

Hello friends, we will be deploying a .Net-based application. This is an everyday use case scenario used by several organizations. We will be using Jenkins as a CICD tool and deploying our application on a Docker Container and Kubernetes cluster. Hope this detailed blog is useful.

Github: <https://github.com/Aj7Ay/DotNet-monitoring.git>

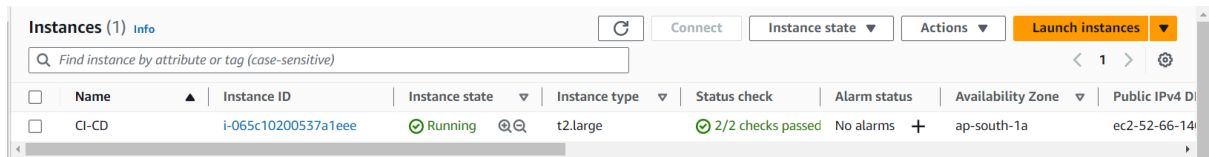
### Steps:-

- Step 1 — Create an Ubuntu T2 Large Instance
- Step 2 — Install Jenkins, Docker and Trivy. Create a Sonarqube Container using Docker.
- Step 3 — Install Plugins like JDK, Sonarqube Scanner, OWASP Dependency Check,
- Step 4 — Create a Pipeline Project in Jenkins using a Declarative Pipeline
- Step 5 — Configure Sonar Server in Manage Jenkins
- Step 6 — we have to install and make the package
- Step 7 — Docker Image Build and Push
- Step 8 — Deploy the image using Docker
- Step 9 — Access the Real World Application
- Step 10 — Kubernetes setup
- Step 11 — Terminate the AWS EC2 Instance

**Now, let's get started and dig deeper into each of these steps:-**

**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.



Instances (1) Info								
Find instance by attribute or tag (case-sensitive)								
	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 D
<input type="checkbox"/>	CI-CD	i-065c10200537a1eee	Running	t2.large	2/2 checks passed	No alarms	ap-south-1a	ec2-52-66-14

## Step 2 — Install Jenkins, Docker and Trivy

### 2A — To Install Jenkins

Connect to your console, and enter these commands to Install Jenkins

```
sudo vi jenkins.sh
```

#enter the below code

```
#!/bin/bash
```

```
sudo apt update -y
```

```
#sudo apt upgrade -y
```

```
wget -O - https://packages.adoptium.net/artifactory/api/gpg/key/public | tee  
/etc/apt/keyrings/adoptium.asc
```

```
echo "deb [signed-by=/etc/apt/keyrings/adoptium.asc]  
https://packages.adoptium.net/artifactory/deb $(awk -F= '/^VERSION_CODENAME/{print$2}'  
/etc/os-release) main" | tee /etc/apt/sources.list.d/adoptium.list
```

```
sudo apt update -y
```

```
sudo apt install temurin-17-jdk -y
```

```
/usr/bin/java --version
```

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \  
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

```
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 -y
```

```
sudo apt-get install jenkins -y
```

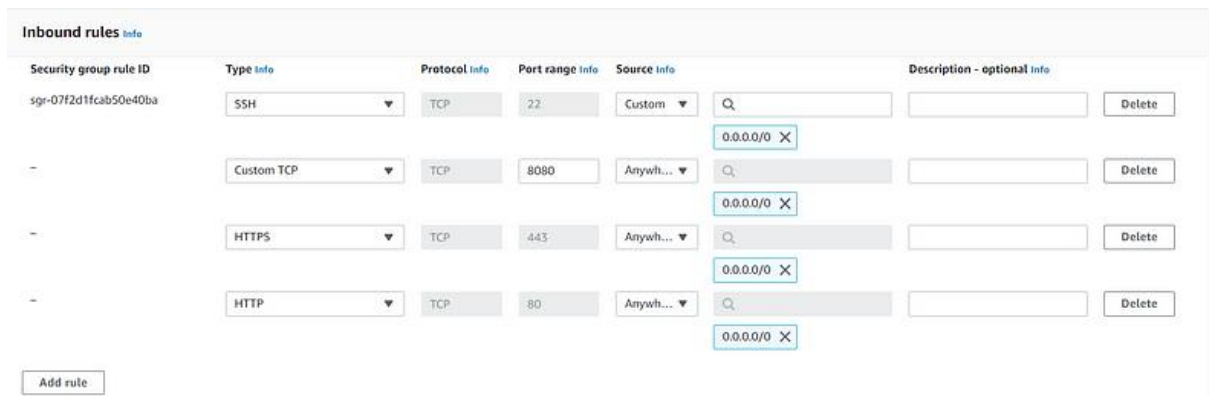
```
sudo systemctl start jenkins
```

```
sudo systemctl status jenkins
```

```
sudo chmod 777 jenkins.sh
```

```
./jenkins.sh
```

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.



The screenshot shows the 'Inbound rules' configuration for an AWS EC2 Security Group. The table lists four rules:

Security group rule ID	Type	Protocol	Port range	Source	Description - optional	Actions
sgr-07f2d1fcab50e40ba	SSH	TCP	22	Custom	0.0.0.0/0	Delete
-	Custom TCP	TCP	8080	Anywh...	0.0.0.0/0	Delete
-	HTTPS	TCP	443	Anywh...	0.0.0.0/0	Delete
-	HTTP	TCP	80	Anywh...	0.0.0.0/0	Delete

An 'Add rule' button is located at the bottom left of the table.

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.

#### Getting Started

## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

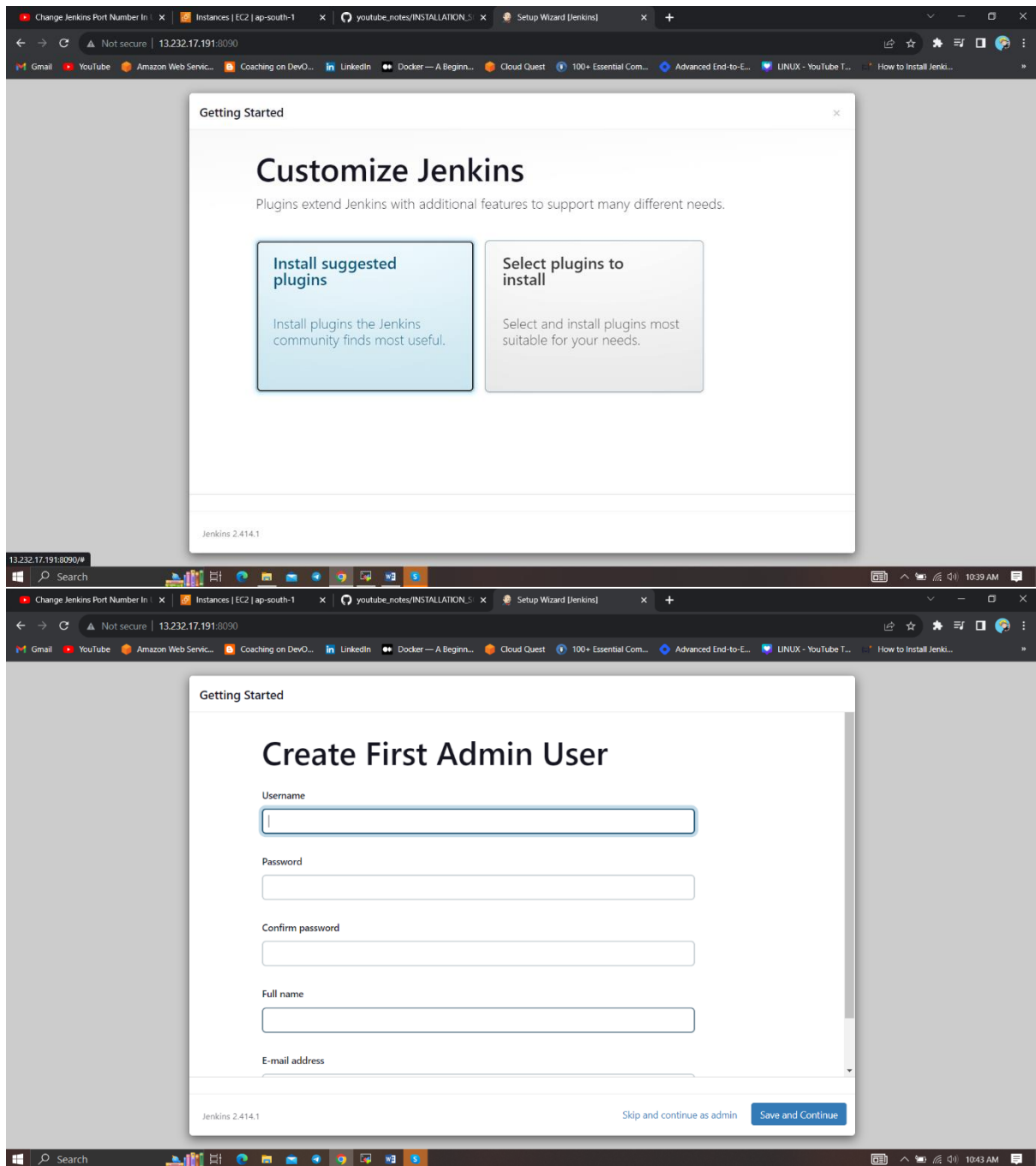
`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

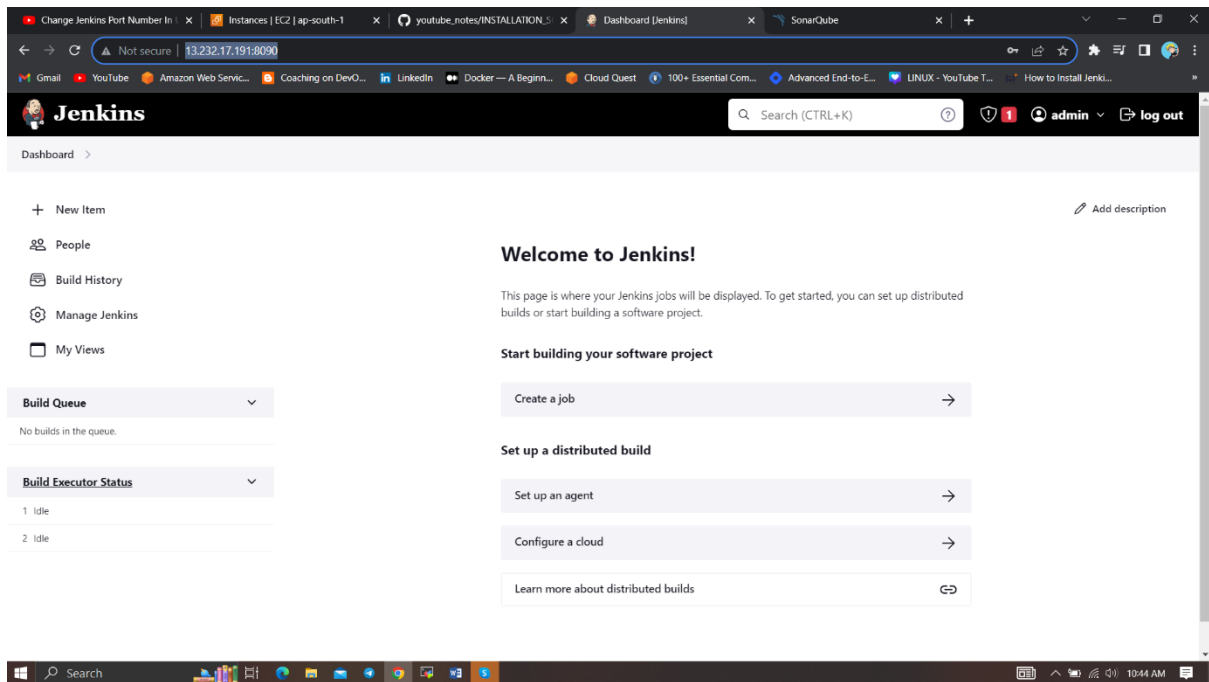
Administrator password

Continue

Jenkins will now get installed and install all the libraries.



Jenkins Getting Started Screen



## 2B — Install Docker

```
sudo apt-get update
```

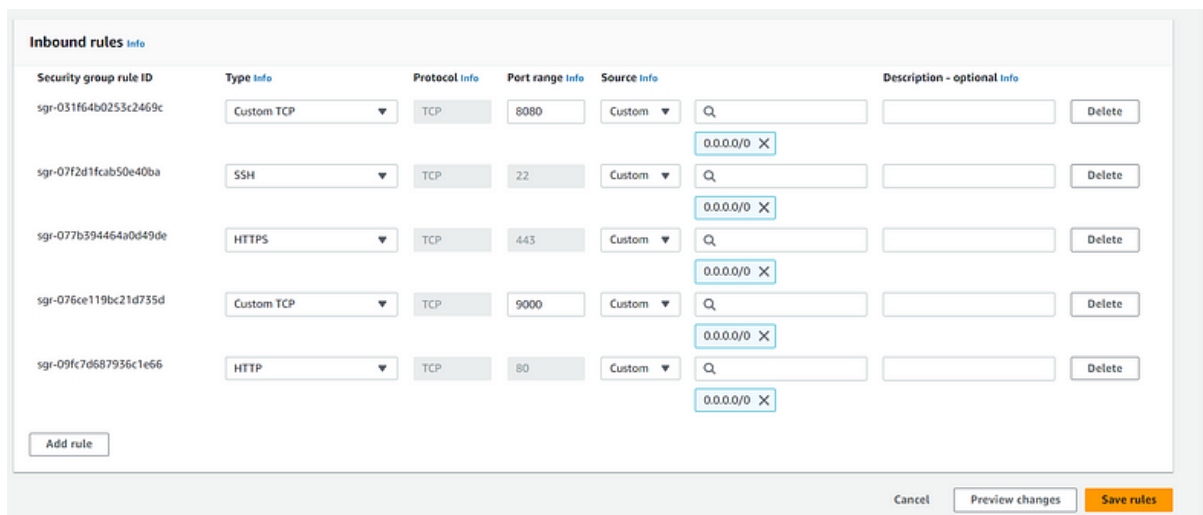
```
sudo apt-get install docker.io -y
```

```
sudo usermod -aG docker $USER
```

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

```
sudo docker ps
```

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



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

```

ubuntu@ip-172-31-42-253:~$ sudo chmod 777 /var/run/docker.sock
ubuntu@ip-172-31-42-253:~$ docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
Unable to find image 'sonarqube:lts-community' locally
lts-community: Pulling from library/sonarqube
44ba2882f8eb: Pull complete
2cabec57fa36: Pull complete
c20481384b6a: Pull complete
bf7b17ee74f8: Pull complete
38617faac714: Pull complete
706f20f58f5e: Pull complete
68a29568c297: Pull complete
Digest: sha256:1a118f8ab960d6c3d4ea0b4455a5a6560654511c88a6816f1603f764d5dcc77c
Status: Downloaded newer image for sonarqube:lts-community
4b60c96bf9ad3d62289436af7f752fdb04993092d0ca3065e2f2e32301b50139
ubuntu@ip-172-31-42-253:~$ docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4b60c96bf9ad	sonarqube:lts-community	"/opt/sonarqube/dock..."	9 seconds ago	Up 5 seconds	0.0.0.0:9000->9000/tcp, :::9000->9000/tcp	sonar

```

ubuntu@ip-172-31-42-253:~$

```

Now our sonarqube is up and running

Enter username and password, click on login and change password

username admin

password admin

## Log in to SonarQube

[Projects](#)
[Issues](#)
[Rules](#)
[Quality Profiles](#)
[Quality Gates](#)
[Administration](#)

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.

Manually

## 2C — Install Trivy

```
sudo apt-get install wget apt-transport-https gnupg lsb-release -y
```

```
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null
```

```
echo "deb [signed-by=/usr/share/keyrings/trivy.gpg] 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 -y

Next, we will log in to Jenkins and start to configure our Pipeline in Jenkins

### Step 3 — Install Plugins like JDK, Sonarqube Scanner, OWASP Dependency Check, Docker.

#### 3A — Install Plugin

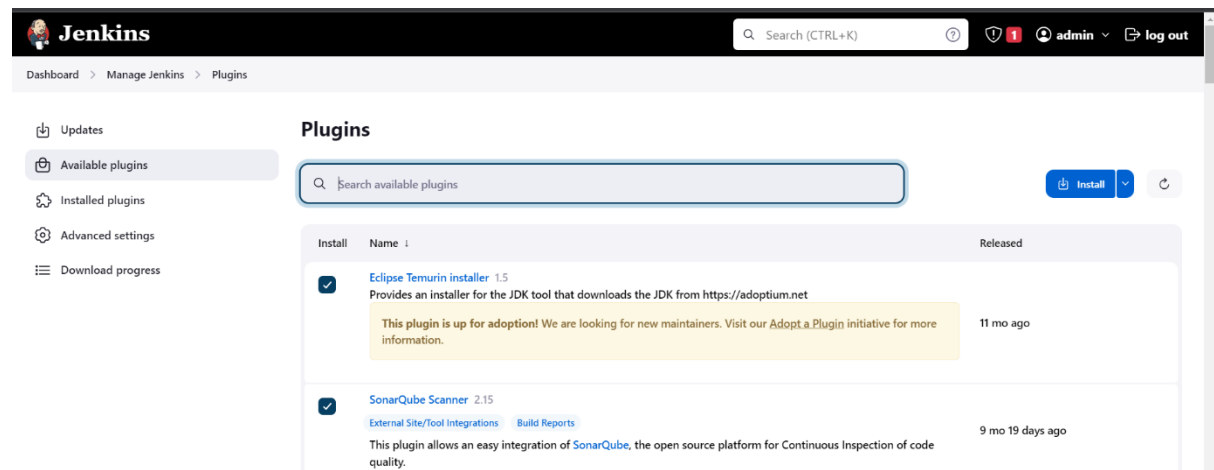
Goto Manage Jenkins → Plugins → Available Plugins →

Install below plugins

1 → Install OWASP (Install without restart)

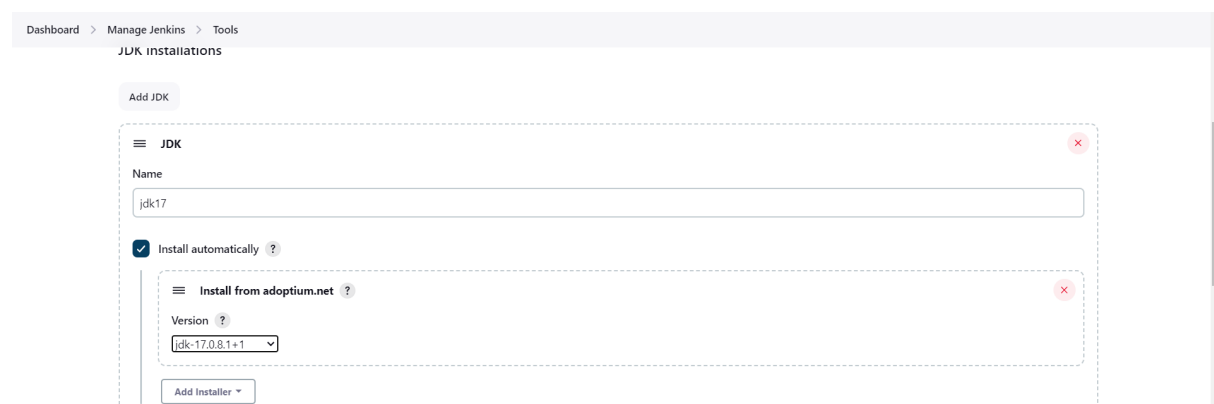
2 → SonarQube Scanner (Install without restart)

3 → 1 → Eclipse Temurin Installer (Install without restart)



#### 3B — Configure Java and Maven in Global Tool Configuration

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



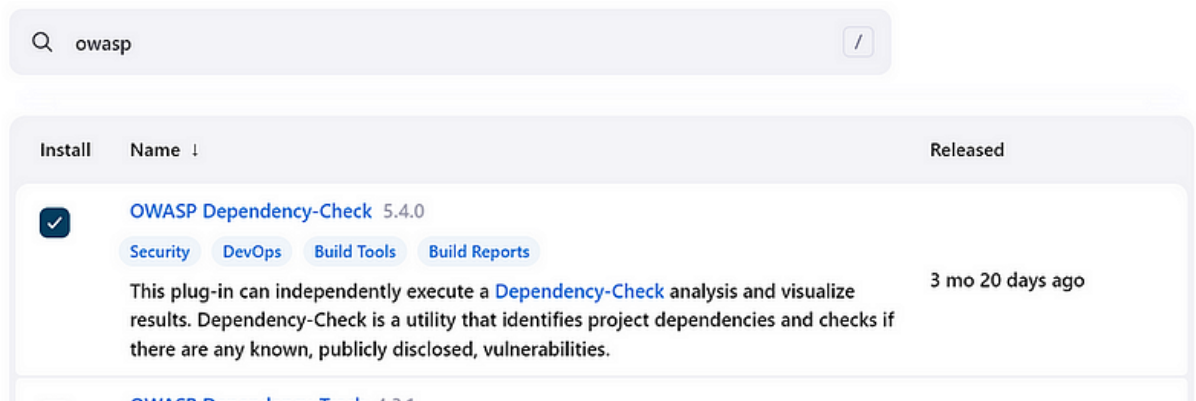
#### 3C — Create a Job

Label it as Dotnet CI-CD, click on Pipeline and OK.

## Step 4 — Install OWASP Dependency Check Plugins

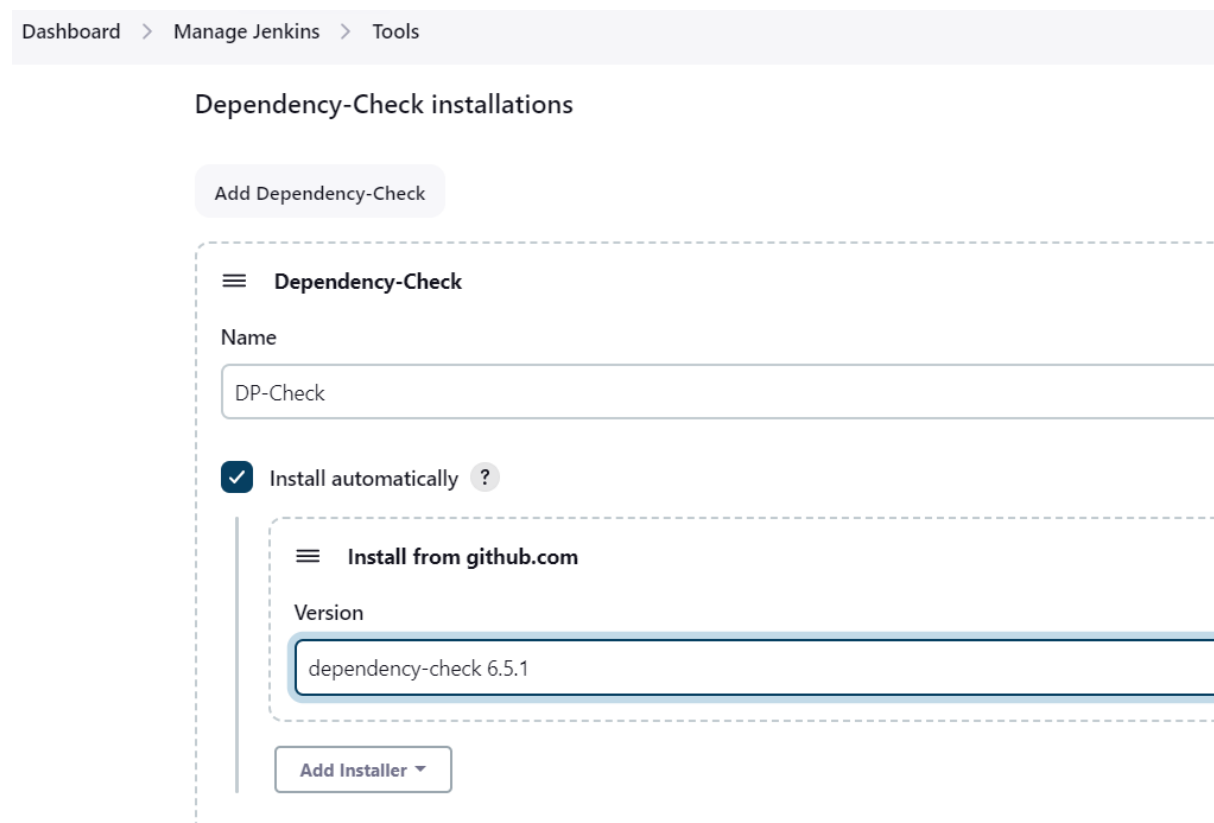
GotoDashboard → Manage Jenkins → Plugins → OWASP Dependency-Check. Click on it and install it without restart.

### Plugins



First, we configured the Plugin and next, we had to configure the Tool

Goto Dashboard → Manage Jenkins → Tools →

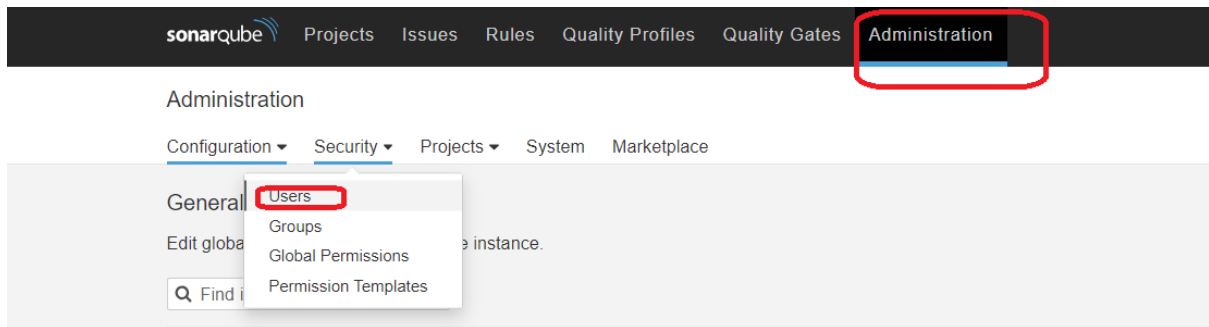


Click on Apply and Save here.

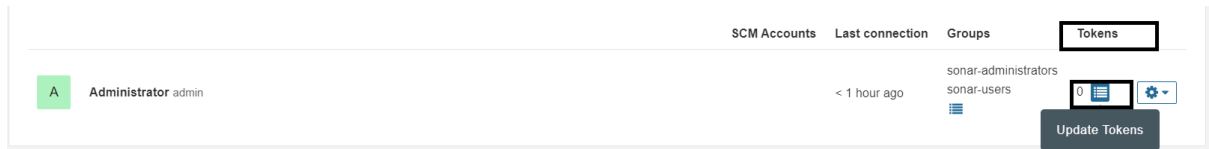
## Step 5 — Configure Sonar Server in Manage Jenkins

Grab the Public IP Address of your EC2 Instance, Sonarqube works on Port 9000, sp <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

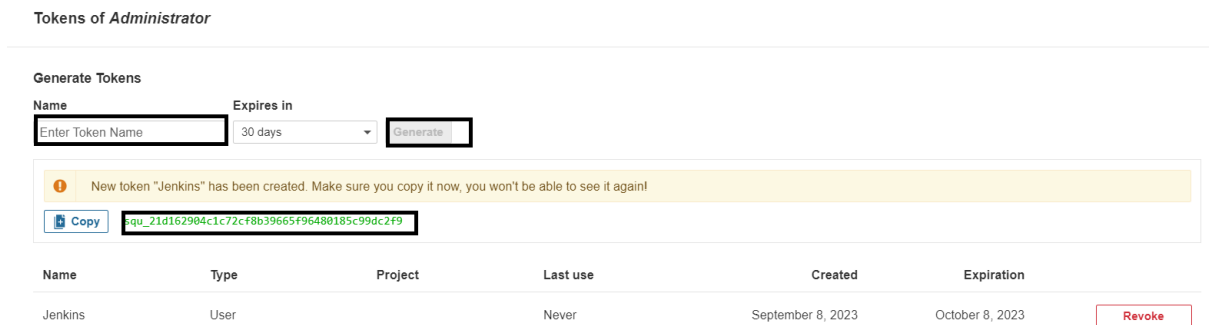




Click on Update Token

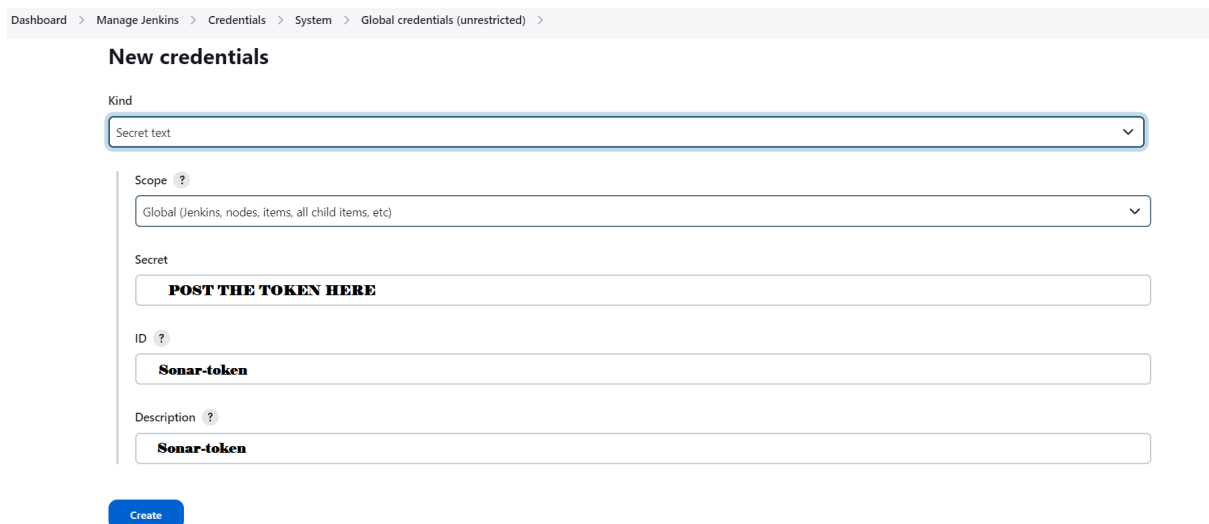


Create a token with a name and generate




Copy this Token

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



You will this page once you click on create

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
 <a href="#">Sonar-token</a>	sonar	Secret text	sonar

Now, go to Dashboard → Manage Jenkins → Configure System

Dashboard > Manage Jenkins > System

### SonarQube servers

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

☐ **Environment variables** Enable injection of SonarQube server configuration as build environment variables

**SonarQube installations**  
List of SonarQube installations

**Name**

**Server URL**  
Default is http://localhost:9000

**Server authentication token**  
SonarQube authentication token. Mandatory when anonymous access is disabled.

[Add](#)

[Save](#) [Apply](#)

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.

Dashboard > Manage Jenkins > Tools

### SonarQube Scanner installations

[Add SonarQube Scanner](#)

**SonarQube Scanner**

**Name**

☒ **Install automatically** ?

**Install from Maven Central**

**Version**

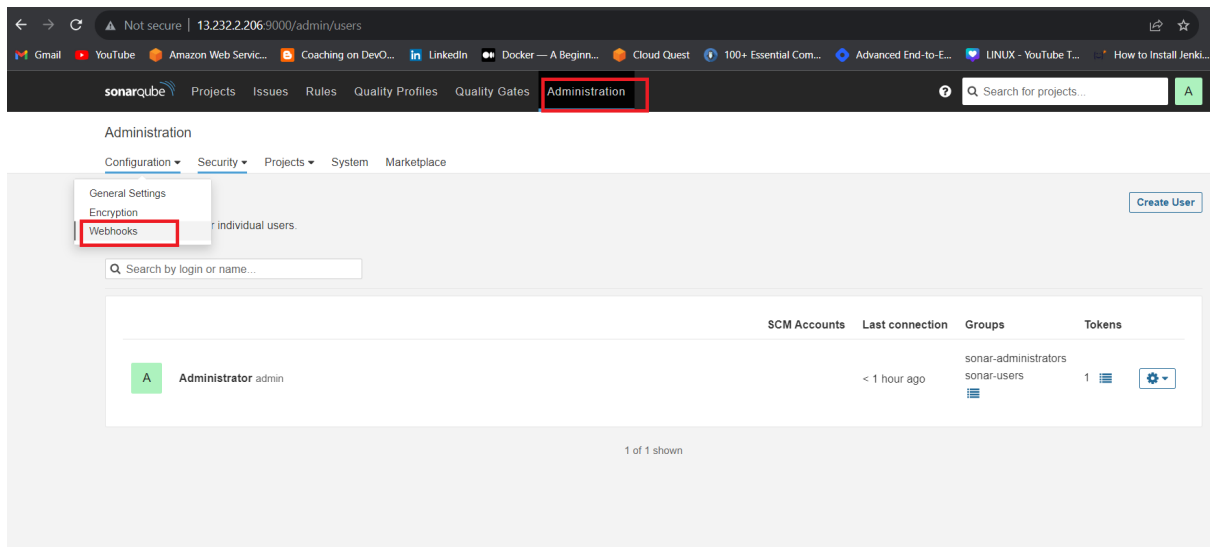
[Add Installer](#)

[Add SonarQube Scanner](#)

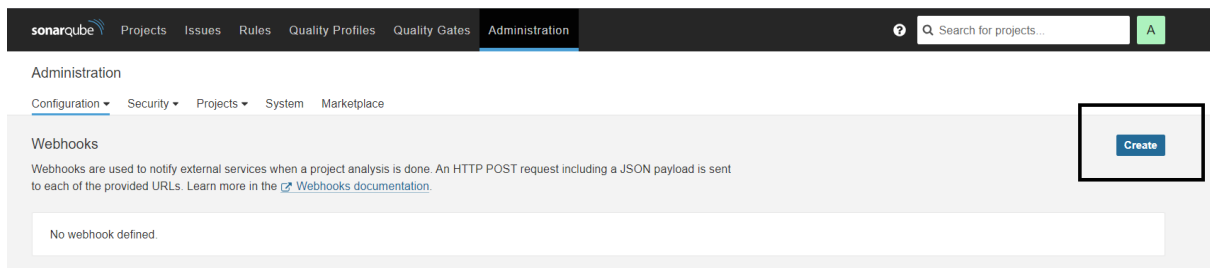
[Save](#) [Apply](#)

In the Sonarqube Dashboard add a quality gate also

Administration→ Configuration→Webhooks



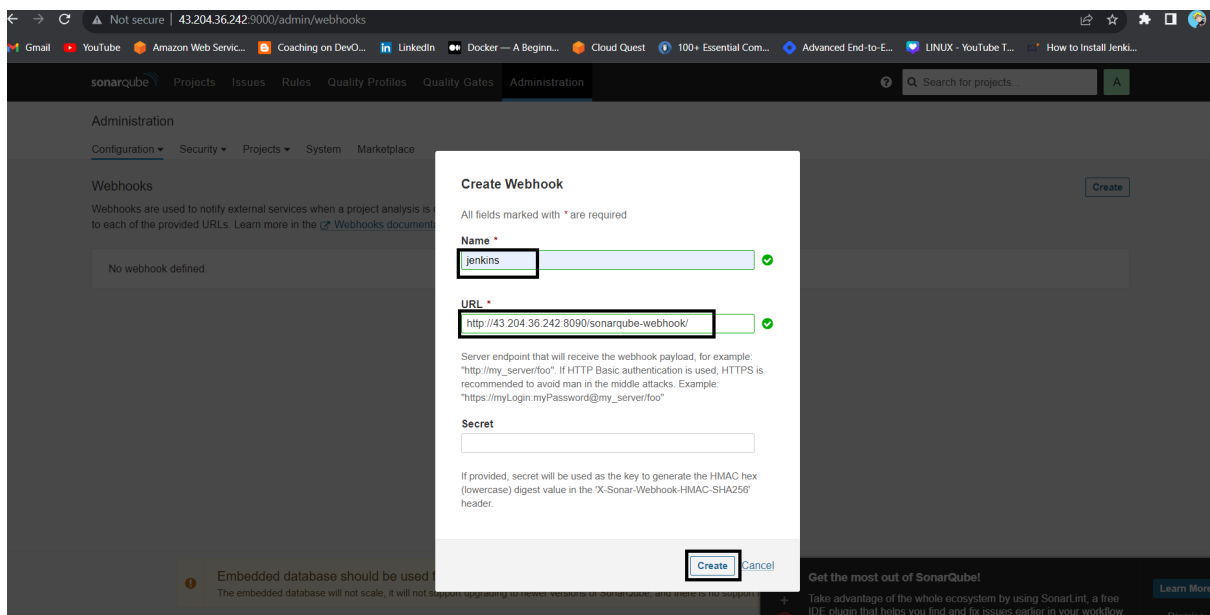
Click on Create



Add details

#in url section of quality gate

http://jenkins-public-ip:8080/sonarqube-webhook/



Let's go to our Pipeline and add the below code 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/Aj7Ay/DotNet-monitoring.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'
            }
        }
    }
}
```

```

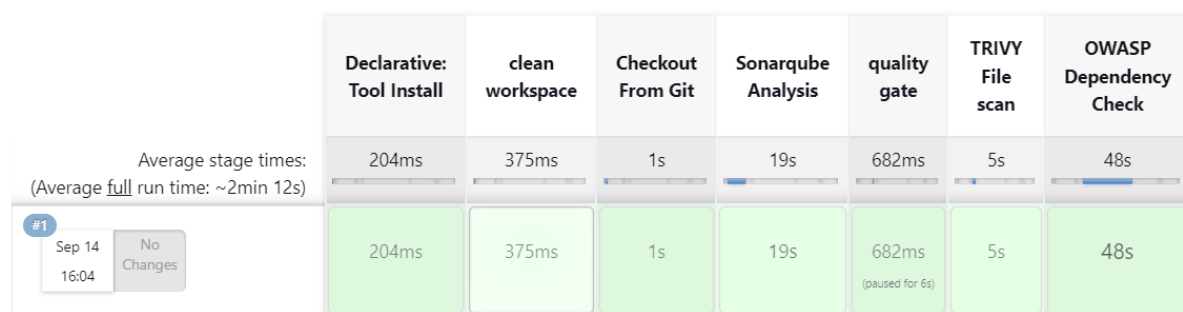
    }
  }
  stage("TRIVY File scan"){
    steps{
      sh "trivy fs . > trivy-fs_report.txt"
    }
  }
  stage("OWASP Dependency Check"){
    steps{
      dependencyCheck additionalArguments: '--scan ./ --format XML ', odciInstallation: 'DP-
Check'

      dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
    }
  }
}

```

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


## Stage View



## SonarQube Quality Gate

Python-Webapp **Passed**

server-side processing: **Success**

 Latest Dependency-Check

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

☆ **Dotnet-Webapp** **Passed** Last analysis: 35 seconds ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
15 <span style="color: orange;">C</span>	0 <span style="color: green;">A</span>	0.0% <span style="color: red;">E</span>	0 <span style="color: green;">A</span>	—	0.0% <span style="color: green;">C</span>	522 <span style="color: blue;">xs</span> HTML, CSS

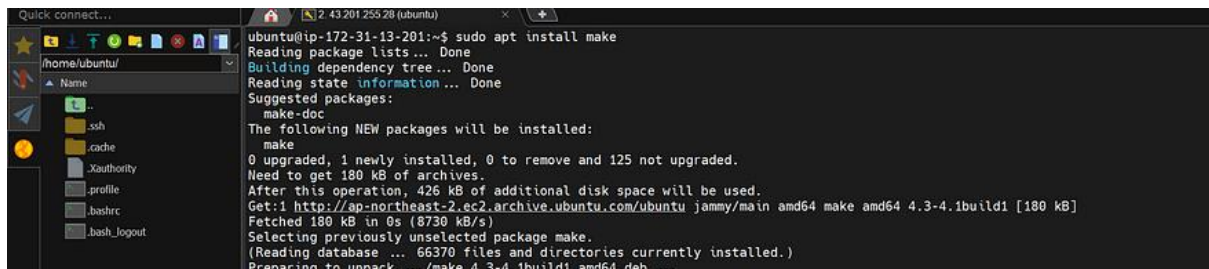
You can see the report has been generated and the status shows as passed. You can see that there are 522 lines. To see a detailed report, you can go to issues.

## Step 6 — we have to install make package

sudo apt install make

# to check version install or not

make -v



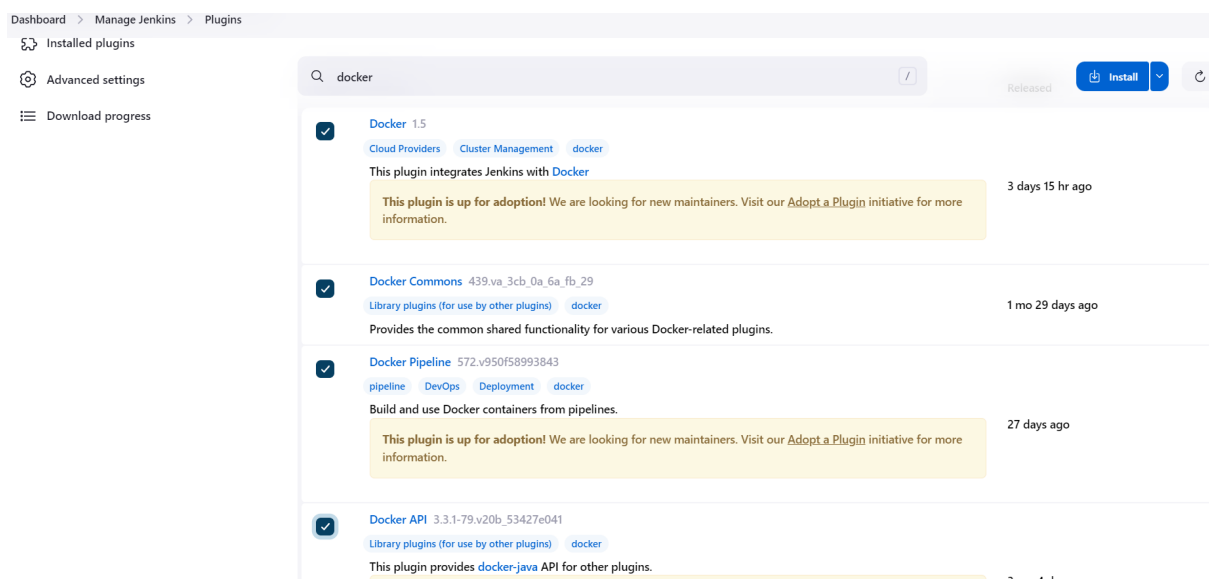
```
ubuntu@ip-172-31-13-201:~$ sudo apt install make
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Suggested packages:
  make-doc
The following NEW packages will be installed:
  make
0 upgraded, 1 newly installed, 0 to remove and 125 not upgraded.
Need to get 180 kB of archives.
After this operation, 426 kB of additional disk space will be used.
Get:1 http://ap-northeast-2.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 make amd64 4.3-4.1build1 [180 kB]
Fetched 180 kB in 0s (8730 kB/s)
Selecting previously unselected package make.
(Reading database ... 66370 files and directories currently installed.)
Preparing to unpack .../make_4.3-4.1build1_amd64.deb ...
```

## Step 7 — 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 →

Dashboard > Manage Jenkins > Tools

### Docker installations

Add Docker

Docker

Name

docker

☒ Install automatically ?

Download from docker.com

Docker version ?

latest

Add Installer ▾

## Add DockerHub Username and Password under Global Credentials

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Kind

Username with password ▾

Scope ?

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

Username ?

sevenajay

☐ Treat username as secret ?

Password ?

.....

ID ?

docker

Description ?

docker

Create

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

```

DotNet-monitoring makefile
Code Blame 70 lines (55 loc) · 2.63 KB
22 @grep -E '^[a-zA-Z_-]+:.*?## .*$$' $(MAKEFILE_LIST) | awk 'BEGIN {FS = ":.*?## "}; {printf "\033[36m%-20s\033[0m %s\n", $$1, $$2}'
23
24 lint: ## 🐞 Lint & format, will not fix but sets exit code on error
25 @dotnet format --help > /dev/null 2> /dev/null || dotnet tool install --global dotnet-format
26 dotnet format --verbosity diag ./src
27
28 image: ## 🚀 Build container image from Dockerfile
29 docker build . --file build/Dockerfile \
30 --tag $(IMAGE_REG)/$(IMAGE_REPO):$(IMAGE_TAG)
31
32 push: ## 📦 Push container image to registry
33 docker push $(IMAGE_REG)/$(IMAGE_REPO):$(IMAGE_TAG)

```

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 to Pipeline Script

```

stage("Docker Build & tag"){
    steps{

```

```



    script{
        withDockerRegistry(credentialsId: 'docker', toolName: 'docker'){
            sh "make image"
        }
    }
}

stage("TRIVY"){
    steps{
        sh "trivy image sevenajay/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

 <b>sevenajay / dotnet-monitoring</b> <b>Description</b> This repository does not have a description 	<b>Docker commands</b> <a data-bbox="1189 1579 1284 1612" href="#">Public View</a> To push a new tag to this repository: <pre>docker push sevenajay/dotnet-monitoring:tagname</pre>
---	---

stage view



Declarative: Tool Install	clean workspace	Checkout From Git	Sonarqube Analysis	quality gate	TRIVY File scan	OWASP Dependency Check	Docker Build & tag	TRIVY	Docker Push
204ms	375ms	1s	19s	682ms	5s	48s	24s	3s	15s
204ms	375ms	1s	19s	682ms (paused for 6s)	5s	48s	24s	3s	15s

## Step 8 — Deploy the image using Docker

Add this stage to your pipeline syntax

```
stage("Deploy to container"){
    steps{
        sh "docker run -d --name dotnet -p 5000:5000 sevenajay/dotnet-monitoring:latest"
    }
}
```

You will see the Stage View like this,

Declarative: Tool Install	clean workspace	Checkout From Git	Sonarqube Analysis	quality gate	TRIVY File scan	OWASP Dependency Check	Docker Build & tag	TRIVY	Docker Push	Deploy to container
204ms	375ms	1s	19s	682ms	5s	48s	24s	3s	15s	1s
204ms	375ms	1s	19s	682ms (paused for 6s)	5s	48s	24s	3s	15s	1s

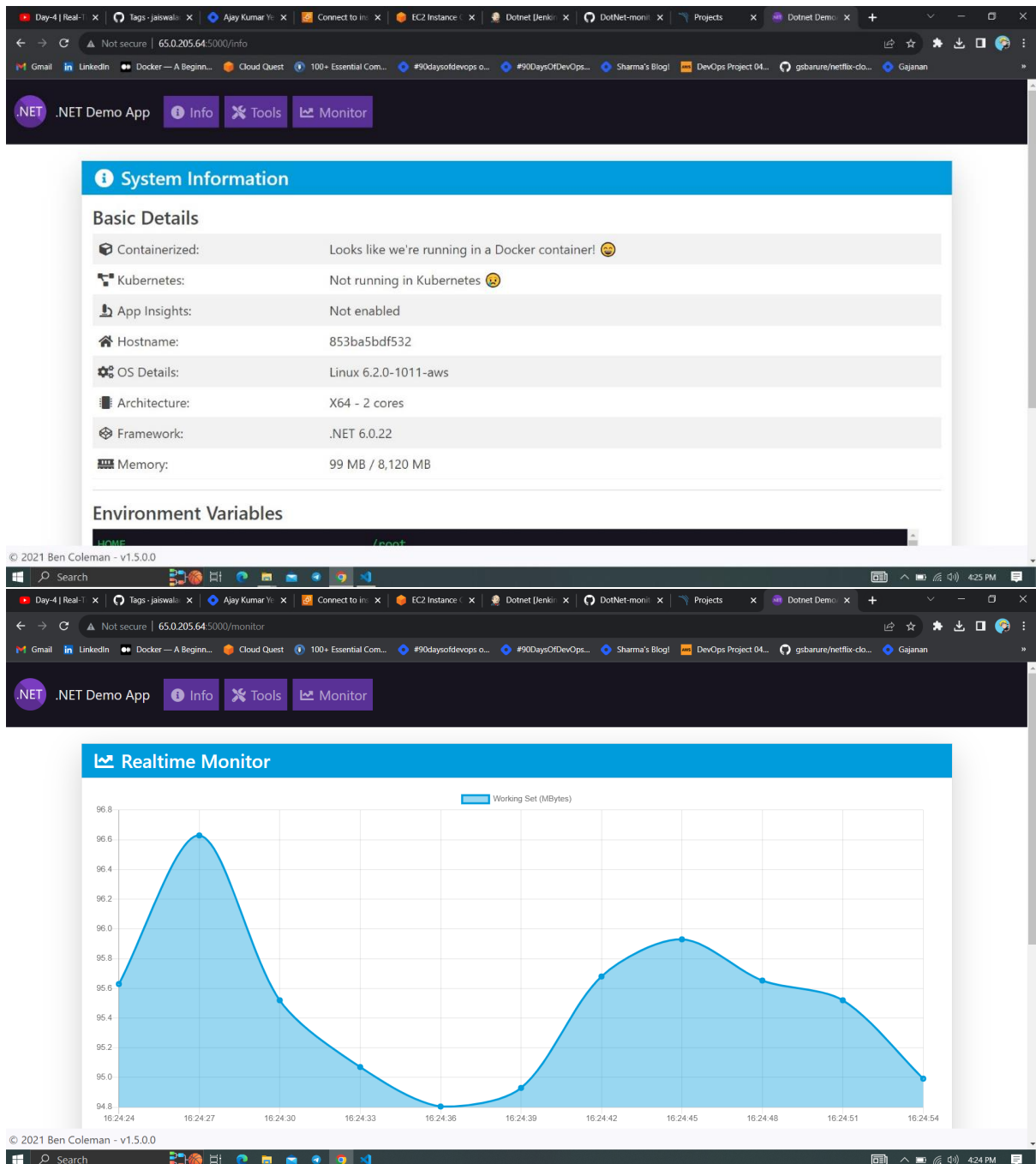
(Add port 5000 to Security Group)

sgr-076ce119bc21d735d	Custom TCP	TCP	9000	Custom	Q		Delete
					0.0.0.0/0	X	
sgr-09fc7d687936c1e66	HTTP	TCP	80	Custom	Q		Delete
					0.0.0.0/0	X	
-	Custom TCP	TCP	5000	Anywh...	Q		Delete
					0.0.0.0/0	X	
Add rule							

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

<public-ip of jenkins:5000>

## Step 9 — Access the Real World Application



## Step 10 —Kubernetes setup

**Take-Two Ubuntu 20.04 instances one for k8s master and the other one for worker.**

Install Kubectl on Jenkins machine also.

### Kubectl on Jenkins to be installed

```
sudo apt update
```

```
sudo apt install curl
```

```
curl -LO https://dl.k8s.io/release/$(curl -L -s  
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
```

```
sudo install -o root -g root -m 0755 kubect1 /usr/local/bin/kubect1
```

```
kubect1 version --client
```

### **Part 1 ———-Master Node———-**

```
sudo su
```

```
hostname master
```

```
bash
```

```
clear
```

### **———-Worker Node———-**

```
sudo su
```

```
hostname worker
```

```
bash
```

```
clear
```

### **Part 2 ———-Both Master & Node ———-**

```
sudo apt-get update
```

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

```
sudo usermod -aG docker Ubuntu
```

```
newgrp docker
```

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

```
sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

```
sudo tee /etc/apt/sources.list.d/kubernetes.list <<EOF
```

```
deb https://apt.kubernetes.io/ kubernetes-xenial main
```

```
EOF
```

```
sudo apt-get update
```

```
sudo apt-get install -y kubelet kubeadm kubect1
```

```
sudo snap install kube-apiserver
```

### **Part 3 ———- Master ———-**

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

```
# in case your in root exit from it and run below commands
```

```
mkdir -p $HOME/.kube
```

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

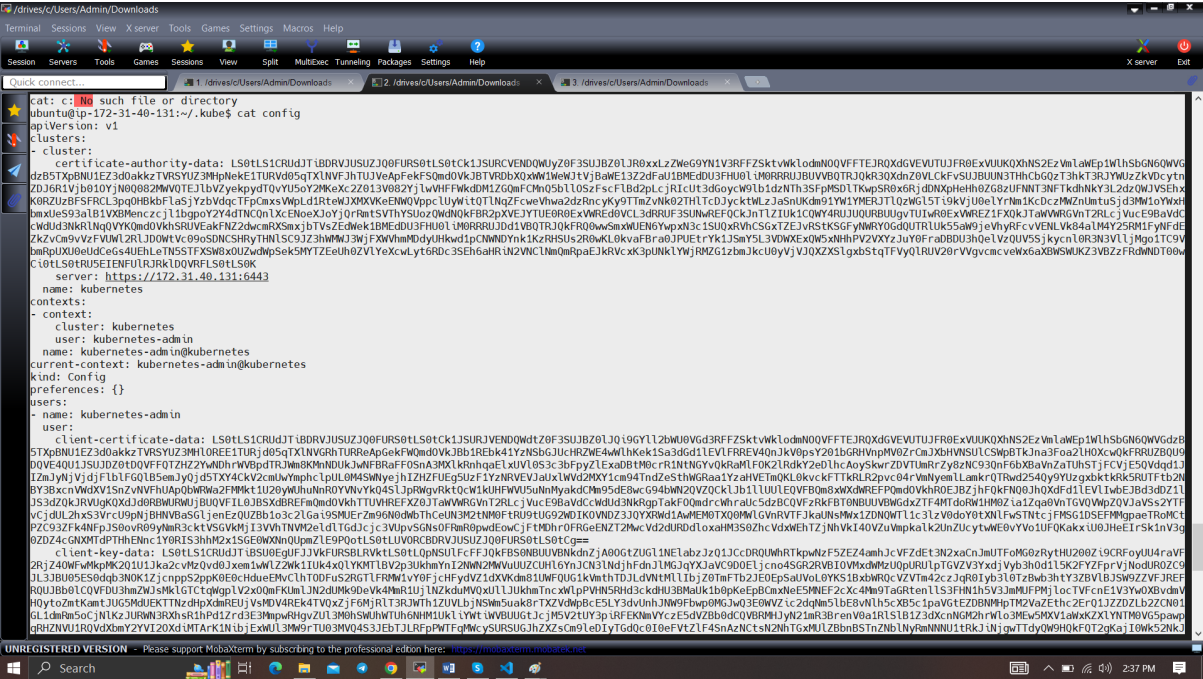
```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

-----Worker Node-----

sudo kubeadm join <master-node-ip>:<master-node-port> --token <token> --discovery-token-ca-cert-hash <hash>

Copy the config file to Jenkins master or the local file manager and save it



copy it and save it in documents or another folder save it as secret-file.txt

Install Kubernetes Plugin, Once it's installed successfully

Dashboard > Manage Jenkins > Plugins

Updates

Available plugins

Installed plugins

Advanced settings

Download progress

Plugins

Kuber

Install

Install	Name	Released
<input checked="" type="checkbox"/>	<div>Kubernetes Credentials 0.11</div> <div>kubernetes <a href="#">credentials</a></div> <div>Common classes for Kubernetes credentials</div>	9 days 16 hr ago
<input checked="" type="checkbox"/>	<div>Kubernetes Client API 6.8.1-224.vd388fca_4db_3b</div> <div>kubernetes <a href="#">Library plugins (for use by other plugins)</a></div> <div>Kubernetes Client API plugin for use by other Jenkins plugins.</div>	9 days 17 hr ago
<input checked="" type="checkbox"/>	<div>Kubernetes 4029.v5712230ccb_f8</div> <div><a href="#">Cloud Providers</a> <a href="#">Cluster Management</a> <a href="#">kubernetes</a> <a href="#">Agent Management</a></div> <div>This plugin integrates Jenkins with Kubernetes</div>	9 days 15 hr ago
<input checked="" type="checkbox"/>	<div>Kubernetes CLI 1.12.1</div> <div>kubernetes</div> <div>Configure kubectl for Kubernetes</div>	8 days 22 hr ago

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

## New credentials

Kind  
Secret file

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

File  
Choose File Secret File.txt

ID ?  
k8s

Description ?  
k8s

Create

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.

ns

Search (CTRL+K)

admin log out

## Pipeline Dotnet

Add description

Disable Project

### Stage View

	Declarative: Tool Install	clean workspace	Checkout From Git	Sonarqube Analysis	quality gate	TRIVY File scan	OWASP Dependency Check	Docker Build & tag	TRIVY	Docker Push	Deploy to container	Deploy to k8s
Average stage times:	14ms	45ms	974ms	17s	339ms	2s	3min 17s	2s	2s	6s	547ms	929ms
(Average full run time: ~3min 59s)												
PS	14ms	45ms	974ms	17s	339ms (skipped for 6s)	2s	3min 17s	2s	2s	6s	547ms	929ms

### Permalinks

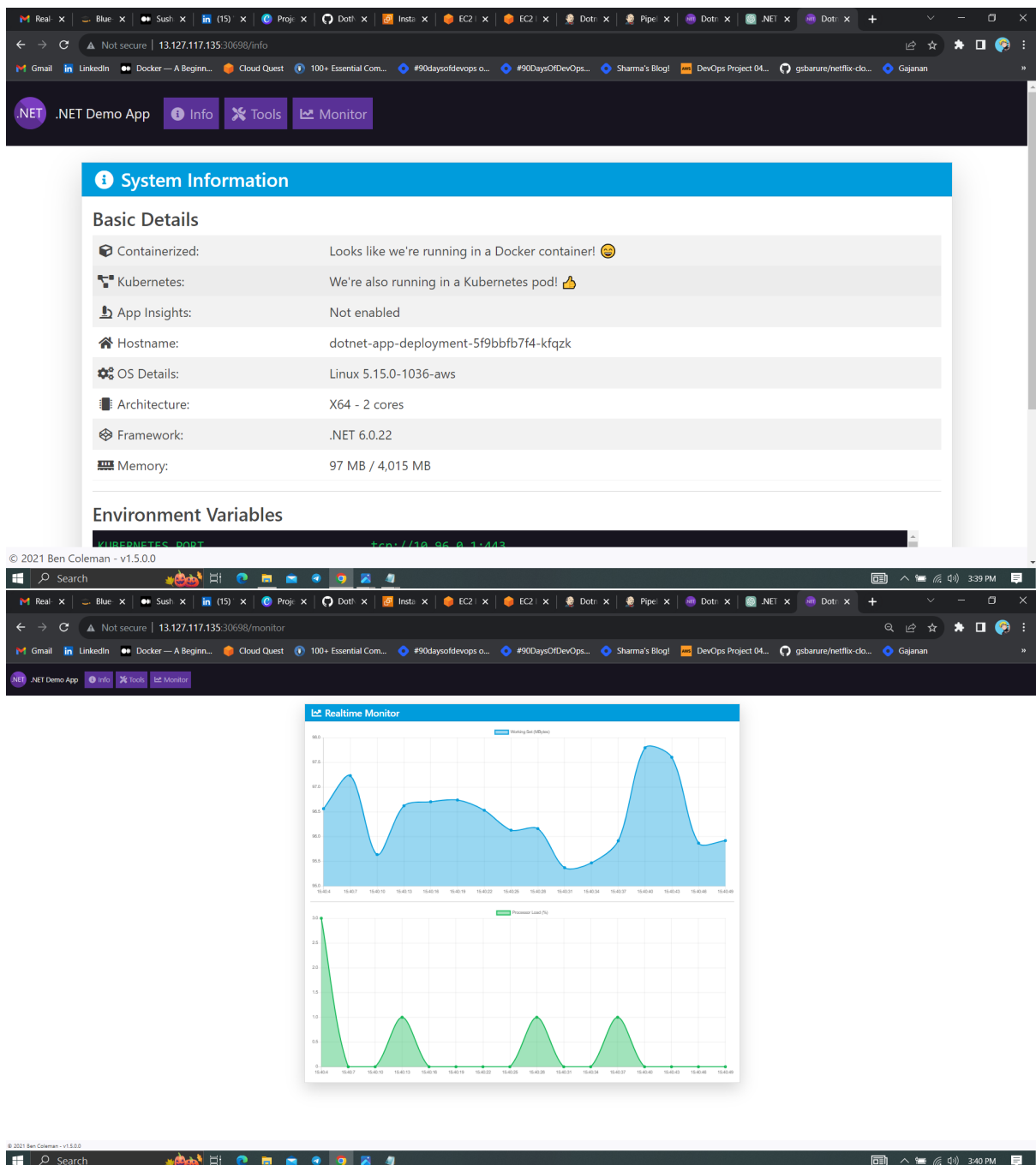
- Last build (#4), 10 min ago
- Last stable build (#4), 10 min ago
- Last successful build (#4), 10 min ago

Output

kubectl get svc

#copy service port

<worker-ip:svc port>



## Step 11 — Terminate the AWS EC2 Instance

Lastly, do not forget to terminate the AWS EC2 Instance.

complete pipeline

```
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/Aj7Ay/DotNet-monitoring.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 sevenajay/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 sevenajay/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'  
        }  
      }  
    }  
  }  
}
```