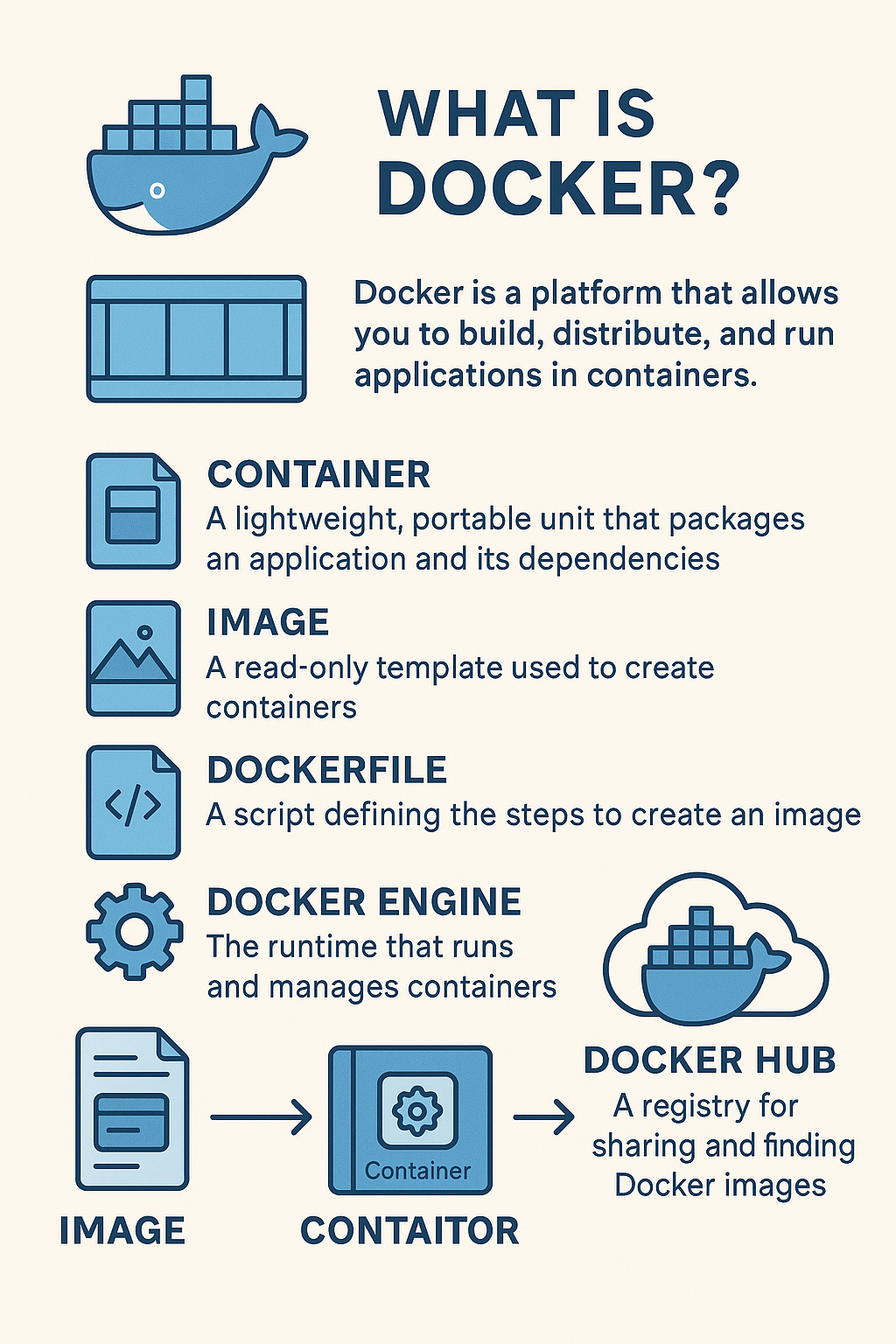
**What is Docker?**

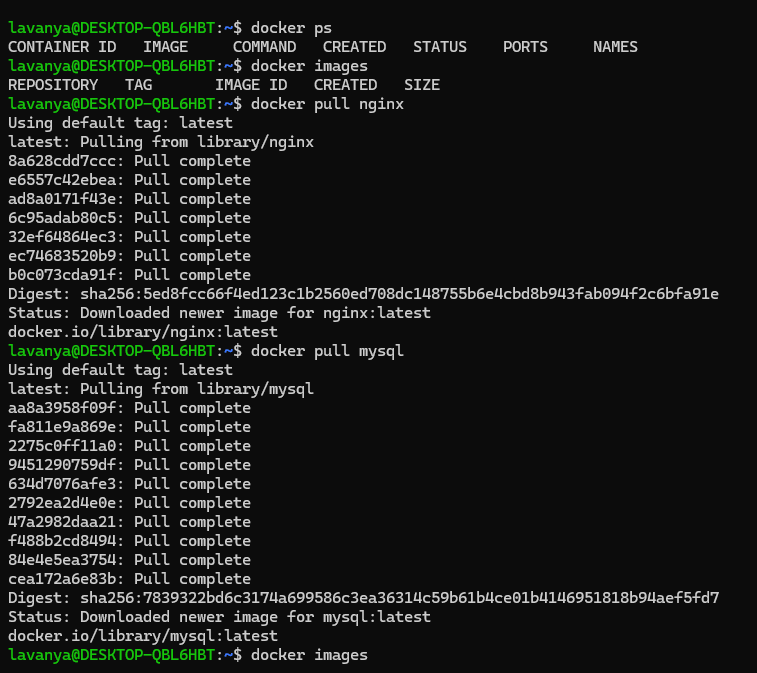
**Docker** is a **platform** that allows developers to **package**, **distribute**, and **run applications** in **containers**. Containers are lightweight, portable units that include everything needed to run the software—**code, runtime, system tools, libraries, and settings**.

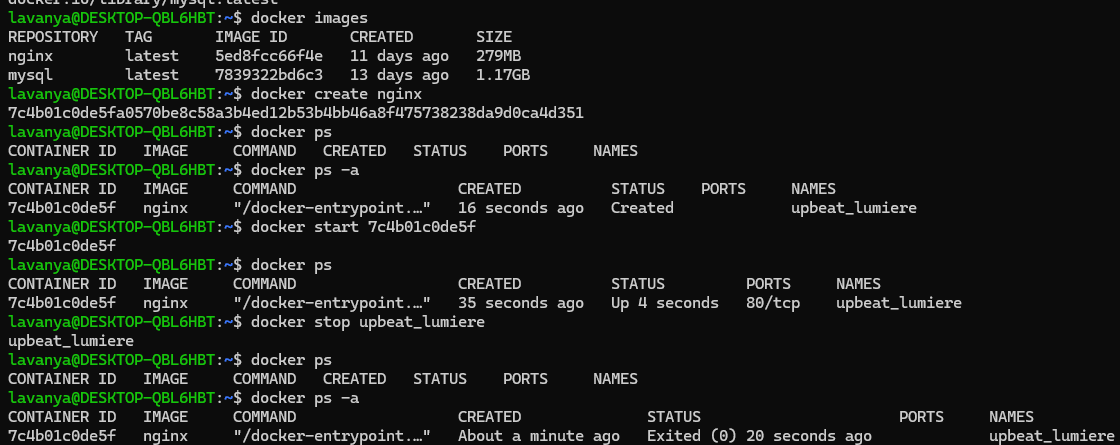
| **Concept** | **Description** |
| --- | --- |
| **Container** | A lightweight, standalone, executable unit that includes an application and its dependencies. |
| **Image** | A snapshot or template used to create containers. Immutable and portable. |
| **Dockerfile** | A script of instructions on how to build a Docker image. |
| **Docker Engine** | The runtime that builds and runs containers. |
| **Docker Hub** | A cloud-based registry where Docker images are stored and shared. |

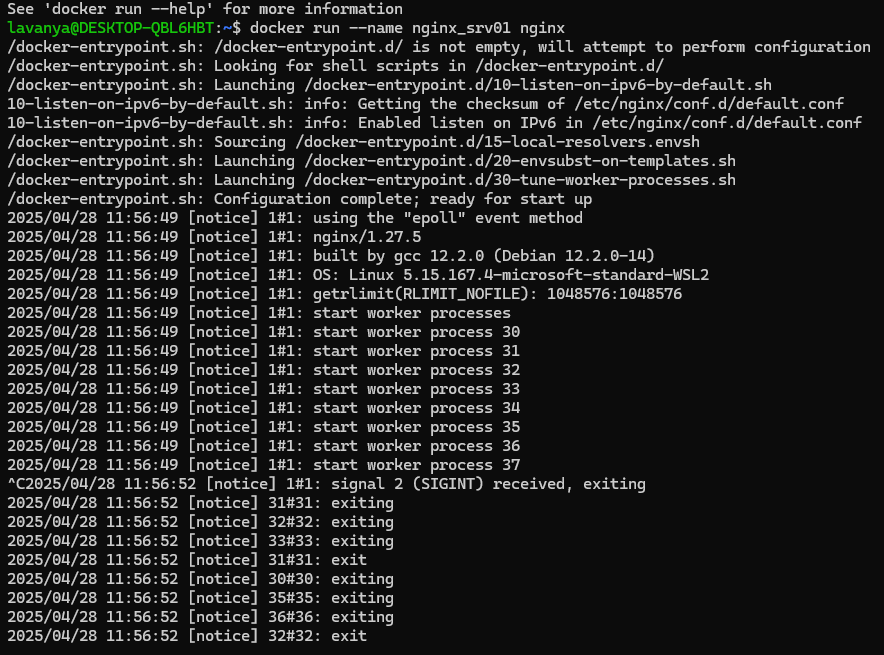
**Why Use Docker?**

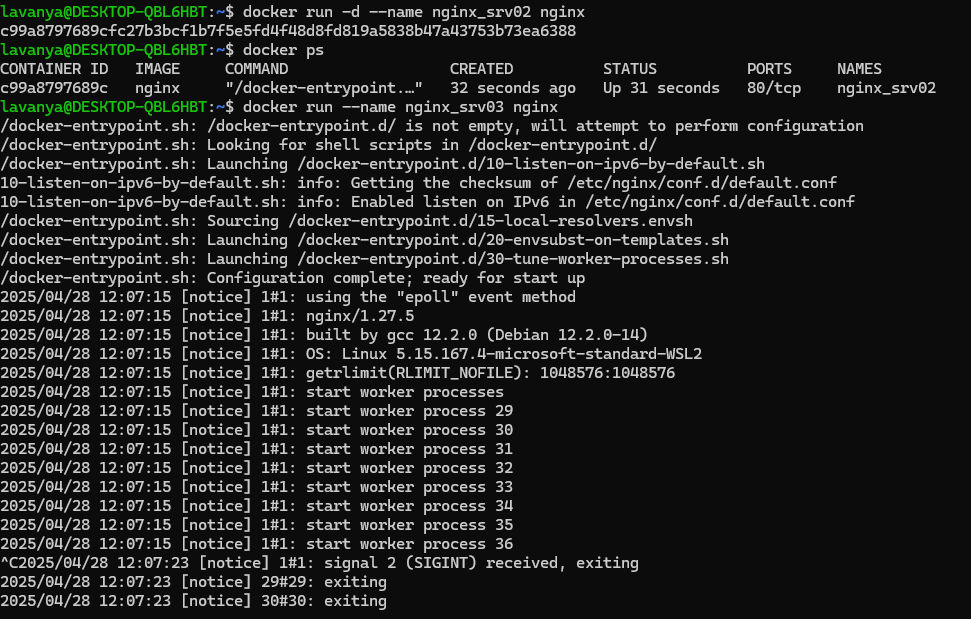
* ✅ **Consistency** across environments (dev, test, prod)
* ✅ **Isolation** between applications
* ✅ **Portability** (runs anywhere: laptop, server, cloud)
* ✅ **Efficiency** (uses less resources than VMs)
* ✅ **Rapid deployment** and scaling

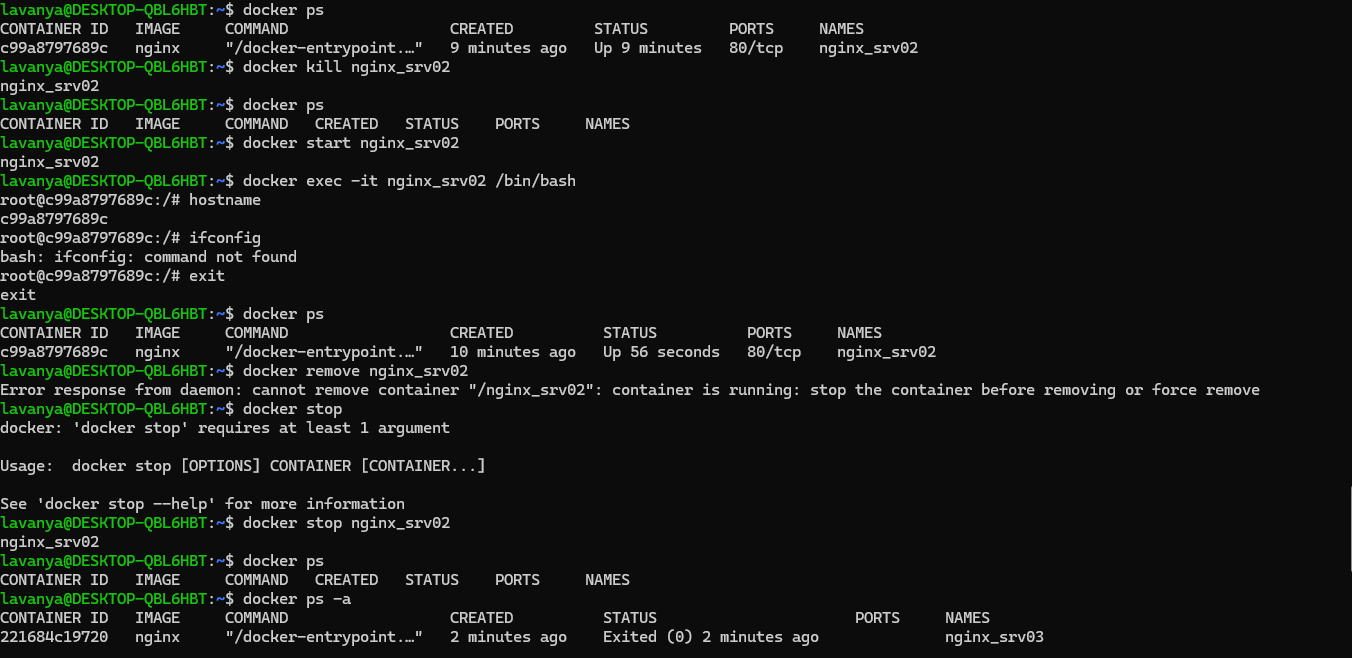


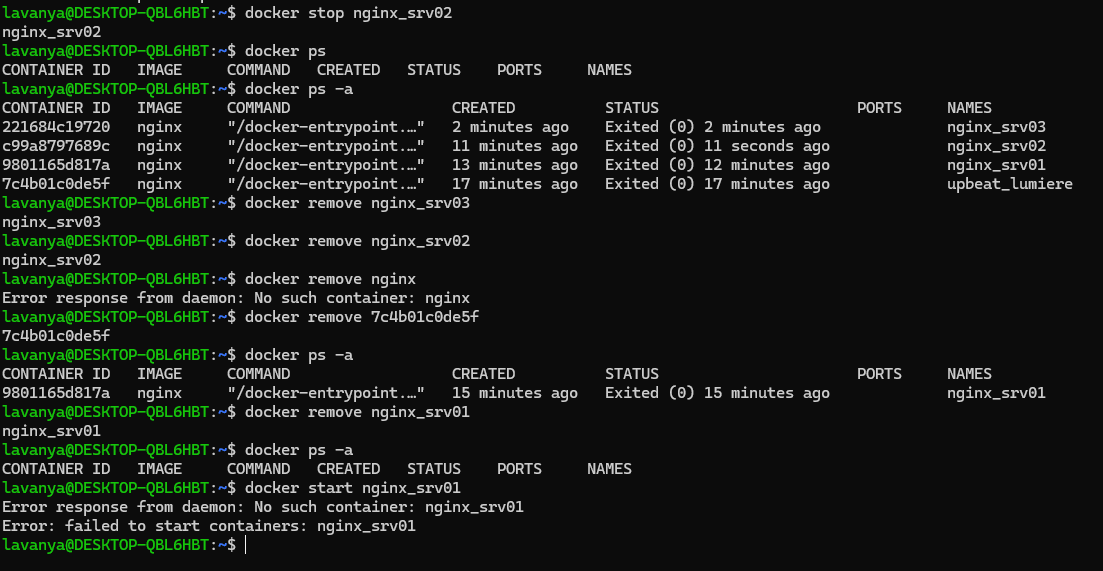






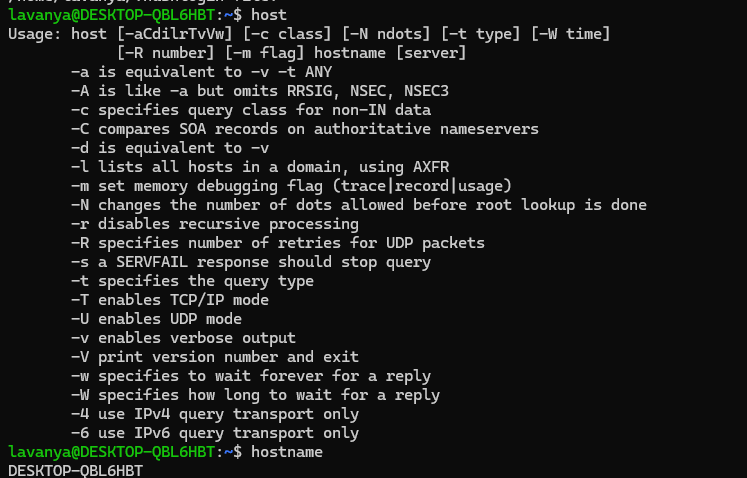


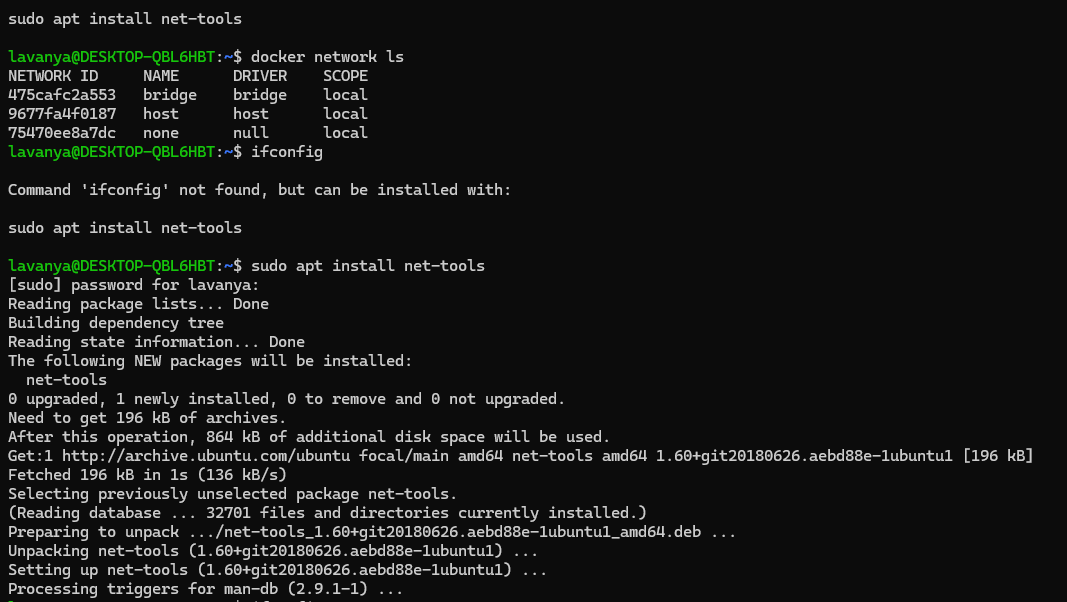


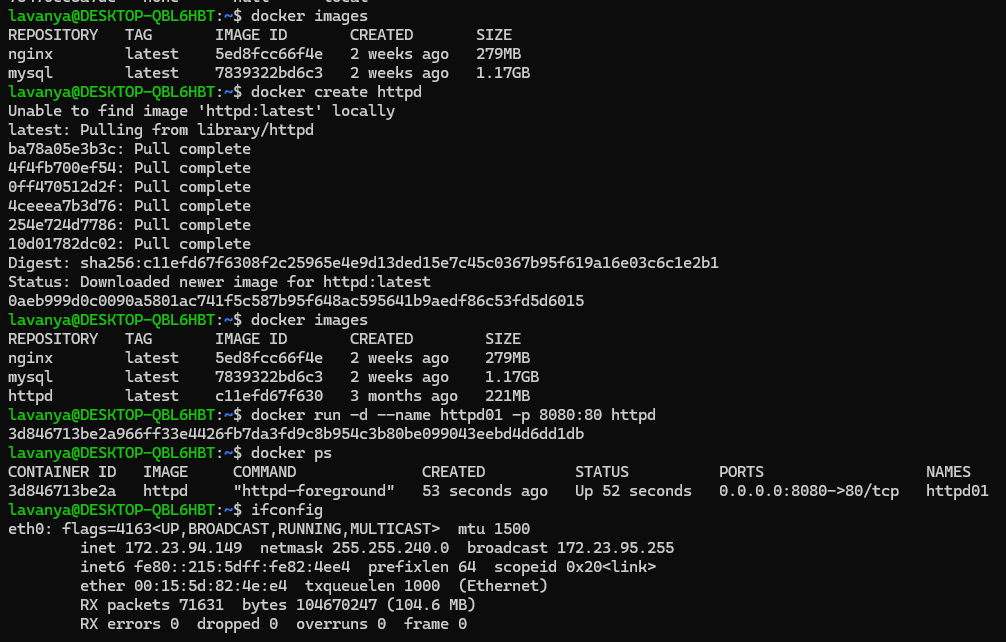


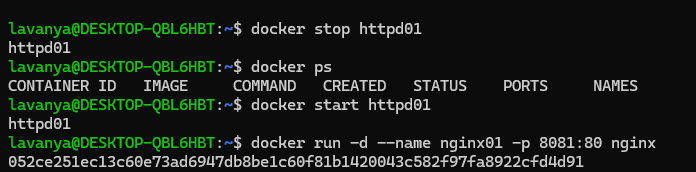
DOCKER NETWORKING :

1. Bridge driver
2. Host driver
3. None driver









lavanya@DESKTOP-QBL6HBT:~$ **docker inspect bridge**

[

{

"Name": "bridge",

"Id": "475cafc2a5534223d6485a534e17e4aaacf42a62aef77357393f54e53c0b3a18",

"Created": "2025-04-28T11:17:39.988403731Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv4": true,

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": null,

"Config": [

{

"Subnet": "172.17.0.0/16",

"Gateway": "172.17.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"052ce251ec13c60e73ad6947db8be1c60f81b1420043c582f97fa8922cfd4d91": {

"Name": "nginx01",

"EndpointID": "bd4a343c653638cb4096cbb8a40aa9d513a4a0d1ac701aff5ff4b1d087e98ecd",

"MacAddress": "36:22:9a:38:89:95",

"IPv4Address": "172.17.0.3/16",

"IPv6Address": ""

},

"3d846713be2a966ff33e4426fb7da3fd9c8b954c3b80be099043eebd4d6dd1db": {

"Name": "httpd01",

"EndpointID": "6f4d7bb7ea1aef5e0723e5788b030faa3755f14e8d94c64852c6b52588deefef",

"MacAddress": "8e:f1:14:36:fc:e0",

"IPv4Address": "172.17.0.2/16",

"IPv6Address": ""

}

},

"Options": {

"com.docker.network.bridge.default\_bridge": "true",

"com.docker.network.bridge.enable\_icc": "true",

"com.docker.network.bridge.enable\_ip\_masquerade": "true",

"com.docker.network.bridge.host\_binding\_ipv4": "0.0.0.0",

"com.docker.network.bridge.name": "docker0",

"com.docker.network.driver.mtu": "1500"

},

"Labels": {}

}

]

lavanya@DESKTOP-QBL6HBT:~$ docker run -d --name httpd02 --net host httpd

656b2a106ad3e96355d4e2a22d497732b7f6b23e21bda0c6d1071be9980ac25a

lavanya@DESKTOP-QBL6HBT:~$ ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.23.94.149 netmask 255.255.240.0 broadcast 172.23.95.255

inet6 fe80::215:5dff:fe82:4ee4 prefixlen 64 scopeid 0x20<link>

ether 00:15:5d:82:4e:e4 txqueuelen 1000 (Ethernet)

RX packets 72427 bytes 104769010 (104.7 MB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 19380 bytes 1540802 (1.5 MB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

inet6 ::1 prefixlen 128 scopeid 0x10<host>

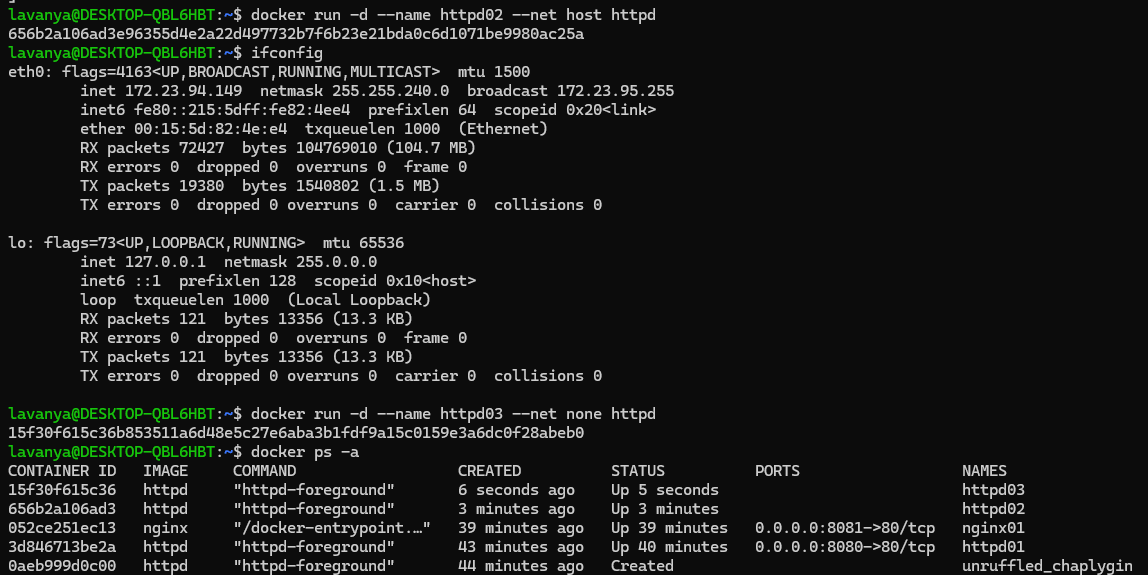
loop txqueuelen 1000 (Local Loopback)

RX packets 121 bytes 13356 (13.3 KB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 121 bytes 13356 (13.3 KB)

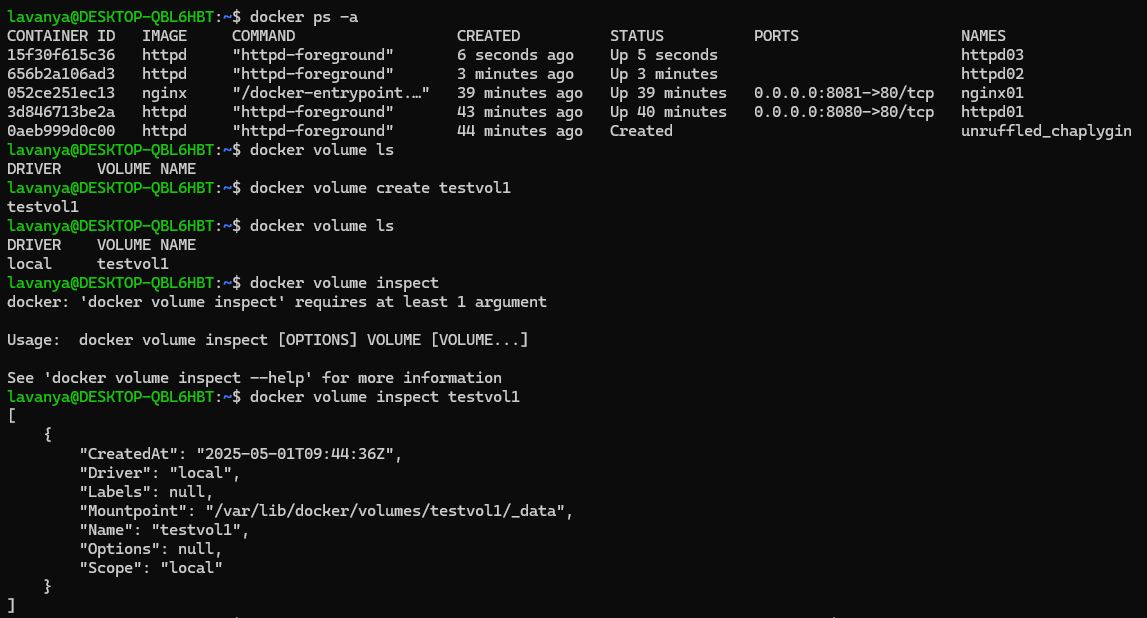
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

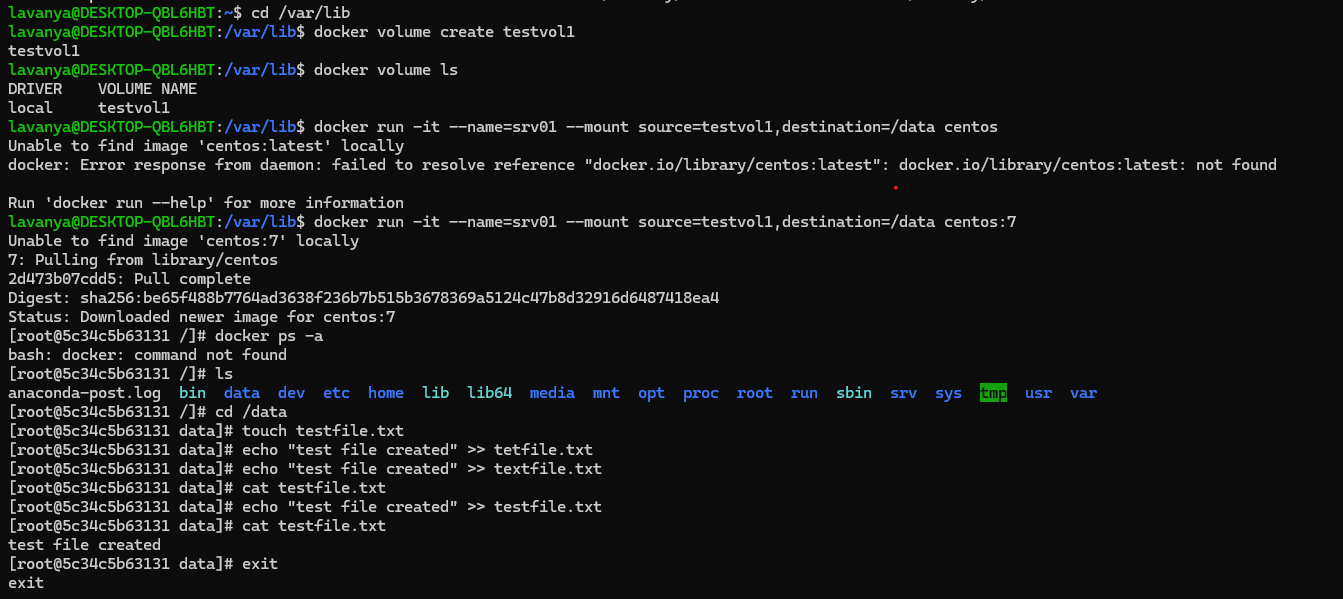


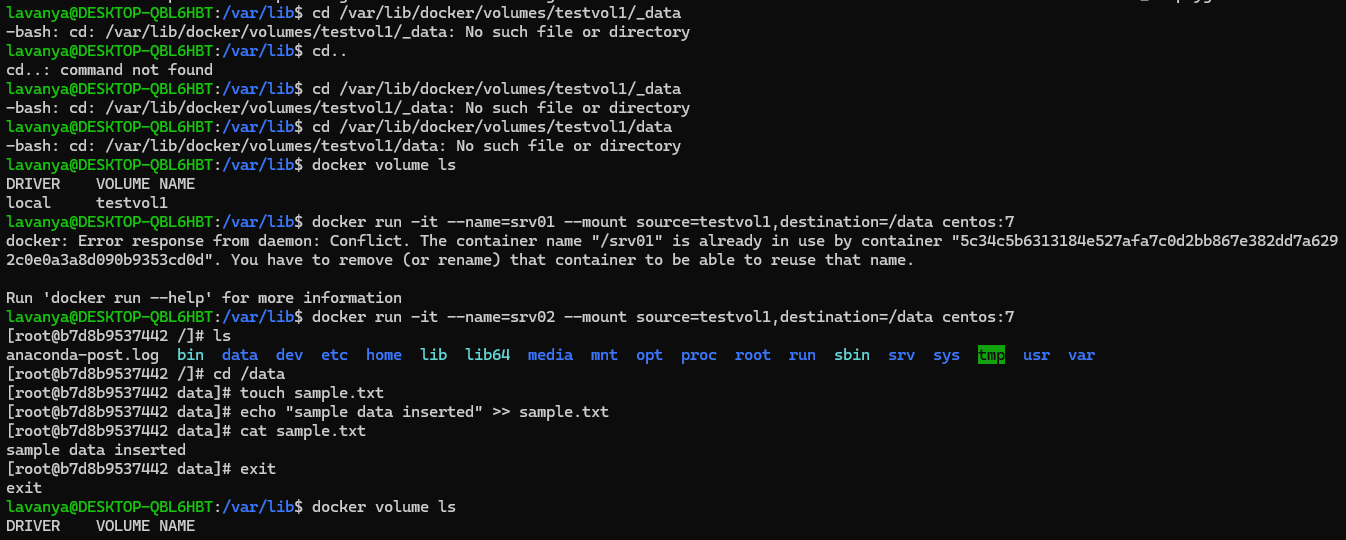
To open that copy the ip address of it with the port and run it’s respective page is opened -> Bridge

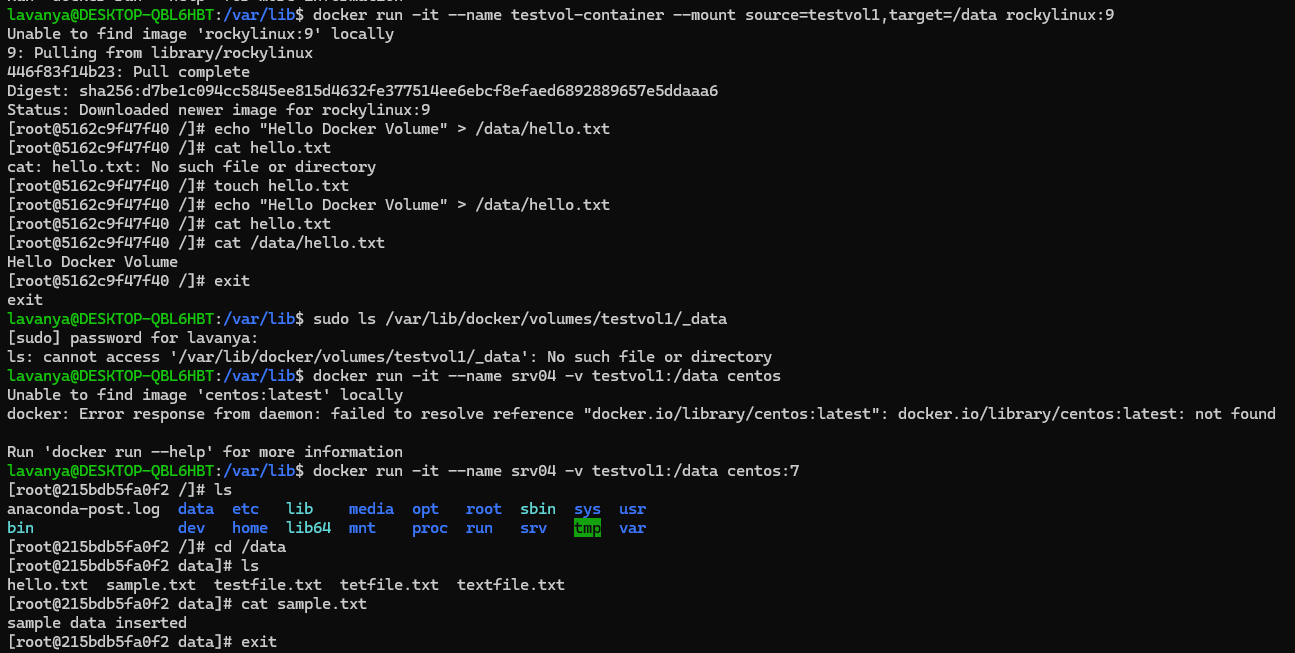
To open that copy only the ip address of it -> host

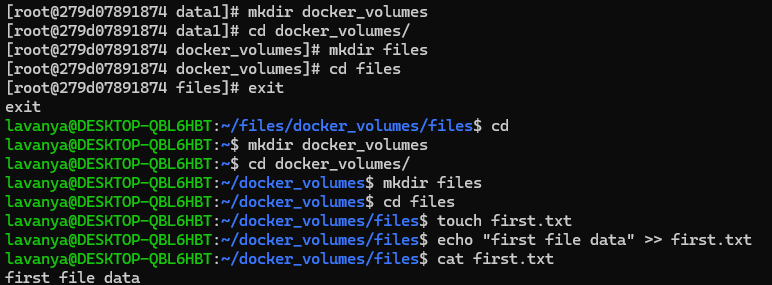
DOCKER VOLUMES





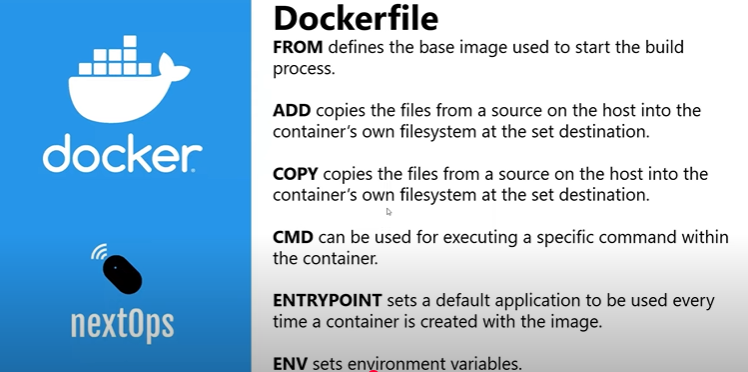


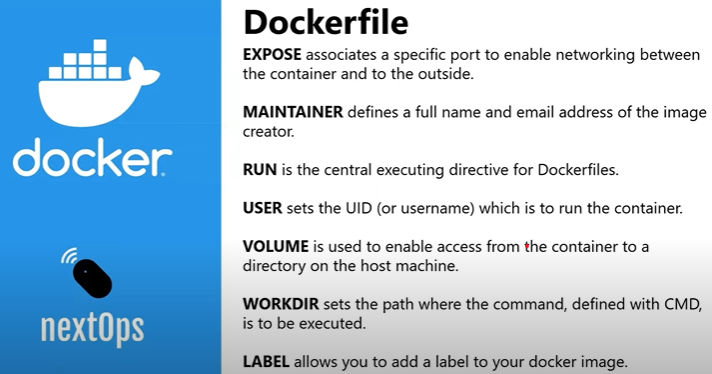




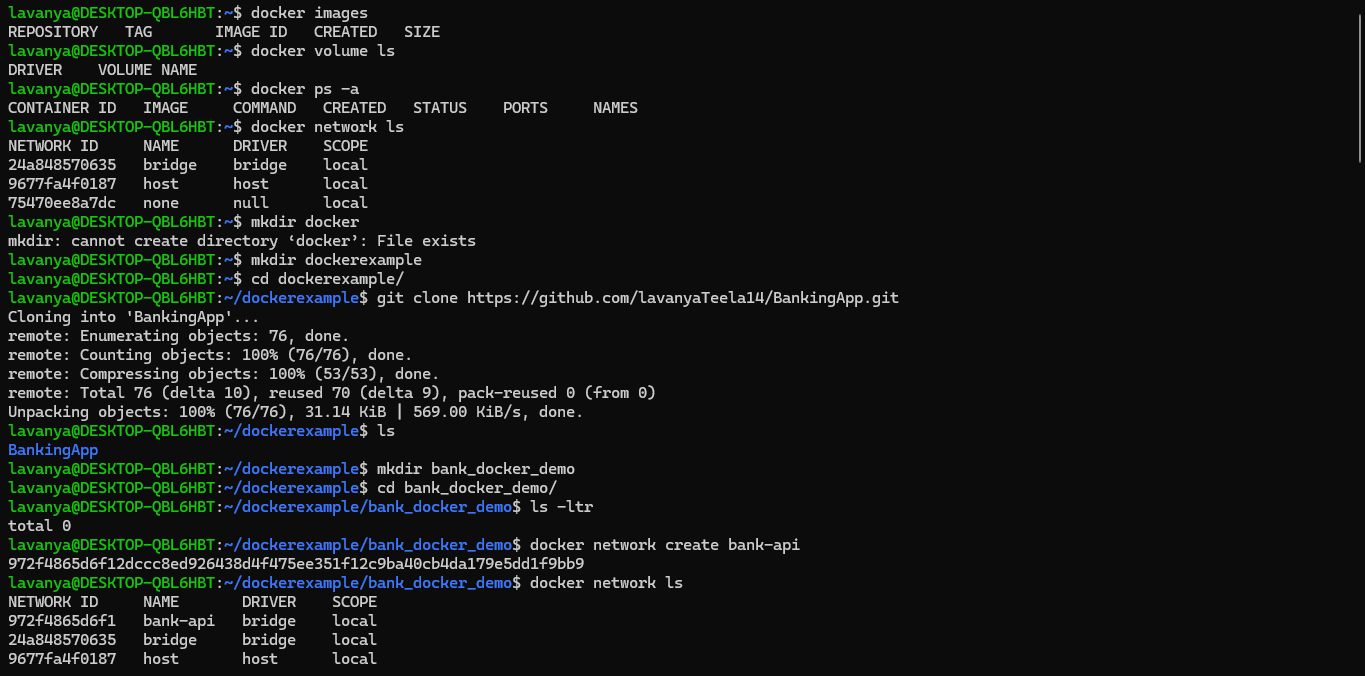
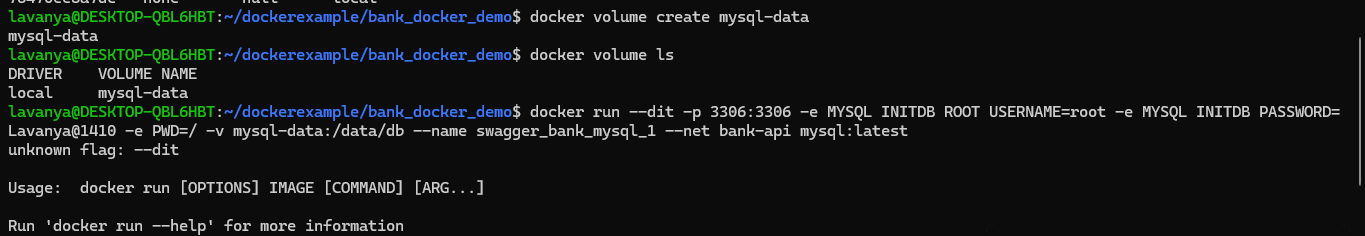


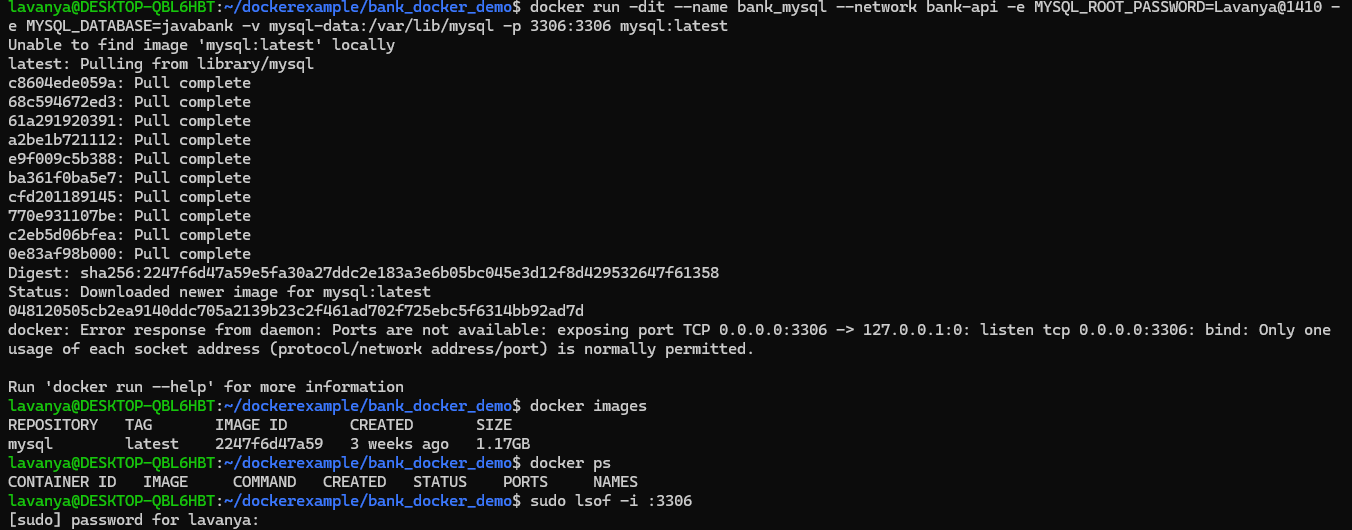
DOCKER FILES :

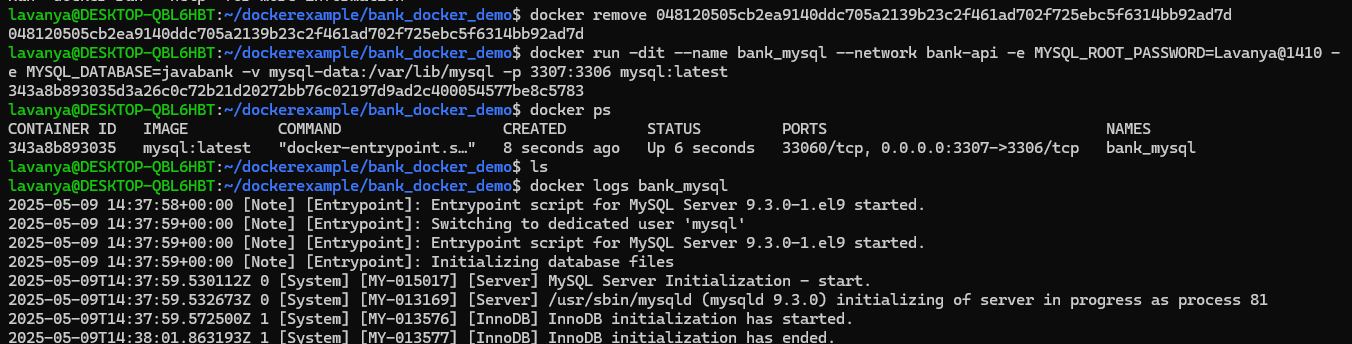




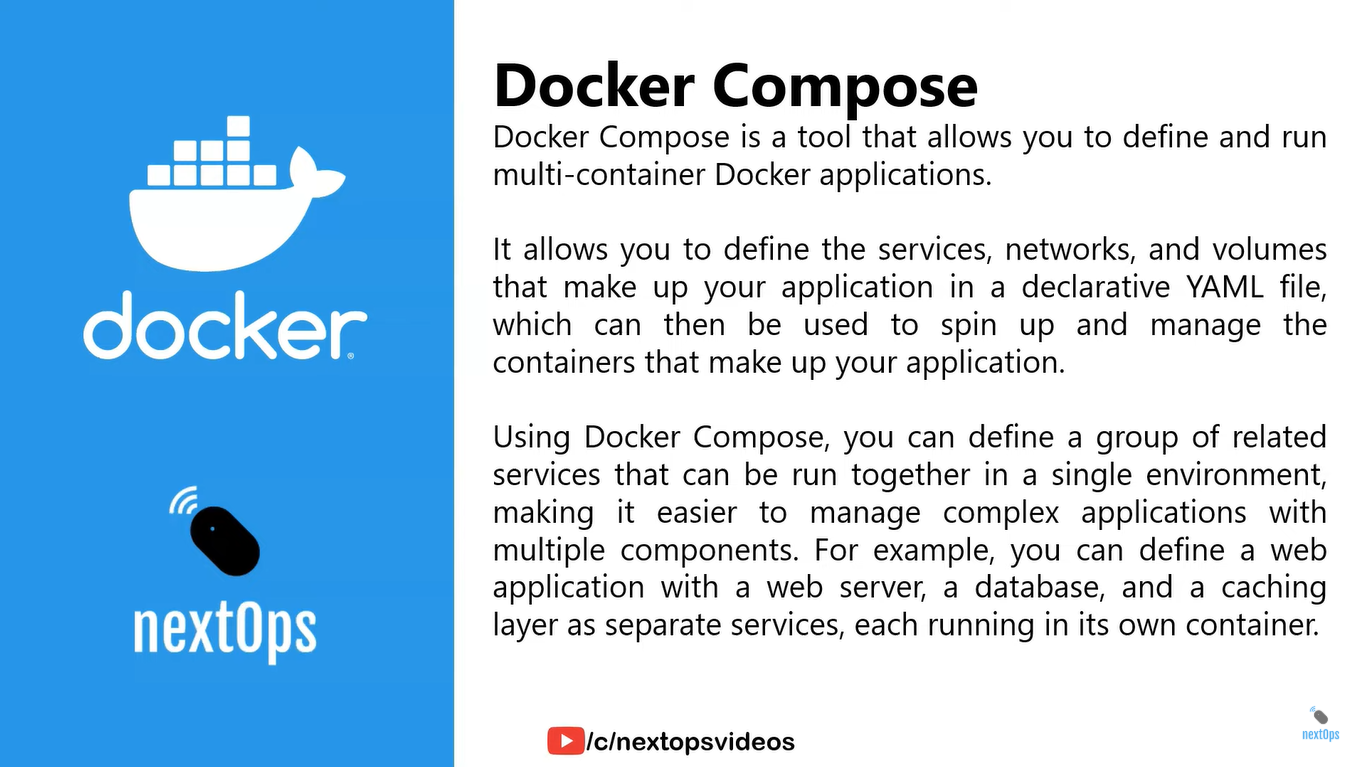
Multi container appl :





Docker compose :



Firstly created Dockerfile in root folder

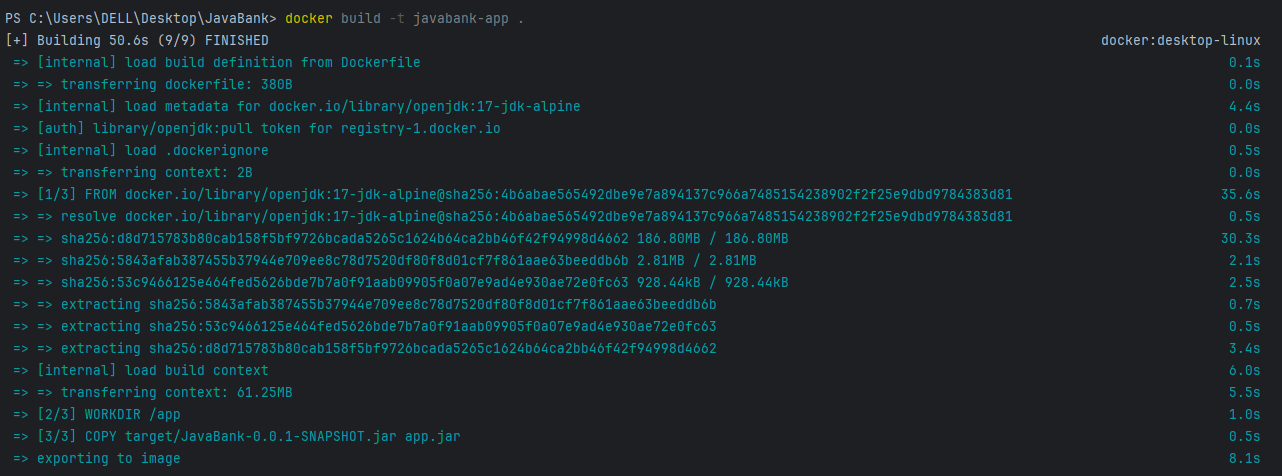
Secondly created docker-compose.yml file in root folder

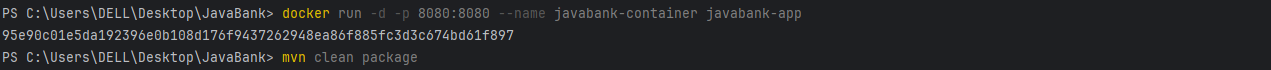
Then run docker build :  
docker build -t javabank-app .

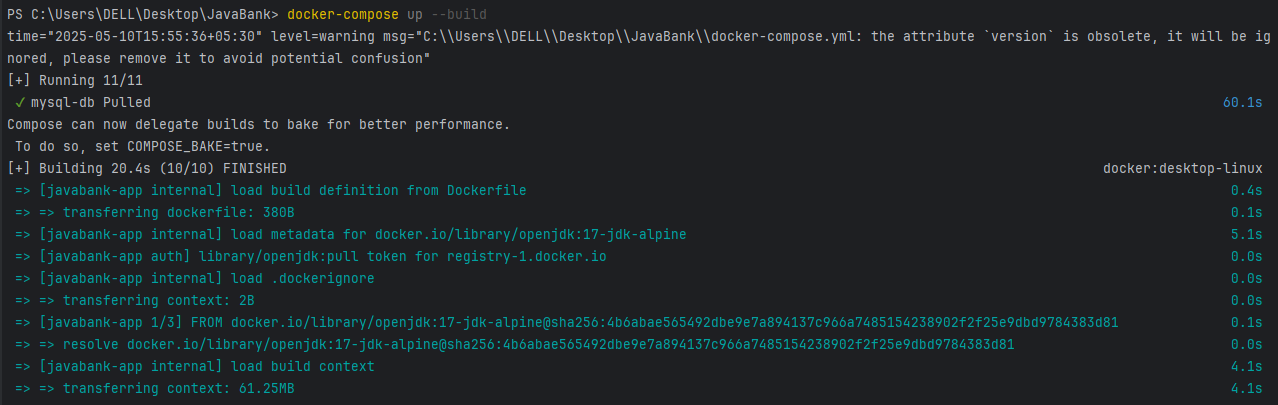
docker run -d -p 8080:8080 --name javabank-container javabank-app

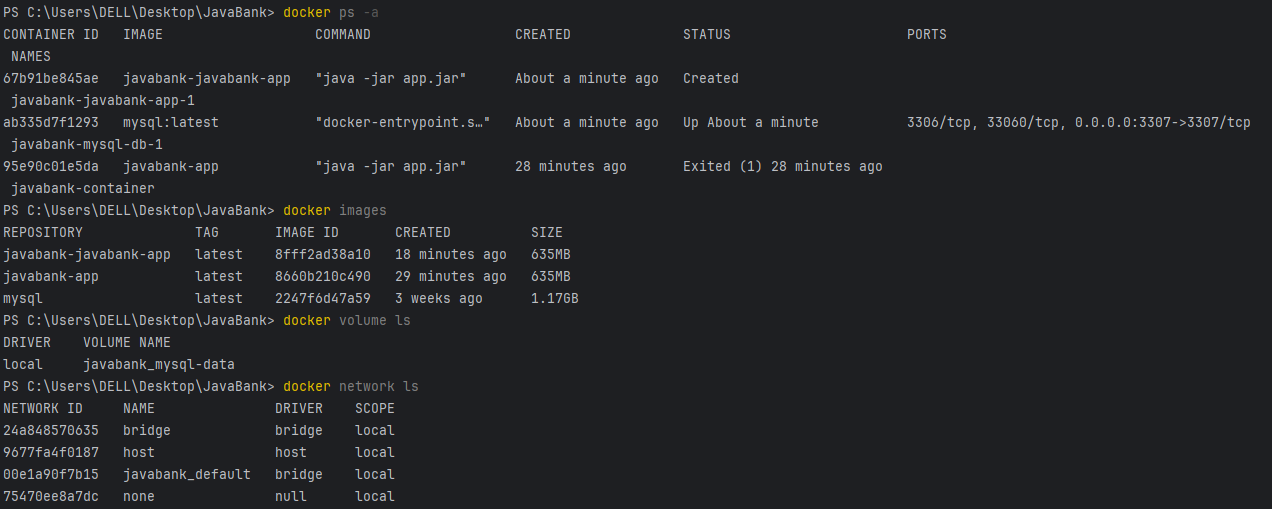
mvn clean package

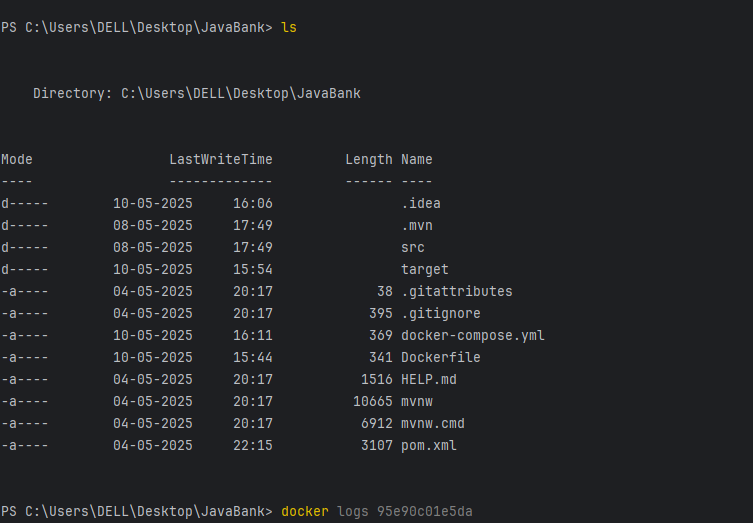
docker-compose up –build



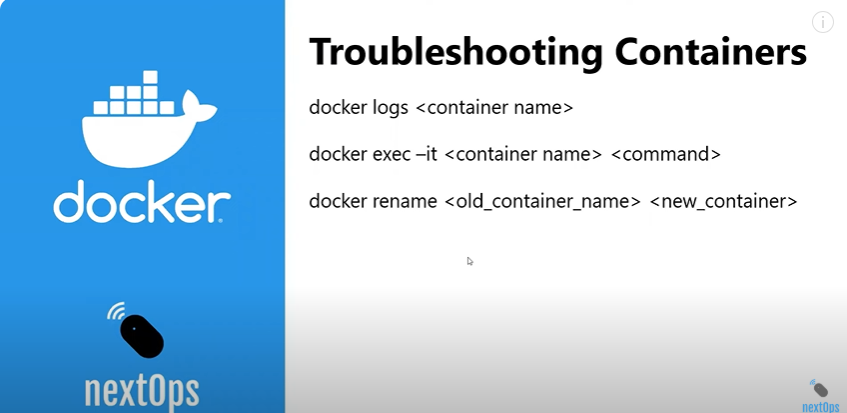


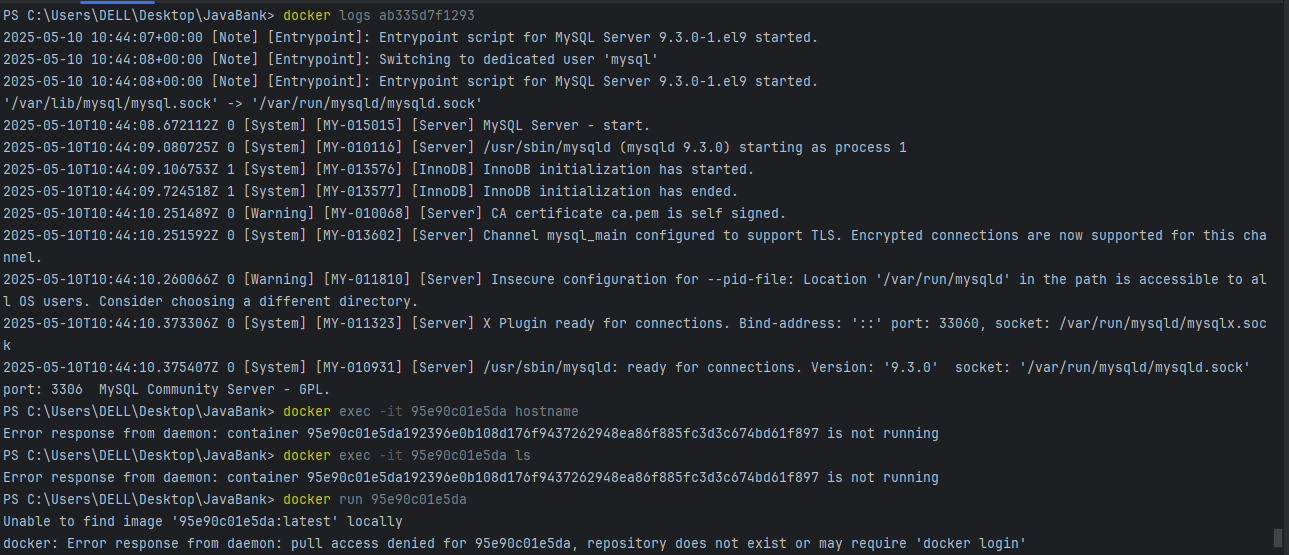




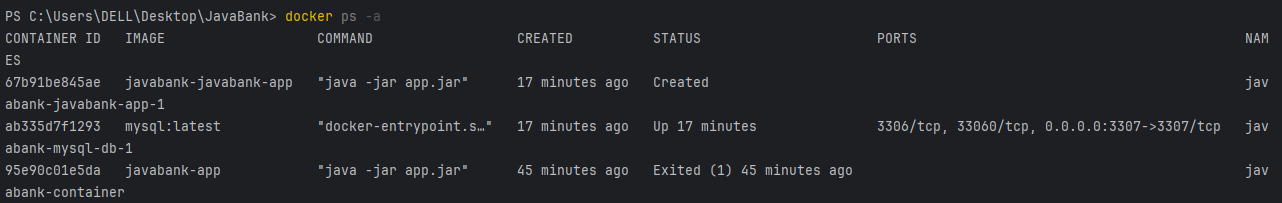


DOCKER TROUBLESHOOTING :

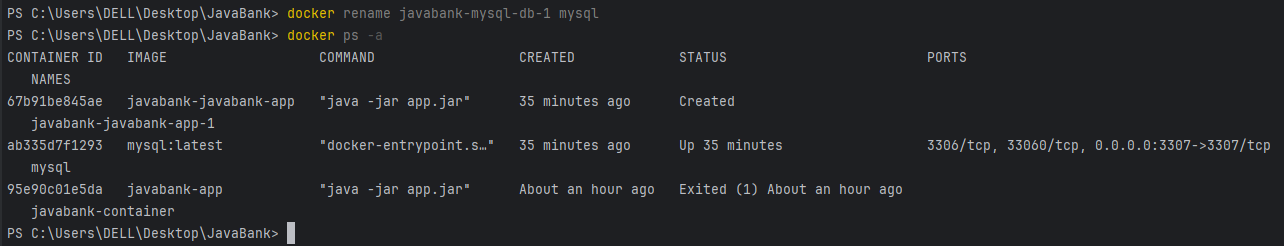


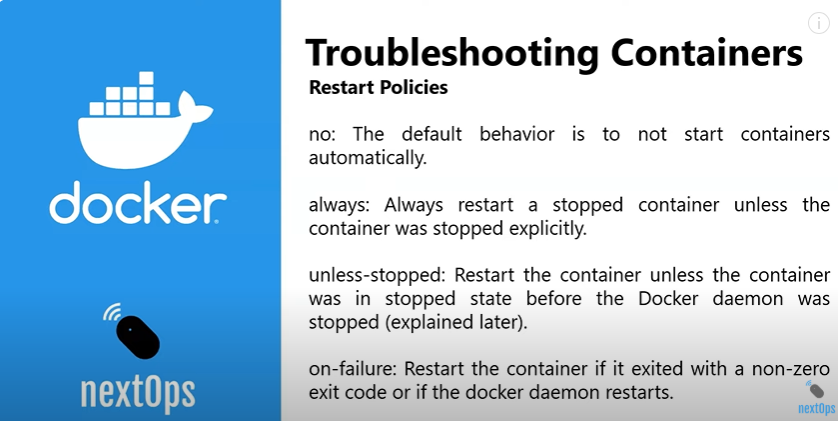


Before renaming:



After renaming:





troubleshooting containers

**ChatGPT said:**

To troubleshoot Docker containers effectively, here are key steps and commands you can use:

**🔍 1. List Containers**

See all running and stopped containers:

bash

CopyEdit

docker ps -a

**📄 2. View Logs**

To check logs for a container (replace container\_id or name):

bash

CopyEdit

docker logs <container\_id>

For continuous real-time logs:

bash

CopyEdit

docker logs -f <container\_id>

**🛠️ 3. Inspect Container Details**

Check configuration, environment, network, volume bindings:

bash

CopyEdit

docker inspect <container\_id>

**🧪 4. Exec into Running Container**

Run a shell inside the container:

bash

CopyEdit

docker exec -it <container\_id> /bin/bash

# or for Alpine/BusyBox-based containers

docker exec -it <container\_id> sh

**🛑 5. Restart / Stop / Remove Containers**

Stop:

bash

CopyEdit

docker stop <container\_id>

Restart:

bash

CopyEdit

docker restart <container\_id>

Remove (must stop first if running):

bash

CopyEdit

docker rm <container\_id>

**⚙️ 6. Check Container’s Port Bindings**

To ensure your ports are mapped correctly:

bash

CopyEdit

docker port <container\_id>

**🔄 7. Check if Database Container is Reachable**

If your app fails to connect to MySQL:

bash

CopyEdit

docker exec -it <container\_id> mysql -u root -p

Or test network reachability:

bash

CopyEdit

docker exec -it <app\_container> ping <db\_container\_name>

Sure! Here’s an **end-to-end explanation** of how to **troubleshoot Docker containers**, especially in a setup like yours where you're running a Spring Boot app with a MySQL database in Docker.

**✅ Scenario Overview**

You have:

* A **Spring Boot app** container.
* A **MySQL container**.
* A **Docker network** and **volume**.
* Issues like port conflicts or app failing to connect to the DB.

**🧩 1. Check All Docker Resources**

Before troubleshooting, list all active Docker resources.

bash

CopyEdit

docker ps -a # See all containers (running or stopped)

docker images # List all images

docker network ls # List all networks

docker volume ls # List all volumes

**🧪 2. Inspect the Problematic Container**

Get detailed configuration and status.

bash

CopyEdit

docker inspect <container\_id\_or\_name>

Look for:

* Ports – are they exposed and bound correctly?
* NetworkMode – is the container attached to the right network?
* Mounts – are volumes correctly mounted?

**📄 3. Check Container Logs**

Check logs to understand errors (especially important for Spring Boot or DB containers):

bash

CopyEdit

docker logs <container\_id>

For continuous logs:

bash

CopyEdit

docker logs -f <container\_id>

**Example:**

If your app can't connect to MySQL, logs might show:

pgsql

CopyEdit

Communications link failure

The driver has not received any packets from the server.

This usually means:

* MySQL container isn't running.
* App can't resolve MySQL container name.
* Network misconfiguration.

**🔍 4. Ping or Connect Between Containers**

Exec into the **app container** and ping the **MySQL container**:

bash

CopyEdit

docker exec -it <app\_container> /bin/bash

ping <mysql\_container\_name>

Or test the DB login:

bash

CopyEdit

mysql -u root -p -h <mysql\_container\_name> -P 3306

**⚙️ 5. Restart or Remove Faulty Containers**

If a container is stuck or misbehaving:

bash

CopyEdit

docker restart <container\_id>

docker stop <container\_id>

docker rm <container\_id>

You may need to **force remove** a container:

bash

CopyEdit

docker rm -f <container\_id>

**🧼 6. Free Up Ports (Optional)**

If you're getting:

swift

CopyEdit

bind: Only one usage of each socket address (protocol/network address/port) is normally permitted

It means another container is already using the port.

Either:

* Stop the existing container:

bash

CopyEdit

docker stop <old\_container>

docker rm <old\_container>

* Or use a different port (if you're okay with changing it):

bash

CopyEdit

-p 3307:3306

**🔁 7. Rebuild and Re-run Containers**

If you've updated code or configurations, rebuild your image:

bash

CopyEdit

mvn clean package # If using Maven

docker build -t myapp .

docker run --name myapp-container -p 8080:8080 --network bank-api myapp

For MySQL:

bash

CopyEdit

docker run --name mysql-db \

-e MYSQL\_ROOT\_PASSWORD=Lavanya@1410 \

-e MYSQL\_DATABASE=javabank \

-v mysql-data:/var/lib/mysql \

--network bank-api \

-p 3306:3306 \

-d mysql:latest

**🗑️ 8. Remove Images/Containers/Volumes (Optional)**

Remove image:

bash

CopyEdit

docker rmi <image\_id>

Force-remove if still referenced:

bash

CopyEdit

docker rm -f <container\_id>

docker rmi -f <image\_id>

**🎯 Common Gotchas**

| **Problem** | **Cause** | **Fix** |
| --- | --- | --- |
| Port already in use | Another container uses it | docker ps, stop & remove old |
| App can't connect to DB | Wrong hostname / no network | Use --network bank-api, connect via container name |
| DB init fails | Volume corruption or config error | Remove volume, recreate it |
| Container restarts | Bad config, missing files | Check docker logs, fix Dockerfile or volumes |

Bottom of Form

**Complete Docker Command Cheat Sheet :**

**General commands :**

| **Task** | **Command** |
| --- | --- |
| Check Docker version | docker --version |
| Check Docker info | docker info |
| Get Docker help | docker help |
| **Working with images :** |  |
| Task | Command |
| List images | docker images |
| Pull image | docker pull <image> |
| Build image from Dockerfile | docker build -t <name> . |
| Tag an image | docker tag <image> <new\_name> |
| Remove image | docker rmi <image> |
| Remove all images | docker rmi $(docker images -q) |

**Working with containers:**

| **Task** | **Command** |
| --- | --- |
| List running containers | docker ps |
| List all containers (incl. stopped) | docker ps -a |
| Start container | docker start <container> |
| Stop container | docker stop <container> |
| Restart container | docker restart <container> |
| Pause container | docker pause <container> |
| Unpause container | docker unpause <container> |
| Remove container | docker rm <container> |
| Remove all containers | docker rm $(docker ps -aq) |
| Run container | docker run <image> |
| Run with port mapping | docker run -p 8080:80 <image> |
| Run with volume | docker run -v <host\_dir>:<container\_dir> <image> |
| Run in interactive mode | docker run -it <image> bash |
| Run in detached mode | docker run -d <image> |
| View container logs | docker logs <container> |
| Exec command inside container | docker exec -it <container> bash |

Working with volumes :

| **Task** | **Command** |
| --- | --- |
| List volumes | docker volume ls |
| Create volume | docker volume create <name> |
| Inspect volume | docker volume inspect <name> |
| Remove volume | docker volume rm <name> |
| Remove all volumes | docker volume rm $(docker volume ls -q) |

**Working with networks:**

| **Task** | **Command** |
| --- | --- |
| List networks | docker network ls |
| Create network | docker network create <name> |
| Inspect network | docker network inspect <name> |
| Remove network | docker network rm <name> |
| Remove all networks | docker network rm $(docker network ls -q) |

**Docker system cleanup :**

| **Task** | **Command** |
| --- | --- |
| Remove all stopped containers | docker container prune |
| Remove all unused images | docker image prune |
| Remove all unused volumes | docker volume prune |
| Remove all unused networks | docker network prune |
| Remove all unused data (containers, images, volumes, networks) | docker system prune |
| Remove everything (including unused volumes) | docker system prune -a --volumes |

**Docker compose :**

| **Task** | **Command** |
| --- | --- |
| Start services | docker-compose up |
| Start in detached mode | docker-compose up -d |
| Stop services | docker-compose down |
| Build services | docker-compose build |
| View logs | docker-compose logs |
| Execute command in service | docker-compose exec <service> bash |