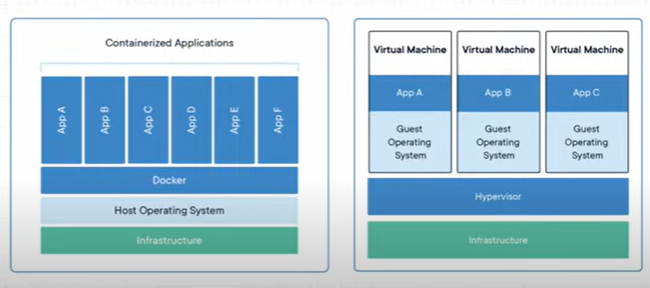
**Docker**

**VM vs Docker**

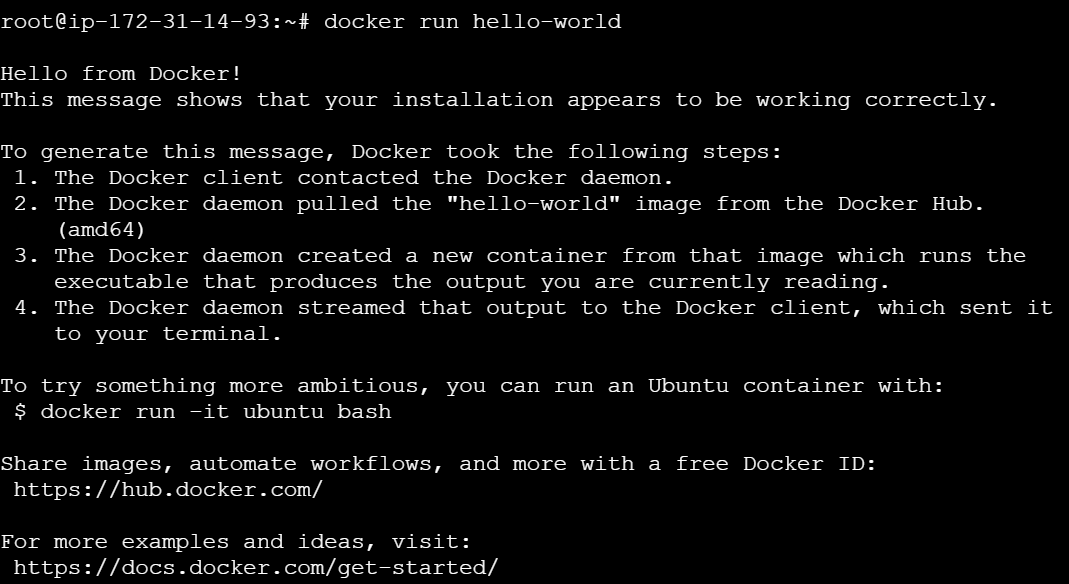
****

**Installation in Linux Ubuntu AWS EC2**

* curl -fsSL https://get.docker.com -o get-docker.sh
* sudo sh get-docker.sh

**To get the access for a particular user to work on docker**

* sudo usermod –aG docker user name

****



if we want to directly run a image we can use,

* docker run ‘image’

or else we can pull the image by mentioning image tag and then we can run

* docker image pull ubuntu:latest # here latest is a tag
* docker run -it 4e5021d210f6 /bin/bash # see above example in screenshot

# here ‘-it’ is used in the command because to make the current terminal as console for the ubuntu

image which we are using

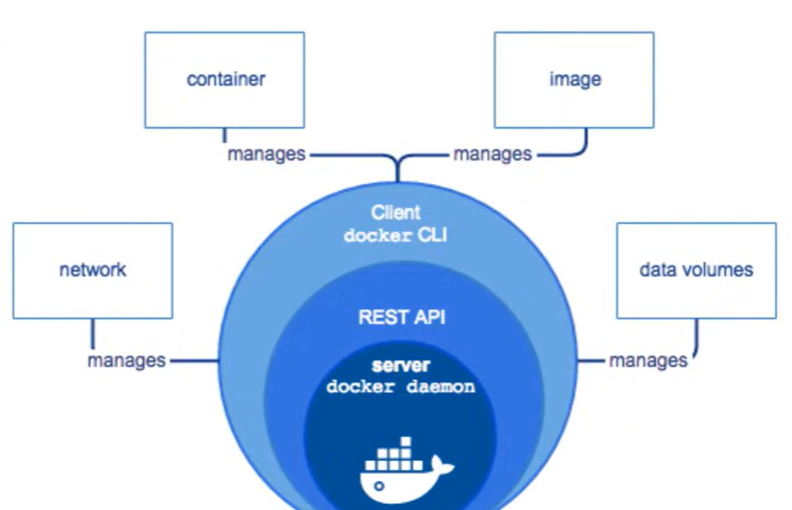
**To list all images**

* docker image ls

ex:

A black screen with white text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.  
  
A diagram of containers with text

AI-generated content may be incorrect.

# With one image we can create multiple container and we can run, let say we can run 100 containers

also with that one image.

# Let say a application is running in one container with particular image, in another container we can

store the data of this application also.

But if we delete the container where application data is present, then data also we be lost.

# so in order to not to loose the data we use **Volumes**.

# now we if we want to store the data in the containers we have different storage drivers

basically. One of the famous driver is **Overlay2**

**Docker Commands**

# to list the running containers

* docker ps

# to list all containers which are running and exited

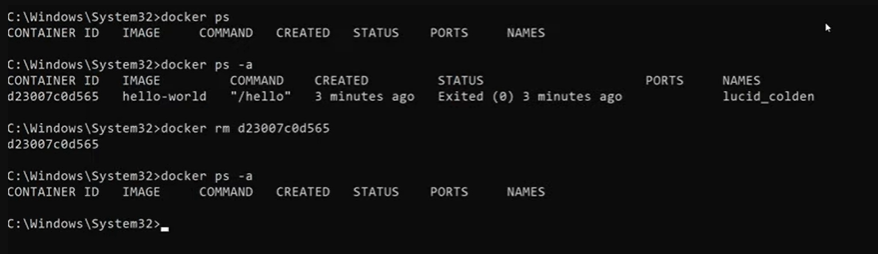
* docker ps -a

# to stop the container

* docker stop ‘container id’

# to remove the container

* docker rm ‘container id’



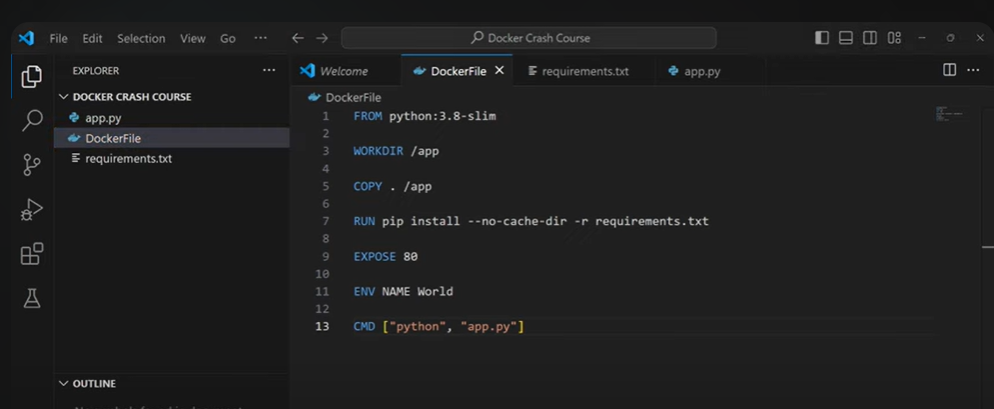
# to list all images

* docker image ls

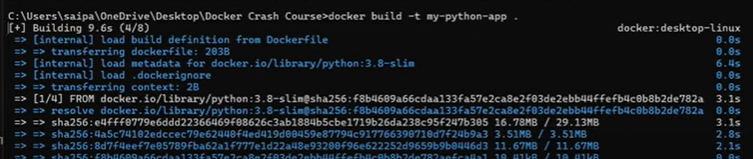
# to remove image

* docker rmi ‘image id’

Creating a Dockerfile



# to build the Dockerfile



Here ‘.’ Is context or the current directory



# to run this image

A screen shot of a computer program

AI-generated content may be incorrect.

Here -p is for port,

here 4000 is your host OS or local machine port number, that which you want to run on.

80 is the port number that you exposed in Dockerfile

Docker Compose

Whenever a larger enterprise application if we are running in docker then, multiple of services will be

there in the application. We tend to use multiple containers. So at this point we use docker compose.

Docker compose config is return in docker-compose.yml file

Example:

A screenshot of a computer

AI-generated content may be incorrect.

Now if you want to execute something on the existing container,

A screenshot of a computer

AI-generated content may be incorrect.

Here ‘exec’ means execute

* Two ways to Dockerize the application

1. Creating and running a app through docker commands, docker compose, Dockerfile
2. First after app is created, then creating a Dockerfile and building an image and run an image.

some commands

Dockerfile, index.html

* docker build -t simpleapp:latest .
* docker run -it -p 8080:80 simpleapp:latest

if you are using windows pc, here 8080 is localhost port number, 80 is application port number, to test it

hit “localhost:8080” in the browser.

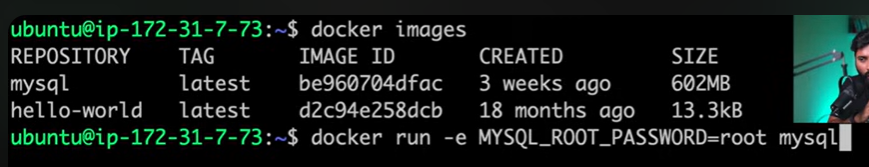
if you are using EC2 add 8080 in security group and to test it hit “Public IP of EC2:8080” in the browser.

**Docker Network**

A computer with a drawing of ships and words

AI-generated content may be incorrect. A black screen with white text

AI-generated content may be incorrect.

* docker pull mysql
* 

Detach mode

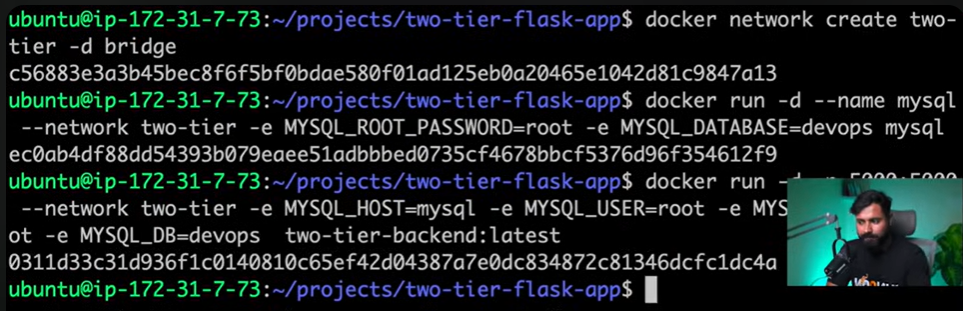
* Docker run -d -e MYSQL\_ROOT\_PASSWORD=root mysql

--------------

**Creating two-tier- application and mysql server in same custom network**

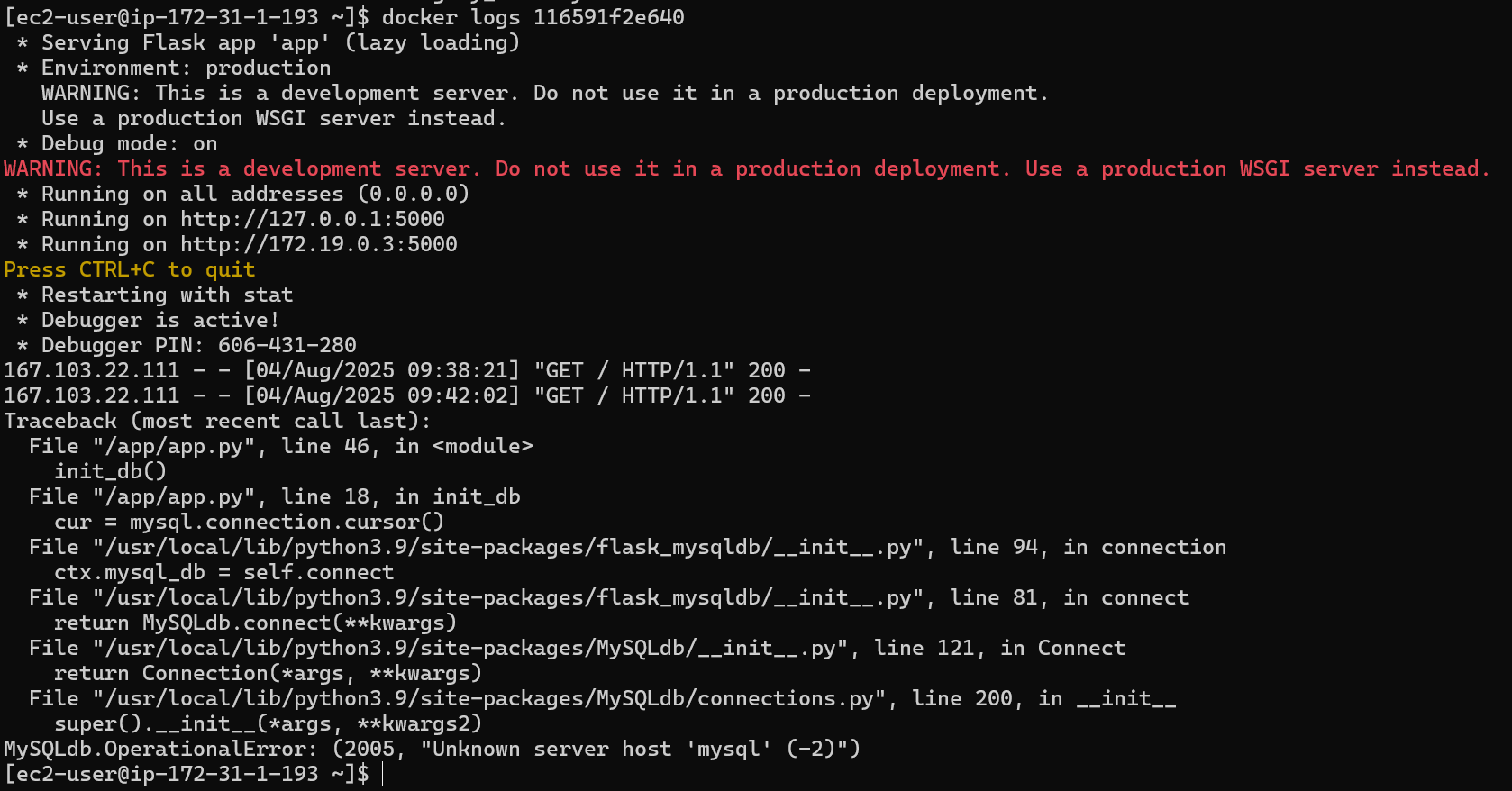
A screen shot of a computer program

AI-generated content may be incorrect.

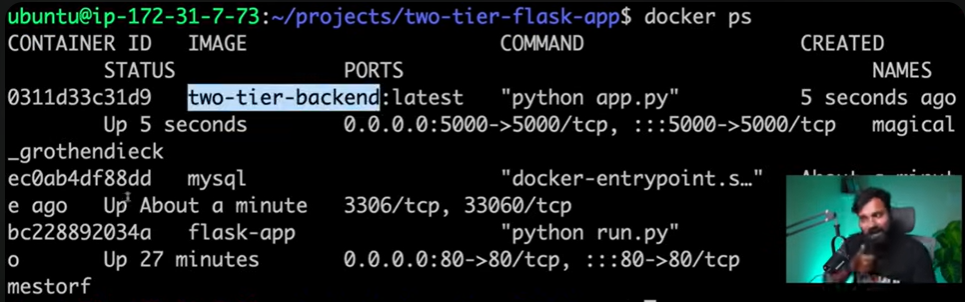


* **Command 1**: docker run -d –name mysql –network two-tier-network -e MYSQL\_ROOT\_PASSWORD=root -e MYSQL\_DATABASE=devops mysql
* **Command 2:** docker run -d -p 5000:5000 --network two-tier-network -e MYSQL\_HOST=mysql -e MYSQL\_USER=root -e MYSQL\_PASSWORD=root -e MYSQL\_DB=devops two-tier-backend:latest

# run the above command 1 first, in order to run command 2 command or else you will get following errors



Now two-tier-backend and mysql are running



* docker logs “container\_id”
* docker network ls
* docker network inspect two-tire

--

* docker exec -it “mysql container\_id” bash
* mysql -u root -p
* Enter password: root
* Show databases;
* Use devops;
* Show tables;
* Select \* from messages;

-----

* docker restart “mysql container\_id”

still data will be available in the database.

----

Now stop the mysql container and delete the container

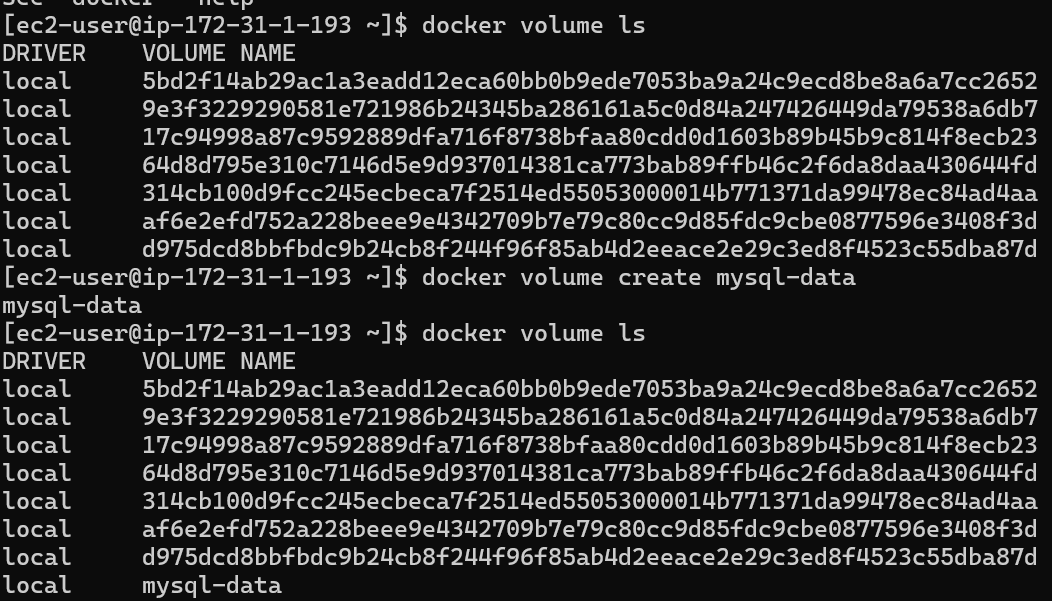
Now data wont be available in the database.

* docker stop “mysql container\_id” && docker rm “mysql container\_id”

**Docker Volumes and Storage**

so whenever mysql container exited or restarts, all its data will be lost

so in order to persist that data we use volumes.



A screen shot of a computer

AI-generated content may be incorrect.

When you create a volume it creates it in our ec2 host at that “mountpoint” location,

Now we need to bind that path to the container path with the below command

* docker run -d --name mysql --network two-tier-network -v mysql-data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=root -e MYSQL\_DATABASE=devops mysql

after running mysql container newly

restart two-tier-backend container

**Running Docker Containers Forever**

To ensure Docker containers run continuously or restart automatically after exiting, follow these strategies:

1. Use Docker Restart Policies

Docker provides built-in restart policies to control container behavior after it stops or crashes.

Command:

docker run -d --restart unless-stopped your-image-name

Restart Policy Options:

- no: Do not restart the container (default).

- on-failure: Restart only if the container exits with a non-zero status.

- always: Always restart the container if it stops.

- unless-stopped: Restart always unless explicitly stopped by the user.

2. Using Docker Compose

Docker Compose allows you to define restart policies in a YAML configuration file.

Example docker-compose.yml:

version: '3'

services:

my-service:

image: your-image-name

restart: unless-stopped

Command to run:

docker-compose up -d

Handling Exited Containers

**If a container has already exited, you can make it run continuously by applying the following steps:**

1. Restart and Apply Restart Policy

Commands:

docker start <container\_name\_or\_id>

docker update --restart unless-stopped <container\_name\_or\_id>

This ensures the container restarts automatically unless manually stopped.

2. Recreate with Restart Policy

If the container was created without a restart policy, remove and recreate it:

docker rm <container\_name\_or\_id>

docker run -d --restart unless-stopped your-image-name

3. Check Why the Container Exited

Use logs to diagnose the issue:

docker logs <container\_name\_or\_id>

Common reasons:

- The main process inside the container finished or crashed.

- Configuration errors or missing dependencies.

Pro Tip

Use long-running processes inside containers (e.g., web servers) or keep them alive with commands like:

tail -f /dev/null

Note: This is a workaround and not recommended for production.

**Connection of a 2 Tier Project**  
  
  
Notes Prepared From below youtube videos

* <https://www.youtube.com/watch?v=ITYYOTK0WE8>
* <https://youtu.be/GGaDSAMeopo?si=55AvDftMkvqYhLFd>
* <https://www.youtube.com/watch?v=QskegXV-90U>