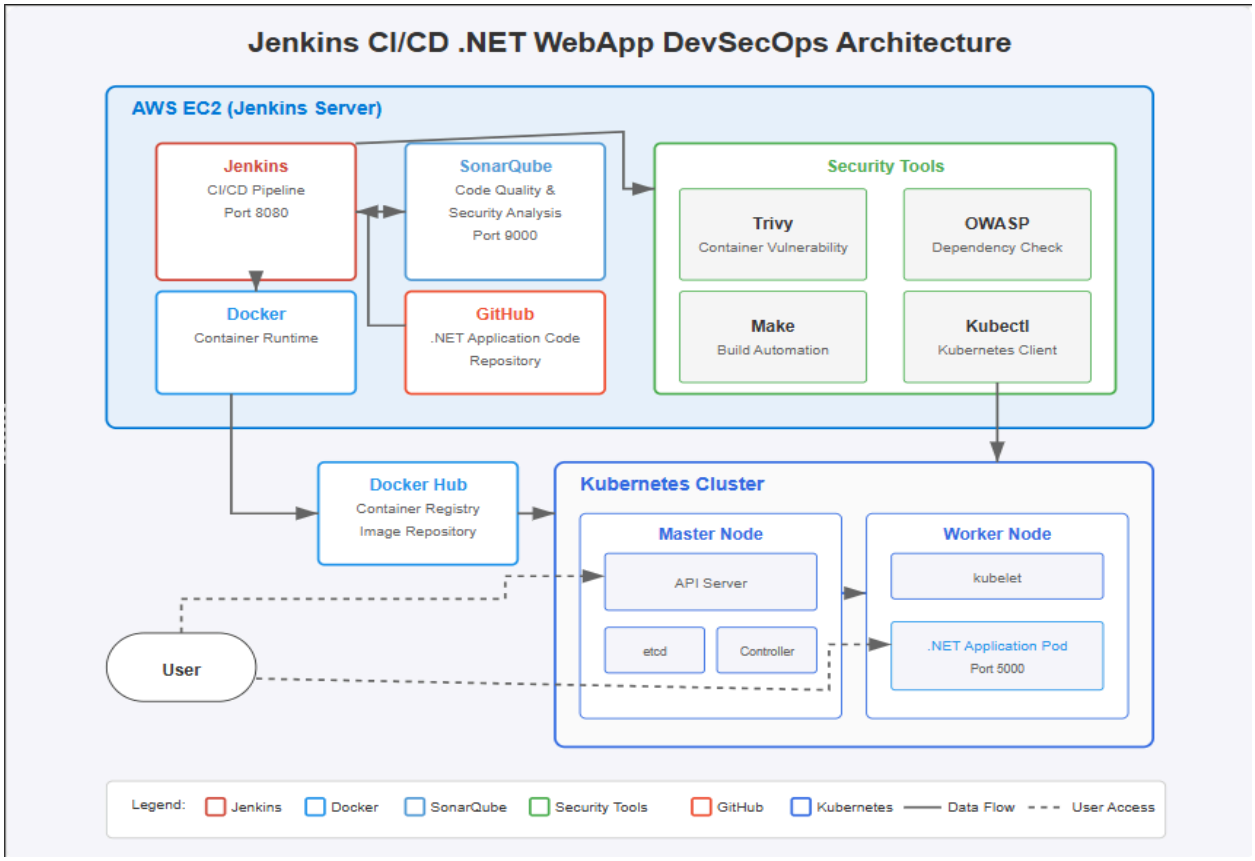
Vadakathi Muhammed Suhaib
Technical Apprentice
Emp ID: X48GRSTML
muhammed.suhaib@cprime.com

Jenkins CI/CD | .NET WebApp DevSecOps Capstone Project



Jenkins CI/CD | .NET WebApp DevSecOps Capstone Project

Introduction

This documentation provides a comprehensive guide for implementing a CI/CD pipeline for a .NET application using Jenkins. The project demonstrates a real-world DevSecOps implementation by deploying a .NET-based application using Jenkins as a CICD tool and deploying the application on Docker Containers and a Kubernetes cluster.

Tools Used

1. AWS EC2

Description: Amazon Elastic Compute Cloud (EC2) provides scalable computing capacity in the Amazon Web Services cloud.
Usage: Hosting the Jenkins server and Kubernetes nodes.

2. Jenkins

Description: An open-source automation server that enables developers to build, test, and deploy their software.
Usage: Creating and managing our CI/CD pipeline.

3. Docker

Description: A platform for developing, shipping, and running applications inside containers.
Usage: Containerizing our .NET application and running SonarQube.

4. SonarQube

Description: An open-source platform for continuous inspection of code quality.
Usage: Performing code quality and security analysis.

5. Trivy

Description: A simple and comprehensive vulnerability scanner for containers and other artifacts.
Usage: Scanning Docker images for vulnerabilities.

6. OWASP Dependency Check

Description: A utility that identifies project dependencies and checks if there are any known, publicly disclosed vulnerabilities.
Usage: Checking for vulnerabilities in project dependencies.

7. Kubernetes

Description: An open-source system for automating deployment, scaling, and management of containerized applications.
Usage: Orchestrating our containerized application.

8. Make

Description: A build automation tool that automatically builds executable programs and libraries from source code.
Usage: Simplifying Docker image building and pushing commands.

Importance of DevSecOps

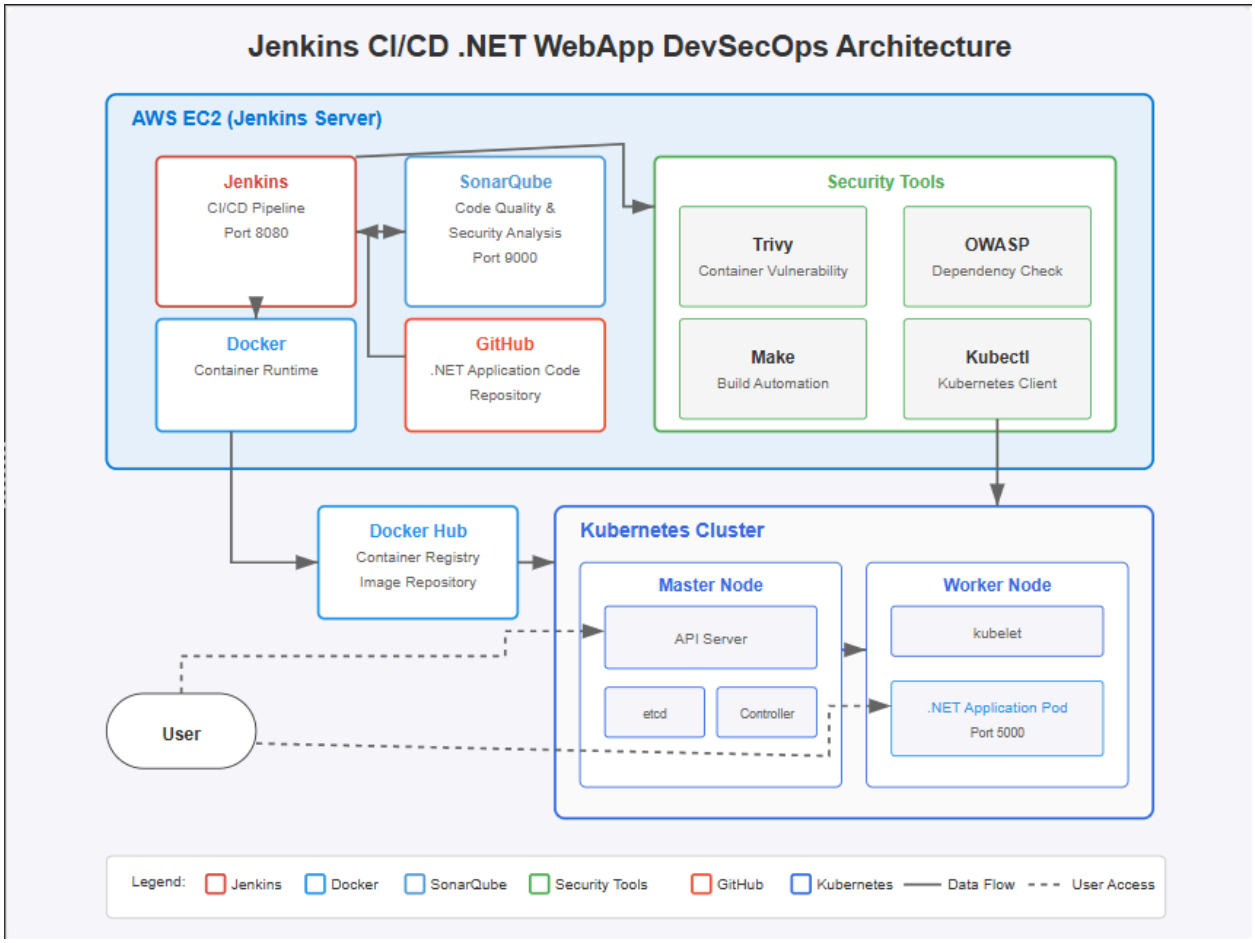
DevSecOps integrates security practices within the DevOps process. This integration is crucial because:

1. **Early Security Integration**: Security checks are performed throughout the development process, not just at the end.
2. **Continuous Security**: Regular security testing helps identify vulnerabilities early.
3. **Automated Security**: Security tests are automated, ensuring consistency and reducing human error.
4. **Compliance Management**: Helps ensure that applications meet regulatory requirements.
5. **Risk Reduction**: Reduces the risk of security breaches by identifying and addressing vulnerabilities early.

In this project, DevSecOps principles are applied through:

- SonarQube for code quality and security analysis
- Trivy for container vulnerability scanning
- OWASP Dependency Check for identifying vulnerabilities in dependencies

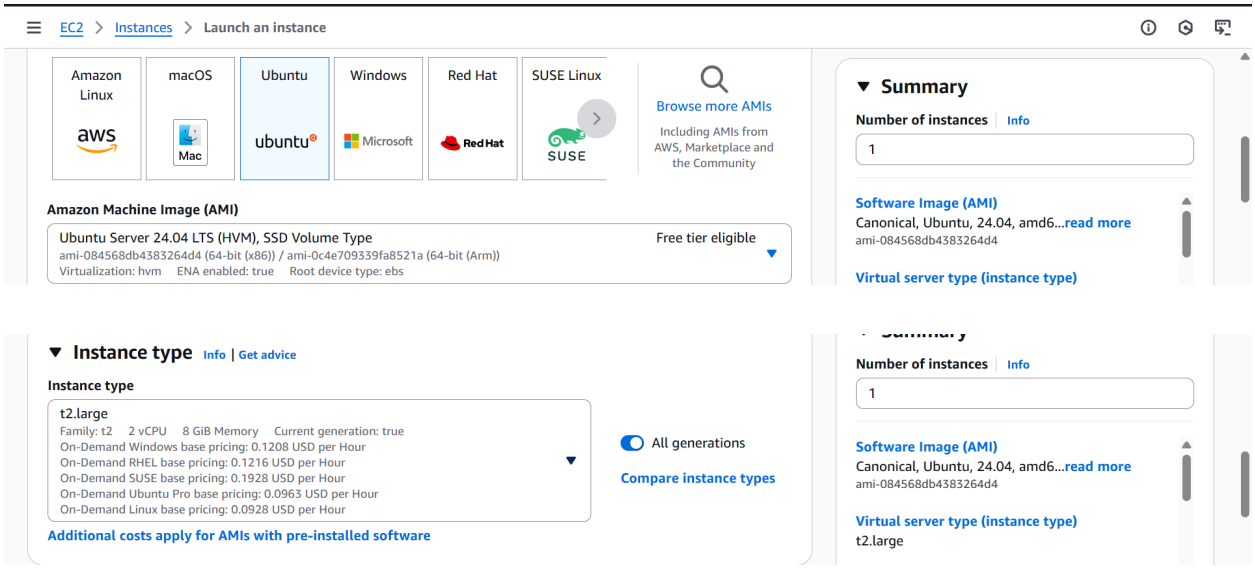
System Architecture



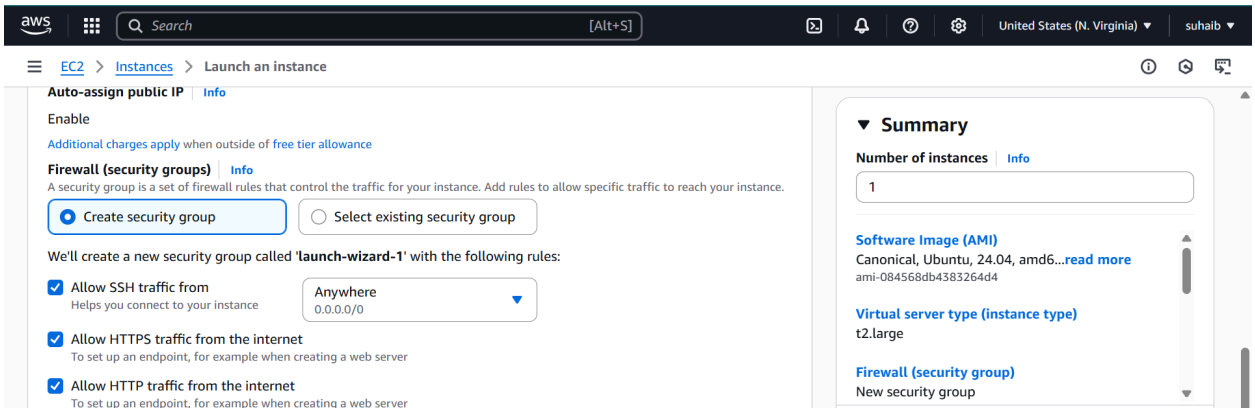
Step-by-Step Implementation Guide

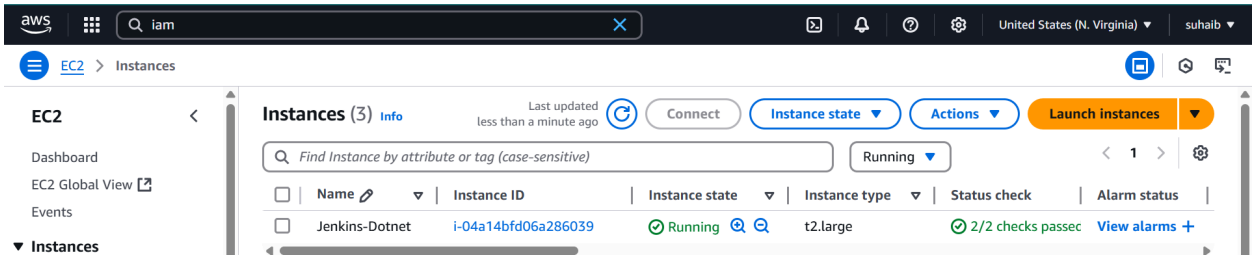
Step 1: Launch an AWS T2 Large Instance

1. Use Ubuntu as the base image



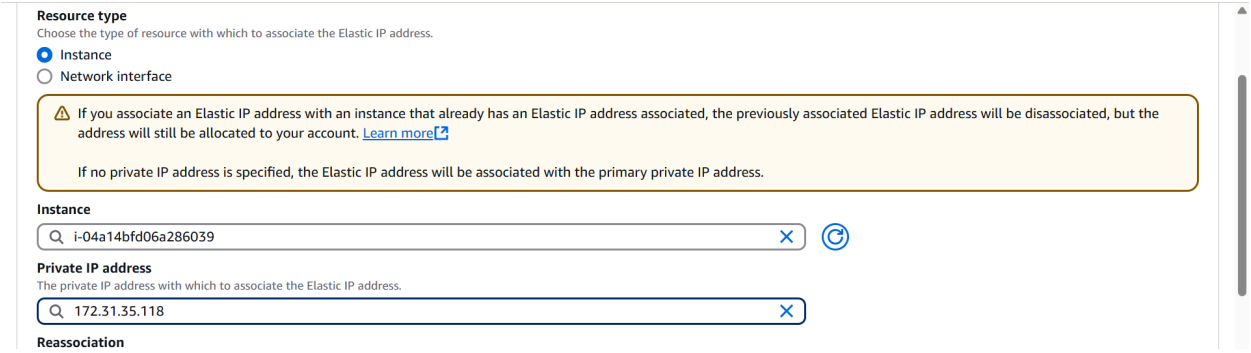
2. Create a new key pair or use an existing one
3. Enable HTTP and HTTPS settings in the Security Group





4. Open ports 8080 (Jenkins), 9000 (SonarQube), and 5000 (Application) in the Security Group

5. Associate elastic IP

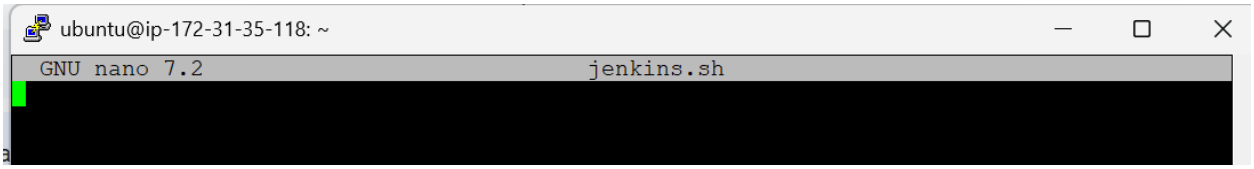


Step 2: Install Jenkins, Docker, and Trivy

2A - Install Jenkins

Create a script file to install Jenkins:

```
sudo nano jenkins.sh
```



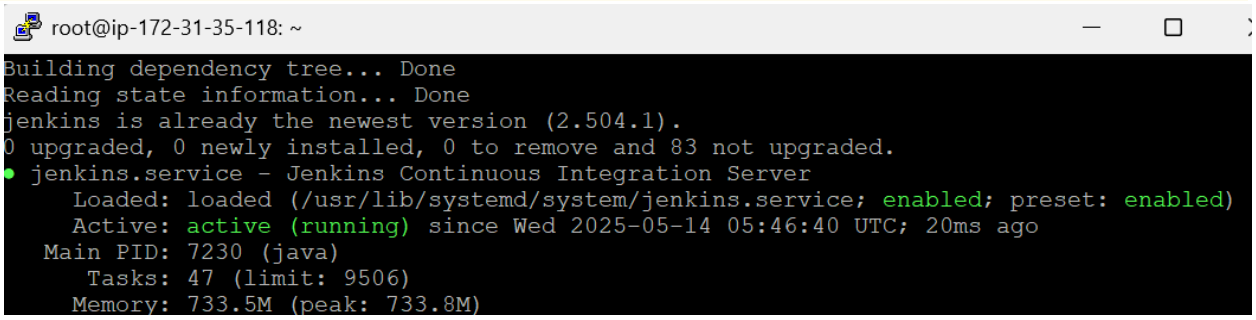
Add the following content:

```
#!/bin/bash
sudo apt update -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
```

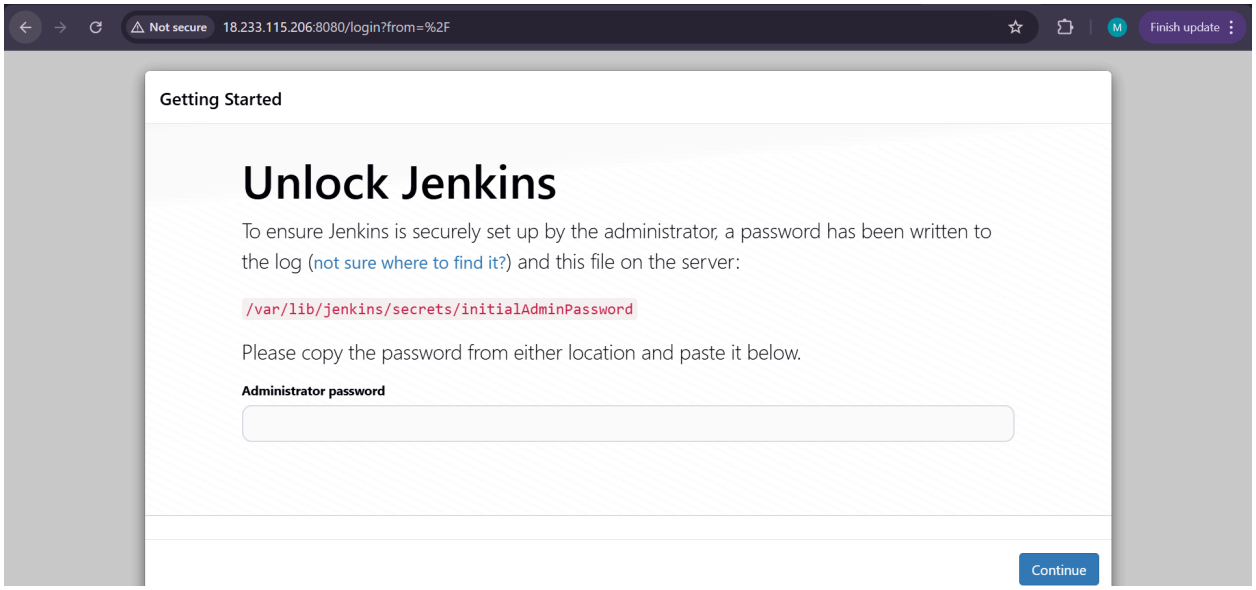
Make the script executable and run it:

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



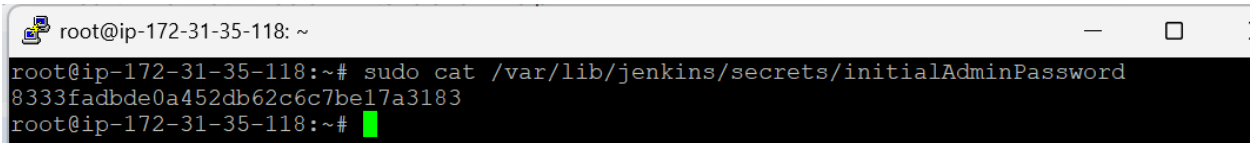
Access Jenkins from your browser:

```
http://<EC2-Public-IP-Address>:8080
http://18.233.115.206:8080
```



Get the Jenkins admin password:

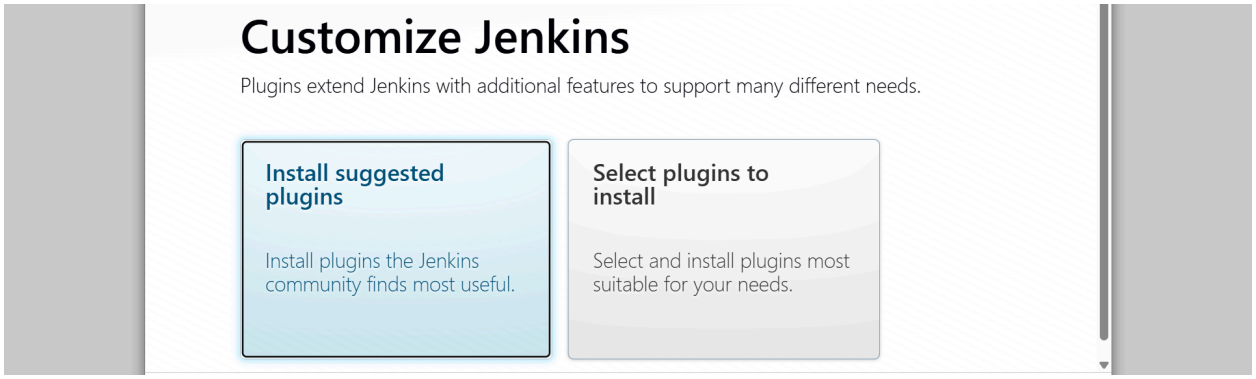
```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

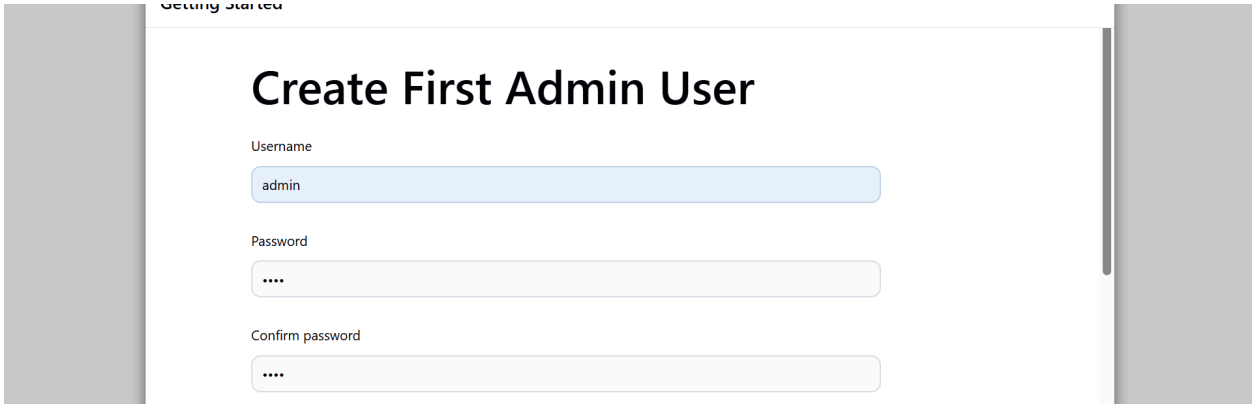


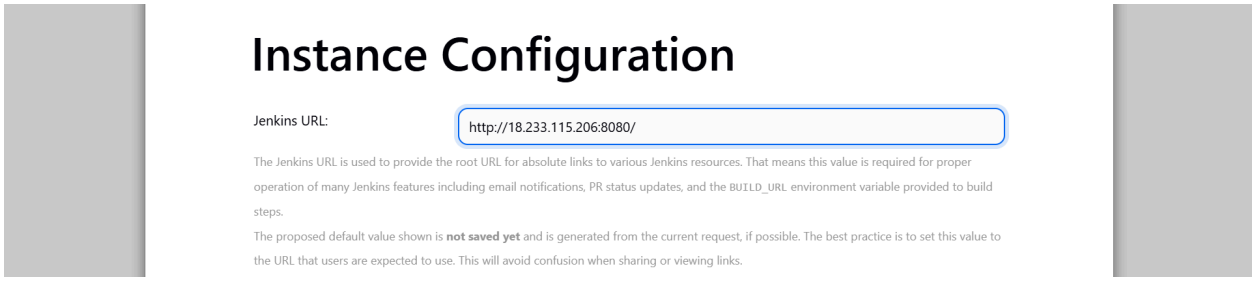

```
8333fadbde0a452db62c6c7be17a3183
```

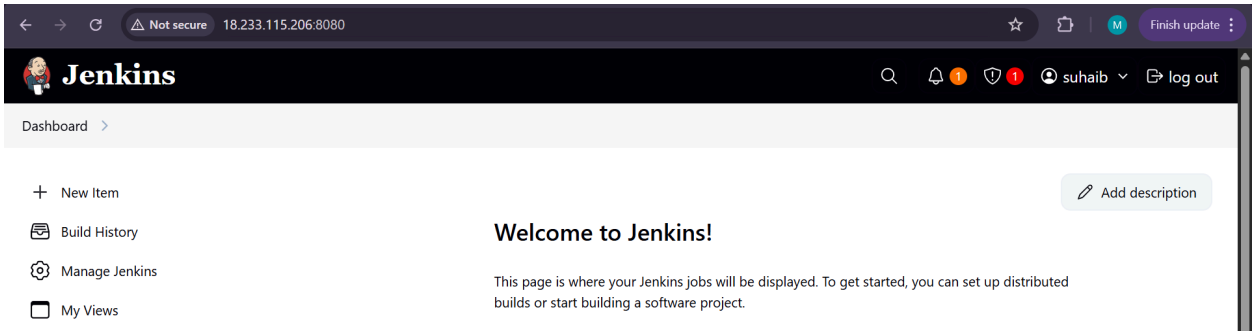
Complete the Jenkins setup by:

1. Entering the admin password
2. Installing suggested plugins
3. Creating an admin user
4. Configuring the instance









2B - Install Docker

Install Docker and run a SonarQube container:

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

```
root@ip-172-31-35-118:~# sudo apt-get update
sudo apt-get install docker.io -y
sudo usermod -aG docker $USER
newgrp docker
sudo chmod 777 /var/run/docker.sock
sudo docker ps
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Ign:4 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:5 https://pkg.jenkins.io/debian-stable binary/ Release
```

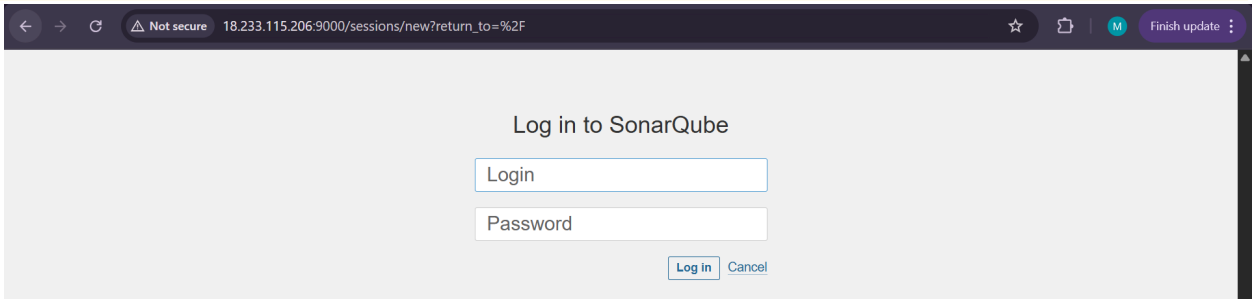
```
root@ip-172-31-35-118:~# docker version
Client:
 Version:           26.1.3
 API version:       1.45
 Go version:        go1.22.2
 Git commit:        26.1.3-0ubuntu1~24.04.1
 Built:             Mon Oct 14 14:29:26 2024
 OS/Arch:           linux/amd64
 Context:           default

Server:
 Engine:
  Version:          26.1.3
  API version:      1.45 (minimum version 1.24)
  Go version:       go1.22.2
  Git commit:       26.1.3-0ubuntu1~24.04.1
```

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

```
root@ip-172-31-35-118:~# 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
215ed5a63843: Pull complete
094bfcb4db7a: Pull complete
4df791be4da6: Pull complete
97a8e80e60c2: Pull complete
7be47dbb02a7: Pull complete
61c8ff67f948: Pull complete
2737e808f9c3: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:93f94e0ea6148cd02d52378049dad85d0f2d6c90c485578317f76addf2af9f3d
Status: Downloaded newer image for sonarqube:lts-community
0e7e47e937ef69b1dc61726d0234ac5c130e172f224f31aee7f3f12afd51b302
```

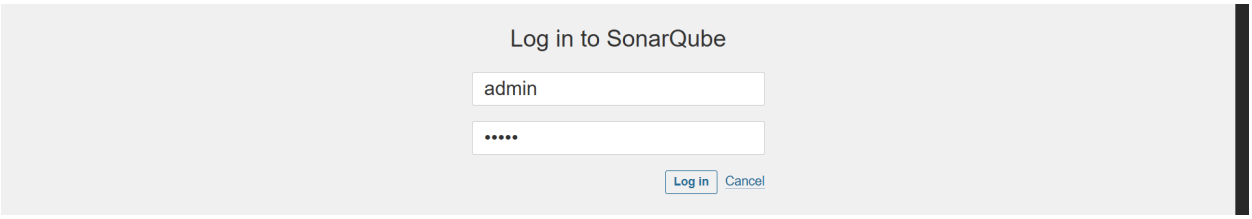
Access SonarQube from your browser:

```
http://<EC2-Public-IP-Address>:9000
http://18.233.115.206:9000
```

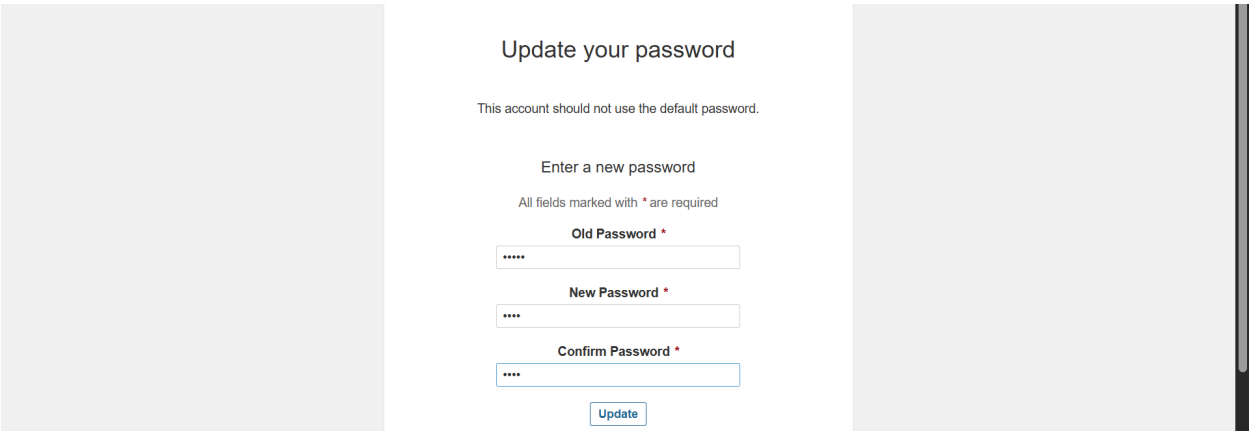


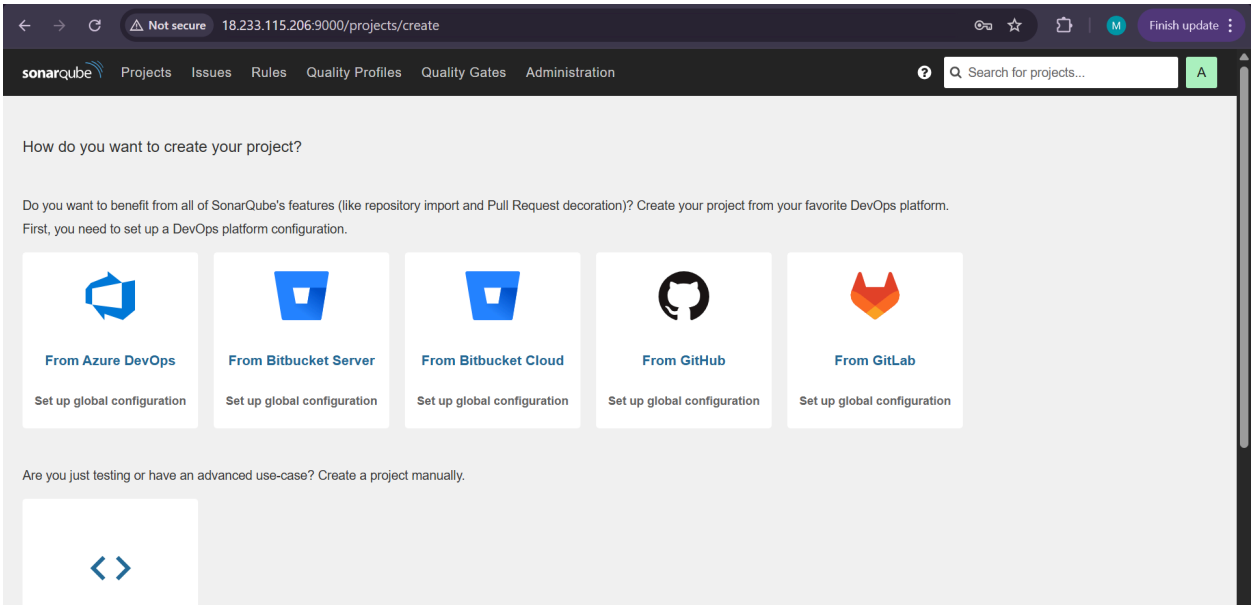
Default credentials:

- Username: admin
- Password: admin



After logging in, change the default password.(root)





2C - Install Trivy

Install Trivy vulnerability scanner:

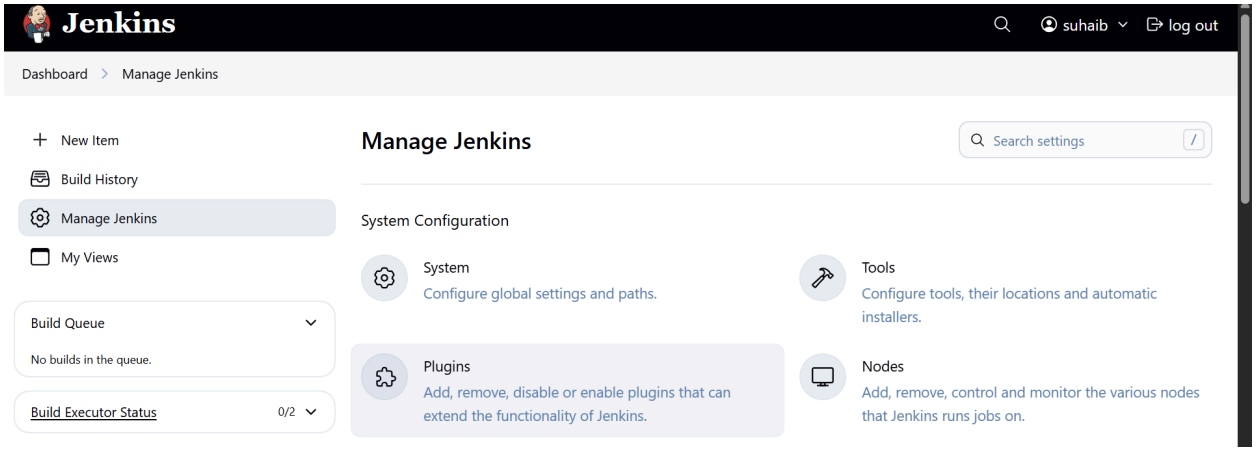
```
sudo apt-get install wget apt-transport-https gnupg lsb-release
-y
wget -qO -
https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg
--dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null
echo "deb [signed-by=/usr/share/keyrings/trivy.gpg]
https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc)
main" | sudo tee -a /etc/apt/sources.list.d/trivy.list
sudo apt-get update
sudo apt-get install trivy -y
trivy version
```

```
root@ip-172-31-35-118:~# trivy version
Version: 0.62.1
root@ip-172-31-35-118:~#
```

Step 3: Configure Jenkins Plugins and Tools

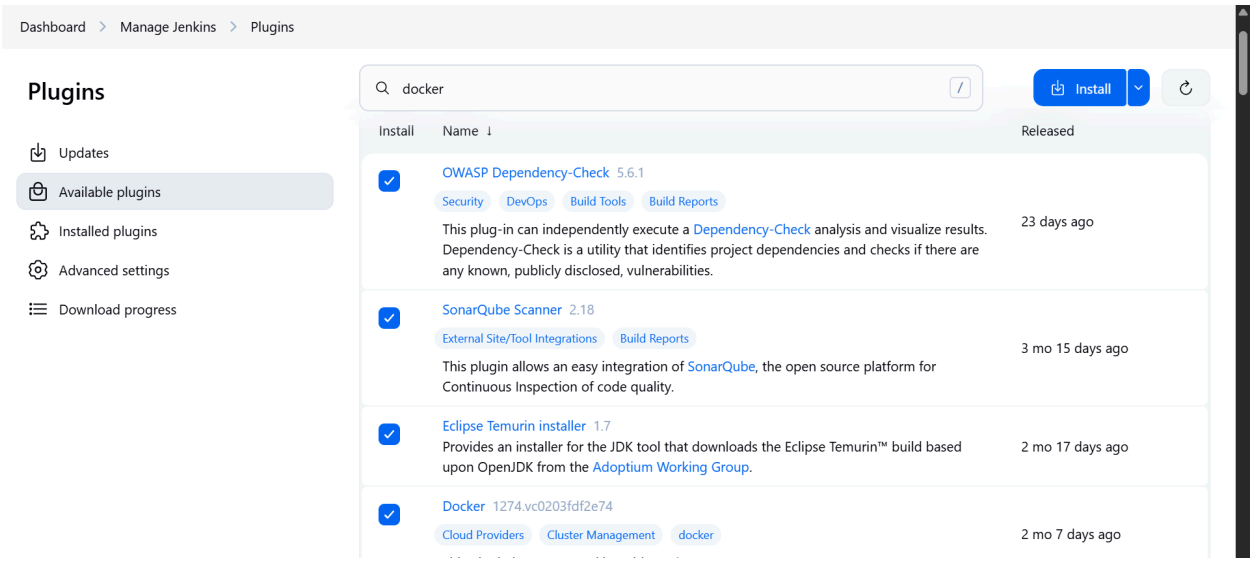
3A - Install Required Plugins

Go to Manage Jenkins → Plugins → Available Plugins and install:



1. OWASP Dependency-Check
2. SonarQube Scanner
3. Eclipse Temurin Installer

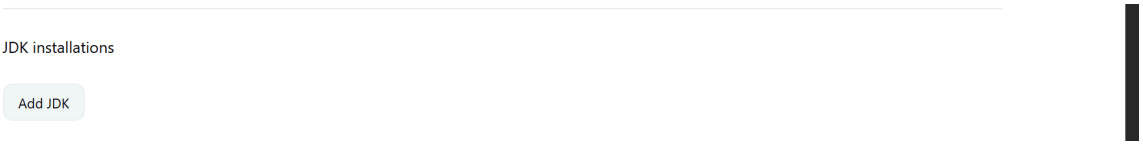
4. Docker plugins (Docker, Docker Commons, Docker Pipeline, Docker API, docker-build-step)



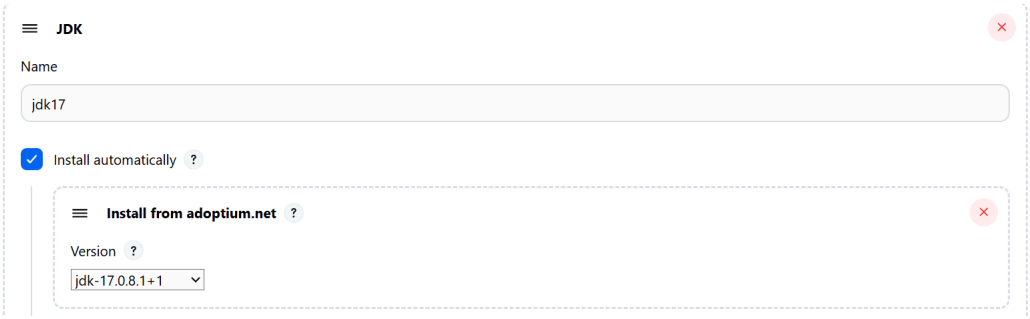
Restart after installation

3B - Configure Java in Global Tool Configuration

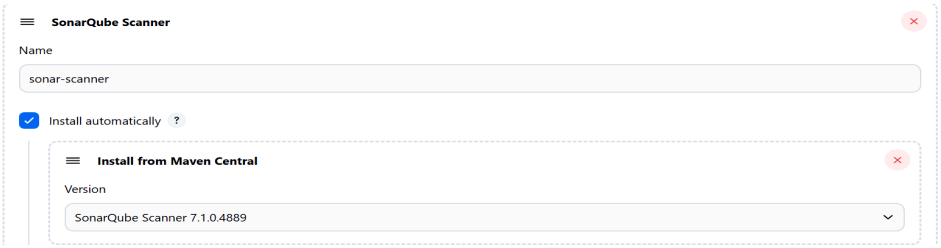
Go to Manage Jenkins → Tools:



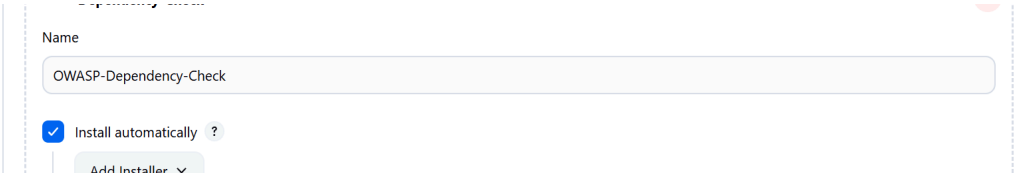
1. Add JDK
   Name: jdk17
   - Install automatically: Check
   - Select "Install from adoptium.netr"
   - Version: JDK-17



2. Add SonarQube Scanner
   Name: sonar-scanner
   - Install automatically: Check
   - Version: Latest



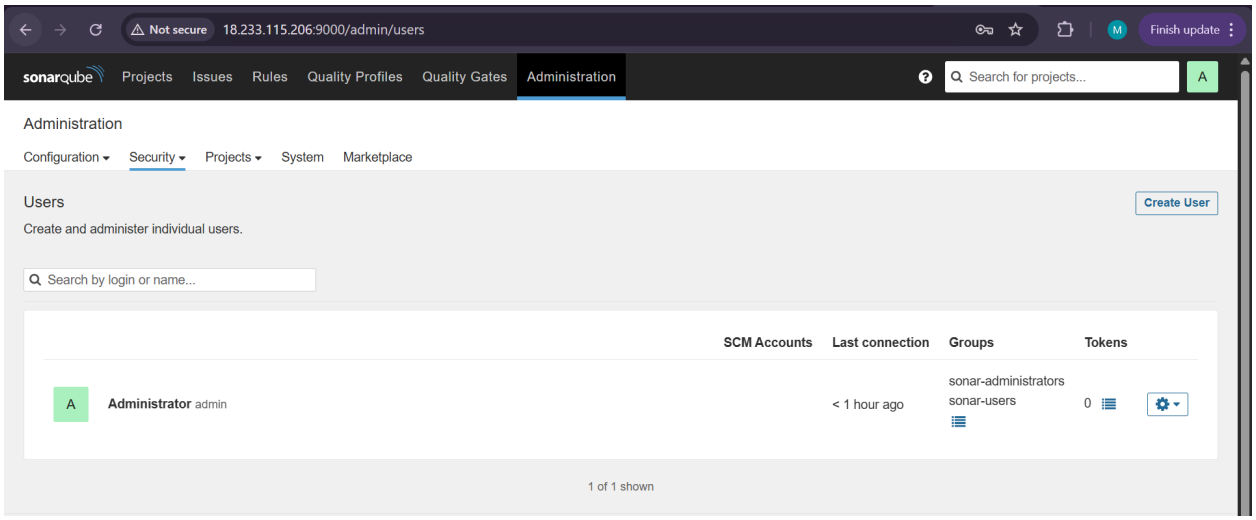
3. Add OWASP Dependency-Check
   Name: OWASP-Dependency-Check
   - Install automatically: Check
   - Version: Latest



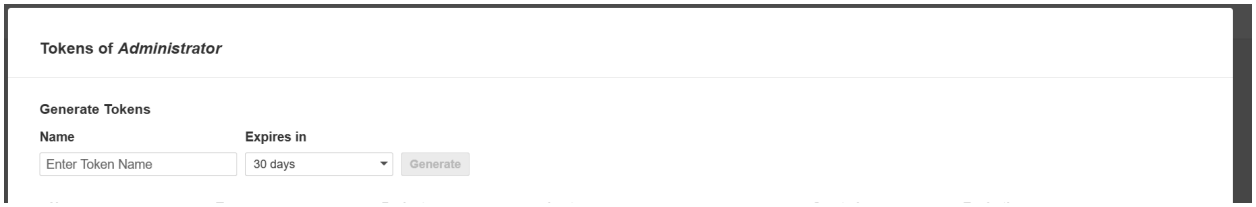
4. Apply and Save

Step 4: Configure SonarQube Server

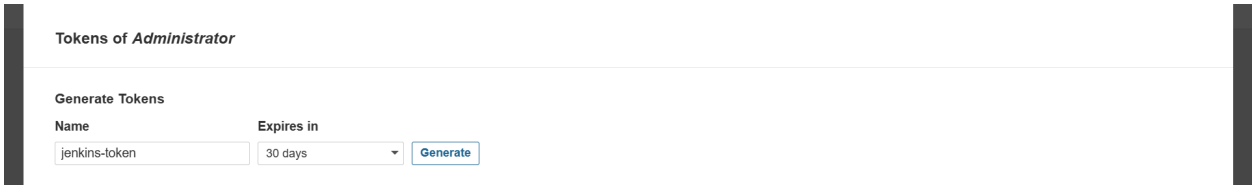
1. Create a token in SonarQube:
   - Go to SonarQube → Administration → Security → Users



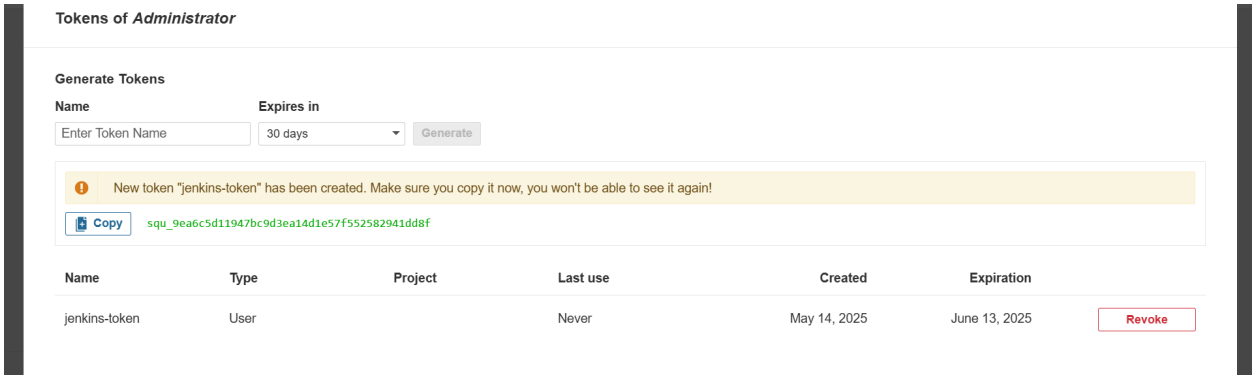
- Click on Tokens and Update Token



- Give it a name and click on Generate Token

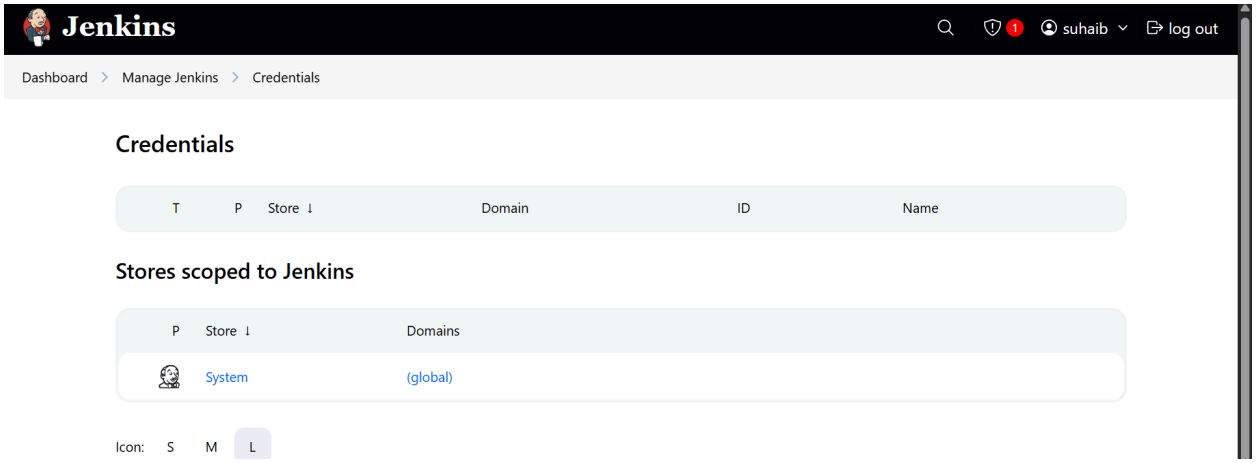


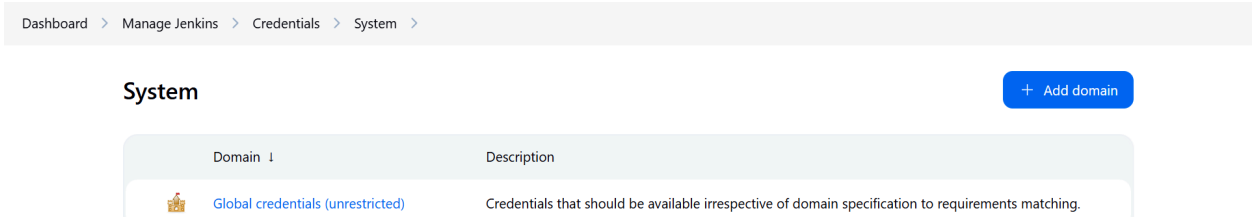
- Copy the generated token

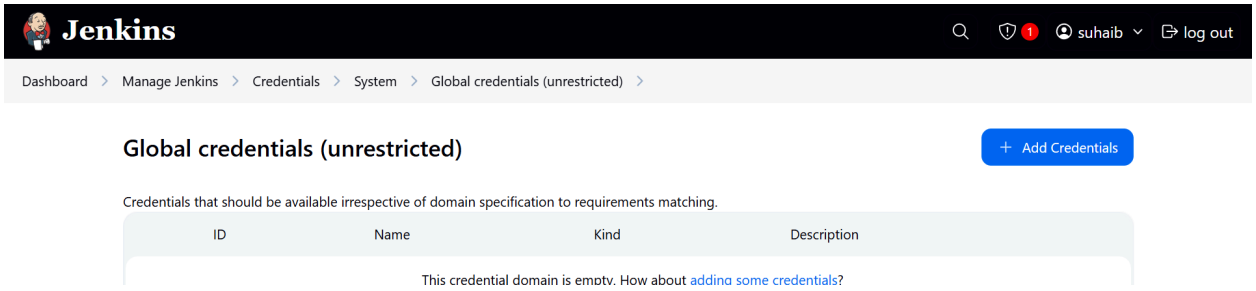


```
squ_9ea6c5d11947bc9d3ea14d1e57f552582941dd8f
```

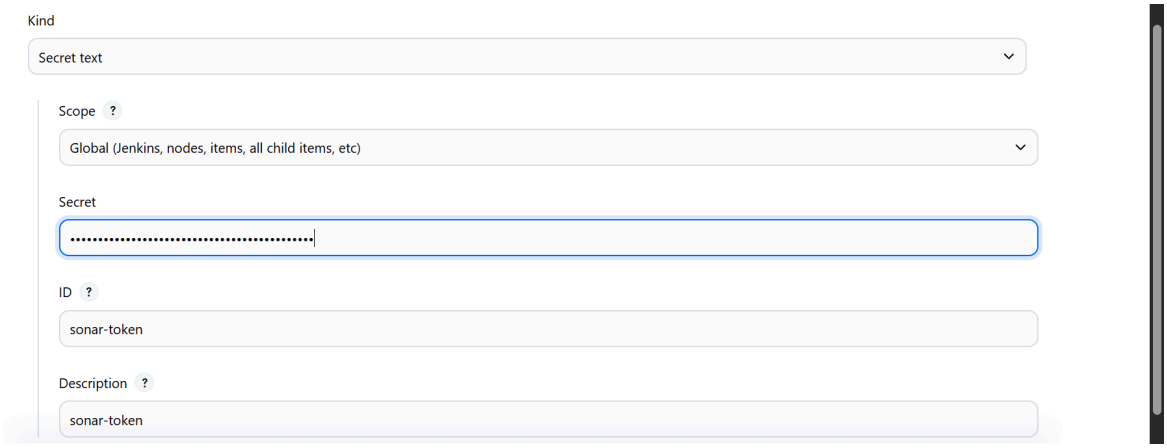
2. Add the token to Jenkins:
    - Go to Jenkins → Manage Jenkins → Credentials → System → Global Credentials → Add Credentials → Secret Text

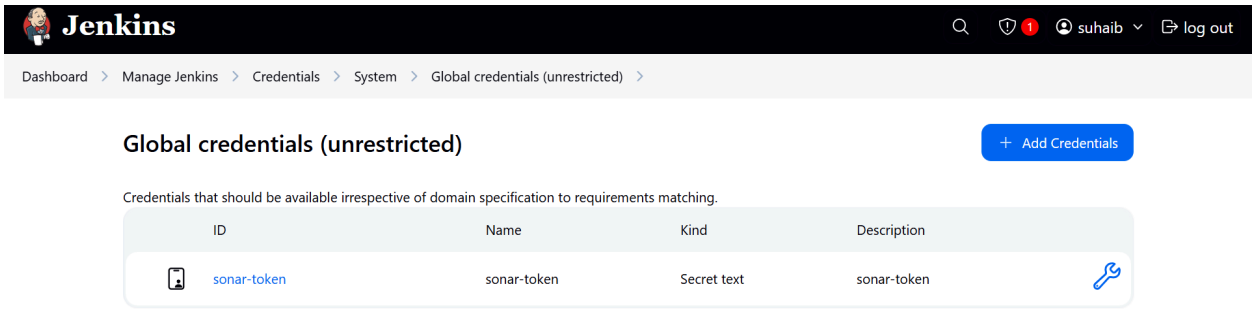




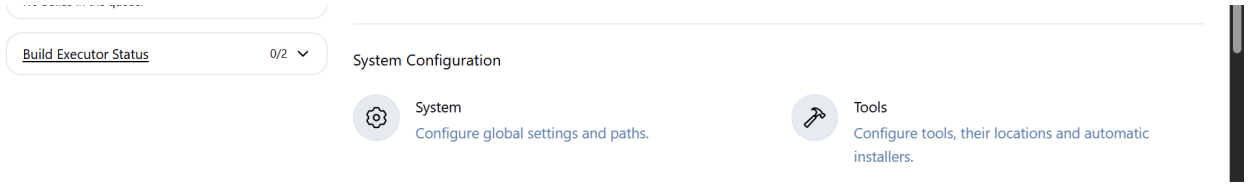


- ID: sonar-token
- Description: sonar-token
- Secret: Paste the token you copied
- Click on Create

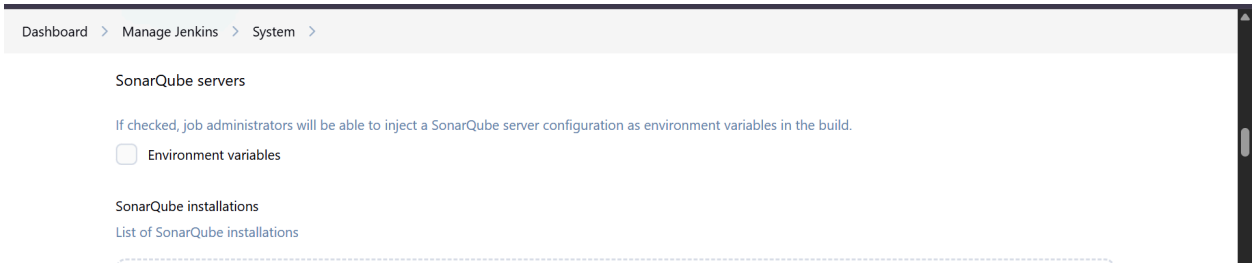




3. Configure SonarQube in Jenkins:
    - Go to Jenkins → Manage Jenkins → System

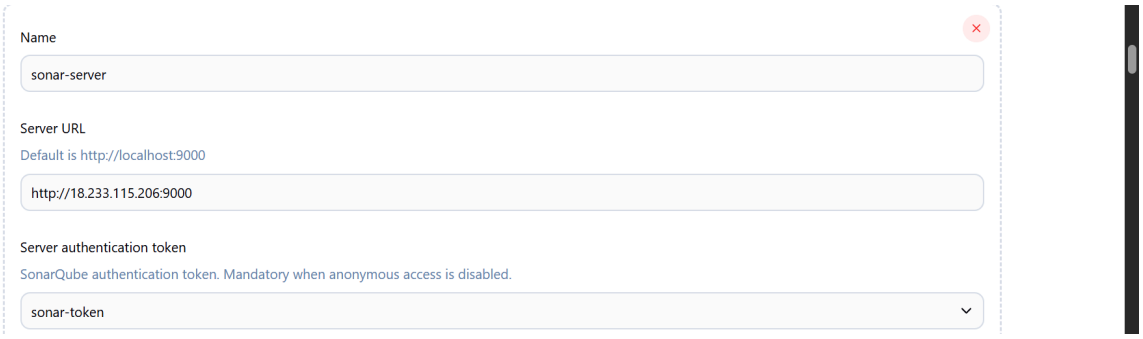


    - Scroll down to SonarQube servers
    - Click on Add SonarQube

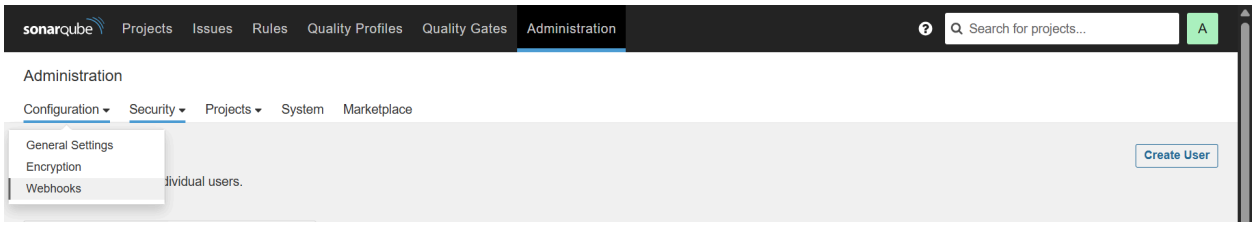


    - Name: sonar-server
    - Server URL: http://<EC2-Public-IP>:9000

- Server authentication token: Select the sonar-token credential
- Apply and Save

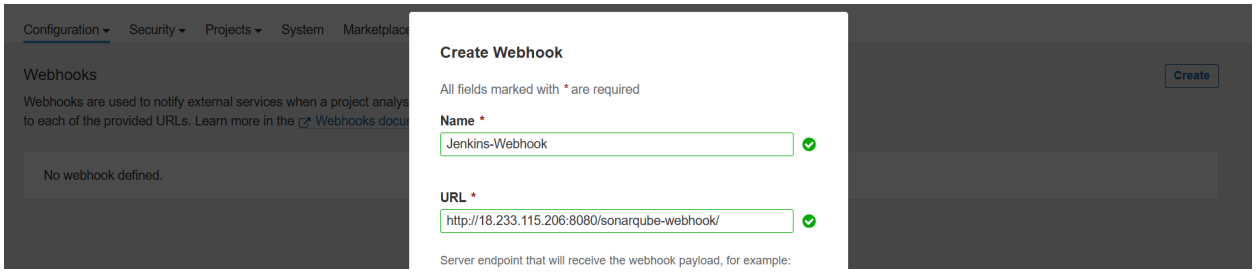


4. Configure Webhook in SonarQube:
   - Go to SonarQube → Administration → Configuration → Webhooks

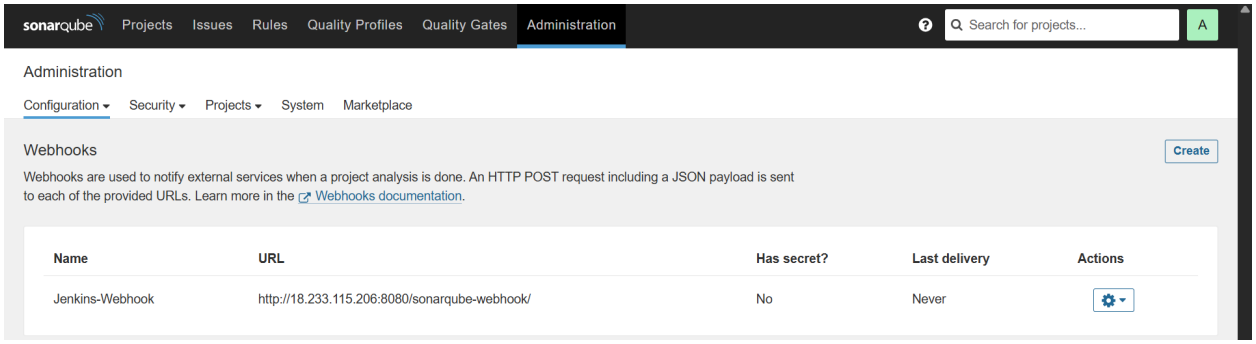


   - Click on Create

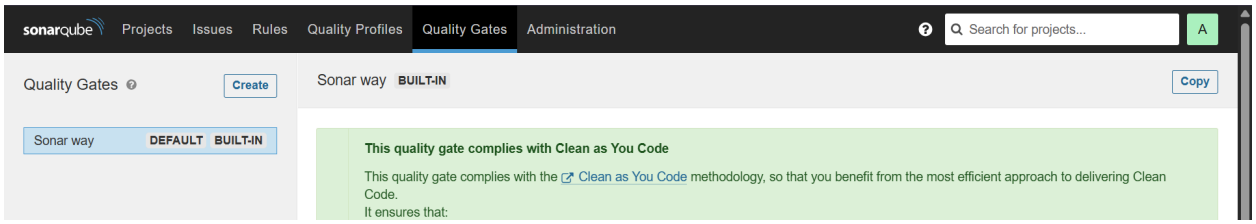
   - Name: Jenkins Webhook
   - URL: http://<Jenkins-Public-IP>:8080/sonarqube-webhook/



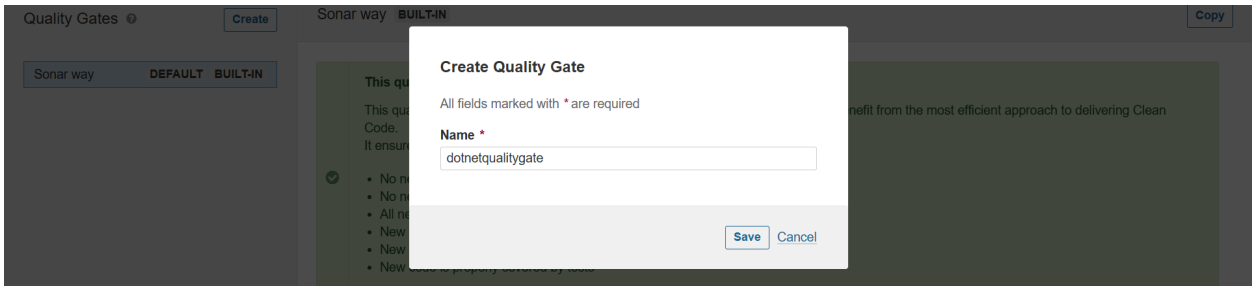
   - Click on Create



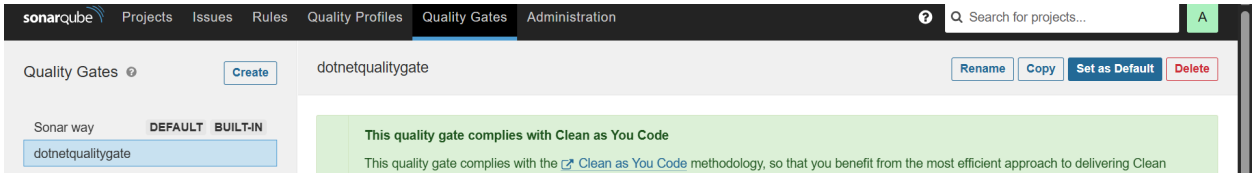
5. Create a Quality Gate in SonarQube:
    - Go to SonarQube → Quality Gates
    - Click on Create

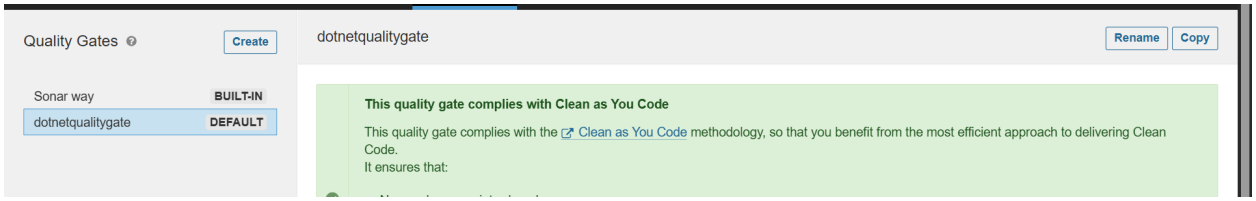


    - Give it a name and set appropriate conditions



    - Make it the default quality gate





Step 5: Install Make Package

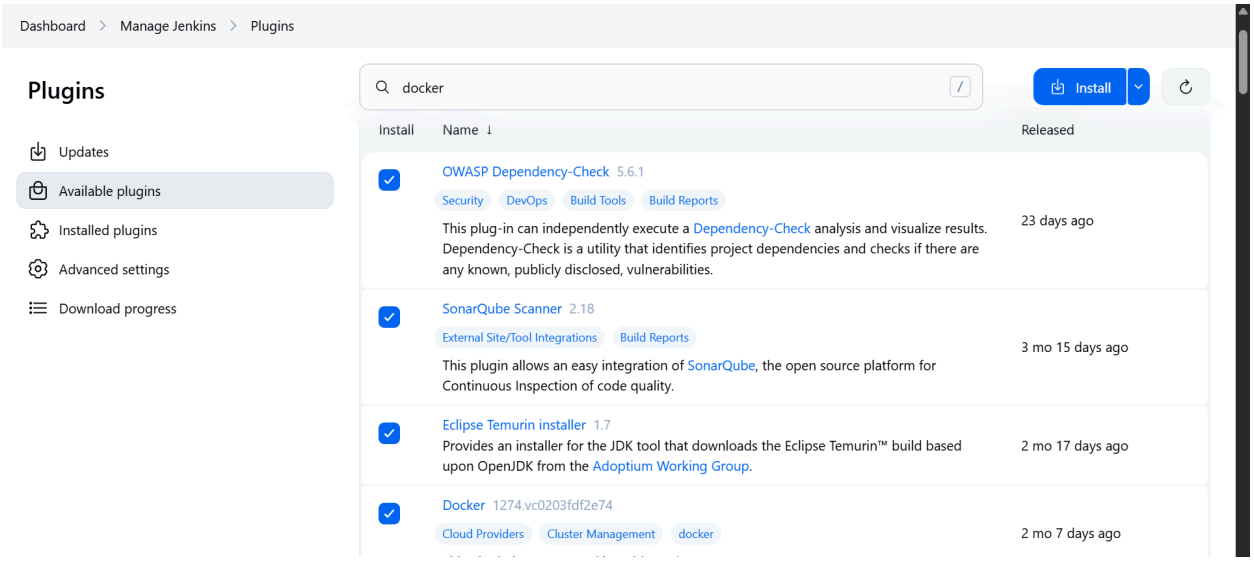
Install the make package:

```
sudo apt install make
# to check version install or not
make -v
```

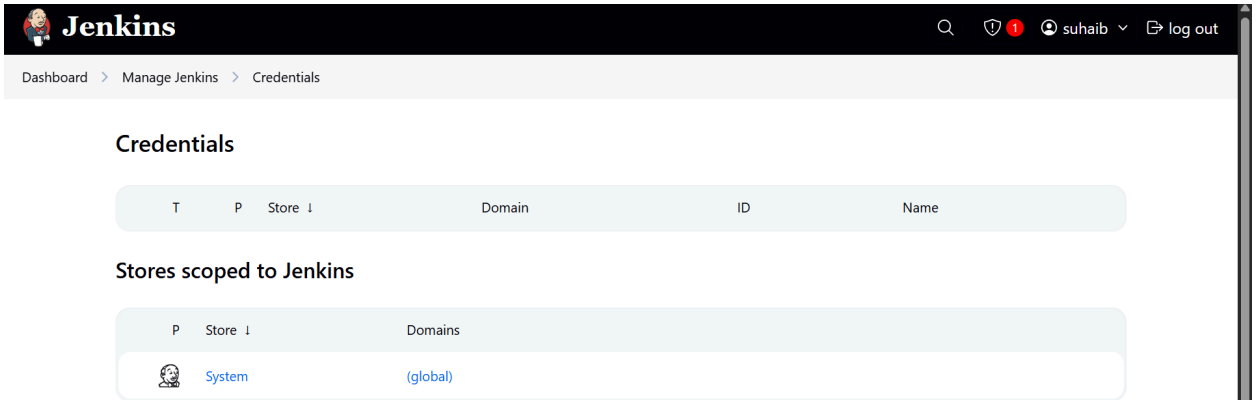
```
ubuntu@ip-172-31-35-118:~$ make -v
GNU Make 4.3
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
ubuntu@ip-172-31-35-118:~$
```

Step 6: Configure Docker Credentials in Jenkins

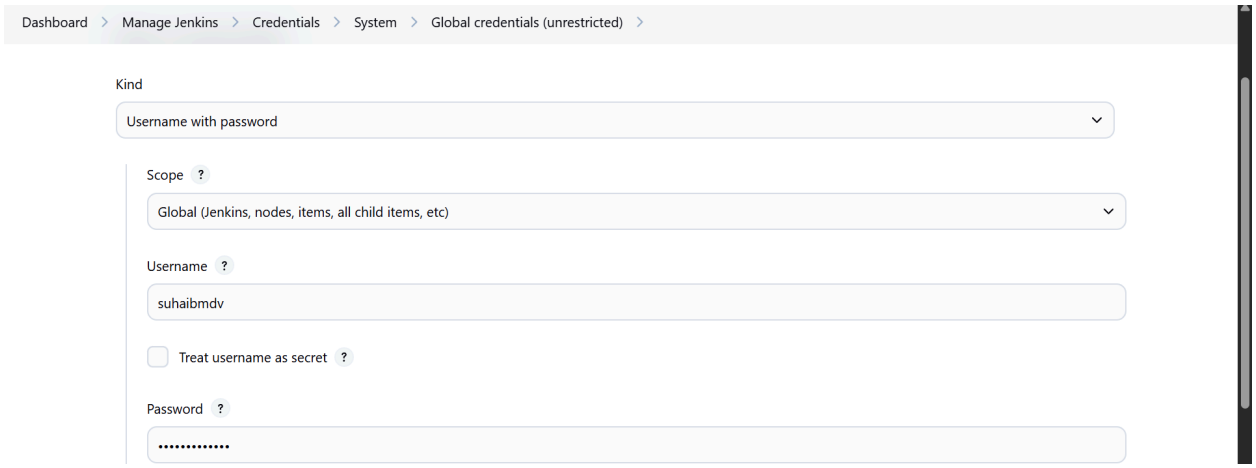
1. Install Docker plugins in Jenkins:
    - Docker
    - Docker Commons
    - Docker Pipeline
    - Docker API
    - docker-build-step

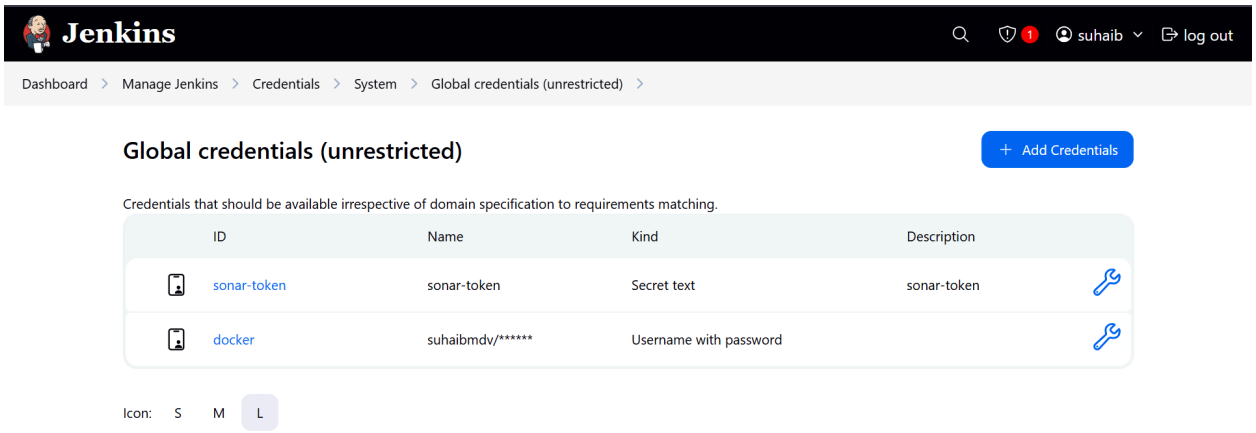


2. Add DockerHub credentials:
    - Go to Jenkins → Manage Jenkins → Credentials → Add Credentials

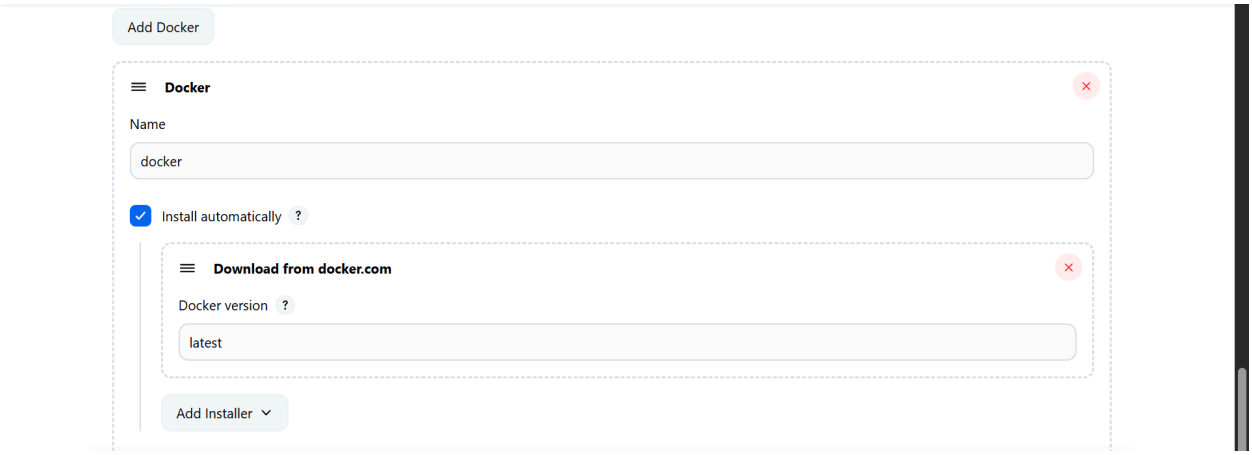


- Kind: Username with password
- ID: docker
- Username: Your DockerHub username
- Password: Your DockerHub password
- Click on Create





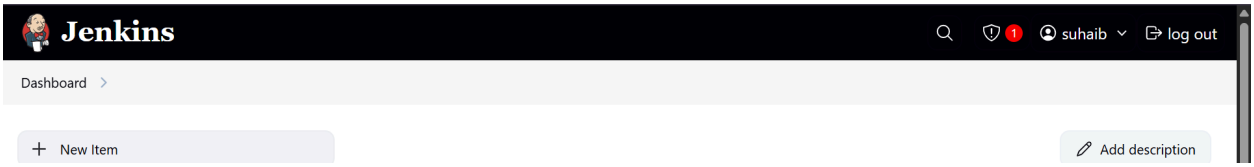
3. Install Docker Tool:
    - Go to Jenkins → Manage Jenkins → Tools→ Docker and Install it



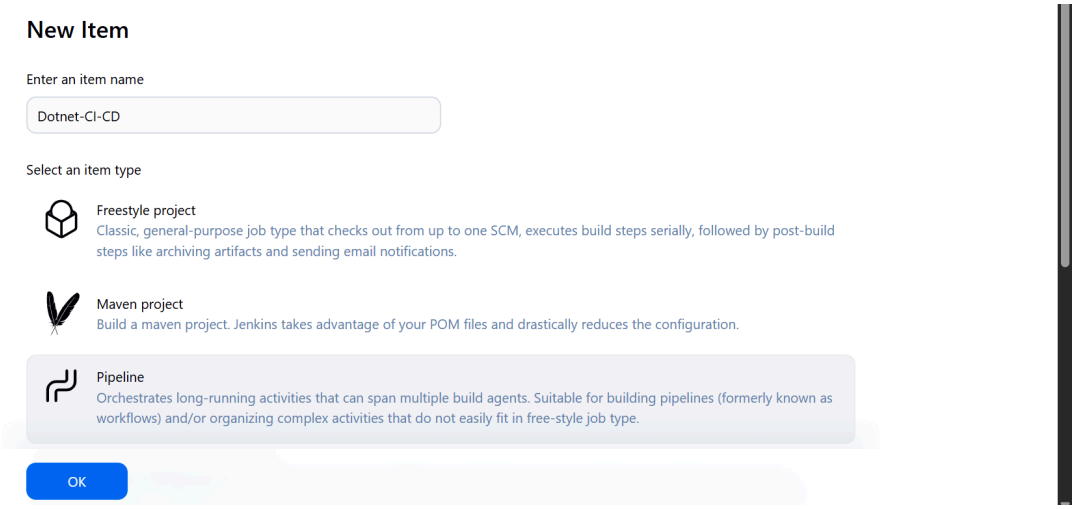
Step 7: Create a Jenkins Pipeline

Create a new Pipeline job in Jenkins:

1. Click on "New Item"



2. Enter "Dotnet-CI-CD" as the name
3. Select "Pipeline" and click OK

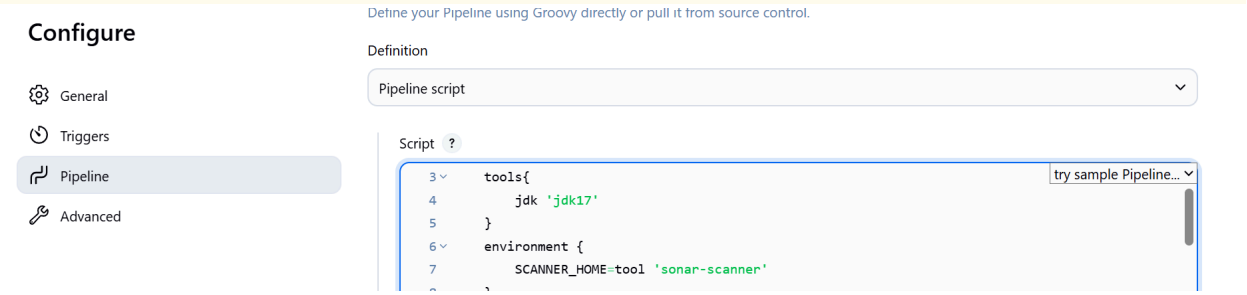


4. In the Pipeline section, select "Pipeline script" and add the following script:

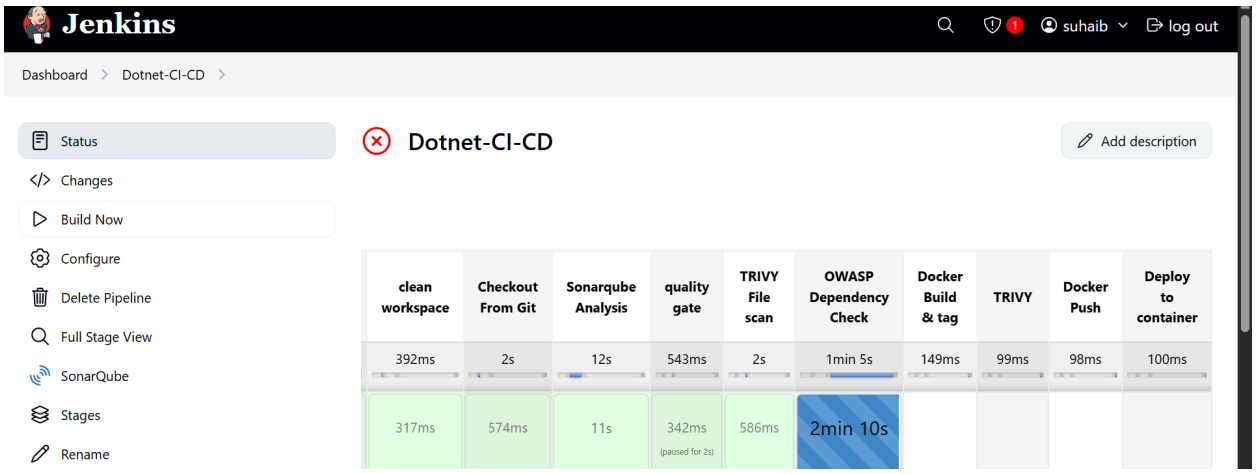
```
pipeline{
    agent any
    tools{
        jdk 'jdk17'
    }
    environment {
        SCANNER_HOME=tool 'sonar-scanner'
    }
    stages {
        stage('clean workspace'){
            steps{
                cleanWs()
            }
        }
        stage('Checkout From Git'){
            steps{
                git branch: 'main', url:
'https://github.com/suhaib-md/Jenkins-CI-CD-.NET-WebApp-DevSecOp
s'
            }
        }
        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: 'OWASP-Dependency-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-image-report.txt"
            }
        }
        stage("Docker Push"){
            steps{
                script{
```

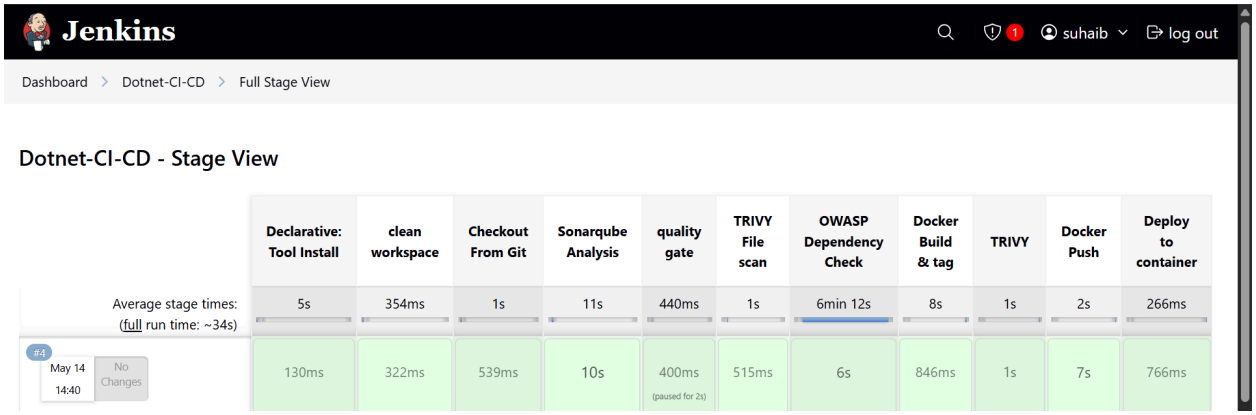
```
                    withDockerRegistry(credentialsId: 'docker', toolName: 'docker'){
                        sh "make push"
                    }
                }
            }
        }
        stage("Deploy to container"){
            steps{
                sh "docker run -d --name dotnet -p 5000:5000 sevenajay/dotnet-monitoring:latest"
            }
        }
    }
}
```



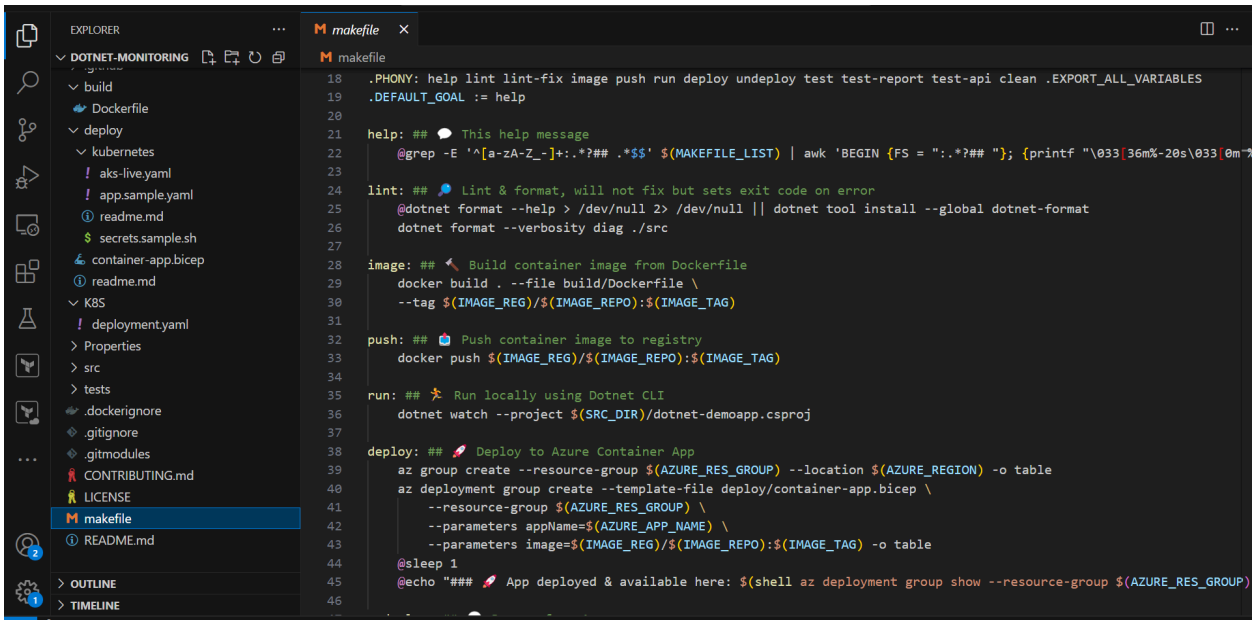
Click on Build now and see if the pipeline works till now



Wait for the pipeline to get executed

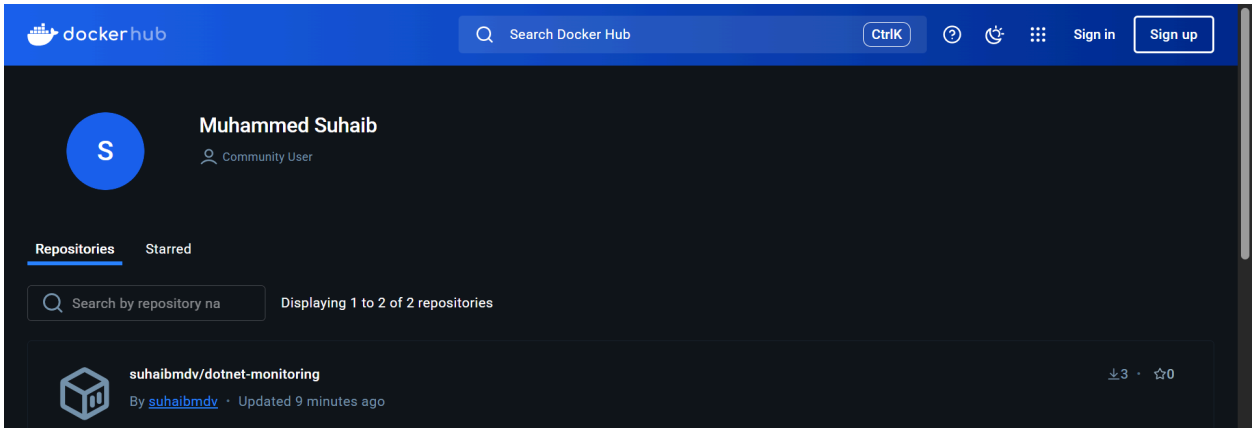


In the makefile, we already defined some conditions to build, tag and push images to dockerhub.

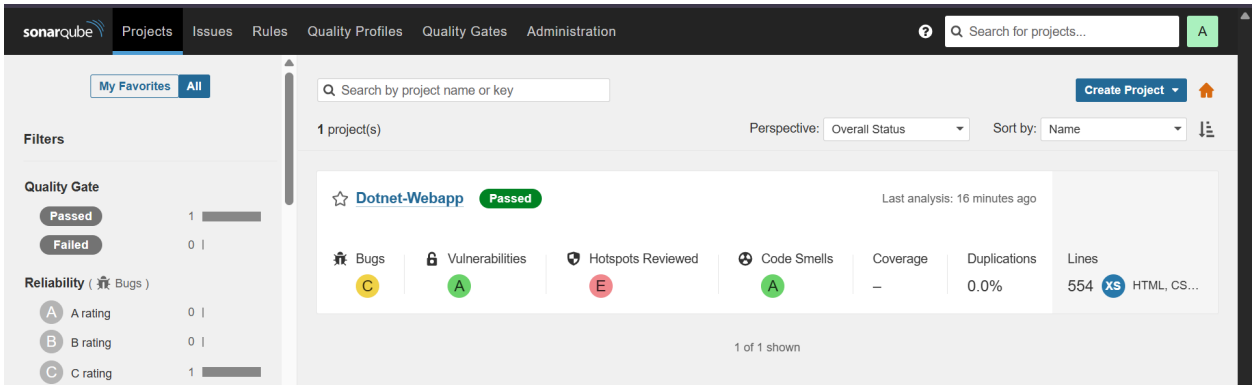


That's why we are using make image and make a push in the place of docker build -t and docker push

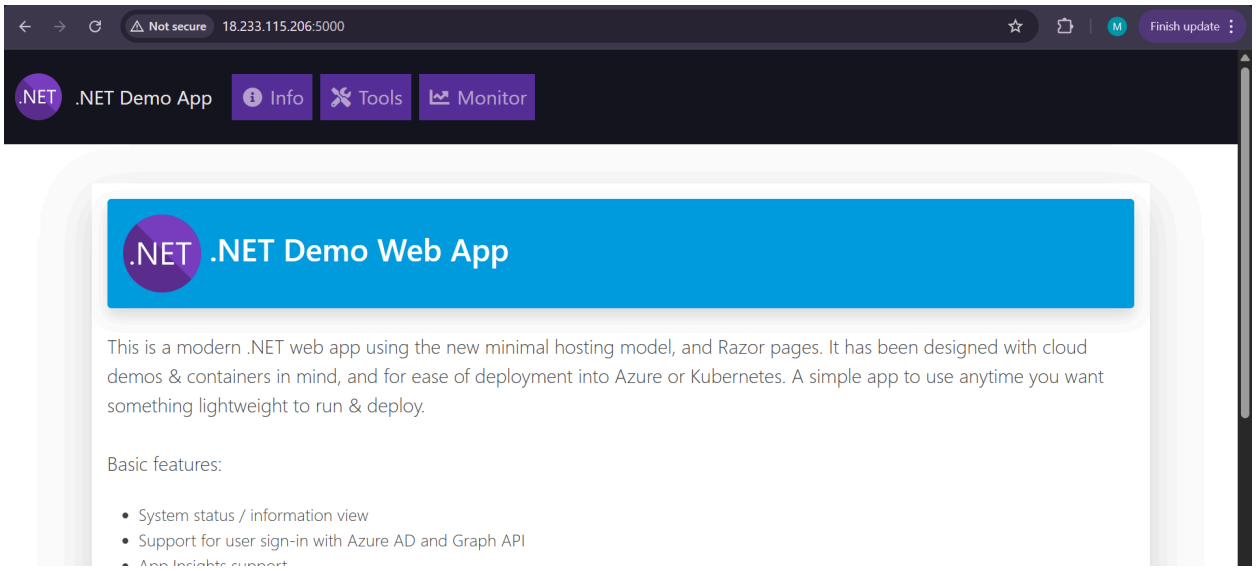
We can see that the image is created in docker hub

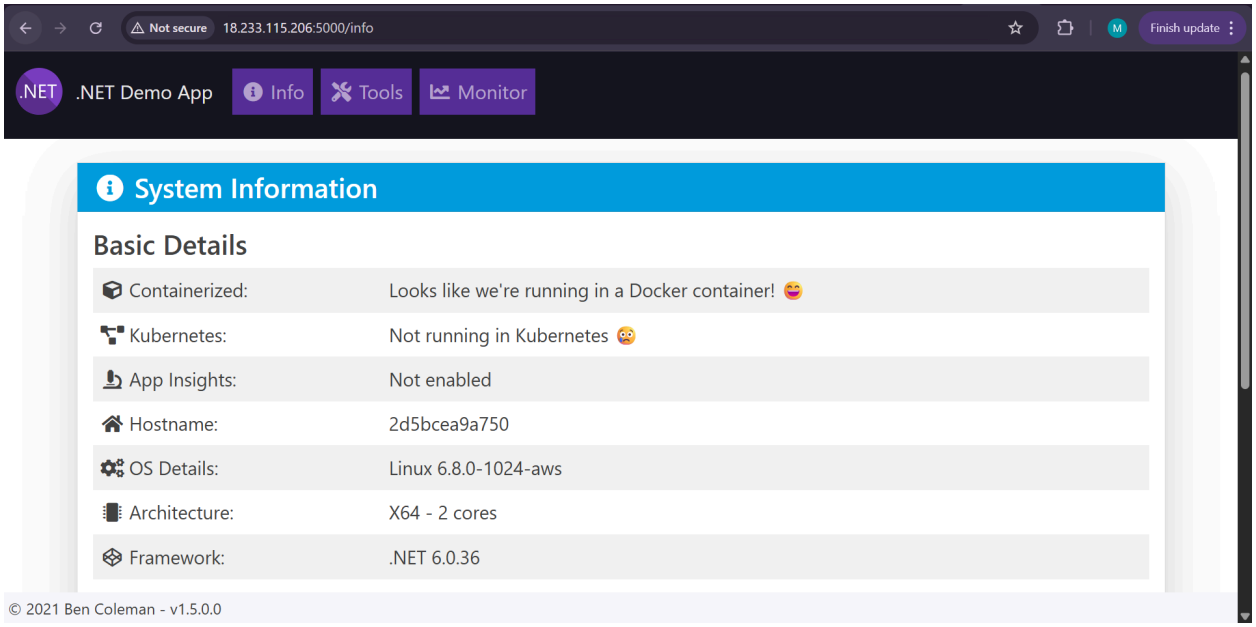


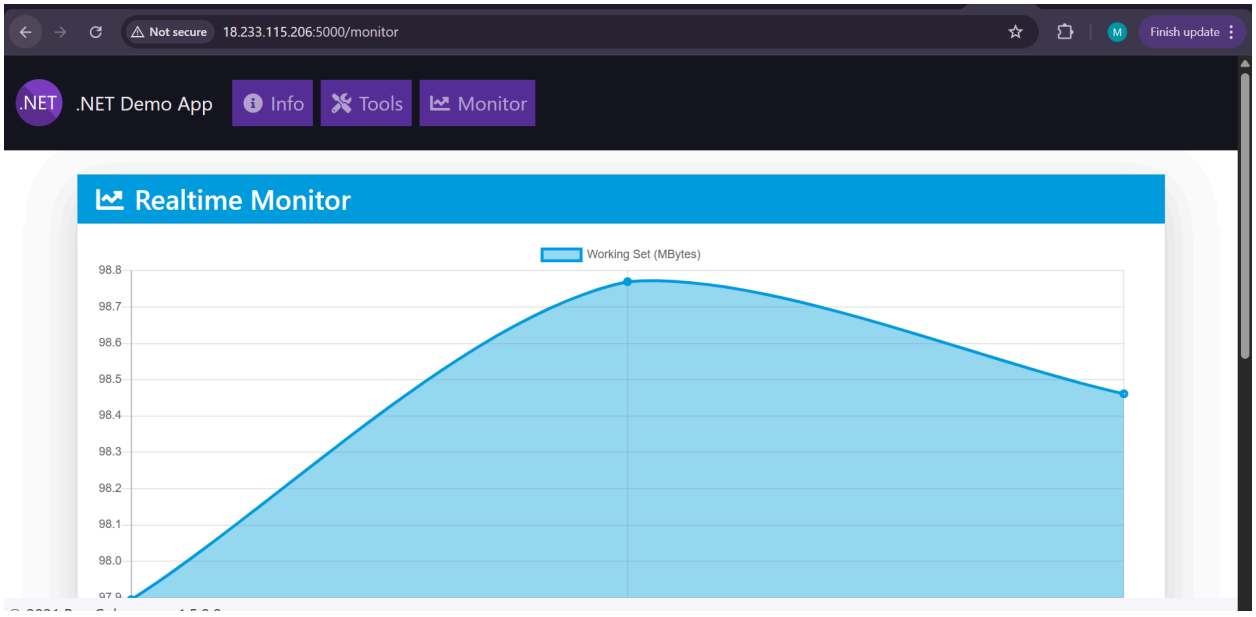
We can get the details in the SonarQube Projects page

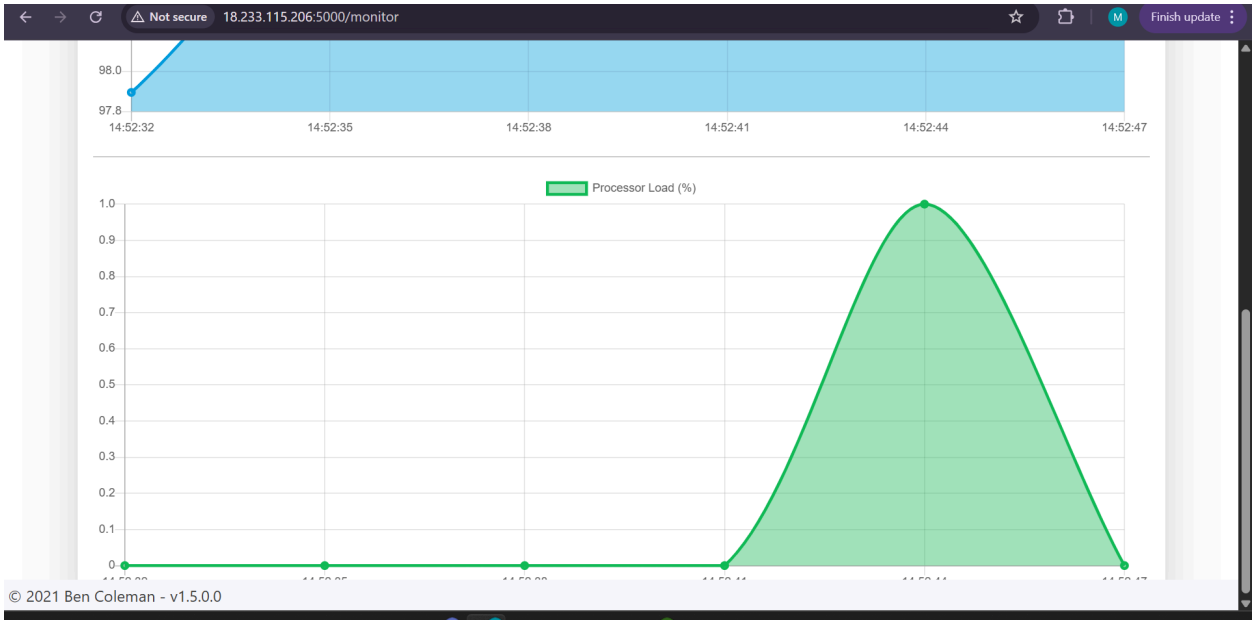


We can access the application on port 5000 of the ip address <public-ip of jenkins:5000>




Not secure
18.233.115.206:5000/info
Finish update
.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:
2d5bcea9a750
OS Details:
Linux 6.8.0-1024-aws
Architecture:
X64 - 2 cores
Framework:
.NET 6.0.36
© 2021 Ben Coleman - v1.5.0.0


Not secure
18.233.115.206:5000/monitor
Finish update
.NET
.NET Demo App
Info
Tools
Monitor
Realtime Monitor
Working Set (MBytes)
98.8
98.7
98.6
98.5
98.4
98.3
98.2
98.1
98.0



Step 8: Setup Kubernetes Cluster

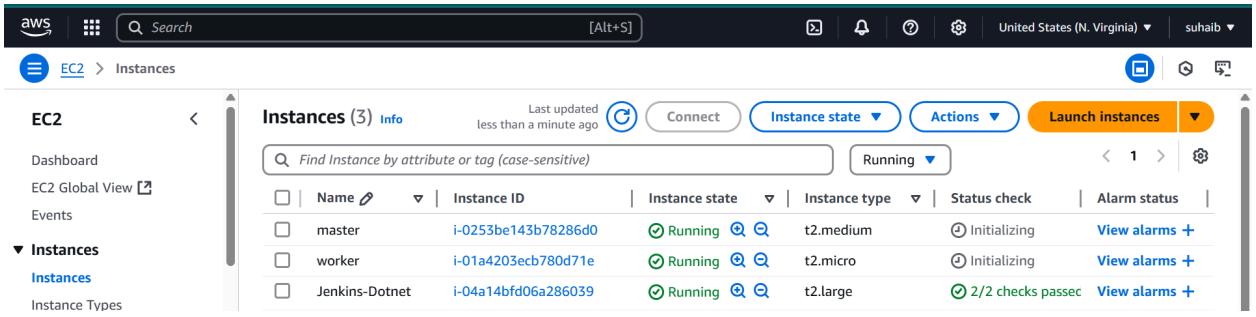
Step 8.1: Install kubectl on Jenkins Server

```
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 kubectl
/usr/local/bin/kubectl
kubectl version --client
```

```
Reading state information... Done
44 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (8.5.0-2ubuntu10.6).
curl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 44 not upgraded.
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0   2107      0 --:--:-- --:--:-- --:--:--  2123
100 57.3M  100 57.3M    0     0  95.4M      0 --:--:-- --:--:-- --:--:-- 95.4M
Client Version: v1.33.0
Kustomize Version: v5.6.0
ubuntu@ip-172-31-35-118:~$
```

Step 8.2: Setup Kubernetes Master Node

Launch a new Ubuntu 20.04 instance and run the following commands:

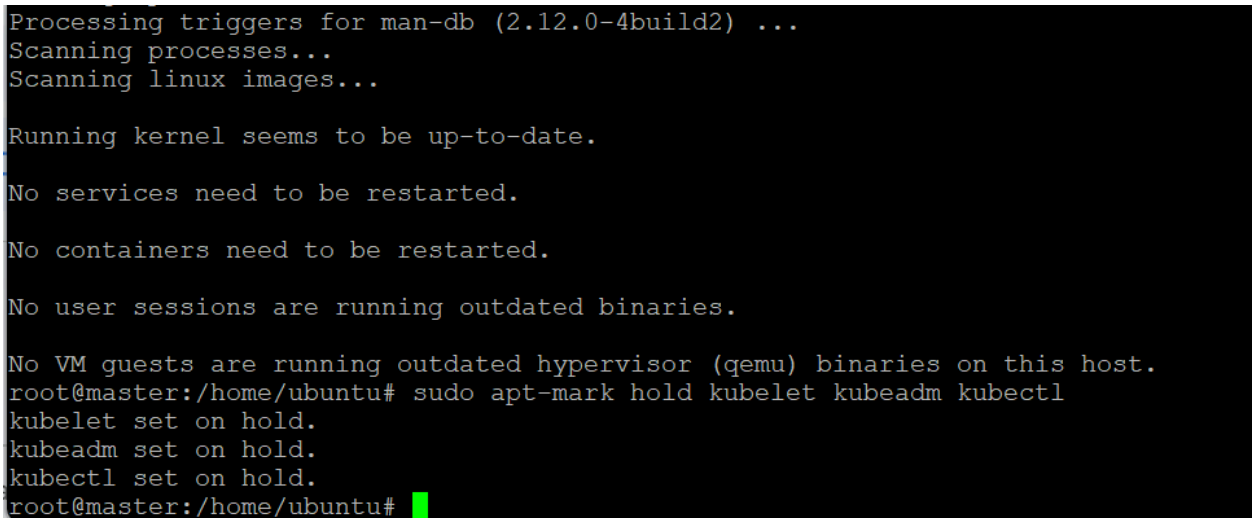



```
sudo su
hostname master
bash
clear
```


```
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
```

```
Setting up docker.io (26.1.3-0ubuntu1~24.04.1) ...
info: Selecting GID from range 100 to 999 ...
info: Adding group `docker' (GID 113) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /us
r/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/li
b/systemd/system/docker.socket.
Processing triggers for dbus (1.14.10-4ubuntu4.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.
```

```
# Install Kubernetes packages
sudo apt install -y apt-transport-https ca-certificates curl gpg

# Add Kubernetes GPG key
sudo mkdir -p /etc/apt/keyrings
curl -fsSL
https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo
gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg


# Add Kubernetes repo
echo "deb
[signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] \
https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /" | \
sudo tee /etc/apt/sources.list.d/kubernetes.list > /dev/null

# Update and install Kubernetes tools
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```



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

```
To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as
 root:

kubeadm join 172.31.16.125:6443 --token 6fu5zm.2r3bviuqups9nvgw \
        --discovery-token-ca-cert-hash sha256:55d72b504d2b77522c04c96a14fae837d4
0fac3ce04e216b4fcd4cc58fd0ec1d
root@master:/home/ubuntu#
```

Note the `kubeadm join` command that will be displayed after initialization. You'll need this for the worker node.

```
kubeadm join 172.31.29.105:6443 --token zun96o.fz68qzkq0mru8vb9
\
        --discovery-token-ca-cert-hash
sha256:4cb14662f40e4e4d423c3c4cf0a01ea33edd6a2b01836f0ce6a4f87cd
f729f3f
```

```
# Exit from root and run as a regular user
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Setup flannel

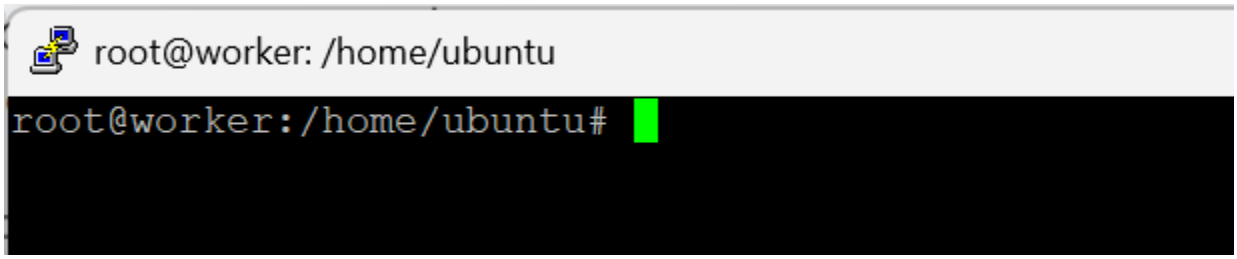
```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Document
ation/kube-flannel.yml
```

```
ubuntu@ip-172-31-29-105:~$ mkdir -p $HOME/.kube
ubuntu@ip-172-31-29-105:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/con
fig
ubuntu@ip-172-31-29-105:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-29-105:~$ kubectl apply -f https://raw.githubusercontent.com/co
reos/flannel/master/Documentation/kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
ubuntu@ip-172-31-29-105:~$
```

Step 8.3: Setup Kubernetes Worker Node

Launch another Ubuntu 20.04 instance and run the following commands:

```
sudo su
hostname worker
bash
clear
```



Setup Docker

```
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
```

```
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service -
 /usr/lib/systemd/system/ubuntu-fan.service.
Setting up docker.io (26.1.3-0ubuntu1~24.04.1) ...
info: Selecting GID from range 100 to 999 ...
info: Adding group `docker' (GID 113) ...
```

Setup Kubernetes

```
# Install Kubernetes packages
sudo apt install -y apt-transport-https ca-certificates curl gpg

# Add Kubernetes GPG key
sudo mkdir -p /etc/apt/keyrings
curl -fsSL
https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo
gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

# Add Kubernetes repo
echo "deb
[signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] \
https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /" | \
sudo tee /etc/apt/sources.list.d/kubernetes.list > /dev/null

# Update and install Kubernetes tools
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Use the join command from the master node output which has a structure like the following:

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

```
kubeadm join 172.31.29.105:6443 --token zun96o.fz68qzkq0mru8vb9
\
        --discovery-token-ca-cert-hash
sha256:4cb14662f40e4e4d423c3c4cf0a01ea33edd6a2b01836f0ce6a4f87cd
f729f3f
```

```
root@worker:/home/ubuntu# kubeadm join 172.31.29.105:6443 --token zun96o.fz68qzk
q0mru8vb9 \
        --discovery-token-ca-cert-hash sha256:4cb14662f40e4e4d423c3c4cf0a01ea33e
dd6a2b01836f0ce6a4f87cdf729f3f
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system g
et cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.y
aml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/ku
belet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

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

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

root@worker:/home/ubuntu#
```

Step 8.4: Copy Kubernetes Config to Jenkins

Copy the Kubernetes config file from the master node to the Jenkins server:

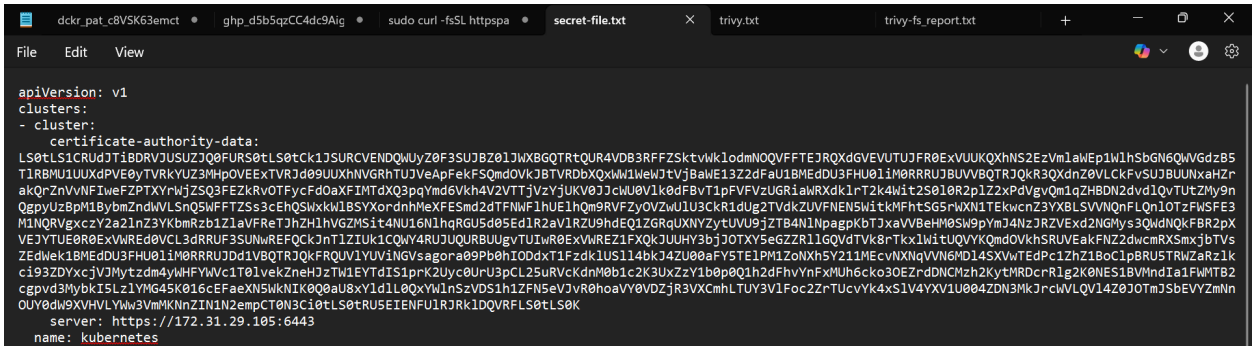
On the Kubernetes master node:

```
cat $HOME/.kube/config
```

1. Copy the output.

2. Save this content on your local machine as `secret-file.txt`.

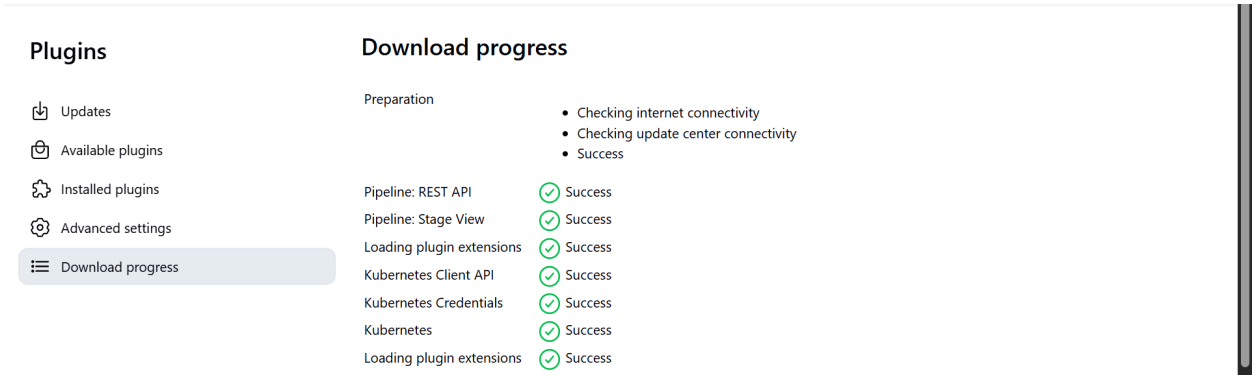
```
ubuntu@ip-172-31-29-105:~$ cat $HOME/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVEND
QWUyZ0F3SUJBZ0lJWXBGQTRtQUR4VDB3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhN
S2EzVmlaWEp1WlhSbGN6QWVGdzB5TlRBMU1UUXdPVE0yTVRkYUZ3MHpOVEExTVRJd09UUXhNVGRhTUJV
eApFekFSQmdOVkJBTVRDbXQxWW1WeWJtVjBaWE13Z2dFaU1BMEdDU3FHU0liM0RRRUJBUVVBQTRJQkR3
QXdnZ0VLCkFvSUJBUUNxaHZrakQrZnVvNFIweFZPTXYrWjZSQ3FEZkRvOTFycFdOaXFIMTdXQ3pqYmd6
Vkh4V2VTTjVzYjUKV0JJcWU0Vlk0dFBvT1pFVFVzUGRiaWRXdklrT2k4Wit2S0l0R2plZ2xPdVgvQm1q
ZHBDN2dvdlQvTUtZMy9nQgpyUzBpM1BybmZndWVLSnQ5WFFTZSs3cEhQSWxkWlBSYXordnhMeXFESmd2
dTFNWFlhUElhQm9RVFZyOVZwUlU3CkR1dUg2TVdkZUVFNEN5WitkMFhtSG5rWXN1TEkwcnZ3YXBLSVVN
QnFLQnlOTzFWSFE3M1NQRVgxczY2a2lnZ3YKbmRzb1ZlaVFReTJhZHlhVGZMSit4NU16NlhqRGU5d05E
dlR2aVlRZU9hdEQ1ZGRqUXNYZytUVU9jZTB4NlNpagpKbTJxaVVBeHM0SW9pYmJ4NzJRZVExd2NGMys3
OWdNQkFBR2pXVEJYTUE0R0ExVWRFd0VCL3dRRUF3SUNwREFOQkJnTlZIUk1CQWY4RUJUQURBUUgvTUIw
```



Step 8.5: Install Kubernetes Plugin in Jenkins

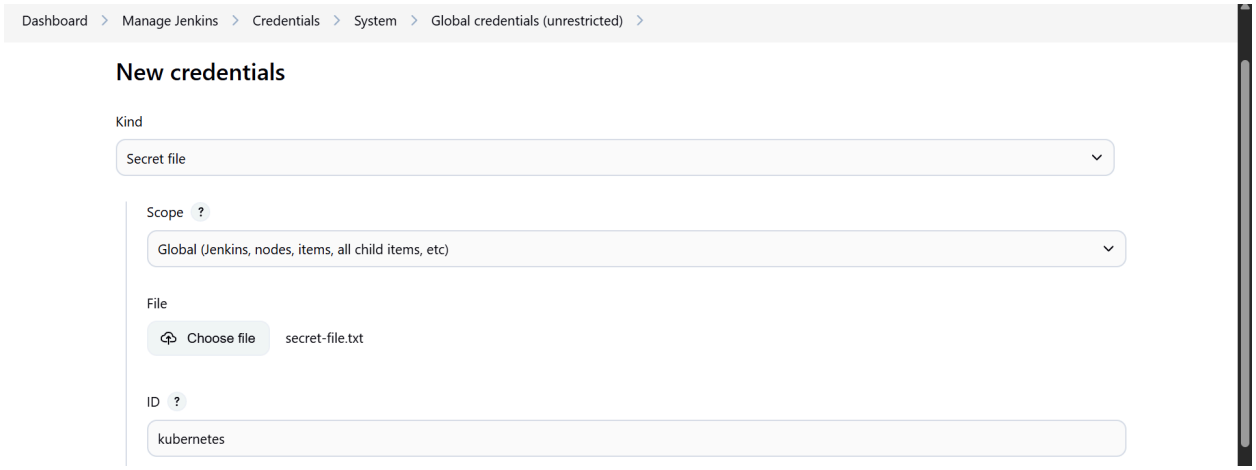
1. Go to Jenkins → Manage Jenkins → Plugins → Available plugins

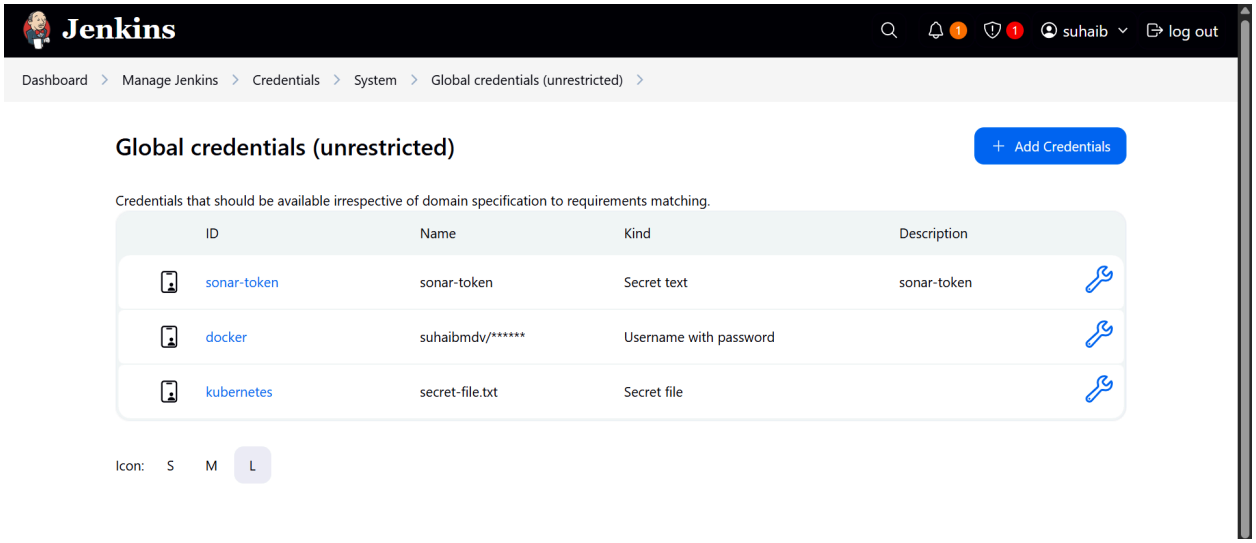
2. Search for "Kubernetes" and install the plugin. Also add kubernetes credentials



3. Add Kubernetes credentials:

    - Go to Jenkins → Manage Jenkins → Credentials → Add Credentials
    - Kind: Secret file
    - ID: kubernetes
    - File: Upload the `secret-file.txt` file we saved earlier
    - Click on Create





Step 9: Update Jenkins Pipeline for Kubernetes Deployment

1. Create a `deployment.yaml` file in the K8S directory of your git repository:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dotnet-app-deployment
spec:
  replicas: 1 # Set the desired number of replicas
  selector:
    matchLabels:
```

```
      app: dotnet-app
  template:
    metadata:
      labels:
        app: dotnet-app
    spec:
      containers:
        - name: dotnet-app
          image: suhaibmdv/dotnet-monitoring:latest # Replace
with your Docker image name
          ports:
            - containerPort: 5000 # Expose the port specified in
your Dockerfile

---
apiVersion: v1
kind: Service
metadata:
  name: dotnet-app-service
spec:
  selector:
    app: dotnet-app
  ports:
    - protocol: TCP
      port: 80 # Service port
      targetPort: 5000 # Port in the container
  type: NodePort # Use NodePort service type
```

2. Update the Jenkins pipeline to include Kubernetes deployment:

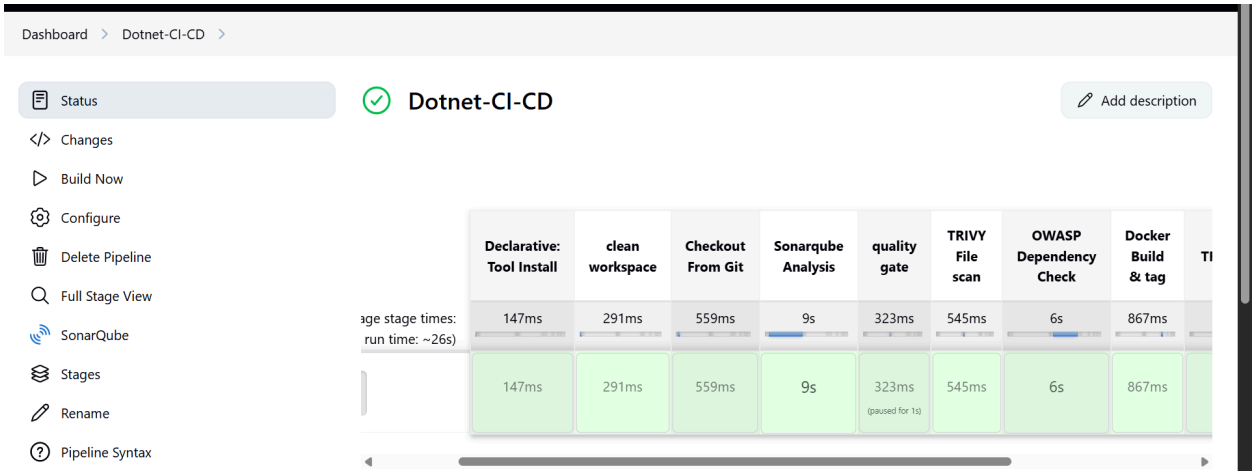
```
pipeline{
    agent any
    tools{
        jdk 'jdk17'
    }
    environment {
```

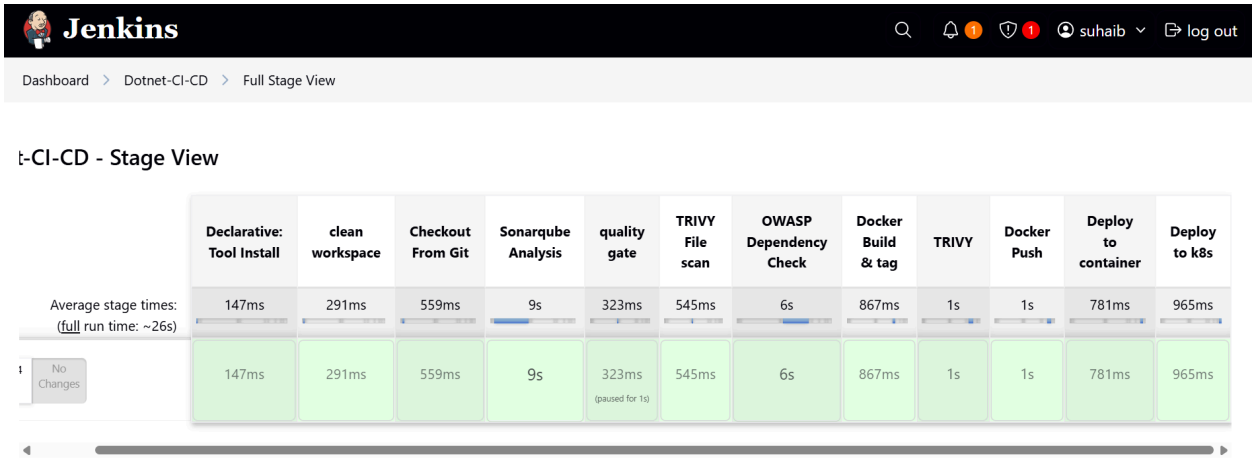
```
        SCANNER_HOME=tool 'sonar-scanner'
    }
    stages {
        stage('clean workspace'){
            steps{
                cleanWs()
            }
        }
        stage('Checkout From Git'){
            steps{
                git branch: 'main', url:
'https://github.com/suhaib-md/Jenkins-CI-CD-.NET-WebApp-DevSecOp
s'
            }
        }
        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: 'OWASP-Dependency-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
suhaibmdv/dotnet-monitoring:latest > trivy.txt"
            }
        }
        stage("Docker Push"){
            steps{
                script{
                   withDockerRegistry(credentialsId: 'docker',
toolName: 'docker'){
                       sh "make push"
                    }
                }
            }
        }
        stage("Deploy to container"){
```

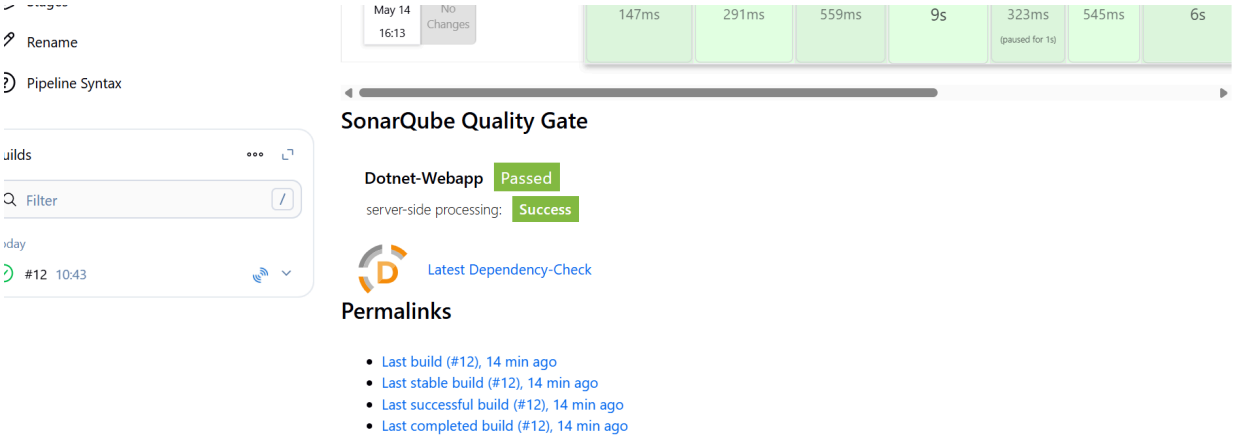
```
            steps{
                sh "docker run -d --name dotnet -p 5000:5000
suhaibmdv/dotnet-monitoring:latest"
            }
        }
        stage('Deploy to k8s'){
            steps{
                dir('K8S') {
                  withKubeConfig(caCertificate: '', clusterName:
'', contextName: '', credentialsId: 'kubernetes', namespace: '',
restrictKubeConfigAccess: false, serverUrl: '') {
                    sh 'kubectl apply -f deployment.yaml'
                  }
                }
            }
        }
    }
}
```

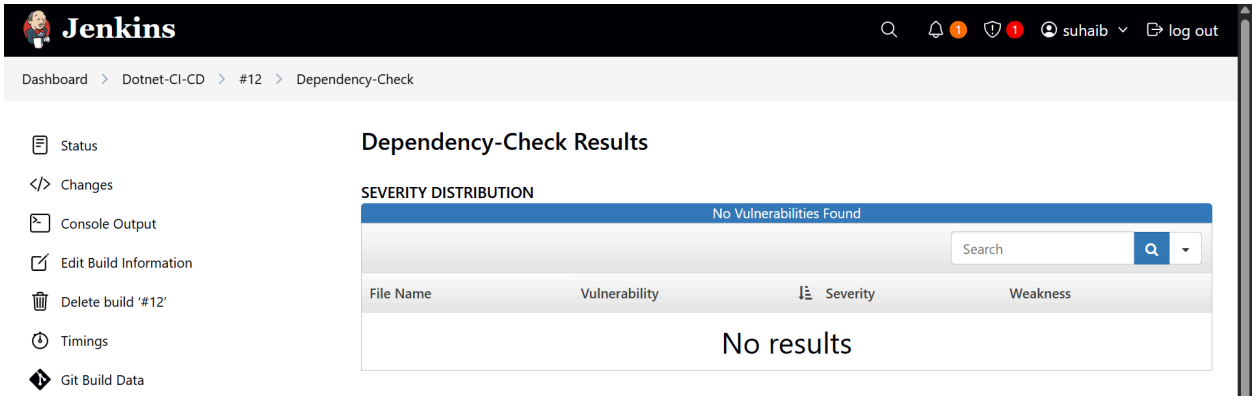
3. Click on Build Now and wait for the pipeline to complete executing



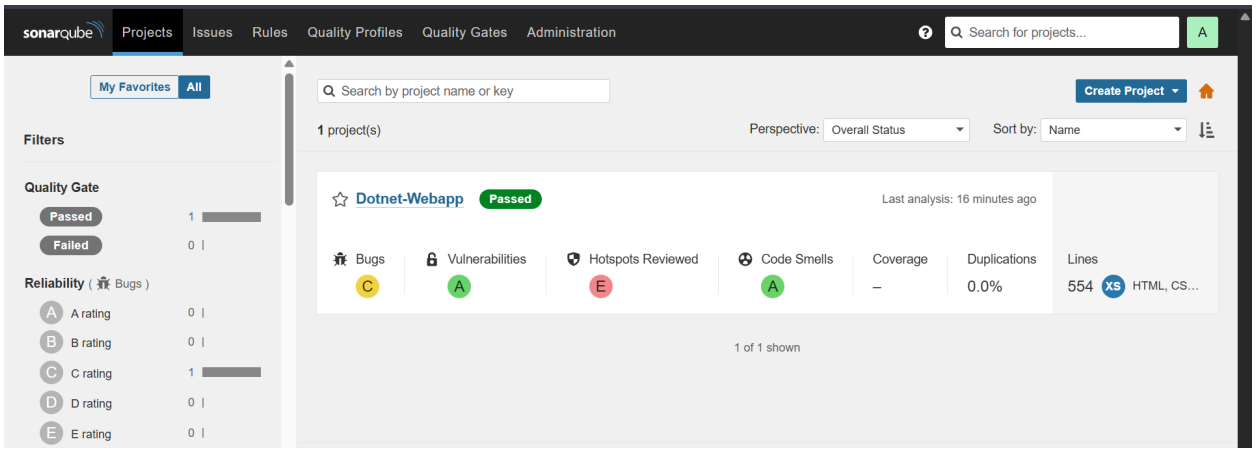


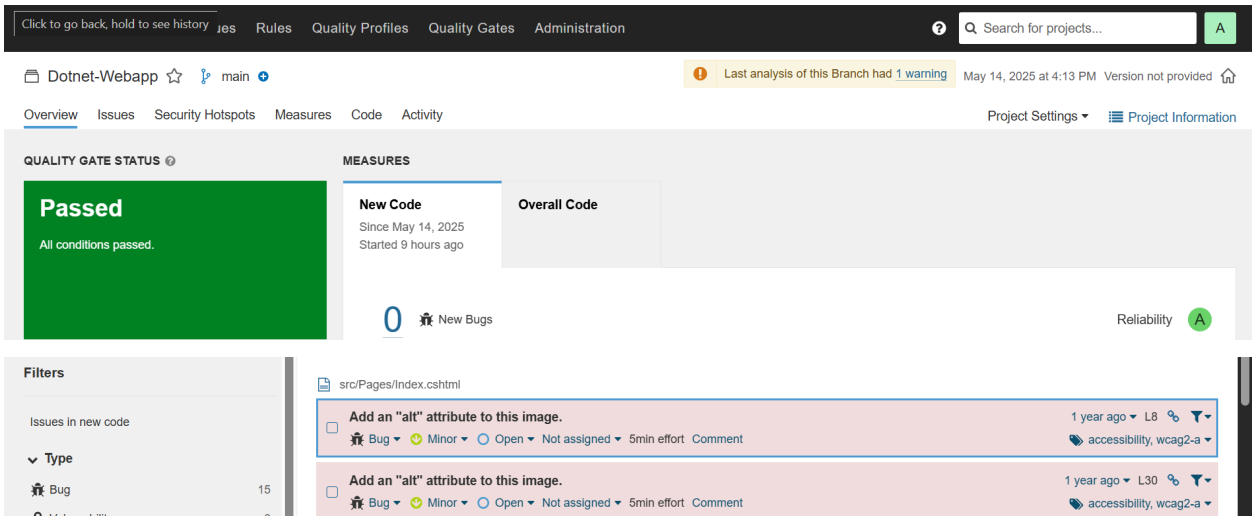
We can look at the OWASP Despendency result at the dependency section





We can look at the Sonarqube Scan results at the Projects section of the Sonarqube dashboard

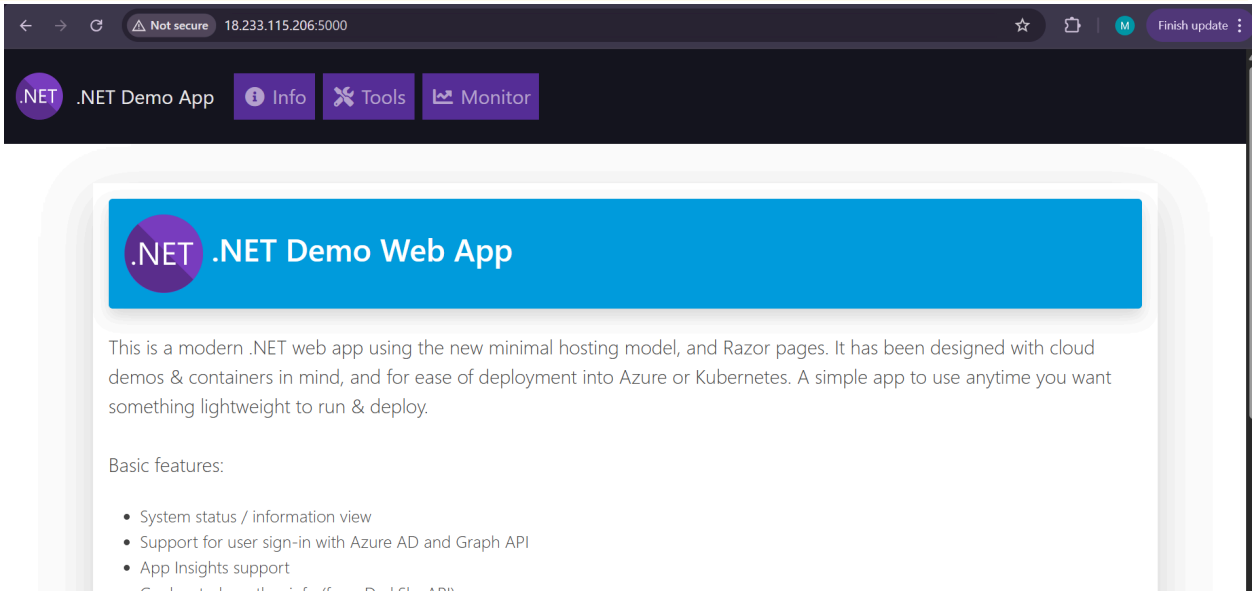


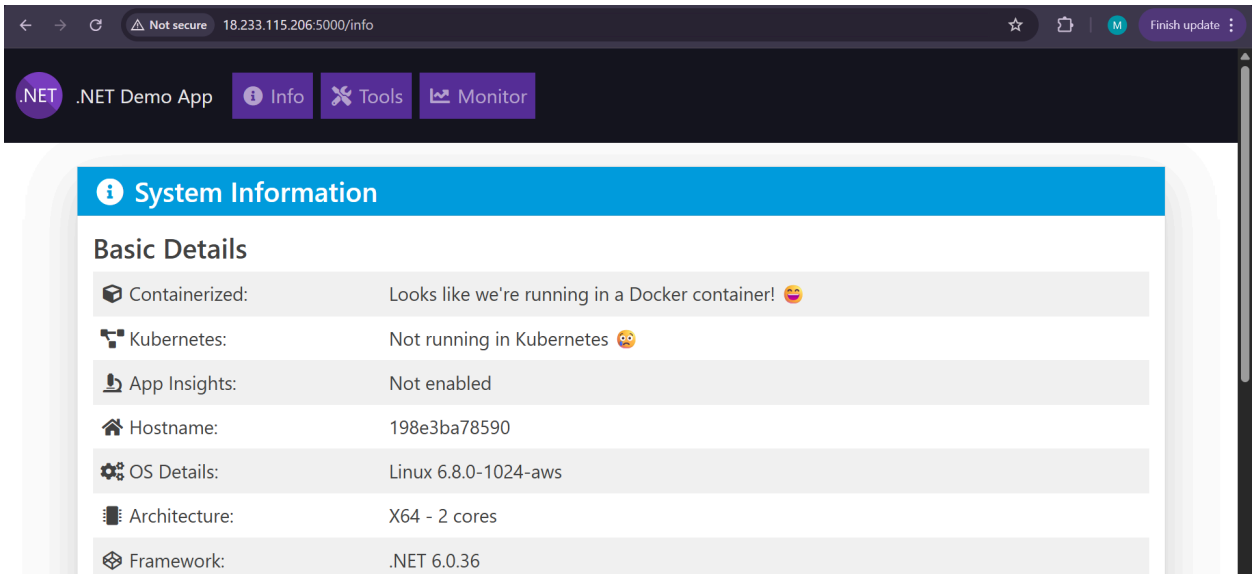


Step 10: Access the Application

For Docker deployment:

```
http://<Jenkins-EC2-Public-IP>:5000
http://18.233.115.206:5000/
```





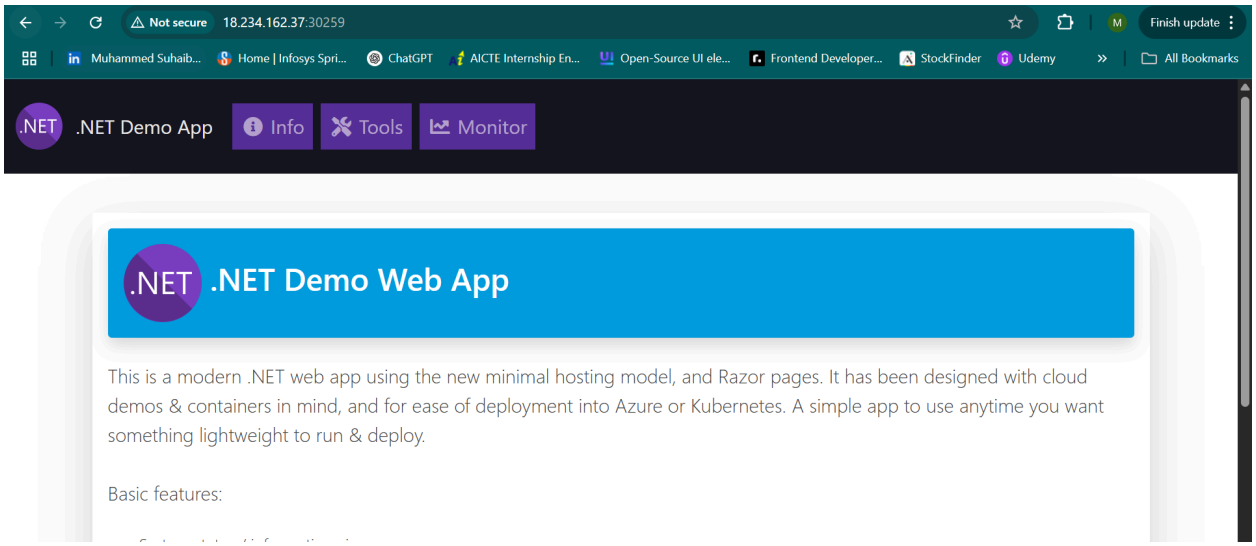
For Kubernetes deployment:

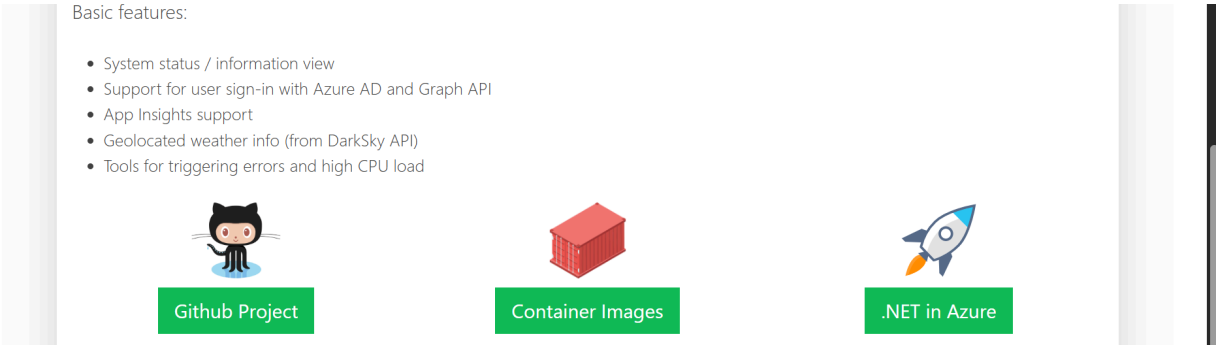
First get the service port using the following command in the master machine

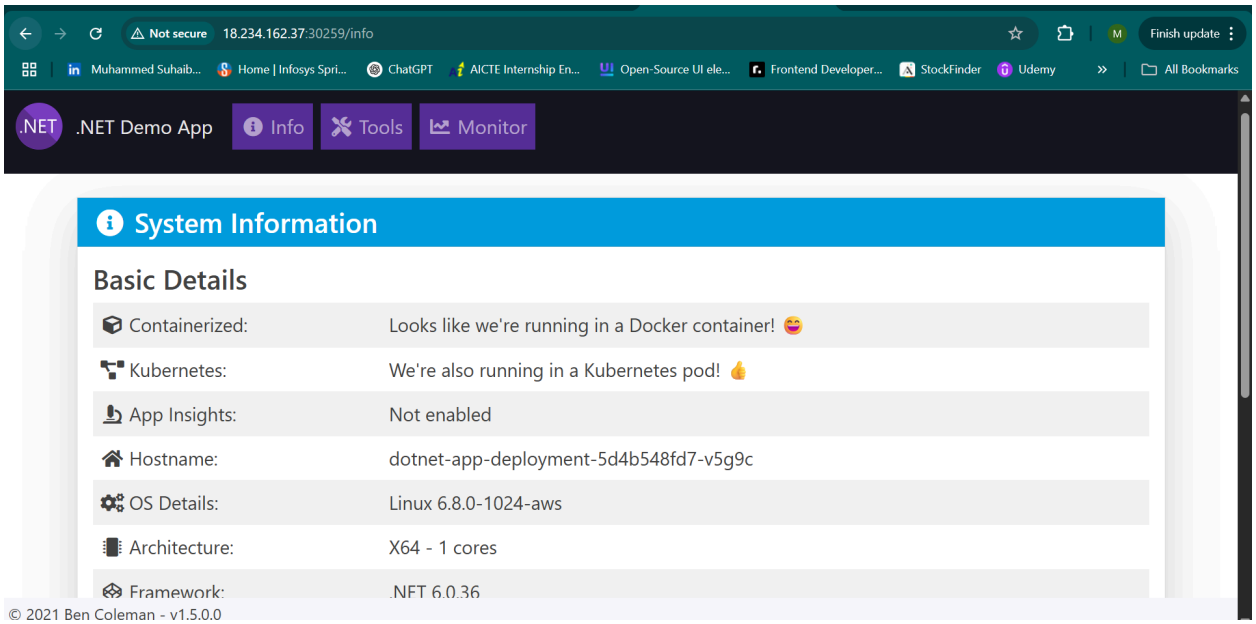
```
kubectl get svc
#copy service port
<worker-ip:svc port>
```

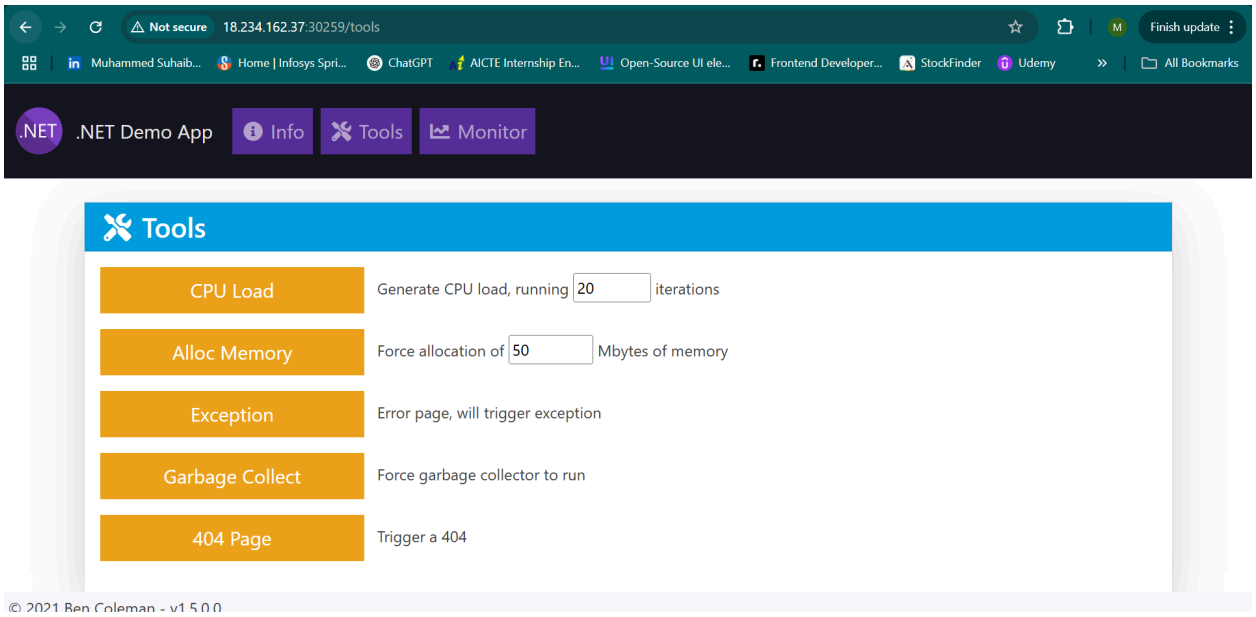
```
ubuntu@master:~$ kubectl get svc
NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
dotnet-app-service   NodePort    10.106.188.214   <none>        80:30259/TCP
10m
kubernetes           ClusterIP   10.96.0.1        <none>        443/TCP
72m
ubuntu@master:~$
```

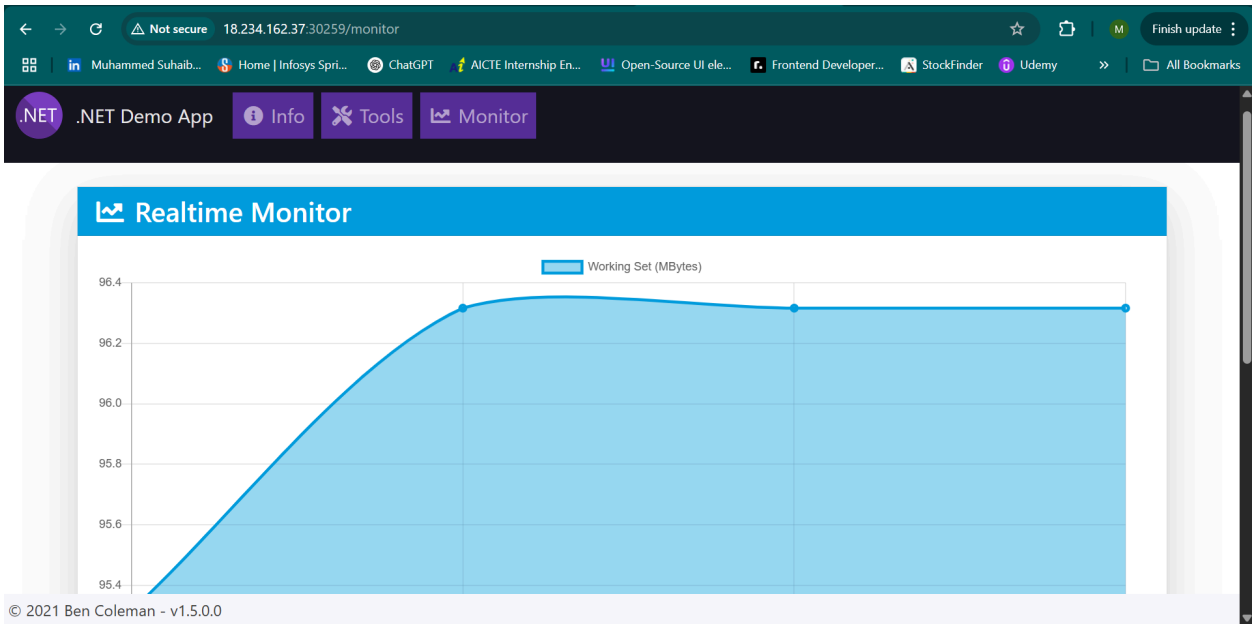
```
http://<Worker-Node-IP>:31000
http://18.234.162.37:30259/
```

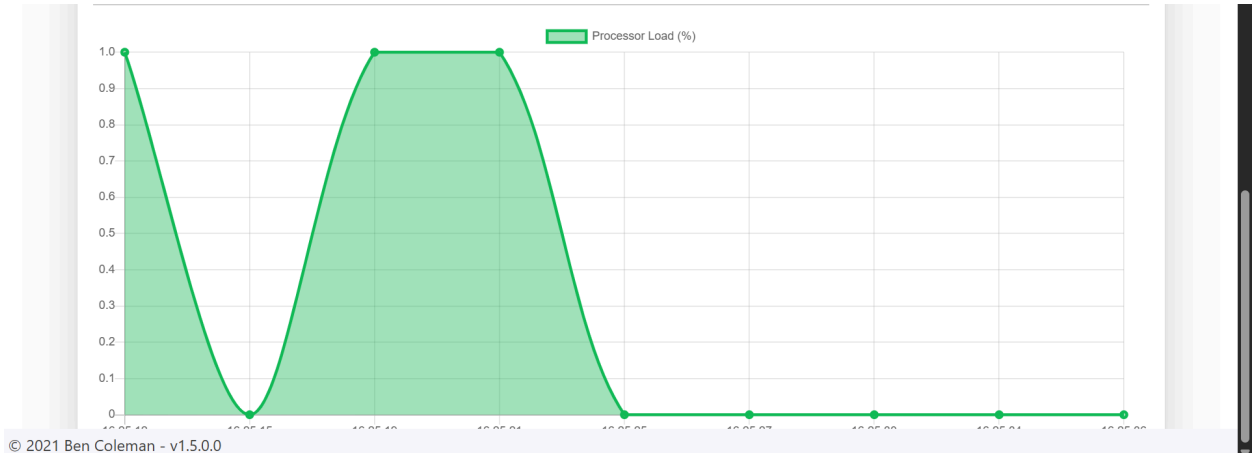





Not secure 18.234.162.37:30259/info
Finish update
Muhammed Suhaib...
Home | Infosys Spri...
ChatGPT
AICTE Internship En...
Open-Source UI ele...
Frontend Developer...
StockFinder
Udemy
All Bookmarks
.NET
.NET Demo App
Info
Tools
Monitor
System Information
Basic Details
Containerized:
Looks like we're running in a Docker container!
Kubernetes:
We're also running in a Kubernetes pod!
App Insights:
Not enabled
Hostname:
dotnet-app-deployment-5d4b548fd7-v5g9c
OS Details:
Linux 6.8.0-1024-aws
Architecture:
X64 - 1 cores
Framework:
.NET 6.0.36
© 2021 Ben Coleman - v1.5.0.0

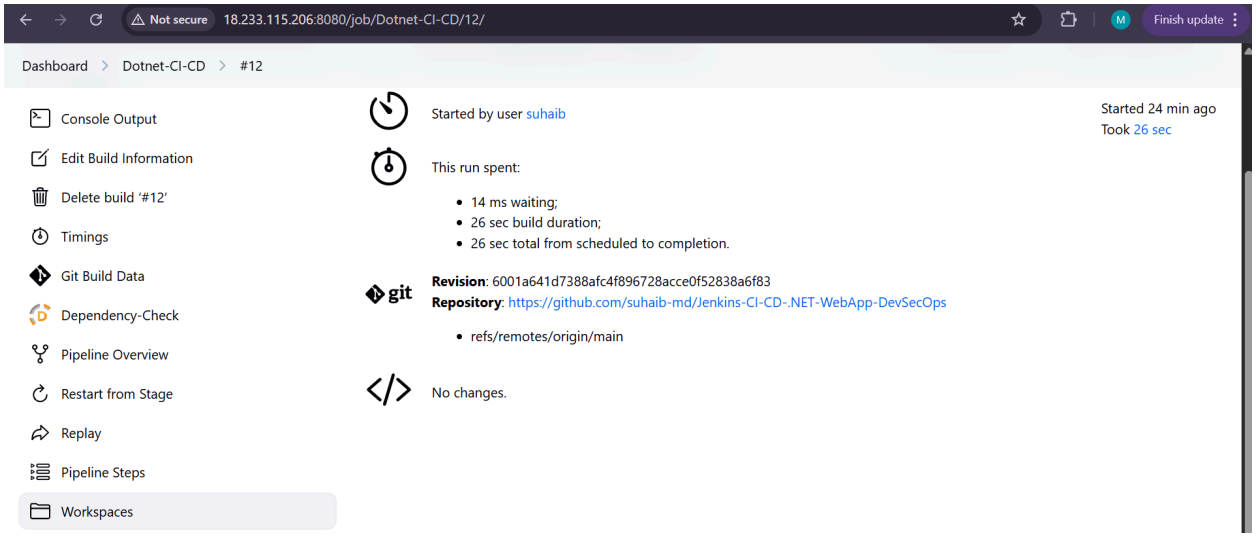

Not secure 18.234.162.37:30259/tools
Finish update
Muhammed Suhaib...
Home | Infosys Spri...
ChatGPT
AICTE Internship En...
Open-Source UI ele...
Frontend Developer...
StockFinder
Udemy
All Bookmarks
.NET
.NET Demo App
Info
Tools
Monitor
Tools
CPU Load
Generate CPU load, running 20 iterations
Alloc Memory
Force allocation of 50 Mbytes of memory
Exception
Error page, will trigger exception
Garbage Collect
Force garbage collector to run
404 Page
Trigger a 404
© 2021 Ben Coleman - v1.5.0.0


Not secure 18.234.162.37:30259/monitor
Finish update
Muhammed Suhaib...
Home | Infosys Spri...
ChatGPT
AICTE Internship En...
Open-Source UI ele...
Frontend Developer...
StockFinder
Udemy
All Bookmarks
.NET
.NET Demo App
Info
Tools
Monitor
Realtime Monitor
Working Set (MBytes)
96.4
96.2
96.0
95.8
95.6
95.4
© 2021 Ben Coleman - v1.5.0.0

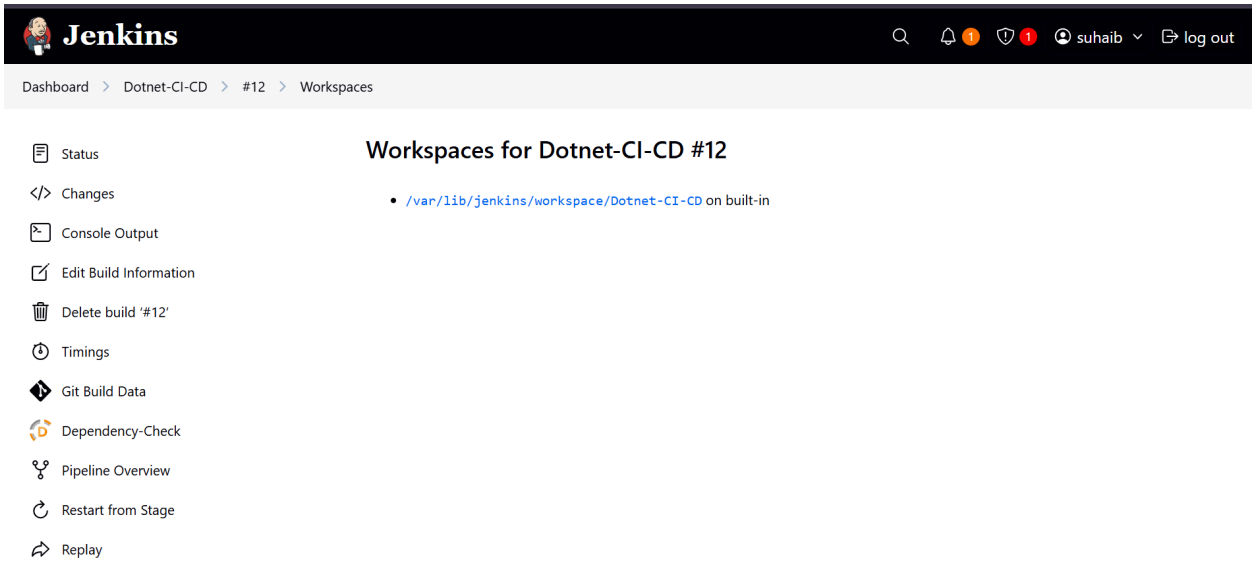

Processor Load (%)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
© 2021 Ben Coleman - v1.5.0.0

For Trivy Results:

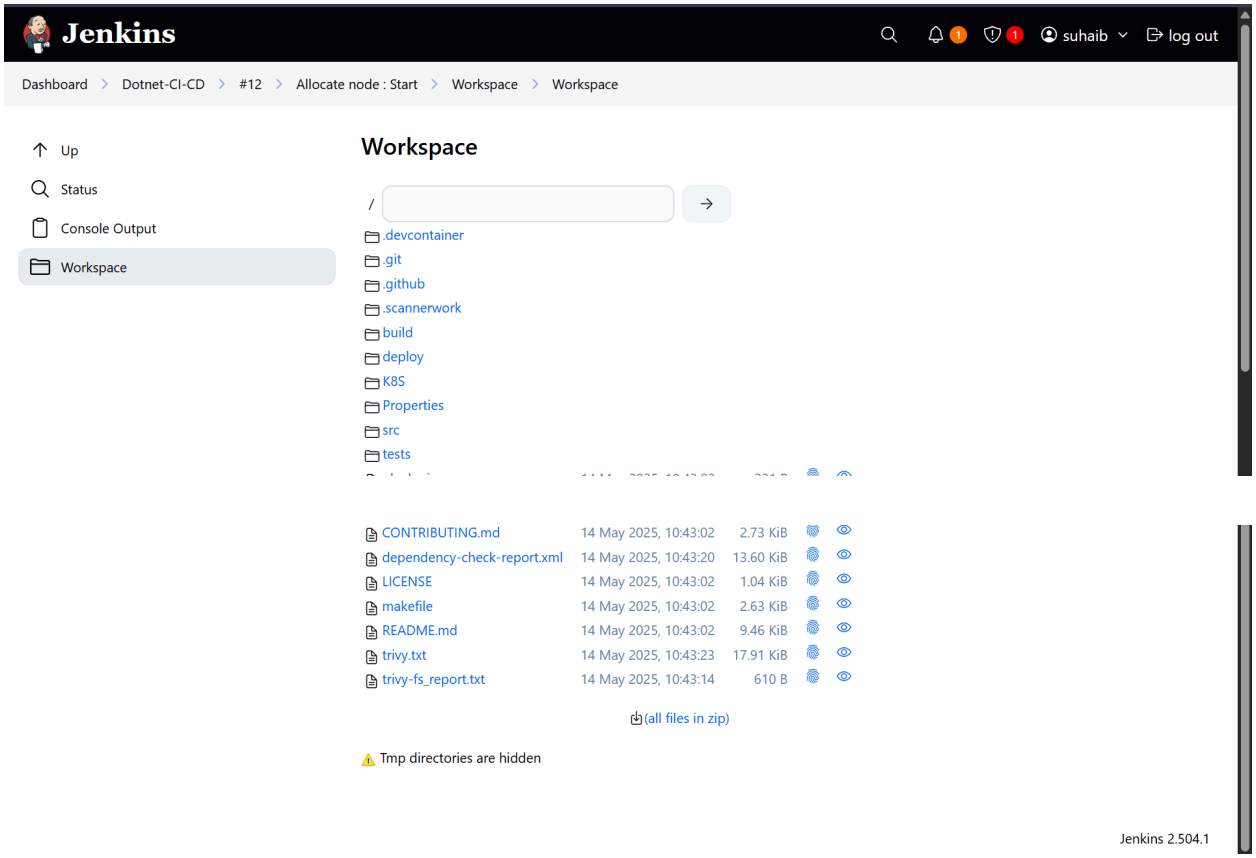
Go to the successful pipeline build page and select workspaces from the left hand side menu



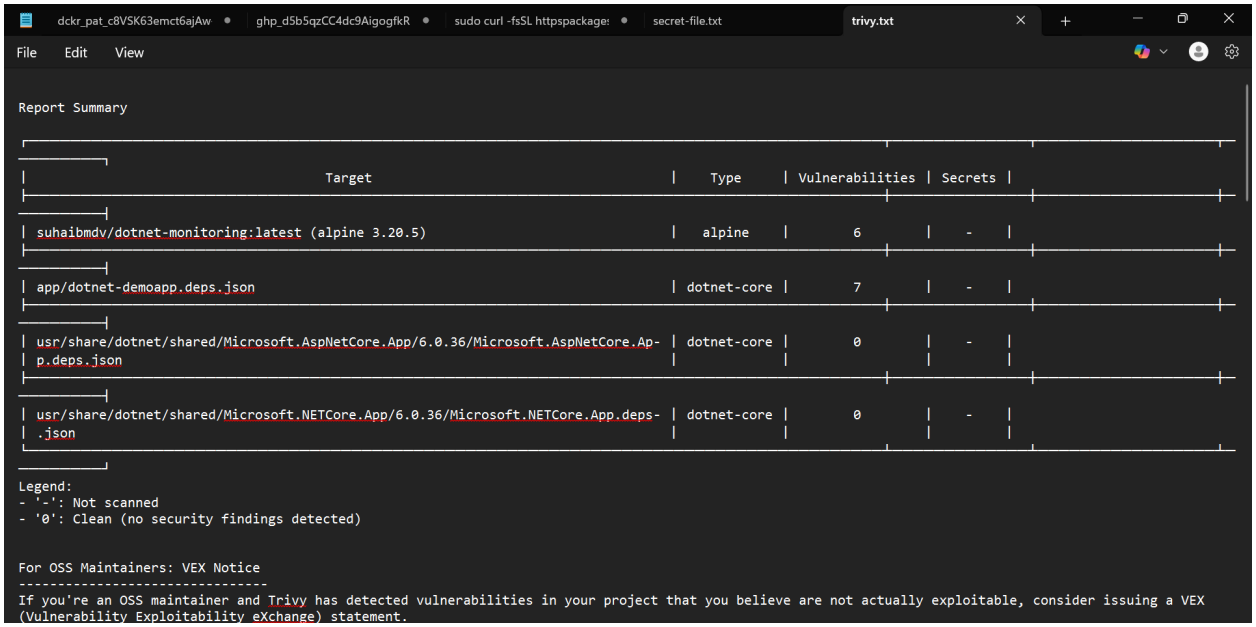
Click the link to view the workspace files



You can view all the files here. Trivy stores its scan results in this workspace.



To view the trivy scan results, download the folder as zip, extract and view as text file.

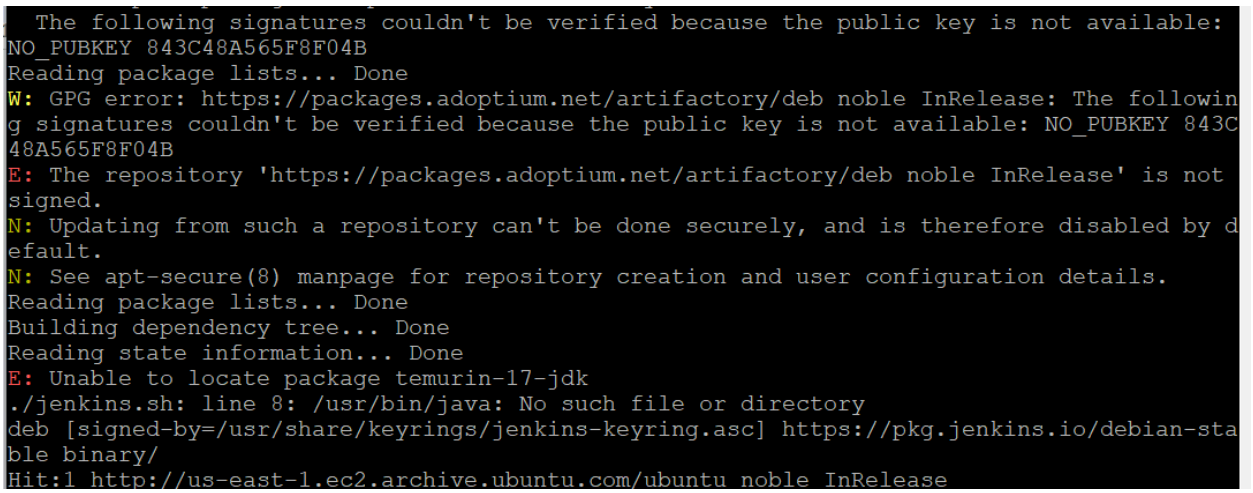


Step 11: Clean Up

When done with the project, terminate all AWS EC2 instances to avoid unnecessary charges.

Troubleshooting

Error in Public Key

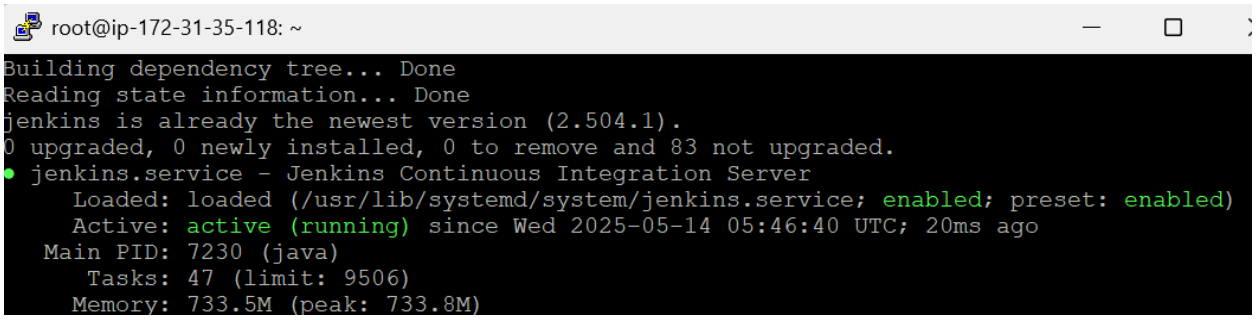


Changed to the current public key and GPG

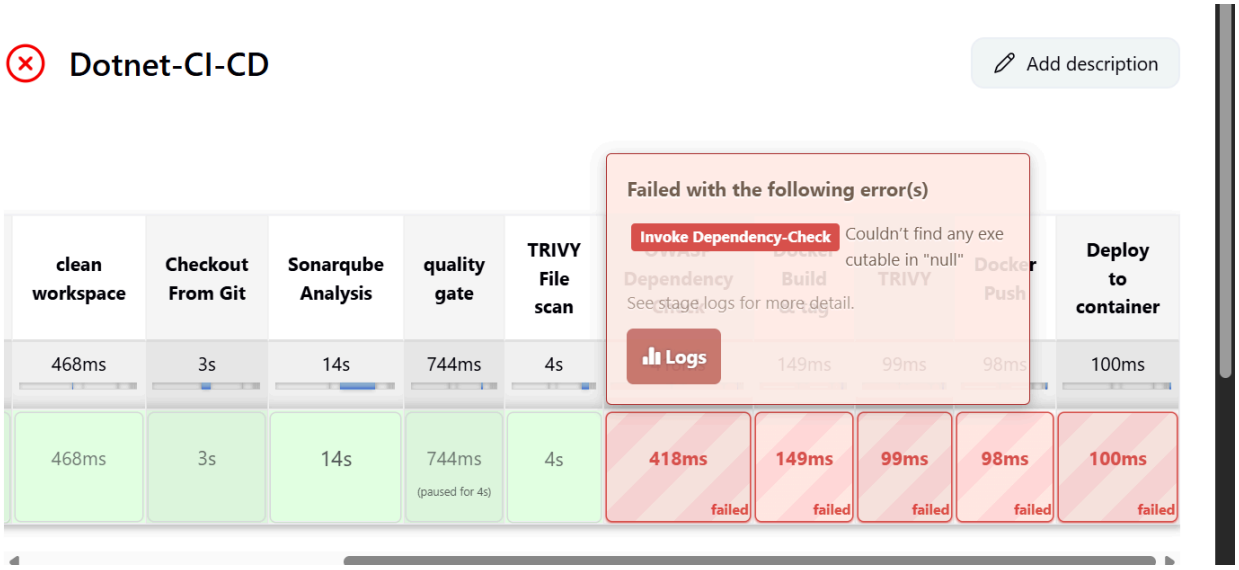
The versions which worked:

```
#!/bin/bash
sudo apt update -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
```

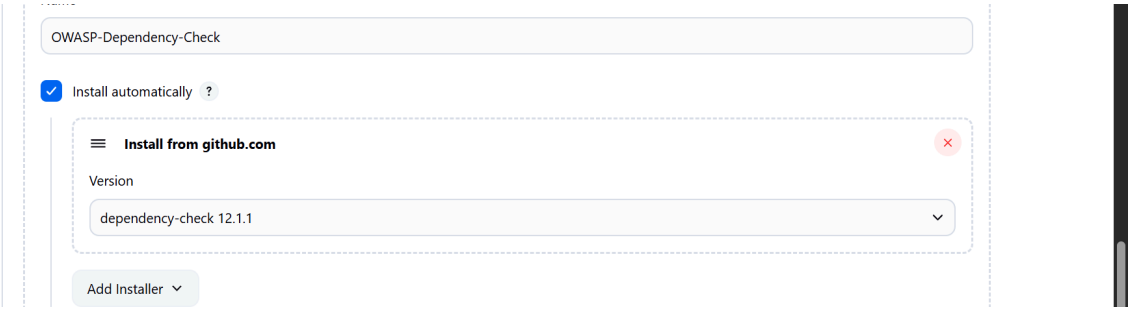
```
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo
tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
https://pkg.jenkins.io/debian-stable binary/ | sudo tee
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update -y
sudo apt-get install jenkins -y
sudo systemctl start jenkins
sudo systemctl status jenkins
#install docker
sudo apt-get update
sudo apt-get install docker.io -y
sudo usermod -aG docker ubuntu
newgrp docker
sudo chmod 777 /var/run/docker.sock
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
# 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
```



Error while running pipeline



It is due to improper setup of owasp dependency check in jenkins tools. Make sure you properly install it by selecting an installer.



Error while running pipeline (Docker)

```
Warning: failed to get default registry endpoint from daemon
(Got permission denied while trying to connect to the Docker
daemon socket at unix:///var/run/docker.sock: Get
http://%2Fvar%2Frun%2Fdocker.sock/v1.29/info: dial unix
/var/run/docker.sock: connect: permission denied). Using system
default: https://index.docker.io/v1/
Got permission denied while trying to connect to the Docker
daemon socket at unix:///var/run/docker.sock: Post
http://%2Fvar%2Frun%2Fdocker.sock/v1.29/auth: dial unix
```

```
/var/run/docker.sock: connect: permission denied
[Pipeline] // withDockerRegistry
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (TRIVY)
Stage "TRIVY" skipped due to earlier failure(s)
[Pipeline] getContext
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Docker Push)
Stage "Docker Push" skipped due to earlier failure(s)
[Pipeline] getContext
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy to container)
Stage "Deploy to container" skipped due to earlier failure(s)
[Pipeline] getContext
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: docker login failed
Finished: FAILURE
```

The error is:

```
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.29/auth: dial unix /var/run/docker.sock: connect: permission denied
```

This is a common permission issue with Docker. The Jenkins user doesn't have the necessary permissions to access the Docker daemon socket. here's how to fix this:

Solution to the Docker permission denied error:

Add the Jenkins user to the Docker group:

```
sudo usermod -aG docker jenkins
```

Restart the Jenkins service to apply the changes:

```
sudo systemctl restart jenkins
```

Alternatively, you can change the permissions of the Docker socket (less secure approach):

```
sudo chmod 666 /var/run/docker.sock
```

```
ubuntu@ip-172-31-35-118:~$ sudo chmod 666 /var/run/docker.sock
ubuntu@ip-172-31-35-118:~$
```

Error while installing kubernetes

```
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Err:6 https://packages.cloud.google.com/apt kubernetes-xenial Release
  404  Not Found [IP: 142.250.31.139 443]
Reading package lists... Done
E: The repository 'https://apt.kubernetes.io kubernetes-xenial Release' does n
 have a Release file.
N: Updating from such a repository can't be done securely, and is therefore di
bled by default.
N: See apt-secure(8) manpage for repository creation and user configuration de
ils.
```

This error is because the correct keys are not used for our version of Ubuntu. To solve this the correct version is used.

```
# Install Kubernetes packages
sudo apt install -y apt-transport-https ca-certificates curl gpg

# Add Kubernetes GPG key
sudo mkdir -p /etc/apt/keyrings
curl -fsSL
https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo
gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

# Add Kubernetes repo
echo "deb
[signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] \
https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /" | \
sudo tee /etc/apt/sources.list.d/kubernetes.list > /dev/null

# Update and install Kubernetes tools
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

GitHub Repository

The code for this project is available at:

https://github.com/suhaib-md/Jenkins-CI-CD-.NET-WebApp-DevSecOps

Conclusion

This DevSecOps project demonstrates how to set up a complete CI/CD pipeline for a .NET application using Jenkins, Docker, and Kubernetes. By following these steps, you've integrated security scanning with SonarQube, Trivy, and OWASP Dependency Check into the pipeline, ensuring that the application is not only built and deployed automatically but also checked for potential security vulnerabilities at various stages of the development lifecycle.

The pipeline shows the practical application of DevSecOps principles by:

1. Performing code quality and security analysis with SonarQube
2. Checking dependencies for vulnerabilities with OWASP Dependency Check
3. Scanning the file system and Docker images for vulnerabilities with Trivy
4. Automating the build, test, and deployment processes with Jenkins
5. Containerizing the application with Docker
6. Orchestrating containers with Kubernetes

This comprehensive approach helps ensure that security is an integral part of the development process rather than an afterthought.