Docker 3

11 November 2025 15:23

Dockerfile creation: Create a simple Dockerfile to build a custom image with the following specifications:
Base image: Ubuntu
Install packages: curl, vim, and git
Set an environment variable: MY_NAME=Your_Name
Build the custom image using docker build and list all available images using docker images.

```
root@DESKTOP-K4UBP7Q:~# mkdir my_ubuntu
root@DESKTOP-K4UBP7Q:~# cd my_ubuntu/
```

```
root@DESKTOP-K4UBP7Q:~/my_ubuntu# vi Dockerfile
```

```
# Use Ubuntu as base image
FROM ubuntu:latest

# Install required packages
RUN apt-get update && apt-get install -y curl vim git && apt-get clean

# Set environment variable
ENV MY_NAME=Vinayak_Deokar

# Default command
CMD ["bash"]
~
```

```
root@DESKTOP-K4UBP7Q:~/my_ubuntu# docker build -t my_ubuntu:1.0 .
[+] Building 0.3s (6/6) FINISHED                                                                                docker:default
 => [internal] load build definition from Dockerfile                                                                      0.1s
 => => transferring dockerfile: 273B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest                                                          0.0s
 => [internal] load .dockerignore                                                                                         0.1s
 => => transferring context: 2B                                                                                           0.0s
 => [1/2] FROM docker.io/library/ubuntu:latest                                                                            0.0s
 => CACHED [2/2] RUN apt-get update && apt-get install -y curl vim git && apt-get clean                                   0.0s
 => exporting to image                                                                                                    0.0s
 => => exporting layers                                                                                                   0.0s
 => => writing image sha256:a4c0c2fe2bf40765de9a6d01dcfb6ce83a7f73ed56bf8929993127b24c85173a                              0.0s
 => => naming to docker.io/library/my_ubuntu:1.0                                                                          0.0s
```

```
root@DESKTOP-K4UBP7Q:~/my_ubuntu# docker images
REPOSITORY                                                      TAG          IMAGE ID       CREATED         SIZE
630493028887.dkr.ecr.ap-south-1.amazonaws.com/mywebapp          latest       a575768357c0   4 days ago      133MB
mywebapp                                                        latest       a575768357c0   4 days ago      133MB
awsecr_image                                                    latest       611d7c78e63b   4 days ago      241MB
public.ecr.aws/b0r7d3x2/kaivalyabachkar/apache2_ecr             vinayak      611d7c78e63b   4 days ago      241MB
vinayakdeokar/ecr_custom                                        latest       5cc65480b008   4 days ago      241MB
vinayakdeokar/dhub_custom                                       latest       796308b8ba44   4 days ago      241MB
vinayakdeokar/dhub_image                                        latest       f6cbc5329ff2   4 days ago      241MB
dhub_image                                                      latest       f6cbc5329ff2   4 days ago      241MB
test_ubuntu                                                     1.0          b7d550dce105   4 days ago      289MB
my_ubuntu                                                       1.0          a4c0c2fe2bf4   4 days ago      289MB
ubuntu                                                          1.0          a4c0c2fe2bf4   4 days ago      289MB
```

Docker networking: Create two Docker containers with different names, and add them to the same custom network. Verify that the containers can communicate with each other using their names as hostnames.

```
root@DESKTOP-K4UBP7Q:~# docker network create my_network
64b4697b0ec70ebe1d156a4af37e14d6b30a5969507aaccacfcf5a7e4fb2f2ac
```

```
root@DESKTOP-K4UBP7Q:~# docker network ls
NETWORK ID     NAME         DRIVER    SCOPE
829ad7d89f3e   bridge       bridge    local
7463eff9a920   host         host      local
64b4697b0ec7   my_network   bridge    local
a1d3c8cc0c47   none         null      local
```

```
root@DESKTOP-K4UBP7Q:~# docker run -itd --name container_1 --network my_network ubuntu bash
fb572d0174ba905ca1311a4dd5a26e052e4c48c67ec83fa52f0e66684f270af2
root@DESKTOP-K4UBP7Q:~# docker run -itd --name container_2 --network my_network ubuntu bash
c74b9c819db25dae33fbc787b2b5e27aa27b86d21534e6694e80b882dd4f0938
```

```
root@DESKTOP-K4UBP7Q:~# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
c74b9c819db2   ubuntu    "bash"    15 seconds ago   Up 15 seconds             container_2
fb572d0174ba   ubuntu    "bash"    22 seconds ago   Up 22 seconds             container_1
```

```
root@DESKTOP-K4UBP7Q:~# docker exec -it container_1 bash
root@fb572d0174ba:/# apt-get update && apt-get install -y iputils-ping
```

apt-get update && apt-get install -y iputils-ping (to run the ping command)

If not install above than getting error command not found

```
root@c74b9c819db2:/# ping -c 3 container_1
bash: ping: command not found
root@c74b9c819db2:/#
```

```
root@fb572d0174ba:/# ping -c 3 container_2
PING container_2 (172.18.0.3) 56(84) bytes of data.
64 bytes from container_2.my_network (172.18.0.3): icmp_seq=1 ttl=64 time=7.63 ms
64 bytes from container_2.my_network (172.18.0.3): icmp_seq=2 ttl=64 time=0.057 ms
64 bytes from container_2.my_network (172.18.0.3): icmp_seq=3 ttl=64 time=0.092 ms

--- container_2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2011ms
rtt min/avg/max/mdev = 0.057/2.591/7.626/3.559 ms
root@fb572d0174ba:/#
```

Data persistence with volumes: Run a Docker container with an attached volume, create a file inside the volume, and verify that the file persists after the container is stopped and removed.

Createe volume

```
root@DESKTOP-K4UBP7Q:~# docker volume create my_volume
my_volume
```

```
root@DESKTOP-K4UBP7Q:~# docker volume ls
DRIVER    VOLUME NAME
local     my_volume
```

Created 1st container with my_volume

```
root@DESKTOP-K4UBP7Q:~# docker run -it --name volume_container -v my_volume:/data ubuntu bash
root@aa5b2903da46:/# ll
```

In that container we created one file and save

```
root@aa5b2903da46:/# cd /data
root@aa5b2903da46:/data# echo "This is persistent data" > persistent.txt
root@aa5b2903da46:/data# cat persistent.txt
This is persistent data
```

After that we delete that 1st container

```
root@DESKTOP-K4UBP7Q:~# docker rm volume_container
volume_container
```

Creating new 2nd conatiner volume2_conatiner
and in that we can see the file that we saved in 1st container

```
root@DESKTOP-K4UBP7Q:~# docker run -it --name volume2_container -v my_volume:/data ubuntu bash
root@84af65e1af55:/# cd /data/
root@84af65e1af55:/data# ll
total 12
drwxr-xr-x 2 root root 4096 Nov 11 10:10 ./
drwxr-xr-x 1 root root 4096 Nov 11 10:11 ../
-rw-r--r-- 1 root root   24 Nov 11 10:10 persistent.txt
root@84af65e1af55:/data# cat persistent.txt
This is persistent data
```

Docker Compose: Create a docker-compose.yml file to run a multi-container application consisting of a web server (e.g., Nginx) and a database server (e.g., MySQL). Ensure that both containers are connected to the same network.

```
root@DESKTOP-K4UBP7Q:~# mkdir my_composeapp
```

```
root@DESKTOP-K4UBP7Q:~# cd my_composeapp/
root@DESKTOP-K4UBP7Q:~/my_composeapp# vi docker-compose.yml
```

```yaml
version: '3.8'

services:
  web:
    image: nginx:latest
    container_name: nginx_server
    ports:
      - "8080:80"
    networks:
      - my_network
    depends_on:
      - db

  db:
    image: mysql:latest
    container_name: mysql_server
    environment:
      MYSQL_ROOT_PASSWORD: root123
      MYSQL_DATABASE: mydb
      MYSQL_USER: user1
      MYSQL_PASSWORD: pass123
    networks:
      - my_network

networks:
  my_network:
    driver: bridge
```

```
root@DESKTOP-K4UBP7Q:~/my_composeapp# sudo apt update
sudo apt install docker-compose -y
```

```
root@DESKTOP-K4UBP7Q:~/my_composeapp# docker-compose up -d
Creating network "my_composeapp_my_network" with driver "bridge"
Pulling db (mysql:latest)...
latest: Pulling from library/mysql
023a182c62a0: Pull complete
f5f78fcd9ccb: Pull complete
494c372d15c3: Pull complete
dcee80f7340c: Pull complete
480d01bd7a6a: Pull complete
834e15e3ed24: Pull complete
c276de9b5571: Pull complete
0cd145fbb449: Pull complete
5a3f7744d0e7: Pull complete
21aa606d8d58: Pull complete
Digest: sha256:569c4128dfa625ac2ac62cdd8af588a3a6a60a049d1a8d8f0fac95880ecdbbe5
Status: Downloaded newer image for mysql:latest
Creating mysql_server ... done
Creating nginx_server ... done
```

```
root@DESKTOP-K4UBP7Q:~/my_composeapp# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                          PORTS
            NAMES
d2109c9469b0   nginx:latest   "/docker-entrypoint.…"   17 seconds ago   Up 16 seconds                   0.0.0.0:8080->80/tcp, [::]:80
80->80/tcp   nginx_server
3e0c33f3dad9   mysql:latest   "docker-entrypoint.s…"   18 seconds ago   Up 17 seconds                   3306/tcp, 33060/tcp
            mysql_server
8b4a060e3229   busybox        "sh -c 'exit 1'"         5 days ago       Restarting (1) 30 seconds ago
            test_fail
```

Go to the container and install the ping packages - **apt-get update && apt-get install -y iputils-ping**
Than docker exec -it nginx_server ping mysql_server
Or we can do it by another way that is go to one container and ping form that container to another container.
But on both container ping packages needs to be install

```
root@DESKTOP-K4UBP7Q:~/my_composeapp#  docker exec -it nginx_server ping mysql_server
PING mysql_server (172.19.0.2) 56(84) bytes of data.
64 bytes from mysql_server.my_composeapp_my_network (172.19.0.2): icmp_seq=1 ttl=64 time=0.332 ms
64 bytes from mysql_server.my_composeapp_my_network (172.19.0.2): icmp_seq=2 ttl=64 time=0.233 ms
64 bytes from mysql_server.my_composeapp_my_network (172.19.0.2): icmp_seq=3 ttl=64 time=0.156 ms
64 bytes from mysql_server.my_composeapp_my_network (172.19.0.2): icmp_seq=4 ttl=64 time=0.178 ms
64 bytes from mysql_server.my_composeapp_my_network (172.19.0.2): icmp_seq=5 ttl=64 time=0.124 ms
64 bytes from mysql_server.my_composeapp_my_network (172.19.0.2): icmp_seq=6 ttl=64 time=0.122 ms
```

Container logging: Run a Docker container that generates log output, and practice using docker logs to view and analyze the logs.

docker run -d --name log_container busybox sh -c "while true; do echo 'This is a log message from a Docker container'; sleep 2; done"

```
root@DESKTOP-K4UBP7Q:~# docker run -d --name log_container busybox sh -c "while true; do echo 'This is a log message from a Docker co
ntainer'; sleep 2; done"
c79b6823adc8c4b85205bffbe1cbfa0b2048f1122b7c628262d3d30eec1635b7
```

```
^Croot@DESKTOP-K4UBP7Q:~# docker logs log_container
This is a log message from a Docker container
This is a log message from a Docker container
This is a log message from a Docker container
This is a log message from a Docker container
```

docker logs --tail 5 log_container - we can see last 5 logs with this
docker logs -f log_container - we can see live logs with this

Updating a containerized application: Update the source code of a simple containerized application, rebuild the Docker image, and deploy the updated version

```
root@DESKTOP-K4UBP7Q:~# docker run -d --name stress_container busybox sh -c "while true; do echo 'Using CPU...'; done"
6fe8eb2e07e7436bab9796e40eea07c0ad5512979a54bbabb3f84fc1b2795637
root@DESKTOP-K4UBP7Q:~# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                          PORTS
          NAMES
6fe8eb2e07e7   busybox        "sh -c 'while true; …"   6 seconds ago    Up 5 seconds
          stress_container
```

Docker stats

```
CONTAINER ID   NAME              CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O         PIDS
ef098fc9aa99   stress_container   74.12%    472KiB / 7.603GiB     0.01%     656B / 126B       0B / 0B           1
d2109c9469b0   nginx_server      0.00%     15.27MiB / 7.603GiB   0.20%     10.2MB / 78.7kB   36.8MB / 32.7MB   13
3e0c33f3dad9   mysql_server      0.56%     428.9MiB / 7.603GiB   5.51%     4.05kB / 1.76kB   71.8MB / 293MB    37
8b4a060e3229   test_fail         0.00%     0B / 0B               0.00%     0B / 0B           0B / 0B           0
13050d79cc31   httpd_no          0.01%     8.84MiB / 7.603GiB    0.11%     3.08kB / 126B     6.38MB / 4.1kB    82
```

```
root@DESKTOP-K4UBP7Q:~# docker top stress_container
UID                 PID                 PPID                C                   STIME
       CMD
root                34025               34001               79                  16:37
       sh -c while true; do echo 'Using CPU...'; done
root@DESKTOP-K4UBP7Q:~#
```

Updating a containerized application: Update the source code of a simple containerized application, rebuild the Docker image, and deploy the updated version while minimizing downtime

Fristly created diretory and create 1 file

```
root@DESKTOP-K4UBP7Q:~# mkdir my_app
root@DESKTOP-K4UBP7Q:~# cd my_app/
root@DESKTOP-K4UBP7Q:~/my_app# echo "<h1>Hello Vinayak - Version 1</h1>" > index.html
```

```
root@DESKTOP-K4UBP7Q:~/my_app# vi Dockerfile
```

```
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/index.html
~
```

Build that image

```
root@DESKTOP-K4UBP7Q:~/my_app# docker build -t mywebapp:versoin_1 .
[+] Building 14.1s (8/8) FINISHED                                                                                docker:default
 => [internal] load build definition from Dockerfile                                                                      0.2s
 => => transferring dockerfile: 105B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine                                                           4.6s
 => [auth] library/nginx:pull token for registry-1.docker.io                                                              0.0s
 => [internal] load .dockerignore                                                                                         0.1s
 => => transferring context: 2B                                                                                           0.0s
 => [internal] load build context                                                                                         0.1s
 => => transferring context: 72B                                                                                          0.0s
 => [1/2] FROM docker.io/library/nginx:alpine@sha256:b3c656d55d7ad751196f21b7fd2e8d4da9cb430e32f646adcf92441b72f82b14     8.6s
```

Than create container and run

```
root@DESKTOP-K4UBP7Q:~/my_app# docker run -d --name webapp_v1 -p 8080:80 mywebapp:versoin_1
113b9838e43c9f13f6ef5ed5c14ca29ec9d72489a7f2609695ee9a74f9f6d39f
```

Then browser serch http://localhost8080
We getting this that's means

Hello Vinayak - Version 1

Now we update in that file index.html (update)

```
root@DESKTOP-K4UBP7Q:~/my_app# echo "<h1>Hello Vinayak - Version 2 (Updated)</h1>" > index.html
```

Then build that image

```
root@DESKTOP-K4UBP7Q:~/my_app# docker build -t mywebapp:version_2 .
[+] Building 2.9s (8/8) FINISHED                                                                                docker:default
 => [internal] load build definition from Dockerfile                                                                      0.1s
 => => transferring dockerfile: 105B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine                                                           2.4s
 => [auth] library/nginx:pull token for registry-1.docker.io                                                              0.0s
 => [internal] load .dockerignore                                                                                         0.0s
 => => transferring context: 2B                                                                                           0.0s
 => [internal] load build context                                                                                         0.0s
 => => transferring context: 82B                                                                                          0.0s
 => CACHED [1/2] FROM docker.io/library/nginx:alpine@sha256:b3c656d55d7ad751196f21b7fd2e8d4da9cb430e32f646adcf92441b72f82b14    0.0s
 => [2/2] COPY index.html /usr/share/nginx/html/index.html                                                                0.1s
 => exporting to image                                                                                                    0.1s
 => => exporting layers                                                                                                   0.1s
 => => writing image sha256:63e419bbc6c86cfa083fd3e5d6191fc1b297f4f0807013f44338064dc0eb5efc                              0.0s
 => => naming to docker.io/library/mywebapp:version_2                                                                     0.0s
```

And then create and run new container

```
root@DESKTOP-K4UBP7Q:~/my_app# docker run -d --name webapp_v2 -p 8081:80 mywebapp:version_2
139ff77acfff12847f50e5767a434847ce9bda02a15b05e9672361f5a399fd2d
root@DESKTOP-K4UBP7Q:~/my_app#
```

Now check on browser http://localhost8081

Hello Vinayak - Version 2 (Updated)

We stop the old container that port is 8081

```
root@DESKTOP-K4UBP7Q:~/my_app# docker stop webapp_v1
webapp_v1
root@DESKTOP-K4UBP7Q:~/my_app# docker rm webapp_v1
webapp_v1
```

Creating new container and give port no 8080

```
root@DESKTOP-K4UBP7Q:~/my_app# docker run -d --name webapp_v2_live -p 8080:80 mywebapp:version_2
7f86a7fc1c7c2a6de0a1d931592bdbc44c6c3d491f2a334a6dd18dc4c2e64e1a
```

We first tested the new version on a different port (8081),
then safely replaced the old version (8080) — without any downtime.

Hello Vinayak - Version 2 (Updated)

This process is known as Rolling Update or Zero Downtime Deployment

Bridge network: Create two Docker containers using different images, such as NGINX and MySQL. Connect them using a user-defined bridge network and ensure they can communicate with each other.

```
root@DESKTOP-K4UBP7Q:~# docker network create my_bridge_network
ed3d24885a3a485f746d5fce9a0f5910c4628f13de2909171ec86fd66e5a2a09
root@DESKTOP-K4UBP7Q:~# docker network ls
NETWORK ID     NAME                   DRIVER    SCOPE
935ed0c7de4d   bridge                 bridge    local
7463eff9a920   host                   host      local
ed3d24885a3a   my_bridge_network      bridge    local
```

```
root@DESKTOP-K4UBP7Q:~# docker run -d --name my_mysql --network my_bridge_network -e MYSQL_ROOT_PASSWORD=root@123 mysql:latest
4cc470e6a96faca9eef22b455626f6696a7aa7d559d4bfe50d605b4b92d1f03e
```

```
root@DESKTOP-K4UBP7Q:~# docker run -d --name my_nginx --network my_bridge_network nginx:latest
6a20f3a97661c8e1cde4e9fcad517a220f4999bf141ad9de39af91a71f51ed8b
```

Go to my_ngnix container

```
root@DESKTOP-K4UBP7Q:~# docker exec -it my_nginx bash
```

```
root@6a20f3a97661:/# ping my_mysql
PING my_mysql (172.20.0.2) 56(84) bytes of data.
64 bytes from my_mysql.my_bridge_network (172.20.0.2): icmp_seq=1 ttl=64 time=1.75 ms
64 bytes from my_mysql.my_bridge_network (172.20.0.2): icmp_seq=2 ttl=64 time=0.127 ms
64 bytes from my_mysql.my_bridge_network (172.20.0.2): icmp_seq=3 ttl=64 time=0.084 ms
64 bytes from my_mysql.my_bridge_network (172.20.0.2): icmp_seq=4 ttl=64 time=0.152 ms
64 bytes from my_mysql.my_bridge_network (172.20.0.2): icmp_seq=5 ttl=64 time=0.094 ms
```

Now run mysql container

```
root@DESKTOP-K4UBP7Q:~# docker exec -it my_mysql bash
```

Install ping packages

```
bash-5.1# microdnf install -y iputils
Package
Installing:
 iputils-20210202-11.0.1.el9_6.3.x86_64
Transaction Summary:
 Installing:        1 packages
 Reinstalling:      0 packages
 Upgrading:         0 packages
 Obsoleting:        0 packages
 Removing:          0 packages
 Downgrading:       0 packages
Running transaction test...
Installing: iputils;20210202-11.0.1.el9_6.3;x86_64;ol9_baseos_latest
Complete.
```

```
bash-5.1# ping my_nginx
PING my_nginx (172.20.0.3) 56(84) bytes of data.
64 bytes from my_nginx.my_bridge_network (172.20.0.3): icmp_seq=1 ttl=64 time=0.168 ms
64 bytes from my_nginx.my_bridge_network (172.20.0.3): icmp_seq=2 ttl=64 time=0.083 ms
64 bytes from my_nginx.my_bridge_network (172.20.0.3): icmp_seq=3 ttl=64 time=0.110 ms
64 bytes from my_nginx.my_bridge_network (172.20.0.3): icmp_seq=4 ttl=64 time=0.052 ms
64 bytes from my_nginx.my_bridge_network (172.20.0.3): icmp_seq=5 ttl=64 time=0.355 ms
```

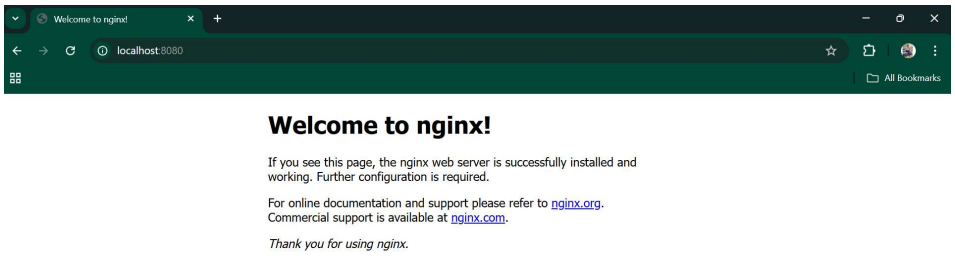
Port mapping: Run a containerized web server (e.g., NGINX or Apache) and map its default port (80 or 443) to a custom port on the host machine. Verify that the web server is accessible through the custom port on the host.

```
root@DESKTOP-K4UBP7Q:~# docker run -d --name my_nginx -p 8080:80 nginx
a788eb0db3cb3960966db3638d14f95f095228d7fd51e0c116a4a0b35c5f3a5f
```

```
root@DESKTOP-K4UBP7Q:~# curl http://localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@DESKTOP-K4UBP7Q:~#
```



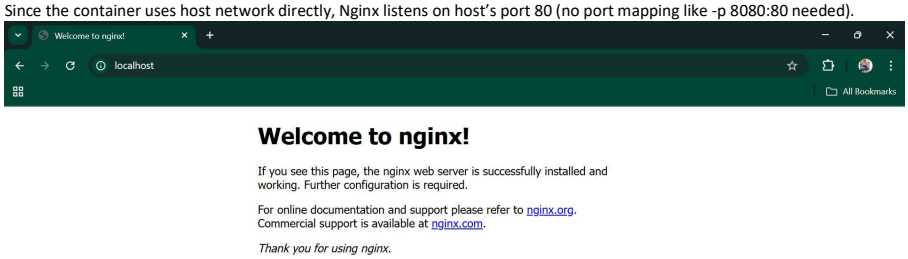
Host networking: Run a container with the host networking mode and compare its network configuration with that of the host machine. Explain the advantages and disadvantages of using host networking for containers.

Normally, Docker containers run in bridge mode — i.e. each container gets its own private IP and communicates with the host via NAT (Network Address Translation).But when you use host networking, the container shares the host's network stack directly.

```
root@DESKTOP-K4UBP7Q:~# docker run -d --name host_net_container --network host nginx
92e2031d7d16d6f18375d4b28b79a4e6dd77fdeb5afbde78fc7e5a093e2d049f
```

```
root@DESKTOP-K4UBP7Q:~# docker inspect host_net_container | grep -i "IPAddress"
                    "IPAddress": "",
```

Since the container uses host network directly, Nginx listens on host's port 80 (no port mapping like -p 8080:80 needed).



11.Named volume: Launch a containerized database (e.g., MySQL or PostgreSQL) using a named volume to store its data. Stop and remove the container, then recreate it using the same named volume. Verify that the data persists across container recreations.

```
root@DESKTOP-K4UBP7Q:~# docker volume create mysql_data
mysql_data
root@DESKTOP-K4UBP7Q:~# docker volume ls
DRIVER    VOLUME NAME
local     5ca8171bec34162e46ced757a3c024aa5c1d9c6b5e811760c14fcb70be695a6e
local     22d56b60a37eebd1f392cd999912450826abb931b6a0eea66b64b805187a619e
local     my_volume
local     mysql_data
```

Creating container 1 with name **my_mysql_db**

```
root@DESKTOP-K4UBP7Q:~# docker run -d --name my_mysql_db -e MYSQL_ROOT_PASSWORD=root@123 -v mysql_data:/var/lib/mysql mysql:latest
acb8ccdf169b7ac0b471a01555c433dc801e7d3dcd6787ecd2caa99d711ca350
```

```
root@DESKTOP-K4UBP7Q:~# docker exec -it my_mysql_db mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.5.0 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> create database vinayakdb;
Query OK, 1 row affected (0.021 sec)
```

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| vinayakdb          |
+--------------------+
```

```
mysql> exit
Bye
```

Then remove this container

```
root@DESKTOP-K4UBP7Q:~# docker stop my_mysql_db
my_mysql_db
root@DESKTOP-K4UBP7Q:~# docker rm my_mysql_db
my_mysql_db
```

And create new container

Creating container 2 with name **my_mysql_db_new**

```
root@DESKTOP-K4UBP7Q:~# docker run -d --name my_mysql_db_new -e MYSQL_ROOT_PASSWORD=root@123 -v mysql_data:/var/lib/mysql mysql:lates
t
4c3a8074c54fa175e75271ccc5f8bc09859e06221b9198ed558c607f7384c864
```

```
root@DESKTOP-K4UBP7Q:~# docker exec -it my_mysql_db_new mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 9.5.0 MySQL Community Server - GPL
```

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| vinayakdb          |
+--------------------+
5 rows in set (0.012 sec)

mysql>
```

A named volume in Docker allows data to persist even after a container is removed. By mounting a volume (e.g., mysql_data:/var/lib/mysql) to a database container, the data remains stored outside the container. When you recreate the container using the same named volume, all previous data is retained.

Bind mount: Create a simple web application with a local directory containing HTML, CSS, and JavaScript files. Run a web server container (e.g., NGINX or Apache) and bind mount the local directory to the container's webroot. Verify that the web application is accessible and update the local files to see if changes are reflected in the container.

```
root@DESKTOP-K4UBP7Q:~# mkdir my_web_app
root@DESKTOP-K4UBP7Q:~# cd my_web_app/
```

Three files created

```
root@DESKTOP-K4UBP7Q:~/my_web_app# echo "<h1>Hello Vinayak - Bind Mount Demo</h1>" > index.html
root@DESKTOP-K4UBP7Q:~/my_web_app# echo "body { background-color: lightblue; text-align: center; }" > style.css
```

```
root@DESKTOP-K4UBP7Q:~/my_web_app# echo "console.log('Bind mount working\!');" > script.js
```

Run NGINX container with a bind mount

```
root@DESKTOP-K4UBP7Q:~/my_web_app# docker run -d --name web_bind_demo1 -p 8080:80 -v $(pwd):/usr/share/nginx/html nginx
5b2889276f5b5ef0d1257a4a305437c5b01d3f0c7041331b512518e575d86d2f
```

Verify in browser

Hello Vinayak - Bind Mount Demo

Test Live Update

```
root@DESKTOP-K4UBP7Q:~/my_web_app# echo "<h1>Updated Version - Live from Host</h1>" > index.html
```

Updated Version - Live from Host

A bind mount connects a local directory to a container directory, allowing real-time file sharing.
When you edit files on your host, the container instantly reflects those changes — perfect for web development.

Volume backups: Using a container with a named volume, create a backup of the volume's data by either exporting it as a tarball or copying the data to a host directory. Restore the data to a new container and verify that the restoration was successful.

```
root@DESKTOP-K4UBP7Q:~# docker volume create my_data_volume
my_data_volume
```

```
root@DESKTOP-K4UBP7Q:~# docker volume ls
DRIVER    VOLUME NAME
local     5ca8171bec34162e46ced757a3c024aa5c1d9c6b5e811760c14fcb70be695a6e
local     22d56b60a37eebd1f392cd999912450826abb931b6a0eea66b64b805187a619e
local     my_data_volume
```

Run a container and add data

```
root@DESKTOP-K4UBP7Q:~# docker run -it --name vol_demo -v my_data_volume:/data ubuntu bash
root@5e26ca176ce4:/# echo "Vinayak volume backup test" > /data/info.txt
```

Backup the volume

```
root@DESKTOP-K4UBP7Q:~# docker run --rm -v my_data_volume:/data -v $(pwd):/backup ubuntu tar cvf /backup/volume_backup.tar -C /data .
./
./info.txt
```

docker run = This runs a new container.
--rm = Automatically removes the container after the command finishes.We use it here because we don't need a permanent container just for backup.
-v my_data_volume:/data = Mounts the Docker named volume my_data_volume to the container path /data. This lets the container access the data stored in that volume.
-v $(pwd):/backup = Mounts your current host directory (output of $(pwd) is the current path) to /backup inside the container.
This is where the backup tar file will be saved on your host machine.
Ubuntu = Uses the Ubuntu image as a temporary container to run the backup command.
tar cvf /backup/volume_backup.tar -C /data .
/backup/volume_backup.tar = path to save the tar file inside container, which maps to host directory.
-C /data . = go to /data directory (volume) and archive all its contents

Restore backup into the volume properly

```
root@DESKTOP-K4UBP7Q:~# docker run --rm -v my_restored_volume:/data -v $(pwd):/backup ubuntu tar xvf /backup/volume_backup.tar -C /
./
./info.txt
```

Verify the restored data

```
root@DESKTOP-K4UBP7Q:~# docker run -it --rm -v my_restored_volume:/data ubuntu ls /data
info.txt
```

Check file content

```
root@DESKTOP-K4UBP7Q:~# docker run -it --rm -v my_restored_volume:/data ubuntu cat /data/info.txt
Vinayak volume backup test
```

Docker Compose networking: Create a Docker Compose file that defines a multi-container application with a frontend, backend, and database. Set up custom networks and volumes for the services and ensure that they can communicate with each other and store data persistently

Create docker-compose file
Insert

```
root@DESKTOP-K4UBP7Q:~# mkdir docker_compose
root@DESKTOP-K4UBP7Q:~# cd docker_compose/
root@DESKTOP-K4UBP7Q:~/docker_compose# vi docker-compose.yml
```

```
version: '3.9'
```

```
services:
 frontend:
  image: nginx:latest
  container_name: frontend
  ports:
   - "8080:80"
  networks:
   - app_network
  depends_on:
   - backend
  volumes:
   - frontend_data:/usr/share/nginx/html

 backend:
  image: node:18
  container_name: backend
  working_dir: /app
  command: sh -c "while true; do echo 'Backend running'; sleep 5; done"
  networks:
   - app_network
  depends_on:
   - db
  volumes:
   - backend_data:/app

 db:
  image: mysql:8
  container_name: db
  environment:
   MYSQL_ROOT_PASSWORD: root123
   MYSQL_DATABASE: mydb
  networks:
   - app_network
  volumes:
   - db_data:/var/lib/mysql

networks:
 app_network:
  driver: bridge

volumes:
 frontend_data:
 backend_data:
 db_data:
```

Compose services up

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker-compose up -d
```

```
Status: Downloaded newer image for node:18
Creating db ... done
Creating backend ... done
Creating frontend ... done
```

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS                   PORTS
           NAMES
9239b316084d   nginx:latest   "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes             0.0.0.0:8080->80/tcp, [::]:808
0->80/tcp   frontend
d516eda2f27e   node:18        "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes
            backend
9a382289cdeb   mysql:8        "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes             3306/tcp, 33060/tcp
            db
```

Network verify

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker network ls
NETWORK ID     NAME                          DRIVER    SCOPE
89eda97a113c   bridge                        bridge    local
036e231a2a5d   docker_compose_app_network    bridge    local
7463eff9a920   host                          host      local
ed3d24885a3a   my_bridge_network             bridge    local
9e25fd5b4934   my_composeapp_my_network      bridge    local
```

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker network inspect docker_compose_app_network
```

```
    },
    "Containers": {
        "9239b316084d1ef584f8265289f9b3ee97ccd60dd8aaf0d0862f4e72fa44fbfe": {
            "Name": "frontend",
            "EndpointID": "f60f7c7c164e34e4d93831c0b2ceceaf39c281ff2f181db98fe00b36519ceef8",
            "MacAddress": "1a:84:84:1e:f1:3b",
            "IPv4Address": "172.21.0.4/16",
            "IPv6Address": ""
        },
        "9a382289cdeb1aa53099c3028df7c66ddd12a375b0166b76305a7e6ead3a04f7": {
            "Name": "db",
            "EndpointID": "8e30c6c5c9761ed2cbc6de46a2ce752865e46d6e07c3e0bac012e941f8cfd404",
            "MacAddress": "ee:23:85:e6:5b:12",
            "IPv4Address": "172.21.0.2/16",
            "IPv6Address": ""
        },
        "d516eda2f27eaa43b40659cb27c7ac11796ceace6616c5aa4dd52a824921f370": {
            "Name": "backend",
            "EndpointID": "c093ee1b73081b0a0472ea4446cb05428d58f3b70160d07a58e71460360aff2e",
            "MacAddress": "0e:49:10:98:1a:9d",
            "IPv4Address": "172.21.0.3/16",
            "IPv6Address": ""
```

Backend logs check

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker-compose logs backend
Attaching to backend
backend     | Backend running
backend     | Backend running
backend     | Backend running
```

Test frontend → backend ping (we need to install ping packge in that container)

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker exec -it frontend ping backend
PING backend (172.21.0.3) 56(84) bytes of data.
64 bytes from backend.docker_compose_app_network (172.21.0.3): icmp_seq=1 ttl=64 time=0.151 ms
64 bytes from backend.docker_compose_app_network (172.21.0.3): icmp_seq=2 ttl=64 time=0.296 ms
64 bytes from backend.docker_compose_app_network (172.21.0.3): icmp_seq=3 ttl=64 time=0.119 ms
```

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker exec -it backend ping db
PING db (172.21.0.2) 56(84) bytes of data.
64 bytes from db.docker_compose_app_network (172.21.0.2): icmp_seq=1 ttl=64 time=0.824 ms
64 bytes from db.docker_compose_app_network (172.21.0.2): icmp_seq=2 ttl=64 time=0.128 ms
```

Persistent volumes verify

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker exec -it backend bash
root@d516eda2f27e:/app# echo "Backend test file" > /app/test.txt
root@d516eda2f27e:/app# exit
```

Stop and remove backend container

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker-compose stop backend
Stopping backend ... done
root@DESKTOP-K4UBP7Q:~/docker_compose# docker-compose rm backend
Going to remove backend
Are you sure? [yN] y
Removing backend ... done
```

Recreate backend:

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker-compose up -d backend
db is up-to-date
Creating backend ... done
```

Check if file exists

```
root@DESKTOP-K4UBP7Q:~/docker_compose# docker exec -it backend ls /app
test.txt
```

This proves volume persistence works.