Experiment -2.2

Student Name: Chayan Gope
UID: 22BDO1036
Branch: AIT-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.
    a. 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.
    b. 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.
    c. 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.
    d. 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.
    e. 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**

  sudo docker network create --driver <driver-name> <network-name>

  1. sudo docker network ls

```
chayan@chayan-virtual-machine:~$ docker network create --driver bridge demo-network
77b70681f180ededecc11f36f9721dc659651a67da6d17a7beec8802af2d6c9a
chayan@chayan-virtual-machine:~$ docker network ls
NETWORK ID     NAME           DRIVER    SCOPE
c1286725460e   bridge         bridge    local
77b70681f180   demo-network   bridge    local
d6ebffb9eed9   host           host      local
35a9dbf1c46d   none           null      local
```

  2. sudo docker network connect <network-name> <container-name or id>

  3. sudo docker network inspect <network-name>

```
chayan@chayan-virtual-machine:~$ docker network connect demo-network f754949a28c6
chayan@chayan-virtual-machine:~$ docker network inspect demo-network
[
    {
        "Name": "demo-network",
        "Id": "90e968393dac028f62755db3b2758e2b048404bf9355290a07566f0c1d0da636",
        "Created": "2024-09-23T12:18:26.21932539+05:30",
        "Scope": "local",
```

```
        "Containers": {
            "51bf0cefc9fde245addafc5b8bf4abdba882400027736ed22ec6051d642a53cb": {
                "Name": "jovial_hertz",
                "EndpointID": "32e81fcd692611049ff7d9fffb37235bb7a1632cd9493fd305d0cb1dd2d5e467",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

4. sudo docker network disconnect <network-name> <container-name>

```
chayan@chayan-virtual-machine:~$ docker network disconnect demo-network 51bf0cefc9fd
```

5. sudo docker network rm <network-name>

```
chayan@chayan-virtual-machine:~$ docker network rm demo-network
demo-network
chayan@chayan-virtual-machine:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
c1286725460e   bridge    bridge    local
d6ebffb9eed9   host      host      local
85a9dbf1c46d   none      null      local
```

6. sudo docker network prune

- **Container to WWW Communication**
    1. Pull an Node image from Docker Hub to create a web server container.
    2. Create a Docker container using the Node image
    3. Access the default webpage running inside the container using https://localhost:80

```
chayan@chayan-virtual-machine:~$ docker pull node
Using default tag: latest
latest: Pulling from library/node
8cd46d290033: Pull complete
2e6afa3f266c: Pull complete
2e66a70da0be: Pull complete
1c8ff076d818: Pull complete
71a2ad2ab1a1: Pull complete
8ed09065f016: Pull complete
8cc9946ce160: Pull complete
fa87db89e50a: Pull complete
Digest: sha256:cbe2d5f94110cea9817dd8c5809d05df49b4bd1aac5203f3594d88665ad37988
Status: Downloaded newer image for node:latest
docker.io/library/node:latest
chayan@chayan-virtual-machine:~$ docker run -dit node
```

- **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 node running on the host machine.
    2. Determine the IP address of the host machine.
    3. Create a Docker container and configure it to communicate with the host machine.

4. 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:

   sudo 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:

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

```
chayan@chayan-virtual-machine:~$ docker network connect demo-network 1ecdc6851cf5
chayan@chayan-virtual-machine:~$ docker network connect demo-network f754949a28c6
chayan@chayan-virtual-machine:~$ docker network inspect demo-network
[
    {
        "Name": "demo-network",
        "Id": "90e968393dac028f62755db3b2758e2b048404bf9355290a07566f0c1d0da636",
        "Created": "2024-09-23T12:18:26.21932539+05:30",
        "Scope": "local",
```

3. Verify that both containers are connected to the same network

```
"Containers": {
    "1ecdc6851cf5675f4c1036f5b0fcf13475caf83840e97a210752aa7265965e1d": {
        "Name": "silly_payne",
        "EndpointID": "279ae96a23effdb40847dc49abbec8d52340790eb30994477c1308193c3218f5",
        "MacAddress": "02:42:ac:13:00:02",
        "IPv4Address": "172.19.0.2/16",
        "IPv6Address": ""
    },
    "f754949a28c6520cb4df66c5d4bf15c20c8d1562535fe45f6973322f6d885482": {
        "Name": "stupefied_visvesvaraya",
        "EndpointID": "1c02bcfa24a6259e52a2857589ce47483f27a6f28701d25151f1955b39c8c42a",
        "MacAddress": "02:42:ac:13:00:03",
        "IPv4Address": "172.19.0.3/16",
        "IPv6Address": ""
    }
```

4. Test the communication between the containers by pinging b/w them.

- **Creating a Container & Communicating to the Web (WWW)**

1. Create a Docker container using the Node image

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 Node 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 node on the host machine.
2. Create a Docker container and configure it to communicate with the host service.
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.			