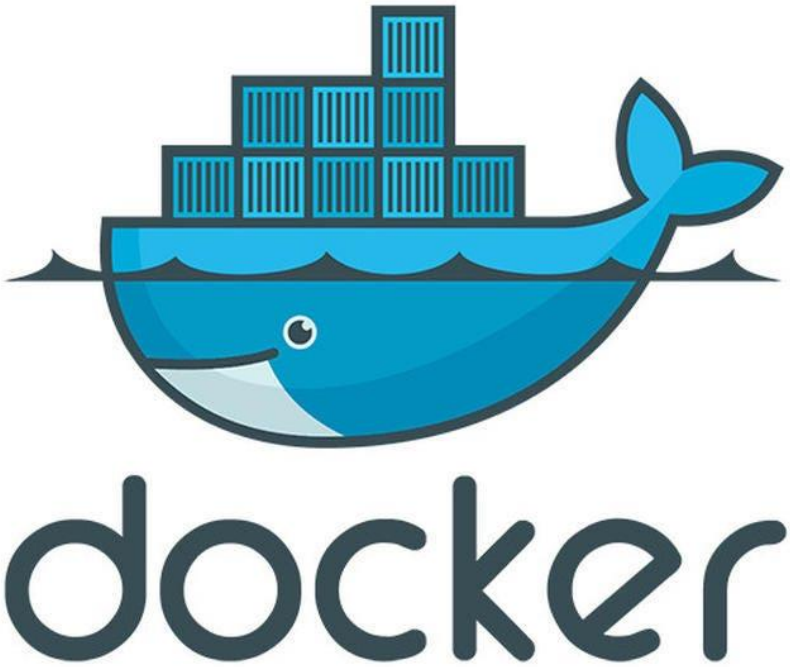


DOCKER FUNDAMENTALS

Why was docker introduced?

- The most common problem that we faced in IT industry was *"The code works on my machine, but it doesn't work on your machine".* The main reason for this problem was due to the difference in computer environments. Docker was aimed to solve this problem.

What is docker?

- Docker is a containerization tool designed to make it easier to create, deploy and run applications by using containers.
- Docker containers are lightweight alternatives to Virtual Machines and it uses the OS of the host. It does not have a Kernel of its own.
- There is no need to pre-allocate any RAM in containers.

Difference between Virtualization and Containerization?

Virtualization	Containerization
It has an OS of its own.	Uses the OS of the host machine.
Starts up slowly	Faster startup
Costly	Lightweight, and hence cost-effective.
Limited Portability between platforms.	Highly portable across different platforms.
Ex – VirtualBox, Hyper-V	Ex - Docker

Additionally, I found this article that further elaborates the difference between virtualization and containerization. Do give it a read.

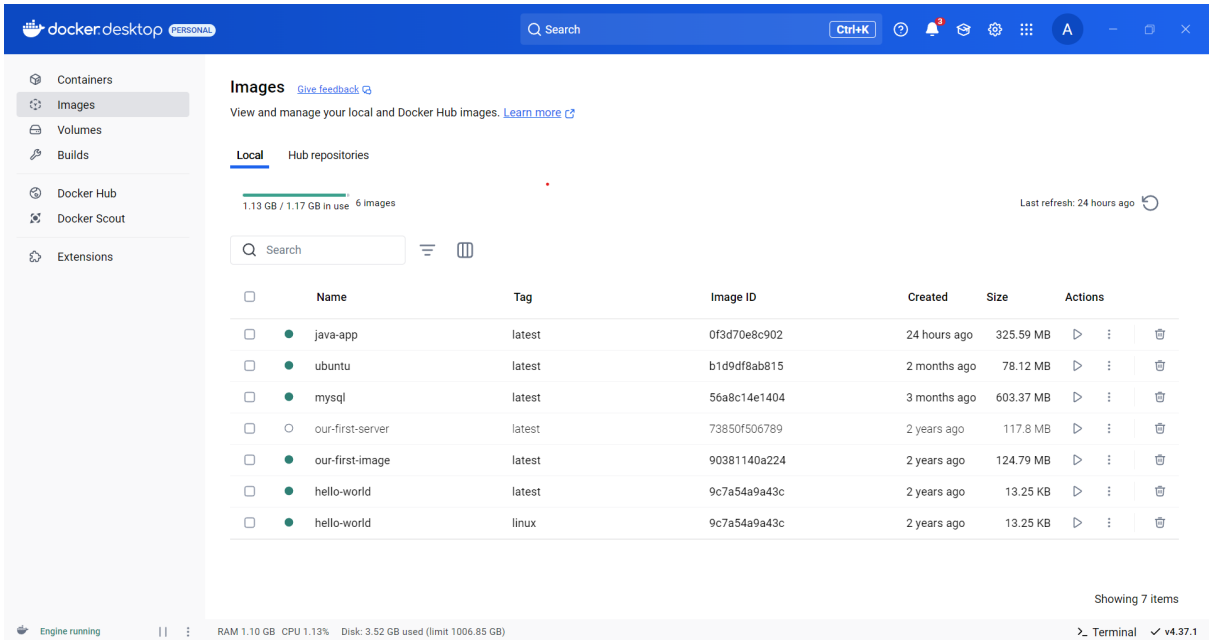
https://medium.com/@kritika.singhal/containerization-vs-virtualization-5aff7495b300

Installing Docker Desktop on your laptop

Step 1: Follow the installation guidelines on this page.

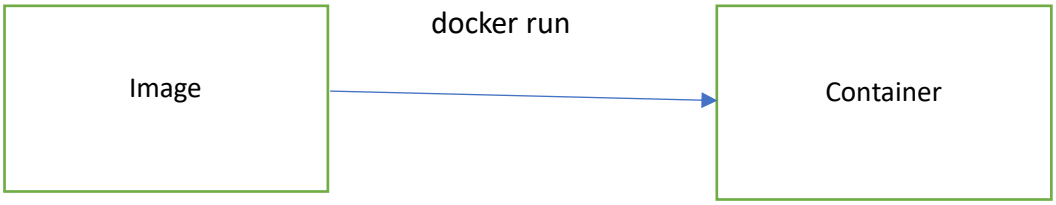
https://docs.docker.com/desktop/setup/install/windows-install/

Step 2: Below is the snapshot of docker desktop.



Before we start the practical, there are two concepts that we need to understand – **Containers** and **Images**.

- ***Images*** – Images are templates/blueprints for containers. It contains the code and required tools to execute the code. Image is basically a stopped container.
- ***Container*** – Container is a standardized unit of software that is used to run and execute the code.



There are two ways to create an image:

- Pull it from Dockerhub (*Dockerhub is a cloud-based repository that is used for finding and sharing the container images*).
- Create a dockerfile.

Creating a docker container

Step 1: Log in to the terminal that is inbuilt in Docker Desktop. You can find it in the bottom right-hand corner.

Step 2: Once you login to the terminal, run the command ***docker --version***

Step 3: Let's pull a nginx image from dockerhub.

```
PS C:\Users\lenovo> docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
fd674058ff8f: Pull complete
566e42bcee1c: Pull complete
2b99b9c5d9e5: Pull complete
bd98674871f5: Pull complete
1e109dd2a0d7: Pull complete
da8cc133ff82: Pull complete
c44f27309ea1: Pull complete
```

.

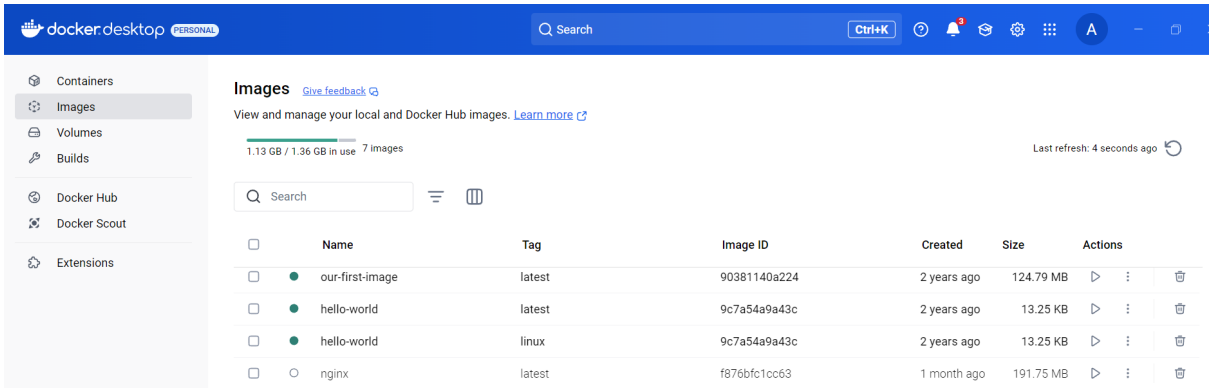
```
c44f27309ea1: Pull complete
Digest: sha256:42e917aaa1b5bb40dd0f6f7f4f857490ac7747d7ef73b391c774a41a8b994f15
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

As you can see, we are able to pull an image from Dockerhub.

Step 4: You can verify the downloaded image by using the command: ***docker image list***

```
PS C:\Users\lenovo> docker image list
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
java-app            latest    0f3d70e8c902   24 hours ago     326MB
nginx               latest    f876bfc1cc63   6 weeks ago      192MB
```

You should be able to see the nginx image. Alternatively, you can check the same in docker desktop.



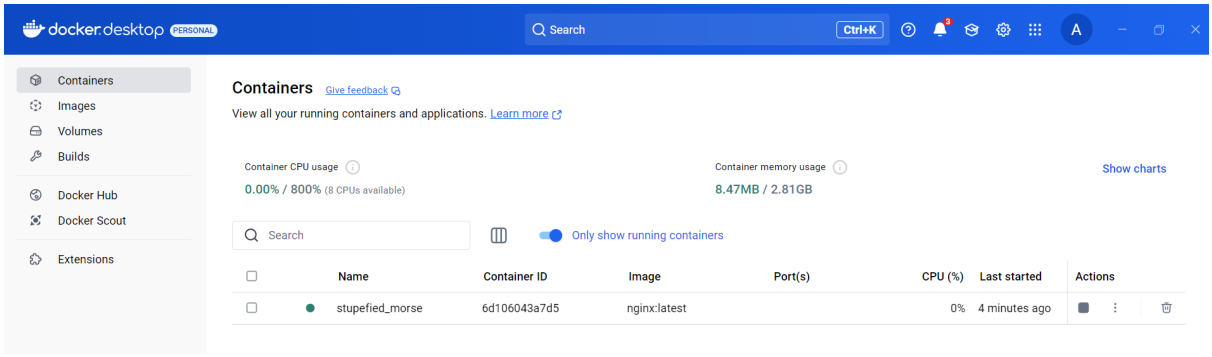
Step 5: Now, let's try to build container by using the command: ***docker run -d nginx:latest*** *(d means detached mode, so that the container processes runs in the background. We can start the container using the command****: docker run nginx:latest*** *as well****.*** *But as soon as we exit, the container would stop.)*

```
PS C:\Users\lenovo> docker run -d nginx:latest
6d106043a7d59da36883b2b81296551e8b27c83588285322ed385d3ac84ad317
```

Step 6: You can verify the running container by the command: **docker ps**

```
PS C:\Users\lenovo> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS     NAMES
6d106043a7d5   nginx:latest   "/docker-entrypoint.…"   7 seconds ago    Up 6 seconds    80/tcp    stupefied_morse
```

Alternatively, you can verify the same in docker desktop as well.



YAYYY!! Now, we have a container with nginx. We can use it to host websites, reverse proxy-server and so on. CONGRATULATIONS!!

Other docker commands:

- **docker run:** To start a new container and interact with it through command line.
  [Ex: docker run hello-world]
- **docker inspect <container_id>:** To view detailed information about a container or image.

```
ubuntu@ip-172-31-3-133:~$ docker inspect 23f320addcf0
[
    {
        "Id": "23f320addcf0fa9813250fe21e1d31e3eba56dad4605d820d6efc6c7d997434d",
        "Created": "2025-01-13T06:22:21.011017992",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
            "-g",
            "daemon off;"
        ],
        "State": {
```

- **docker port <container_id>:** To list the port mappings for a container.

```
ubuntu@ip-172-31-3-133:~$ docker port 23f320addcf0
80/tcp -> 0.0.0.0:80
80/tcp -> [::]:80
```

- **docker stats <container_id>:** To view resource usage statistics for one or more containers.

```
CONTAINER ID   NAME          CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O         PIDS
23f320addcf0   cranky_buck   0.00%     3.207MiB / 957.4MiB   0.33%     1.53kB / 0B   954kB / 12.3kB    2
```

- **docker top <container_id>::** To view the processes running inside the container.

```
ubuntu@ip-172-31-3-133:~$ docker top 23f320addcf0
UID                 PID                 PPID                C                   STIME               TTY                 TIME                CMD
root                3283                3262                0                   06:22               ?                   00:00:00            nginx: m
aster process nginx -g daemon off;
message+            3333                3283                0                   06:22               ?                   00:00:00            nginx: w
orker process
```

- **docker save:** To save an image to a tar archive.

```
ubuntu@ip-172-31-3-133:~$ docker save -o my-image.tar nginx
ubuntu@ip-172-31-3-133:~$ ls
git-tutorial  github  my-image.tar
```

- **docker load:** To load an image from a tar archive.

```
ubuntu@ip-172-31-3-133:~$ docker load -i my-image.tar
Loaded image: nginx:latest
```

- **docker system prune:** To remove all the stopped containers so as to reclaim space.

Before running the command:

```
ubuntu@ip-172-31-3-133:~/github/todo-list-app$ docker ps -a
CONTAINER ID   IMAGE               COMMAND                  CREATED         STATUS                     PORTS                                       NAMES
345ea6e8fefe   todo-app            "docker-entrypoint.s…"   4 seconds ago   Up 2 seconds               0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   peaceful_leakey
4933bdb02f6e   9789e3c719aa        "/bin/sh -c 'dotnet …"   16 hours ago    Exited (145) 16 hours ago                                              hardcore_euclid
a0b51df3d048   ea7f98786413        "/bin/sh -c 'dotnet …"   16 hours ago    Exited (145) 16 hours ago                                              sweet_thompson
19d8af96f12e   c97d2441e3cf        "/bin/sh -c 'dotnet …"   16 hours ago    Exited (145) 16 hours ago                                              priceless_mcclintock
2957d3c5f25c   53befc0a6267        "/bin/sh -c 'dotnet …"   16 hours ago    Exited (145) 16 hours ago                                              nice_nash
ad7c5156288f   5beccbd013b9        "/bin/sh -c 'dotnet …"   16 hours ago    Exited (145) 16 hours ago                                              vigorous_pasteur
0a11f3c74c0e   todo-app:latest     "docker-entrypoint.s…"   16 hours ago    Exited (0) 16 hours ago                                                vigilant_germain
34f47c633bcc   0744a1d4a8df        "/bin/sh -c 'npm ins…"   16 hours ago    Exited (1) 16 hours ago                                                flamboyant_bartik
782a8f6705e3   counter-app:latest  "python cool_counter…"   18 hours ago    Exited (0) 16 hours ago                                                thirsty_snyder
e2ef39b06bec   java-app            "java Main"              19 hours ago    Exited (0) 19 hours ago                                                flamboyant_hertz
23f320addcf0   nginx:latest        "/docker-entrypoint.…"   12 days ago     Exited (0) 11 days ago                                                 cranky_buck
2427b2311231   mysql:latest        "docker-entrypoint.s…"   2 weeks ago     Exited (0) 2 weeks ago                                                 youthful_stonebraker
642b4baa2ca2   mysql:latest        "docker-entrypoint.s…"   2 weeks ago     Exited (1) 2 weeks ago                                                 zealous_ardinghelli
5905ad467576   mysql:latest        "docker-entrypoint.s…"   2 weeks ago     Exited (1) 2 weeks ago                                                 serene_chatterjee
```

After running the command:

```
ubuntu@ip-172-31-3-133:~/github/todo-list-app$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                                       NAMES
345ea6e8fefe   todo-app   "docker-entrypoint.s…"   39 minutes ago   Up 39 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   peaceful_leakey
```

- **docker rmi <image id>:** To delete the images.

```
ubuntu@ip-172-31-3-133:~/github/todo-list-app$ docker rmi b9bd094e43e8 9b8fdbc4debd f876bfc1cc63 56a8c14e1404 16d93ae3411b 1d12470fa662 29de1b9e96c0 264c9bdce361
```

- **docker volume ls:** To list the volumes in your server.

```
ubuntu@ip-172-31-3-133:~$ docker volume ls
DRIVER    VOLUME NAME
local     203ca1c6343219d81b95246cff8a030a812adc84ad1b2e26d1b4f03627c08a4b
local     378bca9e770aa7005bce6b82374cd114d71af33865fafb5cb5d7253b5ac65d69
local     bf883fc8b43bddb6800c7cd78482bbb3b03d3632bccadbc5590b502e857ef582
local     d5f03de0b3870d9578be6968d2aee032719945d7862c7049cd4463e8519013c6
local     d091bf6b7006844851e153a16517483b6f7e5199ec5bd115040f1053471a11b5
local     deb6615688ea87ab88cc1224d727be87036e1e1b41f68e41421453a3a502a401
local     mysql-data
```

- **docker volume inspect <volume_id>:** To list the details about the volume.

```
ubuntu@ip-172-31-3-133:~$ docker volume inspect mysql-data
[
    {
        "CreatedAt": "2025-01-30T06:32:26Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/mysql-data/_data",
        "Name": "mysql-data",
        "Options": null,
        "Scope": "local"
    }
]
```

- **docker volume create <volume_name>:** To create a volume with the desired name.

```
ubuntu@ip-172-31-3-133:~$ docker volume create mysql-data
mysql-data
```

Now, let's understand an important aspect of Docker and that is, ***Dockerfile.***

What is Dockerfile?

- To put it simply, **Dockerfile** is a text-based document that's used to create a container image. It is like a set of instructions for making a container. It tells Docker what base image to use, what commands to run, and what files to include.
  For example, if you are making a container for a website, the Dockerfile might tell Docker to use an official web server image, copy the files for your website into the container, and start the web server when the container starts.

Demo Time:

- **Step 1:** Login to your AWS EC2 Instance.
- **Step 2:** Clone in the below repository from github using the below command:
  git clone https://github.com/dockersamples/todo-list-app.git
- **Step 3:** You should now have a folder with the name **"todo-list-app"**

```
drwxrwxr-x 5 ubuntu ubuntu 4096 Jan 24 14:16 todo-list-app
```

- **Step 4:** Create a dockerfile using the command **: vim Dockerfile**
- **Step 5:** Paste in the below contents:

```
#Base Image
FROM node:14

#Working Directory
WORKDIR /app

#Copy Code
COPY . .

#Install NodeJS
RUN npm install

#Command to run the application
CMD ["node", "src/index.js", "0.0.0.0:3000"]
~
```

You can learn more about these commands at:
https://docs.docker.com/reference/dockerfile/

DIFFERENCE BETWEEN CMD AND ENTRYPOINT:

- **CMD :** Sets default parameters that **can be overridden** from the Docker Command Line Interface (CLI) when a container is running.
- **ENTRYPOINT:** Default parameters that **cannot be overridden** when Docker Containers run with CLI parameters.

➢ **Step 6: Build the image using the below command:**
docker build -t todo-app .

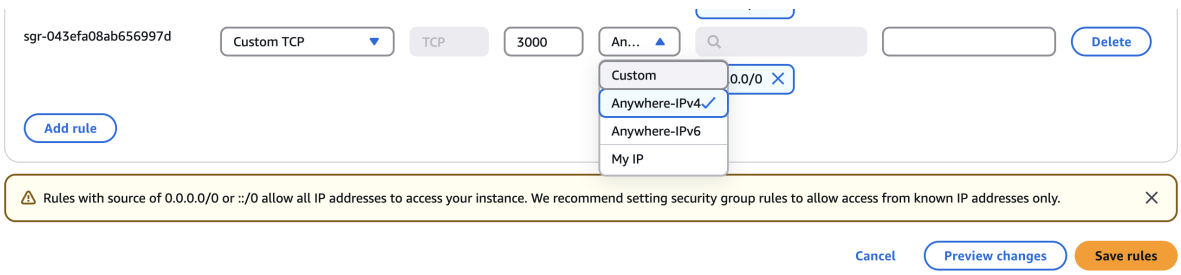
```
[ubuntu@ip-172-31-3-133:~/github/todo-list-app$ docker build -t todo-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  6.474MB
Step 1/5 : FROM node:14
 ---> 1d12470fa662
Step 2/5 : WORKDIR /app
 ---> Using cache
 ---> b47938c76858
Step 3/5 : COPY . .
 ---> Using cache
 ---> d2684968111c
Step 4/5 : RUN npm install
 ---> Using cache
 ---> 0b11c7e48c4f
Step 5/5 : CMD ["node", "src/index.js", "0.0.0.0:3000"]
 ---> Using cache
 ---> 9993e4d4f497
Successfully built 9993e4d4f497
Successfully tagged todo-app:latest
```

➢ **Step 7: Run the container using the below command:**
docker run -d -p 3000:3000 todo-app

```
[ubuntu@ip-172-31-3-133:~/github/todo-list-app$ docker run -d -p 3000:3000 todo-app
345ea6e8fefe2d76c4237bffd214374a12e2e7a93dd72a7667ac2a32613309cb
[ubuntu@ip-172-31-3-133:~/github/todo-list-app$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                                       NAMES
345ea6e8fefe   todo-app       "docker-entrypoint.s…"   4 seconds ago   Up 2 seconds   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   peaceful_leakey
```

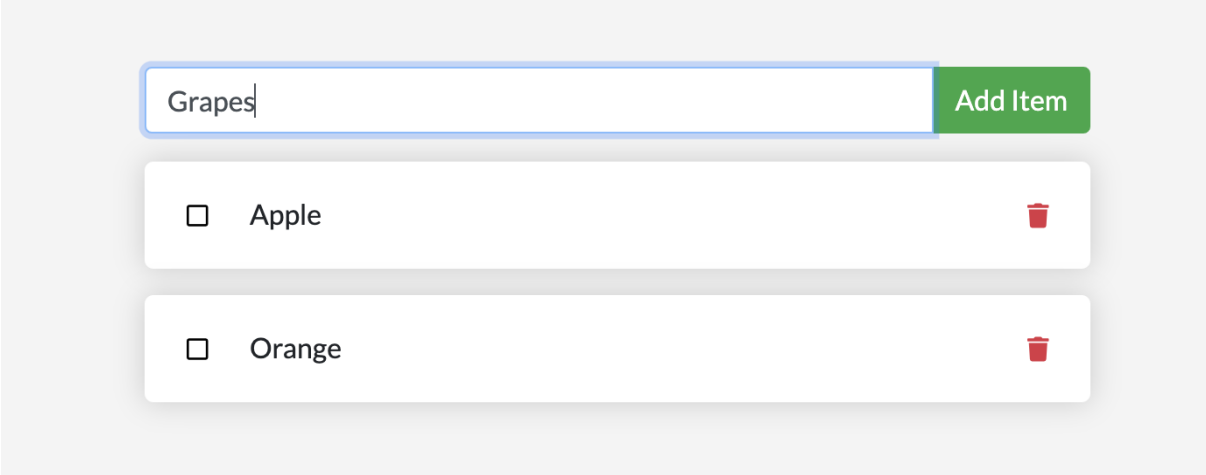
➢ **Step 8:** Since this application runs on port 3000, you need to add this port to the inbound rules of your security group.



Then click on **Save rules.**

- **Step 9:** To verify if the application is running fine, copy the public IP of your AWS EC2 Instance, and paste it on any browser using the below command: http://<your_ec2_public_ip>:3000

- **Step 10: Now, you have a running application that is hosted on a docker container. Amazing right?**



Docker Volume:

Let's try to understand this concept with an example.

Step 1: Create a container of mysql by running the below command:

docker run -d -e MYSQL_ROOT_PASSWORD=root mysql:latest

Note: You need to specify an environmental variable, otherwise the container with exit with the below error

```
ubuntu@ip-172-31-3-133:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS                     PORTS     NAMES
f4de18490a47   mysql:latest   "docker-entrypoint.s…"   7 seconds ago   Exited (1) 6 seconds ago             eloquent_diffie
345ea6e8fefe   todo-app       "docker-entrypoint.s…"   30 hours ago    Exited (0) 29 hours ago              peaceful_leakey
ubuntu@ip-172-31-3-133:~$ docker logs f4de18490a47
2025-01-26 12:49:26+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 9.2.0-1.el9 started.
2025-01-26 12:49:27+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2025-01-26 12:49:27+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 9.2.0-1.el9 started.
2025-01-26 12:49:27+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified
    You need to specify one of the following as an environment variable:
    - MYSQL_ROOT_PASSWORD
    - MYSQL_ALLOW_EMPTY_PASSWORD
    - MYSQL_RANDOM_ROOT_PASSWORD
```

Step 2: To enter inside the docker container, run the below command:

docker exec -it <container_id> bash

Step 3: To login into mysql, enter the below command.

mysql -u root -p

You'll get a prompt to enter the password. You need to enter *root* here.

```
bash-5.1# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 9.2.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Step 4: Create a database and a table by using the below commands.

To get the list of databases in mysql:

show databases;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)
```

To create a database:

create database kyc_database;

```
mysql> create database kyc_database;
Query OK, 1 row affected (0.04 sec)
```

To use the created database:

use kyc_database;

```
mysql> use kyc_database;
Database changed
```

To create a table inside the database:

CREATE TABLE messages (

id INT AUTO_INCREMENT PRIMARY KEY,

message TEXT

);

```
mysql> CREATE TABLE messages (
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> message TEXT
    -> );
Query OK, 0 rows affected (0.05 sec)
```

To insert data in the table:

insert into messages (message) values ("kyc stored"); -- Run this query multiple times to create entry inside the table.

```
mysql> insert into messages (message) values ("kyc stored");
Query OK, 1 row affected (0.02 sec)

mysql> insert into messages (message) values ("kyc stored");
Query OK, 1 row affected (0.01 sec)
```

To list the entries of the table:

select * from messages;

```
mysql> select * from messages;
+----+------------+
| id | message    |
+----+------------+
|  1 | kyc stored |
|  2 | kyc stored |
+----+------------+
2 rows in set (0.00 sec)
```

Step 5: Now, let's stop and remove the container by running the below command.

docker stop <container_id> && docker rm <container_id>

Step 6: Now let's launch a container again by running the command.

docker run -d -e MYSQL_ROOT_PASSWORD=root mysql:latest

Step 7: Now, let try to see if the database we created exists or not.

```
ubuntu@ip-172-31-3-133:~$ docker exec -it b303f0567f09 bash
bash-5.1# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.2.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)
```

OH SNAP!! 😟 Our database is gone. What do we do now? This can be prevented by a concept known as Docker Volume. Below is the definition:

- A Docker volume is a persistent data store for containers, created and managed by Docker. Volumes are used to store data outside the container's writable layer, ensuring that the data is not lost when the container is removed or recreated. This makes volumes a preferred mechanism for persisting data generated by and used by Docker containers. ***Volumes are always mounted on the Host machine.***

Step 8: Now, let's create a backup directory in our Host OS.

mkdir mysql-backup

```
ubuntu@ip-172-31-3-133:~$ mkdir mysql-backup
ubuntu@ip-172-31-3-133:~$ ls
 git-tutorial  github  my-image.tar  mysql-backup
ubuntu@ip-172-31-3-133:~$ cd mysql-backup
ubuntu@ip-172-31-3-133:~/mysql-backup$ pwd
 /home/ubuntu/mysql-backup
```

Step 9: Remove the previously created container and re-launch the container, but this time with a different command.

docker run -d -v /home/ubuntu/mysql-backup:/var/lib/mysql --name mysql -e MYSQL_ROOT_PASSWORD=root mysql:latest

Note: The part before the colon (:) is the path of your host OS and the part after the colon is the path of your docker container. Also, /var/lib/mysql is the default path for mysql.

Step 10: Repeat the above steps of creating database and table.

```
mysql> create database kyc_database;
Query OK, 1 row affected (0.00 sec)

mysql> use kyc_database;
Database changed
mysql> CREATE TABLE messages (
    -> ID INT AUTO_INCREMENT PRIMARY KEY,
    -> message TEXT
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> select * from messages;
Empty set (0.01 sec)

mysql> insert into messages (message) values ("kyc stored");
Query OK, 1 row affected (0.01 sec)

mysql> insert into messages (message) values ("kyc stored");
Query OK, 1 row affected (0.01 sec)

mysql> insert into messages (message) values ("kyc stored");
Query OK, 1 row affected (0.01 sec)

mysql> select * from messages;
+----+------------+
| ID | message    |
+----+------------+
|  1 | kyc stored |
|  2 | kyc stored |
|  3 | kyc stored |
+----+------------+
3 rows in set (0.00 sec)
```

Step 11: Let's remove the container now and verify the contents in the path: /home/ubuntu/mysql-backup in your Host OS.

BOOM!! Now, you should be able to see some contents in this path including the kyc_database that you have created.

Step 12: Now, let's remove the container again and re-launch it with the below command:

docker run -d -v /home/ubuntu/mysql-backup:/var/lib/mysql --name mysql -e MYSQL_ROOT_PASSWORD=root mysql:latest

```
ubuntu@ip-172-31-3-133:~/mysql-backup$ docker stop 27d0c4e42a6c && docker rm 27d0c4e42a6c
27d0c4e42a6c
27d0c4e42a6c
ubuntu@ip-172-31-3-133:~/mysql-backup$ cd
ubuntu@ip-172-31-3-133:~$ docker run -d -v /home/ubuntu/mysql-backup:/var/lib/mysql --name mysql -e MYSQL_ROOT_PASSWORD=root mysql:latest
e8fb746c899d16ca8df780574c4cebec439eb7e03ba45847a8fdac26eb8803fc
ubuntu@ip-172-31-3-133:~$ docker exec -it e8f bash
bash-5.1# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.2.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| kyc_database       |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.02 sec)

mysql>
```

Now, you should be able to see the kyc_database as well. Everything in intact!!

```
mysql> use kyc_database;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from essages;
ERROR 1146 (42S02): Table 'kyc_database.essages' doesn't exist
mysql> select * from messages;
+----+-------------+
| ID | message     |
+----+-------------+
|  1 | kyc stored  |
|  2 | kyc stored  |
|  3 | kyc stored  |
+----+-------------+
3 rows in set (0.00 sec)
```

Friends, that is the concept of Docker Volumes. Do give it a thumbs up, if you have liked it.

Docker Networking:

Docker Networking refers to the ability for containers to connect to and communicate with each other, or to non-Docker workloads.

The following network drivers are available by default, and provide core networking functionality:

- *Bridge* – The default network driver.
- *User-defined bridge* – One that we can create externally to connect the dockers.
- *Host* – Remove network isolation between the container and docker host.
- *Overlay* – Overlay network connect multiple Docker daemons together.
- *MacVLan* – Assign a MAC Address to a container.
- *IPVLan* – Provide full control over both IPv4 and IPv6 addressing.
- *None* – Completely isolate a container from the host and other containers.

Demo Time:

Step 1: Clone the repository using the below command:

git clone https://github.com/LondheShubham153/two-tier-flask-app.git

Step 2: Go through the Dockerfile in this repository and create an image named “flaskapp” by running the below command:

docker build -t flaskapp .

Step 3: Create mysql container by running the below command:

docker run -d --name mysql -e MYSQL_DATABASE=mydb -e MYSQL_ROOT_PASSWORD=admin -v mysql-data:/var/lib/mysql -p 3306:3306 mysql:5.7

Your mysql container should be up and running.

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
46ce40d556b3   mysql:5.7   "docker-entrypoint.s…"   12 seconds ago   Up 10 seconds   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql
```

Step 4: Now, let’s create flaskapp container from the image that we created previously. However, we need to pass certain MYSQL environment variables. The instruction are specified in the README.md file located in : https://github.com/LondheShubham153/two-tier-flask-app/blob/master/README.md

docker run -d --name flaskapp -e MYSQL_HOST=mysql -e MYSQL_USER=root -e MYSQL_PASSWORD=admin -e MYSQL_DB=mysql -p 5000:5000 flaskapp:latest

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker run -d --name flaskapp -e MYSQL_HOST=mysql -e MYSQL_USER=root -e MYSQL_PASSWORD=admin -e MYSQL_DB=mysql -p 5000:5000 flaskapp:latest
68ea974d40ae679476ld4bbd8895ae93f498171bbe0208dfddcd0cf44dd5f424
```

Step 5: However, the container is in Exited mode.

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker ps -a
CONTAINER ID   IMAGE            COMMAND                  CREATED          STATUS                      PORTS                                                  NAMES
68ea974d40ae   flaskapp:latest  "python app.py"          8 seconds ago    Exited (1) 6 seconds ago                                                           flaskapp
46ce40d556b3   mysql:5.7        "docker-entrypoint.s…"   20 minutes ago   Up 20 minutes               0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql
```

You can now check the logs for the container to see what went wrong, by running the command : **docker logs <container_id>**

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker logs 68ea974d40ae
Traceback (most recent call last):
  File "/app/app.py", line 46, in <module>
    init_db()
  File "/app/app.py", line 18, in init_db
    cur = mysql.connection.cursor()
  File "/usr/local/lib/python3.9/site-packages/flask_mysqldb/__init__.py", line 94, in connection
    ctx.mysql_db = self.connect
  File "/usr/local/lib/python3.9/site-packages/flask_mysqldb/__init__.py", line 81, in connect
    return MySQLdb.connect(**kwargs)
  File "/usr/local/lib/python3.9/site-packages/MySQLdb/__init__.py", line 121, in Connect
    return Connection(*args, **kwargs)
  File "/usr/local/lib/python3.9/site-packages/MySQLdb/connections.py", line 200, in __init__
    super().__init__(*args, **kwargs2)
MySQLdb.OperationalError: (2005, "Unknown server host 'mysql' (-2)")
```

If you see, the container has not been able to establish a connection with the MySQL container that we have created previously. Now, let's try to analyse the reason by inspecting the below two containers by using the command: ***docker inspect <container_id>***


```
    },
    "NetworkSettings": {
        "Bridge": "",
        "SandboxID": "4d2890e73ef9f7857f9029f92221d67b1b661c4536b18abf812c1fe900b58ad2",
        "SandboxKey": "/var/run/docker/netns/4d2890e73ef9",
        "Ports": {},
        "HairpinMode": false,
        "LinkLocalIPv6Address": "",
        "LinkLocalIPv6PrefixLen": 0,
        "SecondaryIPAddresses": null,
        "SecondaryIPv6Addresses": null,
        "EndpointID": "",
        "Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "IPAddress": "",
        "IPPrefixLen": 0,
        "IPv6Gateway": "",
        "MacAddress": "",
        "Networks": {
            "bridge": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": null,
                "MacAddress": "",
                "NetworkID": "02547f761ad849d50a299849c179aa56965b7a4fb1fabb30feec3b4da98dbbfc",
                "EndpointID": "",
                "Gateway": "",
                "IPAddress": "",
                "IPPrefixLen": 0,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "DriverOpts": null,
                "DNSNames": null
            }
```


The above snapshot is for the flaskapp container. If you see nothing is configured for this container. There are no endpoints or other parameters. Now, it is understandable that the above 2 containers have a bridge of its own. To solve this issue, we need to create common bridge so as to connect the two containers.

Step 6: Create a network by running the command.

docker network create -d bridge <name_of_network>

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker network create -d bridge twotier
c0e053eedf52180acf869cd02335fbc06899a188539864472f57585d21379038
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
02547f761ad8   bridge    bridge    local
08178b2010cf   host      host      local
5c9c095976d4   none      null      local
c0e053eedf52   twotier   bridge    local
```

Step 7: Inspect the network that we created

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker inspect c0e053eedf52
[
    {
        "Name": "twotier",
        "Id": "c0e053eedf52180acf869cd02335fbc06899a188539864472f57585d21379038",
        "Created": "2025-01-30T07:26:10.124540942Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

If you see, there are no containers.

Step 8: Remove the previously created containers by running the command:

docker stop <container_id> && docker rm <container_id>

Step 9: Now, let's relaunch the container but this time we need to pass a flag **--network=twotier** for both the containers. The command is:

docker run -d --name mysql -e MYSQL_DATABASE=mydb -e MYSQL_ROOT_PASSWORD=admin -v mysql-data:/var/lib/mysql -p 3306:3306 --network=twotier mysql:5.7

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker run -d --name mysql -e MYSQL_DATABASE=mydb -e MYSQL_ROOT_PASSWORD=admin -v mysql-data:/var/lib/mysql -p 3306:3306 --network=twotier
mysql:5.7
```

If you inspect the twotier network now, you should be able to see one container.

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker inspect c0e053eedf52
[
    {
        "Name": "twotier",
        "Id": "c0e053eedf52180acf869cd02335fbc06899a188539864472f57585d21379038",
        "Created": "2025-01-30T07:26:10.124540942Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "c5b9eaa3e07d2b50d64052faf3686356884b0d4eab86cb390fbb7648339989de": {
                "Name": "mysql",
                "EndpointID": "8af756097a41a3920f287d65058ccf73f53f94e4ff65715eb6aae7cf0d0e52a9",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

Step 10: Do the same for the flaskapp container and inspect the network. The command is:

docker run -d --name flaskapp -e MYSQL_HOST=mysql -e MYSQL_USER=root -e MYSQL_PASSWORD=admin -e MYSQL_DB=mysql -p 5000:5000 –network=twotier flaskapp:latest

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker run -d --name flaskapp -e MYSQL_HOST=mysql -e MYSQL_USER=root -e MYSQL_PASSWORD=admin -e MYSQL_DB=mysql -p 5000:5000 --network=twoti
er flaskapp:latest
c7e733a82e865b847fd85143cc7039e514dd08b8d017a093565de8bb56e9462d
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker inspect c0e053eedf52
[
    {
        "Name": "twotier",
        "Id": "c0e053eedf52180acf869cd02335fbc06899a188539864472f57585d21379038",
        "Created": "2025-01-30T07:26:10.124540942Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "c5b9eaa3e07d2b50d64052faf3686356884b0d4eab86cb390fbb7648339989de": {
                "Name": "mysql",
                "EndpointID": "8af756097a41a3920f287d65058ccf73f53f94e4ff65715eb6aae7cf0d0e52a9",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            },
            "c7e733a82e865b847fd85143cc7039e514dd08b8d017a093565de8bb56e9462d": {
                "Name": "flaskapp",
                "EndpointID": "b7ddcee4eb05b93f3f7280cff4d9c583372f53c486b1a977ba33ecbc92122712",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

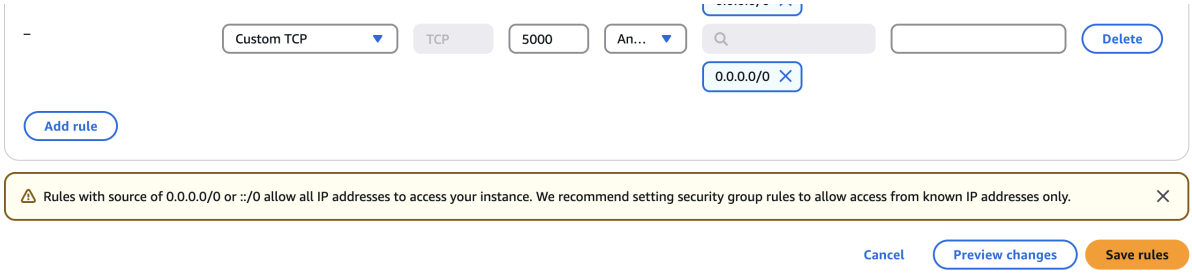
If you notice, there are two containers in the “Containers” tag.

Step 11: Now, that two containers are running.

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
c7e733a82e86   flaskapp:latest   "python app.py"          49 seconds ago   Up 48 seconds   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp              flaskapp
c5b9eaa3e07d   mysql:5.7         "docker-entrypoint.s…"   2 minutes ago    Up 2 minutes    0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql
```

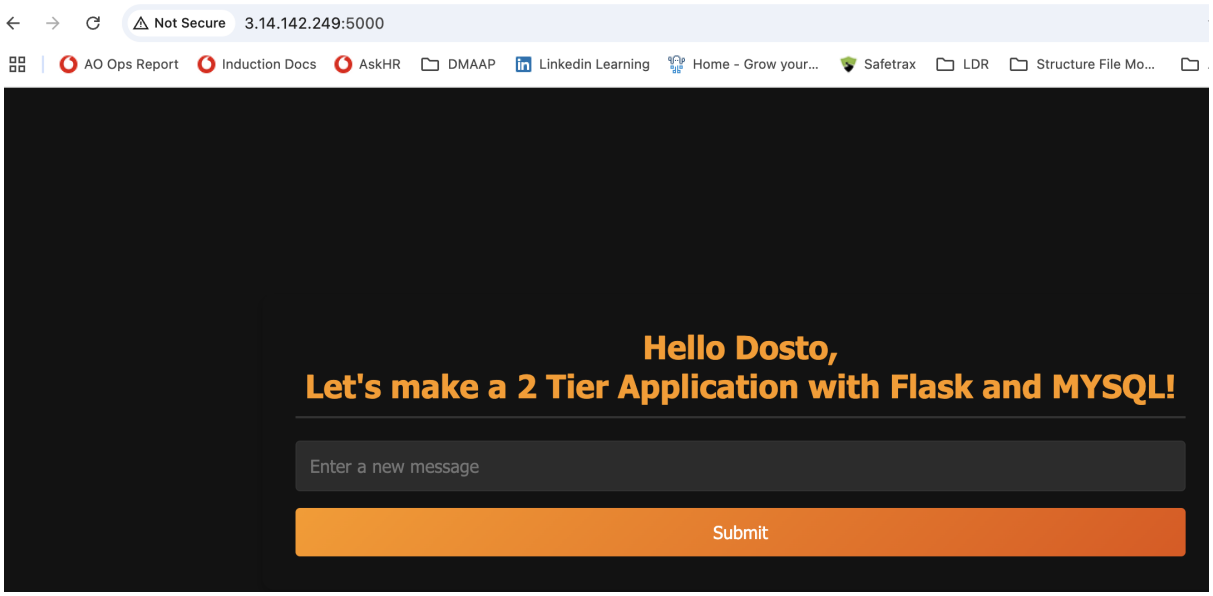
Reason: Because, both these containers are in the same network.

Step 12: Now, let’s try to see if our application is working. For that, we need to add port 5000 to the Inbound rules of our security group of the EC2 instance.



Click on “Save rules” and launch the container in your browser by running :
http://<your_ec2_public_address>:5000/

BOOOOOOOMMM!! You, should be able to see the application running.



Type some messages and verify the same in your mysql container. The messages would be stored in the tables.

Docker Compose:

Docker compose is a utility that allows you to create and manage multiple docker containers at once.

Demo Time:

Step 1: Create a file with the name docker-compose.yml

Now, what is the advantage of docker-compose?

- Previously, we had to run the command docker run and docker build command multiple times after we made changes to Dockerfile or after doing any other changes.
- Using docker compose, we kind of automate the docker start and stop process.

Step 2: Paste in the below contents to this file.

```
version: "3.8"

services:
  mysql:
    image: mysql:5.7
    container_name: mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: admin
      MYSQL_DATABASE: mydb
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - twotier
    ports:
      - 3306:3306
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-uroot", "-padmin"]
      interval: 10s
      timeout: 20s
      retries: 10
      start_period: 60s

  flask-app:
    build:
      context: .
    container_name: flask-app
    restart: always
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: admin
      MYSQL_DB: mydb
    networks:
      - twotier
    ports:
      - 5000:5000
    depends_on:
      - mysql
    healthcheck:
      test: ["CMD-SHELL", "curl -f http://localhost:5000/"]
      interval: 10s
      timeout: 20s
      retries: 10
      start_period: 60s

volumes:
  mysql-data:
networks:
  twotier:
```

Step 3: Install docker compose in your EC2 Instance by running the below command.

apt-get install docker-compose-v2

Step 4: Validate the contents of **docker-compose.yml** by running the below command:

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker compose config
WARN[0000] /home/ubuntu/github/two-tier-flask-app/docker-compose.yml: `version` is obsolete
name: two-tier-flask-app
services:
  flask-app:
    build:
      context: /home/ubuntu/github/two-tier-flask-app
      dockerfile: Dockerfile
    container_name: flask-app
    depends_on:
      mysql:
        condition: service_started
```

If you get any errors, resolve it accordingly.

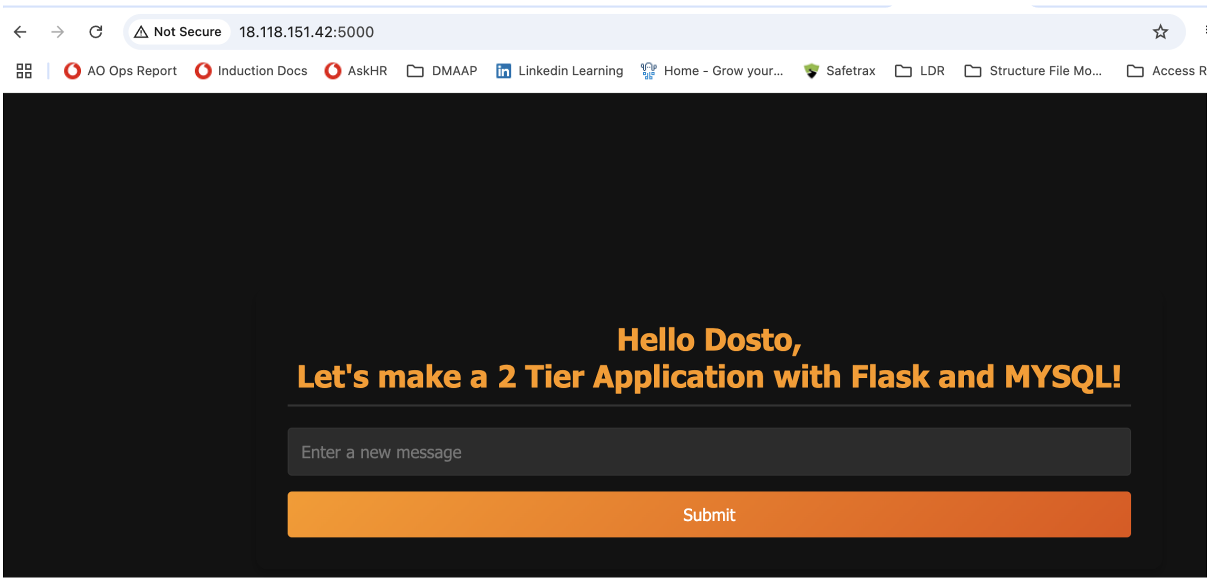
Step 5: Start docker compose by running the command:

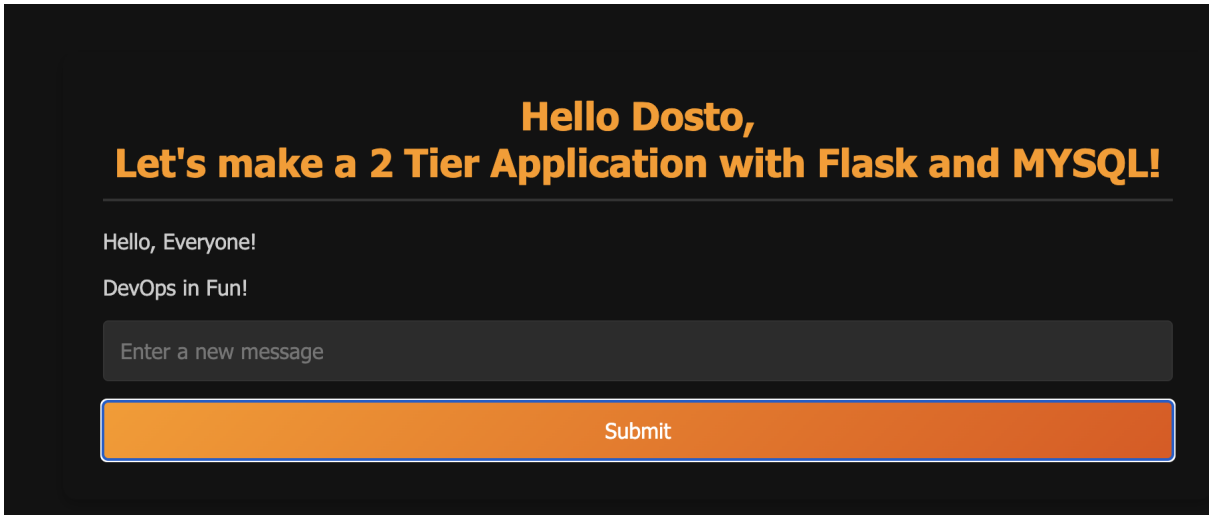
docker compose up -d (To run the app in detached (background) mode).

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker compose up
WARN[0000] /home/ubuntu/github/two-tier-flask-app/docker-compose.yml: `version` is obsolete
[+] Running 2/2
 ✔ Container mysql      Created                                                    0.1s
 ✔ Container flask-app  Created                                                    0.1s
Attaching to flask-app, mysql
mysql      | 2025-01-31 15:23:45+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.44-1.el7 started.
mysql      | 2025-01-31 15:23:46+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
mysql      | 2025-01-31 15:23:46+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.44-1.el7 started.
mysql      | '/var/lib/mysql/mysql.sock' -> '/var/run/mysqld/mysqld.sock'
mysql      | 2025-01-31T15:23:46.850623Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp server option (see documentation for mor
e details).
mysql      | 2025-01-31T15:23:46.852977Z 0 [Note] mysqld (mysqld 5.7.44) starting as process 1 ...
mysql      | 2025-01-31T15:23:46.860593Z 0 [Note] InnoDB: PUNCH HOLE support available
mysql      | 2025-01-31T15:23:46.860997Z 0 [Note] InnoDB: Mutexes and rw_locks use GCC atomic builtins
mysql      | 2025-01-31T15:23:46.861035Z 0 [Note] InnoDB: Uses event mutexes
mysql      | 2025-01-31T15:23:46.861050Z 0 [Note] InnoDB: GCC builtin __atomic_thread_fence() is used for memory barrier
mysql      | 2025-01-31T15:23:46.861060Z 0 [Note] InnoDB: Compressed tables use zlib 1.2.13
mysql      | 2025-01-31T15:23:46.861071Z 0 [Note] InnoDB: Using Linux native AIO
```

Step 6: Now check, if the application is up by running the command:

http://<your_public_ec2_ip>:5000/






```
bash-4.2# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 30
Server version: 5.7.44 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydb               |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> use mydb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from messages;
+----+------------------+
| id | message          |
+----+------------------+
|  1 | Hello, Everyone! |
|  2 | DevOps in Fun!   |
+----+------------------+
2 rows in set (0.00 sec)
```

Multi Stage Build in Docker:

- Multi-stage builds are useful to anyone who has struggled to optimize Dockerfiles while keeping them easy to read and maintain.
- With multi-stage builds, you use multiple FROM statements in your Dockerfile. Each FROM instruction can use a different base image, and each of them begins a new stage of build. You can selectively copy artifacts from one stage to another, leaving behind everything you don't want in the final image.
- The end result is a tiny production image with nothing but the binary inside. None of the build tools required to build the application are included in the resulting image.

NOW, LET'S TRY TO UNDERSTAND THIS WITH A DEMO!

Step 1: Firstly, let create a Dockerfile with the image python:3.9 and check the size of the image. Below are the contents of the Dockerfile.

```
# Use an official Python runtime as the base image
FROM python:3.9

# Set the working directory in the container
WORKDIR /app

# install required packages for system
RUN apt-get update \
    && apt-get upgrade -y \
    && apt-get install -y gcc default-libmysqlclient-dev pkg-config \
    && rm -rf /var/lib/apt/lists/*

# Copy the requirements file into the container
COPY requirements.txt .

# Install app dependencies
RUN pip install mysqlclient
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code
COPY . .

# Specify the command to run your application
CMD ["python", "app.py"]
```

Step 2: We will now create an image from the Dockerfile and check the size. The command is: **docker build -t flask-app-full .**

```
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
flask-app-full  latest    6c5ca4e81fb9   11 minutes ago   1.07GB
```

As can be seen, the size of the image is 1.07 GB, which is quite big.

Step 3: To optimize the size of the image, we will now create a multi-stage docker file. Below is an example for the same.

```
# ---------------- STAGE 1: BUILD STAGE  ---------------- #
# Use an official Python runtime as the base image
FROM python:3.9 AS builder

# Set the working directory in the container
WORKDIR /app

# install required packages for system
RUN apt-get update \
    && apt-get upgrade -y \
    && apt-get install -y gcc default-libmysqlclient-dev pkg-config \
    && rm -rf /var/lib/apt/lists/*

# Copy the requirements file into the container
COPY requirements.txt .

# Install app dependencies
RUN pip install --no-cache-dir -r requirements.txt

# ---------------- STAGE 2: FINAL STAGE  ---------------- #
# Using a smaller image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

#Install runtime dependencies only
RUN apt-get update && \
    apt-get install -y --no-install-recommends libmariadb3 && \
    rm -rf /var/lib/apt/lists/*

#Copy dependencies and application code from the builder stage
COPY --from=builder /usr/local/lib/python3.9/site-packages/ /usr/local/lib/python3.9/site-packages/
COPY . .

# Specify the command to run your application
CMD ["python", "app.py"]
```

We name this Dockerfile as - Dockerfile-multi-stage. Now, let us understand the concepts.

- As we see, there are **two sections** in this file. In the first section, the FROM instruction uses a bigger base image and is used to install the libraries and app dependencies (Building the binary)
- In the second section, the FROM instruction uses a smaller base image and installs the runtime dependencies. The second FROM starts a new build stage with a comparatively smaller python image as its base.
- The COPY command just copies the build artifact (caches) from the previous stage into this new stage.

Step 4: Now, let's build an image with this newly created multi-stage dockerfile. The command to create an image with a file named other than Dockerfile is:

docker build -t flask-app-multi-stage -f dockerfile-multi-stage .

Step 5: Verify the size of this image.

```
ubuntu@ip-172-31-3-133:~/github/two-tier-flask-app$ docker images
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
flask-app-multi-stage   latest    fbc35397f707   6 minutes ago    148MB
```

If you see, this image is smaller in size compared to the previously created image. This shows, how we can optimize the size of the image.

To conclude, below are some of the advantages of using a multi-stage build:

- Optimizes the overall size of the Docker image.
- Removes the burden of creating multiple Dockerfiles for different stages.
- Easy to debug a particular build stage.
- Able to use the previous stage as a new stage in the new environment.
- Ability to use the cached images to make the overall process quicker.
- Reduces the risk of vulnerabilities found as the image size is smaller with the multi-stage builds.

(Source: https://dev.to/pavanbelagatti/what-are-multi-stage-docker-builds-1mi9)

Docker Scout:

Docker Scout is a solution for proactively enhancing your software supply chain security. By analysing your images, Docker Scout compiles an inventory of components, also known as a Software Bill of Materials (SBOM). The SBOM is matched against a continuously updated vulnerability database to pinpoint security weaknesses. (*Source: https://docs.docker.com/scout/*)

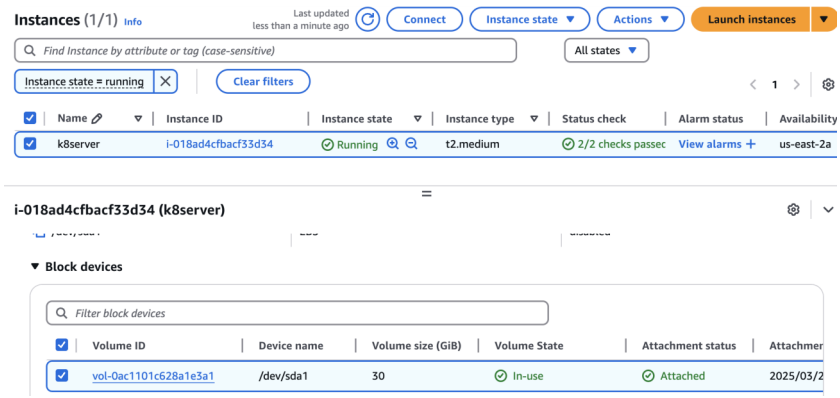
Commands:

docker scout cves <image-name> - Gives a detailed view of all the vulnerabilities for your image.

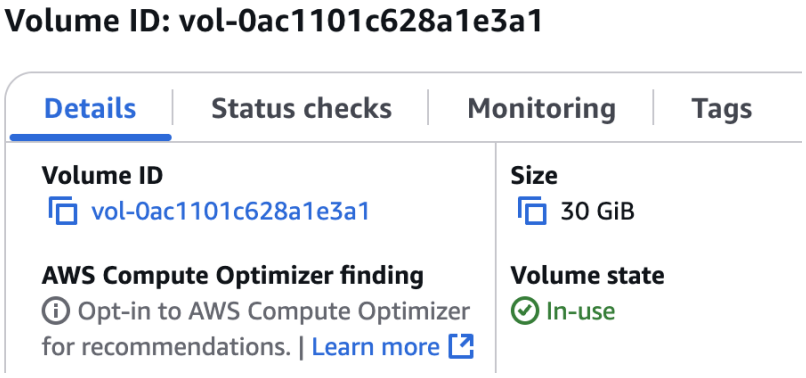
To increase the volume size of EC2 Instance:

As the EC2 instance, that we use is in free-tier, at times we face the issue “No space available on device”. To solve this problem, we can increase the size of the volume that is attached to the EC2 Instance. To do this, we follow the below steps:

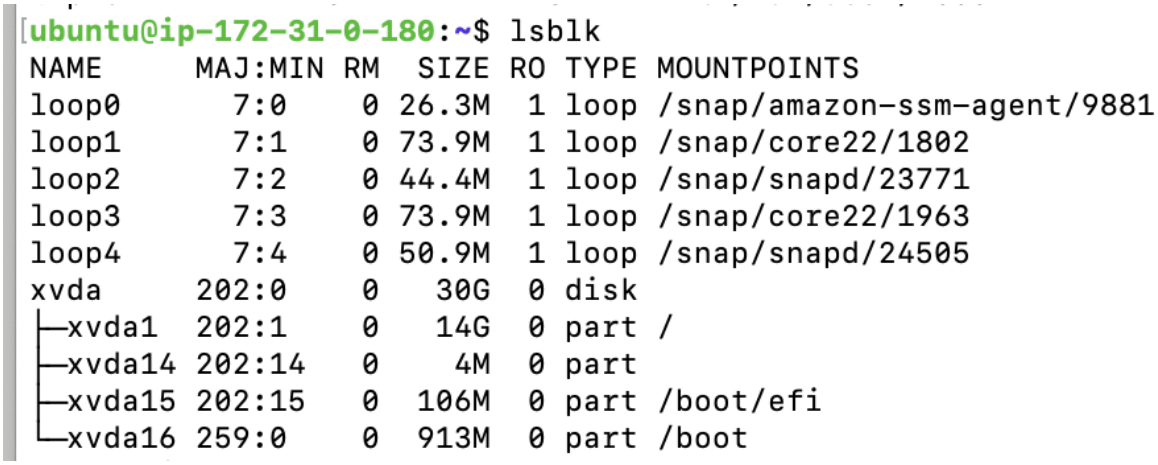
1. Navigate to the “Storage” section of your EC2 Instance and click on the volume.



2. Select the volume and click on **"Actions".** There, you will get an option to "**Modify volume".**
3. Edit the volume as per your need. It will take some time to initialize the volume. Wait until the Volume state is **"In-use"**



4. Now, go to your EC2 instance and run the command: **sudo apt install cloud-guest-utils**
5. If you run the command **"lsblk",** you ll see that the volume is available but not attached.



6. To grow the partition, run the command : **sudo growpart /dev/xvda 1**

```
ubuntu@ip-172-31-0-180:~$ sudo growpart /dev/xvda 1
CHANGED: partition=1 start=2099200 old: size=29358047 end=31457246 new: size=60815327 end=62914526
```

7. To attach the volume to the root(/) mountpoint, run the command: **sudo resize2fs /dev/xvda1**
8. Now, if you run **df -h**, you should be able to see that the total size of the root mountpoint has increased.