Docker Task2

Date: 04/05/24

Q.1 Write a brief explanation of what Docker volumes are and why they are used in containerized environments.

State different types of volumes in Docker and also make a note on difference between them.

Docker volumes are a mechanism for persisting data generated by and used within Docker containers. In containerized environments, where applications run within isolated environments, data persistence becomes a challenge. Docker volumes provide a solution by enabling data to persist beyond the lifecycle of a container.

Docker volumes create a special directory within one or more containers or on the host machine itself, which can be shared among containers or with the host. This directory persists even if the container is stopped or removed.

Docker volumes are used for several reasons:

- **Data Persistence:** They allow data generated and used by containers to persist, ensuring that important information like databases, logs, or user uploads isn't lost when containers are stopped or restarted.
- **Sharing Data:** Volumes enable sharing of data among multiple containers. This is particularly useful in micro-services architectures where different services need access to the same data.
- **Performance:** Docker volumes can provide better performance than other methods of persisting data in containers, such as bind mounts, especially when dealing with large amounts of data or high I/O operations.
- **Backup and Restore:** Volumes facilitate easier backup and restore procedures since data is stored outside the container and can be backed up independently.

# The main types of volumes in Docker:

- **Bind Volumes:** Bind mounts allow you to mount a directory on the host filesystem into a container. Unlike named volumes, bind mounts can be created anywhere on the host filesystem and are not managed by Docker. Bind mounts offer a high degree of flexibility and can be used to share files and directories between the host and containers or between different containers. However, they may not be as portable or easy to manage as named volumes.

- **Named Volumes:** Named volumes are managed by Docker and are stored within a special directory on the host filesystem (typically under /var/lib/docker/volumes). Unlike host volumes, named volumes are not tied to specific paths on the host filesystem. Instead, Docker manages the volume's location and lifecycle. Named volumes are more portable and easier to manage than host volumes, making them a preferred choice for persisting data in production environments.
- **Anonymous Volumes:** Anonymous volumes are similar to named volumes but are not given an explicit name when created. Instead, Docker generates a unique identifier for each anonymous volume. Anonymous volumes are typically used when a container needs to write data to a temporary location that does not need to be preserved beyond the container's lifecycle. They are automatically deleted when the container is removed.

Bind Volume/Host Volume	**Named Volumes**	**Anonymous Volumes**
Host Volumes allow you to mount, directory from the host machine into a container.	Named Volumes are managed by docker & provided away to persist data independantlyof the container lifecycle.	Anonymous Volumes are similar to named volumes but they are not given a user defined name instead docker generates unique identifier for them.
Provide fast i/o performance since there's no intermidiate layer between container & host file system	Docker manages the volume's location & ensure data persistance even if the associated container tois removed.	They are primarily used es for tempory or disposable data that doesn't need to be shared to or persisted beyond the lifecycle of container
Host volumes bind volume are platform-dependent a 4 may lead to portability issues if the directory structure differs across hosts.	These are portables across docker hosts, to making them suitable for production environment.	Docker automatically removes volumes anonymous when associated containers is removed, reducing clutter on the host machine

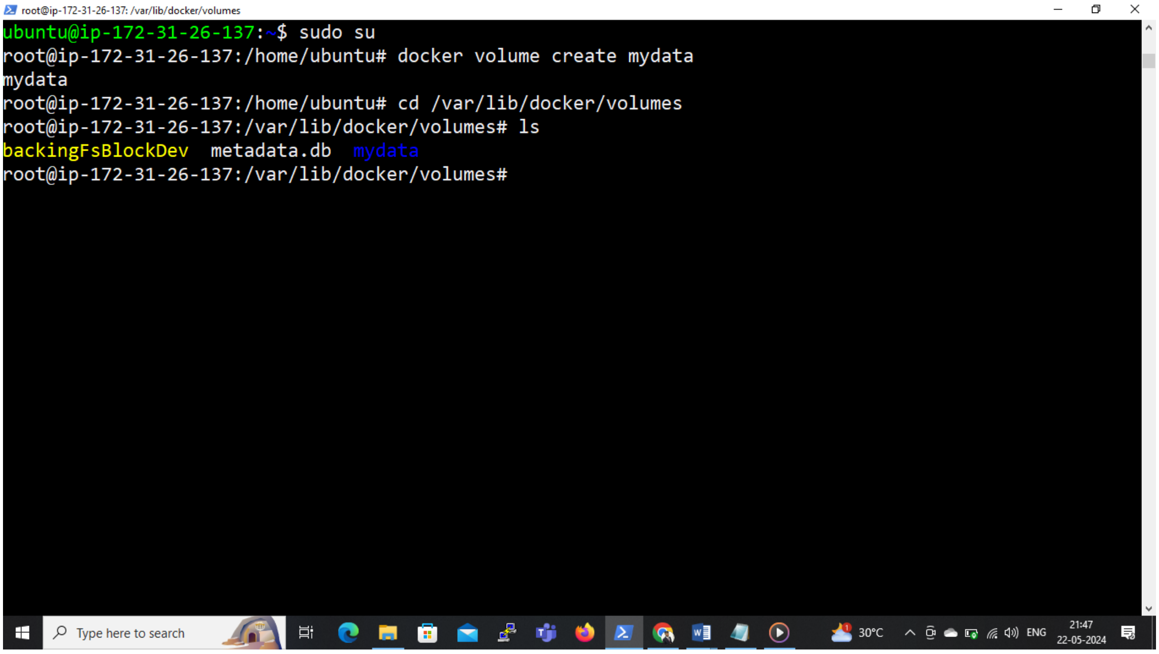
Q.2 Demonstrate the use of Named Volume.

- **Create a Docker Named volume named mydata.**
- **Attach volume to a Nginx Container**
- **Create an HTML file named index.html with some content (e.g., "Hello, Docker Volumes!") on your host machine. Copy this file into the mydata.**
- **Verify that the index.html file is accessible from within the container by starting a simple HTTP request.**

# Create a Named Volume:

Use the following command to create a named volume named mydata:

- docker volume create <volume name>
- docker volume create mydata

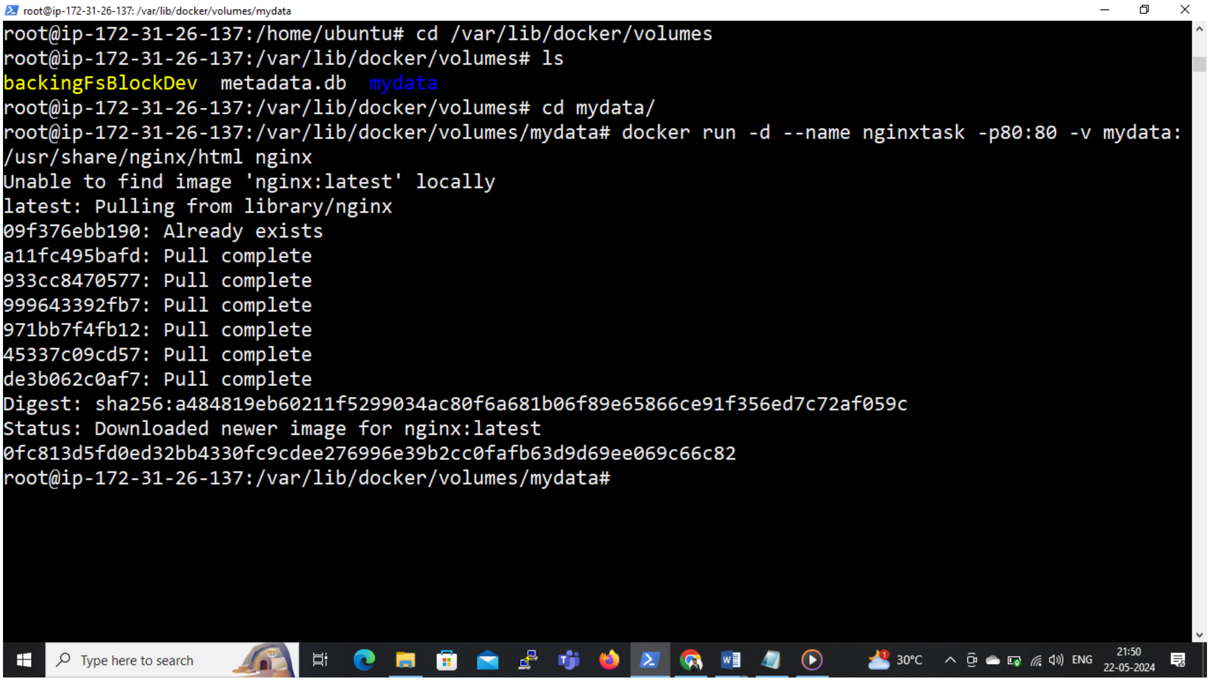


```
ubuntu@ip-172-31-26-137:~$ sudo su
root@ip-172-31-26-137:/home/ubuntu# docker volume create mydata
mydata
root@ip-172-31-26-137:/home/ubuntu# cd /var/lib/docker/volumes
root@ip-172-31-26-137:/var/lib/docker/volumes# ls
backingFsBlockDev  metadata.db  mydata
root@ip-172-31-26-137:/var/lib/docker/volumes#
```

# Attach Volume to an Nginx Container:

Run an Nginx container and mount the mydata named volume to the appropriate directory of nginx:

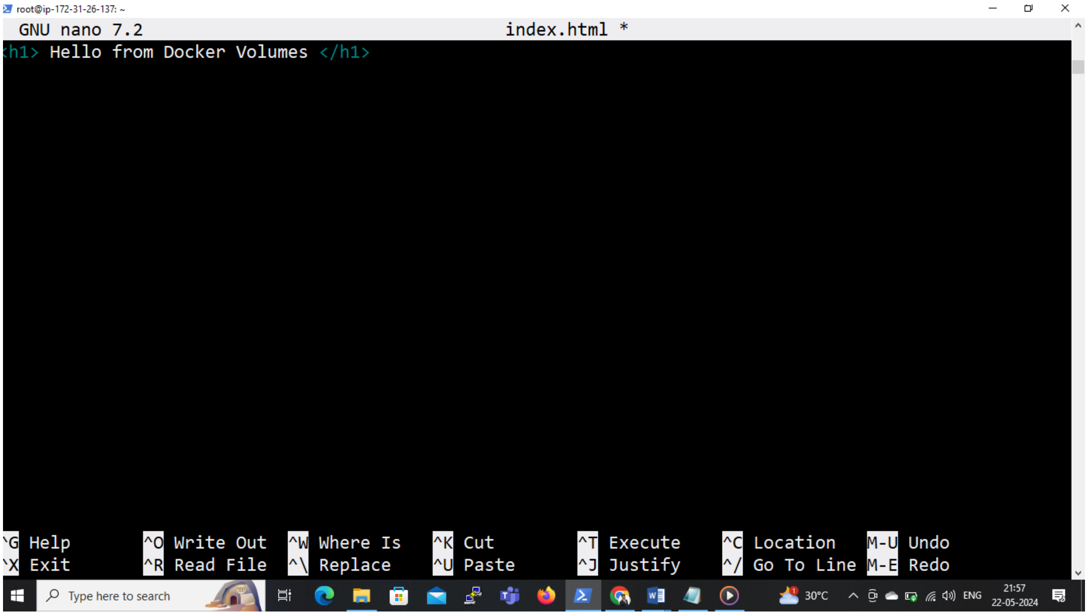
- docker run -d --name nginxtask -p80:80 -v mydata:/usr/share/nginx/html nginx



```
root@ip-172-31-26-137:/home/ubuntu# cd /var/lib/docker/volumes
root@ip-172-31-26-137:/var/lib/docker/volumes# ls
backingFsBlockDev  metadata.db  mydata
root@ip-172-31-26-137:/var/lib/docker/volumes# cd mydata/
root@ip-172-31-26-137:/var/lib/docker/volumes/mydata# docker run -d --name nginxtask -p80:80 -v mydata:
/usr/share/nginx/html nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
09f376ebb190: Already exists
a11fc495bafd: Pull complete
933cc8470577: Pull complete
999643392fb7: Pull complete
971bb7f4fb12: Pull complete
45337c09cd57: Pull complete
de3b062c0af7: Pull complete
Digest: sha256:a484819eb60211f5299034ac80f6a681b06f89e65866ce91f356ed7c72af059c
Status: Downloaded newer image for nginx:latest
0fc813d5fd0ed32bb4330fc9cdee276996e39b2cc0fafb63d9d69ee069c66c82
root@ip-172-31-26-137:/var/lib/docker/volumes/mydata#
```

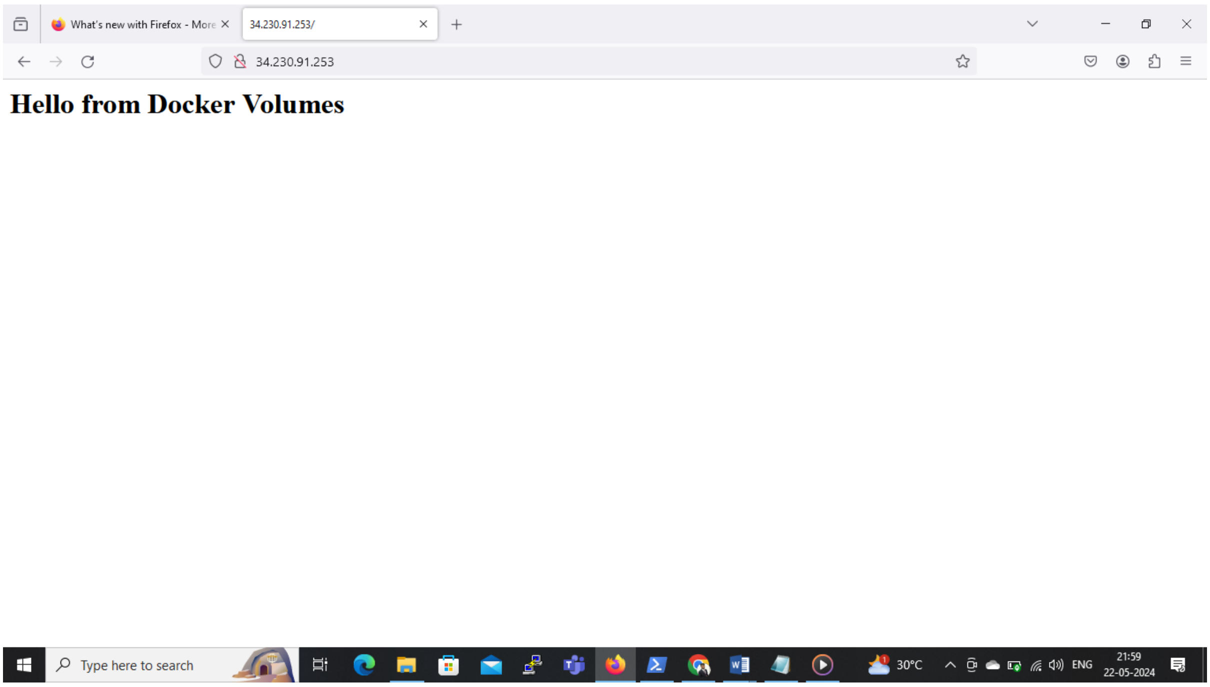
# Create an HTML file named index.html with some content (e.g., "Hello, Docker Volumes!") on your host machine. Copy this file into the mydata.

- ➢ cd
- ➢ docker exec -it nginxtask /bin/bash
- ➢ cd /usr/share/nginx/html
- ➢ rm index.html
- ➢ apt update
- ➢ apt install nano
- ➢ nano index.html



```
<h1> Hello from Docker Volumes </h1>
```

# Verify that the index.html file is accessible from within the container by starting a simple HTTP request.



Hello from Docker Volumes

Q.3 Write a brief explanation of what Docker networks.

Write the difference between host network and bridge network.

Docker Networks:

Docker networks are a fundamental feature of docker that enable communication & connectivity between docker containers running as the same host or across multiple hosts

Essentially, docker networks provide isolated environment for containers to communicate with each other, sing or to how physical, networks enable communication between computers

Key Points:.

- It creates an isolated environment for container allowing them to communicate securely without interference from other containers or external network.
- It facilitate seamless connectivity between containers, enabling them to interact with each other using standard network protocols such as TCP/IP.
- Docker networks support scalability containers to be easily the network added or removed from the network as needed
- There configuration options such as bridge networks, Overlay Networks, & custom networks to suit different use cases & deployment scenarios

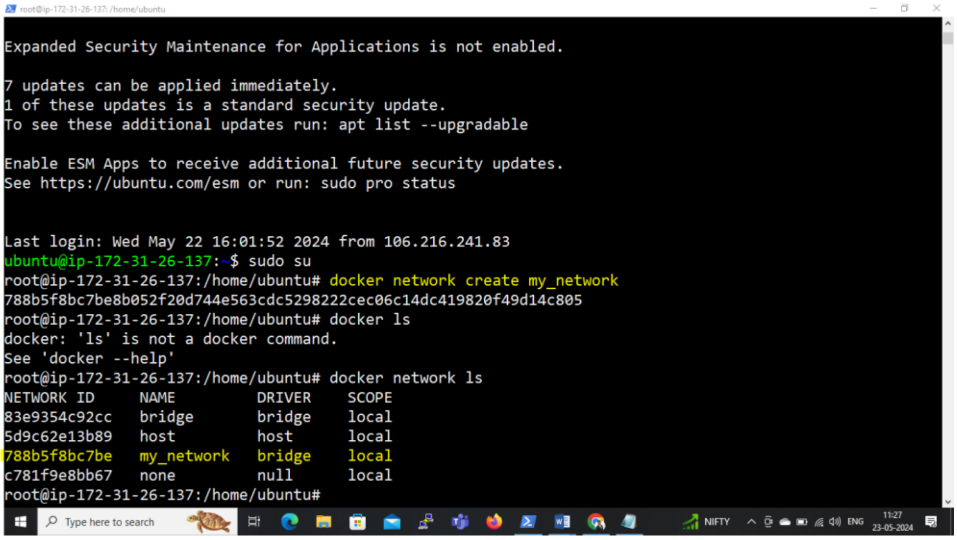
Host Network	Bridge Network
In host network mode, containers share the network namespace with the Docker host.	Bridge network is the default network mode created when Docker is installed.
Containers bypass Docker's network stack and use the host's network stack directly.	Each container connected to a bridge network gets its own unique IP address within the network.
This means that containers in host network mode have access to the same network interfaces and routing tables as the host.	Containers communicate with each other using these IP addresses, and Docker provides network address translation (NAT) to allow containers to access the external network.
Containers can bind to host ports directly, without needing to publish ports or perform port mapping.	By default, containers in a bridge network are isolated from the host network, but they can communicate with the external network through port mapping.
Host network mode offers better network performance because there is no overhead from Docker's network virtualization	Bridge networks provide network isolation and allow multiple containers to share the same IP address range without conflicts.
However, host network mode may pose security risks because containers have direct access to the host's network interfaces, potentially exposing the host to security vulnerabilities.	They are suitable for most use cases where containers need to communicate with each other and with the external network.

Q.4 Demonstrate the use of Custom Network

- **Create a custom bridge network named my_network.**
- **Start two containers, one using the nginx image and another using the httpd image.**
- **Attach both containers to the my_network network.**

# Create a custom bridge network named my_network:

- ➢ docker network create my_network



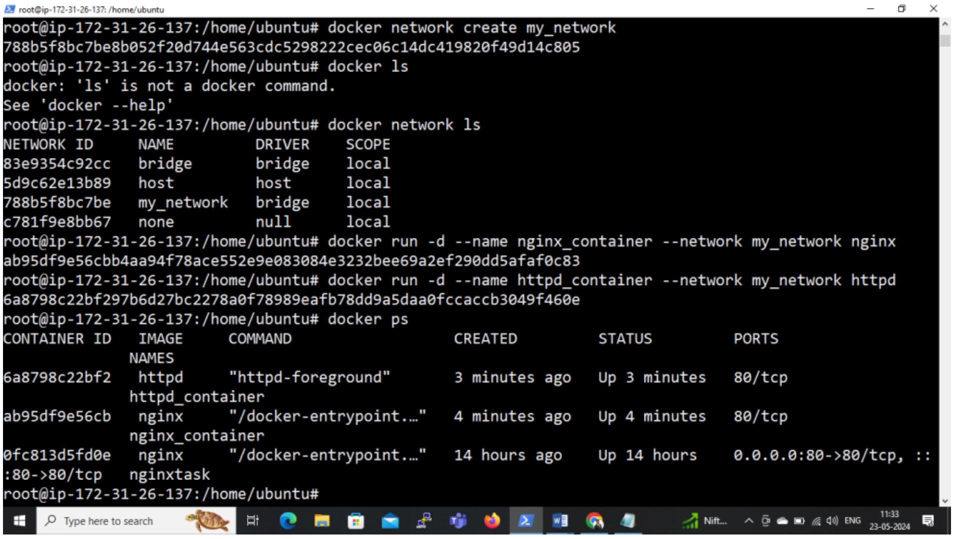
```
Expanded Security Maintenance for Applications is not enabled.

7 updates can be applied immediately.
1 of these updates is a standard security update.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Wed May 22 16:01:52 2024 from 106.216.241.83
ubuntu@ip-172-31-26-137:~$ sudo su
root@ip-172-31-26-137:/home/ubuntu# docker network create my_network
788b5f8bc7be8b052f20d744e563cdc5298222cec06c14dc419820f49d14c805
root@ip-172-31-26-137:/home/ubuntu# docker ls
docker: 'ls' is not a docker command.
See 'docker --help'
root@ip-172-31-26-137:/home/ubuntu# docker network ls
NETWORK ID     NAME         DRIVER    SCOPE
83e9354c92cc   bridge       bridge    local
5d9c62e13b89   host         host      local
788b5f8bc7be   my_network   bridge    local
c781f9e8bb67   none         null      local
root@ip-172-31-26-137:/home/ubuntu#
```

# Start the first container using the nginx image and attach it to the my_network network:

- ➢ docker run -d --name nginx_container --network my_network nginx

# Start the second container using the httpd image and attach it to the my_network network:

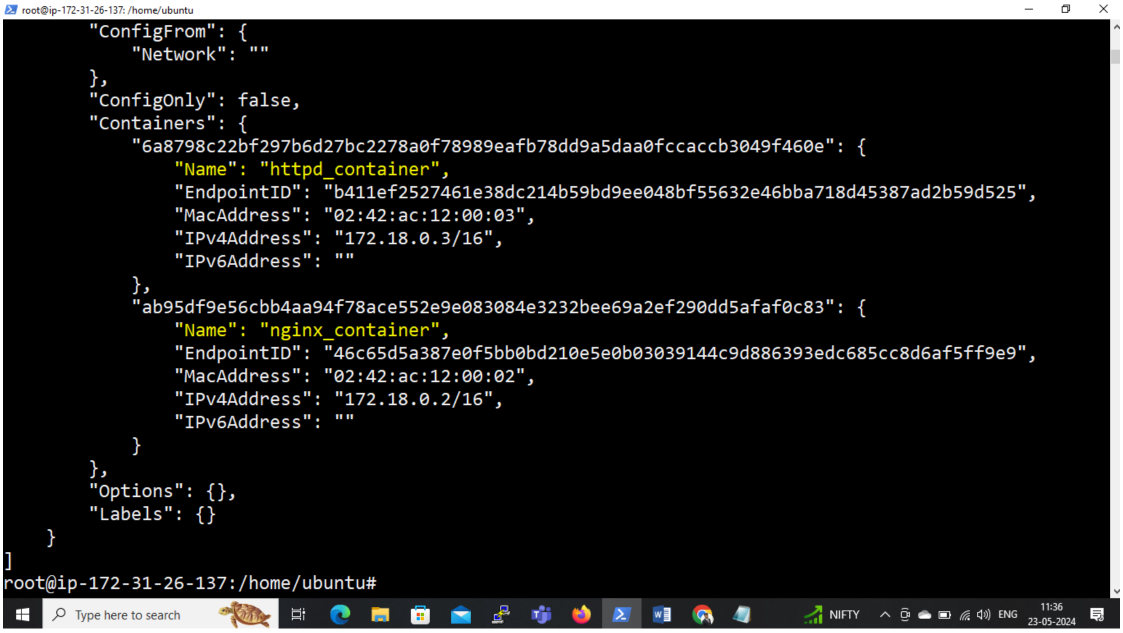
- ➢ docker run -d --name httpd_container --network my_network httpd



```
root@ip-172-31-26-137:/home/ubuntu# docker network create my_network
788b5f8bc7be8b052f20d744e563cdc5298222cec06c14dc419820f49d14c805
root@ip-172-31-26-137:/home/ubuntu# docker ls
docker: 'ls' is not a docker command.
See 'docker --help'
root@ip-172-31-26-137:/home/ubuntu# docker network ls
NETWORK ID     NAME         DRIVER    SCOPE
83e9354c92cc   bridge       bridge    local
5d9c62e13b89   host         host      local
788b5f8bc7be   my_network   bridge    local
c781f9e8bb67   none         null      local
root@ip-172-31-26-137:/home/ubuntu# docker run -d --name nginx_container --network my_network nginx
ab95df9e56cbb4aa94f78ace552e9e083084e3232bee69a2ef290dd5afaf0c83
root@ip-172-31-26-137:/home/ubuntu# docker run -d --name httpd_container --network my_network httpd
6a8798c22bf297b6d27bc2278a0f78989eafb78dd9a5daa0fccaccb3049f460e
root@ip-172-31-26-137:/home/ubuntu# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS
              NAMES
6a8798c22bf2   httpd     "httpd-foreground"       3 minutes ago    Up 3 minutes    80/tcp
              httpd_container
ab95df9e56cb   nginx     "/docker-entrypoint.…"   4 minutes ago    Up 4 minutes    80/tcp
              nginx_container
0fc813d5fd0e   nginx     "/docker-entrypoint.…"   14 hours ago     Up 14 hours     0.0.0.0:80->80/tcp, ::
:80->80/tcp   nginxtask
root@ip-172-31-26-137:/home/ubuntu#
```

# Now both containers (nginx_container and httpd_container) are connected to the my_network network.

- ➢ docker network inspect my_network



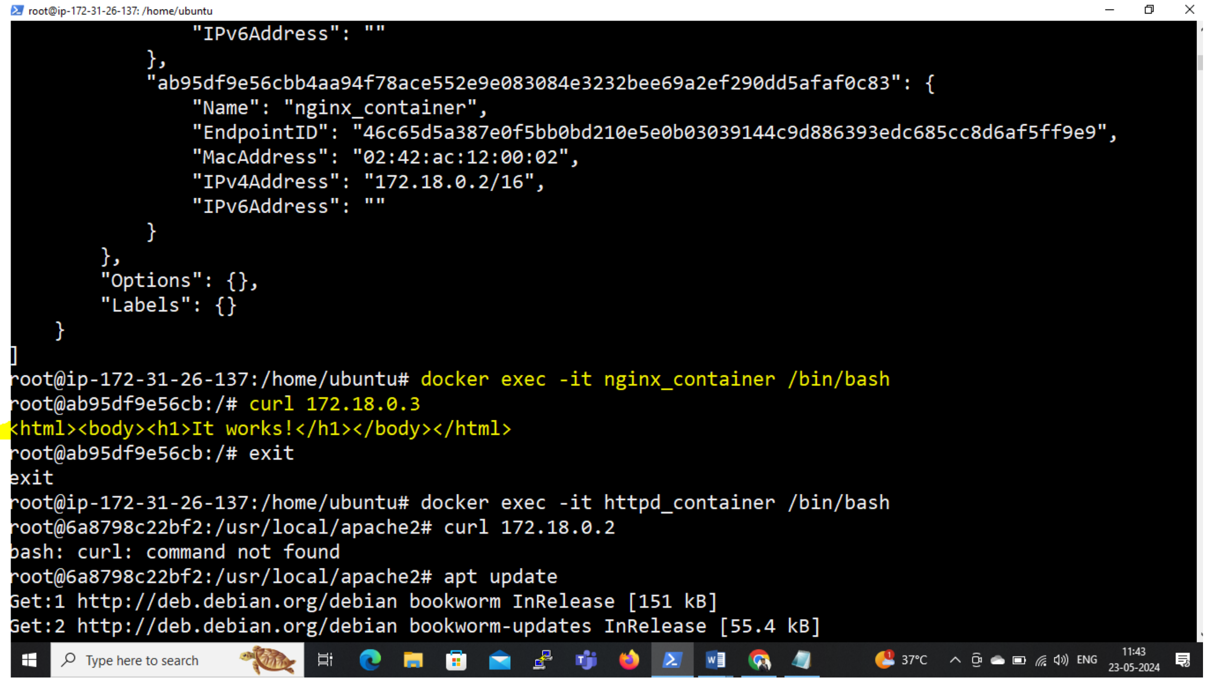
```
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "6a8798c22bf297b6d27bc2278a0f78989eafb78dd9a5daa0fccaccb3049f460e": {
                "Name": "httpd_container",
                "EndpointID": "b411ef2527461e38dc214b59bd9ee048bf55632e46bba718d45387ad2b59d525",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            },
            "ab95df9e56cbb4aa94f78ace552e9e083084e3232bee69a2ef290dd5afaf0c83": {
                "Name": "nginx_container",
                "EndpointID": "46c65d5a387e0f5bb0bd210e5e0b03039144c9d886393edc685cc8d6af5ff9e9",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
root@ip-172-31-26-137:/home/ubuntu#
```

# They can communicate with each other using container names as hostnames.

- ➢ **docker exec -it nginx_container /bin/bash**

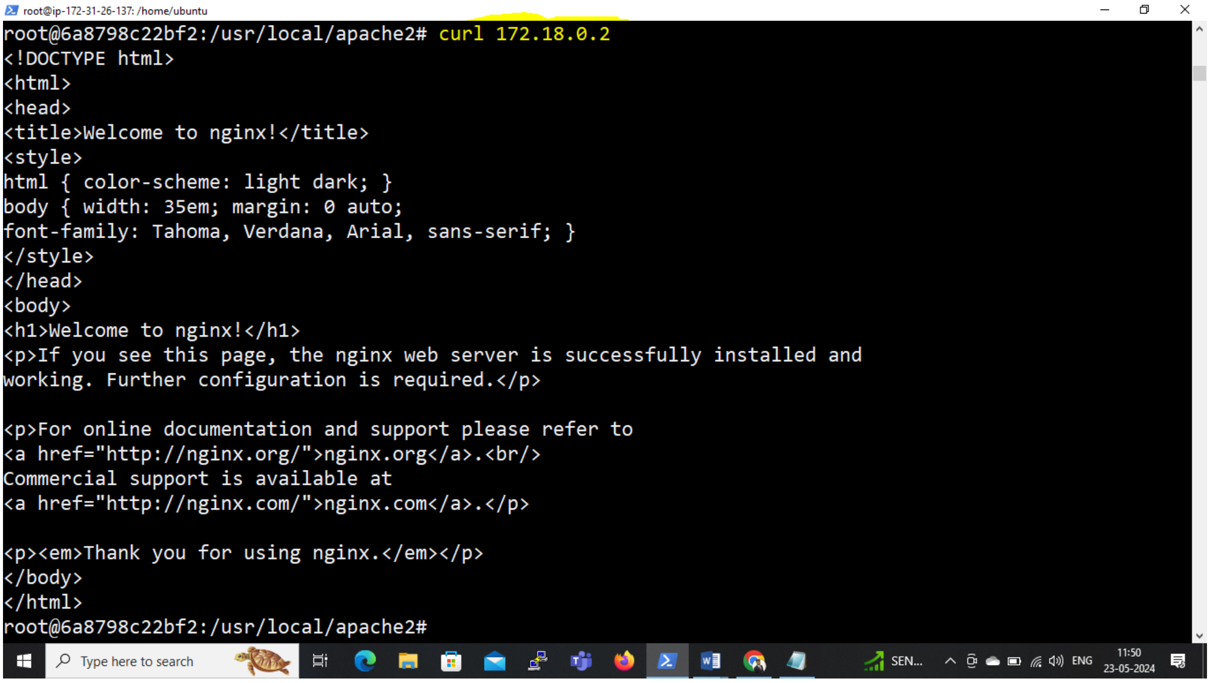
- ➢ **curl 172.18.0.3 (curl IP of httpd_container)**

- ➢ **exit**



```
                "IPv6Address": ""
            },
            "ab95df9e56cbb4aa94f78ace552e9e083084e3232bee69a2ef290dd5afaf0c83": {
                "Name": "nginx_container",
                "EndpointID": "46c65d5a387e0f5bb0bd210e5e0b03039144c9d886393edc685cc8d6af5ff9e9",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
root@ip-172-31-26-137:/home/ubuntu# docker exec -it nginx_container /bin/bash
root@ab95df9e56cb:/# curl 172.18.0.3
<html><body><h1>It works!</h1></body></html>
root@ab95df9e56cb:/# exit
exit
root@ip-172-31-26-137:/home/ubuntu# docker exec -it httpd_container /bin/bash
root@6a8798c22bf2:/usr/local/apache2# curl 172.18.0.2
bash: curl: command not found
root@6a8798c22bf2:/usr/local/apache2# apt update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
```

- **docker exec -it httpd_container /bin/bash**
- **apt update**
- **apt install curl**
- **curl 172.18.0.2 (curl <IP of nginx_container>)**



```
root@6a8798c22bf2:/usr/local/apache2# curl 172.18.0.2
<!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@6a8798c22bf2:/usr/local/apache2#
```