Experiment No. 04

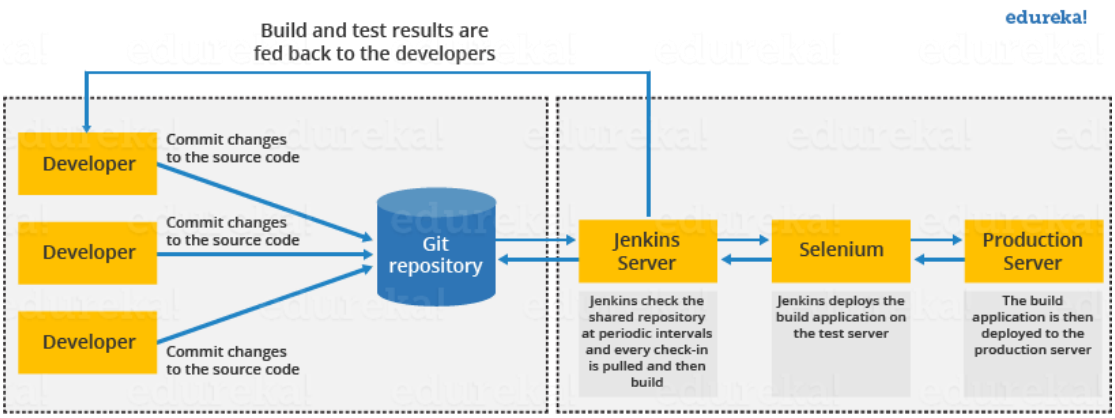
Aim: To Build the pipeline of jobs using Maven / Gradle / Ant in Jenkins, create a pipeline script to Test and deploy an application over the tomcat server and to understand Jenkins Master-Slave Architecture and scale your Jenkins standalone implementation by implementing slave nodes.

Lab Outcome No.: ITL503.3

Lab Outcome: To understand the importance of Jenkins to Build and deploy Software Applications on server environment.

Theory:

Jenkins Architecture



This single Jenkins server was not enough to meet certain requirements like:

- Sometimes you might need several different environments to test your builds. This cannot be done by a single Jenkins server.
- If larger and heavier projects get built on a regular basis then a single Jenkins server cannot simply handle the entire load.

To address the above-stated needs, Jenkins distributed architecture came into the picture.

Tomcat:

- It is an open-source Java servlet container that implements many Java Enterprise Specs such as the Websites API, Java-Server Pages and last but not least, the Java Servlet.
- The complete name of Tomcat is "Apache Tomcat" it was developed in an open, participatory environment and released in 1998 for the very first time. It began as the reference implementation for the very first Java-Server Pages and the Java Servlet API.

- However, it no longer works as the reference implementation for both of these technologies, but it is considered as the first choice among the users even after that. It is still one of the most widely used java-sever due to several capabilities such as good extensibility, proven core engine, and well-test and durable. Here we used the term "servlet" many times, so what is java servlet; it is a kind of software that enables the webserver to handle the dynamic(java-based) content using the Http protocols.

Jenkins Distributed Architecture

Jenkins uses a Master-Slave architecture to manage distributed builds. In this architecture, Master and Slave communicate through TCP/IP protocol.

Jenkins Master

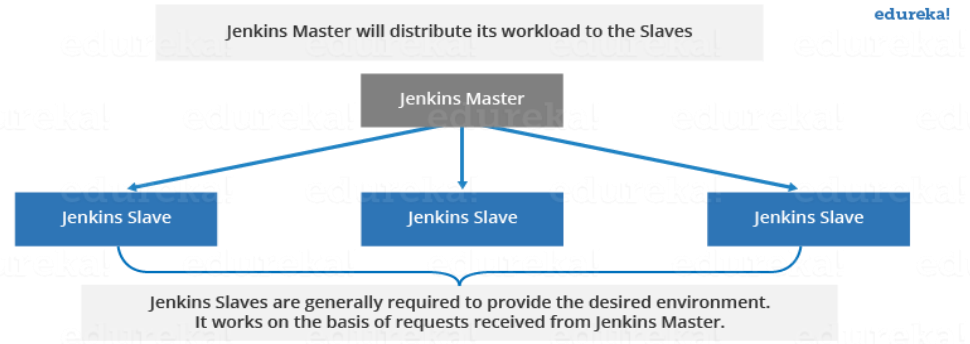
Your main Jenkins server is the Master. The Master's job is to handle:

- Scheduling build jobs.
- Dispatching builds to the slaves for the actual execution.
- Monitor the slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master instance of Jenkins can also execute build jobs directly.

Jenkins Slave

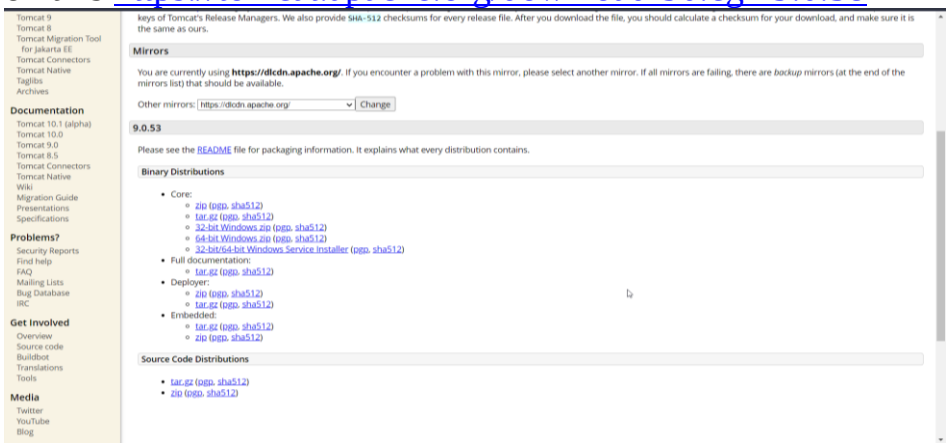
A Slave is a Java executable that runs on a remote machine. Following are the characteristics of Jenkins Slaves:

- It hears requests from the Jenkins Master instance.
- Slaves can run on a variety of operating systems.
- The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
- You can configure a project to always run on a particular Slave machine or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.

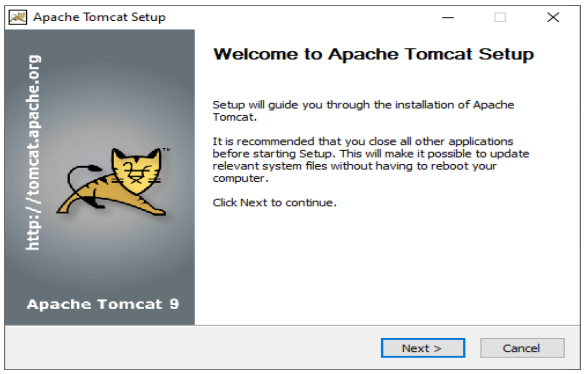


Steps to download and install the Tomcat:

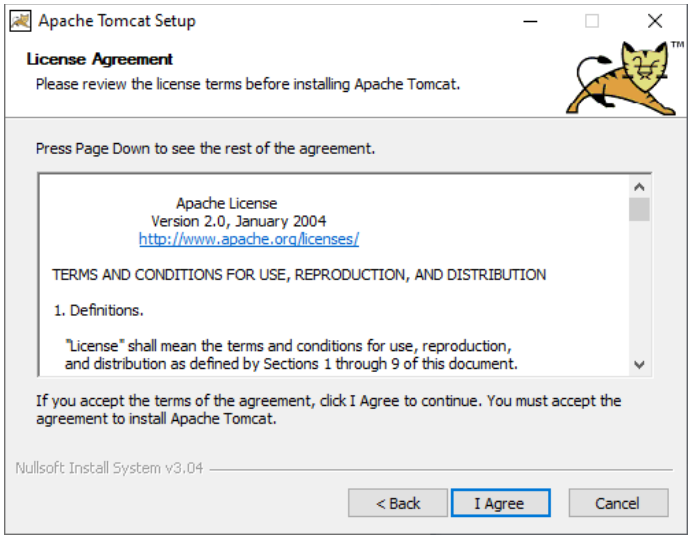
- Open the Google Chrome or any of your web browser and type "download Tomcat for windows" in the search box. You can also go directly on Tomcat's website by clicking on this https://tomcat.apache.org/download-90.cgi#9.0.38



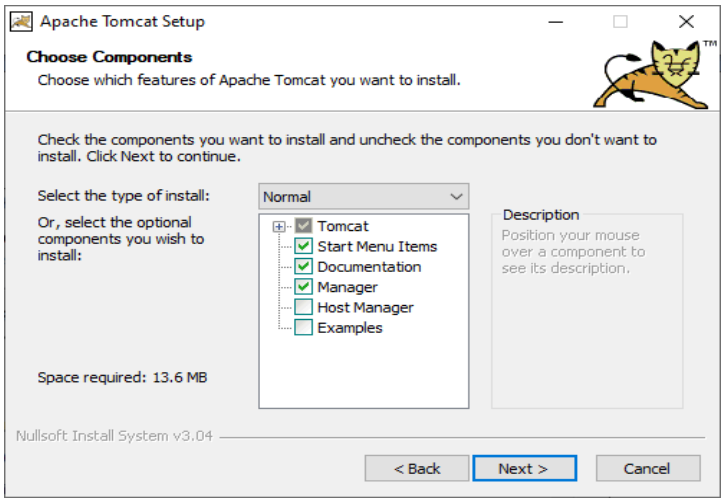
- Go to Download and click on the downloaded file and wait for little until the installation process starts. Once the installation process gets started, click on the "Next" button, as shown below:



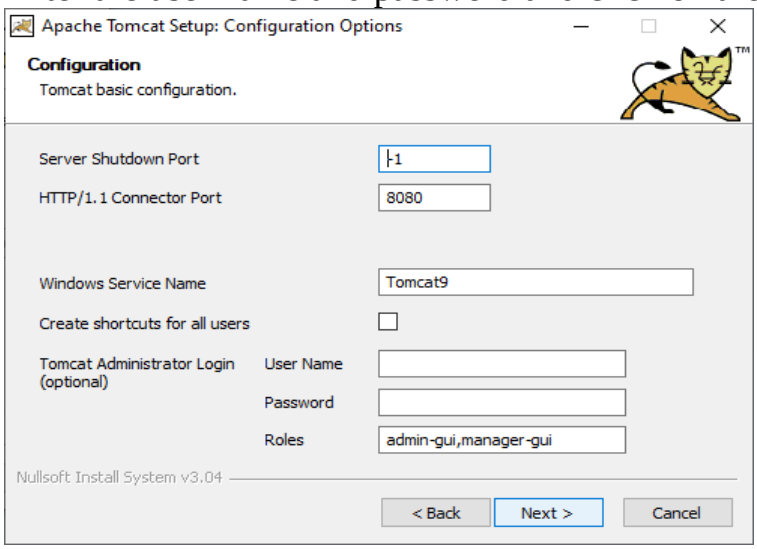
- Click on the button labeled as "I Agree."



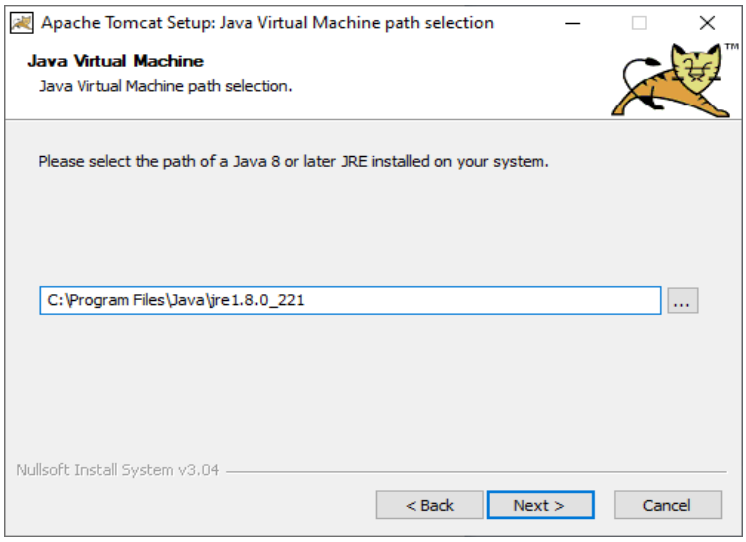
- Click on the "Next" button.



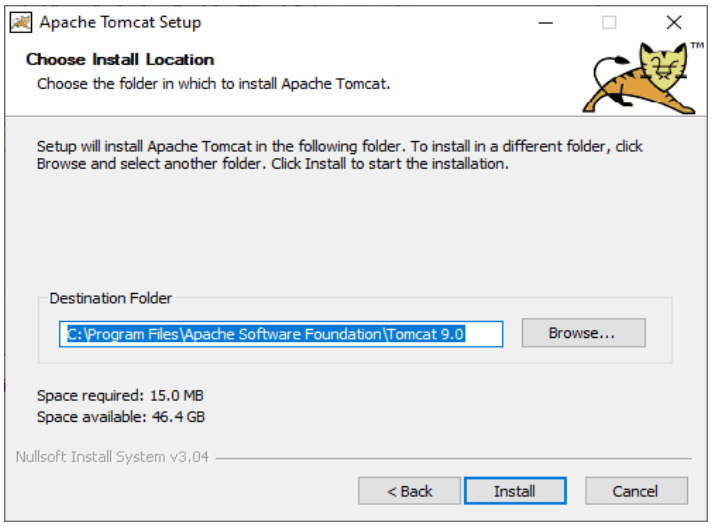
- Enter the user name and password and click on the " Next" button, as shown below:



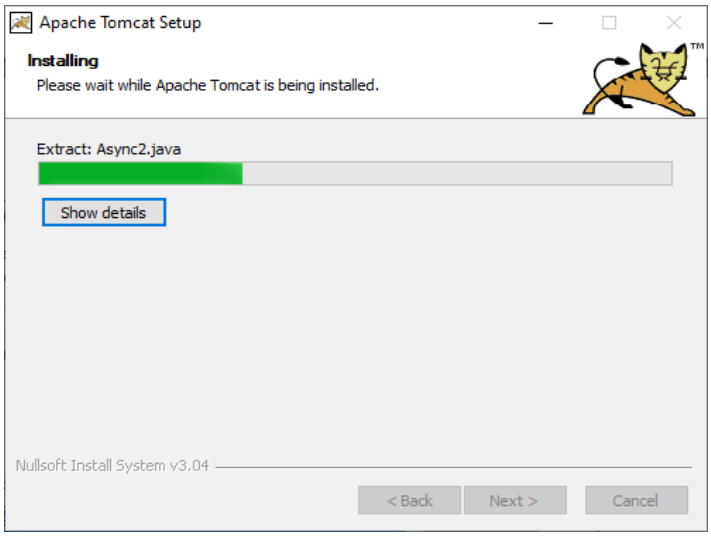
- Then click on the "Next" button again



- Now click on the "Install" button.

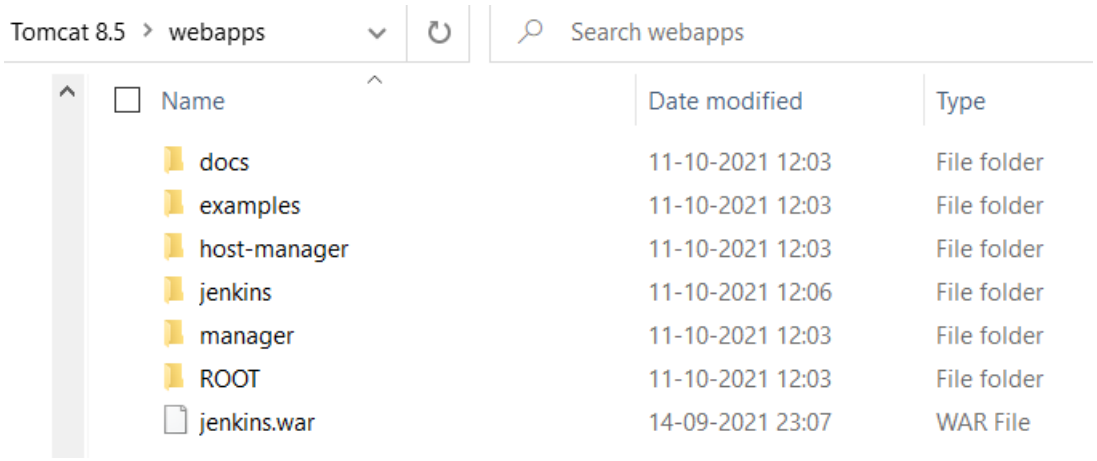


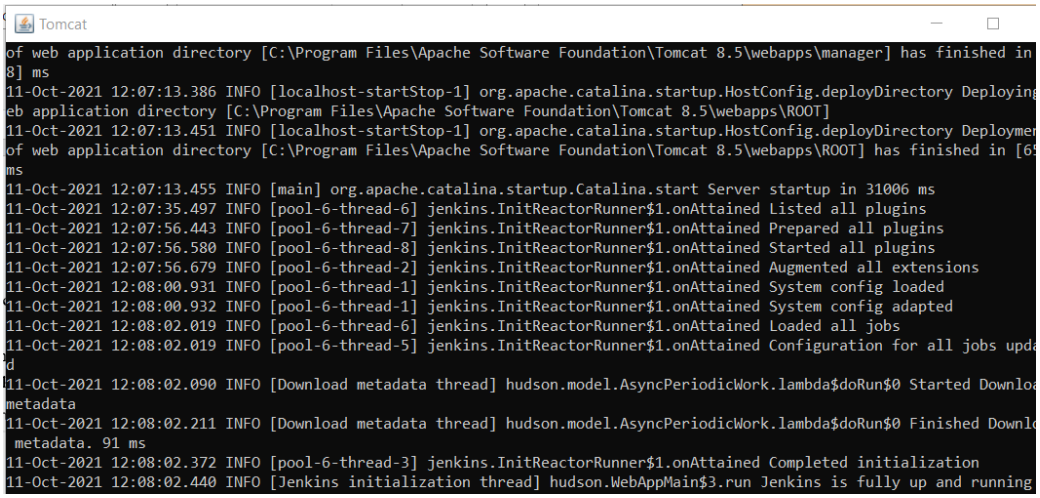
- Wait for some time until the Tomcat gets installed



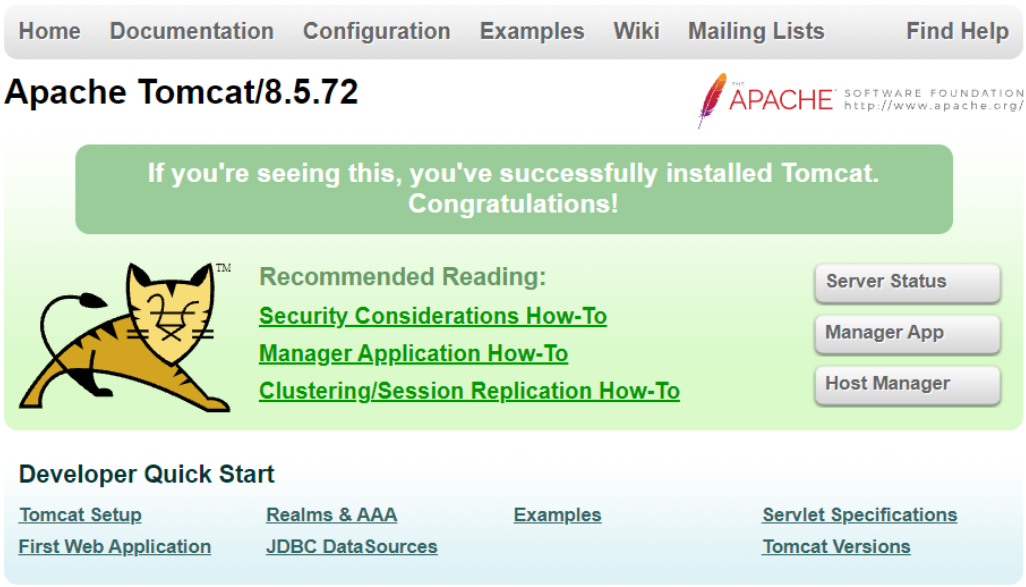
- <u>Integrate Tomcat with Jenkins:</u>

To use tomcat with Jenkins we need to add Jenkins.war file in tomcat. To do that go to tomcat folder click on webapps and paste the Jenkins.war file in that folder then go back and open the bin folder and double click on startup.bat file. This file will open the cmf and configure the Jenkins.

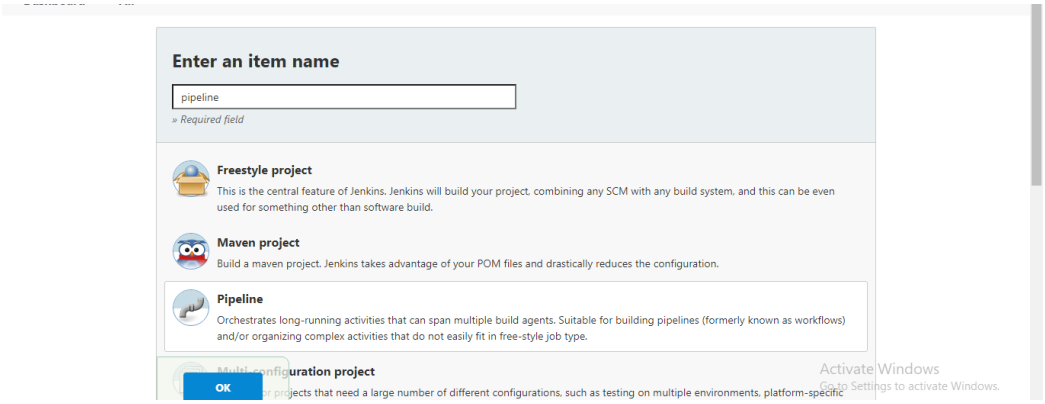


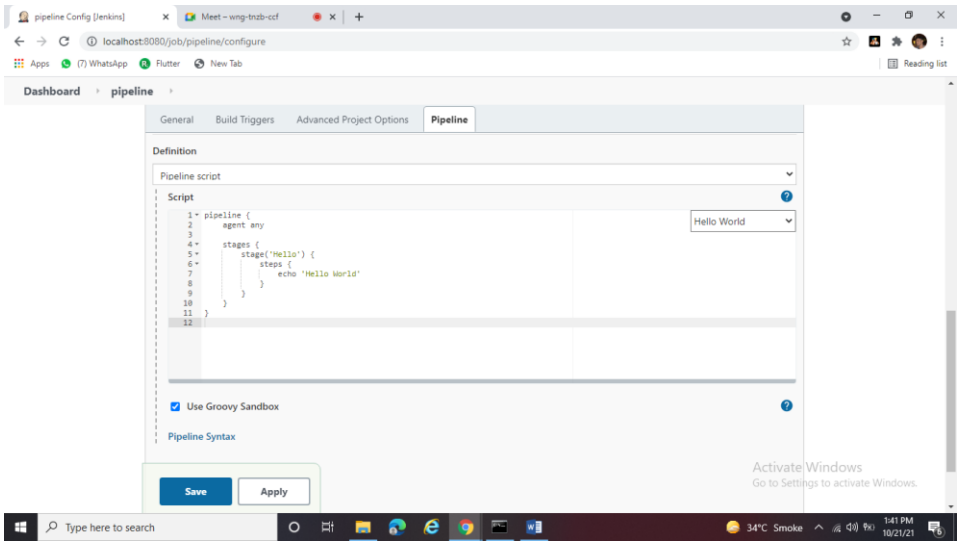


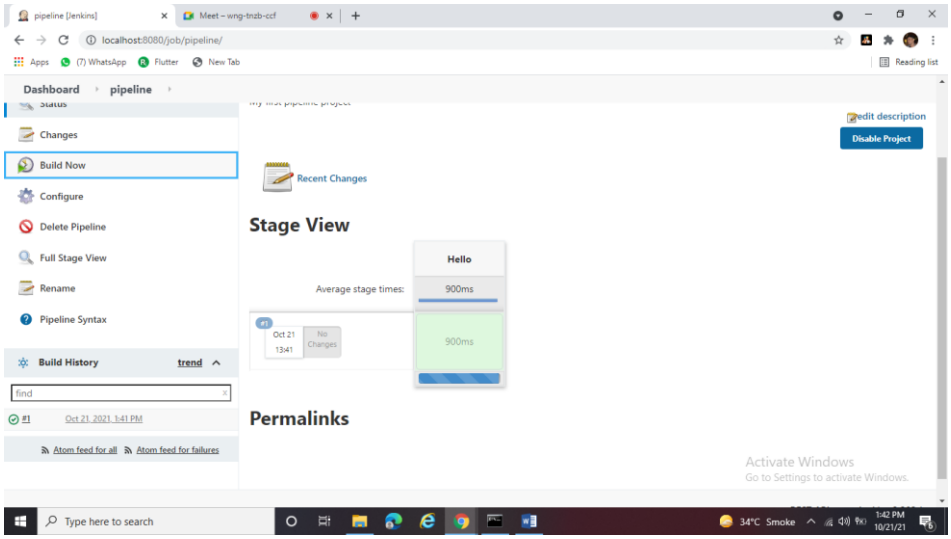
- Now to check the tomcat is installed properly in our system and Jenkins is running or not go to http://localhost:8080/ it will show like this

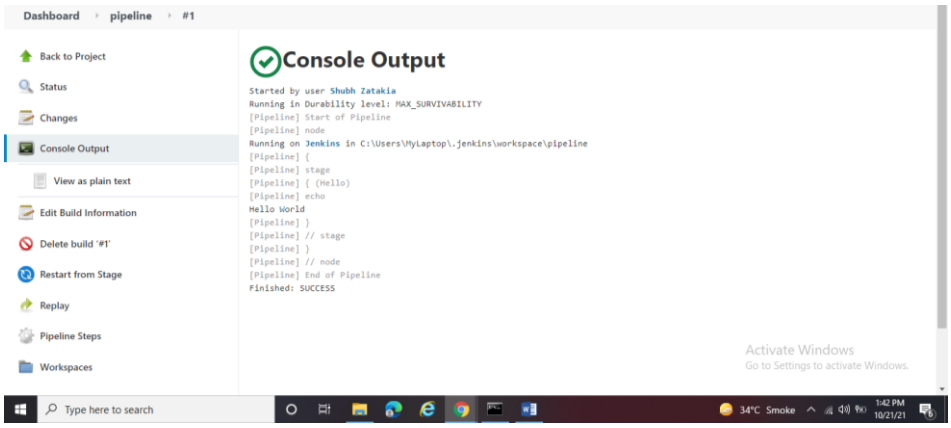


- Creating pipeline in Jenkins:



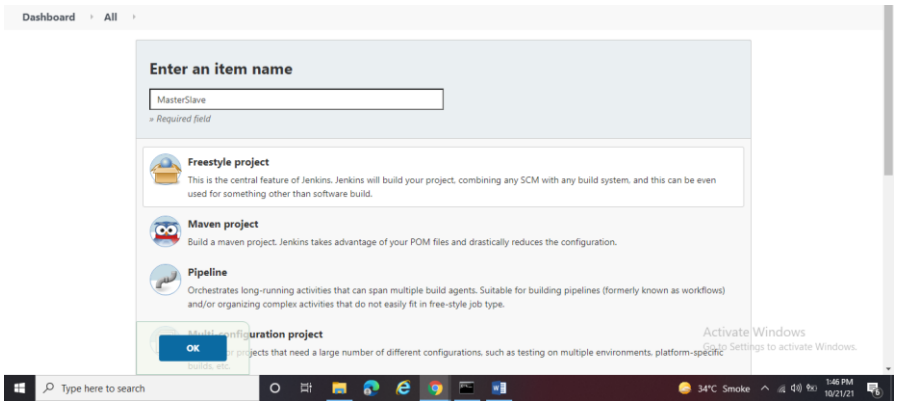




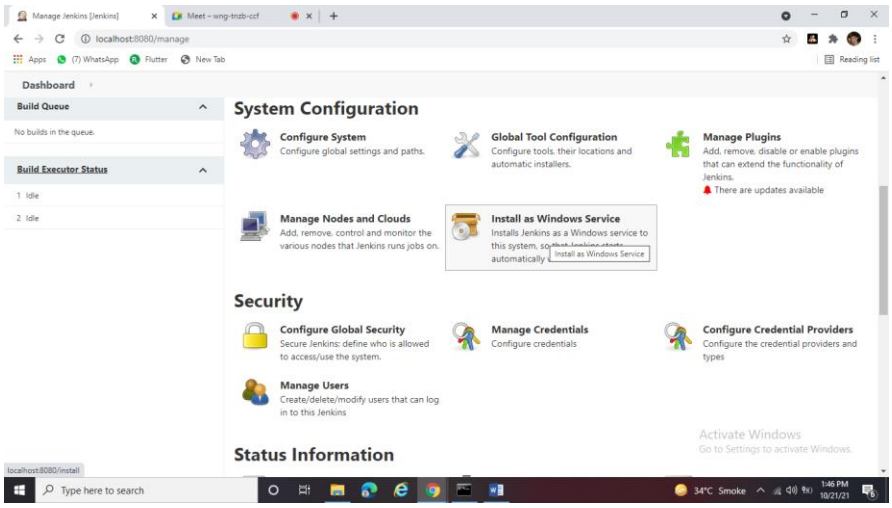


Steps for Creating Master-Slave Architecture:

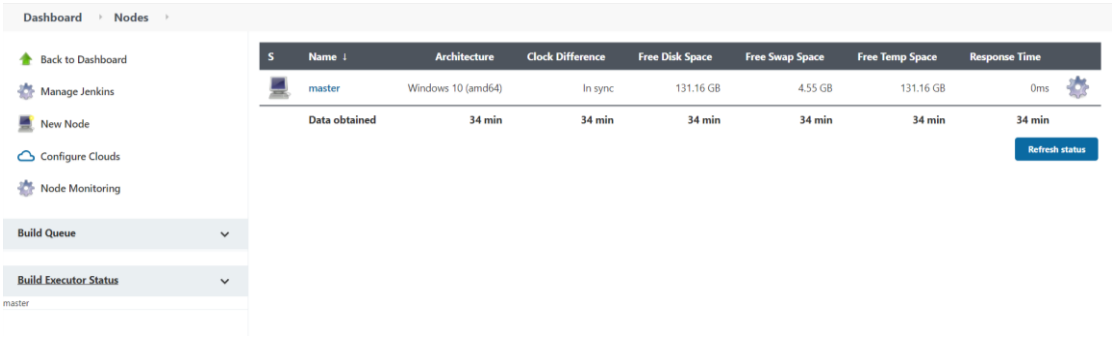
1. Create a simple job first.



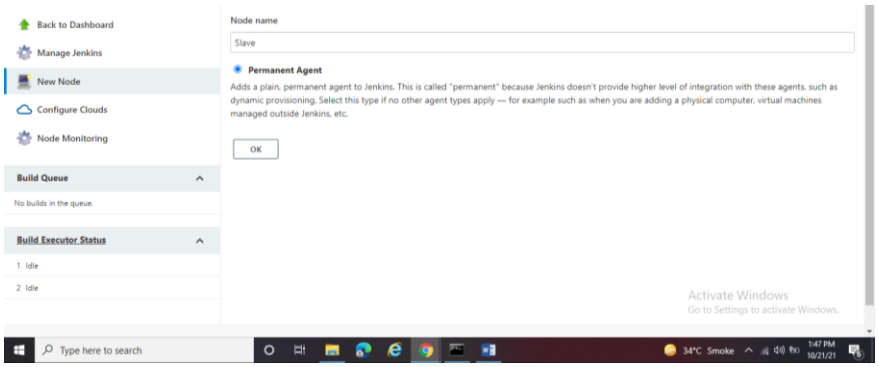
2. Then, for creating slave node go to manage Jenkins and then manage nodes and clouds.



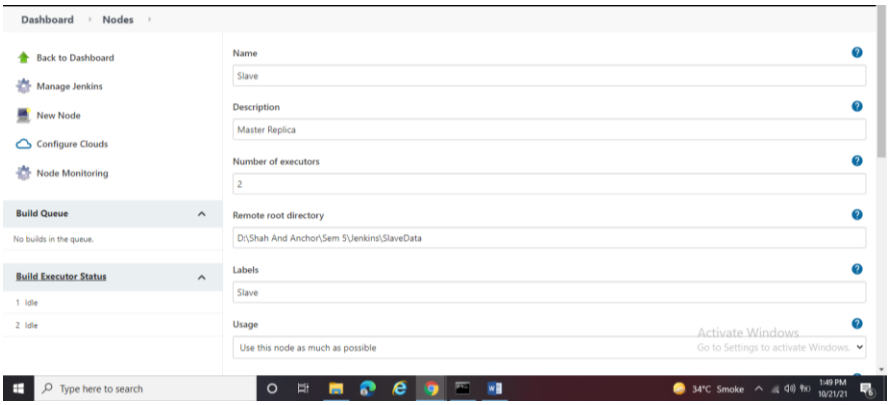
3. Then on click on New Node.

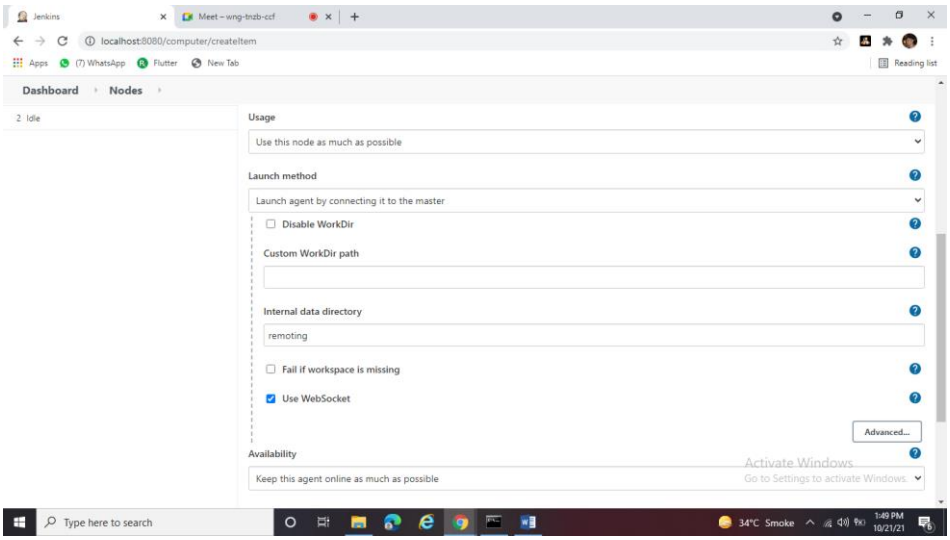


4. Then give name of the node which you want to create and then click on OK.

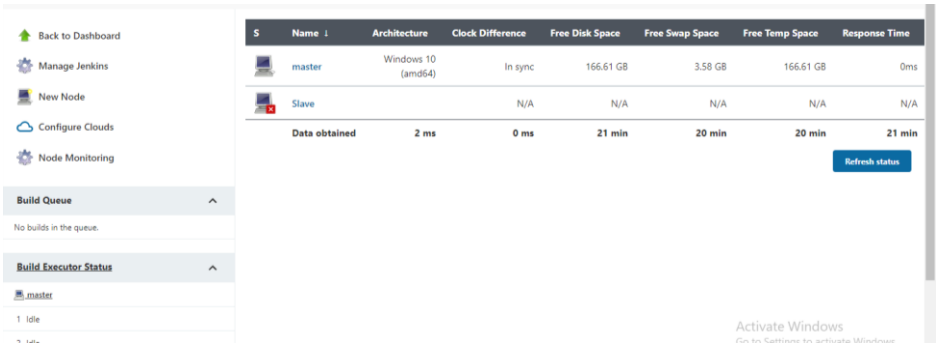


5. Enter the details as required and then click on Save.

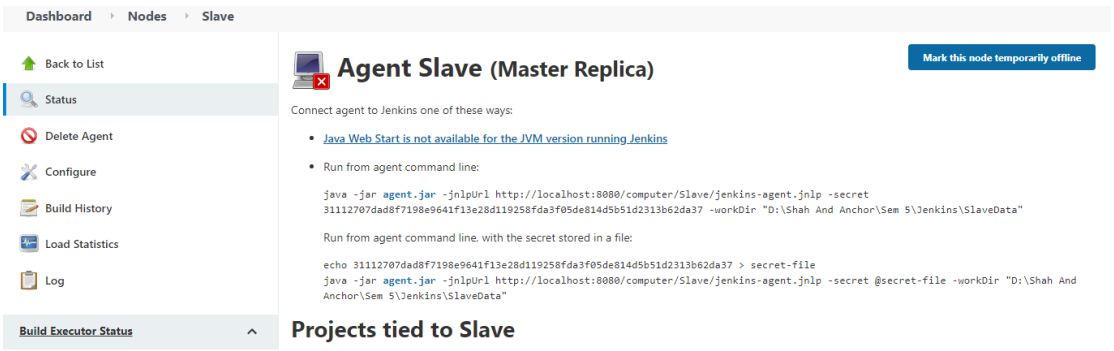




6. The node will be created as follows.

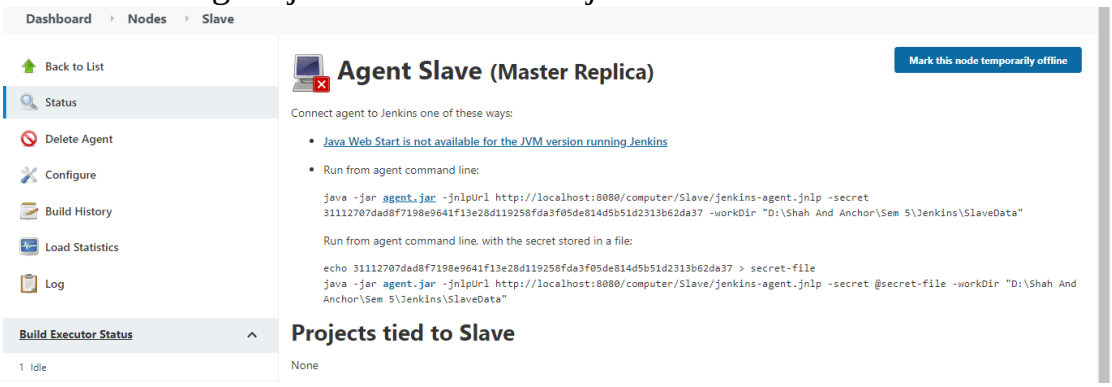


7. Click on the node which you created. The node which you created is not running now.



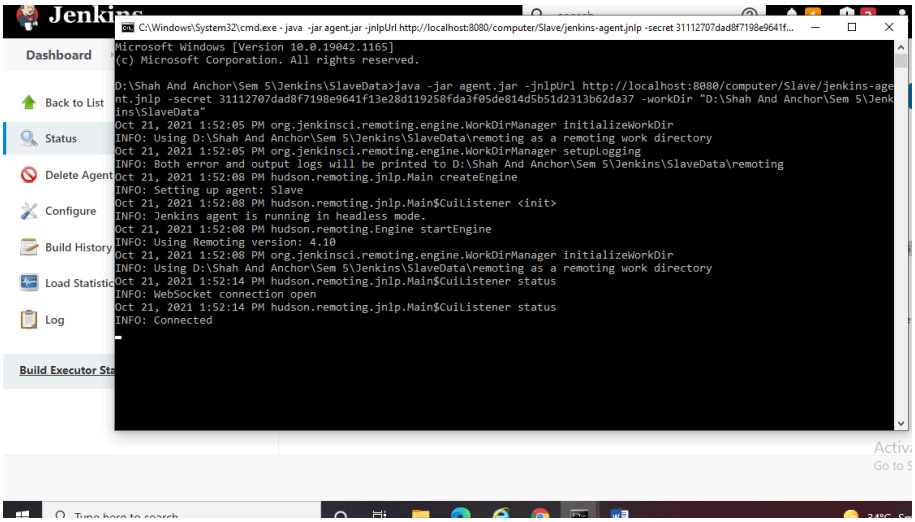
To make the node running we need to follow some more steps

8. Click on the agent-jar to download the jar file.

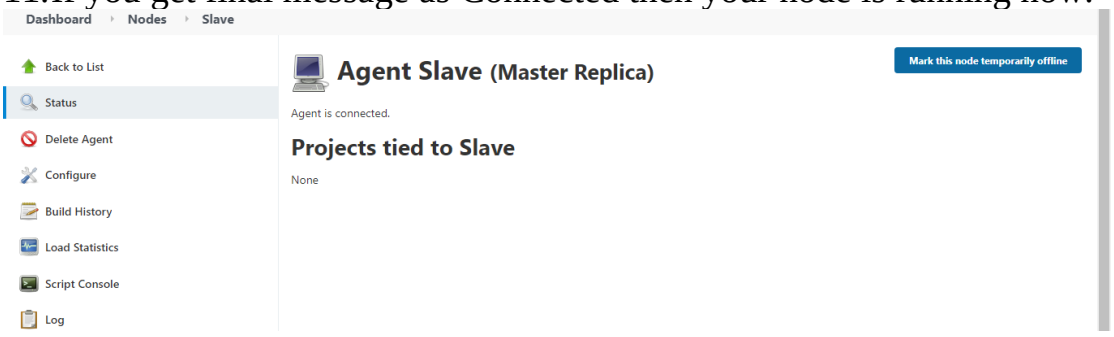


9. Paste this jar file in the path which you provided in previous step.

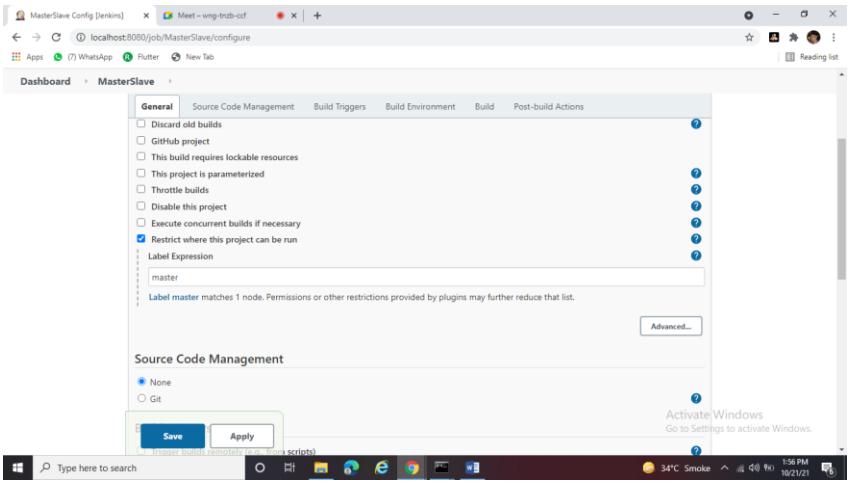
10. Copy the entire command and paste it in the command prompt. Add path of the jar file before the agent-jar.



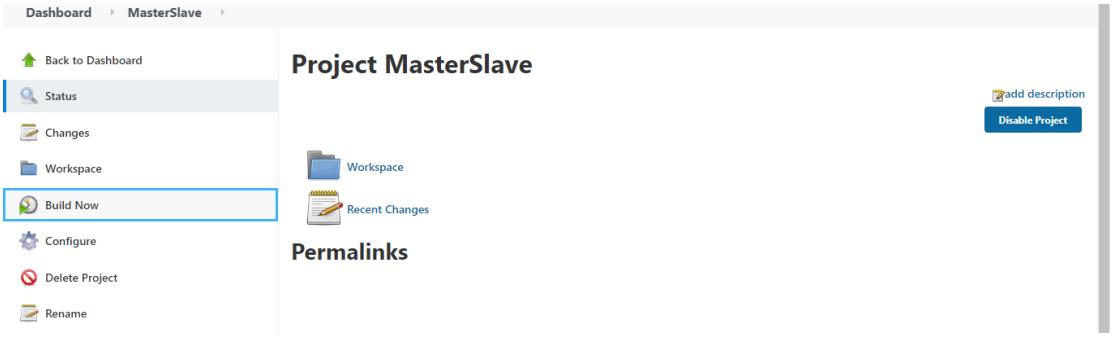
11. If you get final message as Connected then your node is running now.



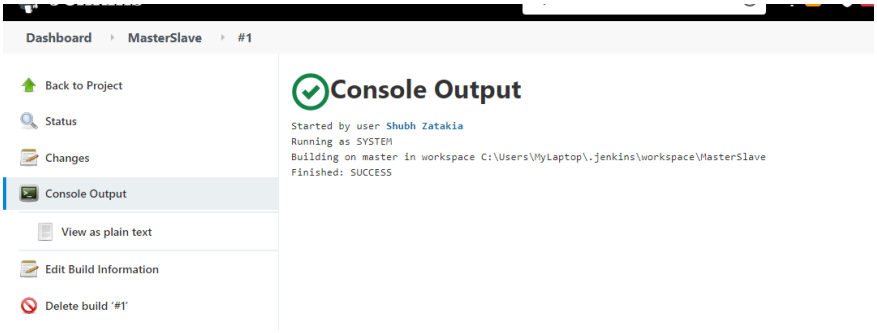
12. To run a job on the node which you created, you need to select any job created and there under Build Environment you need to check the Restrict where the project can be run and then click on Save.



13. Now Click on Build Now option.



14. For Checking whether the build is successful, click on the Console Output. Here you can see where the job is Build.



- **<u>Conclusion</u>:** Successfully created the pipeline of job in Jenkins and Implemented Master-Slave Architecture using Jenkins.

Experiment No. 05

Aim: To Setup and Run Selenium Tests in Jenkins Using Maven.

Lab Outcome No: ITL503.4

Lab Outcome: Understand the importance of Selenium and Jenkins to test Software Applications

Theory:

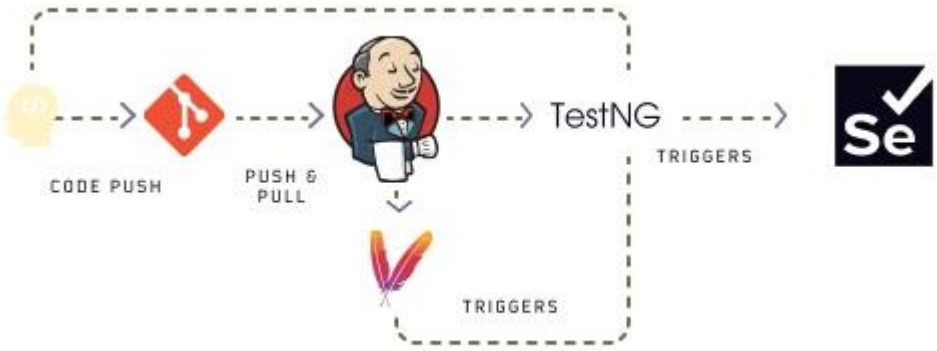
- Maven is a popular build automation tool that is primarily used for Java projects. The advantage of using Maven is that Java libraries and plugins are downloaded on a dynamic basis from the Maven 2 Central Repository.
- The dependencies and other essential build-related information are stored in a pom.xml file of the project. Once downloaded, the dependencies are stored in a local cache (.M2 repository), and the same is used for build generation. This simplifies the build process, as any new project dependency has to be added only in pom.xml, i.e., no manual downloading and installation packages are required.
- Selenium is a widely used test automation framework for validating web applications across different combinations of browsers, platforms, and devices (or emulators). You can refer to our blogs on how Selenium Grid can be used for automated browser testing.
- Jenkins is an open-source CI/CD tool that helps in the automation of activities related to build, test, and deployment. Jenkins has an extensive plugin ecosystem, is open-source, and a passionate community – factors that can be attributed to Jenkins' features.
- In this Maven and Jenkins with Selenium blog, we would be using TestNG, which is a popular test framework that is inspired by JUnit. It is widely used for testing areas such as functional testing, end-to-end testing, and more.

These are the set of tools that we would be using in the demonstration:

- Maven – Project management tool in Java [Link]
- TestNG – Popular test automation framework [Link]
- Selenium WebDriver – Library primarily used for automation of browser interactions [Link]

- Jenkins – Tool for Continuous Integration (CI) and Continuous Deployment (CD) [Link]

Here is the overall flow of information when Selenium, Maven, and Jenkins are integrated:



- Once the developer pushes code into the repository (e.g., GitHub), a Jenkins build is triggered.
- Maven downloads the dependent libraries & packages and starts performing the build. The information related to the test suite is available in testing.xml, and the same is also used in pom.xml.
- A build goal (e.g., install) for running the automated tests is set. The same is achieved through the maven-surefire-plugin.
- The maven-surefire-plugin tells TestNG to run the tests that are under the annotation @Test.
- Depending on the AUT (application under test) and the browser (& OS combination) on which cross browser tests are performed, the Selenium WebDriver invokes the corresponding browser instance and executes the same automation tests.
- Test results are published in HTML Reports, as the HTML Publisher plugin is used for report generation.
- Even if a single test case has failed, the complete test is marked with status ‘Failed.’

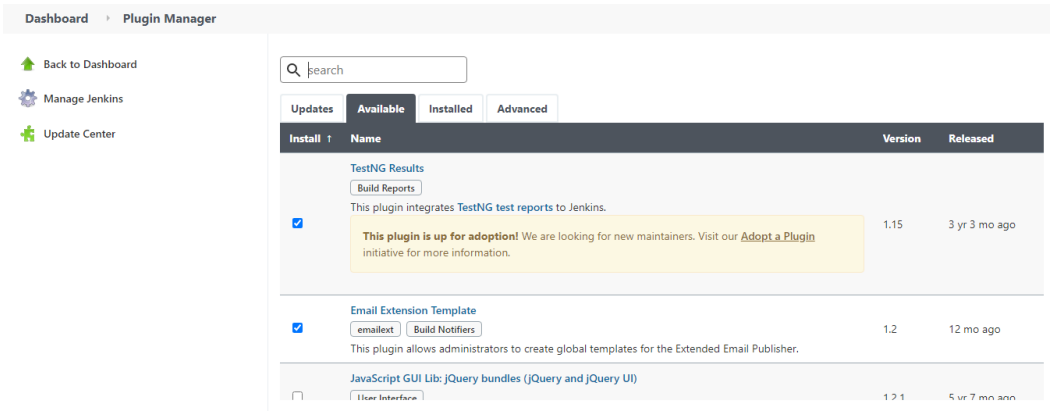
Advantages Of Using Maven & Jenkins with Selenium

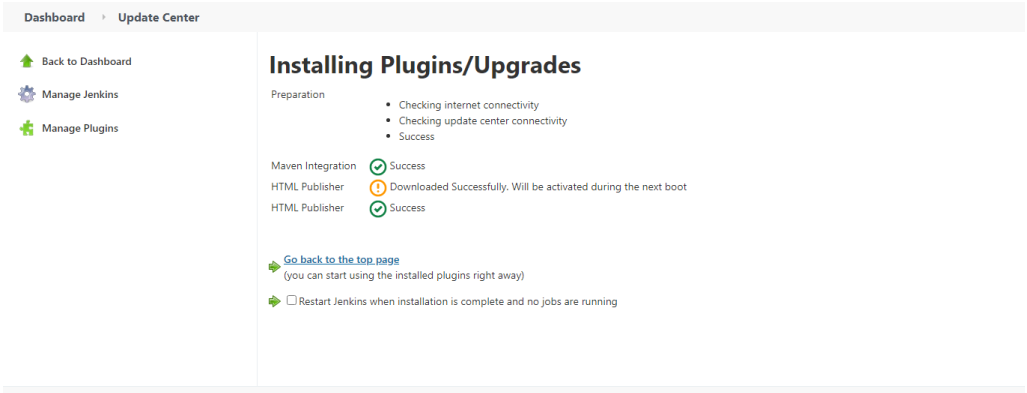
- Before we demonstrate the usage of Selenium Maven Jenkins integration, we will take a look at the main advantages of using Maven and Jenkins with Selenium:
- Whenever a change is made in the implementation, the changes are deployed on the test environment. Automation testing is performed continuously, and developers are kept informed about the build and test stage results.

- Test suites that comprise many test scenarios (or test combinations) might take a longer duration for automation testing. A nightly build run can be scheduled for build and execution on the Jenkins server in such cases.
- As Maven uses pom.xml, it reduces the overall dependency of the manual download of jar files. Also, the ‘Maven Project’ type in Jenkins helps in getting started with Selenium Maven, Jenkins integration with a few configuration settings.
- The Selenium Maven Jenkins integration setup is suited for developing and testing teams distributed across different geographical locations.

Setup Jenkins and Selenium:

1. To run selenium test on Jenkins we need to install necessary plugins and to do that first start your Jenkins and go to manage Jenkins then go to manage plugins and install HTML Publisher and TestNG plugins.

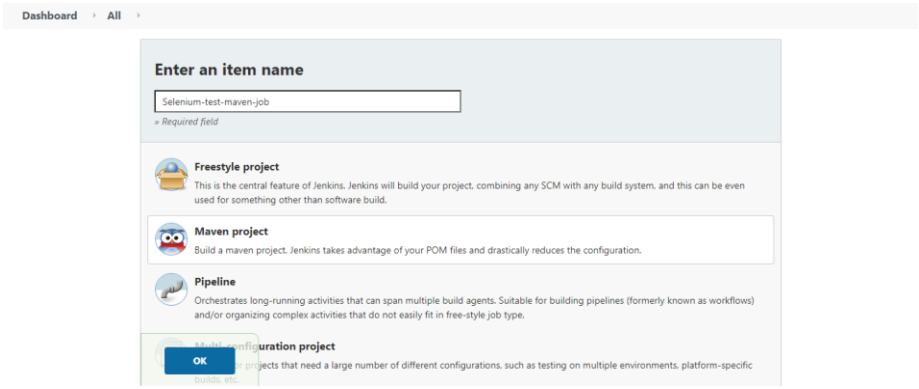




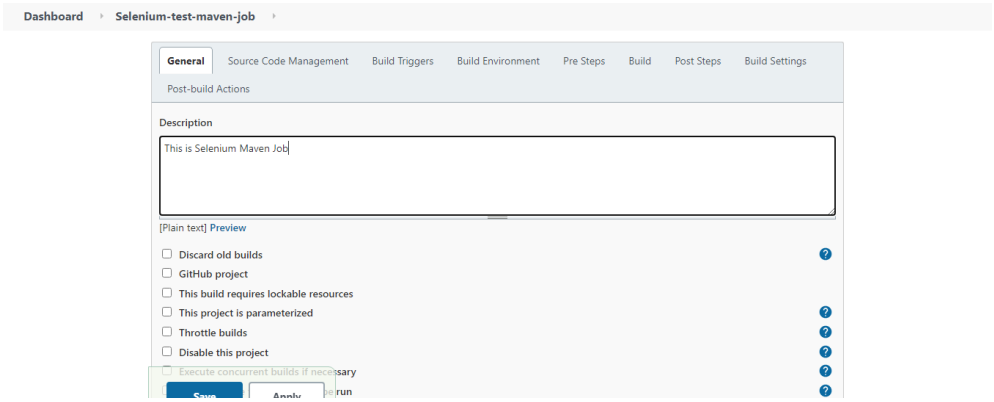
2. Once you've installed both the plugins restart your Jenkins and now you can run selenium test cases on Jenkins.

Run Selenium Tests in Jenkins Using Maven:

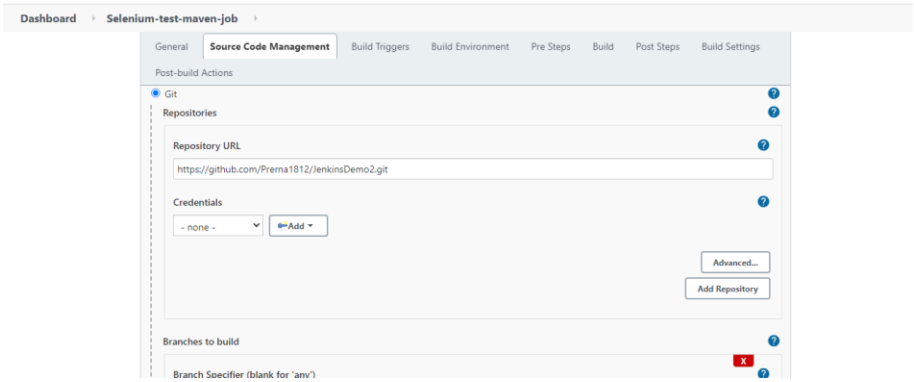
3. Click on new item and select maven job and enter your project name as shown in the figure below



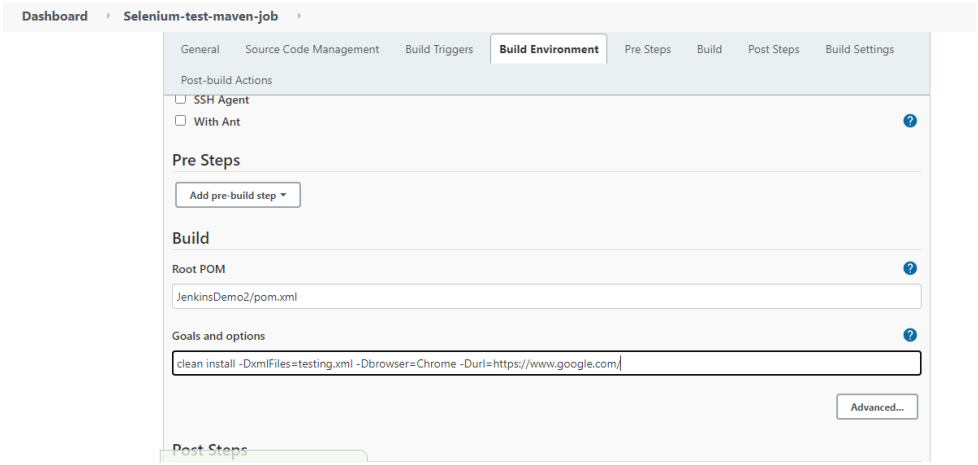
4. Add any description as per your choice

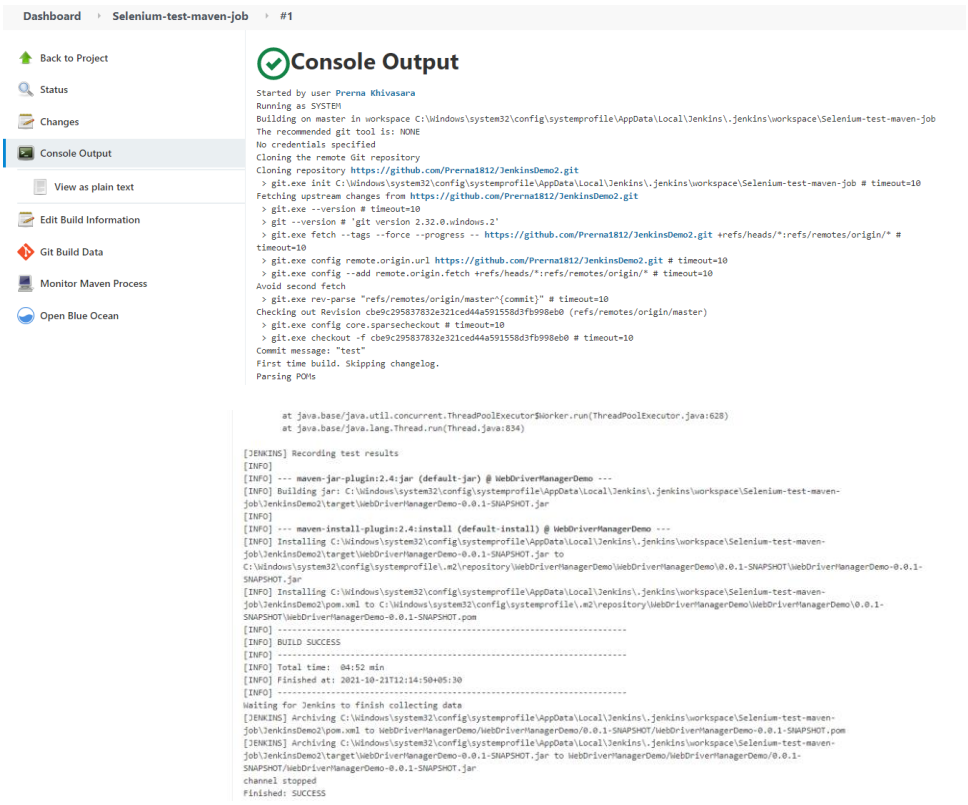


5. Add your git repository link

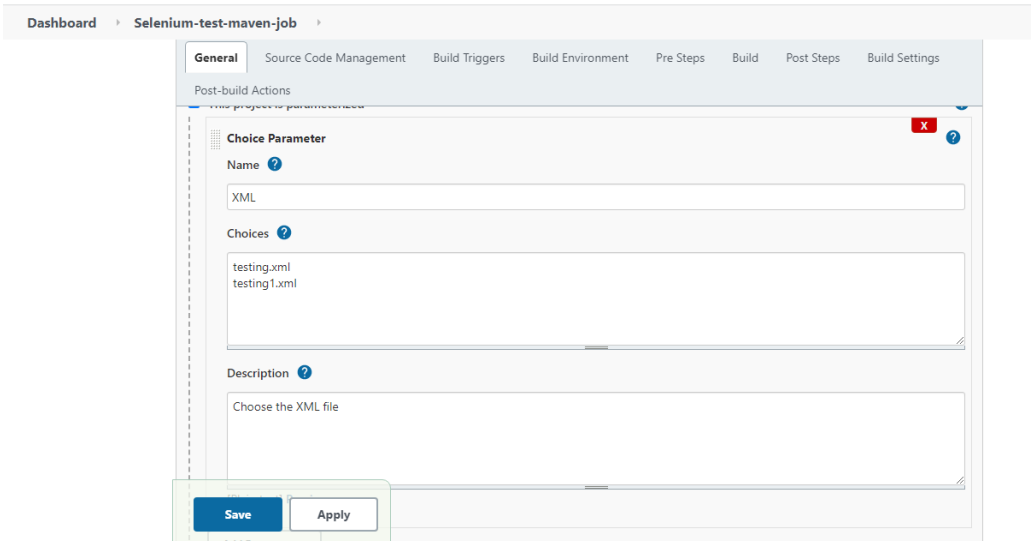


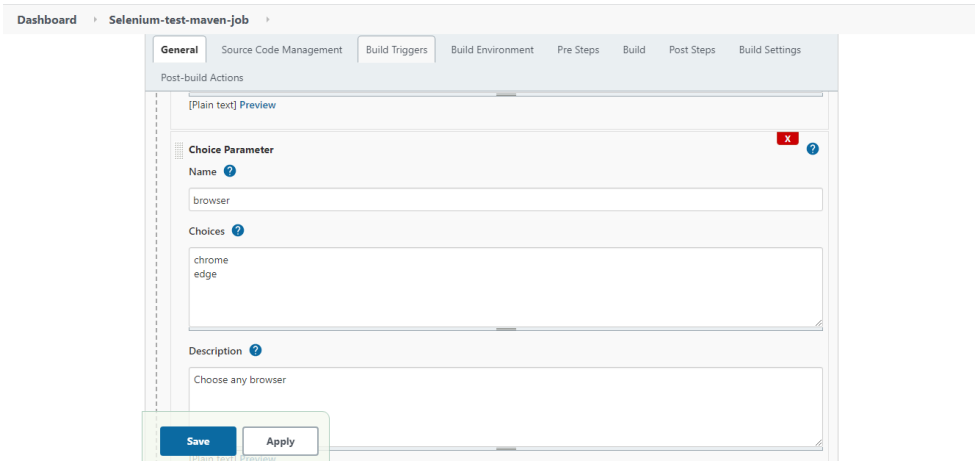
6. We can run the selenium test cases in two different ways. One way is to provide parameters in the build section and then run the code.

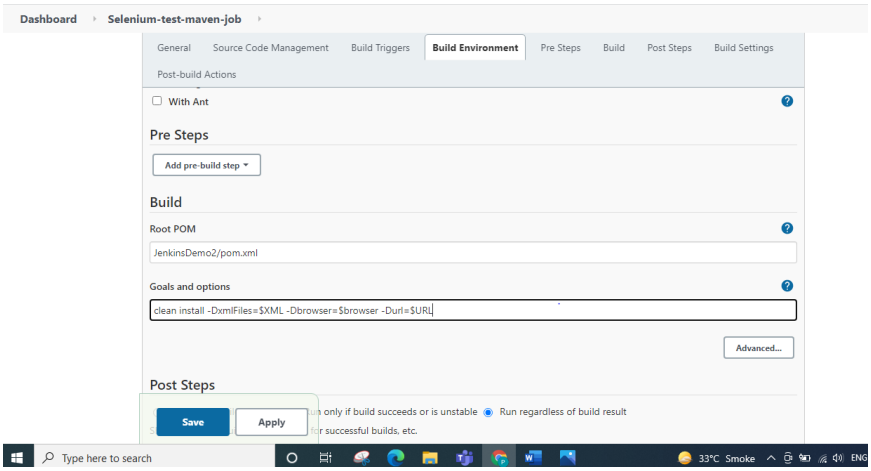




7. And the second one is to provide parameters then enter the key name in the build section and finally execute the project

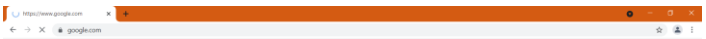


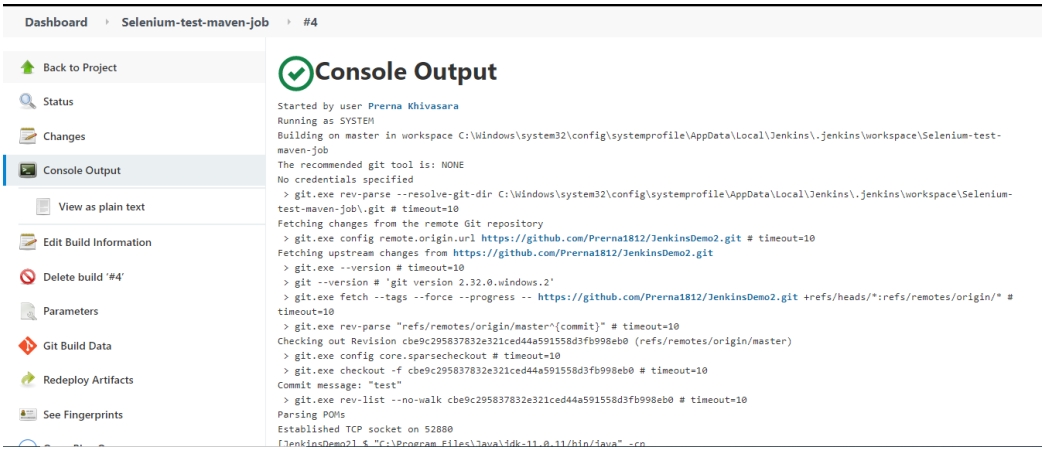




8. By providing the parameters we can see the build option is now change into build with parameters and you can execute the test cases with different options.

9. Once you build the job and if all the parameters are working properly you can see the text cases running. In my case the test case was to run the google chrome three times.





Conclusion:

Learned the concept of selenium. Integrated Jenkins with selenium and installed the necessary plugins. Created a job using maven and deployed the test cases.

Experiment No. 06

Aim: To understand Docker Architecture and Container Life Cycle, install Docker and execute docker commands to manage images and interact with containers.

Lab Outcome No: 5.

Lab Outcome: To understand concept of containerization and Analyze the Containerization of OS images and deployment of applications over Docker.

Theory:

Docker architecture

- Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

The Docker daemon

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

- The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker registries

- A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

- When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Docker objects

- When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Images

- An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.
- You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Containers

- A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
- By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.
- A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

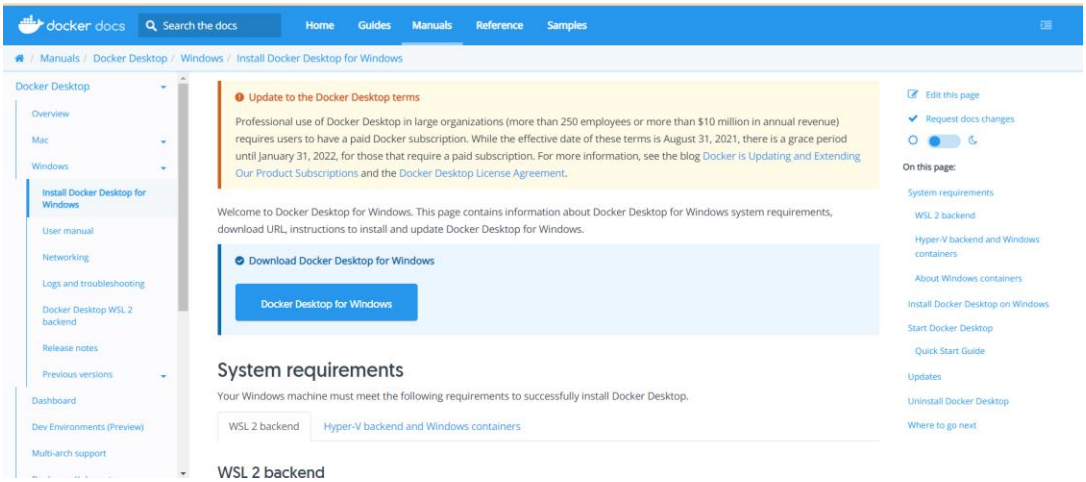
The life cycle of a container

If you working in a DevOps or SRE environment, it is very rare to find a person who has not heard of the name containers. In this article, we talk about the life cycle of a container. For this purpose, we will use docker for a container runtime. Let's have a look at the docker commands first.

- docker build: This is used to build the docker image and then put it to the image registry.
- docker pull: This is used to pull the image build in the above portion from the registry.
- docker run: This will run the image as a docker container
- docker pause: It is used to pause the docker container.
- docker unpause: It is used to unpause the docker container.
- docker stop: Stops the docker container
- docker start: Starts back the docker container.
- docker kill: Kills the docker container.

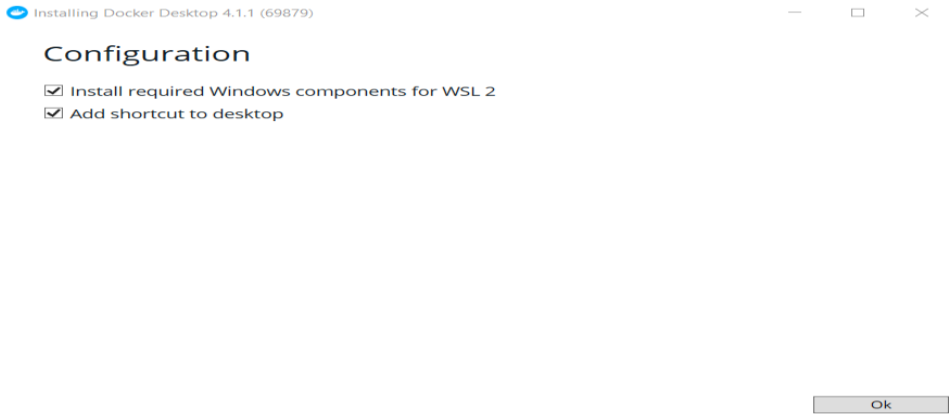
Docker Installation:

1. To install docker go to https://docs.docker.com/desktop/windows/install/ and click on docker desktop for windows. But before installing we need to make sure the system requirements are satisfied, once all the necessary changes is done start download the docker.

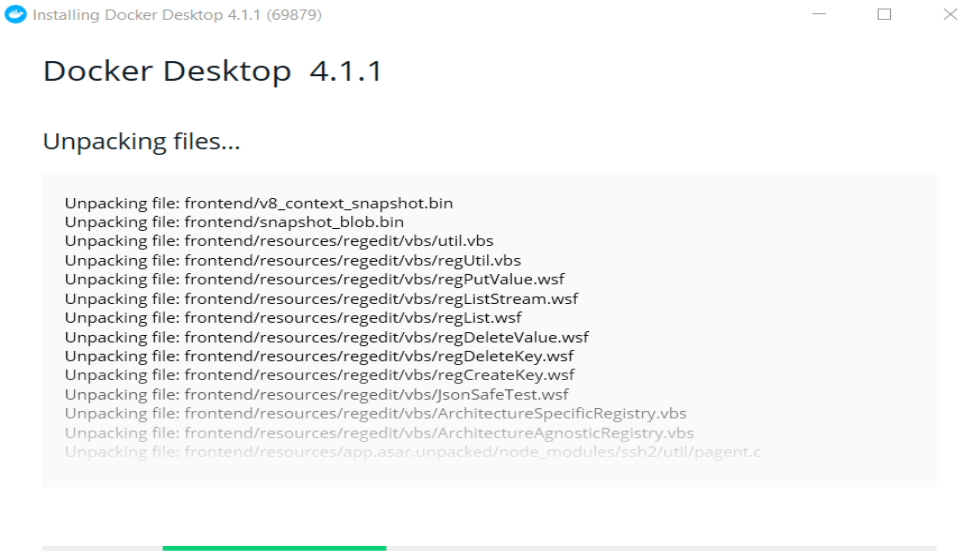


2. Once the docker is downloaded double click on the file and start installing

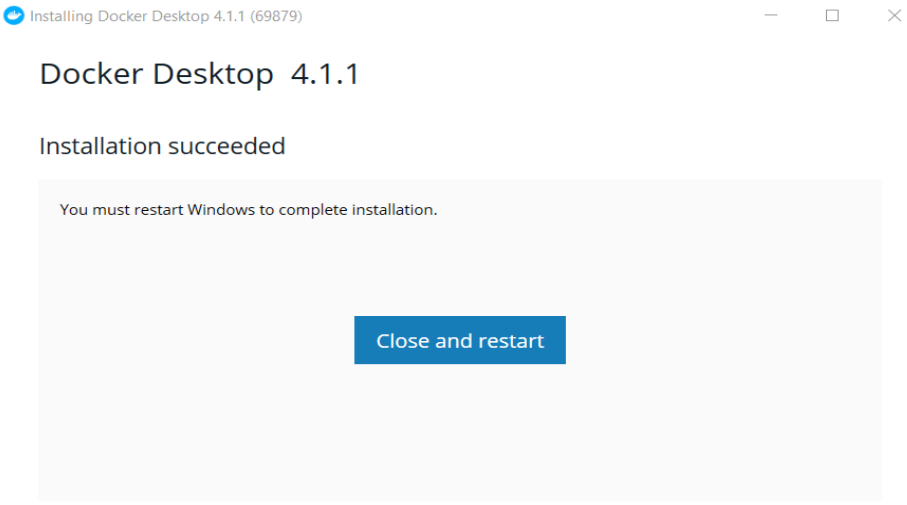
3. While downloading the docker make sure you select both the options and click ok



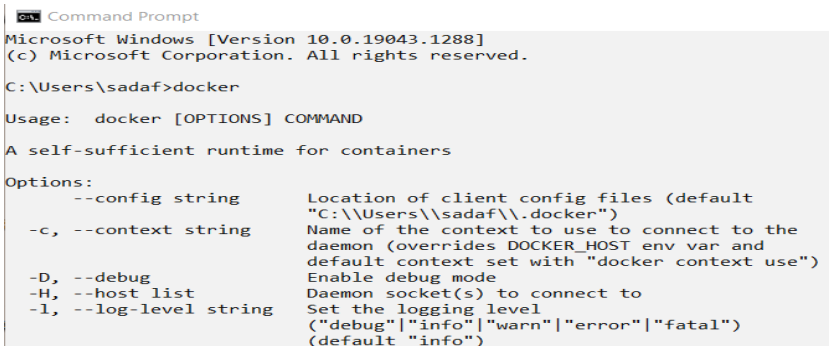
4. After this step the installation process begins.



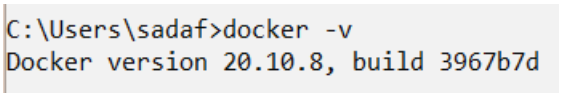
5. Once the docker is installed click on close and restart the PC to update the configurations.



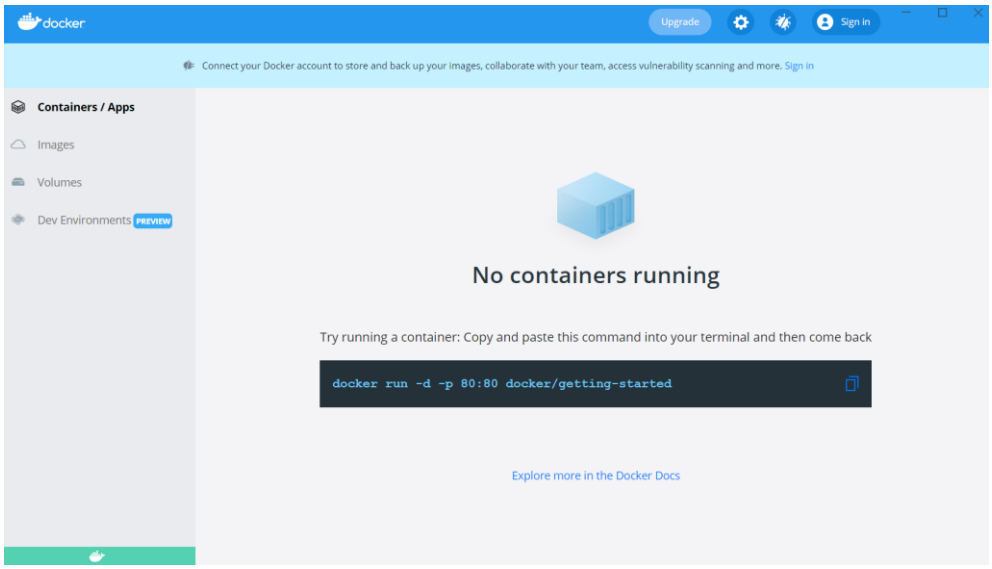
6. Once the system is started open cmd and type docker to check if docker is properly installed or not.



7. Check docker version



8. Once the docker is installed start the app and it will show as Docker engine running



- Docker Images Commands:

Description	Screenshots
docker ps: List containers docker images: List images	C:\Users\sadaf>docker ps CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES C:\Users\sadaf>docker images REPOSITORY TAG IMAGE ID CREATED SIZE
docker pull: Pull an image or a repository from a registry	C:\Users\sadaf>docker pull ubuntu Using default tag: latest latest: Pulling from library/ubuntu 7b1a6ab2e44d: Pull complete Digest: sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322 Status: Downloaded newer image for ubuntu:latest docker.io/library/ubuntu:latest C:\Users\sadaf>docker images REPOSITORY TAG IMAGE ID CREATED SIZE ubuntu latest ba6acccedd29 6 hours ago 72.8MB C:\Users\sadaf>docker pull ubuntu:18.04 18.04: Pulling from library/ubuntu 284055322776: Pull complete Digest: sha256:0fedbd5bd9fb72089c7bbca476949e10593cebed9b1fb9edf5b79dbbacddd7d6 Status: Downloaded newer image for ubuntu:18.04 docker.io/library/ubuntu:18.04 C:\Users\sadaf>docker images REPOSITORY TAG IMAGE ID CREATED SIZE ubuntu latest ba6acccedd29 6 hours ago 72.8MB ubuntu 18.04 5a214d77f5d7 2 weeks ago 63.1MB

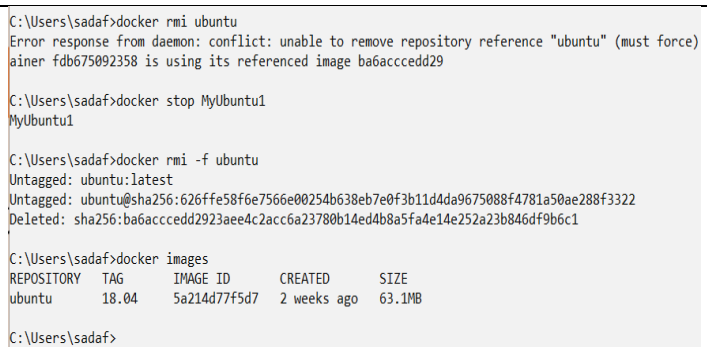
-q: Only show image IDs
-f: Filter output based on conditions provided
-a: Show all images (default hides intermediate images)
--name: Assign a name to the container

```
C:\Users\sadaf>docker images -q
ba6acccedd29
5a214d77f5d7

C:\Users\sadaf>docker images -f "dangling=false"
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       latest    ba6acccedd29   6 hours ago   72.8MB
ubuntu       18.04     5a214d77f5d7   2 weeks ago   63.1MB

C:\Users\sadaf>docker images -f "dangling=true"
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE

C:\Users\sadaf>docker images -f "dangling=false" -q
ba6acccedd29
5a214d77f5d7

C:\Users\sadaf>docker images -a
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       latest    ba6acccedd29   6 hours ago   72.8MB
ubuntu       18.04     5a214d77f5d7   2 weeks ago   63.1MB
```

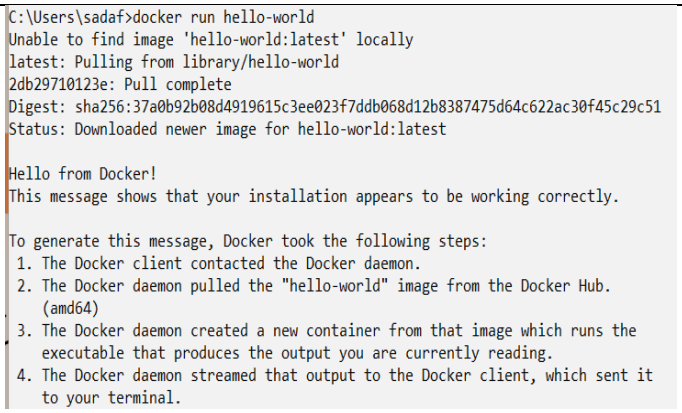
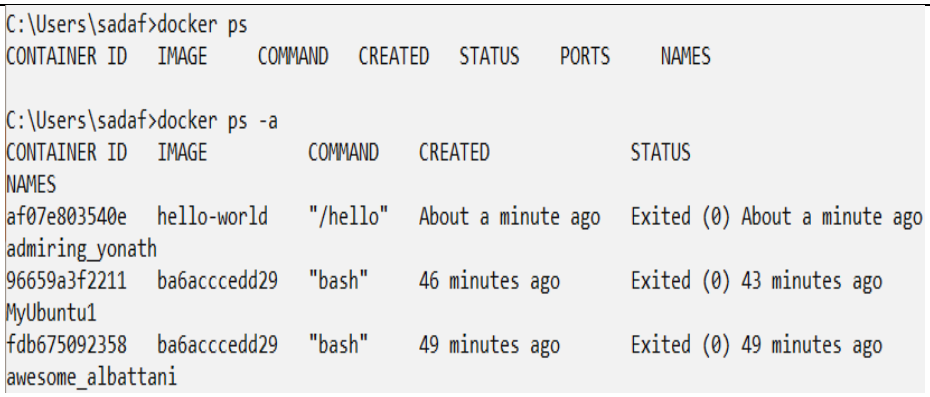
```
C:\Users\sadaf>docker run ubuntu

C:\Users\sadaf>docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS                      PORTS     NAMES
fdb675092358   ubuntu    "bash"    30 seconds ago   Exited (0) 24 seconds ago             awesome_albatt
ani

C:\Users\sadaf>docker run --name MyUbuntu1 -it ubuntu bash
root@96659a3f2211:/#
root@96659a3f2211:/# ls -l
total 48
lrwxrwxrwx   1 root root    7 Oct  6 16:47 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 15  2020 boot
drwxr-xr-x   5 root root  360 Oct 16 06:35 dev
drwxr-xr-x   1 root root 4096 Oct 16 06:35 etc
drwxr-xr-x   2 root root 4096 Apr 15  2020 home
lrwxrwxrwx   1 root root    7 Oct  6 16:47 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Oct  6 16:47 lib32 -> usr/lib32
lrwxrwxrwx   1 root root    9 Oct  6 16:47 lib64 -> usr/lib64
lrwxrwxrwx   1 root root   10 Oct  6 16:47 libx32 -> usr/libx32
drwxr-xr-x   2 root root 4096 Oct  6 16:47 media
drwxr-xr-x   2 root root 4096 Oct  6 16:47 mnt
drwxr-xr-x   2 root root 4096 Oct  6 16:47 opt
dr-xr-xr-x 200 root root    0 Oct 16 06:35 proc
drwx------   2 root root 4096 Oct  6 16:58 root
```

docker inspect: Return low-level information on Docker objects

```
C:\Users\sadaf>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
96659a3f2211   ubuntu    "bash"    47 seconds ago   Up 43 seconds             MyUbuntu1

C:\Users\sadaf>docker inspect ubuntu
[
    {
        "Id": "sha256:ba6acccedd2923aee4c2acc6a23780b14ed4b8a5fa4e14e252a23b846df9b6c1",
        "RepoTags": [
            "ubuntu:latest"
        ],
        "RepoDigests": [
            "ubuntu@sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2021-10-16T00:37:47.578710012Z",
        "Container": "249e88be79ad9986a479c71c15a056946ae26b0c54c1f634f115be6d5f9ba1c8",
        "ContainerConfig": {
            "Hostname": "249e88be79ad",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
```

rmi: Remove one or more images stop: Stop one or more running containers	

- Docker Container Commands:

docker run: Run a command in a new container. The docker run command first creates a writeable container layer over the specified image, and then starts it using the specified command.	
Ps: List containers -a: Show all containers (default shows just running)	
Start: Start one or more stopped containers Stop: Stop one or more running containers	

Pause: Pause all processes within one or more containers Top: Display the running processes of a container	C:\Users\sadaf>docker start MyUbuntu2 MyUbuntu2 C:\Users\sadaf>docker pause MyUbuntu2 MyUbuntu2 C:\Users\sadaf>docker top MyUbuntu2 UID PID PPID C STIME Y TIME CMD root 1466 1444 0 07:28 00:00:00 bash
Stats: Display a live stream of container(s) resource usage statistics	C:\Users\sadaf>docker stats MyUbuntu2 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 836B / 0B 0B / 0B 1 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1 CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS 76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1
Attach: Attach local standard input, output, and error streams to a running container	C:\Users\sadaf>docker attach MyUbuntu2 root@76663ea7ce86:/# ls -l total 48 lrwxrwxrwx 1 root root 7 Oct 6 16:47 bin -> usr/bin drwxr-xr-x 2 root root 4096 Apr 15 2020 boot drwxr-xr-x 5 root root 360 Oct 16 07:28 dev drwxr-xr-x 1 root root 4096 Oct 16 07:23 etc drwxr-xr-x 2 root root 4096 Apr 15 2020 home lrwxrwxrwx 1 root root 7 Oct 6 16:47 lib -> usr/lib lrwxrwxrwx 1 root root 9 Oct 6 16:47 lib32 -> usr/lib32 lrwxrwxrwx 1 root root 9 Oct 6 16:47 lib64 -> usr/lib64 lrwxrwxrwx 1 root root 10 Oct 6 16:47 libx32 -> usr/libx32 drwxr-xr-x 2 root root 4096 Oct 6 16:47 media drwxr-xr-x 2 root root 4096 Oct 6 16:47 mnt drwxr-xr-x 2 root root 4096 Oct 6 16:47 opt dr-xr-xr-x 213 root root 0 Oct 16 07:28 proc drwx------ 1 root root 4096 Oct 16 07:27 root drwxr-xr-x 5 root root 4096 Oct 6 16:58 run lrwxrwxrwx 1 root root 8 Oct 6 16:47 sbin -> usr/sbin drwxr-xr-x 2 root root 4096 Oct 6 16:47 srv dr-xr-xr-x 11 root root 0 Oct 16 07:28 sys drwxrwxrwt 2 root root 4096 Oct 6 16:58 tmp drwxr-xr-x 13 root root 4096 Oct 6 16:47 usr drwxr-xr-x 11 root root 4096 Oct 6 16:58 var root@76663ea7ce86:/# exit exit

Rm: Remove one or more containers History: Show the history of an image/containers	

```
C:\Users\sadaf>docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
76663ea7ce86   ubuntu        "bash"     8 minutes ago    Exited (0) 14 seconds ago             MyUbuntu2
af07e803540e   hello-world   "/hello"   11 minutes ago   Exited (0) 11 minutes ago             admiring_yonath
96659a3f2211   ubuntu        "bash"     56 minutes ago   Exited (0) 52 minutes ago             MyUbuntu1
fdb675092358   ubuntu        "bash"     59 minutes ago   Exited (0) 58 minutes ago             awesome_albattani

C:\Users\sadaf>docker rm MyUbuntu1
MyUbuntu1

C:\Users\sadaf>docker history ubuntu
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
ba6acccedd29   7 hours ago   /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>      7 hours ago   /bin/sh -c #(nop) ADD file:5d68d27cc15a80653…   72.8MB
```

Conclusion:

Learned the concept of docker. Installed docker in our system. Executed docker images commands such as pull, inspect, basic images commands, rmi etc. Created docker containers and execute different containers commands.

Experiment no: 7

Aim: To learn Docker file instructions, build an image for a sample web application using Docker file.

Lab Outcome no: ITL503.5

Lab Outcome: To understand concept of containerization and Analyze the Containerization of OS images and deployment of applications over Docker.

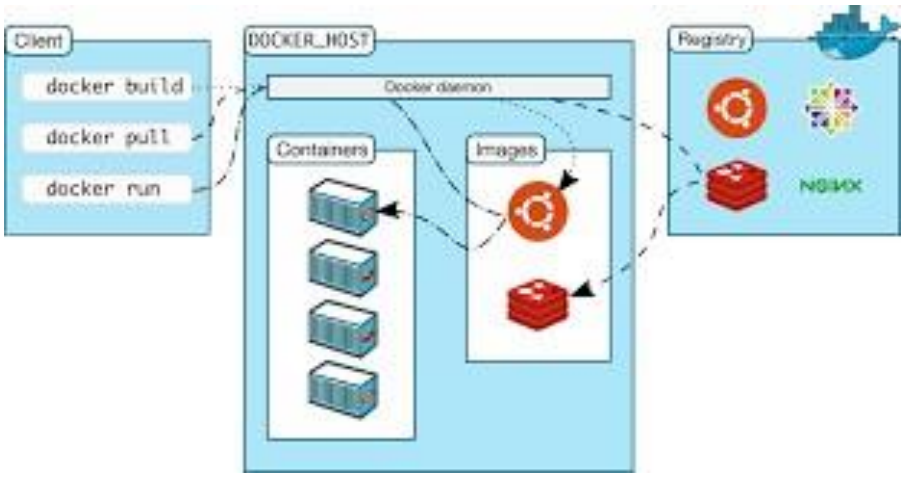
Theory:

Docker Architecture:

What Is Container (Docker)? Containers are a software package into a logical boxwith everything that the application needsto run. That includes the operating system, application code, runtime, system tools, systemlibraries, and etc. Dockercontainers are built off Docker images. Since images are read-only, Docker adds aread-write file system over the read-only file system of the image to create a container. Containers are compared with virtual machines (VMs). VMs are the guest operating system such as Linux or Windows runs on top of a host operatingsystem with virtualized access to the underlying hardware. Containers allow you to package your application together with libraries and other dependencies, providing isolated environments for running your software services.

What is Docker? Docker is an open-source platform based on Linux containers fordeveloping and running applications inside containers. Docker is used to deploy many containers simultaneously on a given host. Containers are very fast and lightweight because they don't need the extra load of a hypervisor as they run
directly within the host machine's kernel. Docker Architecture and Components Docker uses a client-server architecture. The docker client talks to the Docker daemon, which used to building, running, and distributing the Docker containers.

The Docker client and daemon communicate using a REST API, over UNIX sockets,or a network interface.



There are five major components in the Docker architecture:

a) Docker Daemon listens to Docker API requests and manages Docker objectssuch as images, containers, networks and volumes.

b) Docker Clients: With the help of Docker Clients, users can interact with Docker. Docker client provides a command-line interface (CLI) that allowsusers to run, and stop application commands to a Docker daemon.

c) Docker Host provides a complete environment to execute and run applications. It comprises of the Docker daemon, Images, Containers,Networks, and Storage.

d) Docker Registry stores Docker images. Docker Hub is a public registry thatanyone can use, and Docker is configured to use images on Docker Hub bydefault. You can run your own registry on it.

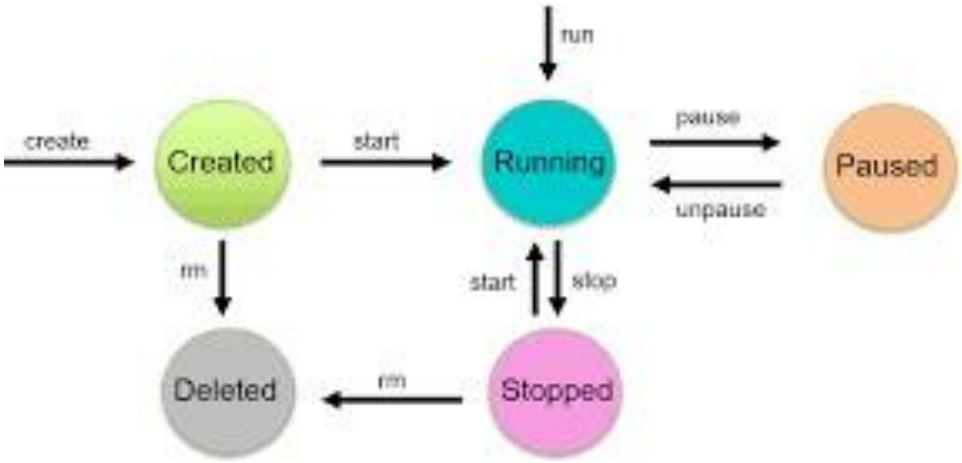
e) Docker Images are read-only templates that you build from a set of

instructions written in Dockerfile. Images define both what you want your packaged application and its dependencies to look like what processes to runwhen it’s launched.

- Docker Engine Components Docker Engine is the layer on which Dockerruns. It is installed on the host machine. It’s a lightweight runtime and tooling that manages containers, images, builds, and more.

There are three components in the Docker Engine:
a) Server: It is the docker daemon called dockerd. It can create and managedocker images, i.e, Containers, networks.
b) Rest API: It is used to instruct docker daemon what to do.
c) Command Line Interface (CLI): It is a client that is used to enter dockercommands

There are different stages when we create a container which is known as Lifecycleof container i.e create, run, pause, delete & stopped.



· The first phase is the created state. Further, the container moves into therunning state while we use the Docker run command.

· We can stop or pause the container, using Docker stop/pause command. And, toput a container back from a stopped state to a running state, we use the Docker run command.
· We can delete a running or stopped container, using Docker rm command.

- Step-By-Step Docker Installation on Windows :

1. Go to the website https://docs.docker.com/docker-for-windows/install/ and download the docker file. Note: A 64-bit processor and 4GB system RAM are thehardware prerequisites required to successfully run Docker on Windows 10.
2. Then, double-click on the Docker Desktop Installer.exe to run the installer. Note: Suppose the installer (Docker Desktop Installer.exe) is not downloaded; youcan get it from Docker Hub and run it whenever required.
3. Once you start the installation process, always enable Hyper-V WindowsFeature on the Configuration page.
4. Then, follow the installation process to allow the installer and wait till theprocess is done.
5. After completion of the installation process, click Close and restart.

Conclusion:

In this experiment, we have learned and studied about Dockerfile. A DockerFile isa text document that contains all the commands a user could call on the command line to assemble images.