TUTOS-INFO
L'informatique par l'exemple

TUTORIEL

DOCKER
Installer Docker et Portainer

SOMMAIRE

EXPERT
DIFFICULTE

1 – INSTALLATION DE DOCKER SUR DEBIAN 11.6 (Bullseye)

La réalisation de ce tutoriel nécessite d'avoir une machine Debian 11.6 (Bullseye) fonctionnelle à disposition. Il est possible d'installer Docker depuis les dépôts Debian mais vous n'aurez pas forcément la dernière version du moteur Docker. Il est donc préférable d'installer le moteur Docker en suivant la procédure « officielle ». Les commandes ci-dessous peuvent être copiées et collées si vous êtes connecté(e) en SSH. **Attention, saisissez « sudo » avant la commande si vous êtes connecté(e) en tant qu'utilisateur** (ici nous nous sommes logués en tant que root).

1. **Mise à jour des dépôts Debian et installation des paquets « ca-certificates », « curl », « gnupg » et « lsb-release » :**

```
apt-get update
apt-get install \
   ca-certificates \
   curl \
   gnupg \
   lsb-release
```

```
root@debian:~# apt-get update
apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

2. **Ajoutez la clé GPG officielle de Docker :**

```
mkdir -m 0755 -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

```
root@debian:~# mkdir -m 0755 -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

3. **Modifiez le « repository » de votre version Debian :**

```
echo \
  "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
root@debian:~# echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. **Mise à jour des dépôts :**

```
apt-get update
```

5. **Installation du moteur Docker, de Containerd et de Docker Compose :**

```
apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Vérifiez la version installée avec la commande « docker --version » :

```
root@debian:~# docker --version
Docker version 23.0.1, build a5ee5b1
```

A ce jour (mars 2023), la dernière version stable de Docker est la version 23.0.1

2 – INSTALLATION DE PORTAINER-CE

Portainer-CE permet de gérer vos conteneurs avec une interface graphique simple et intuitive. Après avoir installé Docker, nous pouvons lancer la création de notre premier conteneur « Portainer-CE » de la façon suivante :

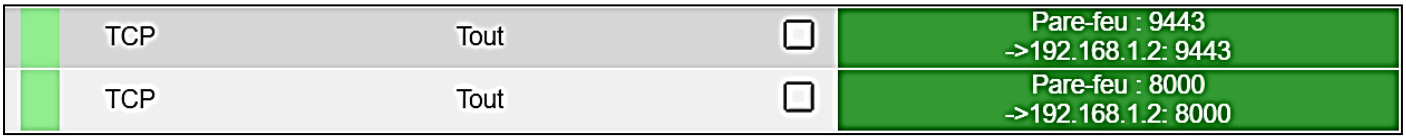
1. **Création d'un volume « portainer_data » :**

docker volume create portainer_data

```
root@debian:~# docker volume create portainer_data
portainer_data
```

2. **Création du conteneur « portainer-ce » :**

Attention, vous devez ouvrir sur votre pare-feu (box, routeur) les ports « 8000 » et « 9443 » et cibler votre machine Debian qui contient le moteur Docker. Ici nous avons utilisé le pare-feu IPFire et ouvert les ports nécessaires :



docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:latest

```
root@debian:~# docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.soc
k -v portainer_data:/data portainer/portainer-ce:latest
Unable to find image 'portainer/portainer-ce:latest' locally
latest: Pulling from portainer/portainer-ce
772227786281: Pull complete
96fd13befc87: Pull complete
b733663f020c: Pull complete
9fbfa87be55d: Pull complete
Digest: sha256:9fa1ec78b4e29d83593cf9720674b72829c9cdc0db7083a962bc30e64e27f64e
Status: Downloaded newer image for portainer/portainer-ce:latest
4348e51d05385b8c0a1b08a9e4eaead60dc2aa961a46fa1a889d53950beade03
```

3. **Accéder à Portainer :**

Pour accéder à Portainer, ouvrez votre navigateur et saisissez dans la barre d'adresse soit votre IP Wan, soit votre domaine et précisez le port 9443 ; par exemple : https://votredomaine:9443

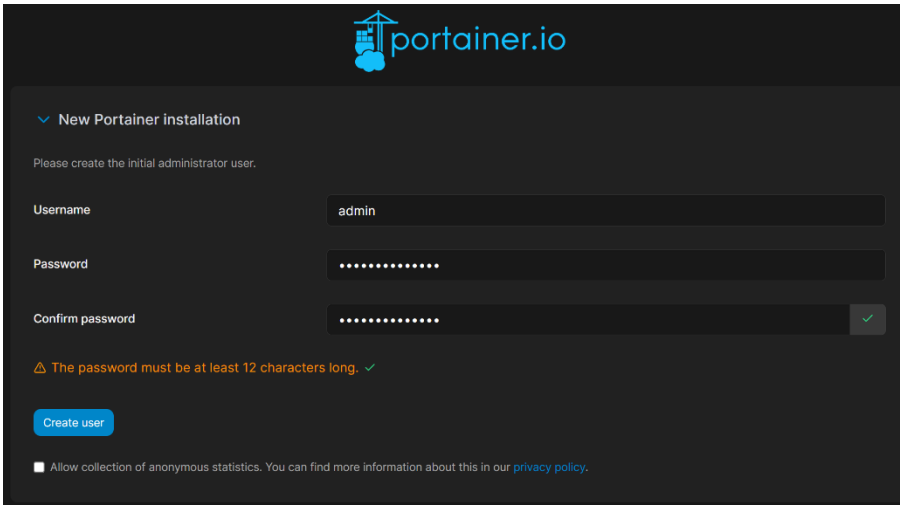
La fenêtre suivante s'affiche :

Il est possible que cette fenêtre ne s'ouvre pas lors de la première connexion et qu'un message vous demande de relancer votre conteneur.

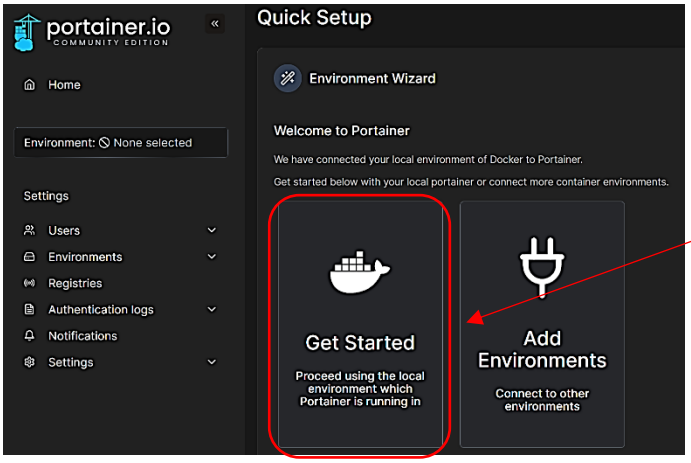
Dans ce cas, saisissez les commandes suivantes sur votre serveur :

docker stop portainer
docker start portainer

Actualisez la page et vous devriez obtenir la fenêtre ci-contre vous demandant de définir un username et un mot de passe fort.

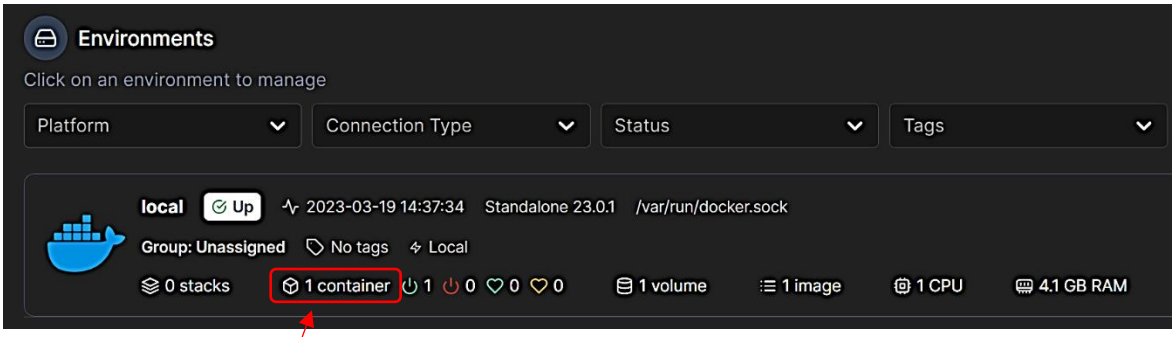


Dans la fenêtre suivante, cliquez sur « Get started » :

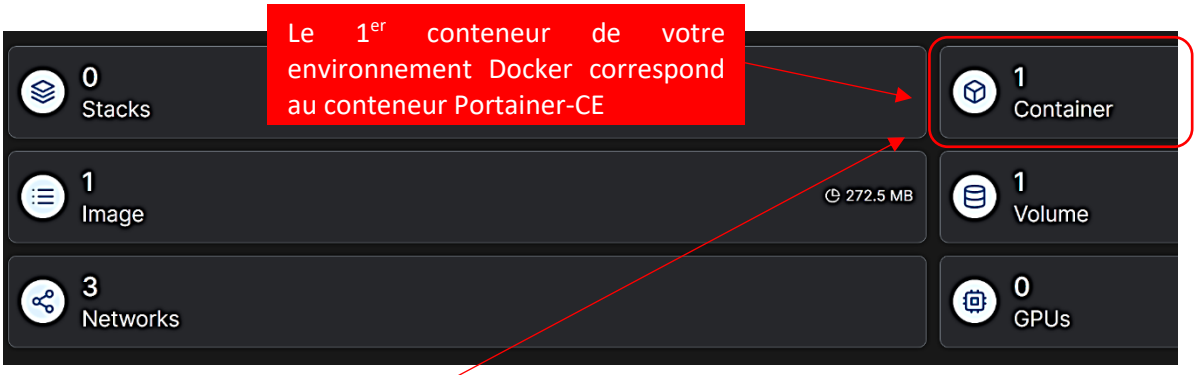


Lors de la première connexion à l'interface de Portainer-CE, cliquez sur « Get started » pour connecter Portainer à votre environnement local Docker.

Votre environnement local s'affiche :

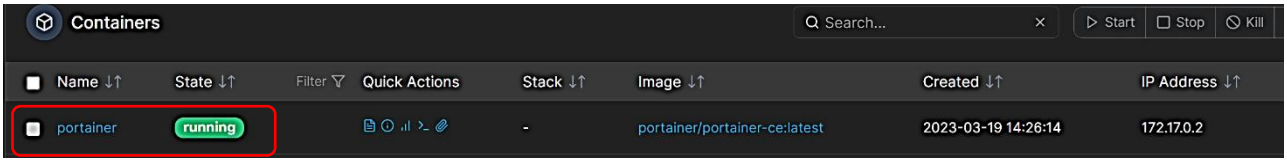


Si vous cliquez sur « 1 container » vous obtenez un détail de votre environnement Docker :



Le 1er conteneur de votre environnement Docker correspond au conteneur Portainer-CE

En cliquant sur le bouton « 1 Container » vous obtenez le détail du conteneur actif :



Votre conteneur « Portainer-CE » est actif en mode « running ». Vous pouvez dorénavant gérer votre environnement Docker via Portainer-CE !

Attention, cette interface intuitive ne dispense pas d'utiliser Docker en mode « cli » (lignes de commandes). Il reste parfois nécessaire de maîtriser les commandes pour certaines opérations plus complexes.

2 – TELECHARGER UNE IMAGE SUR LE DOCKER HUB

Docker met à la disposition des développeurs un service en ligne, baptisé le **Docker Hub**, conçu pour faciliter l'échange d'applications containérisées. Le Docker Hub héberge plus de 100 000 images de containers (Janv. 2020).

Ici, nous allons télécharger l'image ALPINE qui nous permettra de créer notre premier conteneur :

```
docker@docker:~$ docker pull alpine:latest
```

Le TAG « :latest » signifie que nous souhaitons télécharger la dernière version en date.

Une fois l'image téléchargée, il est possible de la lister en faisant « **docker images** » :

```
docker@docker:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              28f6e2705743        3 weeks ago         5.61MB
```

Nom de l'image | Version de l'image | Numéro ID de l'image | Date de mise à jour de l'image | Taille de l'image

L'ensemble des images présentes sur votre machine s'affiche avec les détails.

3 – CREER ET ACTIVER UN CONTENEUR A PARTIR D'UNE IMAGE

A partir de l'image alpine:latest téléchargée, nous créons un premier conteneur avec la commande « **docker run** » :

```
docker@docker:~$ docker run -tid --name alpine1 alpine
04dcc5f3d5bbe8c4b9a84e4fea5f8b80377bcb1ad542684c37197d558a06b9e3
```

Analyse de la commande « **docker run** » :

```
docker run -tid --name alpine1 alpine
```

Nom de la commande | -tid | Ici, on nomme le conteneur « alpine1 » et on indique la source qui est l'image « alpine ».

Les arguments « -tid » signifient :

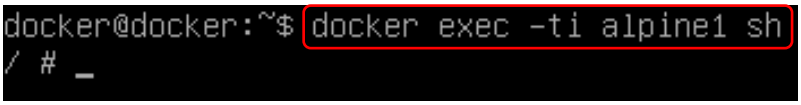
t : on émule un terminal (tty)
i : on active le conteneur
d : on détache le conteneur (fonctionnement en arrière-plan en quelque sorte)

Une fois le conteneur créé, on peut saisir la commande « **docker ps -a** » pour vérifier qu'il est bien en statut « Up » :

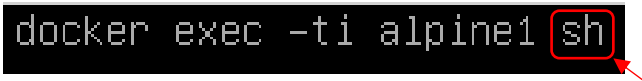
4 – EXECUTER LE CONTENEUR

Ici, nous allons entrer dans le « shell » de notre conteneur Alpine. Pour cela, nous utiliserons la commande indiquée lorsque nous avons fait « docker ps -a » :

Pour entrer en console dans notre conteneur nous utilisons la commande « **docker exec** » :



Analyse de la commande « **docker exec -ti alpine1 sh** » :

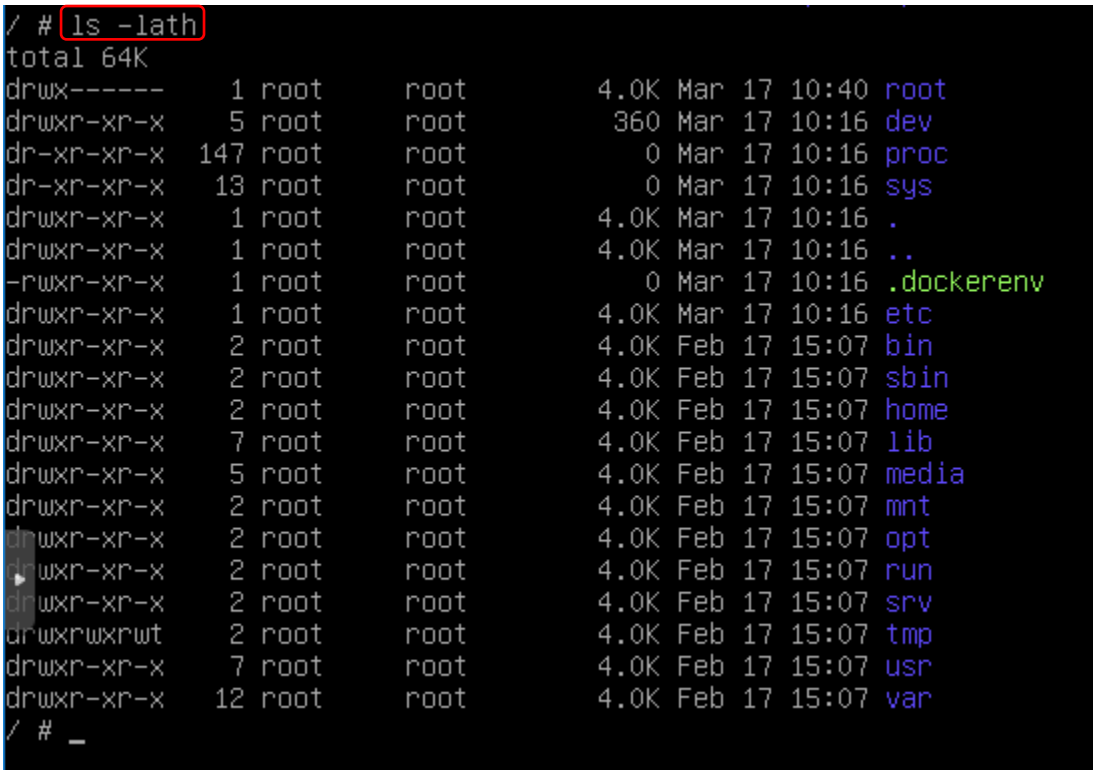


Le « sh » ici, signifie que l'on souhaite entrer dans le shell du conteneur alpine1

Les arguments « -tid » signifient :

t : on émule un terminal (tty)
i : on active le conteneur

Un simple « ls -lath » nous affiche l'arborescence de notre conteneur Alpine :



La sortie du shell s'effectue en saisissant la commande « exit » :

```
/ # exit
docker@docker:~$
```

Nous sommes sortis du shell du conteneur et sommes à nouveau sur l'hôte

5 – LA GESTION DES VOLUMES SUR DOCKER

Conçu à l'origine pour faciliter le déploiement d'applications sans état, Docker est de plus en plus utilisé pour des applications ayant besoin de stocker des données de façon persistante.

Lorsqu'une image Docker est exécutée, le Docker Engine crée un système de fichiers temporaire sur lequel sont stockés l'ensemble des composants et des données générées par le conteneur. Il s'appuie pour cela sur les capacités copy-on-write de l'Union File System. Dans la pratique, la mise en œuvre de ce mécanisme de copy-on-write signifie que lorsque la même image est instanciée à de multiples reprises sur un même hôte, le Docker Engine ne crée par une copie complète de l'image mais ne stocke que les modifications apportées par chaque image en cours d'exécution. Ce mécanisme permet non seulement d'économiser de l'espace, mais aussi de gagner du temps, notamment au démarrage du conteneur - ce qui permet, dans certains cas, d'instancier un conteneur en quelques dixièmes de seconde.

Les conteneurs ont à l'origine été pensés comme un moyen de déployer à grande échelle des micros services, c'est-à-dire **des applications sans état ne nécessitant pas de persistance de leur stockage**.

Dans la pratique, cela signifie que par défaut, **à l'arrêt d'un conteneur, l'espace qu'il occupait et les données générées à l'intérieur du conteneur sont effacés**. Cela convient bien à un micro service mais ne répond pas du tout aux besoins d'applications nécessitant de persister leurs données comme des bases de données ou des applications plus complexes.

CREER UN DOCKER VOLUME (recommandé)

Un volume Docker fournit un mécanisme pour **assurer la persistance des données** d'un conteneur ou lui permettre « **d'échanger** » des données avec d'autres conteneurs partageant le même volume.

Les volumes de données ont l'avantage particulier de survivre à l'arrêt du conteneur et Docker n'efface pas les volumes, même lors de la destruction d'un conteneur associé.

- **docker volume create nom_volume**

Il est à noter que, par défaut, un volume est monté en mode lecture-écriture, mais qu'il est possible de limiter l'accès au seul mode lecture.

Ici, nous créons un volume Docker nommé « volume_alpine1 », à l'aide de la commande « **docker volume create** » :

```
docker@docker:~$ docker volume create volume_alpine1
volume_alpine1
```

En créant un volume Docker, il est possible de connaître son emplacement exact avec la commande « **docker volume inspect** » :

```
docker@docker:~$ docker volume inspect volume_alpine1
[
    {
        "CreatedAt": "2021-03-17T11:47:15+01:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/volume_alpine1/_data",
        "Name": "volume_alpine1",
        "Options": {},
        "Scope": "local"
```

Ici, nous constatons que le volume a été « monté » depuis un emplacement spécifique : **/var/lib/docker/volumes**

Il est important de voir que Docker stocke les volumes dans cet emplacement et que ces volumes devront faire l'objet d'une sauvegarde sur des supports indépendants pour des raisons de sécurité.

ATTACHER UN DOCKER VOLUME A UN CONTENEUR

Nous allons attacher le volume créé (volume_alpine1) à un nouveau conteneur, qui sera nommé Alpine2, de la façon suivante :

```
docker@docker:~$ docker run -tid -v volume_alpine1:/volume_alpine --name alpine2 alpine
1a0b226d882e17a009f9f7db69cd1513e77508a3948b21be815783afccf0bcd0
```

Analyse de la commande :

```
docker run -tid -v volume_alpine1:/volume_alpine --name alpine2 alpine
```

L'argument « -v » signifie que l'on souhaite attacher le volume nommé « volume_alpine1 » et que l'on souhaite le faire apparaître dans le shell du conteneur sous la forme « volume_alpine ».

Il suffit ensuite de se connecter au shell du conteneur pour accéder au volume :

```
docker@docker:~$ docker exec -ti alpine2 sh
/ # ls
bin             home            mnt             root            srv             usr
dev             lib             opt             run             sys             var
etc             media           proc            sbin            tmp             volume_alpine
/ #
```

ATTACHER UN REPERTOIRE LOCAL DE L'HOTE (montage de type « bind »)

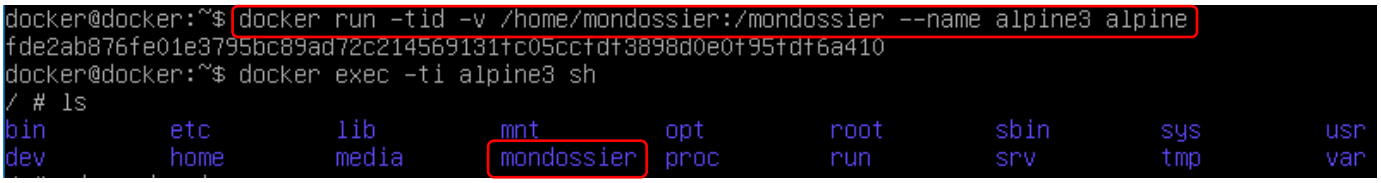
Il est possible d'attacher un répertoire présent sur l'hôte sans passer par la création d'un volume en lançant le conteneur et en lui attachant le répertoire souhaité directement :

Dans la commande suivante, nous attachons le dossier « mon_dossier », présent dans notre dossier « home », dans le conteneur en donnant « nom_montage » au dossier attaché depuis l'hôte :

- **docker run -tid -v /home/mon_dossier:/nom_montage conteneur_nom nom_image**

Pour information, il est possible de monter un **volume de stockage partagé** (iSCSI, FC ou NFS) comme data volume. Docker inclut un concept de plug-ins qui permet aux principaux fabricants de baies de stockage d'intégrer leurs systèmes de stockage avec la technologie de conteneurs.

Ici, nous montons un répertoire local (créé dans notre « home » préalablement et nommé « mondossier » :



L'argument « -v » signifie que l'on souhaite monter le répertoire local situé dans notre « home » et nommé « mondossier » et que l'on souhaite le faire apparaître dans le shell du conteneur sous le nom « mondossier ».

RECAPITULATIF DES COMMANDES ESSENTIELLES

IMAGES ET CONTENEURS DOCKER	
Télécharger une image Docker	docker pull nom_image:latest
Lister les conteneurs	docker ps
Lister les conteneur actifs et inactifs	docker ps -a
Lister les commandes docker	docker
Lancer un conteneur	docker run -tid --name nom_conteneur nom_image
Se connecter à un conteneur (par exemple lancer un shell Alpine)	docker exec -ti nom_conteneur sh
Sortir du shell d'un conteneur	exit
Supprimer des conteneurs	docker rm -f nom_conteneur (ouID) conteneur2 (ou ID2)
Supprimer une image docker	docker rmi nom_image
COMMANDES RESEAU	
Lister les cartes réseau	docker network ls
Récolter des informations sur une carte réseau	docker network inspect nom (ou ID)
Créer un réseau ponté	docker network create --driver bridge nom_pont
Connecter un conteneur à un pont	docker connect nom_pont nom_conteneur
Déconnecter un conteneur d'un pont	docker disconnect nom_pont nom_conteneur
Supprimer un pont	docker network rm nom_pont
Supprimer toutes les cartes réseau	docker network prune -f
Créer un pont réseau	docker network create -d bridge --subnet 172.x.0.0/x
GESTION DU STOCKAGE SUR DOCKER	
LES VOLUMES DE DONNEES	
Créer un volume	docker volume create volume_docker
Lister les volumes	docker volume ls
Inspecter un volume	docker volume inspect volume_nom
Supprimer un volume	docker volume rm volume_nom
Attacher un volume à un conteneur	docker run -tid -v volume_nom:/nom --name nom_conteneur image
ATTACHER UN REPERTOIRE (type Bind)	
Attacher un répertoire local à un conteneur (montage de type « Bind »)	docker run -tid -v /emplacement exact sur l'hôte:/nom nom_conteneur image

LISTER LE SYSTEME DE FICHIERS DE DOCKER	
Lister le système de fichiers Docker	cd /var/lib/docker df -h
COMMANDES UTILES POUR CONTROLES	
Voir les ports en écoute	ss -ntlp
Voir les derniers logs sur un conteneur	docker logs nom_conteneur
Afficher et parcourir les derniers logs sur un conteneur	docker logs nom_conteneur | less
Afficher les derniers logs sur un conteneur en temps réel	docker logs -f nom_conteneur
Afficher les logs ayant le motif « erreur 404 »	docker logs nom_conteneur | grep 404
Retrouver les commandes passées	history | less
Afficher tout ce qui peut être inspecté	docker inspect « tab » « tab » (ici on appuie 2 fois sur la touche TAB
Vérifier les ressources consommées par les conteneurs en temps réel	docker stats

CONSEIL LIE A L'UTILISATION DE DOCKER EN PRODUCTION

Il est conseillé de travailler à partir d'un **utilisateur « dédié » à Docker** plutôt qu'en « root » pour des raisons de sécurité. On procèdera ainsi pour créer cet utilisateur :

1. Création de l'utilisateur (qui aura la charge de l'administration de Docker)
2. Ajout de l'utilisateur au groupe « docker »

Commandes à saisir :

adduser nom_user_docker
usermod -aG docker nom_user_docker

Attention, ne modifiez pas les droits et permissions sur les volumes Docker lorsqu'ils ont été créés avec la commande « docker volume create » !

TP DOCKER

TRAVAIL A REALISER – 1ère partie – FONCTIONS DE BASE ET CREATION D'UN CONTENEUR

Etape	Travail à réaliser	Commande à saisir
1	Installez Docker sur votre machine Debian	apt install docker.io
2	Vérifiez la version de Docker installée	docker --version (2 tirets collés)
3	Faites afficher les commandes Docker	docker
4	Faites afficher l'aide pour une commande spécifique (par exemple la commande « pull »)	docker --help pull
5	Téléchargez l'image de la distribution « Alpine » (version allégée Linux)	docker pull alpine (sans indication supplémentaire, la version « latest » sera téléchargée)
6	Vérifiez que l'image est bien présente dans vos images Docker	docker images
7	Créez et lancez votre premier conteneur Alpine que vous nommerez « alpine1 »	docker run -ti --name alpine1 alpine
8	Sortez du conteneur alpine1	exit
9	Vérifiez le statut de votre conteneur	docker ps -a
10	Tentez de relancer votre conteneur alpine1 : que constatez-vous ?	docker run -ti --name alpine1 alpine *Le conteneur a été détruit et ne se lance plus*

TRAVAIL A REALISER – 2ème partie – MANIPULATIONS SUR LES CONTENEURS

Etape	Travail à réaliser	Commande à saisir
1	Supprimez le conteneur alpine1	docker rm alpine1
2	Vérifiez qu'il n'y a plus de conteneurs présents	docker ps -a
3	Créez un nouveau conteneur alpine1 en faisant en sorte qu'il fonctionne en mode « détaché » (il n'est pas utile de télécharger l'image car elle est déjà présente)	docker run -tid --name alpine1 alpine
4	Vérifiez que le statut du conteneur est bien sur « Up »	docker ps -a
5	Entrez dans le « shell » de votre conteneur alpine1	docker exec -ti alpine sh
6	Une fois dans le conteneur, listez les dossiers du conteneur	ls
7	Ouvrez le dossier « home » et créez un dossier « test » à l'intérieur	cd home mkdir test
8	Tentez d'ouvrir l'éditeur nano pour créer un fichier « monfichier » dans /home/test	nano monfichier
9	Installez nano dans votre conteneur alpine	apk update apk add nano
10	Créez, avec nano, un fichier « monfichier » dans /home/test (saisissez une petite phrase dans le fichier créé)	nano monfichier
11	Faites afficher le contenu du fichier créé	cat monfichier
12	Quittez le conteneur alpine1	exit
13	Vérifiez le statut du conteneur alpine1	docker ps -a
14	Sur votre machine Debian, déplacez-vous dans votre dossier « home » pour constater qu'il n'y a pas de fichier « monfichier » puisque ce dernier est dans le conteneur alpine1	
15	Stoppez le conteneur alpine1	docker stop alpine1
16	Vérifiez le statut du conteneur	docker ps -a
17	Redémarrez votre conteneur alpine1	docker start alpine1
18	Vérifiez que le conteneur alpine1 est bien « Up »	docker ps -a

TRAVAIL A REALISER – 3ème partie – NOTION DE VOLUME SUR DOCKER

Etape	Travail à réaliser	Commande à saisir
1	Créez un volume Docker que vous nommerez « monvolume »	docker volume create monvolume
2	Listez les volumes présents dans Docker	docker volume ls
3	Vérifiez l'emplacement de création de ce volume sur votre machine hôte	docker volume inspect monvolume
4	Créez un nouveau conteneur « alpine2 » et attachez-lui le volume « monvolume ». Le volume apparaîtra dans le shell du conteneur sous le nom « volume_alpine »	docker run -tid -v monvolume:/volume_alpine --name alpine2 alpine
5	Vérifiez le statut de vos conteneurs	docker ps -a
6	Entrez dans le « shell » de votre conteneur alpine2	docker exec -ti alpine2 sh
7	Faites afficher le contenu de votre conteneur	ls

8	Logiquement votre volume est monté « volume_alpine ». Créez, dans ce volume, un fichier avec nano dans lequel vous saisirez une phrase et vérifiez que le fichier est présent	apk update apk add nano cd volume_alpine (et création du fichier)
9	Quittez le conteneur	exit
10	Rendez-vous, sur la machine hôte, dans le volume de votre conteneur pour vérifier que le fichier est bien présent	cd /var/lib/docker/volumes/monvolume/_data
11	Supprimez le conteneur alpine2	docker rm -f alpine2
12	Vérifiez le statut de vos conteneurs	docker ps -a
13	Créez un nouveau conteneur « alpine3 » et attachez-lui le volume « monvolume » qui apparaîtra dans le shell du conteneur sous le nom « volume_alpine »	docker run -tid -v monvolume:/volume_alpine --name alpine3 alpine
14	Entrez dans le shell du conteneur « alpine3 » pour vérifier que le volume « monvolume » a bien été remonté avec son contenu	docker exec -ti alpine3 sh

TRAVAIL A REALISER – 4ème partie – NOTION DE REPERTOIRE LOCAL ATTACHE (montage « bind »)

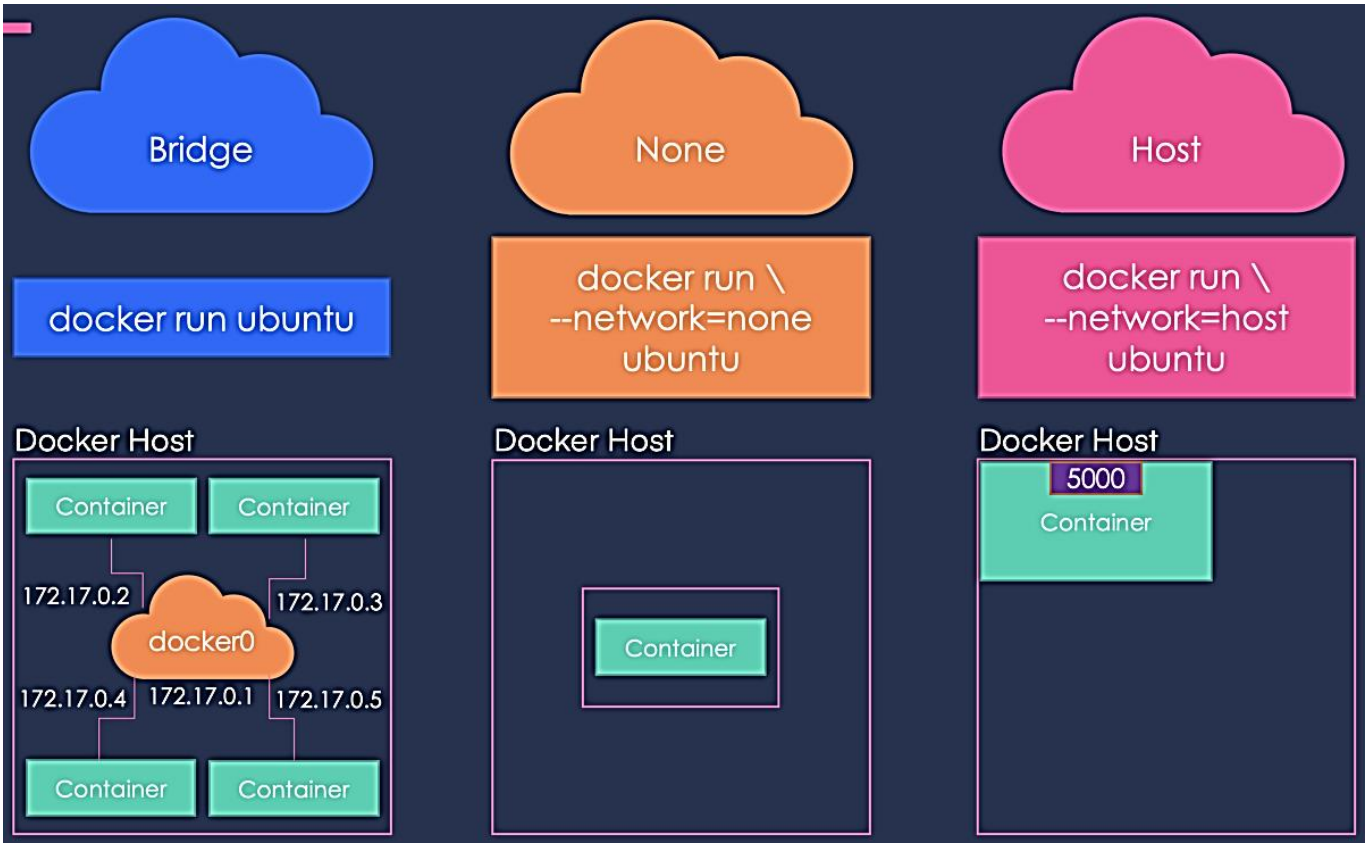
Etape	Travail à réaliser	Commande à saisir
1	Créez dans votre répertoire « home » un dossier nommé « dossier_alpine »	mkdir dossier_alpine
2	Créez un conteneur que vous nommerez « alpinebind » et attachez-lui le dossier « dossier_alpine » précédemment créé dans votre répertoire home	docker run -tid -v /home/dossier_alpine:/dossier_alpine --name alpinebind alpine
3	Entrez dans le shell du conteneur « alpinebind » et vérifiez que le lecteur a bien été attaché	docker exec -ti alpinebind sh
4	Créez, dans le lecteur attaché du conteneur, un dossier « test »	cd dossier_alpine mkdir test
5	Quittez le conteneur	exit
6	Vérifiez que le dossier attaché « dossier_alpine » présent dans votre « home » contient bien le dossier « test » créé depuis le conteneur	cd /home ls
7	Supprimez le conteneur « alpinebind »	rm -f alpinebind
8	Créez un conteneur que vous nommerez « alpinebind2 » et attachez-lui le dossier « dossier_alpine » précédemment créé dans votre répertoire home	docker run -tid -v /home/dossier_alpine:/dossier_alpine --name alpinebind2 alpine
9	Entrez dans le shell du conteneur « alpinebind2 » et vérifiez que le lecteur a bien été attaché avec son contenu	docker exec -ti alpinebind2 sh

6 – LA GESTION DES RESEAUX SUR DOCKER

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** »

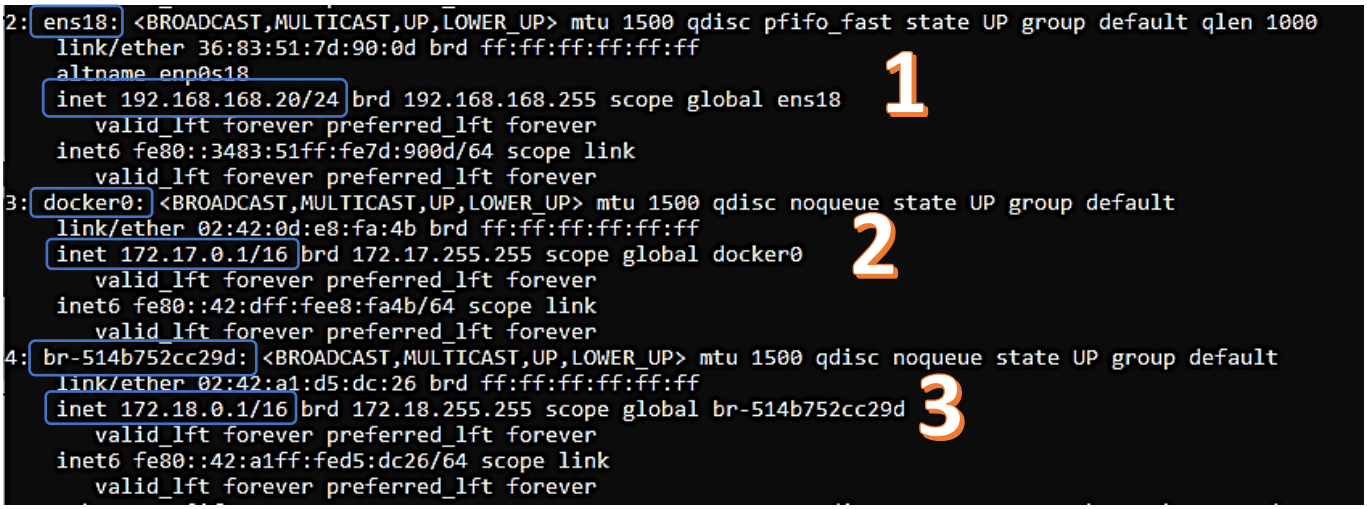


Lors de l'installation de Docker, 3 réseaux sont créés par défaut :

```
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 :



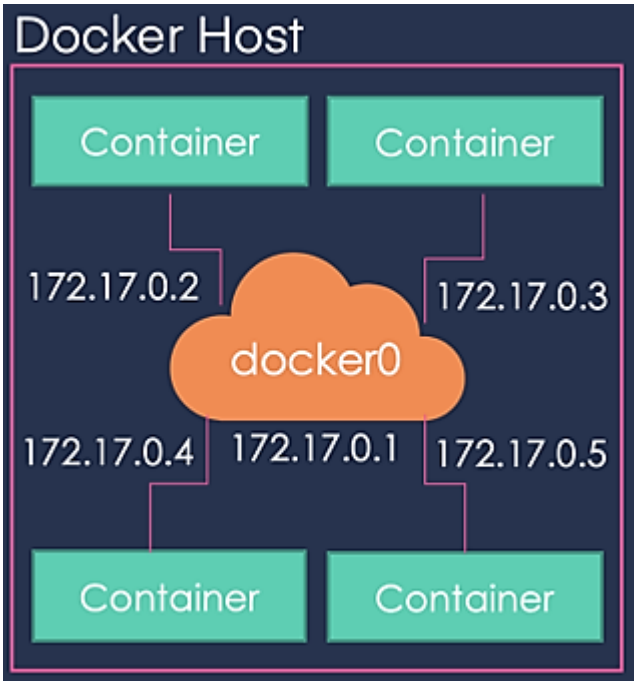
1 « ens18 » correspond à l'interface réseau de notre machine virtuelle Debian qui est connectée à l'hôte (le serveur Proxmox).

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

3 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.

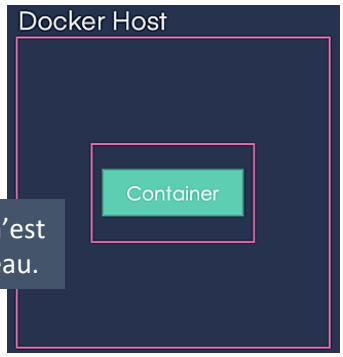
Les conteneurs qui utilisent le driver « Bridge » ne peuvent communiquer qu'entre eux. Pour être accessibles depuis l'extérieur, **un mappage de port est obligatoire**.

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.

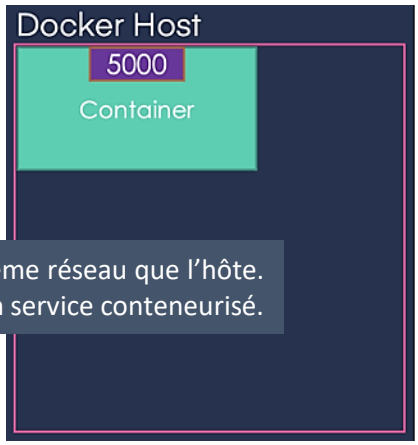
2. Le réseau de type « None »



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.

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

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** ».

En mode « host », le conteneur utilise le même réseau que l'hôte. Il faut « exposer » un port pour accéder à un service conteneurisé.

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 :

```
root@debian-docker:~# docker inspect monréseau2
[
    {
        "Name": "monréseau2",
        "Id": "13e2dc36901229e3ed3d9ed6b7ab9882a4f0f2807c76003ede71f41214e17edc",
        "Created": "2023-03-18T12:10:09.754928908+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.16.0.0/16",
                    "Gateway": "172.16.0.254"
```

Interface réseau créée par l'utilisateur en mode bridge avec un masque et une passerelle spécifiques.

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 :

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 :**

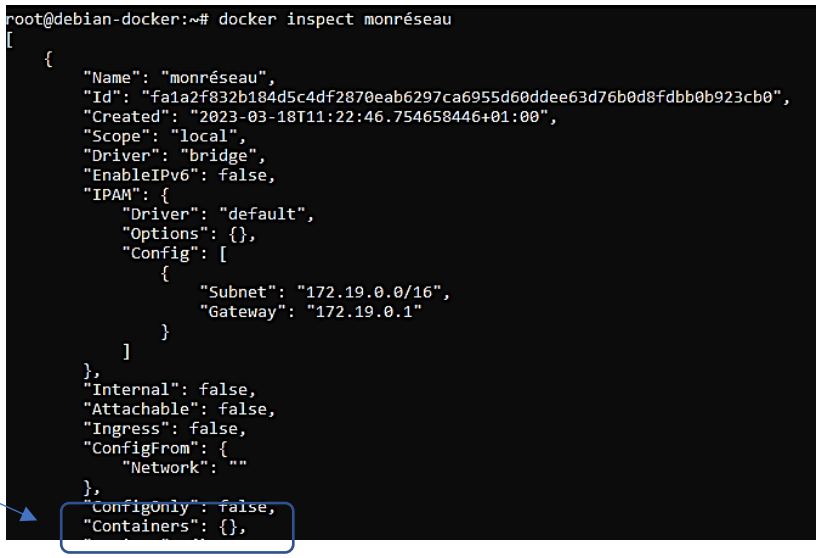
Dans cet exemple, nous déconnectons nos conteneurs de leurs réseaux respectifs :

docker network disconnect monréseau alpine1
docker network disconnect monréseau2 alpine2

```
root@debian-docker:~# docker network disconnect monréseau alpine1
root@debian-docker:~# 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 !