This solution is tested on cent 7.x platform and requires one to be comfortable with Linux environment as align with our devops course pre-requisite.

PS: This document will work with the Jenkins running instance with below state.

JAVA\_HOME,MAVEN\_HOME,GIT configuration must be done in global tool configuration. And Jenkins must be able to pull code from git and able to do maven build successfully.

And requires one target machine where docker must be install and running.(even this docker installation one can do with ansible or puppet or any other CM tools)

On target machine run below command

sudo usermod -aG docker ec2-user # I am using ec2-user for target machine

service docker restart

Till here it was pre-requisite in order to complete “Jenkins CI-CD scripted Pipeline”

Solution document for Jenkins CI-CD Pipeline script

Step 1: Install and start docker service on same Jenkins host machine.

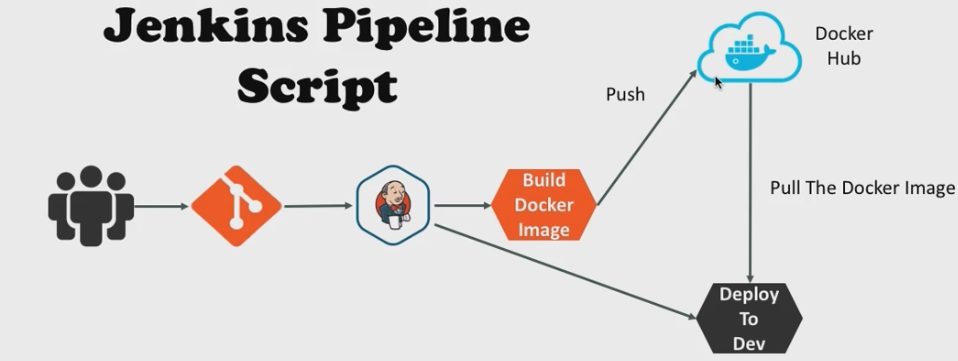
Run below command to ensure Jenkins can use docker service

sudo usermod -aG docker jenkins

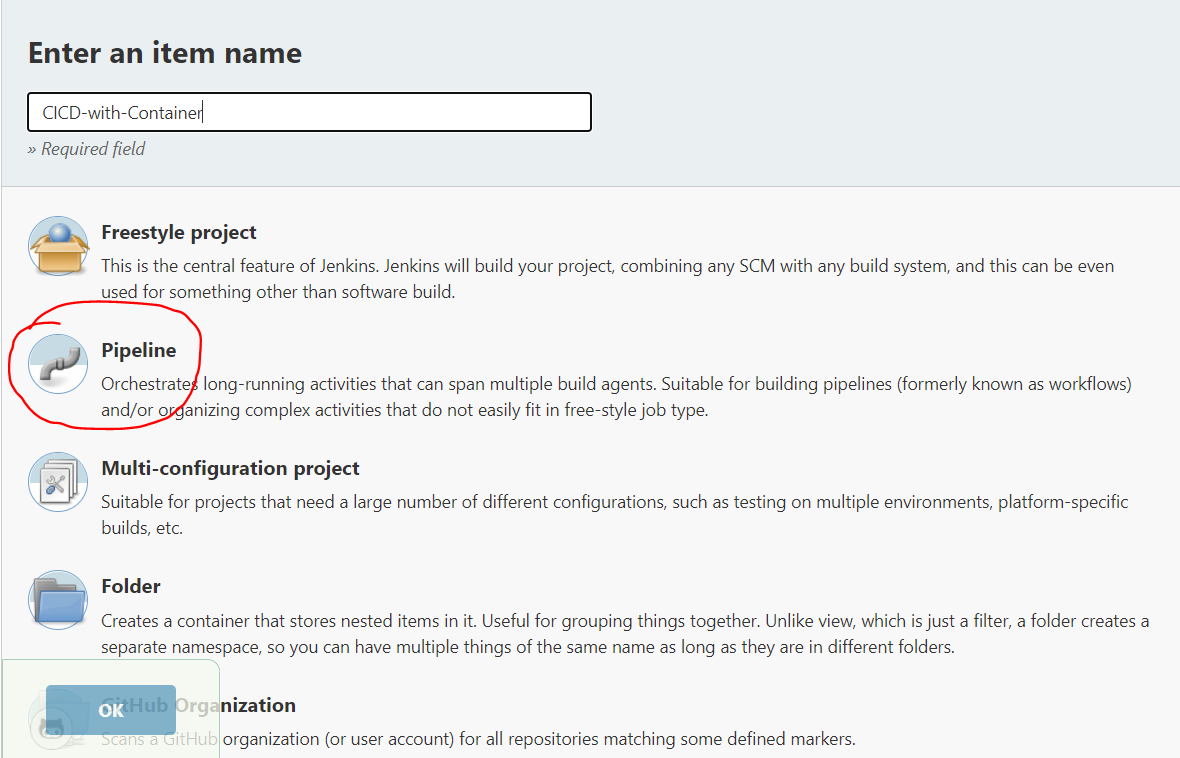
service docker restart

service jenkins restart

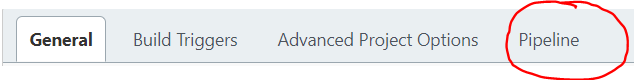
Below diagram depicts the flow of the solution.



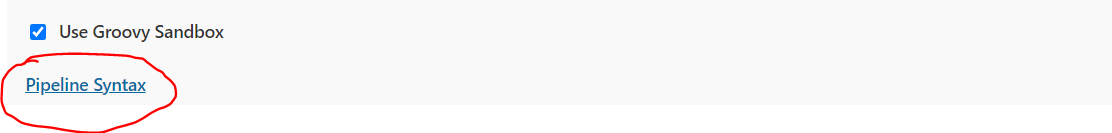
Step 2: Create a new project in Jenkins name as per your preference and type of project must be of pipeline type. One can reference below image.



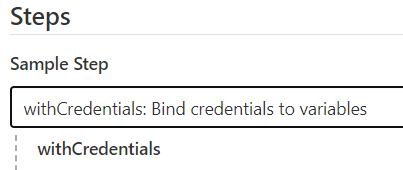
Step 3: Under project configuration select pipeline. One can reference below image.



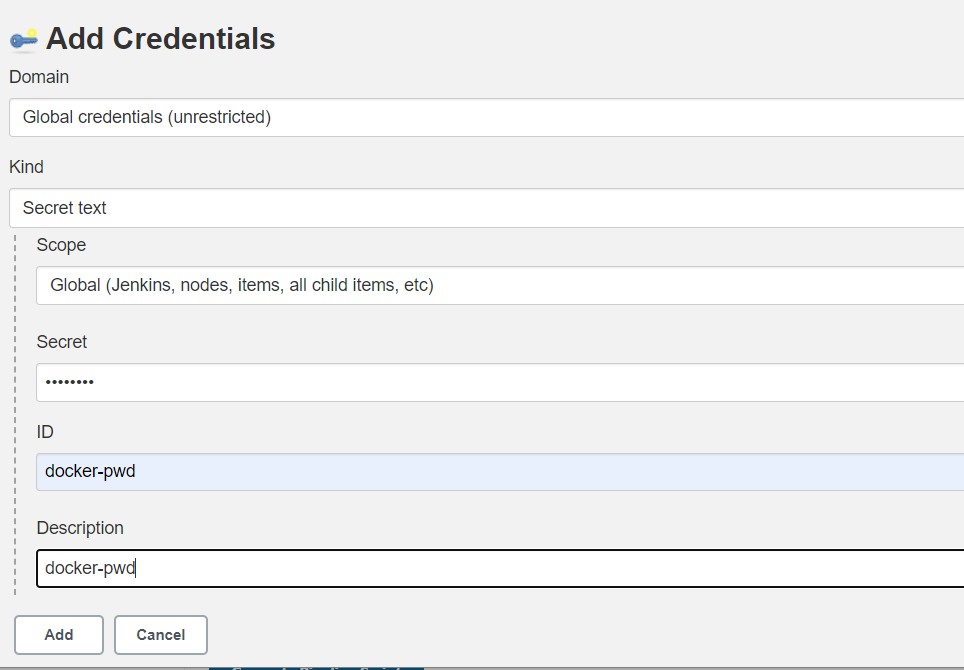
Then click to pipeline syntax it will open a page in new tab. One can reference below image.



Go to new page select below option from Sample step drop down, this step is require for bind docker hub credentials password must not be a plan text.



Click add >> secret text then type variable name as “dockerhubPWD” and click add >> jenkins. And follow below image.



Here from kind drop down select secret text and value of Secret field is your dockerhub password

ID and Description one can put same docker-pwd. Now click to Add. Now click to generate pipeline script it will give one script block like below.

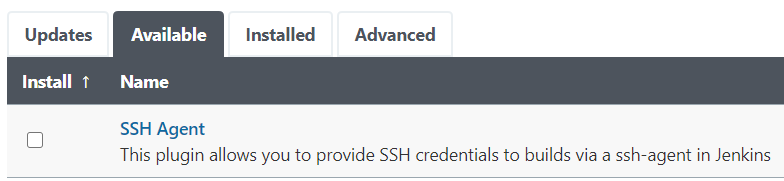
withCredentials([string(credentialsId: 'docker-pwd', variable: 'dockerhubPWD')]) {

// some block

}

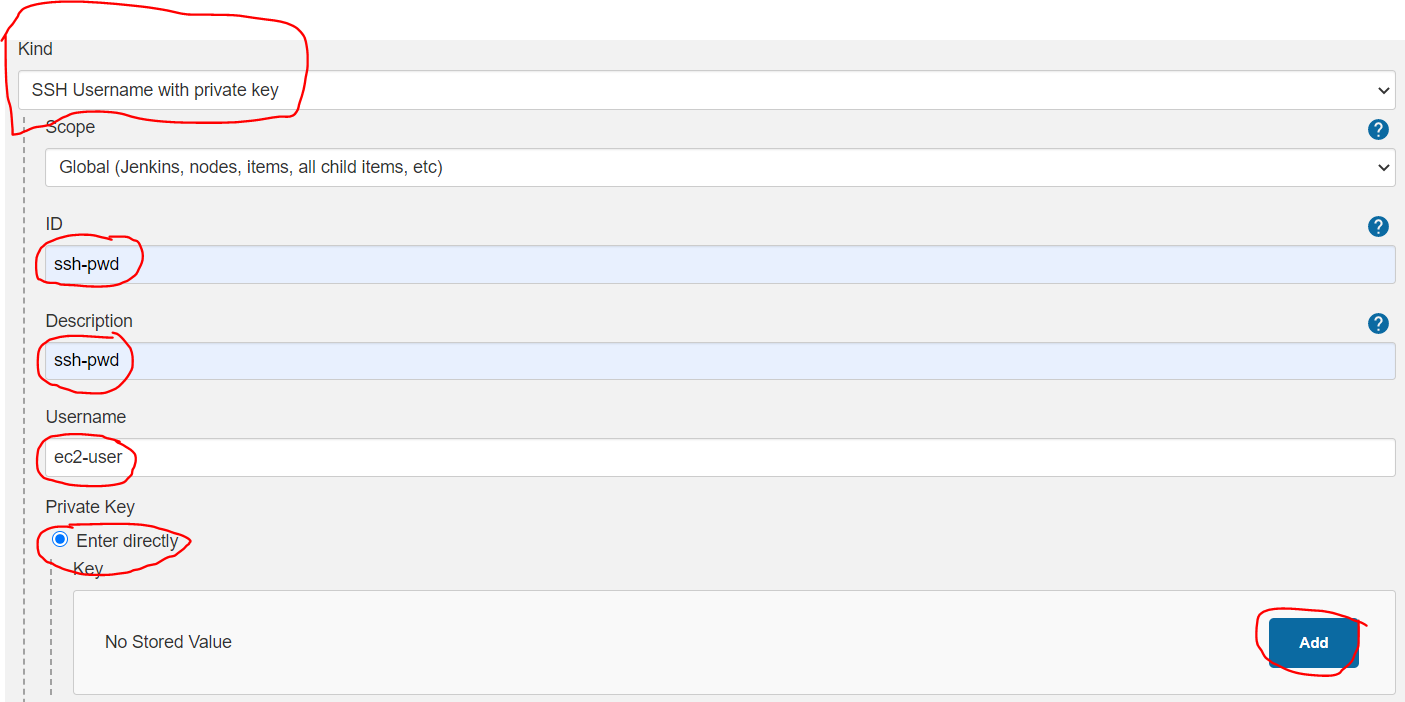
If above line appears then we are done with dockerhub password assignment in a variable named as dockerhubPWD

Step 4: add ssh Agent plugin so that we can connect to target machine for actual deployment where our application will run as a container. Click to manage jenkins >> manage plugins >> select available tab and search for ssh Agent. One can follow below image



Now select above plugin and click to install without restart. Its time to back to second tab of jenkins where we have created variable for dockerhub. We need to create credential for target machine. Select sshAgent from sample drop down, if you don’t find this option even after installation of plugin refresh the page. sshAgent >> add >> jenkins

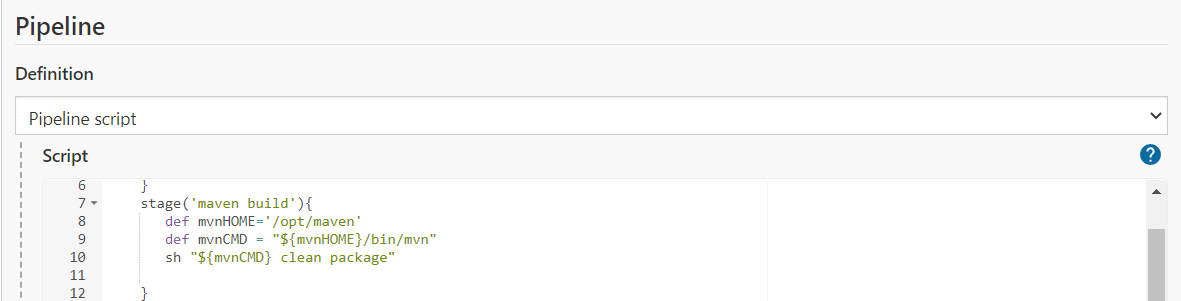
One can follow below image:



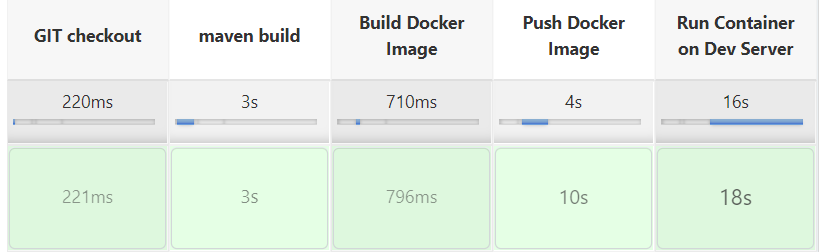
From kind select ssh username with private key, id ssh-pwd same for Description username your target machine user name in my case it is ec2-user then select enter directly radio button and click Add. Then carefully copy paste your .pem file content. Then click to generate pipeline. Now you are done with all required configuration.

Now we are all set to affix below script in pipeline script section of the job:

Reference below image:



Script Explanation: In jenkins scripted pipeline feature is a classic case of infrastructure as code implementation of devops. Here entire script is enclosed within node{}.Node has multiple stages where each and every stage will have specific task to complete below image is depicting stage name as their work



Below is working script one can consume with awareness, different variable name will Stall the script output

node{

stage('GIT checkout'){

git 'https://github.com/chandanchanchal/maven-project.git'

}

stage('maven build'){

def mvnHOME='/opt/maven'

def mvnCMD = "${mvnHOME}/bin/mvn"

sh "${mvnCMD} clean package"

}

stage('Build Docker Image'){

try{

sh 'docker rmi chandanchanchal/my-app:2.0.0'

}catch(err){echo: 'caught error: $err'}

sh 'docker build -t chandanchanchal/my-app:2.0.0 .'

}

stage('Push Docker Image'){

withCredentials([string(credentialsId: 'docker-pwd', variable: 'dockerhubPWD')]) {

sh "docker login -u chandanchanchal -p ${dockerHubPWD}"

}

sh 'docker push chandanchanchal/my-app:2.0.0'

}

stage('Run Container on Dev Server'){

def dockerRun = 'docker run -p 8080:8080 -d --name app1 chandanchanchal/my-app:2.0.0'

def dockerRm = 'docker rm -f app1'

def dockerRmImage= 'docker rmi -f chandanchanchal/my-app:2.0.0'

sshagent(['ssh-pwd']) {

try{

sh "ssh -o StrictHostKeyChecking=no ec2-user@172.31.16.17 ${dockerRm}"

sh "ssh -o StrictHostKeyChecking=no ec2-user@172.31.16.17 ${dockerRmImage}"

}catch(err){echo: 'caught error: $err'}

sh "ssh -o StrictHostKeyChecking=no ec2-user@172.31.16.17 ${dockerRun}"

}

}

}