Lab 7A. Docker Networking

In this lab exercise, you are going to learn about the single host Docker networking and specifically:

- How to examine the network configuration on a Docker host.

Bridge Networking

Following are the three networks created by default on a Docker host:

- bridge
- host
- none

You can examine the existing network configurations by using the Docker CLI, or using tools such as Portainer, if available:

```
docker network ls
```

```
NETWORK ID          NAME                DRIVER              SCOPE
b3d405dd37e4        bridge              bridge              local
7527c821537c        host                host                local
773bea4ca095        none                null                local
```

Creating a new network:

```
docker network create -d bridge mynet
```

Validate that the bridge network by name **`mynet`** is created using:

```
docker network ls
```

```
NETWORK ID          NAME                DRIVER              SCOPE
b3d405dd37e4        bridge              bridge              local
7527c821537c        host                host                local
4e0d9b1a39f8        mynet               bridge              local
773bea4ca095        none                null                local
```

```
docker network inspect mynet
```

```
[
   {
       "Name": "mynet",
       "Id":
"4e0d9b1a39f859af4811986534c91527146bc9d2ce178e5de02473c0f8ce62d5",
       "Created": "2018-05-03T04:44:19.187296148Z",
       "Scope": "local",
       "Driver": "bridge",
       "EnableIPv6": false,
       "IPAM": {
           "Driver": "default",
           "Options": {},
           "Config": [
               {
                   "Subnet": "172.18.0.0/16",
                   "Gateway": "172.18.0.1"
               }
           ]
       },
       "Internal": false,
       "Attachable": false,
       "Ingress": false,
       "ConfigFrom": {
           "Network": ""
       },
       "ConfigOnly": false,
       "Containers": {},
       "Options": {},
       "Labels": {}
   }
]
```

Launching Containers in Different Bridges

Launch two containers `nt01` and `nt02` in a **default** bridge network:

```
docker container run -idt --name nt01 alpine sh
docker container run -idt --name nt02 alpine sh
```

Launch two containers **nt03** and **nt04** in a **mynet** bridge network:

```
docker container run -idt --name nt03 --net mynet alpine sh
docker container run -idt --name nt04 --net mynet alpine sh
```

Now, let's examine if they can interconnect:

```
docker exec nt01 ifconfig eth0
docker exec nt02 ifconfig eth0
docker exec nt03 ifconfig eth0
docker exec nt04 ifconfig eth0
```

Following is a sample configuration from my host :

```
nt01 :  172.17.0.18
nt02 :  172.17.0.19
nt03 :  172.18.0.2
nt04 :  172.18.0.3
```

Note the container IP addresses and try to:

- ping from **nt01** to **nt02**
- ping from **nt01** to **nt03**
- ping from **nt03** to **nt04**
- ping from **nt03** to **nt02**

Replace/update IP addresses based on your setup:

```
docker exec nt01  ping 172.17.0.19
docker exec nt01  ping 172.18.0.2
docker exec nt03  ping 172.17.0.19
docker exec nt03  ping 172.18.0.2
```

You can observe that there are two different subnets/networks even though they run on the same host. **nt01** and **nt02** can connect with each other, and **nt03** and **nt04** can connect as well. However, a connection between containers attached to two different subnets is not possible.

Using None Network Driver

You can create a container with the **none** network, which would essentially mean that the container has no network access. Examine it using the following sequence of commands:

```
docker container run -idt --name nt05 --net none alpine sh

docker exec -it nt05 sh

ifconfig
```

Using Host Network Driver

If you assign a container to a host network, it is going to have access to all the interfaces on the host. Essentially, the container has no network namespace of its own. Examine host network with the following commands:

```
docker container run -idt --name nt05 --net host alpine sh
docker exec -it nt05 sh
ifconfig
```

Troubleshooting with Netshoot

To begin with, read about the netshoot utility here. Netshoot is an extremely useful utility that not only lets you examine the network configurations such as bridges, routing tables, port mappings, etc., but can also help you troubleshoot most of the container networking issues.

Launch netshoot and connect to the host network:

```
docker run -it --net host --privileged nicolaka/netshoot
```

Examine port mapping:

```
iptables -nvL -t nat
```

Traverse host port to container IP and port.

Observe Docker bridge and routing with the following commands:

```
brctl show

ip route show
```

Summary

In this lab, you explored the single host networking with Docker as a software. You also learned about the three default networks, how to create custom networks as well as how to get started with troubleshooting.