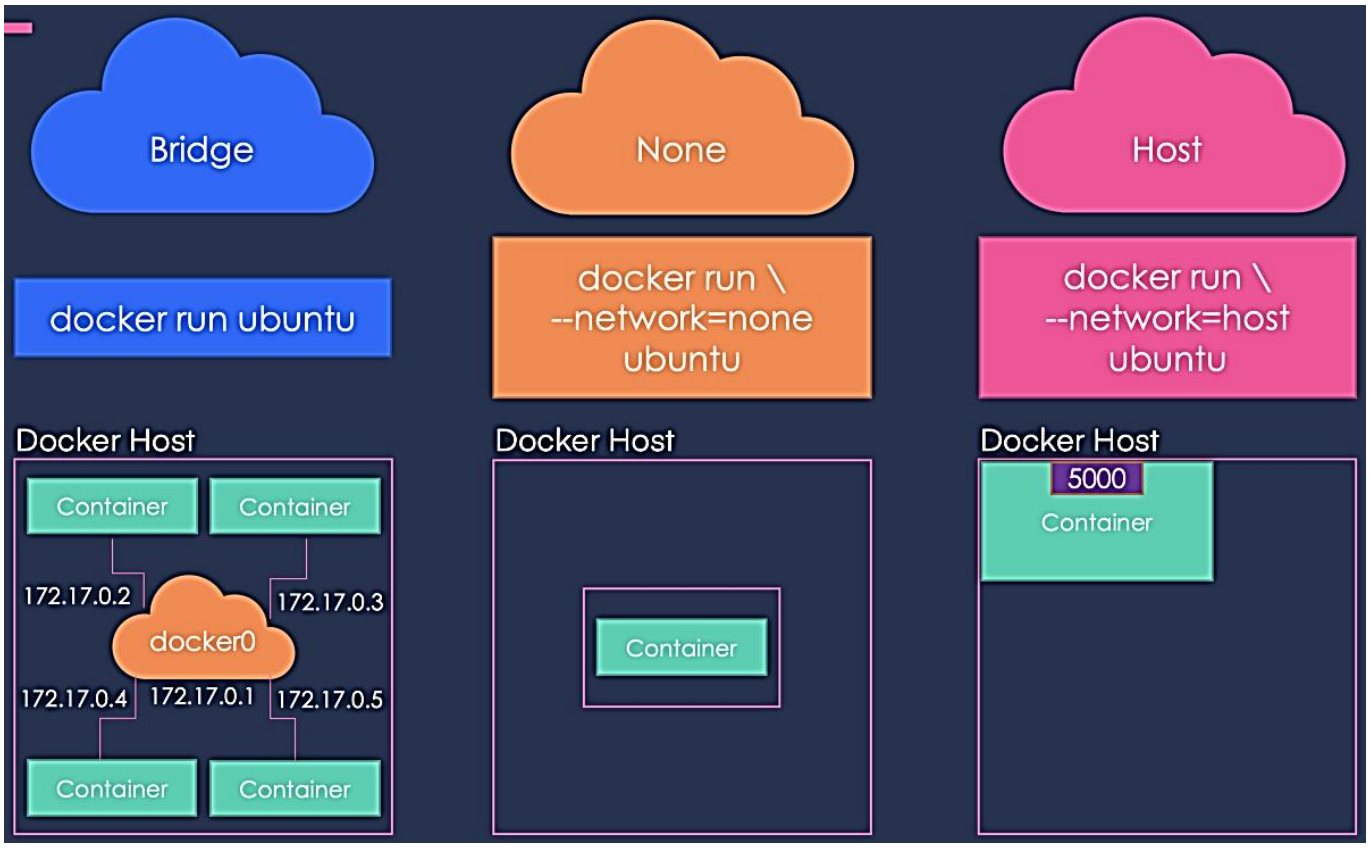
Pour que les conteneurs Docker puissent communiquer entre eux mais aussi avec le monde extérieur, via la machine hôte, une couche réseau est nécessaire. Cette couche réseau permet d'isoler des conteneurs et de créer des applications Docker qui fonctionnent ensemble de manière sécurisée.

Il existe 3 grands types de réseau sur Docker :

- Le réseau de type « **Bridge** »
- Le réseau de type « **None** »
- Le réseau de type « **Host** »



Lors de l'installation de Docker, 3 réseaux sont créés par défaut :

```
root@debian-docker:~# docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
36c710b4d6da   bridge        bridge    local
f65789398574   host          host      local
514b752cc29d   nextcloud-aio bridge    local
262c18388404   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, le conteneur est connecté au Bridge « **docker0 »**. Ce réseau **bridge** permet de fournir un **réseau par défaut**, **172.17.0.0/16 par défaut**, sur lequel seront connectés les conteneurs, ainsi qu'une **passerelle par défaut**, **172.17.0.1**, gérée par l'ordinateur sur lequel est installé Docker pour accéder au reste du réseau et éventuellement à Internet.

En saisissant « ip a » dans la console Debian, on obtient ceci :

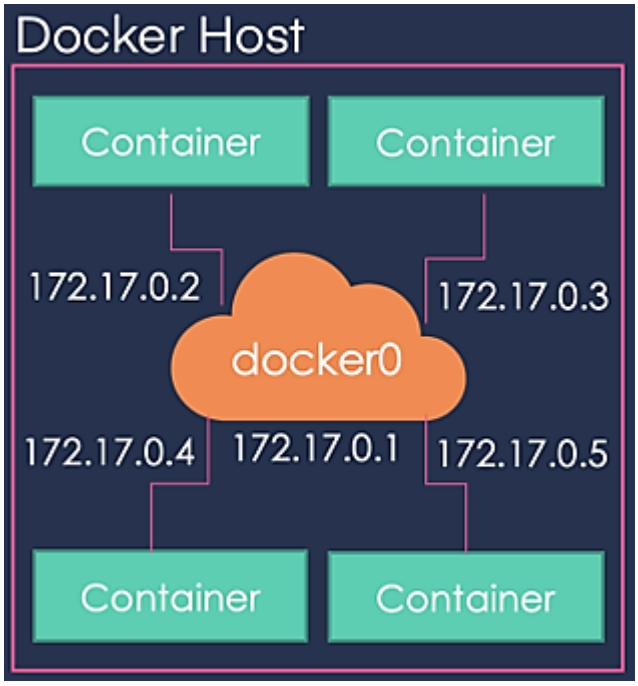
```
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfif
    link/ether 36:83:51:7d:90:0d brd ff:ff:ff:ff:ff:ff
    altname enp0s18
    inet 192.168.168.20/24 brd 192.168.168.255 scope global ens
       valid_lft forever preferred_lft forever
    inet6 fe80::3483:51ff:fe7d:900d/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc no
    link/ether 02:42:0d:e8:fa:4b 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
    inet6 fe80::42:dff:fee8:fa4b/64 scope link
       valid_lft forever preferred_lft forever
4: br-514b752cc29d: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    link/ether 02:42:a1:d5:dc:26 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-514b7
       valid_lft forever preferred_lft forever
    inet6 fe80::42:a1ff:fed5:dc26/64 scope link
       valid_lft forever preferred_lft forever
```

« ens18 » correspond à l'interface réseau de notre machine virtuelle Debian qui est connectée à l'hôte (le serveur Proxmox).

« docker0 » au réseau Docker « Bridge » créé automatiquement lors de l'installation de Docker (réseau par défaut pour les conteneurs).

Br-514b752… » correspond à un réseau « Bridge » créé spécialement pour des conteneurs qui doivent communiquer entre eux.

LES DIFFERENTS TYPES DE RESEAU DOCKER



1. Le réseau de type « BRIDGE »

Docker, une fois installé, crée automatiquement un réseau nommé « **bridge »** connecté à l'interface réseau **docker0.**

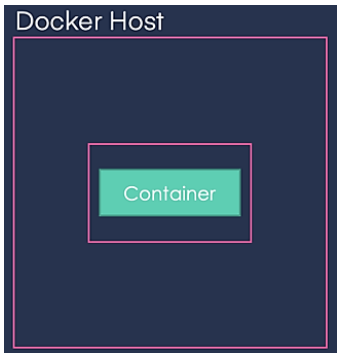
Chaque nouveau conteneur Docker est automatiquement connecté à ce réseau sauf si un réseau personnalisé est spécifié. Le **réseau bridge est le type de réseau le plus couramment utilisé**. Il est limité aux conteneurs d'un hôte unique exécutant le moteur Docker.

Les conteneurs qui utilisent le driver « Bridge » ne peuvent communiquer qu'entre eux. Pour être accessibles depuis l'extérieur, **un mappage de port est obligatoire**.

Exemple de mappage de port lors de la création d'un conteneur « HTTPD » (Apache) :

docker run -tid -p 8000:80 --name web httpd

L'ajout de l'argument **-p 8000:80** permet de rediriger les paquets du port hôte 8000 vers le port 80 du conteneur.



2. Le réseau de type « None »

En mode « none », le conteneur n'est connecté à aucune interface réseau.

C'est un type de réseau permettant **d'interdire toute communication interne et externe avec votre conteneur** car votre conteneur sera dépourvu de toute interface réseau (sauf l'interface loopback). Ce type de réseau peut être utile pour connecter un conteneur web à une base de données par exemple.

3. Le réseau de type « Host »

Ce type de réseau permet aux conteneurs **d'utiliser la même interface réseau que l'hôte**.

Il supprime donc l'isolation réseau entre les conteneurs. **Les conteneurs seront donc accessibles de l'extérieur**.

Il existe d'autres types de réseau sur Docker qui ne feront pas l'objet d'une présentation dans ce document (réseau de type « **Macvlan** » et réseau de type « **Overlay** ».

LES PRINCIPALES COMMANDES LIEES A L'UTILISATION DES RESEAUX DOCKER

1. Créer un réseau Docker nommé « monréseau » et lui affecter le type « Bridge » :

docker network create --driver bridge monréseau

```
root@debian-docker:~# docker network create --driver bridge monréseau
fa1a2f832b184d5c4df2870eab6297ca6955d60ddee63d76b0d8fdbb0b923cb0
```

2. Inspecter un réseau Docker :

docker network inspect monréseau

```
root@debian-docker:~# docker network inspect monréseau
[
    {
        "Name": "monréseau",
        "Id": "fa1a2f832b184d5c4df2870eab6297ca6955d60ddee63d76b0d8fdbb0b923cb0",
        "Created": "2023-03-18T11:22:46.754658446+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
```

L'interface réseau est créée en mode « bridge » avec un masque par défaut et une passerelle par défaut.

Dans cet exemple, Docker a créé le réseau de type bridge « monréseau » avec un adressage IP de type **172.19.0.0/16** car il existait déjà un autre réseau bridgé en 172.18.0.0/16.

3. Lister les réseaux Docker présents :

docker network ls

```
root@debian-docker:~# docker network ls
NETWORK ID     NAME           DRIVER    SCOPE
36c710b4d6da   bridge         bridge    local
f65789398574   host           host      local
fa1a2f832b18   monréseau      bridge    local
514b752cc29d   nextcloud-aio  bridge    local
262c18388404   none           null      local
```

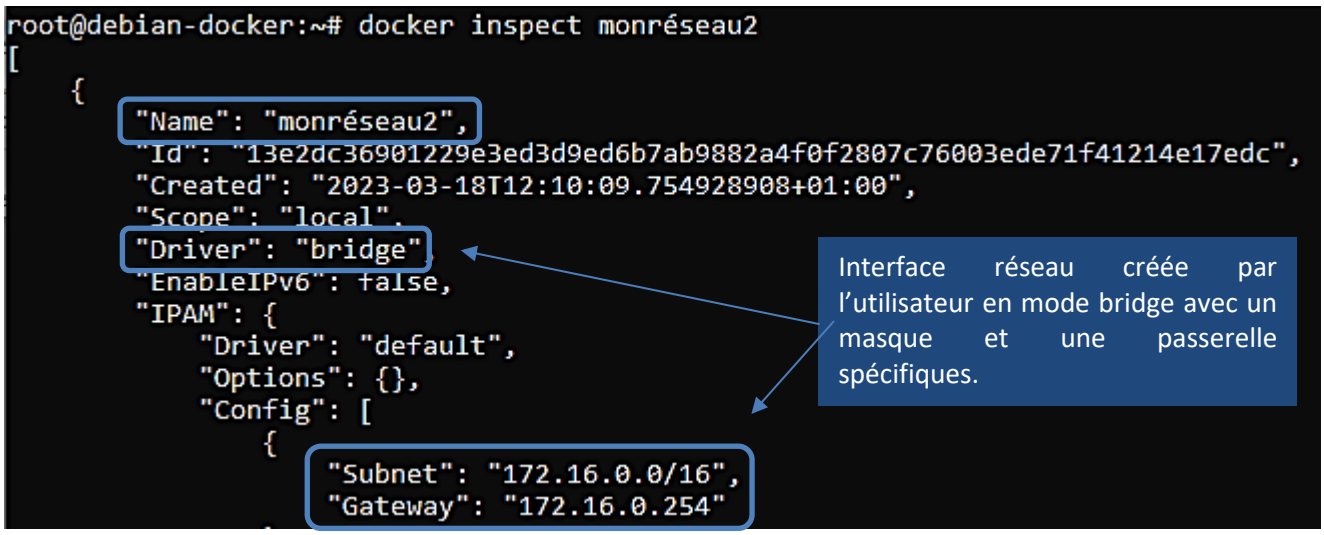
Liste des réseaux disponibles (bridge, host et none par défaut) plus les autres réseaux créés par l'utilisateur.

4. **Créer un réseau de type « bridge » nommé « monréseau2 » avec un masque et une passerelle spécifiques :**

docker network create -d bridge --subnet=172.16.0.0/16 --gateway=172.16.0.254 monréseau2

```
root@debian-docker:~# docker network create -d bridge --subnet=172.16.0.0/16 --gateway=172.16.0.254 monréseau2
13e2dc36901229e3ed3d9ed6b7ab9882a4f0f2807c76003ede71f41214e17edc
```

Si on inspecte le réseau avec « ***docker inspect network monréseau2*** », on constate que l'adressage IP demandé a bien été appliqué :



5. **Affecter un réseau à un conteneur :**

Dans cet exemple, nous avons créé 2 conteneurs « Alpine » que nous relions à chacun de nos réseaux préalablement créés (« monréseau » et « monréseau2 ») :

docker run -tid --name alpine1 --network monréseau alpine

```
root@debian-docker:~# docker run -tid --name alpine1 --network monréseau alpine
64464298039f14054979e713d39a63ab2cb8ab08eea3ac40890f00ab4a833d2d
```

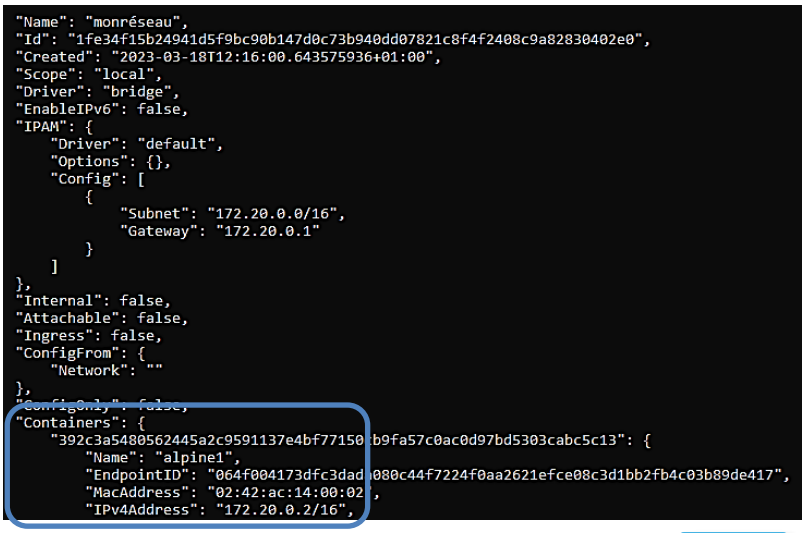
docker run -tid --name alpine2 --network monréseau2 alpine

```
root@debian-docker:~# docker run -tid --name alpine2 --network monréseau2 alpine
e7780be877dd0d32849c5aa083af02d9e42eefe33a5e043d9286c553c6b14f77
```

6. **Vérification de l'affectation des conteneurs aux réseaux spécifiés :**

Si on lance la commande « ***docker inspect monréseau*** » on constate que seul le conteneur « alpine1 » est bien relié à ce réseau :

Le container « alpine1 » est bien relié à l'interface réseau que nous avons créée.

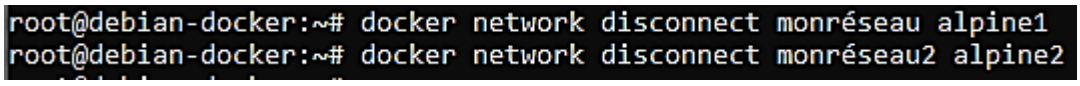


La commande « ***docker inspect monréseau2*** » permet de constater que le conteneur « alpine2 » est lui relié à ce réseau **avec le masque et la passerelle définis préalablement** pour ce réseau :

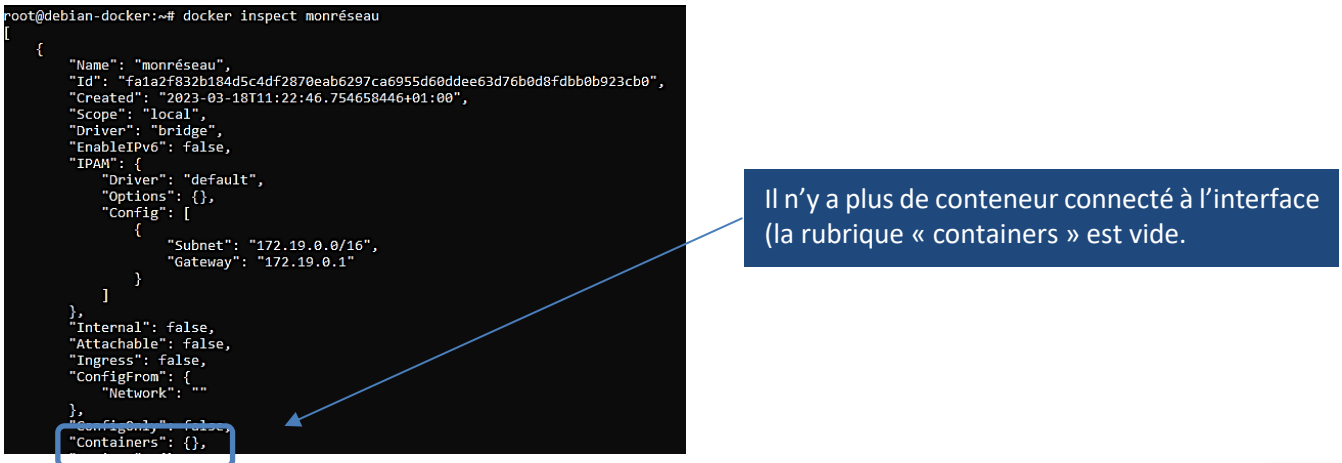
7. **Déconnecter un conteneur de son réseau :**

Dans cet exemple, nous déconnectons nos conteneurs de leurs réseaux respectifs :

docker network disconnect monréseau alpine1
docker network disconnect monréseau2 alpine2



La déconnexion des conteneurs a bien été réalisée puisque la rubrique « Containers » n'affiche plus rien :
docker inspect monréseau



8. **Supprimer un réseau Docker :**

docker network rm monréseau

```
root@debian-docker:~# docker network rm monréseau
monréseau
```

Si on liste les réseaux présents, on constate que le réseau « monréseau » a été supprimé :
docker network ls

```
root@debian-docker:~# docker network ls
NETWORK ID     NAME            DRIVER    SCOPE
36c710b4d6da   bridge          bridge    local
f65789398574   host            host      local
2384e0f80594   monréseau2      bridge    local
514b752cc29d   nextcloud-aio   bridge    local
262c18388404   none            null      local
```

9. **Reconnecter un conteneur au réseau « bridge » par défaut :**

Si vous avez déconnecté un conteneur d'un réseau spécifique et que vous souhaitez le reconnecter au réseau « bridge » par défaut de Docker, il faudra exécuter la commande suivante :

docker network connect bridge alpine1

```
root@debian-docker:~# docker network connect bridge alpine1
```

Le conteneur « alpine1 » a bien été reconnecté sur le réseau « bridge » de Docker :

```
"Containers": {
    282ea143388978f3b839a70f7bdd5e7f3ed50a7969115175ab8da1b13c9fc54a": {
        "Name": "portainer",
        "EndpointID": "1bdef1c9de6ecc007b18e877399e91b195c863f95a9d8949bd5df4467bc89e29",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
    },
    "392c3a5480562445a2c9591137e4bf77150cb9fa57c0ac0d97bd5303cabc5c13": {
        "Name": "alpine1",
        "EndpointID": "a85e0f334cc4b197300f05f8d2c5ef628aec5ed92d17bfef2ead9aee24bd5c8e",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
```

TP A EXECUTER (télécharger au préalable l'image « Alpine » avec *docker pull alpine*

N°	Tâche à réaliser	Commande à exécuter
1	Créez un réseau de type « bridge » nommé « landocker1 »	***docker network create --driver bridge landocker1***
2	Vérifiez l'existence de votre réseau dans Docker	***docker network ls***
3	Créez 2 conteneurs Alpine que vous nommerez « alpine1 » et « alpine2 » et connectez-les au réseau « landocker1 »	***docker run -tid --name alpine1 --network landocker1 alpine*** ***docker run -tid --name alpine2 --network landocker1 alpine***
4	Inspectez votre réseau « landocker1 » et vérifiez les conteneurs connectés	***docker network inspect landocker1***

N°	Tâche à réaliser	Commande à exécuter
5	Exécutez le conteneur « alpine1 » et faites afficher l'adresse IP	***docker exec alpine1 ip a***
6	Lancez un test de ping sur le conteneur « alpine1 »	***docker exec ping -c 4 172.xx.xx.xx***
7	Déconnectez le conteneur « alpine2 » du réseau « landocker1 »	***docker network disconnect landocker1 alpine2***
8	Vérifiez, en l'inspectant, que le réseau « landocker1 » n'a que le conteneur « alpine1 » connecté	***docker network inspect landocker1***
9	Connectez le conteneur « alpine2 » à l'interface « bridge » par défaut de Docker	***docker network connect bridge alpine2***
10	Exécutez le conteneur « alpine2 » pour vérifier son adressage IP qui doit être sur le réseau 172.17.xx.xx de Docker Bridge	***docker exec alpine2 ip a***
11	Faites un test de ping du conteneur « alpine2 » vers le DNS 8.8.8.8	***docker exec alpine2 ping 8.8.8.8***
12	Connectez-vous au shell du conteneur « alpine1 » et tentez de lancer un test de ping vers le conteneur « alpine2 »	***docker exec -ti alpine1 sh***
13	Faites en sorte que les conteneurs « alpine1 » et « alpine2 » soient sur le même réseau « landocker1 » et faites un test de ping pour constater que les machines répondent aux tests de ping	***docker network disconnect bridge alpine2*** ***docker network connect landocker1 alpine2*** ***docker exec -ti alpine2 sh*** ***ping 172.xx.xx.xx***
14	Stopper les conteneurs « alpine1 » et « alpine2 »	***docker stop alpine1 alpine2***
15	Supprimez les conteneurs « alpine1 » et « alpine2 »	***docker rm alpine1 alpine2***
16	Vérifiez que les conteneurs soient bien supprimés et ne soient plus actifs	***docker ps -a***
17	Supprimez le réseau « landocker1 »	***docker network rm landocker1***
18	Vérifiez que le réseau « landocker1 » a bien été supprimé	***docker network ls***
Création d'un mappage de port (exposition de port pour un serveur web par exemple)		
1	Téléchargez l'image « httpd » (image allégée du serveur web Apache)	***docker pull httpd***
2	Créez le conteneur « web » depuis l'image « httpd ». Ce conteneur sera exposé au port 8181 de l'hôte qui redirigera vers le port 80 du conteneur	***docker run -tid -p 8181:80 --name web httpd***
3	Listez les conteneurs actifs et vérifiez que le port 8181 du conteneur « web » est bien exposé	***docker ps -a***

```
CONTAINER ID   IMAGE     COMMAND              CREATED              STATUS
  PORTS
            NAMES
556b80e333fd   httpd     "httpd-foreground"   About a minute ago   Up About a minute
  0.0.0.0:8181->80/tcp, :::8181->80/tcp
            web
```

Ajoutez une règle dans votre pare-feu pour ouvrir le port 8181. Lancez votre navigateur et saisissez votre adresse WAN:8181 ; logiquement vous devriez voir s'afficher le message par défaut du serveur Apache !