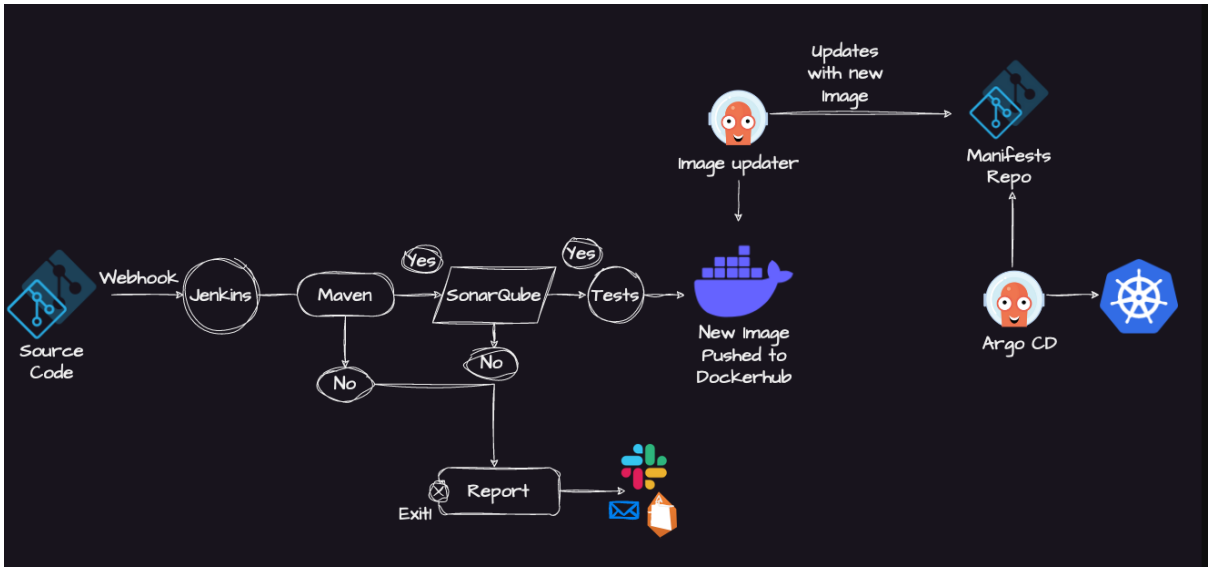
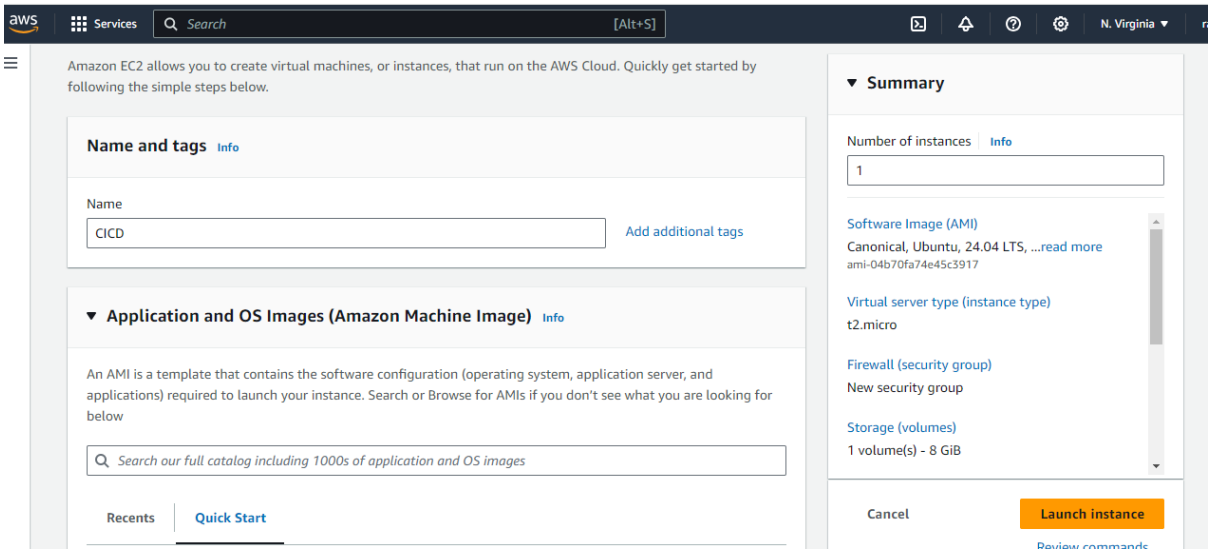
Jenkins Pipeline for Java based application using Maven, SonarQube, Argo CD, Helm and Kubernetes



Installation on EC2 Instance:

- Go to AWS Console
- Instances(running)
- Launch instances



Install Java:

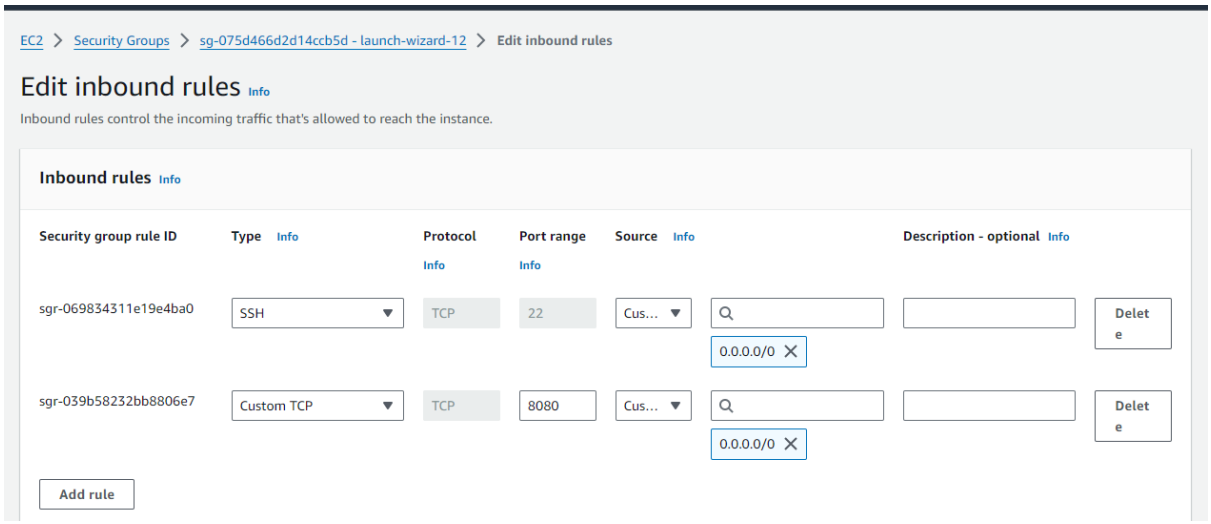
1. Update package lists: sudo apt update
2. Install OpenJDK 11: sudo apt install openjdk-11-jre
3. Verify Java installation: java -version

Install Jenkins:

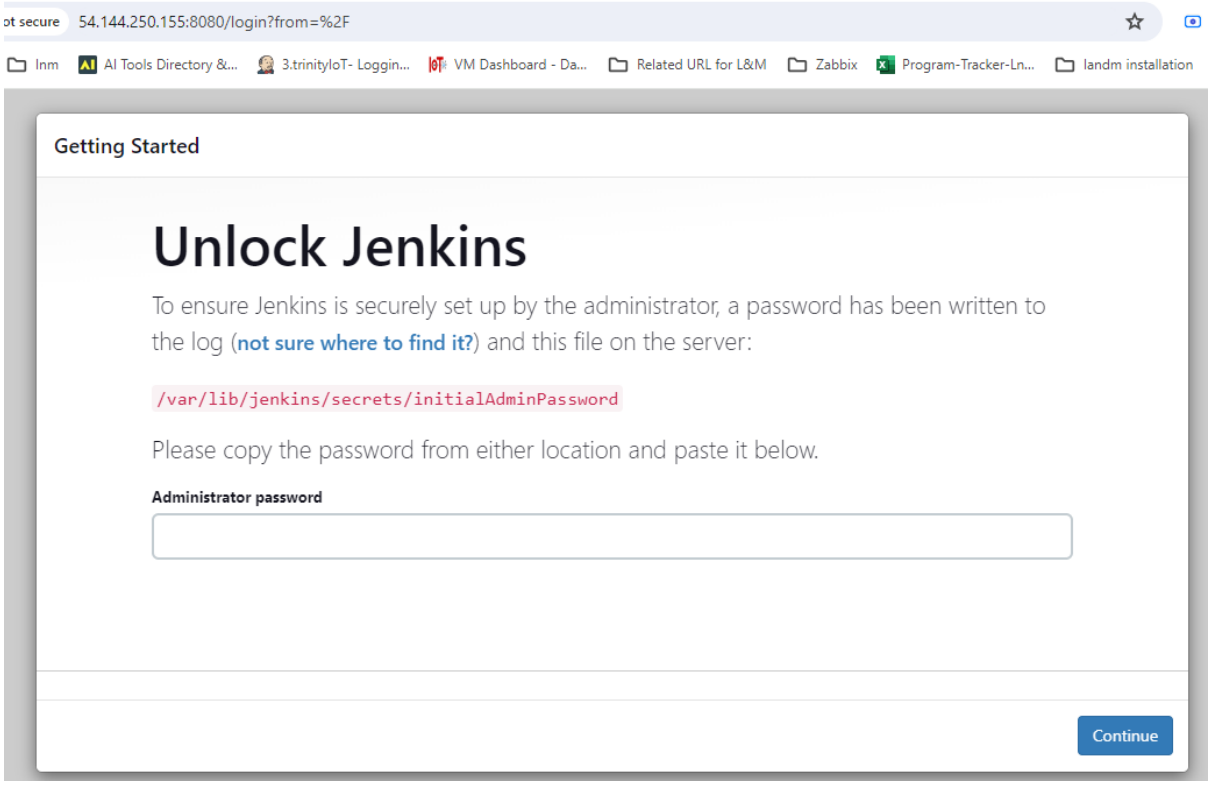
- Import Jenkins repository key: curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
- Add Jenkins repository: echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

- Update package lists: sudo apt-get update
- Install Jenkins: sudo apt-get install jenkins

Note: Ensure inbound traffic rules allow TCP 8080.

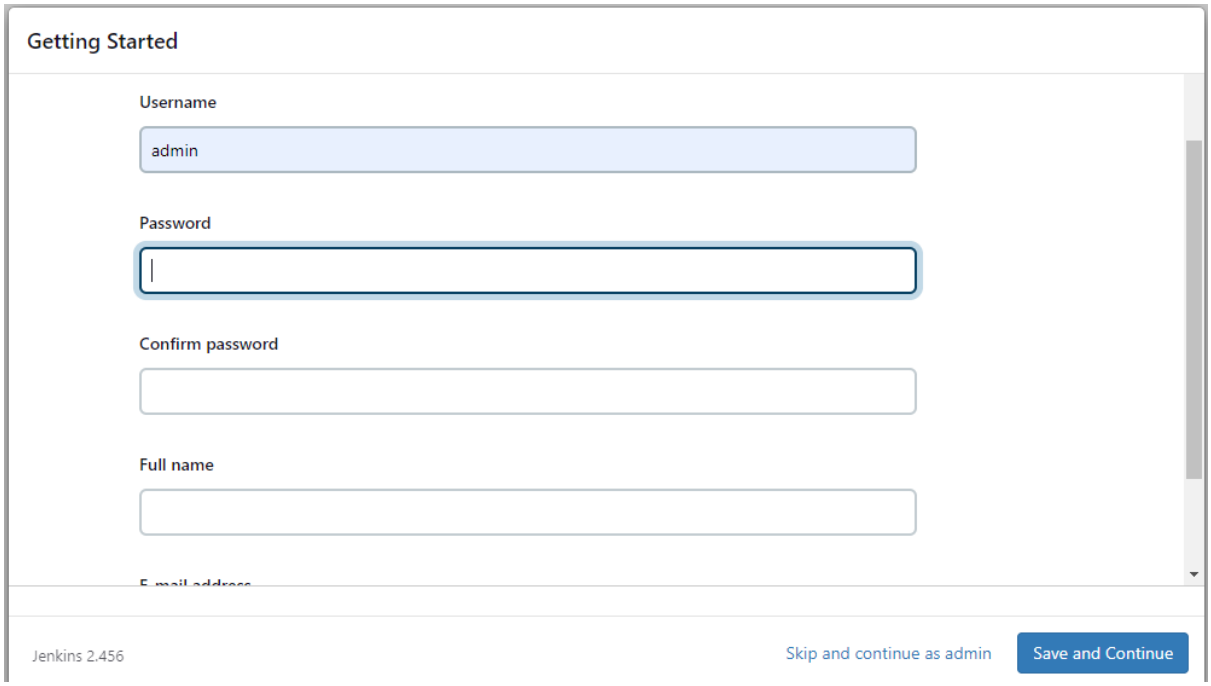


- Access Jenkins:
- Login using: http://<ec2-instance-public-ip>:8080

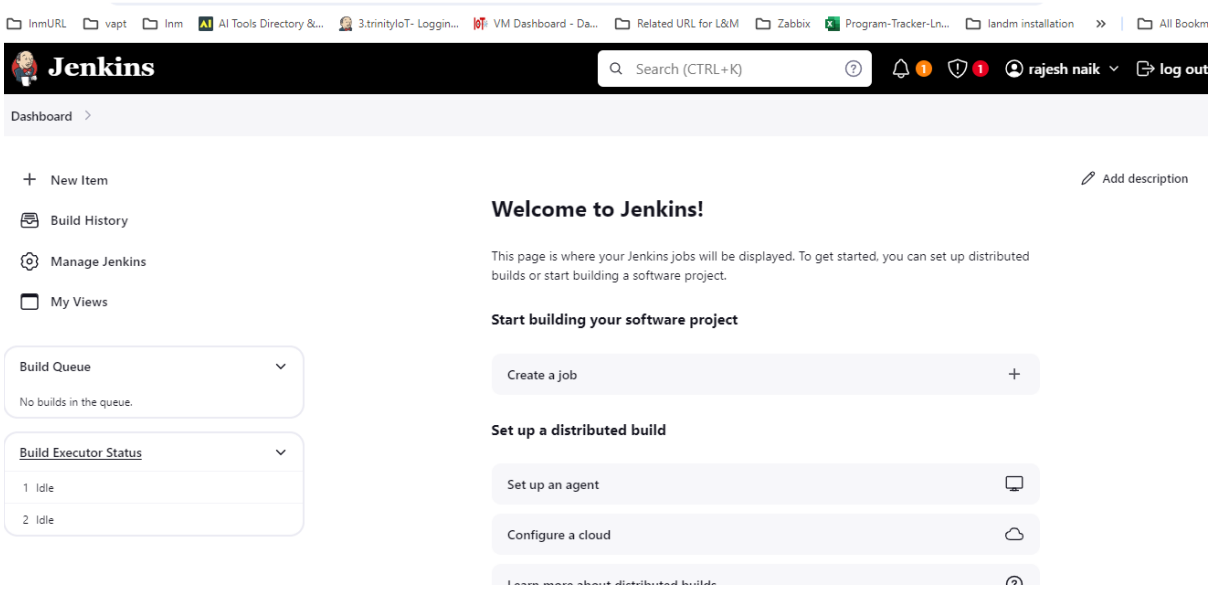


- Retrieve admin password: sudo cat /var/lib/jenkins/secrets/initialAdminPassword

```
root@ip-172-31-61-170:~# cat /var/lib/jenkins/secrets/initialAdminPassword
47e9a83a3b61465faadd2ac1948feb5a
root@ip-172-31-61-170:~#
```



---------->create admin user---------->



----------> **Jenkins Installation is Successful**.---------->

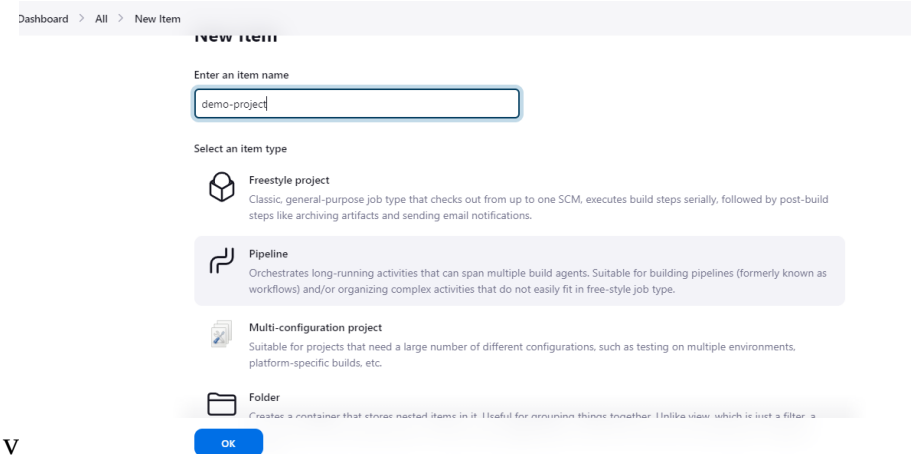
create pipeline

- Navigate to your Jenkins dashboard.
- Click on New Item.
- Enter a name for your pipeline and select Pipeline as the project type.
- Click OK to proceed.

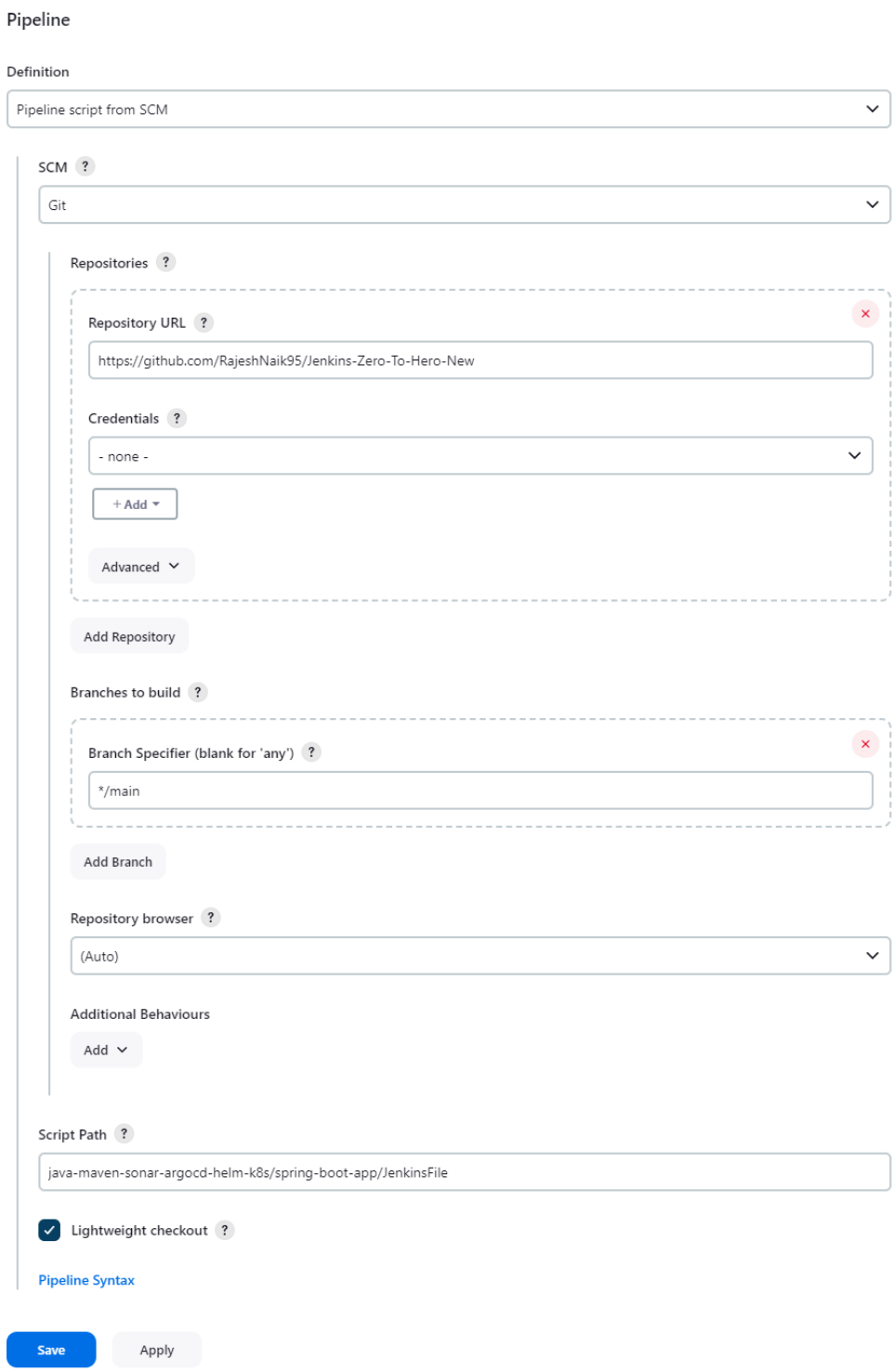
In the pipeline configuration page:

- Scroll down to the Pipeline section.
- In the Definition dropdown, select Pipeline script from SCM.
- Choose your preferred SCM (Git, GitHub, etc.).
- Enter the repository URL where your Jenkinsfile is stored.
- If needed, specify the branch or tag.
- Save the configuration.

Note: Ensure that your Jenkins server has the necessary permissions to access the repository.



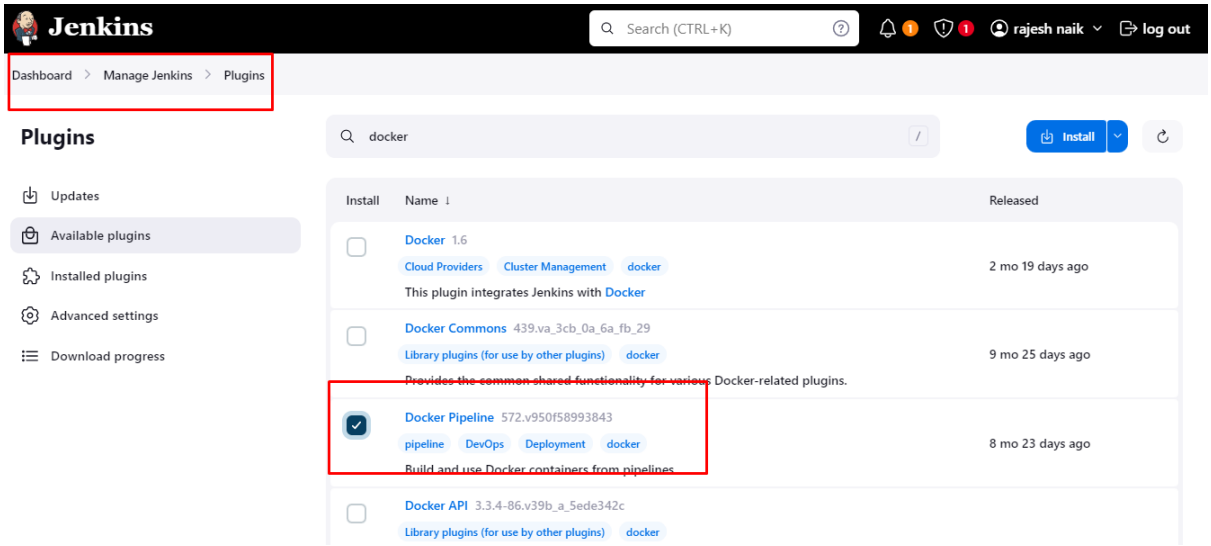
V



In this pipeline we are using docker as agent so for that need to install docker pluggin and docker in jenkins server

Install Docker Plugin:

- Go to the Jenkins dashboard.
- Click on Manage Jenkins in the sidebar.
- Select Manage Plugins.
- Go to the Available tab.
- Search for "Docker" in the filter box.
- Check the box next to Docker Plugin.
- Click Install without restart.



install docker in jenkins server

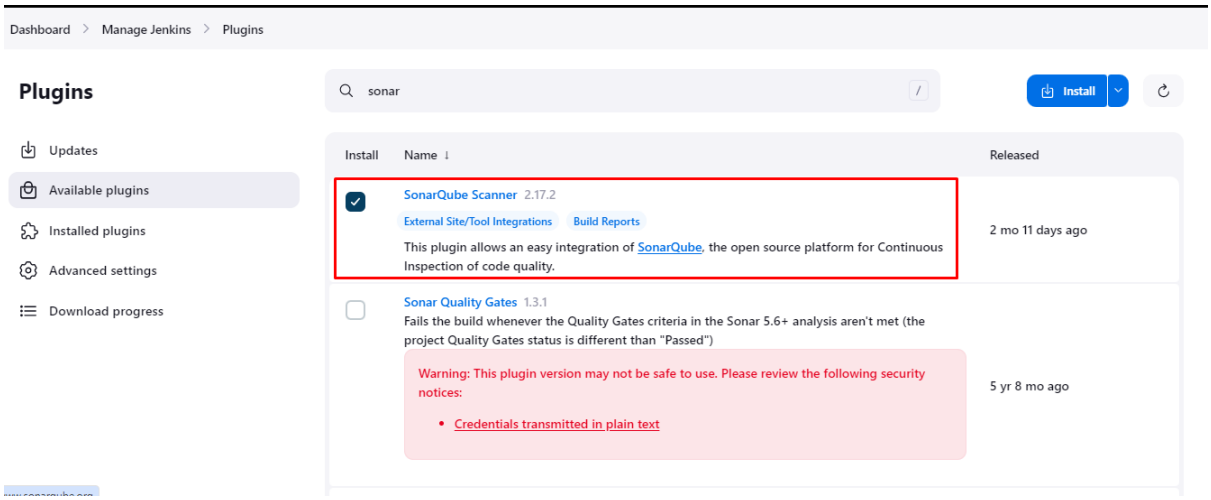
Install Docker:
sudo apt update
sudo apt install docker.io
Grant Permissions:
sudo su -
usermod -aG docker jenkins
usermod -aG docker ubuntu
systemctl restart docker

Once you are done with the above steps, it is better to restart Jenkins.

http://<ec2-instance-public-ip>:8080/restart

Install SonarQube:

- Install SonarQube Scanner Plugin: Jenkins Dashboard -> Manage Jenkins -> Plugins -> Install SonarQube Scanner.



Install and Configure SonarQube Server

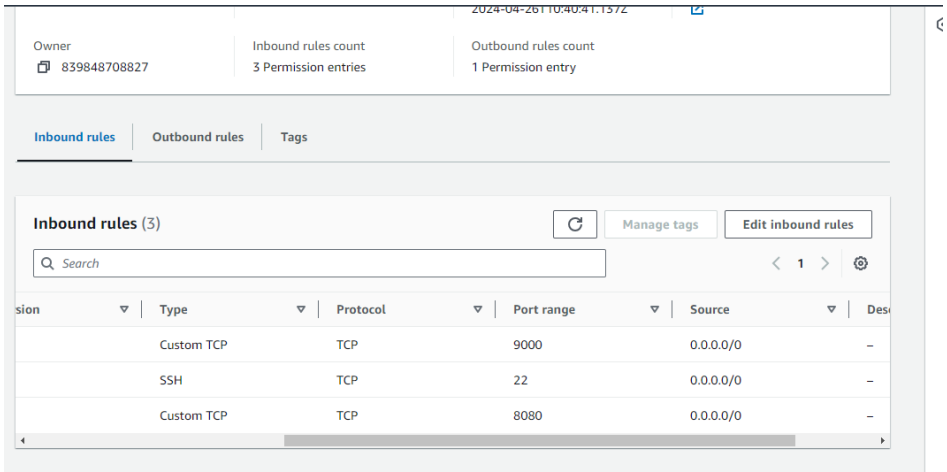
$apt install unzip

```
$adduser sonarqube
su - sonarqube
wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.4.0.54424.zip
unzip *
chmod -R 755 /home/sonarqube/sonarqube-9.4.0.54424
chown -R sonarqube:sonarqube /home/sonarqube/sonarqube-9.4.0.54424
cd sonarqube-9.4.0.54424/bin/linux-x86-64/
./sonar.sh start
```

sonar kube installed it will access in 9000 port

https://ec2 instance ip :9000

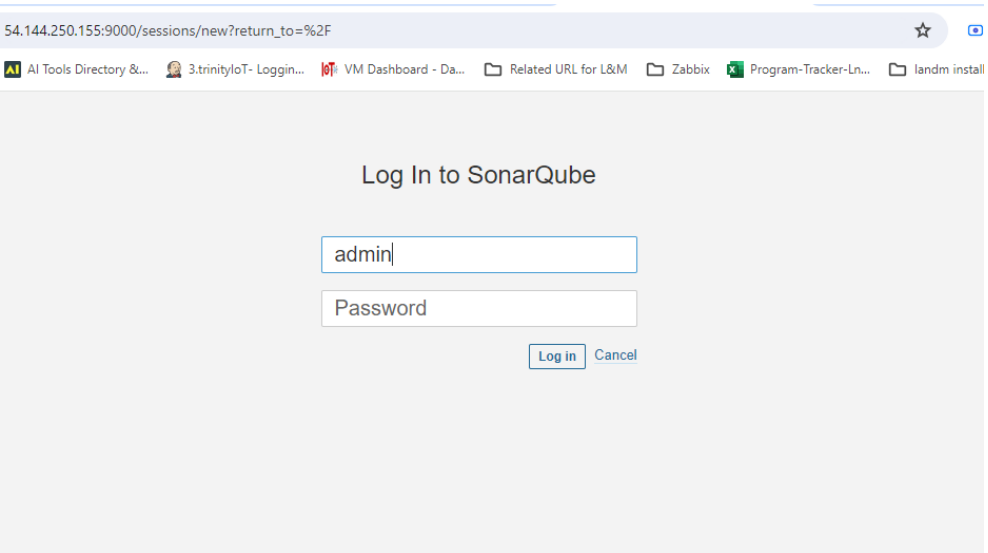
- Add 9000 port in EC2 inbond rule

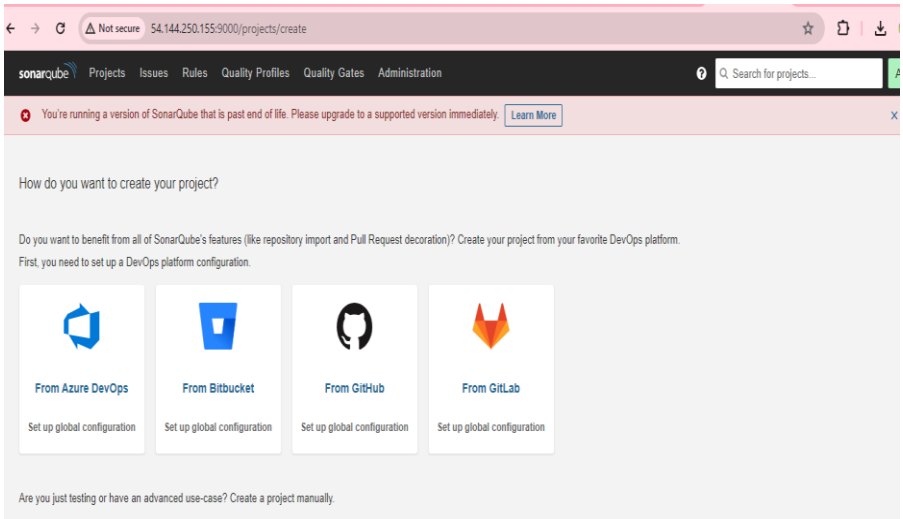


Connect Jenkins with SonarQube:

Now we can acess sonarqkube url

Default User name and password is admin --- admin

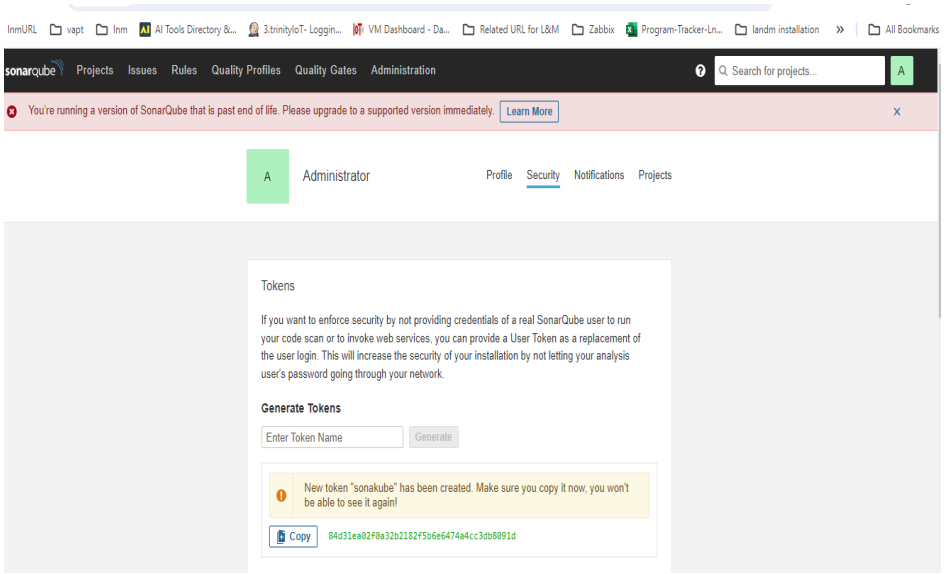




Connect SonarQube with Jenkins:

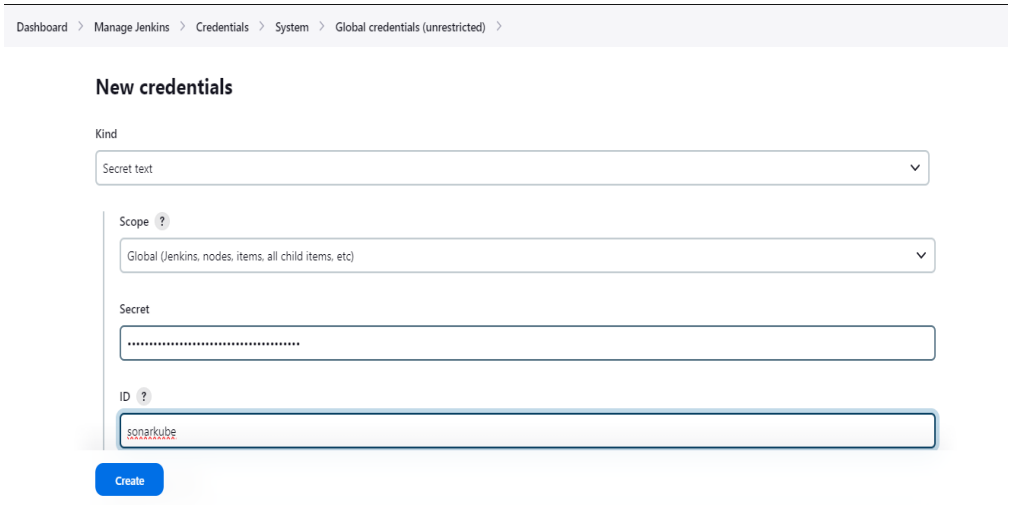
1. Generate API Token in SonarQube:

- Log in to your SonarQube instance.
- Navigate to your profile settings.
- Click on Security or My Account.
- Select Security.
- Scroll down to the Generate Tokens section.
- Provide a name for your token and click Generate.
- Copy the generated token.



Add API Token to Jenkins Credentials:

- Go to your Jenkins dashboard.
- Click on Manage Jenkins in the sidebar.
- Select Manage Credentials.
- Click on (global) or the appropriate domain.
- Click on Add Credentials.
- Choose Secret text as the kind.
- Paste the copied API token in the Secret field.
- Optionally, provide an ID and description.
- Click OK to save the credentials.



Continuous Integration (CI) Setup Complete.

Continuous Deployment (CD) Setup:

Install MiniKube on a new EC2 instance:

Installing Docker:

Update package lists:
sudo apt-get update

Install Docker:
sudo apt-get install docker.io -y
sudo usermod -aG docker $USER && newgrp docker

Update system packages and install MiniKube dependencies:
sudo apt update && sudo apt upgrade
sudo apt install -y curl wget apt-transport-https

Download the latest MiniKube release:
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

Install MiniKube:
sudo install minikube-linux-amd64 /usr/local/bin/minikube

Verify the installation by checking the MiniKube version:
minikube version

You should see an output similar to:
minikube version: v1.32.0
commit: 8220a6eb95f0a4d75f7f2d7b14cef975f050512d

Download kubectl:
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl

Make kubectl executable:
chmod +x kubectl

Move kubectl to /usr/local/bin to make it available system-wide:
sudo mv kubectl /usr/local/bin/

Check the kubectl version:
kubectl version -o yaml

Start MiniKube with Docker driver:
minikube start --driver=docker --force

Check the status of MiniKube:
minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

Setup Argo CD:

- Install Operator Lifecycle Manager (OLM).

curl -sL https://github.com/operator-framework/operator-lifecycle-manager/releases/download/v0.27.0/install.sh | bash -s v0.27.0

- Install Argo CD Operator.

$ kubectl create -f https://operatorhub.io/install/argocd-operator.yaml

$ kubectl get csv -n operators

- Deploy Argo CD controller.
- Access Argo CD UI using generated credentials.

Deploy Argo CD Controller:
Create a YAML file named argocd-basic.yml with the following content:
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
name: example-argocd
labels:
example: basic
spec: {}
Apply the YAML file to create the Argo CD instance:
$kubectl apply -f argocd-basic.yml
Check the status of the Argo CD pods:
$kubectl get pod

```
root@ip-172-31-50-234:/# kubectl get pod
NAME                                          READY   STATUS    RESTARTS   AGE
example-argocd-application-controller-0       1/1     Running   0          31s
example-argocd-redis-c744b8498-pffdp          1/1     Running   0          31s
example-argocd-repo-server-785457f7f8-66dgp   1/1     Running   0          31s
example-argocd-server-655c9579d4-cstwc        1/1     Running   0          31s
root@ip-172-31-50-234:/#
```

Check the services to get the URL for accessing Argo CD:
$kubectl get svc

```
root@ip-172-31-50-234:/# kubectl get svc
NAME                            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
example-argocd-metrics          ClusterIP   10.107.45.7      <none>        8082/TCP            100s
example-argocd-redis            ClusterIP   10.102.88.175    <none>        6379/TCP            100s
example-argocd-repo-server      ClusterIP   10.100.191.165   <none>        8081/TCP,8084/TCP   100s
example-argocd-server           ClusterIP   10.99.176.82     <none>        80/TCP,443/TCP      99s
example-argocd-server-metrics   ClusterIP   10.108.11.113    <none>        8083/TCP            100s
kubernetes                      ClusterIP   10.96.0.1        <none>        443/TCP             3h1m
root@ip-172-31-50-234:/#
```

Edit the service to change the cluster IP to node port for browser access:
$kubectl edit svc example-argocd-server

```
    app.kubernetes.io/managed-by: example-argocd
    app.kubernetes.io/name: example-argocd-server
    app.kubernetes.io/part-of: argocd
  name: example-argocd-server
  namespace: default
  ownerReferences:
  - apiVersion: argoproj.io/v1beta1
    blockOwnerDeletion: true
    controller: true
    kind: ArgoCD
    name: example-argocd
    uid: 8c5aa60f-c951-483d-8127-f580d4cda7be
  resourceVersion: "15166"
  uid: 41f038aa-2da5-4ea0-9045-1237d794bfc1
spec:
  clusterIP: 10.99.176.82
  clusterIPs:
  - 10.99.176.82
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
  - name: https
    port: 443
    protocol: TCP
    targetPort: 8080
  selector:
    app.kubernetes.io/name: example-argocd-server
  sessionAffinity: None
  type: ClusterIP          NodePort
status:
  loadBalancer: {}
-- INSERT --
```

Access Argo CD UI:

Use port-forwarding to access the Argo CD UI:

$kubectl port-forward svc/example-argocd-server -n default --address 0.0.0.0 8080:443

```
root@ip-172-31-50-234:/# kubectl port-forward svc/example-argocd-server -n default --address 0.0.0.0 8080:443
Forwarding from 0.0.0.0:8080 -> 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
```

Use the following credentials to login:

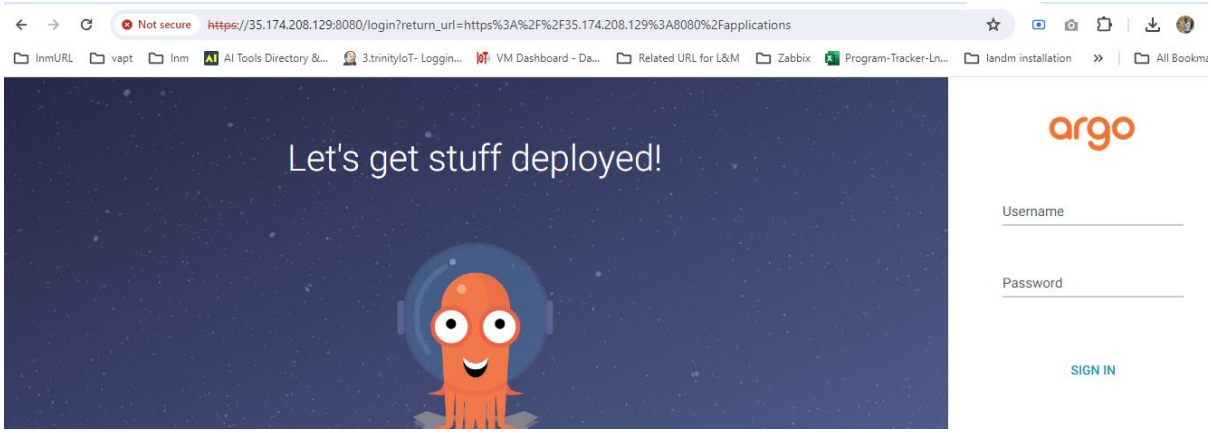
- Username: admin
- Password: (Decrypt the encrypted password using the command below)

echo dlAzcDBaeFJmWUw4RGFuaEd1U0pGYjlYano1Z1dIY3M= | base64 -d

```
root@ip-172-31-50-234:/# kubectl get secret
NAME                                  TYPE                  DATA   AGE
argocd-secret                         Opaque                5      50m
example-argocd-ca                     kubernetes.io/tls     3      50m
example-argocd-cluster                Opaque                1      50m
example-argocd-default-cluster-config Opaque                4      50m
example-argocd-tls                    kubernetes.io/tls     2      50m
```

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving th
# reopened with the relevant failures.
#
apiVersion: v1
data:
  admin.password: dlAzcDBaeFJmWUw4RGFuaEd1U0pGYjlYano1Z1dIY3M=
kind: Secret
metadata:
  creationTimestamp: "2024-05-01T16:26:59Z"
  labels:
    app.kubernetes.io/managed-by: example-argocd
    app.kubernetes.io/name: example-argocd-cluster
    app.kubernetes.io/part-of: argocd
  name: example-argocd-cluster
  namespace: default
  ownerReferences:
  - apiVersion: argoproj.io/v1beta1
    blockOwnerDeletion: true
    controller: true
    kind: ArgoCD
    name: example-argocd
    uid: 8c5aa60f-c951-483d-8127-f580d4cda7be
  resourceVersion: "15140"
  uid: 0daa2e54-7fcf-4560-bf92-73aed379e49b
type: Opaque
~
```

```
root@ip-172-31-50-234:/# echo dlAzcDBaeFJmWUw4RGFuaEd1U0pGYjlYano1Z1dIY3M= | base64 -d
vP3p0ZxRfYL8DanhGuSJFb9Xjz5gWHcsroot@ip-172-31-50-234:/#
```



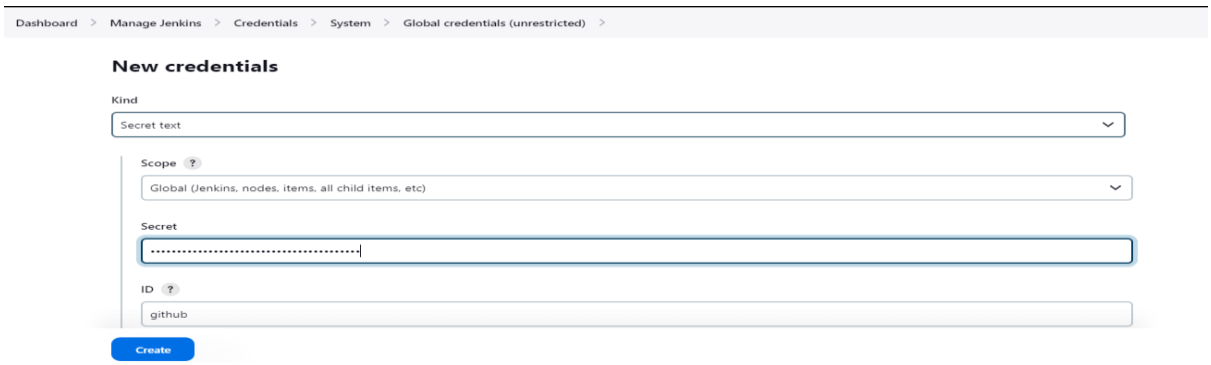
Add Docker Hub and GitHub Credentials in Jenkins:

Docker Hub Credentials:

- Navigate to your Jenkins dashboard.
- Click on Manage Jenkins in the sidebar.
- Select Manage Credentials.
- Click on (global) or the appropriate domain.
- Click on Add Credentials.
- Choose Username with password as the kind.
- Enter your Docker Hub username and password.
- Optionally, provide an ID and description.
- Click OK to save the credentials.

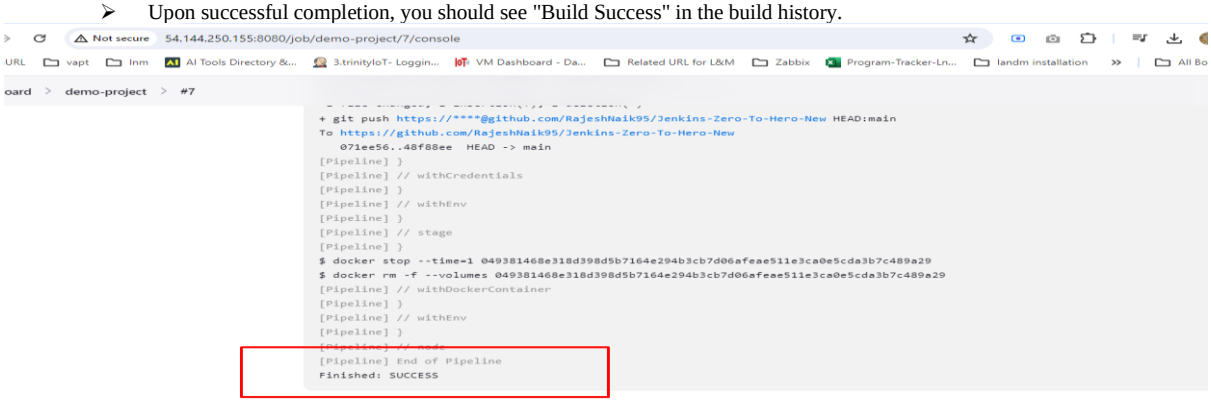
GitHub Credentials:

- Follow the same steps as above but choose Secret text as the kind.
- Enter your GitHub personal access token or password.
- Optionally, provide an ID and description.
- Click OK to save the credentials.



Build Pipeline:

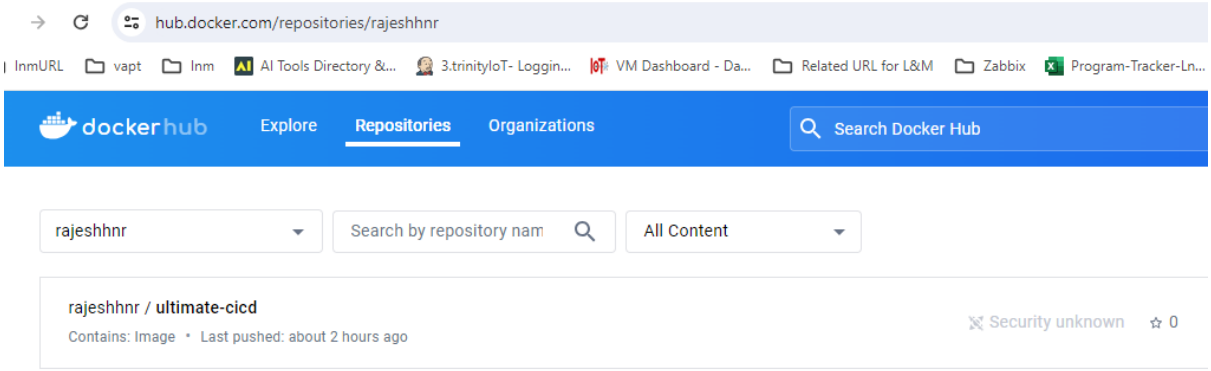
- Ensure all configurations, including linking your Jenkinsfile, setting up Docker and Maven plugins, and defining stages, are completed.
- Once the pipeline is set up, go to the Jenkins dashboard.
- Click on your pipeline project.
- Click on Build Now to trigger a build.
- Jenkins will start the build process. You can monitor the progress in the Jenkins interface.
- Upon successful completion, you should see "Build Success" in the build history.

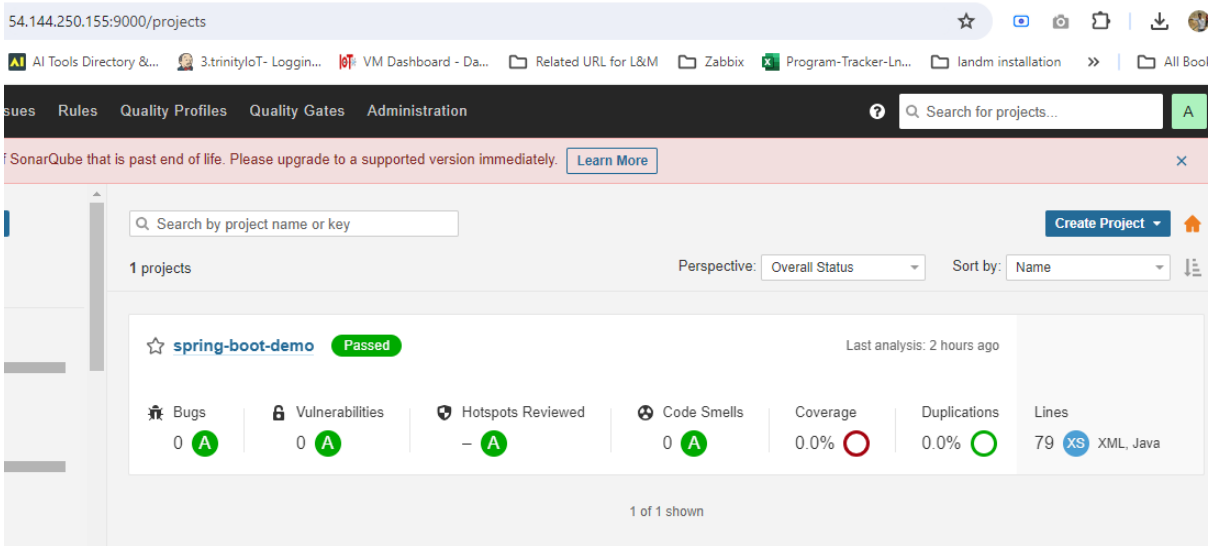


Build Success.....

Post-Build Actions:

- ✓ Verify that Docker images have been updated in your Docker Hub repository.
- ✓ Check the build artifacts and logs for any errors or warnings.
- ✓ If everything looks good, proceed with further deployment steps as per your pipeline configuration.

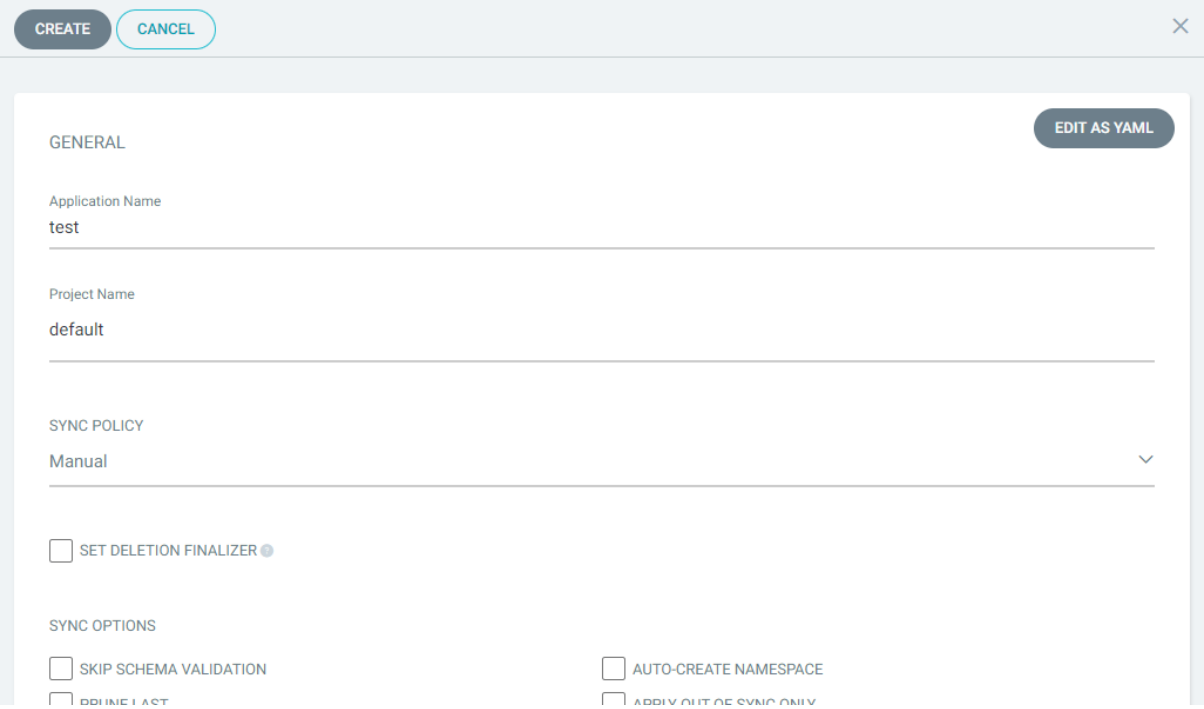


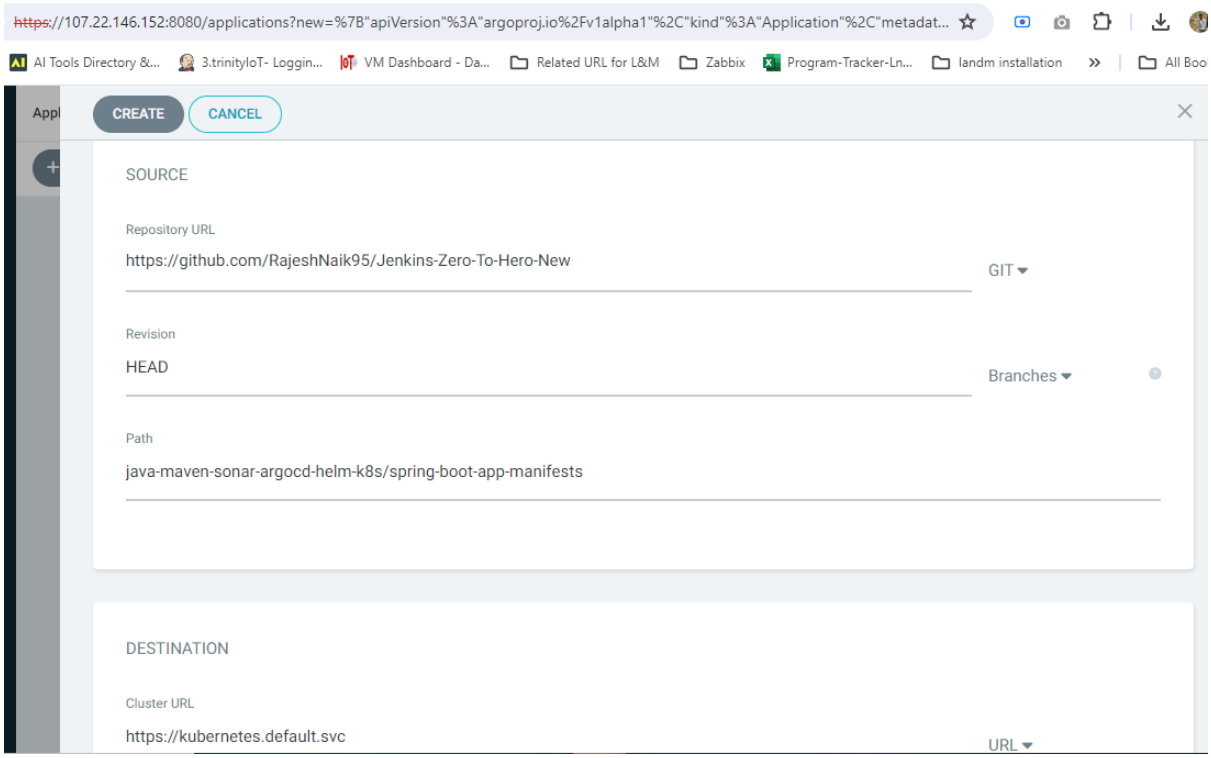


Deploy Application to Minikube Cluster using Argo CD:

Configure Argo CD Application:

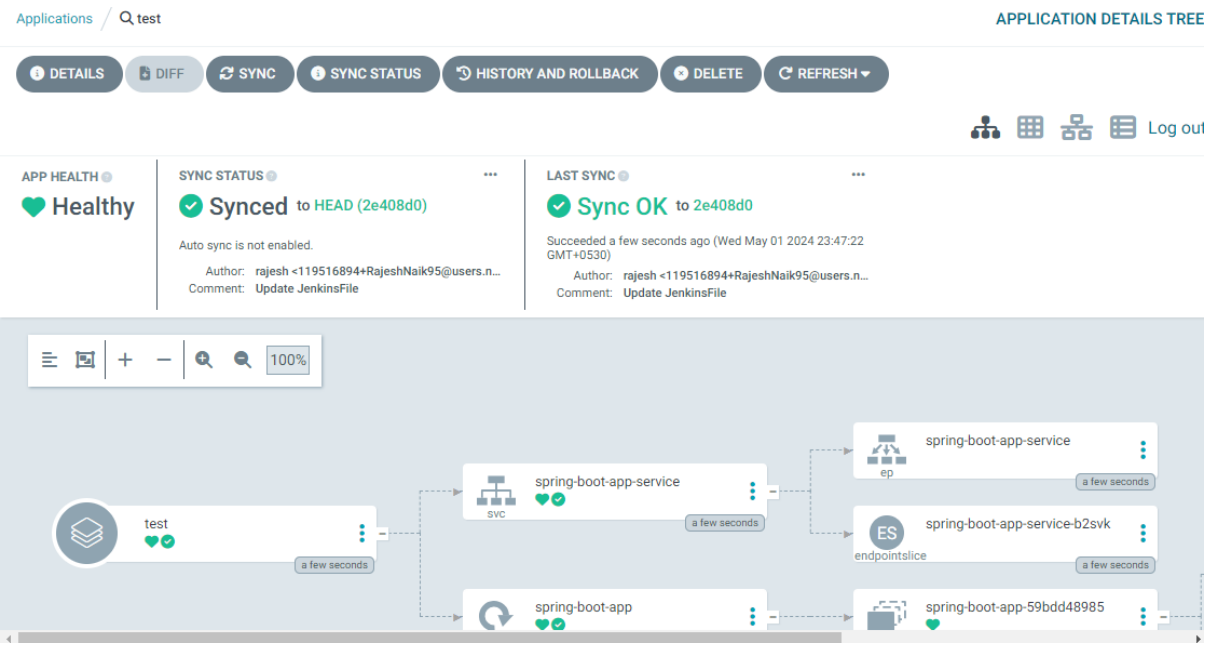
- Open the Argo CD UI in your browser.
- Log in using your credentials.
- Navigate to the Applications tab.
- Click on Create Application.
- Fill in the application details, including the Git repository URL, target revision, and sync policy.
- Click on Create to add the application.





Trigger Synchronization:

- Once the application is created, Argo CD will automatically synchronize with the Git repository.
- You can also manually trigger synchronization by clicking on the Sync button next to the application.



Monitor Deployment:

- Monitor the deployment progress in the Argo CD UI.
- Wait for the application to be deployed successfully.

```
QKvfTPakH5rVN8uwecZBFl7UShY61tJWroot@ip-172-31-49-203:/# kubectl get deploy
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
example-argocd-redis        1/1     1            1           11m
example-argocd-repo-server  1/1     1            1           11m
example-argocd-server       1/1     1            1           11m
spring-boot-app             2/2     2            2           102s
root@ip-172-31-49-203:/# kubectl get pod
NAME                                        READY   STATUS    RESTARTS   AGE
example-argocd-application-controller-0     1/1     Running   0          12m
example-argocd-redis-c744b8498-rgwk7        1/1     Running   0          12m
example-argocd-repo-server-785457f7f8-rxcjf 1/1     Running   0          12m
example-argocd-server-655c9579d4-rk5q6      1/1     Running   0          12m
spring-boot-app-59bdd48985-dn25k            1/1     Running   0          2m16s
spring-boot-app-59bdd48985-q6llc            1/1     Running   0          2m16s
root@ip-172-31-49-203:/#
```

```
root@ip-172-31-49-203:/# curl 192.168.49.2:32619
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"/>
    <link href="/css/main.css?" rel="stylesheet">
    <title>CI/CD Master Class By Abhishek Veeramalla</title>
</head>
<body>

<nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
    <a class="navbar-brand" href="#">Ultimate CI/CD Pipeline using Java</div></a>
</nav>

<main role="main" class="container">
    <div class="starter-template">
        <h1>I have successfuly built a sprint boot application using Maven</h1>
        <p>This application is deployed on to Kubernetes using Argo CD</p>
    </div>
</main>

<script src="/js/main.js"></script>
</body>

</html>
```

Access Application:

- Once the deployment is complete, you can access your application.
- To access a service running in the Minikube cluster, you can use port-forwardin

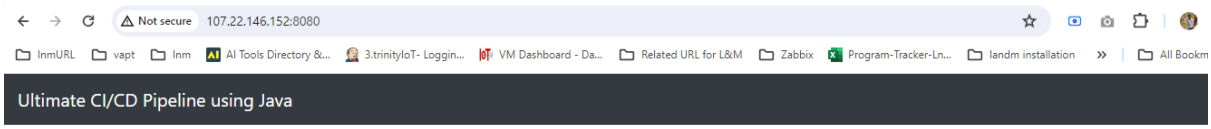
$kubectl get svc

```
root@ip-172-31-49-203:/# kubectl get svc
NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                    AGE
example-argocd-metrics        ClusterIP   10.97.201.46     <none>        8082/TCP                   64m
example-argocd-redis          ClusterIP   10.106.162.0     <none>        6379/TCP                   64m
example-argocd-repo-server    ClusterIP   10.104.90.191    <none>        8081/TCP,8084/TCP          64m
example-argocd-server         NodePort    10.100.32.92     <none>        80:30163/TCP,443:32469/TCP 64m
example-argocd-server-metrics ClusterIP   10.104.127.37    <none>        8083/TCP                   64m
kubernetes                    ClusterIP   10.96.0.1        <none>        443/TCP                    68m
spring-boot-app-service       NodePort    10.104.146.122   <none>        80:32619/TCP               54m
root@ip-172-31-49-203:/# kubectl port-forward --address 0.0.0.0 svc/spring-boot-app-service 8080:80 &
[1] 41639
```

kubectl port-forward --address 0.0.0.0 svc/<service-name> 8080:80
Replace <service-name> with the name of your service.

- Open your browser and navigate to http://localhost:8080 to access your application.

Now we can able to access in browser.....



I have successfuly built a sprint boot application using Maven

This application is deployed on to Kubernetes using Argo CD

THANK You.....