UNIVERSITY OF PETROLEUM & ENERGY STUDIES

Dehradun

ACO LAB

Name- Harsha Agarwal
Sap id- 500096741
Roll no- R2142211158
Batch- B-4
Course- Btech Devops

Submitted to- Dr Hitesh Kumar Sharma

EXPERIMENT 4

AIM: Working with Docker Network

Step 1 - Create Network
The first step is to create a network using the CLI. This network will allow us to attach multiple containers which will be able to discover each other.

In this example, we're going to start by creating a *backend-network*. All containers attached to our backend will be on this network.

Task: Create Network

To start with we create the network with our predefined name.

docker network create backend-network

```
C:\Users\ABC>docker network create backend-network
edf63c6f775353d9a6f061a09042d7405d76636b414ea23bf26e3ada84b4916a
```

Task: Connect To Network
When we launch new containers, we can use the *--net* attribute to assign which network they should be connected to.

docker run -d --name=redis --net=backend-network redis

```
C:\Users\91983>docker run -d --name=redis --net=backend-network redis
e86c2a2e5325f69540b752254ce8fe1ad31c872b9f6cb3353c609a3733125ccb
```

In the next step we'll explore the state of the network.

Step 2 - Network Communication

Unlike using links, *docker network* behave like traditional networks where nodes can be attached/detached.

. Explore using the following two commands and you'll notice it no longer mentions other containers.

docker run --net=backend-network alpine ping -c1 redis

```
C:\Users\91983>docker run --net=backend-network alpine ping -c1 redis
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
96526aa774ef: Pull complete
Digest: sha256:eece025e432126ce23f223450a0326fbebde39cdf496a85d8c016293fc851978
Status: Downloaded newer image for alpine:latest
PING redis (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.564 ms

--- redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.564/0.564/0.564 ms
```

Step 3 - Connect Two Containers

Docker supports multiple networks and containers being attached to more than one network at a time.

For example, let's create a separate network with a Node.js application that communicates with our existing Redis instance.

Task

The first task is to create a new network in the same way.

docker network create frontend-network

```
C:\Users\91983>docker network create frontend-network
10d02e98b93857ba955be44462a8e1f8d16b91c5d86524efd625ca0457401440
```

When using the *connect* command it is possible to attach existing containers to the network.

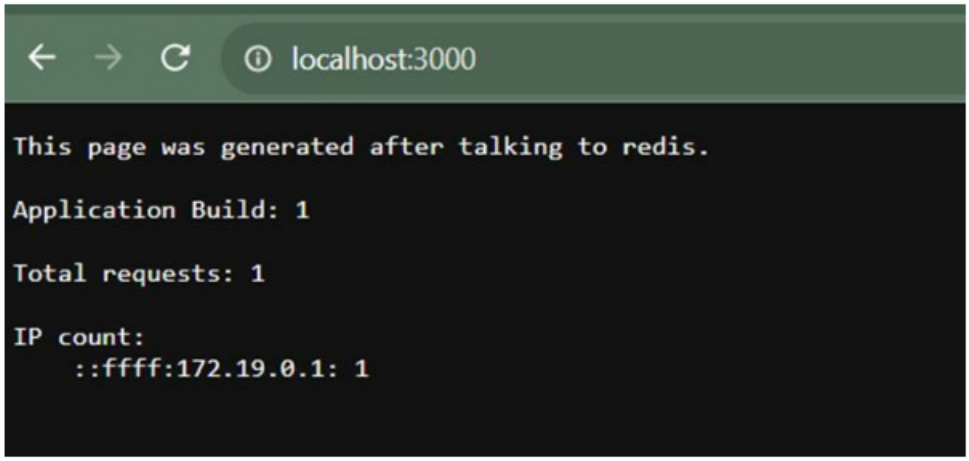
docker network connect frontend-network redis

```
C:\Users\91983>docker network connect frontend-network redis
```

When we launch the web server, given it's attached to the same network it will be able to communicate with our Redis instance.

docker run -d -p 3000:3000 --net=frontend-network katacoda/redis-node-docker-example

You can test it using curl docker:3000



Step 4 - Create Aliases

Links are still supported when using *docker network* and provide a way to define an Alias to the container name. This will give the container an extra DNS entry name and way to be discovered. When using --link the embedded DNS will guarantee that localised lookup result only on that container where the --link is used.

The other approach is to provide an alias when connecting a container to a network.

Connect Container with Alias

The following command will connect our Redis instance to the frontend-network with the alias of *db*.

docker network create frontend-network2

docker network connect --alias db frontend-network2 redis

When containers attempt to access a service via the name db, they will be given the IP address of our Redis container.

docker run --net=frontend-network2 alpine ping -c1 db

```
C:\Users\91983>docker run --net=frontend-network2 alpine ping -c1 db
PING db (172.20.0.2): 56 data bytes
64 bytes from 172.20.0.2: seq=0 ttl=64 time=0.496 ms

--- db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.496/0.496/0.496 ms
```

Step 5 - Disconnect Containers

With our networks created, we can use the CLI to explore the details.

The following command will list all the networks on our host.

docker network ls

```
C:\Users\91983>docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
27130245fc11   backend-network     bridge    local
463659457a3e   bridge              bridge    local
10d02e98b938   frontend-network    bridge    local
efc4ae4660af   frontend-network2   bridge    local
c2ce936c2fa1   host                host      local
1f3825773539   none                null      local
```

We can then explore the network to see which containers are attached and their IP addresses.

docker network inspect frontend-network

```
C:\Users\91983>docker network inspect frontend-network
[
    {
        "Name": "frontend-network",
        "Id": "10d02e98b93857ba955be44462a8e1f8d16b91c5d86524efd625ca0457401440",
        "Created": "2023-11-13T05:24:29.78327665Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "152b9ddc8e4dd6f49151321f24498f94d594332662def6c109ea2b291963befd": {
                "Name": "reverent_blackburn",
                "EndpointID": "5f1e53916c19c2f64df164f635bf918af7316517bb508b4c9d8eb8e18ec7a7da",
                "MacAddress": "02:42:ac:13:00:03",
                "IPv4Address": "172.19.0.3/16",
                "IPv6Address": ""
            },
            "e86c2a2e5325f69540b752254ce8fe1ad31c872b9f6cb3353c609a3733125ccb": {
                "Name": "redis",
                "EndpointID": "0811322f2a9a959140b82a835e1635e97db0e5e766b156fc9fe6eca313ebb2b6",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

The following command disconnects the redis container from the *frontend-network*.

docker network disconnect frontend-network redis

```
C:\Users\91983>docker network disconnect frontend-network redis
```

The following command deletes the created network:

“ docker network rm <network-name> ”

```
C:\Users\91983>docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
27130245fc11   backend-network     bridge    local
463659457a3e   bridge              bridge    local
10d02e98b938   frontend-network    bridge    local
efc4ae4660af   frontend-network2   bridge    local
c2ce936c2fa1   host                host      local
1f3825773539   none                null      local

C:\Users\91983>docker network rm backend-network
backend-network

C:\Users\91983>docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
463659457a3e   bridge              bridge    local
10d02e98b938   frontend-network    bridge    local
efc4ae4660af   frontend-network2   bridge    local
c2ce936c2fa1   host                host      local
1f3825773539   none                null      local

C:\Users\91983>docker network rm frontend-network
frontend-network

C:\Users\91983>docker network rm frontend-network2
frontend-network2

C:\Users\91983>docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
463659457a3e   bridge    bridge    local
c2ce936c2fa1   host      host      local
1f3825773539   none      null      local
```