Docker is a container management service.
It adheres to paradigm "Develop-ship and run anywhere"

Features of Docker
Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.
With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.
You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.

Docker Hub
Docker Hub is a registry service on the cloud that allows you to download Docker images that are built by other communities.
You can also upload your own Docker built images to Docker hub.

Step 1 - First you need to do a simple sign-up on Docker hub.
Step 2 - Once you have signed up, you will be logged into Docker Hub.
Step 3 - Next, let's browse and find the Jenkins image. (look for jenkins/jenkins)
Step 4 - If you scroll down on the same page, you can see the Docker pull command.
docker image ls will list the iamge for you
Step 5 - Now,from cmd run the following command -
--- docker pull hello-world
--- docker run hello-world
docker run hello-world explained
In Docker, everything is based on Images. An image is a combination of a file system and parameters. Let's take an example of the above run command in Docker.
The Docker command is specific and tells the Docker program on the Operating System that something needs to be done.
The run command is used to mention that we want to create an instance of an image, which is then called a container.
Finally, "hello-world" represents the image from which the container is made.

docker run --rm hello-world ----> removes the container after it exits

Giving a name to container
--- docker run --name myubuntu -it ubuntu /bin/bash

Run centos image on our windows machine
--- docker run -it centos /bin/bash

Here, centos is the name of the image we want to download from Docker Hub and install on our machine.
-it is used to mention that we want to run in interactive mode.
/bin/bash is used to run the bash shell once CentOS is up and running.
Type exit to exit and come back to cmd.

Displaying the docker images

---docker images
The output will provide the list of images on the system.
Each image has the following attributes -
TAG - This is used to logically tag images.
Image ID - This is used to uniquely identify the image.
Created - The number of days since the image was created.
Virtual Size - The size of the image.
/*************************
Access the linux docker vm on windows From elevated powershell run below command (--rm -v remove volume after container is removed)
docker run --rm -it -v /:/host alpine
Then when # prompt appears run
chroot /host
#docker images => will display same images as above

ls /var/lib/docker/containers => will show containers

**************************/
Remove the Docker images - docker rmi ImageID
--- docker rmi 7a86f8ffcb25

Get only the image ids of docker images
--- docker images -q

command to see the details of an image or container.(details like Labels,MacAddress,Memory,MemorySwap,NetworkDisabled,User,Volumes etc..)
--- docker inspect jenkins

Docker Containers
Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers.

docker ps
list all the running containers on the machine
--- docker ps
list all the containers on the machine
--- docker ps -a

docker history
see all the commands that were run with an image via a container - docker history ImageID
--- docker history centos

docker top
With this command, you can see the top processes within a container - docker top ContainerID (You can get containerID with docker ps-- Container must be running)
--- docker top 9f215ed0b0d3

docker stop
This command is used to stop a running container - docker stop ContainerId
--- docker stop 9f215ed0b0d3

docker start
This command is used to start a stopped container - docker start ContainerId
--- docker start 9f215ed0b0d3

docker restart
This command is used to restart a stopped/running container - docker restart ContainerId
--- docker restart 9f215ed0b0d3

Using restart policy
To configure the restart policy for a container, use the --restart flag when using the docker run command.
Values for restart can be one of the following
no - Do not automatically restart the container. (the default)
on-failure - Restart the container if it exits due to an error, which manifests as a non-zero exit code.
always - Always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted.
unless-stopped - Similar to always, except that when the container is stopped (manually or otherwise), it is not restarted even after Docker daemon restarts.

--- docker run -d --restart always redis

docker rm
This command is used to delete a container - docker rm ContainerID
--- docker rm 9f215ed0b0d3

docker stats
This command is used to provide the statistics of a running container - docker stats ContainerID
--- docker stats 9f215ed0b0d3
Above cmd will provide CPU and memory utilization of the said Container.

docker attach
This command is used to attach to a running container - docker attach ContainerID - Run this command from separate cmd prompt.
--- docker attach 07b0b6f434fe

docker detach [-d]
docker run -d -it alpine to access a detached container use attach to access this container.

docker pause

This command is used to pause the processes in a running container - docker pause ContainerID
--- docker pause 07b0b6f434fe
If you see the docker ps you will see the container paused state

docker unpause
This command is used to unpause the processes in a running container - docker unpause ContainerID
--- docker unpause 07b0b6f434fe

docker kill
This command is used to kill the processes in a running container - docker kill ContainerID
--- docker kill 07b0b6f434fe

Docker Files
Dockerfile for a simple Java Application
Create a folder java-app create the Dockerfile and Hello.java as mentioned below in this folder.
md java-app

```
# Dockerfile
FROM openjdk:11
MAINTAINER  Ameya Joshi <ameya@email.com>
COPY . /var/www/java
WORKDIR /var/www/java
RUN javac -d . Hello.java
CMD ["java", "test.Hello"]
```

```
package test;
class Hello{
public static void main(String[] args){
System.out.println("This is java app \n by using Docker");
}
}
```

--- docker build -f Dockerfile -t java-app:1 . (If reqd. docker login --username=ameyajoshi --email=amsalways@gmail.com)
-f specifies the Dockerfile. This can be skipped if the filename used at the beginning of this process is Dockerfile.
-t specifies the name of the image. The name java-app, and the 1 after the colon, specify the image tag.
This can be changed to any tag name that makes sense. If no tag is specified "latest" is used as default tag.
Do not forget the .(dot) at the end of command; it specifies the context of the build. The .(dot) at the end of the command specifies the current directory.
The files and directories of current directory will be sent to Docker daemon as a

build artifact.
--- docker run java-app:8
This should display Hello World

docker - Managing Ports
docker run -p hostPort:containerPort container/Image
--- docker inspect jenkins/jenkins will show you the ports on which jenkins is listening in the container.
--- docker run -p 9001:8080 -p 40000:50000 jenkins/jenkins
Here host's 9001 port is mapped with container's 8080 port and host's 40000 port is mapped with container's 50000 port