★] DOCKER (/var | /lib | /docker)

→ docker is containerization platform for developing, packaging, shipping and Running applications

→ It provides the ability to run an application in an isolated environment called container

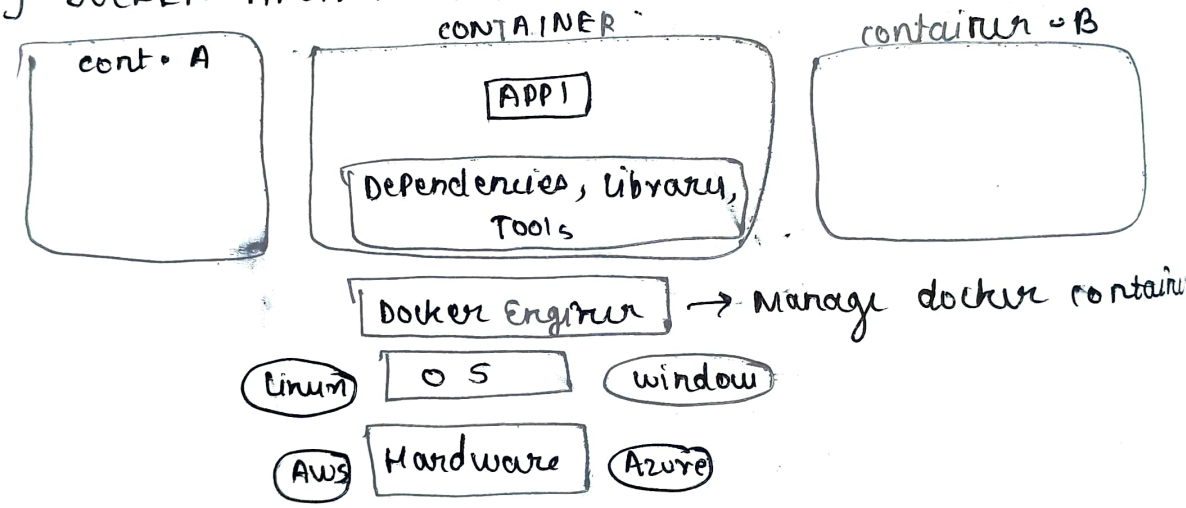
→ makes development & diployment efficient.

★] Container

→ container is a collection of application and all its necessary dependencies & configuration

→ container can easily be shared

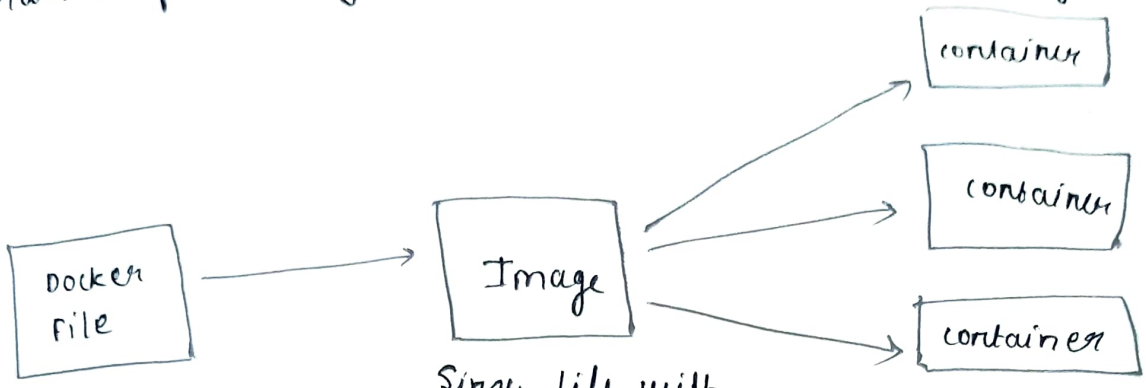
★] DOCKER ARCHITECTURE



containeiner A & B are isolated from each other, Each container is isolated from Each other

DOCKER CONTAINER	VIRTUAL MACHINES
1] Low impact on os, disk	1] High impact on os, disk.
2] Sharing is easy	2] Sharing is hard
3] Encapsulates APP instead of whole machine	3] Encapsulates whole machine

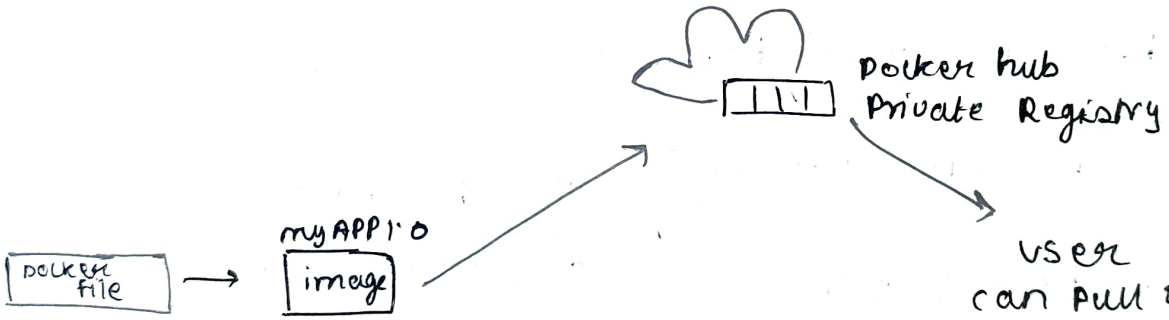
*) Main components of docker



It is a simple text file that contains the instructions to build an image

Singe file with all the dep, lib to Run the Program

Docker Registry → It is central Repository for storing and distributing Docker Images.



*) Docker hub → A Place where many docker images could be found.

Registry → comman Place, docker hub

Repository → different version of software

docker -v → check docker version

System ctl start docker.service → start docker

Registry → for organization (Private hub).

Hub → for Public use.

★) DOCKER FILE

To select base use `FROM`

eg `FROM node`

└ download from docker hub autom...

Layered architecture

`FROM node:20` → use nodejs 20 version.

`WORKDIR /myapp` → create a working directory in contain...

`COPY . .` → copy all files in working directory

`RUN npm install` → how to Run npm in the container

~~`RUN npm start`~~ `EXPOSE 3000` → will work on Port 300...

`CMD ["npm", "start"]` → after Running image this CMD command should Run.

★) DOCKER IMAGE

`docker build .` → Build docker image [.] - says in the same location

`docker build . -f <filename>`

`docker image ls` → list all the images we have

→ docker images →

`docker run <image_id>` → Run your docker image in... the container

`docker ps` → Shows Running container

`docker stop <container_name>` → stop the docker contain...

`docker run -P 3000:3000 <img-id>` → giving access outside the container

Port Binding

*) Running docker Container in detached Mode.

`docker run -d -P 3000:3000 <img-id>` = -d means detached mode

↳ helps to free the terminal, if you will use Run then it'll freeze the terminal & you cannot use more commands

*) Running Multiple docker containers using Single Image

→ If you try to Run two containers on Port 3000 then it will show error, to solve this Problem what you can do is.

`docker run -d -P 3001:3000 <img-id>` → this will Run 3 containers same on the Port 3001, 3002 & 3003.

`-d -P 3002:3000 -11-`

`-d -P 3003:3000 -11-`

`docker rm <cont-name>` → Remove the container

`docker rm <c1><c2><c3>` → delete container c1, c2, c3

`docker run -d --rm -P 3001:3000 <img-id>` → when the container will stop after Running, it will automatically Remove it using `--rm`

`docker run -d --name "Rohit" -p 3001:3000 <img-id>`

→ docker assigns any Random name to container, to avoid this what we can do is, we use `--name "name"`
So it will assign a name to container.

`docker build -t Rohit:01.` → it will basically tag the image which we are building, Rohit:01
→ creates Version of Images (Rohit = Tag, 01 = version)

`docker rm` → to Remove container

`docker rmi` → to Remove docker image

`docker rmi <name:version>` → delete Specific docker image

★) CHANGES in the Project

Make the changes first & create a version tag for it.

`docker build -t Rohit:02.` - it will create Version02 of my file.

★) PRE-DEFINED IMAGES

`docker run -p 8080:80 nginx` → Start the nginx

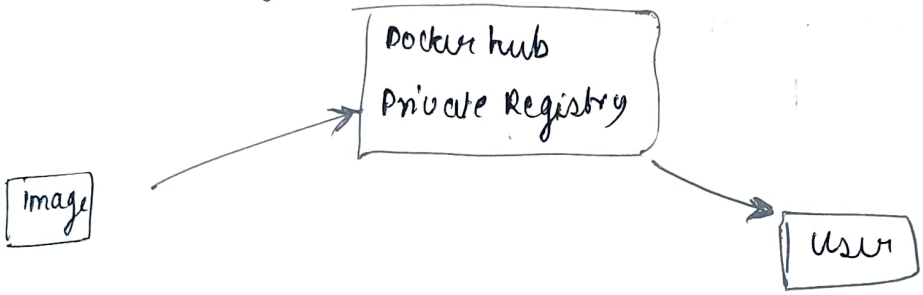
*) Docker Container with interactive mode

→ if there is a Python code that takes the input from the usurse

`docker run -it <img-id>` → Run code in interactive mode because of `-it`

`-it` = interactive terminal

*) Sharing Images DOCKER HUB



2] `docker Push <REPO-name>:tagname` - to Push your docker image into your docker REPO

1] `docker login`

The name of the file should be same as REPOname.

`docker tag <old-name> <new-name>`

→ change docker image name from old to new

*) Using our images Remotely PULL Images

*) DOCKER VOLUMES (-v)

`docker run -it --rm -v myvolume:/myapp/ <img-id>`

creates volume — new name of volume — working directory

→ the command will create a docker volume that can store the data consistently (PERSISTENT DATA STORAGE)

`docker volume ls` → Show list of docker volume

`docker volume inspect`

`docker volume inspect <volume_name>` → Show detail information about volume

*) **Bind Mounts**

```
COPY ./myAPP/
COPY ./server.txt .
```

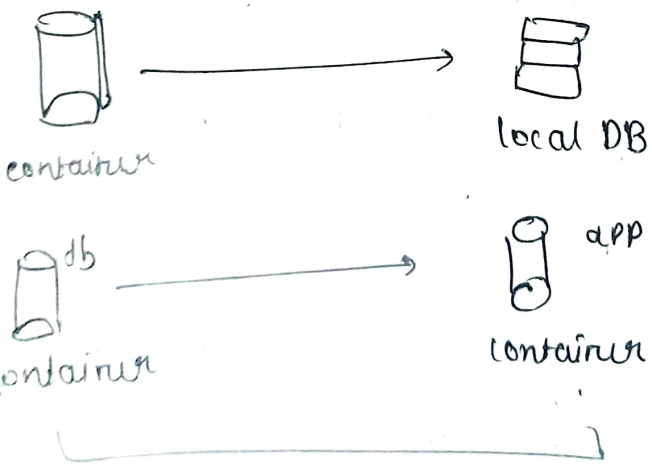
~~docker -v~~

`docker run -v <local_Path_link>:/myAPP/Servers.txt --rm <cont-id>`

→ used to Bind mind local file with the file on docker

*) **.dockerignore** (the files which you don't want to add to docker container)

*) Communication from TO CONTAINERS



How will they connect.

✱) Working with APIs

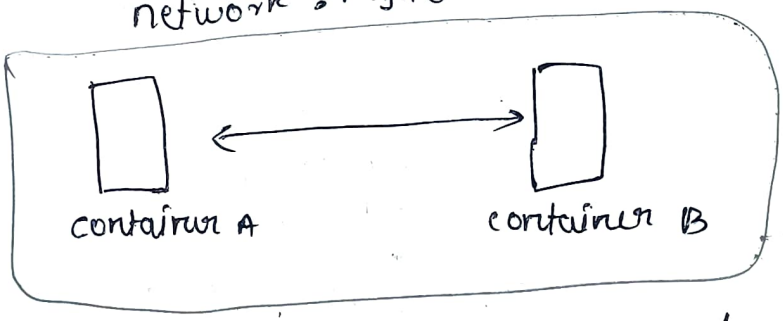
`RUN PIP install Requests`

✱] Container & Local DB.

`docker inspect <cont_name>` → Show info. about container.

✱) Docker network

→ when two containers are dependent on each other then we can Run them in Docker network



→ docker network allows containers to connect and communicate with each others

`docker network create my-net` → create a docker network by name `my-net`

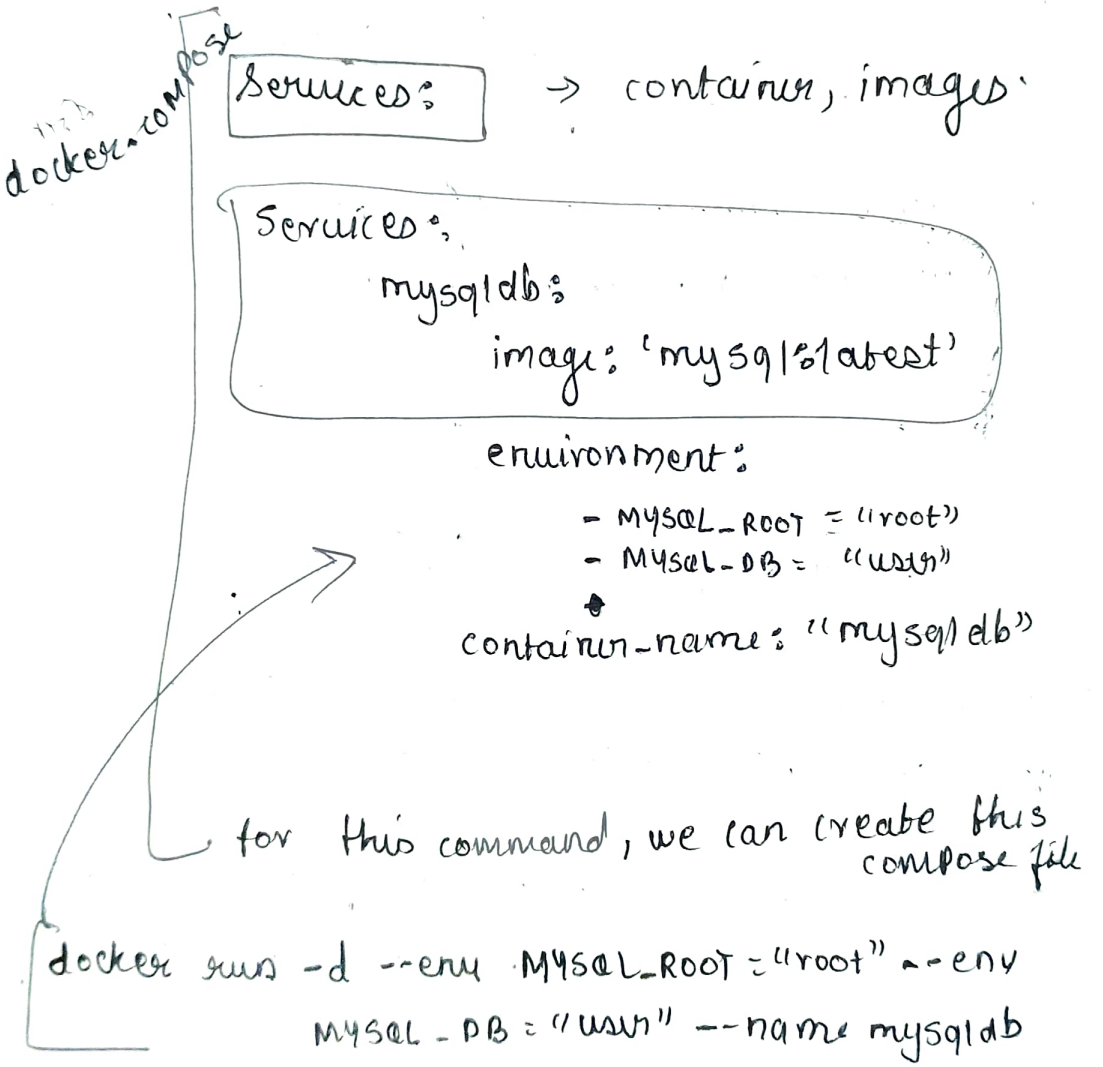
`docker network ls` → Show list of dc nlw.

`--network <nlw_name>` → to use your network

*) DOCKER COMPOSE (docker-compose.yml)

→ it is configuration file which is used to manage our containers. It is a YAML file.

→ used to manage multiple containers Running on same machine.



`docker-compose up` → Run docker-compose file

`docker-compose down` → stop

`./` → current directory

using docker compose we don't have to `--rm` command because after work done it will automatically Remove container.

dependency

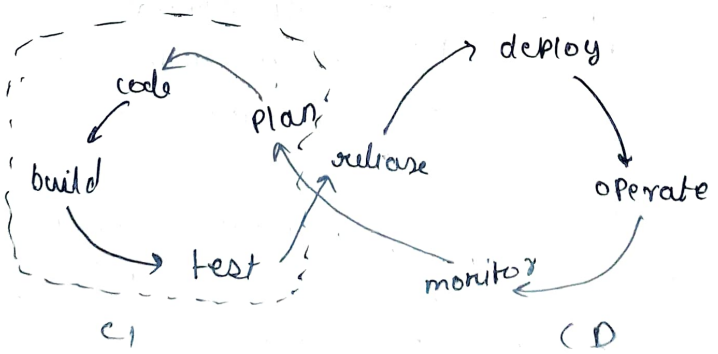
```
depends_on:
  - 'mysqldb'
```

4) docker-compose Network

↳ can automatically create docker network all services, containers which are inside the config file basically shares the same n/w.

RATHER THAN RUNNING LONG COMMANDS YOU CAN USE DOCKER COMPOSE.

4) image - the containerization



Linux container → 2003

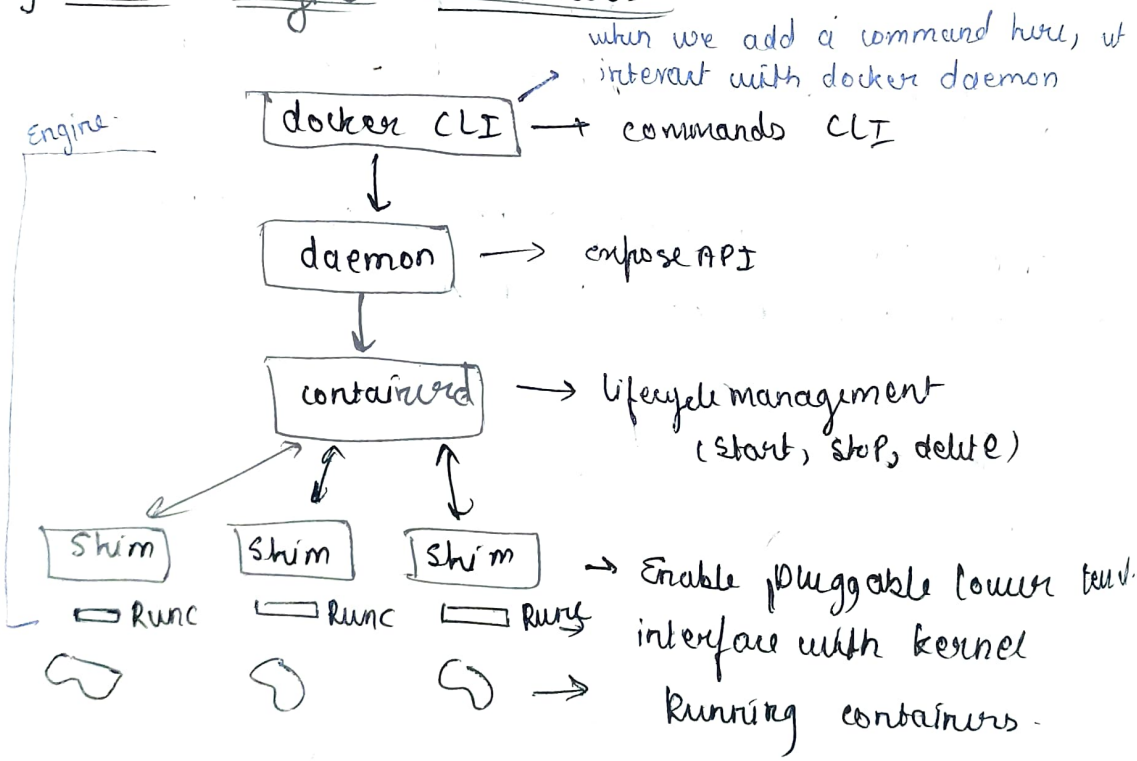
*) Docker container light-weight → because it does not require a whole of operating system. Does not require OS for application

*) docker have pre-defined Images.

*) layers in docker → Each layer means stored memory as cache, so if we run it again it saves time.

*) docker is version control – it maintains all the information in cache.

*] Docker engine components



*] Containerd → is used to Produce containers & builds a Runtime to Run containers
→ management Block, Architecture Person

*) Shim → lower, layer component, team lead
↳ divides the task

*) Runc → Responsible to create container on the basis of instructions.

*) network components → creates a Bridge between

diamon ⟷ components

3 different types of ~~nt~~ n/ws (networks)

1] Bridge n/w → whenever you create a container then Bridge n/w is automatically created.

2] ~~Coast~~ n/w
Most

*] `docker stop $(docker ps -aq)` → stop all docker containers

*] `docker rm $(docker ps -aq)` → Remove all docker containers

*) `docker rmi $(docker images -aq` → Remove all the Images

`docker ps` → List of Running container

`docker ps -a` → list of all containers, stop, Run,

`docker run ubuntu sleep 5` → duration of 5 seconds.

↳ when container starts it runs to sleep command and goes into sleep five seconds to it.

`docker exec <container_name> cat /etc/hosts`

↳ check the contents of a file in a particular container

`docker attach <cont._id>` → if a container is Running in `-d` detach mode then using `attach` you can Run it again in attach mode

`docker inspect <cont._id>` → It returns all details of a container in JSON format

`docker logs <cont._id>` → check logs of a container

`docker network ls` → Show list of docker networks.

`docker run timer` → Prints current time on screen every single seconds

`docker run -P 8080:8080 Jenkins`

A] Memory & CPU (Docker)

`docker Bridge`

`docker inspect` → Shows all data in JSON format & also shows which n/w Bridge

`Ctrl + PQ` → it will exit from container terminal into your own terminal but not stop container

`docker stats` → Shows CPU & memory usage

A] MEMORY OPTIMIZATION

`docker run --itd --memory=512m centos` → Limit the memory to 512m only.

without --memory = Random memory limit will be assign it could be more.

with --memory = Limit memory size.

`docker exec -it <cont_id> /bin/bash` → directly go inside the container.

A] `docker commit <cont_id> docktest:v1` (tag) → this to save the information of the container & what we have done in container. it will be saved in the form of an new [image]

`docker login` → establish the connection between Docker hub and host system.

`docker inspect Bridge` → Show Bridges. (Inspect Bridge

JIRA

1] Issue type scheme → Issue type
2] workflow Scheme → Status & transitions.
3] Issue type screen scheme → custom field mapping
4] field configuration schemes →

`docker Build . -f Dockerfile`

`ENTRY POINT` → will Run as a container

*] ENV Variables in DOCKER

`docker run -e APP_COLOR=blue <cont_name>`

→ adds a environment Variable APP_COLOR

*) DOCKER COMPOSE

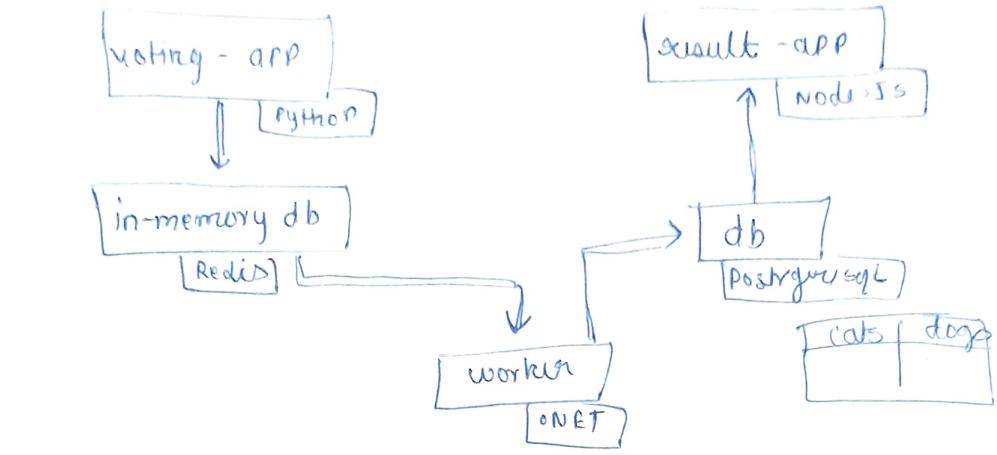
If → docker run Hello
docker run mongodb
docker run Redis:alPine
docker run ansible

```
docker-compose.yml
services:
    web:
      image: "Hello"
    database:
      image: "mongodb"
    messaging:
      image: "Redis:alPine"
    orchestration:
      image: "ansible
```

`docker-compose UP` → to Run the docker compose

`docker-compose down` → shut down docker compose

4) Sample - voting application



-- links → used to link two Containers Together

1] docker run -d --name=redis redis

2] docker run -d --name=db Postgres 9.4

3] docker run -d --name= vote -p 5000:80 --link redis:redis
(host : container)
voting - app

4] docker run -d --name=result -p 5001:80 --link db:db result-app

5] docker run -d --name=worker --link db:db --link Redis:Redis
worker

→ CAN DONE BETTER USING NETWORKING

docker-compose.yml

```
redis:
  image: redis
db:
  image: postgres:9.4
vote:
  image: voting-app
  ports:
    - 5000:80
  links:
    - redis
result:
  image: result-app
  ports:
    - 5001:80
  links:
    - db
workers:
  image: worker
  links:
    - redis
    - db
```

Build

- for the voting application

→ build: ./vote
(location of image)

→ build: ./redis

→ build: ./worker

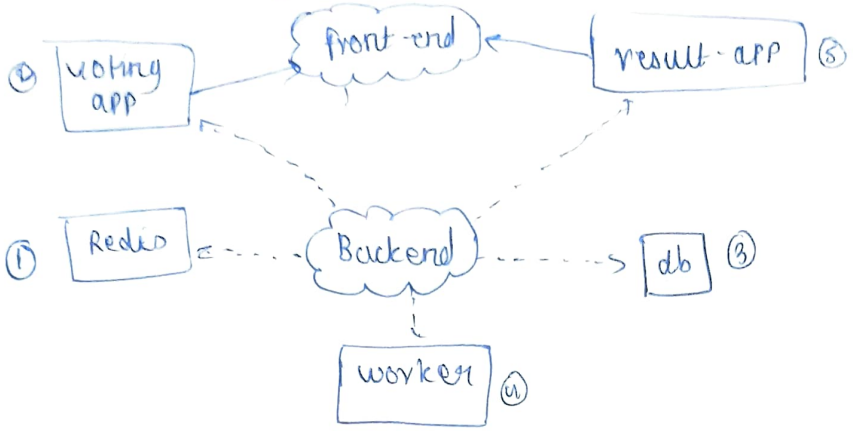
Docker compose - versions

→ version 1: no services section, attach all component to default Bridge network
↳ no depends_on feature

→ version 2: services section available and all the services are defined under the services section. you need to define version: 2 at the top of the file. can create specific Bridge network, no need to use links,
→ depends_on feature

→ version 3: mention version: 3 at the top

A) Docker compose network



docker-compose.yml

```
version: 2

services:
    redis:
        image: Redis
        networks:
            - Back-end
    db:
        image: postgres:9.4
        networks:
            - Back-end
    vote:
        image: voting-app
        networks:
            - front-end
            - Back-end
    Results:
        image: Result-app
        networks:
            - front-end
            - Back-end
networks:
    front-end:
    Back-end:
```

voting-app
5000:80

Result-app
5001:80

*) docker compose is not present By default, you'll need to install it Separately

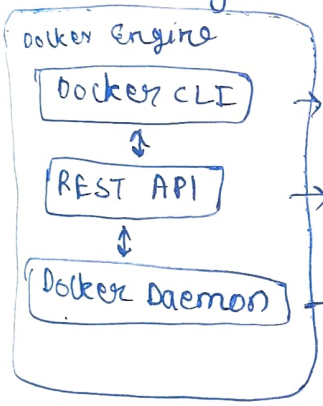
*) docker compose automatically creates a network and add all those files to the created network

Docker Registry = docker.io

*) Private Registry → not accessible to normal users

dot docker login <Private_registry>.io

*) Docker Engine = a docker host with docker installed on it



command line interface, can be on another system & still contact with docker

talks to daemon & Provide instructions

Background process that manages containers, images etc

docker -H=10.123.2.1:2375 run nginx → to Run the commands on a different host

docker uses CPU & memory of the Host Linux

docker run --cpus=.5 ubuntu → the docker will not take more than 50% of the CPU.

docker run --memory=100m ubuntu) limits the memory to 100 mb that docker container can use.

`docker exec <cont_id> ps -ef` → the command → to execute a command inside the docker container

A) DOCKER STORAGE

`/var/lib/docker`

→ docker layered architecture → Save Storage Space

Image layer → Read Only
container layer → Read write
} COPY ON WRITE

`docker volume create data-volume` → creates a new volume under `/var/lib/docker`

`docker run -v data_volume:/var/lib/mysql mysql`

Two ways to create volume.

`docker run \ --mount type=Bind, source=/data/mysql target=/var/lib/mysql mysql`

`ubuntu = AUFS` → Storage drivers

`docker info | more` → tells information about current docker version installed

`docker history`

`docker history <cont-id>` → to see how a docker image is build

`docker system df -v` → Show all the images & their space consumptions

DIRECTORY NAME = CONTAINER ID

4] DOCKER NETWORK

1] Bridge → dfault n/w

2] none — // — --network=none.

3] host `docker run ubuntu --network=host`

(don't have access to external n/w)

`docker network create`
`-- driver Bridge`
`-- Subnet 182.18.0.0/16`
`custom-isolated-network`

→ creating custom n/w

`docker network ls` → Show docker network list

docker use virtual Ethernet Pairs to connect the containers together

`172.12.0.1/24` → Bridge n/w Subnet

5) Docker in windows & MAC

1] Docker toolbox → oracle VM, Docker engine, docker machine, kinimatic cut etc

2] Docker Desktop →

*) Container ~~orf~~ orchestration (Scaling CPU, memory

→ can deploy 100 or 1000's of ~~apps~~ container using a single command

*) container orchestration Tools

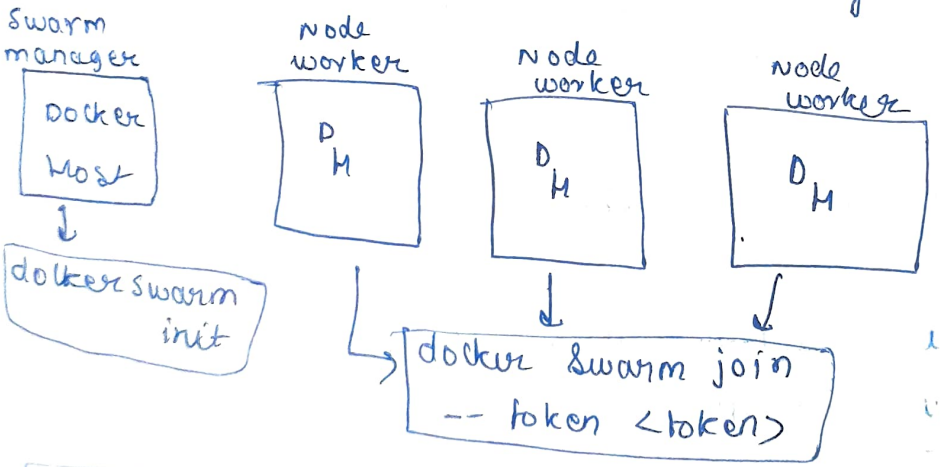
1] Docker Swarm
2] kubernetes – Google (*)
3] Mesos – apache
4] AWS ECS

→ automated management of containarized application across multiple hosts
→ deploying, scaling, managing container across multiple host

if we host our container on different no. of host so if any host goes down then other host will work

*) DOCKER SWARM

→ combines multiple machine into a single cluster



docker service → cmd is used for container orchestration

docker service create --Replicas = 3 my-web

*) features of docker swarm
1] Scaling
2] auto-hosting

1) kubernetes

ephiermal → if we start a container again, the IP will be changed.

→ kubernetes CLI = kubctl

kubectl run --Replicas=1000 my-web

↳ we can Run 1000 instances of same application

kubectl rolling-update my-web --image=web:2

↳ one by one update image

kubectl Rolling-update my-web --rollback

↳ Rollback updates

docker session - 3

→ Port Binding Hosting.

-P → also Port number, but it will take Port number randomly. Very usefull for testing Purpose

use the External IP of the VM → Public IP.

curl 10.124.16.0:80 - will Run the application internally

* If we restrart our docker, it will stop all the Running containers

→ docker update --restart=always <cont-id>

If we restrat the container, it will come back to original State.

*) NETWORK DOCKER

↳ network is used to communicate with multiple containers

`docker attach ~~test~~ <cont_id>`

↳ work same as `exec` command, goes inside container & can execute commands.

→ CUSTOM NETWORK

`docker network create <nw-name>`

↳ creates a custom network, custom n/w can be used for security, so no other can access it

If container are in same network then only they'll be able to communicate. else they won't

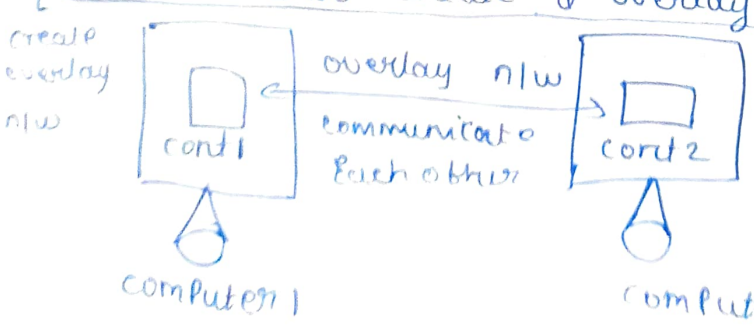
↳ AISO → Automated Service discovery
↳ communicate with use of IP address as well as name.

* Bridge can work on single docker host machine

A] OVERLAY NETWORK

↳ communicate with other container which are at different host machines or computer

`docker network create -d overlay <network_nam>`



→ REPLICA

→ create Replica of your docker container for a proper orchestration.

*] DOCKERFILE (used for automation)

*) CONTAINER ORCHESTRATION

↳ all about managing lifecycle of containers specially in large environment.

1] Provisioning & deployment of container
2] Scaling & Removing containers
3] movement of container from one host to other
4] Load Balancing
5] Health monitoring of containers.

*] minimum 2 web-server should be running all-time

Amazon ECS → container orchestration Tool

docker service ls → shows the list of docker services

*] Node = name of VM Host

COT → automatically detect weather the container is stop or running & if container is stopped then it automatically launch a new instance of container.

(container orchestration)

Swarm cluster → collection of Nodes

manager: Node 01
worker: Node 02
worker: Node 03

→ To deploy your application to a Swarm, you submit a Service definition to a manager node

4) Initialize docker Swarm

1] Manager node command:

```
docker Swarm init --advertise-addr <manager-IP>
```

2] Worker Node command

```
docker Swarm join-token worker
```

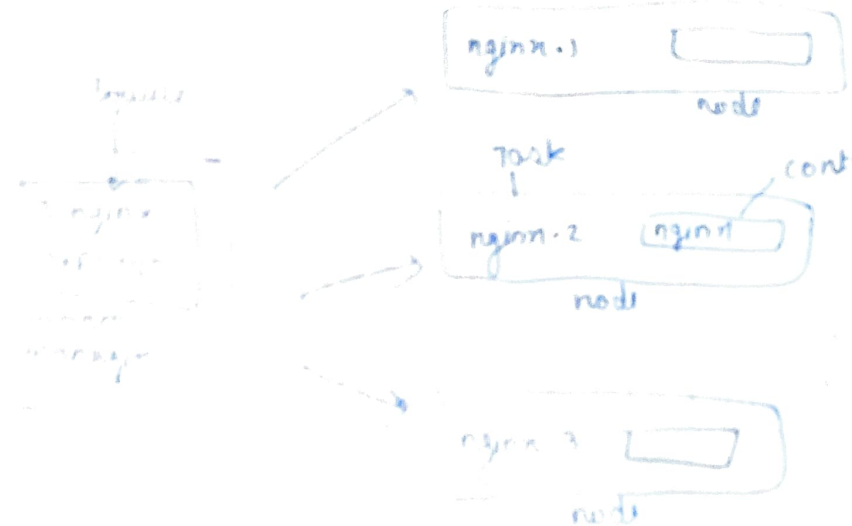
`docker node ls` → Show list of nodes or workers or Slave/manager

Refer gpt

5) Services, Tasks & containers

→ A Service is basically used to create task that gets executed or run on the manager or worker nodes

★ `docker Service create --name web --Replicas 1 nginx`



`docker service ls` → shows list of services

`docker service ps <service_name>` → gives information about service

`docker service rm <service_name>` → Removes a service

*) Scale Services in Swarm

container running in a service = Tasks
Docker CLI can be used to scale

`docker service scale <name> = 5` → Scales upto 5 tasks.

`docker service scale <name> = 1` → Scale down to 1 service

*] Multiple approach to Scale Swarm.

1] docker service scale <name> = 5
2] docker service update --replicas 5 <name>

difference

`docker service scale <name1> = 3 <name2> = 2`

↳ You can scale 2 services at it is possible only with docker service scale command, not with update.

4) Service deployment → Replicated & Global

4] GLOBAL SERVICE → A Service that Runs task on Every node

4] Each time you create or add a node to the Swarm, then orchestrator creates a task & Scheduler assign the task to the new node

docker service create --name antivirus --mode Global -dt ubuntu

↳ mode is global, so it will Run on all the nodes

4) Draining

→ If want Put Some container on maintenance & Stop Some containers but the Service Should be running

docker node update --availability drain <node-name>

↳ drian a node, Stop container, but Run Services on diffrent container.

docker node update --availability active <node-name>

↳ active the Previous node

4) Insfect Swarm Services & Nodes

docker Service insfect <Service-name>

↳ Inspect, but not Easy to Read.

docker Service insfect <Ser-name> --Pretty

↳ Easy to Read due to --Pretty

`docker node inspect <node_id> --pretty` → inspect a Node

3) adding network & Publishing Port to Swarm task

→ Port MAPPING (-- Publish)

```
docker service create --name Romy --Replicas 2 --Publish 8080:80 nginx
```

4) Types of Services → ① Replicated Services
→ ② Global Services

5) Docker Stack

→ a Stack is a group of service or container which are interrelated & share dependencies & can be scaled Together

Stack can compose YAML file

docker Stack deploy – allows user to deploy Services among the Swarm

```
docker stack deploy --compose-file docker-compose.yml mydemo
```
(docker-compose.yml: compose filename; mydemo: Stack name)

`docker stack ps mydemo` → show info. about your stack

`docker stack rm mydemo` → Remove the stack

6) Locking Swarm cluster

→ Swarm cluster have keys TLS, & Security key.

→ Docker lock allows us to have control over keys & Swarm cannot be compromised

```
docker swarm update --autolock=true
```
↳ it will generate a autolock key

`docker swarm unlock` → unlock the swarm & enter the key

`docker swarm unlock-key` → it will show the key

`--rotate` → Rotate key

*) Volume

→ Even if the container is deleted, the volume exist

```
docker service create --name myservice --mount type=volume,
                      source=myvolume,
                      target=/mypath nginx
                                     cont_name
```

*) Control Service Placement

→

Placement constraints

- Replicated & Global Services
- Resource constraint [CPU & memory]
- Placement constraints
- Placement Preference

```
docker service create --name myconst --constraint
                      node.labels.region==blr --Replicas 3 nginx
```

```
docker node update --label-add Region=mumbai Swarm03
```

*) Overlay n/w Security

```
docker network create --opt encrypted --driver overlay
                      my-overlay-secure-network
```

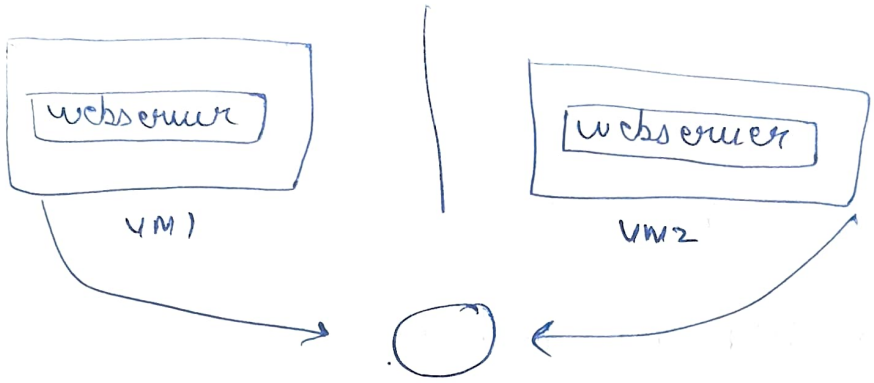
* create Swarm Services using template

```
docker service create --name demoservice
  --hostname="{{.Node.Hostname}}-{{.Service Name}}"
  nginx
```

--hostname
--mount
--env

} — Support Placeholder

* Split Brain Problem



Quorum → how many servers you have within a cluster.

Right no. of Quorum Solves Split Brain problem

always maintain odd number of nodes in cluster

N manager tolerates loss of (N-1)/2 managers

3 (3-1)/2 = 1

`docker system events` → Shows all the Events

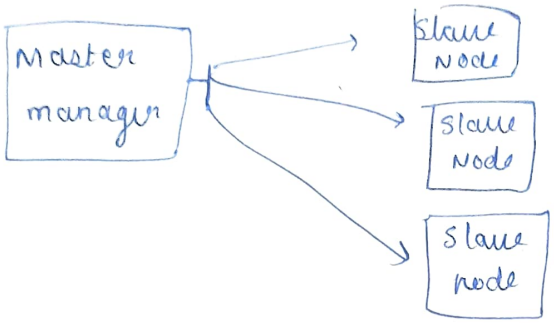
`docker system df` → disk usage

`docker system info` →

Docker Session 3

orchestration Tool uses → Service oriented architecture

1] Docker Swarm → orchestration tool which manages cluster



max → 36 machines can be attached (Nodes) on the Master

Kubernetes → unlimited Nodes can be attached.

Master → High CPU, High memory

```
docker swarm init --advertise-addr <IP of Master>
```

Paste the token link in worker.

```
docker service create -P 8081:80 --name Ronny nginx
```

↳ create a nginx service by name Ronny

```
docker node update --availability Drain <node-id>
```

↳ Drain a particular Node

```
docker node Promote <Node_id>
```
→ Promote as leader

```
docker node demote <node-id>
```
→ demote from leader

docker node ls → Shows which node is leader / worker / Reachable

Dockerfile

FROM Java ⟶ Import from docker hub

WORKDIR /usr/local/bin → creates working directory out off Root folder

COPY test-Prog-jar → copy this file inside container

CMD ["Java", "-Jar", "test-Prog-Jar"] → execute a command

ENTRYPOINT ["Java", "-Jar", "test-Prog-Jar"]

- → cannot enter the terminal of the container, Security Purpose
- → The command that will always Run when the container is start

*) Docker EE → docker enterprise edition

for docker orchestration you need CLI, but using Docker EE, UCP (universal control Plane) you can Easy create, modify container orchestration.

UCP → is used to manage the cluster in docker
↳ helps you manage your docker cluster & application through a single interface

UCP is containerized application that Runs on docker EE

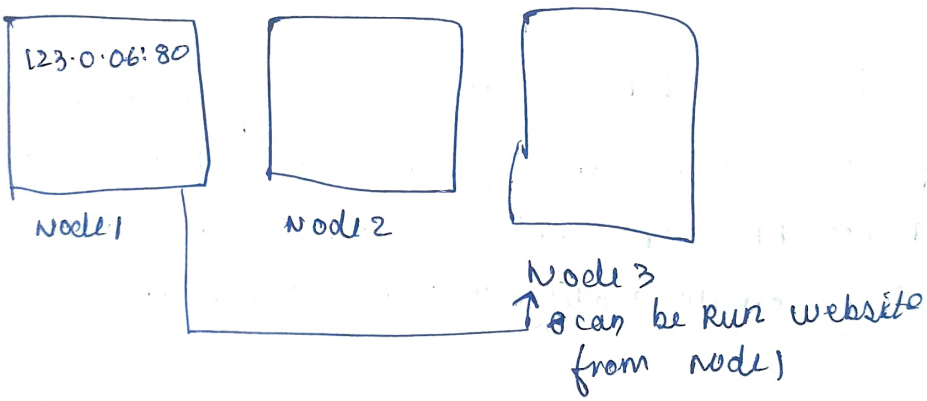
DTR - Docker trusted Registry → Provides a UI for managing the Registry
↳ DTR is a containerized application that runs on the UCP cluster

① Security Scanning
② Image Signing
③ Built-in access control - SSO
④ caching images

*) Swarm Routing Mesh

any node (worker) in the swarm can accept connections on Published Ports for any Service Running in the swarm, even if the Service is not Running on a Particular Node.

eg:- If I run & host a website in Node 1 so that website can be accessed by Node 3 due to Swarm Routing Mesh.



*) DTR can be backed up using AWS S3
*) DTR can be Scale up for high availability

*) Immutable Tags → if any user Pushes the image with a tag, then he can not Push the image with the same tag

* The further away you are from the geographical location where DTR is deploy, the longer it will take to Pull & Push images.

→ DTR caching is used to Resolve the issue
↳ caches are transparent to the users

orchestrator Type - Swarm, kubernetes, mixed

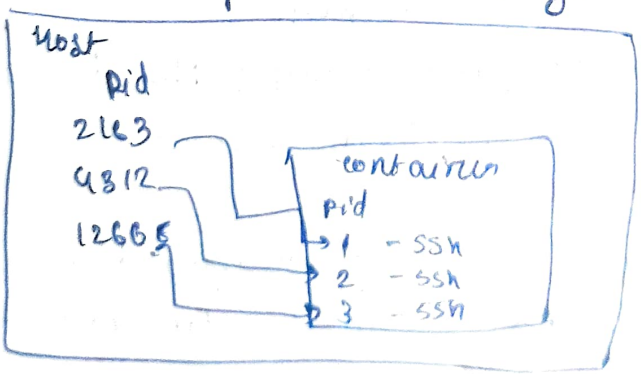
*] CONTAINER SECURITY

→ DTR Provides Security Scan for containers
↳ Scan can be ONPUSH or manually
↳ have a vulnerability Scan Section

*] webhook → Jenkins can be added

*) UCP client Bundles → A client bundle is a group of certificates downloadable directly from the docker universal control Plane (UCP)

*] docker uses a technology called namespace to provide the isolated work space called the container.
→ namespaces provides a layer of isolation



*) Cgroups (control groups) → limit the Resource usage of a collection of Processes (CPU, memory etc)

Limit → Hard limit
Reservation → Soft limit

*) memory Hard limit (cannot use more memory than limit)

```
docker run -dt --name Ron -m 500m nginx
```

Hard limit

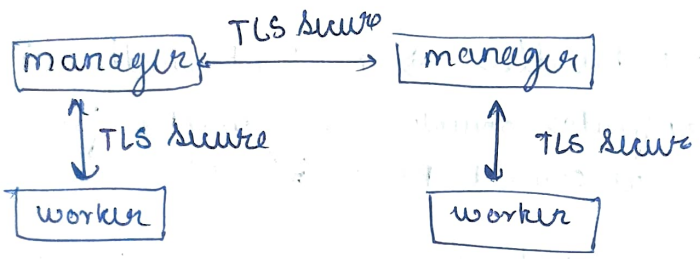
*] memory soft limit (can use more memory than Reservation)

```
docker run ~~-d~~ -it -d -it --name Ron --memory-reservation 250m nginx
```

*) Swarm MTLS

MTLS → mutual Transport layer Security

node in Swarm use MTLS to Securly communicate with the other Nodes.



★ communication ★

```
docker Swarm ca --rotate
```
→ generate a new CA certificate and key

⇒ docker Secret are encrypted during the transit as well as at rest in the docker Swarm.

*] Docker content trust

→ docker content trust basically verifies the integrity of the Publisher of all the data Recieved from a Registry over a channel. This can be done using the digital Signature.

*] Docker Groups

nano /etc/ group

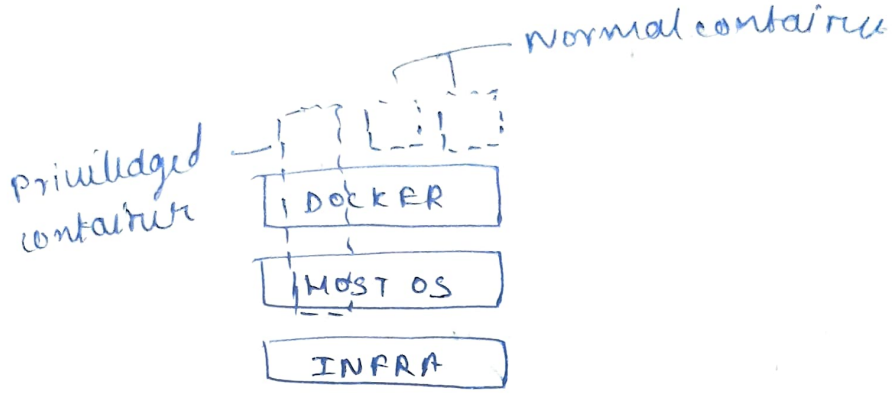
su - new user

usermod -aG docker Ronny

no need for Sudo to Run docker commands

4) Priviledge container

↳ have all the access to the host device as well as can allow other containers to have the access to the host




```
docker run -dt --Priviledge nginx
```
→ Run Prevailedge container

5) Container capabilities

↳ the linux container may have some capabilities & may not have some capabilities

1] add capability

```
docker run -dt --name Ron --cap-add LINUX_IMMUTABLE nginx
```

2] drop capability

```
--cap-drop
```