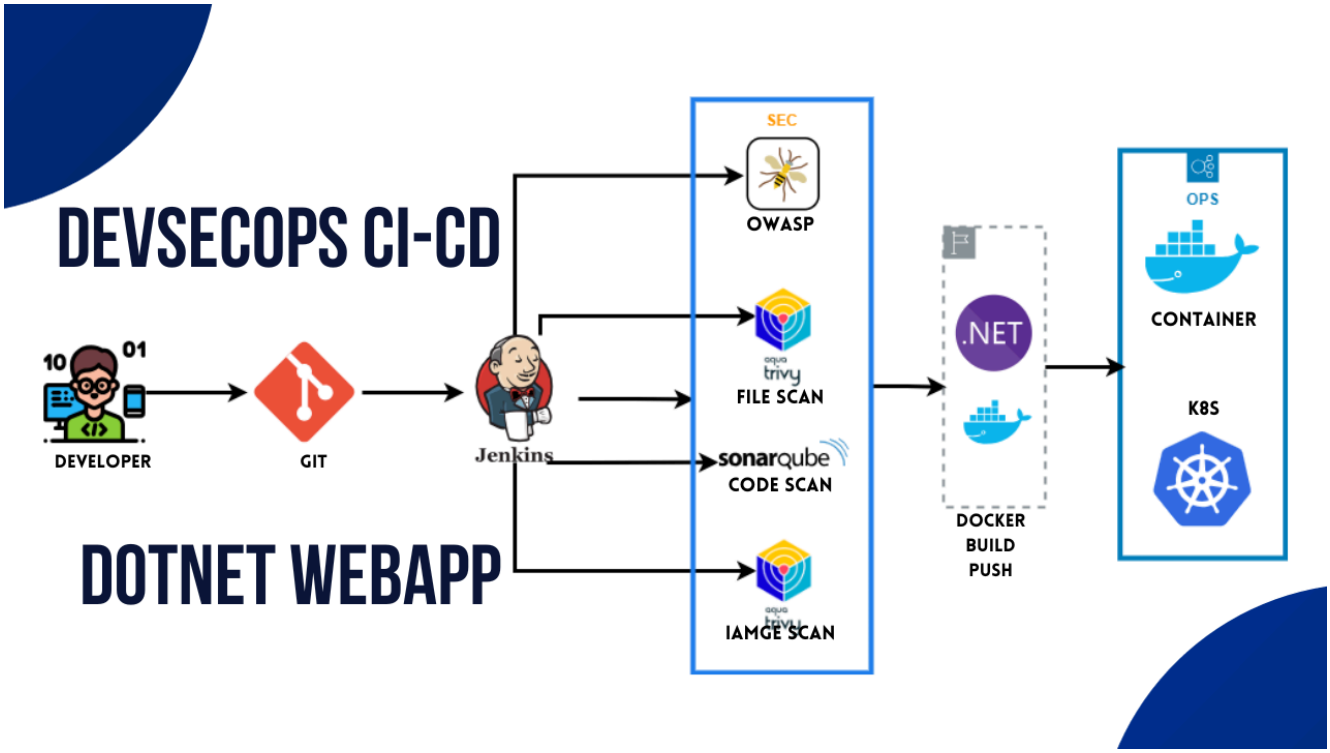
DevOps

Jenkins CI/CD | DOTNET webapp DevSecOps Project

mrcloudbook.com · 8 January 2024



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.

Mr Cloud Book

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

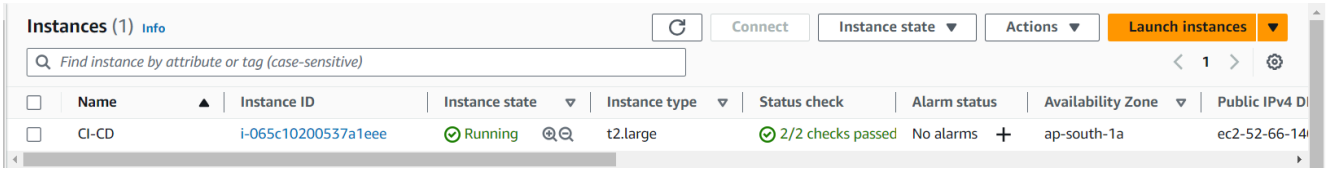
Step 7 – Docker Image Build and Push

```
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.



Step 2 – Install Jenkins, Docker and Trivy

2A – To Install Jenkins

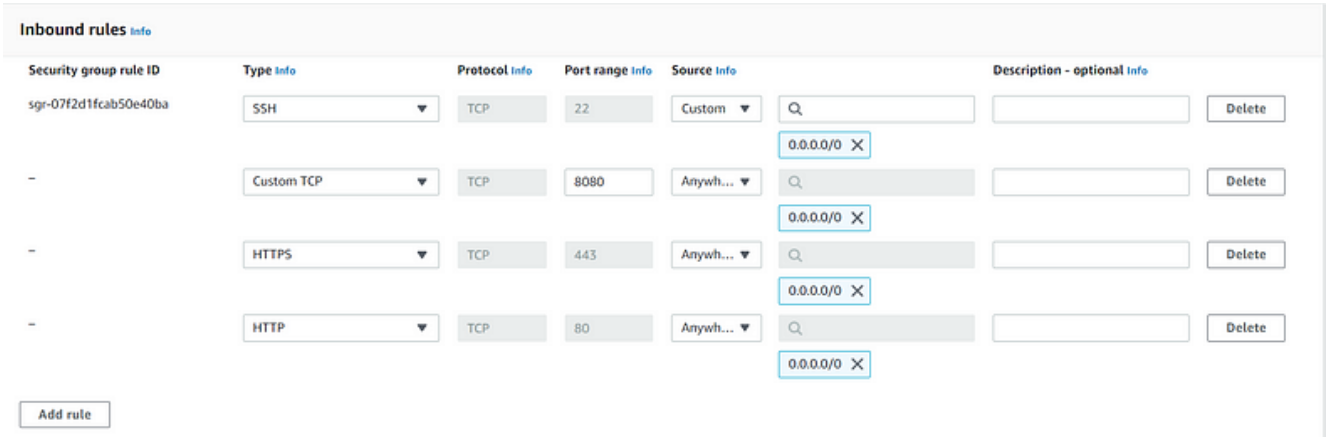
Connect to your console, and enter these commands to Install Jenkins

```
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
echo "deb [signed-by=/etc/apt/keyrings/adoptium.asc] https://packages.ad
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 | su
                  /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 te
                                /etc/apt/sources.list.d/jenkins.list > /de
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
```

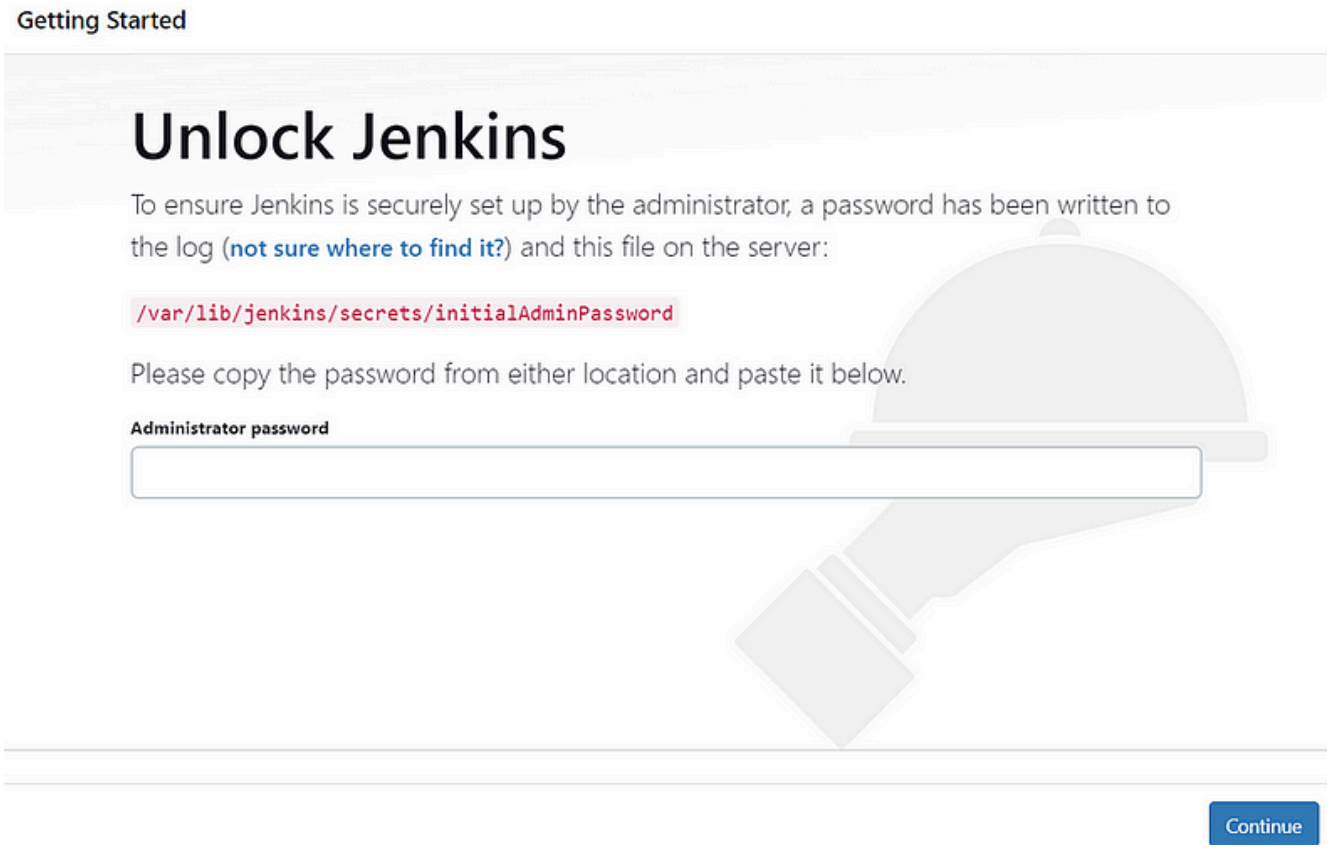
Once Jenkins is installed, you will need to go to your AWS EC2 Security Group and open Inbound Port 8080, since Jenkins works on Port 8080.



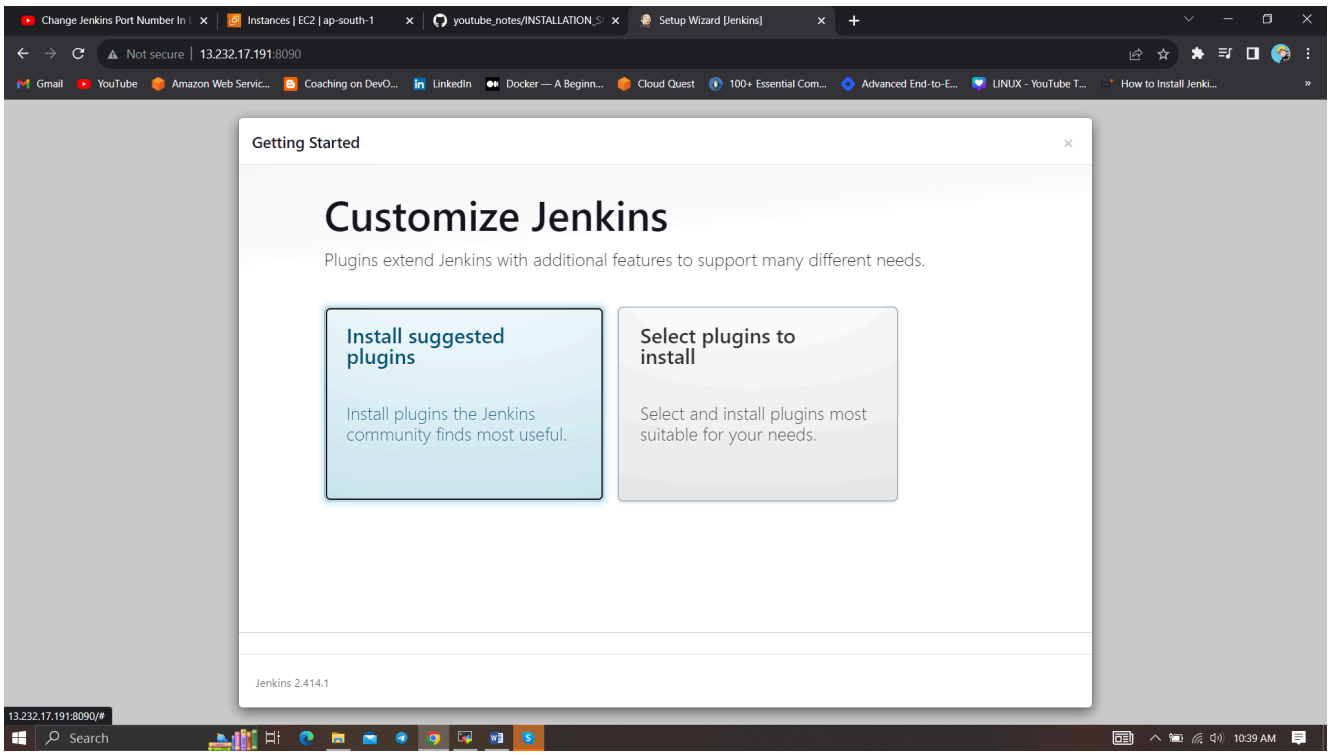
Now, grab your Public IP Address

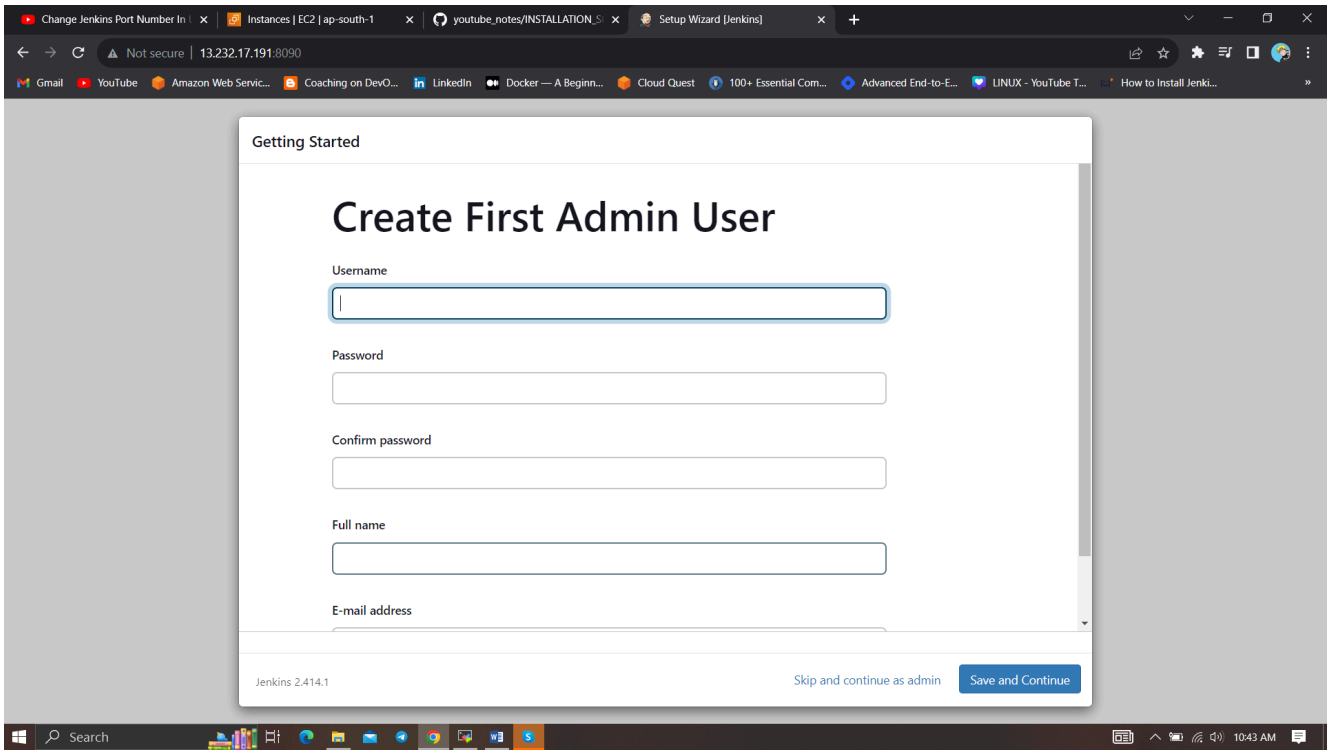
```
EC2 Public IP Address:8080
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

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

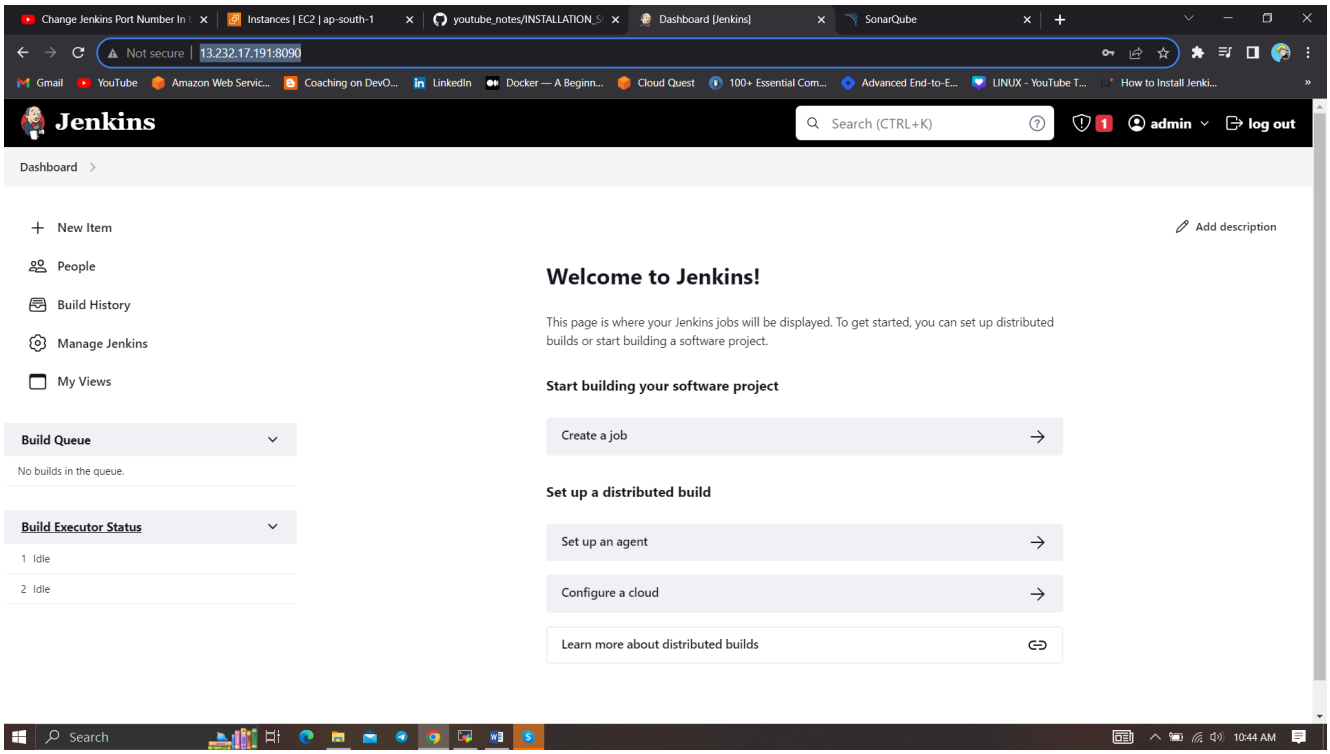


Jenkins will now get installed and install all the libraries.





Jenkins Getting Started Screen

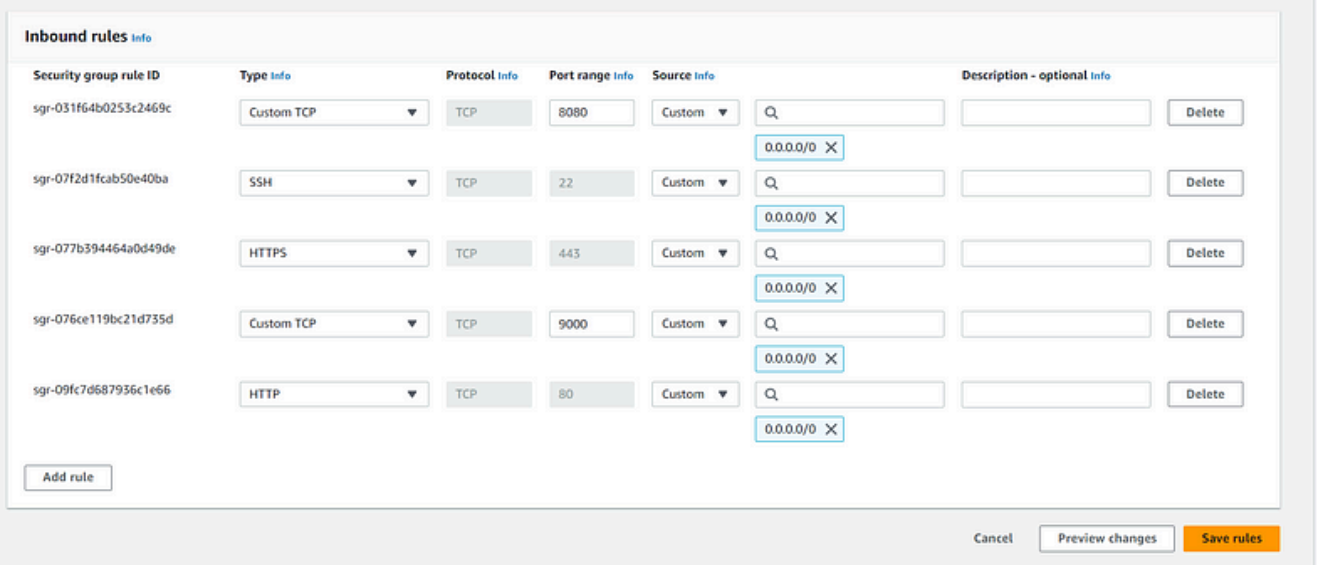


2B – Install Docker

```
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)



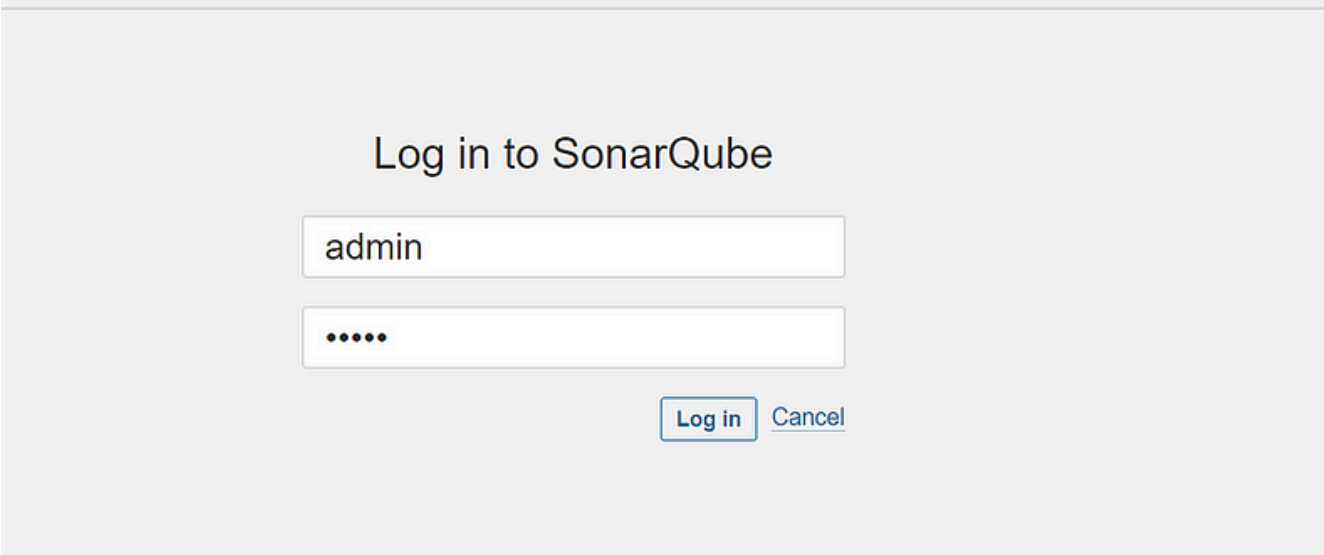

```
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
```

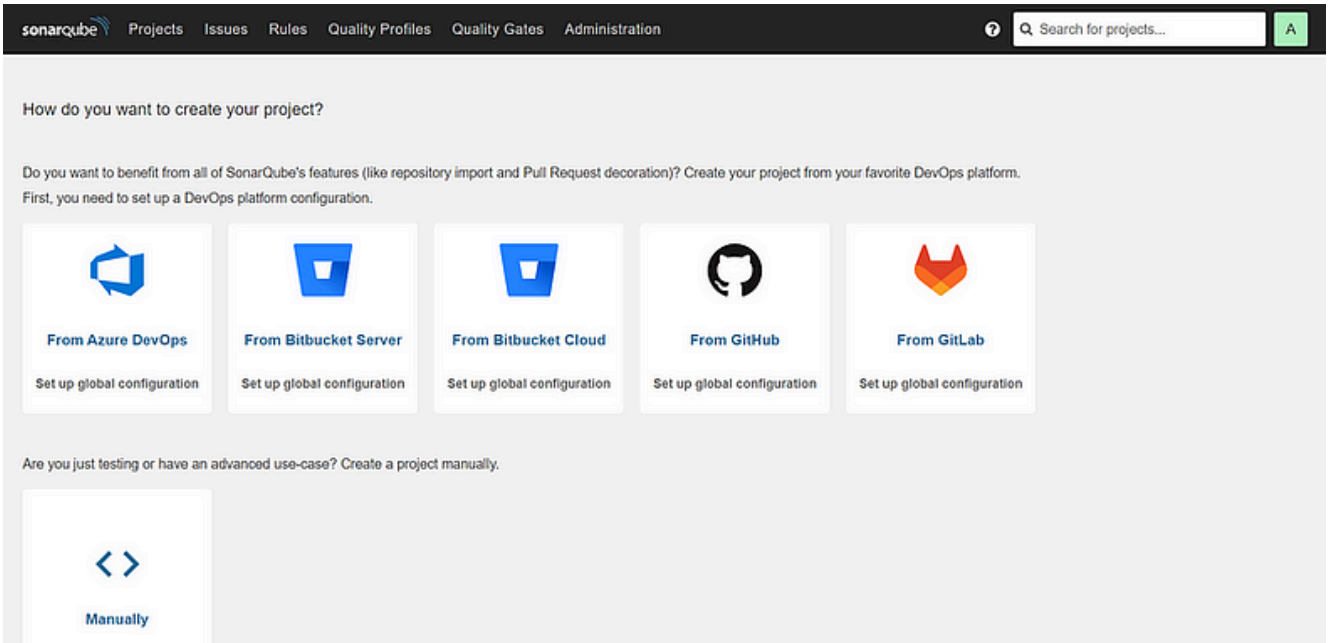
```
ubuntu@ip-172-31-42-253:~$ sudo chmod 777 /var/run/docker.sock
ubuntu@ip-172-31-42-253:~$ docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
Unable to find image 'sonarqube:lts-community' locally
lts-community: Pulling from library/sonarqube
44ba2882f8eb: Pull complete
2cabec57fa36: Pull complete
c20481384b6a: Pull complete
bf7b17ee74f8: Pull complete
38617faac714: Pull complete
706f20f58f5e: Pull complete
65a29568c257: Pull complete
Digest: sha256:1a118f8ab960d6c3d4ea8b4455a5a6560654511c88a6816f1603f764d5dcc77c
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
```





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 | gp
echo "deb [signed-by=/usr/share/keyrings/trivy.gpg] https://aquasecurity
```

```
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

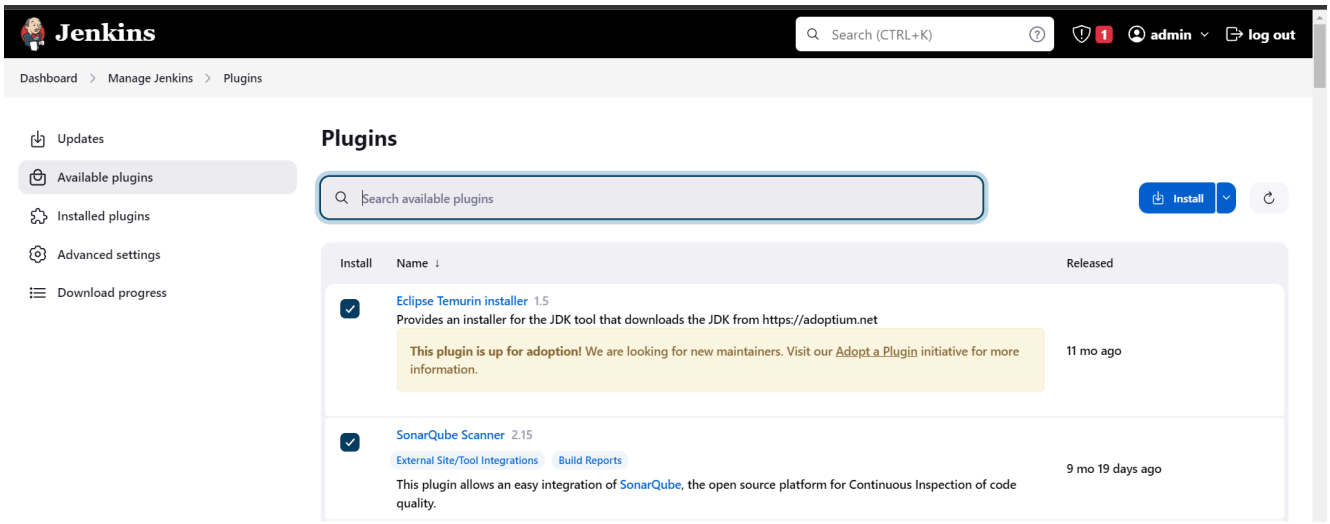
Goto Manage Jenkins →Plugins → Available Plugins →

Install below plugins

1 → Install OWASP ((Install without restart)

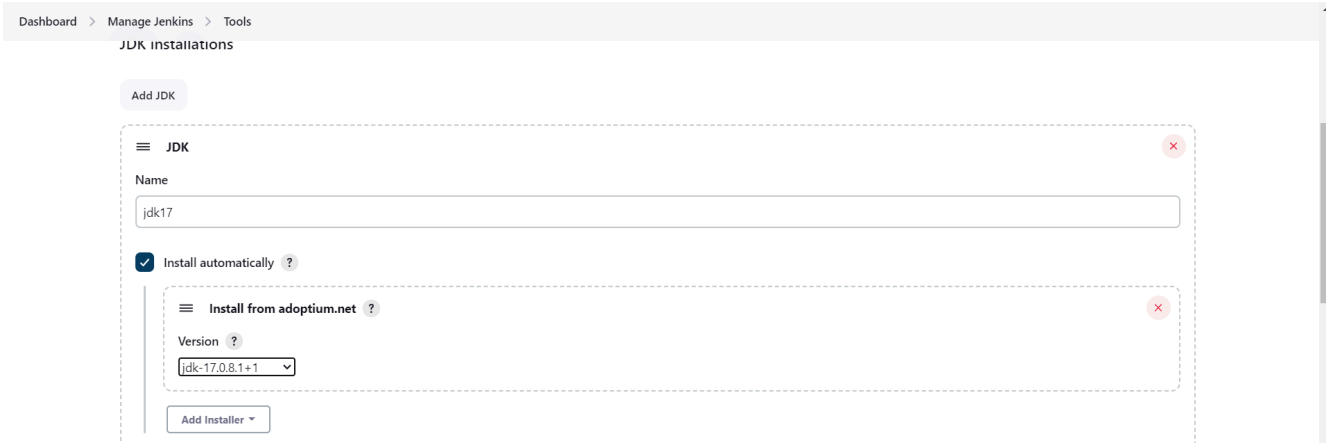
2 → SonarQube Scanner (Install without restart)

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

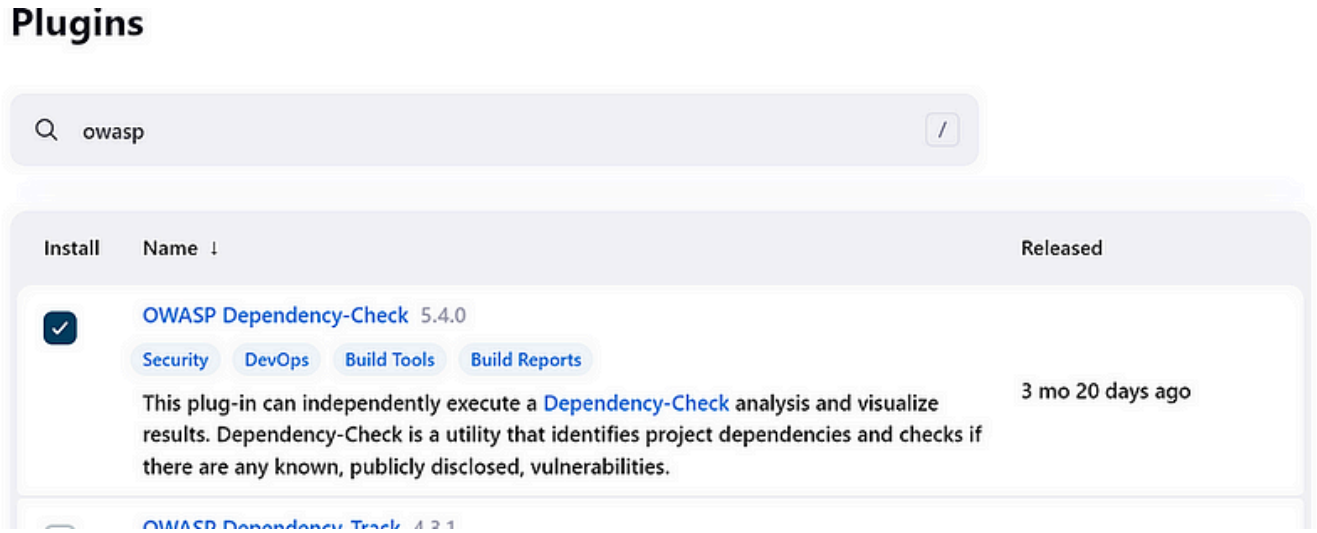


3C – Create a Job

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

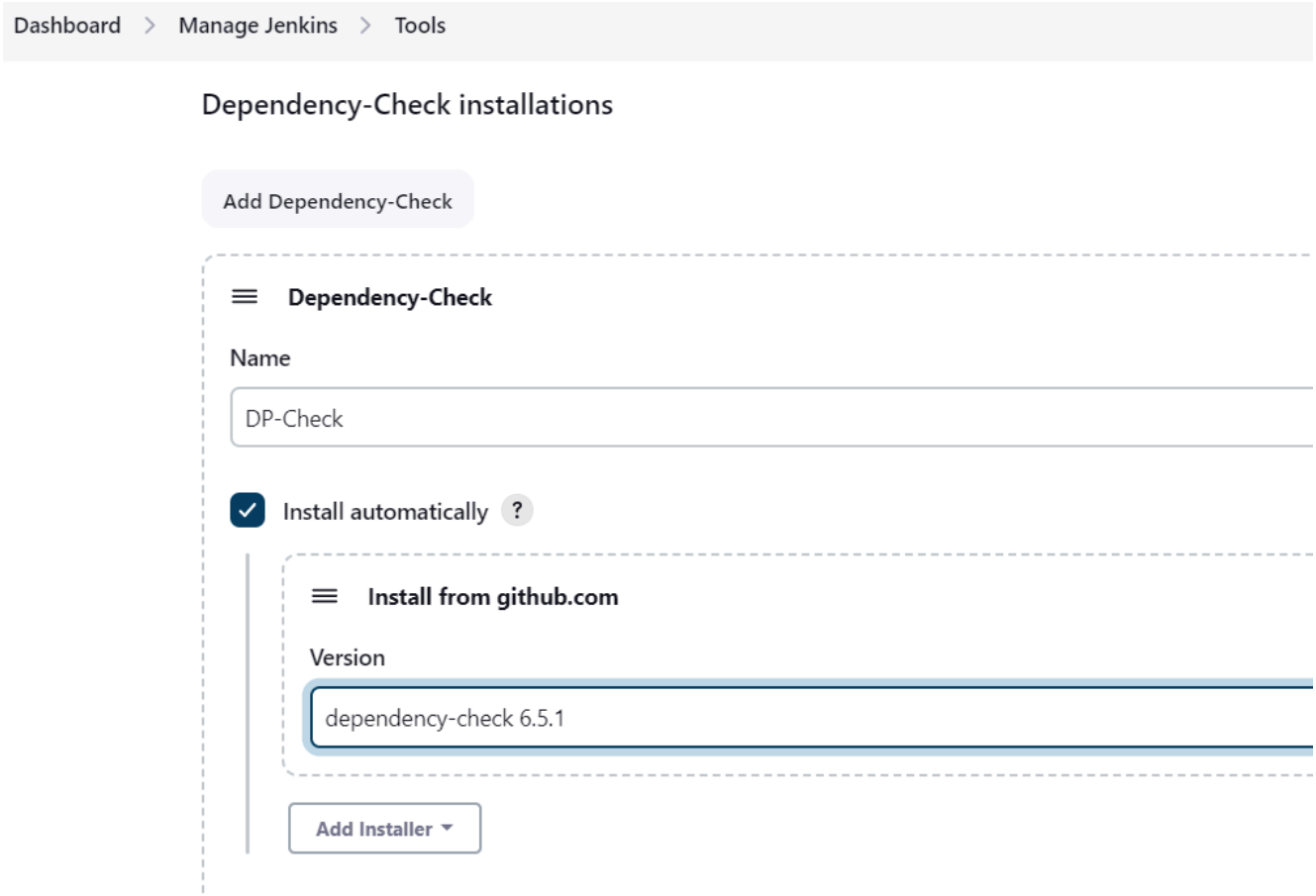
Step 4 – Install OWASP Dependency Check Plugins

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



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

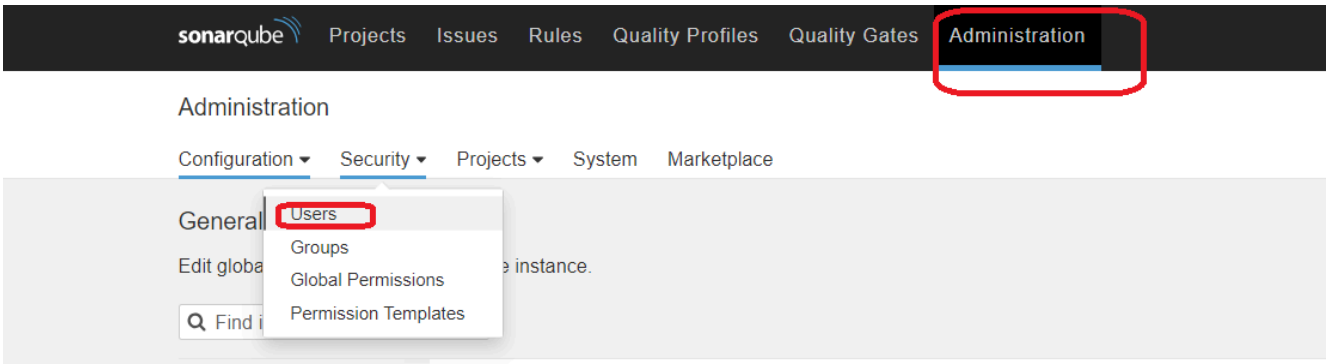
Goto Dashboard → Manage Jenkins → Tools →



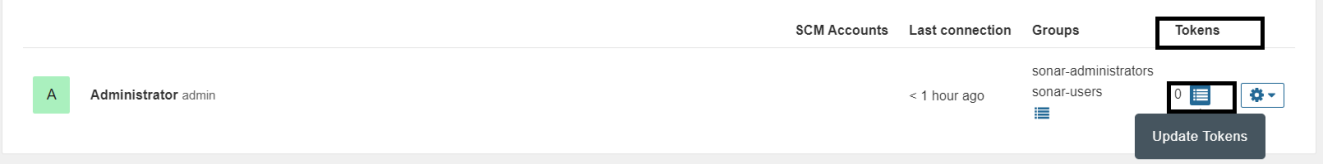
Click on Apply and Save here.

Step 5 – Configure Sonar Server in Manage Jenkins

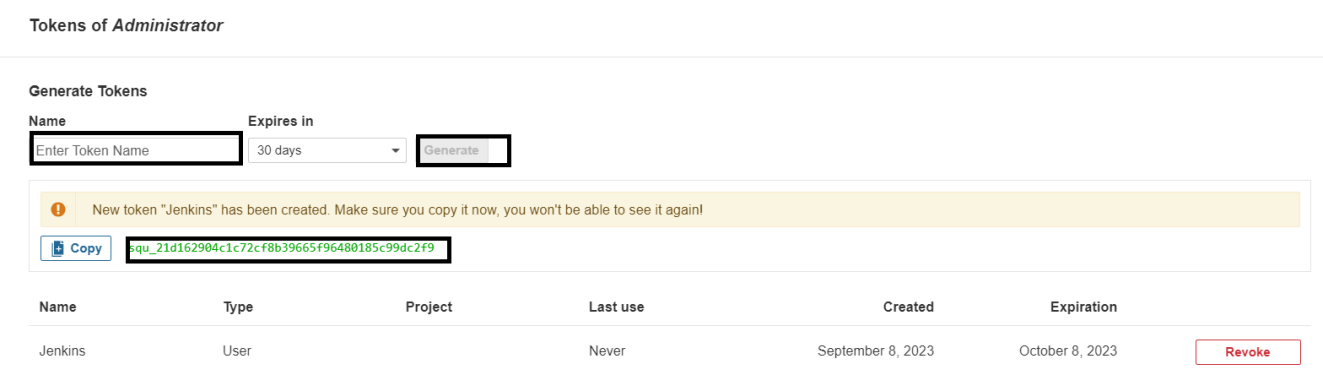
Grab the Public IP Address of your EC2 Instance, Sonarqube works on Port 9000, sp <Public IP>:9000. Goto your Sonarqube Server. Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → and click on Generate Token



Click on Update Token

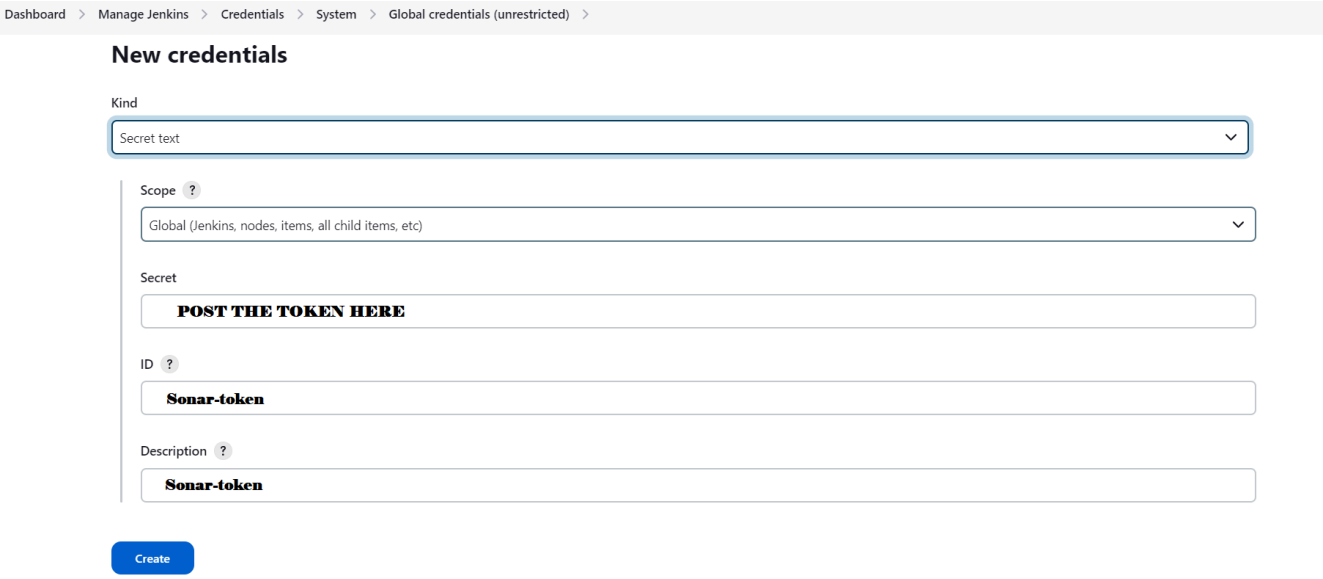


Create a token with a name and generate

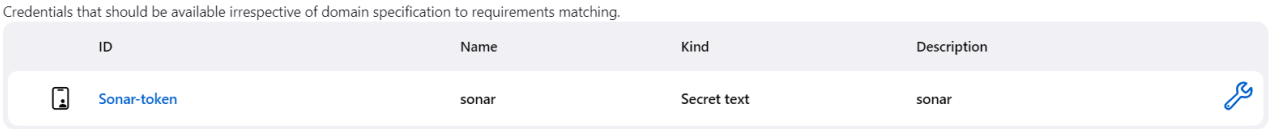


Copy this Token

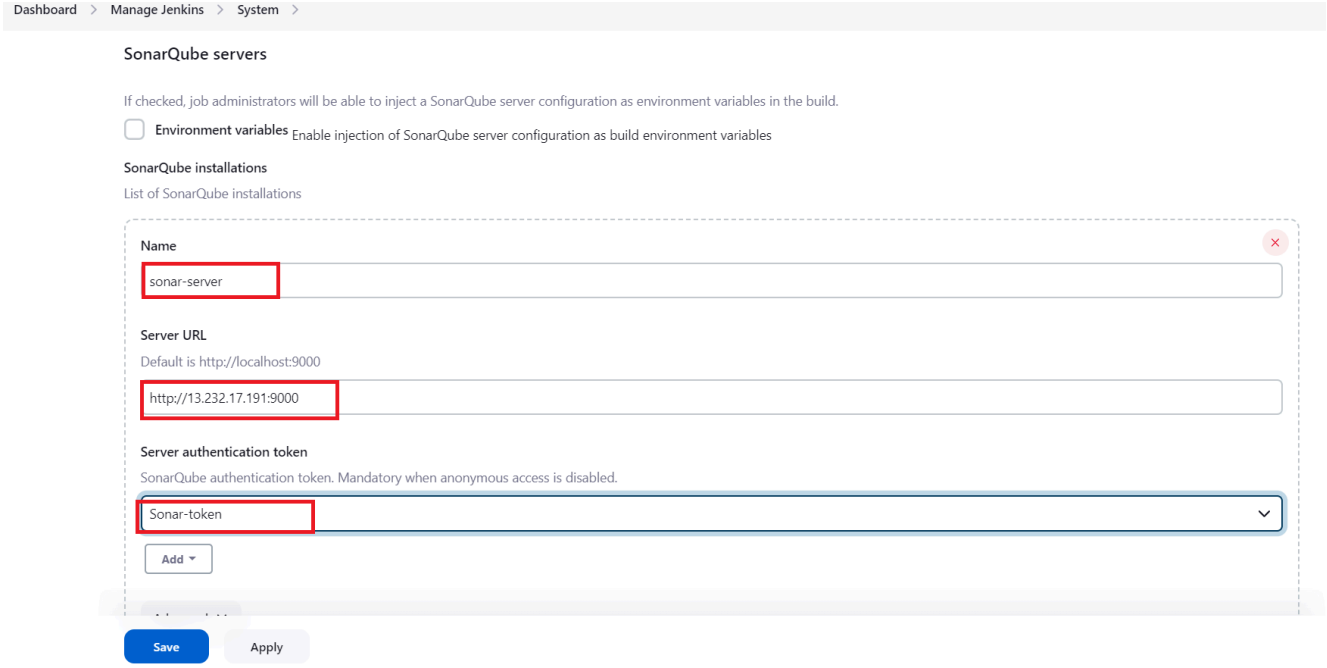
Goto Dashboard → Manage Jenkins → Credentials → Add Secret Text. It should look like this



You will this page once you click on create



Now, go to Dashboard → Manage Jenkins → Configure System

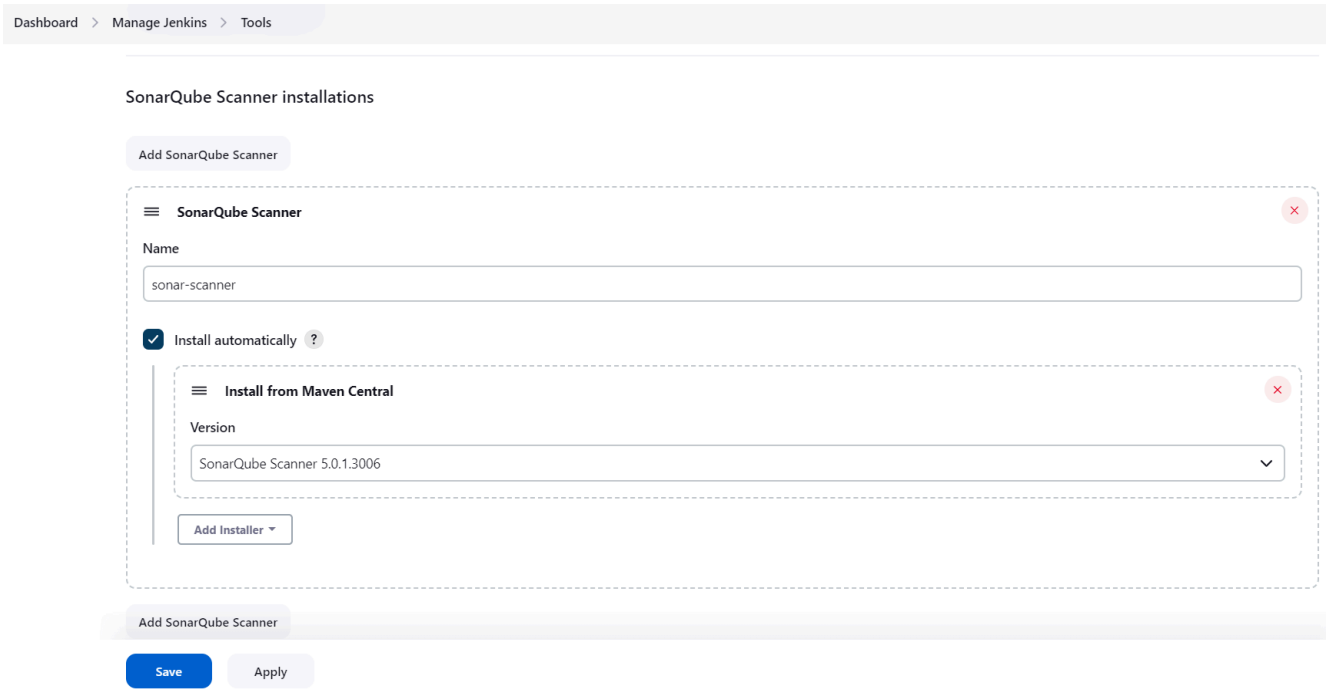


Click on Apply and Save

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

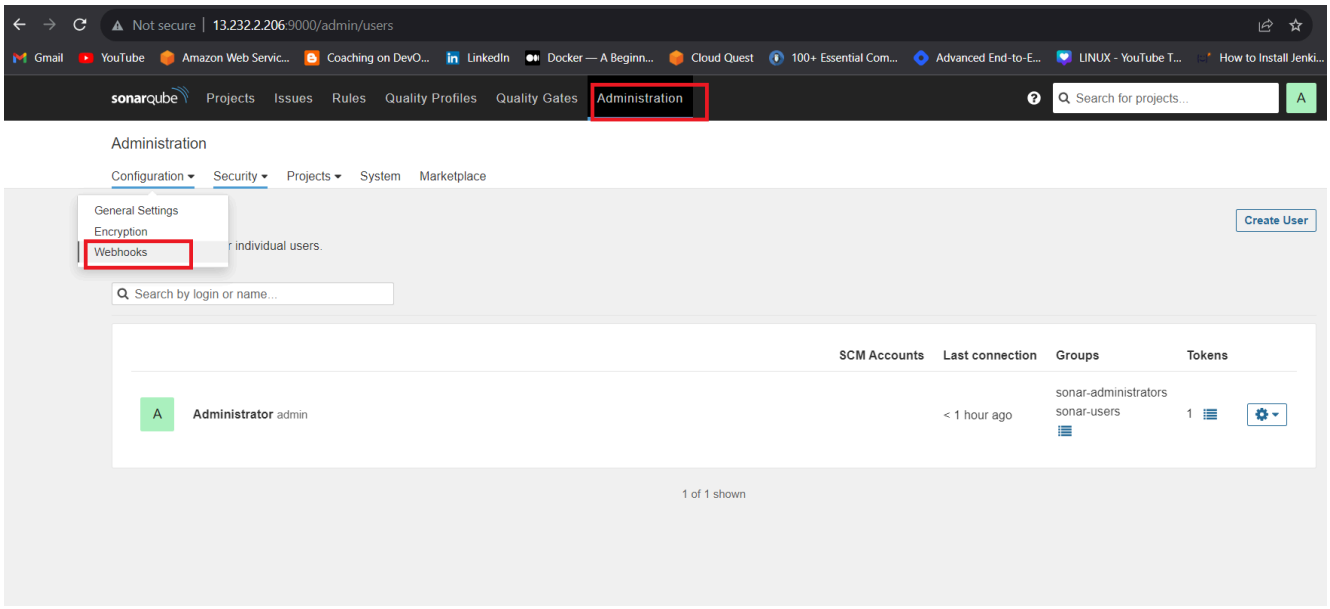
Global Tool Configuration is used to configure different tools that we install using Plugins

We will install a sonar scanner in the tools.

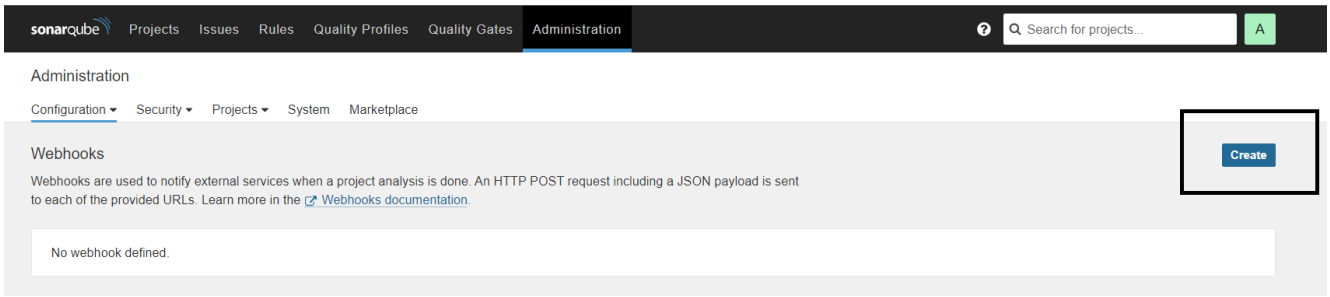


In the Sonarqube Dashboard add a quality gate also

Administration–> Configuration–>Webhooks

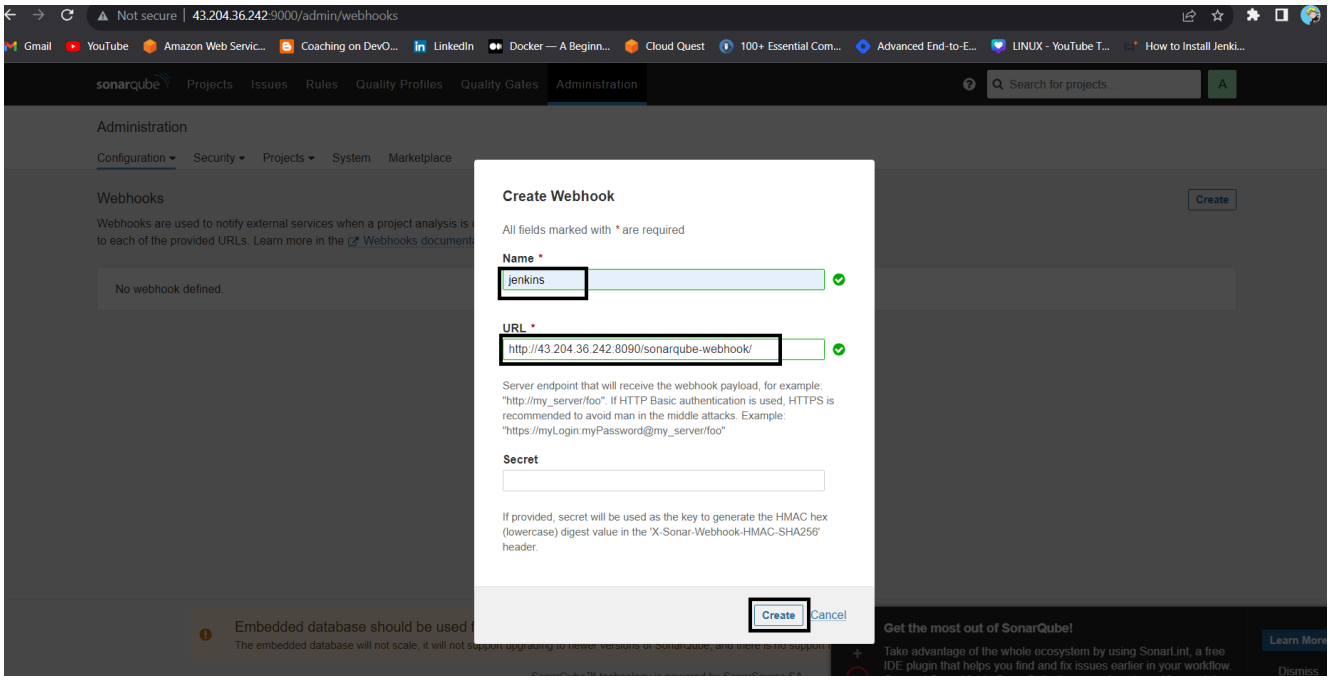


Click on Create



Add details

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

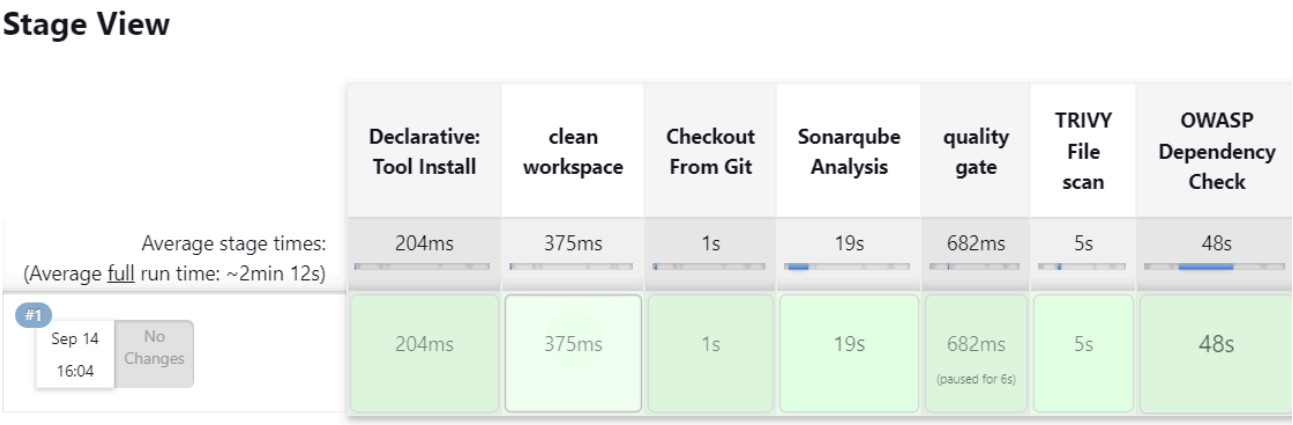


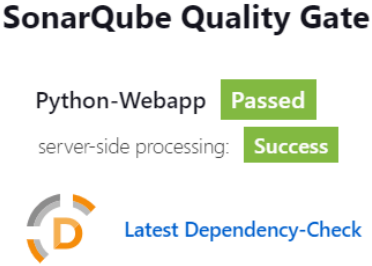
Let's go to our Pipeline and add the below code Pipeline Script.

```
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/DotNe
            }
        }
        stage("Sonarqube Analysis "){
            steps{
                withSonarQubeEnv('sonar-server') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.proje
                    -Dsonar.projectKey=Dotnet-Webapp '''
```

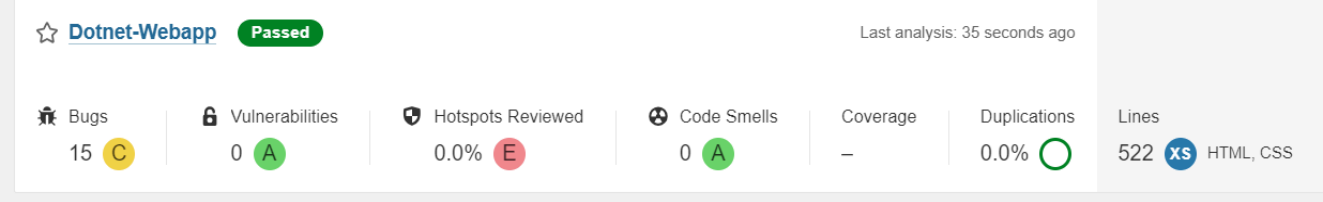
```
            }
        }
    }
    stage("quality gate"){
       steps {
            script {
                waitForQualityGate abortPipeline: false, credentials
            }
        }
    }
    stage("TRIVY File scan"){
        steps{
            sh "trivy fs . > trivy-fs_report.txt"
        }
    }
    stage("OWASP Dependency Check"){
        steps{
            dependencyCheck additionalArguments: '--scan ./ --format
            dependencyCheckPublisher pattern: '**/dependency-check-r
        }
    }
```

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





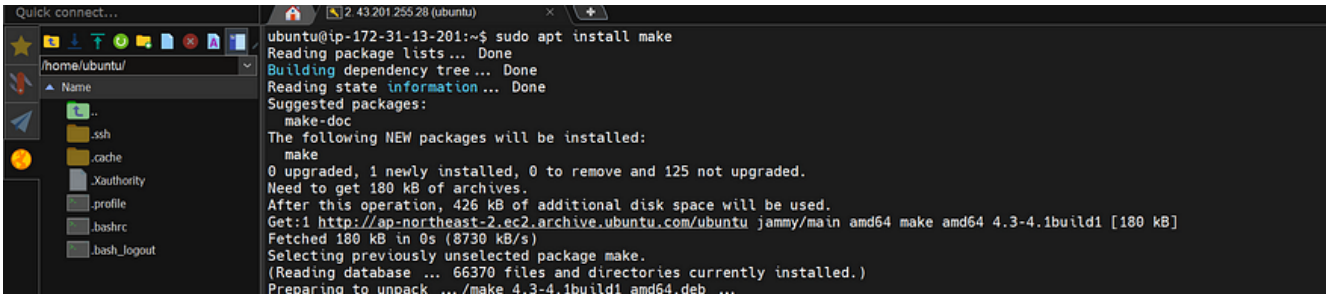
To see the report, you can go to Sonarqube Server and go to Projects.



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
```



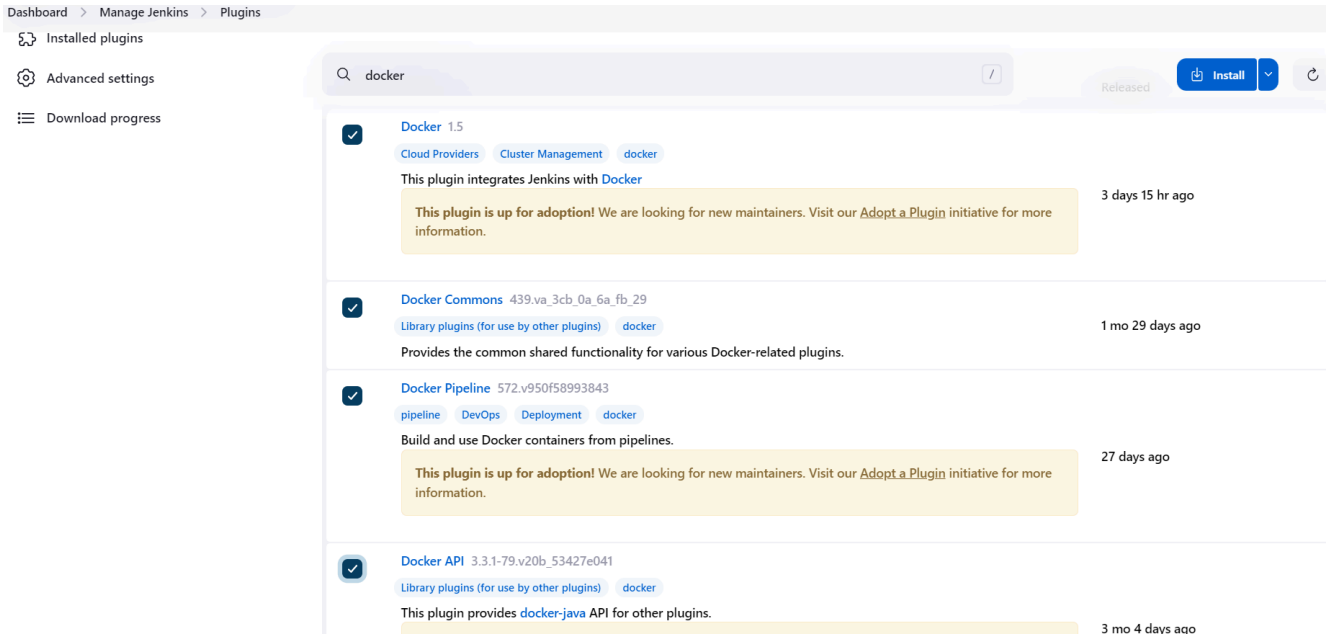
Step 7 – Docker Image Build and Push

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

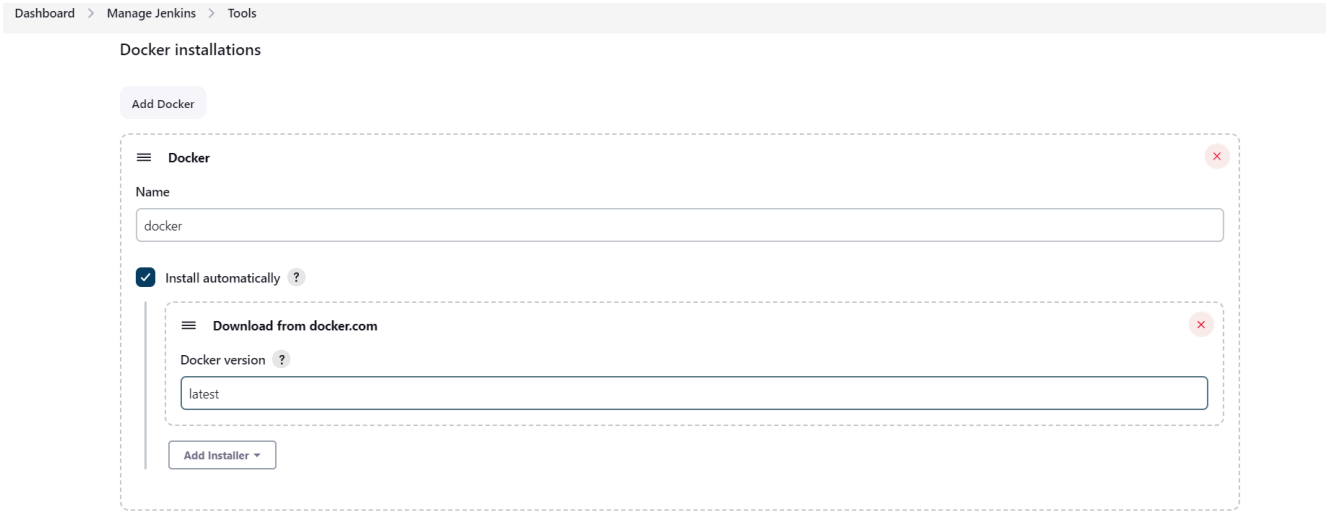
- Docker
- Docker Commons
- Docker Pipeline
- Docker API

- `docker-build-step`

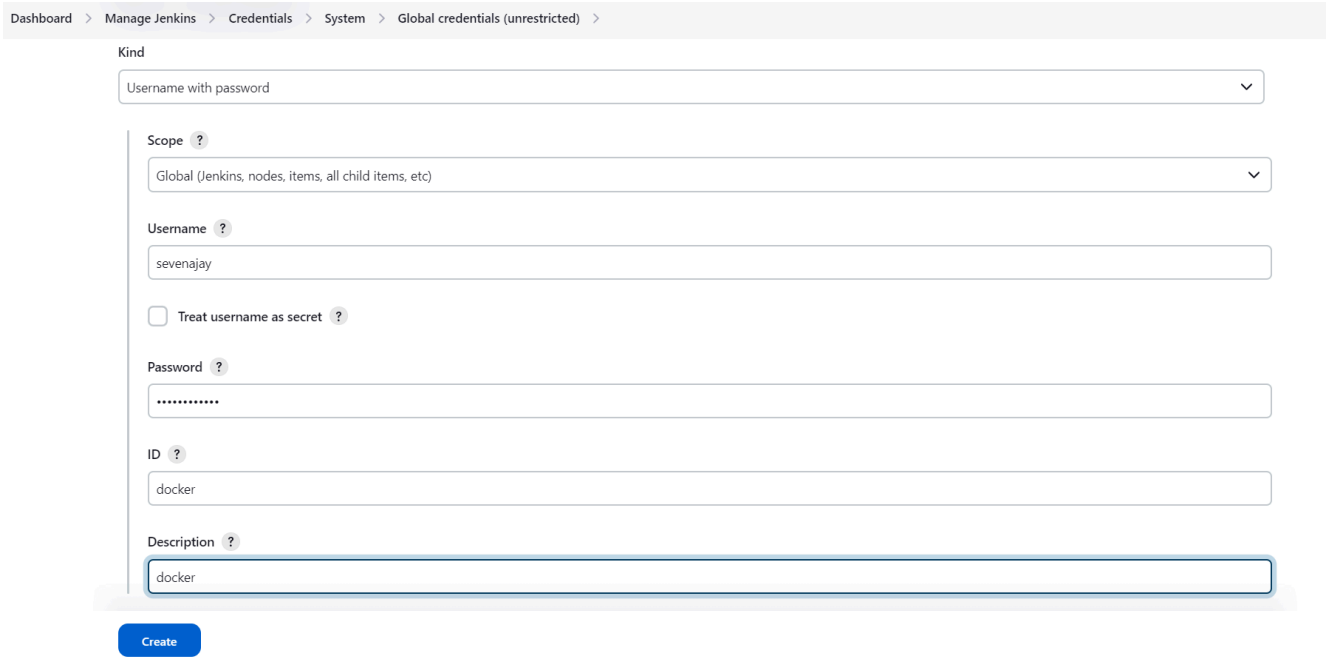
and click on install without restart



Now, goto Dashboard → Manage Jenkins → Tools →



Add DockerHub Username and Password under Global Credentials



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

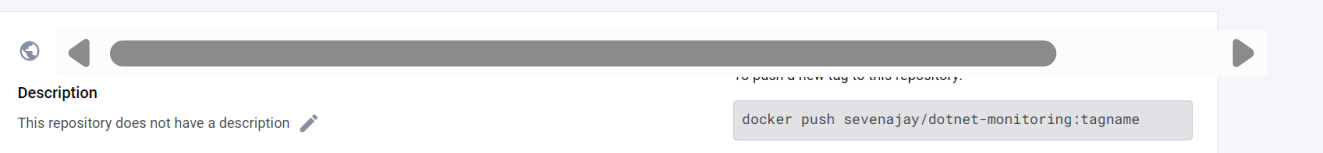
that's why we are using make image and make a push in the place of docker build -t and docker push

Add this stage to Pipeline Script

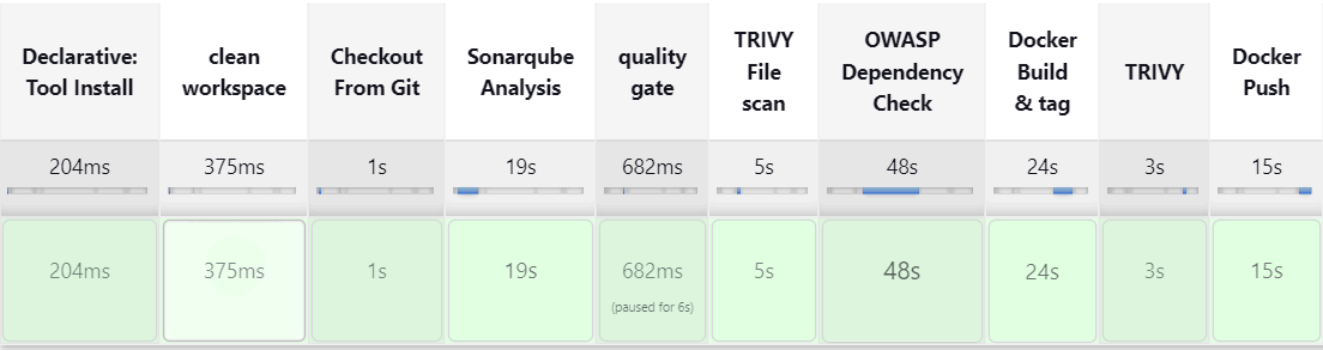
```
stage("Docker Build & tag"){
            steps{
                script{
                   withDockerRegistry(credentialsId: 'docker', toolName
                       sh "make image"
                    }
                }
```

```
        }
    }
    stage("TRIVY"){
        steps{
            sh "trivy image sevenajay/dotnet-monitoring:latest > tr
        }
    }
    stage("Docker Push"){
        steps{
            script{
               withDockerRegistry(credentialsId: 'docker', toolName
                   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

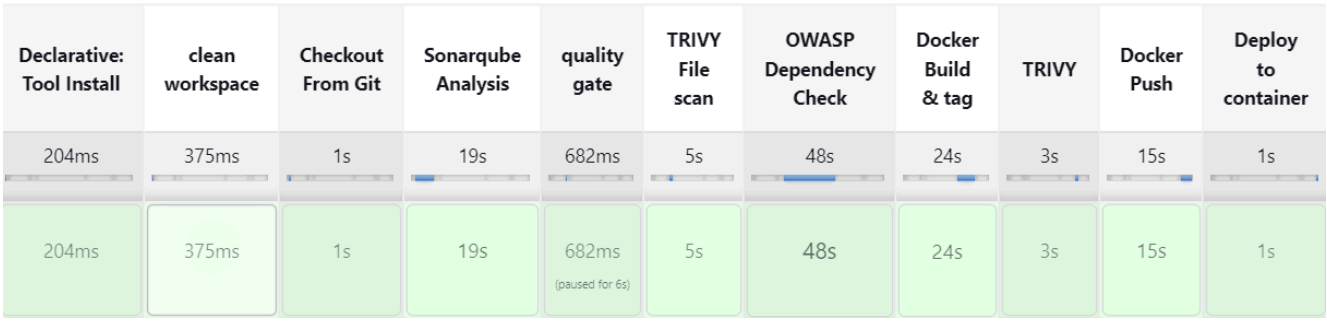


Step 8 – Deploy the image using Docker

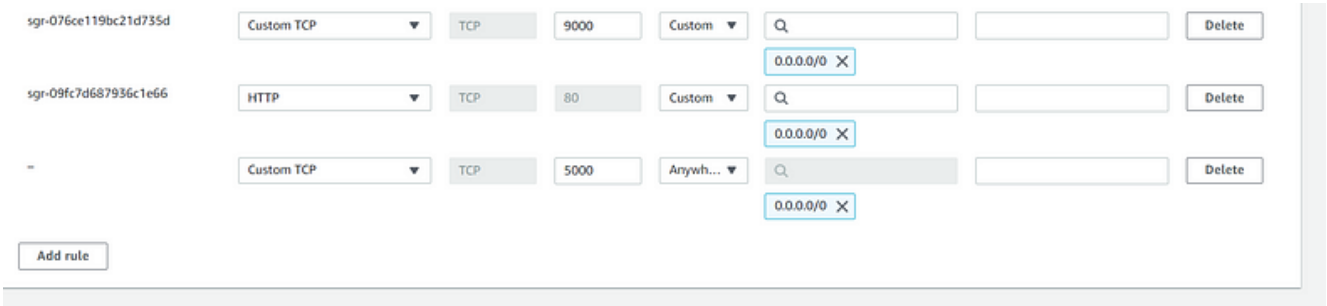
Add this stage to your pipeline syntax

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

You will see the Stage View like this,



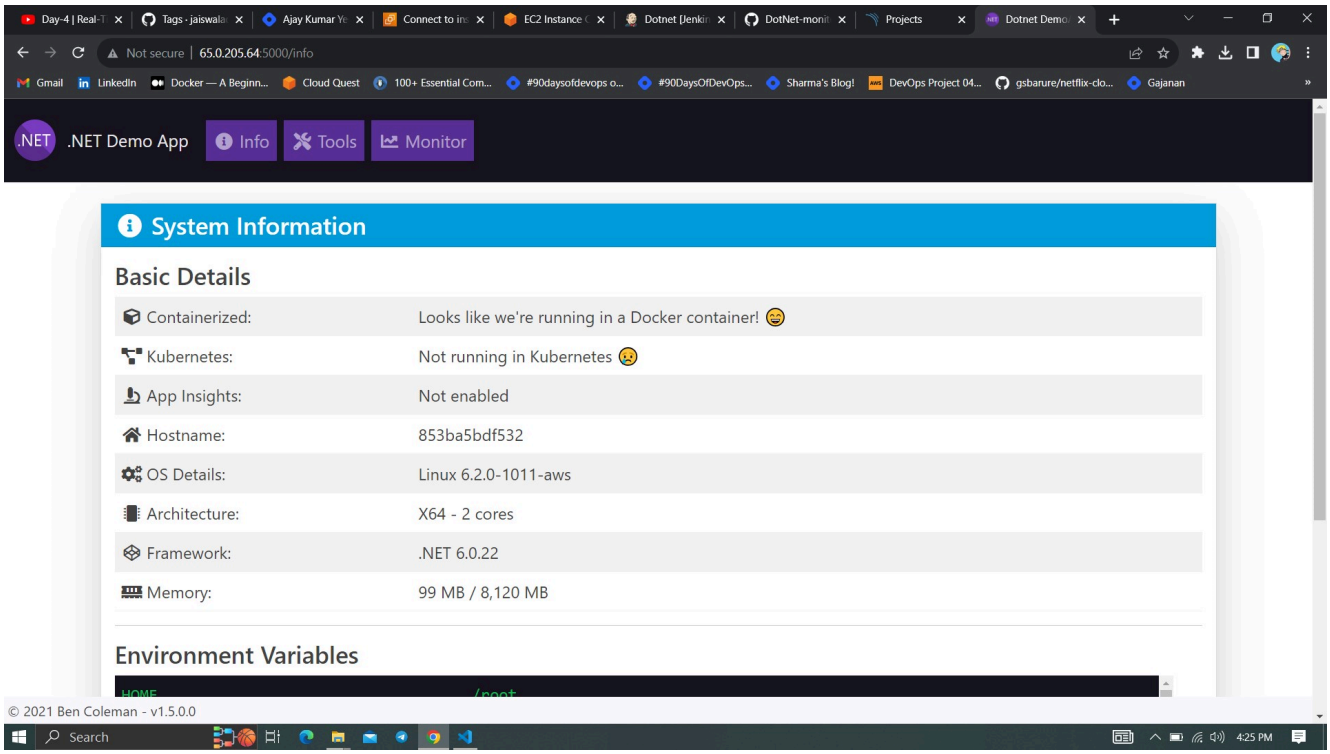
(Add port 5000 to Security Group)

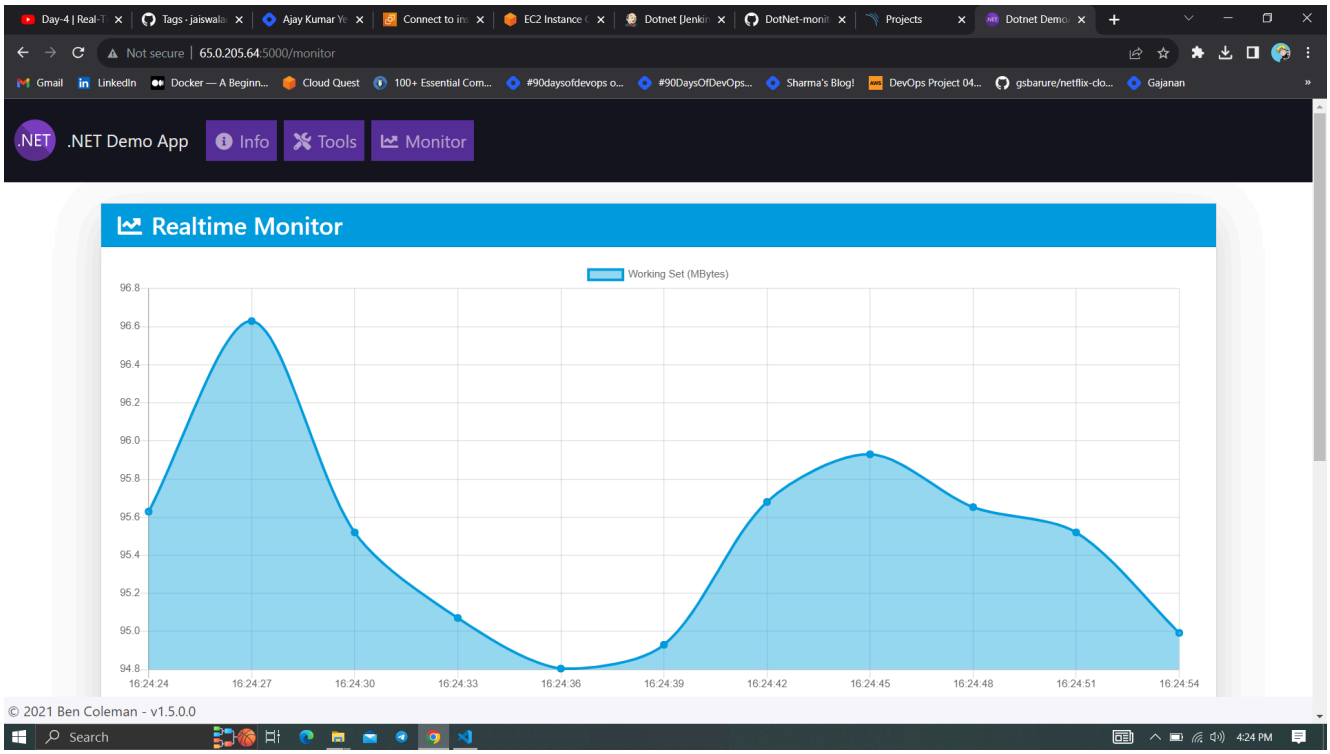


And you can access your application on Port 5000. This is a Real World Application that has all Functional Tabs.

```
<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/releas
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
kubectl 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 | su
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 kubectl
sudo snap install kube-apiserver
```

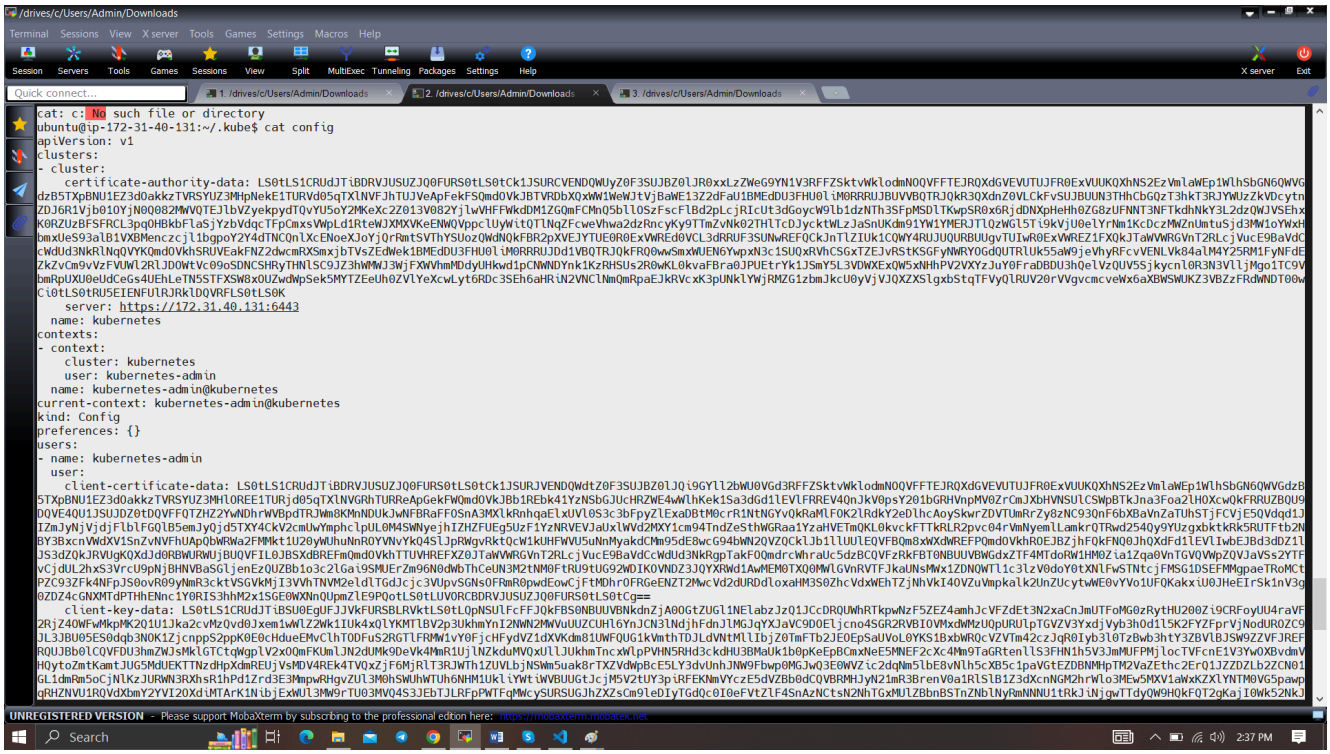
Part 3 — Master

```
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
```

———-Worker Node————

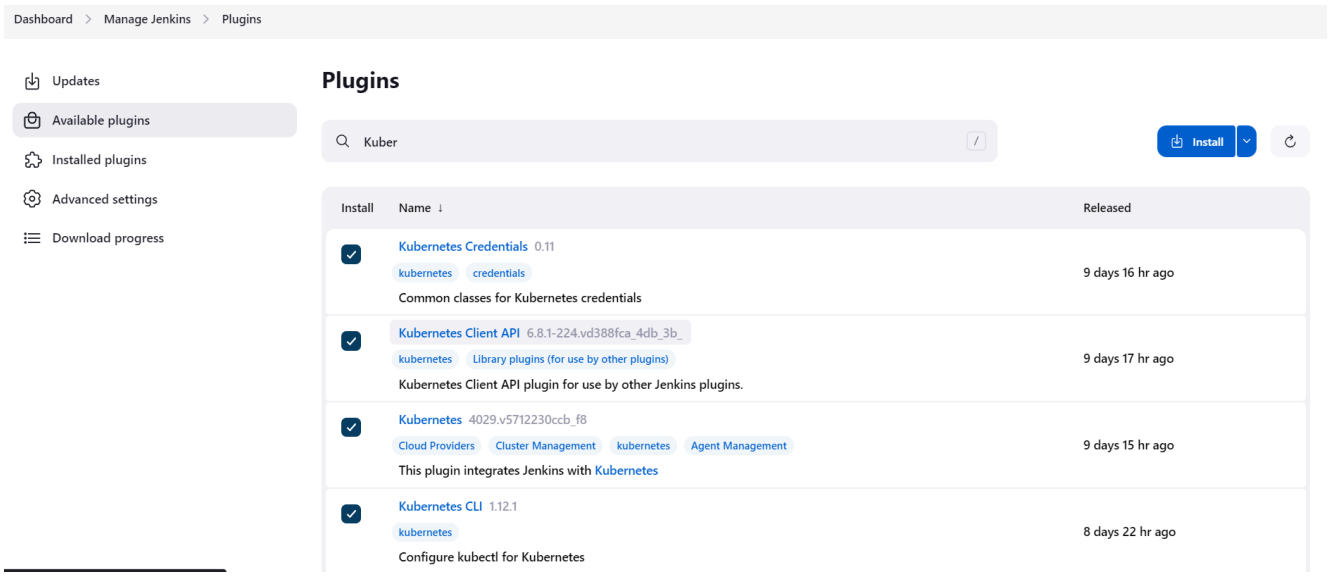
```
sudo kubeadm join <master-node-ip>:<master-node-port> --token <token> --
```

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

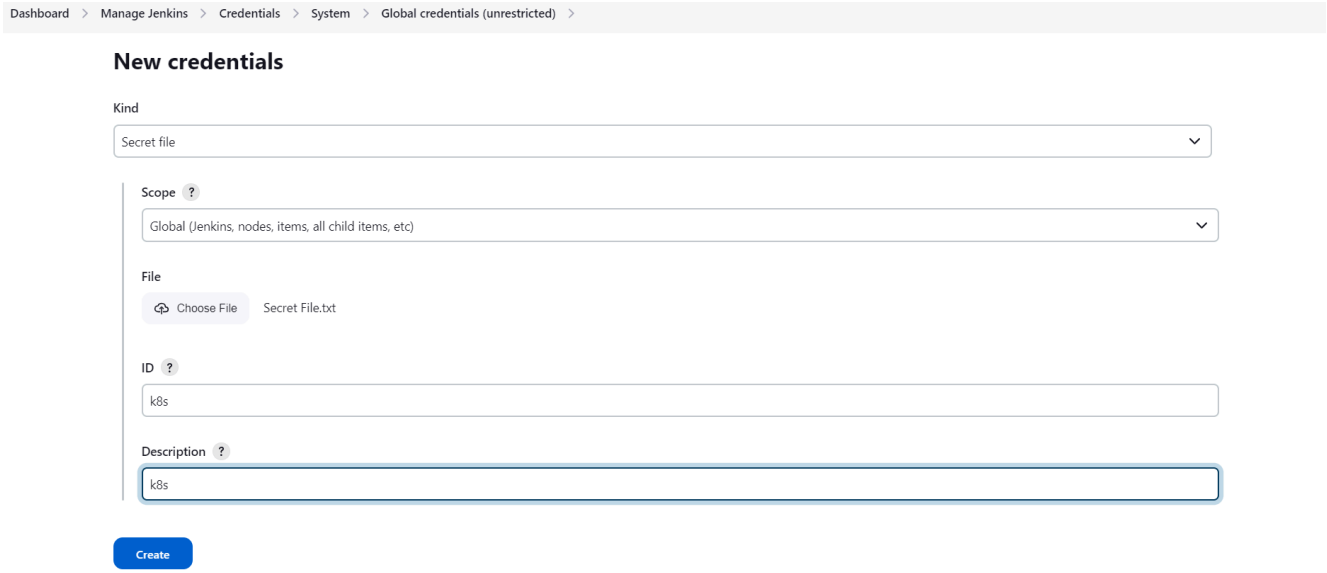


copy it and save it in documents or another folder save it as secret-file.txt

Install Kubernetes Plugin, Once it's installed successfully



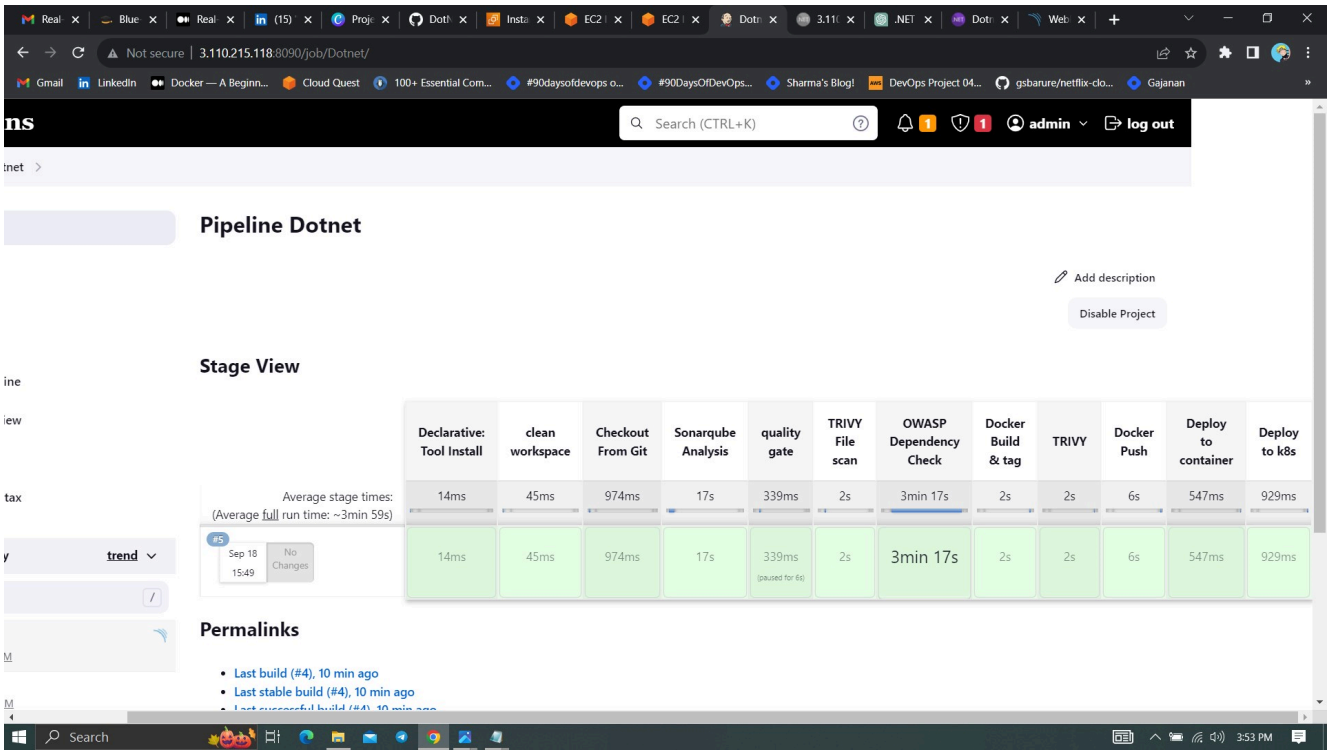
goto manage Jenkins –> manage credentials –> Click on Jenkins global –> add credentials



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

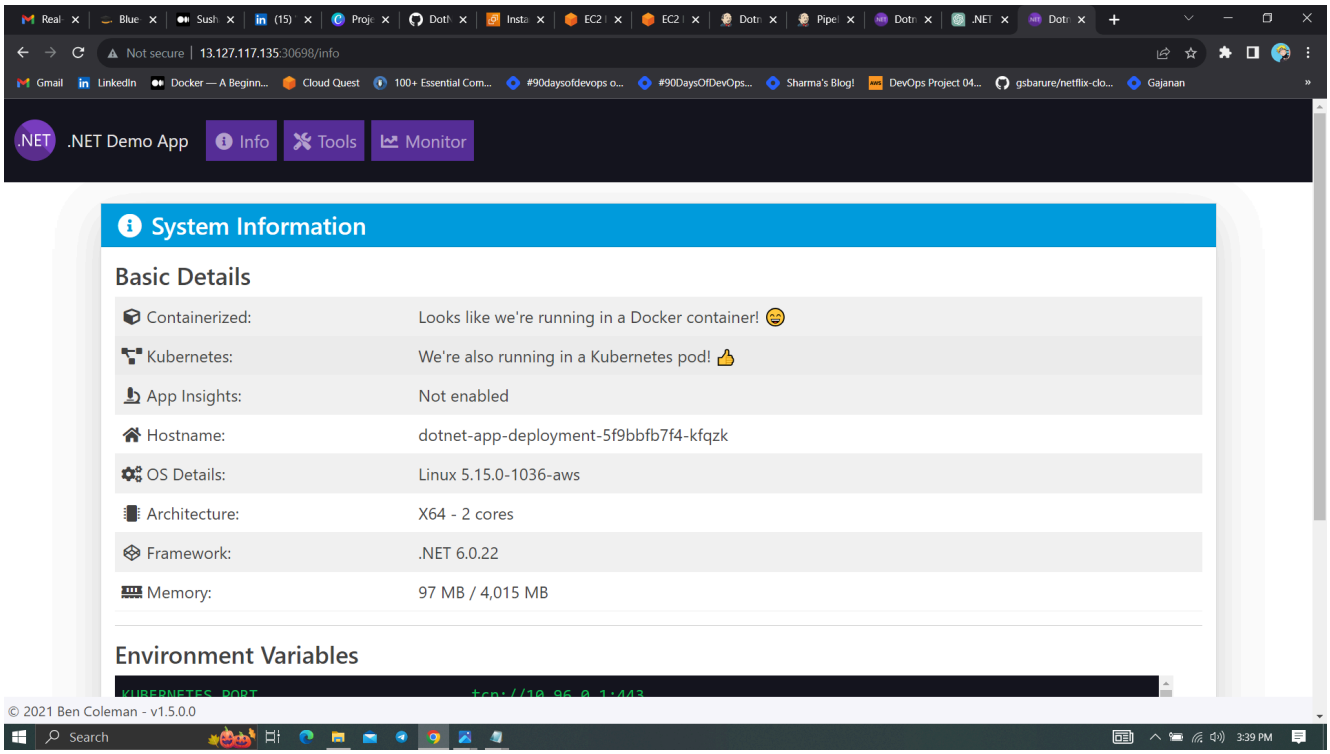
```
stage('Deploy to k8s'){
            steps{
                dir('K8S') {
                  withKubeConfig(caCertificate: '', clusterName: '', cor
                    sh 'kubectl apply -f deployment.yaml'
                   }
                }
            }
        }
```

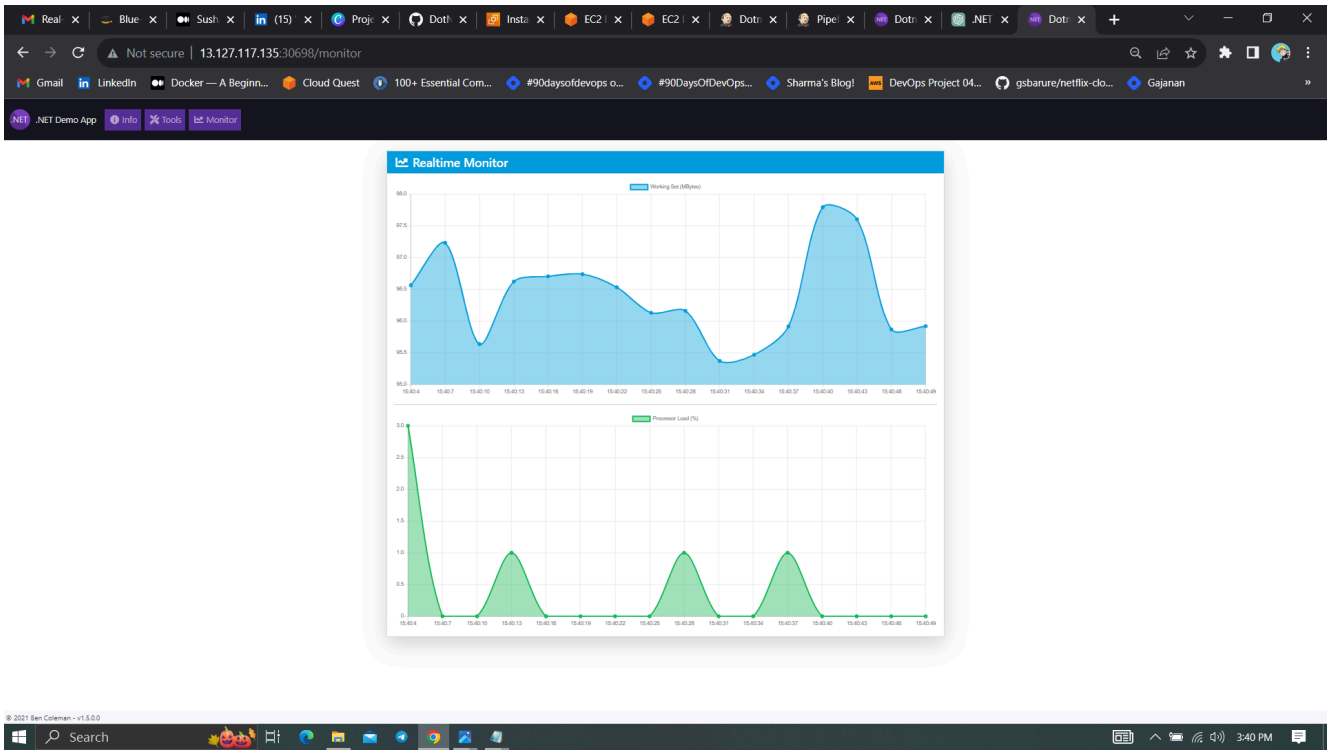
Before starting a new build remove Old containers.



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/DotNe
```

```
        }
    }
    stage("Sonarqube Analysis "){
        steps{
            withSonarQubeEnv('sonar-server') {
                sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.proje
                -Dsonar.projectKey=Dotnet-Webapp '''
            }
        }
    }
    stage("quality gate"){
       steps {
            script {
                waitForQualityGate abortPipeline: false, credentials
            }
        }
    }
    stage("TRIVY File scan"){
        steps{
            sh "trivy fs . > trivy-fs_report.txt"
        }
    }
    stage("OWASP Dependency Check"){
        steps{
            dependencyCheck additionalArguments: '--scan ./ --format
            dependencyCheckPublisher pattern: '**/dependency-check-r
        }
    }
    stage("Docker Build & tag"){
        steps{
            script{
               withDockerRegistry(credentialsId: 'docker', toolName
                   sh "make image"
                }
            }
        }
    }
    stage("TRIVY"){
        steps{
            sh "trivy image sevenajay/dotnet-monitoring:latest > tri
        }
    }
    stage("Docker Push"){
        steps{
            script{
```

```
                    withDockerRegistry(credentialsId: 'docker', toolName
                        sh "make push"
                    }
                }
            }
        }
        stage("Deploy to container"){
            steps{
                sh "docker run -d --name dotnet -p 5000:5000 sevenajay/c
            }
        }
        stage('Deploy to k8s'){
            steps{
                dir('K8S') {
                  withKubeConfig(caCertificate: '', clusterName: '', cor
                    sh 'kubectl apply -f deployment.yaml'
                  }
                }
            }
        }
    }
}
```

Ajay Kumar Yegireddi is a DevSecOps Engineer and System Administrator, with a passion for sharing real-world DevSecOps projects and tasks. **Mr. Cloud Book**, provides hands-on tutorials and practical insights to help others master DevSecOps tools and workflows. Content is designed to bridge the gap between development, security, and operations, making complex concepts easy to understand for both beginners and professionals.

Comments

2 responses to “Jenkins CI/CD | DOTNET webapp DevSecOps Project”

Sudhir Dilip Jadhav
2 February 2024

Getting the below error when I build the pipeline :
****

Started by user Sudhir Dilip Jadhav
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Pipeline-2
[Pipeline] {
[Pipeline] tool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Tool Install)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (clean workspace)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] cleanWs
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] done
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Checkout From Git)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] git
The recommended git tool is: NONE

```
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/Aj7Ay/DotNet-monitoring.git
> git init /var/lib/jenkins/workspace/Pipeline-2 # timeout=10
Fetching upstream changes from https://github.com/Aj7Ay/DotNet-monitoring.git
> git –version # timeout=10
> git –version # 'git version 2.34.1'
> git fetch –tags –force –progress – https://github.com/Aj7Ay/DotNet-monitoring.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/Aj7Ay/DotNet-monitoring.git #
timeout=10
> git config –add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* #
timeout=10
Avoid second fetch
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision e2456b125b06eced47096fbf219c1d50b75c153f
(refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f e2456b125b06eced47096fbf219c1d50b75c153f # timeout=10
> git branch -a -v –no-abbrev # timeout=10
> git checkout -b main e2456b125b06eced47096fbf219c1d50b75c153f # timeout=10
Commit message: "Create deployment.yaml"
First time build. Skipping changelog.
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Sonarqube Analysis )
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] withSonarQubeEnv
Injecting SonarQube environment variables using the configuration: sonar-server
[Pipeline] {
[Pipeline] sh
+ /var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/sonar-
scanner/bin/sonar-scanner -Dsonar.projectName=Dotnet-Webapp -
Dsonar.projectKey=Dotnet-Webapp
Error: A JNI error has occurred, please check your installation and try again
Exception in thread "main" java.lang.UnsupportedClassVersionError:
org/sonarsource/scanner/cli/Main has been compiled by a more recent version of
the Java Runtime (class file version 61.0), this version of the Java Runtime only
recognizes class file versions up to 52.0
```

```
at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass(ClassLoader.java:756)
at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
at java.net.URLClassLoader.defineClass(URLClassLoader.java:473)
at java.net.URLClassLoader.access$100(URLClassLoader.java:74)
at java.net.URLClassLoader$1.run(URLClassLoader.java:369)
at java.net.URLClassLoader$1.run(URLClassLoader.java:363)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:362)
at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:352)
at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
at sun.launcher.LauncherHelper.checkAndLoadMain(LauncherHelper.java:621)
[Pipeline] }
WARN: Unable to locate ‘report-task.txt’ in the workspace. Did the SonarScanner
succeed?
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (quality gate)
Stage “quality gate” skipped due to earlier failure(s)
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (TRIVY File scan)
Stage “TRIVY File scan” skipped due to earlier failure(s)
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (OWASP Dependency Check)
Stage “OWASP Dependency Check” skipped due to earlier failure(s)
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 1
Finished: FAILURE
```

****

Reply

test
28 March 2024

test

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

I'm not a robot
reCAPTCHA
Privacy - Terms

Post Comment

Uncategorized

How to Automate Incident Response : How Q Developer Helped Me Automate a Daily Pain Point

22 July 2025

AI

How to Run Docker Model Runner on Ubuntu 24.04

11 July 2025

AI, DevOps

How to Install docker-ai on Ubuntu 24.04

15 June 2025

Upskill with Ajay: DevSecOps Mastery

Join Mr Cloud book to master DevSecOps through real-world projects. Learn CI/CD, security integration, automation, and more, gaining hands-on skills for industry-level challenges.

Important Links

Privacy Policy

Terms & Conditions

Contact

Resources

Blog

YouTube Channel