Name-Anushka Chamoli

Batch-4 (DevOps)

Sap Id-500097354

Roll No.-R2142211336

EXPERIMENT 4

AIM: Working with Docker Network

Steps to Complete:

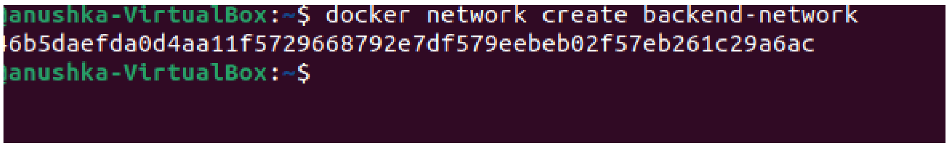
Step 1 - Create Network

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.

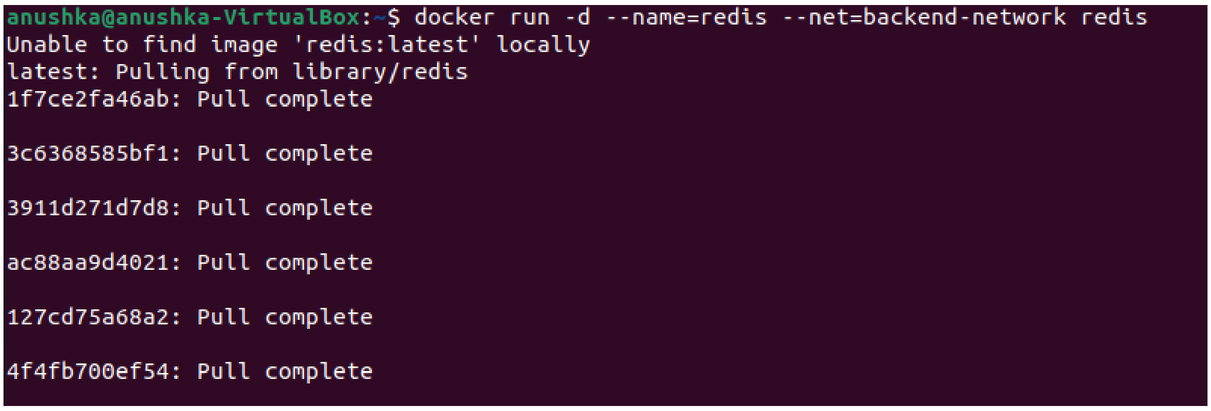
Task: Create Network

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


```
anushka-VirtualBox:~$ docker network create backend-network
6b5daefda0d4aa11f5729668792e7df579eebeb02f57eb261c29a6ac
anushka-VirtualBox:~$
```

Task: Connect To Network

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


```
anushka@anushka-VirtualBox:~$ docker run -d --name=redis --net=backend-network redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
1f7ce2fa46ab: Pull complete

3c6368585bf1: Pull complete

3911d271d7d8: Pull complete

ac88aa9d4021: Pull complete

127cd75a68a2: Pull complete

4f4fb700ef54: Pull complete
```

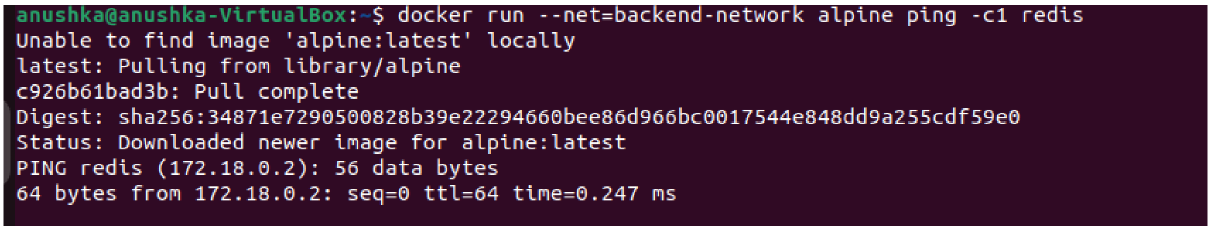
In the next step we'll explore the state of the network.

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.


```
anushka@anushka-VirtualBox:~$ docker run --net=backend-network alpine ping -c1 redis
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
c926b61bad3b: Pull complete
Digest: sha256:34871e7290500828b39e22294660bee86d966bc0017544e848dd9a255cdf59e0
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=0.247 ms
```

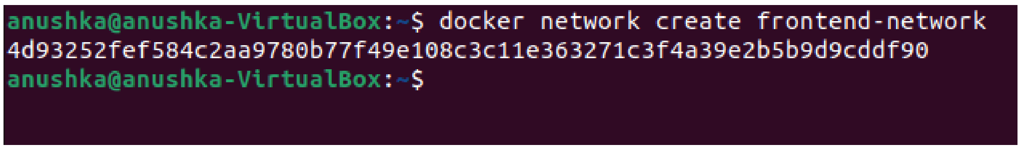
Step 3 - Connect Two Containers

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.

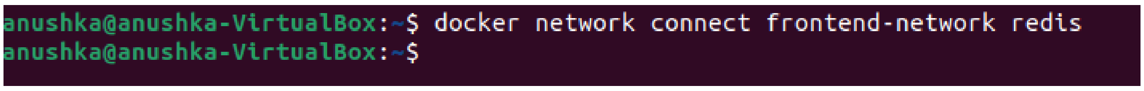
Task

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



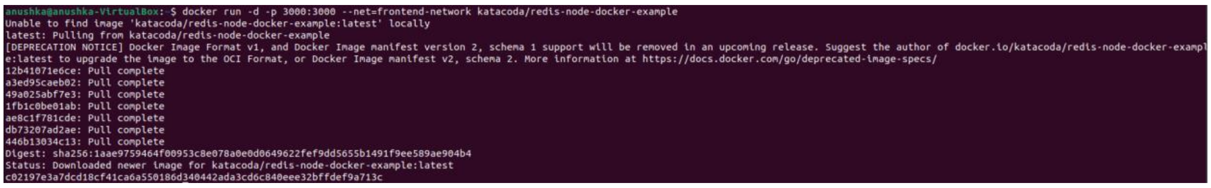
```
anushka@anushka-VirtualBox:~$ docker network create frontend-network
4d93252fef584c2aa9780b77f49e108c3c11e363271c3f4a39e2b5b9d9cddf90
anushka@anushka-VirtualBox:~$
```

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



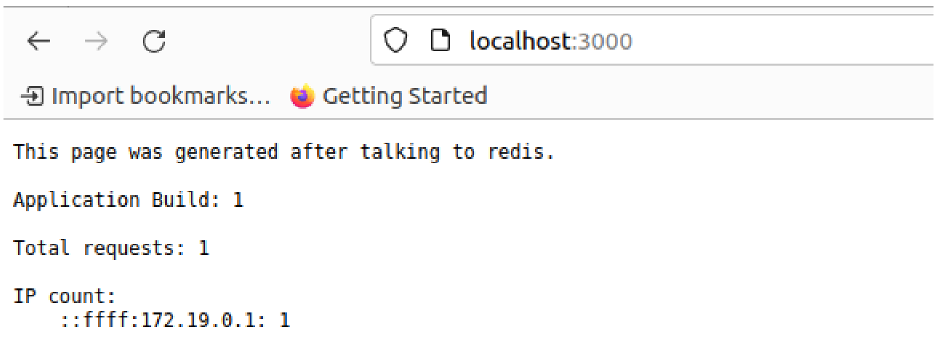
```
anushka@anushka-VirtualBox:~$ docker network connect frontend-network redis
anushka@anushka-VirtualBox:~$
```

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



```
anushka@anushka-VirtualBox:~$ docker run -d -p 3000:3000 --net=frontend-network katacoda/redis-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-exampl
e: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
c02197e3a7dcd18cf41ca6a550186d340442ada3cd6c840eee32bffdef9a713c
```

You can test it using `curl docker:3000`



```
This page was generated after talking to redis.

Application Build: 1

Total requests: 1

IP count:
    ::ffff:172.19.0.1: 1
```

Step 4 - Create Aliases

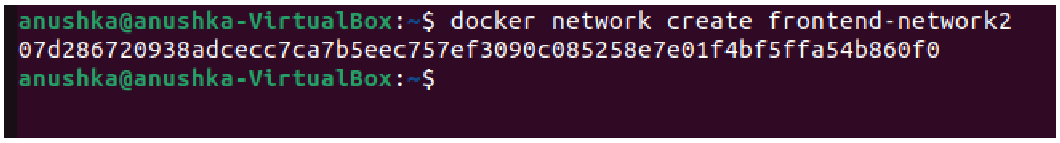
Links are still supported when using *docker network* and provide a way to define an Alias to the container name. This will give the container an extra DNS entry name and way to be discovered. When using --link the embedded DNS will guarantee that localised lookup result only on that container where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

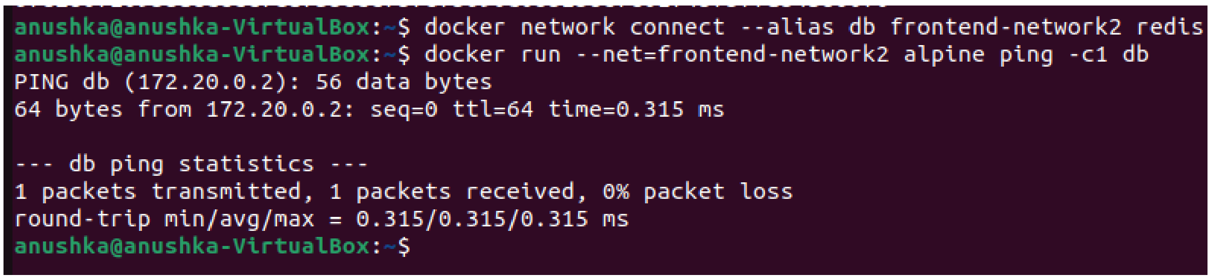


```
anushka@anushka-VirtualBox:~$ docker network create frontend-network2
07d286720938adcecc7ca7b5eec757ef3090c085258e7e01f4bf5ffa54b860f0
anushka@anushka-VirtualBox:~$
```

```
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
```



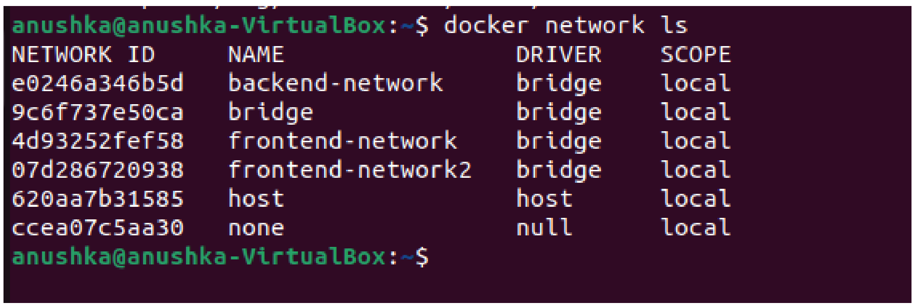
```
anushka@anushka-VirtualBox:~$ docker network connect --alias db frontend-network2 redis
anushka@anushka-VirtualBox:~$ docker run --net=frontend-network2 alpine ping -c1 db
PING db (172.20.0.2): 56 data bytes
64 bytes from 172.20.0.2: seq=0 ttl=64 time=0.315 ms

--- db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.315/0.315/0.315 ms
anushka@anushka-VirtualBox:~$
```

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.

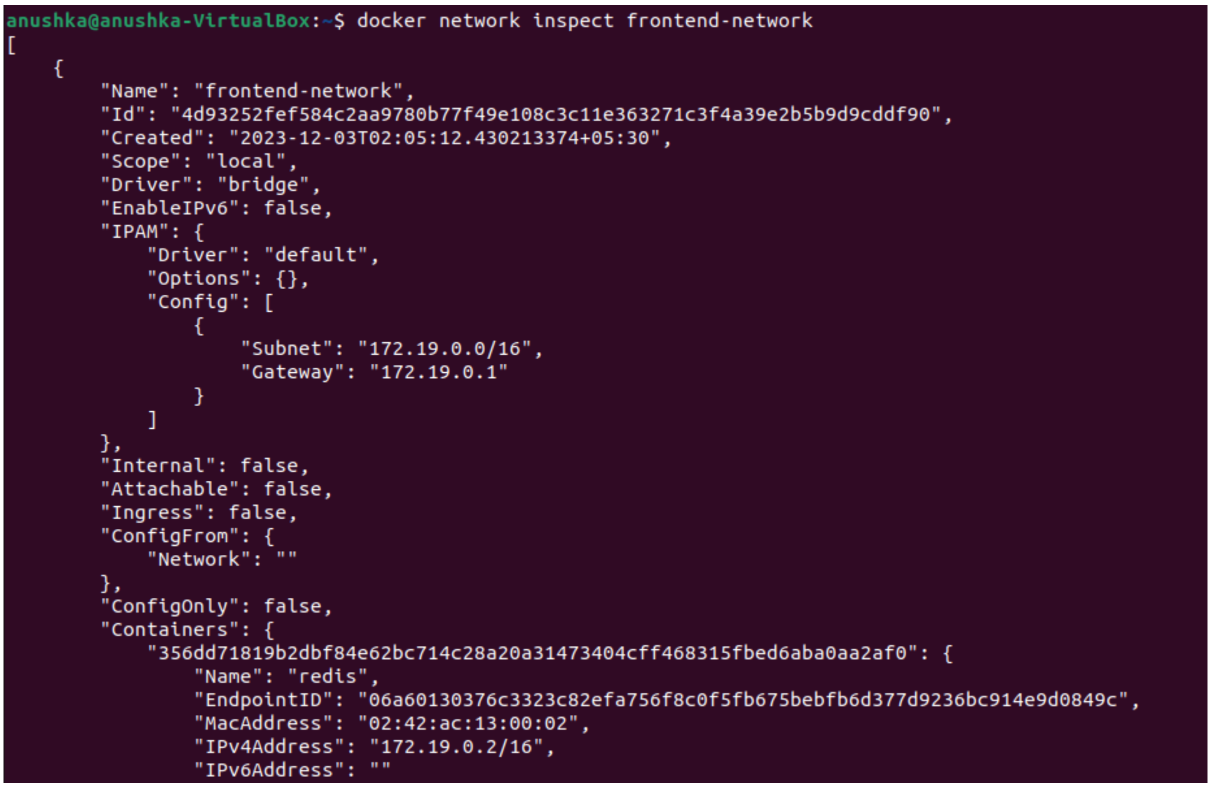
```
docker network ls
```



```
anushka@anushka-VirtualBox:~$ docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
e0246a346b5d   backend-network     bridge    local
9c6f737e50ca   bridge              bridge    local
4d93252fef58   frontend-network    bridge    local
07d286720938   frontend-network2   bridge    local
620aa7b31585   host                host      local
ccea07c5aa30   none                null      local
anushka@anushka-VirtualBox:~$
```

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

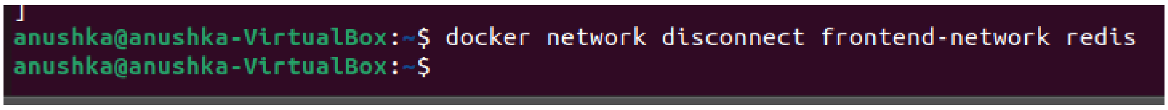
```
docker network inspect frontend-network
```



```
anushka@anushka-VirtualBox:~$ docker network inspect frontend-network
[
    {
        "Name": "frontend-network",
        "Id": "4d93252fef584c2aa9780b77f49e108c3c11e363271c3f4a39e2b5b9d9cddf90",
        "Created": "2023-12-03T02:05:12.430213374+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": {
            "356dd71819b2dbf84e62bc714c28a20a31473404cff468315fbed6aba0aa2af0": {
                "Name": "redis",
                "EndpointID": "06a60130376c3323c82efa756f8c0f5fb675bebfb6d377d9236bc914e9d0849c",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
```

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

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



```
]
anushka@anushka-VirtualBox:~$ docker network disconnect frontend-network redis
anushka@anushka-VirtualBox:~$
```