Docker : le Bridge

Nous allons découvrir ensemble comment sont connectés nos conteneurs sur le réseau, créer des réseaux et utiliser ces réseaux pour nos conteneurs.

Etape 1 : Le réseau Bridge docker0

Lors de l'installation de Docker, trois réseaux sont créés automatiquement.

On peut voir ces réseaux avec la commande **docker network ls**. Un réseau de type bridge est créé :

```
vagrant@mydocker:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
46241e5b65b4        bridge              bridge              local
ff3e57e8ba69        host                host                local
f744a7f07df1        none                null                local
```

Le réseau Bridge est présent sur tous les hôtes Docker. Lors de la création d'un conteneur, si l'on ne spécifie pas un réseau particulier, les conteneurs sont connectés au Bridge **docker0**.

La commande **ifconfig** ou **ip a** fourni les informations sur le réseau (bridge).

```
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:43:6b:dd:48 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

La commande docker **network inspect bridge**, retourne les informations concernant ce réseau :

```
vagrant@mydocker:~$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "46241e5b65b4be694760c600fd86df6fa5fac5815e30f775bc8a3c8d689779f8",
        "Created": "2019-09-22T12:14:59.126516273Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```

Créons deux containers avec l'image **alpine**.

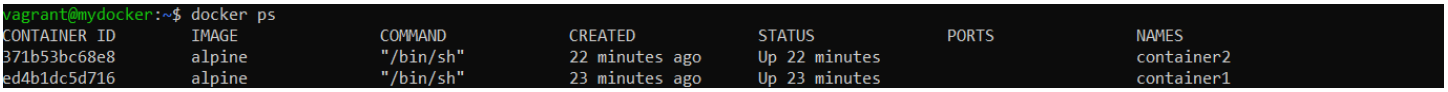
```
# docker run -itd --name=container1 alpine

3386a527aa08b37ea9232cbcace2d2458d49f44bb05a6b775fba7ddd40d8f92c

# docker run -itd --name=container2 alpine

94447ca479852d29aeddca75c28f7104df3c3196d7b6d83061879e339946805c
```

Visualisons les informations du réseau avec **docker network inspect** :

```
docker network inspect bridge
```

Les conteneurs sont connectés au Bridge par défaut **docker0** et peuvent communiquer entre eux par adresse IP, les conteneurs se trouvent alors, sur le même réseau.

Lister les conteneurs en cours d'exécution :



Connexion à l'instance **container1**

Connexion à l'instance **container2,** à partir d'un autre terminal

A partir du shell de **container1** vers un ping vers **container2**

Idem de **container2** vers **container1**

A partir du shell de **container1**, tapez la commande suivante

```
/ # ping container2
ping: bad address 'container2'
```

Docker ne prend pas en charge la découverte automatique du service sur le réseau Bridge par défaut.

Afin que les conteneurs puissent résoudre les adresses IP par les noms d'hôte des conteneurs on crée des réseaux définis par l'utilisateur.

Maintenant que nos tests sont terminés, nous pouvons supprimer nos instances :

```
vagrant@mydocker:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
371b53bc68e8        alpine              "/bin/sh"           About an hour ago   Up About an hour                        container2
ed4b1dc5d716        alpine              "/bin/sh"           About an hour ago   Up About an hour                        container1
vagrant@mydocker:~$ docker rm -f container1 container2
container1
container2
vagrant@mydocker:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
vagrant@mydocker:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
```

Etape 2 : Réseaux définis par l'utilisateur (Réseaux Bridgés)

Il est recommandé d'utiliser des réseaux Bridgés définis par l'utilisateur pour contrôler quels conteneurs peuvent communiquer entre eux, et ainsi permettre la résolution DNS des noms d'hôtes des conteneurs avec les adresses IP.

Un réseau Bridgé est le type de réseau le plus utilisé dans Docker.

Les réseaux Bridgés créés par les utilisateurs sont semblables au réseau bridge par défaut créé à l'installation de Docker **Docker0**.

Cependant de nouvelles fonctionnalités sont ajoutées, la gestion du DNS par exemple.

Création d'un réseau Bridge : test

Voir les réseaux :

```
docker network ls
```

```
vagrant@mydocker:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
46241e5b65b4        bridge              bridge              local
ff3e57e8ba69        host                host                local
f744a7f07df1        none                null                local
```

Création d'un réseau **test**

```
docker network create test
```

```
vagrant@mydocker:~$ docker network create test
fee17abe71dcb94e8579ce343ba0706079dd28f1c588b559018f57fd969329ac
```

Vérification

```
docker network ls
```

```
vagrant@mydocker:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
46241e5b65b4        bridge              bridge              local
ff3e57e8ba69        host                host                local
f744a7f07df1        none                null                local
fee17abe71dc        test                bridge              local
vagrant@mydocker:~$
```

Nous pouvons obtenir quelques informations sur ce réseau :

```
vagrant@mydocker:~$ docker network inspect test
[
    {
        "Name": "test",
        "Id": "fee17abe71dcb94e8579ce343ba0706079dd28f1c588b559018f57fd969329ac",
        "Created": "2019-09-22T14:38:09.340501273Z",
        "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": {}
    }
]
```

Connexion d'un container au réseau **test**, au moment de la création de l'instance

```
docker run --network test -itd --name container1 alpine
```

```
vagrant@mydocker:~$ docker run --network test -itd --name container1 alpine
b37519bf2ea3151e2ff41df68680e5f2ec377fb1f83abc56c596a6195bf977f2
```

Vérification

```
docker network inspect test
```

```
vagrant@mydocker:~$ docker network inspect test
[
    {
        "Name": "test",
        "Id": "fee17abe71dcb94e8579ce343ba0706079dd28f1c588b559018f57fd969329ac",
        "Created": "2019-09-22T14:38:09.340501273Z",
        "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": {
            "b37519bf2ea3151e2ff41df68680e5f2ec377fb1f83abc56c596a6195bf977f2": {
                "Name": "container1",
                "EndpointID": "fd0ff1df0266088d48b1da0d6e331942220d36661a986ccafd2c7759bdc945cd",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

Connexion d'un container (en cours d'exécution) au réseau **test**

```
docker run -itd --name container2 alpine
```

```
vagrant@mydocker:~$ docker run -itd --name container2 alpine
dffc33f74deddbbfec9aed1dd12c7bcdc28a9d26b4018c9fdc9ef406a926845b
```

```
docker network connect test container2
```

```
vagrant@mydocker:~$ docker network connect test container2
```

Vérification

```
docker network inspect test
```


```
    "ConfigOnly": false,
    "Containers": {
        "b37519bf2ea3151e2ff41df68680e5f2ec377fb1f83abc56c596a6195bf977f2": {
            "Name": "container1",
            "EndpointID": "fd0ff1df0266088d48b1da0d6e331942220d36661a986ccafd2c7759bdc945cd",
            "MacAddress": "02:42:ac:12:00:02",
            "IPv4Address": "172.18.0.2/16",
            "IPv6Address": ""
        },
        "dffc33f74deddbbfec9aed1dd12c7bcdc28a9d26b4018c9fdc9ef406a926845b": {
            "Name": "container2",
            "EndpointID": "e331a3f43cfa7842e1ae77b7adc4ab16dacbad16ea8c21b2c0705ed26d559eae",
            "MacAddress": "02:42:ac:12:00:03",
            "IPv4Address": "172.18.0.3/16",
            "IPv6Address": ""
        }
    },
    "Options": {},
    "Labels": {}
}
```


Testons le serveur DNS intégré :

```
docker run -it --name myping --network test debian /bin/bash
```

```
vagrant@mydocker:~$ docker run -it --name myping --network test debian /bin/bash
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
4a56a430b2ba: Pull complete
Digest: sha256:e25b64a9cf82c72080074d6b1bba7329cdd752d51574971fd37731ed164f3345
Status: Downloaded newer image for debian:latest
root@f0d3d6a38770:/#
```

Dans un autre terminal

```
docker network inspect test
```

```
    "Containers": {
        "b37519bf2ea3151e2ff41df68680e5f2ec377fb1f83abc56c596a6195bf977f2": {
            "Name": "container1",
            "EndpointID": "fd0ff1df0266088d48b1da0d6e331942220d36661a986ccafd2c7759bdc945cd",
            "MacAddress": "02:42:ac:12:00:02",
            "IPv4Address": "172.18.0.2/16",
            "IPv6Address": ""
        },
        "dffc33f74deddbbfec9aed1dd12c7bcdc28a9d26b4018c9fdc9ef406a926845b": {
            "Name": "container2",
            "EndpointID": "e331a3f43cfa7842e1ae77b7adc4ab16dacbad16ea8c21b2c0705ed26d559eae",
            "MacAddress": "02:42:ac:12:00:03",
            "IPv4Address": "172.18.0.3/16",
            "IPv6Address": ""
        },
        "f0d3d6a3877097b18661b688e02af6ee9e8d371779626dc246877af2b7049960": {
            "Name": "myping",
            "EndpointID": "8309d3f49b90540abc65d667f1fd7a62d85026f47e8cb7671c14503e4aab714e",
            "MacAddress": "02:42:ac:12:00:04",
            "IPv4Address": "172.18.0.4/16",
            "IPv6Address": ""
        }
    },
    "Options": {},
    "Labels": {}
}
```


Dans le bash de debian

```
root@f0d3d6a38770:/# ping container1
```

```
vagrant@mydocker:~$ docker run -it --name myping --network test debian /bin/bash
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
4a56a430b2ba: Pull complete
Digest: sha256:e25b64a9cf82c72080074d6b1bba7329cdd752d51574971fd37731ed164f3345
Status: Downloaded newer image for debian:latest
root@f0d3d6a38770:/# ping container1
PING container1 (172.18.0.2) 56(84) bytes of data.
64 bytes from container1.test (172.18.0.2): icmp_seq=1 ttl=64 time=0.265 ms
64 bytes from container1.test (172.18.0.2): icmp_seq=2 ttl=64 time=0.106 ms
64 bytes from container1.test (172.18.0.2): icmp_seq=3 ttl=64 time=0.108 ms
^C
--- container1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 0.106/0.159/0.265/0.075 ms
root@f0d3d6a38770:/#
```

Testez avec container2

Maintenant que nos tests sont terminés, nous pouvons supprimer le réseau **test**

```
vagrant@mydocker:~$ docker network rm test
Error response from daemon: error while removing network: network test id fee17abe71dcb94e8579ce343ba0706079dd28f1c588b559018f57fd969329ac has active endpoints
```

Le réseau **test** est toujours utilisé par des conteneurs, nous pouvons voir les conteneurs actifs sur ce réseau :

```
vagrant@mydocker:~$ docker network inspect test
[
    {
        "Name": "test",
        "Id": "fee17abe71dcb94e8579ce343ba0706079dd28f1c588b559018f57fd969329ac",
        "Created": "2019-09-22T14:38:09.340501273Z",
        "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": {
            "b37519bf2ea3151e2ff41df68680e5f2ec377fb1f83abc56c596a6195bf977f2": {
                "Name": "container1",
                "EndpointID": "fd0ff1df0266088d48b1da0d6e331942220d36661a986ccafd2c7759bdc945cd",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            },
            "dffc33f74deddbbfec9aed1dd12c7bcdc28a9d26b4018c9fdc9ef406a926845b": {
                "Name": "container2",
                "EndpointID": "e331a3f43cfa7842e1ae77b7adc4ab16dacbad16ea8c21b2c0705ed26d559eae",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

Il s'agit des conteneurs **container1** et **container2**

```
docker stop container1 && docker rm -f container1
```

```
vagrant@mydocker:~$ docker stop container1 && docker rm -f container1
container1
container1
```

```
docker stop container2 && docker rm -f container2
```

```
vagrant@mydocker:~$ docker stop container2 && docker rm -f container2
container2
container2
```

```
docker network rm test3
```

```
vagrant@mydocker:~$ docker network rm test
test
```

```
docker network ls
```

```
vagrant@mydocker:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
46241e5b65b4        bridge              bridge              local
ff3e57e8ba69        host                host                local
f744a7f07df1        none                null                local
```