Section 6 - Docker Networking Basics

2 - Docker Network CLI commands

Objectives

- Practice the commands for controlling docker networks

Commands overview

- Show networks - `docker network ls`
- Inspect a network - `docker network inspect`
- Create a network - `docker network create --driver`
- Connect a container to a network - `docker network connect`
- Disconnect a container from a network - `docker network disconnect`

Example

- We will create a nginx container "web_server" to use it as a reference for this lecture.

```
docker container run --publish 80:80 --name web_server -d nginx
```

docker network ls

- Use `docker network ls` command to list all networks that have been created on a Docker host

```
# docker network ls
NETWORK ID          NAME        DRIVER     SCOPE
31b82a12c304        bridge      bridge     local
b0897e090893        host        host       local
3163420f3967        none        null       local
```

> Note:
>
> - Remember that sometime the `bridge` network is called `docker0`.
> - If we do not specify any network when we create a container, then the default `bridge` network is used.
> - The `bridge` (docker0) network is using the `bridge` driver.
> - The `bridge` driver is the default network driver when => container on a bridge network go through the NAT firewall to the physical network.

docker network inspect

- Use `docker network inspect` to display detailed information of a network.

```
docker network inspect bridge
    {
        "Name": "bridge", ...
        "Subnet": "172.17.0.0/16",
        "Gateway": "172.17.0.1" ...
        "Containers": {
            "e67416da3c6306e5353f3579827ec98dd2800c1514c6f778525f6a391b2c7
                "Name": "web_server",
                "EndpointID": "20f643a0412a83f624e0395f7ff7516d8535252376c
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16", ...
```

- We can see also the "Subnet": "172.17.0.0/16" and the default "Gateway": "172.17.0.1" use to route out to the physical network.
- In the "Containers" section we can see that there is a container "web_server" connected to this network and that the IP address is "172.17.0.2/16" .

Network driver - host (example) (1)

- The host network `--network host` is a special network that skips the virtual networking of Docker and attaches the container directly to the host interface.
- If you use the host network driver for a container, that container's network stack is not isolated from the Docker host.
- In this case, there NO protection from the NAT firewall that seats in the front of the virtual networks.
- For instance, if you run a container which binds to port 80 and you use host networking, the container's application will be available on port 80 on the host's IP address.
- There is better network performance, since there are not any virtual layers present.
- There are special applications that require access to the physical network interface.

Network driver - host (example) (2)

- Create a container with network `host` driver.

```
# docker container run --network host --name web_server_2 -d nginx
```

> Note: In this example nginx will open port 80 on the physical network interface of the Docker host.. If port 80 on the Docker host is already occupied, then the nginx container will not be successfully started.

Network driver - none (example)

- If you want to completely disable the networking stack on a container, you can use the `--network none` flag when starting the container. Within the container only the loopback device is created. The following example illustrates this.

```
# docker container run --network none  alpine  ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
```

> Notice that no eth0 is created.

docker network create

- create a user-defined bridge network

```
# docker network create my_app_net
8b3050f27566f4f4e5bb01136e81fddef1b4516b5bb143d7e2f412b33c3f5b7e

# docker network ls
NETWORK ID          NAME          DRIVER      SCOPE
31b82a12c304        bridge        bridge      local
b0897e090893        host          host        local
8b3050f27566        my_app_net    bridge      local
3163420f3967        none          null        local
```

> Note: The new network is created with a driver of bridge because this is the default driver.

docker network create (2)

- The next available subnet is automatically assigned to the new user-defined bridge network.

```
$ docker network inspect my_app_net
...
    "Subnet": "172.18.0.0/16",
...
```

> Note:
>
> - The "172.18.0.0/16" subnet is created and the default one is "172.17.0.0/16"
> - All these settings can be changed

docker network create (3)

- create a user-defined network with specific driver

```
# docker network create --driver bridge my-bridge-network
```

- The `--driver` option accepts `bridge` or `overlay` which are the built-in network drivers
- If we have installed a third party network driver, you can specify that DRIVER here also (like Weave).

docker network create (4)

- create a user-defined network with specific options

```
# docker network create --help

Options:
      --attachable           Enable manual container attachment
 ...
  -d, --driver string        Driver to manage the Network (default "bridge
      --gateway strings      IPv4 or IPv6 Gateway for the master subnet
      --ingress              Create swarm routing-mesh network
      --internal             Restrict external access to the network
      --ip-range strings     Allocate container ip from a sub-range
      --ipam-driver string   IP Address Management Driver (default "defaul
      --ipam-opt map         Set IPAM driver specific options (default map
      --ipv6                 Enable IPv6 networking
      --label list           Set metadata on a network
```

From the CLI documentation we can view all possible options that can be used for more advanced scenarios.

Connect a container to a user-defined network (1)

- We can use the `--network` option to connect a container to a user-defined network.

```
# docker container run -d --name new_nginx --network my_app_net nginx
```

Connect a container to a user-defined network (2)

- Verify that the new_nginx container is connected to the `my_app_net` network.

```
# docker network inspect my_app_net
[
 ...
        "Containers": {
            "c1442972be77395c41f86a42d358814816be8a1ce2436bb56a804abdb3ea8
                "Name": "new_nginx",
                "EndpointID": "66287cf5d53fc15bc25c7647993b545d92db6673b8f
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
...
```

Note:
The IP address of the new_nginx container is "172.18.0.2"

Connect/Disconnect a container to a user-defined network

- We don't have to start new containers, just like with the physical network where we can unplug and re-plug in Ethernet devices, we can also do the same with existing networks and existing containers.
- When we connect a container to a new network, then a new NIC is dynamically created

docker network connect (1)

- Use `docker network connect` to connect a container to a network.
- We can connect a container by name or by ID. Once connected, the container can communicate with other containers in the same network.

docker network connect (2)

```
# docker network connect my_app_net web_server
# docker container inspect web_server
[
    {
        "Id": "080b0a6e30ced4745d73a95324e5ba74183ef49462ef0cc69a6b5b46b92(
            "Networks": {
                "bridge": { ...
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16, ...
                },
                "my_app_net": { ...
                    "Gateway": "172.18.0.1",
                    "IPAddress": "172.18.0.3", ...
```

Notes: We can see that the "web_server" container has two IP addresses **172.17.0.2** and **172.18.0.3** related to the two different virtual networks that the container is connected to (bridge and my_app_net).

docker network disconnect

- Use the `docker network disconnect` command to disconnect a container from a network.

```
# docker network disconnect my_app_net web_server
# docker container inspect web_server
...
```

Docker Networks: Default Security

- In the physical world where we create virtual machines and hosts in a network, we often overexpose the ports and networking on our application servers.
- With Docker `bridge` virtual network, you only expose the ports on a host for which you specifically use the `--publish` option.

Exercise

- Ref:
- D_S6_L2_Docker_Network_CLI_commands_ex.md