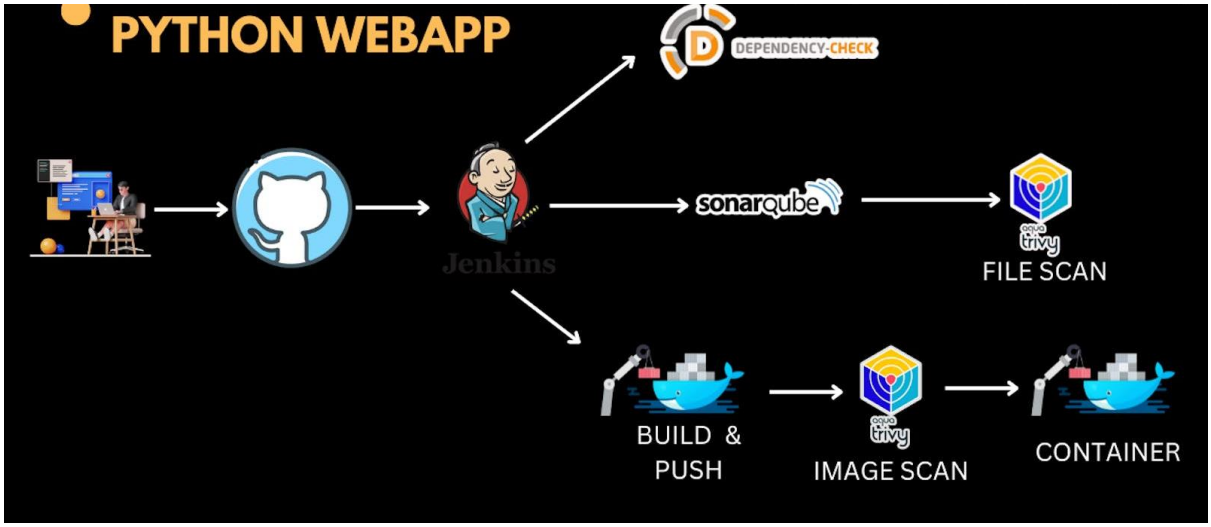
CI-CD DevSecOps project with Jenkins | Python webapp



Step 1 — Launch an AWS T2 Large Instance.

Step 2 — Install Jenkins, Docker and Trivy

2A — To Install Jenkins

2B — Install Docker

2C — Install Trivy

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

3A — Install Plugin

3B — Configure Java and Maven in Global Tool Configuration

3C — Create a Job

Step 4 — Install OWASP Dependency Check Plugins

Step 5 — Configure Sonar Server in Manage Jenkins

Step 6 — we have to install make package

Step 7 — Docker Image Build and Push

Step 8 — Deploy the image using Docker

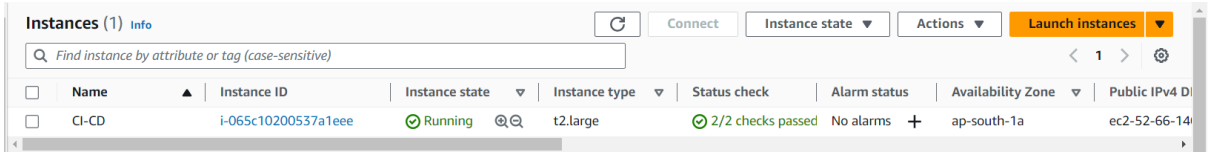
Step 9 — Access the Real World Application

Step 10 — Terminate the AWS EC2 Instance

https://github.com/Milky19/Python-System-Monitoring.git

STEP1:Launch an Ubuntu(22.04) T2 Large Instance

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 and open all ports (not best case to open all ports but just for learning purposes it's okay).



Step 2 — Install Jenkins, Docker and Trivy

2A — To Install Jenkins

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

vi jenkins.sh

#!/bin/bash

sudo apt update -y

#sudo apt upgrade -y

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

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

sudo apt update -y

sudo apt install temurin-17-jdk -y

/usr/bin/java --version

**curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee **

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

**echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] **

**https://pkg.jenkins.io/debian-stable binary/ | sudo tee **

/etc/apt/sources.list.d/jenkins.list > /dev/null

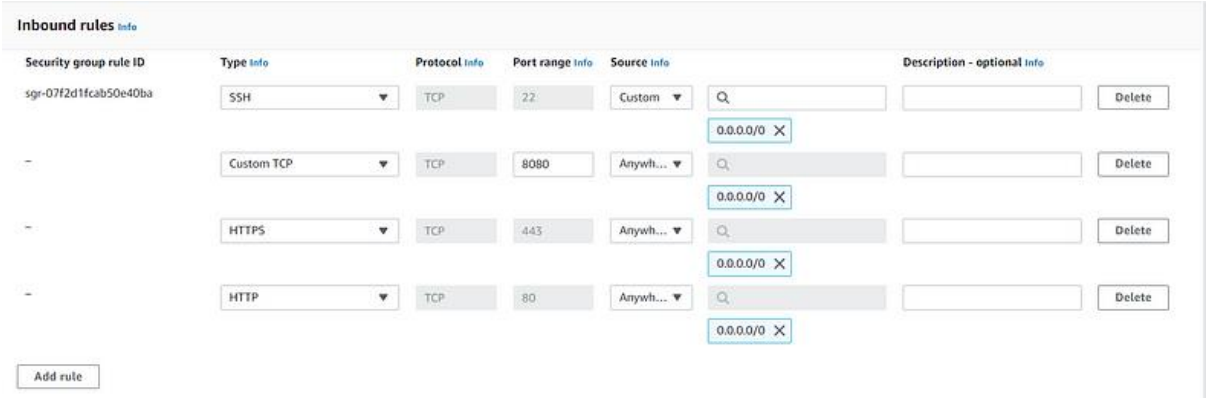
sudo apt-get update -y

sudo apt-get install jenkins -y

sudo systemctl start jenkins

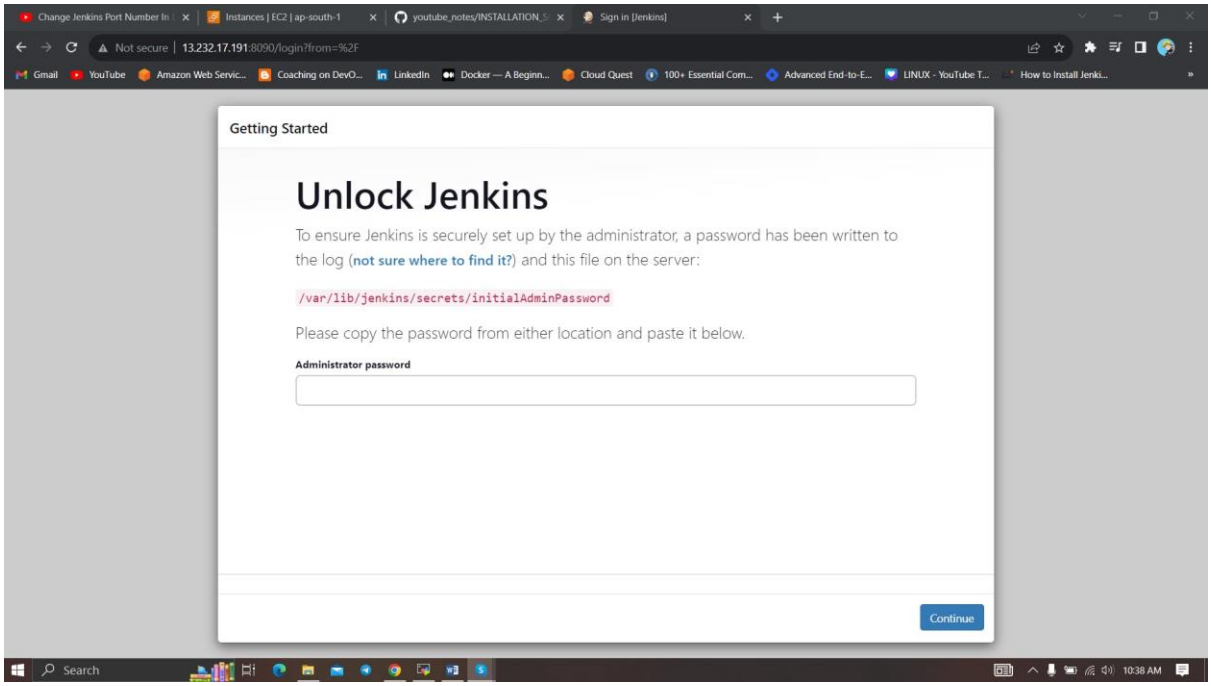
sudo systemctl status jenkins

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.

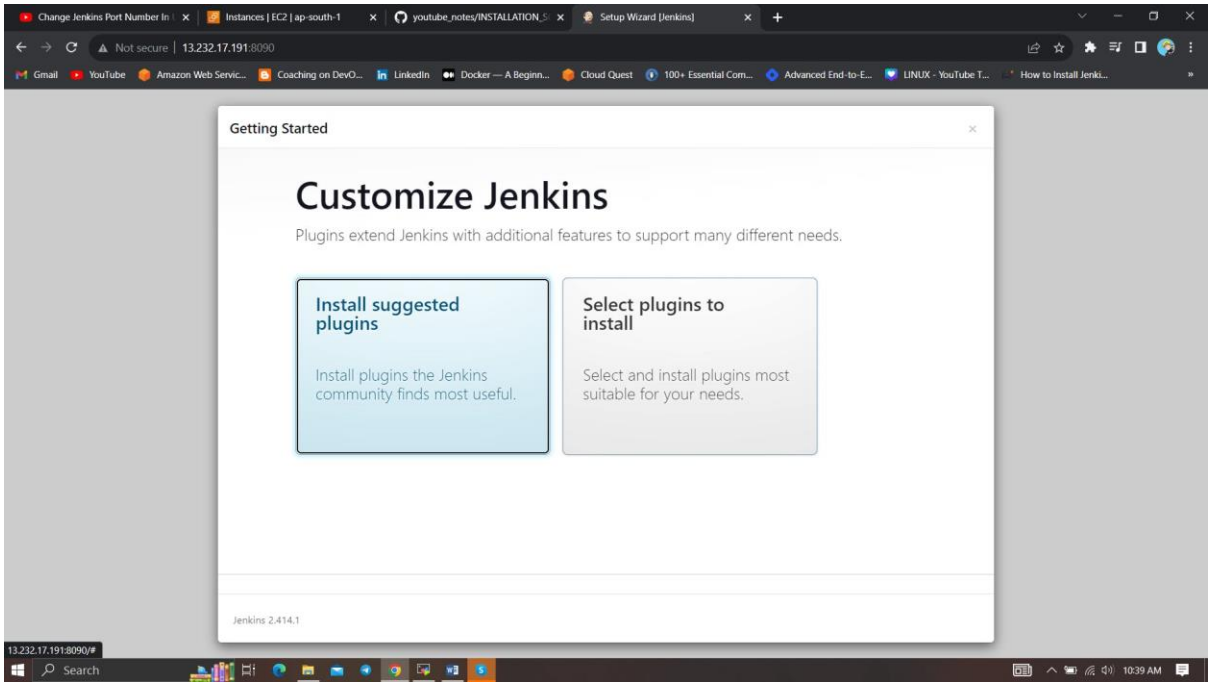


<EC2 Public IP Address:8080>

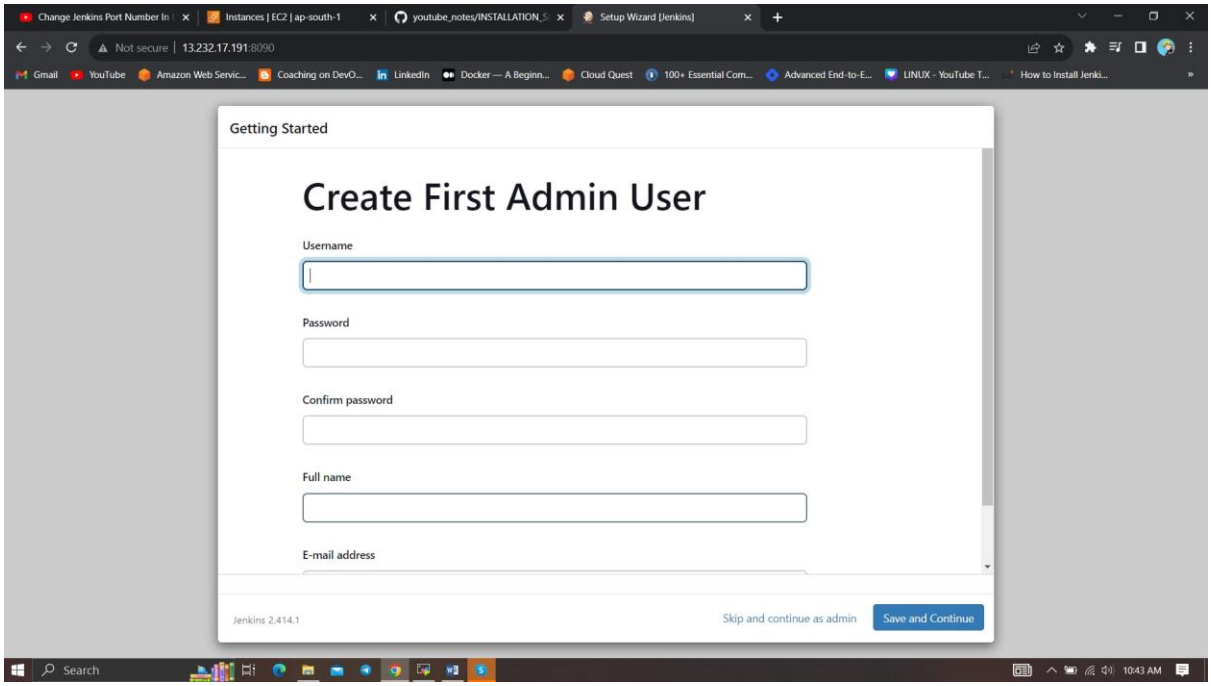
sudo cat /var/lib/jenkins/secrets/initialAdminPassword



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

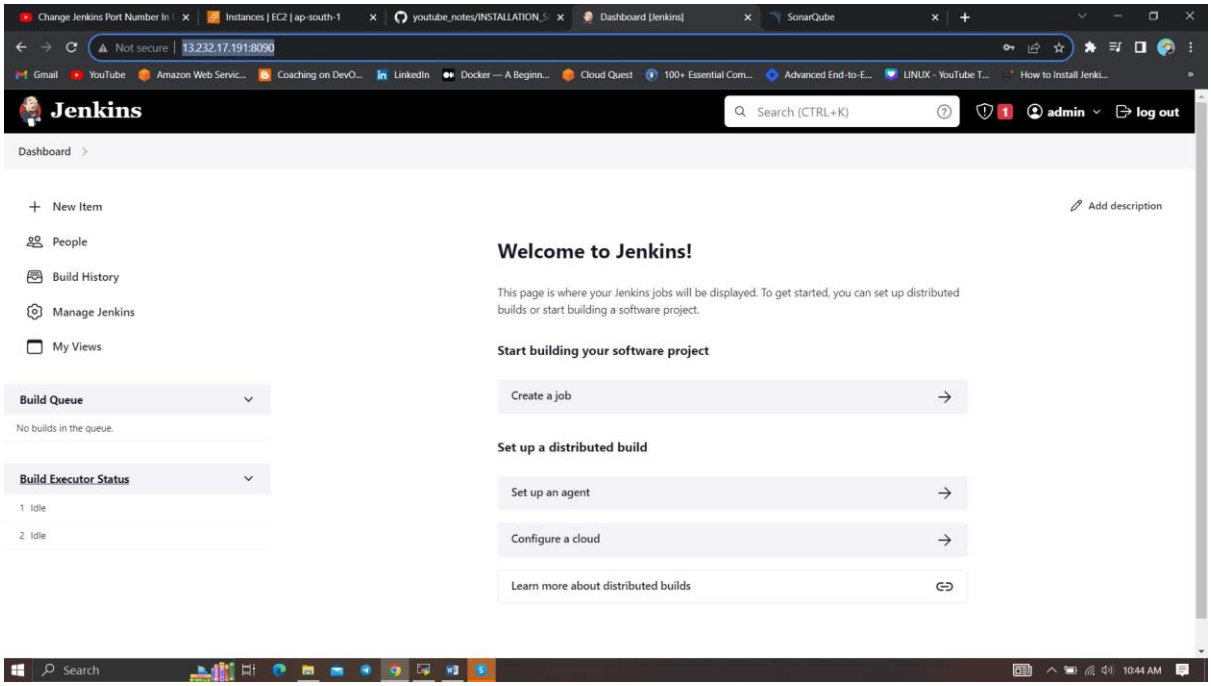


Jenkins will now get installed and install all the libraries.



Create a user click on save and continue.

Jenkins Getting Started Screen.



2B — Install Docker

sudo apt-get update

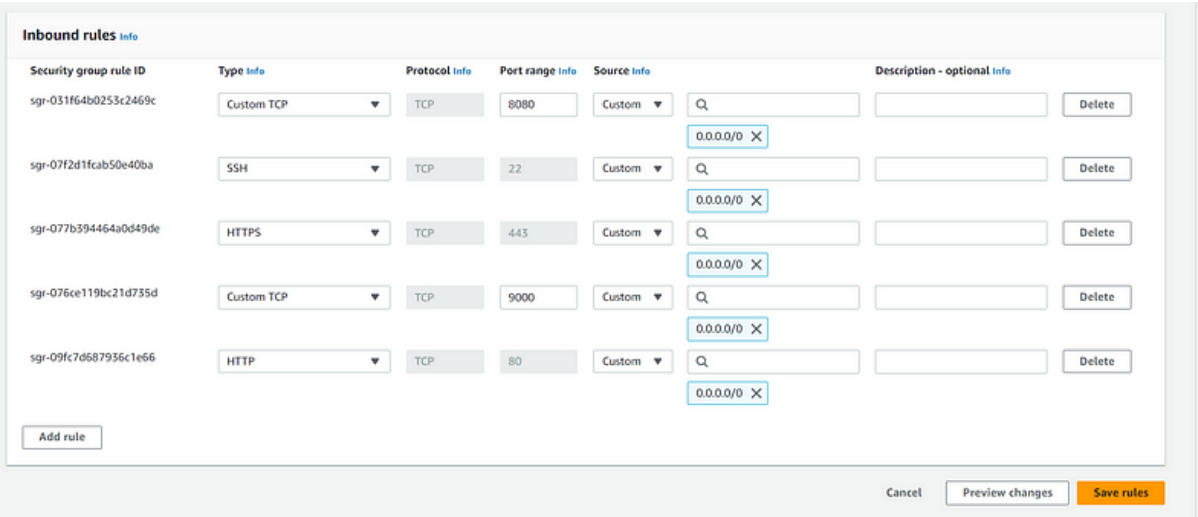
sudo apt-get install docker.io -y

sudo usermod -aG docker $USER #my case is ubuntu

newgrp docker

sudo chmod 777 /var/run/docker.sock

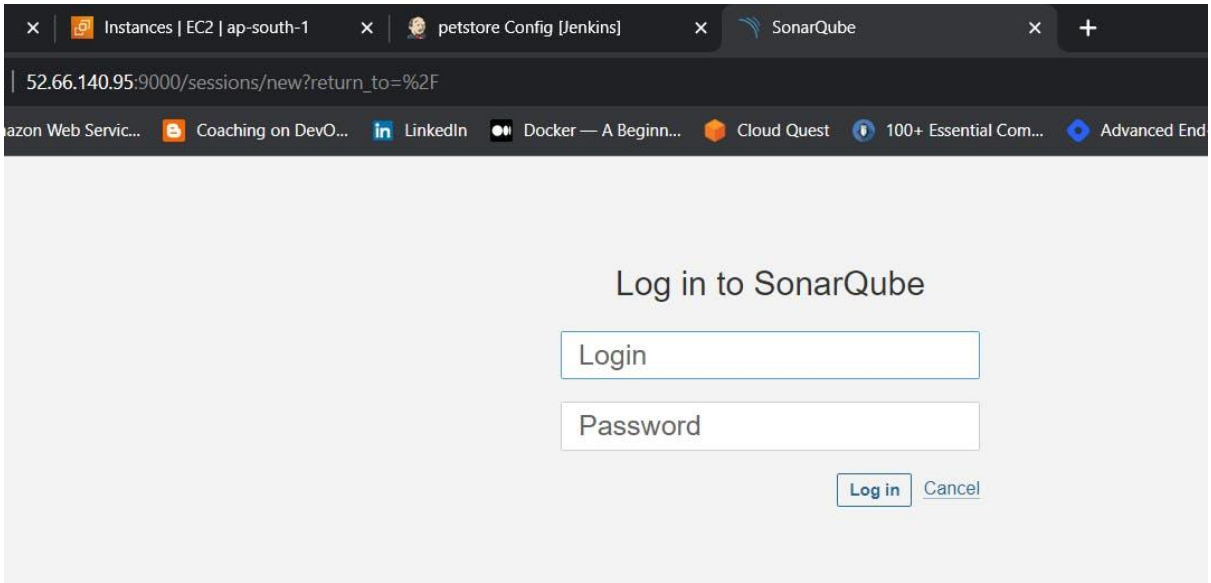
After the docker installation, we create a sonarqube container (Remember to add 9000 ports in the security group).



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:~$
```

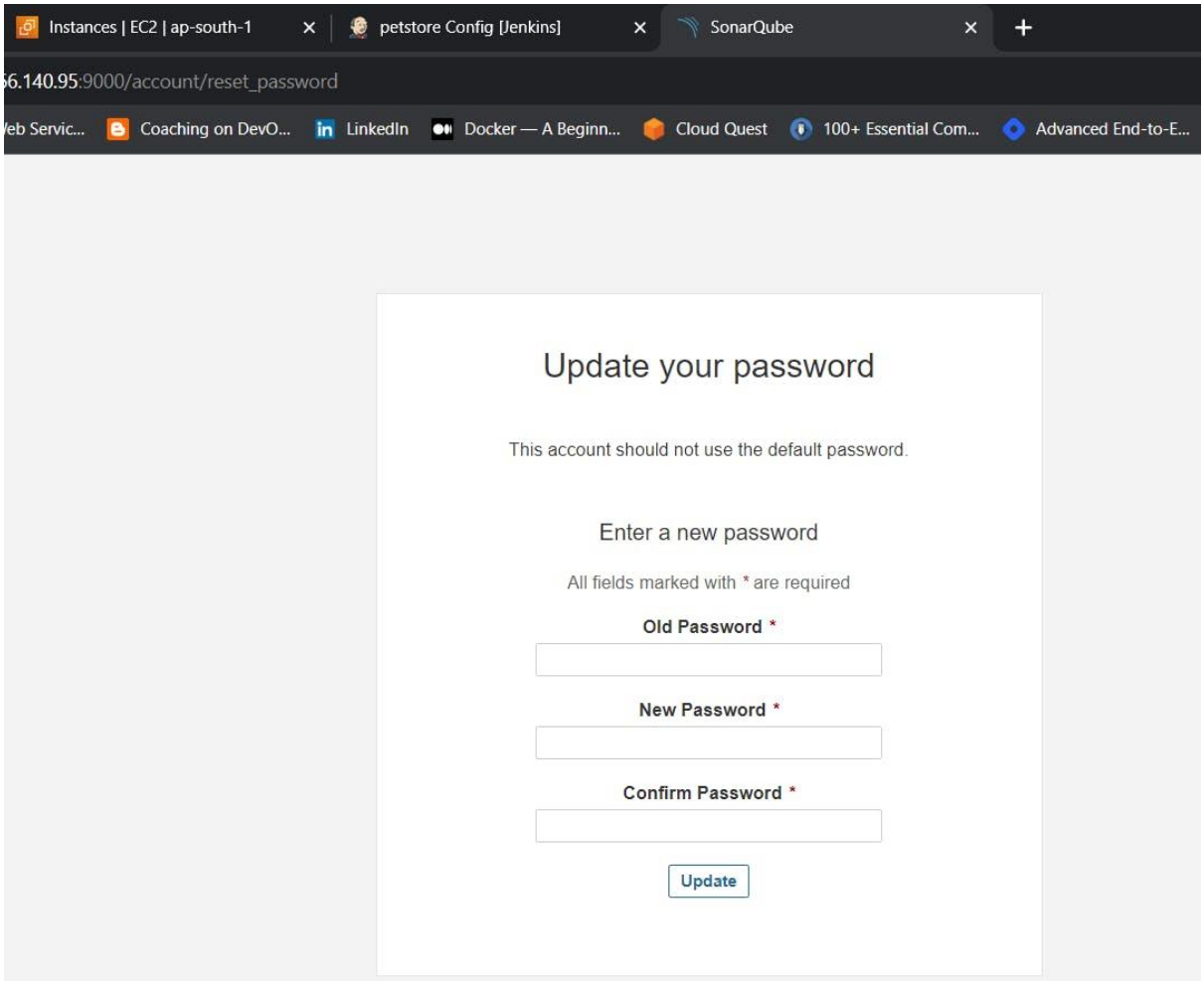
Now our sonarqube is up and running



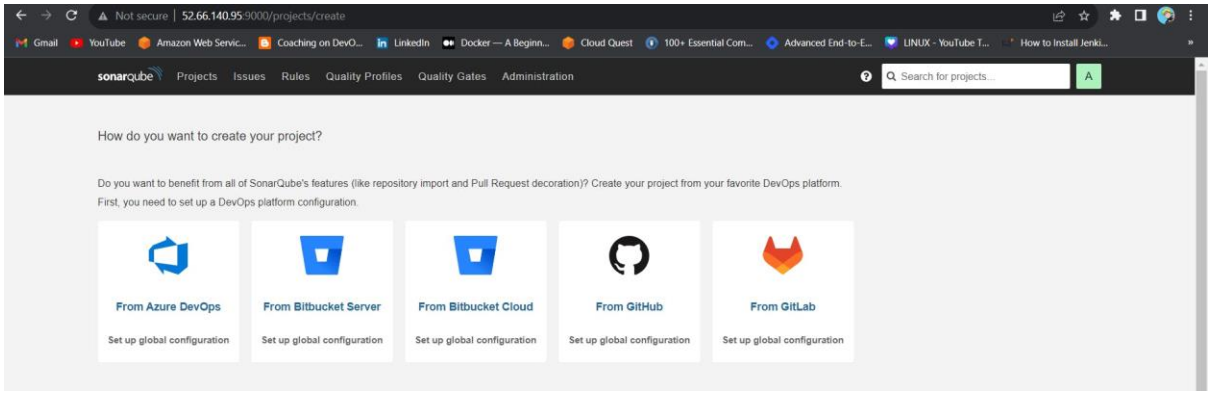
Enter username and password, click on login and change password

username admin

password admin



Update New password, This is Sonar Dashboard.



2C — Install Trivy

vi trivy.sh

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

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

echo "deb [signed-by=/usr/share/keyrings/trivy.gpg] https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main" | sudo tee -a /etc/apt/sources.list.d/trivy.list

sudo apt-get update

sudo apt-get install trivy –y

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

Step 3 — Install Plugins like JDK, Sonarqube Scanner, NodeJs, OWASP Dependency Check

3A — Install Plugin

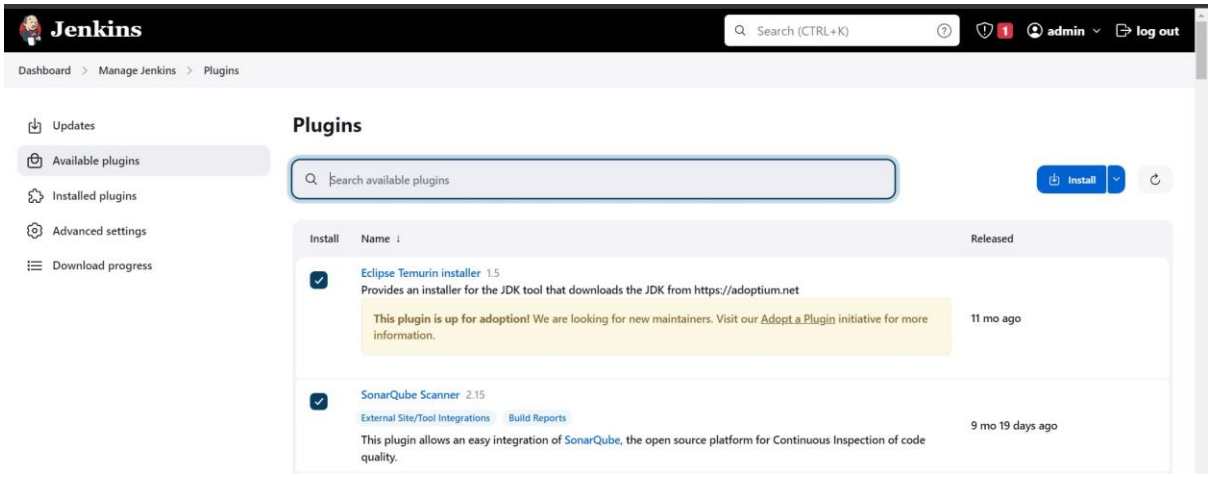
Goto Manage Jenkins →Plugins → Available Plugins →

Install below plugins

1 → Eclipse Temurin Installer (Install without restart)

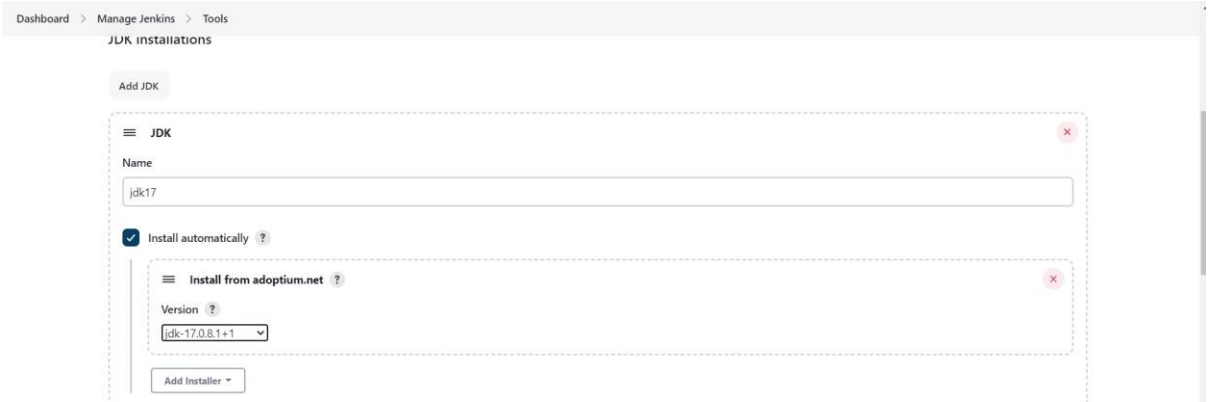
2 → SonarQube Scanner (Install without restart)

3 → NodeJs Plugin (Install Without restart)



3B — Configure Java and Maven in Global Tool Configuration

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

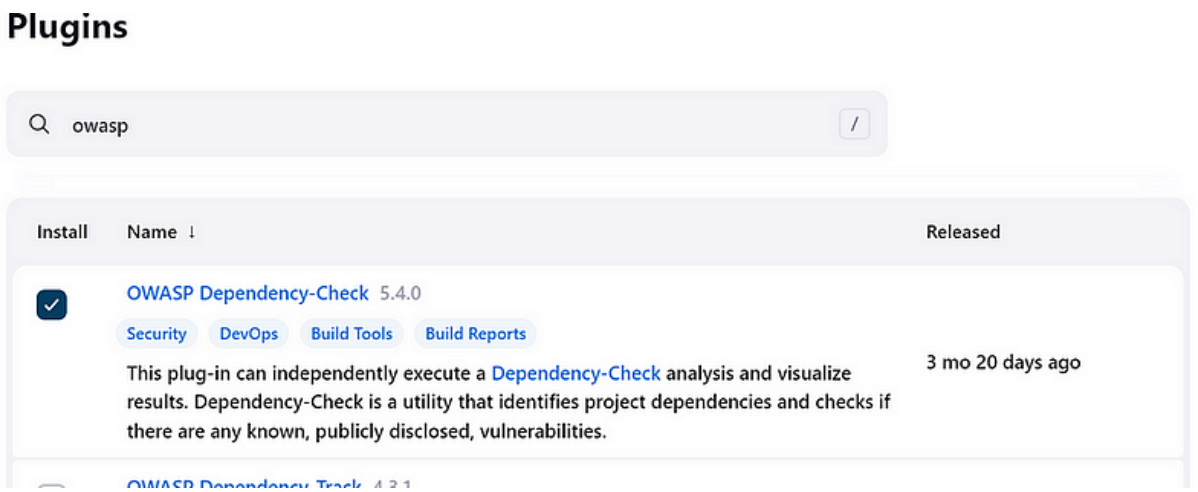


3C — Create a Job

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

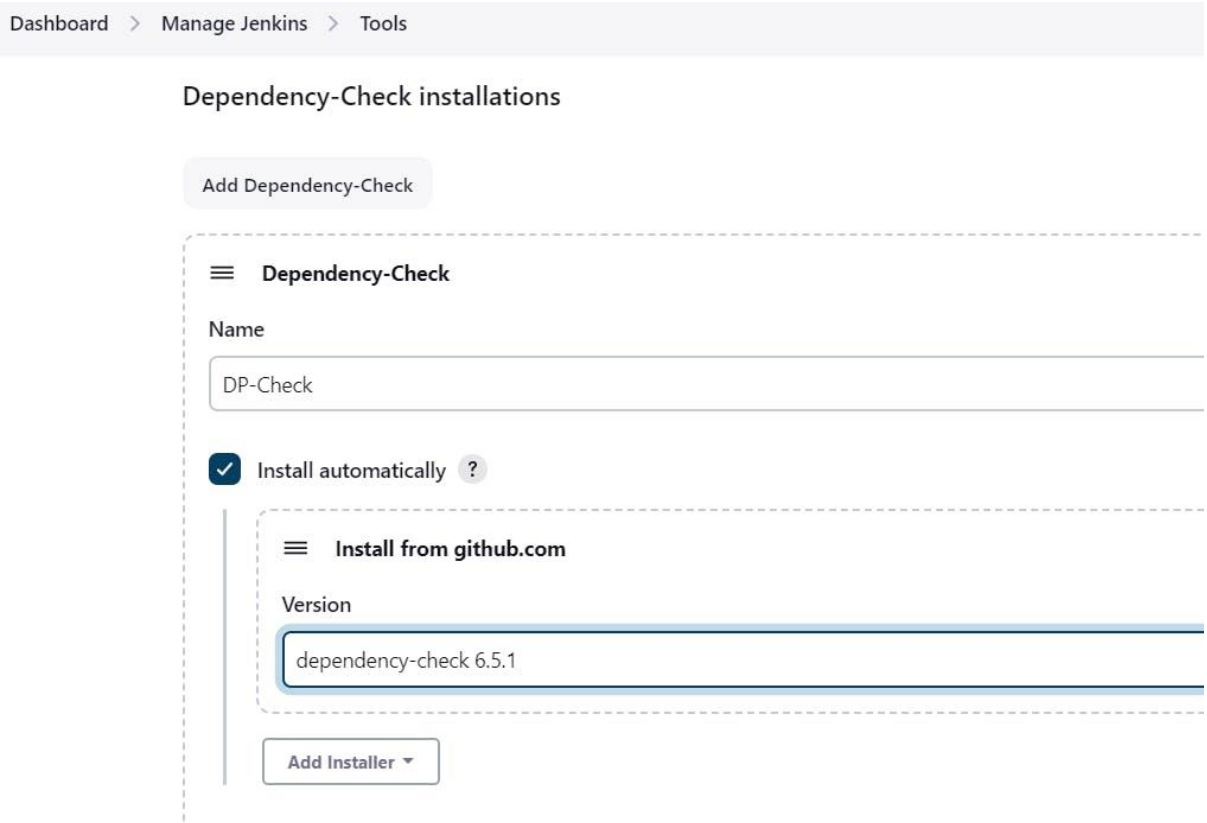
Step 4 — Install OWASP Dependency Check Plugins

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

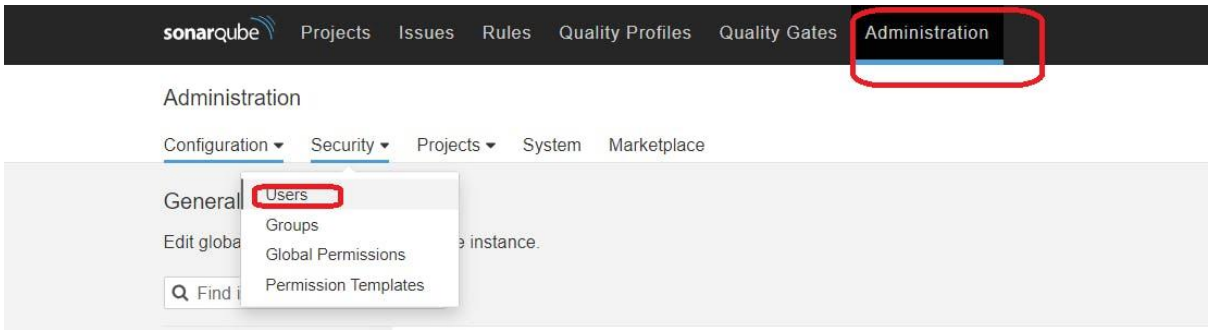
Goto Dashboard → Manage Jenkins → Tools →



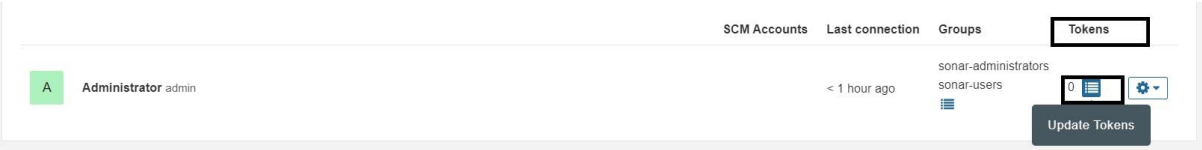
Click on Apply and Save here.

Step 5 — Configure Sonar Server in Manage Jenkins

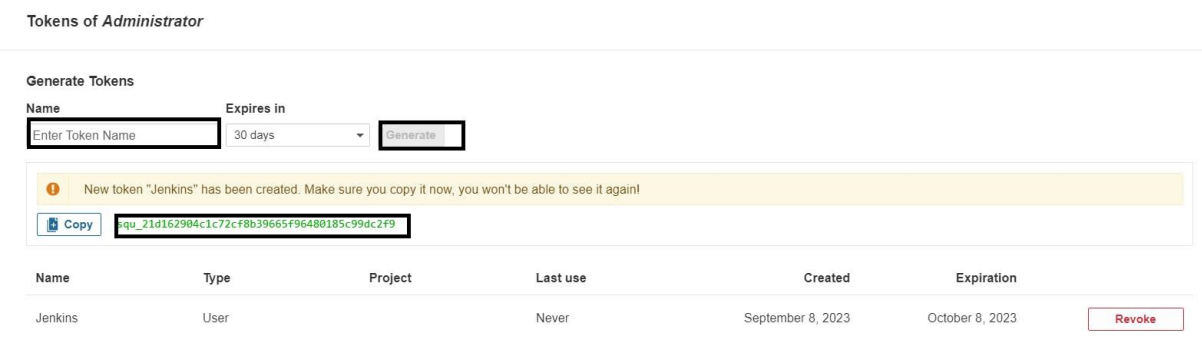
Grab the Public IP Address of your EC2 Instance, Sonarqube works on Port 9000, sp <Public IP>:9000. Goto your Sonarqube Server. Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → and click on Generate Token



Click on Update Token

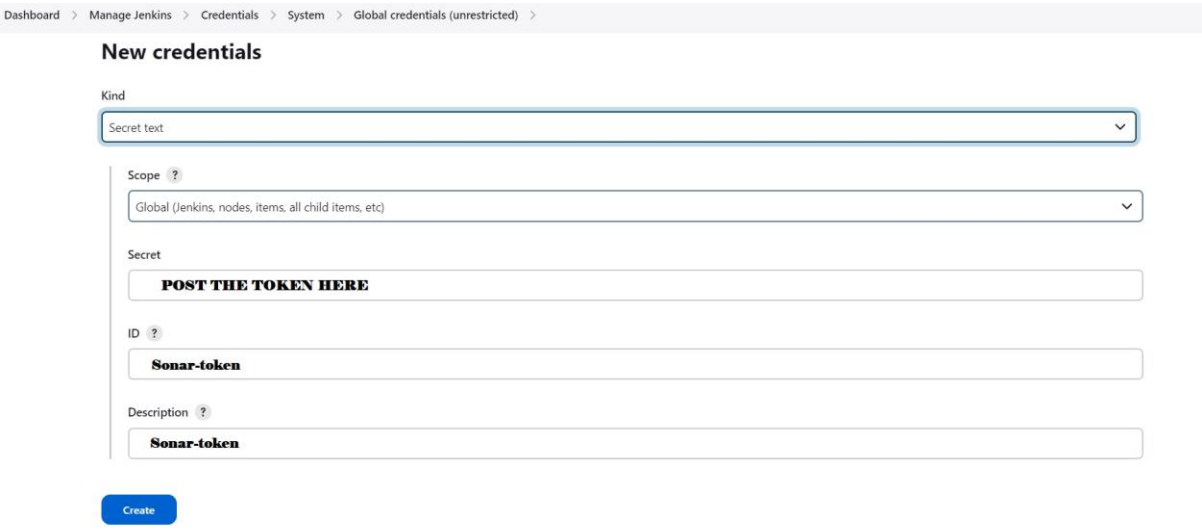


Create a token with a name and generate



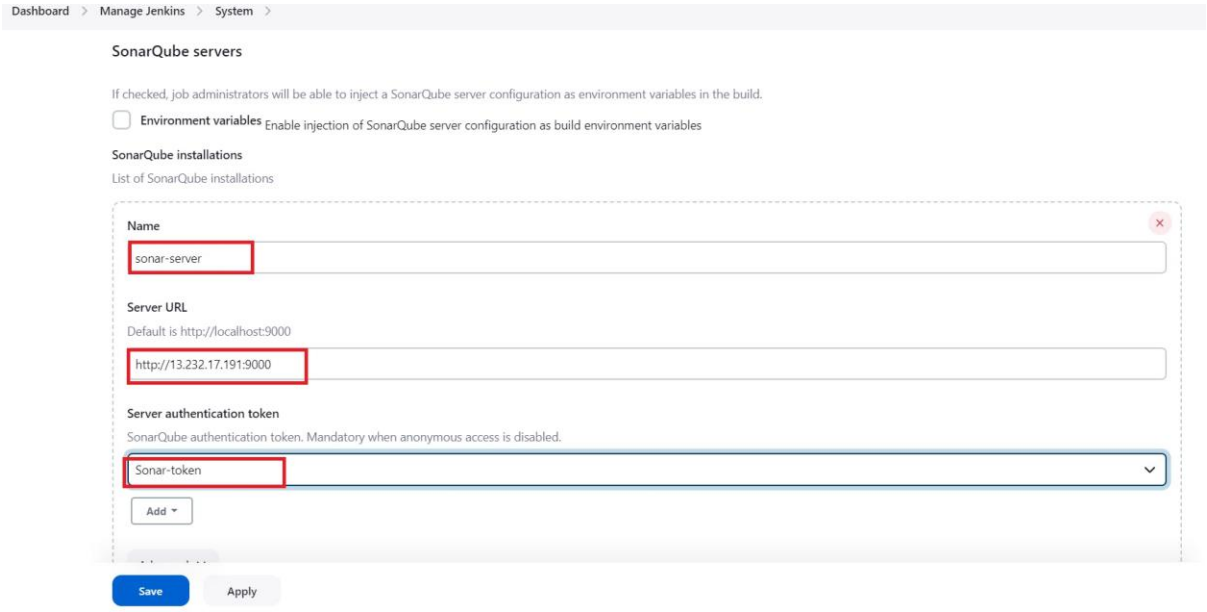
Copy this Token

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

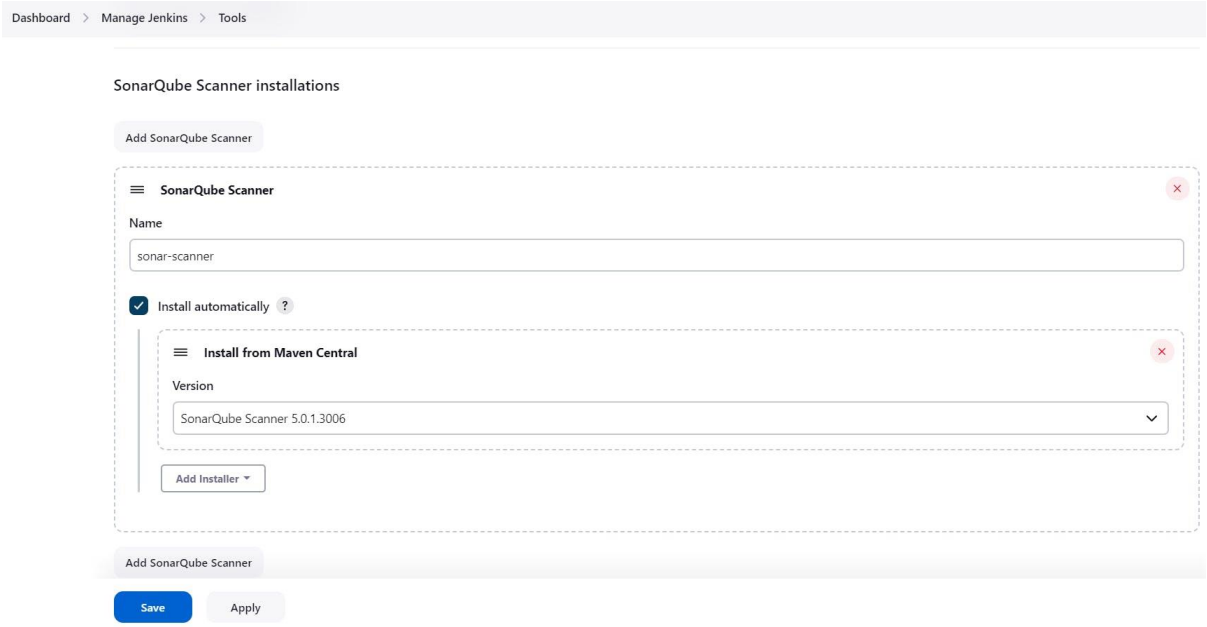


Click on Apply and Save

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

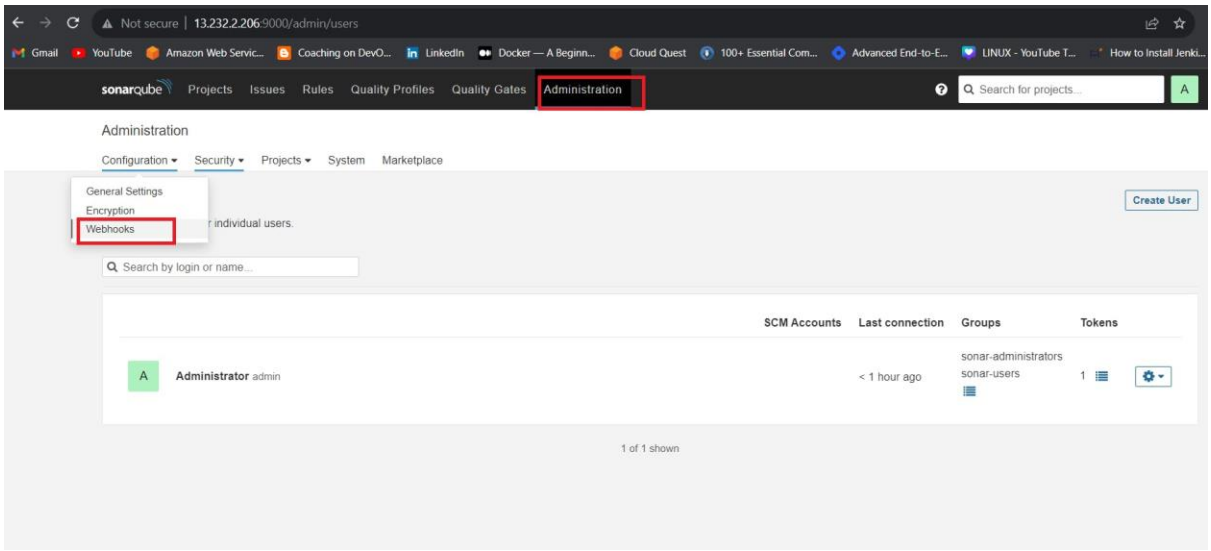
Global Tool Configuration is used to configure different tools that we install using Plugins

We will install a sonar scanner in the tools.

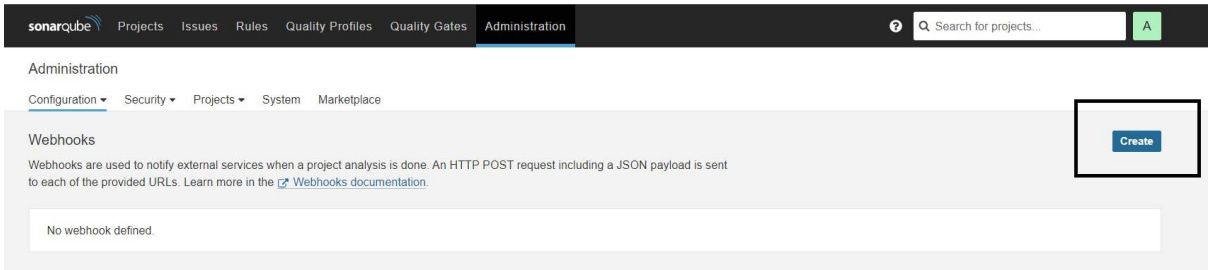


In the Sonarqube Dashboard add a quality gate also

Administration--> Configuration-->Webhooks



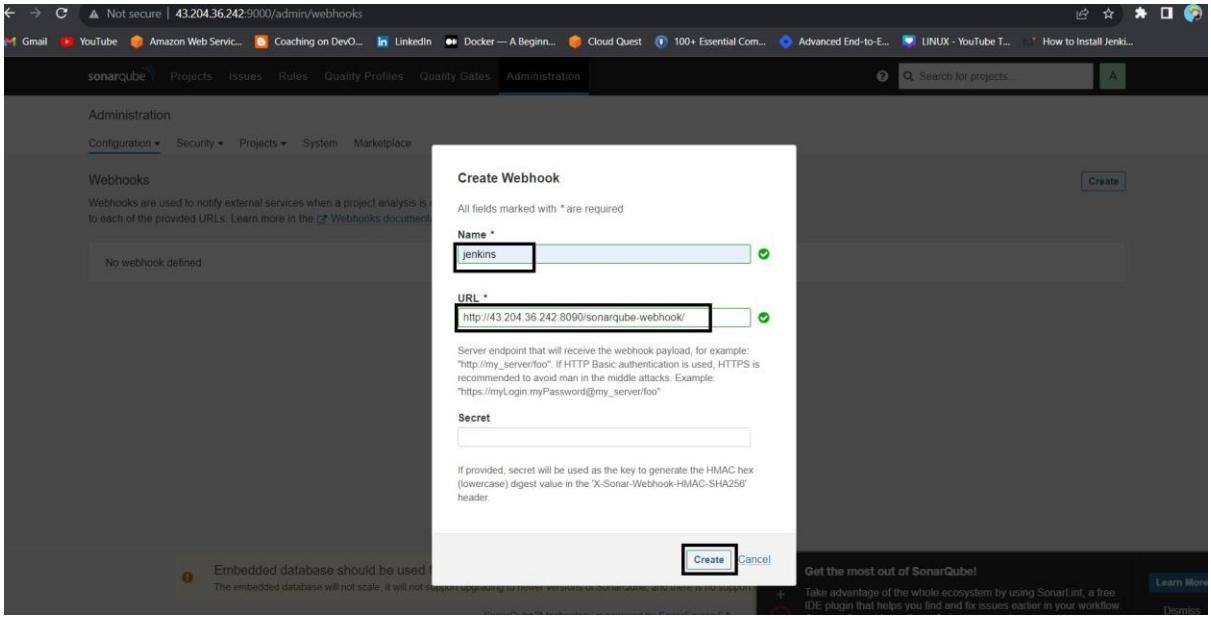
Click on Create



Add details

#in url section of quality gate

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



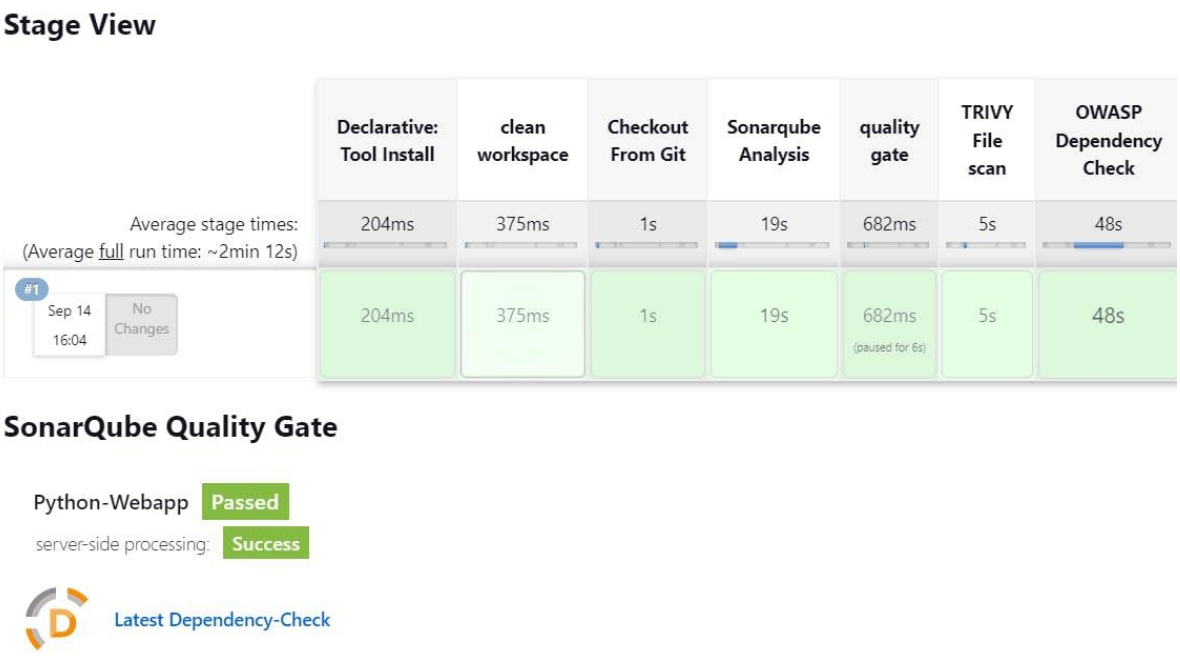
Let's go to our Pipeline and add the below code Pipeline Script.

```
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/Milky19/Python-System-Monitoring.git'
```

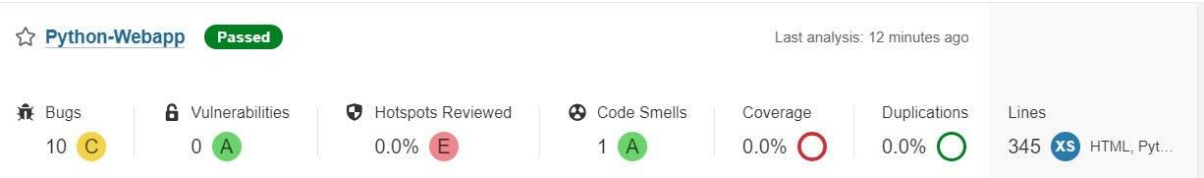
```
        }
    }
    stage("Sonarqube Analysis "){
        steps{
            withSonarQubeEnv('sonar-server') {
                sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Python-Webapp \
                -Dsonar.projectKey=Python-Webapp '''
            }
        }
    }
    stage("quality gate"){
       steps {
            script {
                waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'
            }
        }
    }
    stage("TRIVY File scan"){
        steps{
            sh "trivy fs . > trivy-fs_report.txt"
        }
    }
    stage("OWASP Dependency Check"){
        steps{
            dependencyCheck additionalArguments: '--scan ./ --format XML ', odcInstallation: 'DP-
Check'
            dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
        }
```

```
    }
  }
}
```

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



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



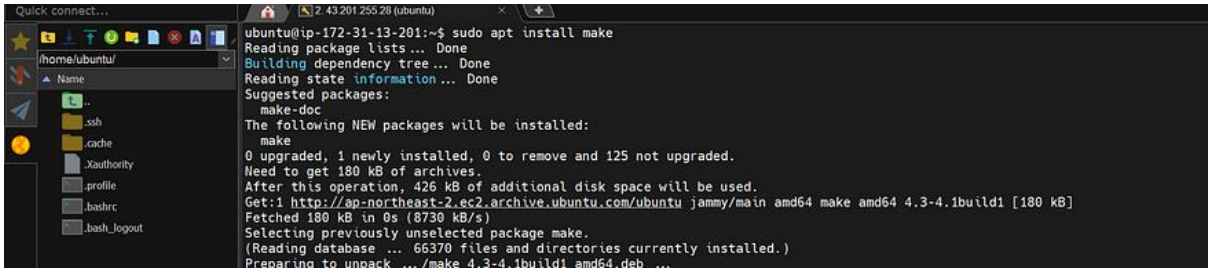
You can see the report has been generated and the status shows as passed. You can see that there are 522 lines. To see a detailed report, you can go to issues.

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

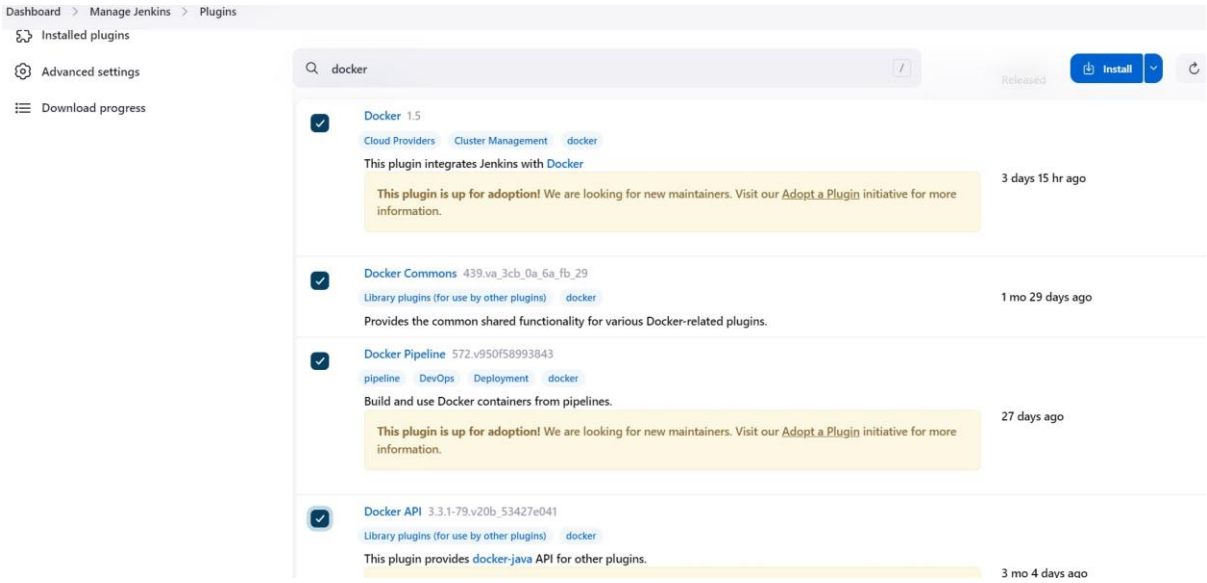
Docker Commons

Docker Pipeline

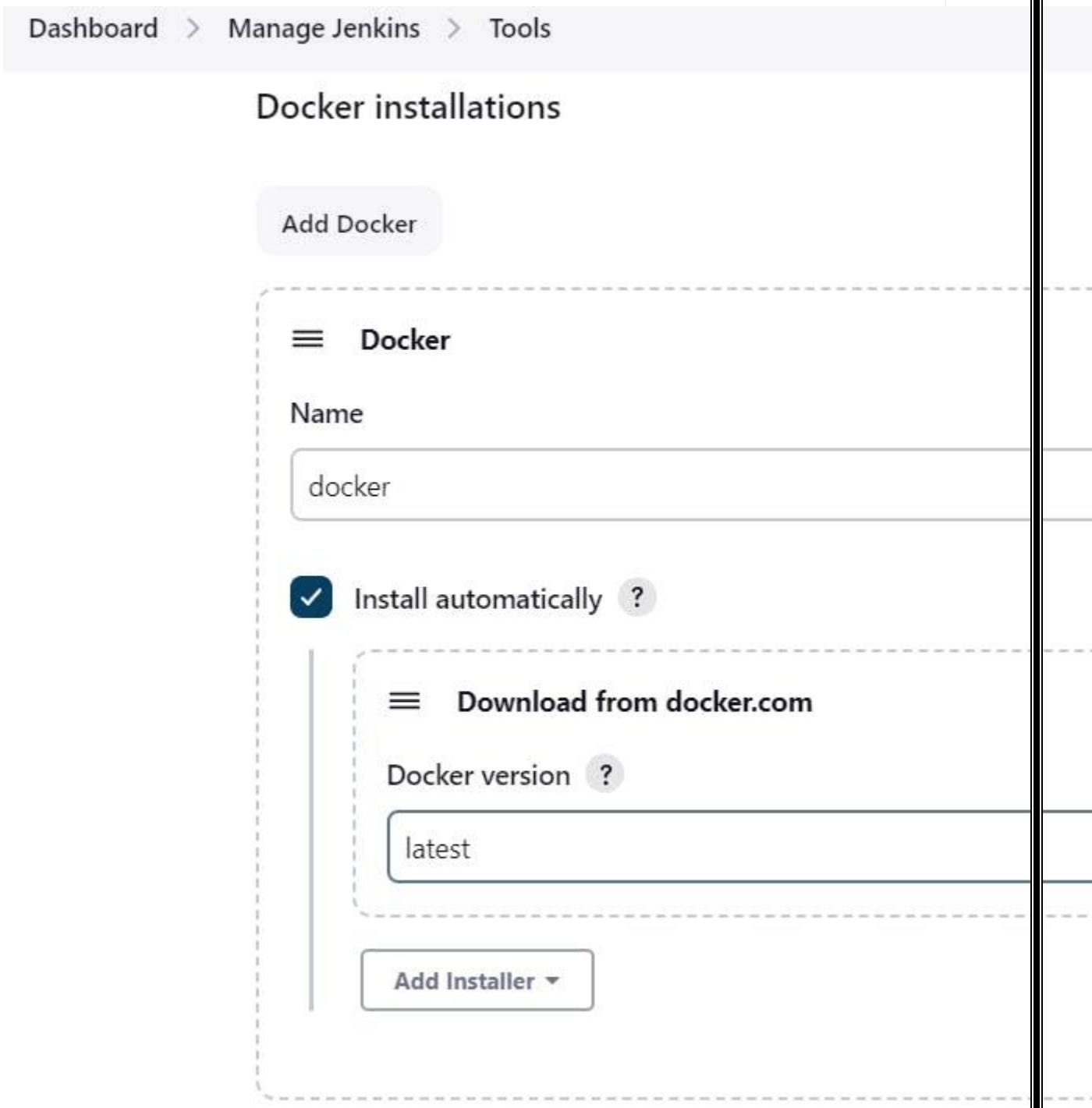
Docker API

docker-build-step

and click on install without restart



Now, goto Dashboard → Manage Jenkins → Tools →



Add DockerHub Username and Password under Global Credentials

Dashboard > Manage Jenkins > Credentials > System > Global credentials (un

Kind

Username with password

Scope ?

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

Username ?

sevenajay

Treat username as secret ?

Password ?

••••••••••••

ID ?

docker

Description ?

docker

Create

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

DotNet-monitoring / makefile

Code Blame 70 lines (55 loc) · 2.63 KB Raw

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

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

Add this stage to Pipeline Script

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

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

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

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

Step 8 — Deploy the image using Docker

Add this stage to your pipeline syntax

```
stage("Deploy to container"){
  steps{
    sh "docker run -d --name python1 -p 5000:5000 hanvitha/python-system-monitoring:latest"
  }
}
```

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

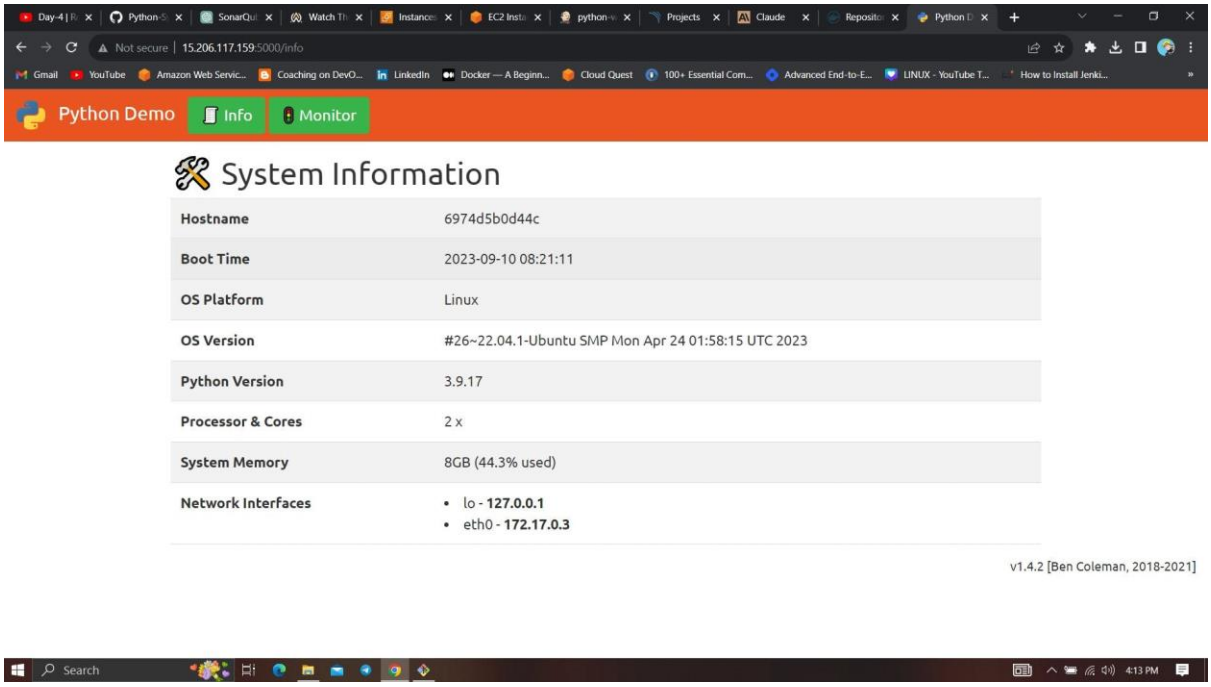
sgr-076ce119bc21d735d	Custom TCP	TCP	9000	Custom	0.0.0.0/0	Delete
sgr-09fc7d687936c1e66	HTTP	TCP	80	Custom	0.0.0.0/0	Delete
-	Custom TCP	TCP	5000	Anywh...	0.0.0.0/0	Delete

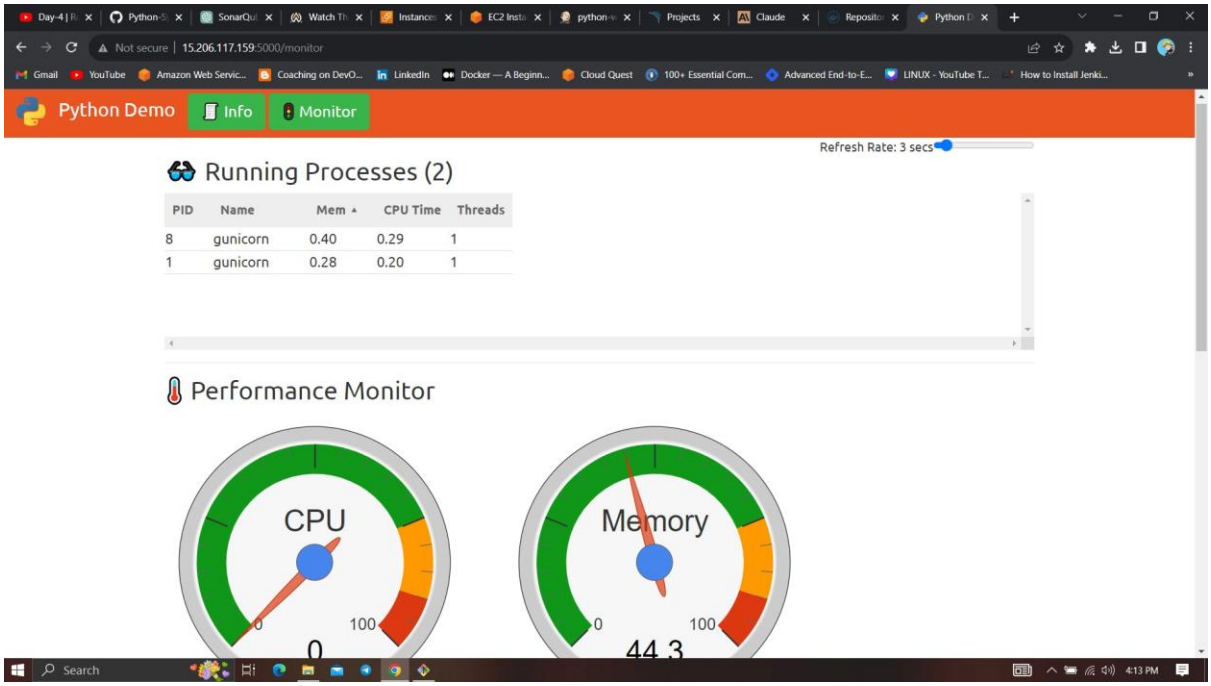
Add rule

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

<public-ip of jenkins:5000>

Step 9 — Access the Real World Application




Not secure | 15.206.117.159:5000/monitor
Python Demo
Info
Monitor
Refresh Rate: 3 secs
Running Processes (2)
PID Name Mem CPU Time Threads
8 gunicorn 0.40 0.29 1
1 gunicorn 0.28 0.20 1
Performance Monitor
CPU
0
100
0
Memory
0
100
44.3