**Building an End-to-End CI/CD Pipeline with AWS, Jenkins, Docker, SonarQube, ArgoCD, and Kubernetes**

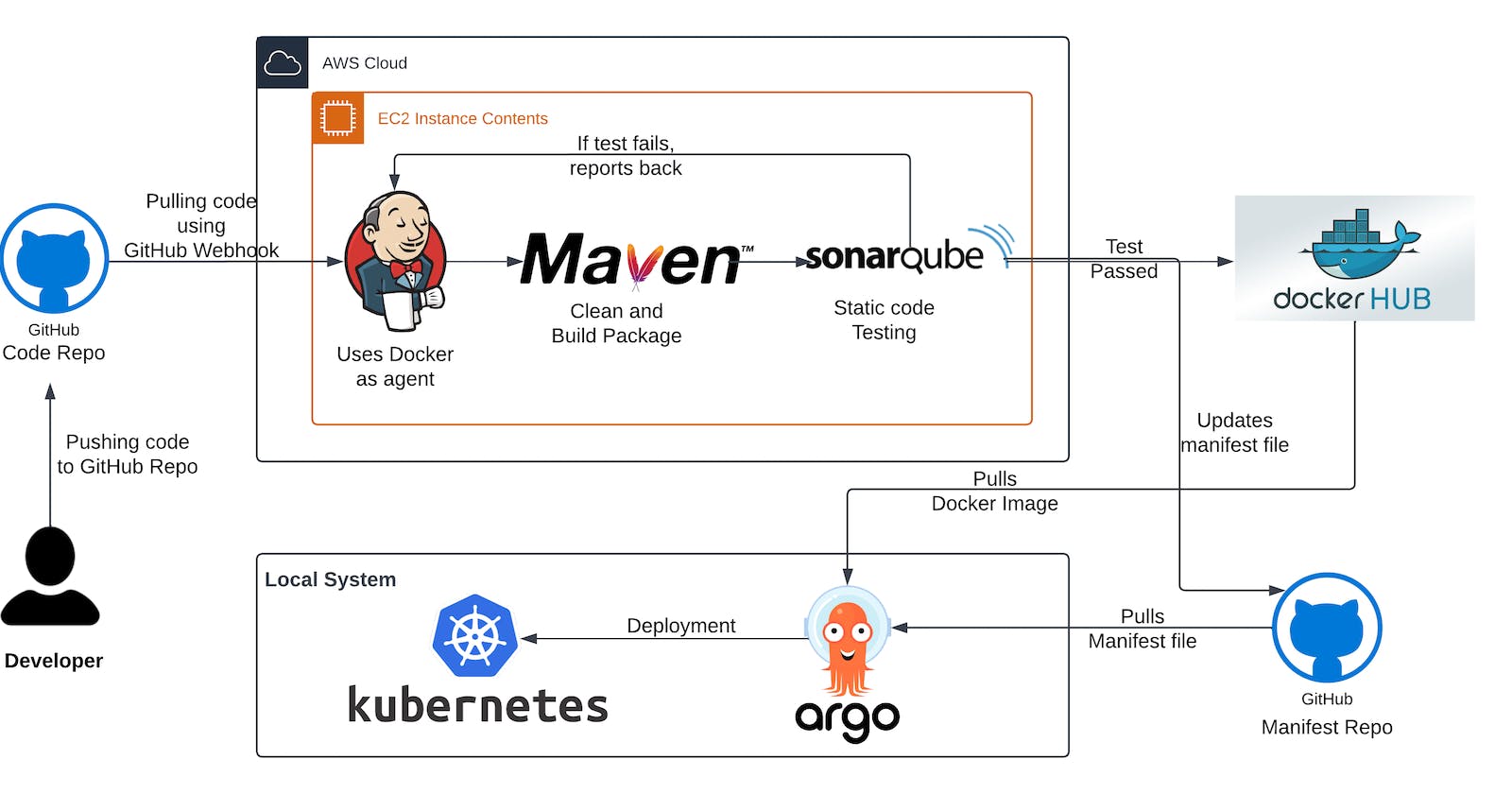


TABLE OF CONTENTS

* [1. Create an EC2 instance in AWS:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-1-create-an-ec2-instance-in-aws)
  + [Allocate the elastic IP address and associate it with the newly created EC2 instance.](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-allocate-the-elastic-ip-address-and-associate-it-with-the-newly-created-ec2-instance)
* [2. Install and configure Jenkins:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-2-install-and-configure-jenkins)
* [3. Fork the GitHub repository and create a webhook:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-3-fork-the-github-repository-and-create-a-webhook)
* [4. Build a new item in Jenkins:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-4-build-a-new-item-in-jenkins)
* [5. Installing plugins:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-5-installing-plugins)
* [6. Install SonarQube in Ubuntu and configure it in Jenkins:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-6-install-sonarqube-in-ubuntu-and-configure-it-in-jenkins)
* [7. Adding Credentials in Jenkins](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-7-adding-credentials-in-jenkins)
  + [Connecting Jenkins and SonarQube:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-connecting-jenkins-and-sonarqube)
  + [Adding credentials for DockerHub in Jenkins:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-adding-credentials-for-dockerhub-in-jenkins)
  + [Adding credentials for github in Jenkins:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-adding-credentials-for-github-in-jenkins)
* [8. Install Docker and configure it in Jenkins:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-8-install-docker-and-configure-it-in-jenkins)
* [9. Writing Jenkinsfile and manifest file:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-9-writing-jenkinsfile-and-manifest-file)
* [10. Build the pipeline:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-10-build-the-pipeline)
* [Now let’s perform continuous deployment using ArgoCD in Kubernetes in the local system](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-now-lets-perform-continuous-deployment-using-argocd-in-kubernetes-in-the-local-system)
* [1. Install Dependencies:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-1-install-dependencies)
* [2. Install kubectl and minikube and start minikube](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-2-install-kubectl-and-minikube-and-start-minikube)
* [3. Installing and configuring Argo CD](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-3-installing-and-configuring-argo-cd)
* [4. Setting up Argo CD for deployment in K8s](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-4-setting-up-argo-cd-for-deployment-in-k8s)
* [5. Testing the deployment from the browser:](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-5-testing-the-deployment-from-the-browser)
* [Clean up](https://rupaks.hashnode.dev/building-an-end-to-end-cicd-pipeline-with-aws-jenkins-docker-sonarqube-argocd-and-kubernetes#heading-clean-up)

Show less

This tutorial guides you through setting up a comprehensive CI/CD pipeline using AWS, Jenkins, Docker, SonarQube, ArgoCD and Kubernetes. It covers creating an EC2 instance, installing/configuring Jenkins and SonarQube, adding credentials, installing Docker, building the pipeline, deploying with ArgoCD in Kubernetes, and performing cleanup. By following this tutorial, you'll gain hands-on experience in automating the build, test, and deployment processes of your applications.

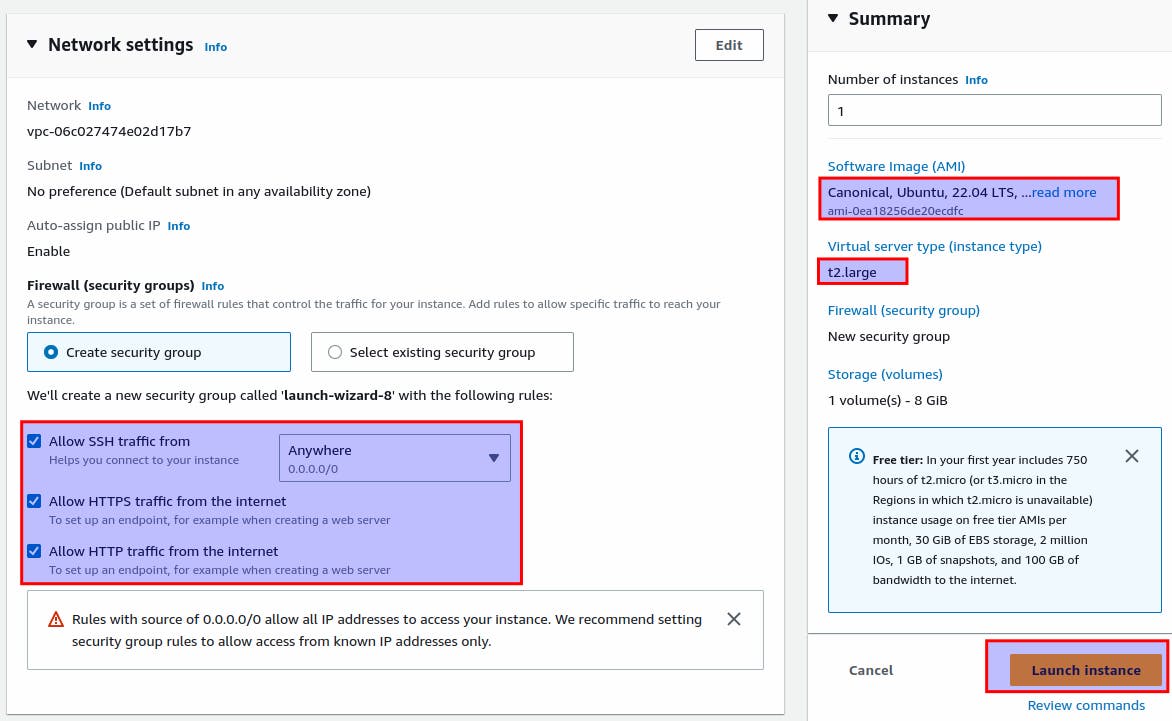
**Prerequisites:**

* An AWS account with appropriate permissions to create EC2 instances and open ports.
* Basic familiarity with Linux commands and AWS EC2.
* Access to the GitHub repository and appropriate permissions to create webhooks.
* A DockerHub account for pushing Docker images.

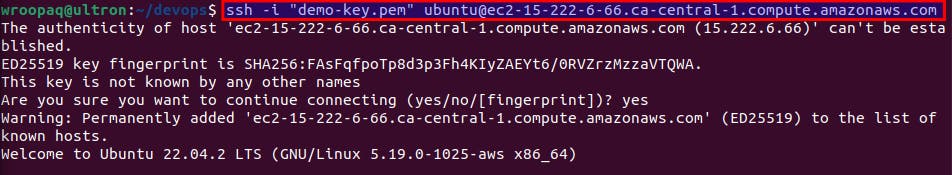
Let's go through the process:

**1. Create an EC2 instance in AWS:**

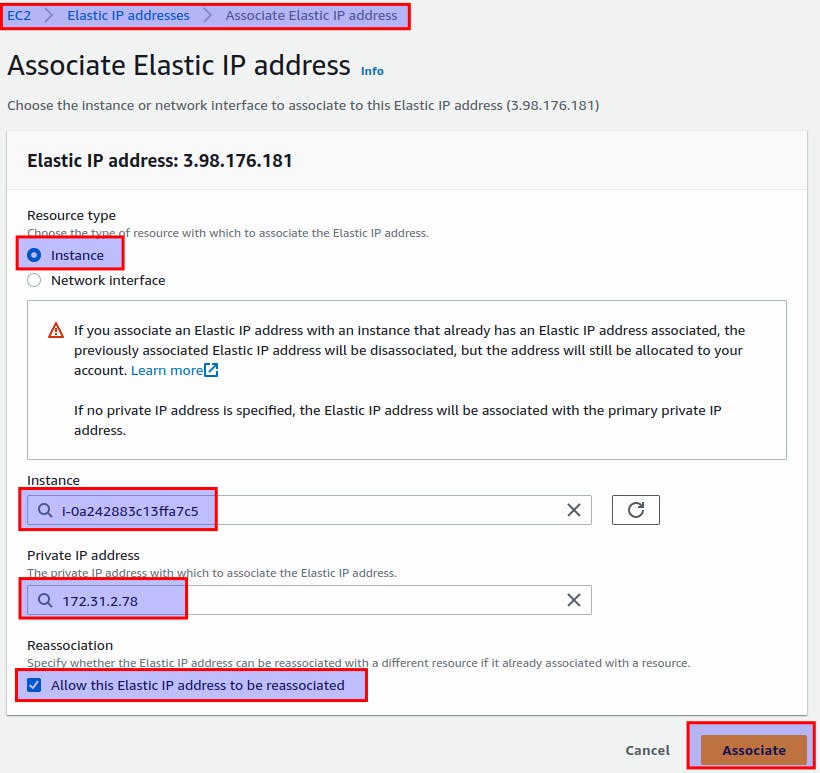
* Log in to the AWS Management Console and navigate to the EC2 service.
* Click on "Launch Instance" and select the Ubuntu t2.large instance type.



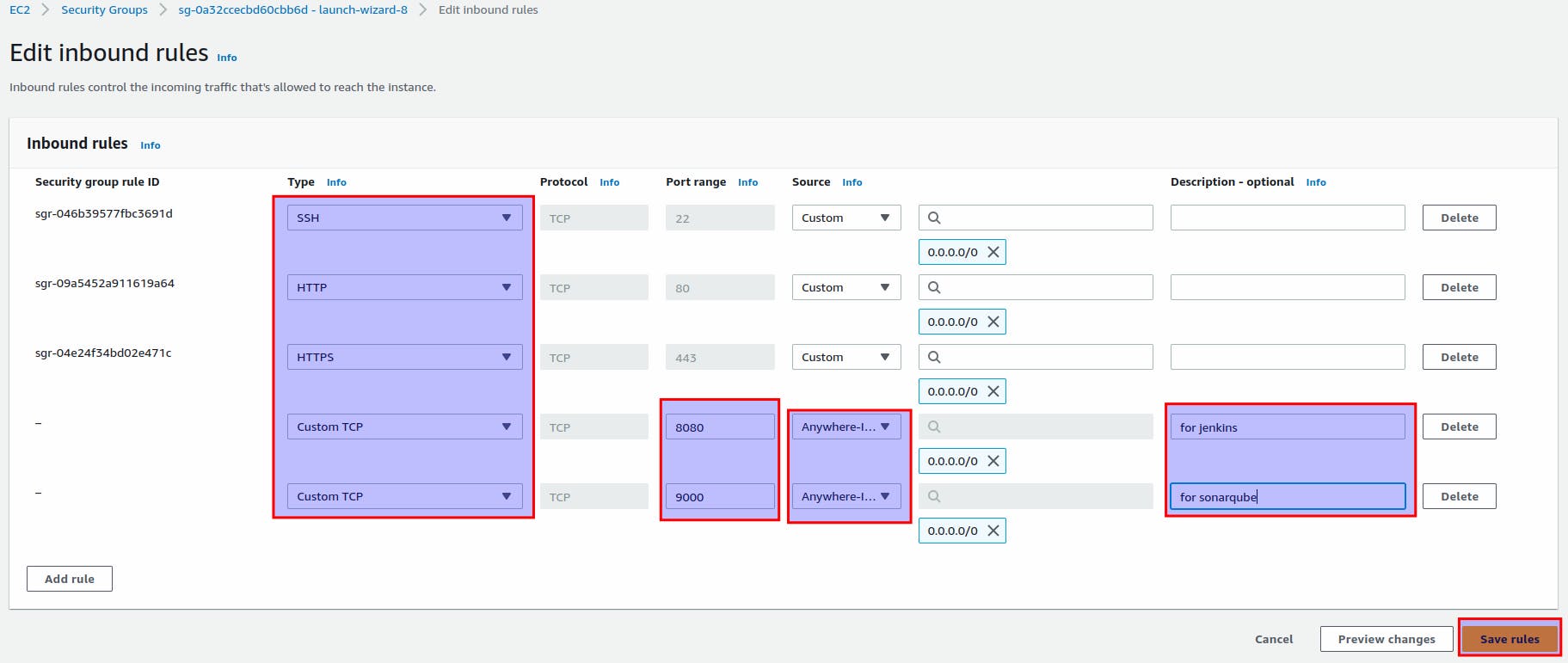
* Launch the instance and make a note of the public IP address.



**Allocate the elastic IP address and associate it with the newly created EC2 instance.**



Edit inbound rules:



Port 22: for SSH

Port 80 and 443: for HTTP and HTTPS

Port 8080: for Jenkins

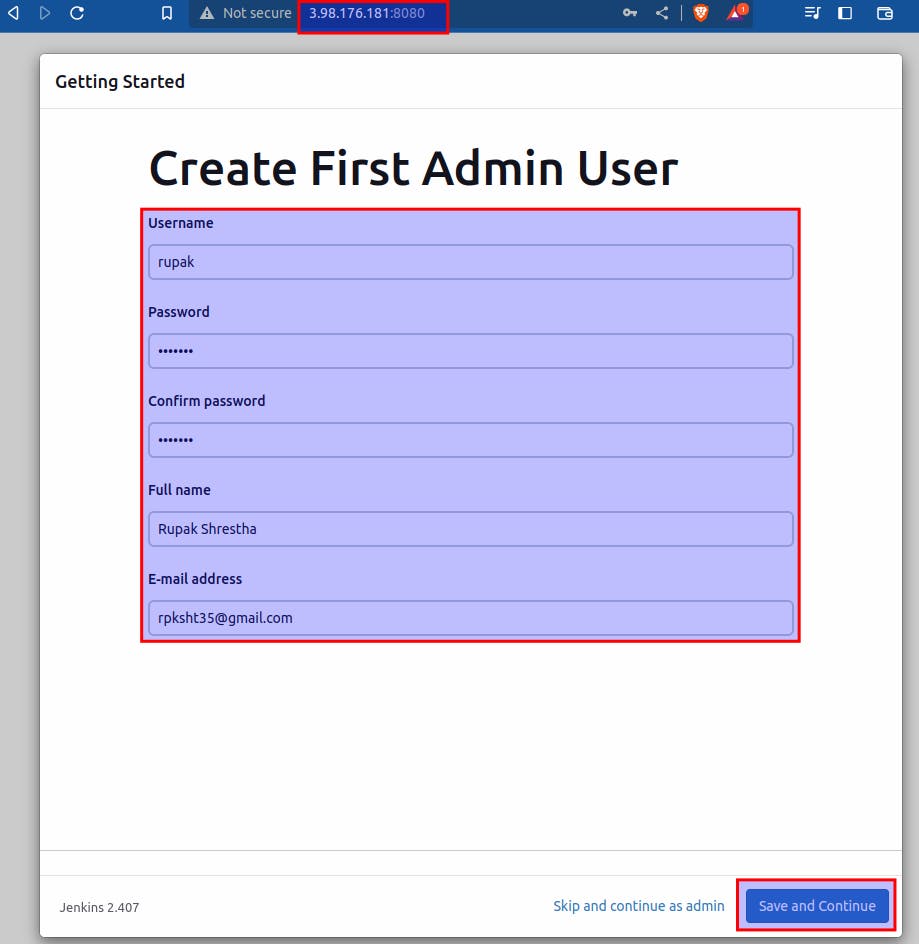
Port 9000: for SonarQube

**2. Install and configure Jenkins:**

* SSH into the EC2 instance using a terminal or SSH client.
* Update the system packages: sudo apt update
* Install Jenkins by following the official Jenkins documentation for Ubuntu: <https://www.jenkins.io/doc/book/installing/linux/>

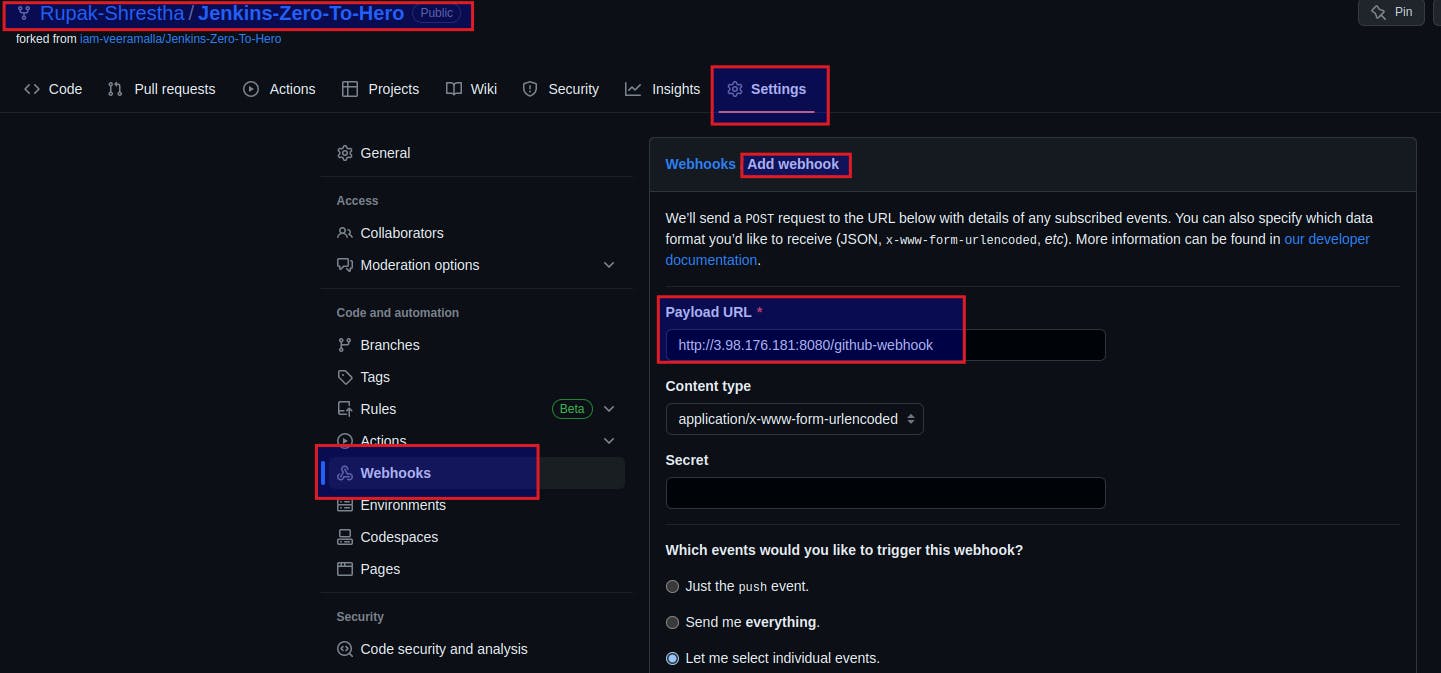
or my previous blog: <https://rupaks.hashnode.dev/jenkins-cicd>

* Once installed, access Jenkins by navigating to http://<public-ip>:8080 in a web browser.
* Follow the on-screen instructions to complete the Jenkins setup, including installing suggested plugins.



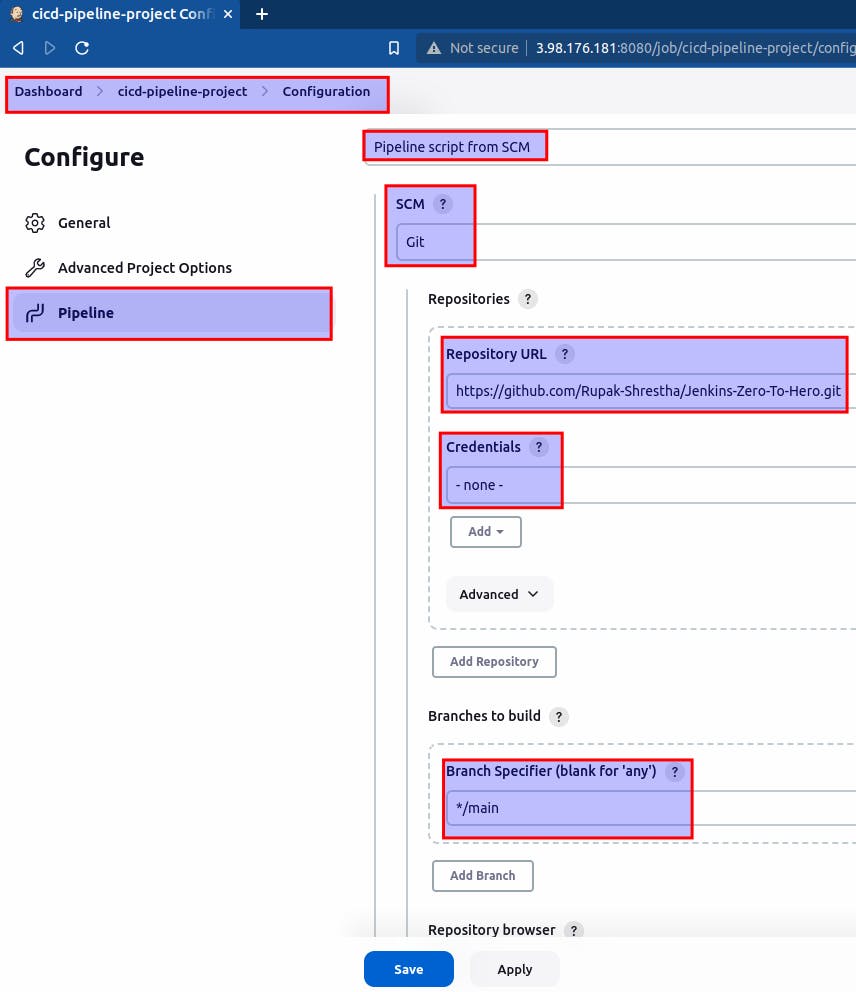
**3. Fork the GitHub repository and create a webhook:**

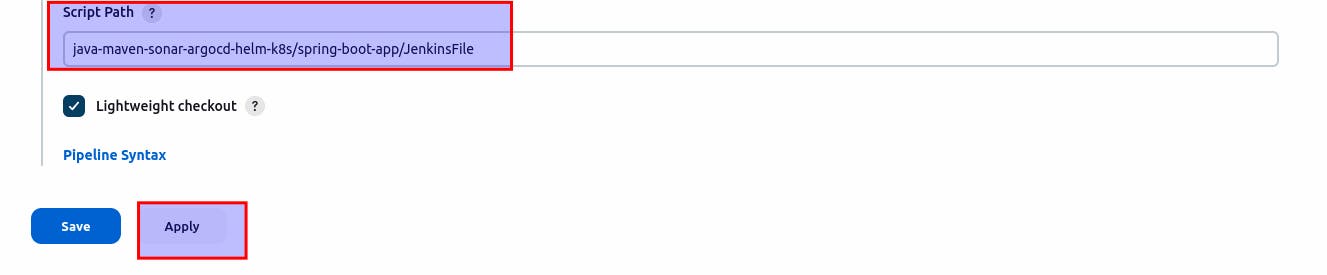
* Fork the repository at <https://github.com/Rupak-Shrestha/Jenkins-Zero-To-Hero> to your GitHub account.
* In your forked repository, go to "Settings" > "Webhooks" > "Add webhook".
* Set the Payload URL to http://<public-ip>:8080/github-webhook/

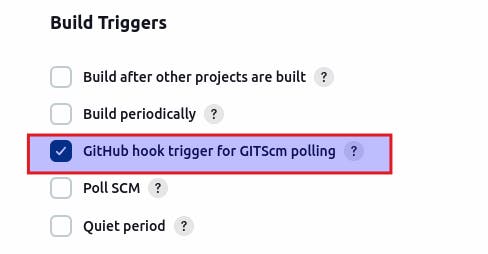


* Select "Just the push event" under "Which events would you like to trigger this webhook?".
* Save the webhook configuration.

**4. Build a new item in Jenkins:**

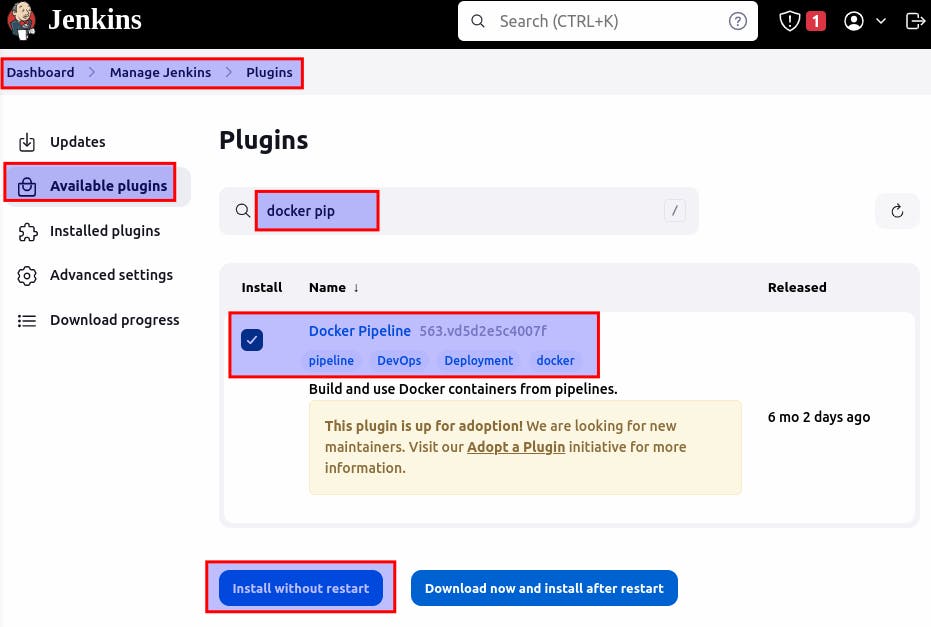




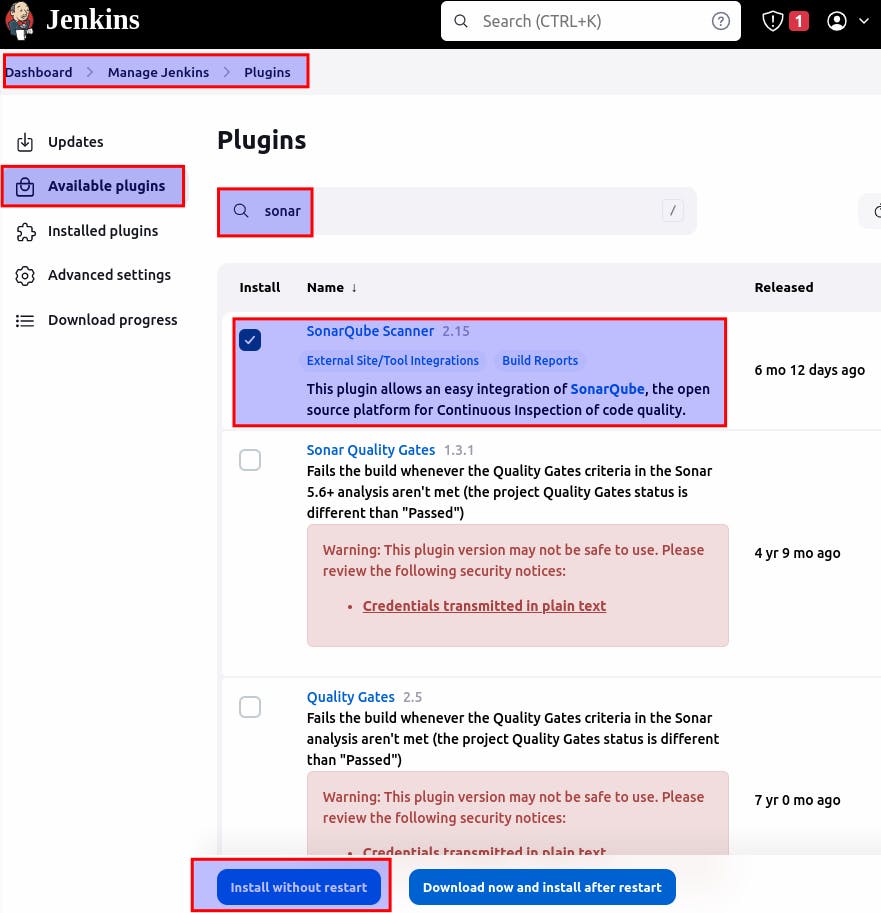


**5. Installing plugins:**

Install the docker pipeline plugin:



Install SonarQube Scanner plugin:



**6. Install SonarQube in Ubuntu and configure it in Jenkins:**

* SSH into the EC2 instance.
* Follow the official SonarQube documentation for Ubuntu installation: <https://docs.sonarqube.org/latest/setup/install-server/>

# Install the 'unzip' package

apt install unzip

# Create a new user named 'sonarqube'

adduser sonarqube

# Switch to the 'sonarqube' user

sudo su - sonarqube

# Download SonarQube zip package

wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.4.0.54424.zip

# Unzip the downloaded package

unzip \*.zip

# Set appropriate permissions for the SonarQube directory

chmod -R 755 /home/sonarqube/sonarqube-9.4.0.54424

# Change ownership of the SonarQube directory to the 'sonarqube' user

chown -R sonarqube:sonarqube /home/sonarqube/sonarqube-9.4.0.54424

# Change the current directory to the SonarQube bin directory

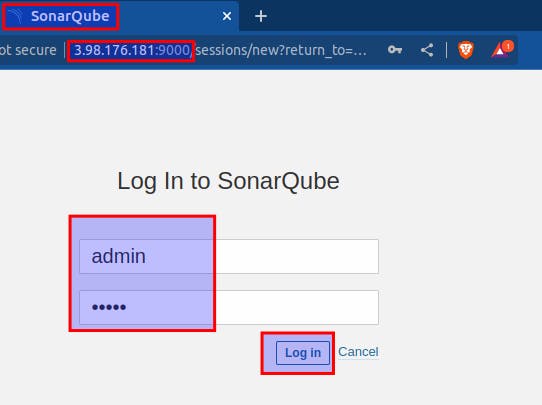
cd sonarqube-9.4.0.54424/bin/linux-x86-64/

# Start SonarQube

./sonar.sh start

* Once SonarQube is installed and running, access it through

http://<public-ip>:9000

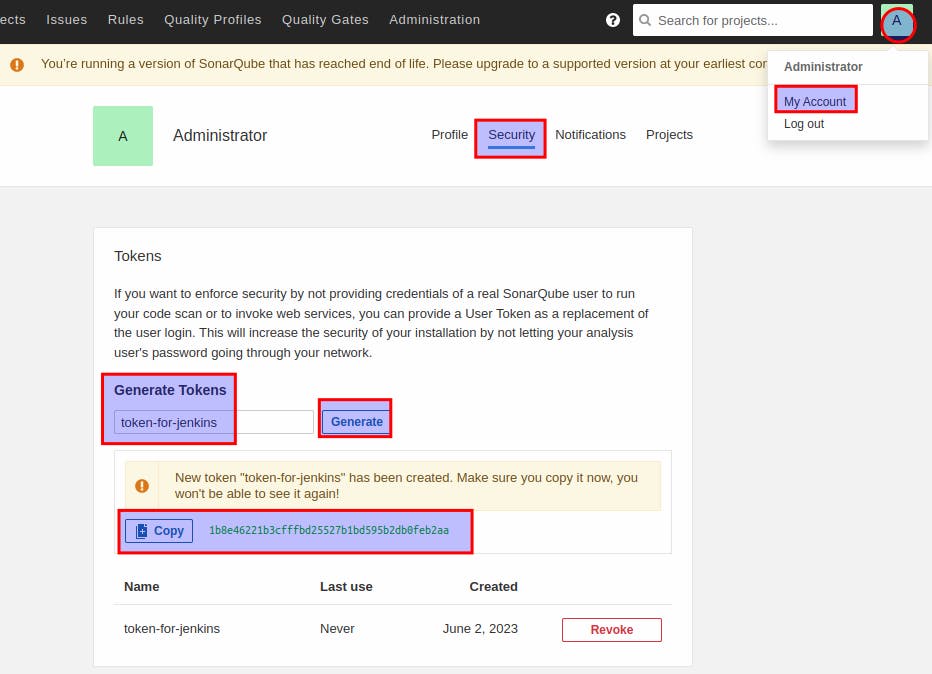


Username and Password both: admin

**7. Adding Credentials in Jenkins**

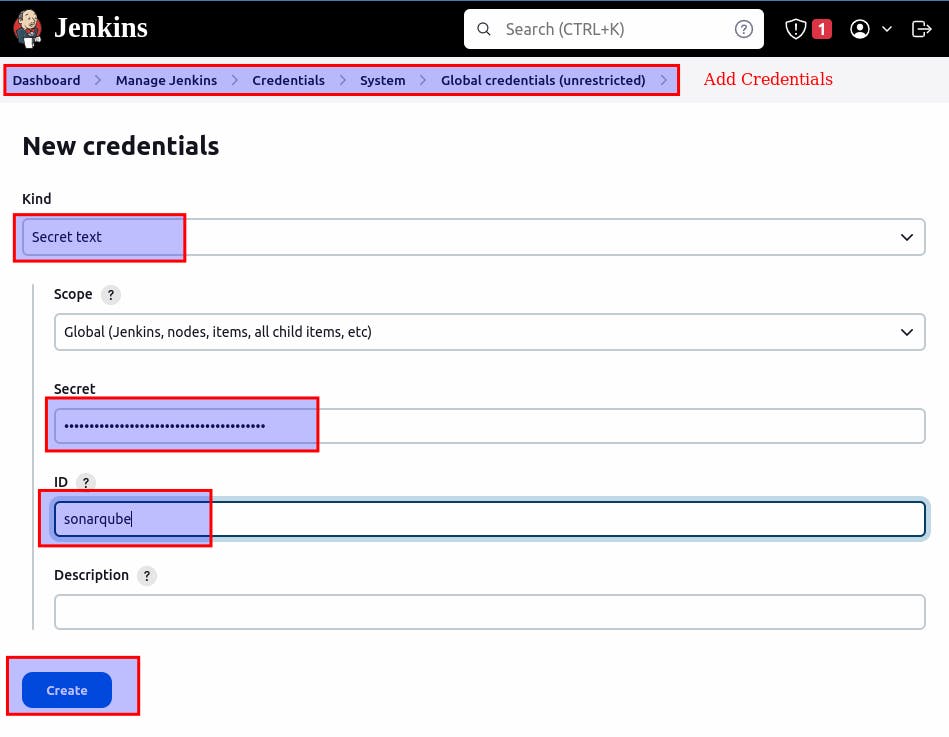
**Connecting Jenkins and SonarQube:**

Generate token in SonarQube:



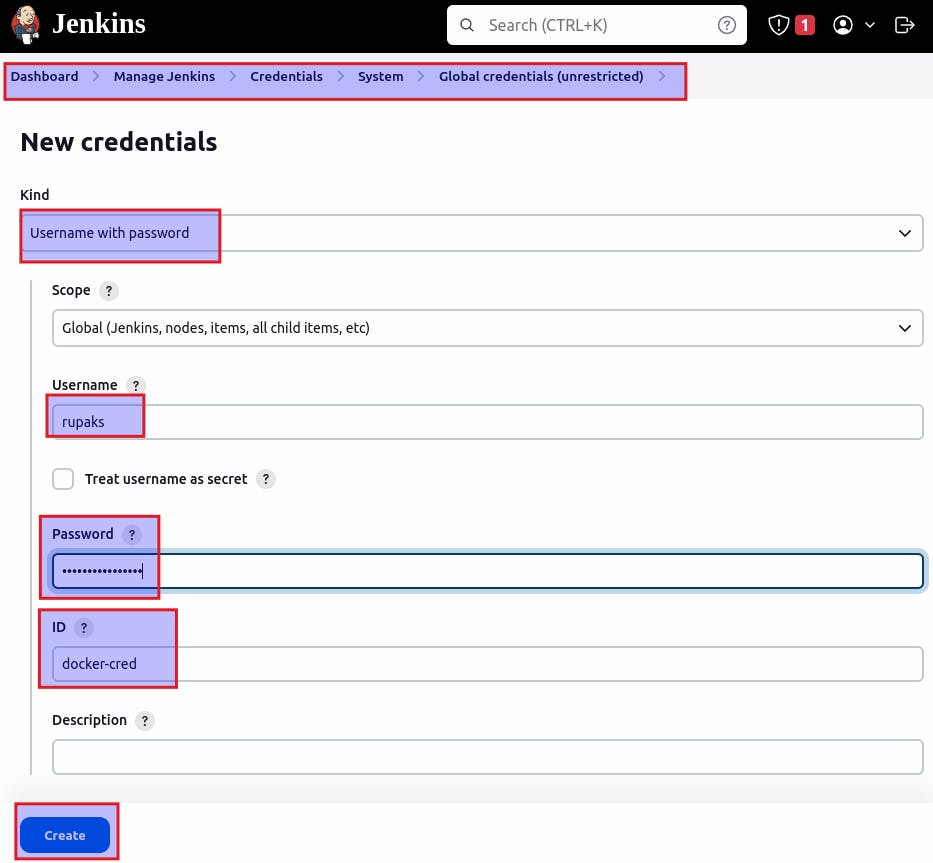
Copy the token.

Create credentials in jenkins:



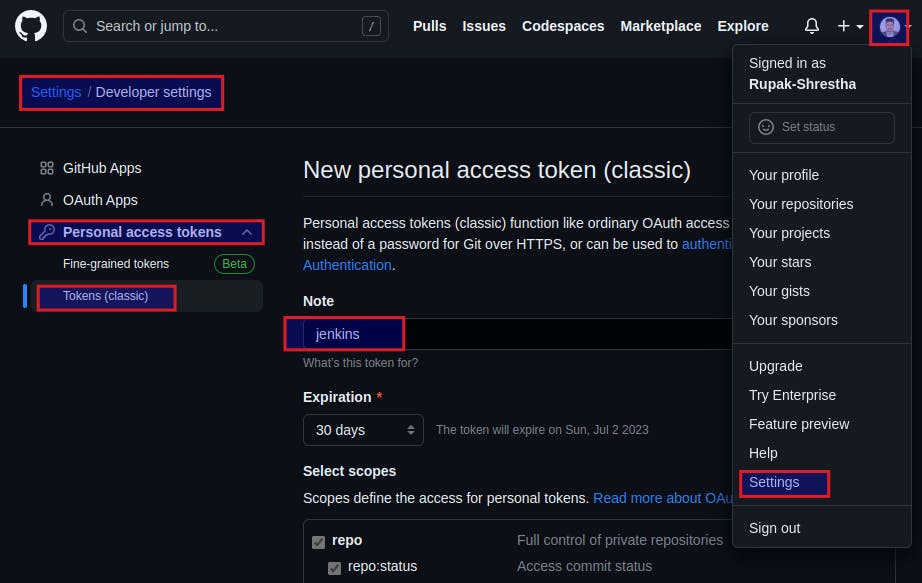
Paste the token in secret.

**Adding credentials for DockerHub in Jenkins:**

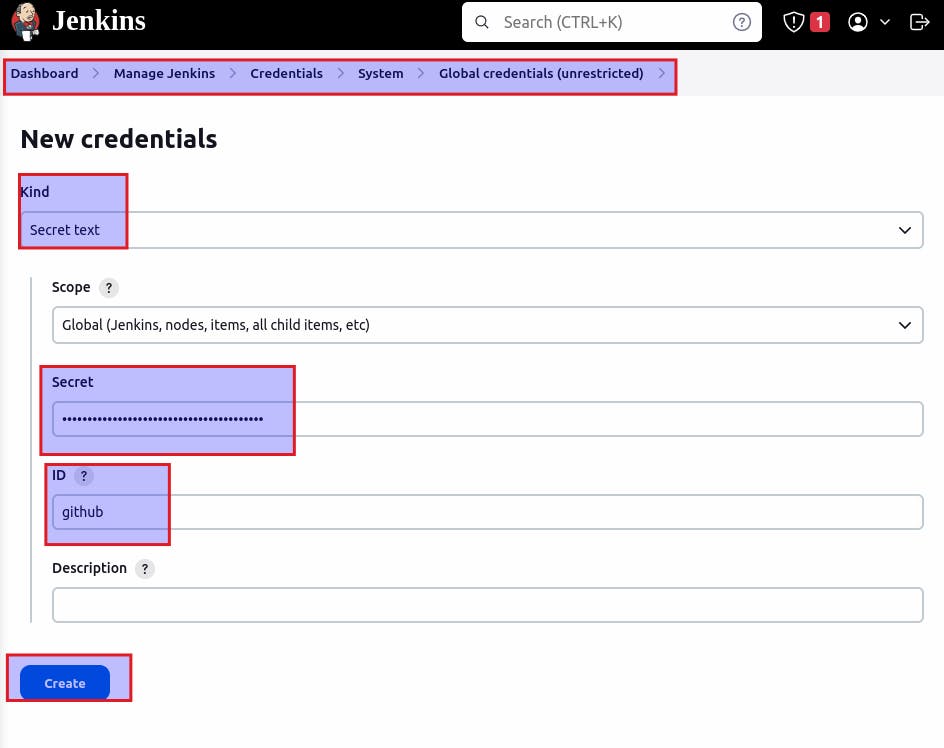


**Adding credentials for github in Jenkins:**

Generate a token:



Copy the token and paste on secret:



**Restart Jenkins.**

**8. Install Docker and configure it in Jenkins:**

* SSH into the EC2 instance.
* Follow the official Docker documentation for Ubuntu installation: [https://docs.docker.com/engine/install/ubuntu](https://docs.docker.com/engine/install/ubuntu/%EF%BF%BCsudo)

# Update the package lists

sudo apt update

# Install Docker

sudo apt install docker.io

# Switch to the root user

sudo su -

# Add the 'jenkins' user to the 'docker' group

usermod -aG docker jenkins

# Add the 'ubuntu' user to the 'docker' group

usermod -aG docker ubuntu

# Grant permissions to the Docker socket

sudo chmod 666 /var/run/docker.sock

# Restart the Docker service

systemctl restart docker

**9. Writing Jenkinsfile and manifest file:**

Jenjinsfile:

pipeline {

agent {

docker {

image 'abhishekf5/maven-abhishek-docker-agent:v1' // Docker image to use for the pipeline

args '--user root -v /var/run/docker.sock:/var/run/docker.sock' // Mount Docker socket to access the host's Docker daemon

}

}

stages {

stage('Checkout') {

steps {

sh 'echo passed' // Print a message

// git branch: 'main', url: 'https://github.com/iam-veeramalla/Jenkins-Zero-To-Hero.git' // Git checkout command

}

}

stage('Build and Test') {

steps {

sh 'ls -ltr' // List files and directories in the current directory

// Build the project and create a JAR file

sh 'cd java-maven-sonar-argocd-helm-k8s/spring-boot-app && mvn clean package'

}

}

stage('Static Code Analysis') {

environment {

SONAR\_URL = "http://3.98.176.181:9000" // SonarQube server URL

}

steps {

withCredentials([string(credentialsId: 'sonarqube', variable: 'SONAR\_AUTH\_TOKEN')]) {

sh 'cd java-maven-sonar-argocd-helm-k8s/spring-boot-app && mvn sonar:sonar -Dsonar.login=$SONAR\_AUTH\_TOKEN -Dsonar.host.url=${SONAR\_URL}' // Run SonarQube analysis

}

}

}

stage('Build and Push Docker Image') {

environment {

DOCKER\_IMAGE = "rupaks/ultimate-cicd:${BUILD\_NUMBER}" // Docker image name with the build number

// DOCKERFILE\_LOCATION = "java-maven-sonar-argocd-helm-k8s/spring-boot-app/Dockerfile" // Dockerfile location

REGISTRY\_CREDENTIALS = credentials('docker-cred') // Docker registry credentials

}

steps {

script {

sh 'cd java-maven-sonar-argocd-helm-k8s/spring-boot-app && docker build -t ${DOCKER\_IMAGE} .' // Build Docker image

def dockerImage = docker.image("${DOCKER\_IMAGE}")

docker.withRegistry('https://index.docker.io/v1/', "docker-cred") {

dockerImage.push() // Push Docker image to the registry

}

}

}

}

stage('Update Deployment File') {

environment {

GIT\_REPO\_NAME = "Jenkins-Zero-To-Hero" // GitHub repository name

GIT\_USER\_NAME = "Rupak-Shrestha" // GitHub username

}

steps {

withCredentials([string(credentialsId: 'github', variable: 'GITHUB\_TOKEN')]) {

sh '''

git config user.email "rpksht35@gmail.com"

git config user.name "Rupak Shrestha"

BUILD\_NUMBER=${BUILD\_NUMBER}

sed -i "s/replaceImageTag/${BUILD\_NUMBER}/g" java-maven-sonar-argocd-helm-k8s/spring-boot-app-manifests/deployment.yml // Replace image tag in the deployment file

git add java-maven-sonar-argocd-helm-k8s/spring-boot-app-manifests/deployment.yml

git commit -m "Update deployment image to version ${BUILD\_NUMBER}"

git push https://${GITHUB\_TOKEN}@github.com/${GIT\_USER\_NAME}/${GIT\_REPO\_NAME} HEAD:main // Push changes to GitHub repository

'''

}

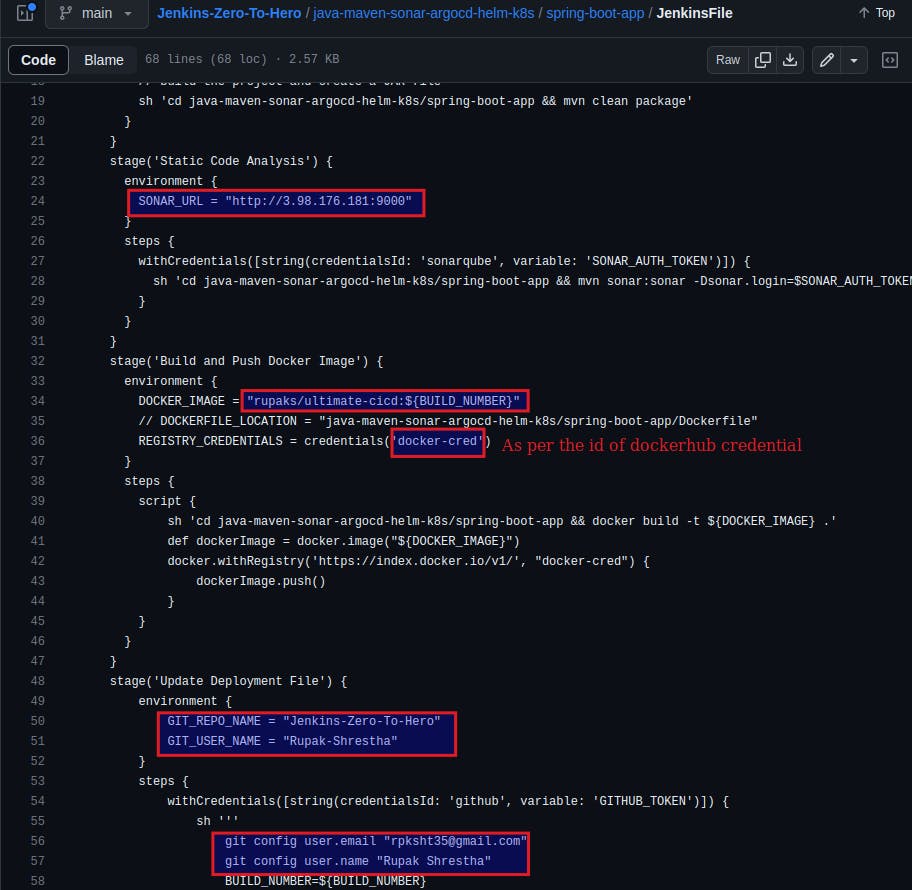
}

}

}

}

Changes to be made in file: java-maven-sonar-argocd-helm-k8s/spring-boot-app/JenkinsFile



Manifest: deployment.yml file

apiVersion: apps/v1

kind: Deployment

metadata:

name: spring-boot-app

labels:

app: spring-boot-app

spec:

replicas: 2 # Number of replicas for the deployment

selector:

matchLabels:

app: spring-boot-app

template:

metadata:

labels:

app: spring-boot-app

spec:

containers:

- name: spring-boot-app

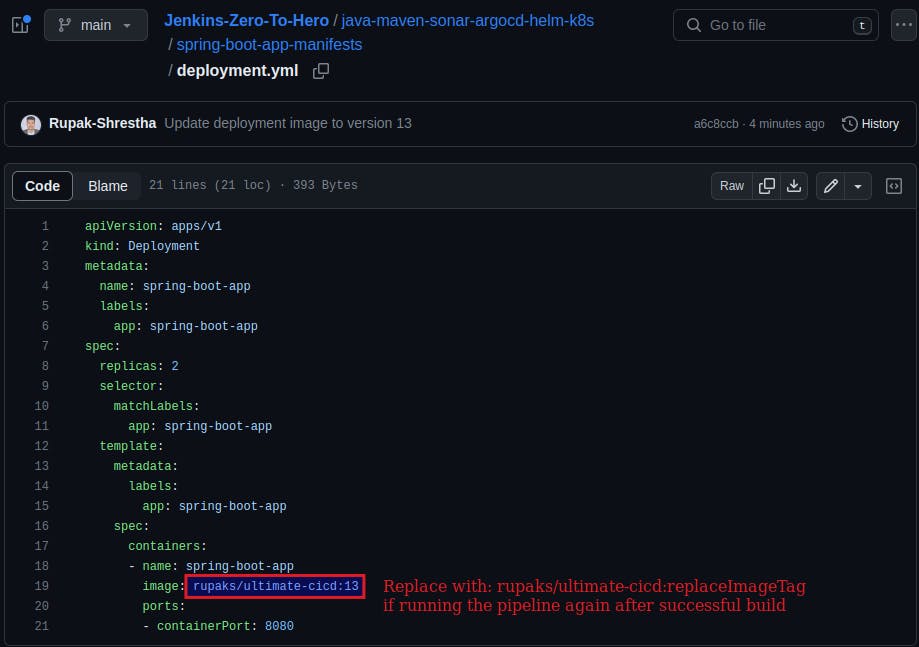
image: rupaks/ultimate-cicd:replaceImageTag # Docker image for the container

ports:

- containerPort: 8080 # Port on which the container listens

Changes to be made in file:

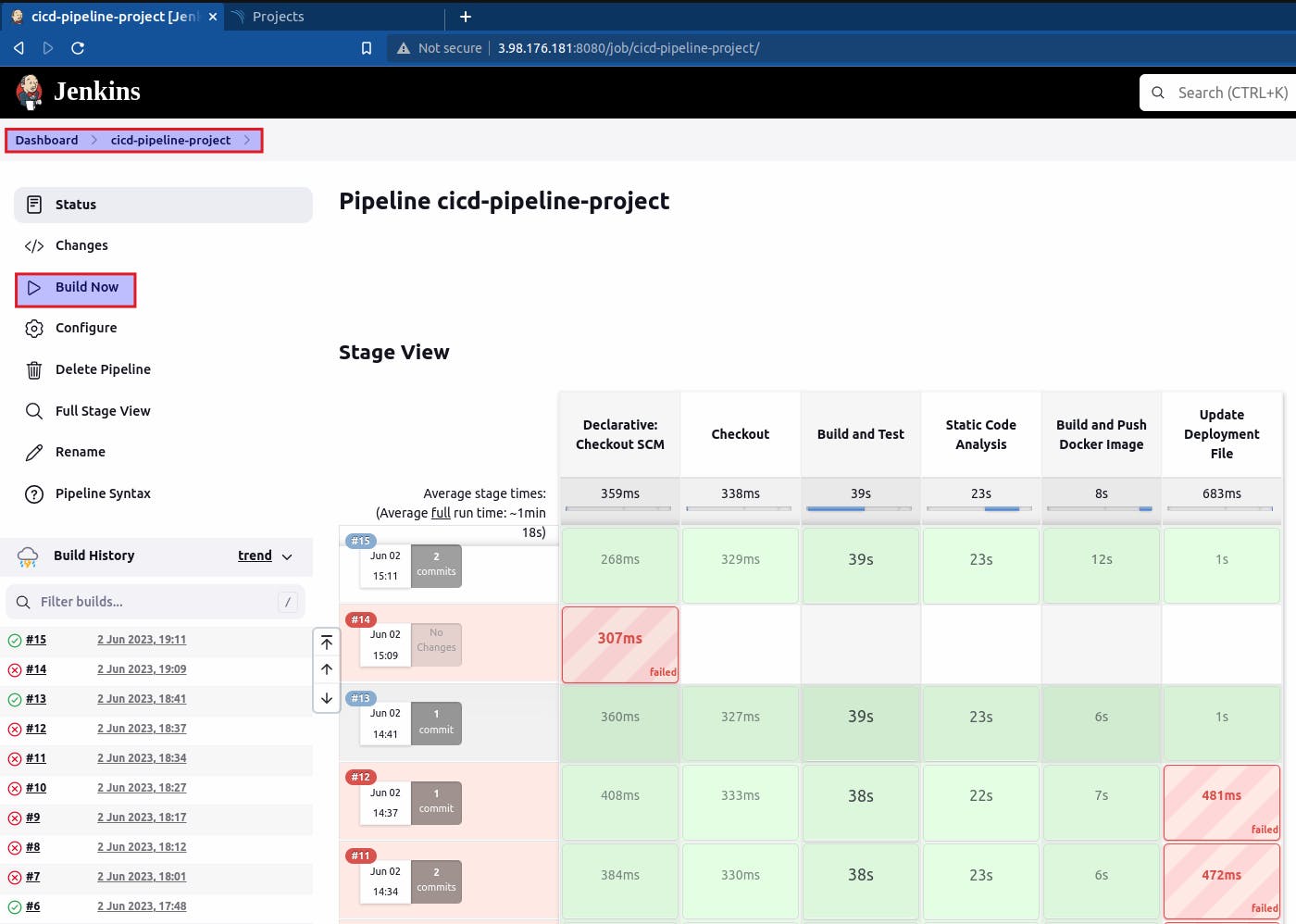
java-maven-sonar-argocd-helm-k8s/spring-boot-app-manifests/deployment.yml



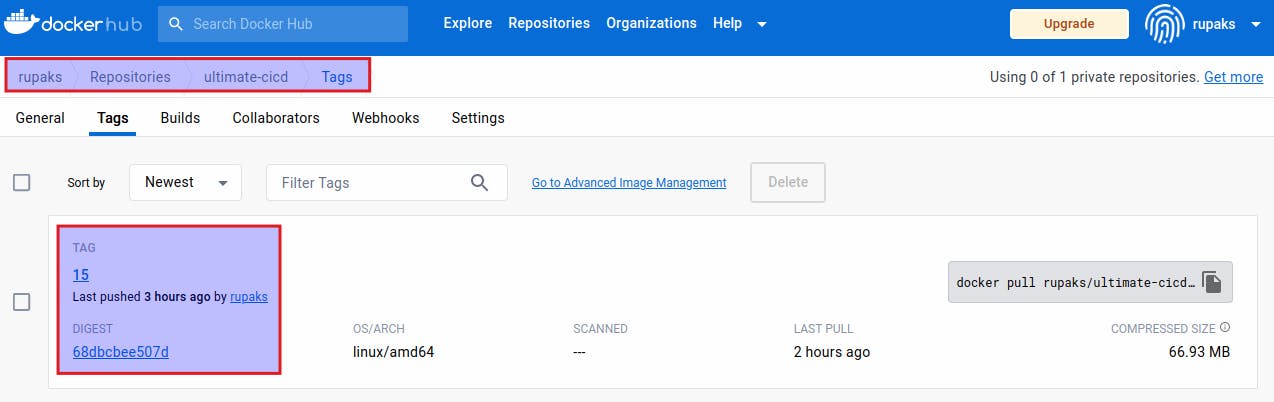
**10. Build the pipeline:**

Create a repository in your dockerhub: ultimate-cicd or as per the changes you made in your groovy script.

Then, build your pipeline.



After the build is successful, an image is pushed to dockerhub on *build and push docker image* stage. We can check it from dockerhub:



**Now let’s perform continuous deployment using ArgoCD in Kubernetes in the local system**

To install and use Minikube on Ubuntu, you can follow these steps:

**1. Install Dependencies:**

* Open a terminal on your Ubuntu machine.

# Update the package list:

sudo apt update

# Install the necessary dependencies:

sudo apt install curl virtualbox

**2. Install kubectl and minikube and start minikube**

kubectl is the command-line tool used to interact with Kubernetes clusters. Install it by running:

sudo snap install kubectl --classic

* Download the Minikube binary using curl:

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

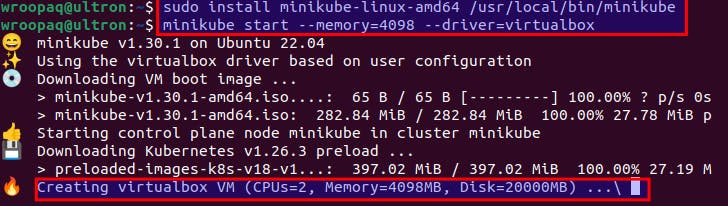
* Make the downloaded binary executable:

sudo install minikube-linux-amd64 /usr/local/bin/minikube

* Start Minikube with the default settings by running the following command:

minikube start --memory=4098 --driver=virtualbox

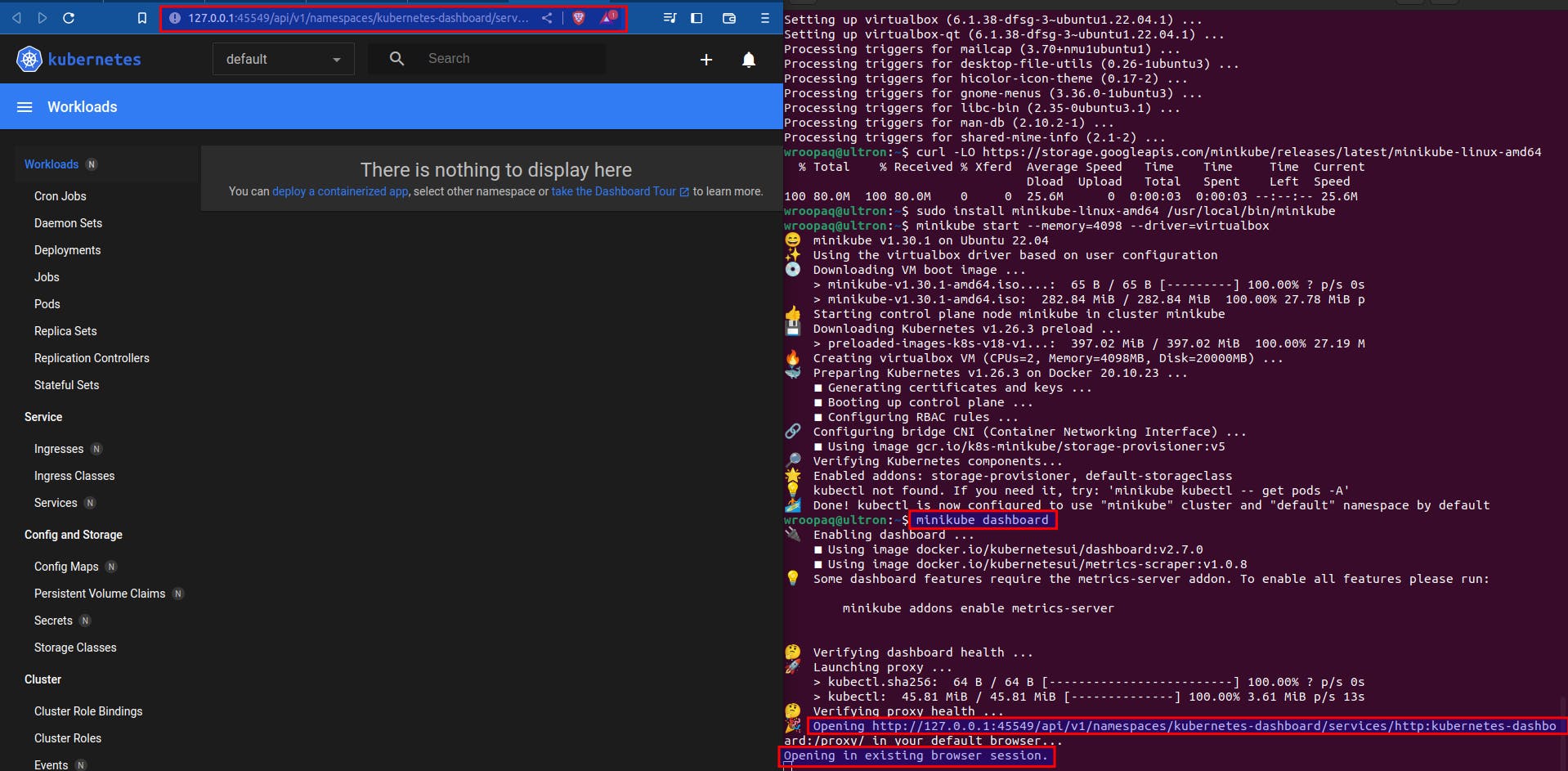
This command will start Minikube with the specified memory allocation using the VirtualBox driver as the virtualization solution on Ubuntu.



* To access the Kubernetes dashboard, run:

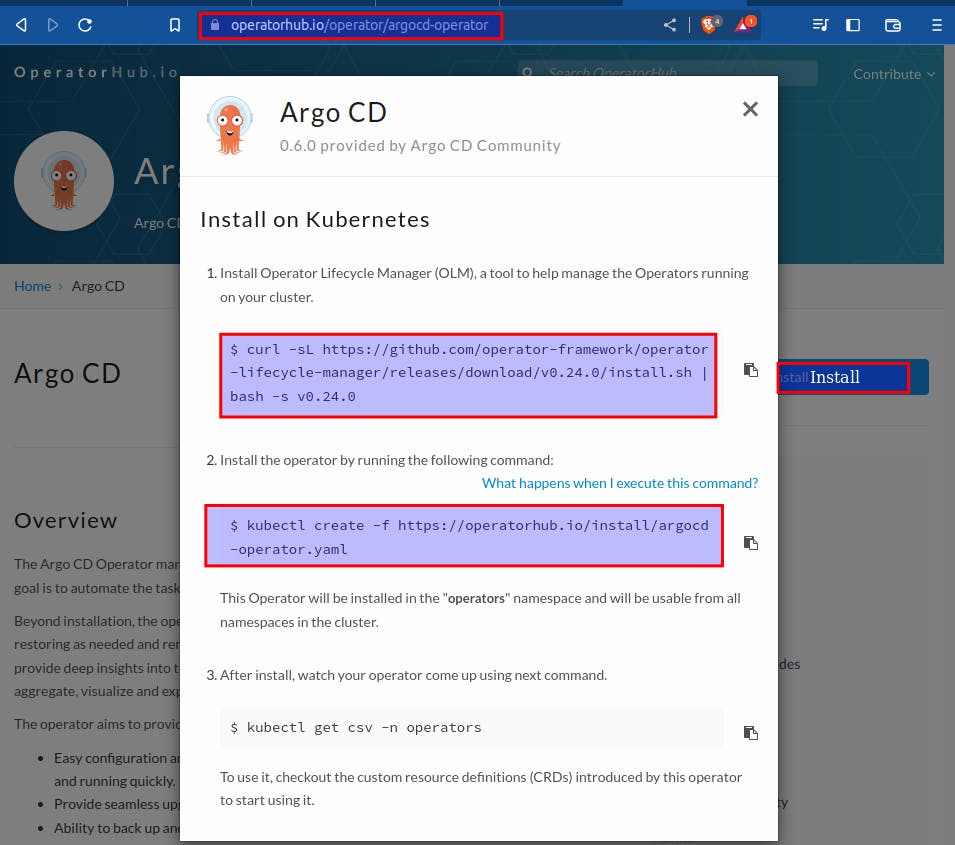
minikube dashboard

This will open the dashboard in your default web browser.



**3. Installing and configuring Argo CD**

Goto: <https://operatorhub.io/operator/argocd-operator>



# Install Operator Lifecycle Manager (OLM)

curl -sL https://github.com/operator-framework/operator-lifecycle-manager/releases/download/v0.24.0/install.sh | bash -s v0.24.0

# Create Argo CD Operator using the provided YAML file

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

# Get the pods in the 'operators' namespace

kubectl get pods -n operators



**Create: 1 file (eg: argocd.yml) and paste the following code into the file:**

**From:** <https://argocd-operator.readthedocs.io/en/latest/usage/basics/>

COPY

COPY

apiVersion: argoproj.io/v1alpha1

kind: ArgoCD

metadata:

name: example-argocd # Name of the Argo CD instance

labels:

example: basic # Labels for identification or categorization

spec: {} # Empty specification, no additional configuration provided

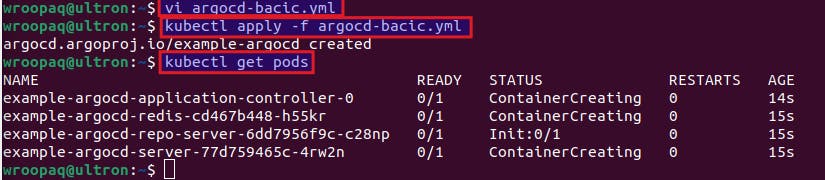
In terminal:

# Apply the configuration in the argocd.yml file

kubectl apply -f argocd.yml

# Get the pods in the cluster

kubectl get pods



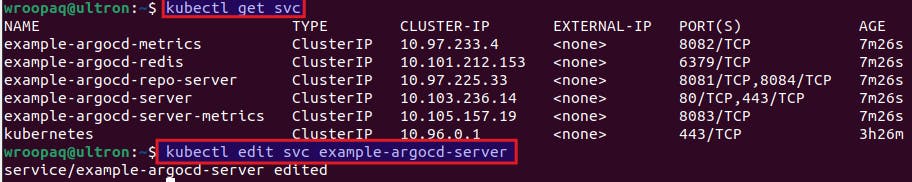
**Edit argocd-server so that the ArgoCD dashboard can be accessible from the browser.**

# Get the list of services in the cluster

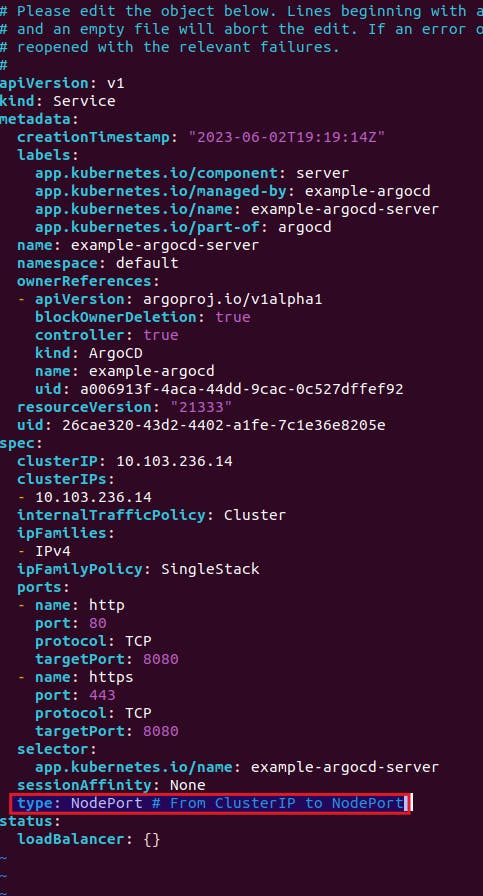
kubectl get svc

# Edit the example-argocd-server service

kubectl edit svc example-argocd-server



Change type: from ClusterIP to NodePort

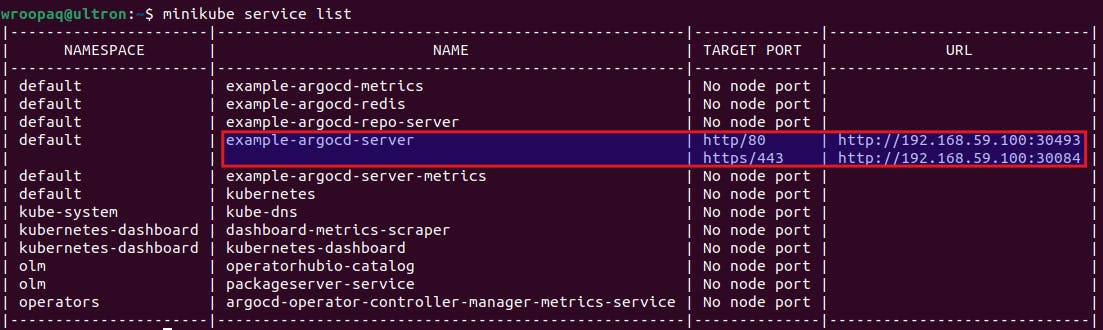


# Get the list of services in the cluster

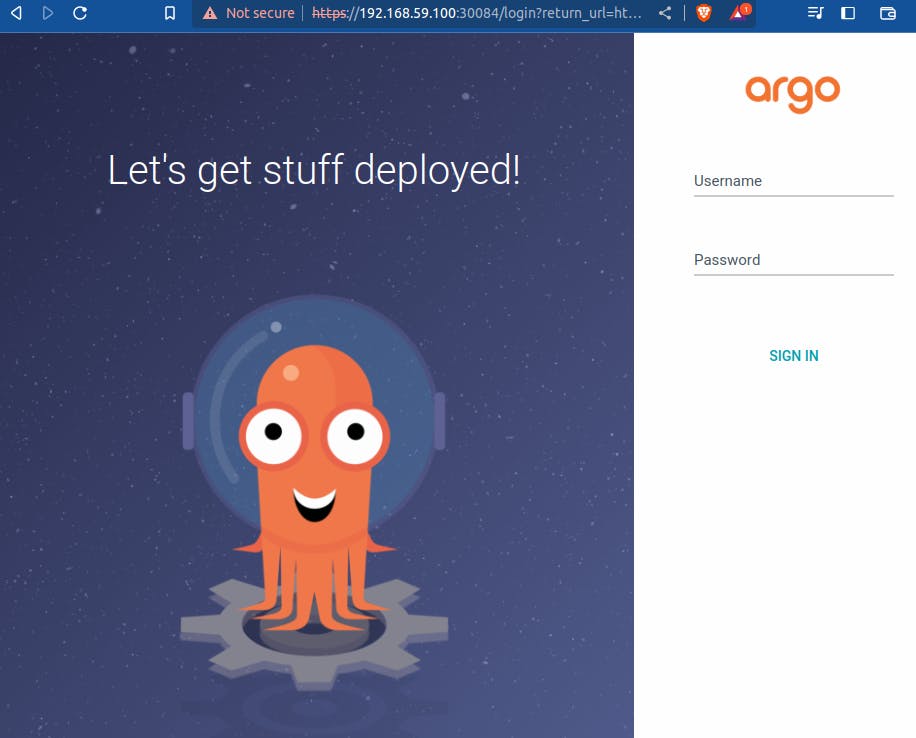
kubectl get svc

# List the services exposed by minikube

minikube service list



You'll get the URL from where you can access the ArgoCD dashboard.



Username: admin

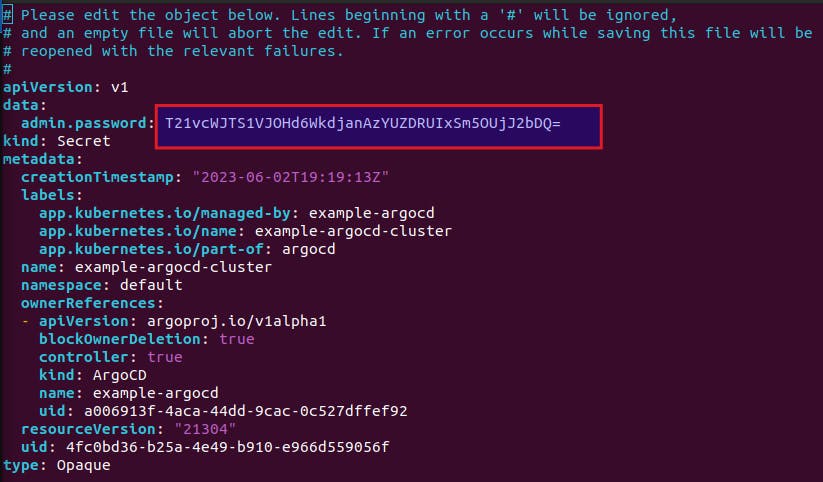
for password:

# Get the list of secrets in the cluster

kubectl get secret

# Edit the example-argocd-cluster secret

kubectl edit secret example-argocd-cluster



copy admin.password

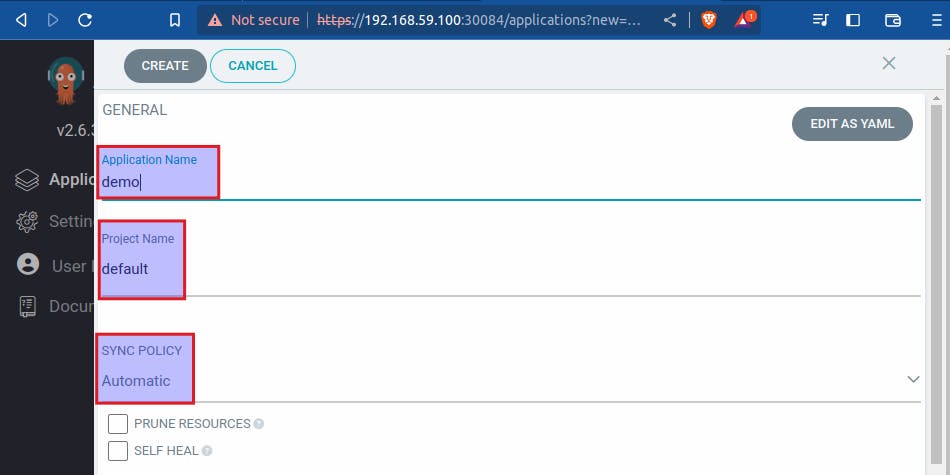
The password is base64 encrypted, so in terminal:

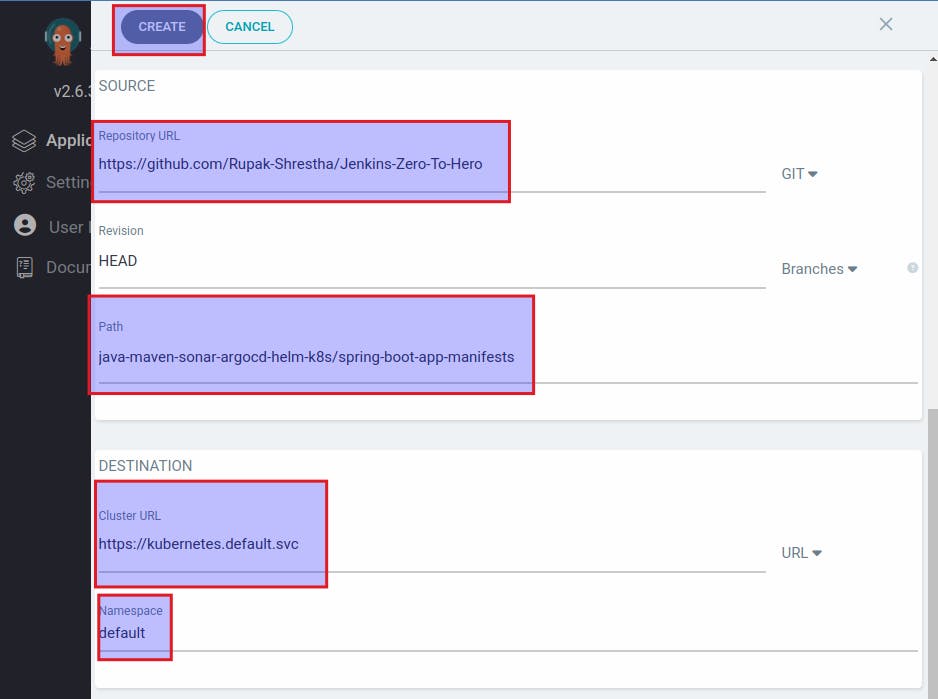
# Decode the base64-encoded string

echo T21vcWJTS1VJOHd6WkdjanAzYUZDRUIxSm5OUjJ2bDQ= | base64 -d

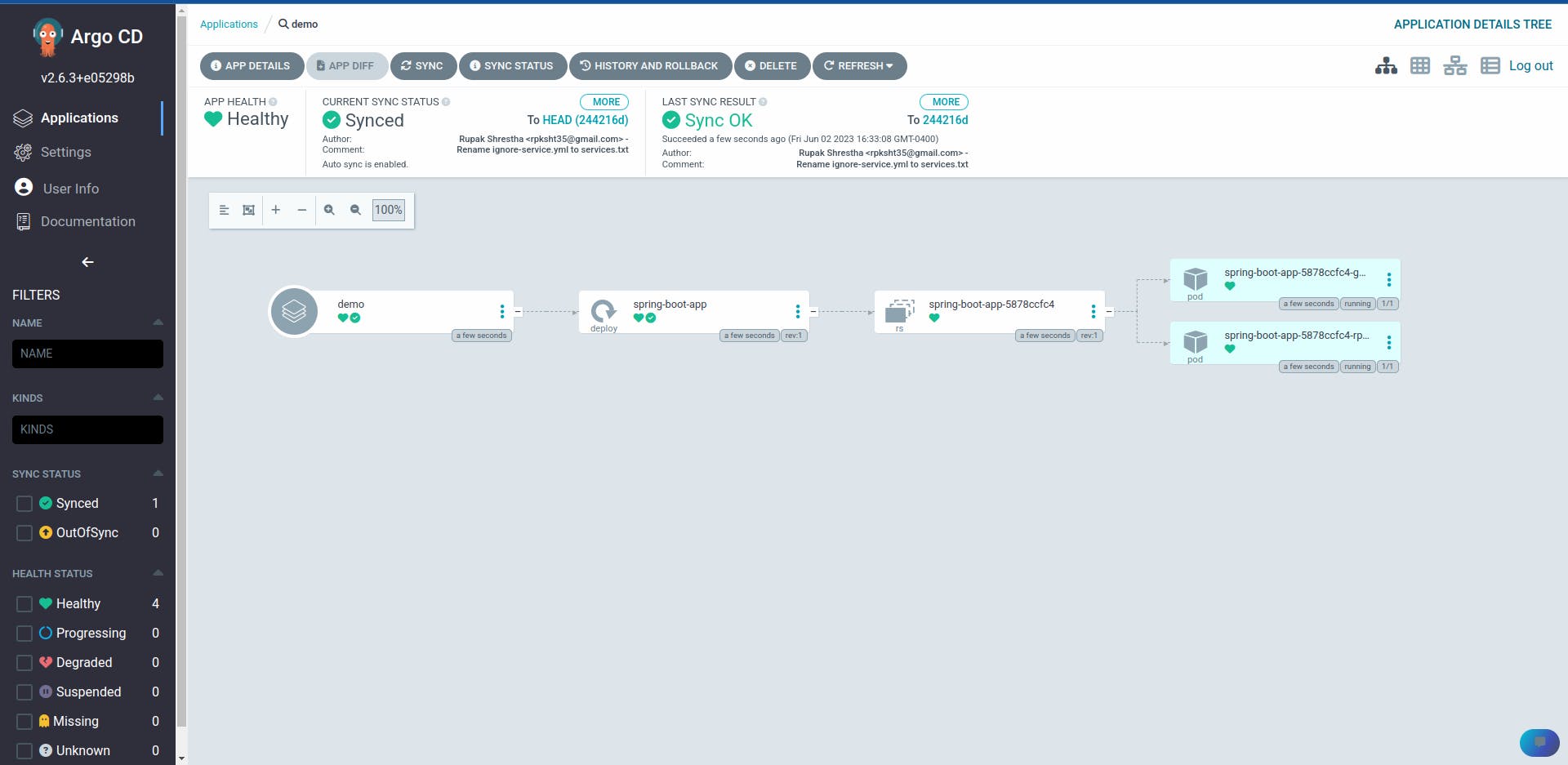
Copy the output and use it as a password.

**4. Setting up Argo CD for deployment in K8s**

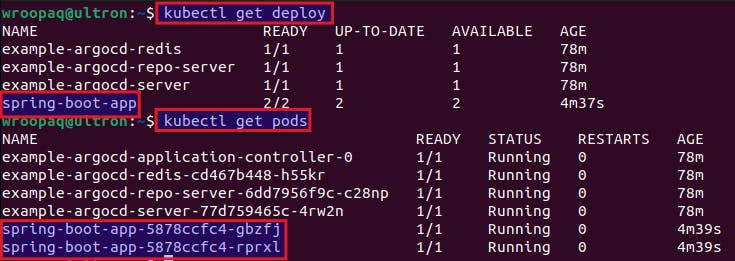




Application deployed successfully:



Checking from terminal:



Two pods are running as the replica is set to 2 in deployment.yml file.

**5. Testing the deployment from the browser:**

# kubectl describe pod <pod\_name>

# Describe the details of the pod with the name spring-boot-app-5878ccfc4-gbzfj

kubectl describe pod spring-boot-app-5878ccfc4-gbzfj

# kubectl port-forward pod/<pod\_name> <local\_port>:<application\_port>

# Forward the local port 8010 to the port 8080 of the pod spring-boot-app-5878ccfc4-gbzfj

kubectl port-forward pod/spring-boot-app-5878ccfc4-gbzfj 8010:8080

Checking the application from the browser:



**Clean up**

1. **Delete the instance**
2. **Disassociate and release elastic ip address**
3. **Stop and Delete Minikube Cluster:**

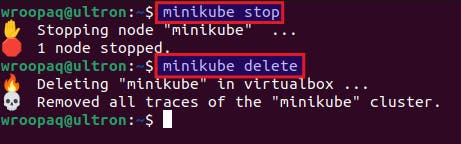
When you're finished working with Minikube, you can stop and delete the cluster by running:

# Stop the minikube cluster

minikube stop

# Delete the minikube cluster

minikube delete



This will stop the cluster and delete its resources.

To summarize, this tutorial offers a detailed and hands-on walkthrough for establishing a resilient CI/CD pipeline utilizing AWS, Jenkins, Docker, SonarQube, and ArgoCD. By following the step-by-step instructions, you can successfully automate the build, test, and deployment processes of your applications. This tutorial equips you with the knowledge and hands-on experience necessary to streamline your software development workflow, increase efficiency, and ensure consistent delivery of high-quality software. Embracing CI/CD practices can significantly enhance your development process and enable you to adapt to the ever-changing demands of the software industry.

Thank you for reading and Happy Learning! 🎉