

TP Administration : conteneurisation (docker)

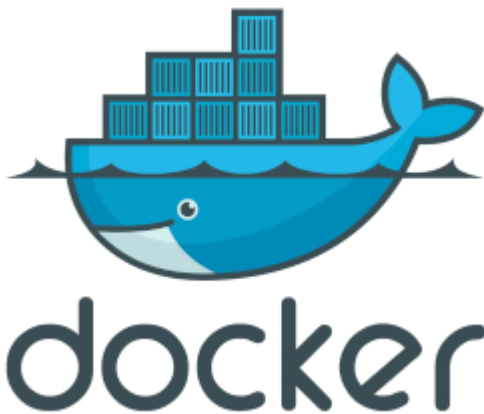
- TP Administration : conteneurisation (docker)
 - Docker
 - Présentation
 - Installation sous Ubuntu
 - Docker Engine
 - Architecture
 - Image
 - Conteneur
 - Technologie
 - Conteneur vs Machine virtuelle
 - Un Conteneur = Un processus
 - Liens
 - Commandes de base
 - Docker Compose
 - Docker Desktop (GUI)
 - Manipulations
 - Hello World!
 - Ubuntu
 - Surveillance
 - Environnement de développement C/C++ (Dockerfile)
 - Environnement de développement web PHP (Docker compose)
 - Étape n°1 : le serveur web
 - Étape n°2 : les fichiers du serveur web
 - Étape n°3 : PHP
 - Étape n°4 : le serveur MySQL/MariaDB
 - Travail demandé
 - Installation et vérification
 - Premier pas
 - Dockerfile
 - Docker compose
 - Bonus

- [Création et publication d'image](#)
- [Le réseau](#)
- [Annexes](#)
 - [Les options](#)
 - [Exemple Dockerfile](#)
 - [Visual Studio Code](#)

Docker

Présentation

[Docker](#) est une plate-forme permettant aux développeurs et aux administrateurs système de créer, d'exécuter et de partager des applications avec des conteneurs.



L'utilisation de conteneurs pour déployer des applications est appelée **conteneurisation**.

La conteneurisation est de plus en plus populaire car les conteneurs sont :

- Flexible : même les applications les plus complexes peuvent être conteneurisées.
- Léger : les conteneurs exploitent et partagent le noyau hôte (efficaces en termes de ressources système vs machines virtuelles).
- Portable : créer localement, déployer sur le cloud et exécuter n'importe où.
- Faible Couplage : les conteneurs sont hautement autonomes et encapsulés (remplacement ou mise à niveau sans perturber les autres).
- Évolutif : augmenter et distribuer automatiquement des répliques de conteneurs dans un centre de données.
- Sécurisé : les conteneurs appliquent des contraintes et des isolements aux processus sans aucune configuration requise de la part de l'utilisateur.

`docker-io` vs `docker-ce` Les anciennes versions de Docker étaient appelées `docker` ou `docker-engine` ou `docker-io`. Le paquet `docker-io` est toujours le nom utilisé par Debian/Ubuntu pour la version Docker fournie sur les dépôts officiels. `docker-ce` est une version certifiée fournie directement par www.docker.com. Docker a une version Entreprise Edition (EE) et une version gratuite Community Edition (CE). Remarque : La principale raison d'utiliser le nom `docker-io` sur la plate-forme Debian/Ubuntu était d'éviter un conflit de nom avec le binaire de la barre d'état système !

Installation sous Ubuntu

Lien : <https://docs.docker.com/engine/install/ubuntu/>

Note

On privilégiera une installation de `docker-ce` avec Ubuntu.

Il est préférable de supprimer une installation précédente de `docker` avant de commencer :

```
sudo apt remove docker docker-engine docker.io
sudo snap remove docker
```

Installation de `docker.io`

```
$ apt-cache policy docker.io
docker.io:
  Installé : (aucun)
  Candidat : 24.0.5-0ubuntu1~22.04.1
  ...
$ sudo apt install docker.io docker-compose-v2 docker-buildx
```

Ou installation de `docker-ce`

Warning

Avant d'installer Docker Community Edition (`docker-ce` de www.docker.com), il faudra peut-être supprimer les anciens paquets :

```
$ sudo apt remove docker docker-engine docker.io containerd runc
```

```
$ sudo apt update
$ sudo apt install ca-certificates curl gnupg
$ sudo install -m 0755 -d /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
$ sudo chmod a+r /etc/apt/keyrings/docker.gpg

$ echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt update

$ sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
```

- Vérification

```
$ systemctl status docker
docker.service - Docker Application Container Engine
Loaded: loaded (/lib/systemd/system/docker.service; enabled;)
Active: active (running) since Fri 2020-05-29 09:01:05 UTC; 56s ago
Docs: https://docs.docker.com
Main PID: 19679 (dockerd)
Tasks: 8
CGroup: /system.slice/docker.service
+-19679 /usr/bin/dockerd -H fd://--containerd=/run/containerd/containerd.sock

$ docker --version
Docker version 19.03.10, build 9424aeae9

$ docker --help
```

💡 Tip

Problème de permission (facultatif) :

```
$ docker images
Got permission denied ...

# Ajouter l'utilisateur au group docker :
$ sudo usermod -aG docker ${USER}

# Redémarrer la session ou :
$ su - ${USER}

$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
```

Docker Engine

[Docker Engine](#) est une technologie de conteneurisation open source pour créer et conteneuriser des applications. Docker Engine agit comme une application client-serveur avec :

- Un **serveur** avec un processus démon `dockerd`

```
$ ps ax | grep docker
25447 ? Ssl 21:52 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/conta.
```

- Un **client** avec une interface en ligne de commande (CLI) client `docker`

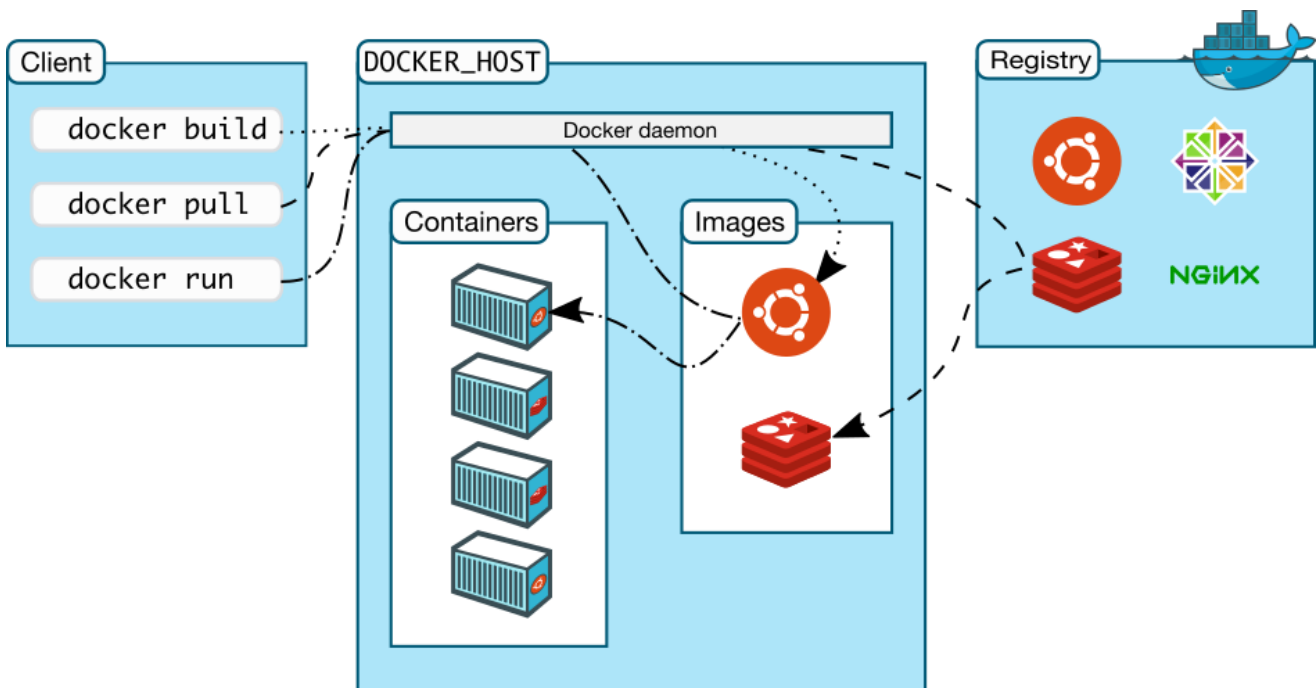
```
$ docker --version
Docker version 19.03.10, build 9424aeae9
```

```
$ docker --help
```

- Une **API** pour les programmes qui s'interfaçent avec le démon Docker.

Architecture

Docker utilise une architecture client/serveur.



Le client Docker discute avec le démon Docker, qui s'occupe de la construction, de l'exécution et de la distribution de vos conteneurs Docker.

Le client et le démon Docker peuvent s'exécuter sur le même système, ou vous pouvez connecter un client Docker à un démon Docker distant.

Image

Une **image** est un modèle (template) en **lecture seule** avec des instructions pour créer un **conteneur** Docker.

Souvent, une image est basée sur une autre image, avec une personnalisation supplémentaire.

Il est possible de créer ses propres images ou utiliser uniquement celles créées par d'autres et publiées dans un registre (registry).

Pour créer une image, on utilise un fichier `Dockerfile` avec une syntaxe simple pour définir les étapes nécessaires pour créer l'image et l'exécuter.

Chaque instruction d'un `Dockerfile` crée un calque dans l'image. Lorsque on modifie le `Dockerfile` et on reconstruit l'image, seuls les calques qui ont été modifiés sont reconstruits.

Cela fait partie de ce qui rend les images si légères, petites et rapides par rapport aux autres technologies de virtualisation.

Conteneur

Un **conteneur** est une instance exécutable d'une **image**.

Un conteneur est défini par son image ainsi que par les options de configuration fournis à la création ou au démarrage.

Il est possible de créer, démarrer, arrêter, déplacer ou supprimer un conteneur à l'aide de la commande `docker`.

On peut connecter un conteneur à un ou plusieurs réseaux, y attacher du stockage ou même créer une nouvelle image en fonction de son état actuel.

Par défaut, un conteneur est relativement bien isolé des autres conteneurs et de sa machine hôte. Vous pouvez contrôler l'isolement du réseau, du stockage ou d'autres sous-systèmes des autres conteneurs ou de la machine hôte.

Lorsqu'un conteneur est supprimé, toute modification de son état qui n'est pas stockée dans un stockage persistant disparaît.

Technologie

Docker est écrit en [Go](#) et profite de plusieurs fonctionnalités du noyau Linux :

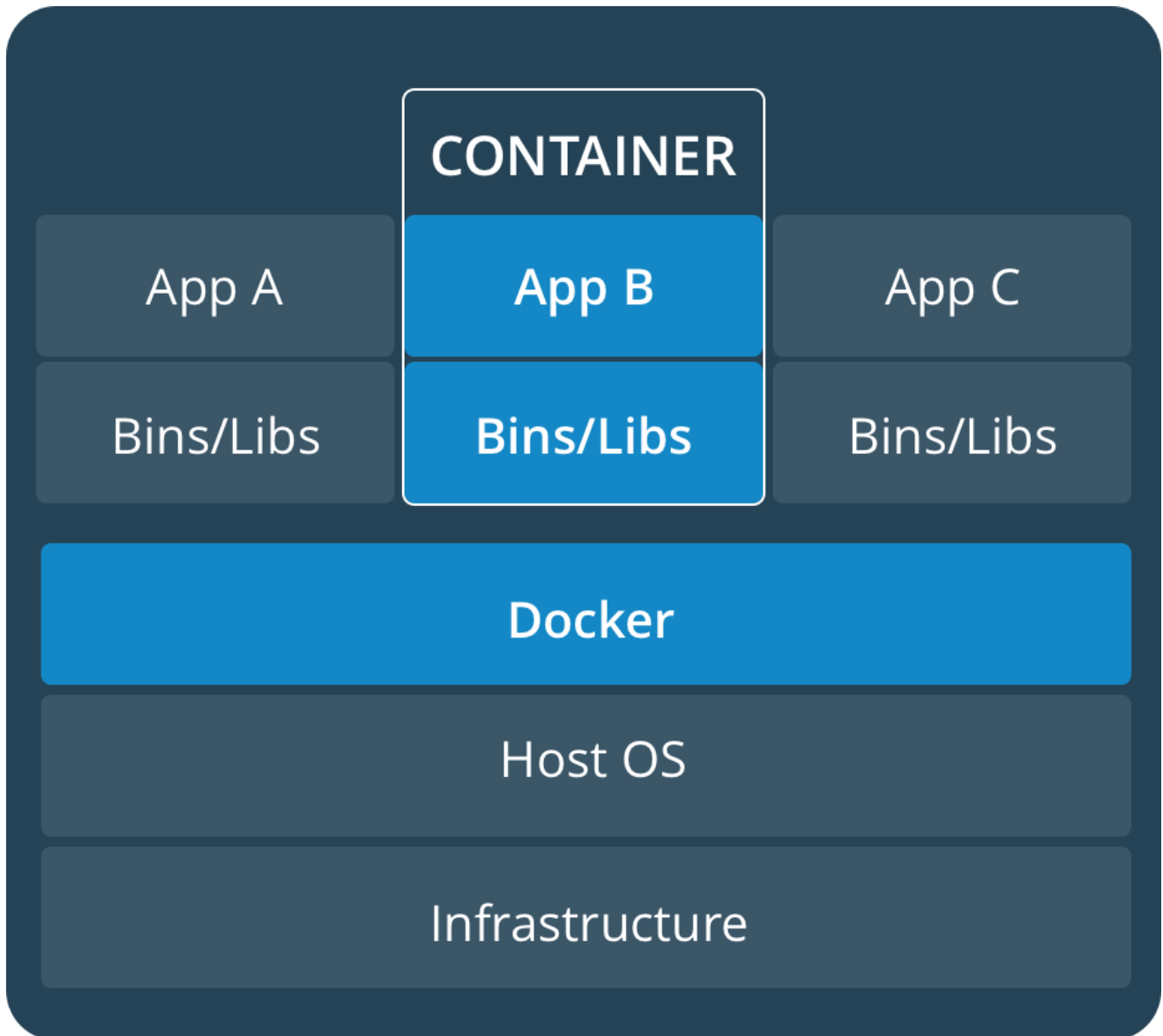
- [Espaces de noms \(namespaces\)](#) : Docker utilise les espaces de noms pour isoler l'espace de travail du conteneur. Chaque aspect d'un conteneur s'exécute dans un espace de noms distinct et son accès est limité à cet espace de noms. [Docker Engine](#) utilise des espaces de noms suivants sous Linux : pid, net, ipc, mnt et uts.
- [Groupes de contrôle \(cgroups\)](#) : Docker utilise les groupes de contrôle ([cgroups](#)) sur Linux pour partager les ressources matérielles disponibles avec les conteneurs et d'appliquer éventuellement des limites et des contraintes.
- [UnionFS](#) : Docker utilise le système de fichiers Union (et plusieurs variantes notamment AUFS, btrfs, vfs et DeviceMapper) pour fournir les blocs de construction des conteneurs. UnionFS fonctionne en créant des couches, ce qui le rend très léger et rapide.

[Docker Engine](#) combine les espaces de noms, les groupes de contrôle et UnionFS dans un [wrapper](#) appelé format de **conteneur**.

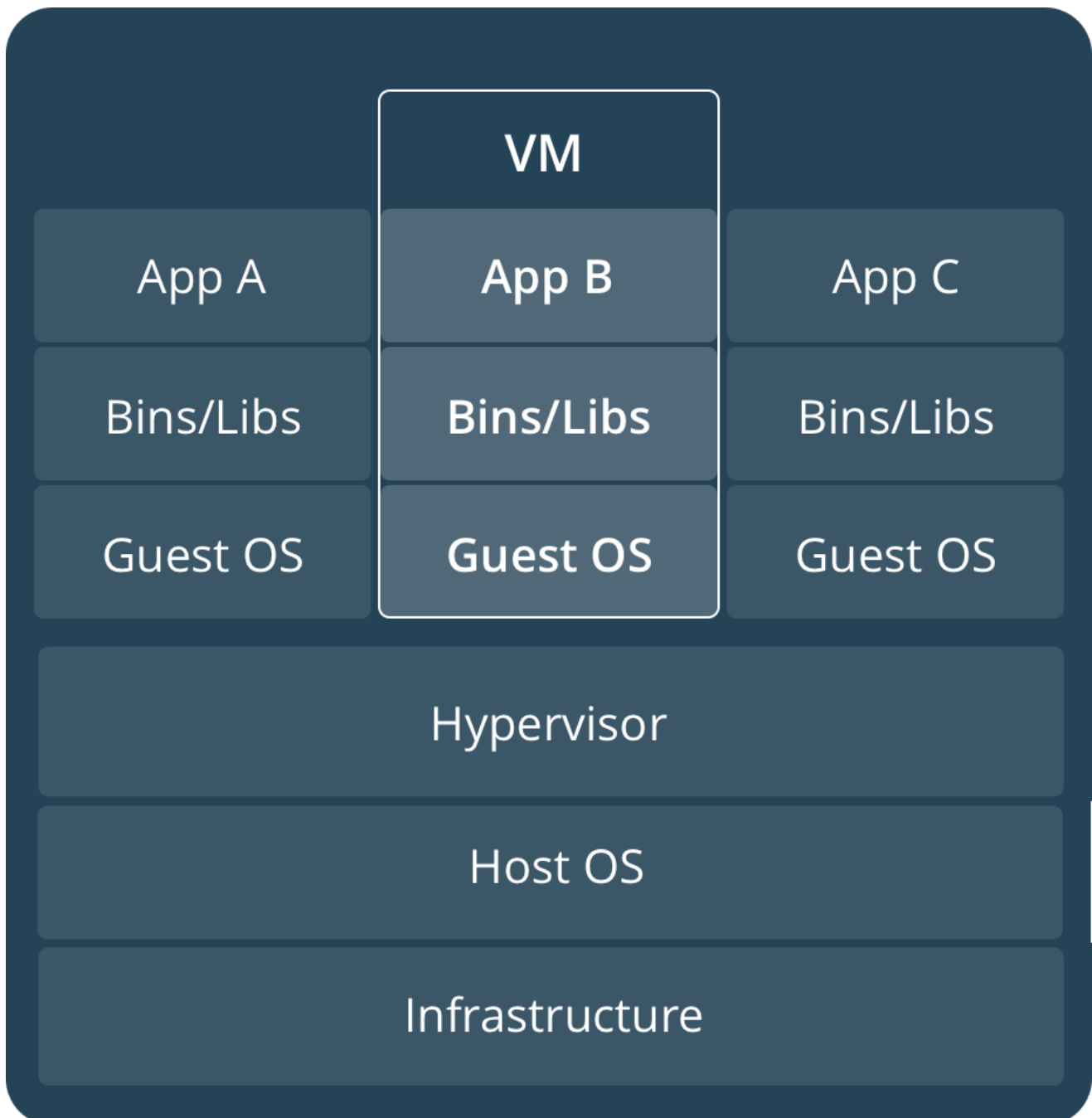
Le format de conteneur par défaut est `libcontainer`. Suivant des principes de normalisations `libcontainer` a été nommé `runc` qui gère maintenant l'exécution du conteneur. Le découpage interne de Docker a fait apparaître une couche `containerd` sous Docker Engine. Ce qui donne au final, le système à couche suivant: `dockerd` (démon) - `containerd` - `runc` - [kernel](#) Linux ([cgroups](#) et [namespaces](#)).

Conteneur vs Machine virtuelle

- Un conteneur s'exécute nativement sous Linux et partage le noyau de la machine hôte avec d'autres conteneurs. Il exécute simplement un processus ce qui le rend léger.



- Une machine virtuelle (VM) exécute un système d'exploitation « invité » complet avec un accès virtuel aux ressources de l'hôte via un **hyperviseur**. En général, les machines virtuelles “consomment” plus de ressources que les besoins réelles de l'application mais offrent une meilleure isolation.



cf. Cours : [cours-virtualisation.pdf](#)

Un Conteneur = Un processus

Le principe de fonctionnement de Docker implique qu'un conteneur représente un seul processus du système.

Évidemment, un processus peut se "forker" en plusieurs processus.

💡 Tip

Il est possible de contourner ce principe : [Run multiple processes in a container](#). Par exemple, en utilisant un gestionnaire de processus comme `supervisord` (qui sera le processus principal pour lui faire gérer plusieurs processus secondaires dans le même conteneur).

Liens

Quelques liens :

- Docker CLI cheat sheet : https://docs.docker.com/get-started/docker_cheatsheet.pdf
- Docker Engine :
 - <https://docs.docker.com/engine/>
 - <https://docs.docker.com/engine/install/>
 - https://docs.docker.com/engine/reference/commandline/container_run/
- Guide de démarrage :
 - <https://docs.docker.com/get-started/part1/>
 - <https://docs.docker.com/get-started/part2/>
 - <https://docs.docker.com/get-started/part3/>
- Dockerfile :
 - <https://docs.docker.com/engine/reference/builder/>
 - <https://docs.docker.com/get-started/part2/#sample-dockerfile>
- Docker Hub : <https://hub.docker.com/>
- Tutoriel d'introduction à Docker : <https://docker-curriculum.com/>
- Ressources sur Docker : <https://github.com/veggie Monk/awesome-docker>
- Utilisation en ligne : <https://training.play-with-docker.com/>

Commandes de base

- Aide :

```
$ docker --help
```

- Lister :

```
# Liste les images déjà téléchargées :  
$ docker images  
REPOSITORY TAG IMAGE ID CREATED SIZE  
  
# Liste tous les conteneurs Docker (en cours d'exécution ou arrêtés) :
```

```
$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

# Affiche les informations
$ docker info
```

- Récupérer une image :

```
# Recherche une image
$ docker search hello-world

# Récupère une image
$ docker pull hello-world
```

Exécuter un conteneur :

```
# Démarre un conteneur à partir d'une image, avec un nom personnalisé :
$ docker run --name <nom_conteneur> image

# Démarre un conteneur en mode interactif :
$ docker run -it --name <nom_conteneur> image shell

# Démarre ou arrête un conteneur existant :
$ docker start|stop <nom_conteneur>

# Ouvre un shell dans un conteneur déjà en cours d'exécution :
$ docker exec -it <nom_conteneur> sh
```

Un conteneur ne reste en vie que si un processus est actif. Pour arrêter un conteneur, il suffit de faire un `exit` (dans le shell de celui-ci) ou un `docker stop`.

Nettoyer :

```
# Arrêter tous les conteneurs
$ docker stop $(docker ps -aq)

# Supprimer tous les conteneurs
$ docker rm $(docker ps -aq)

# Supprimer toutes les images
$ docker rmi $(docker images -q)
```

Tout nettoyer (les conteneurs, réseaux, images non utilisés, et éventuellement, les volumes) :

```
$ docker system prune
```

Pour la journalisation (log), on utilise `docker logs` et, `docker stats` pour l'usage des ressources CPU, mémoire et réseau.

Docker Compose

[Docker Compose](#) est un outil permettant de définir et exécuter des applications à partir de multiples conteneurs.

Lorsque plusieurs conteneurs doivent interagir (par exemple, un serveur web et une base de données), il est indispensable d'utiliser [Docker Compose](#).

La configuration des conteneurs est réalisée dans un fichier `docker-compose.yaml` (au format de représentation de données [YAML](#)).

Le démarrage est assurée par une commande unique `docker compose`.

Docker Desktop (GUI)

Docker Desktop est une application GUI (Graphical User Interface) qui permet de créer, partager et exécuter des conteneurs, applications et images.

Lien : <https://docs.docker.com/desktop/>

Manipulations

 Important

cf. [Travail demandé](#)

Hello World!

Lien : <https://hub.docker.com/>

Évidemment, Docker dispose de son "Hello world!", un conteneur qui ne fait rien de particulier à part afficher le célèbre message :

```
# Démarrer un conteneur
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:c79d06dfdfd3d3eb04cafd0dc2bacab0992ebc243e083cabe208bac4dd7759e
Status: Downloaded newer image for hello-world:latest
```

```

Hello from Docker!
This message shows that your installation appears to be working correctly.
...

# Lister les images
$ docker images
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
hello-world     latest      9c7a54a9a43c  7 months ago  13.3kB

# Lister les conteneurs
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND        CREATED        STATUS
f70adc74fc6a  hello-world   "/hello"      39 seconds ago Exited (0) 39 seconds

# Supprimer le conteneur
$ docker rm f70adc74fc6a

# Supprimer l'image
$ docker rmi hello-world

# Un conteneur avec un nom et qui se supprime automatiquement (mais pas l'image)
$ docker run --rm --name=mon-conteneur hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:c79d06dfdfd3d3eb04cafd0dc2bacab0992ebc243e083cabe208bac4dd7759e
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

$ docker ps -a
CONTAINER ID   IMAGE          COMMAND        CREATED        STATUS        PORTS        NAMES

$ docker images
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
hello-world     latest      9c7a54a9a43c  7 months ago  13.3kB

```

Ubuntu

Rechercher une image Ubuntu :

```

$ docker search ubuntu
NAME                DESCRIPTION
ubuntu              Ubuntu is a Debian-based Linux operating sys...
websphere-liberty  WebSphere Liberty multi-architecture images ...
open-liberty        Open Liberty multi-architecture images based...
neurodebian         NeuroDebian provides neuroscience research s...
ubuntu-debootstrap DEPRECATED; use "ubuntu" instead

```

```

ubuntu-upstart          DEPRECATED, as is Upstart (find other proces...
ubuntu/nginx           Nginx, a high-performance reverse proxy & we...
ubuntu/squid           Squid is a caching proxy for the Web. Long-t...
...

```

```
# Rechercher une image Ubuntu officielle
```

```
$ docker search --filter "is-official=true" ubuntu
```

NAME	DESCRIPTION	STARS
ubuntu	Ubuntu is a Debian-based Linux operating sys...	16657
websphere-liberty	WebSphere Liberty multi-architecture images ...	297
open-liberty	Open Liberty multi-architecture images based...	62
neurodebian	NeuroDebian provides neuroscience research s...	105
ubuntu-debootstrap	DEPRECATED; use "ubuntu" instead	52
ubuntu-upstart	DEPRECATED, as is Upstart (find other proces...	115

Récupérer une image Ubuntu :

```

$ docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
58690f9b18fc: Pull complete
b51569e7c507: Pull complete
da8ef40b9eca: Pull complete
fb15d46c38dc: Pull complete
Digest: sha256:1f1a2d56de1d604801a9671f301190704c25d604a416f59e03c04f5c6ffee0d
Status: Downloaded newer image for ubuntu:16.04
docker.io/library/ubuntu:16.04

```

What's Next?

View a summary of image vulnerabilities and recommendations → [docker scout](#) [q](#)

```
# Lister les images
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	9c7a54a9a43c	7 months ago	13.3kB
ubuntu	16.04	b6f507652425	2 years ago	135MB

Démarrer le conteneur (avec un nom) en mode interactif (shell bash) :

```
$ docker run -it --name=mon-ubuntu-1604 ubuntu:16.04 /bin/bash
```

```
root@6662637532cd:/# cat /etc/os-release
```

```
NAME="Ubuntu"
```

```
VERSION="16.04.7 LTS (Xenial Xerus)"
```

```
...
```

```
root@6662637532cd:/# apt-get update
```

```
Get:1 http://archive.ubuntu.com/ubuntu xenial >InRelease [247 kB]
```

```
...
```

```
root@6662637532cd:/# exit
```

```
# Lister les conteneurs
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
6662637532cd	ubuntu:16.04	"/bin/bash"	2 minutes ago	Exited (0) 47 seconds

```
# Démarrer un conteneur nommé
```

```
$ docker start mon-ubuntu-1604  
mon-ubuntu-1604
```

```
# Ouvrir un shell
```

```
$ docker exec -it mon-ubuntu-1604 /bin/bash
```

```
# Exécuter une commande
```

```
$ docker exec mon-ubuntu-1604 cat /etc/os-release  
NAME="Ubuntu"  
VERSION="16.04.7 LTS (Xenial Xerus)"  
...
```

```
# Lister les conteneurs
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
6662637532cd	ubuntu:16.04	"/bin/bash"	6 minutes ago	Up 3 minutes	

```
# Arrêter un conteneur
```

```
$ docker stop mon-ubuntu-1604  
mon-ubuntu-1604
```

```
# Lister les conteneurs
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
6662637532cd	ubuntu:16.04	"/bin/bash"	6 minutes ago	Exited (0) 1 second

```
# Supprimer le conteneur et l'image
```

```
$ docker rm 6662637532cd  
$ docker rmi b6f507652425
```

Surveillance

```
# Lister tous les conteneurs Docker
```

```
$ docker ps -a
```

```
# Visualiser l'activité d'un conteneur
```

```
$ docker logs -ft <CONTAINER ID>
```

```
# Visualiser les statistiques CPU/Mémoire/Réseau
$ docker stats <CONTAINER ID>
```

Environnement de développement C/C++ (Dockerfile)

Lien : <https://docs.docker.com/engine/reference/builder/>

Pour créer une nouvelle image (et éventuellement la publier), il faut fournir un fichier `DockerFile`.

Les instructions de base d'un `DockerFile` :

Instruction	Description	Exemple	Remarques
<code>FROM</code>	Spécifie l'image Docker à utiliser	<code>FROM ubuntu:20.04</code>	Obligatoire, généralement une seule
<code>RUN</code>	Construit l'image	<code>RUN apt-get update && apt-get install -y g++ make git</code>	L'utilité principale est d'installer les fonctionnalités pré-requises
<code>COPY</code>	Copie de fichiers de l'hôte vers le conteneur	<code>COPY . /src/</code>	
<code>ADD</code>	idem <code>COPY</code>	<code>ADD . /src/</code>	<code>ADD entra</code>
<code>WORKDIR</code>	Définit le répertoire courant (idem à la commande <code>cd</code>)	<code>WORKDIR /src</code>	
<code>CMD</code>	Exécute une commande au démarrage du conteneur (cf. <code>ENTRYPOINT</code>)	<code>CMD ["make", "tests"]</code>	Voir aussi : <code>CMD make tests</code>

Pour `CMD` :

Format	Syntaxe	Exemple	Commande exécutée

Format	Syntaxe	Exemple	Commande exécutée
terminal	CMD <commande>	CMD make tests	/bin/sh -c "make tests"
exécution	CMD ["executable", "param1", "param2"]	CMD ["make", "tests"]	make tests

⚠ Important

Dans le format "terminal", l'exécutable `/bin/sh` doit être présent dans l'image pour que la `<commande>` puisse s'exécuter. Lors d'un arrêt "normal" du conteneur, seul le processus (PID 1) sera arrêté par un signal `SIGTERM` (`sh` dans le format terminal) et les autres processus seront simplement "tués" (avec `SIGKILL`).

Si besoin :

```
$ mkdir ubuntu-dev
$ cd ubuntu-dev
$ touch Dockerfile
```

- Éditer le fichier `DockerFile` :

```
$ cd ubuntu-dev
$ vim Dockerfile
```

```
FROM ubuntu:20.04
RUN apt-get update && apt-get install -y g++ make git
RUN rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
COPY . /src/
WORKDIR /src
CMD ["make", "tests"]
```

- Créer une image `ubuntu-dev` (ici avec le `login` `tvaira`) :

```
$ docker build -t tvaira/ubuntu-dev .
[+] Building 1.0s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 224B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:20.04
```

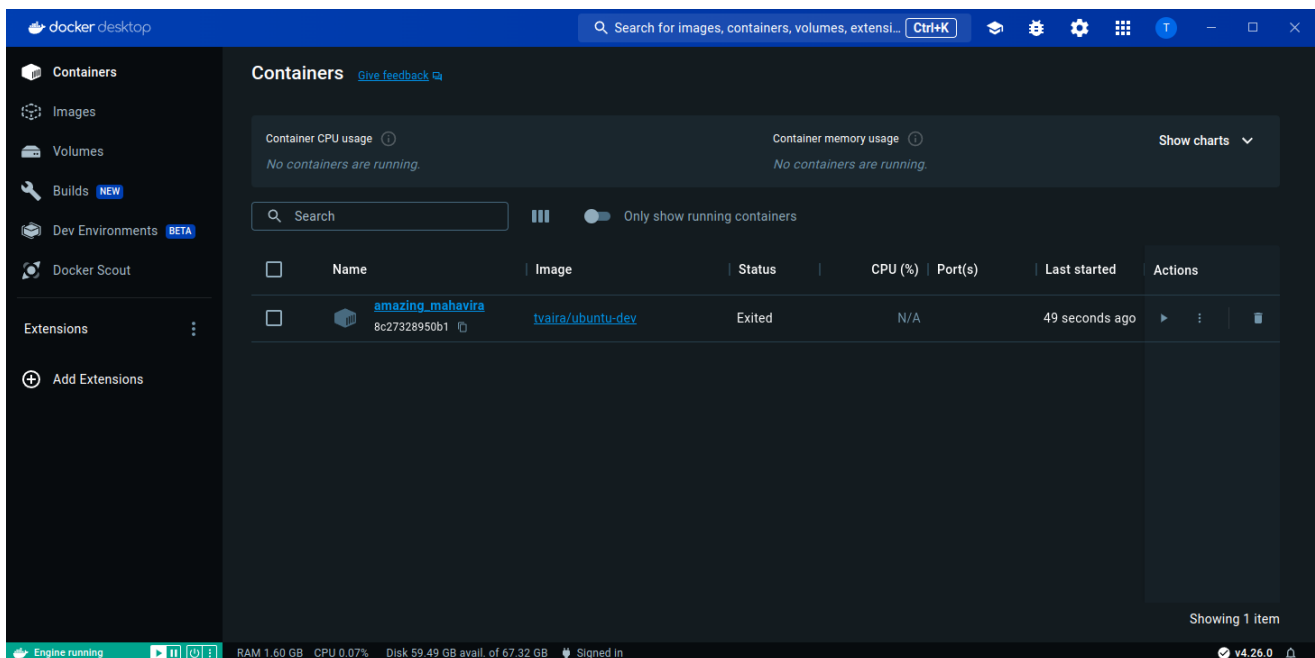
```

=> [1/5] FROM docker.io/library/ubuntu:20.04@sha256:f5c3e53367f142fab0b499085!
=> [internal] load build context
=> => transferring context: 10.24MB
=> CACHED [2/5] RUN apt-get update && apt-get install -y g++ make git
=> CACHED [3/5] RUN rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
=> [4/5] COPY . /src/
=> [5/5] WORKDIR /src
=> exporting to image
=> => exporting layers
=> => writing image sha256:a2c673be9dce75ec307f01642be59cd84be97d429937baae01!
=> => naming to docker.io/tvaira/ubuntu-dev

```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
tvaira/ubuntu-dev	latest	b296e91219e0	20 hours ago	412MB



- Démarrer un conteneur à partir de l'image créée :

Rappel : On peut ajouter l'option `--rm` à la commande `docker run` si on souhaite supprimer automatiquement le conteneur à sa sortie.

```

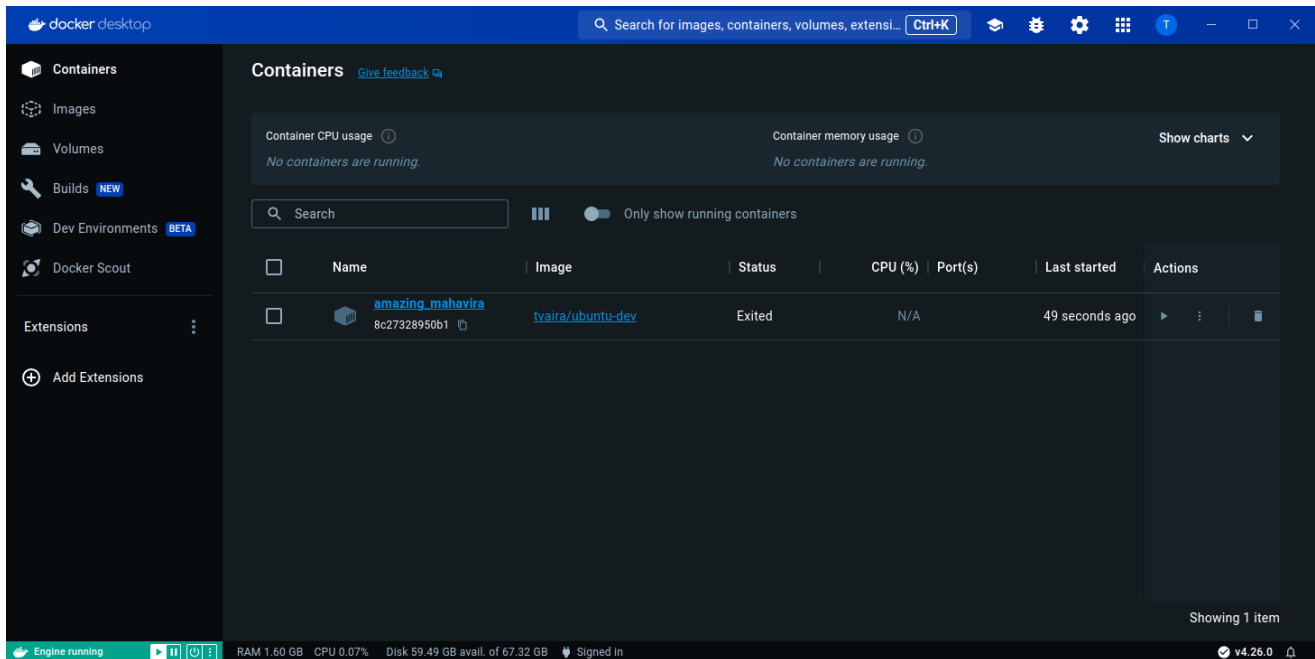
$ docker run -t tvaira/ubuntu-dev
g++ -std=c++11 -Wall -I. -I./tests -c      -c -o tests/testUnitairePoint.o tests/
g++ -std=c++11 -Wall -I. -I./tests -c      -c -o tests/testsUnitaires.o tests/t
g++ -std=c++11 -Wall -I. -I./tests -c Point.cpp
g++ -o tests/testsUnitaires.out tests/testUnitairePoint.o tests/testsUnitaires
./tests/testsUnitaires.out

```

```
=====
All tests passed (11 assertions in 5 test cases)
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
8c27328950b1	tvaira/ubuntu-dev	"make tests"	26 seconds ago	Exited (0)



Le conteneur est dans l'état `Exited` car un conteneur ne reste en vie que si un processus est actif. Ici, la commande `make` s'est exécutée puis terminée.

On peut exécuter un shell ou des commandes en mode interactif :

- Où suis-je ?

```
$ docker run -ti tvaira/ubuntu-dev pwd
/src
```

- Quels sont les fichiers ?

```
$ docker run -ti tvaira/ubuntu-dev ls -l
total 24
-rw-rw-r-- 1 root root 185 Dec 8 09:34 Dockerfile
-rw-rw-r-- 1 root root 956 Dec 6 13:15 Makefile
-rw-rw-r-- 1 root root 854 Dec 7 14:01 Point.cpp
-rw-r----- 1 root root 506 Dec 6 13:04 Point.h
-rw-rw-r-- 1 root root 1184 Dec 6 13:08 testPoint.cpp
drwxrwxr-x 2 root root 4096 Dec 8 09:43 tests
```

- un shell :

```
$ docker run -ti tvaira/ubuntu-dev /bin/bash
root@2e7c8319edb1:/src# make
```

```
...
root@2e7c8319edb1:/src# exit
```

Et donc :

```
$ docker ps -a
CONTAINER ID   IMAGE                COMMAND              CREATED        STATUS
2e7c8319edb1   tvaira/ubuntu-dev   "/bin/bash"         38 seconds ago Exited
10358078d473   tvaira/ubuntu-dev   "ls -l"             59 seconds ago Exited
a8e1ed40e167   tvaira/ubuntu-dev   "pwd"               About a minute ago Exited
8c27328950b1   tvaira/ubuntu-dev   "make tests"        5 minutes ago  Exited
```

⚠ Warning

L'image a été créée en copiant les fichiers de développement du répertoire de l'hôte. Si on modifie ces fichiers, ils ne seront pas modifiés dans l'image !

Pour éviter de recréer une image, il est possible d'utiliser un **volume** :

```
$ vim Point.cpp

$ docker run --rm -t -v ./src tvaira/ubuntu-dev
g++ -std=c++11 -Wall -I. -I./tests -c Point.cpp
g++ -o tests/testsUnitaires.out tests/testUnitairePoint.o tests/testsUnitaires
./tests/testsUnitaires.out

~~~~~
testsUnitaires.out is a Catch v2.13.7 host application.
Run with -? for options

-----
2 : Constructeur d'initialisation
-----
tests/testUnitairePoint.cpp:12
.....

tests/testUnitairePoint.cpp:17: FAILED:
  CHECK( p1.getX() == Approx(1.) )
with expansion:
  0.0 == Approx( 1.0 )
...
=====
test cases: 5 | 2 passed | 3 failed
assertions: 11 | 7 passed | 4 failed

make: *** [Makefile:28: tests] Error 4
```

Environnement de développement web PHP (Docker compose)

Si besoin :

```
$ mkdir php-dev
$ cd php-dev
$ touch docker-compose.yml
```

Étape n°1 : le serveur web

On va créer un service web avec [NGINX](#).

Note

[NGINX](#) est un logiciel libre de serveur Web. Depuis avril 2019, nginx est le serveur web le plus utilisé au monde d'après Netcraft, ou le deuxième serveur le plus utilisé d'après W3techs. [NGINX](#) est un **système asynchrone** par opposition aux serveurs synchrones (comme [Apache HTTP Server](#)) où chaque requête est traitée par un processus dédié. Au lieu d'exploiter une architecture parallèle et un multiplexage temporel des tâches par le système d'exploitation, NGINX utilise les changements d'état pour gérer plusieurs connexions en même temps ; le traitement de chaque requête est découpé en de nombreuses mini-tâches et permet ainsi de réaliser un multiplexage efficace entre les connexions. Comme [Apache](#), [NGINX](#) est très modulaire : un noyau minimal et de nombreux modules, venant compléter les fonctions de base. Par contre, ces modules sont liés au serveur lors de la compilation. NGINX ne supporte pas les bibliothèques dynamiques partagées.

Liens :

- <http://nginx.org/en/>
- <https://www.nginx.com/resources/wiki/>

Initialiser un fichier `docker-compose.yml` avec le contenu suivant :

```
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
```

Instruction	Description
-------------	-------------

Instruction	Description
<code>version</code>	La version de YAML
<code>services</code>	La liste de tous les services qui seront gérés
<code>web</code>	Le service web
<code>image</code>	L'image utilisée pour le service, ici la dernière (latest) image officielle NGINX . Il est possible de préciser une version, par exemple <code>nginx:1.18.0</code>
<code>ports</code>	Le transfert de port entre la machine locale (HOST) et le conteneur (CONTAINER) avec la syntaxe suivante <code>[HOST:]CONTAINER</code> . Ici, le port <code>80</code> du serveur <code>nginx</code> du conteneur sera accessible sur le port <code>8080</code> de la machine locale.

Tester l'exécution du serveur.

Pour créer et démarrer le conteneur :

```
$ docker compose up -d
[+] Running 8/8
  ✓ web 7 layers [#####]          0B/0B      Pulled
...
[+] Running 2/2
  ✓ Network php-dev_default      Created
  ✓ Container php-dev-web-1      Started

$ docker compose ls
NAME                STATUS              CONFIG FILES
php-dev             running(1)          ../tp-admin-docker/php-dev/docker-com

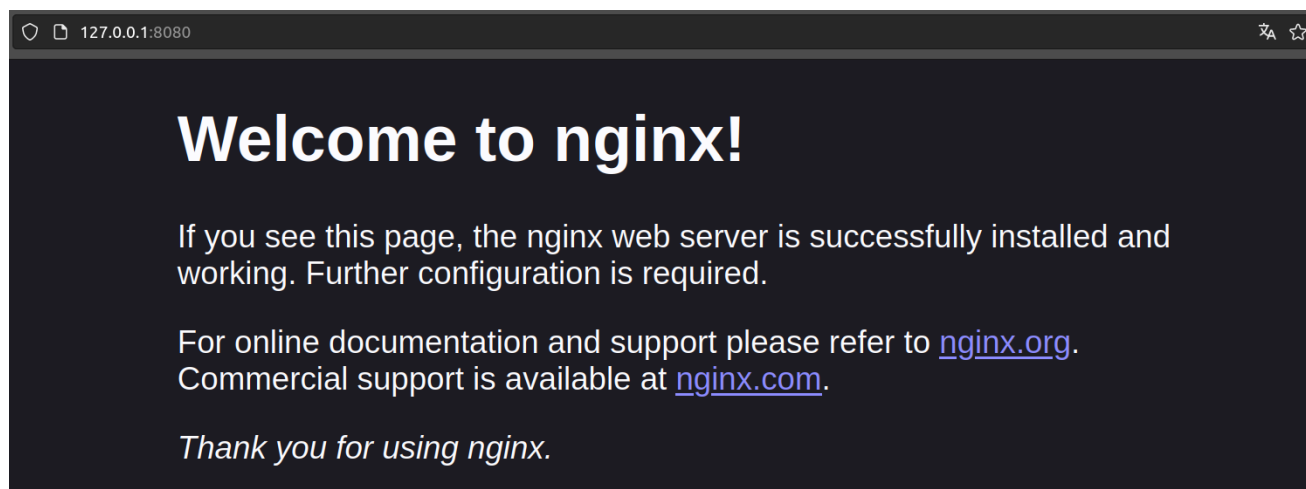
$ docker compose ps
$ docker ps -a
CONTAINER ID   IMAGE             COMMAND                                     CREATED        STATUS
171e35dca825   nginx:latest     "/docker-entrypoint..."                 49 seconds ago Up 47 s

$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest   a8758716bb6a  3 months ago  187MB

$ docker compose top
php-dev-web-1
UID          PID        PPID         C    STIME   TTY   TIME      CMD
root         1500170    1500149     0    13:54  ?    00:00:00  nginx: master pro
systemd+    1500219    1500170     0    13:54  ?    00:00:00  nginx: worker pro
...
```

L'option `-d` est important car elle permet de démarrer le conteneur en mode **détaché** (en arrière-plan).

Le serveur web **NGINX** sera donc accessible sur la machine locale :
`http://127.0.0.1:8080`



Étape n°2 : les fichiers du serveur web

Rappel : comme le serveur fonctionne dans un conteneur, il n'a accès à aucun fichier de la machine locale. Cependant, Docker permet de spécifier un volume (un fichier ou un répertoire) de la machine locale qui sera partagé avec le conteneur.

On a besoin de deux volumes :

- pour la configuration du serveur nginx (le fichier `nginx.conf`), et
- pour les fichiers de l'application web (le répertoire `app` où on placera les scripts PHP).

Modifier le fichier `docker-compose.yml` avec le contenu suivant :

```
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/nginx.conf
      - ./app:/app
```

Instruction	Description
-------------	-------------

Instruction	Description
<code>volumes</code>	Les chemins hôtes de montage ou des volumes nommés qui sont accessibles par des conteneurs de service avec la syntaxe suivante : <code>VOLUME:CONTAINER_PATH[:ACCESS_MODE]</code>

Lien : <https://docs.docker.com/compose/compose-file/05-services/#volumes>

Créer le fichier `nginx.conf` :

```
server {  
    listen 80 default_server;  
    root /app/public;  
}
```

Directive	Description
<code>listen</code>	Le port d'écoute par défaut du serveur
<code>root</code>	La racine des fichiers accessibles par le serveur

Créer le fichier `app/public/index.html` :

```
$ mkdir -p app/public  
$ echo '<h1>Hello, World!</h1>' > app/public/index.html
```

Redémarrer le service web.

```
$ docker compose stop web  
$ docker compose up -d
```



Étape n°3 : PHP

On va ajouter un service pour [PHP](#).

Alors qu'Apache intègre l'interpréteur PHP dans chaque requête, NGINX nécessite un programme externe pour gérer le traitement PHP et agir comme un pont (comme un serveur de proxy inverse) entre l'interpréteur PHP et le serveur web. pour obtenir de meilleures performances globales. Le gestionnaire de processus PHP-FPM (FastCGI Process Manager) interprète alors les requêtes HTTP qu'il reçoit de NGINX et exécute les scripts PHP pour générer les réponses HTTP correspondantes pour que NGINX les renvoie au client HTTP.

Il faut utiliser PHP-FPM (FastCGI Process Manager) avec NGINX.

Modifier le fichier `docker-compose.yml` avec le contenu suivant :

```
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/nginx.conf
      - ./app:/app
  php:
    image: php:fpm-latest
    volumes:
      - ./app:/app
```

On utilise ici la dernière version mais on pourrait spécifier une autre version :

`php:7.3-fpm` , `php:7.4-fpm` , `php:8.0-fpm` ...

Modifier le fichier `nginx.conf` :

```
server {
    listen 80 default_server;
    root /app/public;

    index index.php index.html index.htm;

    location ~ \.php$ {
        fastcgi_pass php:9000;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

Directive	Description
-----------	-------------

Directive	Description
<code>index</code>	Liste de recherche pour définir la "page" d'accueil (racine du serveur)
<code>location</code>	Extension des scripts pour le service php

Créer le script `app/public/index.php` :

```
<?php

// PHP
echo "<p>Version de PHP : ".PHP_VERSION."</p>";
phpinfo();

?>
```

Redémarrer le service web.

```
$ docker compose stop web
$ docker compose up -d
```

Version de PHP : 8.3.2

PHP Version 8.3.2	
System	Linux 0165a6248ca4 5.19.0-32-generic #33~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon Jan 30 17:03:34 UTC 2 x86_64
Build Date	Jan 19 2024 22:47:35
Build System	Linux - Docker
Build Provider	https://github.com/docker-library/php
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--with-pic' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-iconv' '--with-openssl' '--with-readline' '--with-zlib' '--disable-phpdbg' '--with-pear' '--with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--enable-fpm' '--with-fpm-user=www-data' '--with-fpm-group=www-data' 'build_alias=x86_64-linux-gnu'
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-fpm.ini, /usr/local/etc/php/conf.d/docker-php-ext-sodium.ini
PHP API	20230831

Étape n°4 : le serveur MySQL/MariaDB

MySQL est un système de gestion de bases de données relationnelles (SGBDR) SQL.

Il est distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, Informix et Microsoft SQL Server.

MariaDB est un [système de gestion de bases de données SQL](#) édité sous licence GPL. Il s'agit d'un fork communautaire de [MySQL](#).

En 2009, à la suite du rachat de [MySQL](#) par Sun Microsystems et des annonces du rachat de Sun Microsystems par Oracle Corporation, Michael Widenius, fondateur de MySQL, quitte cette société pour lancer le projet [MariaDB](#), dans une démarche visant à remplacer MySQL tout en assurant l'interopérabilité. Le nom vient de la 2e fille de Michael Widenius, Maria (la première s'appelant My). MariaDB a été choisi par défaut sur les distributions « Debian ».

On va ajouter un service pour [MySQL](#).

Modifier le fichier `docker-compose.yml` avec le contenu suivant :

```
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/nginx.conf
      - ./app:/app
  php:
    image: php:fpm-latest
    volumes:
      - ./app:/app
  mysql:
    image: mariadb:latest
    environment:
      MYSQL_ROOT_PASSWORD: 'password'
      MYSQL_USER: 'bts'
      MYSQL_PASSWORD: 'password'
      MYSQL_DATABASE: 'bd'
    volumes:
      - mysqldata:/var/lib/mysql
    ports:
      - 3306:3306
volumes:
  mysqldata: {}
```

Instruction	Description
-------------	-------------

Instruction	Description
<code>environment</code>	Les variables d'environnement définies dans le conteneur avec la syntaxe <code>VARIABLE: 'valeur'</code>

```
<?php

// PHP
echo "<p>Version de PHP : ".PHP_VERSION."</p>";
//phpinfo();

// MySQL
$pdo = new PDO('mysql:dbname=bd;host=mysql', 'bts', 'password', [PDO::ATTR_ERR
$resultat = $requete->fetch();
echo "<p>Version de MySQL : ".$resultat['Value']."</p>";

?>
```

Redémarrer le service web.

```
$ docker compose stop web php
$ docker compose up -d
```

Comme il n'y a pas de pilote PHP MySQL installé, on obtient un message d'erreur :

```
Fatal error: Uncaught PDOException: could not find driver in /app/public/index
#0 /app/public/index.php(8): PDO->__construct('mysql:dbname=bd...', 'bts', Ob
#1 {main} thrown in /app/public/index.php on line 8
```

L'[extension PHP PDO \(Data Objects\)](#) définit une interface d'abstraction pour accéder à une base de données depuis [PHP](#). Chaque pilote de base de données implémenté dans l'interface PDO peut utiliser des fonctionnalités spécifiques de chacune des bases de données en utilisant des extensions de fonctions. Il faut donc un [driver](#) PDO spécifique à la base de données pour accéder au serveur de base de données, ici [PDO_MYSQL](#).

Il faut modifier le service php pour gérer l'installation du [driver PDO_MYSQL](#).

Modifier le fichier `docker-compose.yml` avec le contenu suivant :

```
version: '3'
services:
  web:
```

```

    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/nginx.conf
      - ./app:/app
  php:
    build:
      context: ./php
      dockerfile: Dockerfile
    volumes:
      - ./app:/app
  mysql:
    image: mariadb:latest
    environment:
      MYSQL_ROOT_PASSWORD: 'password'
      MYSQL_USER: 'bts'
      MYSQL_PASSWORD: 'password'
      MYSQL_DATABASE: 'bd'
    volumes:
      - mysqldata:/var/lib/mysql
    ports:
      - 3306:3306
volumes:
  mysqldata: {}

```

Instruction	Description
<code>build</code>	Configuration de construction pour créer une image de conteneur
<code>context</code>	Le répertoire pour la configuration
<code>dockerfile</code>	Le nom du fichier <code>Dockerfile</code>

Créer le fichier `./php/Dockerfile` avec le contenu suivant :

```

FROM php:fpm
RUN docker-php-ext-install pdo pdo_mysql

```

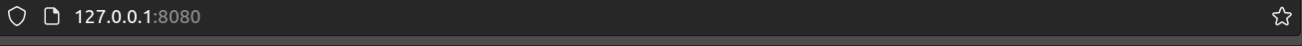
`docker-php-ext-install` est un script fourni permettant d'installer des extensions PHP. Pour l'[extension mysqli](#), il faudra : `RUN docker-php-ext-install mysqli`

Redémarrer le service web.

```

$ docker compose stop web php
$ docker compose up -d

```



Version de PHP : 8.3.2

Version de MySQL : 11.2.2-MariaDB-1:11.2.2+maria~ubu2204

Travail demandé

Installation et vérification

Question 1. Installer `docker-ce` et `docker-desktop` (facultatif).

Question 2. Déterminer les versions installées (les paquets, `docker`, `docker-compose` et Docker Engine).

Question 3. Vérifier l'état du service docker (`docker.service`).

Question 4. Vérifier le bon fonctionnement de Docker en démarrant un conteneur à partir de l'image `hello-world`.

Question 5. Lister les images et tous les conteneurs présents localement.

Question 6. Supprimer le conteneur et l'image `hello-world`.

Premier pas

Question 7. Rechercher les images `alpine` officielles. Qu'est-ce qu'"Alpine Linux" ? Quelle est sa principale particularité ? Comparer la taille de l'image "Alpine Linux" avec la taille d'une image "Ubuntu".

Question 8. Démarrer un conteneur (avec le nom `mon-alpine`) en mode interactif (un shell `sh`) à partir de la dernière version (`latest`) d' `alpine` officielle.

Question 9. Identifier la version d' `alpine` avec la commande `cat /etc/os-release`.

Question 10. Quel est le compte de la session `alpine` ?

Question 11. Visualiser l'activité (`log`) du conteneur.

Vous pouvez supprimer le conteneur et l'image.

Dockerfile

Question 12. Fournir le fichier `Dockerfile` pour créer une image `ubuntu-22.04` (basée sur Ubuntu 22.04 LTS) contenant les paquets `g++ make git`.

Question 13. Démarrer un conteneur à partir de l'image créée en mode interactif puis vérifier l'installation des paquets.

Question 14. Supprimer le conteneur et l'image.

Question 15. Réaliser la manipulation "Environnement de développement C/C++"

Docker compose

Question 16. Fournir le fichier `docker-compose.yml` pour créer un service web avec nginx (version 1.25.0) sur le port `5000` de la machine locale.

Question 17. Démarrer le serveur. Afficher les images, conteneurs et processus.

Question 18. Tester le serveur avec un client HTTP. Fournir le résultat obtenu avec `curl`. Puis arrêter le service web.

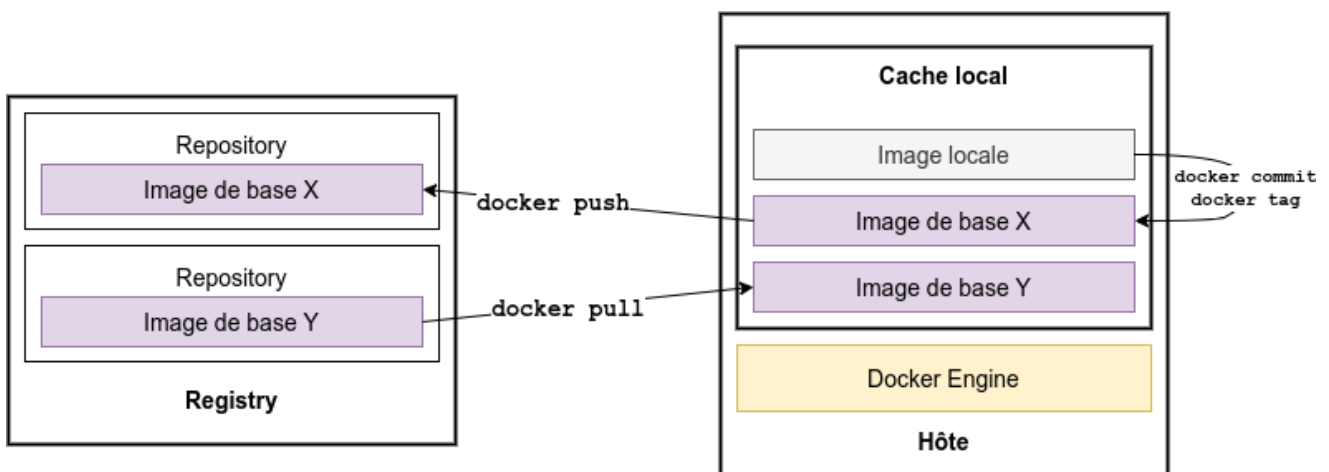
Question 19. Terminer la manipulation "Environnement de développement web PHP"

Question 20. Arrêter les services. Puis nettoyer les conteneurs, les ressources et les images.

| cf. Bonus ...

Bonus

Création et publication d'image



Modification d'un conteneur, par exemple :

```
root@54fa5656a835:/src# apt update
root@54fa5656a835:/src# apt install clang-format doxygen
root@54fa5656a835:/src# exit
```

Puis :

```
# Lister pour récupérer le CONTAINER ID
$ docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS
54fa5656a835   tvaira/ubuntu-dev   "/bin/bash"            2 minutes ago   Exited (0)
...

# Commit d'une nouvelle image
docker commit -m "ajout clang-format et doxygen" -a "tvaira" 54fa5656a835 tvai
sha256:58aa9341586381e9a788a4a8a3df4a4c9be3a03147af8c584bc6177ac866b8d7

# Lister les images
$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
tvaira/ubuntu-dev   latest      58aa93415863     2 seconds ago   691MB
...

# Se connecter à Docker Hub https://hub.docker.com/
$ docker login -u tvaira
Password: *****
WARNING! Your password will be stored unencrypted in /home/tv/.docker/config.j
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded

# Tag d'une nouvelle image
#$ docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]

# Transmet l'image à Docker Hub https://hub.docker.com/ (par défaut avec le tag
$ docker push tvaira/ubuntu-dev
```

Dans [Docker Hub](https://hub.docker.com/) (<https://hub.docker.com/>)

The screenshot shows the Docker Hub interface. At the top, there is a navigation bar with 'dockerhub', 'Explore', 'Repositories', and 'Organizations'. A search bar contains 'Search Docker Hub' and a 'ctrl+K' button. Below the navigation bar, there is a search filter for 'tvaira' and a 'Create repository' button. The main content area displays two repository entries:

Repository Name	Status	Stars	Downloads	Visibility
tvaira / ubuntu-dev Contains: Image Last pushed: 9 minutes ago	Inactive	0	0	Public
tvaira / server-ubuntu2004-gns3 Contains: Image Last pushed: 2 years ago	Inactive	0	3	Public

The screenshot shows the Docker Hub interface for the repository 'tvaira / ubuntu-dev'. At the top, there's a navigation bar with 'docker hub', 'Explore', 'Repositories', and 'Organizations'. A search bar is on the right. Below the navigation, the breadcrumb 'tvaira / Repositories / ubuntu-dev / General' is visible. The main content area has tabs for 'General', 'Tags', 'Builds', 'Collaborators', 'Webhooks', and 'Settings'. A light blue banner prompts to 'Add a short description for this repository'. Below that, the repository name 'tvaira / ubuntu-dev' is shown with a 'Public View' button. The 'Description' section states 'This repository does not have a description'. The 'Last pushed' time is '9 minutes ago'. To the right, 'Docker commands' shows 'docker push tvaira/ubuntu-dev:tagname'. Below this, there are sections for 'Tags' (listing 'latest' as the only tag) and 'Automated Builds'.

This screenshot shows the specific tag page for 'tvaira/ubuntu-dev:latest'. The header is similar to the previous screenshot. The main content features a 3D cube icon for the tag. The tag name 'tvaira/ubuntu-dev:latest' is prominent, with a 'Delete Tag' button to its right. Below the tag name, the 'DIGEST' is shown as 'sha256:cccb6a8b619e06ecf0fc3bb354c3e80ddb10df02b7fdc9a1403e86f1923f599'. A table provides details: OS/ARCH is 'linux/amd64', COMPRESSED SIZE is '256.03 MB', LAST PUSHED is '9 minutes ago by tvaira', and TYPE is 'Image'.

Le réseau

Docker prend en charge différents types de réseaux ce qui permet de faire communiquer les conteneurs entre eux ainsi qu'avec la machine locale en choisissant le niveau d'isolation.

Lien : <https://docs.docker.com/network/>

Les pilotes de réseau suivants sont disponibles :

Pilote	Description
none	Aucune interface réseau (sauf <u>loopback</u>) dans le conteneur ce qui permet une isolation complète
bridge	Le pilote de réseau par défaut qui permet aux conteneurs de communiquer sans les rendre accessibles depuis l'extérieur (sauf avec un mappage de ports)

Pilote	Description
host	Utilise la même interface réseau que la machine locale ce qui retire l'isolation du réseau entre le conteneur et la machine locale
overlay	<u>Overlay networks connect multiple Docker daemons together</u>
ipvlan	Permet un contrôle sur l'adressage IPv4 et IPv6
macvlan	Permet d'attribuer une adresse MAC à un conteneur

L'installation Docker provoque la création d'un réseau `bridge` connecté à l'interface réseau `docker0` :

```
$ ifconfig docker0
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:2aff:fea9:8515 prefixlen 64 scopeid 0x20<link>
    ether 02:42:2a:a9:85:15 txqueuelen 0 (Ethernet)
    RX packets 8700 bytes 471925 (471.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 9048 bytes 116658042 (116.6 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

$ ip addr show docker0
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:2a:a9:85:15 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::42:2aff:fea9:8515/64 scope link
        valid_lft forever preferred_lft forever

$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
c102518d86f6       bridge             bridge              local
...

$ docker network inspect c102518d86f6
[
  {
    "Name": "bridge",
    "Id": "c102518d86f6b383a503b963344d2668f4a3a3ff95396543ffcc2581228edd7",
    "Created": "2024-01-27T06:47:23.96856338+01:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
```

```
        {
            "Subnet": "172.17.0.0/16",
            "Gateway": "172.17.0.1"
        }
    ]
},
"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": {}
}
]
```

Chaque nouveau conteneur Docker est automatiquement connecté par défaut à ce réseau.

Le driver `bridge` permet aux conteneurs de communiquer entre eux. Ils ne seront accessibles depuis l'extérieur seulement avec un mappage de ports (avec l'instruction `ports` du fichier `docker-compose.yml` ou l'option `-p` de la commande `docker run`).

Docker fournit une commande spécifique pour gérer les réseaux :

```
$ docker help network
```

```
Usage: docker network COMMAND
```

```
Manage networks
```

```
Commands:
```

```
  connect    Connect a container to a network
  create     Create a network
  disconnect  Disconnect a container from a network
  inspect    Display detailed information on one or more networks
  ls         List networks
  prune      Remove all unused networks
```

```
rm          Remove one or more networks
```

Run `'docker network COMMAND --help'` for more information on a `command`.

Pour créer un nouveau réseau :

```
$ docker network create --help
```

```
Usage: docker network create [OPTIONS] NETWORK
```

Create a network

Options:

```
--attachable          Enable manual container attachment
--aux-address map     Auxiliary IPv4 or IPv6 addresses used by Network
--config-from string  The network from which to copy the configuration
--config-only         Create a configuration only network
-d, --driver string   Driver to manage the Network (default "bridge")
--gateway strings     IPv4 or IPv6 Gateway for the master subnet
--ingress             Create swarm routing-mesh network
--internal            Restrict external access to the network
--ip-range strings    Allocate container ip from a sub-range
--ipam-driver string  IP Address Management Driver (default "default")
--ipam-opt map        Set IPAM driver specific options (default map[])
--ipv6               Enable IPv6 networking
--label list          Set metadata on a network
-o, --opt map         Set driver specific options (default map[])
--scope string        Control the network's scope
--subnet strings      Subnet in CIDR format that represents a network s
```

Création d'un nouveau réseau :

```
$ docker network create --driver bridge mon-reseau --subnet=172.16.0.0/16 --ga
```

```
$ docker network ls
```

```
NETWORK ID          NAME                DRIVER              SCOPE
...
3eb96898c800       mon-reseau         bridge              local
```

```
$ docker network inspect 3eb96898c800
```

```
[
  {
    "Name": "mon-reseau",
    "Id": "3eb96898c800b0a3b5cdbb0ec73d1543532108b68b19c03b8a04f01e76eb710",
    "Created": "2024-01-30T17:56:08.284296181+01:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
```

```

    "Driver": "default",
    "Options": {},
    "Config": [
      {
        "Subnet": "172.16.0.0/16",
        "Gateway": "172.16.0.1"
      }
    ],
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]

```

Le choix du réseau peut se réaliser avec l'option `--network` de la commande `docker run` :

```

$ docker run --help
...
--network network          Connect a container to a network
...
--add-host list           Add a custom host-to-IP mapping (host
--ip string                IPv4 address (e.g., 172.30.100.104)
--ip6 string              IPv6 address (e.g., 2001:db8::33)
--mac-address string      Container MAC address (e.g., 92:d0:c6
...

```

Deux conteneurs connectés au nouveau réseau :

```

$ docker run -dit --rm --network=mon-reseau --name=mon-conteneur1 alpine
$ docker run -dit --rm --network=mon-reseau --name=mon-conteneur2 alpine

$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS
2c9d601be3e2   alpine   "/bin/sh" 7 seconds ago  Up 6 seconds
3fedbc94e405   alpine   "/bin/sh" 12 seconds ago  Up 11 seconds

```

Communication entre les deux conteneurs :

```
$ docker exec mon-conteneur1 ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
48: eth0@if49: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:10:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.2/16 brd 172.16.255.255 scope global eth0
        valid_lft forever preferred_lft forever

$ docker exec mon-conteneur2 ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
50: eth0@if51: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:10:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.3/16 brd 172.16.255.255 scope global eth0
        valid_lft forever preferred_lft forever

$ docker exec mon-conteneur1 ping -c 1 172.16.0.3
PING 172.16.0.3 (172.16.0.3): 56 data bytes
64 bytes from 172.16.0.3: seq=0 ttl=64 time=0.094 ms

--- 172.16.0.3 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.094/0.094/0.094 ms
```

Nettoyage des ressources réseaux :

```
$ docker network disconnect mon-reseau mon-conteneur1
$ docker network disconnect mon-reseau mon-conteneur2

$ docker network rm mon-reseau
```

Nettoyage des conteneurs et des images :

```
$ docker stop $(docker ps -aq)

$ docker rmi $(docker images -q)
```

Annexes

Les options

<u>CLI</u>	<u>Dockerfile</u>	
<code>\$ docker container run [OPTIONS] IMAGE [CMD] [ARG...]</code>	CMD (<i>only one</i>)	
Les options importantes :		
<code>--name</code>	Assign a name to the container	
<code>--interactive , -i</code>	Keep STDIN open even if not attached	
<code>--tty , -t</code>	Allocate a pseudo-TTY	
<code>--env , -e</code>	Set environment variables	ENV
<code>--rm</code>	Automatically remove	
<code>--user , -u</code>	Username or UID (<name uid>[:<group gid>])	USER
<code>--volume , -v</code>	Bind mount a volume	VOLUME
<code>--workdir , -w</code>	Working directory inside the container	WORKDIR
Voir aussi :		
<code># Copy files/folders between a container and the local filesystem</code>	ADD	
<code>\$ docker container cp</code>	COPY	
	Voir aussi : RUN, ENTRYPOINT, SHELL	

```
$ docker run -it --user=$(id -u $USER):$(id -g $USER)
--workdir="/home/$USER" --volume="/home/$USER:/home/$USER"
--volume="/etc/group:/etc/group:ro" --volume="/etc/passwd:/etc/passwd:ro"
--volume="/etc/shadow:/etc/shadow:ro"
--volume="/etc/sudoers.d:/etc/sudoers.d:ro" test /bin/bash
tv@b94a9d926fae:~$ id
uid=1026(tv) gid=65536(tv) groups=65536(tv)
tv@b94a9d926fae:~$ pwd
/home/tv
tv@b94a9d926fae:~$ exit
$ cat /etc/group | grep -E '^tv'
tv:x:65536:
$ cat /etc/passwd | grep -E '^tv'
tv:x:1026:65536:Vaira Thierry,,,:/home/tv:/bin/bash
```

Exemple Dockerfile

```
FROM ubuntu:20.04
LABEL maintainer="tvaira@free.fr"
ARG DEBIAN_FRONTEND=noninteractive
RUN apt--y sudo locales g++
RUN apt-get install -y libcppunit-dev
RUN apt-get install -y bash-completion vim git doxygen-gui graphviz
RUN sed -i -e 's/# en_US.UTF-8 UTF-8/fr_FR.UTF-8 UTF-8/' /etc/locale.gen && \
dpkg-reconfigure --frontend=noninteractive locales && \
update-locale LANG=fr_FR.UTF-8
ENV LANG fr_FR.UTF-8
ENV LANGUAGE fr_FR:fr
ENV LC_ALL fr_FR.UTF-8
RUN echo "Set disable_coredump false" >> /etc/sudo.conf
RUN rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

```
ENV USERNAME iris
ENV PASSWORD password
RUN export uid=1026 gid=65536 && \
useradd --uid ${uid} --create-home $USERNAME && \
echo "$USERNAME:$PASSWORD" | chpasswd && \
usermod --shell /bin/bash $USERNAME && \
usermod -aG sudo $USERNAME && \
echo "$USERNAME ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers.d/$USERNAME && \
chmod 0440 /etc/sudoers.d/$USERNAME && \
groupmod --gid ${gid} $USERNAME && \
chown ${uid}:${gid} -R /home/$USERNAME && \
mkdir -p /home/$USERNAME/Projets && chown ${uid}:${gid} -R /home/$USERNAME/Pro
USER $USERNAME
ENV HOME /home/$USERNAME
WORKDIR /home/$USERNAME/Projets
CMD /bin/bash
```

Visual Studio Code

L'[extension Dev Containers](#) de Visual Studio Code vous permet d'utiliser un conteneur Docker comme environnement de développement complet.

Liens :

- <https://code.visualstudio.com/docs/devcontainers/containers>
- <https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-containers>
- <https://code.visualstudio.com/docs/devcontainers/tutorial>

Thierry Vaira : [thierry\(dot\)vaira\(at\)gmail\(dot\)com](mailto:thierry(dot)vaira(at)gmail(dot)com)