Docker Tutorial

Docker Tutorial provides basic and advanced concepts of Docker. Our Docker Tutorial is designed for both beginners as well as professionals.

Docker is a centralized platform for packaging, deploying, and running applications. Before Docker, many users face the problem that a particular code is running in the developer's system but not in the user's system. So, the main reason to develop docker is to help developers to develop applications easily, ship them into containers, and can be deployed anywhere.

Docker was firstly released in March 2013. It is used in the Deployment stage of the software development life cycle that's why it can efficiently resolve issues related to the application deployment

What is Docker?

Docker is an **open-source centralized platform designed** to create, deploy, and run applications. Docker uses **container** on the host's operating system to run applications. It allows applications to use the same **Linux kernel** as a system on the host computer, rather than creating a whole virtual operating system. Containers ensure that our application works in any environment like development, test, or production.

Docker includes components such as **Docker client, Docker server, Docker machine, Docker hub, Docker composes,** etc.

Let's understand the Docker containers and virtual machine.

Docker Containers

Docker containers are the **lightweight** alternatives of the virtual machine. It allows developers to package up the application with all its libraries and dependencies, and ship it as a single package. The advantage of using a docker container is that you don't need to allocate any RAM and disk space for the applications. It automatically generates storage and space according to the application requirement.

Virtual Machine

A virtual machine is a software that allows us to install and use other operating systems (Windows, Linux, and Debian) simultaneously on our machine. The operating system in which virtual machine runs are called virtualized operating systems. These virtualized operating systems can run programs and preforms tasks that we perform in a real operating system.

Containers Vs. Virtual Machine

Containers Vs. Virtual Machine

Containers	**Virtual Machine**
Integration in a container is faster and cheap.	Integration in virtual is slow and costly.
No wastage of memory.	Wastage of memory.
It uses the same kernel, but different distribution.	It uses multiple independent operating systems.

Why Docker?

Docker is designed to benefit both the Developer and System Administrator. There are the following reasons to use Docker -

- Docker allows us to easily install and run software without worrying about setup or dependencies.
- Developers use Docker to eliminate machine problems, i.e. "**but code is worked on my laptop**." when working on code together with co-workers.
- Operators use Docker to run and manage apps in isolated containers for better compute density.
- Enterprises use Docker to securely built agile software delivery pipelines to ship new application features faster and more securely.
- Since docker is not only used for the deployment, but it is also a great platform for development, that's why we can efficiently increase our customer's satisfaction.

Advantages of Docker

There are the following advantages of Docker -

- It runs the container in seconds instead of minutes.
- It uses less memory.
- It provides lightweight virtualization.
- It does not a require full operating system to run applications.
- It uses application dependencies to reduce the risk.
- Docker allows you to use a remote repository to share your container with others.
- It provides continuous deployment and testing environment.

Disadvantages of Docker

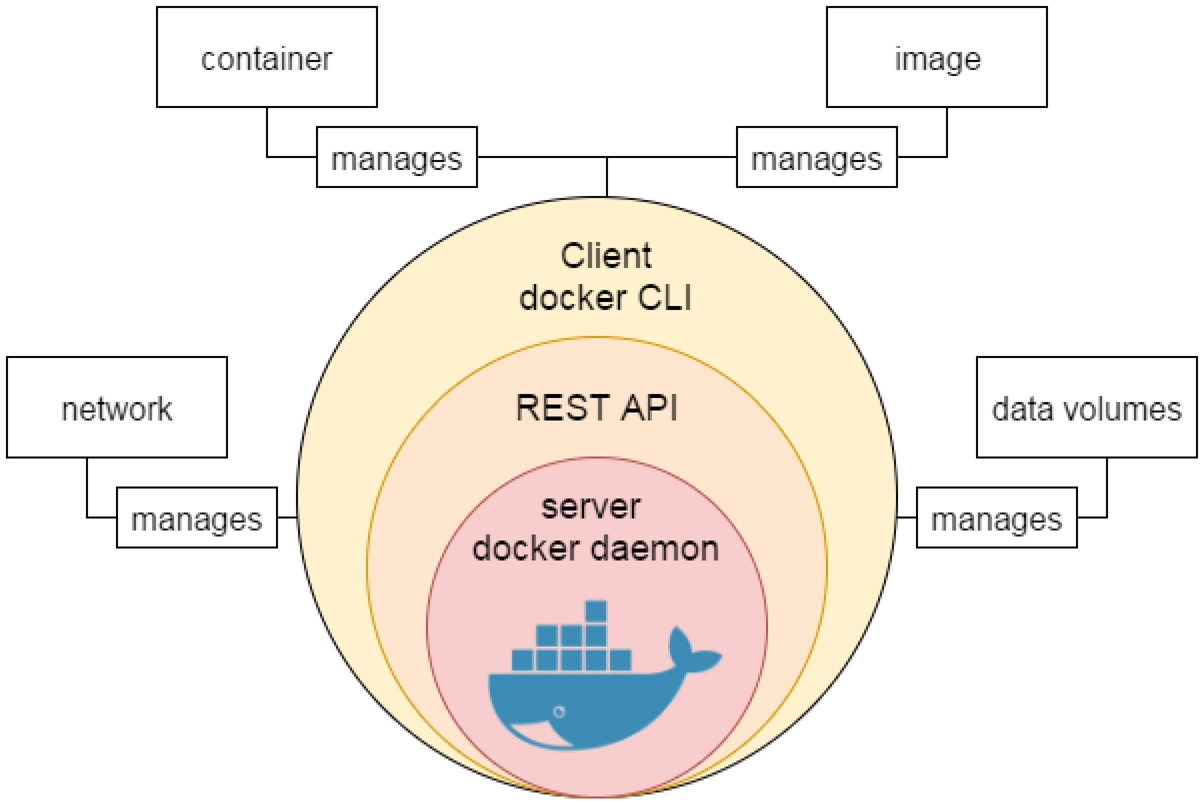
There are the following disadvantages of Docker -

- It increases complexity due to an additional layer.
- In Docker, it is difficult to manage large amount of containers.
- Some features such as container self -registration, containers self-inspects, copying files form host to the container, and more are missing in the Docker.
- Docker is not a good solution for applications that require rich graphical interface.

- Docker provides cross-platform compatibility means if an application is designed to run in a Docker container on Windows, then it can't run on Linux or vice versa.

Docker Engine

It is a client server application that contains the following major components.



Docker Features

Although Docker provides lots of features, we are listing some major features which are given below.

- Easy and Faster Configuration
- Increase productivity
- Application Isolation
- Swarm
- Routing Mesh

- Services
- Security Management

Easy and Faster Configuration

This is a key feature of docker that helps us to configure the system easily and faster.

We can deploy our code in less time and effort. As Docker can be used in a wide variety of environments, the requirements of the infrastructure are no longer linked with the environment of the application.

Increase productivity

By easing technical configuration and rapid deployment of application. No doubt it has increase productivity. Docker not only helps to execute the application in isolated environment but also it has reduced the resources.

Application Isolation

It provides containers that are used to run applications in isolation environment. Each container is independent to another and allows us to execute any kind of application.

Swarm

It is a clustering and scheduling tool for Docker containers. Swarm uses the Docker API as its front end, which helps us to use various tools to control it. It also helps us to control a cluster of Docker hosts as a single virtual host. It's a self-organizing group of engines that is used to enable pluggable backends.

Routing Mesh

It routes the incoming requests for published ports on available nodes to an active container. This feature enables the connection even if there is no task is running on the node.

Services

Services is a list of tasks that lets us specify the state of the container inside a cluster. Each task represents one instance of a container that should be running and Swarm schedules them across nodes.

Security Management

It allows us to save secrets into the swarm itself and then choose to give services access to certain secrets.

It includes some important commands to the engine like secret inspect, secret create etc.

Docker Architecture

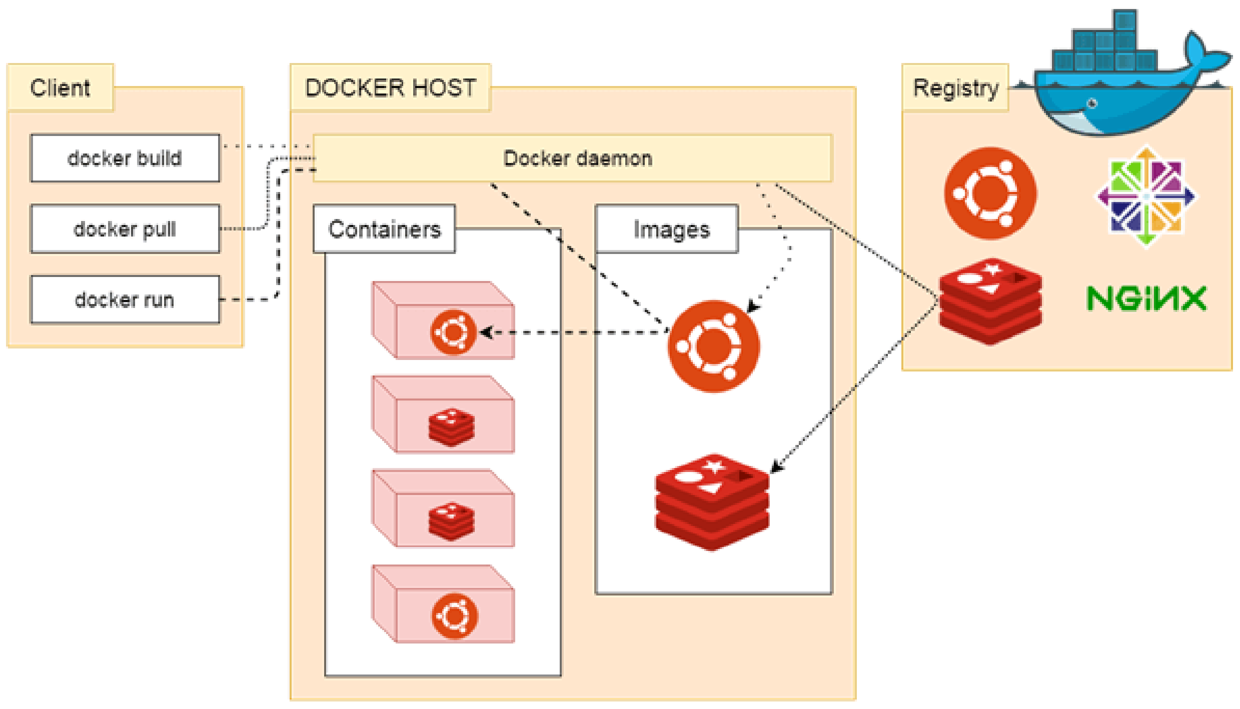
Before learning the Docker architecture, first, you should know about the Docker Daemon.

What is Docker daemon?

Docker daemon runs on the host operating system. It is responsible for running containers to manage docker services. Docker daemon communicates with other daemons. It offers various Docker objects such as images, containers, networking, and storage. s

Docker architecture

Docker follows Client-Server architecture, which includes the three main components that are **Docker Client**, **Docker Host**, and **Docker Registry**.



1. Docker Client

Docker client uses **commands** and **REST APIs** to communicate with the Docker Daemon (Server). When a client runs any docker command on the docker client terminal, the client terminal sends these docker commands to the Docker daemon. Docker daemon receives these commands from the docker client in the form of command and REST API's request.

Docker Client uses Command Line Interface (CLI) to run the following commands -

docker build

docker pull

docker run

2. Docker Host

Docker Host is used to provide an environment to execute and run applications. It contains the docker daemon, images, containers, networks, and storage.

3. Docker Registry

Docker Registry manages and stores the Docker images.

There are two types of registries in the Docker -

Pubic Registry - Public Registry is also called as **Docker hub**.

Private Registry - It is used to share images within the enterprise.

Docker Objects

There are the following Docker Objects -

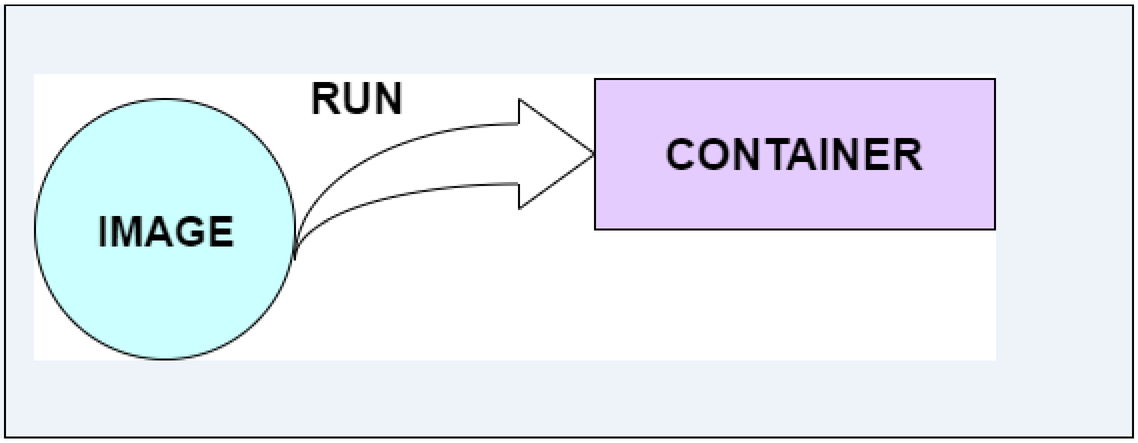
Docker Images

Docker images are the **read-only binary templates** used to create Docker Containers. It uses a private container registry to share container images within the enterprise and also uses public container registry to share container images within the whole world. Metadata is also used by docket images to describe the container's abilities.

Docker Containers

Containers are the structural units of Docker, which is used to hold the entire package that is needed to run the application. The advantage of containers is that it requires very less resources.

In other words, we can say that the image is a template, and the container is a copy of that template.



Docker Networking

Using Docker Networking, an isolated package can be communicated. Docker contains the following network drivers -

- **Bridge -** Bridge is a default network driver for the container. It is used when multiple docker communicates with the same docker host.
- **Host -** It is used when we don't need for network isolation between the container and the host.
- **None -** It disables all the networking.
- **Overlay -** Overlay offers Swarm services to communicate with each other. It enables containers to run on the different docker host.
- **Macvlan -** Macvlan is used when we want to assign MAC addresses to the containers.

Docker Storage

Docker Storage is used to store data on the container. Docker offers the following options for the Storage -

- **Data Volume -** Data Volume provides the ability to create persistence storage. It also allows us to name volumes, list volumes, and containers associates with the volumes.
- **Directory Mounts -** It is one of the best options for docker storage. It mounts a host's directory into a container.
- **Storage Plugins -** It provides an ability to connect to external storage platforms.

Docker Container and Image

Docker container is a running instance of an image. You can use Command Line Interface (CLI) commands to run, start, stop, move, or delete a container. You can also provide configuration for the network and environment variables. Docker container is an isolated and secure application platform, but it can share and access to resources running in a different host or container.

An image is a read-only template with instructions for creating a Docker container. A docker image is described in text file called a **Dockerfile**, which has a simple, well-defined syntax. An image does not have states and never changes. Docker Engine provides the core Docker technology that enables images and containers.

You can understand container and image with the help of the following command

```
$ docker run hello-world
```

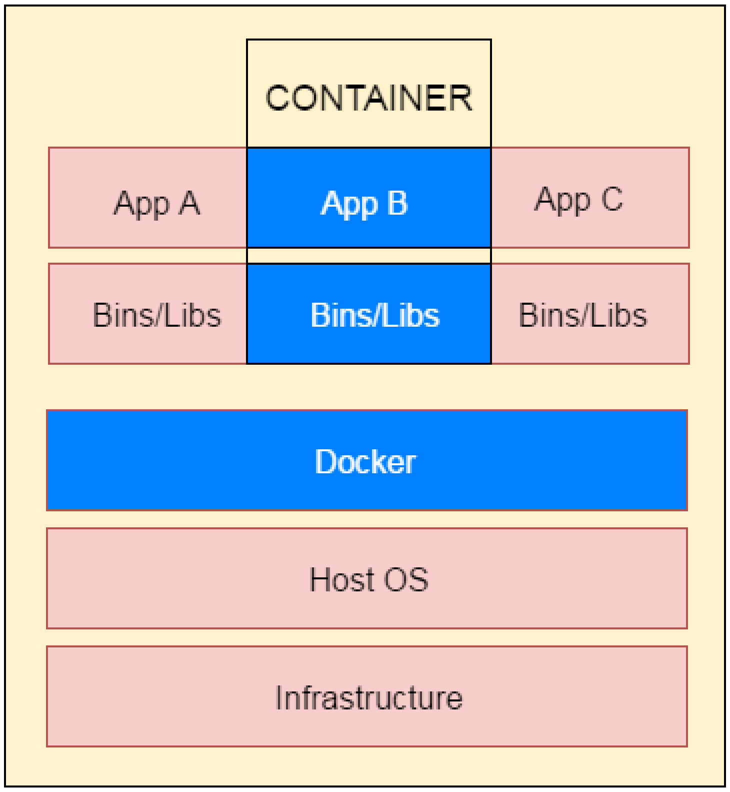
The above command **docker run hello-world** has three parts.

1) **docker:** It is docker engine and used to run docker program. It tells to the operating system that you are running docker program.

2) **run:** This subcommand is used to create and run a docker container.

3) **hello-world:** It is a name of an image. You need to specify the name of an image which is to load into the container.

Docker Container



Docker Dockerfile

A Dockerfile is a text document that contains commands that are used to assemble an image. We can use any command that call on the command line. Docker builds images automatically by reading the instructions from the Dockerfile.

The docker build command is used to build an image from the Dockerfile. You can use the -f flag with docker build to point to a Dockerfile anywhere in your file system.

```
$ docker build -f /path/to/a/Dockerfile .
```

Dockerfile Instructions

The instructions are not case-sensitive but you must follow conventions which recommend to use uppercase.

Docker runs instructions of Dockerfile in top to bottom order. The first instruction must be **FROM** in order to specify the Base Image.

A statement begin with # treated as a comment. You can use RUN, CMD, FROM, EXPOSE, ENV etc instructions in your Dockerfile.

Here, we are listing some commonly used instructions.

FROM

This instruction is used to set the Base Image for the subsequent instructions. A valid Dockerfile must have FROM as its first instruction.

Ex.

```
FROM ubuntu
```

LABEL

We can add labels to an image to organize images of our project. We need to use LABEL instruction to set label for the image.

Ex.

```
LABEL vendorl = "JavaTpoint"
```

RUN

This instruction is used to execute any command of the current image.

```
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

CMD

This is used to execute application by the image. We should use CMD always in the following form

```
CMD ["executable", "param1", "param2"?]
```

This is preferred way to use CMD. There can be only one CMD in a Dockerfile. If we use more than one CMD, only last one will execute.

COPY

This instruction is used to copy new files or directories from source to the filesystem of the container at the destination.

Ex.

1. COPY abc/ /xyz

Rules

- The source path must be inside the context of the build. We cannot COPY ../something /something because the first step of a docker build is to send the context directory (and subdirectories) to the docker daemon.
- If source is a directory, the entire contents of the directory are copied including filesystem metadata.

WORKDIR

The WORKDIR is used to set the working directory for any RUN, CMD and COPY instruction that follows it in the Dockerfile. If work directory does not exist, it will be created by default.

We can use WORKDIR multiple times in a Dockerfile.

Ex.

Docker Compose

It is a tool which is used to create and start Docker application by using a single command. We can use it to file to configure our application's services.

It is a great tool for development, testing, and staging environments.

It provides the following commands for managing the whole lifecycle of our application.

- Start, stop and rebuild services
- View the status of running services
- Stream the log output of running services

- Run a one-off command on a service

Docker Storage Driver

Docker provides us pluggable storage driver architecture. It gives us the flexibility to "plug in" the storage driver in our Docker. It is completely bases on the Linux filesystem.

To implement, we must set driver at the docker daemon start time. The Docker daemon can only run one storage driver and all containers created by that daemon instance use the same storage driver.

The following table contains the Docker storage driver.

Technology	Storage driver name
OverlayFS	overlay or overlay2
AUFS	aufs
Btrfs	btrfs
Device Mapper	devicemapper
VFS	vfs
ZFS	zfs

The Backing Filesystem is **extfs**. The **extfs** means that the **overlay** storage driver is operating on the top of the filesystem.

The backing filesystem refers to the filesystem that was used to create the Docker host's local storage area under **/var/lib/docker** directory.

The following table contains storage drivers that must match the host?s backing filesystem.

Storage driver	Commonly used on	Disabled on
overlay	ext4xfs	btrfsaufsoverlayzfseCryptfs
overlay2	ext4xfs	btrfsaufsoverlayzfseCryptfs
aufs	ext4xfs	btrfsaufseCryptfs
btrfs	btrfsonly	N/A
devicemapper	Direct-lvm	N/A
vfs	debugging only	N/A
zfs	zfsonly	N/A