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
```

```
C:\Users\an626>docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
e95ba0221c1a   bridge    bridge    local
e0d81941894e   host      host      local
67739ef8f520   none      null      local
```

1.2. Inspect the Bridge Network

```
docker network inspect bridge
```

```
C:\Users\an626>docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "e95ba0221c1ad3051ac73efc723d1f115be5a6ce8cd11977db33830a53749
c80",
        "Created": "2024-09-09T06:09:40.122892751Z",
        "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
```

```
C:\Users\an626>docker network create my_bridge
54143f5713d2e899110062c9c1e6bd7ad31b3b4a9289d6ed860aa77147660a3f
```

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

```
C:\Users\an626>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
7a0447c012b1dfa2e18fbbd7fd5e6061b21a37d1ed2bacd7fb2a0ae0b68bca1d
```

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

```
C:\Users\an626>docker run -dit --name container2 --network my_bridge busybox
44d34b26d1e73329fd713ad369b72f62eac243c24d4ce3049ef80b493f9838fb
```

2.3. Test Container Communication

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

docker exec -it container1 ping container2

```
C:\Users\an626>
C:\Users\an626>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=0.112 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.155 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.132 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.408 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.122 ms
^C
--- container2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.112/0.185/0.408 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
```

```
C:\Users\an626>docker run -d --name host_network_container --network host nginx
aeca9b8f66807899bda8639e0c631dc5bddea822419554a1ef3e55730e888733
```

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\an626>docker run -d --name host_network_container --network host nginx
aeca9b8f66807899bda8639e0c631dc5bddea822419554a1ef3e55730e888733

C:\Users\an626>docker network inspect host
[
    {
        "Name": "host",
        "Id": "e0d81941894edfb17669410e2297e3e385803415b4c1a1a0ab217c017cd23332",
        "Created": "2023-11-20T04:58:21.405993393Z",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": []
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "aeca9b8f66807899bda8639e0c631dc5bddea822419554a1ef3e55730e888733": {
                "Name": "host_network_container",
                "EndpointID": "bb511bac054b93e52c06f2124fcb50a7e031b5eaf6cd881c2d0d368a3607
c157",
                "MacAddress": "",
                "IPv4Address": "",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

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\an626>docker network disconnect my_bridge container1
```

4.2. Remove Networks

To remove the user-defined network:

docker network rm my_bridge

```
C:\Users\an626>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\an626>docker rm -f container1 container2 host_network_container
container1
container2
host_network_container
```