

# EXPERIMENT 4

**Name- Radhika Rajesh Thakkar**

**SAPID- 500098212**

**Roll No- R2142211496**

**Batch-DevOps B4**

**AIM: Working with Docker Network**

**Steps to Complete:**

## Step 1 - Create Network

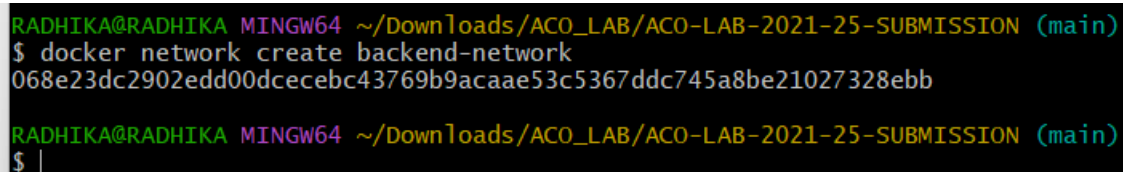
The first step is to create a network using the CLI. This network will allow us to attach multiple containers which will be able to discover each other.

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
```

A terminal window with a black background and green text. The prompt is 'RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO\_LAB/ACO-LAB-2021-25-SUBMISSION (main)'. The command '\$ docker network create backend-network' is entered, followed by a long alphanumeric string '068e23dc2902edd00dcecebc43769b9acaae53c5367ddc745a8be21027328ebb'. The prompt is repeated on the next line.

```
RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ docker network create backend-network
068e23dc2902edd00dcecebc43769b9acaae53c5367ddc745a8be21027328ebb
RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$
```

## 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
```

```
RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ docker run -d --name=redis1 --net=backend-network redis
2d70434932fa527d410e514e510770725a123fadac8c49bed36de075dfc766ac

RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$
```

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.

### Task: Explore

The first thing you'll notice is that Docker no longer assigns environment variables or updates the hosts file of containers. 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
```

```
RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ docker run --net=backend-network alpine ping -c1 redis1
PING redis1 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.108 ms

--- redis1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.108/0.108/0.108 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
```

```

RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ docker network create frontend-network
330127647cca061b75a2f6b6b95e50f2d70bcec9ce566e75853464a917a2e521

RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$

```

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

```
docker network connect frontend-network redis
```

```

RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ 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
```

```

$ docker run -d -p 3000:3000 --net=frontend-network katacoda/redis-node-docker-example
Unable to find image 'katakoda/redis-node-docker-example:latest' locally
latest: Pulling from katacoda/redis-node-docker-example
Image docker.io/katakoda/redis-node-docker-example:latest uses outdated schema1
manifest format. Please upgrade to a schema2 image for better future compatibility.
More information at https://docs.docker.com/registry/spec/deprecated-schema-v1/
12b41071e6ce: Pulling fs layer
a3ed95caeb02: Pulling fs layer
49a025abf7e3: Pulling fs layer
1fb1c0be01ab: Pulling fs layer
ae8c1f781cde: Pulling fs layer
db73207ad2ae: Pulling fs layer
446b13034c13: Pulling fs layer
1fb1c0be01ab: Waiting
ae8c1f781cde: Waiting
db73207ad2ae: Waiting
446b13034c13: Waiting
a3ed95caeb02: Download complete
12b41071e6ce: Verifying Checksum
12b41071e6ce: Download complete
12b41071e6ce: Pull complete
a3ed95caeb02: Pull complete
1fb1c0be01ab: Download complete
49a025abf7e3: Verifying Checksum
49a025abf7e3: Download complete
49a025abf7e3: Pull complete
1fb1c0be01ab: Pull complete
ae8c1f781cde: Verifying Checksum
ae8c1f781cde: Download complete
ae8c1f781cde: Pull complete
db73207ad2ae: Verifying Checksum
db73207ad2ae: Download complete
db73207ad2ae: Pull complete
446b13034c13: Verifying Checksum
446b13034c13: Download complete
446b13034c13: Pull complete
Digest: sha256:1aae9759464f00953c8e078a0e0d0649622fef9dd5655b1491f9ee589ae904b4
Status: Downloaded newer image for katacoda/redis-node-docker-example:latest
be0f0cffa08cf29834c39d8eab88de2280a1cb1ac5ff4915ae41e6dcbfdf4692

```

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
```

```
RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ docker network create frontend-network2
5ebdb45bbd7f1935b23bf0e55185e4913fc493d28438adacd5a184df617881ca

RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ |
```

```
docker network connect --alias db frontend-network2 redis
```

```
RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
h$ docker network connect --alias db frontend-network2 redis

RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$
```

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
```

```
RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ 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=0.073 ms

--- db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.073/0.073/0.073 ms

RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ |
```

## 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
```

```
RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ docker network ls
NETWORK ID          NAME                DRIVER             SCOPE
068e23dc2902        backend-network     bridge             local
ac5c77caf3aa        bridge              bridge             local
330127647cca        frontend-network    bridge             local
5ebdb45bbd7f        frontend-network2   bridge             local
098fbbdb7df9dc      host                host               local
dddfdbd975f8        none                null              local

RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$
```

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

```
docker network inspect frontend-network
```

```

RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ docker network inspect frontend-network
[
  {
    "Name": "frontend-network",
    "Id": "330127647cca061b75a2f6b6b95e50f2d70bcec9ce566e75853464a917a2e521"
  },
  {
    "Created": "2023-10-25T17:30:28.501638685Z",
    "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": {},
    "Options": {},
    "Labels": {}
  }
]
RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ |

```

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

```
docker network disconnect frontend-network redis
```

```

RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$ docker network disconnect frontend-network redis
RADHIKA@RADHIKA MINGW64 ~/Downloads/ACO_LAB/ACO-LAB-2021-25-SUBMISSION (main)
$

```