***Docker Container Management Service => Build | Ship | Run containers***

Docker is a containerization platform uses software containerization technology.

Docker doesn’t use hypervisor

### cgroups and Linux namespaces are at the heart of containers today, including Docker.

### Need of Docker

1. It is very clear that the environment in which application or a service developed or running is different on the Production environment.

***Development Env.! = Production Env.***

1. In today’s world of rapid development, multiple technology stacks and faster release model, Containers are a must to satisfy the needs of the client.
2. Is it possible to run multiple versions of the same platform on a single host.

Advantages

Time Saving | Cost reducing solution | no need to configure hardware and spend time installing operating system and software

Docker image

Docker image is a read only template, used to create containers.

A Docker image is made up of filesystems layered over each other.

At the base is a **boot filesystem, bootfs**, which resembles the typical Linux/Unix boot filesystem.

Docker next layers a **root filesystem, rootfs,** on top of the boot filesystem. The rootfs can be one or more operating systems (e.g., a Debian or Ubuntu filesystem, the root filesystem stays in read-only mode Docker images are stored in Docker Hub (i.e., public image repository) or your local repository.

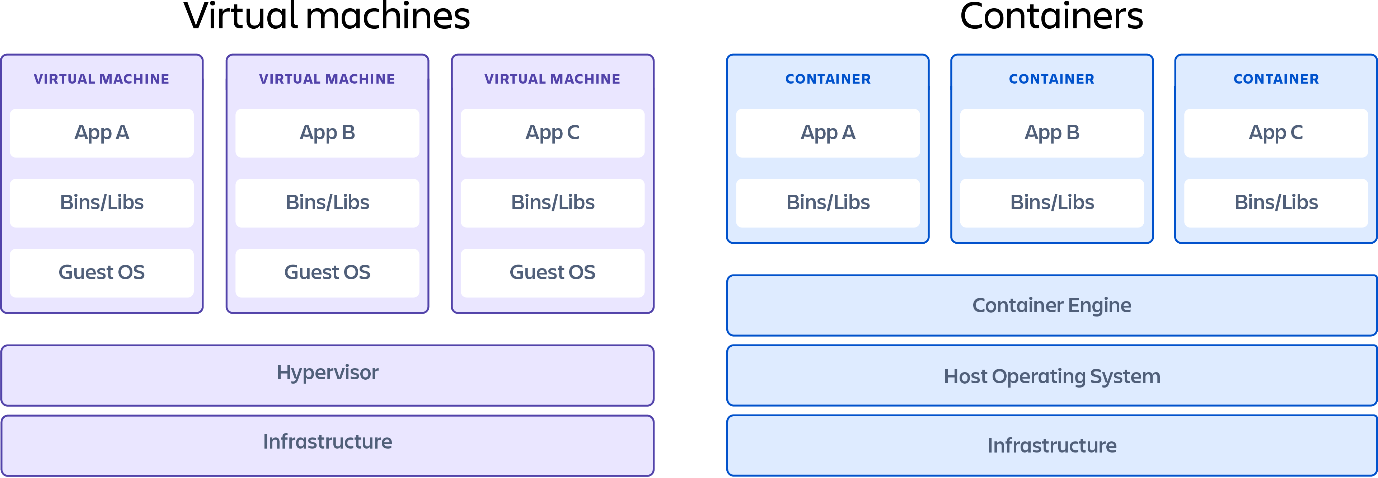
Layer of Images (Mostly Linux base Image, because small in size, Application image on top)

**Advantage of layers** is that we don’t need to install all the layer of similar application when two different versions are downloaded. As the first download will download all the layers and in next time it will only download the changed layers or delta layers (difference among two).

Containers

A container is a loosely isolated environment used to build & run software packages.

These software package include the code and all dependencies to run application quickly and reliably on any computing environment.



Host Operating System

Docker Terminologies

**Docker Client –** Primary Interface to Docker Engine (CLI)

**Docker Engine/ Docker Daemon** – This is a lightweight runtime, that builds and runs Docker containers. | Host Level process | REST based service.

**Docker registry** - Server for Docker daemon to fetch repositories and images | Public or Private

**Docker Hub** – This is a docker registry where the Docker images and repository are stored and also shared with other Docker users.

**Docker Machine** – This is a tool which lets user install Docker Engine on Virtual hosts.

Docker Machine is **a tool for provisioning and managing your Dockerized hosts** (hosts with Docker Engine on them).

**Docker Swarm** - Provides native clustering capabilities and turns a group of Docker engines into a single, virtual Docker Engine.

**Docker Compose** - is a tool for defining and running multi-container Docker applications

**Image** - Read-only layer of file-system that never changes (Blue-print of a running container)

**Container** - Self-contained environment build using one or more images (created from Docker image)

**Dockerfile** - A file containing Docker commands to configure and build image from existing image and other ingredients.

**Repositories** - Set of images on local Docker or Register Server

**Volume**- Provides external persistent storage for containers

To check status of the docker service on Ubuntu

**sudo service docker status**

To start a service To stop a service

**sudo service docker start sudo service docker stop**

# Command

**docker run hello-world:** To check whether docker has installed successfully or not. By running hello-world image.

**docker run <ImageName>:** To pull image if locally not available and then start the container. [pull + run]

**docker ps**: (process state) show running containers

**docker ps -a:** shows all the container irrespective of running and exited process state.

**docker info**: Display system-wide information

**docker version**: Show the Docker version information

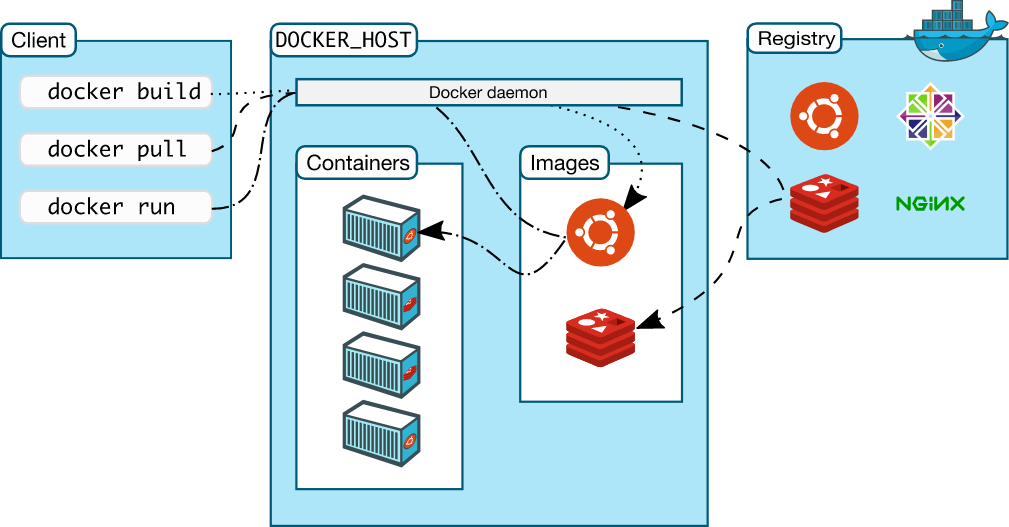
**docker images**: List images stored locally docker image

**docker pull <imageName>**: Fetch the image from Docker hub

**docker rmi < imageName >**: Delete or remove the image from local machine

**docker inspect < imageName >**: View details available in the image

# Docker Architecture



### How to open a terminal into a running container

* Use ***docker ps***to get the name of the existing container
* Use the command ***docker exec -it <container name> /bin/bash*** to get a bash shell in the container
* Generically, use ***docker exec -it <container name> <command>*** to execute whatever command you specify in the container.
* At end type exit to close the container terminal

### Keep Your Container Running in The Background

You have to use two combinations, one after the other: ctrl+p followed by ctrl+q.

Or use docker ps -l –q

# Containers

Name of the image

***docker image pull ubuntu***

Fetch an image named ubuntu from official docker repository/ registry

Start a new container based on image name provided

Name of the container

Command to run within the container

***docker container run –it --name myUbuntu ubuntu /bin/bash***

Start the container in Interactive terminal

Image name pulled before

***docker image pull nginx***

Run the container in detached mode, command prompt will be provided to user.

-p is used for specific port whereas -P is used for random port

***docker container run –d –p 8085:80 --name webserver nginx***

Provide port at which the web server must run. 8085 is HOST port 80 is CONTAINER port. Used to bind the host computer port with the container port.

**docker container ls** = Provide list of containers running currently in the host

**docker container ls –a** = Provide list of containers irrespective of the status of the container.

**docker stop <container Name/ ID>** = Stop the running container

**docker start <container Name/ ID>** = Starts the stopped container

**docker inspect <container Name/ ID>** = Provide details (internal) of container

**docker logs <container Name/ ID>** to see what is running inside the container | used for debugging a container

**docker top <container Name/ ID>** = Provide processes running inside the container

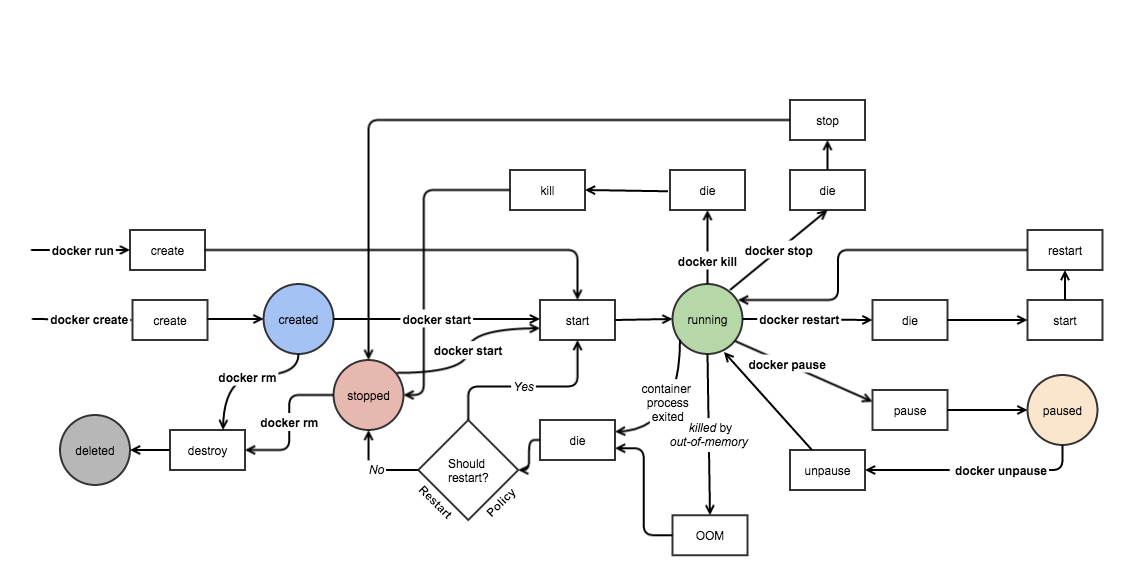
**docker port <container Name/ ID>** = Provide details of the port mapped

**docker rm –f <container Name/ ID>** = Remove/delete/terminate container

-f is used for forcefully removing the container if the container is in running state else not needed if it’s in stopped state.

**docker stats** **<container Name/ ID>** = Display a live stream of container resource usage statistics

# Container Lifecycle



Creating Image

**docker commit** = build an image from existing running container.

**docker build** = create an image from dockerfile by executing the build step given in the file.

**docker import/ load** = create a base image by importing from a tarball (import is mainly used for creating base image) | load preserve meta data whereas import recreate metadata.

Deploying application on Docker host

• Create a Dockerfile that specifies what your app needs to run (i.e., define an image).

• Connect to the Docker Host.

• Using the Dockerfile, create an image in the Host.

• Create a new container using this image.

• Start the container. Your app is now “Dockerized”!

• Optionally, you can take a snapshot of this container. This will create a new image which you can then push into a Registry for later use

# Underlying technology for Docker

• Docker is written in Go and makes use of several kernel features.

• Namespaces – provide the isolated workspaces we call the container.

• When you run a container, Docker creates separate set of namespaces for that container.

• This provides level of isolation, each aspect of a container runs in its own namespace.

**• Some of the namespaces Docker engine uses are as below** –

➢ The pid (process ID) namespace for process isolation.

➢ The net namespace for managing network interfaces.

➢ The ipc (InterProcess Communication) namespace for managing access to IPC resources.

➢ The mnt namespace for Managing mount-points.

➢ The uts namespace for isolating kernel and version identifiers

**• Control Groups (cgroups)**

‒ Allow the Docker to share the available hardware resources to containers.

‒ This ensures Docker containers are good multi-tenant citizens on the host.

**• Union File Systems**

‒ File Systems that operate by creating layers.

‒ This makes them very lightweight and fast.

‒ These serve as the building blocks of containers

# Docker images and Containers

• When a new container is created, a thin writable layer is added on top of underlying stack.

• This layer is called container layer.

• All changes made to the running container writing new files, modifying existing files, and deleting files are written to this thin writable container layer

Volume

*Folder in physical host file system is mounted into the virtual file system of Docker*.

**Anonymous volume** -v /var/lib/mysql/sata

**Named Volume** -v name: /var/lib/mysql/sata

**Host volume** ‒ sudo docker run --name docker-nginx -p 80:80 -d -v ~/docker-nginx/html:/usr/share/nginx/html nginx

• -v specifies that we're linking a volume

• The part to the left of the : is the location of our file/directory on our virtual machine (~/docker-nginx/html)

• The part to the right of the : is the location that we are linking to in our container (/usr/share/nginx/html)

**sudo docker login:** To connect with docker hub (remote repository)

**sudo docker pull <ImageName>:** to download an image from docker hub

**sudo docker push <ImageName>:** to push an image to docker hub

DockerFile

Is a blue print for images.

Every instruction in docker file act as a layer.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence

Starting by basing it on other image

We can have multiple RUN commands but single CMD command.

Entry point Linux command

Copy from host

Inside container

Optional defining env var

Building image from DockerFile

**docker build -t <ImageName> .** = . represent current directory/ folder

Docker Compose

Used to manage multiple container all at once in a structured way using yaml file.

Graphical user interface, application

Description automatically generated

Image name

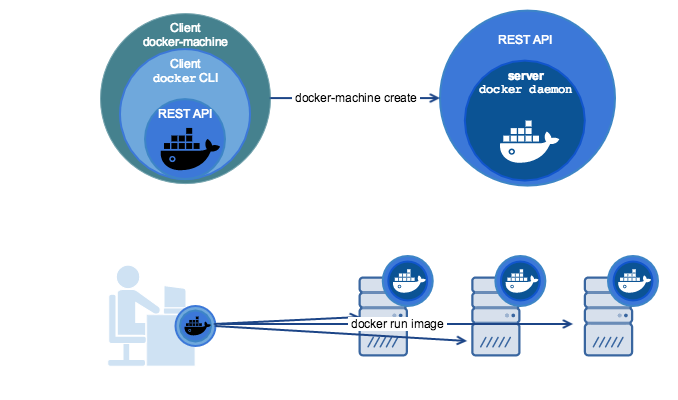
First Container name

Starting container using compose file

**docker-compose -f <x.yaml> up**: to start all the container inside the x.yaml file

**docker-compose -f <x.yaml> down**: to stop all the container inside the x.yaml file

Docker Machine



NOTE

1 process per container