# Docker Tutorial



# Introduction

## What Is Docker?

According to docker :

> docker allows you to package your applications with all of its dependencies into a standardized unit for  software development.

In a detailed manner , Docker is a new open source Container-Based technology that automate the deployment of applications inside isolated Linux  container including required runtime  including required dependencies of the application. Simply , it allows you to pack your applications with the required dependencies into a container and allow rapid deployment.

## Why Docker?

- Docker Containers Are one of the hottest technologies for applications development nowadays.
- Docker allows you to run many containers simultaneously on any host and can run anywhere whenever it is installed.
- Docker reduces run and deployment time.
- docker provides a safe way to run applications securely isolated in a container and packaged with all dependencies and libraries.
- the dockerized applications can run on any virtual platform  or bare metal without any modifications.

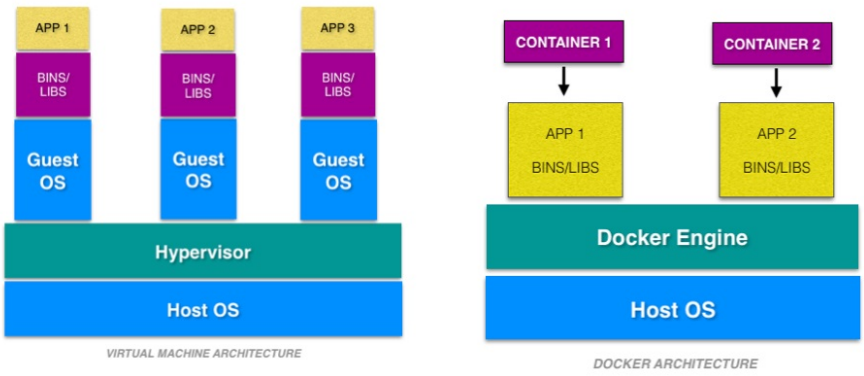
## Docker Benefits for Developers:

- Faster applications delivery.
- Applications portability .
- Can be used as versions Control.
- Sharing containers among team members using repositories.

## Docker Benefits for DevOps:

- Easier deploying and scaling.
- Efficient management of applications life cycle.
- Light weight footprint: docker images are very small.
- Simplified maintenance: by reducing efforts and risk of problems with applications dependencies.

## Docker Vs Virtual Machines:




Virtual machines are more resource-intensive than Docker containers as the virtual machines need to load the entire OS to start. The lightweight architecture of Docker containers is less resource-intensive than virtual machines.

In the case of virtual machines, resources like CPU, memory, and I/O may not be allocated permanently to containers — unlike in the case of containers, where the resource usage with the load or traffic.

Scaling up and duplicating containers in simple and easy as compared to virtual machines because there is no need to install an operating system in them.

Apart from the major differences, some other differences are summarized below:

| Criteria | Docker | Virtual Machines |
|---|---|---|
| Boot-Time | Boots in a few seconds. | It takes a few minutes for VMs to boot. |
| Runs on | Dockers make use of the existing kernel on Linux machines | VMs make use of the hypervisor. |
| Memory Efficiency | No space is needed to visualize, hence less memory. | Requires entire OS to be loaded before starting the surface, so less efficient. |
| Isolation | Prone to adversities as no provisions for isolation systems. | Interference possibility is minimum because of the efficient isolation mechanism. |
| Deployment | Deploying is easy as only a single image, containerized can be used across all platforms. | Deployment is comparatively lengthy as separate instances are responsible for execution. |
| Usage | Docker has a complex usage mechanism consisting of both third party and docker managed tools. | Tools are easy to use and simpler to work with. |

For more information visit this link: Get Started With Docker

## Docker Components

Docker has set of components that constitutes its building block including:

| Component | Description |
| --- | --- |
| Docker Daemon | Runs on a host machine and responsible for docker operations without direct interaction from the user. |
| Docker Client | Accept commands from users and communicates with docker daemon through CLI or REST API. |
| Docker Image | Is a template or blueprint that contains all the data and metadata needed to run the containers that are launched from a certain image. |
| Docker Container | A box that contains all large and tiny details which are needed for a dockerized application to run. Each container is created from a certain docker image and is an isolated and secure application platform. |
| Docker Registry | A public or private hosted collection of tagged images that together create the file system for a container. |
| Docker Hub | A Software as a Service (SaaS) platform for sharing and managing docker images. |
| Docker Dockerfile | A text file containing a specific syntax for building docker Images. |
| Docker Swarm | A clustering and scheduling tool for docker containers. |

For more information visit this link: Docker Components

# Docker Installation

## Docker Editions

Docker is coming in two editions including:

- Docker Community Edition (CE):

  Docker-CE is for the developers and DevOps community. Each month docker releases an Edge version with latest features and in addition to quarterly releases with stable versions. it is available for the following operating systems:

  1. Ubuntu
  2. Debian
  3. CentOS
  4. Windows 10
  5. MacOS
  6. Microsoft Azure
  7. AWS
- Docker Enterprise Edition (EE):

  Docker-EE is designed for enterprise development who build, ship and run business-critical applications in production environment.

For more information Visit this link: Docker Editions

## Docker for Linux

Docker engine is built on the top of Linux kernel, which means that docker runs directly in any Linux distribution. there are several ways to install Docker engine om Linux Based OS Including :

1. Installation Scripts provided by Docker community.
2. Docker Package included in the official Linux distribution (but might not be the latest version).
3. Packages supplied by docker for any Linux distributions.
4. Binary download from Docker repository.

To install Docker on a Linux distribution, please refer to the official docs.

## Docker for Microsoft Windows

Docker desktop for windows requires Microsoft Windows 10  professional or enterprise 64-bit, to install:

- Download the Setup file from here DockerDesktop4Windows
- Run The setup file **Docker Desktop Installer** and wait until it downloads the required packages.
- You will encounter 3 options during Docker installation: Tick just the first two options.
    - The third option is to use Windows as the host OS of Docker, this **IS NOT** recommended, since most images aren't supported on Windows based Docker
- Wait until the installation is completed and then restart your PC

  **Start enjoying Docker Services**

For more information visit this link: Docker for Windows

# Installation Validation

## Linux

1. Open a terminal
2. Go to Powershell
3. Run

   ```
   docker -v
   ```

   Expected Output:

   Docker version 19.03.5, build 633a0ea

## Windows

1. Start Docker Desktop App, wait until it's finished initializing
2. Go to Start Menu > Powerhell
3. Run

```
docker -v
```

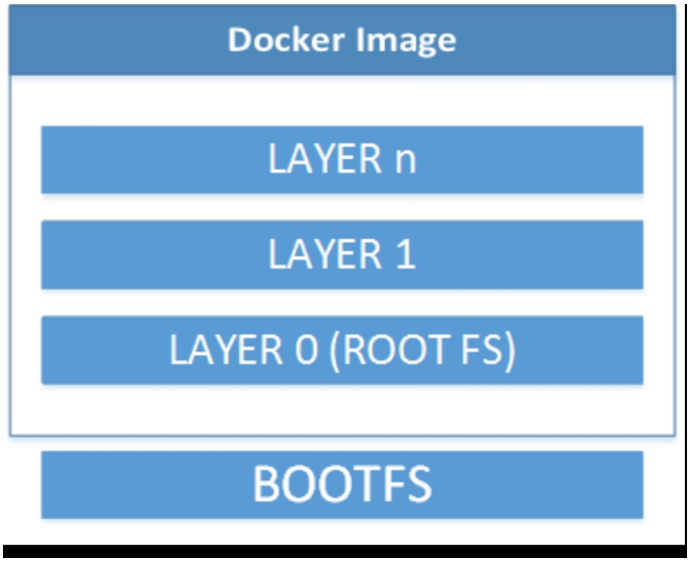
Expected Output:

Docker version 19.03.5 Build 633a0ea

# Docker Image

## What is docker image ?

An image is a template or blueprint that contains all the data and metadata needed to run the containers which are launched from a certain image. You can think of it as Class in OOP, the image is comprised of a layers, each layer is a set of filesystem changes and has a unique ID upon its creation. Further, Docker image is immutable (Read-only template), the following figure descries Docker Image architecture:




## Basic Docker image Commands

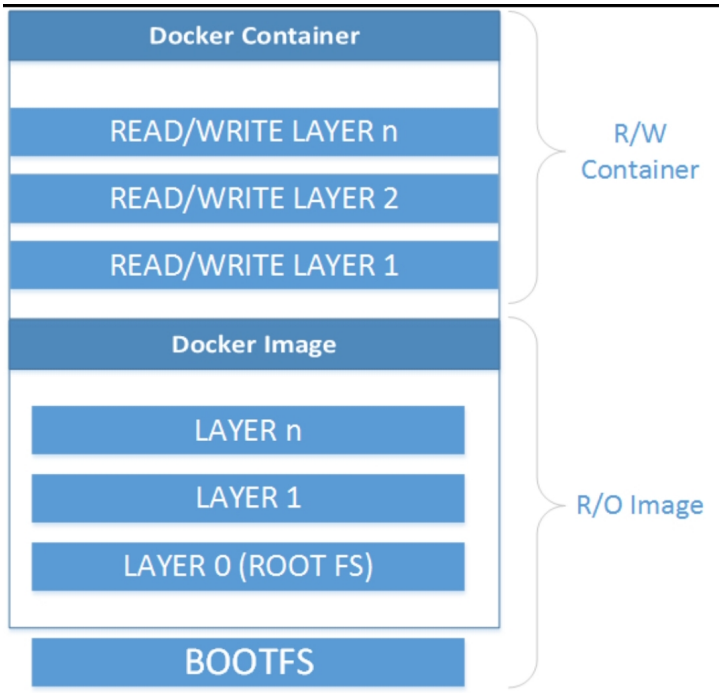
| Command | Description |
| --- | --- |
| docker image ls | List all available images. |
| docker image build | Build an image from a Dockerfile. |
| docker container commit | Create a new image from a container's changes, pausing it temporarily if it is running. |
| docker image rm | Remove one or more image. |
| docker image history | Show the history if an image. |
| docker image tag | Tags an image into a repository. |
| docker image import | Import the contents from a tarball to create file system image. |
| docker image export | Export a container's file system as a tar archive. |
| docker image search | Search the docker hub for specific images. |

For more information visit this link: Docker Image

# Docker Containers

## What is Docker Container?

Docker containers are created from docker images , you can think of it as a class instance in OOP. It differs from Docker image in creating a writable layer on the top of docker image that enables the user to commit changes on the on containerized image and create a new image out of it. Further it is also possible to have several running  containers from the same image, each of the running containers is an isolated and secure application environment, the following figure descries Docker Image architecture:




## Basic Docker container commands

| Command | Description |
| --- | --- |
| docker container create | Create a new command but do not start it |
| docker container run | Create and start a container in on shoot. |
| docker container rm | Remove one or more containers |
| docker container start | Start one or more stopped containers |
| docker container stop | Stop a running container |
| docker container restart | Restart a running container |
| docker container kill | Kill a running container. |
| docker container attach | Attach/connect to a running container. |
| Docker container exec | Perform commands in a running container. |
| docker container ps | Show all running containers. |
| docker container logs | Display the logs of certain container. |
| docker container inspect | Return a low-level information about certain container. |
| docker container top | Display the running processes of a container. |
| docker container port | List port mappings or a specific mapping for the container. |
| docker container stats | Display a live stream of a container's resource usage statistics. |
| docker container diff | Inspect changes on a container's filesystem. |

For more information visit this link: Docker Containers

# Docker Network

## What is Docker Network?

Docker network is a connection and communication way between docker containers and the outside world via host machine. Alternatively, it is a communication passage through which all isolated containers communicate with each other in various situations to perform the required actions.

## Docker Network Types

1. **Bridge Network:**

   When you start Docker, a default bridge network is created automatically. A newly-started containers will connect automatically to it. You can also create user-defined custom bridge networks. User-defined bridge networks are superior to the default bridge network.

2. **Host Network:**

   Host network remove network isolation between the container and the Docker host, and use the host's networking directly. If you run a container which binds to port 80 and you use host networking, the container's application is available on port 80 on the host's IP address. Means you will not be able to run multiple web containers on the same host, on the same port as the port is now common to all containers in the host network.

3. **Overlay network :**

   Creates an internal private network that spans across all the nodes participating in the swarm cluster. So, Overlay networks facilitate communication between a docker swarm service and a standalone container, or between two standalone containers on different Docker Daemons.

4. **Macvlan network :**

   Some applications, especially legacy applications or applications which monitor network traffic, expect to be directly connected to the physical network. In this type of situation, you can use the Macvlan network driver to assign a MAC address to each container's virtual network interface, making it appear to be a physical network interface directly connected to the physical network.

5. **None Network :**

   In this kind of network, containers are not attached to any network and do not have any access to the external network or other containers. So, this network is used when you want to completely disable the networking stack on a container.

## Basic Docker network commands

| Command | Description |
| --- | --- |
| docker network connect | Connect a container to a network. |
| docker network create | Create a new network. |
| docker network disconnect | Disconnect a container from a network. |
| docker network inspect | Display detailed information on one or more networks. |
| docker network ls | List all networks. |
| docker network prune | Remove unused networks. |
| docker network rm | Remove one or more network. |

For more information visit this link: Docker Network

# Docker Volumes

## What is Docker Volume?

Docker volumes are directories or filesystem that exist outside the containers with main purpose of persisting the container data. the main role of the volumes is to store the changes that happen inside a container due to various operations on your local to be readily available in case the container has been removed.

## Basic Docker volume commands

| Command | Description |
| --- | --- |
| docker volume create | Create a new volume |
| docker volume inspect | Display detailed information on one or more volumes. |
| docker volume ls | List all volumes. |
| docker volume prune | Remove unused volumes. |
| docker volume rm | Remove on or more volumes. |

For more information visit this link: Docker volume

# Docker HUB

## What is Docker HUB?

Docker HUB is a Software as a Service (SaaS) platform provided by Docker Company that holds public and private Docker images to be downloaded Docker user for their application development.

## Docker HUB Features:

1. Central repository for public/private images.
2. Automatically creates new images on changes in source code repository.
3. Work-groups creation to mange access to image repositories.
4. Lifecycle workflow automation.

For more information visit this link: Docker HUB Docs

## Simple example:

Searching for MySQL image on from public repository (Docker Hub), You can either directly search the docker HUB website to search for the desired image or doing it from the terminal or windows powershell as shown in the example which describe the looking up for python image:

```
docker search python
```

once you found the desired image, you have to download it using the following code:

```
docker pull python
```

# Docker Private Registry

## What is Docker Private Registry?

Docker private Registry is storage system used to store the private images on a local machine. The registry server is container provided Docker company. It is highly recommended to use registry version 2 which has a greater advancement in security, reliability and efficiency over version 1.

## Docker Registry Advantages:

1. Securely deployed in your local environment.
2. faster deployment due to the reduced latency of pulling and pushing images.

To create a local registry, apply the following command

```
docker run -d -p 5000:5000 --restart=always --name registry -v
data:/var/lib/registry registry :2
```

in the previous command, the registry data is persisted as a docker volume on the container filesystem. and thus if you want to change the location of the registry volume, just change the director after the parameter `-v` .

An alternative solution for private registry is to create an account on the `docker hub` from the following link which provides a free-to-use, hosted Registry, plus additional features (organization accounts, automated builds, and more).

For more information visit this link: Docker registry Docs

# DockerFile

## What is Dockerfile:

Simply, Dockerfile is the image creation recipe. More technically, Dockerfile is a text based file (script) that includes all commands you would normally execute manually in order to build a docker image, these commands are called Dockerfile instructions. Dockerfiles are always aligned on any base image and enables docker daemons to build images automatically by reading the instructions from Dockerfile.

## Dockerfile Instructions:

Dockerfiles instruction are the blueprint for creating a docker image, there are 19 command to perform in a DockerFile as shown in the table bellow but it is not necessarily  to use them in every Dockerfile, it depends on your need:

| Instruction | Description |
| --- | --- |
| FROM | Define the base image to start building image on top of it. |
| MAINTAINER | Declare the author of the generated image. |
| ENV | Define one or more environment variables in the image. |
| RUN | Execute any commands in a new layer the top of the current image and commit the results. |
| CMD | Provide defaults for an executing containers. |
| EXPOSE | Used to associate a specified port to enable networking between the running process inside the container and the host |
| ADD | copy new files, directories, or remote file to the container. Not highly recommended instruction because it invalidates the cache. |
| LABEL | add metadata to an image |
| COPY | Copy New files or directories to container. |
| ENTRYPOINT | Configure a new that will run as an executable. |
| VOLUME | Used to enable access from your container to a directory on the host machine. |
| USER | Set the username for the following commands : RUN, CMD and ENTRYPOINT. |
| WORKDIR | Set the working directory |
| ARG | Defines a variable that users can pass at build-time to the builder with the `docker build` command |
| ONBUILD | Adds an instruction to be executed later, when the image is used as the base for another build |
| STOPSIGNAL | Sets the system call signal that will be sent to the container to exit |
| HEALTHCHECK | Tells Docker how to test a container to check that it is still working |
| SHELL | Allows the default shell used for the shell form of commands to be overridden. |

The previous table may somehow vague or ambiguous to draw a complete picture about the Dockerfiles. For this Reason, there will be a complete example on how to create a Dockerfile.

For more information visit this link: Dockerfile Docs

# Conclusion

Up to this point,  a large proportion of the theoretical part of Docker technology has been covered, such brief introduction enables the reader to have a clear idea about docker as well as its related issues  and concepts. In cases there is a need for extra information and further reading, a Docker documentation link is provided at the end of each section that route you directly toward the documentation provided by docker for the same topic. An extra hands-on example on how to use docker commands is provided [here]

---