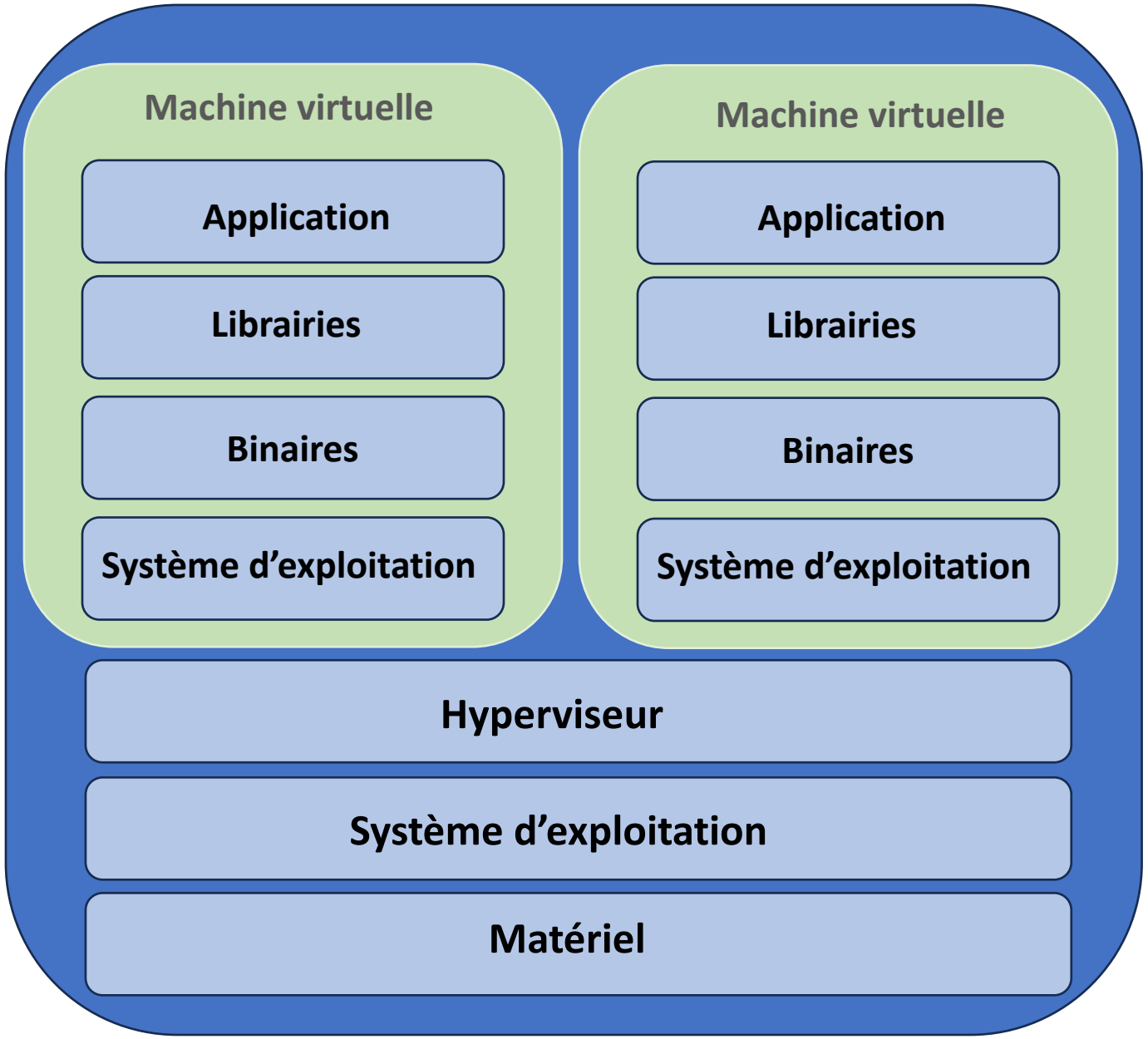
Virtualisation matérielle de type VMWare, VirtualBox ou HyperV.



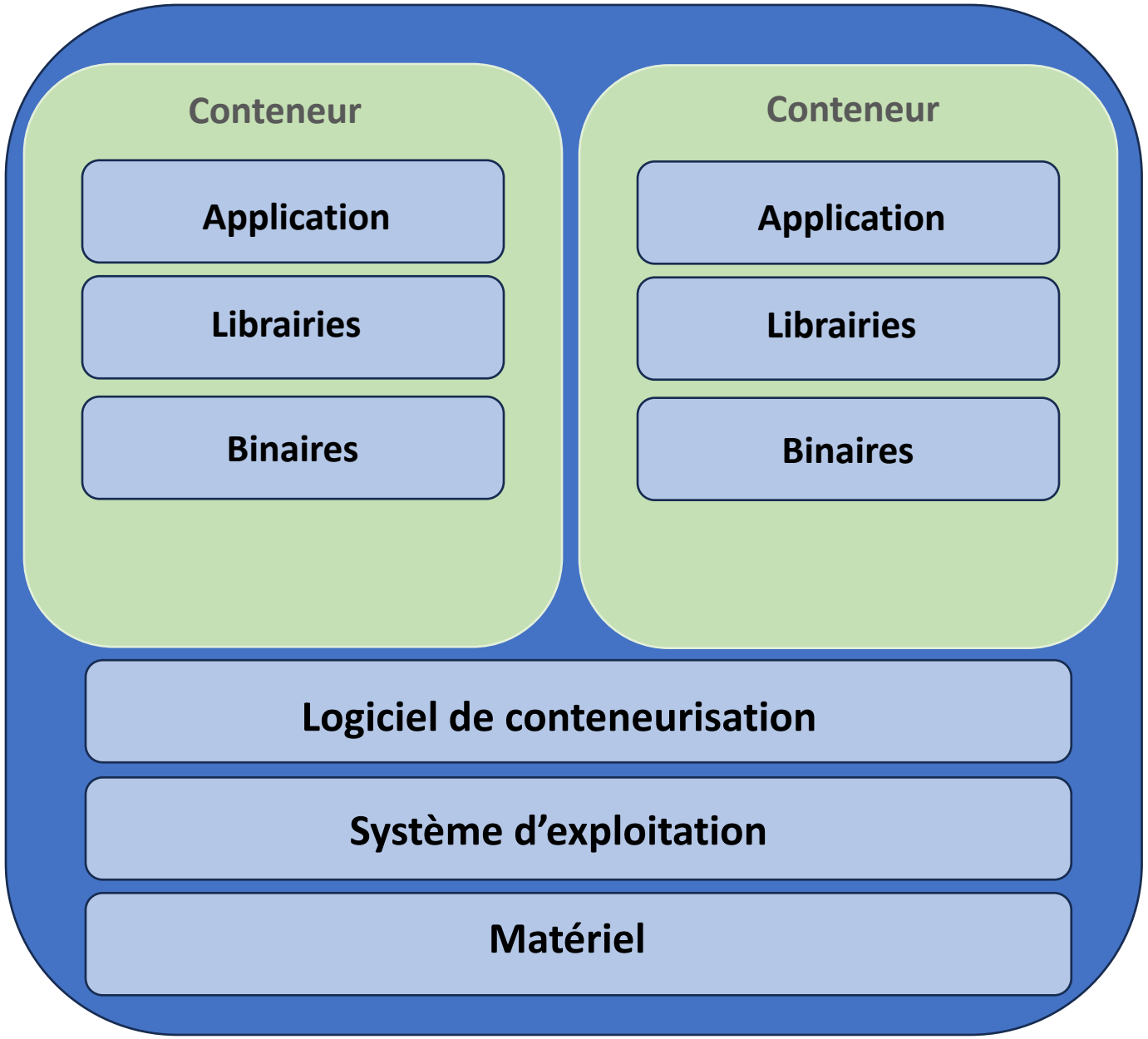
Dans les faits, comme nous le voyons, cette virtualisation matérielle offre un niveau d'isolation élevé.

Inconvénients :

le poids d'une machine virtuelle est très important.
Une machine virtuelle consommera difficilement moins de quelques Go de mémoire.

La distribution de ce type de package demandera une bande passante réseau conséquente

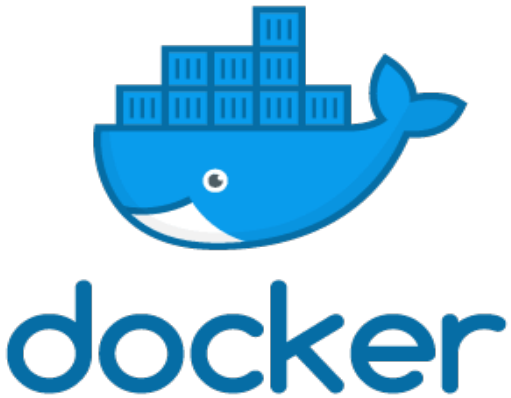
Les conteneurs : Doker, Podman, LXC, …



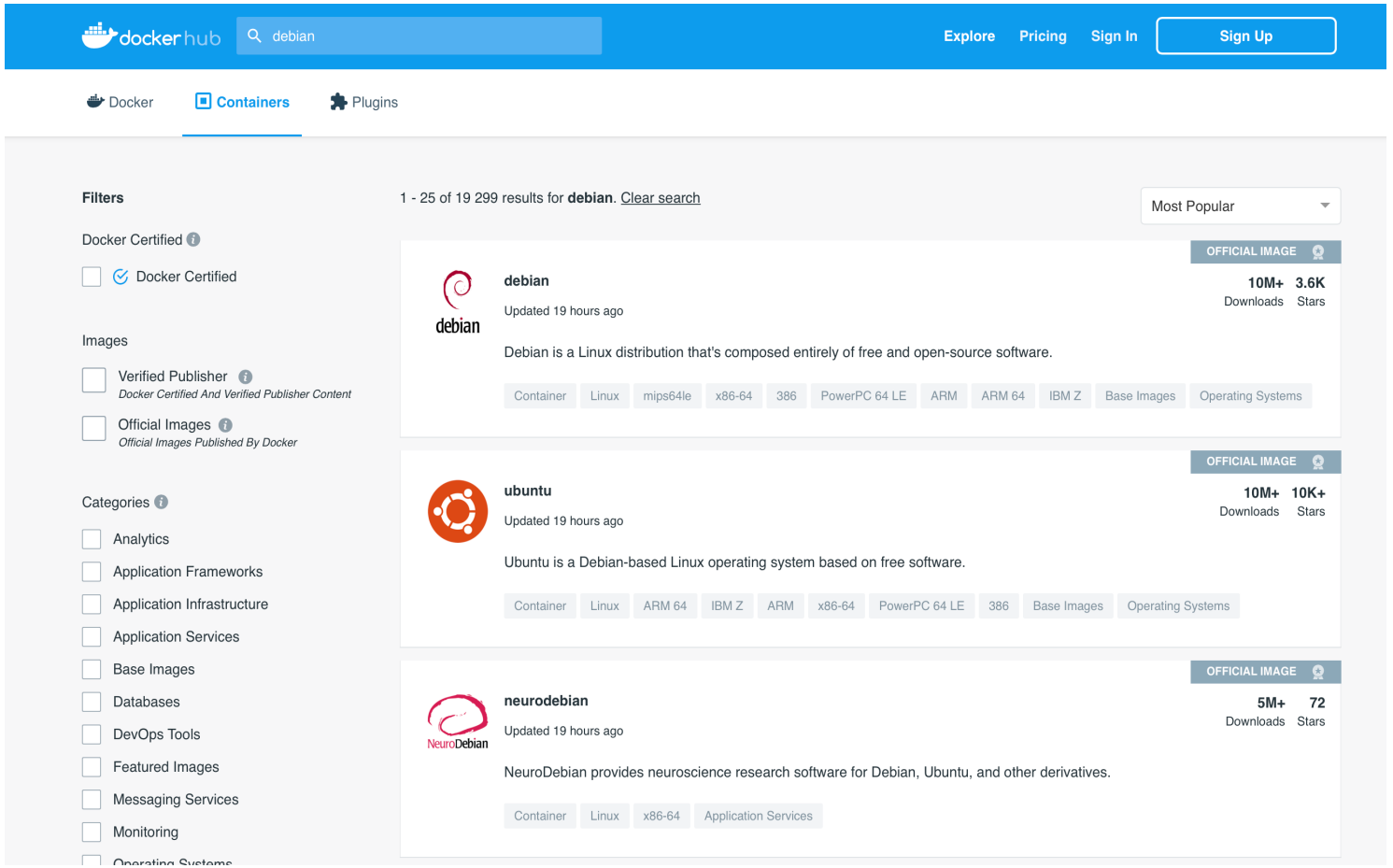
Une architecture à base de conteneurs offre une solution de compromis. Le conteneur offre l'isolation permettant à un développeur d'embarquer l'ensemble des dépendances logicielles dont il a besoin (y compris les dépendances de niveau OS).
De plus, un conteneur s'appuie sur le noyau (kernel) du système d'exploitation hôte1.

Il est donc très léger et démarre presque aussi vite que le processus qu'il encapsule. Le nombre de conteneurs qu'un même hôte peut exécuter est donc nettement plus élevé que son équivalent en machines virtuelles.

Docker



Docker permet de profiter de la technologie des conteneurs en profitant d'une API simple. Les commandes sont très simples à retenir et permettent de mettre en place un système de conteneur en place rapidement grâce à un hub collaboratif qui permet à tout le monde de partager des images qui pourront ensuite être utilisées comme base pour construire ces conteneurs.



Pour commencer à créer notre premier conteneur, on doit commencer par trouver une image qui va nous servir de base pour construire notre conteneur. Pour cela il faut se rendre sur le Hub Docker.

Création et de lancement d’un conteneur à partir d'une image donnée :

Pour lancer un conteneur, il suffit de taper : **`docker run <IMAGE>`**

Exemple pour une image debian : **`docker run –it debian`**

Options de base :

- **-d**, **--detach** : Lance le conteneur en arrière-plan (mode détaché).
- **--name** : Attribue un nom au conteneur, qui peut être utilisé pour le référencer ultérieurement.
- **-t**, **--tty** : Alloue un pseudo-TTY, ce qui simule un terminal, comme si vous étiez connecté en local au conteneur.
- **-i**, **--interactive** : Garde le STDIN ouvert, même si non attaché, ce qui permet d'interagir avec le conteneur.

Lister les conteneurs :

Si vous voulez voir tous les conteneurs, il faudra faire un :

```
docker ps -a #montre tous les conteneurs
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
58ed0202566e	eclipse-mosquitto	"/docker-entrypoint.…"	3 days ago	Up 24 hours	0.0.0.0:1883->1883/tcp, 0.0.0.0:8883->8883/tcp	mosquitto_broker
7020a402c25b	ubuntu:latest	"/bin/sh -c 'apt-get…"	3 days ago	Up 24 hours		mosquitto_client
d84cf5dbbb4b	b5529f640b6e	"bash"	6 days ago	Exited (0) 6 days ago		gracious_noether
f2e88ca30bdc	ubuntu	"bash"	5 weeks ago	Exited (255) 5 weeks ago		hardcore_lichterman
a128a539505f	ubuntu	"bash"	3 months ago	Exited (130) 3 months ago		lwt_simulator
b1b1fd0b06a8	alexandreoda/burpsuite	"/bin/sh -c 'java -j…"	14 months ago	Exited (10) 14 months ago		burpsuite
c7a5c4b59db8	ubuntu	"bash"	14 months ago	Exited (255) 5 weeks ago		user2
8a806d1d7232	openalpr	"alpr -c eu h786poj.…"	2 years ago	Exited (0) 13 months ago		jolly_allen

```
docker ps # montre tous les conteneurs en cours d'exécution
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
58ed0202566e	eclipse-mosquitto	"/docker-entrypoint.…"	3 days ago	Up 24 hours	0.0.0.0:1883->1883/tcp, 0.0.0.0:8883->8883/tcp	mosquitto_broker
7020a402c25b	ubuntu:latest	"/bin/sh -c 'apt-get…"	3 days ago	Up 24 hours		mosquitto_client

À chaque fois que vous faites un docker run un nouveau conteneur est lancé à partir de l'image que vous avez demandé.

Gérer les conteneurs:

Pour sortir d'un conteneur, saisissez exit : **root@b0f3efdd0b4f:/# exit**
Dans ce cas on sort et éteint le conteneur.

Pour sortir d'un conteneur sans l'éteindre, on utilise la combinaison de touche Ctrl-p puis Ctrl-q.

Pour relancer un conteneur fermé : **docker start <NOM DU CONTENEUR>**

Pour entrer dans un conteneur lancé : **docker attach <NOM DU CONTENEUR>**

Pour stopper un conteneur : **docker stop <NOM DU CONTENEUR>**

Pour supprimer un conteneur : **docker rm <NOM DU CONTENEUR>**

Pour exécuter une commande dans un conteneur : **docker exec <NOM DU CONTENEUR> commande**

Pour exécuter un shell interactif dans un conteneur déjà en cours d'exécution, vous pouvez utiliser :
docker exec –it <NOM DU CONTENEUR> /bin/bash

Gérer les images :

- Pour télécharger une image depuis Docker Hub ou un autre registre d'images :

`docker pull <NOM DU CONTENEUR>`

- Pour supprimer une image : **`docker rmi <NOM DE L'IMAGE>`**

- Pour lister les images locales : **`docker images`**

- Pour obtenir des informations détaillées sur une image spécifique :

`docker inspect <NOM DE L'IMAGE>`

- Pour supprimer toutes les images non utilisées : **`docker image prune`**

- Pour créer une image à partir d'un conteneur :

`docker [conteneur_id] [nouveau_nom_de_l'image]`

Exemple : **`docker commit c3f279d17e0a mynewimage:tag`**

LES VOLUMES

Les conteneurs Docker sont éphémères, ce qui signifie que toutes les données stockées à l'intérieur du conteneur sont perdues une fois que le conteneur est détruit. Pour conserver ces données, Docker permet de créer des volumes.

Un volume est un emplacement sur le disque hôte qui est monté dans le conteneur. Les données dans un volume survivent aux redémarrages et aux suppressions de conteneurs, et peuvent être partagées entre plusieurs conteneurs.

Pour créer un volume persistant commun à l'hôte et au conteneur lors de la création du conteneur.

```
docker run –v /path/in/host:/path/in/container ––name mycontainer
ubuntu:16.04
```

Gérer les volumes :

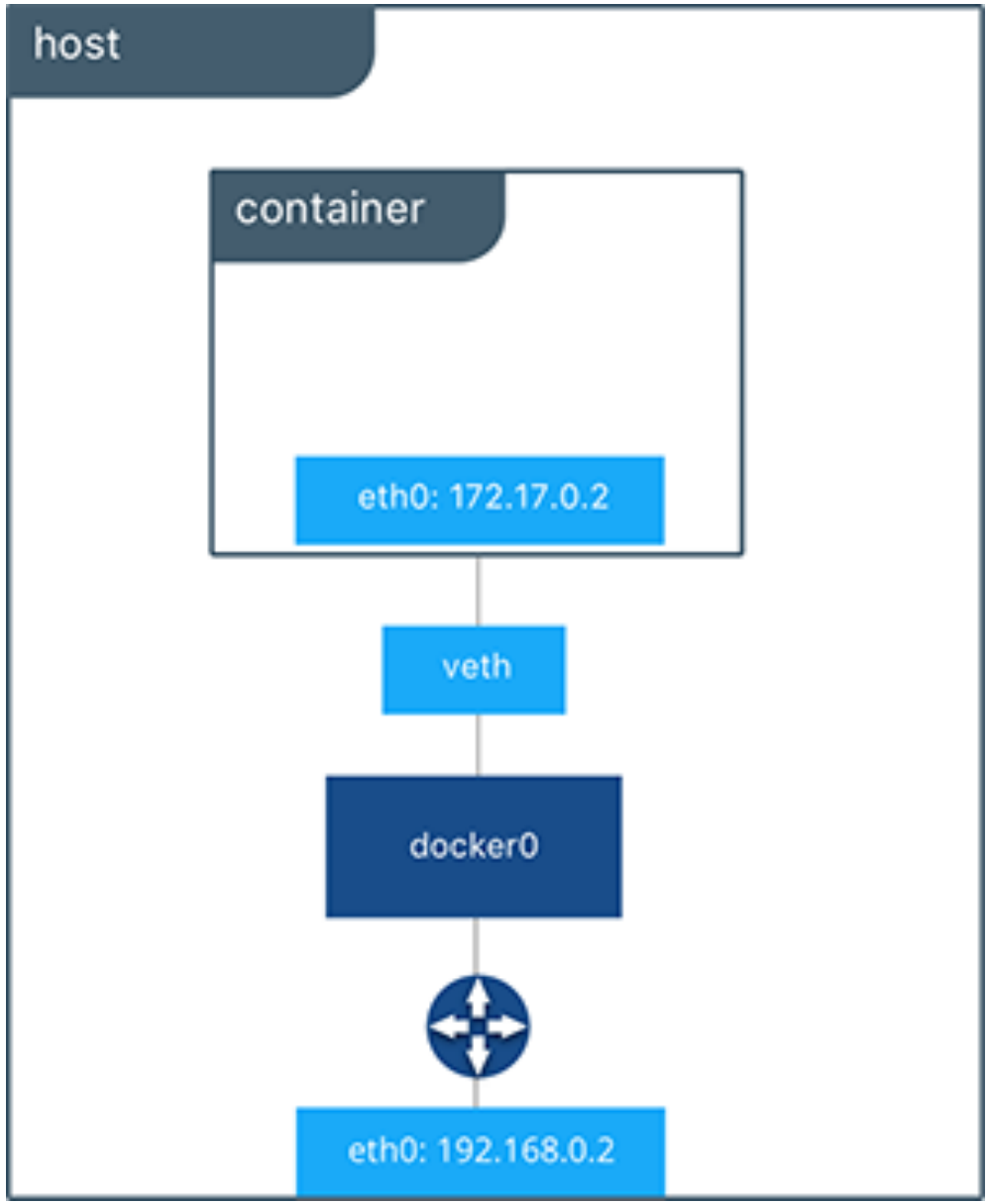
Lister les volumes : **docker volume ls**

```
DRIVER    VOLUME NAME
local     2fe31b53ecf96a928dbd0bf20d45f4a9516f800db05423e64ec8e51877003c43
local     b22a15920806b862e15ba4b2de7fb1c58ef1fa261c2cd4516084f29472cbc93f
```

Inspecter un volume : **docker volume inspect nom_du_volume**

docker volume inspect 2fe31b53ecf96a928dbd0bf20d45f4a9516f800db05423e64ec8e51877003c43


```
[
    {
        "CreatedAt": "2024-04-30T13:36:28Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/2fe31b53ecf96a928dbd0bf20d45f4a9516f800db05423e64ec8e51877003c43/_data",
        "Name": "2fe31b53ecf96a928dbd0bf20d45f4a9516f800db05423e64ec8e51877003c43",
        "Options": null,
        "Scope": "local"
    }
]
```



Le driver bridge :

Lorsque vous installez Docker pour la première fois, il crée automatiquement un réseau bridge 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.

Les conteneurs qui utilisent ce driver, ne peuvent communiquer qu'entre eux, cependant ils ne sont pas accessibles depuis l'extérieur.

Pour que les conteneurs sur le réseau bridge puissent communiquer ou être accessibles du monde extérieur, vous devez configurer le mappage de port.

Mapper les ports :

Voici comment vous pouvez définir le mapping de ports lors de la création ou de l'exécution d'un conteneur Docker :

Utilisation de la commande docker run

Pour mapper un port, vous utilisez l'option -p ou --publish dans la commande docker run. Cette option permet de définir un mapping entre un port sur l'hôte et un port dans le conteneur.

Format de base
Le format de base pour le mapping de ports est :

```
docker run –p 8080:80 –d <IMAGE>
```

Considérations de sécurité
Lors du mapping de ports, il est important de considérer les implications de sécurité, surtout quand vous exposez des ports sur des machines en production. Assurez-vous de n'exposer que les ports nécessaires et d'utiliser des mécanismes de sécurité appropriés comme des pare-feu ou des règles de filtrage IP pour protéger les accès non autorisés.

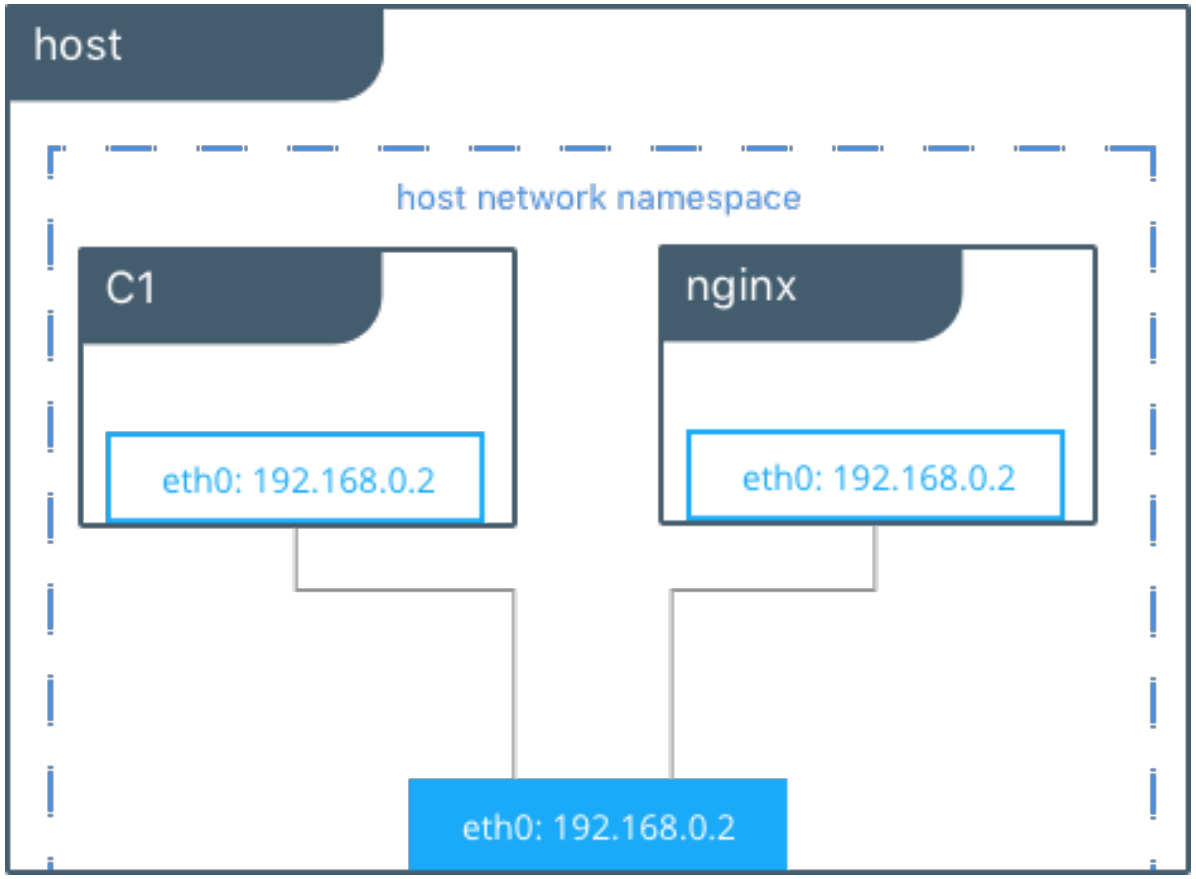
Cas concret :

docker run –it –p 88:88 ––name mycontainer ubuntu:16.04

Je lance un conteneur basé sur l'image ubuntu version 16.04, je le nomme mycontainer, je la lance en interaction avec un terminal, je mappe le port 88 local vers le port 88 du conteneur.

- Le drapeau --it permet de lancer l'image en interaction avec un terminal.
- Le drapeau --name permet de donner un nom à notre conteneur pour le retrouver plus facilement par la suite
- Le drapeau -p permet de mapper un port local sur un port du conteneur

Si je regarde maintenant l'IP de ma machine sur le port 88 je vois bel et bien mon application dans mycontainer qui fonctionne correctement. Le type de réseau utilisé est bridge car il n'est pas précisé dans la commande.



Le driver host :

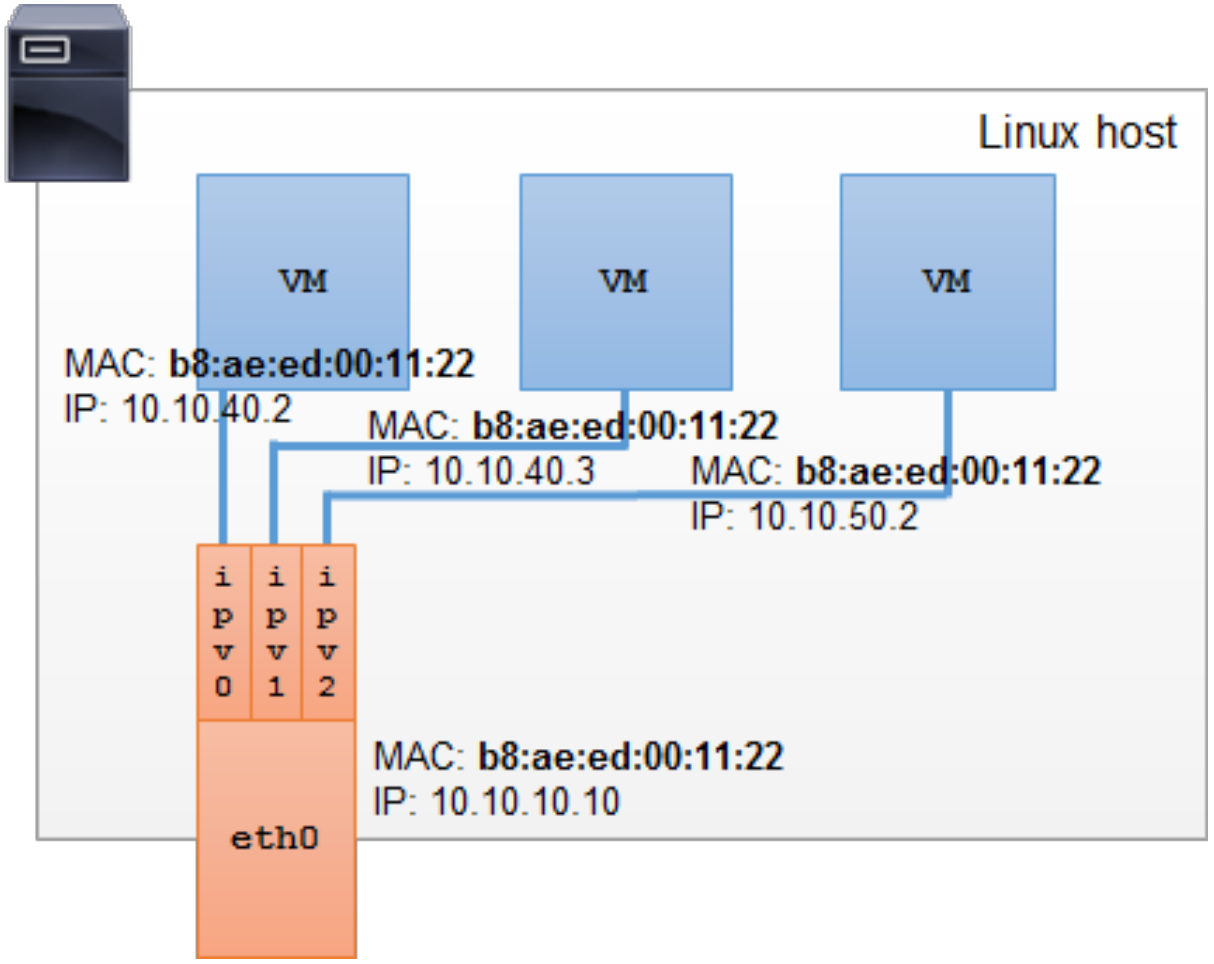
Ce type de réseau permet aux conteneurs d'utiliser la même interface que l'hôte. Il supprime donc l'isolation réseau entre les conteneurs et seront par défaut accessibles de l'extérieur. De ce fait, il prendra la même IP que votre machine hôte.

```
docker run –it ––net host ––name
mycontainer ubuntu:16.04
```

Je n’ai pas besoin ici de mapper de port.

Le driver None :

C'est le type de réseau idéal, si vous souhaitez 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 serveur permet de rendre le conteneur comme un périphérique physique sur votre réseau. Le moteur Docker route le trafic vers les conteneurs en fonction de leurs adresses MAC.

Toute les sous-interface partagent l'adresse MAC avec l'interface parent, mais utilisent une adresse IP distincte.

Création d'un réseau ipvlan :

docker network create -d ipvlan --subnet=10.1.10.0/24 –-gateway=10.1.10.1 -o parent=eno1 myipvlan

Création d'un conteneur ubuntu sur le réseau ipvlan créé précédemment :

docker run –-name='votre_nom' –-hostname='votre_nom' –-net=myipvlan –-ip=10.1.10.10 -it ubuntu:16.04