Docker

- open source, centralised platform designed to create, deploy and run applications.
- It allows applications to use the same Linux kernel as the system on host computer, rather than creating a whole virtual OS.
- Containers ensure that our application works in any environment like development, test or production.
- Previously - code works on my system but not on user's system.
- Docker resolves this issue

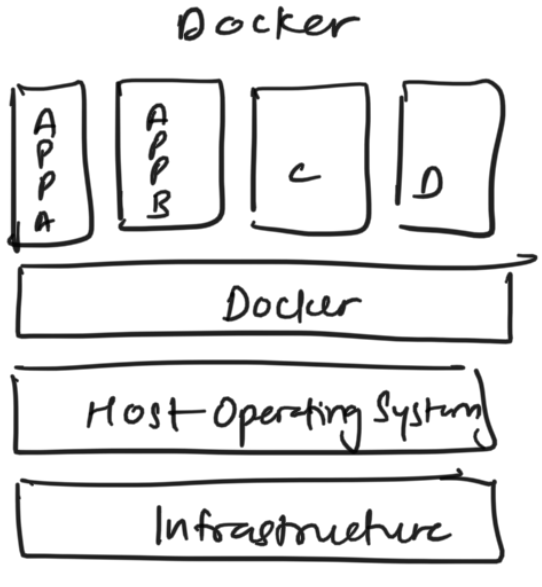
Docker containers

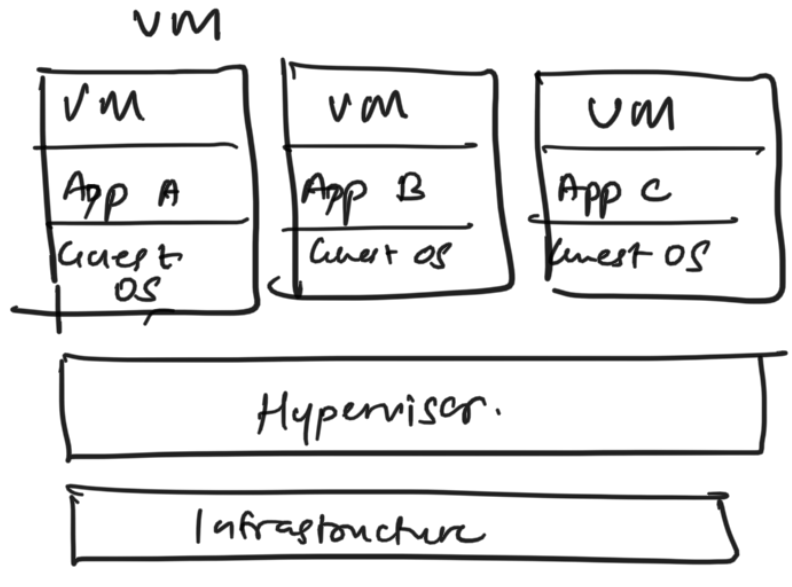
- lightweight alternatives of the virtual machine.
- allows developers to package up the application along with its libraries, dependencies and ship it as a single package
- Adv - you dont need to allocate RAM and disk space for apps. It automatically generates storage and space according to application requirement.
- multiple containers can run on same machine, share OS kernel each running as isolated process in user space

Virtual Machine

- allows to use other OS simultaneously on our machine.
- the virtualized OS can function similar to our real OS.
- includes a full copy of OS, taking tens of GBs, slowdown boot

Containers	Virtual Machine.
Integration - faster, cheaper	Integration - slow, costly.
no wastage of memory	wastage of memory.
uses the same kernel, but diff. distribution	It uses multiple independent OS.



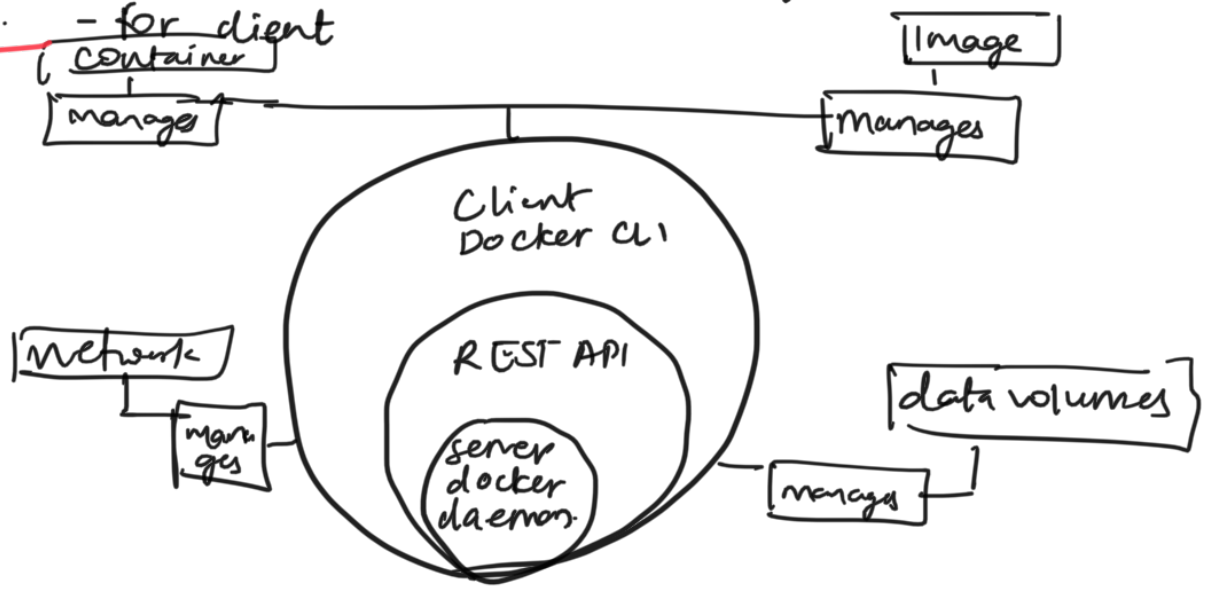


Why Docker?

- easy install & run apps. dont worry about setup, dependencies.

- runs in seconds
- less memory use, less resources => less disk space
- does not require full OS to run apps.
- dont need a full OS

Docker Engine
- A client server app with following components.
- A server - long running program called daemon process.
- REST API - specify interfaces that prog. can use to talk to daemon & instruct it.
- CLI - for client



Docker Image
- a template for creating an environment of your choice
- a snapshot, can create multiple snapshot/versions and choose from them
- read only, binary templates.

Docker container.
- structural units of docker, which is used to hold entire pkg that is needed to run the application..
- Image - template Container - a copy of that template.
- running instance of an image.

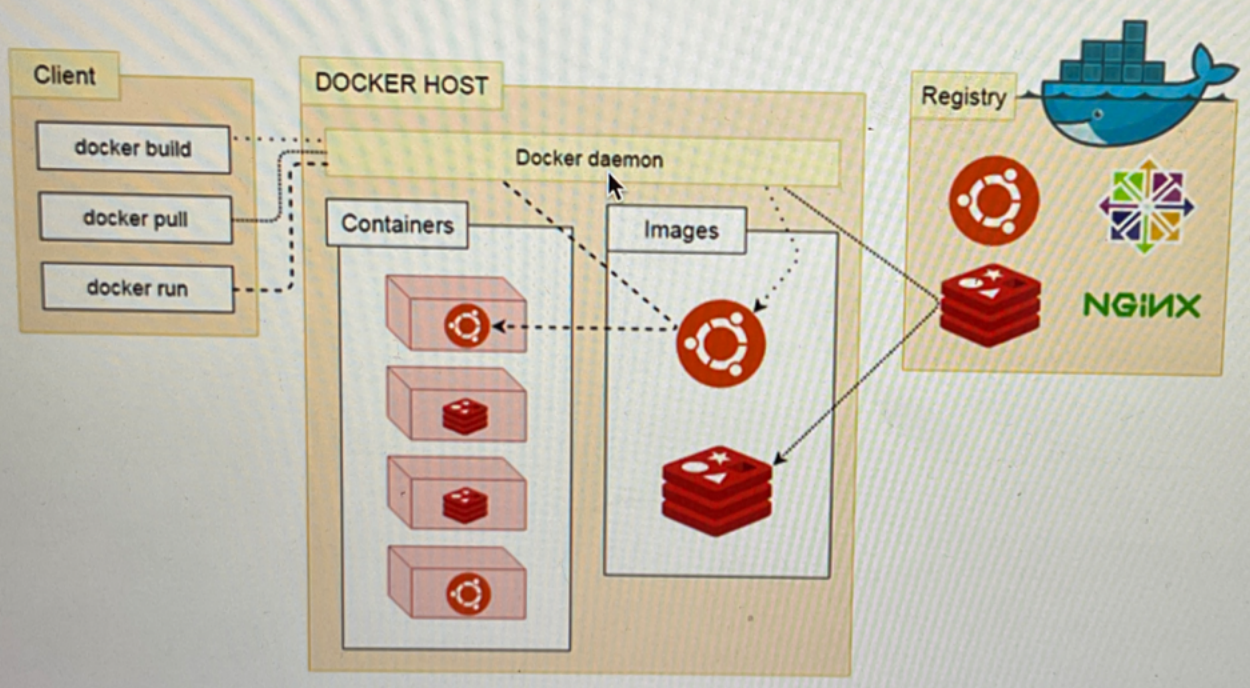
Docker daemon
- runs on host OS
- responsible for running containers to manage docker services
- communicates with other daemons.
- offers various docker objects such as images, containers, networking, storage.

Docker Architecture -

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 commands and REST API to communicate with docker daemon (server)

Docker Host
- provides env to run and execute apps.

Docker Registry
- manages & stores docker images.
Public Registry - Dockerhub
Private Registry - used to share images with the enterprise

Dockerfile - docker image is described in a text file called dockerfile

docker pull nginx → pulls an image.
docker images → lists all images.
docker run nginx:latest → run an image, creates container
(image name, tag)
docker container ls → lists all containers running
or docker ps
docker run -d nginx:latest → run in detached mode.
docker stop containerID → stops container with that id.
docker run -d -p 8080:80 nginx:latest
whenever we type 8080 on localhost we want to map it to port 80 on container.

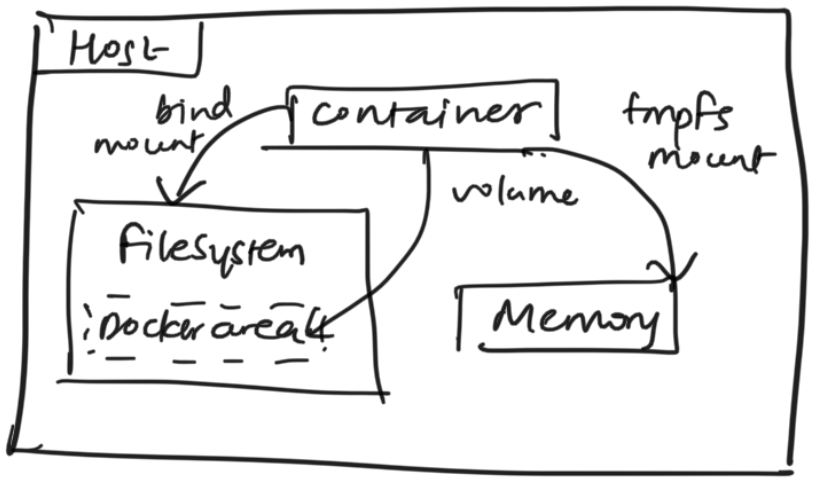
we can also map multiple ports.
docker run -d -p 8080:80 -p 3000:80 nginx:latest
(expose multiple ports.)

docker stop containerID/name
docker ps -a // show all containers
docker rm containerID/name → delete a container.
docker ps -aq → all container, only numeric id
docker rm $(docker ps -aq) → remove all containers
(it wont remove running containers)
docker rm -f $(docker ps -aq) → force remove
(will remove running ones too)
docker run --name website -d -p 3000:80 nginx:latest
(assign a name to container)

export FORMAT = " "
docker ps --format=FORMAT // formats text o/p of ps.

Volumes

- allows to share data - files, folders
- allows share data b/w host & container
- and also b/w containers



volume source dest

docker run --name some-nginx -v some/content:/usr/share/nginx/html:ro -d nginx

dest — read only

Inside dest we can share our own files

create a folder website on your desktop
Create a index.html inside it

cd website
docker run --name website -v $(pwd):/usr/share/nginx/html -d -p 8080:80 nginx:latest

go inside container
docker exec -it website bash.
cd /usr/share/nginx/html - you will see index.html.
touch about.html.
If you will see it on your desktop too.
Hence a volume is created.

Dockerfile
- Build your own image
- Has everything your app needs to run
- OS, Software, App code

Dockerfile.
FROM nginx:latest → base image that you want to use.
ADD . /usr/share/nginx/html → This is where static content should live for nginx
src (everything in curr dir) dest

docker build --tag. website:latest .
-t website:latest .
name version loc^n to dockerfile

docker image ls
docker run --name website -p 8080:80 -d website:latest

```
FROM node:latest        → use node image
WORKDIR /app            → set up workdir, if not exist make on.
                          all commands will run in app.
ADD . .                 → added from src to app dir
RUN npm install
CMD node index.js
```

Dockerignore → like gitignore

.dockerignore.

```
node_modules
Dockerfile
.git
*.js            all .js files
/folder/**      all folder
```

Caching and layers.

Every command in Dockerfile will create layers

If we make only small changes we can utilize caching and skip certain steps (eg. npm install)

ADD package.json . → add in dockerfile before npm install.

Alpine

→ Optimize size of image created

- Simple, smaller, secure

docker pull node:lts-alpine - alpine version of node

nginx:alpine — nginx

* docker image rm imageid

In Dockerfile

```
FROM nginx:alpine
node:alpine
```

Tags, Versions, Tagging.

→ Allows you to control image version

→ Avoid breaking changes

(change from node 8 → node 12 say)

→ Safe

```
FROM node:10.16.1-alpine
          specific version
```

Build & run

Tagging override

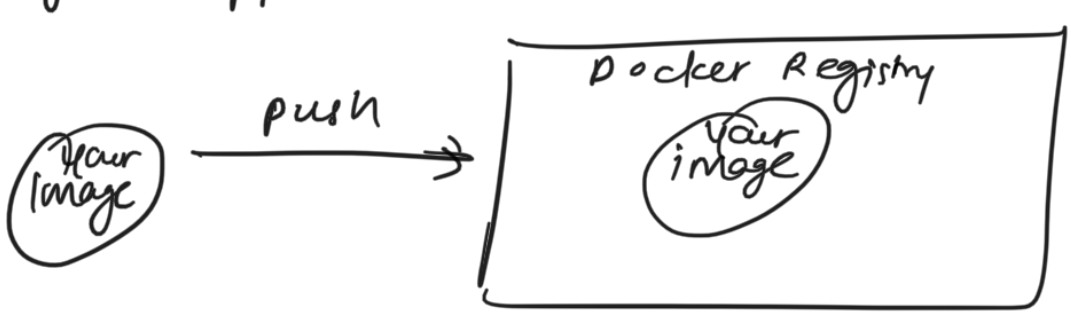
Whenever you make change to image & use same tag it will override the image, mark it as <none> in docker images ls.

docker tag amigoscode-website:latest amigo-website1
website 2

2 versions - 1, 2
1 amigoscode-website:latest

V1 and V2 will be different

Docker Registries
- Highly scalable server side application that stores and lets you distribute Docker images.
- used in your CICD pipelines.
- Run your applications.



- Private/Public Registries
- Dockerhub - Public
- quay.io
- Amazon EC2 container registry

Go to Dockerhub
create Repository

docker push harshali1504l/myapp:tagname

docker login

* Tag your image properly **
docker tag nodeapp:v1 harshali1504l/myapp1:v1
username repo

Pulling Images

docker rmi harshali1504l/myapp1:v1 → removes ah image

docker pull harshali/myapp1. → pulls latest image by defaur as no tag is mentioned.

& then run.

Docker Inspect

docker Inspect containerID/name → Inspect the container.

Docker logs

docker logs containerID/name → console.log().

You can see all requests GET, PUT etc.

docker logs -f containerID/name → Follows the app.

if u click, a GET Req
we get GET in logs.

Docker exec

To get into the container to debug, etc.

docker exec -it containerID

interactive — Allocates a sudo tty

docker exec -it containerID /bin/sh

— you can find this cmd in inspect

You will go to your myapp1.

If cd..

→ so to linux file directory structure