*] 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**

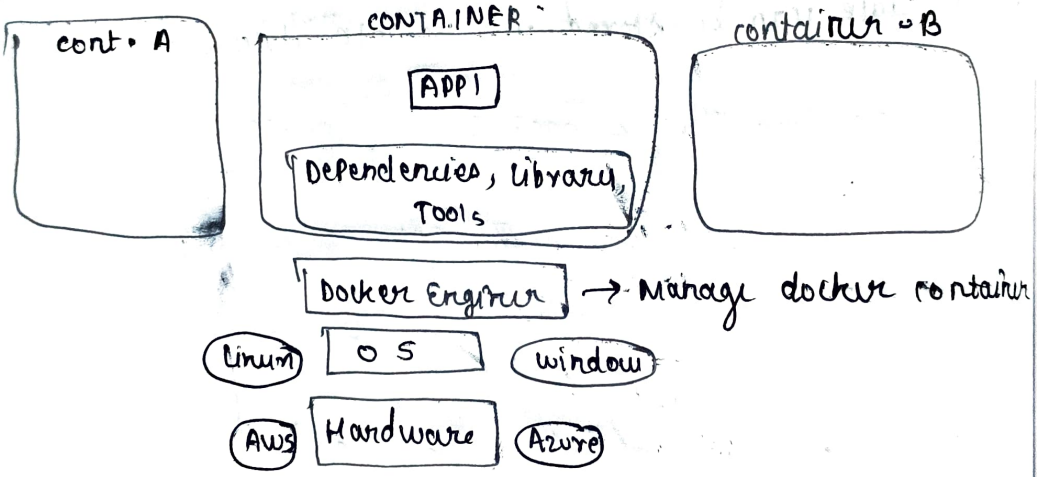
→ makes development & diployment efficient.

*] Container

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

→ container can easily be Shared

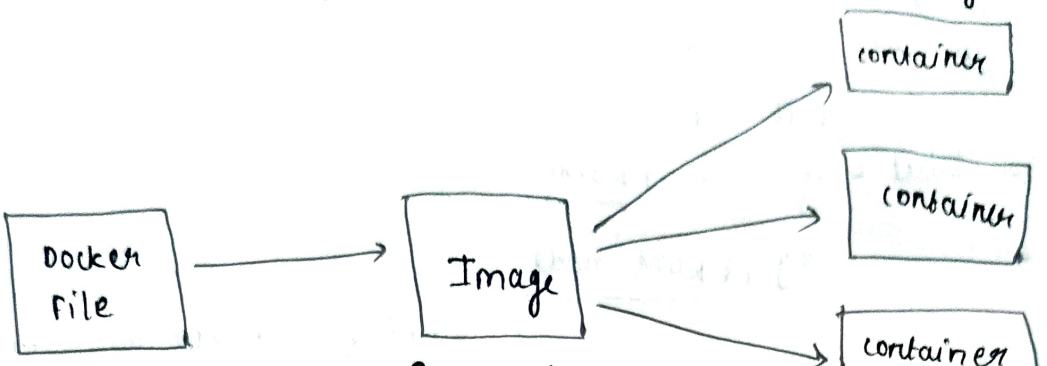
*] DOCKER ARCHITECTURE



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

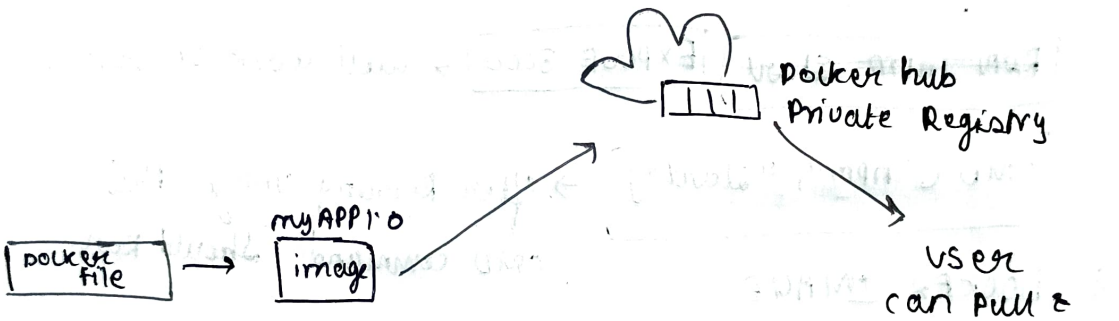
*) Main components of docker



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

Single 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 automatically

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

`WORKDIR /myapp` → create a working directory in container

`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 3000

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

★) DOCKER IMAGE

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

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

↳ `docker images` →

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

`docker ps` → Shows Running container

`docker stop <container_name>` → Stop the docker container

`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>
   ——  ||  ——   3002:3000  —||—
                3003:3000  —||—
```

→ this will Run 3 containers on the Port 3001, 3002 & 3003.

`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 were building, Rohit : 01
↳ creates Version of Images (Tag : 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

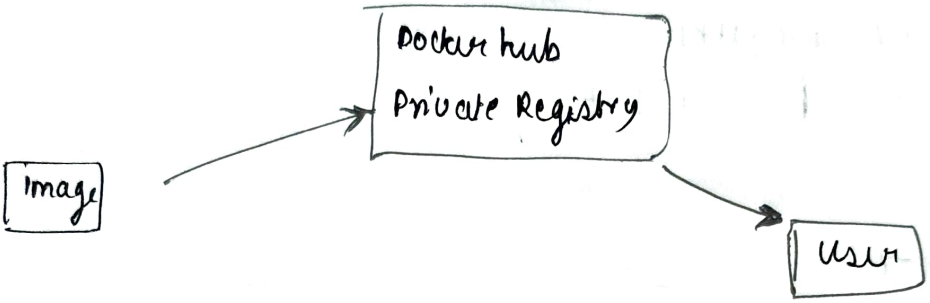
*) Docker Container with interactive mode

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

`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
- newname 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 .
```

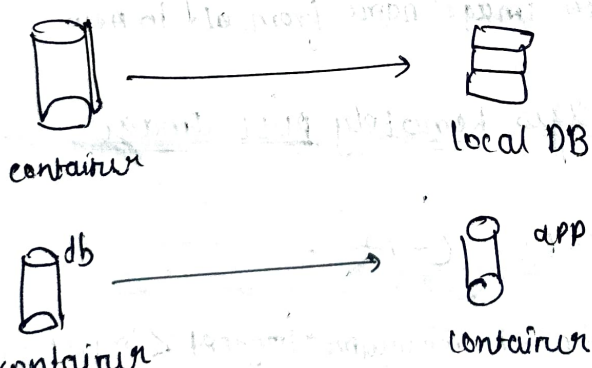
~~docker -v~~

`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 1 TO CONTAINERS



How will they connect.

★) working with API's

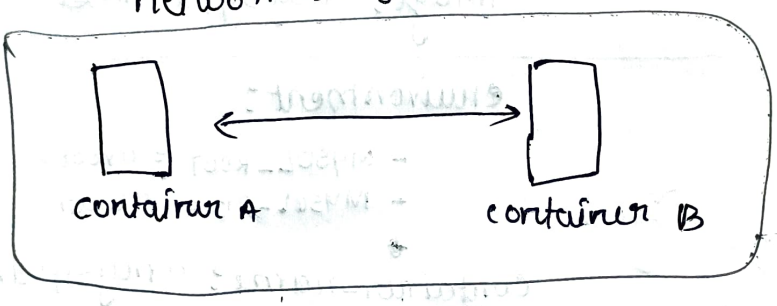
`RUN PIP install Requests`

★] Container & Local DB.

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

★) Docker network

→ when two containers are dipendent 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.yml

```
services:
```
→ container, images.

```
services:
    mysqldb:
        image: 'mysql:latest'
        environment:
            - MYSQL_ROOT = "root"
            - MYSQL_DB = "usr"
        container_name: "mysqldb"
```

for this command, we can create this compose file

```
docker run -d --env MYSQL_ROOT="root" --env
MYSQL_DB="usr" --name mysqldb
```

`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:

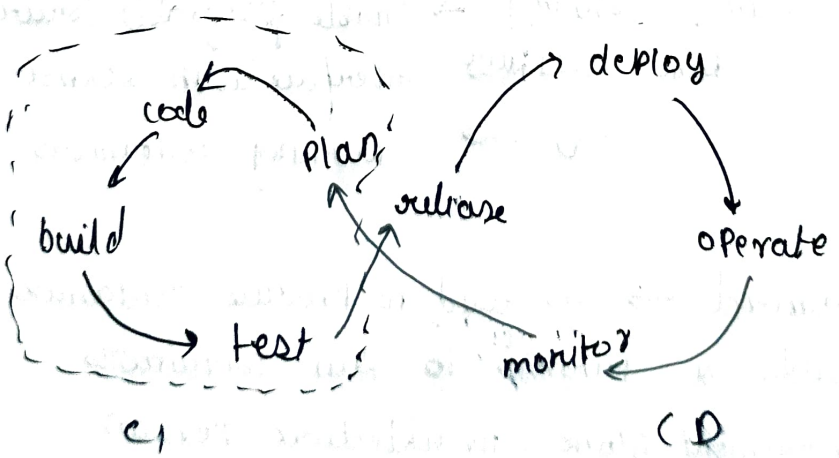
- 'mysqldb'

*) 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.

*) image - the containerization



Linux container → 2003

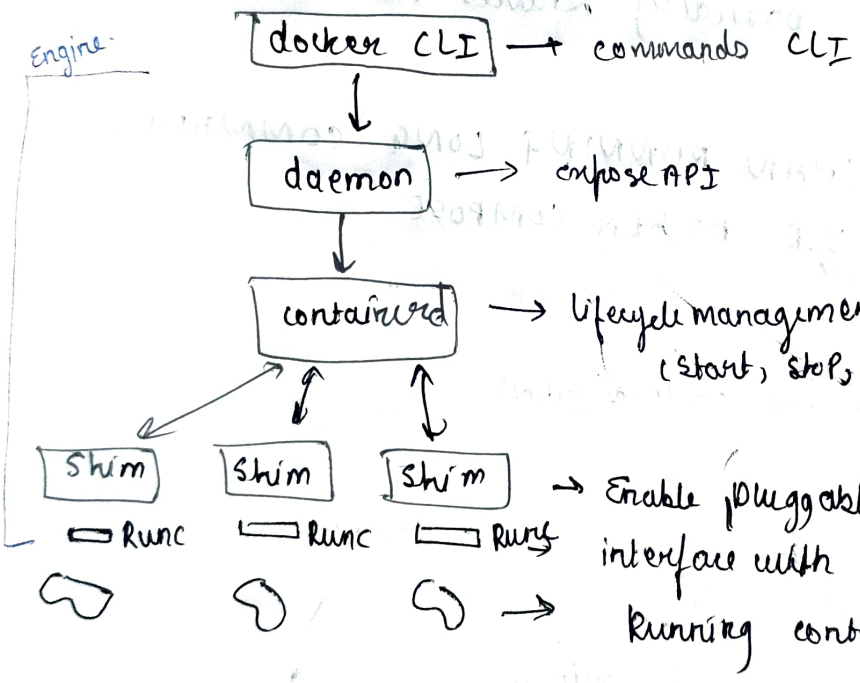
*) Docker container light-weight → because it does not require a whole of operating system. Does not require OS for application

*) 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

daemon ⟷ 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