Name: Bhavesh Sanjiv Kapur

SapID: 500105635

Roll no. R2142220057

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
```

```
Last login: Fri Sep 13 07:45:02 on ttys000
(base) ➜  ~ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
ac25be573b9b   bridge    bridge    local
5baec11479bc   host      host      local
5c9757e832e4   none      null      local
(base) ➜  ~ _
```

1.2. Inspect the Bridge Network

docker network inspect bridged

```
(base) ➜  ~ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "ac25be573b9b53a39d273788e4a4e7467a951b7219da78151cdd0568966be86a",
        "Created": "2024-09-13T02:17:01.686213458Z",
        "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": {}
    }
]
(base) ➜  ~ _
```

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 bridgeBhavesh
```

```
(base) ➜  ~ docker network create bridgeBhavesh
a0bb3a8aa804081ba0542b5df71e00cf586ec7a70f871d48784324f5e7422aae
(base) ➜  ~ _
```

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
```

```
(base) ➜  ~ docker run -dit --name contBhavesh1 --network bridgeBhavesh busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
835f85a6d665: Pull complete
Digest: sha256:c230832bd3b0be59a6c47ed64294f9ce71e91b327957920b6929a0caa8353140
Status: Downloaded newer image for busybox:latest
62cd28006882c3b0c162262396e43913a0c2e2d36bfff904f7de24e4b90fdeb6
(base) ➜  ~ docker run -dit --name contBhavesh2 --network bridgeBhavesh busybox
3dbdf243cb9a74e2de384d0df4d682e62ed193968d5460680e67d4cdc0de7528
(base) ➜  ~ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
3dbdf243cb9a   busybox   "sh"      5 seconds ago    Up 4 seconds              contBhavesh2
62cd28006882   busybox   "sh"      35 seconds ago   Up 34 seconds             contBhavesh1
(base) ➜  ~ _
```

2.3. Test Container Communication

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

docker exec -it container1 ping container2

```
(base) ➜  ~ docker exec -it contBhavesh1 ping contBhavesh2
PING contBhavesh2 (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=1.177 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.230 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.231 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.259 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.300 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.400 ms
^C
--- contBhavesh2 ping statistics ---
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 0.230/0.432/1.177 ms
(base) ➜  ~ _
```

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

```
(base) ➜  ~ docker run -d --name hostContBhavesh --network host nginx
ab053707774e69cd43d9bfefd715f92c080ccabc67a7d9655bdaf5f57348b8b3
(base) ➜  ~ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
ab053707774e   nginx     "/docker-entrypoint.…"   2 seconds ago    Up 1 second               hostContBhavesh
3dbdf243cb9a   busybox   "sh"                     31 minutes ago   Up 31 minutes             contBhavesh2
62cd28006882   busybox   "sh"                     32 minutes ago   Up 32 minutes             contBhavesh1
(base) ➜  ~ _
```

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

```
(base) ➜  ~ docker network inspect host
[
    {
        "Name": "host",
        "Id": "5baec11479bc7e04d6a630bf0aa8242e039111343d817a9b567d89ebafcab3e2",
        "Created": "2023-11-04T09:07:35.683854167Z",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": []
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "ab053707774e69cd43d9bfefd715f92c080ccabc67a7d9655bdaf5f57348b8b3": {
                "Name": "hostContBhavesh",
                "EndpointID": "ece9fc824581dc24b98c5f065830b53352ed493b0a20de5ac2c466bc64dc71a1",
                "MacAddress": "",
                "IPv4Address": "",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
(base) ➜  ~ _
```

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
```

```
(base) ➜  ~ docker network disconnect bridgeBhavesh contBhavesh2
(base) ➜  ~ docker network rm bridgeBhavesh
bridgeBhavesh
(base) ➜  ~ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
ac25be573b9b   bridge    bridge    local
5baec11479bc   host      host      local
5c9757e832e4   none      null      local
(base) ➜  ~ _
```

Step 5: Clean Up

Stop and remove all containers created during this exercise:

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

```
(base) ➜  ~ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
ab053707774e   nginx     "/docker-entrypoint.…"   6 minutes ago    Up 6 minutes              hostContBhavesh
3dbdf243cb9a   busybox   "sh"                     37 minutes ago   Up 37 minutes             contBhavesh2
62cd28006882   busybox   "sh"                     38 minutes ago   Up 15 seconds             contBhavesh1
(base) ➜  ~ docker container stop 3db
3db
(base) ➜  ~ docker container stop 62c
62c
(base) ➜  ~ docker container stop ab0
ab0
(base) ➜  ~ docker rm -f 3db 62c ab0
3db
62c
ab0
(base) ➜  ~ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
(base) ➜  ~ _
```