**DOCKER**

**Introduction to Docker**

Docker is a set of platforms as a service (PaaS) products that use the Operating system level visualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than a virtual machine.

**Difference between Docker Containers and Virtual Machines**

**1. Docker Containers**

* Docker Containers contain binaries, libraries, and configuration files along with the application itself.
* They don’t contain a guest OS for each container and rely on the underlying OS kernel, which makes the containers lightweight.
* Containers share resources with other containers in the same host OS and provide OS-level process isolation.

**2. Virtual Machines**

* Virtual Machines (VMs) run on Hypervisors, which allow multiple Virtual Machines to run on a single machine along with its own operating system.
* Each VM has its own copy of an operating system along with the application and necessary binaries, which makes it significantly larger and it requires more resources.
* They provide Hardware-level process isolation and are slow to boot.

**Important Terminologies in Docker**

**1. Docker Image**

* It is a file, comprised of multiple layers, used to execute code in a Docker container.
* They are a set of instructions used to create docker containers.

**2. Docker Container**

* It is a runtime instance of an image.
* Allows developers to package applications with all parts needed such as libraries and other dependencies.

**3. Docker file**

* It is a text document that contains necessary commands which on execution helps assemble a Docker Image.
* Docker image is created using a Docker file.

**4. Docker Engine**

* The software that hosts the containers is named Docker Engine.
* Docker Engine is a client-server based application
* The docker engine has **3 main** components:
  + **Server**: It is responsible for creating and managing Docker images, containers, networks, and volumes on the Docker. It is referred to as a daemon process.
  + **REST API**: It specifies how the applications can interact with the Server and instructs it what to do.
  + **Client**: The Client is a docker command-line interface (CLI), that allows us to interact with Docker using the docker commands.

**5. Docker Hub**

* Docker Hub is the official online repository where you can find other Docker Images that are available for use.
* It makes it easy to find, manage, and share container images with others.

**Installing Docker on Ubuntu**

**1. Remove old version of Docker**

$ sudo apt-get remove docker docker-engine docker.io containerd runc

**2. Installing Docker Engine**

$ sudo apt-get update

$ sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

$ sudo apt-key fingerprint 0EBFCD88

$ sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable nightly test"

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io

 Check if docker is successfully installed in your system

$ sudo docker run hello-world

**Create an application in Docker**

**1. Create a folder with 2 files (Dockerfile and main.py file) in it.**

* Dockerfile
* main.py

**2. Edit main.py with the below code.**

Python3

#!/usr/bin/env python

Print(“Docker and RU Rocks!”)

**3. Edit Dockerfile with the below commands.**

FROM python:latest

COPY main.py /

CMD [ "python", "./main.py" ]

**4. Create a Docker image.**

Once you have created and edited the main.py file and the Dockerfile, create your image to contain your application.

$ docker build -t python-test .

The ‘-t’ option allows to define the name of your image. ‘python-test’ is the name we have chosen for the image.

**5. Run the Docker image**

Once the image is created, your code is ready to launch.

$ docker run python-test

**Push an image to Docker Hub**

**1.** Create an Account on Docker Hub.

**2.** Click on the “Create Repository” button, put the name of the file, and click on “Create”.

**3.** Now will “tag our image” and “push it to the Docker Hub repository” which we just created.

Now, run the below command to list docker images:

$ docker images

The above will give us this result

REPOSITORY TAG IMAGE\_ID CREATED SIZE afrozchakure/python-test latest c7857f97ebbd 2 hours ago 933MB

Image ID is used to tag the image. The syntax to tag the image is:

docker tag <image-id> <your dockerhub username>/python-test:latest

$ docker tag c7857f97ebbd afrozchakure/python-test:latest

**4.**Push image to Docker Hub repository

$ docker push afrozchakure/python-test

**Fetch and run the image from Docker Hub**

**1.**To remove all versions of a particular image from our local system, we use the Image ID for it.

$ docker rmi -f af939ee31fdc

**2.**Now run the image, it will fetch the image from the docker hub if it doesn’t exist on your local machine.

$ docker run afrozchakure/python-test

**Virtualization with Docker Containers**

In a software driven world where omnipresence and ease of deployment with minimum overheads are the major requirements, the cloud promptly takes its place in every picture. Containers are creating their mark in this vast expanse of cloud space with the world’s top technology and IT establishments relying on the concept for their infrastructural necessities. Tech giants like Facebook, Google and Microsoft use containers in their streamlined processes to facilitate a secure and easy deployment into the cloud production environments. This deployment with containers offers a technique, which abstracts the application from the run-time environment; much like virtual machines and this is done, of course by virtualization.

The two core concepts to be explored here are:

1. How containers are built on virtualization technique
2. How they offer an alternative to virtual machines

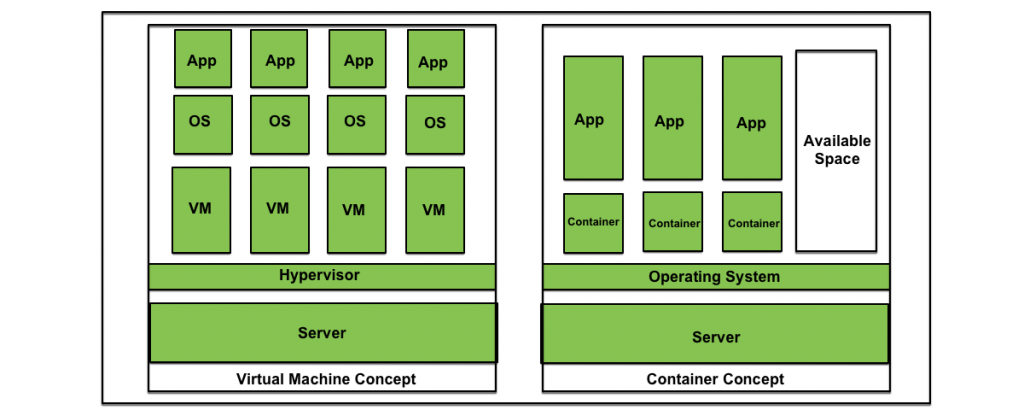
**1.** To answer how containers enable virtualization let us discuss in detail what is virtualization and how containers come into play with this concept.

Virtualization is a technique, which essentially creates an illusion of a resource such as a desktop, storage, network or an operating system. Devices, applications and human users possess the capability of interacting with these resources. This illusion also called virtualization expands the capabilities of traditional systems, which are limited by their own physical resources.

Now, containers enable this virtualization for applications that are deployed in them. Applications in containers run independently, isolated from any physical resource. Containers virtualize the OS, CPU, memory, storage and network resources there by providing a controlled environment that can be scaled up or down as required. A container also packages the application along with its dependencies and necessary files, which enables the application to be deployed on any environment without having to configure the server, hardware or software

This still sounds similar to virtualization implemented through virtual machines right?  
So before we get into what more containers can do let us clear up how containers differ from virtual machines

**2.** A virtual machine is a form of hardware virtualization. The hardware is logically separated from the other resources. The hardware can be any system such as a desktop (with hardware and its own OS) called the host machine, on which several virtual machines or guest machines can run, each with their own separate operating systems. This is made possible by a firmware called the hypervisor.



Containers like virtual machines, run on a host machines. They can also be controlled as needed and provide isolation of application from the run-time environment. However, containers aim at operating system virtualization. Unlike virtual machines, in a containerized system guest operating systems need not be installed on the host system. Instead, the underlying host operating system is shared by all instances of containers containing the application code. This makes containers light weight and reduces operating system overheads associated with virtual machines such as patching, upgrading, driver support etc. Moreover, it reduces the amount of resources of the host machine consumed by the presence of individual operating systems.

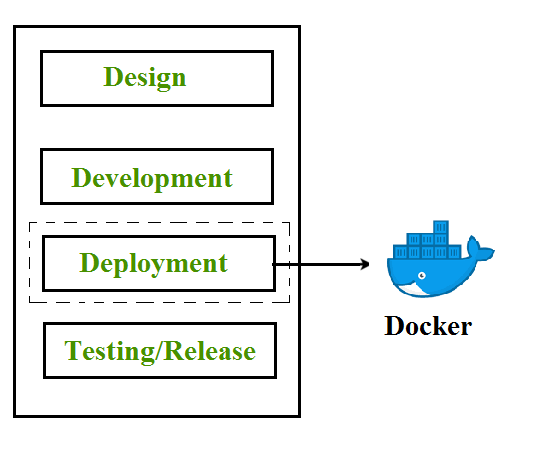
**Benefits of containers:**

* Applications can be deployed without any worry about the run time environment. As a result, an application can easily be moved through the software development cycle and can run anywhere, for example, on Mac OS, Linux, Windows and even in data centers. This results in less expenditure of time on examining the environment and more time can be time on developing new functionality.
* Multiple containers with applications can be run on the same instance of physical resource sharing an operating system. These containers being lightweight are fast and efficiently utilize the computing resources available.
* Containers are isolated from one another, which gives the developer the leeway to split application services into different containers. These containers do not share any dependencies and each can be manipulated and updated by the developer at will.
* Containers come with the inbuilt capability of version control.

# Containerization using Docker

**Docker** is the containerization platform which is used to package your application and all its dependencies together in the form of containers so to make sure that your application works seamlessly in any environment which can be development or test or production. Docker is a tool designed to make it easier to create, deploy, run applications by using containers. 

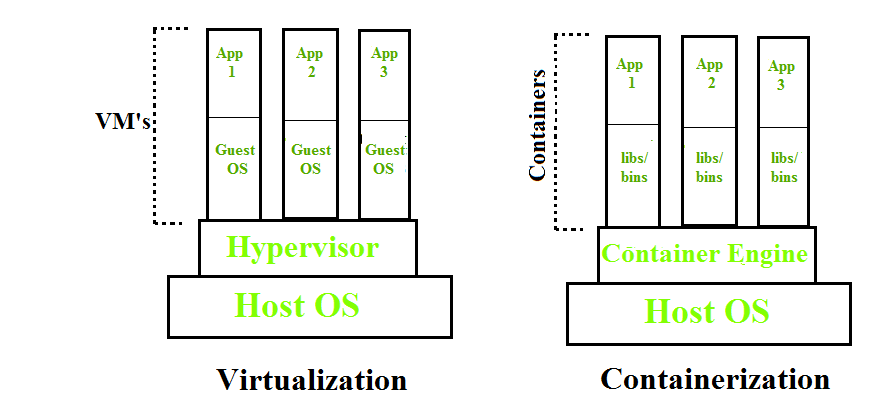
Docker is the world’s leading software container platform. It was launched in 2013 by a company called Dotcloud, Inc which was later renamed as Docker, Inc. It is written in the Go language. It has been just six years since Docker was launched yet communities have already shifted to it from VM’s. Docker is designed to benefit both developers and system administrators making it a part of many DevOps toolchains. Developers can write code without worrying about the testing and production environment. Sysadmins need not worry about infrastructure as Docker can easily scale up and scale down the number of systems. Docker comes into play at the deployment stage of the software development cycle. 



## Containerization

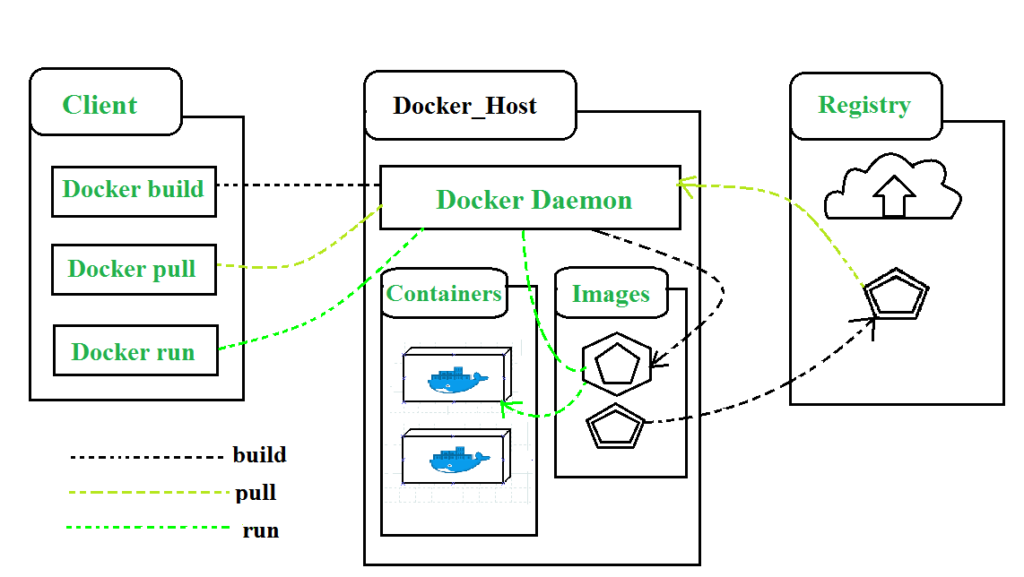
Containerization is OS-based virtualization which creates multiple virtual units in the user space, known as Containers. Containers share the same host kernel but are isolated from each other through private namespaces and resource control mechanisms at the OS level. Container-based Virtualization provides a different level of abstraction in terms of virtualization and isolation when compared with hypervisors. Hypervisors use a lot of hardware which results in overhead in terms of virtualizing hardware and virtual device drivers. A full operating-system (e.g -Linux, Windows) run on top of this virtualized hardware in each virtual machine instance.   
But in contrast, containers implement isolation of processes at the operating system level, thus avoiding such overhead. These containers run on top of the same shared operating system kernel of the underlying host machine and one or more processes can be run within each container. In containers you don’t have to pre-allocate any RAM, it is allocated dynamically during the creation of containers while in VM’s you need to first pre-allocate the memory and then create the virtual machine. Containerization has better resource utilization compared to VMs and a short boot-up process. It is the next evolution in virtualization.

Containers are able to run virtually anywhere, greatly easy development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or bare metal, on a developer’s machine or in data centers on-premises; and of course, in the public cloud. Containers virtualize CPU, memory, storage, and network resources at the OS-level, providing developers with a sandboxed view of the OS logically isolated from other applications. Docker is the most popular open-source container format available and is supported on Google Cloud Platform and by Google Kubernetes Engine.



## Docker Architecture

Docker architecture consists of Docker client, Docker Daemon running on Docker Host, and Docker Hub repository. Docker has client-server architecture in which the client communicates with the Docker Daemon running on the Docker Host using a combination of REST APIs, Socket IO, and TCP. If we have to build the Docker image, then we use the client to execute the build command to Docker Daemon then Docker Daemon builds an image based on given inputs and saves it into the Docker registry. If you don’t want to create an image then just execute the pull command from the client and then Docker Daemon will pull the image from the Docker Hub and finally if we want to run the image then execute the run command from the client which will create the container.

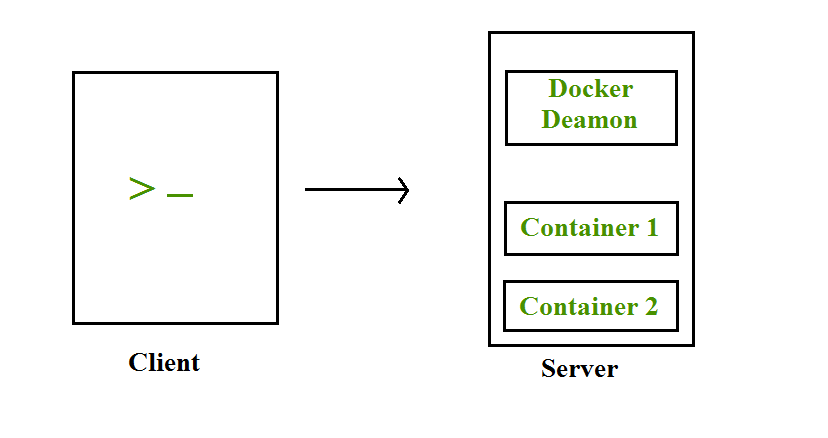


## **Components of Docker**

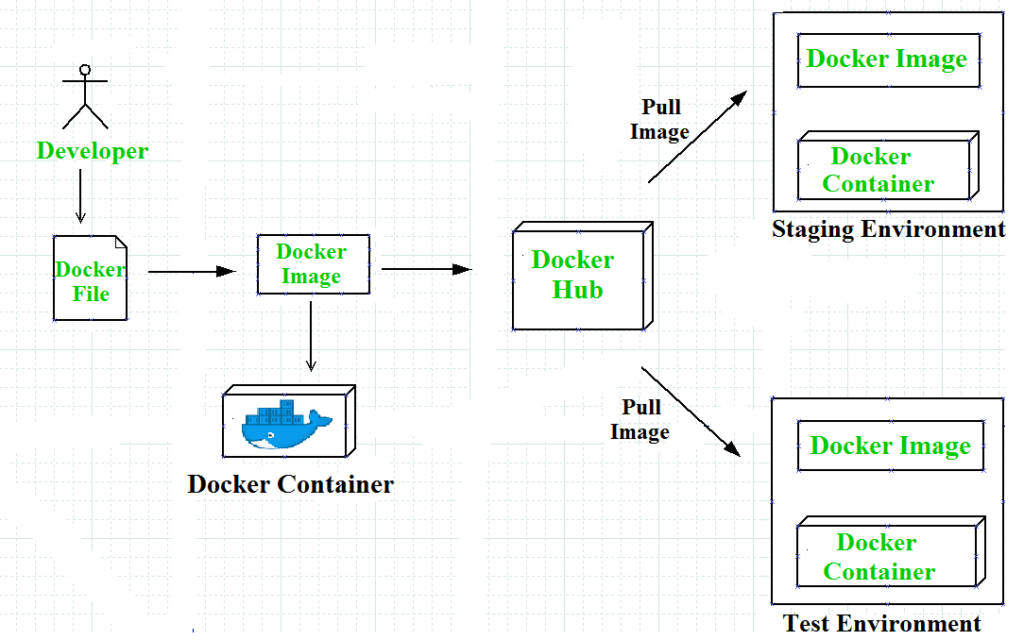
The main components of Docker include – Docker clients and servers, Docker images, Dockerfile, Docker Registries, and Docker containers. These components are explained in

Details in the below section:

1. **Docker Clients and Servers**– Docker has a client-server architecture. The Docker Daemon/Server consists of all containers. The Docker Daemon/Server receives the request from the Docker client through CLI or REST APIs and thus processes the request accordingly. Docker client and Daemon can be present on the same host or different host.



1. **Docker Images**– Docker images are used to build docker containers by using a read-only template. The foundation of every image is a base image for eg. base images such as – ubuntu14.04 LTS, Fedora 20. Base images can also be created from scratch and then required applications can be added to the base image by modifying it thus this process of creating a new image is called “committing the change”.
2. **Docker File**– Dockerfile is a text file that contains a series of instructions on how to build your Docker image. This image contains all the project code and its dependencies. The same Docker image can be used to spin ‘n’ number of containers each with modification to the underlying image. The final image can be uploaded to Docker Hub and share among various collaborators for testing and deployment. The set of commands that you need to use in your Docker File are FROM, CMD, ENTRYPOINT, VOLUME, ENV, and many more.
3. **Docker Registries**– Docker Registry is a storage component for Docker images. We can store the images in either public/private repositories so that multiple users can collaborate in building the application. Docker Hub is Docker’s own cloud repository. Docker Hub is called a public registry where everyone can pull available images and push their own images without creating an image from scratch.
4. **Docker Containers**– Docker Containers are runtime instances of Docker images. Containers contain the whole kit required for an application, so the application can be run in an isolated way. For eg.- Suppose there is an image of Ubuntu OS with NGINX SERVER when this image is run with docker run command, then a container will be created and NGINX SERVER will be running on Ubuntu OS.



## **Docker Compose**

Docker Compose is a tool with which we can create a multi-container application. It makes it easier to configure and   
run applications made up of multiple containers. For example, suppose you had an application that required WordPress and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately. We define a multi-container application in a YAML file. With the docker-compose up command, we can start the application in the foreground. Docker-compose will look for the docker-compose.yaml file in the current folder to start the application. By adding the -d option to the docker-compose up command, we can start the application in the background. Creating a docker-compose.yaml file for WordPress application :

#cat docker-compose.yaml

version: ’2’

services:

db:

image: mysql:5.7

volumes:db\_data:/var/lib/mysql

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: wordpress

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

wordpress:

depends\_on:

- db

image: wordpress:latest

ports:

- "8000:80"

restart: always

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_PASSWORD: wordpress

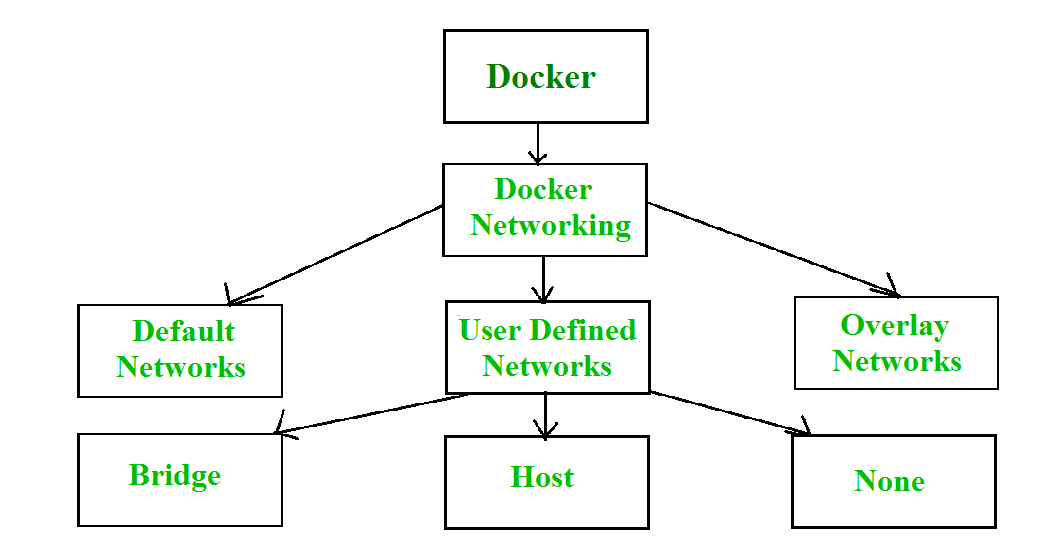
volumes:

db\_data:

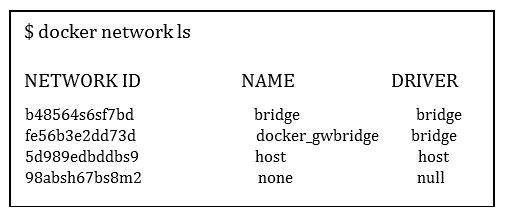
In this docker-compose.yaml file, we have the following ports section for the WordPress container, which means that we are going to map the host’s 8000 port with the container’s 80 port. So that host can access the application with its IP and port no.

## Docker Networks

When we create and run a container, Docker by itself assigns an IP address to it, by default. Most of the time, it is required to create and deploy Docker networks as per our needs. So, Docker let us design the network as per our requirements. There are three types of Docker networks- default networks, user-defined networks, and overlay networks. 



To get list of all the default networks that Docker creates, we run the command shown below – 



There are three types of networks in Docker – 

1. **Bridged network**: When a new Docker container is created without the –network argument, Docker by default connects the container with the bridge network. In bridged networks, all the containers in a single host can connect to each other through their IP addresses. Bridge network is created when the span of Docker hosts is one i.e. when all containers run on a single host. We need an overlay network to create a network that has a span of more than one Docker host.
2. **Host network**: When a new Docker container is created with the –network=host argument it pushes the container into the host network stack where the Docker daemon is running. All interfaces of the host are accessible from the container which is assigned to the host network.
3. **None network**: When a new Docker container is created with the –network=none argument it puts the Docker container in its own network stack. So, in this none network, no IP addresses are assigned to the container, because of which they cannot communicate with each other.

We can assign any one of the networks to the Docker containers. The –network option of the ‘docker run’ command is used to assign a specific network to the container. 

$docker run --network ="network name"

To get detailed information about a particular network we use the command- 

$docker network inspect "network name"

## Advantages of Docker –

Docker has become popular nowadays because of the benefits provided by Docker containers. The main advantages of Docker are: 

1. **Speed** – The speed of Docker containers compared to a virtual machine is very fast. The time required to build a container is very fast because they are tiny and lightweight. Development, testing, and deployment can be done faster as containers are small. Containers can be pushed for testing once they have been built and then from there on to the production environment.
2. **Portability** – The applications that are built inside docker containers are extremely portable. These portable applications can easily be moved anywhere as a single element and their performance also remains the same.
3. **Scalability** – Docker has the ability that it can be deployed in several physical servers, data servers, and cloud platforms. It can also be run on every Linux machine. Containers can easily be moved from a cloud environment to localhost and from there back to cloud again at a fast pace.
4. **Density** – Docker uses the resources that are available more efficiently because it does not use a hypervisor. This is the reason that more containers can be run on a single host as compared to virtual machines. Docker Containers have higher performance because of their high density and no overhead wastage of resources.