# Docker

Docker is regarded as an open-source platform which is used by software developers to deploy the application using containers. Containers are lightweight and isolated in nature.

Docker comes into play when an application is deployed in Environment-1 and that needs to be deployed in Environment-2(suppose a new computer with different OS). We need to install packages, software, DB and everything for running the application to that new environment. That is a very risky and tedious job because sometimes the versions may change in some application. Besides this, to deploy anything in PROD, it is risky to install all the necessary software before deploying the code. So to solve the above issue, Docker is helpful.

## Container

A container is a package which comprises of the code and the software that need to be installed to run the same. It creates a bundle of the code and its dependencies which can be installed in a foreign machine from the Host machine where it is created.

If there is a version mismatch in the dependencies (software) of the application, then Docker container will allow us to create a new version of the application with updated version without affecting the older version.

### Features of Containers:

1. Portable
2. Lightweight

## Image

Docker Image is an executable file which contains the instruction to build the container. Its just like the Class and object concept in OOPs where Class comprises of the Blueprint and objects are nothing but instances of the given blueprint. Similarly, Image is a blueprint which can be considered as a “Class” and Containers are the instances which can be referred as “Objects”. One single Image has multiple containers which are shared across the developers to be installed in the local machine.

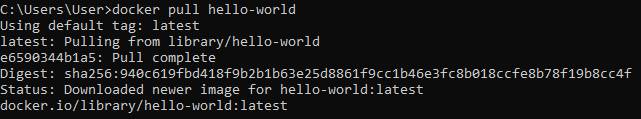
Docker Image is a static snapshot of what the local developer environment should look like while Container is a running instance of the associated Docker Image.

### How to setup Docker in our local machine

* Visit the Website - <https://www.docker.com/>
* Download the Docker application suitable for your machine.
* Install the same using the “Recommended setup”

### Practice Docker Hub Public images

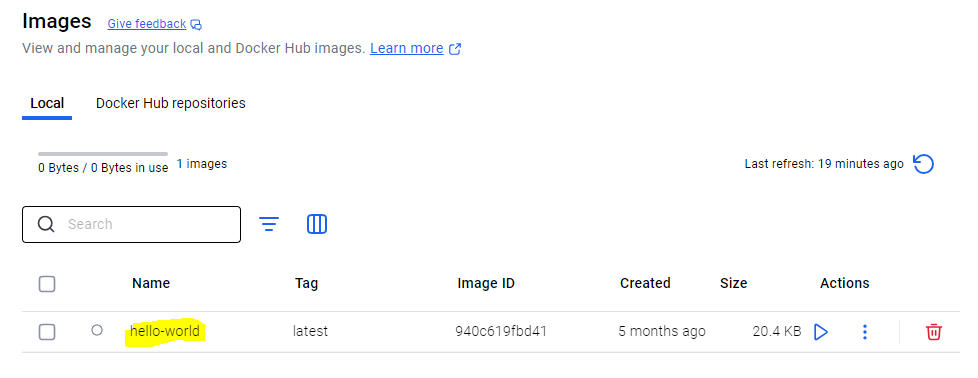
* Visit the Docker Hub website - <https://hub.docker.com/> to access public Docker images – Optional Step
* Search for “Hello-world” in the above website.
* If we want to pull any image then simply we should write the below code in our command prompt.
  + Open “cmd” prompt and write the below code
  + docker pull <image name>
  + eg. docker pull hello-world



* We can check the images that are present in our Docker application
  + Run the below code in command prompt
  + docker images



* Even the pulled image is visible in Docker application

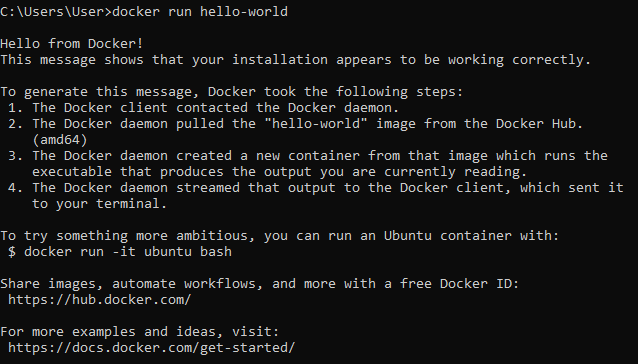


## How to create a container

As we run the image an instance of the image will get created which is referred as “Container”. The container can be referred across different machines to deploy the code.

### Run the Docker container from the above image

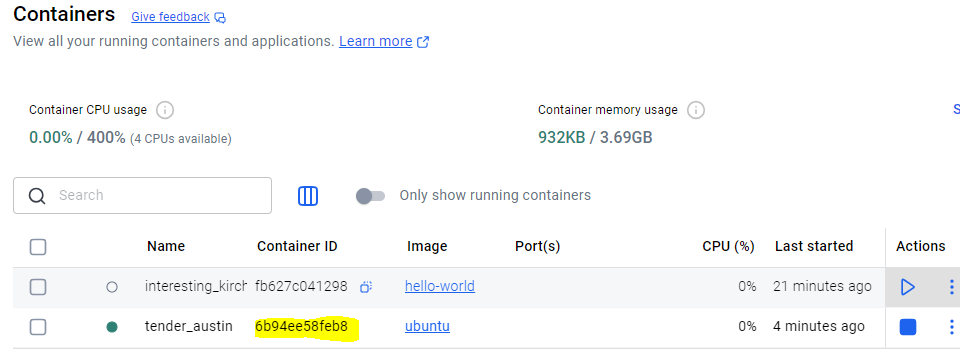
* Run the docker image to create a container using below code
  + docker run <image\_name>
    - eg: docker run hello-world



* In order to run the Docker images in Interactive mode, we should use “-it” before the image name.
  + docker run –it <image\_name>
    - Eg: docker run –it Ubuntu

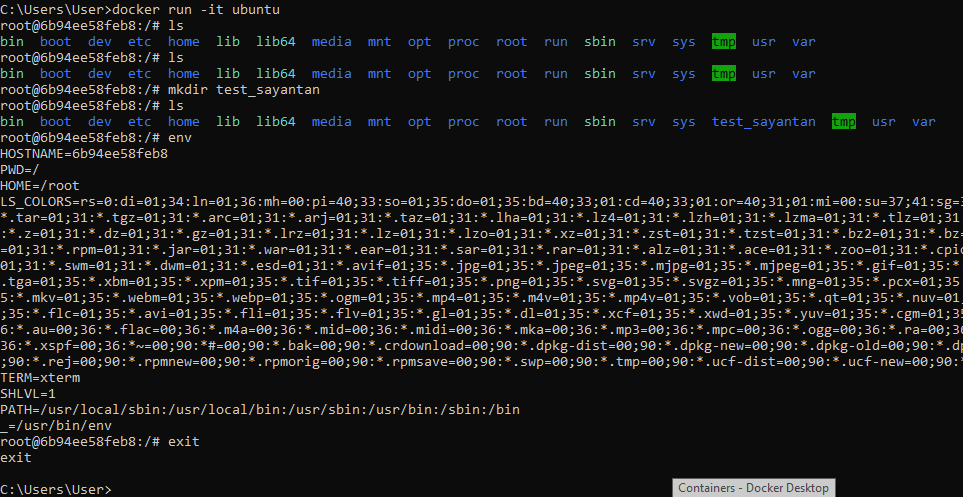


Here we can see that the docker has entered inside the container

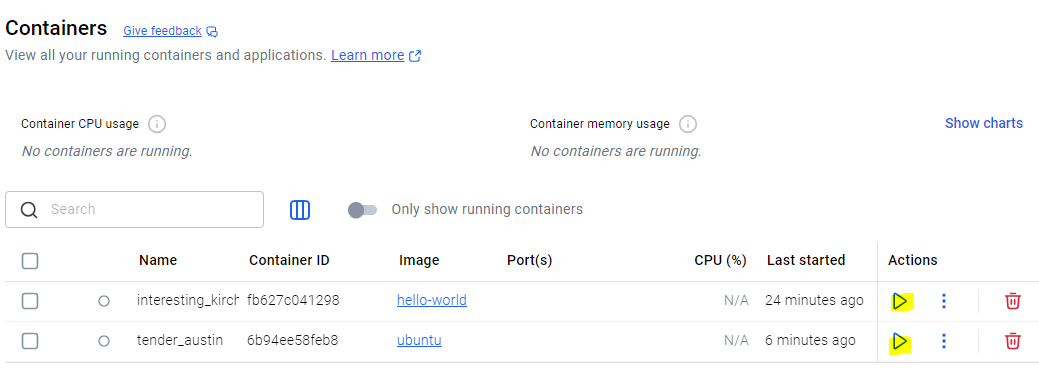


The container id is actually matching

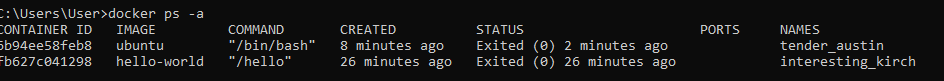
We can run different commands in Ubuntu container as shown below and once we type “Exit”, we will come out of the container



Along with that, the container will stop running in Docker UI as well as shown below



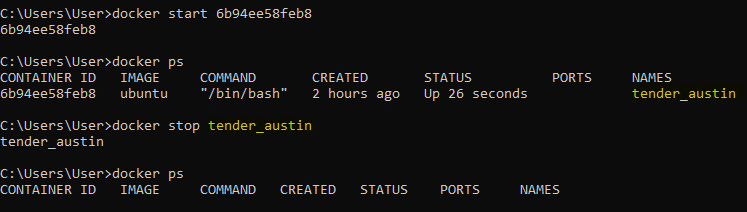
* To view the containers that had been created so far, we will run the below command
  + docker ps –a



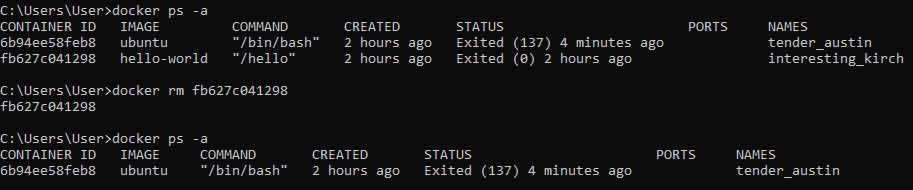
* To view the running containers
  + docker ps



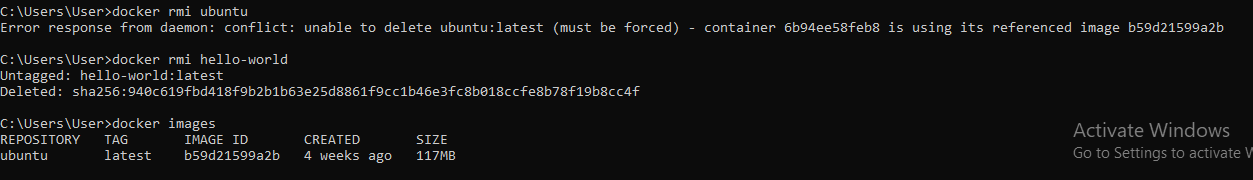
* For running and stooping any existing container
  + docker start <container\_name or container\_id>
    - eg : docker start Ubuntu



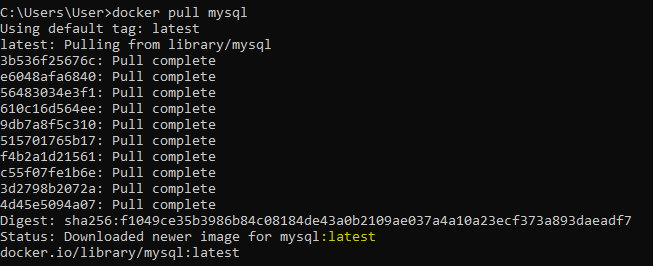
* To remove or destroy any container we need to write
  + docker rm <container name or container id>
    - eg : docker ps –a hello-world



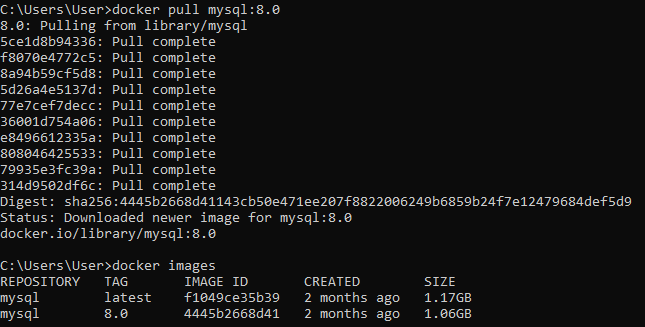
* To remove or destroy any image we will write below command but firstly we need to destroy the corresponding container before destroying the image else it will throw an error.
  + docker rmi <image name or image id>
    - docker rmi hello-world



**Note: If we are just running “docker pull <imagename>” then the latest version of the image will get pulled by default.**



* Now if we need to pull any specific version of the said image, we need to mention that in the command itself
  + docker pull mysql:8.0 --- Here version of mysql is selected as 8.0



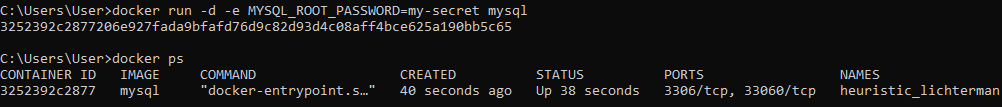
**Detach Mode – Here we can run any image in the background. By default all the containers run in attach mode but if we need to run any container separately in detach mode then we need to run:**

**docker run –d <Image\_Name>**

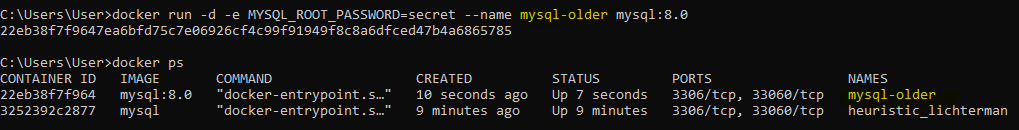
**We can run environment variables (if required) while executing the image. The environment variable can be declared using “-e <variable>” in the docker run command.**

**docker run –d –e <env variable> <Image name>**

**docker run –e <env variable> <Image name>**



* We can also provide custom name to the containers using the attribute “—name” in the run command
  + docker run –d –e <variable> -- name <preferred name> <Image name>

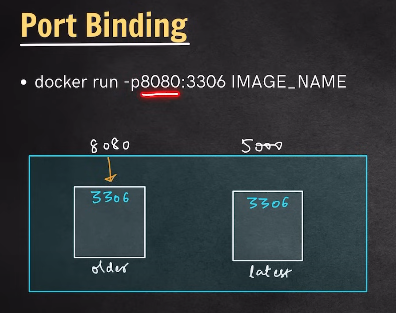


## Layers of Image

The Image is a composition of multiple layers which are built one above the other. The layers get installed while we pull the image from Docker. The bottom layer is considered as “Base Layer” which is usually a Linux system while the topmost layer is the Container Layer which is actually editable. Apart from the container layer, all other layers of an Image are Read-only and immutable in nature.

## Ports of Docker Containers – How to bind ports

The ports shown in the Docker containers are the ports which are respective to the container and they are different from the port that is being used for the machine. As a result we may see similar ports for two different containers.

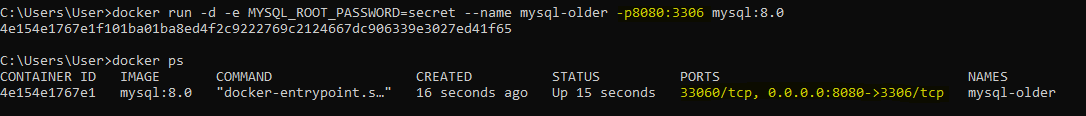


Here we have bind the 8080 port of the system with the 3306 port of the docker container.

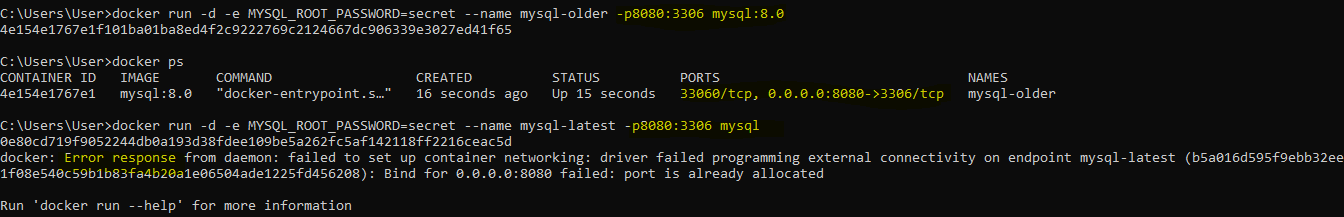
Syntax to bind ports – docker run –p<host port>:<container port> <image\_name>

Eg : docker run –p8080:3600 mysql

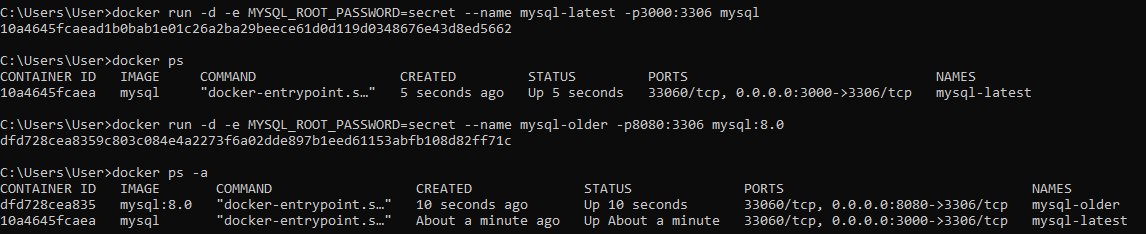
**What is Port Binding? – When we map the Host port with the container port in docker, it is known as Port Binding.**



Now if we are going to bind the same port for a different container, we will see the error as the host port is already bind with one of the container



We can bind different ports of the Host machine with different ports of the container

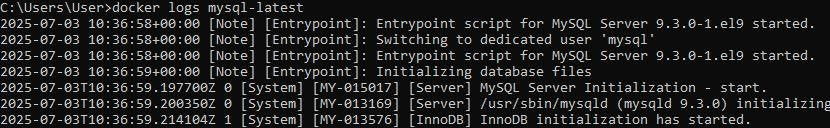


## Troubleshooting Containers

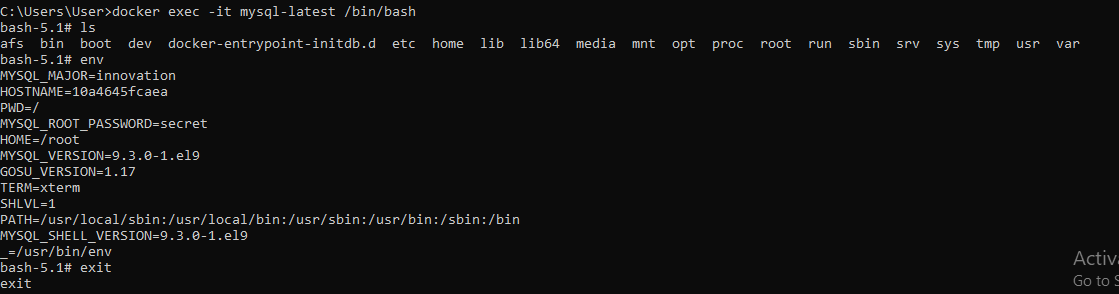
In order to troubleshoot any issues related to Docker containers, we need to go through the logs associated to the docker containers in order to identify the root cause.

### Commands to get the logs of containers

* The below command executes the logs related to the specific docker container
  + docker logs <container name>
    - eg: docker logs mysql-latest



* If we want to run additional commands on an already running container, we need to open the bash or shell in interactive mode.
  + docker exec –it <container id> /bin/bash
  + docker exec –it <container id> /bin/sh
    - ex: docker exec –it mysql-latest /bin/bash



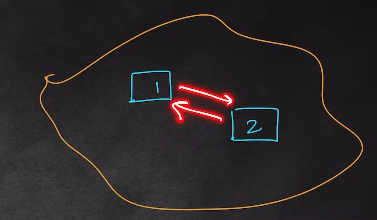
The exit command executed in the last line brings out of the interactive bash or shell but not from the container

## Difference between Docker and VM

|  |  |
| --- | --- |
| **Docker** | **Virtual Machines** |
| Docker just acts upon the Application layer and hence it has less overhead which makes it lightweight in nature. | VM acts as a complete package virtualizing both Application Layer and Host OS Kernel. Hence it’s not lightweight like Docker. |
| Docker is faster and smaller in size | VMs are larger and slower compared to Docker |
| Docker basically is hosted on Linux OS. But it works on any OS as Docker Desktop adds a hypervisor to create a Linux Kernel to run Docker commands | VMs are compatible to any OS as it provides the OS Kernel in itself alongside the application layer. |

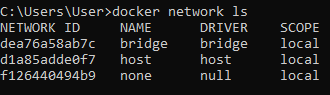
## Docker Network

Docker has the ability to setup an isolated network to setup communication or interaction between two separate containers without need of any port or local host.

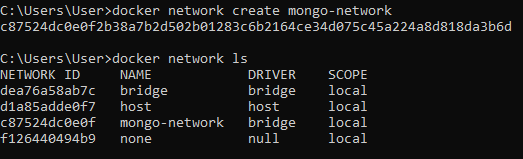


### Command for Docker network

* To view all the available Docker network in system
  + docker network ls



* If we want to create a new Docker network in system
  + docker create <network name>
    - eg: docker create mongo-network

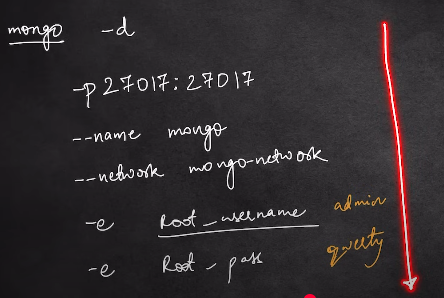


## Developing using Docker

We will setup Mongo DB and Mongo-Express containers using Docker. Mongo DB is the database and Mongo-Express is the interactive admin interface which is required to fetch any information by connecting to the Mongo DB. Now since both the containers need to interact among themselves to fetch required information so it’s essential to setup an isolated Docker network.

### Setting up Mongo DB and Mongo-Express

The official Mongo DB and Mongo-express Image are available in Docker hub (<https://hub.docker.com/_/mongo>).

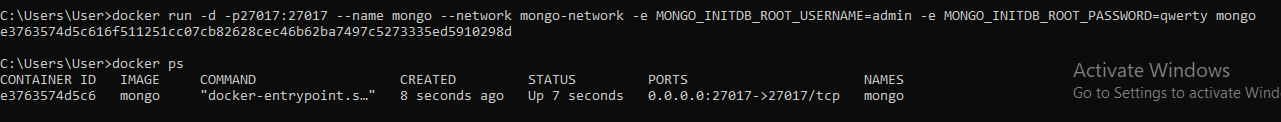


#### Mongo DB:-

We need to setup Mongo in detach mode (-d) with a custom name (--name mongo) under a docker network (--network mongo-network) having some environment variables as username and password (-e root\_username admin -e root\_password qwerty)

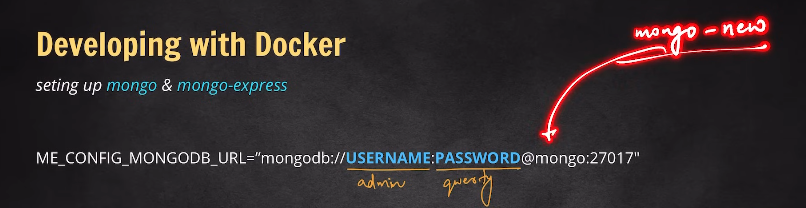
**Code –**

docker run –d –p27017:27017 –name mongo –network mongo-network –e MONGO\_INITDB\_ROOT\_USERNAME=admin –e MONGO\_INITDB\_ROOT\_PASSWORD=qwerty mongo



#### Mongo Express:-

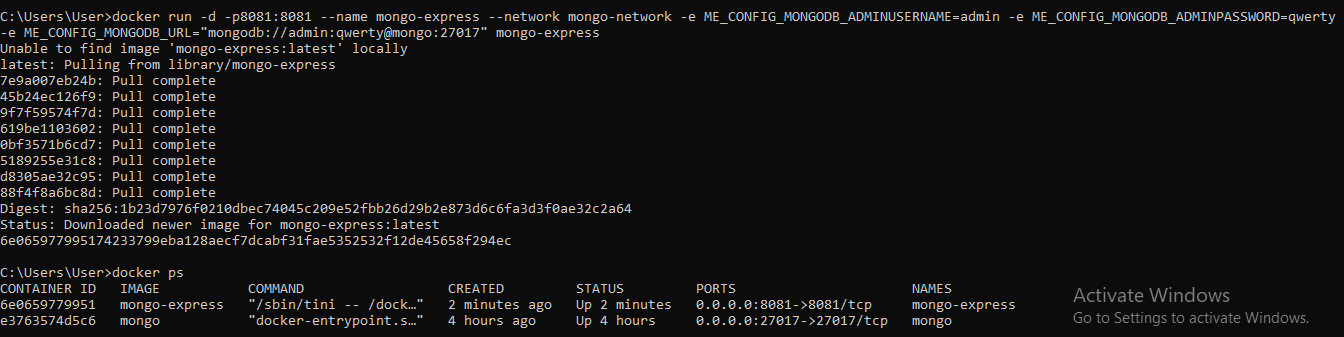
We need to setup Mongo in detach mode (-d) with a custom name (--name mongo) under a docker network (--network mongo-network) having some environment variables as username, password (-e root\_username admin –e root\_password qwerty) and one more as MongoDBURL



Here mongo/mongo-new is the name of the container and 27017 is port number

**Code -**

docker run -d -p8081:8081 --name mongo-express --network mongo-network -e ME\_CONFIG\_MONGODB\_ADMINUSERNAME=admin -e ME\_CONFIG\_MONGODB\_ADMINPASSWORD=qwerty -e ME\_CONFIG\_MONGODB\_URL="mongodb://admin:qwerty@mongo:27017" mongo-express

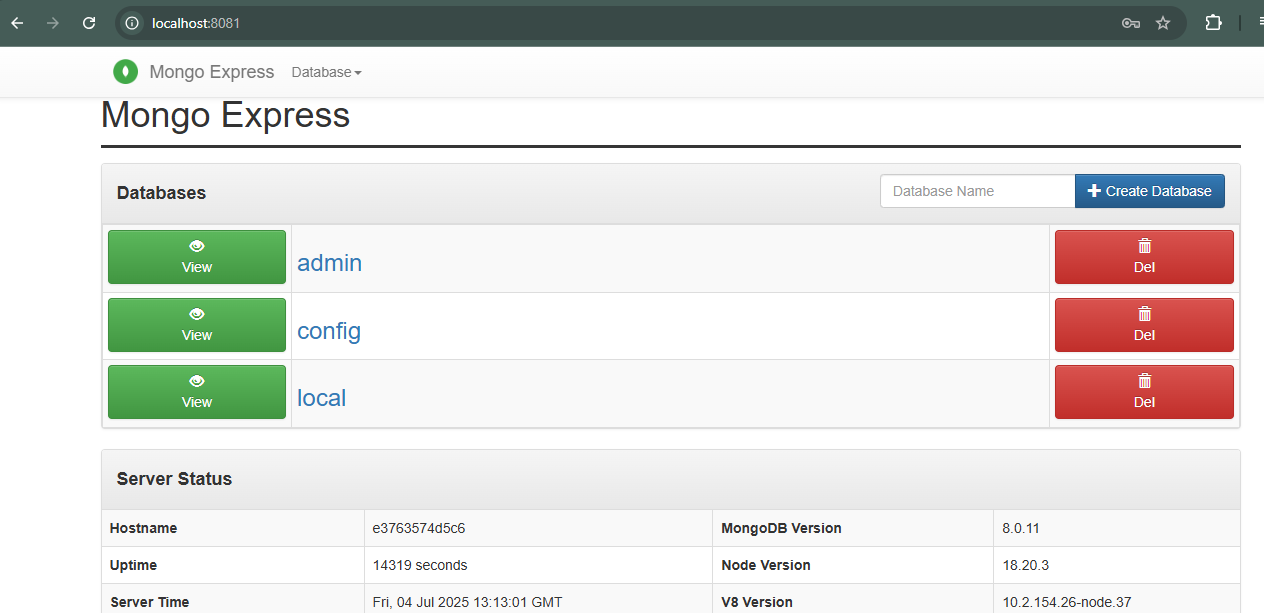


We will be able to validate whether the containers are running by connecting the port 8081 using localhost in the URL. As we type the URL “localhost:8081”, it will prompt for a credential and the default credential is –

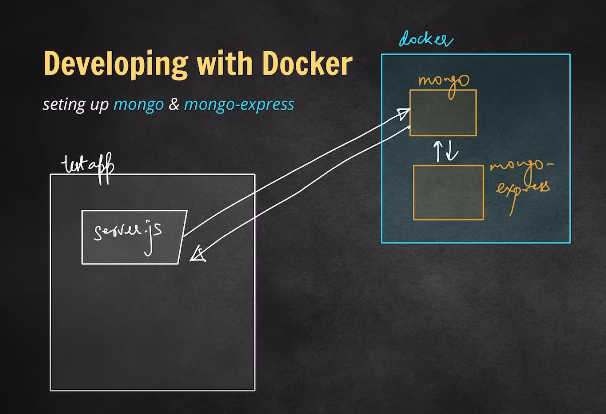
Username – admin

Password – pass

Once the same is validated the below screen will be visible to us



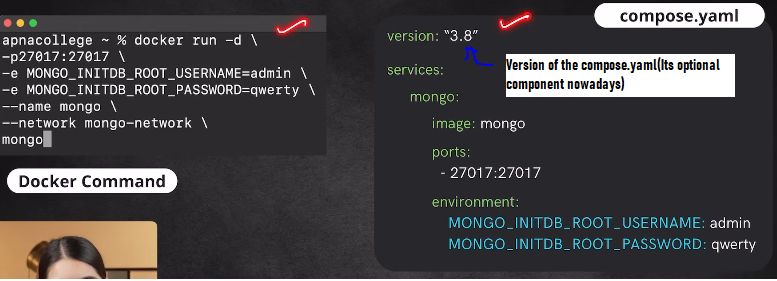
So Mongo Express provides us the graphical interface of the Mongo DB database which we have containerized under single docker network.



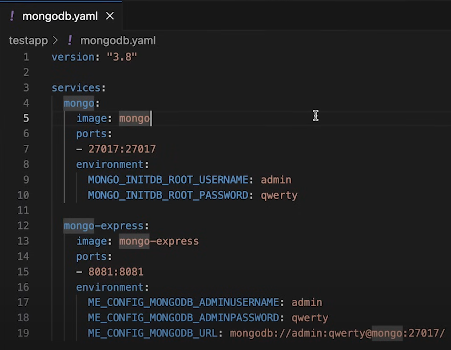
Basically Mongo and Mongo-Express are interacting among themselves since they are created under same docker network. So whatever being inserted in mongo is being shown in Mongo-Express.

## Docker Compose

Docker compose is used to define and run multi container application at a time. Here we convert all the commands associated with the container in a single file which is saved in the format of <file name>.yaml (Yet Another Markup Language). Now instead of running the docker command from terminal, we can run it from the aforementioned file. The benefit of running the command using the said file makes it more organized and the commands can be edited easily.



The commands executed in “Developing using Docker” section to setup a “Mongo” and “mongo-express” container can be enlisted in a YAML file which can be shared across. The snapshot of the YAML file highlighting the commands is shared below



### Important commands under Docker compose

* If we want to start the container in detach mode, the below command need to be executed for the respective YAML file
  + docker compose –f <filename.yaml> up –d
* If we want to remove the docker container permanently then we need to run the below command for the respective YAML file
  + docker compose –f <filename.yaml> down

1. It is noteworthy that in order to run the docker compose commands, we need to go to the folder where the docker compose file i.e. the <File name.yaml> file is located.
2. It is to be noted that inside the Docker compose YAML file, we never create the Docker network where the docker container should run. Docker compose creates the network without being mentioned explicitly.