ASSIGNMENT 3

# Name: Faizan Ahmad

# Roll No: 221434448

# CSCS 351 Section: A

# Submitted to Professor Dr. Saad Bin Saleem

# Jenkins Installation

Stage 1) Got to https://www.jenkins.io/download/and select the stage.

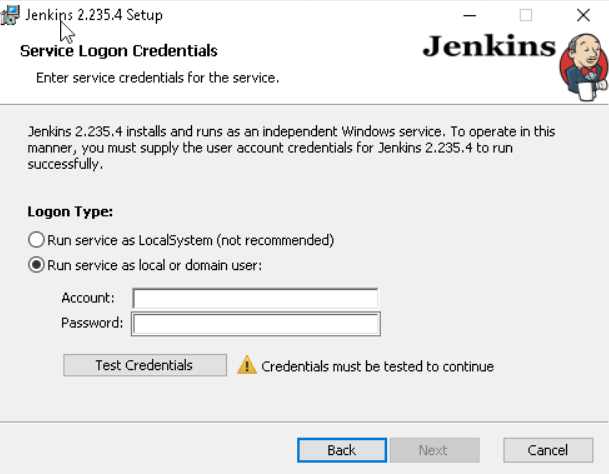
Stage 2) Go to download area and unfasten the bundle and run it.

Stage 3) In the Jenkin Setup screen, click Next.

Stage 4) Choose where you need to have the Jenkins occurrence introduced (default area is C:\Program Files (x86)\Jenkins), then, at that point, click on Next button.

Stage 5) Click on the Install button.

Stage 6) Once introduce is finished, click Finish.



# Creating CI/CD pipeline & sharing the screenshots of right configurations & sharing the screenshots of running pipeline successfully

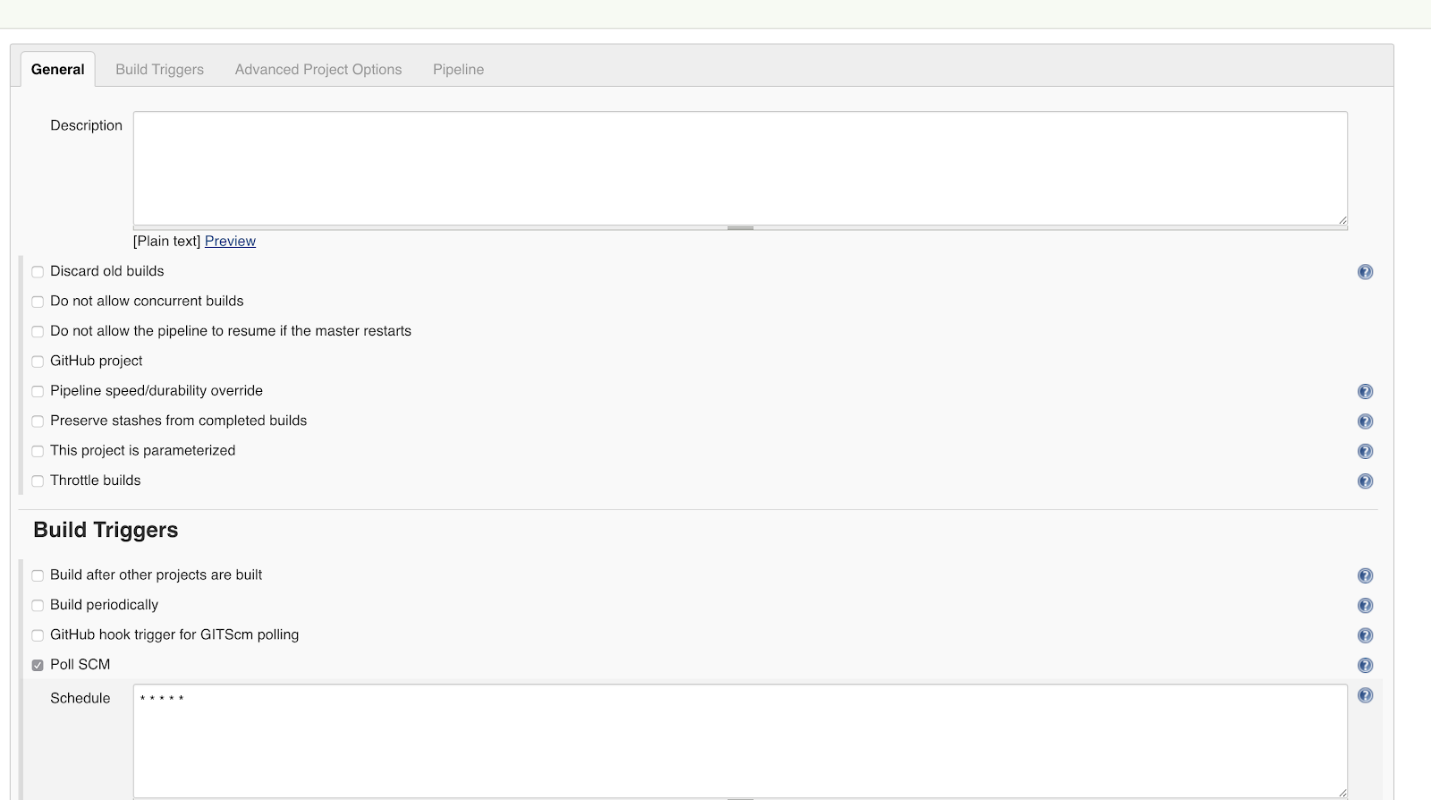
Stage 1: Execute Jenkins as a Java parallel.

Stage 2: Create another Jenkins work.

Stage 3: Create a pipeline work.

Stage 4: Configure and execute a pipeline work through an immediate content.

Stage 5: Configure and execute a pipeline work with SCM.



The settings that we changed are:

• We involved the Poll SCM as the form trigger; setting this choice trains Jenkins to really look at the Git store on an intermittent premise (consistently as shown by \* \* \* \* \*). If the repo has changed since the last survey, the occupation is set off.

• In the actual pipeline, we determined the archive URL and the qualifications. The branch is ace.

• We are adding all the work's code in a Jenkinsfile that is put away in a similar vault as the code.

Stage 6: Configure Jenkins Credentials For GitHub

Go to/accreditations/store/framework/area/\_/newCredentials and add the qualifications to the two targets. Ensure that you give a significant ID and depiction to each in light of the fact that you'll reference them later.

Stage 7: Create the JenkinsFile

The Jenkinsfile educates Jenkins about how to fabricate, test, dockerize, distribute, and convey our application.

**Jenkinsfile**

pipeline {

specialist any

climate {

vault = "magalixcorp/k8scicd"

GOCACHE = "/tmp"

}

stages {

stage('Build') {

specialist {

docker {

picture 'golang'

}

}

steps {

// Make our venture catalog.

sh 'cd ${GOPATH}/src'

sh 'mkdir - p ${GOPATH}/src/hi world'

// Duplicate all records in our Jenkins work area to our task registry.

sh 'cp - r ${WORKSPACE}/\* ${GOPATH}/src/hi world'

// Assemble the application.

sh 'go form'

}

}

stage('Test') {

specialist {

docker {

picture 'golang'

}

}

steps {

// Make our venture index.

sh 'album ${GOPATH}/src'

sh 'mkdir - p ${GOPATH}/src/hi world'

// Duplicate all records in our Jenkins work area to our task registry.

sh 'cp - r ${WORKSPACE}/\* ${GOPATH}/src/hi world'

// Eliminate reserved test results.

sh 'go clean - reserve'

// Run Unit Tests.

sh 'go test ./... - v - short'

}

}

stage('Publish') {

climate {

registryCredential = 'dockerhub'

}

steps{

script {

def appimage = docker.build vault + ":$BUILD\_NUMBER"

docker.withRegistry( '', registryCredential ) {

appimage.push()

appimage.push('latest')

}

}

}

}

stage ('Deploy') {

steps {

script{

def image\_id = vault + ":$BUILD\_NUMBER"

sh "ansible-playbook playbook.yml - - extra-vars \"image\_id=${image\_id}\""

}

}

}

}

}

The record is more straightforward than it looks. Fundamentally, the pipeline contains four phases:

Construct is where we fabricate the Go parallel and guarantee that everything seems OK in the form cycle.

Test is where we apply a basic UAT test to guarantee that the application functions true to form.

Distribute, where the Docker picture is fabricated and pushed to the library. From that point onward, any climate can utilize it.

Send, here Ansible is summoned to contact Kubernetes and apply the definition records.

**Testing Our CI/CD Pipeline**

The last piece of this is where we really put our work under a magnifying glass. We will commit our code to GitHub and guarantee that our code travels through the pipeline until it arrives at the bunch:

Add our documents: git add \*

Commit our changes: git commit - m "Beginning commit"

Push to GitHub: git push

On Jenkins, we can either trust that the work will get set off consequently, or we can simply tap on "Form Now".

On the off chance that the occupation succeeds, we can look at our sent application utilizing the accompanying command:

kubectl get nodes -o wide

Stage 1 can be named "Construct," "Assemble Information," or whatever, and a comparable thought is applied for the other stage blocks. "Step" basically expresses what to execute, and this can be a straightforward print order (e.g., reverberation "Hi, World"), a program-execution order (e.g., java HelloWorld), a shell-execution order (e.g., chmod 755 Hello), or some other order — for however long it is perceived as an executable order through the Jenkins climate. The Jenkins pipeline is given as a systematized script ordinarily called a Jenkinsfile, albeit the record name can be unique. Here is an illustration of a straightforward Jenkins pipeline record.

**Illustration of Jenkins pipeline script**

pipeline {

stages {

stage("Build") {

steps {

/ Simply print a Hello, Pipeline to the control center

reverberation "Hi, Pipeline!"

/ Gather a Java document. This requires JDKconfiguration from Jenkins

javac HelloWorld.java

/ Execute the incorporated Java parallel called HelloWorld. This requires JDK design from Jenkins

java HelloWorld

/ Executes the Apache Maven orders, clean then bundle. This requires Apache Maven

design from Jenkins

mvn clean bundle ./HelloPackage

/ List the records in current catalog way by executing a default shell order

sh "ls - ltr"

}

}

/ What's more, next stages to characterize further...

} / End of stages

} / End of pipeline

