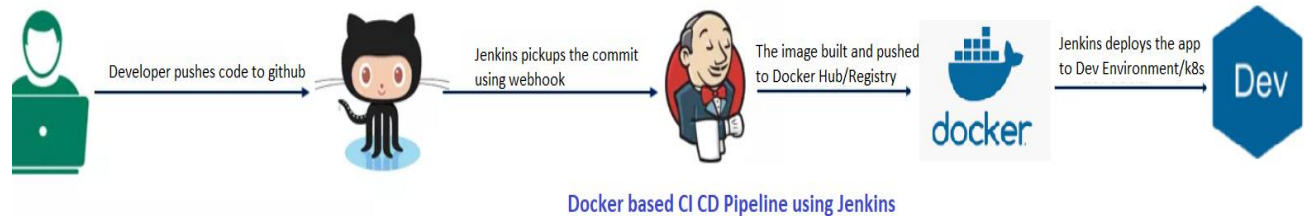


Jenkins-Docker Continuous Integration & Continuous Deployment Pipeline



1. Create Jenkins VM and install all the suggested plugins. **(VM1-Jenkins)**
2. Under Available Manage Plugins section, search for docker, there are multiple Docker plugins, docker compose, docker build plugins. Please install all docker plugins. **(VM1-Jenkins)**
3. Install Docker on Jenkins instance so that we can create docker images. **(VM1-Jenkins)**


```
yum install docker  
service docker start  
systemctl enable docker
```
4. Configure JDK and Maven. **(VM1-Jenkins)**
5. Since we will perform some operations such as checkout codebase and pushing an image to Docker Hub, we need to define the Docker Hub Credentials. **(VM1-Jenkins)**

These definitions are performed under Jenkins Home Page -> Credentials -> Jenkins-> Global credentials -> Add Credentials menu and provide id and password and update ID as "dockerHubAccount"

Jenkins > Credentials > System > Global credentials (unrestricted) >

[Back to credential domains](#)

[Add Credentials](#)

Kind: Username with password

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

Username: pkw0301

Password:

ID: dockerHubAccount

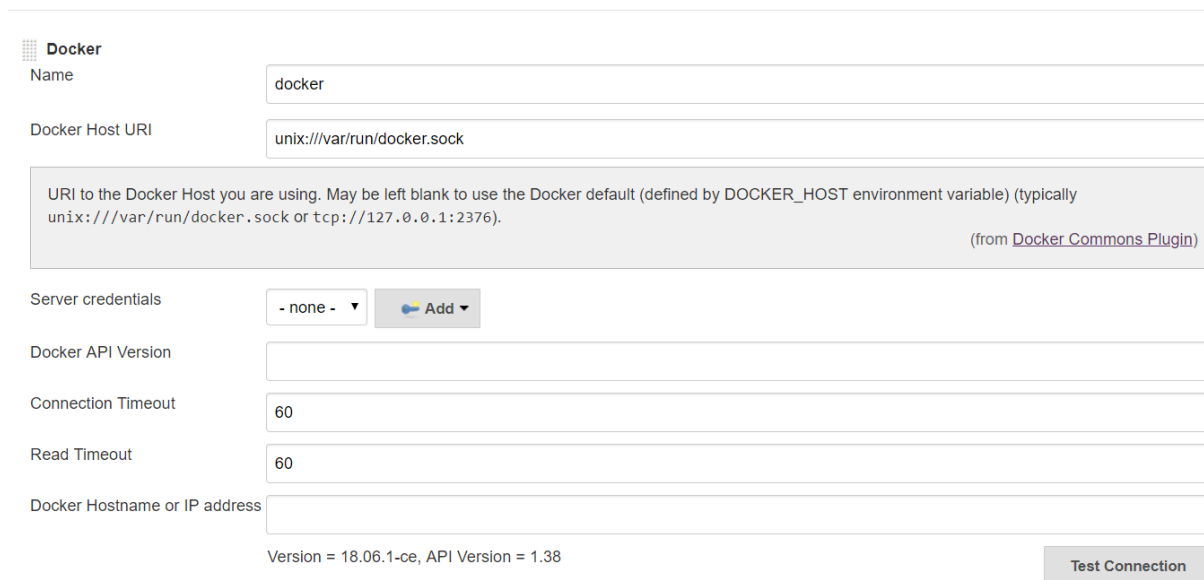
6. Select **Configure System** to access the main Jenkins settings. **(VM1-Jenkins)**

At the bottom, there is a dropdown called **Add a new cloud**. Select **Docker** from the list.
Now we need docker host uri, the docker.sock file is owned by root and does not allow write permissions by other. You have to make it such that Jenkins can read/write to that socket file when mounted.

To enable the docker host URL you need to change permission on your Jenkins ec2 **instance**.

chmod 777 /var/run/docker.sock

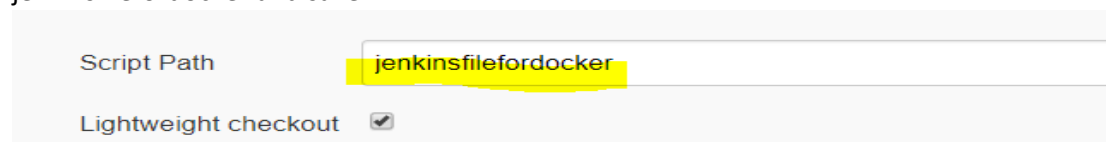
Once you change permission then at dashboard, under Docker Host URI put **unix:///var/run/docker.sock**
And click on Test Connection, it will display installed docker and API version.



7. We have installed docker on Jenkins instance, now we are good to build docker images **(VM1-Jenkins)**

Create a **Pipeline** Jenkins job, you can name it as docker-ci-cd

Source Code URL: <https://github.com/prakashk0301/maven-project/> and Jenkinsfile name as jenkinsfilefordocker and save.



8. From Pipeline Syntax->select git under Sample Step. Provide all the details. **(VM1-Jenkins)**

Jenkins > docker-cdci > Pipeline Syntax

[Back](#)
[Snippet Generator](#)
[Declarative Directive Generator](#)
[Declarative Online Documentation](#)
[Steps Reference](#)
[Global Variables Reference](#)
[Online Documentation](#)
[Examples Reference](#)
[IntelliJ IDEA GDSDL](#)

Overview

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step **Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

Steps

Sample Step

Repository URL

Branch

Credentials [Add](#)

☒ Include in polling?

☒ Include in changelog?

[Generate Pipeline Script](#)

git 'https://github.com/prakashk0301/maven-project'

- Open Github repo and create a file called **jenkinsfilefordocker** and paste Pipeline syntax content and commit your changes. **(VM1-Jenkins)**

[Create new file](#) [Upload files](#) [Find File](#) [Clone or download](#)

```
node{
  stage('scm checkout') {
    git 'https://github.com/prakashk0301/maven-project'
  }
}
```

```
1
2 node{
3   stage('scm checkout') {
4     git 'https://github.com/prakashk0301/maven-project'
5   }
6
7
8 }
```

- Save and build your job (just to verify)

- In case of multiple branches and if you want to build through to any specific branch then

```

1  node{
2      stage ('scm checkout') {
3          git 'https://github.com/prakashk0301/maven-project'
4      }
5
6      stage('Checkout to different branch') {
7          sh "git checkout try-docker"
8      }
9  }
10
11
12 }

```

12. It's time to build the job and create a package. From Pipeline syntax search for **shell script**, and Create a script for maven package goal, and add script in docker-jenkinsfile file and build.

Steps

Sample Step

Shell Script

[Generate Pipeline Script](#)

sh label: "", script: 'mvn clean package'

```

node{

    stage ('scm checkout') {

        git ('https://github.com/prakashk0301/maven-project')

    }

    stage ('Checkout to different branch') {

        sh "git branch -r"

        sh "git checkout master"

    }

    stage ('package stage') {

        sh label: "", script: 'mvn clean package '

    }

}

```

13. We can define a new stage for docker image creation. Our docker images will contains Jenkins job's artifact.

```
stage ('docker image build') {
    sh 'docker build -t pkw0301/prakash-app:1.0.0 .'
}
```

```
1  node{
2
3      stage ('scm checkout') {
4          git ('https://github.com/prakashk0301/maven-project')
5      }
6
7      stage('Checkout to different branch') {
8          sh "git branch -r"
9          sh "git checkout try-docker"
10     }
11     stage ('package stage') {
12         sh label: '', script: 'mvn clean package '
13     }
14     stage ('docker image build') {
15         sh 'docker build -t pkw0301/prakash-app:1.0.0 .'
16     }
17
18 }
19 }
```

In order to copy Jenkins job artifact we have to write a Dockerfile, please keep it with your source code.

```
FROM tomcat:8
```

```
COPY /webapp/target/*.war /usr/local/tomcat/webapps/
```

server	file created
webapp	Update index.jsp
.gitignore	file created
Dockerfile	Create Dockerfile
README.md	file created
jenkinsfilefordocker	Update and rename.
pom.xml	Update pom.xml

14. Save and build job, ssh your Jenkins instance and verify Docker images.(If you get any permission denied error then please follow below commands and re-run your job) **(VM1-Jenkins)**

```
chmod 777 /var/run/docker.sock
docker images
```

15. Push Docker image to Dockerhub/Registry. To do that we have to generate a script.
Pipeline syntax-> select **withcredentials** from drop down -> secret text - > select **dockerHubaccount** -> generate script **(VM1-Jenkins)**

Steps

Sample Step | withCredentials: Bind credentials to variables

Secret values are masked on a best-effort basis to prevent accidental disclosure. See the inline help for details and usage guidelines.

Bindings

||| **Secret text**

Variable

Credentials

```
withCredentials([string(credentialsId: 'dockerhubaccount', variable: '')]) {
  // some block
}
```

Add stage to push docker images

```
stage ('Push Docker image to DockerHub') {
  withCredentials([string(credentialsId: 'dockerhubaccount', variable: 'dockerhubaccount')]) {
    sh "docker login -u pkw0301 -p ${dockerhubaccount}"
  }
  sh 'docker push pkw0301/prakash-app:1.0.0'
}
```

Please provide your dockerhub id during login command, so that you also can login and push images. Once you add stage, build you job. Verify docker images by login to dockerhub.

- Now we have pushed docker images to dockerhub, In order to deploy these images to Development/QA/Prod environment, we need instances

Create an ec2 instance for Development environment and install docker. **(VM2-Dev)**

```
yum install docker
service docker start
systemctl enable docker
chmod 777 /var/run/docker.sock
```

From Pipeline syntax select sshagent->Add->Jenkins->

Kind: SSH Username with private key

ID: deploy-to-dev-docker

Username: ec2-user

Private key: please paste pem key of dev instance

Sample Step **sshagent SSH Agent**

```

node {
  sshagent (credentials: ['deploy-dev']) {
    sh 'ssh -o StrictHostKeyChecking=no -l cloudbees 192.168.1.106 uname -a'
  }
}

```

Multiple credentials could be passed in the array but it is not supported using Snippet Generator.

(from [SSH Agent Plugin](#))

ec2-user (deploy-to-dev-docker) **Add**

☐ Ignore missing credentials

Generate Pipeline Script

```

sshagent(['deploy-to-dev-docker']) {
  // some block
}

```

Update your Jenkinsfile and add a new stage “deploy-to-dev”

```

stage ('Deploy to Dev') {

    def dockerRun = 'docker run -d -p 9000:8080 --name my-tomcat-app pkw0301/prakash-app:1.0.0'

    sshagent(['deploy-to-dev-docker']) {

        sh "ssh -o StrictHostKeyChecking=no ec2-user@172.31.46.1 ${dockerRun}"

    }

}

```

Save your changes and build.

You can access docker based application by the following URL.

<VM2 dev public ip>:9000/webapp

Done 😊

(You also can refer my jenkinsfile for docker ci/cd

Github URL: <https://github.com/prakashk0301/maven-project/tree/ci-cd-with-docker>

Branch: ci-cd-with-docker)