APPLICATION CONTAINERIZATION AND ORCHESTRATION LAB

STUDENT
SACHIN AGGARWAL
500097500
B4

SUBMITTED TO
DR.HITESH KUMAR SHARMA

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.

```
Sachins-Air:~ sachinaggarwal$ docker network create backend-network
9178baec259c2f023ec0731f40f5de333c647e03dd8222a675916e7c7491d300
```

Task: Connect To Network

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

```
Sachins-Air:~ sachinaggarwal$ docker run -d --name=redis --net=backend-network r
edis
2b6de0b604fa1ebad5c340db4ea2f7196179ec93496fdd0dea46c5cbbef3adde
```

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.

```
Sachins-Air:~ sachinaggarwal$ docker run --net=backend-network alpine ping -c1 r
edis
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
579b34f0a95b: 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=0.178 ms

--- redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.178/0.178/0.178 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.

```
Sachins-Air:~ sachinaggarwal$ docker network create frontend-network
132e47b95a950411f3b9524c64c7f1684e4c5eebaf2e3e66289a1d3f9b965a91
```

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

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

```
Sachins-Air:~ sachinaggarwal$ docker network connect frontend-network redis
Sachins-Air:~ sachinaggarwal$ 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
Image docker.io/katacoda/redis-node-docker-example:latest uses outdated schema1
manifest format. Please upgrade to a schema2 image for better future compatibili
ty. More information at https://docs.docker.com/registry/spec/deprecated-schema-
v1/
12b41071e6ce: Pull complete
a3ed95caeb02: Pull complete
49a025abf7e3: Pull complete
1fb1c0be01ab: Pull complete
ae8c1f781cde: Pull complete
```

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 results 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*.

```
Sachins-Air:~ sachinaggarwal$ docker network create frontend-network2
56002c0de6d54c302db5a59bea7810c309b87237d7ee49fcc1d0aae08091f6de
Sachins-Air:~ sachinaggarwal$
```

```
Sachins-Air:~ sachinaggarwal$ docker network connect --alias db frontend-network
2 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
```

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.

```
Sachins-Air:~ sachinaggarwal$ docker network ls
NETWORK ID     NAME               DRIVER    SCOPE
9178baec259c   backend-network    bridge    local
70d4bf9a85a8   bridge             bridge    local
132e47b95a95   frontend-network   bridge    local
56002c0de6d5   frontend-network2  bridge    local
806f6334e938   host               host      local
147d9f87c49f   none               null      local
```

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

```
Sachins-Air:~ sachinaggarwal$ docker network inspect frontend-network
[
    {
        "Name": "frontend-network",
        "Id": "132e47b95a950411f3b9524c64c7f1684e4c5eebaf2e3e66289a1d3f9b965a91
',
        "Created": "2023-09-30T17:14:10.966671423Z",
        "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": {}
    }
]
```

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

```
Sachins-Air:~ sachinaggarwal$ docker network disconnect frontend-network redis
```