

# Deploying ReactJS application in Kubernetes with Devsecops CI/CD pipeline

By: Shivam Joshi

Email: [shivam9joshi@gmail.com](mailto:shivam9joshi@gmail.com)

Linkedin: [www.linkedin.com/in/shivamgjoshi](https://www.linkedin.com/in/shivamgjoshi)

Tools and technologies:

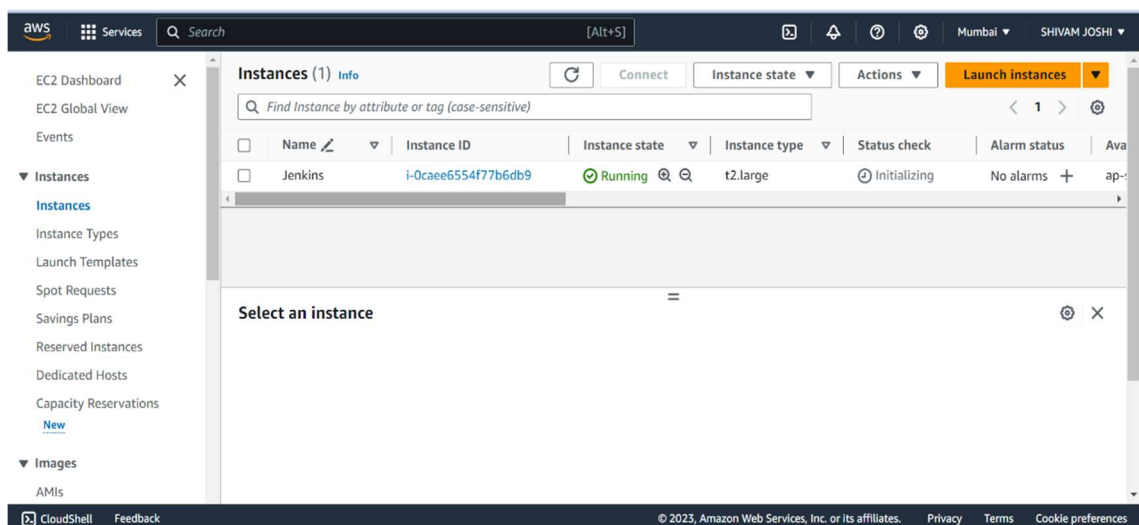
1. AWS EC2 instance (for Jenkins/Sonarqube server):
  - Operating system : Ubuntu(22.04) T2 Large Instance
  - Storage : 15 GB

For Kubernetes Master/Worker :

- Operating system : Ubuntu 20.04 T2.Medium Instance
  - Storage : 15 GB
2. Jenkins
  3. Sonarqube
  4. Trivy
  5. Docker
  6. Kubernetes

## Steps:

1. Launch an Ubuntu(22.04) T2 Large Instance for setting up Jenkins.

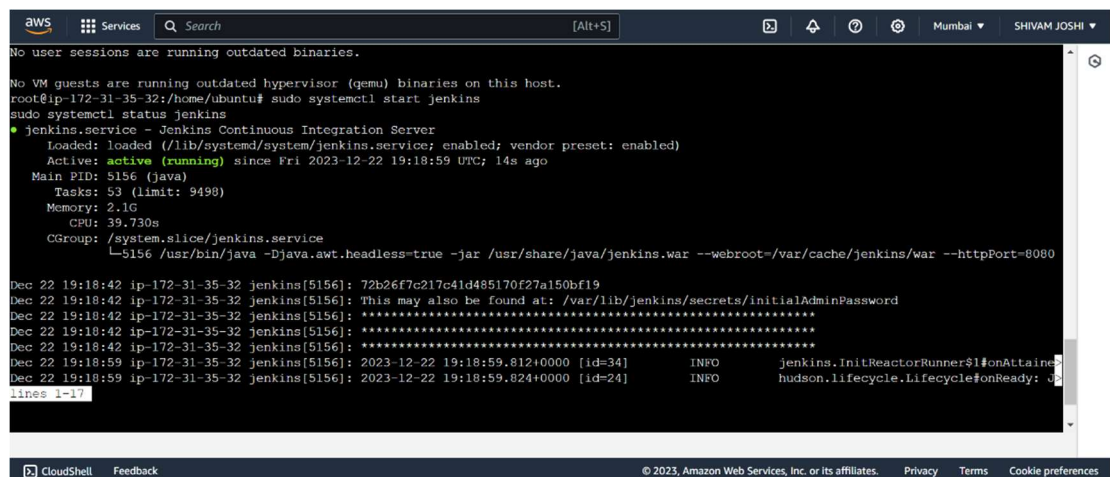


2. Install Jenkins on the EC2 instance  
**sudo apt update -y**  
**sudo apt install openjdk-17-jre -y**

```
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 install jenkins -y
sudo systemctl start jenkins
sudo systemctl status Jenkins
```

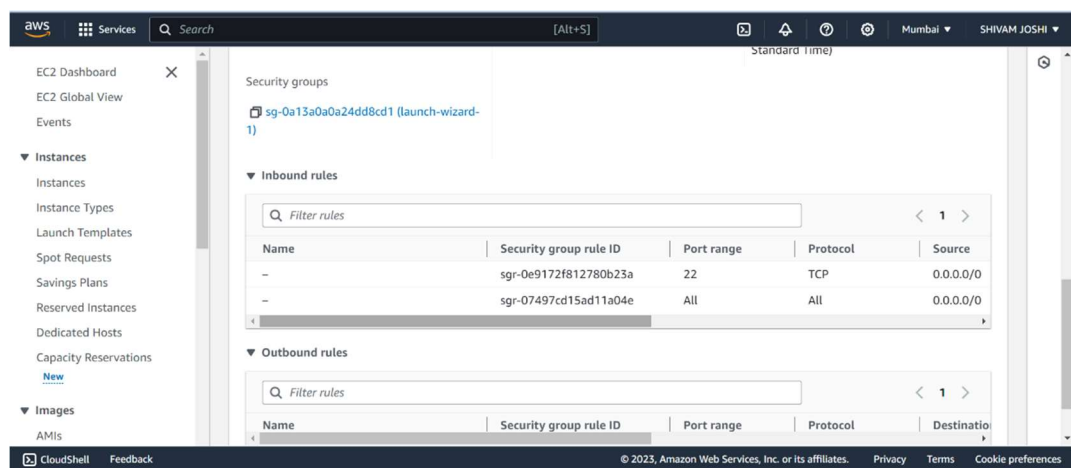


The screenshot shows the AWS CloudShell interface with a terminal window. The terminal output displays the status of the Jenkins service, which is active and running. It also shows the command to start Jenkins and the resulting logs, including the initial admin password and the Jenkins service starting up.

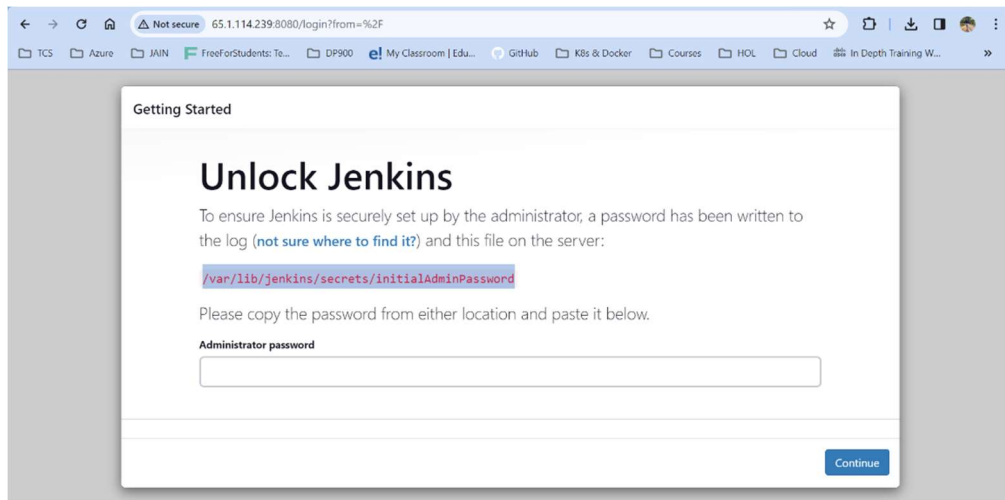
```
No user sessions are running outdated binaries.
No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-172-31-35-32:/home/ubuntu# sudo systemctl start jenkins
sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2023-12-22 19:18:59 UTC; 14s ago
     Main PID: 5156 (java)
       Tasks: 53 (limit: 9490)
        Memory: 2.1G
         CPU: 39.730s
    CGroup: /system.slice/jenkins.service
            └─5156 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Dec 22 19:18:42 ip-172-31-35-32 jenkins[5156]: 72b26f7c217c41d485170f27a150bf19
Dec 22 19:18:42 ip-172-31-35-32 jenkins[5156]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Dec 22 19:18:42 ip-172-31-35-32 jenkins[5156]: *****
Dec 22 19:18:42 ip-172-31-35-32 jenkins[5156]: *****
Dec 22 19:18:42 ip-172-31-35-32 jenkins[5156]: *****
Dec 22 19:18:59 ip-172-31-35-32 jenkins[5156]: 2023-12-22 19:18:59.812+0000 [id=34] INFO jenkins.InitReactorRunner$1 onAttaine
Dec 22 19:18:59 ip-172-31-35-32 jenkins[5156]: 2023-12-22 19:18:59.824+0000 [id=24] INFO hudson.lifecycle.Lifecycle#onReady: J
lines 1-17
```

3. Go to AWS EC2, Security Group and open Inbound Port 8080.



Type<EC2 Public IP Address:8080> in web browser to access Jenkins server.

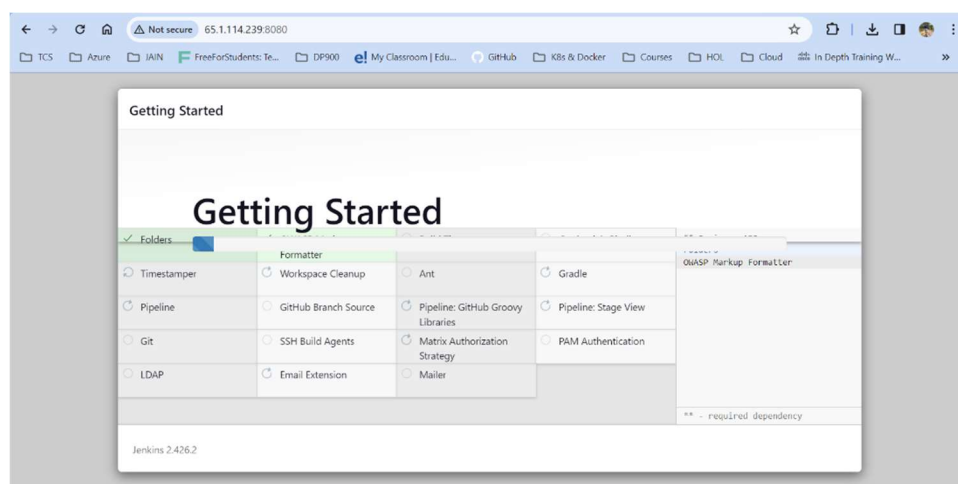


Copy the displayed path and paste it on the EC2 terminal as displayed below.  
**sudo cat /var/lib/jenkins/secrets/initialAdminPassword**

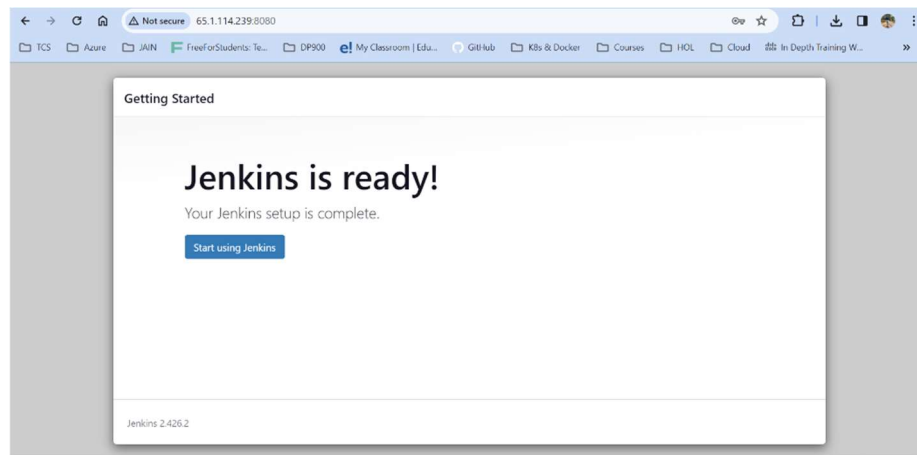
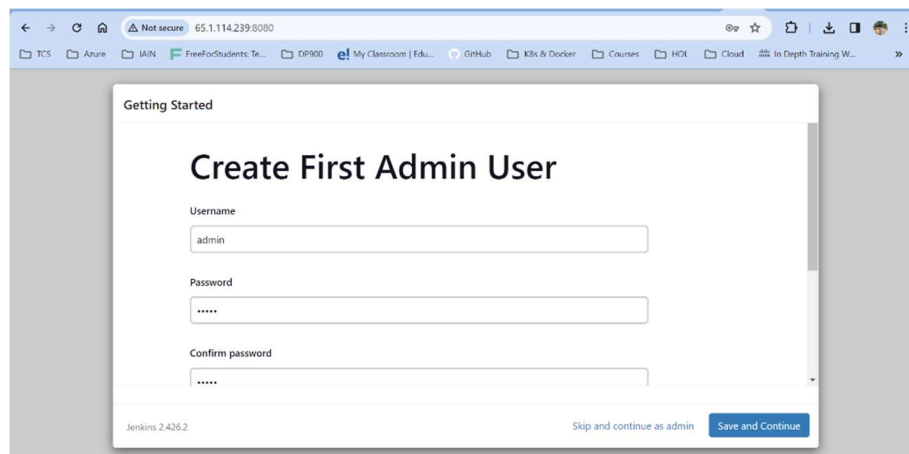
Copy the token and paste it in Jenkins administrator password section.



#### 4. Install the suggested plugins



5. Now create a user in Jenkins with own username, password and email.



6. You're doing really well! Let now install **Docker** and **Sonarqube**

```
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
```

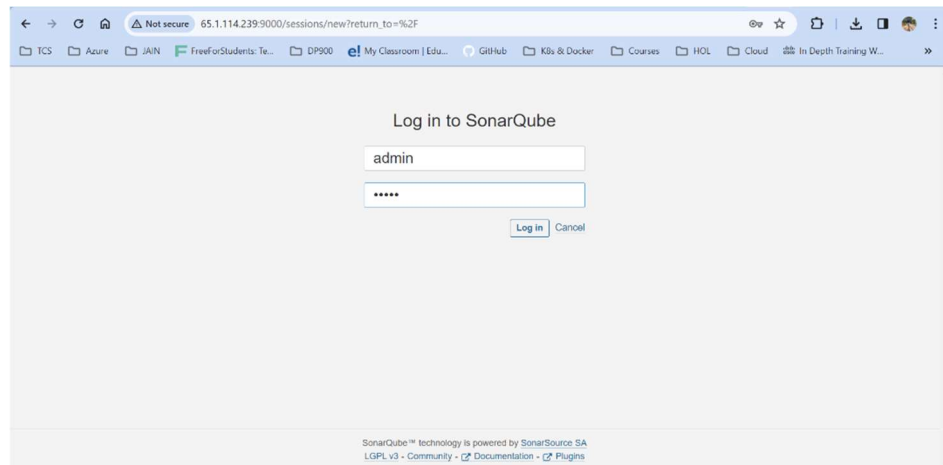
7. After the docker installation, we will install sonarqube using a Sonarqube container (Remember to add 9000 ports in the security group as done for Jenkins).

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

8. Enter public ip of EC2 instance and the port number in web browser.  
<public-ip-of-EC2:9000>

9. Enter username and password, click on login and change password

**Note:** For the first time the default credentials will be username= admin and password =admin, following this update new password.



10. Our infrastructure needs some security, so now we will 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**

11. Now install plugins like JDK, Sonarqube Scanner, NodeJS, OWASP dependency check.

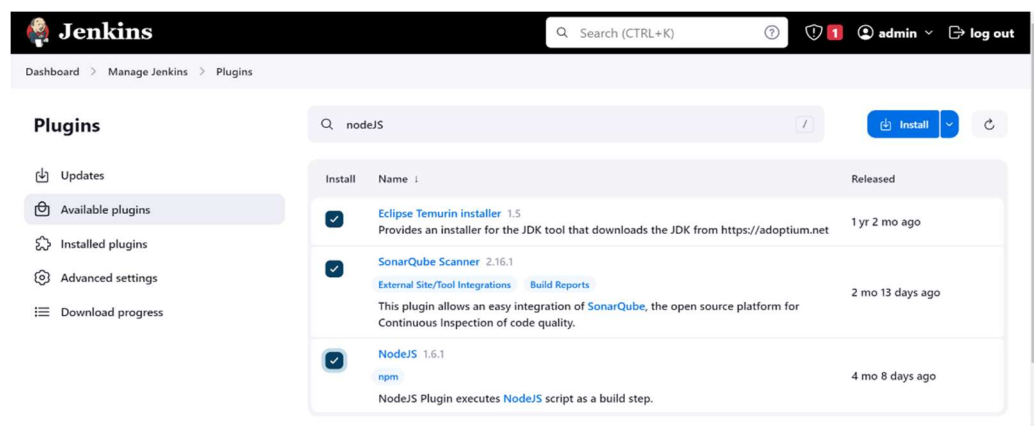
Install plugins

Go to Manage Jenkins → Plugins → Available Plugins → Install below plugins

→ Eclipse Temurin Installer (Install without restart)

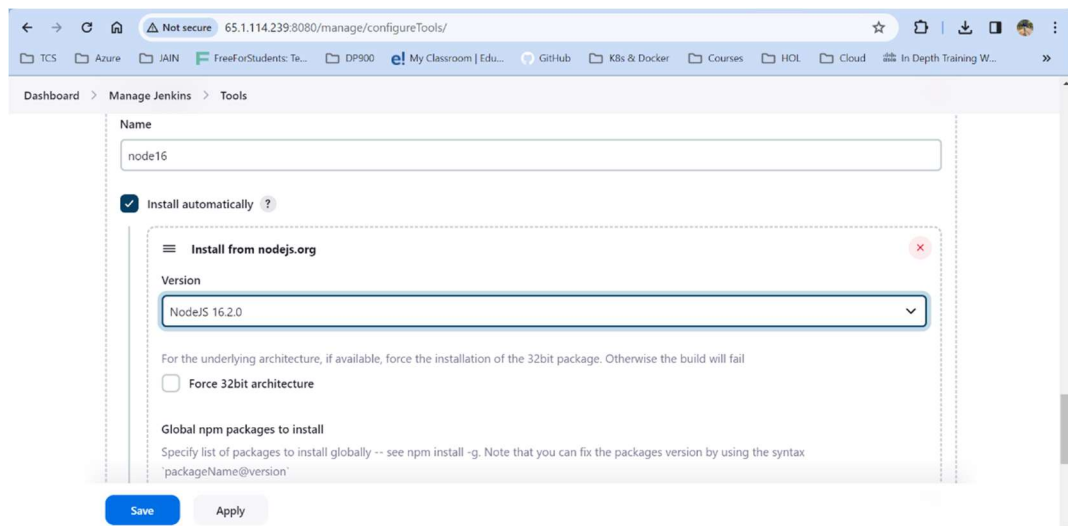
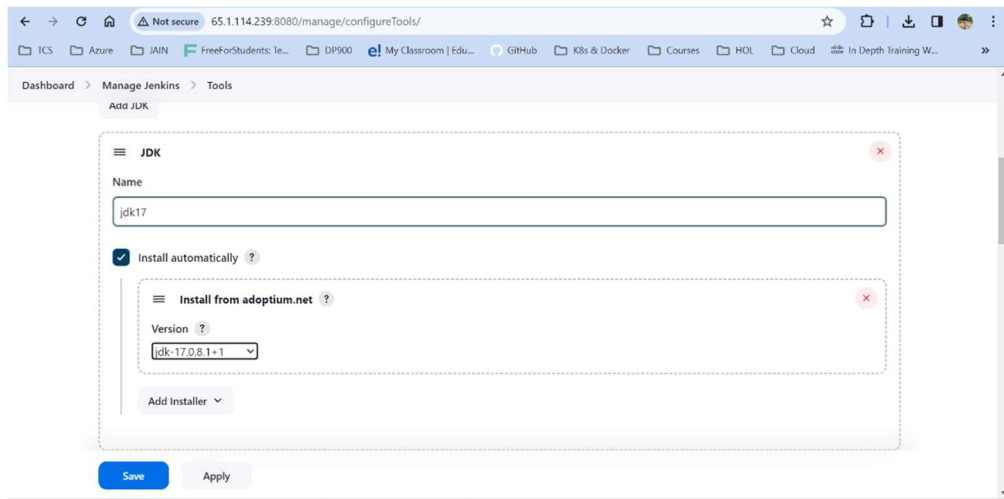
→ Sonarqube Scanner (Install without restart)

→ NodeJS Plugin (Install Without restart)



## 12. Configure Java and Nodejs in Global tool configuration

Goto Manage Jenkins → Tools → Install JDK(17) and NodeJs(16)→ Click on Apply and Save.



## 13. Create a Job

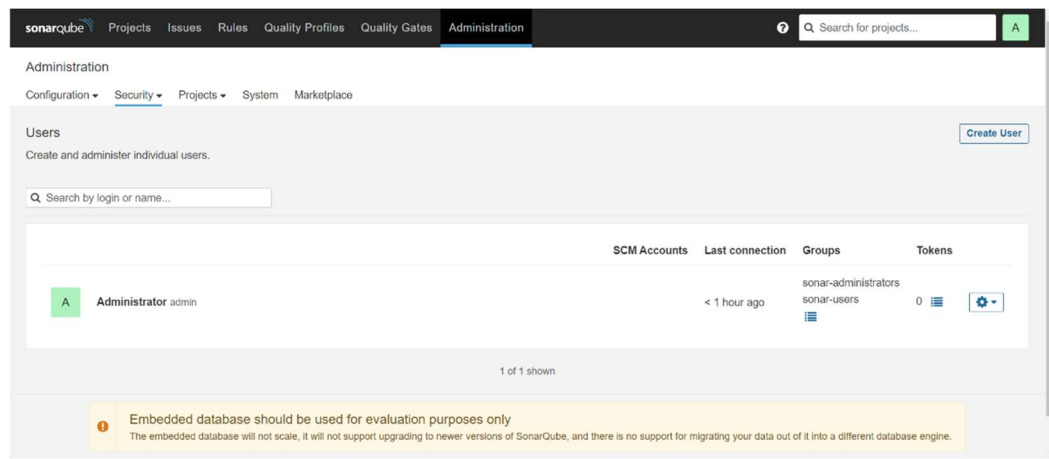
create a job as Devsecops\_demo Name, select pipeline and click ok.

## 14. Configure Sonar Server in Manage Jenkins

Copy the Public IP of your EC2 instance, Sonarqube works on port 9000, <Public IP EC2>:9000.

Goto your Sonarqube Server.

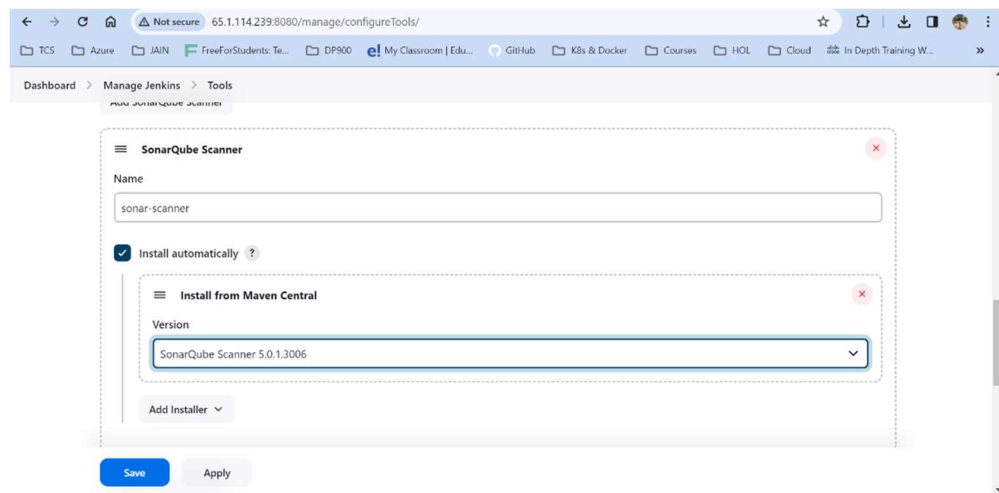
Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → click on Generate Token→Copy the token.



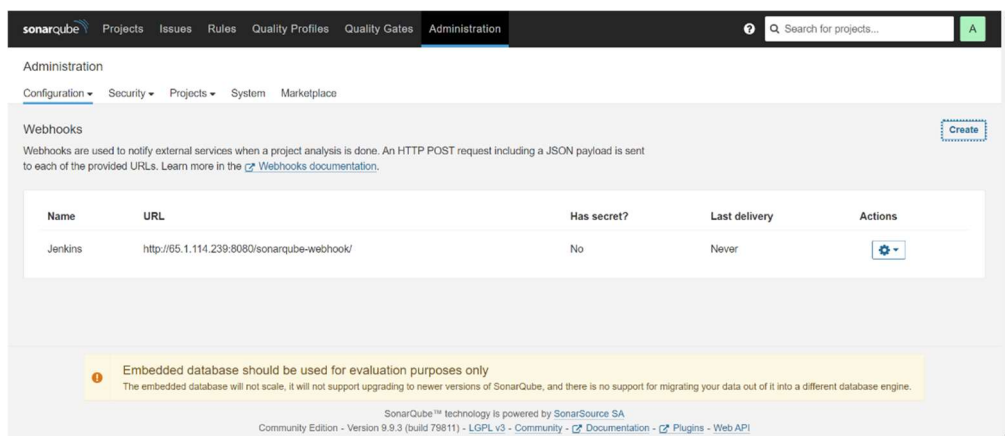
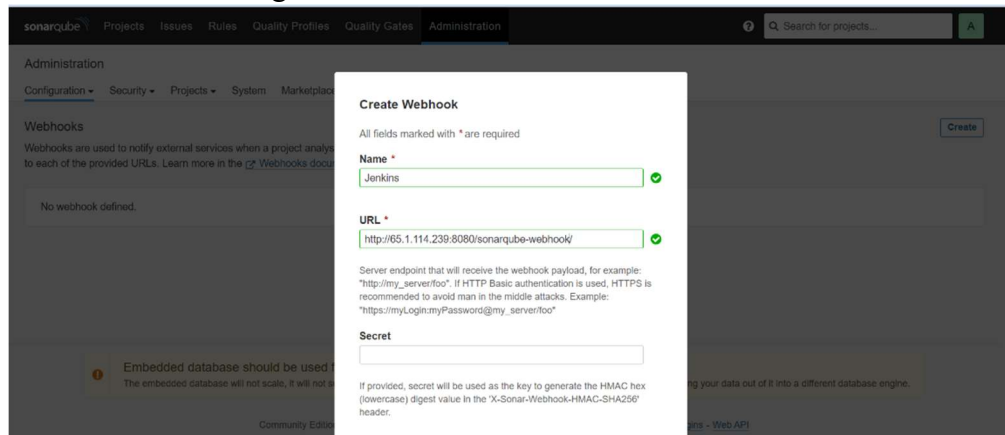
Goto Jenkins Dashboard → Manage Jenkins → Credentials → Add Secret Text.

Now, go to Dashboard → Manage Jenkins → System and Add like the below image. Click on Apply and Save.

15. We will install a sonar scanner in the tools.



16. In the Sonarqube Dashboard add a quality gate also  
Administration → Configuration → Webhooks



17. Let's go to our Pipeline and add the script in our Pipeline Script.

<https://github.com/AWS-AZURE-Bootcamp5/Devsecops-Project1/blob/main/Jenkinsfile1>



Dashboard > Devsecops\_project > Configuration

### Configure

- General
- Advanced Project Options
- Pipeline**

Definition: Pipeline script

```

1 = pipeline{
2   agent any
3   tools{
4     jdk 'jdk17'
5     nodejs 'node16'
6   }
7   environment {
8     SCANNER_HOME=tool 'sonar-scanner'
9   }
10  stages {
11    stage('clean workspace'){
12      steps{
13        cleanws()
14      }
15    }
16    stage('checkout from git'){
17
18    }
19  }
20 }
  
```

☒ Use Groovy Sandbox

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

---

Dashboard > Devsecops\_project > Configure

[Delete Pipeline](#)

[Full Stage View](#)

[Rename](#)

[Pipeline Syntax](#)

### Stage View

Average stage times:

	Declarative: Tool Install	clean workspace	Checkout from Git	Sonarqube Analysis	quality gate	Install Dependencies	OWASP FS SCAN
30s	493ms	1s	112ms	57ms	57ms	57ms	

Build History: trend

Filter builds...

#2 Dec 22, 2023, 8:12 PM

#1 Dec 22, 2023, 8:08 PM

[Atom feed for all](#) [Atom feed for failures](#)

### Permalinks

## 18. Install OWASP Dependency Check Plugins

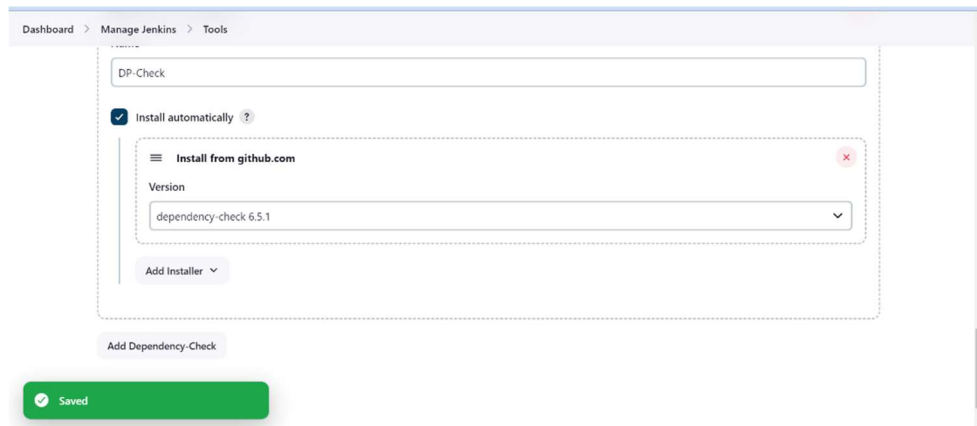
Go to Dashboard → Manage Jenkins → Plugins → OWASP Dependency-Check. Click on it and install it without restart.

Dashboard > Manage Jenkins > Plugins

Pipeline: REST API	Success
Pipeline: Stage View	Success
Git	Success
SSH Build Agents	Success
Matrix Authorization Strategy	Success
PAM Authentication	Success
LDAP	Success
Email Extension	Success
Mailer	Success
Loading plugin extensions	Success
Eclipse Temurin installer	Success
SonarQube Scanner	Success
Config File Provider	Success
NodeJS	Success
Loading plugin extensions	Success
OWASP Dependency-Check	Success
Loading plugin extensions	Success

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

Go to Dashboard → Manage Jenkins → Tools



Now go configure → Pipeline and add OWASP and TRIVY stage to your pipeline and build.

## 19. 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:

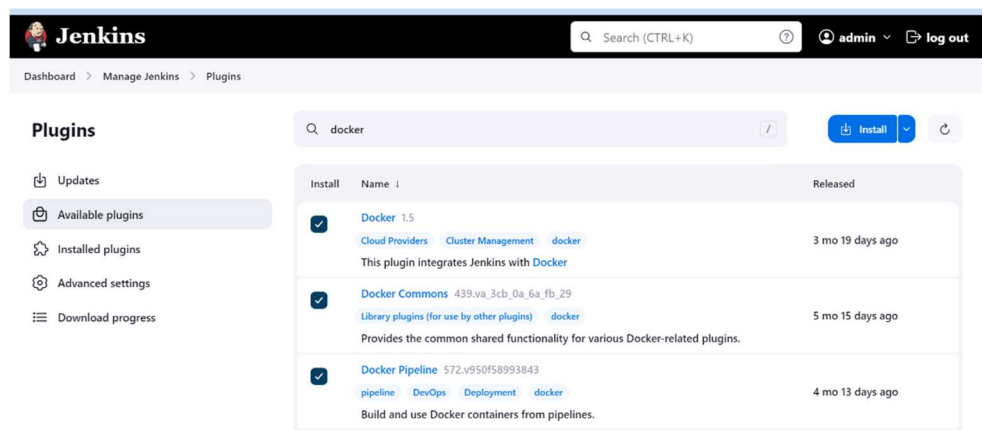
Docker

Docker Commons

Docker Pipeline

Docker API

Docker-build-step



Now, goto Dashboard → Manage Jenkins → Tools

Dashboard > Manage Jenkins > Tools

Docker

Name

docker

☒ Install automatically ?

Download from docker.com

Docker version ?

latest

Add Installer ▾

Save

Apply

Add Docker Hub Username and Password under Global Credentials

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

### New credentials

Kind  
Username with password ▾

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

Username ?  
[REDACTED]

☐ Treat username as secret ?

Password ?  
[REDACTED]

Create

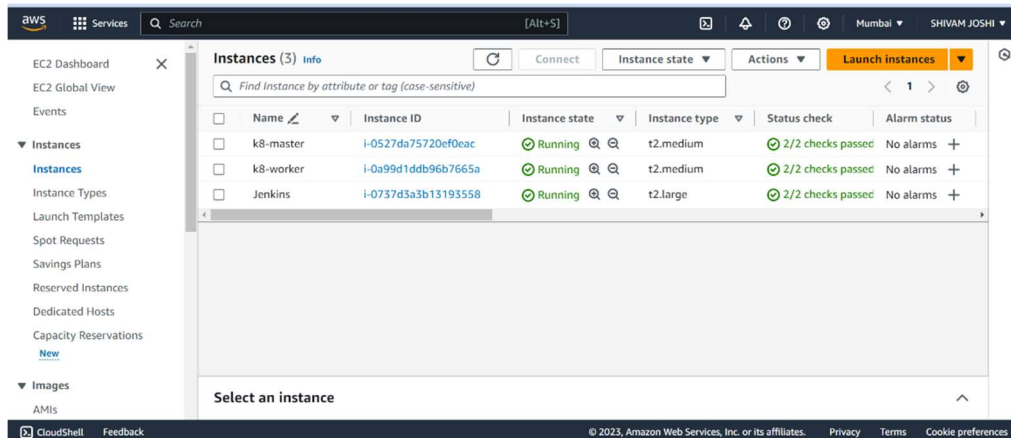
Execute the pipeline again.

		Declarative: Tool Install	clean workspace	Checkout from Git	Sonarqube Analysis	quality gate	Install Dependencies	OWASP FS SCAN	TRIVY FS SCAN	Docker Build & Push	TRIVY	Deploy to container	Deploy to kubernets
Average stage times:		6s	386ms	2s	20s	348ms	2min 47s	3min 55s	11s	56s	25s	263ms	142ms
130	No Changes	146ms	298ms	1s	23s	321ms	1min 39s	2min 42s	23s	1min 40s	2min 7s	1s	419ms failed
137	No Changes	219ms	410ms	1s	28s	348ms	2min 6s	2min 38s	23s	3min 2s failed	56ms failed	48ms failed	49ms failed
137	No Changes	155ms	415ms	1s	24s	497ms	2min 34s	8min 8s	8s aborted	129ms aborted	119ms aborted	103ms aborted	112ms aborted
134	No Changes	185ms	317ms	4s	28s	517ms (paused for 501ms)	7min 33s	6min 6s failed	56ms failed	82ms failed	72ms failed	62ms failed	66ms failed

## 20. Kubernetes setup

Deploy 2 Ubuntu 20.04 instances for kubernetes **master** and **worker**. T2.Medium 15 GB

Install Kubectl on Jenkins machine aswell.



- **sudo apt update**
- **sudo apt install curl -y**
- **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**

21. Set hostname for Master node

- **sudo hostnamectl set-hostname K8s-Master**

22. Set hostname for Master node

- **sudo hostnamectl set-hostname K8s-Worker**

23. Install Kubeadm/Kubelet/kubectl on Master and Worker

- **sudo apt-get update**
- **sudo apt-get install -y docker.io**
- **sudo usermod -aG docker Ubuntu**
- **newgrp docker**
- **sudo chmod 777 /var/run/docker.sock**
- **sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -**
- **sudo tee /etc/apt/sources.list.d/kubernetes.list <<EOF**  
**deb https://apt.kubernetes.io/ kubernetes-xenial main # 3lines same command**  
**EOF**
- **sudo apt-get update**
- **sudo apt-get install -y kubelet kubeadm kubectl**
- **sudo snap install kube-apiserver**

24. On Master node

- **sudo kubeadm init --pod-network-cidr=10.244.0.0/16**
- **mkdir -p \$HOME/.kube**
- **sudo cp -i /etc/kubernetes/admin.conf \$HOME/.kube/config**
- **sudo chown \$(id -u):\$(id -g) \$HOME/.kube/config**

- **kubectl apply -f**

<https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

25. You will get the token and the command like below after executing the above command

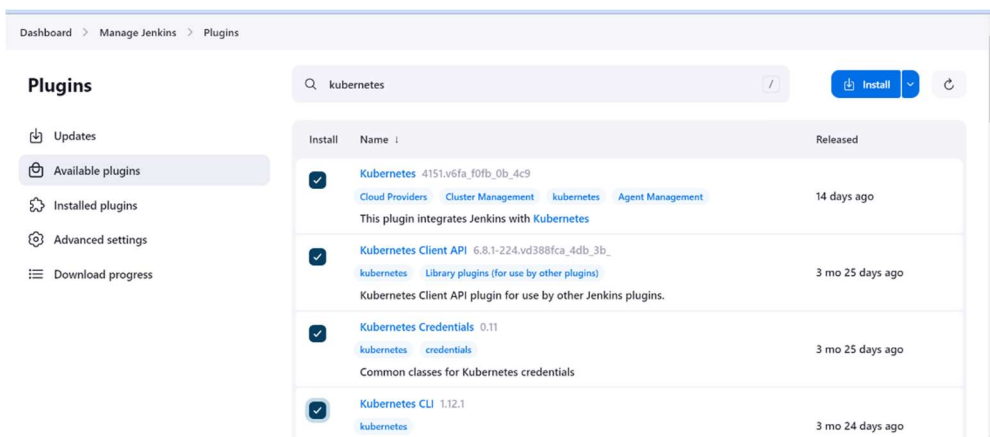
- ```
- sudo kubeadm join <master-node-ip>:<master-node-port> --token <token>
--discovery-token-ca-cert-hash <hash>
```

Copy above token and paste in worker node

26. Copy the config file from K8 MASTER to the local laptop and save it with a name secret-file.txt and use this at the Kuberenetes credential section.

[illegible]

## 27. Install Kubernetes plugin



28. Go to manage Jenkins → manage credentials → Click on Jenkins global → add credentials.

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

### New credentials

Kind  
Secret file

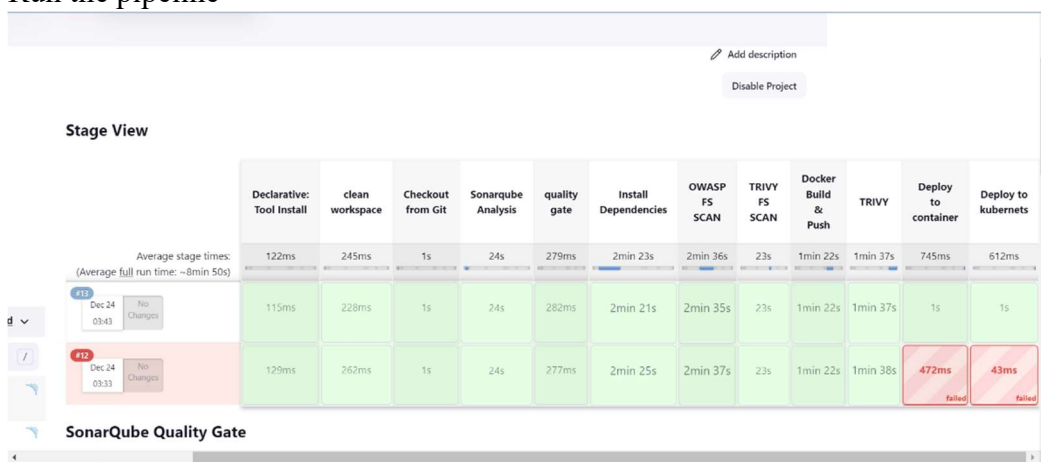
Scope ?  
Global (Jenkins, nodes, items, all child items, etc)

File  
Choose File secret-file.txt

ID ?  
k8s

Create

## 29. Run the pipeline



Hurray!!! We have successfully deployed the CI/CD pipeline. Congratulations, our efforts have paid off.

## 30. In the Kubernetes cluster give this command

- **kubectl get all**
- **kubectl get svc**

## 31. Access from a Web browser with

- **<public-ip-of-slave:service port>**

## IMPORTANT:

If incase you get some error while running the pipeline at the Kubernetes stage:

+ kubectl apply -f deployment.yaml

error: error validating "deployment.yaml": error validating data: failed to download openapi: Get "https://172.31.47.5:6443/openapi/v2?timeout=32s": dial tcp 172.31.47.5:6443: i/o timeout; if you choose to ignore these errors, turn validation off with --validate=false

|                      | Declarative:<br>Tool Install | clean<br>workspace | Checkout<br>from Git | Sonarqube<br>Analysis | quality<br>gate | Install<br>Dependencies | OVVASP<br>FS<br>SCAN | TRIVY<br>FS<br>SCAN | Docker<br>Build<br>&<br>Push | TRIVY    | Deploy<br>to<br>container | Deploy to<br>kubernetes |
|----------------------|------------------------------|--------------------|----------------------|-----------------------|-----------------|-------------------------|----------------------|---------------------|------------------------------|----------|---------------------------|-------------------------|
| Average stage times: | 3s                           | 371ms              | 1s                   | 23s                   | 336ms           | 2min 44s                | 3min 23s             | 16s                 | 1min 9s                      | 1min 6s  | 439ms                     | 7s                      |
| 3<br>No<br>Changes   | 165ms                        | 378ms              | 1s                   | 25s                   | 313ms           | 2min 24s                | 2min 41s             | 23s                 | 1min 25s                     | 1min 54s | 544ms<br>failed           | 57ms<br>failed          |
| 3<br>No<br>Changes   | 194ms                        | 258ms              | 1s                   | 25s                   | 325ms           | 2min 54s                | 2min 46s             | 23s                 | 1min 27s                     | 1min 53s | 1s                        | 1min 10s<br>failed      |
| 3<br>No<br>Changes   | 155ms                        | 321ms              | 2s                   | 26s                   | 347ms           | 2min 37s                | 2min 39s             | 23s                 | 1min 27s                     | 2min 22s | 532ms<br>failed           | 50ms<br>failed          |
| 3<br>No<br>Changes   | 189ms                        | 453ms              | 1s                   | 26s                   | 302ms           | 2min 52s                | 2min 45s             | 23s                 | 1min 25s                     | 1min 38s | 527ms<br>failed           | 50ms<br>failed          |

| Stage Logs (Deploy to kubernetes)                                                                                                                                                                                                                                                                                     |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| Use a tool from a predefined Tool Installation -- jdk17 (self time 21ms)                                                                                                                                                                                                                                              |  |
| Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step. (self time 36ms)                                                                                                                                                                                     |  |
| Use a tool from a predefined Tool Installation -- node16 (self time 68ms)                                                                                                                                                                                                                                             |  |
| Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step. (self time 46ms)                                                                                                                                                                                     |  |
| Shell Script -- kubectl apply -f deployment.yaml (self time 1min 7s)                                                                                                                                                                                                                                                  |  |
| + kubectl apply -f deployment.yaml<br>error: error validating "deployment.yaml": error validating data: failed to download openapi: Get "https://172.31.47.5:6443/openapi/v2?timeout=32s": dial tcp 172.31.47.5:6443: i/o timeout;<br>if you choose to ignore these errors, turn validation off with --validate=false |  |

Follow the below method to troubleshoot the error:

Error Message:

- "error validating 'deployment.yaml'": There's an issue with the validation of a file named "deployment.yaml."
- "error validating data: failed to download openapi:": The specific problem is a failure to download OpenAPI data, which is needed for validation.
- "Get 'https://172.31.47.5:6443/openapi/v2?timeout=32s': dial tcp 172.31.47.5:6443: i/o timeout": The attempt to retrieve the OpenAPI data from the specified server and port resulted in a timeout, indicating a connection issue.

#### Possible Causes:

1. Network Connectivity Issues:
  - Check if the server at 172.31.47.5 is reachable and the port 6443 is open.
  - Verify network connectivity and firewall rules.
2. Server-Side Issues:
  - The server might be down or experiencing problems.
  - The OpenAPI endpoint might not be configured correctly.
3. Timeout Configurations:
  - The timeout value of 32 seconds might be too short for the connection.
4. Validation Tool Issues:
  - The validation tool itself might have bugs or configuration issues.

#### Resolving the Error:

1. Check Network Connectivity:
  - Use tools like ping or telnet to test connectivity to the server and port.
2. Investigate Server Status:
  - Review server logs for errors or indications of problems.
  - Ensure the OpenAPI endpoint is running correctly.
3. Adjust Timeouts (if applicable):
  - If possible, increase the timeout value for the validation tool.
4. Consider Disabling Validation (Temporary Workaround):
  - If necessary, use the `--validate=false` flag to temporarily bypass validation, but proceed with caution as this could mask potential issues.

**Please terminate your instance to avoid extra billing cost.**

References: <https://www.youtube.com/live/mdbS5Hu1NnQ?si=T3uu4VBVhspEQ6PR>