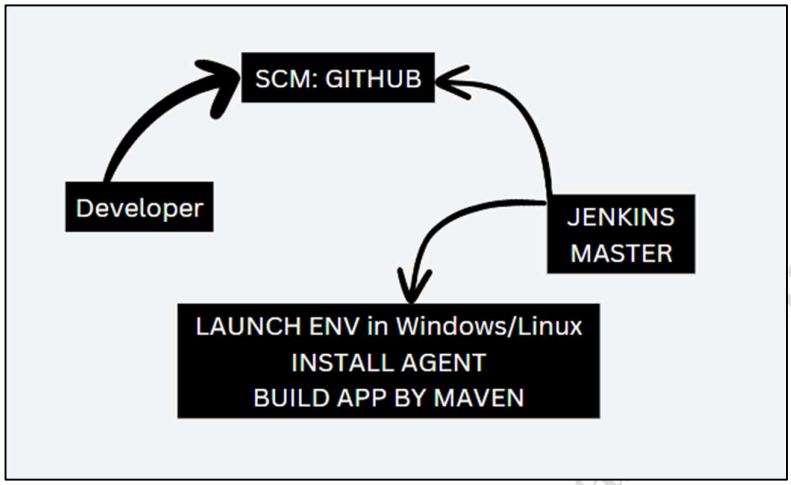
MASTER SLAVE ARCHITECTURE IN JENKINS USING DOCKER

In the Previous Document, I have explained about how we can set up the Master Slave Architecture in Jenkins using Linux Machine and Windows Machine. But in this document, I am going to show about How we can set up the Master Slave Architecture in Jenkins using Docker. Before applying this setting, let's understand why we need to this setup?



In this scenario, the process begins with a developer writing code and pushing it to GitHub. Once the code is on GitHub, Jenkins comes into play. Jenkins retrieves the code from GitHub and prepares the environment on a Windows or Linux system. This environment setup usually takes around 5 minutes. Following this, agent software needs to be installed on the Windows/Linux operating system. This software enables communication with the Jenkins Master. This installation process also takes approximately 5 minutes. After this setup, the developer's task enters a queue and awaits its turn. The entire environment setup process initially consumes about 10 minutes.

In practical situations, developers frequently perform testing, which necessitates a fresh environment for accurate results. Relying on an outdated environment can lead to issues when the application is in production.

Another issue arises when agents are created manually. If developers don't have code to test or build at that moment, the operating system remains idle. This becomes a concern especially if the OS is hosted in the cloud, as the developer continues to incur costs for unused resources.

To address these challenges, Jenkins offers a solution through Dynamic Slaves. These slaves are generated on-the-fly as new jobs arrive. This approach optimizes resource utilization and ensures that resources are allocated precisely when needed, thus reducing wastage and costs.

With Dynamic Slaves, I'll be using Docker as a tool to create a super quick setup, taking just 1 or 2 seconds. This setup will then help us build our job and put our application on this special "Container." This smart trick lets us bring our work to the market fast and without any hassle.

For Using Docker, I am going to Launch one OS and install Docker Engine on top of that OS.

```
[ec2-user@ip-172-31-46-152 ~]$ sudo su -
[root@ip-172-31-46-152 ~]# rpm -q docker
package docker is not installed
[root@ip-172-31-46-152 ~]# yum install docker -y
```

After installation, I am going to start the docker engine service and enable it. By using command.

systemctl enable docker --now

```
[root@ip-172-31-46-152 ~]# systemctl enable docker --now
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[root@ip-172-31-46-152 ~]# systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: disabled)
     Active: active (running) since Thu 2023-08-24 11:34:52 UTC; 3s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
    Process: 15344 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
    Process: 15359 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
   Main PID: 15365 (dockerd)
      Tasks: 7 (limit: 1114)
     Memory: 29.4M
        CPU: 282ms
     CGroup: /system.slice/docker.service
             └─15365 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536
```

And we can see that The Docker service is active and running.

Before Moving further, I am going to share some small information about docker. In Docker, while Launching the Container we need Image (For Example: Windows 11 ISO) From that Image we can create multiple OS with different name, and every OS has their OWN IP and Storage.

In Docker, if you want to launch new OS which is also known as **container**, for that we need 1 Image which we can pull from https://hub.docker.com. And form Here I am going to pull Ubuntu OS image. By using command: **docker pull ubuntu.**

```
[root@ip-172-31-46-152 ~]# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
b237fe92c417: Pull complete
Digest: sha256:ec050c32e4a6085b423d36ecd025c0d3ff00c38ab93a3d71a460ff1c44fa6d77
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
[root@ip-172-31-46-152 ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       latest    01f29b872827   2 weeks ago   77.8MB
```

And after applying the command Docker will pull the latest image from docker Hub and store in my local OS. and we can see this by using **docker images** command.

For Launching any container using docker we have command available.

docker run -it --name container-name Image-name: version. # -it means interactive terminal.

```
[root@ip-172-31-46-152 ~]# docker run -it --name os1 ubuntu
root@e67866fc3ae1:/# ps
    PID TTY          TIME CMD
      1 pts/0    00:00:00 bash
      9 pts/0    00:00:00 ps
root@e67866fc3ae1:/# cal
bash: cal: command not found
root@e67866fc3ae1:/# date
Thu Aug 24 11:46:29 UTC 2023
root@e67866fc3ae1:/#
```

If we don't mention the version, then it will launch the container with the latest version, and you can see that after the command executed then we came inside the container and inside the container I have run some commands and some commands are not available by default so we can install those commands.

In future we will require multiple software like git, maven which we can download manually every time if we do that then again it will take lots of time so it's better to go for Docker file.

A Dockerfile is like a recipe or set of instructions that you give to Docker, a technology used to create and manage containers. A Dockerfile tells Docker how to build a container image, which is a snapshot of an environment where your application can run.

Think of it as building a customized computer setup that has everything your application needs to run. The Dockerfile contains a series of commands that Docker follows to assemble this setup. These commands might include:

1. **Starting Point:** You specify a base image to begin with. This base image could be an operating system like Linux.
2. **Copying Files:** You can copy files from your local computer into the container image.
3. **Running Commands:** You can execute commands like installing software, setting up configurations, and more, just like you would on a regular computer.
4. **Setting Environment:** You can define environment variables that your application will use.
5. **Exposing Ports:** If your application needs to communicate over specific network ports, you can specify that in the Dockerfile.
6. **Starting Application:** You can indicate what command or script should be run when the container starts.
7. **Cleanup and Optimization:** You can perform clean-up steps to reduce the size of the image and remove temporary files.

Once the Dockerfile is ready, you use the `**docker build**` command to create a container image based on its instructions. This image can then be used to create containers that run your application in a consistent and isolated environment. The benefit of Dockerfile is that they capture all the setup and configuration steps needed for your application to work, making it easy to reproduce the same environment on different machines or platforms. This is particularly helpful in development, testing, and deployment scenarios.

Now I am going to create a Docker file:

```
# Use the Ubuntu 18.04 image as the base
FROM ubuntu:18.04

# Update package lists and install necessary packages
RUN apt-get update && \
    DEBIAN_FRONTEND=noninteractive apt-get -y upgrade && \
    DEBIAN_FRONTEND=noninteractive apt-get install -q -y git openssh-server openjdk-
11-jdk maven && \
    apt-get clean

# Configure SSH server and create necessary directories
RUN sed -i 's|session    required     pam_loginuid.so|session    optional
pam_loginuid.so|g' /etc/pam.d/sshd && \
    mkdir -p /var/run/sshd && \
    useradd -m -d /home/jenkins -p $(openssl passwd -1 jenkins) jenkins && \
    mkdir -p /home/jenkins/.m2 && \
    mkdir -p /home/jenkins/.ssh

# Change ownership of directories
RUN chown -R jenkins:jenkins /home/jenkins/.m2/ && \
    chown -R jenkins:jenkins /home/jenkins/.ssh/

# Expose SSH port
EXPOSE 22

# Start SSH server
CMD ["/usr/sbin/sshd", "-D"]
```

This code sets up a Docker image with Ubuntu 18.04 as the starting point. It does the following:

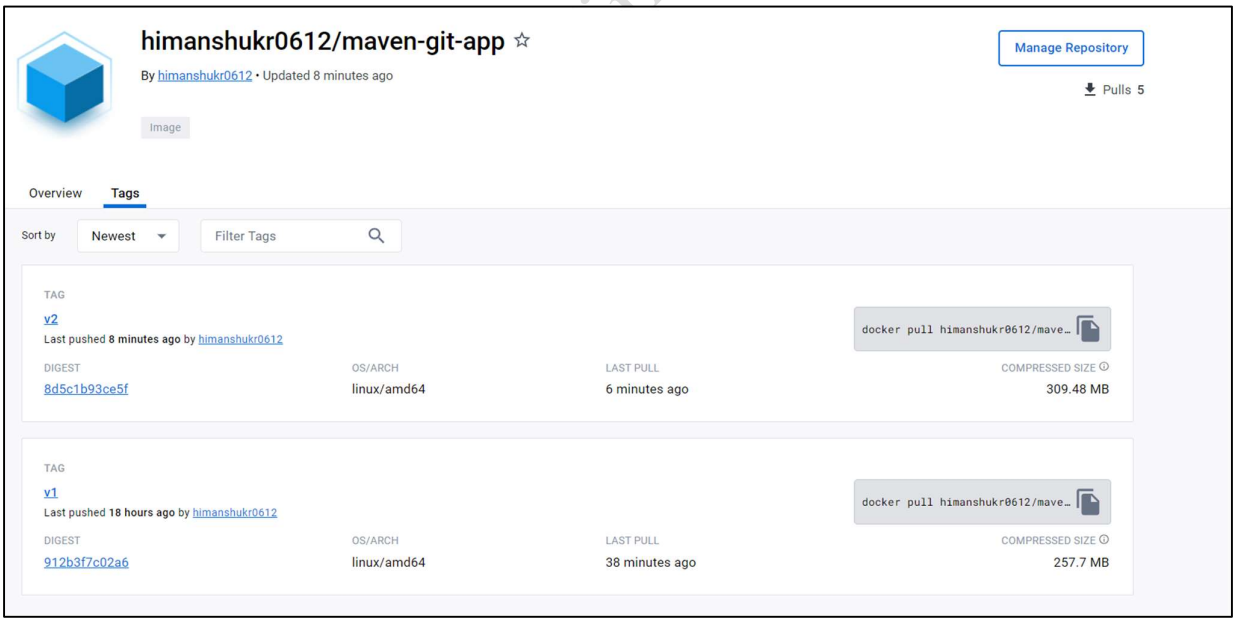
1. Uses Ubuntu 18.04 as the base image.
2. Updates package information and installs necessary packages like Git, SSH server, OpenJDK 11, and Maven.
3. Configures the SSH server, creating essential directories, and creating a user named "jenkins" with a specific password.
4. Changes ownership of user directories to ensure proper access.
5. Exposes port 22 for SSH connections.
6. Starts the SSH server, which enables remote access to the container.

I am going to build one Image which I am going to use to build my Maven application.

docker build himanshukr0612/maven-git-app:v2 .

```
Step 3/6 : RUN sed -i 's|session    required     pam_loginuid.so|session    optional     pam_loginuid.so|g' /etc/pam.d/sshd &&     mkdir -p /var/run/sshd &&     useradd -m -
p $(openssl passwd -1 jenkins) jenkins &&     mkdir -p /home/jenkins/.m2 &&     mkdir -p /home/jenkins/.ssh
 ---> Running in 1106b4469990
Removing intermediate container 1106b4469990
 ---> dedcd8883aa2
Step 4/6 : RUN chown -R jenkins:jenkins /home/jenkins/.m2/ &&     chown -R jenkins:jenkins /home/jenkins/.ssh/
 ---> Running in 54372abb770a
Removing intermediate container 54372abb770a
 ---> 3d0d9b5a5110
Step 5/6 : EXPOSE 22
 ---> Running in a182023bafce
Removing intermediate container a182023bafce
 ---> d8936f6a706b
Step 6/6 : CMD ["/usr/sbin/sshd", "-D"]
 ---> Running in f1bee122b790
Removing intermediate container f1bee122b790
 ---> 26c1d913d7db
Successfully built 26c1d913d7db
Successfully tagged himanshukr0612/maven-git-app:v2
[root@ip-172-31-46-152 ~]#
[root@ip-172-31-46-152 ~]# docker push himanshukr0612/maven-git-app:v2
The push refers to repository [docker.io/himanshukr0612/maven-git-app]
```

After this I uploaded this image in my Docker Hub.



Now Let's try to build the Application by using this image.

```
[root@ip-172-31-46-152 ~]# docker images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
himanshukr0612/maven-git-app  v2        26c1d913d7db   10 minutes ago   717MB
himanshukr0612/maven-git-app  v1        9f07bb269536   19 hours ago     673MB
ubuntu                        18.04     f9a80a55f492   2 months ago     63.2MB
ubuntu                        16.04     b6f507652425   24 months ago    135MB
[root@ip-172-31-46-152 ~]#
```

Currently I have this image available in my Local OS. Now, I am going to launch 1 Container.

For launching the Container, I have run the command for that but here I am not able to execute other commands. So here I am just coming back to my Base OS by pressing (**ctrl + p + q**).

```
[root@ip-172-31-46-152 ~]# docker run -it --name app-test himanshukr0612/maven-git-app:v2
```

Now let's see what is going on. When I launch this container, this container executes command SSHD service in the background.

```
[root@ip-172-31-46-152 ~]# docker history himanshukr0612/maven-git-app:v2
IMAGE          CREATED          CREATED BY                                      SIZE      COMMENT
26c1d913d7db   12 minutes ago   /bin/sh -c #(nop)  CMD ["/usr/sbin/sshd" "-D…   0B
d8936f6a706b   12 minutes ago   /bin/sh -c #(nop)  EXPOSE 22                    0B
3d0d9b5a5110   12 minutes ago   /bin/sh -c chown -R jenkins:jenkins /home/je…   0B
dedcd8883aa2   12 minutes ago   /bin/sh -c sed -i 's|session    required    …   401kB
00d35e098c6a   12 minutes ago   /bin/sh -c apt-get update &&     DEBIAN_FRON…   654MB
f9a80a55f492   2 months ago     /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      2 months ago     /bin/sh -c #(nop) ADD file:3c74e7e08cbf9a876…   63.2MB
<missing>      2 months ago     /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      2 months ago     /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      2 months ago     /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH     0B
<missing>      2 months ago     /bin/sh -c #(nop)  ARG RELEASE                  0B
[root@ip-172-31-46-152 ~]#
```

And while creating this image I have also created 1 User **jenkins** with the password **jenkins.** Now, I am trying to login with that user by using SSH. For this I need to know about the Container IP. which we can get by **docker inspect container-name**

```
[root@ip-172-31-46-152 ~]# docker inspect app-test
[
    {
        "Id": "d9b9aa655c1c0db1547e6ca02bfedb3287432c1bcc2e2aa604b9a0fe48c97783",
        "Created": "2023-08-25T06:53:10.584573096Z",
        "Path": "/usr/sbin/sshd",
        "Args": [
            "-D"
        ],
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
```

The IP of the Container is 172.17.0.2 and Now I am Login to this IP with jenkins user.

```
[root@ip-172-31-46-152 ~]# ssh -l jenkins 172.17.0.2                                  1
The authenticity of host '172.17.0.2 (172.17.0.2)' can't be established.
ED25519 key fingerprint is SHA256:ZagqDUfNyDeX0/uzIxEwva9KyV36Y4NwTLfYL2g0PxM.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes              2
Warning: Permanently added '172.17.0.2' (ED25519) to the list of known hosts.
jenkins@172.17.0.2's password:                                                        3
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 6.1.41-63.114.amzn2023.x86_64 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

$
```

1. Here You can See I used SSH and try to login with jenkins user on IP 172.17.0.2.
2. After that Here I need to manually type **Yes** for allowing connection through SSH.
3. Then I have eneterd password for jenkins which is jenkins which I created while creating this image.

Now I am in my Container and Here I have git and maven installed.

```
$ git --version
git version 2.17.1
$ mvn -version
Apache Maven 3.6.0
Maven home: /usr/share/maven
Java version: 11.0.19, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-amd64
Default locale: en_US, platform encoding: ANSI_X3.4-1968
OS name: "linux", version: "6.1.41-63.114.amzn2023.x86_64", arch: "amd64", family: "unix"
$
```

Now, I am going to download the Code from my GitHub where I have stored my Maven Application.

```
$ git clone https://github.com/Himanshu-kr-007/Java-Maven-App.git
Cloning into 'Java-Maven-App'...
remote: Enumerating objects: 28, done.
remote: Counting objects: 100% (28/28), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 28 (delta 2), reused 28 (delta 2), pack-reused 0
Unpacking objects: 100% (28/28), done.
$ ls
Java-Maven-App
```

Now I am going to Build this application by going inside the directory and then run the command mvn clean package.

```
$ cd Java-Maven-App
$ ls
Dockerfile  Jenkinsfile  README.md  jenkins  pom.xml  src
$ mvn clean package
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.inject.internal.cglib.core.$ReflectUtils$1 (file:/usr/share/maven/lib/guice.jar) to method java.lang.ClassLoader.defineClass(java.lang.String
,byte[],int,int,java.security.ProtectionDomain)
WARNING: Please consider reporting this to the maintainers of com.google.inject.internal.cglib.core.$ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.mycompany.app:my-app >----------------------
[INFO] Building my-app 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
```

When the Code is executed successfully then it generates the output in target folder.

```
[INFO] Building jar: /home/jenkins/Java-Maven-App/target/my-app-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.325 s
[INFO] Finished at: 2023-08-25T07:38:47Z
[INFO] ------------------------------------------------------------------------
$ ls
Dockerfile  Jenkinsfile  README.md  jenkins  pom.xml  src  target
$ cd target
$ ls
classes  generated-sources  generated-test-sources  maven-archiver  maven-status  my-app-1.0-SNAPSHOT.jar  surefire-reports  test-classes
$ java -jar my*.jar
Hello World!
```

Here we can see the application is Built Successfully, Now I am going to do this thing with the help of automation using Jenkins.

But if we want to use this Image by using Jenkins, then Jenkins needs to connect with Docker from another System and by chance Docker By default does not support other system to connect with them. For this we need to configure it in such a way that Jenkins will connect with themselves.

```
[root@ip-172-31-46-152 ~]# systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: disabled)
     Active: active (running) since Thu 2023-08-24 11:34:52 UTC; 1h 14min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
    Process: 15344 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
    Process: 15359 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
   Main PID: 15365 (dockerd)
      Tasks: 9 (limit: 1114)
     Memory: 427.5M
        CPU: 48.441s
     CGroup: /system.slice/docker.service
             └─15365 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536
```

Here I am going to edit the Configuration file of Docker. Which is present in the given location **/usr/lib/systemd/system/docker.service.**

```
[root@ip-172-31-46-152 ~]# vim /usr/lib/systemd/system/docker.service
[root@ip-172-31-46-152 ~]#
[root@ip-172-31-46-152 ~]#
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock $OPTIONS $DOCKER_STORAGE_OPTIONS $DOCKER_ADD_RUNTIMES
```

I've made changes to the configuration file located at /usr/lib/systemd/system/docker.service. Specifically, I've added -H tcp://0.0.0.0:4243. This change allows Jenkins to connect using any IP address on Port 4243. However, it's recommended to enhance security by specifying the IP of our Jenkins machine. This approach ensures that only connections from the specified IP are allowed, preventing unauthorized access.

```
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:4243 -H fd:// --containerd=/run/containerd/containerd.sock $OPTIONS $DOCKER_STORAGE_OPTIONS $DOCKER_ADD_RUNTIMES
```

After that, we need to reload the daemon and restart the docker service.

```
[root@ip-172-31-46-152 ~]# systemctl daemon-reload
[root@ip-172-31-46-152 ~]# systemctl restart docker
[root@ip-172-31-46-152 ~]#
```

Now I am going to remove all the Images and Container from My Machine.

```
[root@ip-172-31-46-152 ~]# docker ps -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS          PORTS     NAMES
3e4295090a90   himanshukr0612/maven-git-app:v2   "/usr/sbin/sshd -D"   20 minutes ago   Up 14 minutes   22/tcp    app-test
[root@ip-172-31-46-152 ~]# docker rm app-test
Error response from daemon: You cannot remove a running container 3e4295090a9063c1e25e6d6324a16e147924d5dc2e3cc9775d61c6d4c2480115. Stop the container before attempting removal or force remo
ve
[root@ip-172-31-46-152 ~]# docker stop app-test
app-test
[root@ip-172-31-46-152 ~]# docker rm app-test
app-test
```

For removing the container, we need to stop the container first then only we can remove it. And then removing the image

```
REPOSITORY                     TAG       IMAGE ID       CREATED              SIZE
himanshukr0612/maven-git-app   v2        26c1d913d7db   About an hour ago    717MB
himanshukr0612/maven-git-app   v1        9f07bb269536   20 hours ago         673MB
ubuntu                         18.04     f9a80a55f492   2 months ago         63.2MB
ubuntu                         16.04     b6f507652425   24 months ago        135MB
```

```
[root@ip-172-31-46-152 ~]# docker rmi himanshukr0612/maven-git-app:v2
Untagged: himanshukr0612/maven-git-app:v2
Untagged: himanshukr0612/maven-git-app@sha256:8d5c1b93ce5fe1164f16e27c794504c032235c9a10c78f9fe247acfc87eaf2a4
Deleted: sha256:26c1d913d7db16baf99ae71aab8ff6e572b0f79420c3417e92cec5d20147acfd
Deleted: sha256:d8936f6a706b41f831cff6d8889d48f4c7be0f7317490d7ee8eb5d4b73e3cace
Deleted: sha256:3d0d9b5a51103f51cf3153139d32da1f8aedfbec6a0607d04945a82c697f42b2
Deleted: sha256:cd9c0b20c299e923dc8084ba7e130afe5053c15180265d0e7d82a9b5387eabf3
Deleted: sha256:dedcd8883aa2c3fc6429b3e30a9b2dbbbb8f7fc3e7e90d1b7a8c837a258e26f5
Deleted: sha256:18ce8c21fc72313b9528554d1b7f0813d6db4dd5122946de0e8f36fb3a5deda3
Deleted: sha256:00d35e098c6af844a893a3098e6858dbf2dacef63d5a0a75ac9aabf1fb19f2e3
Deleted: sha256:59fcbb842cefa080eb6d29ca2b0e456d352f462d67210f15f42f2e14ce34c078
[root@ip-172-31-46-152 ~]# docker rmi himanshukr0612/maven-git-app:v1
Untagged: himanshukr0612/maven-git-app:v1
Untagged: himanshukr0612/maven-git-app@sha256:912b3f7c02a6a3e54d4bdc0e78f6c42283ee444fb49f98a87696d0e3faeed82d
Deleted: sha256:9f07bb269536373e3b90e48408f8de0b22050f437b3fe8f630b38d3e14e5e8ce
Deleted: sha256:b2865fc7099dbd2a55f64c5a0dae5632f778772e91447c04d8c51a987b3be30e
Deleted: sha256:070da7ce7f45d033b343fac362352fa54c8105e4c4361fb9bb8f3ccfcaaa960b
Deleted: sha256:537077e8610a95e828d9f87e6b4f058562e3e0a74da08c2d0a5e9365ec4d56dc
[root@ip-172-31-46-152 ~]# docker rmi ubuntu:18.04
Untagged: ubuntu:18.04
Untagged: ubuntu@sha256:152dc042452c496007f07ca9127571cb9c29697f42acbfad72324b2bb2e43c98
Deleted: sha256:f9a80a55f492e823bf5d51f1bd5f87ea3eed1cb31788686aa99a2fb61a27af6a
Deleted: sha256:548a79621a426b4eb077c926eabac5a8620c454fb230640253e1b44dc7dd7562
[root@ip-172-31-46-152 ~]# docker rmi ubuntu:16.04
Untagged: ubuntu:16.04
Untagged: ubuntu@sha256:1f1a2d56de1d604801a9671f301190704c25d604a416f59e03c04f5c6ffee0d6
Deleted: sha256:b6f50765242581c887ff1acc2511fa2d885c52d8fb3ac8c4bba131fd86567f2e
Deleted: sha256:0214f4b057d78b44fd12702828152f67c0ce115f9346acc63acdf997cab7e7c8
Deleted: sha256:1b9d0485372c5562fa614d5b35766f6c442539bcee9825a6e90d1158c3299a61
Deleted: sha256:3c0f34be6eb98057c607b9080237cce0be0b86f52d51ba620dc018a3d421baea
Deleted: sha256:be96a3f634de79f523f07c7e4e0216c28af45eb5776e7a6238a2392f71e01069
[root@ip-172-31-46-152 ~]#
```

```
[root@ip-172-31-46-152 ~]# docker images
REPOSITORY   TAG        IMAGE ID   CREATED   SIZE
[root@ip-172-31-46-152 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ip-172-31-46-152 ~]#
```

Now I don't have any image and container running in my OS.

Before doing the further setup, we need to allow the Inbound rule for port number 4243 in the Docker OS.

Inbound rules (3) | Manage tags | Edit inbound rules

Filter security group rules

Security group rule...	IP version	Type	Protocol	Port range	Source
sgr-05cb0e763e91bb4ce	IPv4	All traffic	All	All	0.0.0.0/0
sgr-0cfabd941d6b4b30e	IPv4	Custom TCP	TCP	4243	0.0.0.0/0
sgr-01cbd53be085c17...	IPv4	SSH	TCP	22	0.0.0.0/0

Now I am testing the connectivity, by running the command: **curl DockerIP:4243/version**

```
[root@ip-172-31-45-173 ~]# curl 13.234.136.29:4243/version
{"Platform":{"Name":""},"Components":[{"Name":"Engine","Version":"20.10.25","Details":{"ApiVersion":"1.41","Arch":"amd64","BuildTime":"2023-07-05T00:00:00.000
000000+00:00","Experimental":"false","GitCommit":"5df983c","GoVersion":"go1.19.9","KernelVersion":"6.1.41-63.114.amzn2023.x86_64","MinAPIVersion":"1.12","Os":
"linux"}},{"Name":"containerd","Version":"1.7.2","Details":{"GitCommit":"0cae528dd6cb557f7201036e9f43420650207b58"}},{"Name":"runc","Version":"1.1.7","Details
":{"GitCommit":"f19387a6bec4944c770f7668ab51c4348d9c2f38"}},{"Name":"docker-init","Version":"0.19.0","Details":{"GitCommit":"de40ad0"}}],"Version":"20.10.25",
"ApiVersion":"1.41","MinAPIVersion":"1.12","GitCommit":"5df983c","GoVersion":"go1.19.9","Os":"linux","Arch":"amd64","KernelVersion":"6.1.41-63.114.amzn2023.x8
6_64","BuildTime":"2023-07-05T00:00:00.000000000+00:00"}
[root@ip-172-31-45-173 ~]#
```

It will give the information about Docker version which is currently installed and here the connection is also established.

For Using Docker in Jenkins, we need to install the plugins for Docker in Jenkins.

Go to Manage Jenkins – Plugins – Search Docker in Available Plugins.

Updates
Available plugins
Installed plugins
Advanced settings

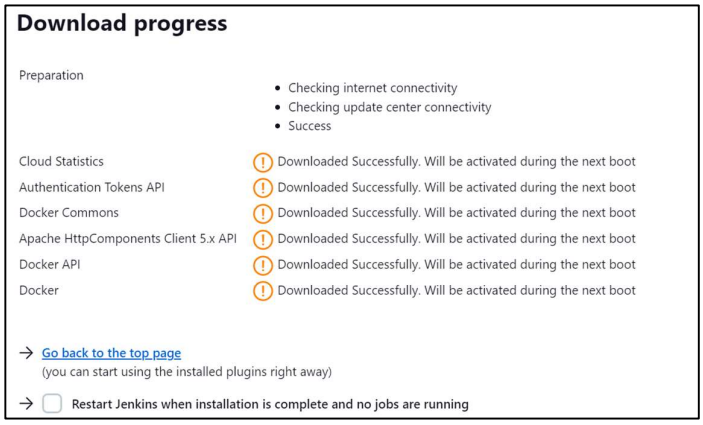
Plugins

Docker

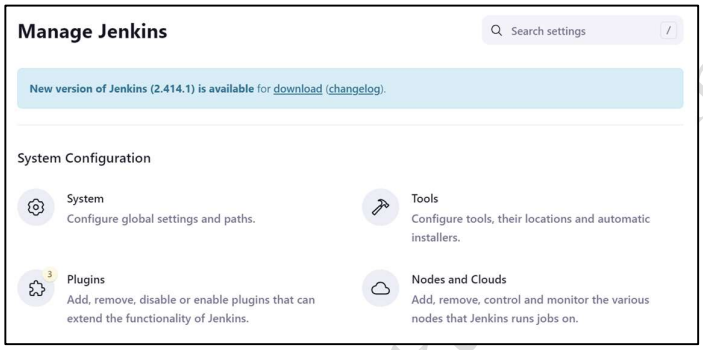
Install	Name	Released
☑	Docker 1.4 Cloud Providers · Cluster Management · docker This plugin integrates Jenkins with Docker **This plugin is up for adoption!** We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.	2 mo 19 days ago

Install without restart | Download now and install after restart | Update information obtained: 13 min ago | Check now

Select Download now and install after restart and then click on restart Jenkins.

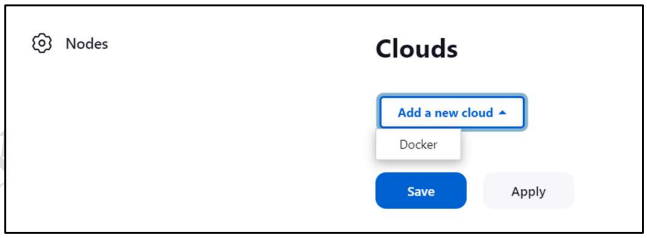


Now Login inside the Jenkins. Go to Manage Jenkins – Select Nodes and Clouds.

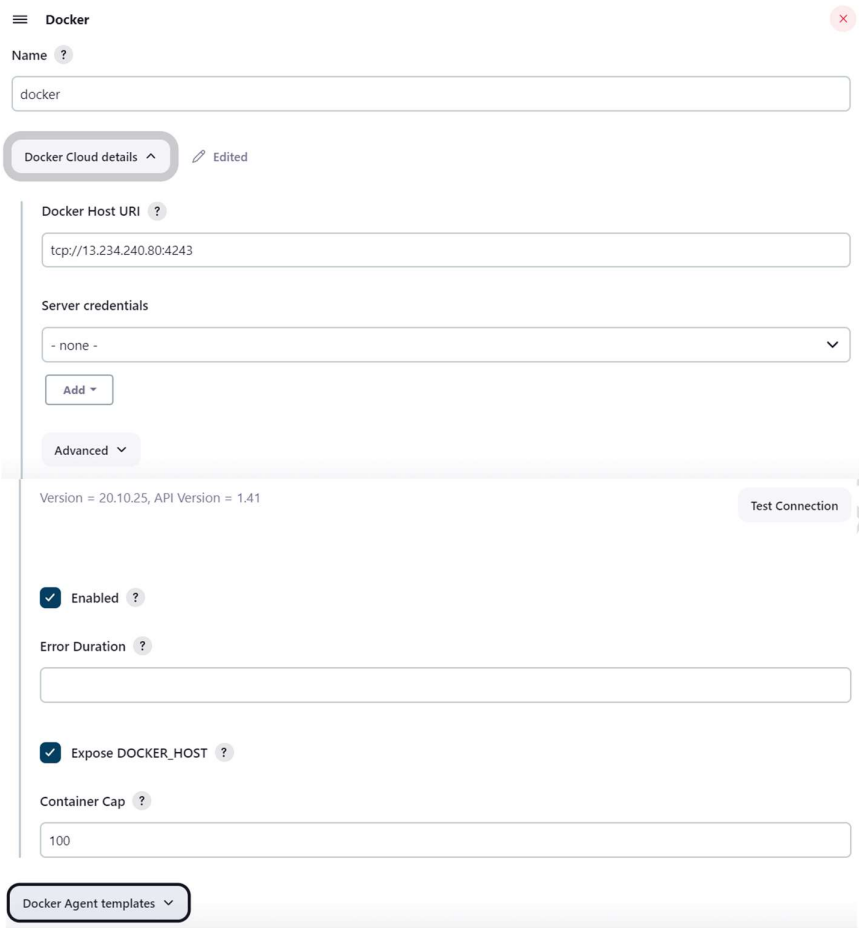


Click on Clouds from left panel.

Click on Docker



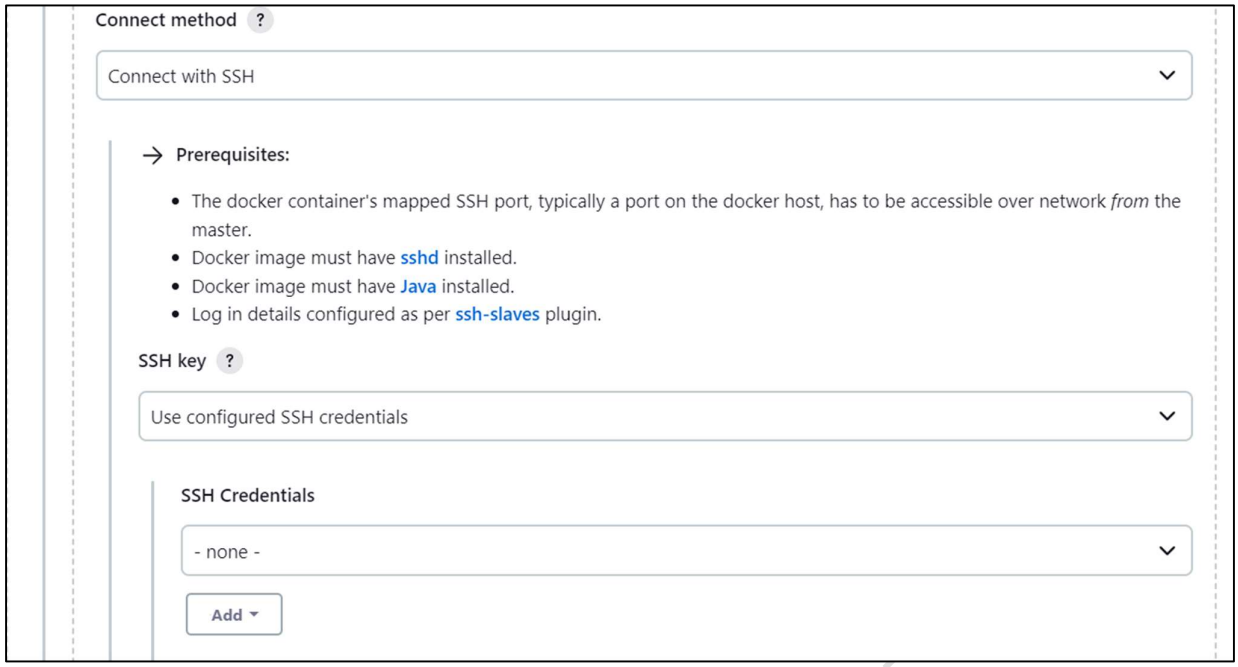
After that, I have to enter the IP address with port Number using TCP protocol.



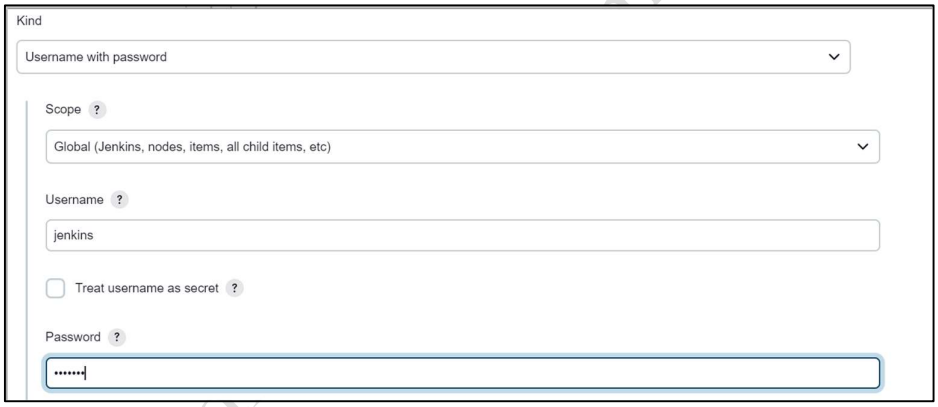
After that I clicked on Test Connection and then I can see information about my Docker Version. After that click on **Enabled** checkbox and enable the **Expose DOCKER_HOST** checkbox. Then click on Docker Agent templates and then click on Add Docker template.

After that I mentioned the information about my Docker Image.

Scroll down, and you will find an option for connect Method. And for this select the option Connect with SSH and then for SSH Key use Configure SSH Credentials.



Click on Add – Select Jenkins – and Enter the Username and password. In my case, while creating the Image using Dockerfile I have created a **user** called **jenkins** and set the **password** as **jenkins** itself.



Then Add this credential in the Jenkins. And don't forget to mention Host Key Verification strategy as None.

Search for Remote file system root and mention the path for that. In my case this is **/home/Jenkins.**

Remote File System Root ?

/home/jenkins

And in Usage Select **only build jobs with label expressions matching this node.**

Usage ?

Only build jobs with label expressions matching this node

Then click on Save button. Now I am going to create the Job as freestyle.

Enter an item name

DockerJob

» *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.

Adding the GitHub Link for my maven project and selecting the Project where I would like to run. And I have selected Docker-Maven.

Repositories ?

Repository URL ?

https://github.com/Himanshu-kr-007/Java-Maven-App.git/

Credentials ?

- none -

Add

Advanced

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

This code is present in my main branch.

In the Build Step I have selected first: Invoke top-level Maven Target and set the goal as clean package. And then execute the shell command to run the package.

Here I am using the Docker which is going to launch on demand basis after the Job executed then this container will delete from the cloud but here, I would like to save the output in my Workspace. For that I have given the post build action which will save the artifact after the container is terminated.

Now I am going to Build this Job. And after that behind the scenes It will start downloading the Image in my Docker Slave OS till that time the Job will go in the Queue

```
[root@ip-172-31-46-152 ~]# docker images
REPOSITORY                      TAG       IMAGE ID       CREATED             SIZE
himanshukr0612/maven-git-app    v2        26c1d913d7db   About an hour ago   717MB
[root@ip-172-31-46-152 ~]#
```

Now, Jenkins will download the Image in my Docker OS and by using that image it starts creating the container and then starts execution of the job. We can also see this from Build Executor status.

```
[root@ip-172-31-46-152 ~]# docker ps -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS
  NAMES
dae1431369bb   himanshukr0612/maven-git-app:v2   "/usr/sbin/sshd -D -…"   4 seconds ago   Up 3 seconds   0.0.0.0:32769->22/tcp, :::32769->22/tcp
  wizardly_wescoff
[root@ip-172-31-46-152 ~]#
```

From the Log We can see the Agen is connected and online.

```
<===[JENKINS REMOTING CAPACITY]===>channel started
Remoting version: 3107.v665000b_51092
Launcher: DockerDelegatingComputerLauncher
Communication Protocol: Standard in/out
This is a Unix agent
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by jenkins.slaves.StandardOutputSwapper$ChannelSwapper to constructor java.io.FileDescriptor(int)
WARNING: Please consider reporting this to the maintainers of jenkins.slaves.StandardOutputSwapper$ChannelSwapper
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Evacuated stdout
Agent successfully connected and online
```

```
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.24/plexus-utils-3.0.24.jar (247
kB at 578 kB/s)
[□[1;34mINFO□[m] Building jar: /home/jenkins/workspace/DockerJob/target/my-app-1.0-SNAPSHOT.jar
[□[1;34mINFO□[m] □[1m------------------------------------------------------------------------□[m
[□[1;34mINFO□[m] □[1;32mBUILD SUCCESS□[m
[□[1;34mINFO□[m] □[1m------------------------------------------------------------------------□[m
[□[1;34mINFO□[m] Total time:  12.909 s
[□[1;34mINFO□[m] Finished at: 2023-08-25T07:56:54Z
[□[1;34mINFO□[m] □[1m------------------------------------------------------------------------□[m
[DockerJob] $ /bin/sh -xe /tmp/jenkins6440876335551266721.sh
+ java -jar target/my-app-1.0-SNAPSHOT.jar
Hello World!
Archiving artifacts
Finished: SUCCESS
```

After the connection is success, then It's start building the Job. And run the Program also and after that it stores the Artifact in my Jenkins OS. From Message we can see that.

After the job is executed successfully then the Container is automatically removed.

```
[root@ip-172-31-46-152 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ip-172-31-46-152 ~]#
```

Artifacts of DockerJob #2

/ target /

my-app-1.0-SNAPSHOT.jar Aug 25, 2023, 7:56:54 AM 2.61 KB

(all files in zip)

We can see the Cloud statistics also.

Cloud Statistics

Cloud	Template	Overall success rate	Current success rate	Sample count
docker		100%	100%	2
	Docker-Maven-Image (himanshukr0612/maven-git-app:v2)	100%	100%	2

In this context, archiving artifacts plays a crucial role. This is because once the job is completed, the container is deleted. It's worth noting that the container is removed after the job is carried out. However, through Jenkins' Post Build feature, we're able to retain and save the artifacts even after the container is gone.

THANK YOU