Name: Shweta Singh
Sap Id: 500098159
Course & Batch: Btech CSE(DevOps)-B4

Submitted to: Dr Hitesh Kumar Sharma

EXPERIMENT 4

AIM: Working with Docker Network

Steps to Complete:

Step 1 - Create Network

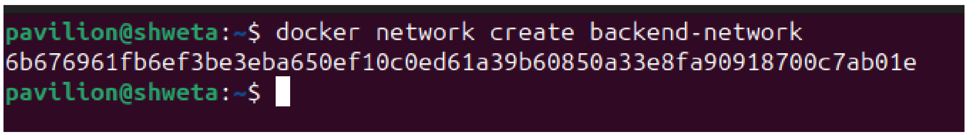
The first step is to create a network using the CLI. This network will allow us to attach multiple containers which will be able to discover each other.

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
```

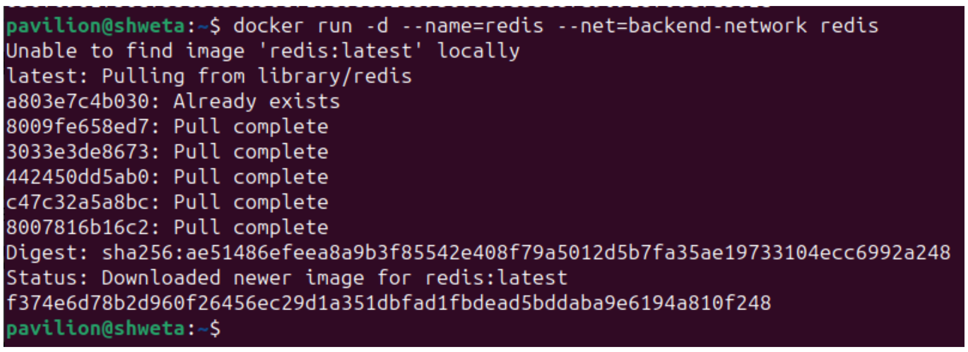


```
pavilion@shweta:~$ docker network create backend-network
6b676961fb6ef3be3eba650ef10c0ed61a39b60850a33e8fa90918700c7ab01e
pavilion@shweta:~$ 
```

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
```



```
pavilion@shweta:~$ docker run -d --name=redis --net=backend-network redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
a803e7c4b030: Already exists
8009fe658ed7: Pull complete
3033e3de8673: Pull complete
442450dd5ab0: Pull complete
c47c32a5a8bc: Pull complete
8007816b16c2: Pull complete
Digest: sha256:ae51486efeea8a9b3f85542e408f79a5012d5b7fa35ae19733104ecc6992a248
Status: Downloaded newer image for redis:latest
f374e6d78b2d960f26456ec29d1a351dbfad1fbdead5bddaba9e6194a810f248
pavilion@shweta:~$
```

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

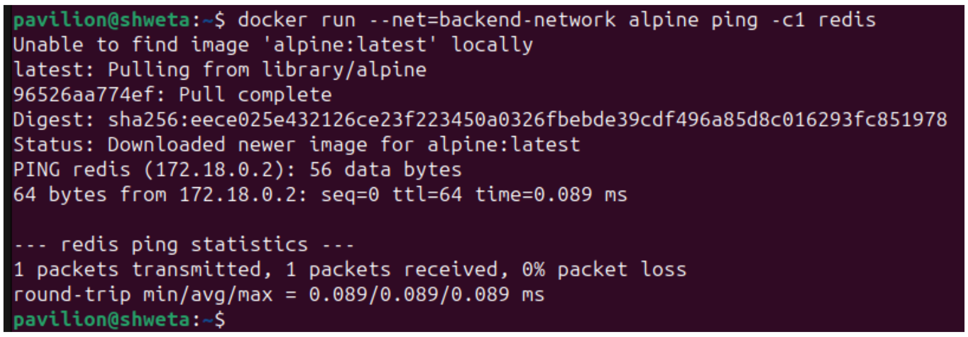
Step 2 - Network Communication

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
```



```
pavilion@shweta:~$ 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=0.089 ms

--- redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.089/0.089/0.089 ms
pavilion@shweta:~$
```

Step 3 - Connect Two Containers

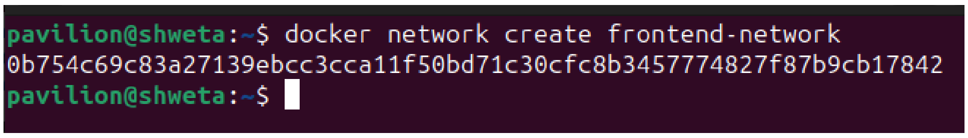
Docker supports multiple networks and containers being attached to more than one network at a time.

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`



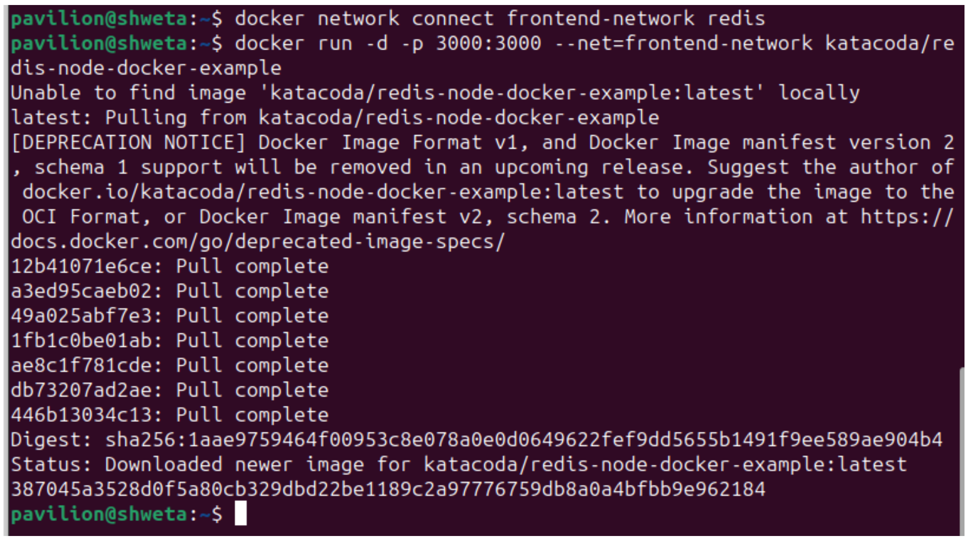
```
pavilion@shweta:~$ docker network create frontend-network
0b754c69c83a27139ebcc3cca11f50bd71c30cfc8b3457774827f87b9cb17842
pavilion@shweta:~$ 
```

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

`docker network connect frontend-network redis`

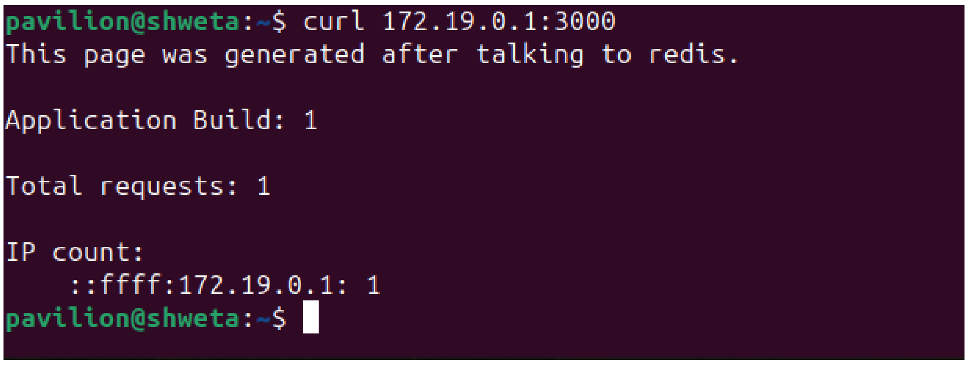
When we launch the web server, given it's attached to the same network it will be able to communicate with our Redis instance.

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



```
pavilion@shweta:~$ docker network connect frontend-network redis
pavilion@shweta:~$ docker run -d -p 3000:3000 --net=frontend-network katacoda/re
dis-node-docker-example
Unable to find image 'katacoda/redis-node-docker-example:latest' locally
latest: Pulling from katacoda/redis-node-docker-example
[DEPRECATION NOTICE] Docker Image Format v1, and Docker Image manifest version 2
, schema 1 support will be removed in an upcoming release. Suggest the author of
 docker.io/katacoda/redis-node-docker-example:latest to upgrade the image to the
 OCI Format, or Docker Image manifest v2, schema 2. More information at https://
docs.docker.com/go/deprecated-image-specs/
12b41071e6ce: Pull complete
a3ed95caeb02: Pull complete
49a025abf7e3: Pull complete
1fb1c0be01ab: Pull complete
ae8c1f781cde: Pull complete
db73207ad2ae: Pull complete
446b13034c13: Pull complete
Digest: sha256:1aae9759464f00953c8e078a0e0d0649622fef9dd5655b1491f9ee589ae904b4
Status: Downloaded newer image for katacoda/redis-node-docker-example:latest
387045a3528d0f5a80cb329dbd22be1189c2a97776759db8a0a4bfbb9e962184
pavilion@shweta:~$ 
```

You can test it using `curl docker:3000`



```
pavilion@shweta:~$ curl 172.19.0.1:3000
This page was generated after talking to redis.

Application Build: 1

Total requests: 1

IP count:
    ::ffff:172.19.0.1: 1
pavilion@shweta:~$
```

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

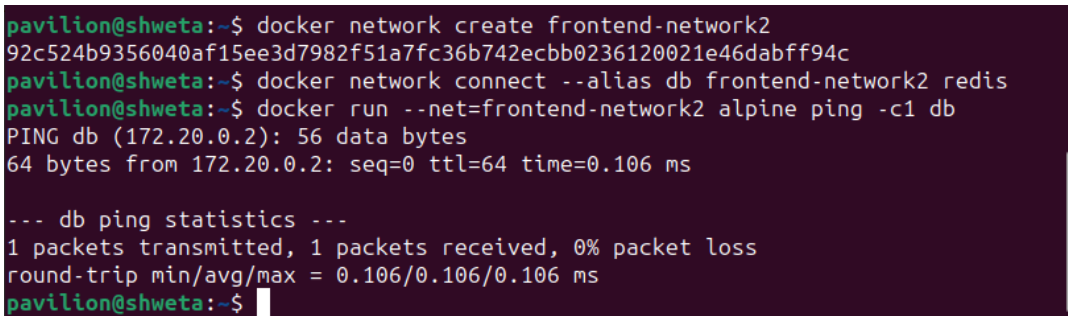
The following command will connect our Redis instance to the frontend-network with the alias of *db*.

```
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
```



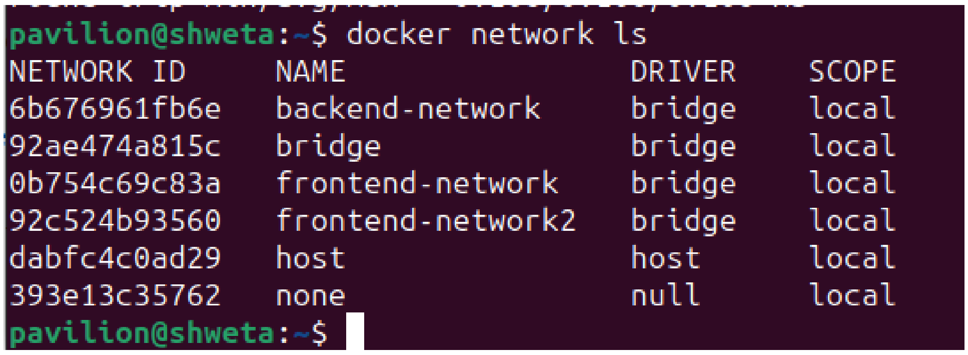
```
pavilion@shweta:~$ docker network create frontend-network2
92c524b9356040af15ee3d7982f51a7fc36b742ecbb0236120021e46dabff94c
pavilion@shweta:~$ docker network connect --alias db frontend-network2 redis
pavilion@shweta:~$ docker run --net=frontend-network2 alpine ping -c1 db
PING db (172.20.0.2): 56 data bytes
64 bytes from 172.20.0.2: seq=0 ttl=64 time=0.106 ms

--- db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.106/0.106/0.106 ms
pavilion@shweta:~$
```

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.

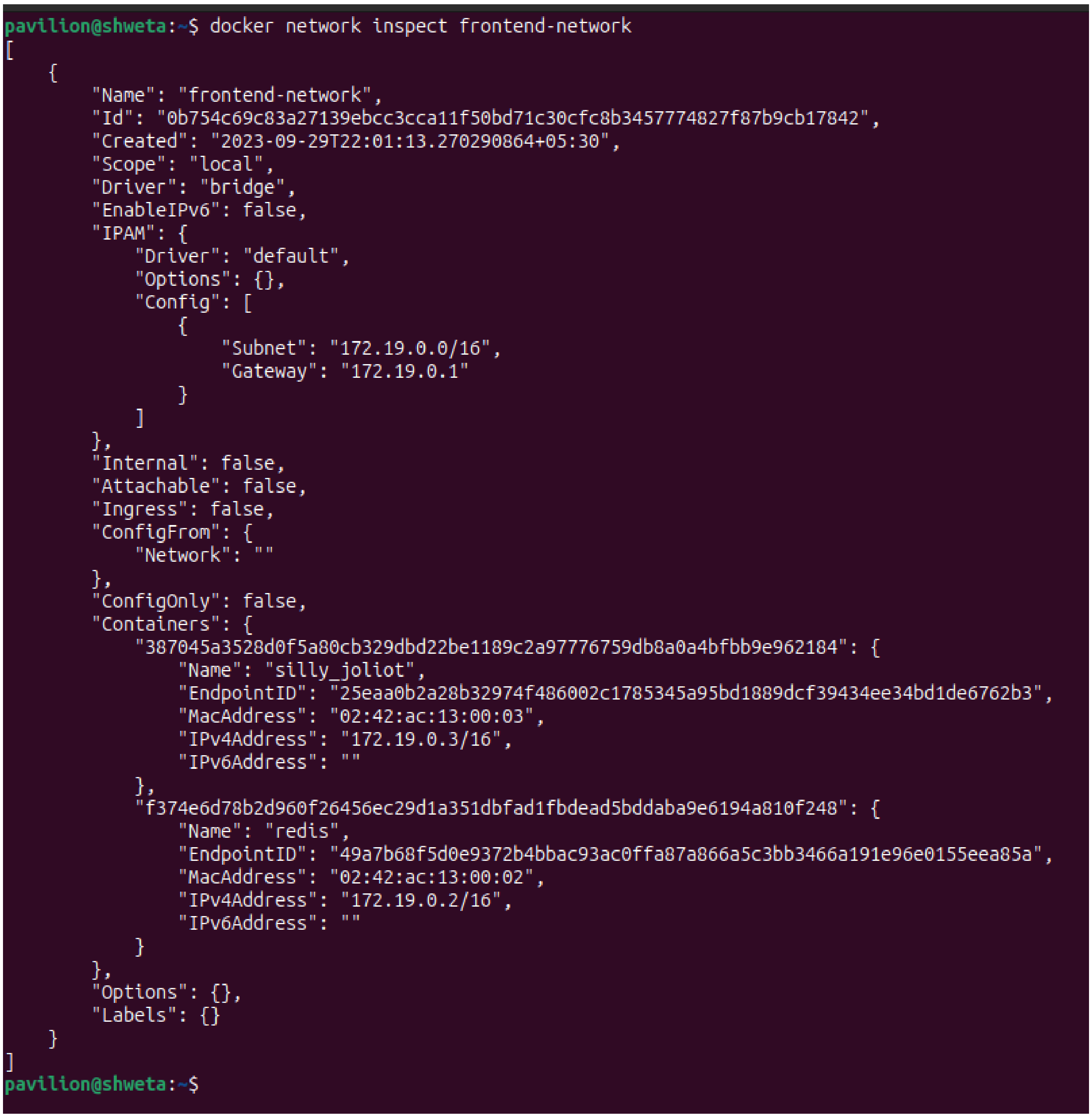
```
docker network ls
```



```
pavilion@shweta:~$ docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
6b676961fb6e   backend-network     bridge    local
92ae474a815c   bridge              bridge    local
0b754c69c83a   frontend-network    bridge    local
92c524b93560   frontend-network2   bridge    local
dabfc4c0ad29   host                host      local
393e13c35762   none                null      local
pavilion@shweta:~$
```

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

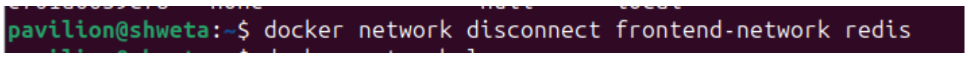
```
docker network inspect frontend-network
```



```
pavilion@shweta:~$ docker network inspect frontend-network
[
    {
        "Name": "frontend-network",
        "Id": "0b754c69c83a27139ebcc3cca11f50bd71c30cfc8b3457774827f87b9cb17842",
        "Created": "2023-09-29T22:01:13.270290864+05:30",
        "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": {
            "387045a3528d0f5a80cb329dbd22be1189c2a97776759db8a0a4bfbb9e962184": {
                "Name": "silly_joliot",
                "EndpointID": "25eaa0b2a28b32974f486002c1785345a95bd1889dcf39434ee34bd1de6762b3",
                "MacAddress": "02:42:ac:13:00:03",
                "IPv4Address": "172.19.0.3/16",
                "IPv6Address": ""
            },
            "f374e6d78b2d960f26456ec29d1a351dbfad1fbdead5bddaba9e6194a810f248": {
                "Name": "redis",
                "EndpointID": "49a7b68f5d0e9372b4bbac93ac0ffa87a866a5c3bb3466a191e96e0155eea85a",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
pavilion@shweta:~$
```

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

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



```
pavilion@shweta:~$ docker network disconnect frontend-network redis
```