Docker is a container management service. The keywords of Docker are develop, ship and run anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.

The initial release of Docker was in March 2013 and since then, it has become the buzzword for modern world development, especially in the face of Agile-based projects.

Docker, a subset of the Moby project, is a software framework for building, running, and managing containers on servers and the cloud. The term "docker" may refer to either the tools (the commands and a daemon) or to the Dockerfile file format.

It used to be that when you wanted to run a web application, you bought a server, installed Linux, set up a LAMP stack, and ran the app. If your app got popular, you practiced good load balancing by setting up a second server to ensure the application wouldn't crash from too much traffic.

Times have changed, though, and instead of focusing on single servers, the Internet is built upon arrays of inter-dependent and redundant servers in a system commonly called "the cloud". Thanks to innovations like Linux kernel namespaces and cgroups, the concept of a server could be removed from the constraints of hardware and instead became, essentially, a piece of software. These software-based servers are called containers, and they're a hybrid mix of the Linux OS they're running on plus a hyper-localized runtime environment (the contents of the container).

Docker, a subset of the Moby project, is a software framework for building, running, and managing containers on servers and the cloud. The term "docker" may refer to either the tools (the commands and a daemon) or to the Dockerfile file format.

It used to be that when you wanted to run a web application, you bought a server, installed Linux, set up a LAMP stack, and ran the app. If your app got popular, you practiced good load balancing by setting up a second server to ensure the application wouldn't crash from too much traffic.

Times have changed, though, and instead of focusing on single servers, the Internet is built upon arrays of inter-dependent and redundant servers in a system commonly called "the cloud". Thanks to innovations like Linux kernel namespaces and cgroups, the concept of a server could be removed from the constraints of hardware and instead became, essentially, a piece of software. These software-based servers are called containers, and they're a hybrid mix of the Linux OS they're running on plus a hyper-localized runtime environment (the contents of the container)

PS C:\Users\Kranti\Desktop> docker --help

Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:

run Create and run a new container from an image

exec Execute a command in a running container

ps List containers

build Build an image from a Dockerfile

pull Download an image from a registry

push Upload an image to a registry

images List images

login Log in to a registry

logout Log out from a registry

search Search Docker Hub for images

version Show the Docker version information

info Display system-wide information

Management Commands:

builder Manage builds

buildx\* Docker Buildx (Docker Inc., v0.11.0)

compose\* Docker Compose (Docker Inc., v2.19.1)

container Manage containers

context Manage contexts

dev\* Docker Dev Environments (Docker Inc., v0.1.0)

extension\* Manages Docker extensions (Docker Inc., v0.2.20)

image Manage images

init\* Creates Docker-related starter files for your project (Docker Inc., v0.1.0-beta.6)

manifest Manage Docker image manifests and manifest lists

network Manage networks

plugin Manage plugins

sbom\* View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc., 0.6.0)

scan\* Docker Scan (Docker Inc., v0.26.0)

scout\* Command line tool for Docker Scout (Docker Inc., 0.16.1)

system Manage Docker

trust Manage trust on Docker images

volume Manage volumes

Swarm Commands:

swarm Manage Swarm

Commands:

attach Attach local standard input, output, and error streams to a running container

commit Create a new image from a container's changes

cp Copy files/folders between a container and the local filesystem

create Create a new container

diff Inspect changes to files or directories on a container's filesystem

events Get real time events from the server

export Export a container's filesystem as a tar archive

history Show the history of an image

import Import the contents from a tarball to create a filesystem image

inspect Return low-level information on Docker objects

kill Kill one or more running containers

load Load an image from a tar archive or STDIN

logs Fetch the logs of a container

pause Pause all processes within one or more containers

port List port mappings or a specific mapping for the container

rename Rename a container

restart Restart one or more containers

rm Remove one or more containers

rmi Remove one or more images

save Save one or more images to a tar archive (streamed to STDOUT by default)

start Start one or more stopped containers

stats Display a live stream of container(s) resource usage statistics

stop Stop one or more running containers

tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

top Display the running processes of a container

unpause Unpause all processes within one or more containers

update Update configuration of one or more containers

wait Block until one or more containers stop, then print their exit codes

Global Options:

--config string Location of client config files (default

"C:\\Users\\Kranti\\.docker")

-c, --context string Name of the context to use to connect to the

daemon (overrides DOCKER\_HOST env var and

default context set with "docker context use")

-D, --debug Enable debug mode

-H, --host list Daemon socket to connect to

-l, --log-level string Set the logging level ("debug", "info",

"warn", "error", "fatal") (default "info")

--tls Use TLS; implied by --tlsverify

--tlscacert string Trust certs signed only by this CA (default

"C:\\Users\\Kranti\\.docker\\ca.pem")

--tlscert string Path to TLS certificate file (default

"C:\\Users\\Kranti\\.docker\\cert.pem")

--tlskey string Path to TLS key file (default

"C:\\Users\\Kranti\\.docker\\key.pem")

--tlsverify Use TLS and verify the remote

-v, --version Print version information and quit

Run 'docker COMMAND --help' for more information on a command.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1] docker pull mysql

2] docker run --name mysqlDb -e MYSQL\_ROOT\_PASSWORD=root -d mysql

3] docker exec -it mysqlDb bash

4] mysql -p

use the password to start

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

31] create a folder

open in vscode

create a Dockerfile

32] install docker extension

33] docker build -t my\_ubuntu\_image .

FROM ubuntu

MAINTAINER shital

RUN apt update

CMD [ "echo","this is my ubuntu image" ]

34] docker pull ubuntu

35] docker run --name ubuntu my\_ubuntu\_image

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*Docker using Java

1] docker pull openjdk

2] create a folder and add

Dockerfile

Test.java

3] docker build -t myjavaimage .

4] docker images

5] docker run --name javaProject myjavaimage

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*