Lab Exercise 4- Working with Docker Networking

Name: Aditya Tomar

SAP: 500106015

E.NO: R2142221060

Batch: B-2(DevOps)

Step 1: Understanding Docker Default Networks

Docker provides three default networks:

- bridge: The default network when a container starts.
- host: Bypasses Docker's network isolation and attaches the container directly to the host network.
- none: No networking is available for the container.

1.1. Inspect Default Networks

Check Docker's default networks using:

```
docker network ls
```

Terminal

```
adityatomar@Adityas-Air-3 ~ % docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
edbbd2fd5a55   bridge    bridge    local
b105c931f9a9   host      host      local
868b13639d3d   none      null      local
adityatomar@Adityas-Air-3 ~ %
```

1.2. Inspect the Bridge Network

```
docker network inspect bridge
```

```
adityatomar@Adityas-Air-3 ~ % docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "edbbd2fd5a55a7bf45e969f413cd5bfaed2224efd78d7a5109f5f30a940c685d",
        "Created": "2024-09-09T05:51:44.311416625Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
```

This command will show detailed information about the bridge network, including the connected containers and IP address ranges.

Step 2: Create and Use a Bridge Network

2.1. Create a User-Defined Bridge Network

A user-defined bridge network allows containers to communicate by name instead of IP.

```
docker network create my_bridge
```

```
adityatomar@Adityas-Air-3 ~ % docker network create my_bridge
a497b1b9209a3d7877095b37689c2512db32df44fbb766ca762d332b09186bc3
adityatomar@Adityas-Air-3 ~ %
```

2.2. Run Containers on the User-Defined Network

Start two containers on the newly created my_bridge network:

```
docker run -dit --name container1 --network my_bridge busybox
```

```
adityatomar@Adityas-Air-3 ~ % docker run -dit --name container1 --network my_bridge busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
75e8ca8f509f: Pull complete
Digest: sha256:34b191d63fbc93e25e275bfccf1b5365664e5ac28f06d974e8d50090fbb49f41
Status: Downloaded newer image for busybox:latest
7231f0b3b3adcae870021c996ef8e1e5f62a9591cf6b474890222cd025bbb83a
```

docker run -dit --name container2 --network my_bridge busybox

```
adityatomar@Adityas-Air-3 ~ % docker run -dit --name container2 --network my_bridge busybox

d5a5bc3fa683e5fb867feae5609ffa68bf301a080ced97ea858d032c39a8c20a
```

2.3. Test Container Communication

Execute a ping command from container1 to container2 using container names:

docker exec -it container1 ping container2

```
adityatomar@Adityas-Air-3 ~ % docker exec -it container1 ping container2

PING container2 (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.186 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.186 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.194 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.246 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.242 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.272 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.215 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.181 ms
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.242 ms
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.240 ms
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.270 ms
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.302 ms
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.306 ms
64 bytes from 172.18.0.3: seq=13 ttl=64 time=0.302 ms
64 bytes from 172.18.0.3: seq=14 ttl=64 time=0.251 ms
64 bytes from 172.18.0.3: seq=15 ttl=64 time=0.334 ms
64 bytes from 172.18.0.3: seq=16 ttl=64 time=0.263 ms
```

The containers should be able to communicate since they are on the same network.

Step 3: Create and Use a Host Network

3.1. Run a Container Using the Host Network

The host network allows the container to use the host machine's networking stack:

```
docker run -d --name host_network_container --network host nginx
```

```
adityatomar@Adityas-Air-3 ~ % docker run -d --name host_network_container --network host nginx

Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
92c3b3500be6: Pull complete
ee57511b3c68: Pull complete
33791ce134bf: Pull complete
cc4f24efc205: Pull complete
3cad04a21c99: Pull complete
486c5264d3ad: Pull complete
b3fd15a82525: Pull complete
Digest: sha256:04ba374043ccd2fc5c593885c0eacddebabd5ca375f9323666f28dfd5a9710e3
Status: Downloaded newer image for nginx:latest
0b54ba7a4ddaf22a060dd2d24801f5aa8fc4038ccec8148ed9006121b5bae499
```

Access the NGINX server via localhost:80 in your browser to verify the container is using the host network.

3.2. Check Network

```
docker network inspect host
```

```
adityatomar@Adityas-Air-3 ~ % docker network inspect host
[
    {
        "Name": "host",
        "Id": "b105c931f9a9ab99e6bf5558adf79f0db1e894bde08d92368230bdecce2e93e8",
        "Created": "2024-08-11T14:26:36.057801291Z",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": null
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
```

Step 4: Disconnect and Remove Networks

4.1. Disconnect Containers from Networks

To disconnect container1 from my_bridge:

```
docker network disconnect my_bridge container1
```

```
adityatomar@Adityas-Air-3 ~ % docker network disconnect my_bridge container1
adityatomar@Adityas-Air-3 ~ % docker network disconnect my_bridge container1
Error response from daemon: container 7231f0b3b3adcae870021c996ef8e1e5f62a9591cf6b474890222cd025bbb83a
 is not connected to network my_bridge
```

4.2. Remove Networks

To remove the user-defined network:

```
docker network rm my_bridge
```

```
adityatomar@Adityas-Air-3 ~ % docker network rm my_bridge
my_bridge
```

Step 5: Clean Up

Stop and remove all containers created during this exercise:

```
docker rm -f container1 container2 host_network_container
```

```
adityatomar@Adityas-Air-3 ~ % docker rm -f container1 container2 host_network_container
container1
container2
host_network_container
```