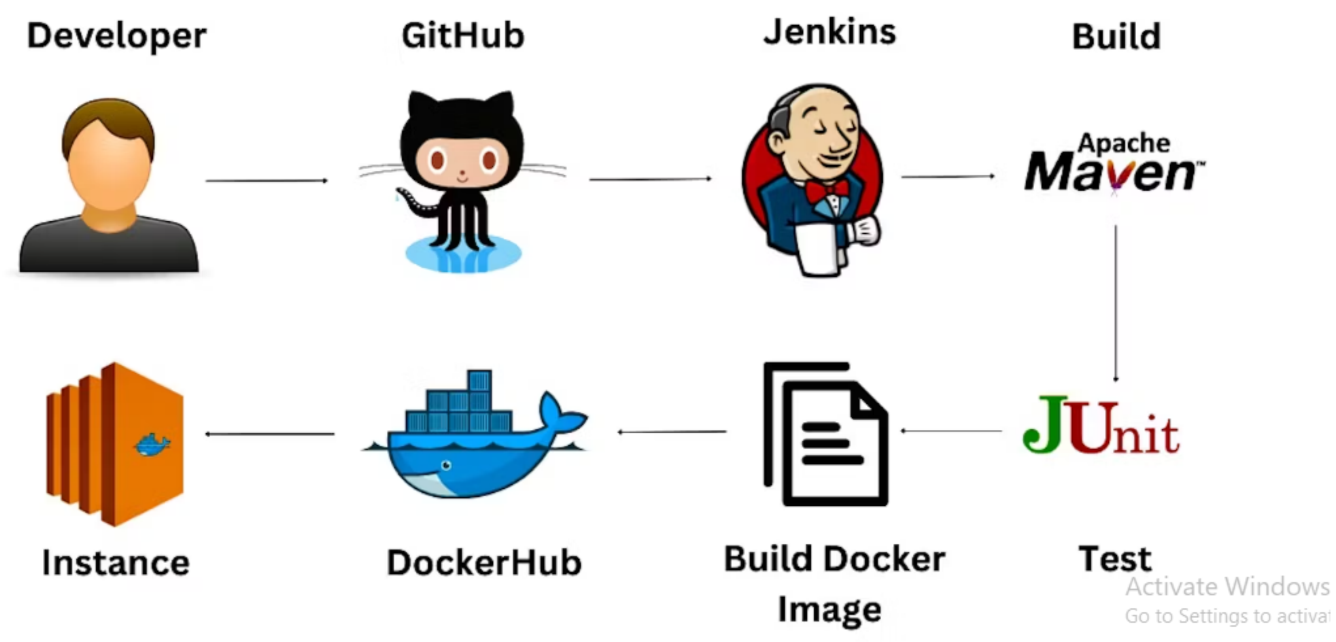
Automated CI/CD pipeline for Java Project



In this article, we will be creating an automated CI/CD pipeline for your Java project using Jenkins, Docker, and AWS. With this pipeline, your project will be automatically built, tested, and deployed to your AWS EC2 instance every time you push code to your GitHub repository.

Prerequisite:

- GitHub and DockerHub account
- AWS account and knowledge of EC2 Instances
- Knowledge of Jenkins, Docker, and Maven

Step 1: Setup Jenkins Server

First set up the Jenkins server to create a pipeline. Launch an EC2 instance and install Java and Jenkins into it. **Follow this article to set up a Jenkins server**. You need to install Maven and Docker as well. Maven to build the Java project and Docker to build the docker image in Jenkins. SSH into your instance and install maven and Docker using the below commands. For Jenkins, I am using an Ubuntu instance.

COPY

```
# Install Maven in Jenkins
sudo apt-get install maven -y
## Update packages
sudo apt-get update
```

```
## Install Docker
sudo apt-get install docker.io -y
## Add Jenkins user to Docker group
sudo usermod -a -G docker jenkins
```

Set JAVA_HOME and MAVEN_HOME using the below commands. To set the variable permanently, you should add it to the .bashrc file in your home directory.

```
echo "export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64" >> ~/.bashrc
echo "export MAVEN_HOME=/usr/share/maven" >> ~/.bashrc
```

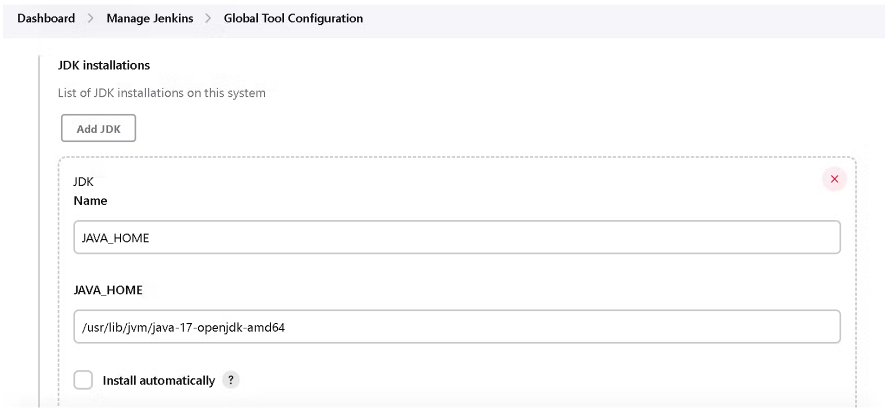
1.1. Install Plugins

Install plugins to integrate Jenkins with GitHub, Maven, and EC2. Go to **Manage Jenkins**, and select **Manage plugins.** Under available plugins search for the below plugins and **Install without restart**

1. **Git**
2. **Pipeline maven integration**
3. **Pipeline stage view**
4. **SSH Agent**

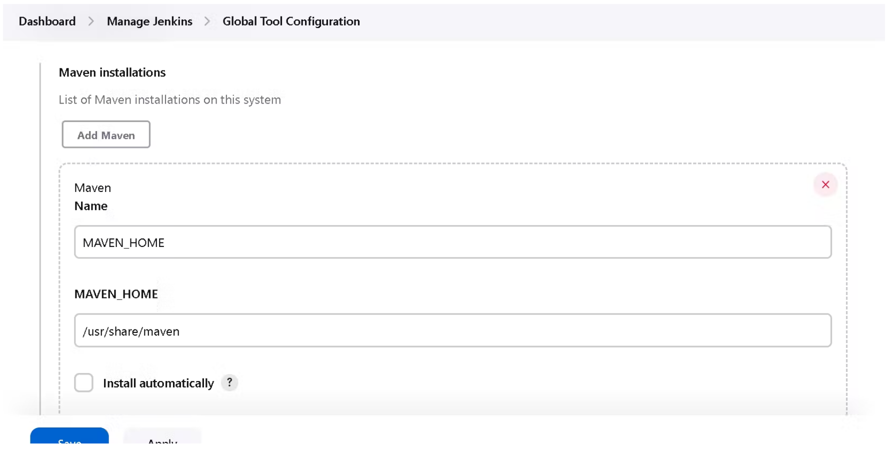
1.2. Configure Java and Maven in Jenkins

Go to **Manage Jenkins,** select **Global tool configuration**, and scroll down to add **JDK** and **Maven** path that we exported in the above steps as shown below. Uncheck the **Install automatically** checkbox, Give **Name** and **Path** and click **Save**.


Dashboard > Manage Jenkins > Global Tool Configuration
JDK installations
List of JDK installations on this system
Add JDK
JDK
Install automatically

Name: JAVA_HOME

JAVA_HOME: /usr/lib/jvm/java-17-openjdk-amd64


Dashboard > Manage Jenkins > Global Tool Configuration

Maven installations

List of Maven installations on this system

Add Maven

Maven
Name

MAVEN_HOME

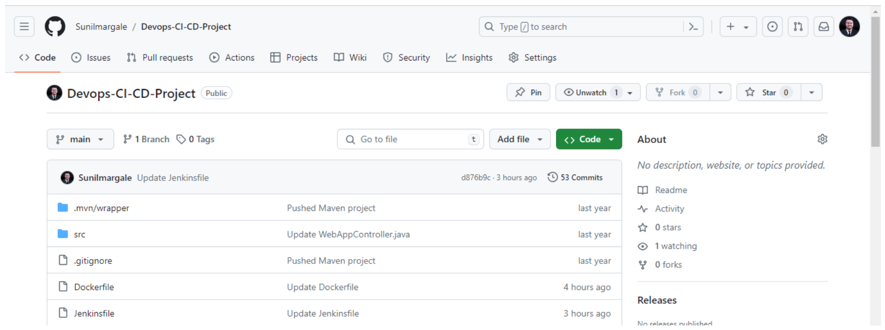
MAVEN_HOME

/usr/share/maven

☐ Install automatically ?

1.3. Create Webhook

Webhook in Jenkins triggers the pipeline automatically when any changes are done in the GitHub repository like commit and push. Go to Jenkins dashboard and copy the URL in the browser. Now go to GitHub repository settings. In the left pane select **Webhooks.**


Sunilmargale / Devops-CI-CD-Project

Code · Issues · Pull requests · Actions · Projects · Wiki · Security · Insights · Settings

Devops-CI-CD-Project Public

Pin · Unwatch 1 · Fork 0 · Star 0

main · 1 Branch · 0 Tags · Go to file · Add file · Code

Sunilmargale Update Jenkinsfile d876b9c · 3 hours ago 53 Commits

.mvn/wrapper	Pushed Maven project	last year
src	Update WebAppController.java	last year
.gitignore	Pushed Maven project	last year
Dockerfile	Update Dockerfile	4 hours ago
Jenkinsfile	Update Jenkinsfile	3 hours ago

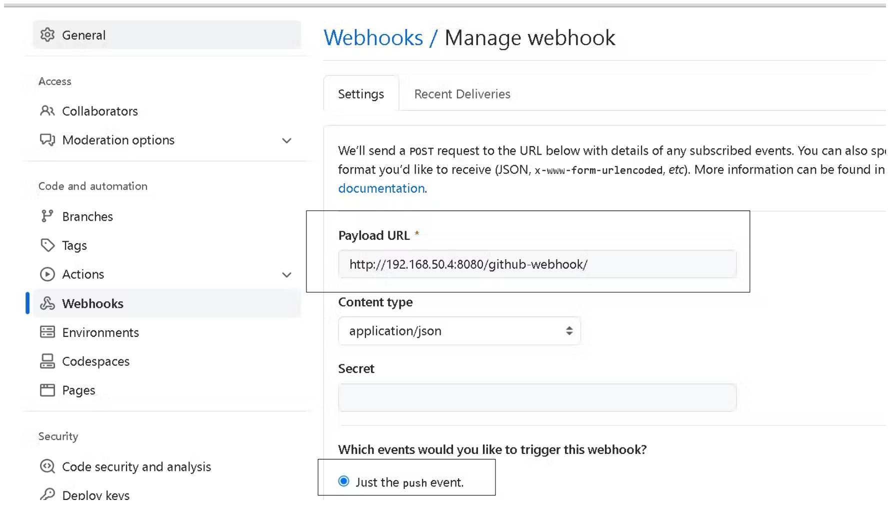
About

No description, website, or topics provided.

Readme · Activity · 0 stars · 1 watching · 0 forks

Releases

Click **Add webhook** and paste the Jenkins URL in **the Payload URL** by appending the URL with **/github-webhook/** as shown below. Select the events when you want to trigger the pipeline, I have selected **Just the push event** and click **Add webhook.**



Webhooks / Manage webhook

Settings · Recent Deliveries

We'll send a POST request to the URL below with details of any subscribed events. You can also spe format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in documentation.

Payload URL *

http://192.168.50.4:8080/github-webhook/

Content type

application/json

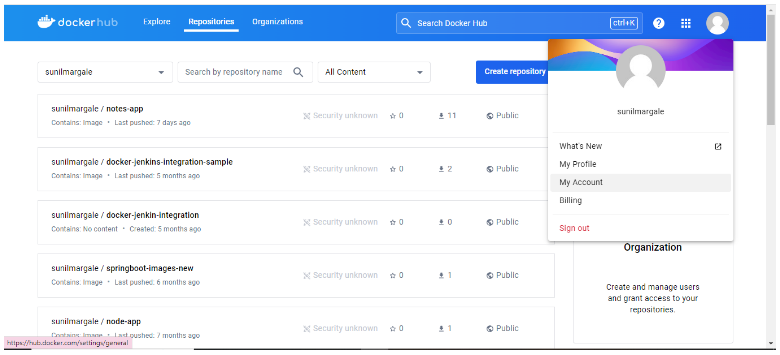
Secret

Which events would you like to trigger this webhook?

● Just the push event.

Step 2: Add DockerHub credential in Jenkins

To build and push the Docker image to DockerHub, we need to add docker credentials in Jenkins. Go to your DockerHub account and select the dropdown near username and click **Account settings.**



sunilmargale / notes-app
Contains: Image • Last pushed: 7 days ago — Security unknown — 0 — 11 — Public

sunilmargale / docker-jenkins-integration-sample
Contains: Image • Last pushed: 5 months ago — Security unknown — 0 — 2 — Public

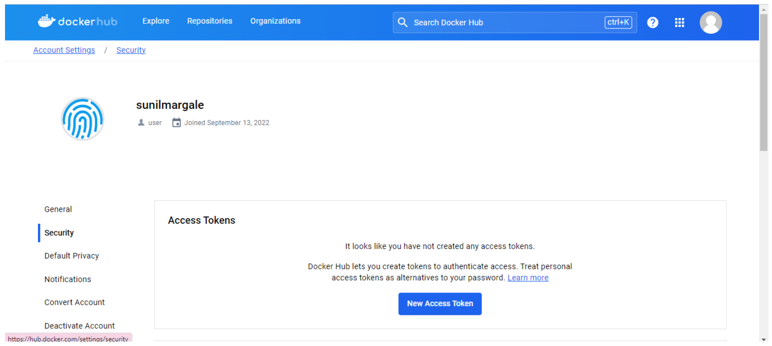
sunilmargale / docker-jenkin-integration
Contains: No content • Created: 5 months ago — Security unknown — 0 — 0 — Public

sunilmargale / springboot-images-new
Contains: Image • Last pushed: 6 months ago — Security unknown — 0 — 1 — Public

sunilmargale / node-app
Contains: Image • Last pushed: 7 months ago — Security unknown — 0 — 1 — Public

https://hub.docker.com/settings/general

Click **Security** in the left panel and click **New Access Token.** Give a **description** and click **Generate**. Copy the token and close. Keep this token as we will be adding this to the Jenkins credential in the next steps.



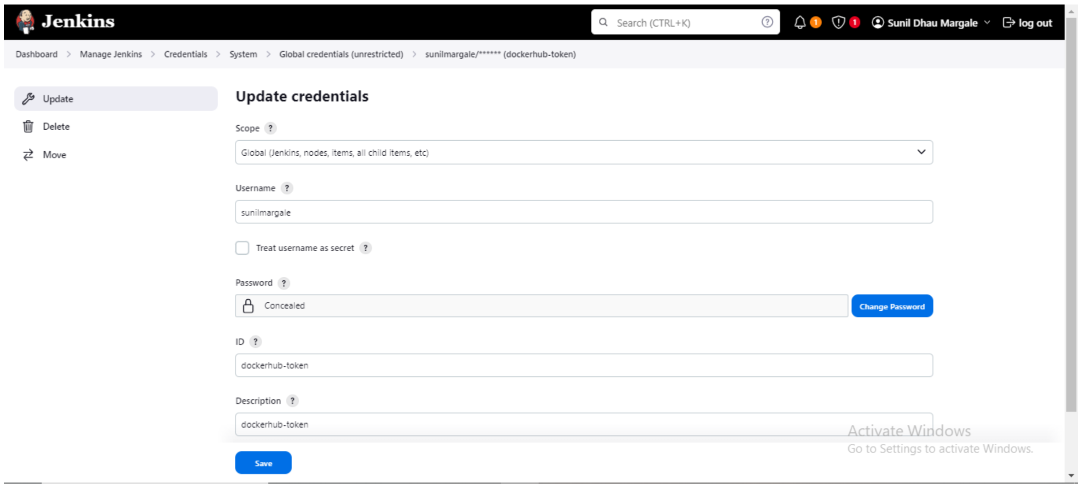
Access Tokens

It looks like you have not created any access tokens.

Docker Hub lets you create tokens to authenticate access. Treat personal access tokens as alternatives to your password. Learn more

https://hub.docker.com/settings/security

Go to the **Jenkins dashboard** → **Manage Jenkins** → **Manage credentials**. Click **System** → **Global credentials** and then click **Add credentials.** Select the kind as **Username and Password** and enter your DockerHub username and enter the token in **Password** that we have generated in dockerHub. Give the ID and description of your choice but remember this ID as we will be using it in Jenkinsfile and click **Create**.


Jenkins
Search (CTRL+K)
Sunil Dhau Margale
log out
Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) > sunilmargale/****** (dockerhub-token)
Update
Delete
Move
Change Password
Save
Activate Windows
Go to Settings to activate Windows.

Update credentials

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

Username: sunilmargale

☐ Treat username as secret

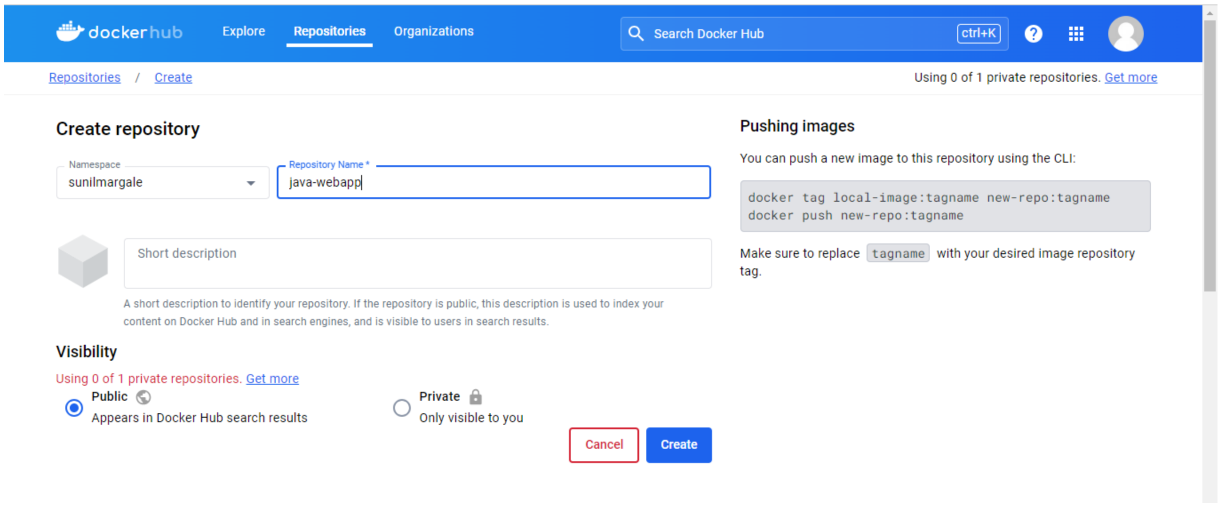
Password: Concealed

ID: dockerhub-token

Description: dockerhub-token

2.1. Create a Repository in DockerHub

Go to the DockerHub registry, select **Repositories** and click **Create repository.** Give a name to the repository and select visibility as **Public** and click **Create**.



Repositories / Create

Using 0 of 1 private repositories. Get more

Create repository

Namespace: sunilmargale

Repository Name *: java-webapp

Short description

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. Get more

[✓ Public] Appears in Docker Hub search results

Private Only visible to you

Pushing images

You can push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to replace `tagname` with your desired image repository tag.

Step 3: Setup Docker in EC2 Instance

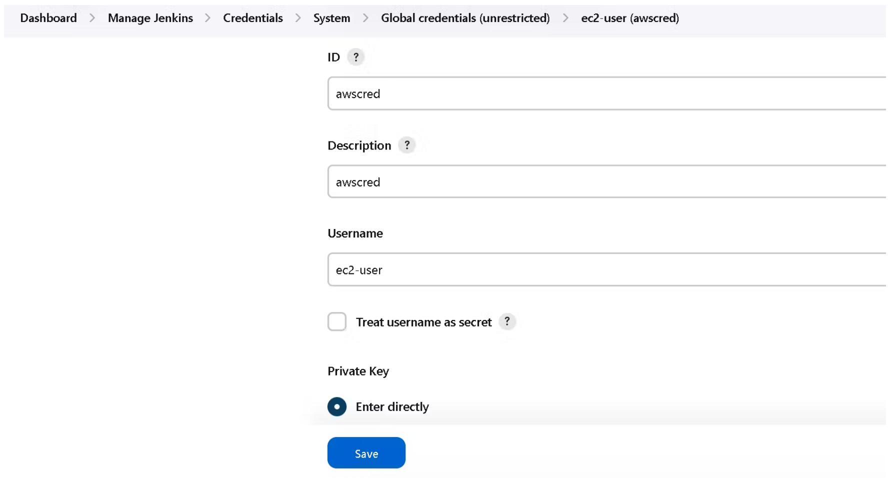
Launch an EC2 instance and install docker into it. Open port 22 to SSH into the instance and open port 8081 as I will be exposing my Java application to port 8081. After successful deployment, you will be able to access the application in the browser using `http://PUBLIC_IP:8081/hello`

I have launched the Amazon Linux instance and setup Docker using the below commands:

```
# Update packages
sudo yum update
# Install Docker
sudo yum install docker
# Add ec2-user to Docer group
sudo usermod -a -G docker ec2-user
# Enable docker service at AMI boot time
sudo systemctl enable docker.service
# Start the Docker service
sudo systemctl start docker.service
```

3.1. Add EC2 credentials in Jenkins

Go to **Jenkins Dashboard → Manage Jenkins → Manage Credentials**. Click on Global and **Add Credentials** and select **SSH Username with the private key** under **Kind**. Enter the ID and Description of your choice. Make note of this ID as we will be using it in Jenkinsfile. Enter the username of the EC2 instance that you have launched. For the Amazon Linux instance, the username is ec2-user. Select **Enter directly** and click **Add** and copy-paste the private key that you have created while launching an instance and click **Create**.


Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) > ec2-user (awscred)

ID: awscred

Description: awscred

Username: ec2-user

Treat username as secret [ ]

Private Key

● Enter directly

Step 4: Write Jenkinsfile and Dockerfile

Write Jenkinsfile with all the below steps to fetch, build, test and deploy Java application. Remember to add the below Jenkinsfile in the root directory of your project in GitHub.

GitHub Project

```
pipeline {
  agent any

  environment {
    DOCKERHUB_CREDENTIALS = credentials('docker-hub-cred')
    REMOTE_SERVER = '15.206.163.92'
    REMOTE_USER = 'ubuntu'
  }

  // Fetch code from GitHub

  stages {
    stage('checkout') {
      steps {
        git branch: 'main', url: 'https://github.com/Sunilmargale/Devops-CI-CD-Project.git'

      }
    }

   // Build Java application

    stage('Maven Build') {
      steps {
        sh 'mvn clean install'
      }

     // Post building archive Java application

      post {
        success {
          archiveArtifacts artifacts: '**/target/*.jar'
        }
      }
    }

    // Test Java application

    stage('Maven Test') {
      steps {
        sh 'mvn test'
      }
    }

   // Build docker image in Jenkins

    stage('Build Docker Image') {

      steps {
        sh 'docker build -t java-webapp:latest .'
        sh 'docker tag javawebapp sunilmargale/java-webapp:latest'
      }
    }

   // Login to DockerHub before pushing docker Image

    stage('Login to DockerHub') {
      steps {
        sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u    $DOCKERHUB_CREDENTIALS_USR --password-stdin'
      }
    }

   // Push image to DockerHub registry

    stage('Push Image to dockerHUb') {
      steps {
        sh 'docker push pasunilmargale/java-webapp:latest'
      }
      post {
        always {
          sh 'docker logout'
        }
      }

    }

   // Pull docker image from DockerHub and run in EC2 instance

    stage('Deploy Docker image to AWS instance') {
      steps {
        script {
          sshagent(credentials: ['awscred']) {
          sh "ssh -o StrictHostKeyChecking=no ${REMOTE_USER}@${REMOTE_SERVER} 'docker stop javaApp || true && do
      sh "ssh -o StrictHostKeyChecking=no ${REMOTE_USER}@${REMOTE_SERVER}'docker pull sunilmargale/java-webapp'"
          sh "ssh -o StrictHostKeyChecking=no ${REMOTE_USER}@${REMOTE_SERVER}'docker run --name javaApp -d -p 80
          }
        }
```

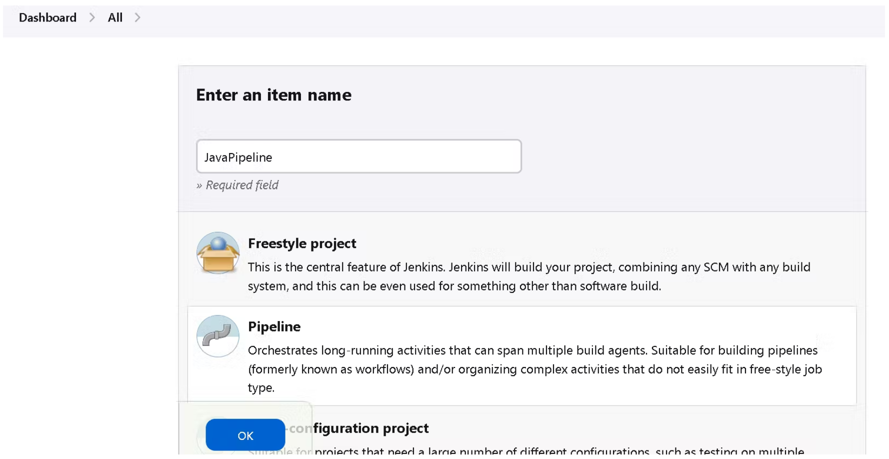
```
            }
          }
        }
      }
```

Also, write Dockerfile with the instructions to build the Java project and keep this file in the root directory of the project.

```
#Define your base image
FROM eclipse-temurin:17-jdk-focal

#Maintainer of this image
LABEL maintainer="Sunil Margale"

#Copying Jar file from target folder
COPY target/web-services.jar web-services.jar

#Expose app to outer world on this port
EXPOSE 8081

#Run executable with this command
ENTRYPOINT ["java", "-jar", "web-services.jar"]
```

Step 5: Create a Jenkins pipeline

Go to Jenkins Dashboard click **New Item → Give a name to the pipeline → Select Pipeline → Click Ok**



Enter an item name

JavaPipeline

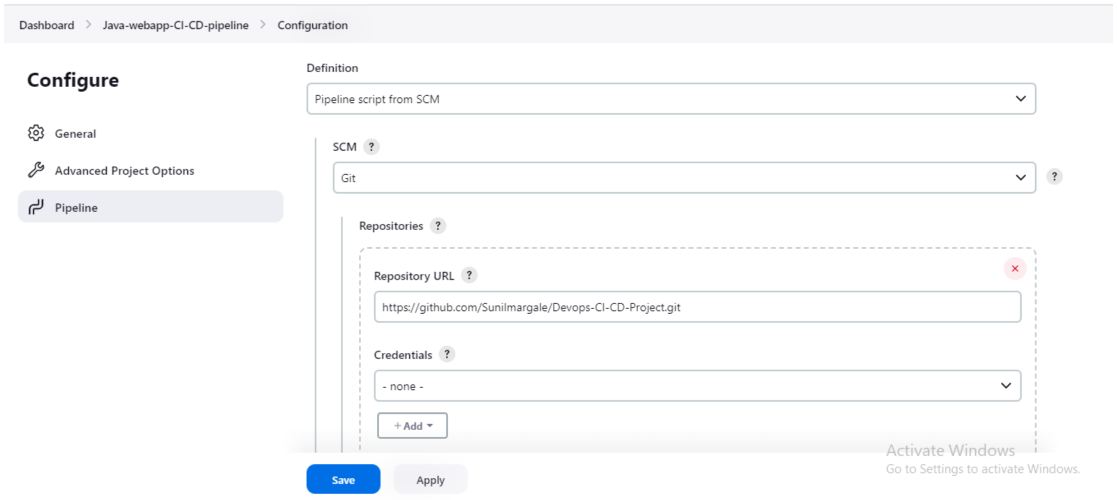
» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple

Add Description of your **pipeline → Build Triggers → GitHub hook trigger for GITScm polling**. Scroll to the last in the Pipeline section and from the dropdown select **Pipeline script from SCM.** Under SCM, select Git and enter your GitHub project repository URL. If your GitHub repository is private then add credentials.


Dashboard > Java-webapp-CI-CD-pipeline > Configuration

Configure

Definition: Pipeline script from SCM

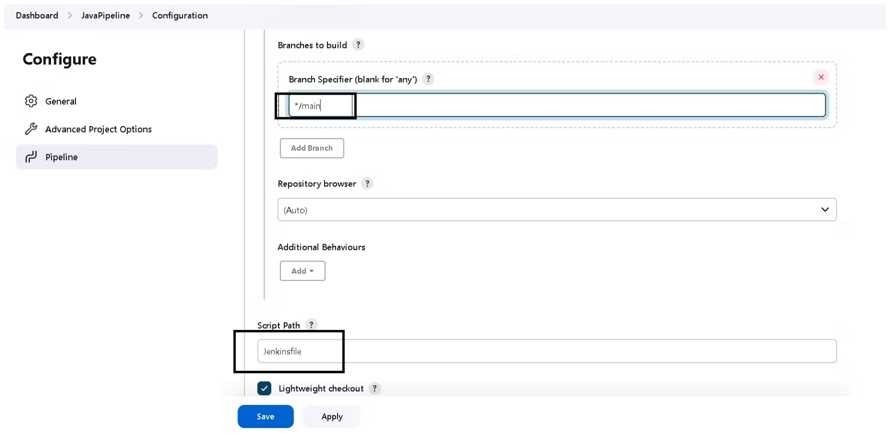
SCM: Git

Repositories

Repository URL: https://github.com/Sunilmargale/Devops-CI-CD-Project.git

Credentials: - none -

Also, enter the branch name in **Branches to build** and the Jenkinsfile name in **Script Path** and click **Save.** Finally, Click **Build Now** to run the pipeline.


Dashboard > JavaPipeline > Configuration

Configure

Branches to build

Branch Specifier (blank for 'any'): */main

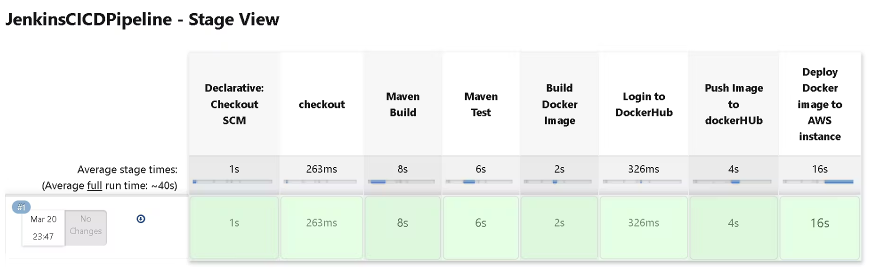
Repository browser: (Auto)

Additional Behaviours

Script Path: Jenkinsfile

Lightweight checkout ✓

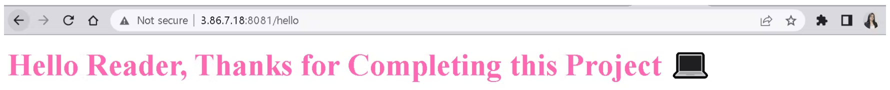
The Jenkins pipeline run successfully and the docker image got deployed to the EC2 instance.


JenkinsCICDPipeline - Stage View

	Declarative: Checkout SCM	checkout	Maven Build	Maven Test	Build Docker Image	Login to DockerHub	Push Image to dockerHUb	Deploy Docker image to AWS instance
Average stage times: (Average full run time: ~40s)	1s	263ms	8s	6s	2s	326ms	4s	16s
#1 Mar 20 23:47 No Changes	1s	263ms	8s	6s	2s	326ms	4s	16s

Access your Java Application in the browser using the public IP of the AWS instance at port 8081

publicIP:8081/hello


Not secure | 3.86.7.18:8081/hello

Hello Reader, Thanks for Completing this Project 💻