Experiment -2.2

Student Name: Ayush Pandey
UID: 22BDO10038
Branch: CSE-DevOps
Section/Group: 22BCD-1(A)
Semester: 5
Date of Performance: 23-09-24
Subject Name: Docker and Kubernetes
Subject Code: 22CSH-343

1. Aim/Overview of the practical:

To Setup
i. Container to WWW Communication,
ii. Container to Local Host Machine Communication,
iii. Container to Container Communication,
iv. Creating a Container & Communicating to the Web (WWW),
v. Container to Host Communication Work,
vi. Container to Container Communication using Docker Desktop.

2. Apparatus: PC, Docker Engine, DockerHub, Ubuntu Linux

3. Steps for experiment/practical:

Docker Networking:

1. It allows you to create a Network of Docker Containers managed by a master node called the manager.
2. Containers inside the Docker Network can talk to each other by sharing packets of information.
3. The Docker network is a virtual network created by Docker to enable communication between Docker containers.
4. If two containers are running on the same network or host, they can communicate with each other without the need for ports to be exposed to the host machine.
5. A network driver defines how containers interact with each other, with the host system, and with external networks.

- Bridge (default)
  Use Case: Containers on the same host.
  How It Works: Containers connected to the same bridge network can communicate with

each other, but they are isolated from other networks by default. This is the default driver if no network is specified when creating a container

- Host
  Use Case: For cases where the container should share the host's network stack.
  How It Works: The container uses the host's network directly, which means it doesn't get its own IP address. Instead, it shares the host's IP and ports.

- Overlay
  Use Case: Multi-host Docker deployments, especially in Docker Swarm.
  How It Works: Allows containers running on different hosts to communicate with each other, using a distributed network across the Docker Swarm cluster. It creates an encrypted network overlay over the physical infrastructure.

- Macvlan
  Use Case: Direct communication with physical networks.
  How It Works: Assigns a MAC address to each container, making it appear as a physical device on the network. This is useful when you want to bypass the Docker host's network stack.

- None
  Use Case: Total network isolation
  How It Works: The container doesn't get any network interface, effectively isolating it from any network.

- **Docker Network Commands**
  1. docker network create –driver<driver-name> <network-name>

```
ayush@Linux:~$ sudo docker network create --driver bridge demo-networks
9f983b462e8e3c3e8cd6e55a2cc41b411def49f9f95295bf49e27850f82b70d3
```

2. docker network ls

```
ayush@Linux:~$ docker network ls
NETWORK ID     NAME            DRIVER    SCOPE
ebba4984d89a   bridge          bridge    local
b44f15b9095c   demo-network    bridge    local
9f983b462e8e   demo-networks   bridge    local
ba2a42c1cafa   host            host      local
da6492661b43   none            null      local
```

3. docker network connect <network-name> <container-name>

```
ayush@Linux:~$ sudo docker network connect demo-networks 7c62ba724552
```

4. docker network inspect <network-name>

```
ayush@Linux:~$ sudo docker network inspect demo-networks
[
    {
        "Name": "demo-networks",
        "Id": "9f983b462e8e3c3e8cd6e55a2cc41b411def49f9f95295bf49e27850f82b70d3",
        "Created": "2024-10-20T22:47:35.205864873+05:30",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.20.0.0/16",
                    "Gateway": "172.20.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "7c62ba724552c4a6533362c8c34695a8c587e6a5c4e1b6af5b80ab113e036545": {
                "Name": "network-demo",
                "EndpointID": "914f38a9e6a8b29df9659142cfe25f3967473ef3da5c117112c78b32de5e547d",
                "MacAddress": "02:42:ac:14:00:02",
                "IPv4Address": "172.20.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

5. docker network disconnect <network-name> <container-name>
6. docker network rm <network-name>

```
ayush@Linux:~$ docker network disconnect demo-networks 7c62ba724552
ayush@Linux:~$ docker network rm demo-networks
demo-networks
ayush@Linux:~$ sudo docker network ls
NETWORK ID     NAME           DRIVER    SCOPE
ebba4984d89a   bridge         bridge    local
b44f15b9095c   demo-network   bridge    local
ba2a42c1cafa   host           host      local
da6492661b43   none           null      local
```

7. docker network prune

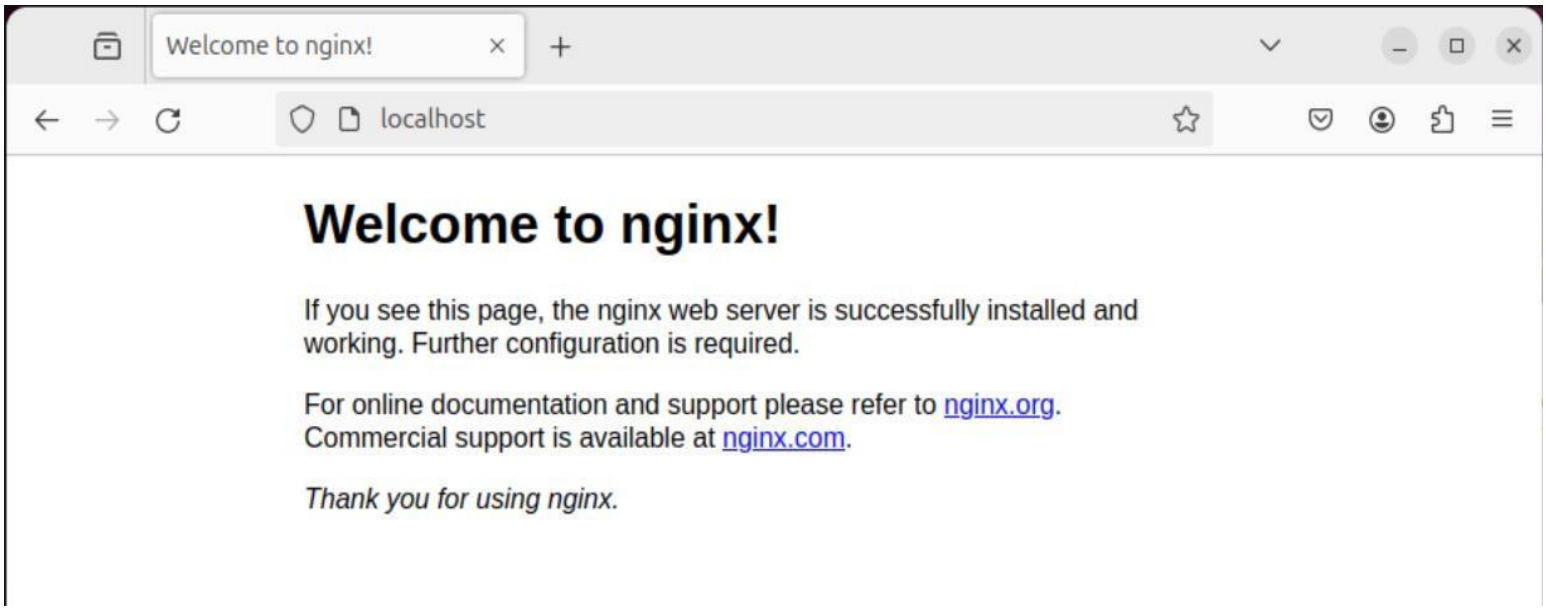
```
ayush@Linux:~$ docker network prune
WARNING! This will remove all custom networks not used by at least one container
.
Are you sure you want to continue? [y/N] y
Deleted Networks:
minikube
demo-network

ayush@Linux:~$ docker network ls

NETWORK ID     NAME      DRIVER    SCOPE
f48b90b1dafe   bridge    bridge    local
ba2a42c1cafa   host      host      local
da6492661b43   none      null      local
```

- **Container to WWW Communication**

1. Pull a Nginx image from Docker Hub to create a web server container.
2. Create a Docker container using the Nginx image
3. Access the default webpage running inside the container using https://localhost:80

```
ayush@Linux:~$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
a480a496ba95: Pull complete
f3ace1b8ce45: Pull complete
11d6fdd0e8a7: Pull complete
f1091da6fd5c: Pull complete
40eea07b53d8: Pull complete
6476794e50f4: Pull complete
70850b3ec6b2: Pull complete
Digest: sha256:28402db69fec7c17e179ea87882667f1e054391138f77ffaf0c3eb388efc3ffb
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

```
ayush@Linux:~$ docker run -dit --name default-web -p 80:80 nginx
41ac3b6ce1ddfdc50541de82b9e04dec0dddd3823de30a45d34ae8edff05f68c
```



- **Container to Local Host Machine Communication**

1. Identify the service running on the host machine that you want to communicate with from the container. Have the default web-page of nginx running on the host machine
2. Determine the IP address of the host machine.

```
ayush@Linux:~$ ifconfig
cali2cb837795e7: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet6 fe80::ecee:eeff:feee:eeee  prefixlen 64  scopeid 0x20<link>
        ether ee:ee:ee:ee:ee:ee  txqueuelen 0  (Ethernet)
        RX packets 199  bytes 21223 (21.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 201  bytes 210947 (210.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

cali52aa690cb21: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet6 fe80::ecee:eeff:feee:eeee  prefixlen 64  scopeid 0x20<link>
        ether ee:ee:ee:ee:ee:ee  txqueuelen 0  (Ethernet)
        RX packets 214  bytes 21615 (21.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 227  bytes 30789 (30.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

cali5c900ed0750: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet6 fe80::ecee:eeff:feee:eeee  prefixlen 64  scopeid 0x20<link>
        ether ee:ee:ee:ee:ee:ee  txqueuelen 0  (Ethernet)
        RX packets 11  bytes 866 (866.0 B)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 21  bytes 2892 (2.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:43:f4:4c:29  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Create a Docker container and configure it to communicate with the host machine.

Test the container's communication with the host machine. Run the following command to access a service (e.g., a web server) running on the host machine from within the container:

docker exec my_local_app curl http://host_machine_IP

- **Container to Container Communication**

1. Create two Docker containers that need to communicate with each other:

```
ayush@Linux:~$ docker run -dit --name contain1 nginx
15f2911a6954ae2cb2071c10b341f807e310160ffd84148b2ded189a8250bd10
ayush@Linux:~$ docker run -dit --name contain2 nginx
da5efb17d6f22257284f329adf4f7114c98bd841a815af8cbb3eb18e6980957b
ayush@Linux:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
da5efb17d6f2   nginx     "/docker-entrypoint.…"   5 seconds ago    Up 4 seconds    80/tcp    contain2
15f2911a6954   nginx     "/docker-entrypoint.…"   12 seconds ago   Up 11 seconds   80/tcp    contain1
```

2. Create a user-defined bridge network and attach both containers to the user-defined network:

```
ayush@Linux:~$ docker network create contain-bridge
2782a8e4781f4bb04005d951319272b562cb77c623ca899ef8b11f9182dc95a8
ayush@Linux:~$ docker network connect contain-bridge contain1
ayush@Linux:~$ docker network connect contain-bridge contain2
```

3. Verify that both containers are connected to the same network. Both the containers would be visible under containers when we inspect the docker network.

```
ayush@Linux:~$ docker inspect contain-bridge
[
    {
        "Name": "contain-bridge",
        "Id": "2782a8e4781f4bb04005d951319272b562cb77c623ca899ef8b11f9182dc95a8",
        "Created": "2024-10-22T23:04:46.167382131+05:30",
        "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": {
            "15f2911a6954ae2cb2071c10b341f807e310160ffd84148b2ded189a8250bd10": {
                "Name": "contain1",
                "EndpointID": "88bd5e80110ac8d1df4ccbaf9027656639372eb3dadaad725e962c7ed1eb7214",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
```

4. Test the communication between the containers by pinging b/w them. Send 4 packets by docker exec <container_name> ping -c 4 <container_name>

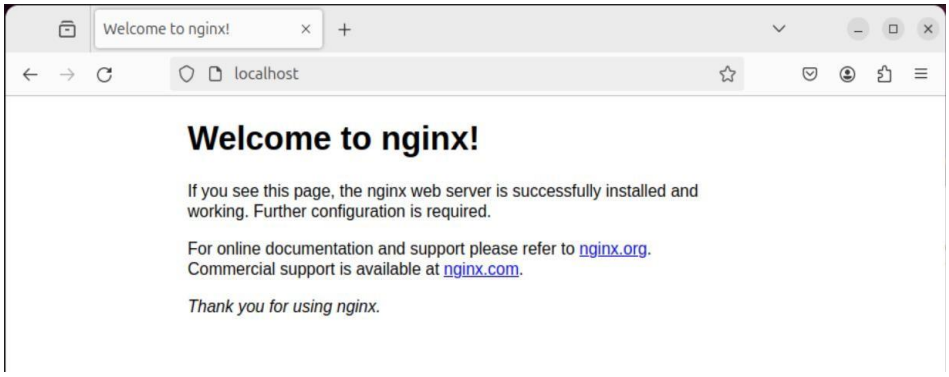
```
ayush@Linux:~$ docker exec contain1 ping -c 4 contain2
PING contain2 (172.18.0.3) 56(84) bytes of data.
64 bytes from contain2.contain-bridge (172.18.0.3): icmp_seq=1 ttl=64 time=
0.138 ms
64 bytes from contain2.contain-bridge (172.18.0.3): icmp_seq=2 ttl=64 time=
0.054 ms
64 bytes from contain2.contain-bridge (172.18.0.3): icmp_seq=3 ttl=64 time=
0.044 ms
64 bytes from contain2.contain-bridge (172.18.0.3): icmp_seq=4 ttl=64 time=
0.144 ms

--- contain2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3015ms
rtt min/avg/max/mdev = 0.044/0.095/0.144/0.046 ms
```

Creating a Container & Communicating to the Web (WWW)

1. Create a Docker container using Nginx image

```
ayush@Linux:~$ docker run -dit --name contain3 -p 80:80 nginx
664177df78e3efaec158253a9347bb1c1da5c1155ebc85eeb1698fb7375783a5
```

2. Access the web server running inside the container at the address http://localhost:80/



3. If everything is set up correctly, you should see the default Nginx welcome page in your web browser.

- **Container to Host Communication Work**

1. Identify the service running on the host machine that you want to communicate with from the container. For this, a web-server running on nginx on the host machine.

2. Create a Docker container and configure it to communicate with the host service.

```
ayush@Linux:~$ docker run -dit --name contain3 -p 80:80 nginx
664177df78e3efaec158253a9347bb1c1da5c1155ebc85eeb1698fb7375783a5
```

3. Access the service running on the host from the container

   a. Inside the container, you can now access the web server running on the host machine using the host IP address and the mapped port (8080).

Learning outcomes (What I have learnt):

1. I have learnt the concept of containerization.

2. I have learnt to configure Docker to work with different environments.
3. I have learnt how to build docker images using Dockerfile.
4. I have learnt the purpose of Dockerfile and its advantages.

5. I have learnt how Dockerfile can help in creating CI/CD pipelines.

Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			