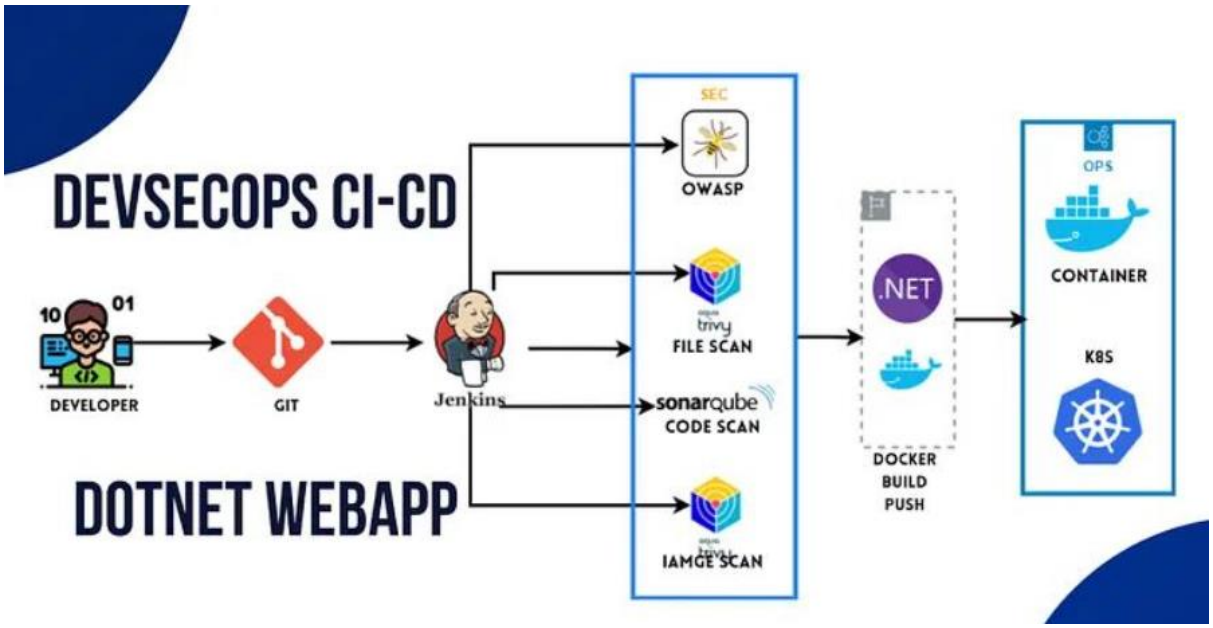
Real-Time DevSecOps Pipeline for a DotNet Web App



Steps:-

Step 1 — Create an Ubuntu T2 Large Instance with 30GB storage

Step 2 — Install Jenkins, Docker and Trivy. Create a Sonarqube Container using Docker.

Step 3 — Install Plugins like JDK, Sonarqube Scanner

Step 4 — Install OWASP Dependency Check Plugins

Step 5 — Configure Sonar Server in Manage Jenkins

Step 6— Create a Pipeline Project in Jenkins using Declarative Pipeline

Step 7 — Install make package

Step 8— Docker Image Build and Push

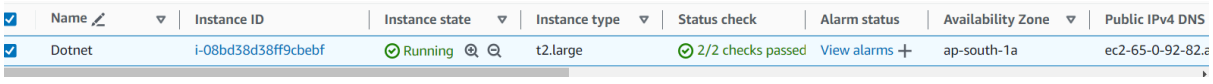
Step 9 — Deploy the image using Docker

Step 10—Access the Real World Application

Step 11— Kubernetes Set Up

Step 12 — Terminate the AWS EC2 Instance

Step 1 — Launch an AWS T2 Large Instance. Use the image as Ubuntu. You can create a new key pair or use an existing one. Enable HTTP and HTTPS settings in the Security Group.



let's connect to your ec2 via ssh using command ssh -i "laptopkey.pem" ubuntu@ec2-65-0-92-82.ap-south-1.compute.amazonaws.com

run the following commands

a. sudo su

b. apt update

clone the github repo by

git clone https://github.com/sai241194/DotNet.git

Step 2 — Install Jenkins, Docker and Trivy. Create a Sonarqube Container using Docker.

2A — To Install Java and Jenkins

Connect to your console, and enter these commands to Install Java latest version and Jenkins

sudo apt update
sudo apt install openjdk-17-jdk
sudo apt install openjdk-17-jre

```
root@ip-172-31-39-3:~# java --version
openjdk 17.0.10 2024-01-16
OpenJDK Runtime Environment (build 17.0.10+7-Ubuntu-122.04.1)
OpenJDK 64-Bit Server VM (build 17.0.10+7-Ubuntu-122.04.1, mixed mode, sharing
root@ip-172-31-39-3:~#
```

**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

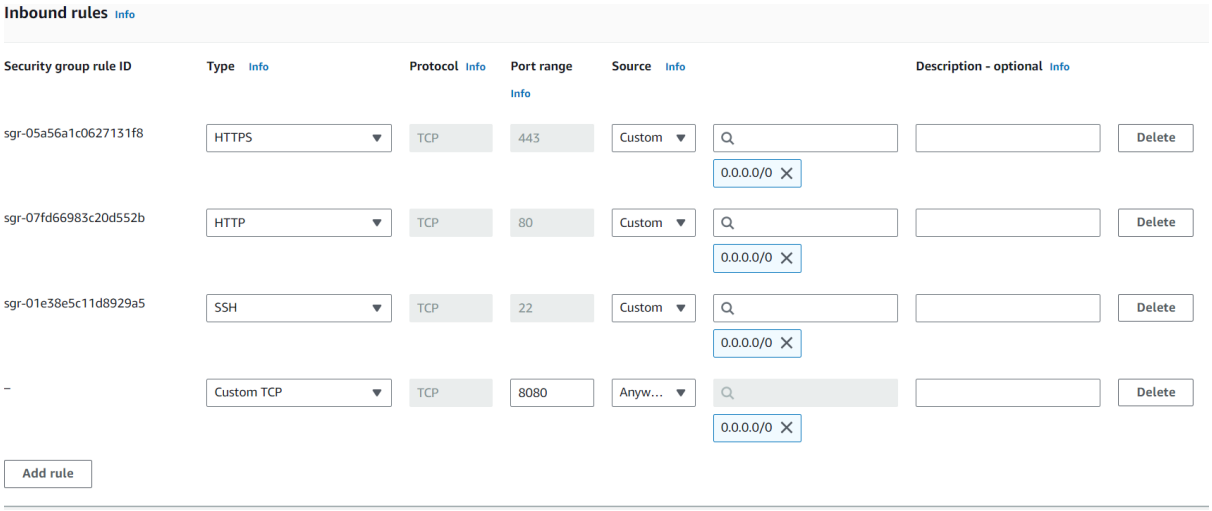
sudo apt-get install jenkins

```
root@ip-172-31-39-3:~# jenkins --version
2.440.2
```

sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo systemctl status Jenkins

```
oot@ip-172-31-39-3:~# sudo systemctl start jenkins
oot@ip-172-31-39-3:~# sudo systemctl status jenkins
 jenkins.service - Jenkins Continuous Integration Server
    Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor pres
    Active: active (running) since Sat 2024-04-06 18:47:40 UTC; 45s ago
  Main PID: 5849 (java)
     Tasks: 52 (limit: 9498)
    Memory: 2.1G
       CPU: 42.016s
    CGroup: /system.slice/jenkins.service
            └─5849 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/jav

pr 06 18:47:21 ip-172-31-39-3 jenkins[5849]: 486315199b3443c3b56176ad8f524952
pr 06 18:47:21 ip-172-31-39-3 jenkins[5849]: This may also be found at: /var/
pr 06 18:47:21 ip-172-31-39-3 jenkins[5849]: ********************************
pr 06 18:47:21 ip-172-31-39-3 jenkins[5849]: ********************************
pr 06 18:47:21 ip-172-31-39-3 jenkins[5849]: ********************************
```

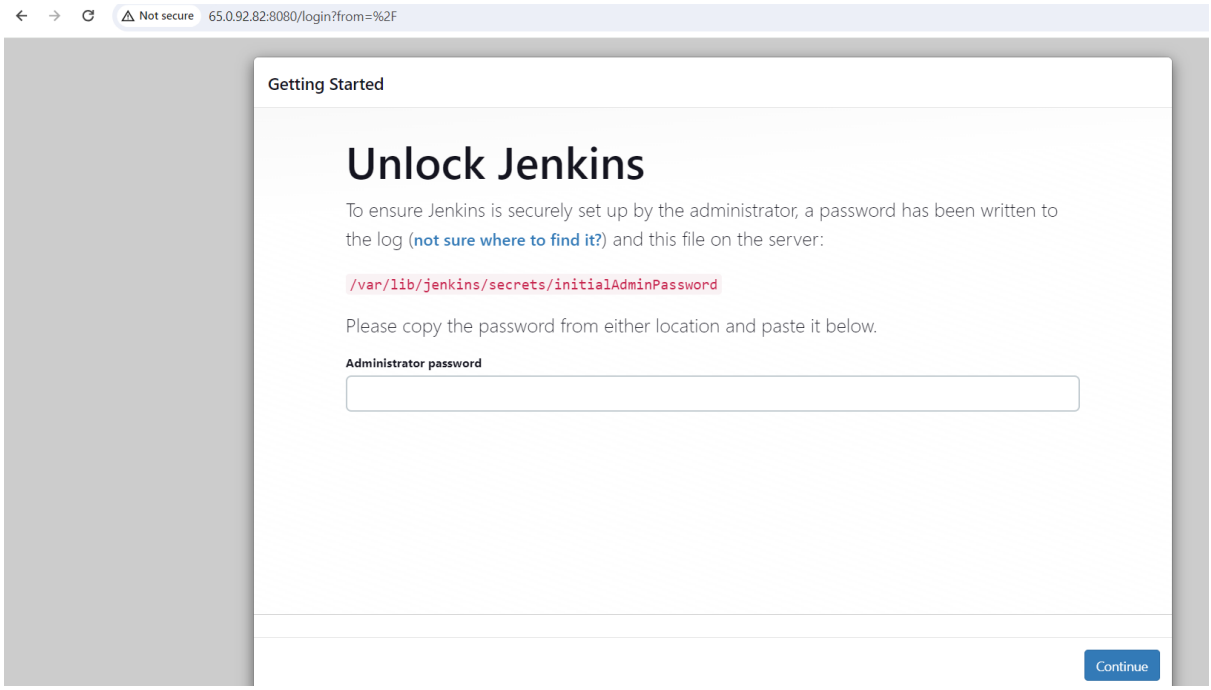
Once Jenkins is installed, you will need to go to your AWS EC2 Security Group and open Inbound Port 8080, since Jenkins works on Port 8080



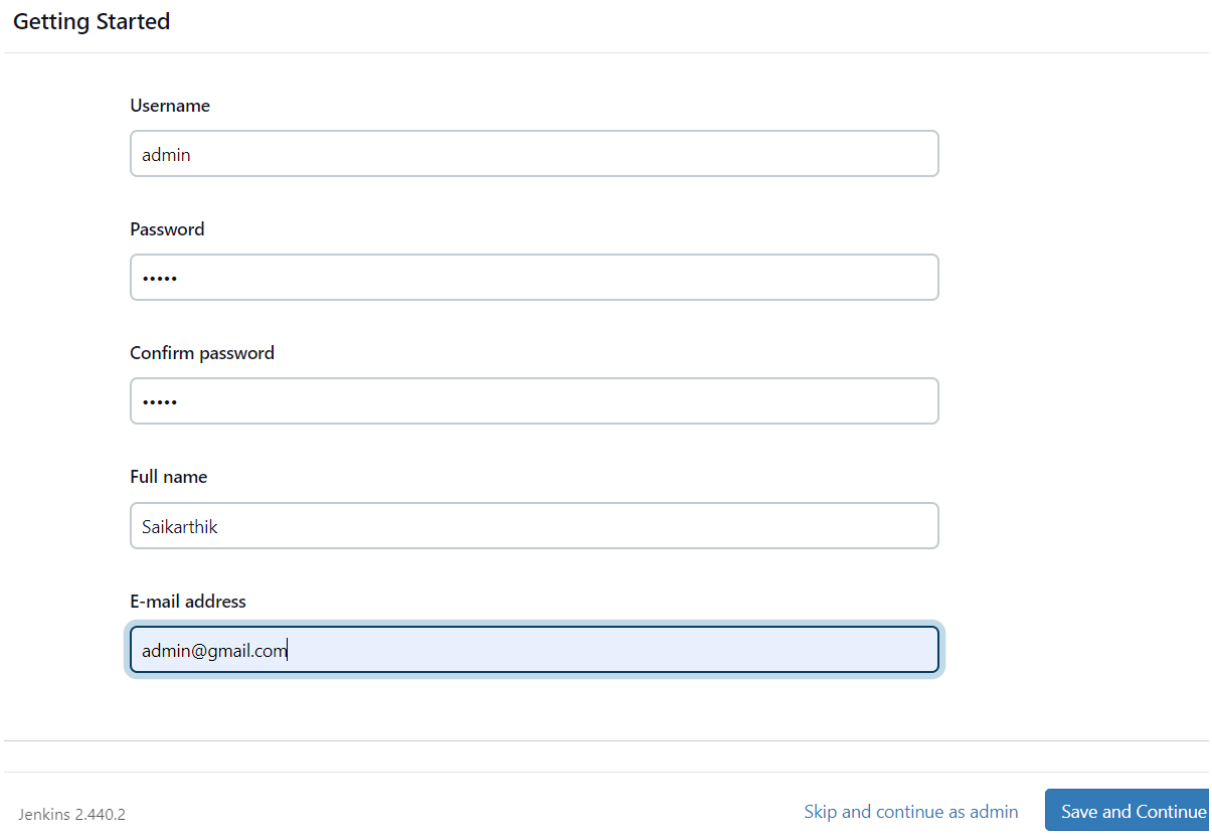
Now, grab your Public IP Address

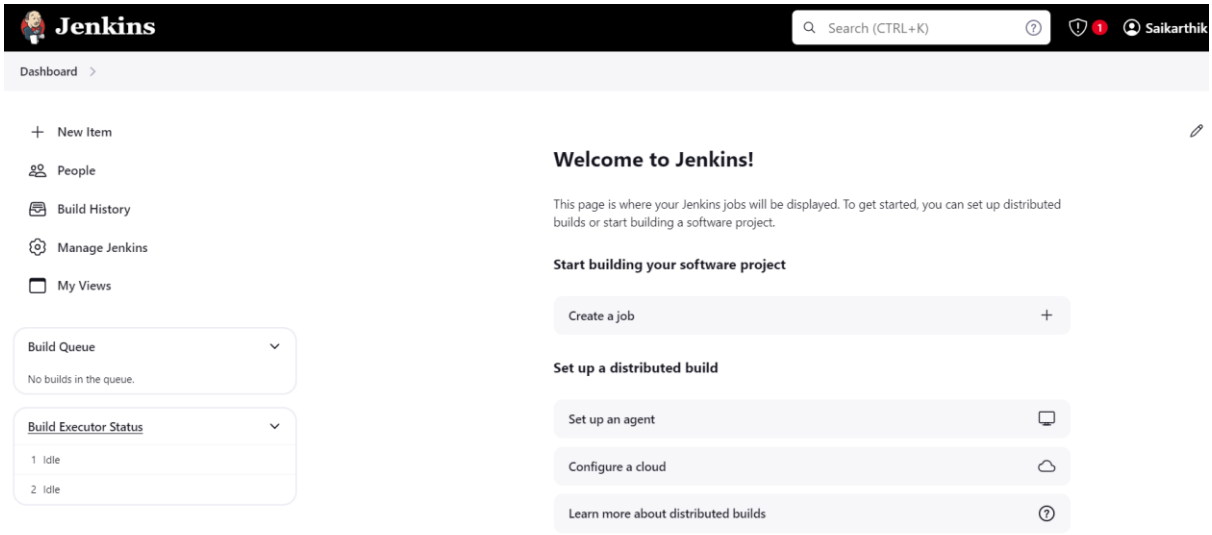
```
<EC2 Public IP Address:8080>
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Unlock Jenkins using an administrative password and install the required plugins.



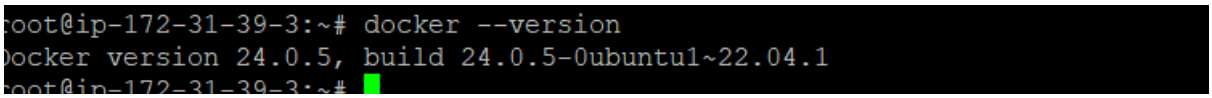
Jenkins will now get installed and install all the libraries.



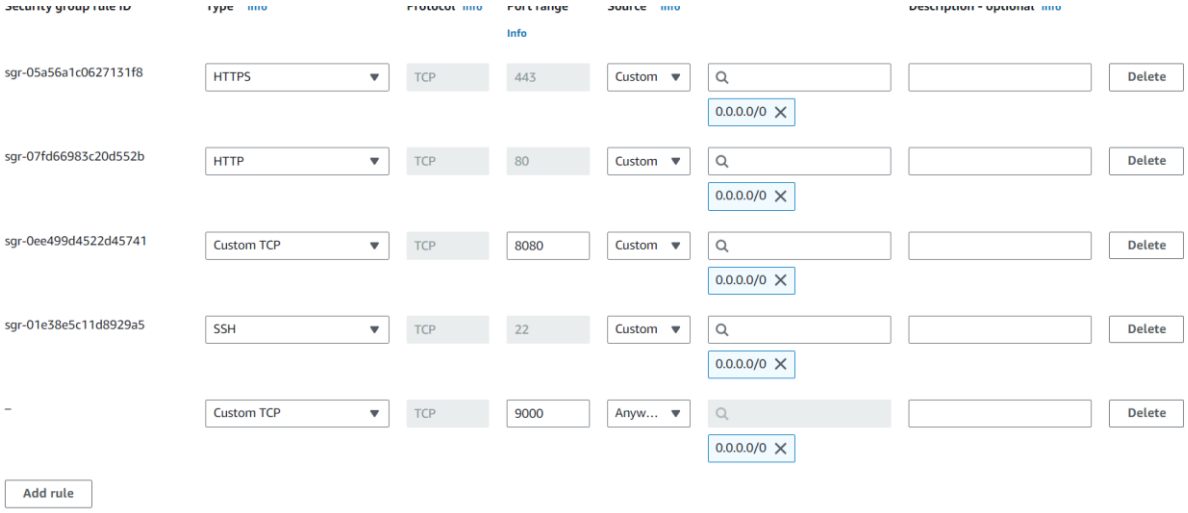


2B — Install Docker

a. **apt-get install docker.io -y**
b. **usermod -aG docker $USER** # Replace with your username e.g ‘ubuntu’
c. **newgrp docker**
d. **sudo chmod 777 /var/run/docker.sock**



After the docker installation, we create a sonarqube container (Remember added 9000 port in the security group)



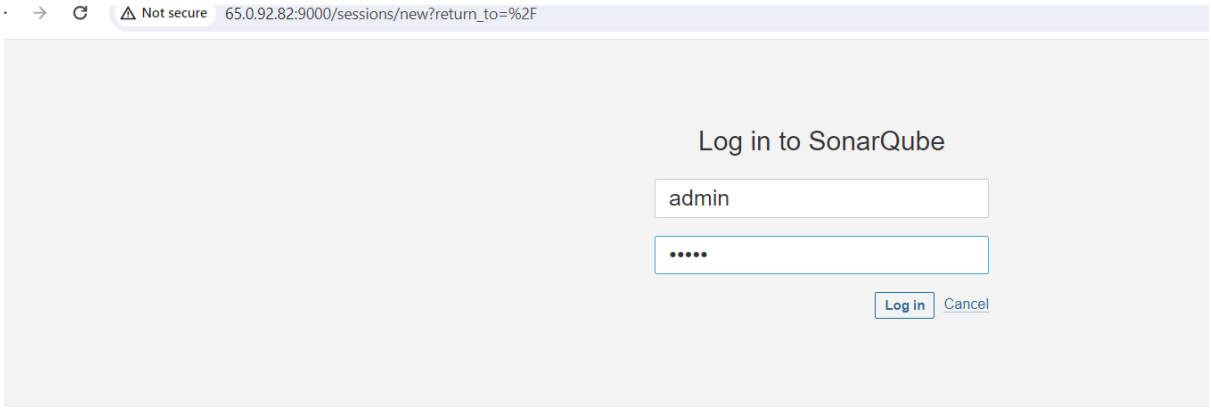
run the following commands to install and run the conatainer of sonarqube on port no. 9000

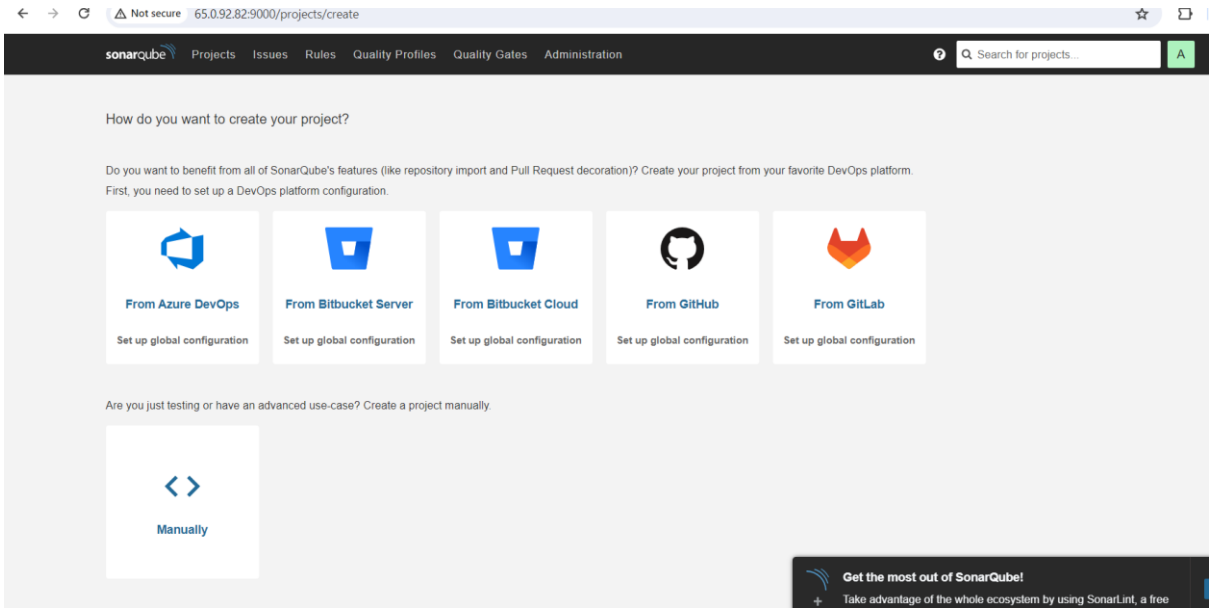
docker run -itd --name sonar -p 9000:9000 sonarqube:lts-community

```
root@ip-172-31-39-3:~# docker run -itd --name sonar -p 9000:9000 sonarqube:lts-c
ommunity
Unable to find image 'sonarqube:lts-community' locally
lts-community: Pulling from library/sonarqube
23828d760c7b: Pull complete
cfcf356fa6e6: Pull complete
b345ba1396e8: Pull complete
33e0a682e40f: Pull complete
9fa5752204b5: Pull complete
70c5a1f7be5b: Pull complete
c7f4e3bed4fe: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:7380e0ec0ebe276e829df5bfbaaf7e58e44eb5adb1873382905577c8d35a0a2c
Status: Downloaded newer image for sonarqube:lts-community
2b974ac86b1478aae25fc9969b1679e3aba05209451ed29816e0710e9a77ef6d
root@ip-172-31-39-3:~#
```

Now browse **http://your_public_ip:9000** you will see **sonarqube is running**

username =admin
password=admin





2C — Install Trivy

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 /etc/apt/sources.list.d/trivy.list

sudo apt-get update

sudo apt-get install trivy

```
root@ip-172-31-39-3:~# trivy --version
Version: 0.50.1
root@ip-172-31-39-3:~#
```

now your trivy is ready to check and scan your image for any vulnerabilities

to check run the following commands

a. trivy image <image id>

```
root@ip-172-31-39-3:~# trivy image 1e64814986e1
2024-04-06T19:11:45.398Z        INFO    Need to update DB
2024-04-06T19:11:45.398Z        INFO    DB Repository: ghcr.io/aquasecurity/trivy-db:2
2024-04-06T19:11:45.398Z        INFO    Downloading DB...
44.86 MiB / 44.86 MiB [-------------------------------------------------------------------------------------------------------------------------------------
2024-04-06T19:11:49.447Z        INFO    Vulnerability scanning is enabled
2024-04-06T19:11:49.447Z        INFO    Secret scanning is enabled
2024-04-06T19:11:49.447Z        INFO    If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2024-04-06T19:11:49.447Z        INFO    Please see also https://aquasecurity.github.io/trivy/v0.50/docs/scanner/secret/#recommendation for faster
2024-04-06T19:11:57.791Z        INFO    Java DB Repository: ghcr.io/aquasecurity/trivy-java-db:1
2024-04-06T19:11:57.791Z        INFO    Downloading the Java DB...
519.32 MiB / 519.32 MiB [---------------------------------------------------------------------------------------------------------------------------------
2024-04-06T19:12:35.372Z        INFO    The Java DB is cached for 3 days. If you want to update the database more frequently, the '--reset' flag
2024-04-06T19:12:35.988Z        INFO    Detected OS: ubuntu
2024-04-06T19:12:35.988Z        INFO    Detecting Ubuntu vulnerabilities...
2024-04-06T19:12:35.995Z        INFO    Number of language-specific files: 1
2024-04-06T19:12:35.995Z        INFO    Detecting jar vulnerabilities...
```

Step 3 — Install some suggested plugins for pipeline to run without errors

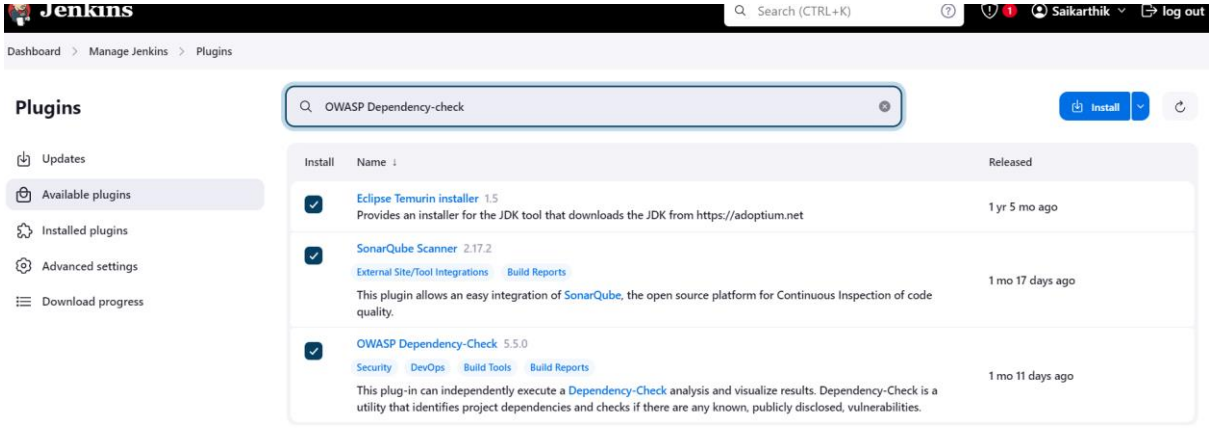
Goto Manage Jenkins →Plugins → Available Plugins →

Install below plugins

1 → Eclipse Temurin Installer (Install without restart)

2 → SonarQube Scanner (Install without restart)

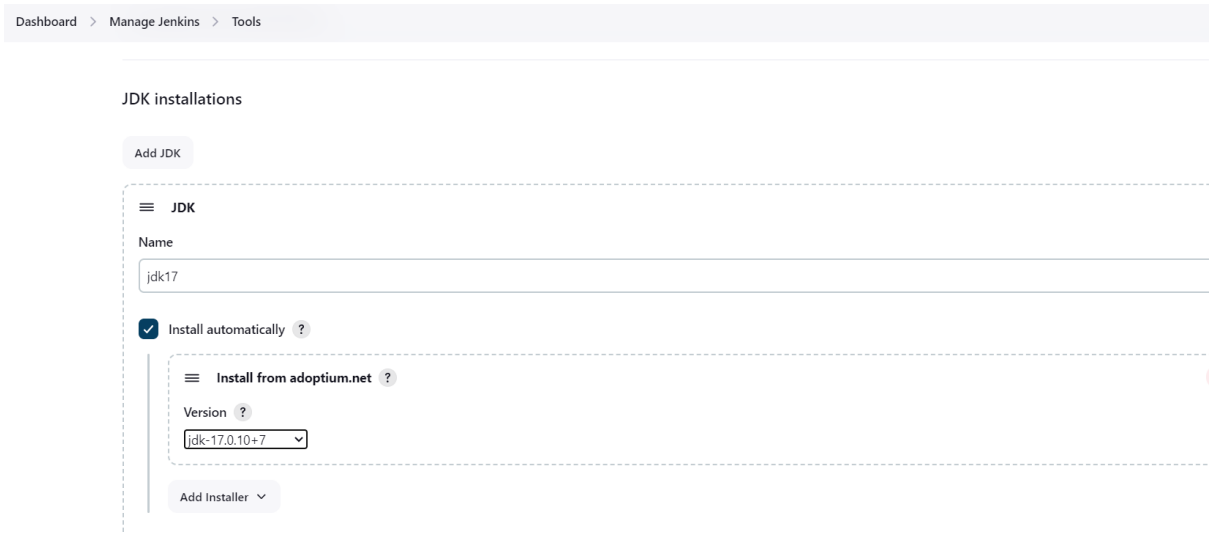
3 → OWASP Dependency Check (Install without restart)



3A — Configure Java in Global Tool Configuration

Goto Manage Jenkins → Tools → Install JDK17→ Click on Apply and Save

1. click on add jdk and select installer adoptium.net
2. choose jdk-17.0.10+7 version and in name section enter jdk 17

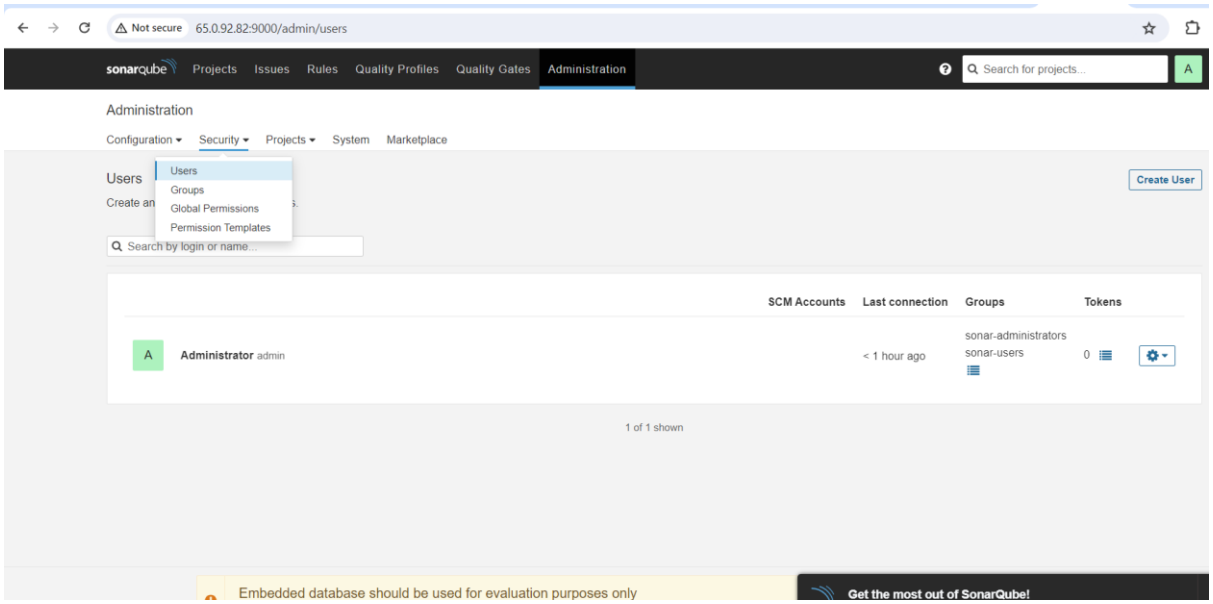


Next is to add dependency check

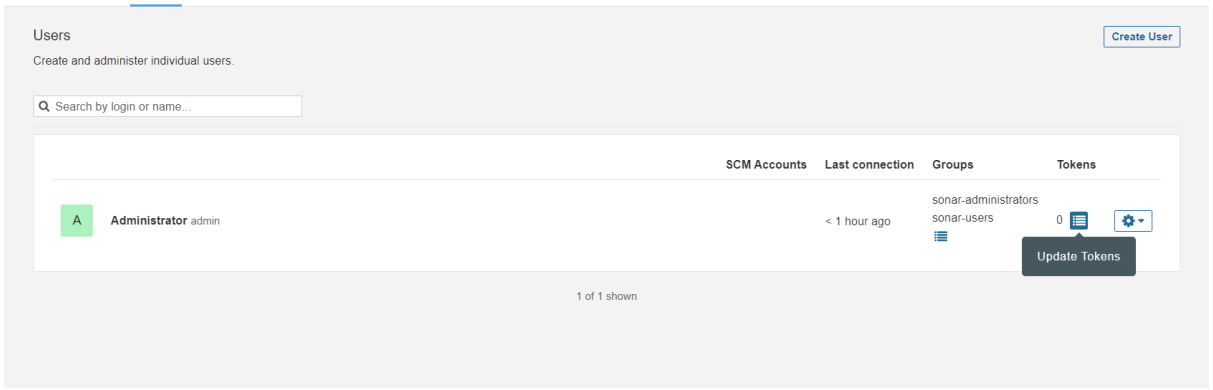
1. add dependency check
2. name = DP-Check
3. from add installer select install from github.com

Click on apply and Save here.

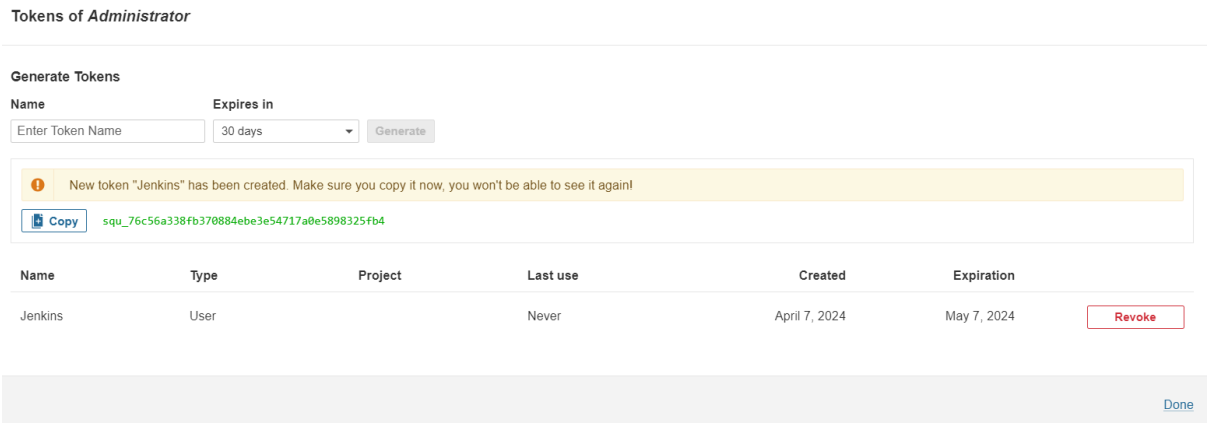
Grab the Public IP Address of your EC2 Instance, Sonarqube works on Port 9000, <Public IP>:9000. Goto your Sonarqube Server. Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → and click on Generate Token



Click on Update Token



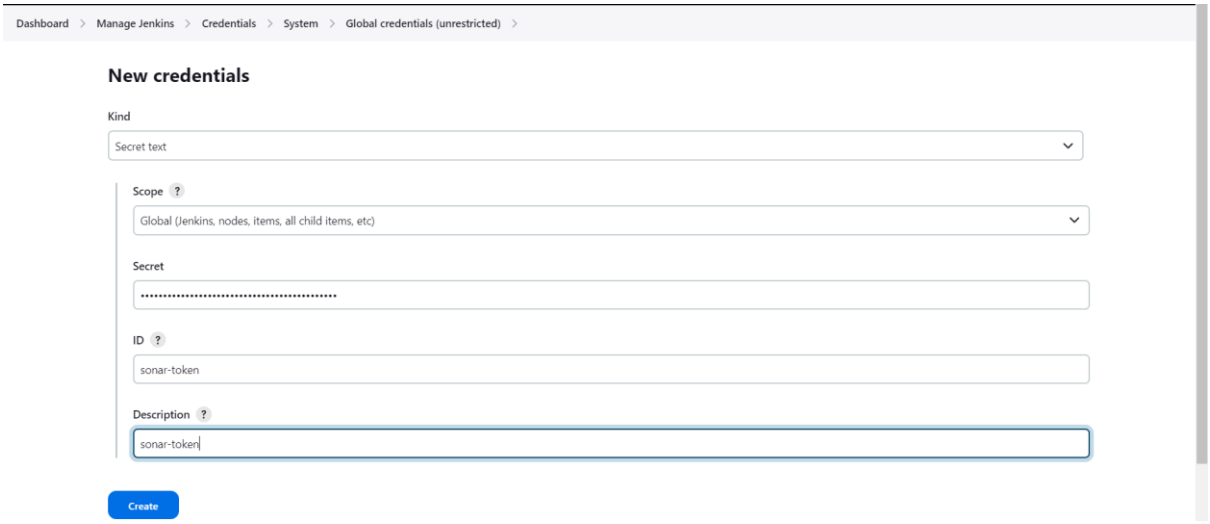
Create a token with a name and generate



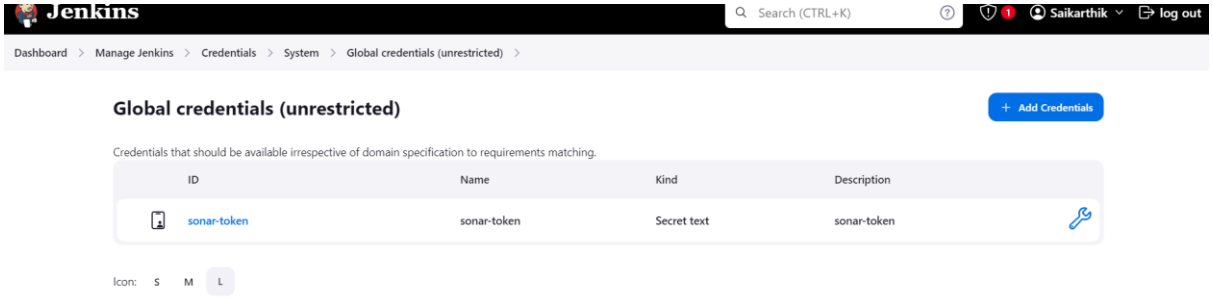
Copy this Token

Goto Dashboard → Manage Jenkins → Credentials → Add Secret Text. It should look like this

Enter the Token in secret field



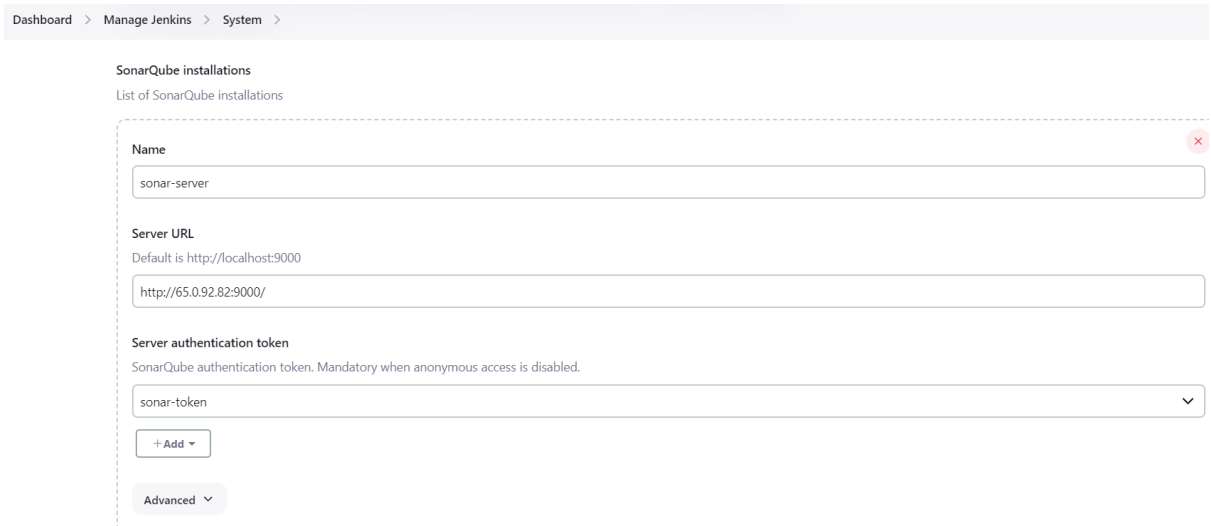
You will this page once you click on create



Step 5 — Configure Sonar Server in Manage Jenkins

Now, go to Dashboard → Manage Jenkins → Configure System→ add sonarqube servers

1. name =sonar-server
2. server_url=http://public_ip:9000
3. server authentication token = sonar-token



Click on Apply and Save

The Configure System option is used in Jenkins to configure different server

Global Tool Configuration is used to configure different tools that we install using Plugins

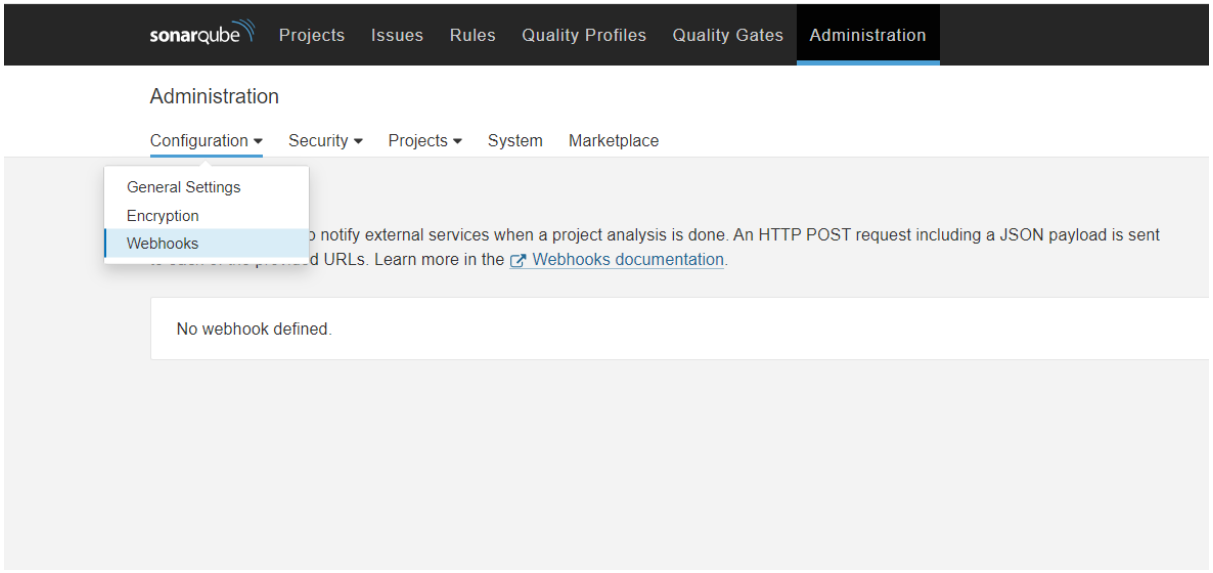
We will install a sonar scanner in the tools.

Go To Dashboard → Manage Jenkins → Tools

1. add sonar scanner
2. name =sonar-scanner

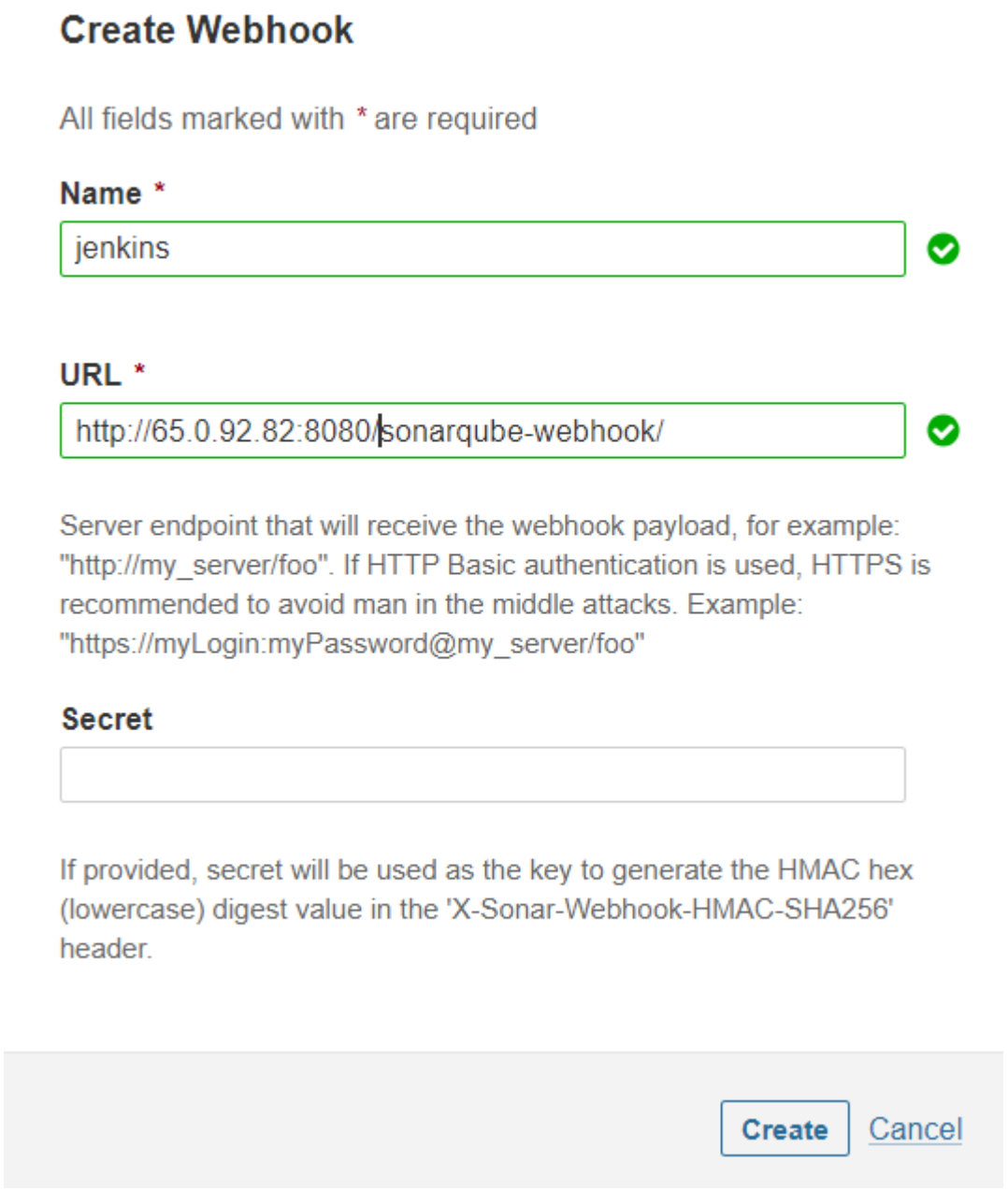
In the Sonarqube Dashboard add a quality gate also

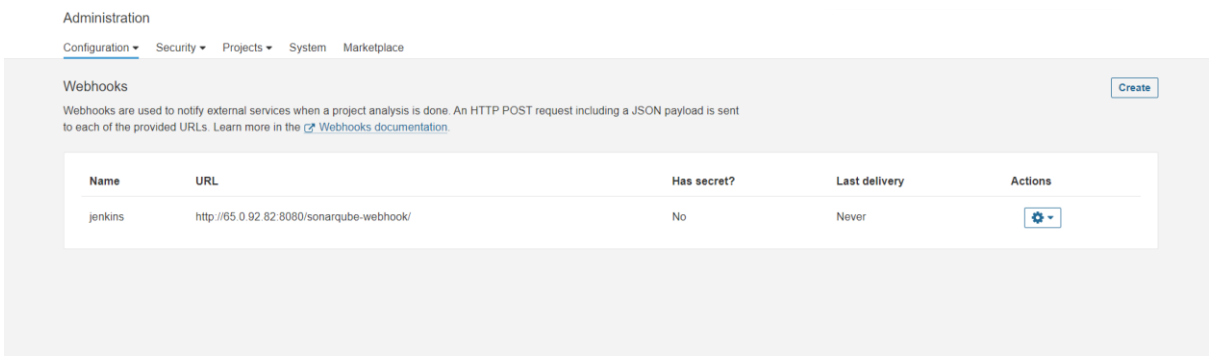
Administration → Configuration →Webhooks



Click on create

#in url section of quality gate
<http://jenkins-public-ip:8080>/sonarqube-webhook/





Click on Apply and Save here.

Step 6 — Create a Pipeline Project in Jenkins using Declarative Pipeline

1. **go to new item** →**select pipeline** →in the name section type **dotnet**
2. scroll down to the pipeline script and copy paste the following code

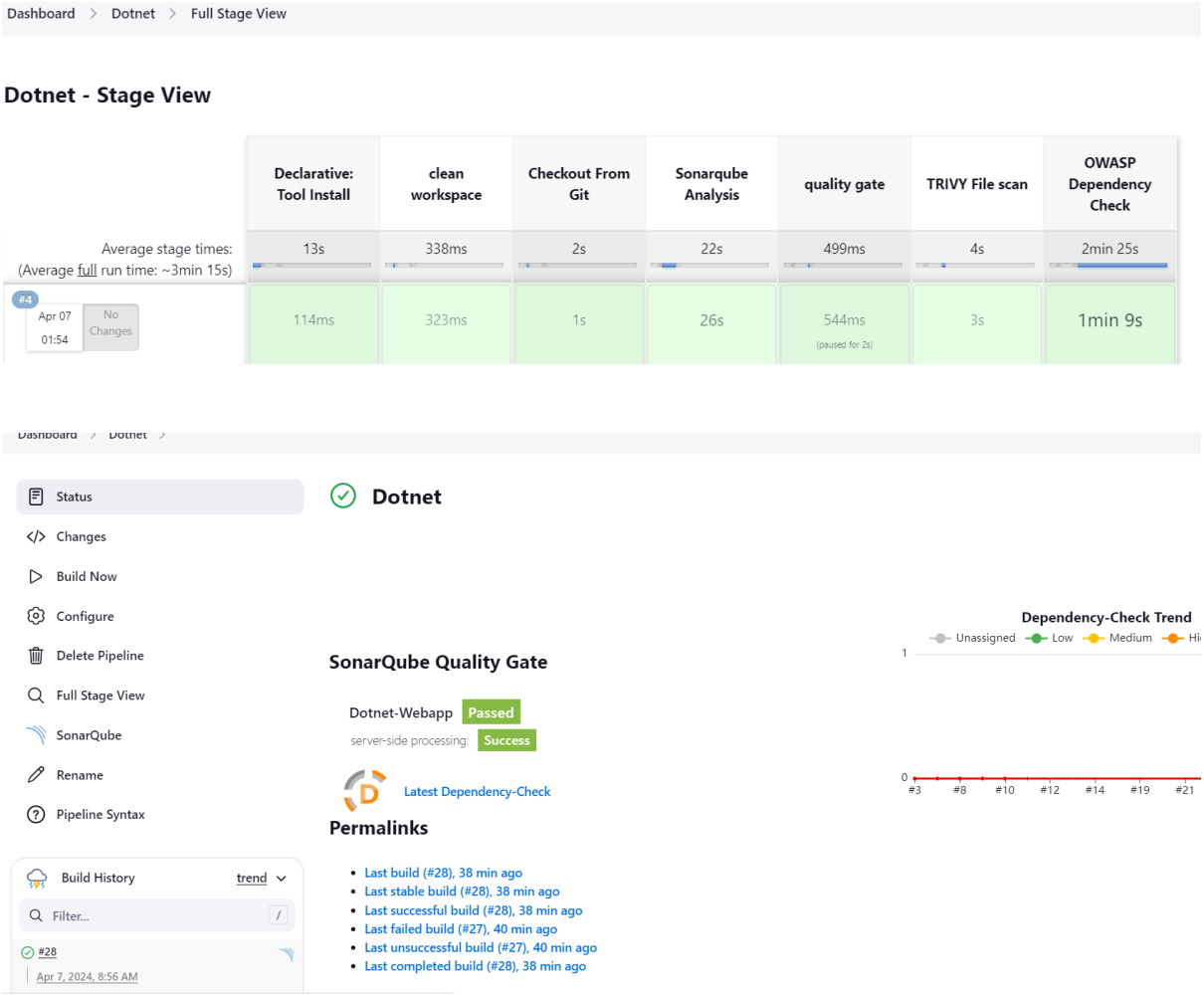
```
pipeline {
    agent any
    tools {
        jdk 'jdk17'
    }
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }
    stages {
        stage('clean workspace') {
            steps {
```

```
            cleanWs()
        }
    }
    stage('Checkout From Git') {
        steps {
            git branch: 'main', url:
'https://github.com/sai241194/DotNet.git'
        }
    }
    stage("Sonarqube Analysis ") {
        steps {
            withSonarQubeEnv('sonar-server') {
                sh """$SCANNER_HOME/bin/sonar-scanner -
Dsonar.projectName=Dotnet-Webapp \
                    -Dsonar.projectKey=Dotnet-Webapp"""
            }
        }
    }
    stage("quality gate") {
```

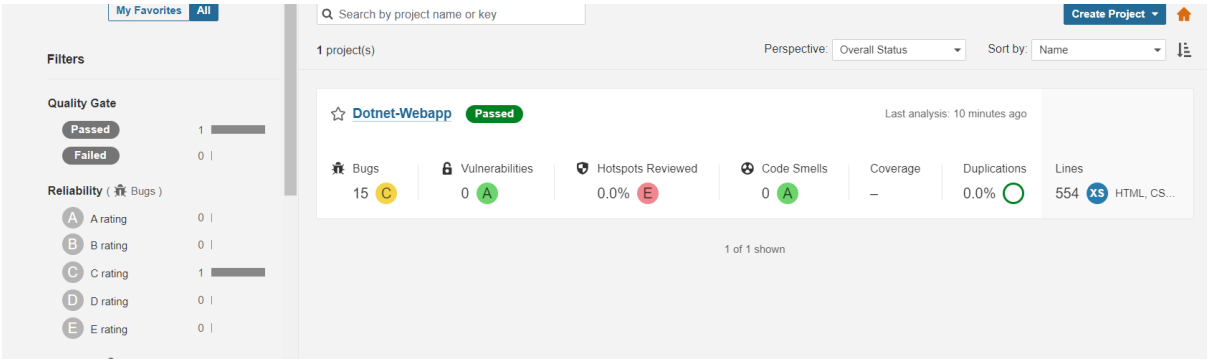
```
        steps {
            script {
                waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'
            }
        }
    }
    stage("TRIVY File scan") {
        steps {
            sh "trivy fs . > trivy-fs_report.txt"
        }
    }
    stage("OWASP Dependency Check") {
        steps {
            dependencyCheck additionalArguments: '--scan ./ --format XML ', odcInstallation: 'DP-Check'
            dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
        }
```

```
        }
    }
}
```

Click on Build now, you will see the stage view like this



To see the report, you can go to Sonarqube Server and go to Projects.



Step 7 — Install make package

sudo apt install make

to check version install or not

make -v

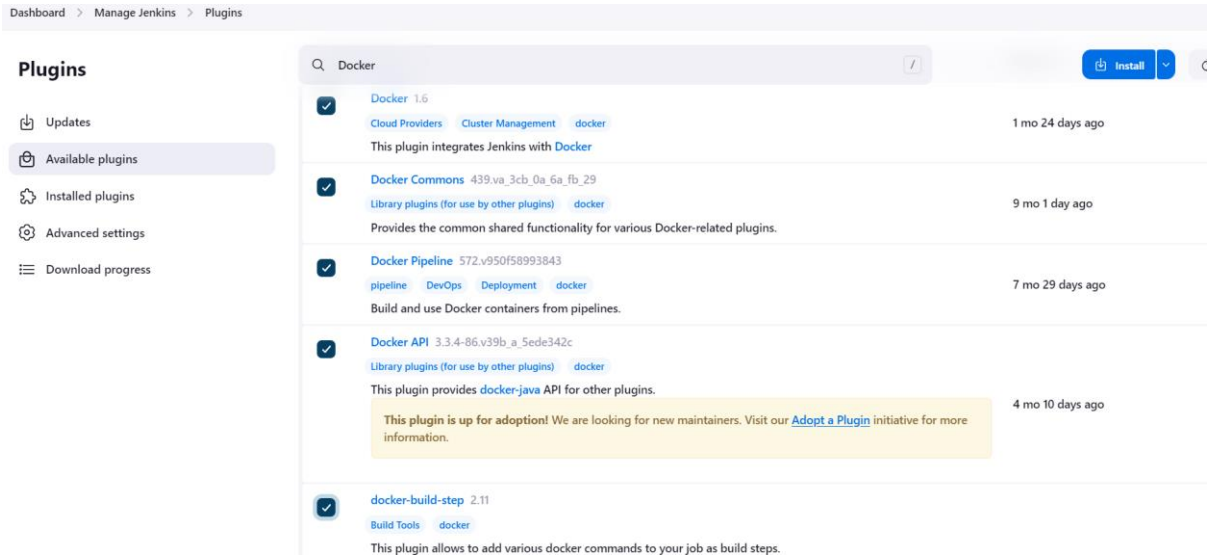
```
root@ip-172-31-39-3:~# make -v
GNU Make 4.3
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
root@ip-172-31-39-3:~#
```

Step 8 — Docker Image Build and Push

We need to install the Docker tool in our system, Goto Dashboard → Manage Plugins → Available plugins → Search for Docker and install these plugins

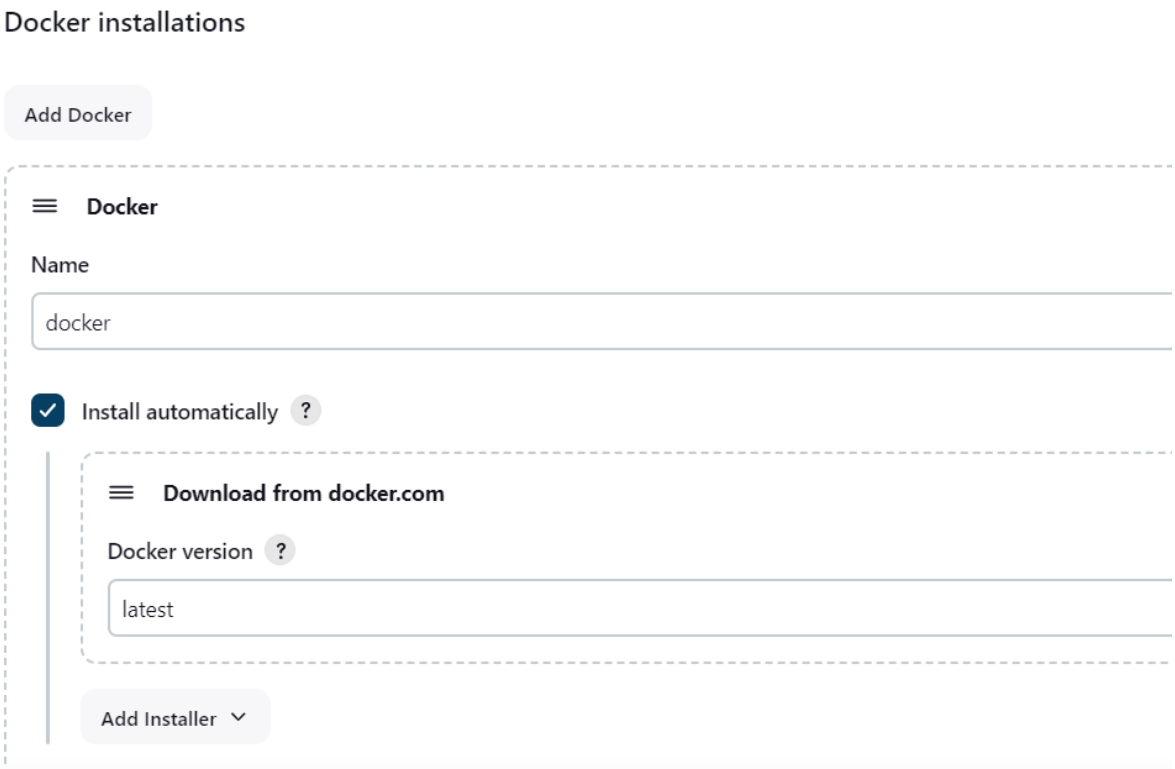
- `Docker`
- `Docker Commons`
- `Docker Pipeline`
- `Docker API`
- `docker-build-step`

and click on install without restart



Now, goto Dashboard → Manage Jenkins → Tools

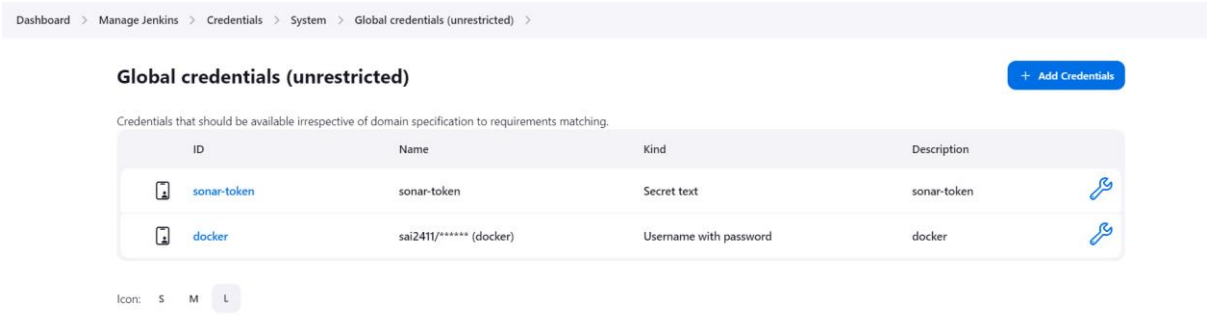
1. click on add docker
2. name=docker
3. add installer =download from docker.com



Click on apply and save

Now go to Dashboard → Manage Jenkins → Credentials

Add DockerHub Username and Password under Global Credentials



In the makefile, we already defined some conditions to build, tag and push images to dockerhub.

that’s why we are using make image and make a push in the place of docker build -t and docker push

Add this stage which is highlighted in bold to Pipeline Script

```
pipeline {
    agent any
    tools {
        jdk 'jdk17'
```

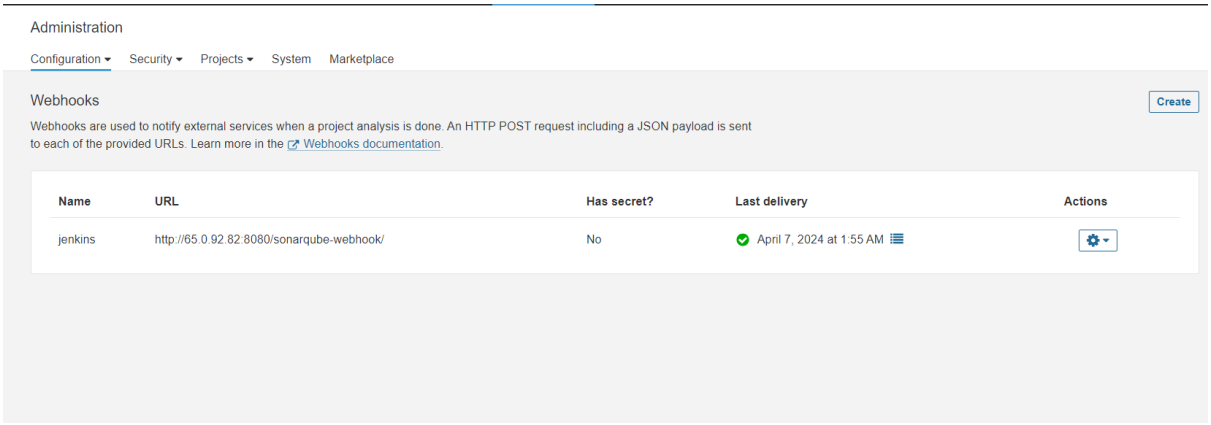
```
}
environment {
    SCANNER_HOME = tool 'sonar-scanner'
}
stages {
    stage('clean workspace') {
        steps {
            cleanWs()
        }
    }
    stage('Checkout From Git') {
        steps {
            git branch: 'main', url: 'https://github.com/sai241194/DotNet.git'
        }
    }
    stage("Sonarqube Analysis ") {
        steps {
            withSonarQubeEnv('sonar-server') {
```

```
                sh """$SCANNER_HOME/bin/sonar-scanner -
Dsonar.projectName=Dotnet-Webapp \
                -Dsonar.projectKey=Dotnet-Webapp"""
            }
        }
    }
    stage("quality gate") {
        steps {
            script {
                waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'
            }
        }
    }
    stage("TRIVY File scan") {
        steps {
            sh "trivy fs . > trivy-fs_report.txt"
        }
    }
    stage("OWASP Dependency Check") {
```

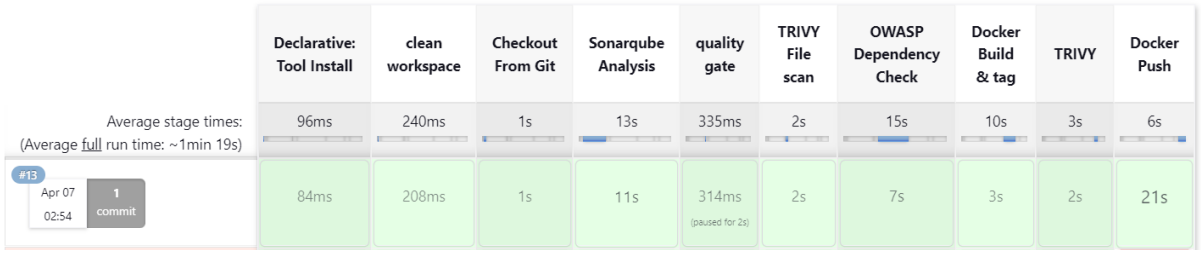
```
        steps {
            dependencyCheck additionalArguments: '--scan ./ --format XML ', odcInstallation: 'DP-Check'
            dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
        }
    }
    stage("Docker Build & tag") {
      steps {
        script {
          withDockerRegistry(credentialsId: 'docker', toolName: 'docker'){
            sh "make image"
          }
        }
      }
    }
    stage("TRIVY") {
      steps {
        sh "trivy image sai2411/dotnet-monitoring:latest > trivy.txt"
      }
```

```
        }
        stage("Docker Push") {
            steps {
                script {
                    withDockerRegistry(credentialsId: 'docker', toolName: 'docker'){
                        sh "make push"
                    }
                }
            }
        }
    }
}
```

When all stages in docker are successfully created then you will see the result You log in to Dockerhub, and you will see a new image is created



Stage View



Step 9 — Deploy the image using Docker

Add this stage to your above pipeline syntax

stage("Deploy to container"){

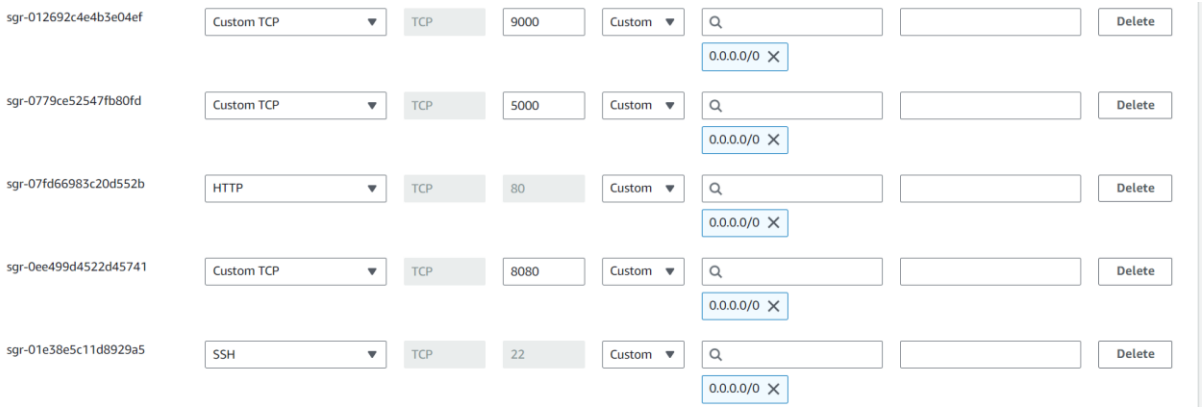
steps{

sh "docker run -itd --name dotnet -p 5000:5000 sai2411/dotnet-monitoring:latest"

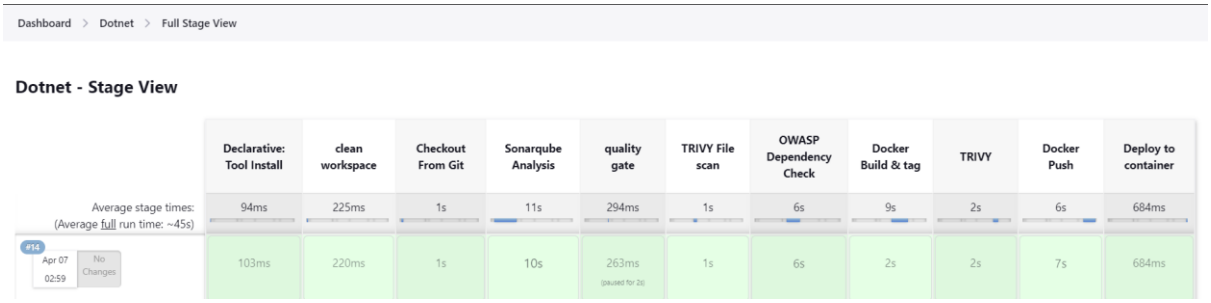
}

}

(Add port 5000 to Security Group)

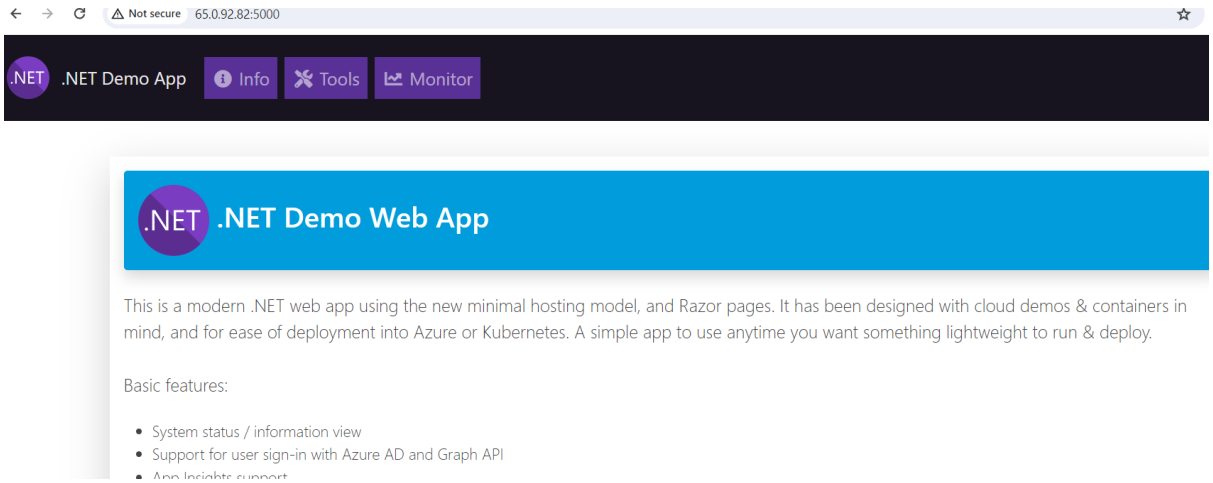


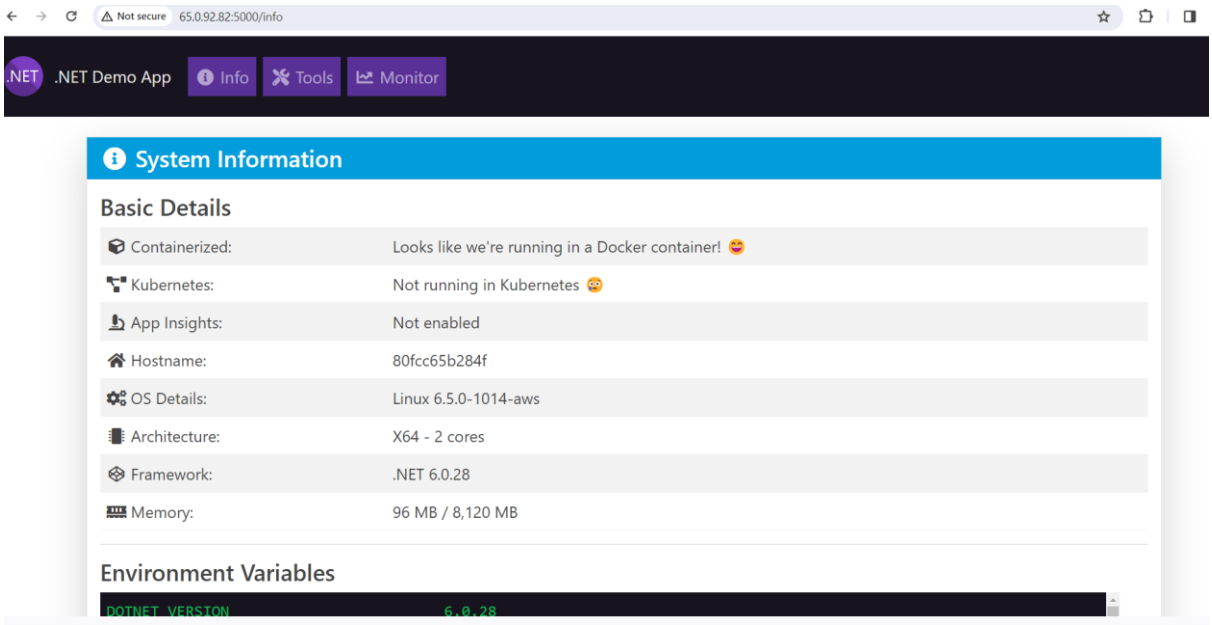
You will see the Stage View like this,

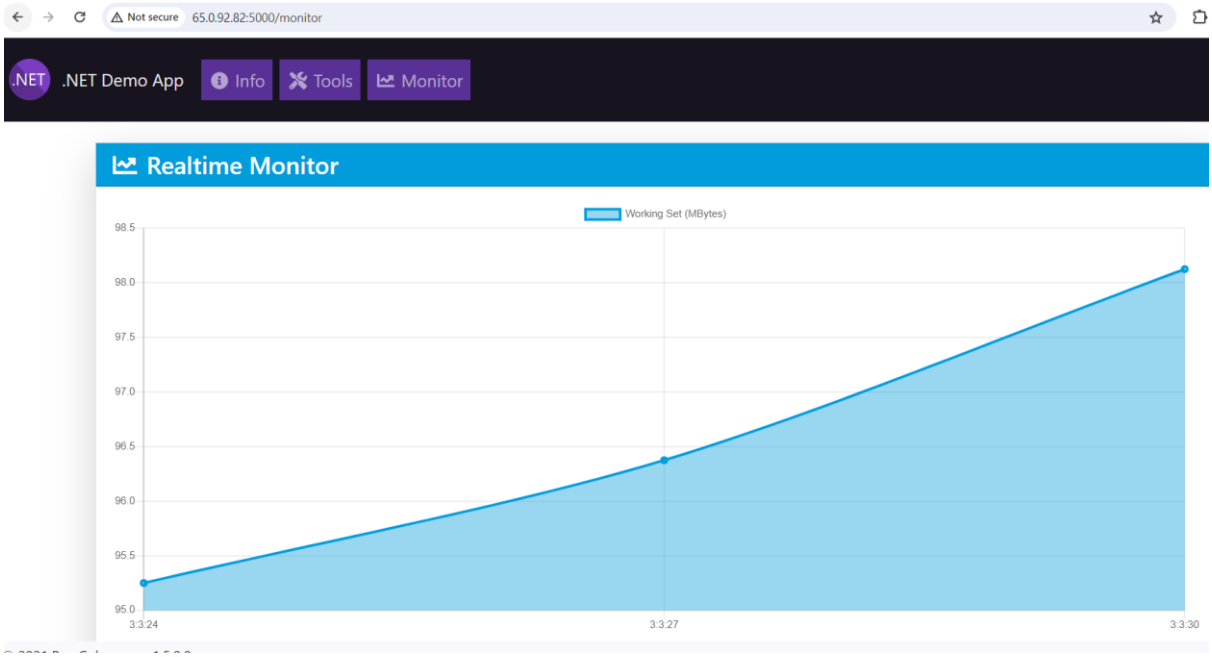


And you can access your application on Port 5000. This is a Real World Application that has all Functional Tabs.

Step 10 — Access the Real World Application


Not secure 65.0.92.82:5000
.NET .NET Demo App
Info
Tools
Monitor
.NET .NET Demo Web App
This is a modern .NET web app using the new minimal hosting model, and Razor pages. It has been designed with cloud demos & containers in mind, and for ease of deployment into Azure or Kubernetes. A simple app to use anytime you want something lightweight to run & deploy.
Basic features:
System status / information view
Support for user sign-in with Azure AD and Graph API


Not secure 65.0.92.82:5000/info
.NET .NET Demo App
Info
Tools
Monitor
System Information
Basic Details
Containerized: Looks like we're running in a Docker container!
Kubernetes: Not running in Kubernetes
App Insights: Not enabled
Hostname: 80fcc65b284f
OS Details: Linux 6.5.0-1014-aws
Architecture: X64 - 2 cores
Framework: .NET 6.0.28
Memory: 96 MB / 8,120 MB
Environment Variables
DOTNET_VERSION 6.0.28



Step 11 — Kubernetes Set Up

Install Kops and Kubectl on Jenkins machine as well.

```
curl -Lo kops https://github.com/kubernetes/kops/releases/download/$(curl -s
https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag_name |
cut -d '"' -f 4)/kops-linux-amd64
chmod +x kops
sudo mv kops /usr/local/bin/kops
```

```
curl -LO https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
chmod +x kubectl
sudo mv kops /usr/local/bin/kubectl
```

Create one IAM user and give full admin access

Login with AWS CLI and install AWS CLI using apt install awscli -y

Give aws configure and give details (AWS Access Key ID,Secret Access Key,region and output format)

Now we need to export our access key and secret key

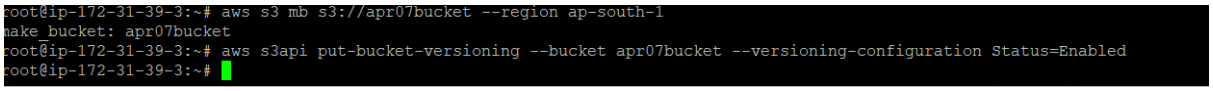
```
export AWS_ACCESS_KEY_ID=$(aws configure get
aws_access_key_id)
export AWS_SECRET_ACCESS_KEY=$(aws configure get
aws_secret_access_key)
```

Next create S3 bucket and enable versionong

aws s3 mb s3://apr07bucket --region ap-south-1

mb=make bucket

aws s3api put-bucket-versioning --bucket sep16bucket --versioning-configuration Status=Enabled



Next create sshkey for machines

ssh-keygen

id_rsa=private key

id_rsa.pub=public Key

We are now going to give name for cluster by using DNS or gossip-based cluster

DNS:

```
 # export NAME=aws.sai.com
  # export KOPS_STATE_STORE=s3://apr07bucket
```

Or

Gossip-based: free domain shared by K8'S

For a gossip-based cluster, make sure the name ends with k8s.local. For example:

```
  # export NAME=sai.k8s.local
  # export KOPS_STATE_STORE=s3://apr07bucket
```

Creating Cluster now

kops create cluster --zones ap-south-1b ${NAME}

Control Plane-master
Node-worker

To edit node:

kops edit ig --name=sai.k8s.local nodes-ap-south-1b

ig=instance group

To edit control plane:

kops edit ig --name=sai.k8s.local control-plane-ap-south-1b

Update cluster:

kops update cluster --name sai.k8s.local --yes –admin

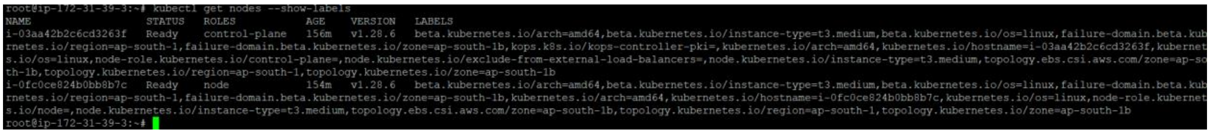
List clusters: **kops get cluster**

```
root@ip-172-31-39-3:~# kops get cluster
NAME              CLOUD    ZONES
sai.k8s.local     aws      ap-south-1b
root@ip-172-31-39-3:~#
```

Validate Cluster:**kops validate cluster**

```
root@ip-172-31-39-3:~# kops validate cluster
Using cluster from kubectl context: sai.k8s.local

Validating cluster sai.k8s.local

INSTANCE GROUPS
NAME                            ROLE            MACHINETYPE     MIN     MAX     SUBNETS
control-plane-ap-south-1b       ControlPlane    t3.medium       1       1       ap-south-1b
nodes-ap-south-1b               Node            t3.medium       1       1       ap-south-1b

NODE STATUS
NAME                    ROLE            READY
i-03aa42b2c6cd3263f     control-plane   True
i-0fc0ce824b0bb8b7c     node            True

Your cluster sai.k8s.local is ready
root@ip-172-31-39-3:~#
```

List Nodes: **kubectl get nodes --show-labels**



Now Give this command in CLI

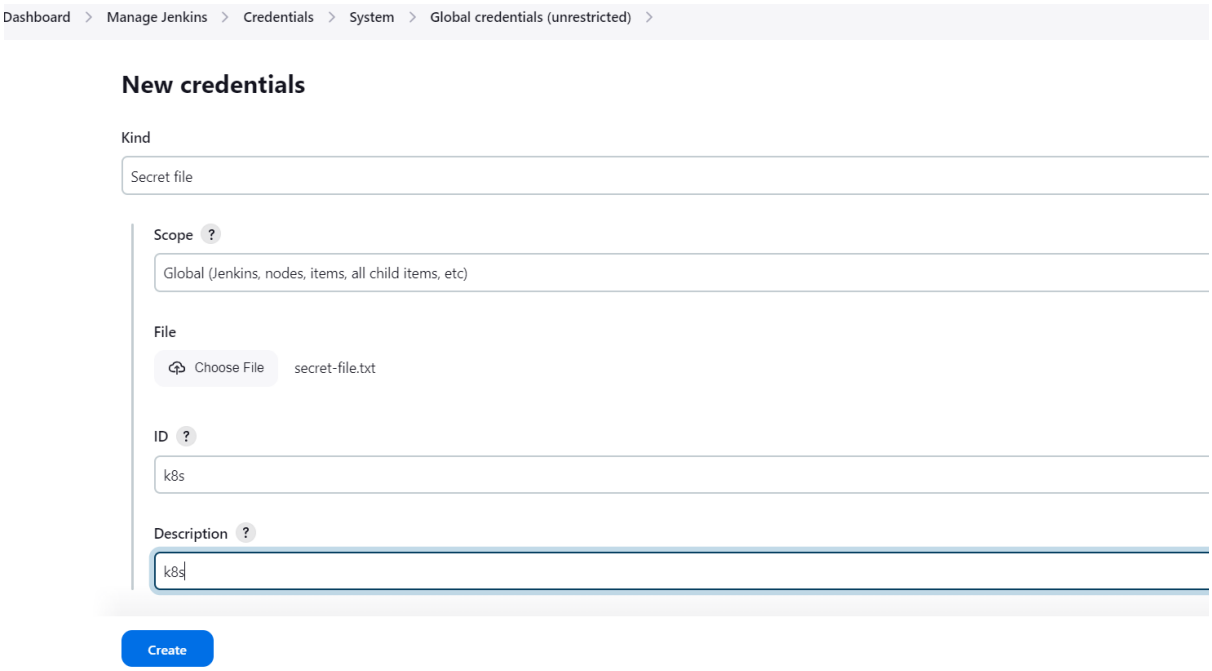
cat /root/.kube/config

Copy the config file to Jenkins master or the local file manager and save it as secret-file.txt

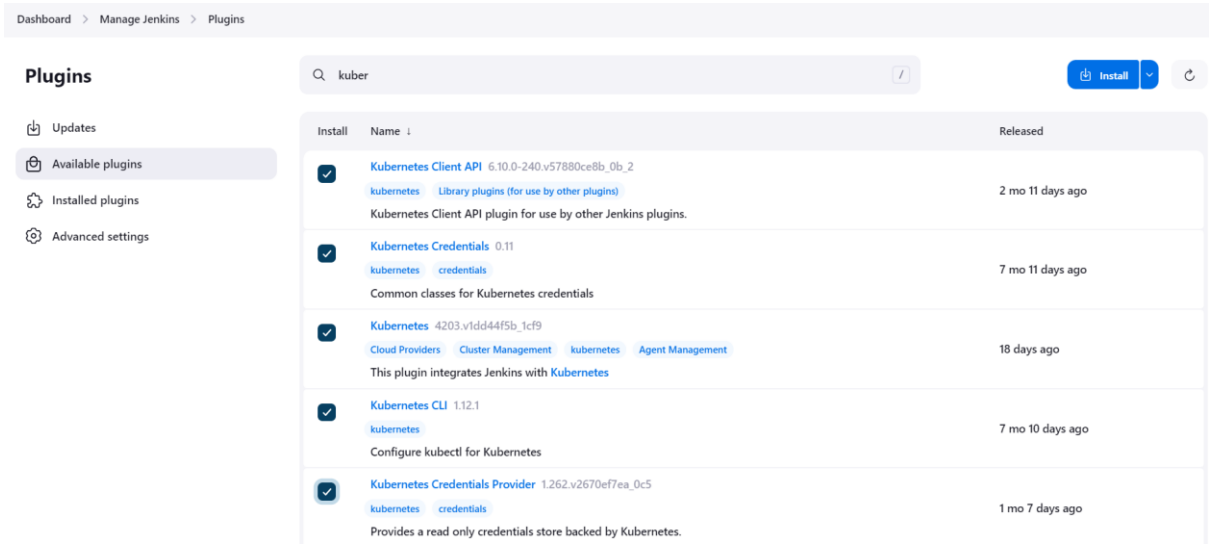
Now, goto the Master Node

Install Kubernetes Plugin, Once it’s installed successfully

goto manage Jenkins → manage credentials → Click on Jenkins global → add credentials



Install Kubernetes Plugin, Once it's installed successfully



The final step to deploy on the Kubernetes cluster, add this stage to the pipeline.

stage('Deploy to k8s'){

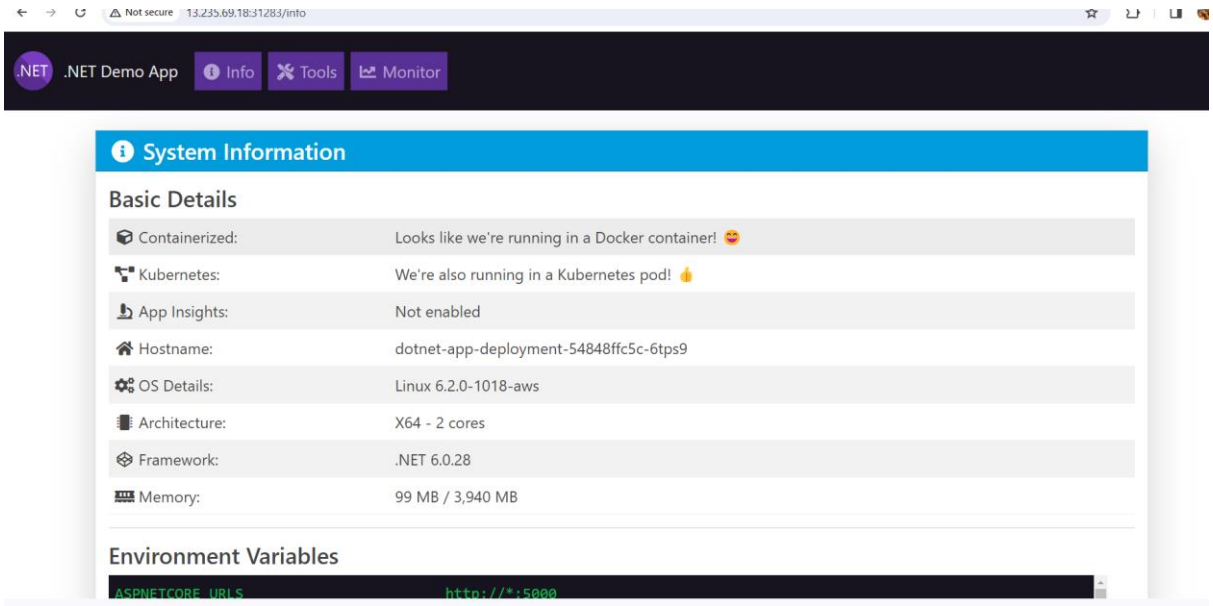
```
        steps{
            dir('K8S') {
              withKubeConfig(caCertificate: '', clusterName: '', contextName: '', credentialsId: 'k8s', namespace: '', restrictKubeConfigAccess: false, serverUrl: '') {
                sh 'kubectl apply -f deployment.yaml'
              }
            }
        }
    }
```

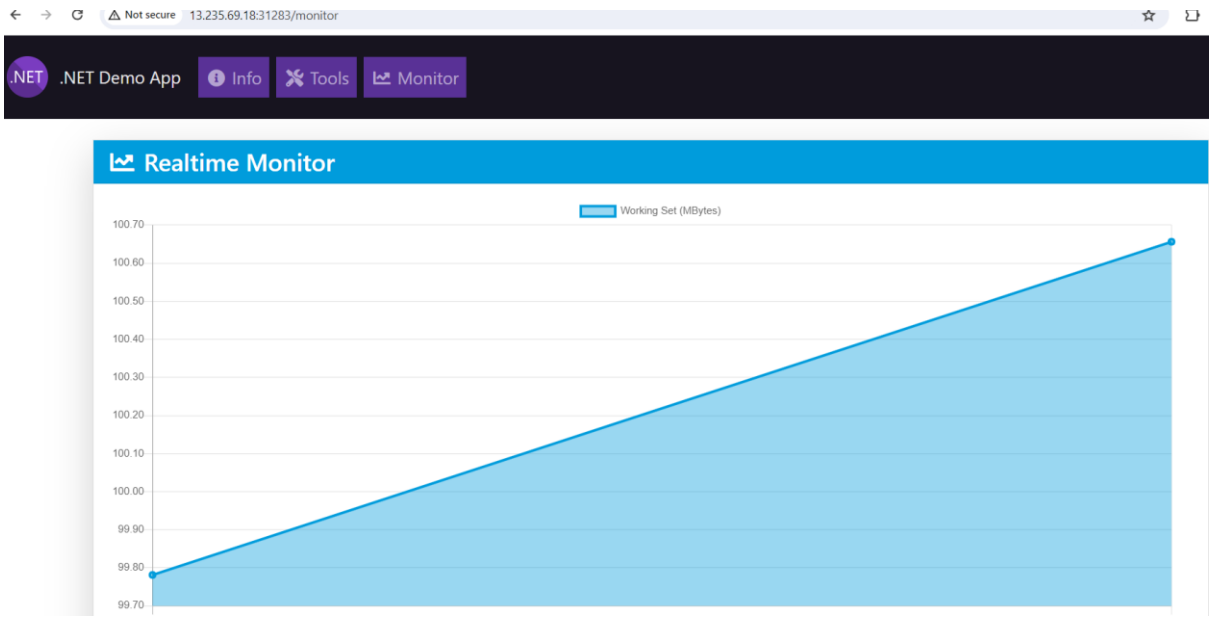
Before starting a new build remove Old containers.

kubectl get svc
#copy service port
<worker-ip:svc port>

< http://13.235.69.18:31283/>

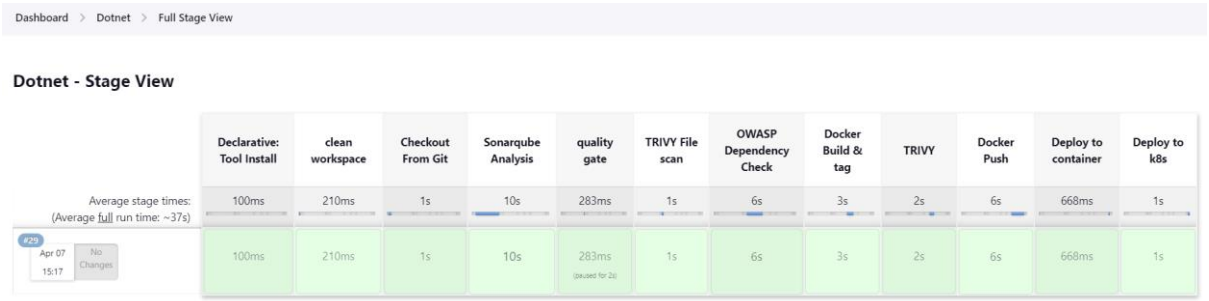
```
root@ip-172-31-39-3:~# kubectl get svc
NAME                TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
dotnet-app-service  NodePort    100.65.8.187   <none>        80:31283/TCP   18m
kubernetes          ClusterIP   100.64.0.1     <none>        443/TCP        159m
root@ip-172-31-39-3:~#
```





Step 12 — Terminate the AWS EC2 Instance

The complete pipeline script




```
pipeline{
    agent any
    tools{
        jdk 'jdk17'
    }
    environment {
        SCANNER_HOME=tool 'sonar-scanner'
    }
    stages {
        stage('clean workspace'){
            steps{
                cleanWs()
```

```
        }
    }
    stage('Checkout From Git'){
        steps{
            git branch: 'main', url:
'https://github.com/sai241194/DotNet.git'
        }
    }
    stage("Sonarqube Analysis "){
        steps{
            withSonarQubeEnv('sonar-server') {
                sh '" $SCANNER_HOME/bin/sonar-scanner -
Dsonar.projectName=Dotnet-Webapp \
                -Dsonar.projectKey=Dotnet-Webapp "'
            }
        }
    }
    stage("quality gate"){
        steps {
```

```
            script {
                waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'
            }
        }
    }
    stage("TRIVY File scan"){
        steps{
            sh "trivy fs . > trivy-fs_report.txt"
        }
    }
    stage("OWASP Dependency Check"){
        steps{
            dependencyCheck additionalArguments: '--scan ./ --format XML ', odcInstallation: 'DP-Check'
            dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
        }
    }
```

```
        stage("Docker Build & tag"){
            steps{
                script{
                    withDockerRegistry(credentialsId: 'docker', toolName: 'docker'){
                        sh "make image"
                    }
                }
            }
        }
        stage("TRIVY"){
            steps{
                sh "trivy image sai2411/dotnet-monitoring:latest > trivy.txt"
            }
        }
        stage("Docker Push"){
            steps{
                script{
```

```
            withDockerRegistry(credentialsId: 'docker',
toolName: 'docker'){

                sh "make push"

            }

        }

    }

    stage("Deploy to container"){

        steps{

            sh "docker run -d --name dotnet -p 5000:5000
sai2411/dotnet-monitoring:latest"

        }

    }

    stage('Deploy to k8s'){

        steps{

            dir('K8S') {

              withKubeConfig(caCertificate: '', clusterName: '',
contextName: '', credentialsId: 'k8s', namespace: '',
restrictKubeConfigAccess: false, serverUrl: '') {

                sh 'kubectl apply -f deployment.yaml'
```

```
            }
        }
    }
}
}
```

Here is the GitHub for this project

https://github.com/sai241194/DotNet