



TP Docker - part2

BUT2 – IUT Limoges 2025

R4.A.08 : Virtualisation

Develop faster. Run anywhere.

The most-loved Tool in Stack Overflow's 2022 Developer Survey.

Documentation officielle : <https://docs.docker.com/>

Docker est une plateforme permettant de lancer certaines applications dans des conteneurs logiciels.

Selon la firme de recherche sur l'industrie 451 Research, « Docker est un outil qui peut emballer une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur ». Il ne s'agit pas de virtualisation, mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement. Cette approche permet d'accroître la flexibilité et la portabilité d'exécution d'une application, laquelle va pouvoir tourner de façon fiable et prévisible sur une grande variété de machines hôtes, que ce soit sur la machine locale, un cloud privé ou public, une machine nue, etc.

Techniquement, Docker étend le format de conteneur Linux standard, LXC, avec une API de haut niveau fournissant une solution pratique de virtualisation qui exécute les processus de façon isolée. Pour ce faire, Docker utilise entre autres LXC, cgroups et le noyau Linux lui-même. Contrairement aux machines virtuelles traditionnelles, un conteneur Docker n'inclut pas de système d'exploitation, mais s'appuie au contraire sur les fonctionnalités du système d'exploitation fournies par la machine hôte.

La technologie de conteneur de Docker peut être utilisée pour étendre des systèmes distribués de façon à ce qu'ils s'exécutent de manière autonome depuis une seule machine physique ou une seule instance par nœud. Cela permet aux nœuds d'être déployés au fur et à mesure que les ressources sont disponibles, offrant un déploiement transparent et similaire aux PaaS pour des systèmes comme Apache Cassandra, Riak, ou d'autres systèmes distribués.

Table des matières

A. Les volumes et les bind mounts.....	3
1. Volumes vs bind mounts.....	4
2. Liste des commandes concernant les volumes.....	5
Exemple de Portainer.....	5
La création du volume.....	5
La création du conteneur Portainer.....	6
Persistance des données pour une instance de Dokuwiki.....	6
B. Le réseau dans Docker.....	8
Le driver bridge.....	8
Le driver host.....	9
Le driver overlay.....	10
Le driver IPvlan.....	10
Le driver Macvlan.....	11
Le driver none :.....	11
C. Applications multi-conteneurs.....	14
1. Introduction.....	14
2. Docker compose pour l'instance Dokuwiki.....	17
3. Getting-started.....	18
4. Application multi-conteneurs.....	18
4.1. stack wordpress.....	18
4.2. stack crud.....	19
D. ANNEXES.....	20

A. Les volumes et les bind mounts

Les données générées à l'intérieur des conteneurs docker sont éphémères, elles sont perdues si le conteneur est supprimé.

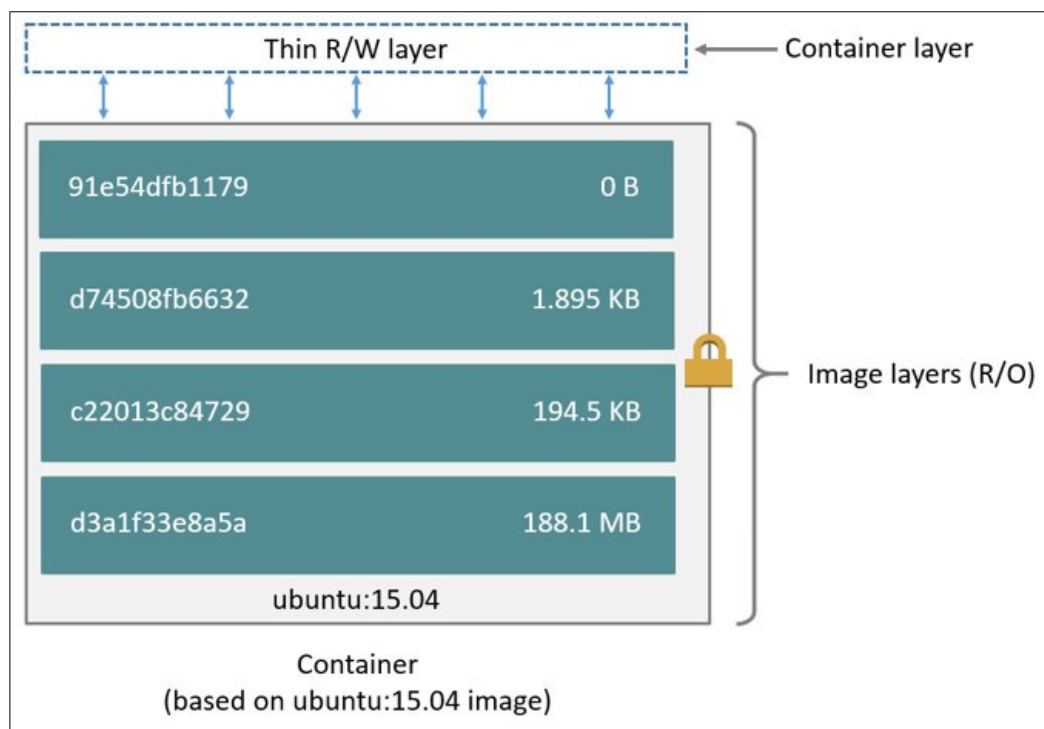
Comme vu dans la première partie, une image Docker est composée d'un ensemble de couches en lecture seule.

Lorsque qu'un conteneur est instancié à partir d'une image, une « couche conteneur » est ajoutée en haut de cette pile en lecture/écriture (RW). C'est dans cette couche que sont effectuées les modifications éventuelles (e.g. lorsque des modifications sont faites par des commandes après avoir ouvert un shell dans un conteneur)

Si un fichier est modifié dans un conteneur, Docker crée une copie du fichier de la couche en lecture seule (RO) vers la couche de conteneur en RW.

Si un fichier est créé dans un conteneur, Docker crée le fichier uniquement sur la couche de conteneur (RW).

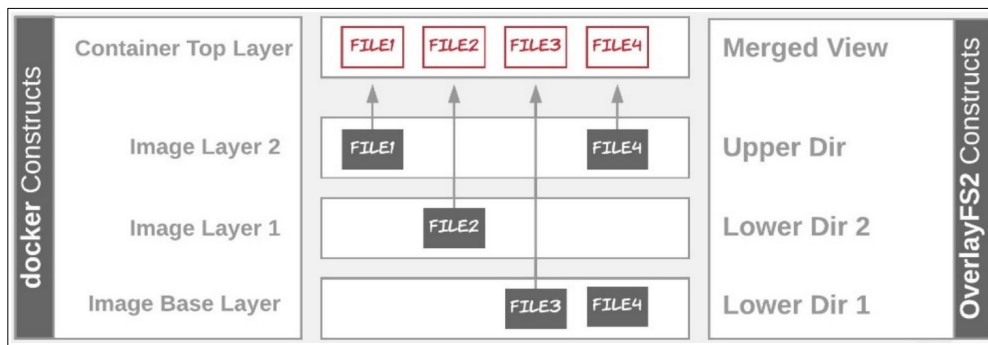
Si un fichier est supprimé dans un conteneur, supprime le fichier uniquement sur la couche de conteneur (RW). Le fichier éventuellement présent sur les couches en lecture seule ne sont pas affectés.



<https://docs.docker.com/storage/storagedriver/>

Cette « union » de couches est appelée "Union File System".

Ainsi, un nouveau conteneur instancié à partir de la même image qu'un conteneur modifié sera quant à lui non modifié car tous les changements ont eu lieu sur la couche de conteneur en RW.



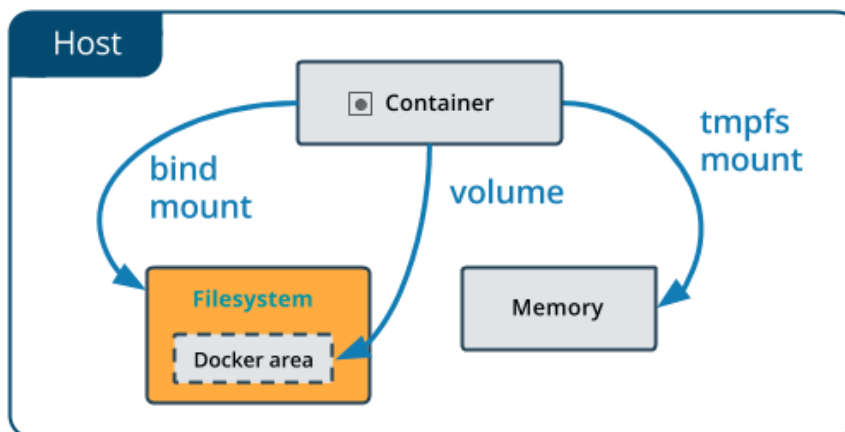
Exemple UFS : <https://ibm.github.io/docker101/lab-3/>

1. Volumes vs bind mounts

Une des conséquences du fonctionnement en couches en RO, est que lorsqu'un conteneur est supprimé, toutes les modifications apportées lors de l'exécution de celui-ci (génération de logs, modification d'une base de données, etc.) sont perdus.

Pour permettre aux données générées par une application conteneurisée par Docker, il existe plusieurs solutions plus ou moins équivalentes :

- Les **Volumes**
- Les **Bind mounts** (mapping de point de montage)



<https://docs.docker.com/storage/bind-mounts/>

Le principe et le fonctionnement dans les deux cas est le même à la différence que les volumes sont gérés par Docker contrairement aux points de montage.

Il s'agit de fournir un espace de stockage pour le conteneur en dehors du système de fichier unifié du conteneur (c'est à dire sur l'hôte). Cela présente certains avantages :

- Stockage cohérent des données,
- Pas d'augmentation de la taille du conteneur,
- Les données ne dépendent pas du cycle de vie du conteneur.

Un bind mount consiste à mapper une partie du système de fichier de la machine hôte avec un répertoire du conteneur.

	Hôte	Conteneur
Ex : mapper	/home/htdocs/app	→ /var/www/html

Un **volume** est un répertoire géré par Docker qui permet de faire la même chose. La suite s'attarde sur les commandes de manipulation des volumes.

2. Liste des commandes concernant les volumes

```
std@kathara:~$ docker volume --help

Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes

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

Exemple de Portainer.

Lors de l'installation de Portainer, vous avez réalisé 2 étapes :

1. La création d'un volume
2. L'instanciation de la dernière image officielle de Portainer

La création du volume

```
std@kathara:~$ docker volume create portainer_data
portainer_data
```

Cela crée un volume (un répertoire géré par Docker) qui sera partagé entre l'hôte et le ou les conteneur(s) qui vont utiliser ce volume.

Pour visualiser les volumes Docker, on utilise la commande suivante :

```
std@kathara:~$ docker volume ls
DRIVER      VOLUME NAME
local       portainer_data
```

On peut obtenir des informations sur un volume avec la commande inspect.

La commande `prune` permet de supprimer tous les volumes locaux non utilisés et la commande `rm` permet de supprimer un volume avec son nom.

La création du conteneur Portainer

```
std@kathara:~$ 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
```

```
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  
a8c0a48581d0b091484aaf199cebdb53b6da293f1707d82ca57b059b5e3bc421
```

-p 8000:8000 -p 9443:9443 : mappe les ports 8000 et 9443 de l'hôte avec les ports 8000 et 9443 du conteneur.

-v /var/run/docker.sock:/var/run/docker.sock : Binde le socket docker du conteneur et de l'hôte pour que Portainer puisse utiliser l'API Docker et gérer les conteneurs qui tournent dans l'environnement local.

-v portainer_data:/data : Monte le volume local **portainer_data** avec le répertoire **/data** dans le conteneur.

TAF

- ✓ Inspecter le volume créé pour portainer,
- ✓ Dans quel répertoire est-il réellement situé sur la machine ?
- ✓ Essayer de le supprimer par la commande `docker volume rm`. Que se passe-t-il ?

Persistance des données pour une instance de Dokuwiki

Image Dokuwiki : <https://hub.docker.com/r/linuxserver/dokuwiki>

Dokuwiki est un moteur de wiki open-source écrit en PHP. Les données des pages du wiki sont stockées dans des fichiers textes. Il est donc important de penser à la persistance des données si on souhaite mettre en œuvre un wiki dans un conteneur Docker.

Sur la page DockerHub de LinuxServer, une image dokuwiki est disponible.

Dans les instructions, on trouve la ligne de commande pour créer un conteneur à partir de cette image :

```
docker run -d \
  --name=dokuwiki \
  -e PUID=1000 \
  -e PGID=1000 \
  -e TZ=Etc/UTC \
  -p 80:80 \
  -p 443:443 \
  -v /path/to/appdata/config:/config \
  --restart unless-stopped \
  lscr.io/linuxserver/dokuwiki:latest
```

En une ligne ça donne :

```
docker run -d --name=dokuwiki -e PUID=1000 -e PGID=1000 -e TZ=Etc/UTC -p 80:80 -p 443:443 -v /path/to/appdata/config:/config --restart unless-stopped lscr.io/linuxserver/dokuwiki:latest
```

TAF

En utilisant un **volume Docker**

- ✓ Créer un conteneur Dokuwiki (adapter les ports et le volume),
- ✓ Installer le wiki et créer une ou deux pages,
- ✓ Visualiser les données stockées dans le volume ou point de montage,
- ✓ Supprimer le conteneur puis en créer un nouveau,
- ✓ Constater le résultat.

Un volume peut être partagé entre plusieurs conteneurs. Il est possible d'utiliser le paramètre **--volume-from <CT-ID>** dans la commande docker run pour qu'un conteneur utilise les mêmes volumes que le conteneur <CT-ID>

TAF

En utilisant un **volume Docker**

- ✓ Créer un second conteneur dokuwiki avec le même volume que le premier conteneur (attention au port).
- ✓ Modifier une page dans le premier conteneur
- ✓ Rafraîchir le second wiki et constater le résultat.

B. Le réseau dans Docker

Documentation : <https://docs.docker.com/network/>
<https://www.docker.com/blog/understanding-docker-networking-drivers-use-cases/>
<https://collabnix.com/a-beginners-guide-to-docker-networking/>

Dans la partie suivante, nous verrons une introduction aux applications multi-conteneurs. Comme le nom l'indique relativement bien, il s'agit de séparer les différents services utilisés par une application dans différents conteneurs (e.g. une application web / php / mariadb).

Il va donc falloir que ces conteneurs puissent communiquer entre eux. Il en est de même avec les conteneurs seuls pour leur communication avec l'hôte et l'extérieur.

Docker prend en charge différents types de réseaux (appelés drivers) qui sont adaptés à certains cas d'utilisation et fournissent différentes fonctionnalités

Les différents drivers sont :

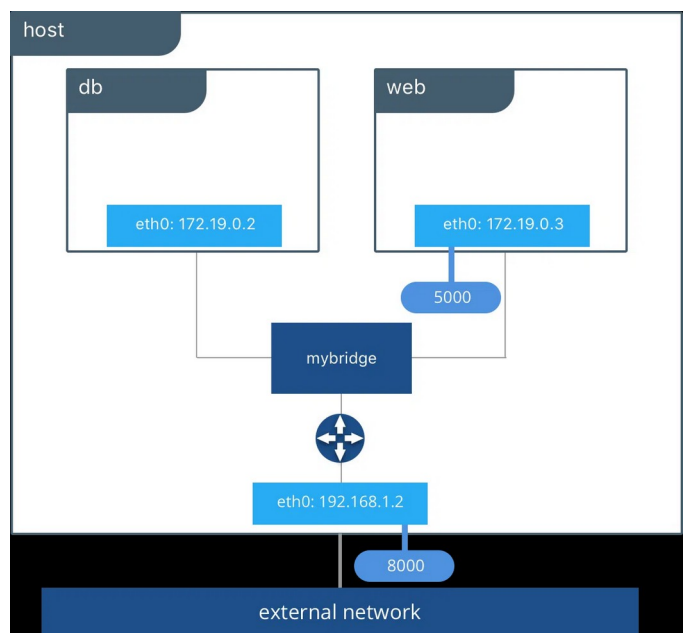
- bridge
- host
- overlay
- ipvlan
- macvlan
- none
- plugins tiers

Le driver **bridge**

Les réseaux « bridge » sont créés par l'utilisateur (il existe un réseau bridge par défaut) et sont une bonne solution lorsque vous avez besoin que plusieurs conteneurs communiquent sur le même hôte Docker.

Le réseau bridge par défaut nommé « bridge » est 172.17.0.0/16

```
"Name": "bridge",
  "Config": [
    {
      "Subnet": "172.17.0.0/16",
      "Gateway": "172.17.0.1"
    }
  ],
}
```



Créer un nouveau réseau bridge

```
docker network create -d bridge mybridge
```

ou

```
docker network create -d bridge --subnet=172.16.42.0/24 --gateway=172.16.42.1 mybridge
```



```

    "Name": "mybridge",
    "Id": "f8fc4d1d9ef5a6771db2a134b49dc0c81acfab039bd327592430517d377ed783",
    "Created": "2023-02-21T22:33:54.3716522Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.16.42.0/24",
          "Gateway": "172.16.42.1"
        }
      ]
    }
  },

```

Utiliser un réseau bridge créé

```
docker run -d --net mybridge --name web1 nginx
```

```
docker inspect web1
```

```

"Networks": {
  "mybridge": {
    "IPAMConfig": null,
    "Links": null,
    "Aliases": [
      "9dbd53a270c2"
    ],
    "NetworkID": "f8fc4d1d9ef5a6771db2a13...",
    "EndpointID": "c92f752fb7db2cffddc903...",
    "Gateway": "172.16.42.1",
    "IPAddress": "172.16.42.3",
    "IPPrefixLen": 24,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "MacAddress": "02:42:ac:10:2a:03",
    "DriverOpts": null
  }
}

```

Le driver **host**

Il est une solution lorsque la pile réseau du conteneur ne doit pas être isolée de l'hôte Docker, mais que vous souhaitez que d'autres aspects du conteneur soient isolés. (Ne fonctionne que sous Linux)

Le conteneur n'aura pas d'adresse IP et sera « confondu » avec l'hôte. Si un conteneur héberge une application qui tourne sur le port 80, alors cette application sera accessible par l'URL : <http://localhost:80> sans mappage de port.

```
"Name": "host",
```

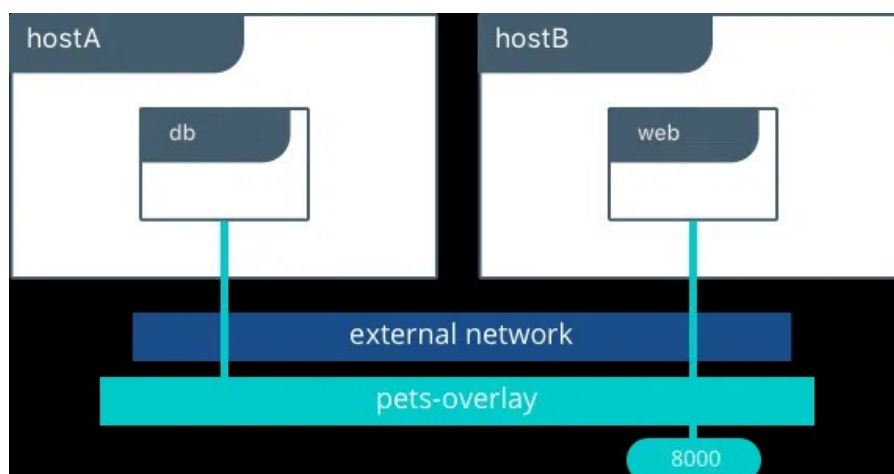
```

    "Id": "73ec10ce59915830117dcc729483b9315dab2bedaca7a5efd507d55b065e8fbd",
    "Created": "2023-02-10T21:10:16.6571971Z",
    "Scope": "local",
    "Driver": "host",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": []
    },
    "Containers": {
      "98fcdd5d46e9174d4baf8...": {
        "Name": "gracious_ramanujan",
        "EndpointID": "a75320247763a0946df688b...",
        "MacAddress": "",
        "IPv4Address": "",
        "IPv6Address": ""
      }
    }
  },

```

Le driver **overlay**

Il est la solution lorsque vous avez besoin que des conteneurs exécutés sur différents hôtes Docker puissent communiquer, ou lorsque plusieurs applications fonctionnent ensemble à l'aide de Swarm par exemple.

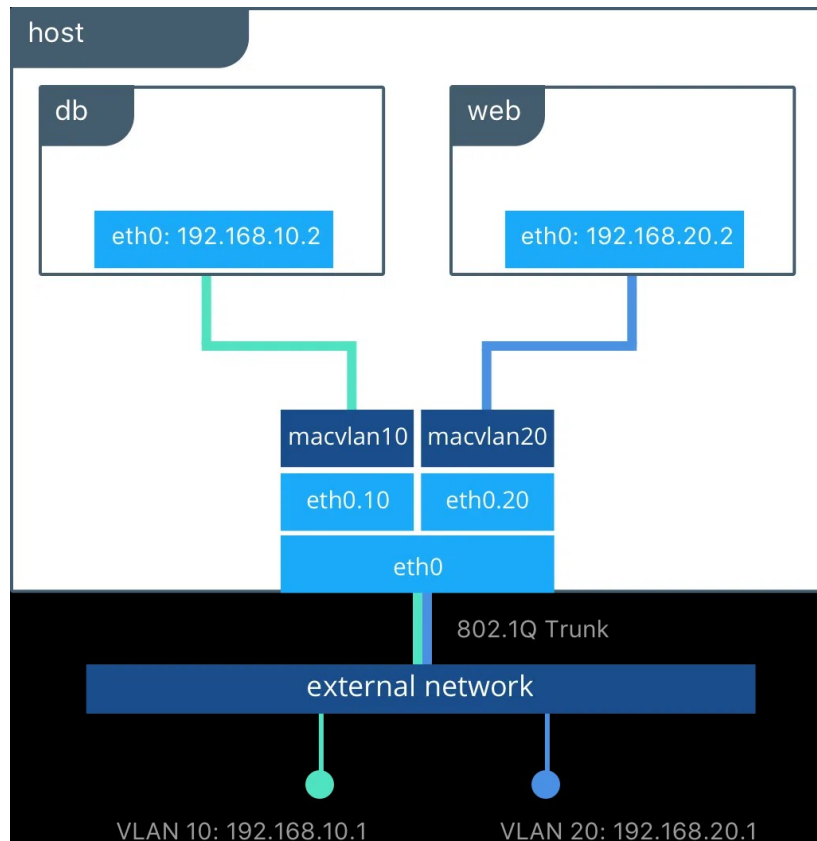


Le driver **IPvlan**

Il est généralement utilisé pour les applications qui s'exécutent sur l'hôte, mais ayant besoin d'une adresse IP distincte, différente de l'hôte (e.g. On souhaite faire tourner plusieurs applications sur le même port).

Le driver **Macvlan**

Il est la solution lorsque vous avez besoin que vos conteneurs soient vus de l'extérieur comme des hôtes physiques sur votre réseau, chacun avec une adresse MAC unique afin de les affecter à un VLAN par exemple. Macvlan = Ipvlan + MAC unique.



Le driver **none** :

Le conteneur sera totalement isolé des autres conteneurs et de l'extérieur.

– Les drivers réseau **tiers** vous permettent d'intégrer Docker à des piles réseau spécialisées.

```
Windows PowerShell
bash-5.1# ip a | grep inet
    inet 127.0.0.1/8 scope host lo
    inet 172.16.42.6/24 brd 172.16.42.255 scope global eth0
bash-5.1# ping -c2 172.16.42.7
PING 172.16.42.7 (172.16.42.7) 56(84) bytes of data.
64 bytes from 172.16.42.7: icmp_seq=1 ttl=64 time=0.055 ms
64 bytes from 172.16.42.7: icmp_seq=2 ttl=64 time=0.040 ms

--- 172.16.42.7 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.040/0.072/0.105/0.032 ms
bash-5.1#

Windows PowerShell
bash-5.1# ip a | grep inet
    inet 127.0.0.1/8 scope host lo
    inet 172.16.42.7/24 brd 172.16.42.255 scope global eth0
bash-5.1# ping -c2 172.16.42.6
PING 172.16.42.6 (172.16.42.6) 56(84) bytes of data.
64 bytes from 172.16.42.6: icmp_seq=1 ttl=64 time=0.105 ms
64 bytes from 172.16.42.6: icmp_seq=2 ttl=64 time=0.040 ms

--- 172.16.42.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.040/0.072/0.105/0.032 ms
bash-5.1#
```

Test de communication entre deux conteneurs bridgés

Inspection du bridge par défaut (bridge)

```
std@kathara:~$ docker network inspect bridge
```

```
[
  {
    "Name": "bridge",
    "Id": "3377faa6a5ee978e17f3e5fe2406e29afa2b3dd67bc99334f948aa484c4ea3bf",
    "Created": "2023-02-26T17:09:06.9940609Z",
    "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": ""
    },
    "Containers": {
      "1bebf303491c80fb19e4a9dca2118b04e7dba84786be1e09bda8da507399f948": {
        "Name": "portainer",
        "EndpointID": "1c43df0b8d59cf44db30...",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
      },
      "2dfa69f7b93fe53fdfce2d97a84a25939973df5d3c4c24250855b2484240a9ec": {
        "Name": "php-app1",
        "EndpointID": "fedde419a307b140a306...",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
      }
    }
  }
]
```

```

    },
    "a6aee30f2aa53a82aa988542f7b7475817095e37aeb96d8d95c2b58b5f72cc7d": {
      "Name": "doku_mount2",
      "EndpointID": "29b0a140183dcd341b20...",
      "MacAddress": "02:42:ac:11:00:05",
      "IPv4Address": "172.17.0.5/16",
      "IPv6Address": ""
    },
    "d615e8ce3bacf8b6628e8c686915d1834104e3fe70e0412577b3106ce28ef5b6": {
      "Name": "doku_mount",
      "EndpointID": "5bfcd9ad6f5bb914ee3e...",
      "MacAddress": "02:42:ac:11:00:04",
      "IPv4Address": "172.17.0.4/16",
      "IPv6Address": ""
    }
  },
  "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": {}
}
]

```

Containers in network			
Container Name	IPv4 Address	IPv6 Address	MacAddress
doku_mount2	172.17.0.5/16	-	02:42:ac:11:00:05
doku_mount	172.17.0.4/16	-	02:42:ac:11:00:04
php-app1	172.17.0.3/16	-	02:42:ac:11:00:03
portainer	172.17.0.2/16	-	02:42:ac:11:00:02

Visualisation dans Portainer

C. Applications multi-conteneurs

1. Introduction

Documentation : <https://docs.docker.com/compose/>

Il n'est plus nécessaire d'installer docker-compose, il est intégré à Docker désormais. Voir docker-compose VS docker compose : <https://docs.docker.com/compose/migrate/>

[OBSOLÈTE] Installation (Debian 12)

```
curl -s https://api.github.com/repos/docker/compose/releases/latest | grep  
browser_download_url | grep docker-compose-linux-x86_64 | cut -d '"' -f 4 |  
wget -qi -
```

```
chmod +x docker-compose-Linux-x86_64  
mv docker-compose-Linux-x86_64 /usr/bin/docker-compose
```

```
docker-compose --version
```

```
Docker Compose version v2.16.0
```

Docker Compose est un outil qui permet de définir une infrastructure d'application multi-conteneurs. Dans un fichier au format YAML, il est possible de définir un ensemble de conteneurs et de dépendances qui permettra à tous les composants d'une application d'interagir entre eux. L'ensemble des conteneurs sera donc défini dans un seul fichier YAML (avec ou non des fichiers Dockerfile pour chaque conteneur) et de démarrer la pile (**stack**) de conteneurs en une commande.

Exemple simple : une application web va avoir besoin d'un service http (apache, nginx, caddy, etc.) d'un interpréteur php et d'un serveur de base de données.

Un fichier docker-compose.**yaml** est composé d'un ensemble structuré de clés/valeurs définissant chaque conteneur et ses paramètres. (!\ l'indentation est importante)

Exemple (issu du gettingstarted de Docker.com)

```
version: "3.1"
services:
  web:
    build: .
    ports:
      - "8000:5000"
    volumes:
      - ./code
    environment:
      FLASK_DEBUG: "true"

  redis:
    image: "redis:alpine"
```

version : version pour le format du fichier compose (voir <https://docs.docker.com/compose/compose-file/compose-versioning/>)

services : définit la liste des services (conteneurs qui seront créés)
Ici, un service web et un service redis.

web : nom du premier service (!= nom du conteneur)

build : . : Définit le contexte à partir duquel seront exécuter les commandes Docker, notamment pour construire les images avec des Dockerfile.

context : . ou ./web, ... en fonction de l'architecture des répertoires du projet.

ports : le mapping de port entre l'hôte et le conteneur

volume : pour les volumes/points de montage utilisés par le conteneurisation

environnement : définition de variables d'environnement pour le conteneur

redis : nom du second service

image : image pour créer le conteneur

Il existe de très nombreuses autres clés pour écrire un fichier compose : container_name, restart, depends_on, networks, ...

Se référer à la documentation : <https://docs.docker.com/compose/compose-file/>

Lancer l'application à partir du docker-compose.yml.

```
std@kathara:~$ docker compose up -d
```

```
fls@cygnus:~/Bureau/docker/compose$ sudo docker-compose up -d
[+] Running 3/7
.: redis Pulling
  #: 63b65145d645 Pull complete
  #: 6a83e1b979d3 Pull complete
  #: 33568fda55fd Pull complete
  *: 92c907937b14 Downloading [=====] 7.553MB/8.669MB
  *: ae96d2ab3885 Download complete
  *: fbd5435e8d0e Download complete
```

...

```
=> [5/6] RUN pip install -r requirements.txt 6.9s
=> [6/6] COPY . . 0.0s
=> exporting to image 1.6s
=> => exporting layers 1.6s
=> => writing image
sha256:5c6fd555969b127abc6f13d4e26784b8f0d6d5c94532cdd3f2f135ce4295e29d 0.0s
=> => naming to docker.io/library/compose-web 0.0s
[+] Running 3/3
  :: Network compose_default Created 0.1s
  :: Container compose-redis-1 Started 0.9s
  :: Container compose-web-1 Started
```

```
fls@cygnus:~/Bureau/docker/compose$ docker ps
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
3eff58d10d25 compose-web "flask run" 46 seconds ago Up 45 seconds
0.0.0.0:8020->5000/tcp, :::8020->5000/tcp compose-web-1
5a01ba2c80b3 redis:alpine "docker-entrypoint.s..." 46 seconds ago Up 45 seconds
6379/tcp compose-redis-1
```

```
std@kathara:~$ docker compose --help
```

Usage: docker compose [OPTIONS] COMMAND

Define and run multi-container applications with Docker.

Options:

--ansi string Control when to print ANSI control characters ("never"|"always"|"auto") (default "auto")

--compatibility Run compose in backward compatibility mode

--dry-run Execute command in dry run mode

--env-file stringArray Specify an alternate environment file.

-f, --file stringArray Compose configuration files

--parallel int Control max parallelism, -1 for unlimited (default -1)

--profile stringArray Specify a profile to enable

--progress string Set type of progress output (auto, tty, plain, quiet) (default "auto")

--project-directory string Specify an alternate working directory (default: the path of the, first specified, Compose file)

-p, --project-name string Project name

Commands:

attach Attach local standard input, output, and error streams to a service's running container.

build Build or rebuild services

config Parse, resolve and render compose file in canonical format

cp Copy files/folders between a service container and the local filesystem

create Creates containers for a service.

down Stop and remove containers, networks

events Receive real time events from containers.

exec Execute a command in a running container.


```
images    List images used by the created containers
kill      Force stop service containers.
logs      View output from containers
ls        List running compose projects
pause     Pause services
port      Print the public port for a port binding.
ps        List containers
pull      Pull service images
push      Push service images
restart   Restart service containers
rm        Removes stopped service containers
run       Run a one-off command on a service.
scale     Scale services
start     Start services
stats     Display a live stream of container(s) resource usage statistics
stop      Stop services
top       Display the running processes
unpause   Unpause services
up        Create and start containers
version   Show the Docker Compose version information
wait      Block until the first service container stops
watch     Watch build context for service and rebuild/refresh containers when files are updated
```

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

2. Docker compose pour l'instance Dokuwiki

Sur la page DockerHub de l'image Dokuwiki testée précédemment, une alternative (recommandée) à la commande docker run est donnée. Il s'agit d'un fichier docker-compose pour lancer l'application. Il s'agit ici d'un docker-compose pour un seul conteneur.

Lien : <https://hub.docker.com/r/linuxserver/dokuwiki>

```
---
version: "2.1"
services:
  dokuwiki:
    image: lscr.io/linuxserver/dokuwiki:latest
    container_name: dokuwiki
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Etc/UTC
    volumes:
      - /path/to/appdata/config:/config
    ports:
      - 80:80
      - 443:443
    restart: unless-stopped
```

Il s'agit de l'équivalent de la commande

```
docker run -d --name=dokuwiki -e PUID=1000 -e PGID=1000 -e TZ=Etc/UTC -p 80:80 -p 443:443 -v /path/to/appdata/config:/config --restart unless-stopped lscr.io/linuxserver/dokuwiki:latest
```

TAF

- ✓ Créer un nouveau répertoire
- ✓ Créer le fichier docker-compose.yml ci-dessus.
- ✓ Construire et lancer l'application dokuwiki à partir du docker-compose

3. Getting-started

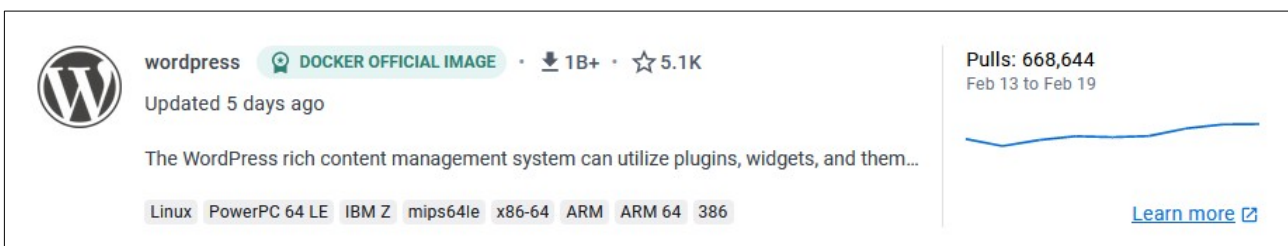
TAF

- ✓ Réaliser le tutoriel officiel de docker-composer sur Docker.com
<https://docs.docker.com/compose/gettingstarted/>
Il s'agit d'une petite application web python avec un base de données redis.

4. Application multi-conteneurs

4.1. stack wordpress

En première page de DockerHub, on trouve une image de wordpress



wordpress DOCKER OFFICIAL IMAGE · 1B+ · 5.1K

Updated 5 days ago

The WordPress rich content management system can utilize plugins, widgets, and them...

Linux PowerPC 64 LE IBM Z mips64le x86-64 ARM ARM 64 386

Pulls: 668,644
Feb 13 to Feb 19

[Learn more](#)

TAF

- ✓ Lire la documentation de l'image et lancer/tester une stack wordpress + mysql.

TAF

- ✓ Rajouter un service phpmyadmin à la stack wordpress pour gérer la base de données de wordpress.

4.2. stack crud

TAF

- ✓ Créer une stack pour déployer l'application CRUD vue en cours.
- ✓ Ajouter un service phpmyadmin.

```
std@docker:~/crud$ tree
```

```
├── db
│   ├── database.sql
│   └── Dockerfile
├── docker-compose.yml
└── web
    ├── config.php
    ├── create.php
    ├── delete.php
    ├── Dockerfile
    ├── error.php
    ├── index.php
    ├── read.php
    └── update.php
```

```
3 directories, 11 files
```

Dashboard

localhost:8888/index.php

Liste des étudiants

+ Ajouter

#	Nom	Ecole	Age	Action
2	mlx	Palma de Majorque	42	

D. ANNEXES

Une simple application apache/PHP (Celle utilisée dans le TP Proxmox) avec stockage des données de la base dans un volume

Le Dockerfile

```
FROM debian:stable-slim
ARG DOCUMENTROOT="/var/www/html"
RUN apt-get update -y && \
    apt-get install -y apache2
RUN apt-get install -y mariadb-server
ADD db/app-db.sql /
RUN service mariadb start && mariadb < /app-db.sql
RUN apt-get install -y \
    php-mysql \
    php && \
    rm -f ${DOCUMENTROOT}/index.html && \
    apt-get autoclean -y
ADD app ${DOCUMENTROOT}
EXPOSE 80
WORKDIR ${DOCUMENTROOT}
ENTRYPOINT apache2ctl -D FOREGROUND
```

Le build

```
docker build -t php-app:1.0 . --no-cache
[+] Building 37.7s (12/12) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/debian:stable-slim
=> CACHED [1/7] FROM docker.io/library/debian:stable-slim@sha256:f711bda490b4e5803ee7f634483c4e6fa7dae54102654f2c231ca58eb233a2f1
=> [internal] load build context
=> => transferring context: 181B
=> [2/7] RUN apt-get update -y && apt-get install -y apache2
=> [3/7] RUN apt-get install -y mariadb-server
=> [4/7] ADD db/app-db.sql /
=> [5/7] RUN apt-get install -y php-mysql php && rm -f /var/www/html/index.html && apt-get autoclean -y
=> [6/7] ADD app /var/www/html
=> [7/7] WORKDIR /var/www/html
=> exporting to image
=> => exporting layers
=> => writing image sha256:acb31adef14f96c9680849f914e062d79ff6ef4ecc8baec4ae67bf202ea3f89b7
=> => naming to docker.io/library/php-app:1.0
```

```
docker run -d -p 8787:80 --name php-app1 php-app:1.0
```

```
docker volume create php-app-db
```

```
php-app-db
```

```
docker run -d -p 8787:80 -v php-app-db:/var/lib/mysql --name php-app1 php-app:1.0
```

```
d64dabb81179192e076bbf5f9cd34b2d1e0624f3e7abae3427b3a399795e6759
```