Docker network

Les networks ont plusieurs utilités, créer un réseau overlay entre plusieurs machines par exemple, ou alors remplacer les links en permettant à tous les conteneurs d'un même réseau de communiquer par leurs noms.

Les types de réseaux

Nous avons de base, 4 types de networks :

- bridge : Crée un réseau interne pour vos conteneurs.
- host : Ce type de réseau permet au conteneurs d'avoir la même interface que l'hôte.
- none : Comme le nom l'indique, aucun réseau pour les conteneurs.
- overlay : Réseau interne entre plusieurs hôtes.

Bien évidemment il existe des plugins pour étendre ces possibilités.

La commande *docker info* affiche de nombreuses informations intéressantes sur une installation Docker.

Exécutez la commande docker info et recherchez la liste des plugins réseau.

```
[root@localhost ~]# docker info | grep -i -A 2 Plugins
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
```

Par défaut, nous avons déjà un réseau bridge, un réseau host et un réseau none. Nous ne pouvons pas créer de réseau *host* ou *none* supplémentaires.

```
[root@localhost ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
f17e455a7170        bridge              bridge              local
6488b8e4a815        host                host                local
984761d2b859        none                null                local
```

La sortie ci-dessus montre que le réseau *bridge* est associé au pilote *bridge*. Il est important de noter que le réseau et le pilote sont connectés, mais ils ne sont pas identiques. Dans cet exemple, le réseau et le pilote ont le même nom - mais ce n'est pas la même chose!

La sortie ci-dessus montre également que le réseau *bridge* est délimité localement. Cela signifie que le réseau existe uniquement sur cet hôte Docker. Cela est vrai pour tous les réseaux utilisant le pilote *bridge* - le pilote *bridge* fournit un réseau à hôte unique.

Tous les réseaux créés avec le pilote *bridge* sont basés sur un *bridge* Linux (alias un commutateur virtuel).

Avec la commande brctl répertoriez les ponts Linux sur votre hôte Docker.

```
[root@localhost ~]# brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242caebc5b3       no
```

```
[root@localhost ~]# ip addr show docker0
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
state DOWN group default
    link/ether 02:42:ca:eb:c5:b3 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
```

Création d'un network

Nous allons directement attaquer en créant un réseau, que nous appellerons test :

```
[root@localhost ~]# docker network create test
4d5ce9fcdd33e4c16189515af23c0987ec464b865927d8947e070b6486cb456c
[root@localhost ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
f17e455a7170        bridge              bridge              local
6488b8e4a815        host                host                local
984761d2b859        none                null                local
4d5ce9fcdd33        test                bridge              local
```

Nous pouvons obtenir quelques informations sur ce réseau :

```
[root@localhost ~]# docker network inspect test
[
    {
        "Name": "test",
        "Id":
"4d5ce9fcdd33e4c16189515af23c0987ec464b865927d8947e070b6486cb456c",
        "Created": "2020-03-22T17:18:05.990069864+01:00",
        "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,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Comme nous pouvons le voir, le réseau a comme sous-réseau 172.18.0.0/16, et comme gateway 172.18.0.1. Les conteneurs auront donc des IP attribuées entre 172.18.0.2 et 172.18.255.254 (172.18.255.255 étant le broadcast).

Nous pouvons évidemment choisir ce sous-réseau, les IP à attribuer aux conteneurs, etc... Pour ceci nous avons plusieurs options, comme *--subnet*, *--gateway*, ou *--ip-range* par exemple, ce qui donnerait :

```
[root@localhost ~]# docker network create --subnet 10.0.50.0/24 --gateway
10.0.50.254 --ip-range 10.0.50.0/28 test2
e05cddb74152738f5a1e78ccd99ef4f004abb518cacf792134e20511abcac8ef
[root@localhost ~]# docker network inspect test2
[
...
            "Config": [
                {
                    "Subnet": "10.0.50.0/24",
                    "IPRange": "10.0.50.0/28",
                    "Gateway": "10.0.50.254"
                }
...
]
```

Là vous avez créé un réseau 10.0.50.0/24 (donc de 10.0.50.0 à 10.0.50.255), mis la passerelle en 10.0.50.254, et un IPRange en 10.0.50.0/28 (donc une attribution de 10.0.50.1 à 10.0.50.14).

Pour chaque réseau créé, docker nous crée une interface sur l'hôte :

```
[root@localhost ~]# ifconfig
br-4d5ce9fcdd33: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 0.0.0.0
        ether 02:42:49:28:7d:43  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-e05cddb74152: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 10.0.50.254  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 02:42:ec:4f:20:5e  txqueuelen 0  (Ethernet)
        RX packets 18  bytes 1260 (1.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1260 (1.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 0.0.0.0
        ether 02:42:ca:eb:c5:b3  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Nous pouvons également choisir ce nom, avec l'argument -o :

```
[root@localhost ~]# docker network create --subnet 192.168.200.0/24 -o
"com.docker.network.bridge.name=br-test3" test3
a7a5a4a633cc50e30542876e452fee0bf7b9988ccaa12f90ef2ba76c6612d11c
[root@localhost ~]# ifconfig
…
br-test3: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.200.1  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 02:42:e7:59:b4:34  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
```

```
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Utilisation des networks avec les conteneurs

Pour attacher un réseau à un conteneur :

```
[root@localhost ~]# docker run -ti --network test3 --name ctest alpine sh
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:C0:A8:C8:02
          inet addr:192.168.200.2  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Nous sommes bien sur le réseau test3 que nous avons précédemment créé.

Nous pouvons également ajouter un réseau supplémentaire avec *docker network connect*.

Détachez-vous du conteneur avec Ctrl-p + Ctrl-q et revenez à l'invite de commande de la machine hôte.

```
[root@localhost ~]# docker network connect test ctest
[root@localhost ~]# docker attach ctest
Puis dans le conteneur :
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:C0:A8:C8:02
          inet addr:192.168.200.2  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth1      Link encap:Ethernet  HWaddr 02:42:AC:12:00:02
          inet addr:172.18.0.2  Bcast:0.0.0.0  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

L'un des avantages des networks, c'est qu'il n'est plus nécessaire de créer des liens entre les conteneurs, et donc deux conteneurs peuvent communiquer ensemble sans soucis. Pour faire un test, créons un conteneur *dest* puis un conteneur qui le ping qui s'appellera c*ping* :

```
[root@localhost ~]# docker run -d --network test3 --name dest alpine ping
8.8.8.8
```

NB : ping 8.8.8.8 est là juste pour faire tourner le conteneur en arrière plan.

```
[root@localhost ~]# docker run -ti --network test3 --name cping alpine ping
dest
PING dest (192.168.200.2): 56 data bytes
64 bytes from 192.168.200.2: seq=0 ttl=64 time=0.210 ms
64 bytes from 192.168.200.2: seq=1 ttl=64 time=0.183 ms
64 bytes from 192.168.200.2: seq=2 ttl=64 time=0.174 ms
```

```
64 bytes from 192.168.200.2: seq=3 ttl=64 time=0.180 ms
64 bytes from 192.168.200.2: seq=4 ttl=64 time=0.182 ms
^C
--- dest ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.174/0.185/0.210 ms
```

Essayons maintenant de comprendre comment une requête ping au conteneur *dest* à partir du conteneur c*ping* est-elle possible.

```
[root@localhost ~]# docker run -ti --network test3 --name cping alpine sh
/ # cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
192.168.200.3   4a1d06f256fa
/ # hostname
4a1d06f256fa
/ # ping -c 2 dest
PING dest (192.168.200.2): 56 data bytes
64 bytes from 192.168.200.2: seq=0 ttl=64 time=0.191 ms
64 bytes from 192.168.200.2: seq=1 ttl=64 time=0.172 ms

--- dest ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.172/0.181/0.191 ms
```

A part l'entrée à la fin, qui est en fait l'adresse IP résolvant le nom d'hôte du conteneur local, 4a1d06f256fa est l'ID du conteneur; il n'y a aucun signe d'entrée pour *dest*.

Ensuite, vérifions /etc/resolv.conf:

```
/ # cat /etc/resolv.conf
search localdomain
nameserver 127.0.0.11
options ndots:0
/ #
```

Cela renvoie ce que nous recherchons; comme vous pouvez le voir, nous utilisons un serveur de noms local: *nameserver 127.0.0.11*

```
/ # nslookup dest 127.0.0.11
Server:         127.0.0.11
Address:        127.0.0.11:53

Non-authoritative answer:

Non-authoritative answer:
Name:   dest
Address: 192.168.200.2
```

Détachez-vous du conteneur avec Ctrl-p + Ctrl-q et revenez à l'invite de commande de la machine hôte.

Inspectons de plus près le réseau *test3* :

```
[root@localhost ~]# docker network inspect test3
[
    {
        "Name": "test3",
        "Id":
"a7a5a4a633cc50e30542876e452fee0bf7b9988ccaa12f90ef2ba76c6612d11c",
        "Created": "2020-03-22T17:23:57.302158216+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.200.0/24"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Containers": {

"4a1d06f256fa786da4c97bef81bcb875d4ee21343967593653120d970808c01a": {
                "Name": "cping",
                "EndpointID":
"26180e7052fe7d1867706d2d2ec99c7f071782b2ee23bb6a2be9d94aa646bb63",
                "MacAddress": "02:42:c0:a8:c8:03",
                "IPv4Address": "192.168.200.3/24",
                "IPv6Address": ""
            },

"7d0b75f91d17ee3ab9868856f36aaa7918467ef66a7f5e78a1d6d88b4e62592d": {
                "Name": "dest",
                "EndpointID":
"ebc7db3f39bc539fa2956dbeb619847852da1570b5a6f8eabd3a91cbd879bf1c",
                "MacAddress": "02:42:c0:a8:c8:02",
                "IPv4Address": "192.168.200.2/24",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.name": "br-test3"
        },
        "Labels": {}
    }
]
```

Comme vous pouvez le voir, il contient des informations sur l'adressage réseau utilisé dans la section IPAM, ainsi que des détails sur chacun des deux conteneurs exécutés sur le réseau.

La gestion des adresses IP (IPAM) est un moyen de planifier, de suivre et de gérer les adresses IP au sein du réseau. IPAM possède à la fois des services DNS et DHCP, de sorte que chaque service est notifié des changements dans l'autre. Par exemple, DHCP attribue une adresse à un conteneur. Le service DNS est ensuite mis à jour pour renvoyer l'adresse IP attribuée par DHCP chaque fois qu'une recherche est effectuée sur ce conteneur.

Nettoyage:

Maintenant que nos tests sont terminés, nous pouvons supprimer nos réseaux et conteneurs :

```
[root@localhost ~]# docker container stop $(docker container ls -q)
[root@localhost ~]# docker container prune
[root@localhost ~]# docker network prune
```