Deploying ReactJS application in Kubernetes with Devsecops CI/CD pipeline

By: Shivam Joshi

Email: shivam9joshi@gmail.com

Linkedin: 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

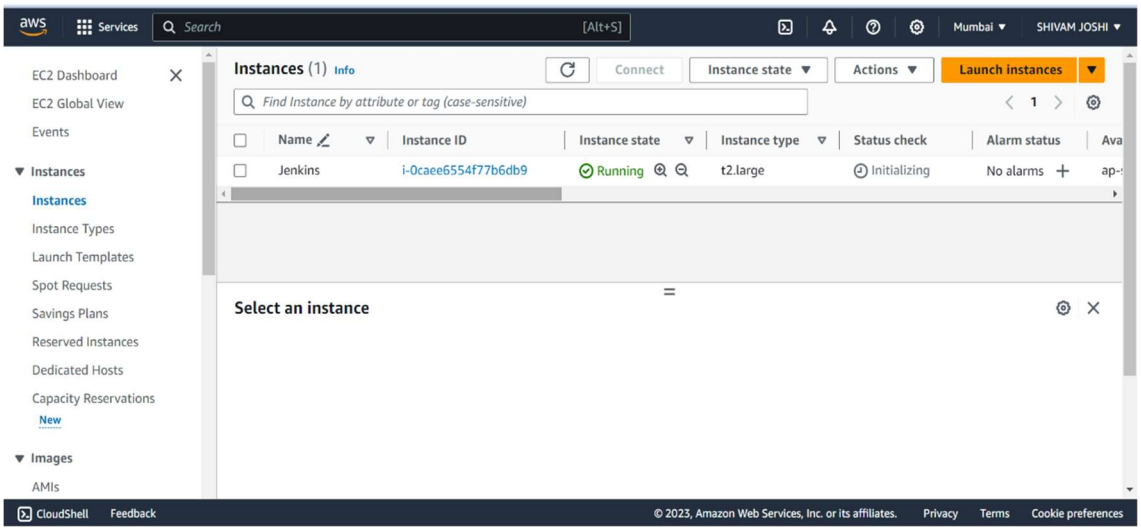
   For Kubernetes Master/Worker :

   - Operating system : Ubuntu 20.04 T2.Meduim 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.

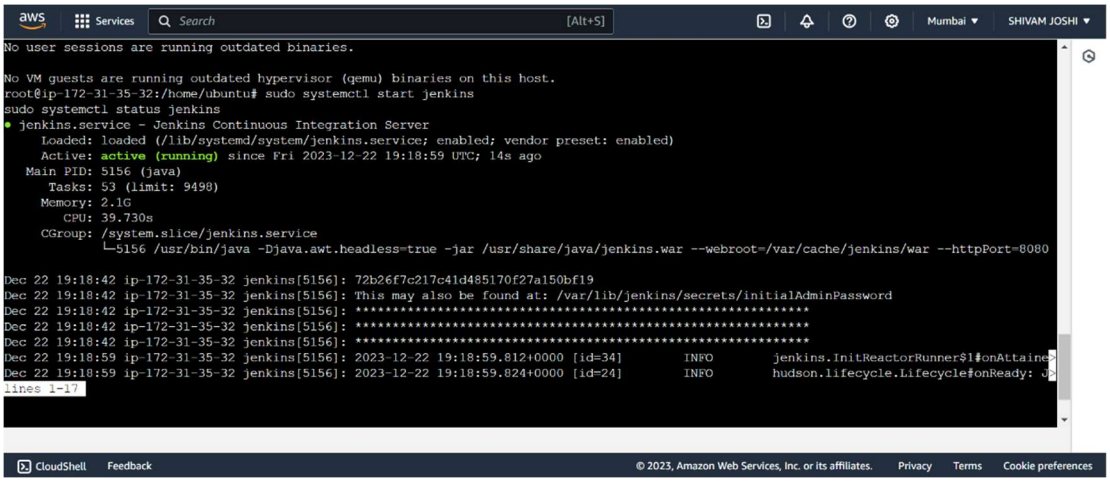


2. Install Jenkins on the EC2 instance
   sudo apt update -y
   sudo apt install openjdk-17-jre -y

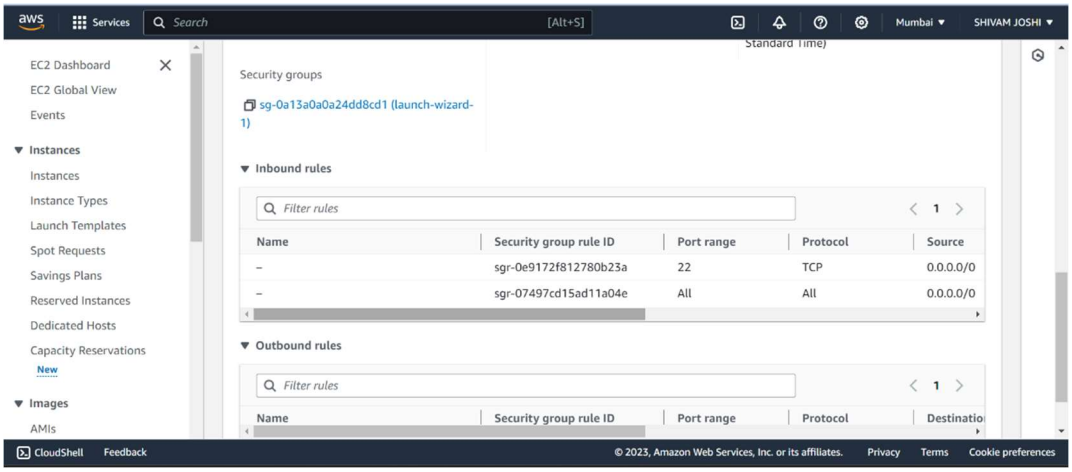
**curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null**

**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**

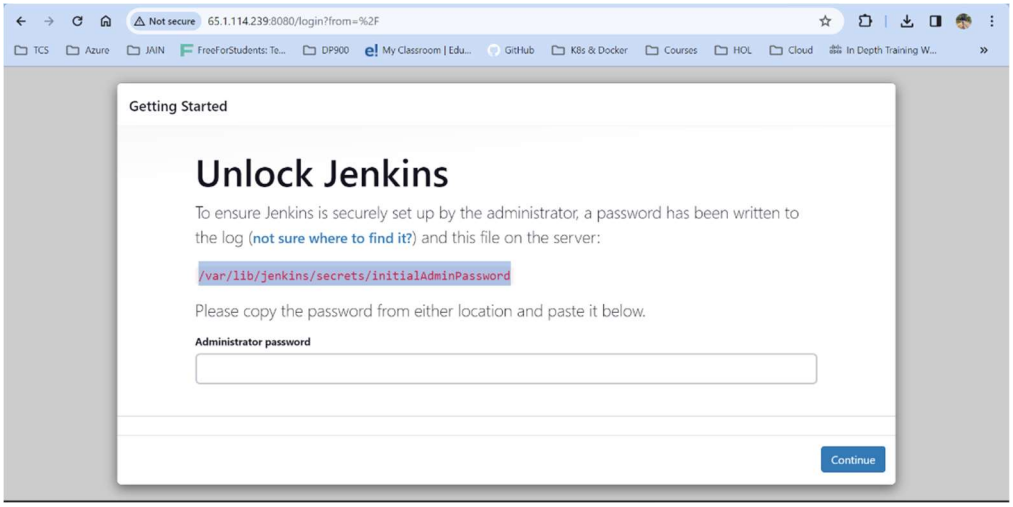
sudo apt-get install jenkins -y
sudo systemctl start jenkins
sudo systemctl status Jenkins



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

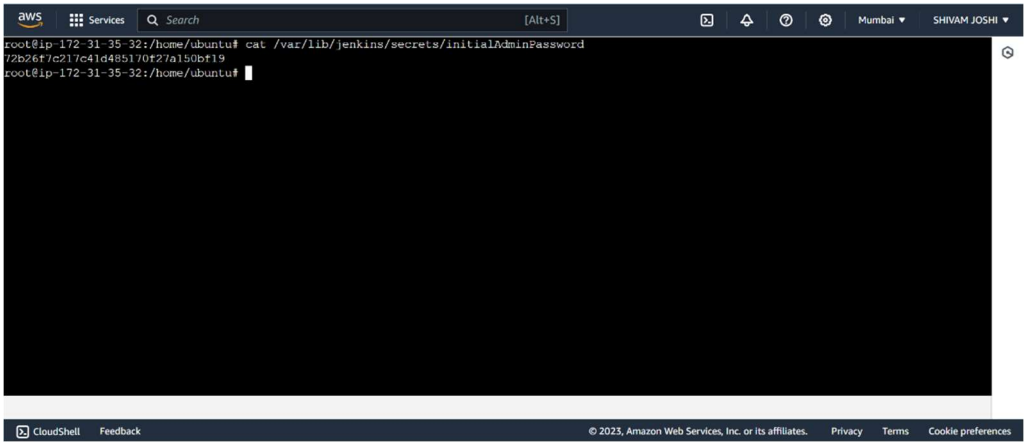


Type<EC2 Public IP Address:8080> in web browser to access Jenkins server.

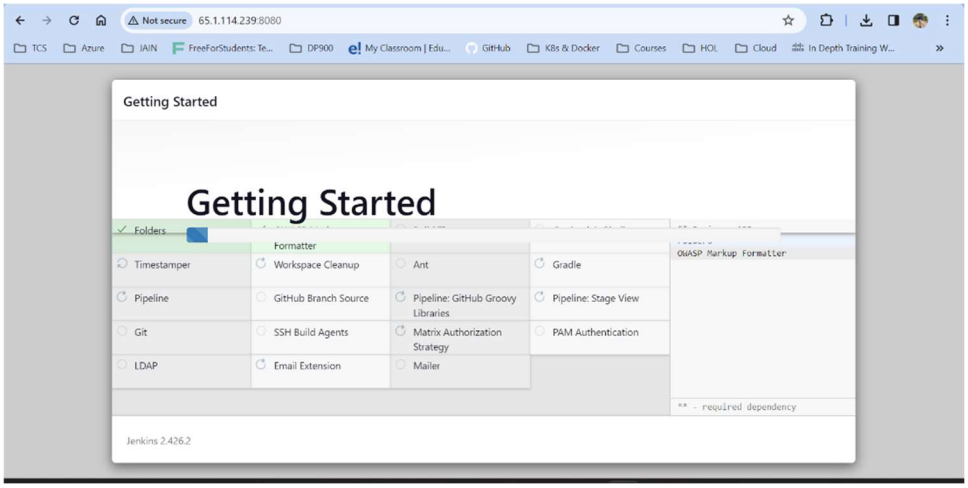


Copy the displayed path and paste it on the EC2 terminal as displayed below.
sudo cat /var/lib/jenkins/secrets/initialAdminPassword

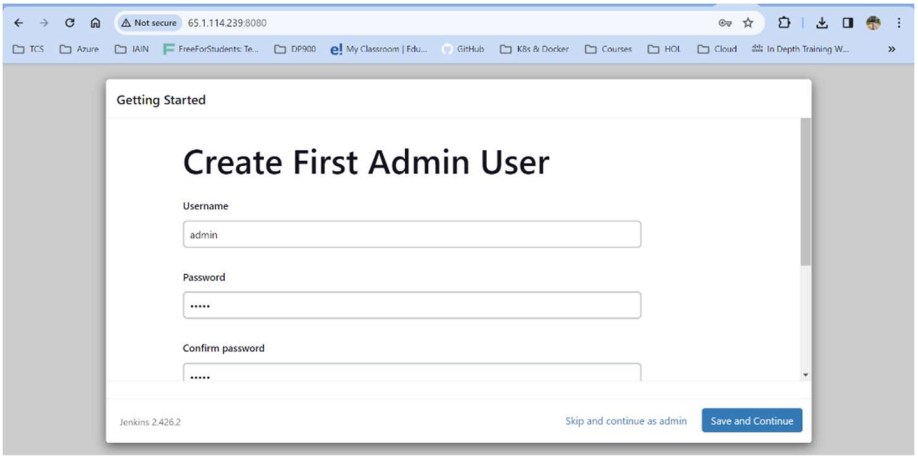
Copy the token and paste it in Jenkins administrator password section.

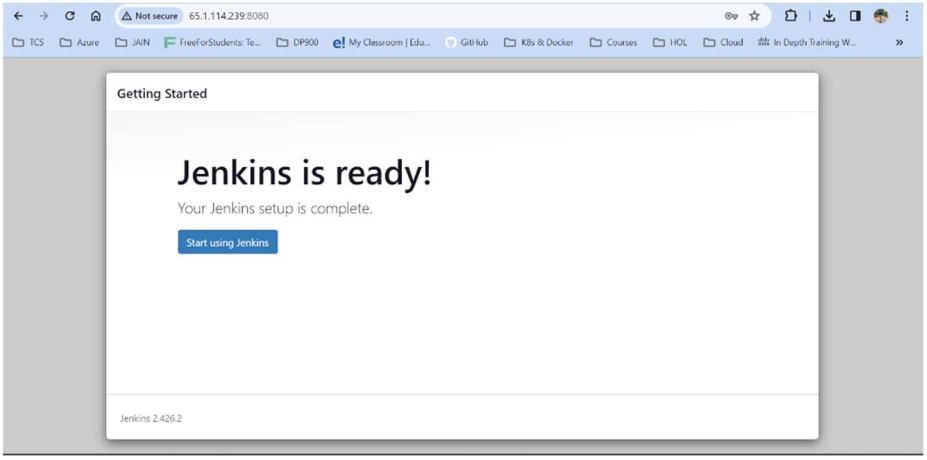


4. Install the suggested plugins



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

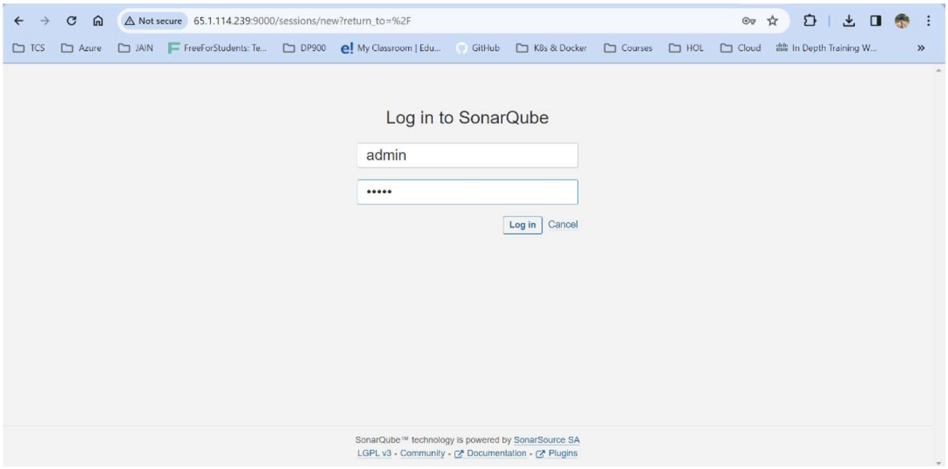
7. After the docker installation, we will install sonarqube using a Sonarqube container (Remember to add 9000 ports in the security group as done for Jenkins).

   - **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>

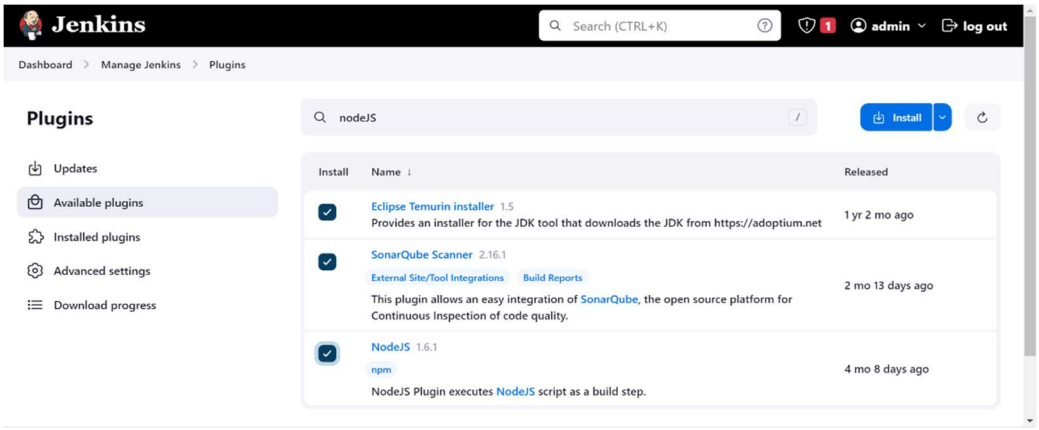
9. Enter username and password, click on login and change password

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

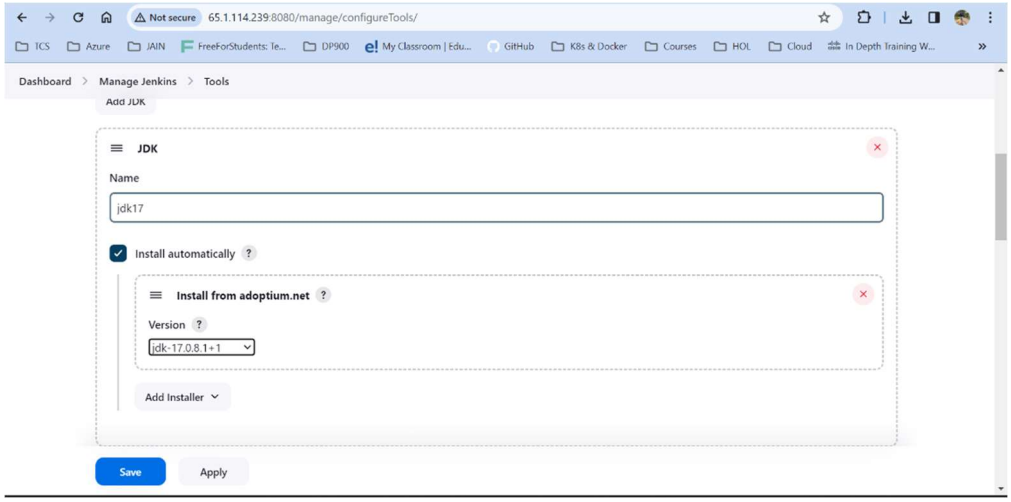


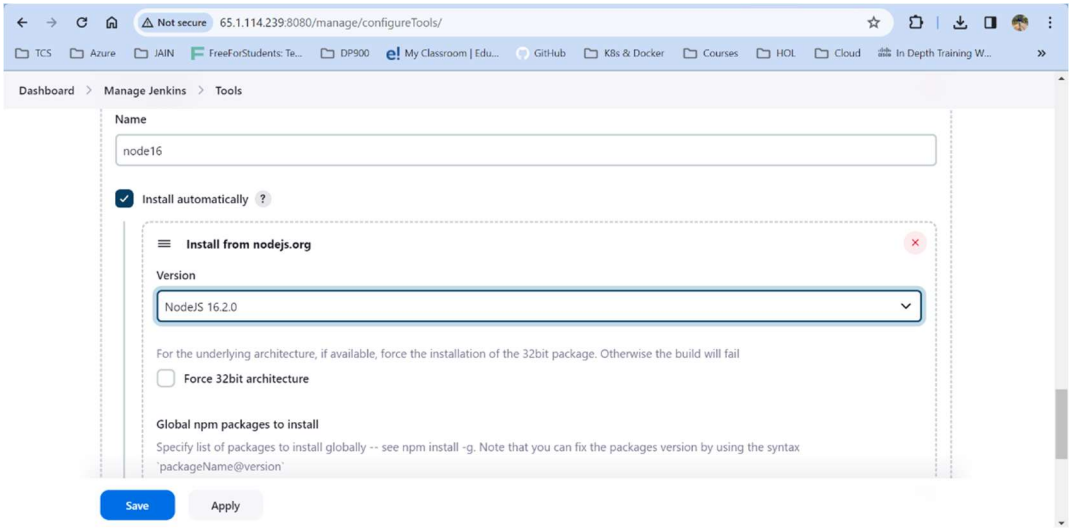
10. Our infrastructure needs some security, so now we will install **Trivy.**
    - **sudo apt-get install wget apt-transport-https gnupg lsb-release -y**
    - **wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null**
    - **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)



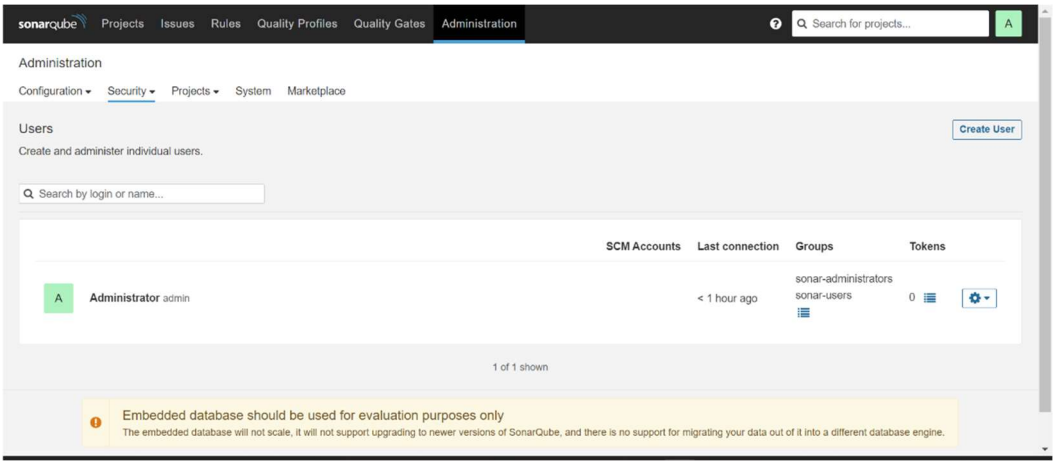
12. Configure Java and Nodejs in Global tool configuration
    Goto Manage Jenkins → Tools → Install JDK(17) and NodeJs(16)→ Click on Apply and Save.





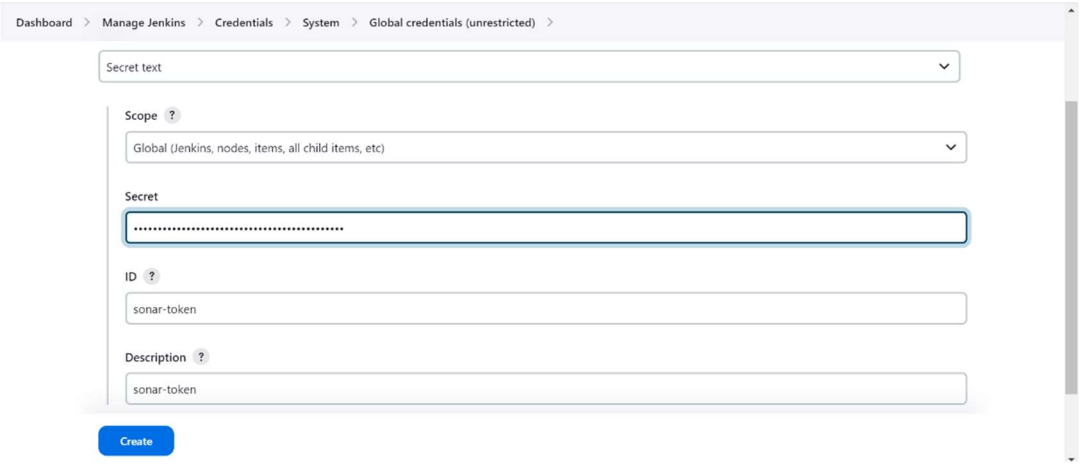
13. Create a Job
    create a job as Devsecops_demo Name, select pipeline and click ok.

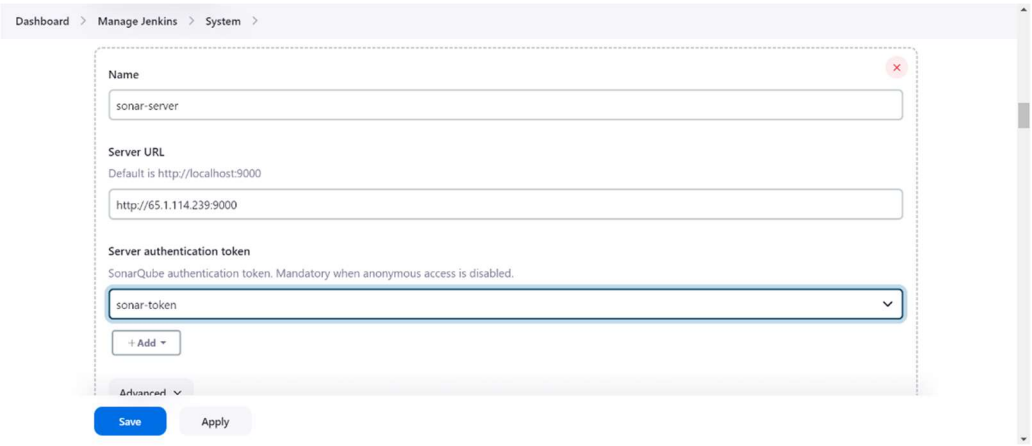
14. Configure Sonar Server in Manage Jenkins
    Copy the Public IP of your EC2 instance, Sonarqube works on port 9000, <Public IP EC2>:9000.
    Goto your Sonarqube Server.
    Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → click on Generate Token→Copy the token.



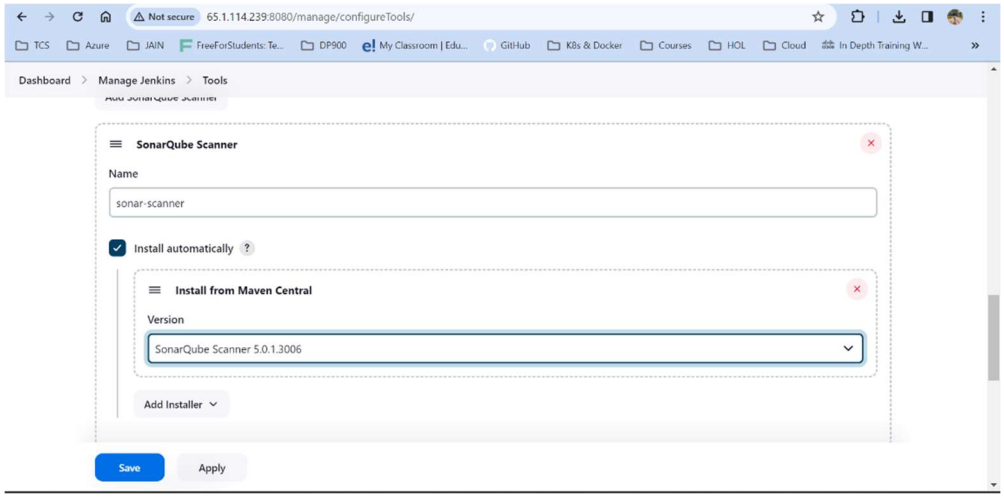
Goto Jenkins Dashboard → Manage Jenkins → Credentials → Add Secret Text.

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

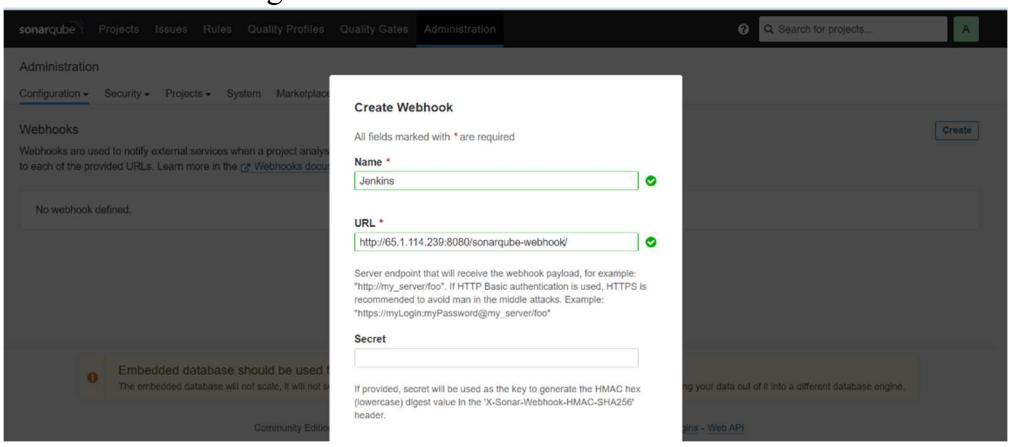


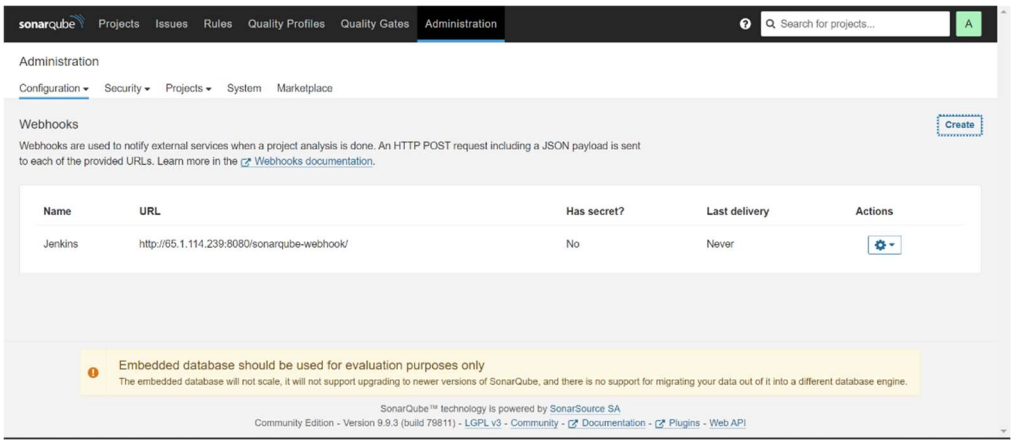


15. We will install a sonar scanner in the tools.

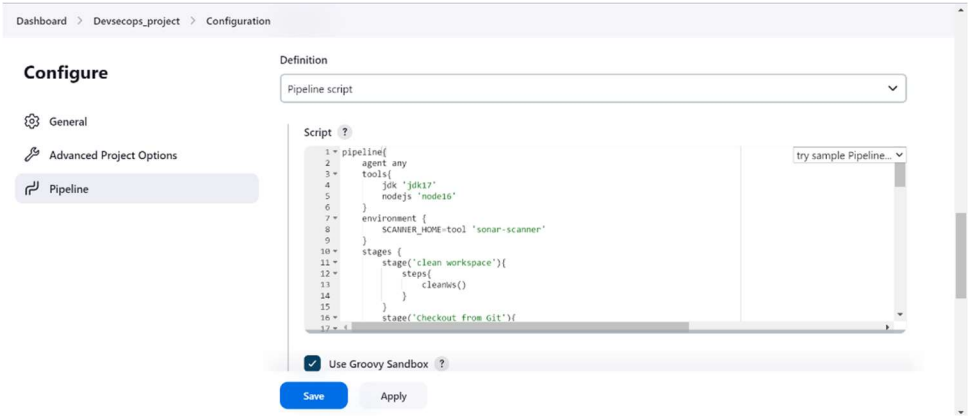


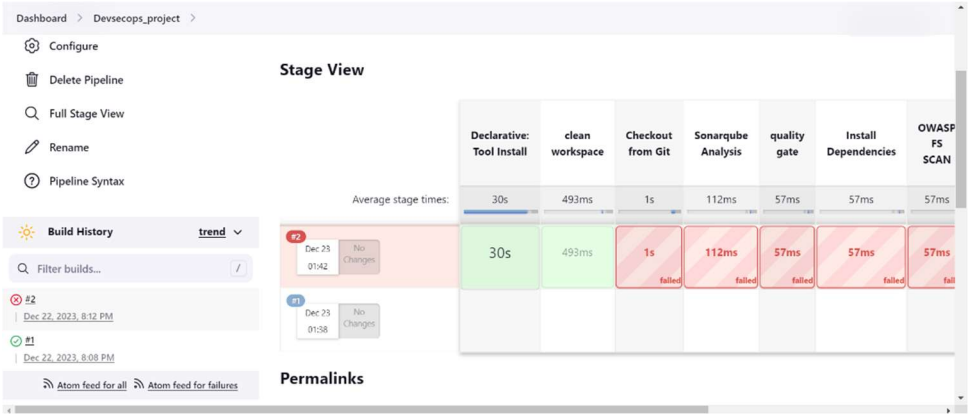
16. In the Sonarqube Dashboard add a quality gate also
    Administration → Configuration →Webhooks



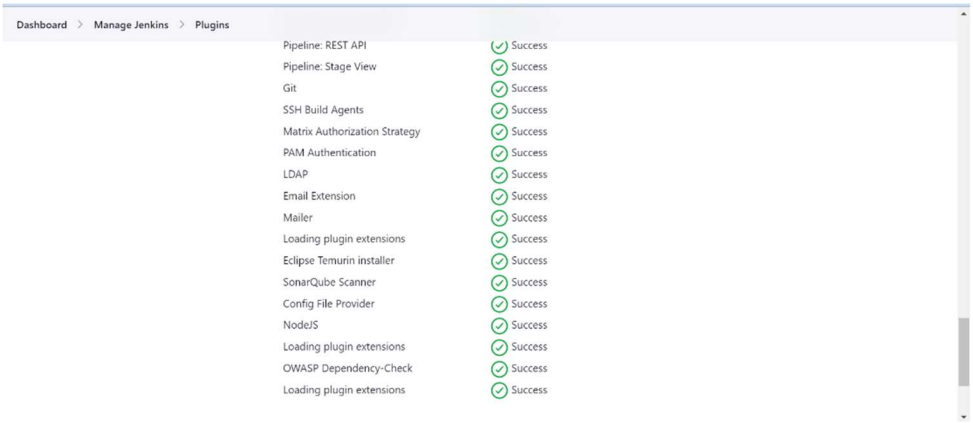


17. Let's go to our Pipeline and add the script in our Pipeline Script.
    https://github.com/AWS-AZURE-Bootcamp5/Devsecops-Project1/blob/main/Jenkinsfile1



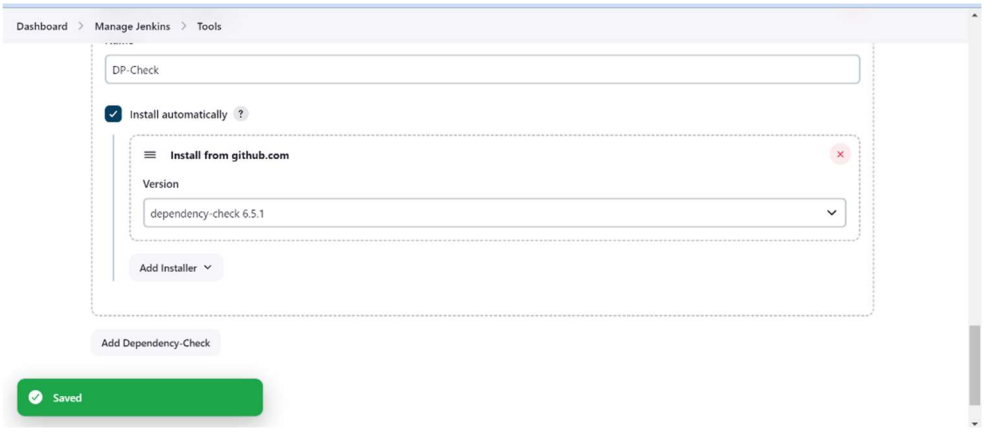


18. Install OWASP Dependency Check Plugins
    Go to Dashboard → Manage Jenkins → Plugins → OWASP Dependency-Check. Click on it and install it without restart.



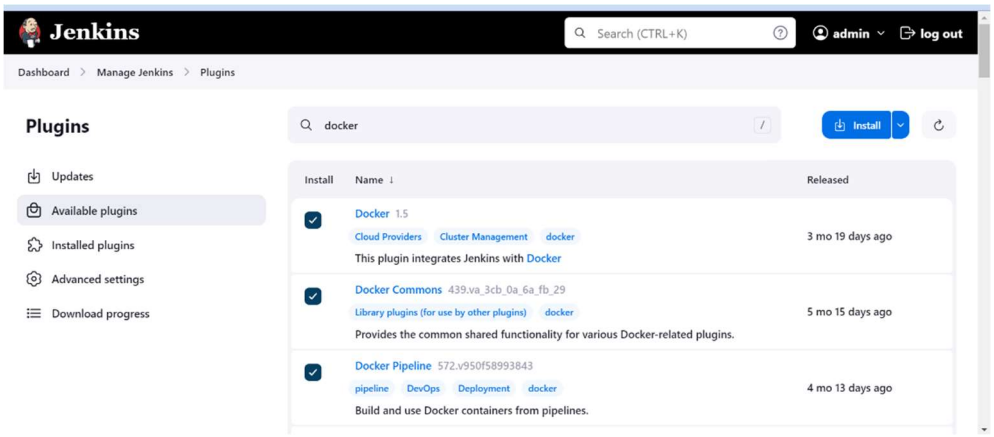
First, we configured the Plugin and next, we had to configure the Tool

Go to Dashboard → Manage Jenkins → Tools



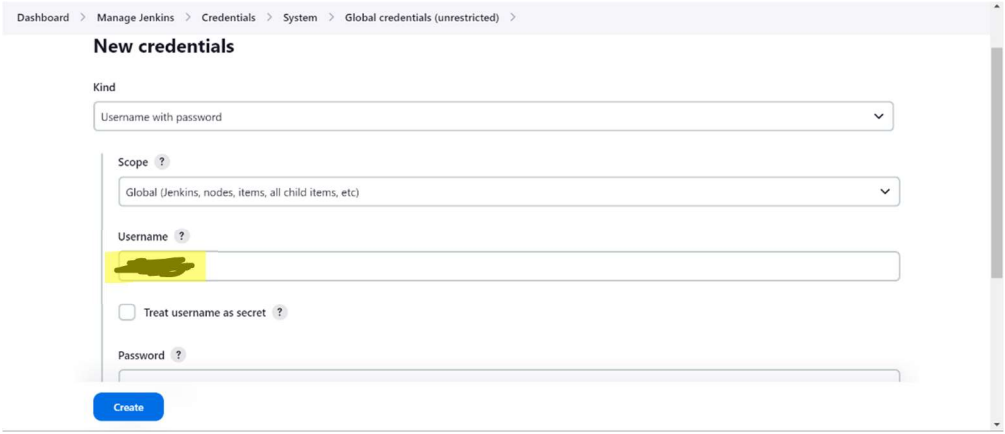
Now go configure → Pipeline and add OWASP and TRIVY stage to your pipeline and build.

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

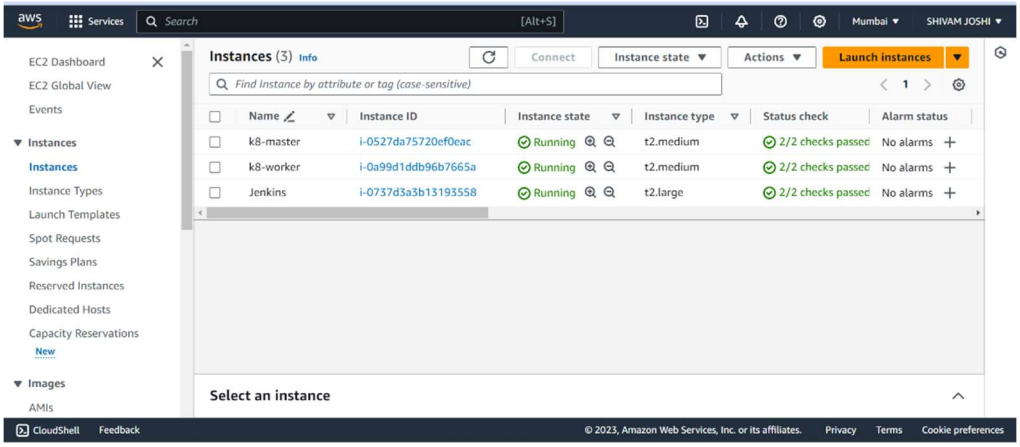
Add Docker Hub Username and Password under Global Credentials



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
No Changes	146ms	298ms	1s	23s	321ms	1min 39s	2min 42s	23s	1min 40s	2min 7s	1s	419ms failed
No Changes	219ms	410ms	1s	28s	348ms	2min 6s	2min 38s	23s	3min 2s failed	56ms failed	48ms failed	49ms failed
No Changes	155ms	415ms	1s	24s	497ms	2min 34s	8min 8s	8s aborted	129ms aborted	119ms aborted	103ms aborted	112ms aborted
No Changes	185ms	317ms	4s	28s	517ms (paused for 841ms)	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**

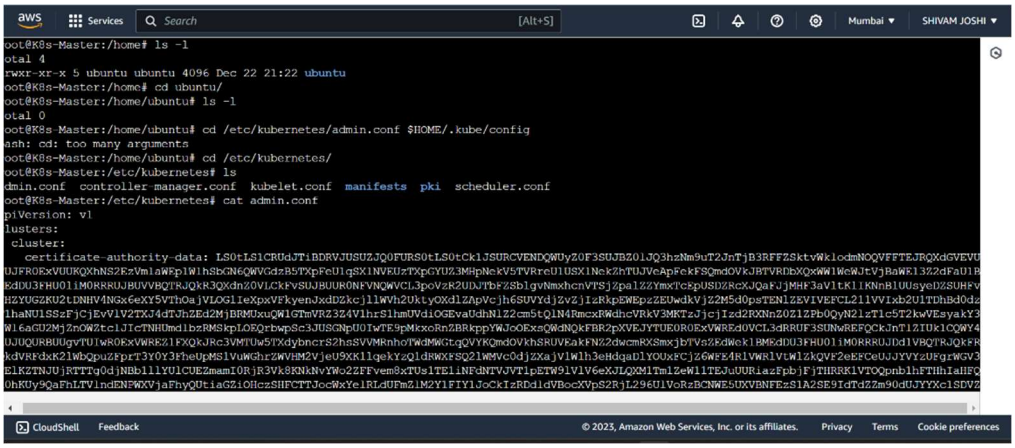
24. On Master node
    - **sudo kubeadm init --pod-network-cidr=10.244.0.0/16**
    - **mkdir -p $HOME/.kube**
    - **sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**
    - **sudo chown $(id -u):$(id -g) $HOME/.kube/config**

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

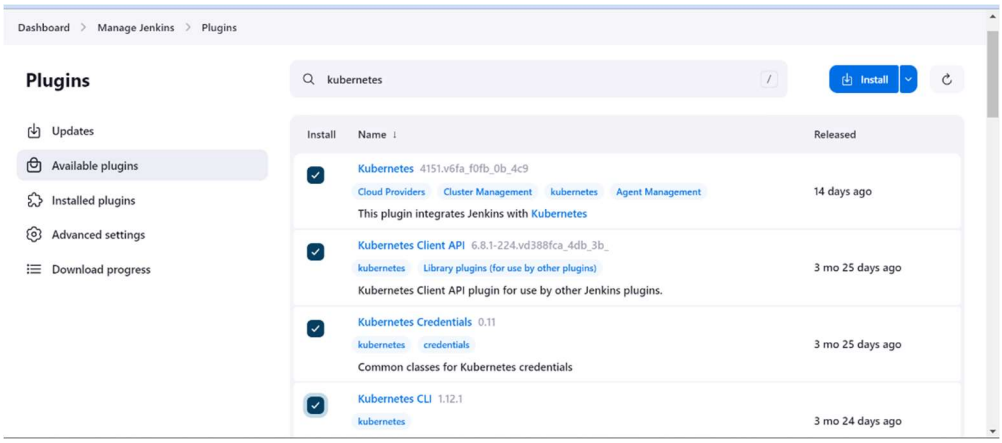
25. You will get the token and the command like below after executing the above command
    - **sudo kubeadm join <master-node-ip>:<master-node-port> --token <token> --discovery-token-ca-cert-hash <hash>**

    Copy above token and paste in worker node

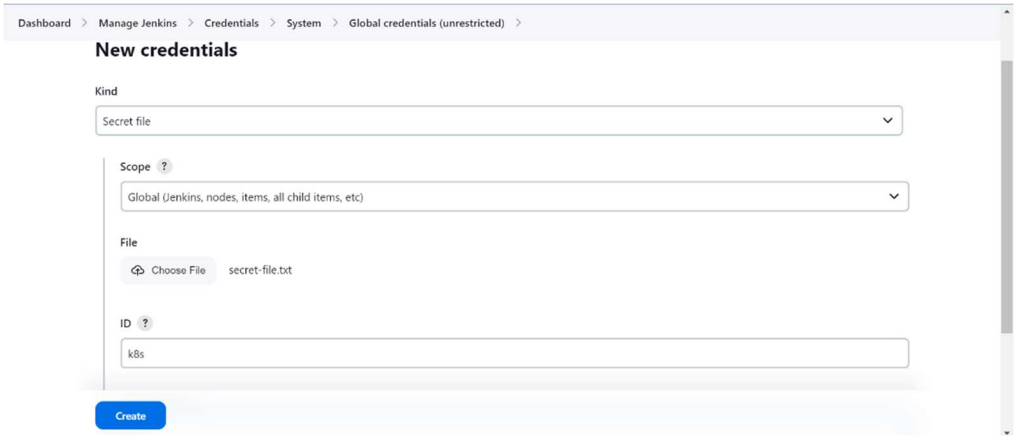
26. Copy the config file from K8 MASTER to the local laptop and save it with a name secret-file.txt and use this at the Kuberenetes credential section.



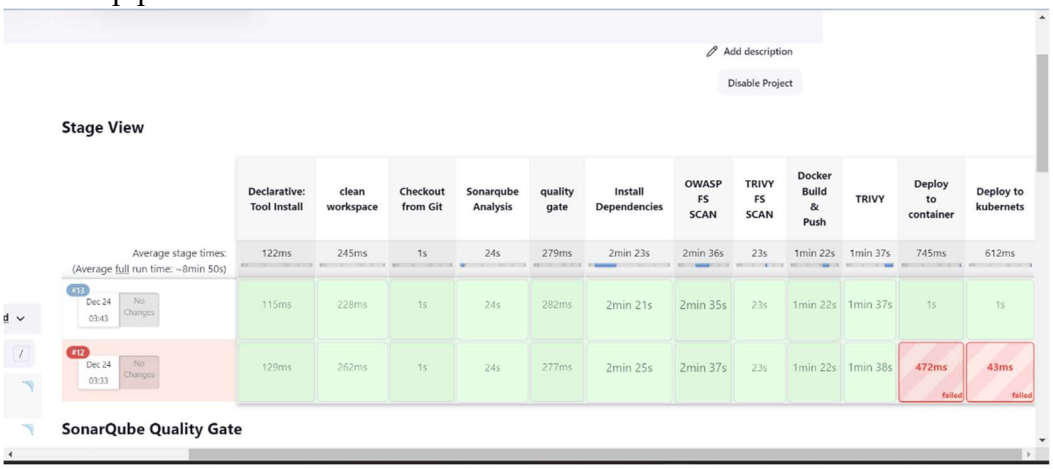
27. Install Kubernetes plugin



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



29. Run the pipeline



Hurrayy!!! 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**

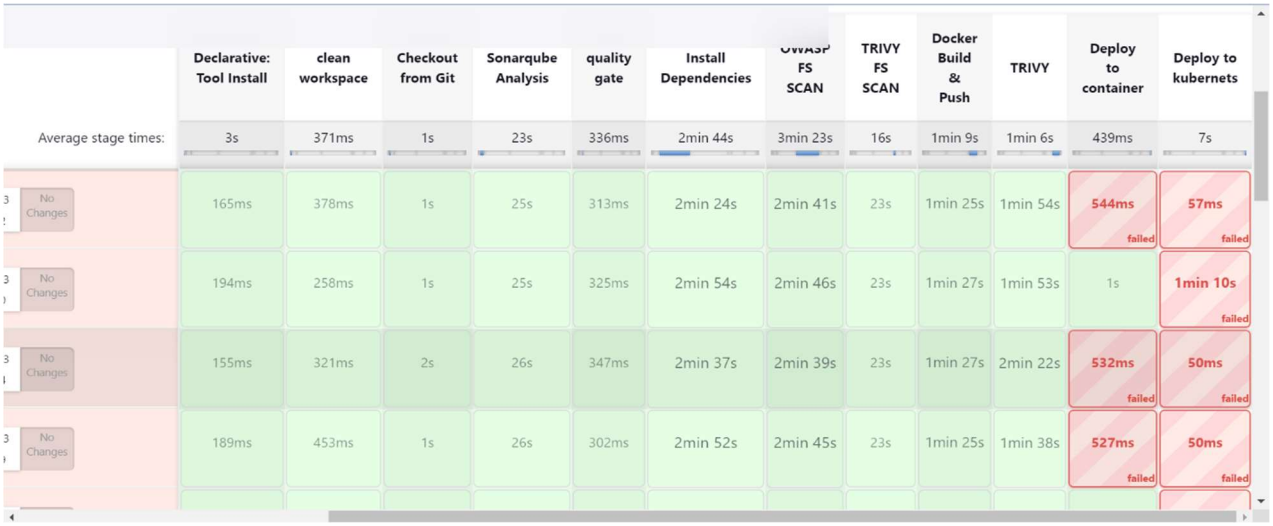
31. Access from a Web browser with
    - **<public-ip-of-slave:service port>**

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: 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:	3s	371ms	1s	23s	336ms	2min 44s	3min 23s	16s	1min 9s	1min 6s	439ms	7s
No Changes	165ms	378ms	1s	25s	313ms	2min 24s	2min 41s	23s	1min 25s	1min 54s	544ms failed	57ms failed
No Changes	194ms	258ms	1s	25s	325ms	2min 54s	2min 46s	23s	1min 27s	1min 53s	1s	1min 10s failed
No Changes	155ms	321ms	2s	26s	347ms	2min 37s	2min 39s	23s	1min 27s	2min 22s	532ms failed	50ms failed
No Changes	189ms	453ms	1s	26s	302ms	2min 52s	2min 45s	23s	1min 25s	1min 38s	527ms failed	50ms failed

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