Name-Ansh Tyagi

sap id-500105272

Roll no- R2142220033

Batch- B2 Devops Non-Honors

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
```

```
Terminal
❯ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
65f74333500b   bridge    bridge    local
06edda09eda5   host      host      local
b4403dea57ac   none      null      local

~ ..... 10:53:33 PM
❯
```

1.2. Inspect the Bridge Network

docker network inspect bridge

```
❯ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "65f74333500b29fdffb2819eabd1ed241deb5d3e1e7f10f4bfa49dad2f2262f0",
        "Created": "2024-10-06T17:03:08.368251375Z",
        "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": "65535"
        },
        "Labels": {}
    }
]

~ ..... 10:54:18 PM
❯
```

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

```
Terminal
❯ docker network create my_bridge
6ab7ae1bc1750be1f3d7e6ce2681af91d66b451440e2faa0fe440d1046d6f7f7

~ ..... 10:56:00 PM
❯
```

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

```
Terminal
❯ docker run -dit --name container1 --network my_bridge busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
1523c6c3dc4c: Pull complete
Digest: sha256:768e5c6f5cb6db0794eec98dc7a967f40631746c32232b78a3105fb946f3ab83
Status: Downloaded newer image for busybox:latest
268b4ad7232db571b4b071fe7508f37b6c5af88d1686805e4dc6c2551e3c17a0

~ ..... 18s 10:57:04 PM
```

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

```
~ ..... 18s 10:57:04 PM
❯ docker run -dit --name container2 --network my_bridge busybox
21442ad5676b6fe997a4f1376b32ed4c9588c4ea08327c1bf8049d2dda53d80e

~ ..... 10:57:31 PM
❯
```

2.3. Test Container Communication

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

docker exec -it container1 ping container2

Terminal

```
❯ 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=0.354 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.262 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.181 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.203 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.281 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.246 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.282 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.238 ms
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.236 ms
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.203 ms
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.131 ms
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.230 ms
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.240 ms
64 bytes from 172.18.0.3: seq=13 ttl=64 time=0.499 ms
64 bytes from 172.18.0.3: seq=14 ttl=64 time=0.212 ms
64 bytes from 172.18.0.3: seq=15 ttl=64 time=0.190 ms
64 bytes from 172.18.0.3: seq=16 ttl=64 time=0.335 ms
64 bytes from 172.18.0.3: seq=17 ttl=64 time=0.292 ms
64 bytes from 172.18.0.3: seq=18 ttl=64 time=0.125 ms
```

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

Step 3: Create and Use a Host Network3.1. 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

```
Terminal

❯ docker run -d --name host_network_container --network host nginx
72d81f9bf85fbee2bf835a2413ec433dda11eaac0342580f71457a23dd736a35
```

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

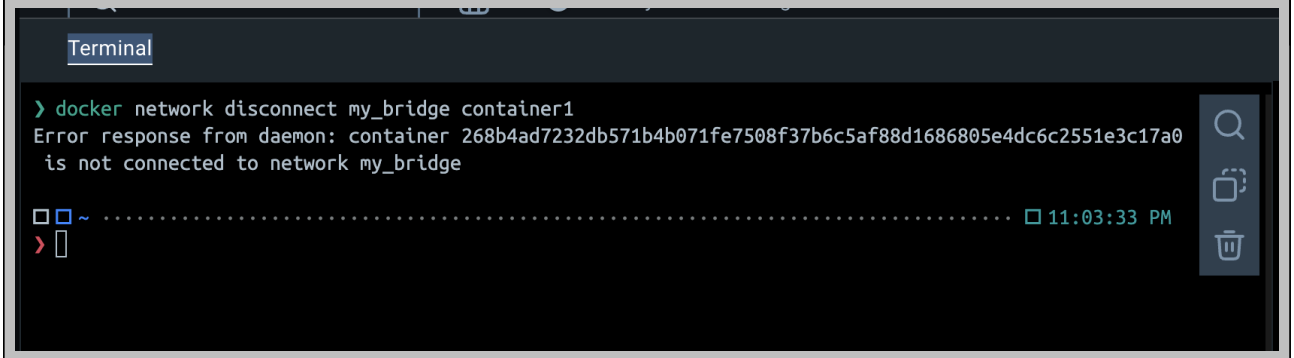
```
Terminal

❯ docker network inspect host
[
    {
        "Name": "host",
        "Id": "06edda09eda5f18bcb204b642a179a36b0691b0f4b7afceb38abdb2341d4bc7e",
        "Created": "2024-10-06T17:03:08.360463292Z",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": null
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "72d81f9bf85fbee2bf835a2413ec433dda11eaac0342580f71457a23dd736a35": {
                "Name": "host_network_container",
                "EndpointID": "905c0abb984904b0b1735f02d7482724323b02f2f49b1018d3a49d4b63a101c6",
                "MacAddress": "",
                "IPv4Address": "",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]

~ ..... 11:02:39 PM
❯
```

Step 4: Disconnect and Remove Networks

4.1. Disconnect Containers from Networks

To disconnect container1 from my_bridge:

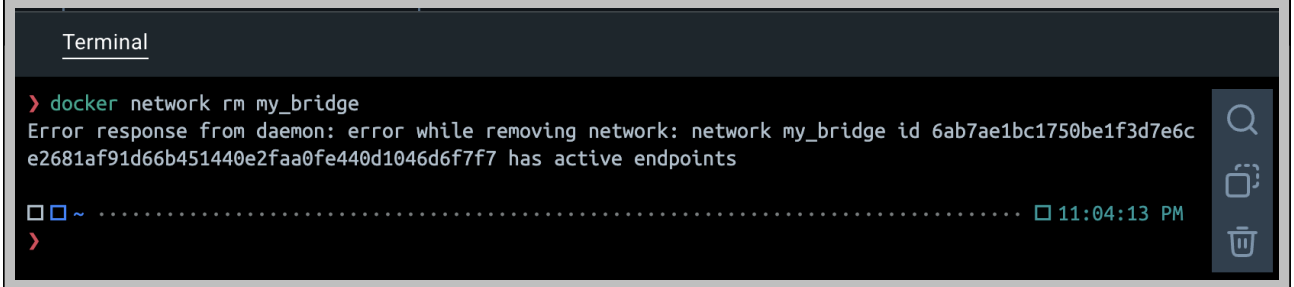
```
docker network disconnect my_bridge container1
```



4.2. Remove Networks

To remove the user-defined network:

```
docker network rm my_bridge
```



Step 4: Clean Up

Stop and remove all containers created during this exercise:

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

```
~ ..... 11:04:13 PM
❯ docker rm -f container1 container2
container1
container2

~ ..... 11:04:49 PM
❯
```