Lab Exercise 4- Working with Docker Networking

Step 1: Understanding Docker Default Networks

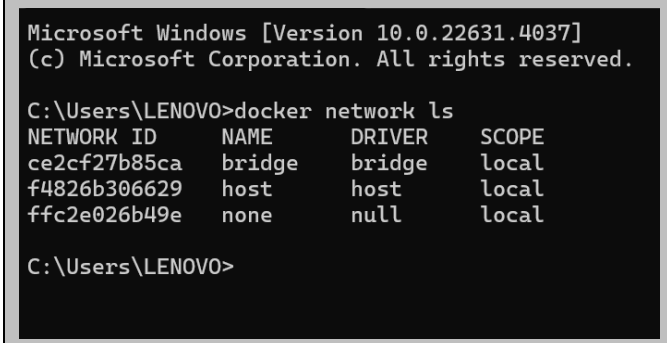
Docker provides three default networks:

- 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
```

```
C:\Users\LENOVO>docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "ce2cf27b85caf6353e0f002d6a821e3bc47e9130116c9b540423f2138e19f112",
        "Created": "2024-09-09T05:46:04.837053099Z",
        "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": {}
    }
]

C:\Users\LENOVO>
```

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
```

```
C:\Users\LENOVO>
C:\Users\LENOVO>docker network create my_bridge
8c2f2ccad1f8f4e8926140871cd7af10f0d4fc336fe0dc83fe2cabacceb3b08e
```

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
```

```
C:\Users\LENOVO>docker run -dit --name container1 --network my_bridge busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
3d1a87f2317d: Pull complete
Digest: sha256:34b191d63fbc93e25e275bfccf1b5365664e5ac28f06d974e8d50090fbb49f41
Status: Downloaded newer image for busybox:latest
582eb9d2dbfaab36049bfef575fce7d333413ae2dcdf8159dd84f4f6a3a2cab6

C:\Users\LENOVO>docker run -dit --name container2 --network my_bridge busybox
12b42c7d78630e3213c97fbd68c979a1bcc468f7218e9472a41b402a141c0862
```

2.3. Test Container Communication

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

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

```
C:\Users\LENOVO>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=0.218 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.062 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.067 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.069 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.062 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.066 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.069 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.078 ms
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.053 ms
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.075 ms
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.058 ms
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.073 ms
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.062 ms
64 bytes from 172.18.0.3: seq=13 ttl=64 time=0.079 ms
64 bytes from 172.18.0.3: seq=14 ttl=64 time=0.065 ms
64 bytes from 172.18.0.3: seq=15 ttl=64 time=0.062 ms
64 bytes from 172.18.0.3: seq=16 ttl=64 time=0.065 ms
64 bytes from 172.18.0.3: seq=17 ttl=64 time=0.063 ms
64 bytes from 172.18.0.3: seq=18 ttl=64 time=0.066 ms
64 bytes from 172.18.0.3: seq=19 ttl=64 time=0.073 ms
^C
--- container2 ping statistics ---
20 packets transmitted, 20 packets received, 0% packet loss
round-trip min/avg/max = 0.053/0.074/0.218 ms

What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug container1
    Learn more at https://docs.docker.com/go/debug-cli/

C:\Users\LENOVO>
```

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
```

```
C:\Users\LENOVO>docker run -d --name host_network_container --network host nginx
460a239e7ee5d4458ccc184fa79581d7016594b5b65a917dcf13b15821a4b2b3
```

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

```
C:\Users\LENOVO>docker network inspect host
[
    {
        "Name": "host",
        "Id": "f4826b30662950d03593478ce9e99e810d83fc0954d30b613fc12a861e677b5f",
        "Created": "2024-09-02T05:42:49.973566719Z",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": null
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "460a239e7ee5d4458ccc184fa79581d7016594b5b65a917dcf13b15821a4b2b3": {
                "Name": "host_network_container",
                "EndpointID": "5e5a7af9a907e44f1ff00c66b23ab06d58ea3bce29522fe4397673dc711ecba0",
                "MacAddress": "",
                "IPv4Address": "",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]

C:\Users\LENOVO>
```

Step 4: Disconnect and Remove Networks

4.1. Disconnect Containers from Networks

To disconnect container1 from my_bridge:

docker network disconnect my_bridge container1

```
C:\Users\LENOVO>docker network disconnect my_bridge container1
Error response from daemon: container 582eb9d2dbfaab36049bfef575fce7d333413ae2dcdf8159dd84f4f6a3a2cab6 is not connected to network my_bridge
```

```
C:\Users\LENOVO>docker network inspect my_bridge
[
    {
        "Name": "my_bridge",
        "Id": "8c2f2ccad1f8f4e8926140871cd7af10f0d4fc336fe0dc83fe2cabacceb3b08e",
        "Created": "2024-09-09T05:50:23.706214929Z",
        "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": {},
        "Options": {},
        "Labels": {}
    }
]

C:\Users\LENOVO>
```

4.2. Remove Networks

To remove the user-defined network:

```
docker network rm my_bridge
```

```
C:\Users\LENOVO>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
```

```
C:\Users\LENOVO>docker rm -f container1 container2 host_network_container
container1
container2
host_network_container
```