Lab Exercise 4

Working with Docker Networking

Step 1: Understanding Docker Default Networks

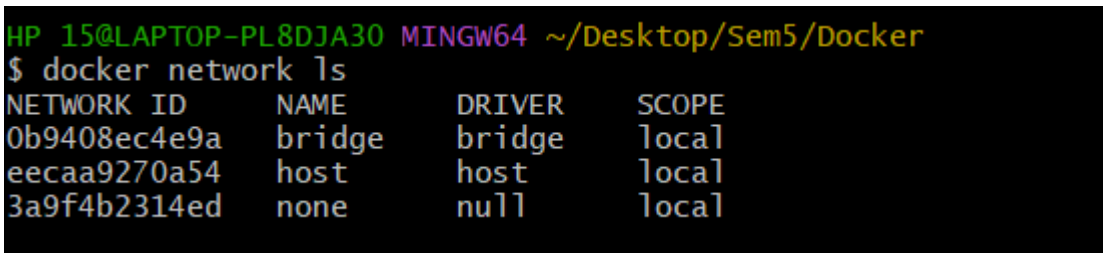
Docker provides three default networks:

- 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
```



1.2. Inspect the Bridge Network

```
docker network inspect bridge
```

```
HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "0b9408ec4e9a95cff296076db1ac2c2e6d1f314060cc3ad13c6a6d1029be9e71"
,
        "Created": "2024-11-21T15:58:00.207706366Z",
        "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,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```

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
```

```
HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker network create akshit_bridge
37bf9598b9838eb55099b0c959e68555aeb651a768a107750a406e9a76047c6a

HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker network ls
NETWORK ID     NAME            DRIVER    SCOPE
37bf9598b983   akshit_bridge   bridge    local
0b9408ec4e9a   bridge          bridge    local
eecaa9270a54   host            host      local
3a9f4b2314ed   none            null      local
```

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

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

```
HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker run -dit --name container1 --network akshit_bridge busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
430378704d12: Pull complete
Digest: sha256:db142d433cdde11f10ae479dbf92f3b13d693fd1c91053da9979728cceb1dc68
Status: Downloaded newer image for busybox:latest
21e1cd396781f42b249a1e0aac8bb7c4e322ca986078bb41724fe215c2f266e6

HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker run -dit --name container2 --network akshit_bridge busybox
89afdf2fbacfa782a00092033fc927b961bed671e78e20ff446b5bf6991073f6
```

2.3. Test Container Communication

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

```
docker exec -it container1 ping container2
```

```
HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ 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=1.030 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.069 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.084 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.107 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.084 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.067 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.107 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.128 ms
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.083 ms
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.128 ms
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.125 ms
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.123 ms
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.390 ms
^X64 bytes from 172.18.0.3: seq=13 ttl=64 time=0.071 ms
64 bytes from 172.18.0.3: seq=14 ttl=64 time=0.078 ms
64 bytes from 172.18.0.3: seq=15 ttl=64 time=0.170 ms
^C
--- container2 ping statistics ---
16 packets transmitted, 16 packets received, 0% packet loss
round-trip min/avg/max = 0.067/0.177/1.030 ms
```

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

Step 3: Disconnect and Remove Networks

3.1. Disconnect Containers from Networks

To disconnect container1 from my_bridge:

```
docker network disconnect my_bridge container1
```

```
HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker network disconnect akshit_bridge container1

HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker exec -it container1 ping container2
ping: bad address 'container2'
```

4.2. Remove Networks

To remove the user-defined network:

```
docker network rm my_bridge
```

```
HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker network rm akshit_bridge
Error response from daemon: error while removing network: network akshit_bridge
id 37bf9598b9838eb55099b0c959e68555aeb651a768a107750a406e9a76047c6a has active e
ndpoints

HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker network disconnect akshit_bridge container2

HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker network rm akshit_bridge
akshit_bridge
```

Step 4: Clean Up

Stop and remove all containers created during this exercise:

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

```
HP 15@LAPTOP-PL8DJA30 MINGW64 ~/Desktop/Sem5/Docker
$ docker rm -f container1 container2
container1
container2
```