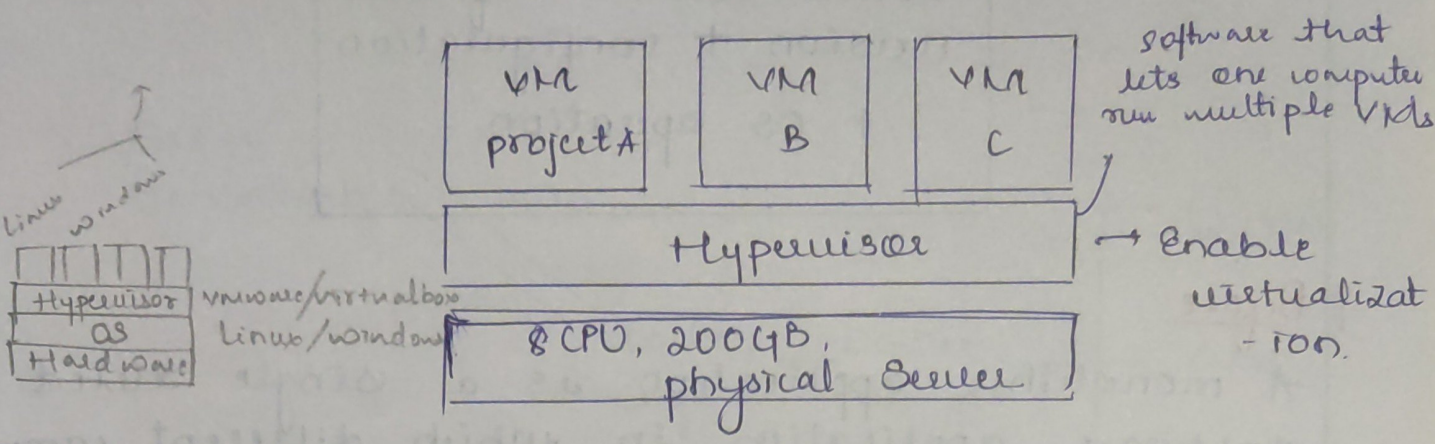
DOCKER

Docker

Virtualization :→



- Before virtualisation companies used to buy the physical server as per the project requirement which was too expensive

= physical server = setup = very difficult

Virtualization make virtual machines as per the project requirement

↳ creating a virtual of something like a computer, storage or network inside another system.

Containerization :→

Developer write the code → operation → Server

↳ to deploy ↗

↓

errors occur

↓

It says it is working properly here ←

Containerization comes into the picture

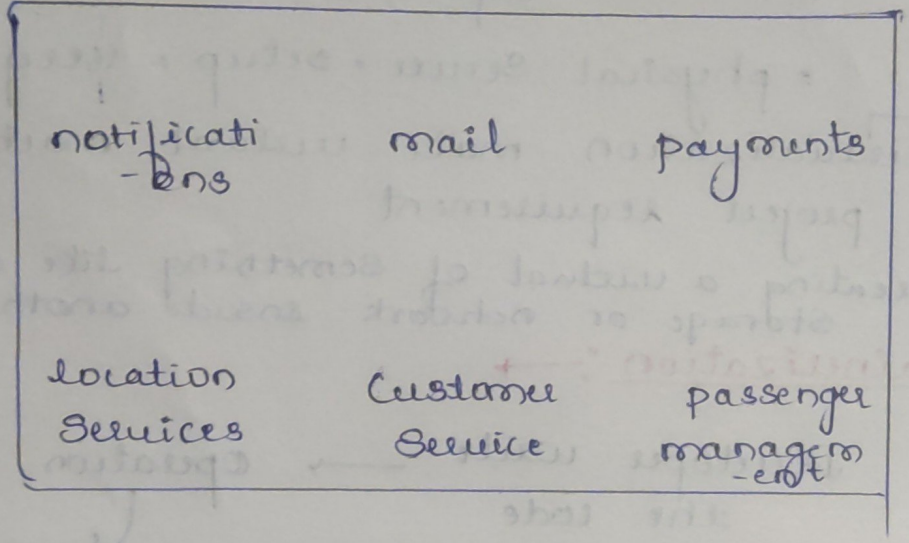
Common reasons :

- Compatibility issue
- Software version
- Dependencies
- Configuration issues
- different OS versions.

Source code + Software version + configuration + OS operation.

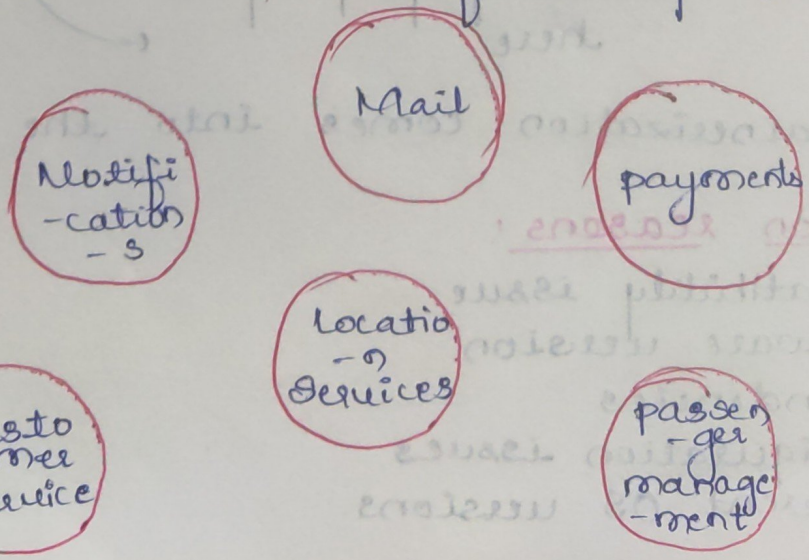
* Before!

A monolithic application is a single tiered software application in which different component are combined into a single program which resides in a single platform.

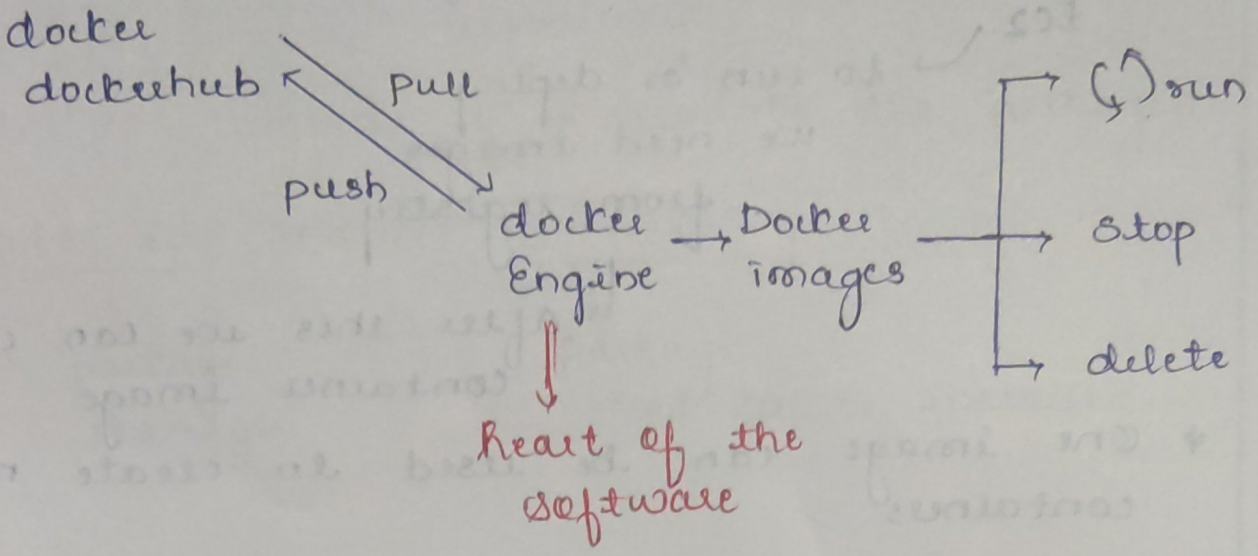


* After :→

Microservices are a software development architectural style that structures an application as a collection of loosely coupled services



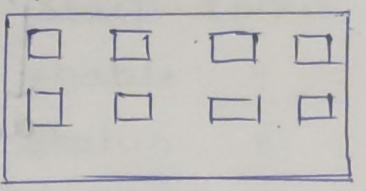
* Docker Lifecycle :→



* Container image :→

Template that holds everything needed to run an application code, runtime, system tools & settings.

Container images is present in image registry



Ex. VMWare / Virtualbox is used to install the Operating System

ISO file, OS image

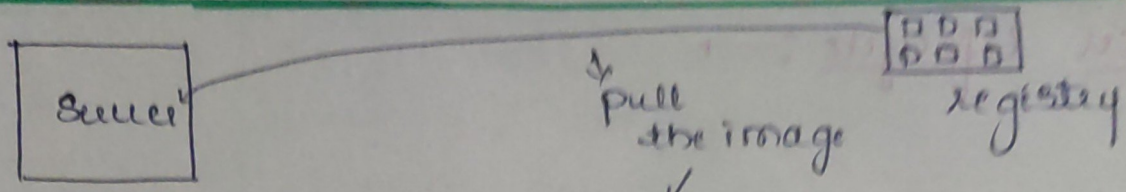
Same docker image is used.

"Dockerhub" → is a image registry, where we can find images & download them

↳ used to deploy

↳ it is a public images registry where we can find the images

Ex:→ ubuntu, centos



EC2 → to run or deploy we need images from registry

↳ after this we can create container image

* One image can be used to create multiple containers

Create EC2 for docker installation

Create EC2 instance (ubuntu)

↓ connect

↓ terminal: Sudo apt update -y (Superuser do)

↓ advanced package tool

Sudo apt install docker.io y

Sudo systemctl start docker

" " enable "

Sudo docker info

Sudo systemctl status docker

Sudo docker pull ubuntu

Sudo docker run -it --name=mycontainer ubuntu bash

inside the container control+P control+Q → Exit

* "Sudo apt update -y" => updates the packages indexes in the OS which are available in the remote package repository

* "Sudo apt upgrade -y" => update the packages

* "Sudo apt install docker.io -y"
  - apt: package manager
  - docker.io: package need to be installed.

* Sudo docker ps
  - it will generate container id, image, names

* Docker Installation

```
Sudo apt update -y
Sudo apt install docker.io -y
Sudo Systemctl start docker
Sudo Systemctl enable "
  "    "      status  "
Sudo docker info
```

* To pull the docker image
  docker pull image-name

* To Search for image in dockerhub
  docker search image-name

* To list the images available in dockerhost
  docker image ls

* To deploy container with container image in interactive mode.

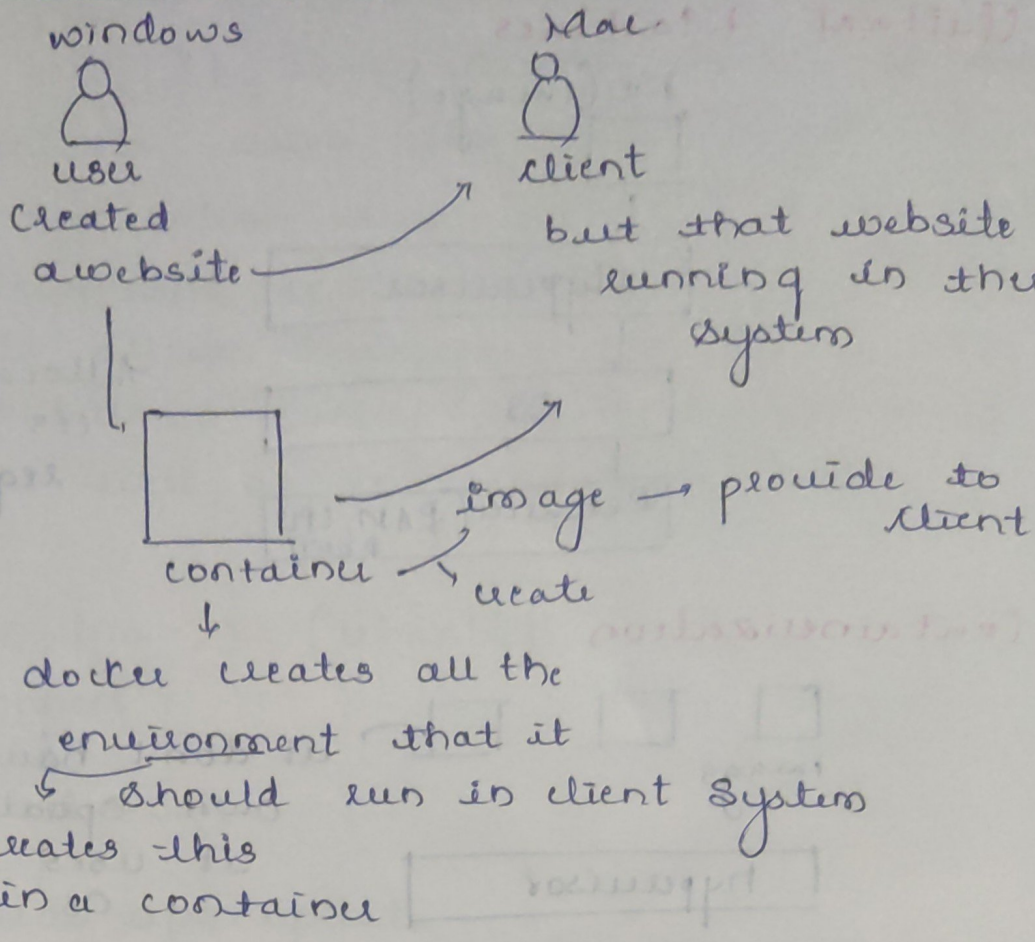
docker run -it ~~-d~~ --name = container_name image_name command_to_execute_when_container_will_be_deployed

Eg docker run -t --name = mycontainer ubuntu bash

* To logout from container.

ctrl+P & ctrl+Q

* Docker image Vs Docker container

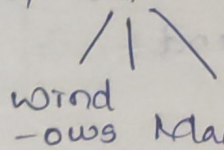


- This problem was faced the company named dot cloud.
  ↳ they created this docker tool

* Virtualization Vs Containerization

Virtual Machines

Hypervisor, OS



It needs a OS to run OS

Ex: I have windows OS to run linux OS one on the other

heavy.

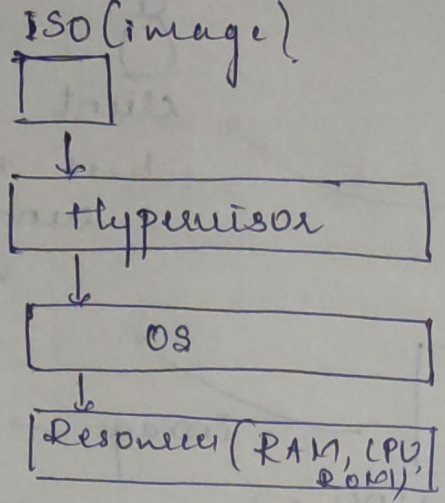
Container docker, podman, containerd.

Docker Engine → not OS

It uses host based Operating System.

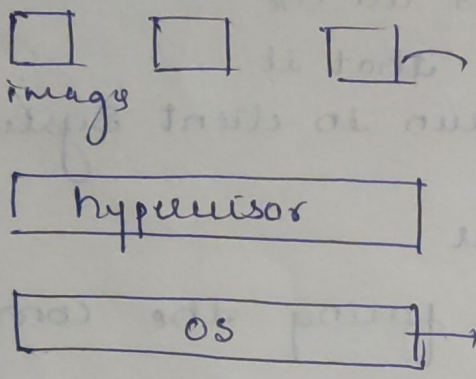
Lightweight

Virtual Machines



Allocate the resources as per the requirement

Containerization



it don't haves its own Operating Systems it uses host based OS

* Docker Architecture :→

Docker Engine → it required.

Docker Daemon → dockerd → containerd

Docker CLI → works to backend = command line interface

Docker client/desktop

↳ API

* docker daemon uses dockerd, containerd

↓ ↳ tool

to run this we need commands "CLI"

now directly communicate

API → tells the docker Engine that this is the requirement

Docker = Containerization tool designed to help the developers build, share & run containers applications.

Docker desktop:

It is a client which is used to communicate with the docker engine.

* It is an application that provides a complete Docker development environment on your local machine. It allows developers to build, run & manage containers easily using a graphical interface as well as command-line tools.

Create EC2 instance (ubuntu)

connect
↓
cmd ⇒ ssh
↓
Sudo apt update
Sudo apt-get install docker-io
↳ local system
Sudo systemctl status docker
docker ps → process status
↳ it shows all the running containers
↳ error

```
Sudo usermod -aG docker $USER
```
↳ add the current user

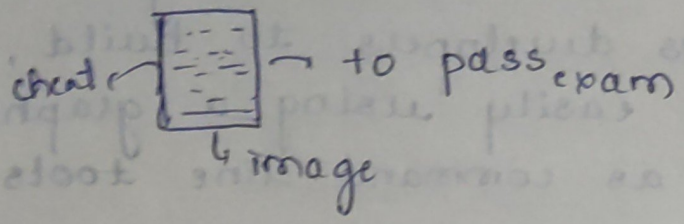
```
newgrp docker
```
→ refreshes the command

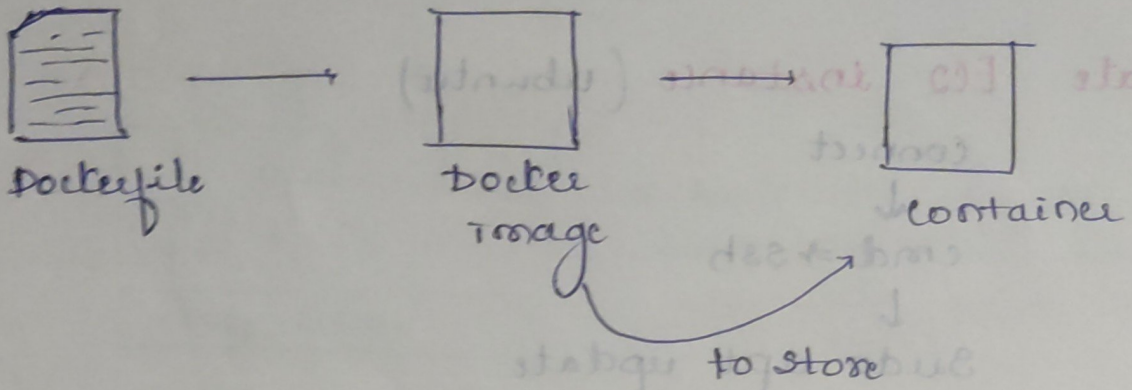
docker ps
↳ now no error

Docker Images & Containers

* docker ps ⇒ it sends the ~~command~~ details to the containerd to show the commands

* Docker image & Containers





- Everytime no need to have dockerfile to make image, we can used prebuild ~~images~~ dockerfile to pull images.

* prebuild dockerfile to pull images

- docker login ⇒ in local terminal the docker will access the desktop docker
  - ↳ if you forget your password.
    - dockerhub ⇒ login the same details which you have for docker desktop
      - settings
      - ↓ Generate tokens
      - ↓ provide name (read, write, delete)
      - ↓ Copy & paste in docker login terminal

↳ once can be done after only if you just run docker run hello-world it automatically pulls

* docker pull hello-world.
  ↳ from repository download the image

+ docker images
  ↳ it shows the images now.

docker run hello-world.

* Steps: →

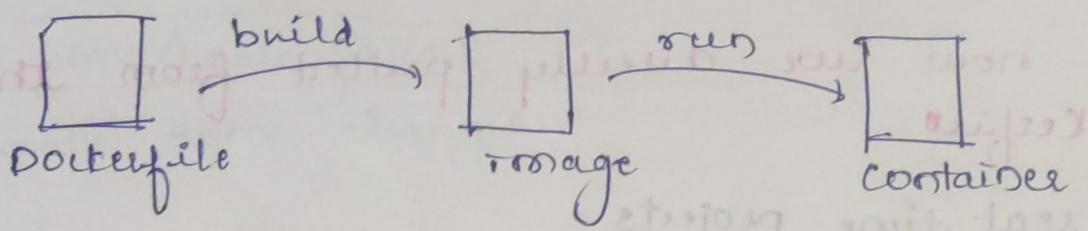
1. The docker client contacted the docker daemon
2. The docker daemon pulled the "hello-world" image from the docker hub.
3. The docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The docker daemon streamed that output to the docker client, which sent it to the terminal

* Another pull command: →

docker pull mysql.
↓
1st it downloads then create the images

docker images

Important



- Build a dockerfile creates images
- run " " " container

- my sql is a server → need password, username,

```
docker run -e MYSQL_ROOT_PASSWORD=root mysql
```

-e → creates environment

no space

↳ after this command you can't exist so create one more terminal connect to EC2 again

- `docker images`
  ↳ you can see the mysql image running
- `docker ps`
  ↳ copy the container ID of mysql to stop
- `docker stop paste`
  ↳ 1st terminal gets unblock

* In case if you don't want to run it in another terminal use

```
docker run -d -e MYSQL_ROOT_PASSWORD=root mysql
```

↳ detach & run in the same terminal.

* Till now we directly pulled from the dockerfile.

In real-time projects
we need to create our own dockerfile

Here we are using java

mkdir projects
cd ~
ls
git clone https://github.com/LondheShubham153/Simple-java-docker.git

cd Simple-java-docker-~~git~~
ls

`rm -v Dockerfile`
↳ verbose shows the deleted file

- `Vim Dockerfile`

```
# pull a base image which gives all required tools & libraries
FROM openjdk:17-jdk-alpine
# create a folder where the app code will be stored.
WORKDIR /app
# copy the source code from your host machine to your container
COPY src/Main.java /app/Main.java
# compile the application code
RUN javac ~~src/~~Main.java
# run the application
CMD ["java", "Main"]
```

build ← RUN, COPY, WORKDIR, FROM
command inject ← CMD
↳ overwrite

- `docker build -t java-app .` → context
↳ After dockerfile we build the image
- docker image
- `docker run java-app`

- `Vim src/Main.java`
↳ make some changes

- docker run java-app
↳ no changes seen ⇒ need to update the image

- `docker build -t java-app .`

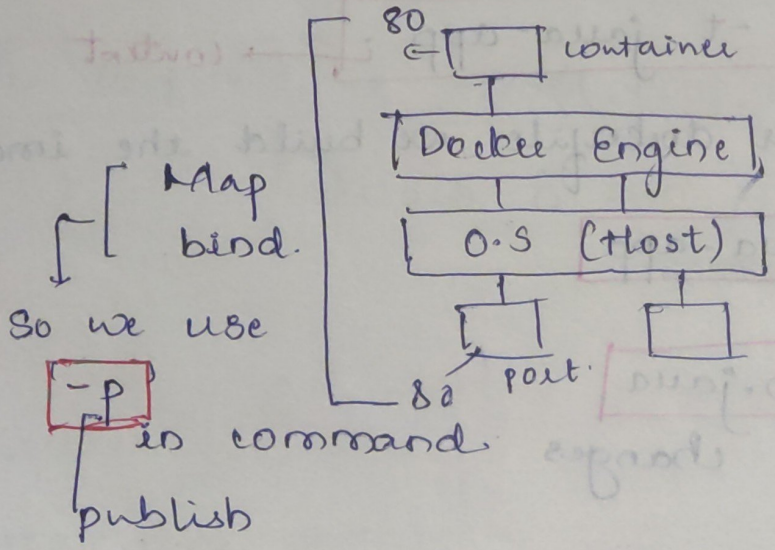
- docker run java-app
  ↳ now the changes can be seen.
  "Vaishnavi, Looking very beautiful"

* Write a dockerfile for python.

vim Dockerfile

```
FROM python:3.7
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
ENTRYPOINT ["python", "run.py"]
CMD ["run.py"]
```

cd projects
flask-app
docker build -t flask-app .




```
docker run -d -p 80:80 flask-app
```

Edit Security group - 80

Access → IP address

after docker PS "____"

* "docker attach" c/p
  - on the browser if I write '/' or refresh or make changes it displays on terminal
  - block the terminal, go to another terminal & unblock it

* docker stop copy & paste
  - id

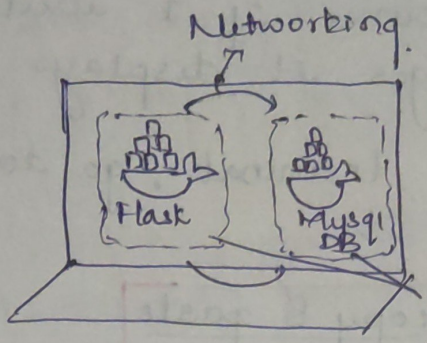
* you can start if you want
  - docker start c/p

* Enter into the mysql
  - docker ps
    - mysql
      - Enter into it
        - docker exec -it c/p bash
          - interactive terminal
            - bash-5.1# mysql -u root -p
              - password:
                - show database
                - create " devops
                - exit
                - exit

docker run c/p

* Containers to keep always running.
  - docker run -itd ubuntu
    - interactive terminal detach.
  - docker run ubuntu => runs & stops immediately

Docker Networking.



Networking is required to communicate with containers

Isolated (Can't Communicate)

Docker Network :→

- Host ——→ local & host of Docker engine should be same.
- default Bridge
  - host wants to communicate to flask the bridge is created
- User defined Bridge (custom)
- None [No network]
- MACVLAN (docker Swarm)
- IPVLAN :
- Overlay

→ docker Swarm

docker Swarm Ex: we created 1 container in mysql in same way we can create multiple in clusters

`docker network ls`

* Create your own network :→

```
docker network create mynetwork -d bridge
```
↳ driver

```
docker network ls
```

* project two-tier application :→

```
cd projects
git clone https://github.com/LondheShubham/two-tier-flask-app.git
```
↓ c/p

```
ls
cd two-tier-flask-app
docker build -t two-tier-backend .
docker images
```

* `docker run -d -p 5000:5000 two-tier-backend`

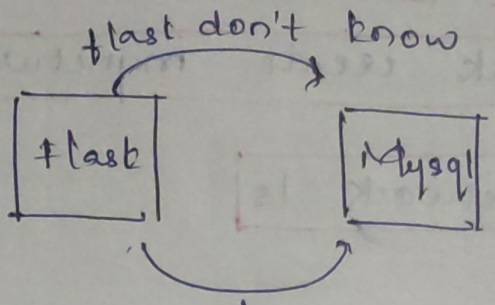
```
docker run -d -p 5000:5000 -e MYSQL_HOST=mysql
-e MYSQL_USER=root -e MYSQL_PASSWORD=root -e
MYSQL_DB=devops two-tier-backend:latest
```
↑ tag

```
docker ps
```
↳ can't see the container

```
docker ps -a
```
↳ copy code of backend latest

```
docker logs paste here
```

Get error ⇒ because



↳ To remove the error we should bring them on same network

- Create a network for flask

```
docker ps
docker stop c/p && docker rm c/p
```

↳ remove mysql for this moment

```
docker network create two-tier -d bridge
```

↳ create a network to connect flask & mysql.

```
docker run -d --name mysql --network
two-tier -e MYSQL_ROOT_PASSWORD=root -e
MYSQL_DATABASE=devops mysql
```

```
docker run -d -p 5000:5000 --network
two-tier -e MYSQL_HOST=mysql -e MYSQL_
USER=root -e MYSQL_PASSWORD=root -e
MYSQL_DB=devops two-tier-backend:latest
```

```
docker ps
```

↳ now we can see the two-tier-backend is running

* To check if the n/w of Flask & mysql is in one network.

docker network ls

docker network inspect two-tier

↳ it will show both the containers are on same network / not

Edit inbound rules

↳ add 5000

copy ip-address ⇒ paste on chrome with :5000.

↳ Hello Namaskaar Nyc devops

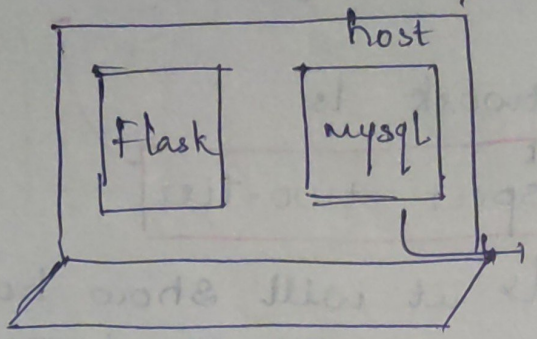
```
docker exec -t c/p bash
mysql -u root -p
use devops;
Select * from messages;
```

Seen here

* If you delete the data will be completed deleted. you can't access back again.

↳ So we use volumes.

Docker Volumes & Storage



By volumes we can recover data as it binds with the host

if it crashes data gone Completely

* Create volumes

docker volume create mysql-data

docker inspect mysql-data

docker ps

↳ mysql = remove

docker stop c/p && docker rm c/p

docker restart c/p

↳ of flask app

↳ docker run -d --name mysql --network two-tier

(1) -v mysql-data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=devops mysql

volume

docker inspect mysql-data

↳ copy path

Sudo su

cd paste

ls

↳ Here you can see that the data is completely secured.

exit

② docker run same.

docker ps
↳ two-tier-backend:latest
↳ c/p
docker restart ↓ paste
↳ data remains same.

Another form to Create Volumes

ubuntu@ip-172-31-7-73 = mkdir volumes
ls
cd volumes
mkdir mysql
ls
pwd
cd mysql
pwd
cd
cd projects/two-tier-flask-app
docker ps
[docker stop mysql && docker rm sql.]

docker run -d --name mysql --network two-tier -v /home/ubuntu/volumes/mysql:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=devops mysql.

cd
cd volumes/
ls
cd mysql/
↳ All data backedup

Docker Compose

* Docker compose :- It is a tool used to automate easily
  ↳ used to make 1 or more containers

ubuntu/projects/two-tier-flask-app = vim docker-compose.

TO install docker-compose
= ~~sudo~~
sudo apt-get install docker-compose-V2

docker compose up
↳ run the docker compose command.
↳ docker build & run automatically.

docker compose up -d
↳ detach

Docker Registry

* Docker registry :- It is a storage & distribution system for docker images. It allows you to push, pull & manage container images.

docker System prune → deletes unused images, containers & volumes

1st docker images

2nd docker rmi c/p all the images you want to delete

* `docker container prune` => delete all stopped containers
* `docker image prune -a` => Delete all unused images
* `docker volume prune` => Delete all unused volumes

* Delete docker images at a time
  - => `docker images -aq`
    - ↳ lists all the images id
  - * => `docker rmi -f $(docker images -aq)`
    - `rmi` → remove images
    - `-f` → forcefully

`docker compose down` => stopps the containers & then deletes
- ↳ deletion

`docker compose up -d --build` => forcefully builds containers in the 'docker compose

docker ps
Ex: two-tier-flask-app - "___"

`docker attach` paste
- ↳ directly you can see the changes made on browser through ip

wanr to push from local system to dockerhub

`docker login` ⇒ Ensure with docker login

1st name of image

`docker image tag two-tier-flask-app-flask:latest trainwithshubham/two-tier-backend:latest`

username of docker created → `vaishnavitikke` → meaning

renames the image

`docker images`

↳ it shows repository present in the local system.

`docker push vaishnavitikke/two-tier-backend:latest`

↳ push from local to dockerhub

↓

you can see it in docker

Multi Stage Docker Builds

```
cd projects
ls
```

↳ flask-app-ecs simple-java-docker two-tier-flask-app

```
cd flask-app-ecs
docker build -t flask-app:latest .
```

Vim Dockerfile

↳ large image - python 3.7 is required to download the packages, requirements dependencies

if we r using this the image generated will be large only

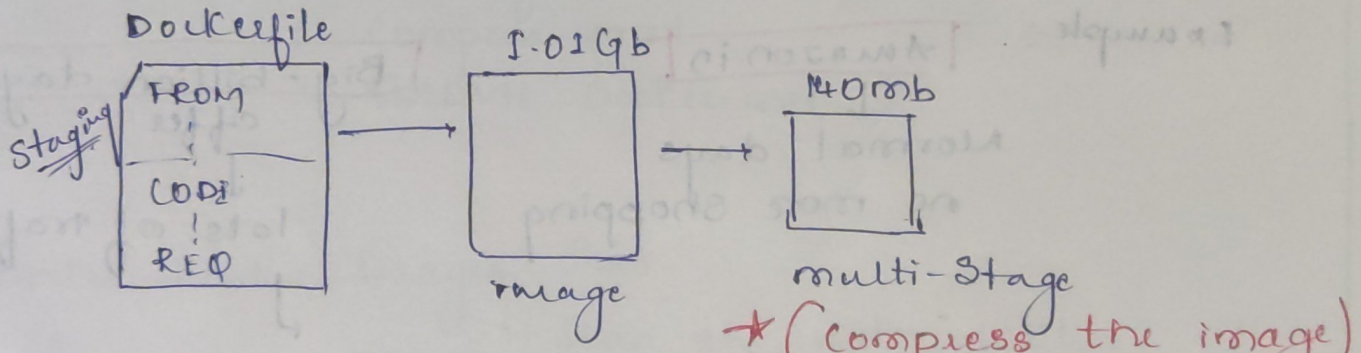
but after installed no need we can go with small image (slim)

vim Dockerfile → large

```
FROM python:3.7 AS builder
WORKDIR /app
COPY requirements.txt.
RUN pip install -r requirements.txt
FROM python:3.7-slim → makes Small.
WORKDIR /app
COPY --from=builder /usr/local/lib/python3.7
/site-packages/ /usr/local/lib/python3.7/site
                                -packages/
```

`docker build -t flask-app-mini .`

docker images



* (Compress the image)

* Monitoring & logging in docker

```
docker run -d -p 80:80 flask-app-mini:latest
docker ps
docker logs paste
nohup docker attach paste &
  ↳ runs backend & store it in file
ls
↳ nohup.out ← refresh, error, on chrome
cat nohup.out
```

* Docker is not used in production because

docker stop 4b
↳ if I don't write complete id only starting 2alpha-bets or number if I write it gets stop

docker rm 4b => same it deletes easily.

↳ it crashes easily

- So the docker is used with kubernetes Orchestration.

* Kubernetes → in background of it Docker containers are only running.
  - Autoheal
  - Autoscale

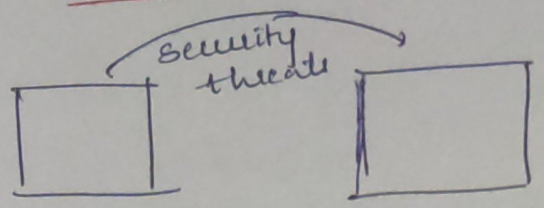
Example:

Amazon.in
↓
Normal days
no more shopping

Big-billion-days offer
↓
lots of traffic
↓
So the container gets crashed in docker recovered or Autoheal by kubernetes immediately

Same with traffic
If lots of traffic present on docker kubernetes helps to Auto-Scale easily

* Docker Scout



to securely connect to the web or create 2-tier flask app we need. Scout tool for security

- It will scans is there any security threats in base image or the final image.

* Docker init

↳ it helps in creating

1. .dockerignore
2. Dockerfile
3. compose.yaml
4. ~~README~~.Docker.md.

* Docker Scout = Images Scan
  ↳ Show the vulnerabilities

* Docker Image = read only templates that contain instructi-ons for creating a container.

* Docker container = an isolated process with all the files it needs to run.

* " Volume = persistant data storage for containers