Lab Exercise 4- Working with Docker Networking

Name: Aarushi
SAP ID: 500105028
Rollno.: R2142220004
Batch: DevSecOps B1:H

Step 1: Understanding Docker Default Networks

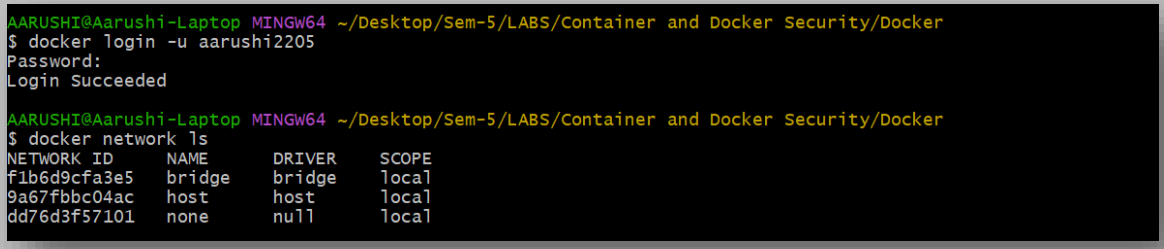
Docker provides three default networks:

- 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

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "f1b6d9cfa3e5febb1e6edb9aabb795f075dc392157bdf93eea07a210dd7af7bc",
        "Created": "2024-11-09T18:47:00.92186145Z",
        "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

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker network create Aarushi_bridge
16e0b1f3a3ffeac38de5c8db4a394f6e21231882c515ea3101e726253994db7a
```

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

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker run -dit --name container1 --network Aarushi_bridge busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
a46fbb00284b: Pull complete
Digest: sha256:768e5c6f5cb6db0794eec98dc7a967f40631746c32232b78a3105fb946f3ab83
Status: Downloaded newer image for busybox:latest
d761049d2c19b0a3b95bc1b17fac8f16a2870c6189dd519b5603b7913057d5b9
```

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

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker run -dit --name container2 --network Aarushi_bridge busybox
94a01334f92d9d1dda082eb3ea426783385d94f4567b19723da2e50f025af981

AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
94a01334f92d   busybox   "sh"      12 seconds ago   Up 7 seconds              container2
d761049d2c19   busybox   "sh"      12 minutes ago   Up 12 minutes             container1
```

2.3. Test Container Communication

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

docker exec -it container1 ping container2

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/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=0.643 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.153 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.199 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.217 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.138 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.136 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.135 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.125 ms
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.168 ms
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.165 ms
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.264 ms
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.139 ms
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.161 ms
64 bytes from 172.18.0.3: seq=13 ttl=64 time=0.157 ms
64 bytes from 172.18.0.3: seq=14 ttl=64 time=0.121 ms
64 bytes from 172.18.0.3: seq=15 ttl=64 time=0.128 ms
64 bytes from 172.18.0.3: seq=16 ttl=64 time=0.316 ms
64 bytes from 172.18.0.3: seq=17 ttl=64 time=0.230 ms
64 bytes from 172.18.0.3: seq=18 ttl=64 time=0.132 ms
64 bytes from 172.18.0.3: seq=19 ttl=64 time=0.144 ms
^C
--- container2 ping statistics ---
20 packets transmitted, 20 packets received, 0% packet loss
round-trip min/avg/max = 0.121/0.193/0.643 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

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker run -d --name host_network_container --network host nginx
762dde29df975689ba9c7718393f2c391f0bfa68712133080b8c3074e4389c93
```

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

docker network inspect host

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker network inspect host
[
    {
        "Name": "host",
        "Id": "9a67fbbc04ac53eb0cb1710b3448ba763a94fb399c94d667bfa7f846c836177b",
        "Created": "2024-08-30T06:11:04.974297228Z",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": null
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "762dde29df975689ba9c7718393f2c391f0bfa68712133080b8c3074e4389c93": {
                "Name": "host_network_container",
                "EndpointID": "a9462977ce0feffe83f26e78e8f17f49096748af6b803e1bbebe0c9c85c43035",
                "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
```

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker network disconnect Aarushi_bridge container1
```

4.2. Remove Networks

To remove the user-defined network:

```
docker network rm my_bridge
```

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker network disconnect Aarushi_bridge container2
```

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker network rm Aarushi_bridge
Aarushi_bridge
```

Step 5: Clean Up

Stop and remove all containers created during this exercise:

docker rm -f container1 container2

```
AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker rm -f container1 container2 host_network_container
container1
container2
host_network_container

AARUSHI@Aarushi-Laptop MINGW64 ~/Desktop/Sem-5/LABS/Container and Docker Security/Docker
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```