



R4.A.08 : Virtualisation

BUT 2 – Semestre 4

Volume horaire

- Cours : 4.5h
- TD : 6h
- TP : 8h

contact : sautour.iut@gmail.com

Descriptif détaillé

Objectif

L'objectif de cette ressource est de comprendre les principes et enjeux de la virtualisation en informatique et d'être capable de déployer une solution de virtualisation. Cette ressource permettra de découvrir les techniques et outils utilisées pour la virtualisation de systèmes, amenant au déploiement de plateformes facilitant l'intégration et l'administration de services.

Savoirs de référence étudiés

- Types de virtualisation (serveur, application, réseau...)
- Outils de la virtualisation (hypervision, conteneurs...)
- Architectures virtualisées
- Les différents savoirs de référence pourront être approfondis

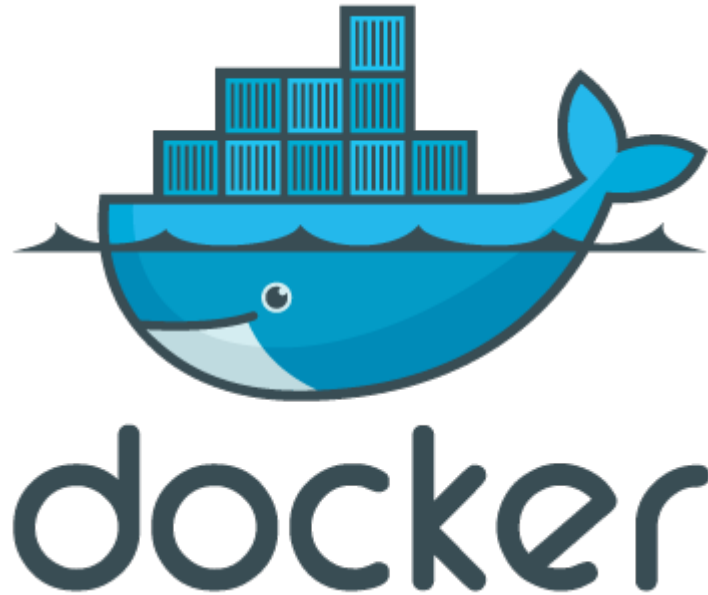
Indications de mise en œuvre

Cette ressource est largement identique à la ressource R4.B.08 et peut être mutualisée en partie, mais avec des horaires différents.

Virtualisation

conteneurs

hyperviseur



Docker

Part 1.

Docker

Docker est la star incontestée (pour l'instant) des outils utilisés pour la conteneurisation d'applications.

Il permet (entre autre) :

- de s'affranchir des problèmes de compatibilité entre environnement de développement et de production (« Mais puisque je te dis que chez moi ça marche ! »)
- de simplifier l'intégration, la livraison et le déploiement continus
- faciliter l'approche DevOps

Philosophie Docker :

- Un conteneur est immuable (enfin devrait être)
- Un conteneur = une application (un service [micro])

Docker - Terminologie

image : un modèle immuable constitué d'un système de fichiers par « couches » et de métadonnées au format JSON qui peut être utilisé pour créer des conteneurs Docker. Elle contient le code de l'application, l'environnement d'exécution de l'application, les bibliothèques nécessaires à l'application, les fichiers de configuration, etc.

conteneur : une instance d'une image en cours d'exécution.

volume : espace de stockage pour un ou plusieurs conteneurs.

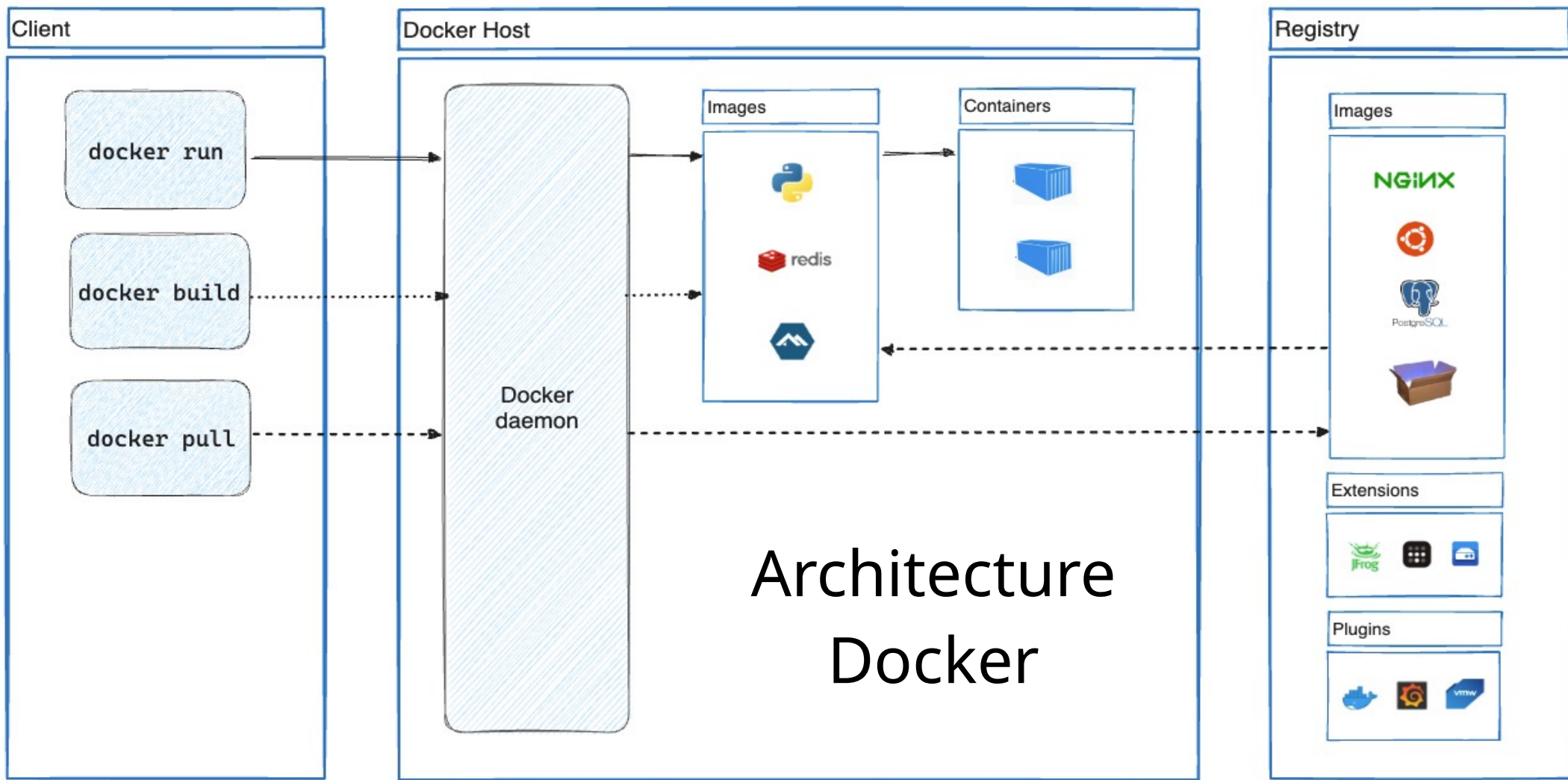
Docker - Terminologie

registre : (En anglais registry) Serveur de stockages d'images Docker versionnées. Le plus connu étant DockerHub

compose : outils pour définir et créer des applications multi-container.

dockerfile : Fichier texte. C'est un modèle de création d'une image à partir d'une image de base et d'instructions pour ajouter des couches à cette image.

Docker Hub : Le registre officiel d'images proposé par Docker Inc.



Docker CLI Reference

<https://docs.docker.com/reference/>

Docker CLI - infos

docker --version : affiche la version du moteur Docker

```
fls@docker:~$ docker --version  
Docker version 24.0.7, build afdd53b
```

docker compose version : affiche la version de docker compose

```
fls@docker:~$ docker compose version  
Docker Compose version v2.21.0
```

docker info : affiche toutes les informations système concernant l'installation de Docker

Docker CLI - help

docker --help
ou **docker :**
affiche la liste
des commandes
Docker
et des options
globales

```
Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run          Create and run a new container from an image
  [...]

Management Commands:
  compose*     Docker Compose (Docker Inc., v2.21.0)
  [...]

Swarm Commands:
  swarm        Manage Swarm

Commands:
  attach       Attach local standard input, output, and error streams to a running container
  [...]

Global Options:
  [...]
  -v, --version      Print version information and quit

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

Docker CLI - help

docker <cmd> --help :

affiche la liste des
sous-commandes
et des options
d'une commande

```
fls@docker:~$ docker exec --help
```

```
Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

```
Execute a command in a running container
```

```
Aliases:
```

```
docker container exec, docker exec
```

```
Options:
```

-d, --detach	Detached mode: run command in the background
--detach-keys string	Override the key sequence for detaching a container
-e, --env list	Set environment variables
--env-file list	Read in a file of environment variables
-i, --interactive	Keep STDIN open even if not attached
--privileged	Give extended privileges to the command
-t, --tty	Allocate a pseudo-TTY
-u, --user string	Username or UID (format: "<name uid>[:<group gid>]")
-w, --workdir string	Working directory inside the container

Docker CLI - images

docker images : affiche la liste des images locales

```
fls@docker:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
crud	1.0	89b7f3bb7cb7	15 hours ago	555MB
webapp	1.1	a3d5938e1ca0	18 hours ago	294MB
webapp	1.0	19d52d27573d	18 hours ago	294MB
nextcloud	latest	d67e7d5f3fb7	3 days ago	1.2GB
registry	2	ff1857193a0b	2 weeks ago	25.4MB
mariadb	10.6	3b3ad3b80a5c	3 weeks ago	395MB
portainer/portainer-ce	latest	d7f7a88e1acc	6 weeks ago	294MB

docker search : recherche une image sur Docker Hub.

```
fls@docker:~$ docker search debian --filter is-official=true
```

NAME	DESCRIPTION	STARS	OFFICIAL
ubuntu	Ubuntu is a Debian-based Linux operating sys...	16547	[OK]
debian	Debian is a Linux distribution that's compos...	4827	[OK]
neurodebian	NeuroDebian provides neuroscience research s...	105	[OK]

Docker CLI – image vs images

```
fls@docker:~$ docker image --help
```

```
Usage:  docker image COMMAND
```

```
Manage images
```

```
Commands:
```

build	Build an image from a Dockerfile
history	Show the history of an image
import	Import the contents from a tarball to create a filesystem image
inspect	Display detailed information on one or more images
load	Load an image from a tar archive or STDIN
ls	List images
prune	Remove unused images
pull	Download an image from a registry
push	Upload an image to a registry
rm	Remove one or more images
save	Save one or more images to a tar archive (streamed to STDOUT by default)
tag	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Docker CLI - image

docker pull debian : télécharge l'image debian:**latest** depuis un registre (e.g. Dockerhub)

Par défaut si on ne spécifie pas de tag dans le nom d'une image, le tag **latest** est choisi

docker pull debian:11.8 : télécharge l'image debian:11.8

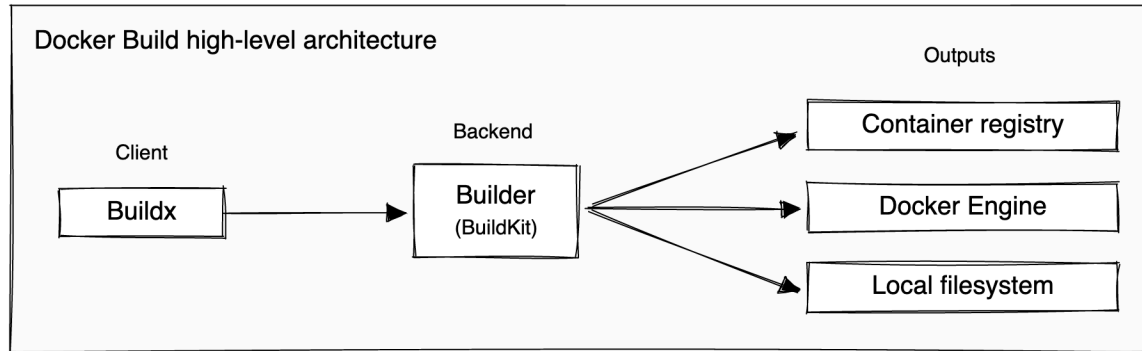
Combo with bash

docker rmi -f \$(docker images -q) : supprime toutes les images d'un coup

Docker CLI - Docker Build architecture

Docker Build implements a **client-server** architecture, where:

- **Buildx** is the client and the user interface for running and managing builds
- **BuildKit** is the server, or builder, that handles the build execution.



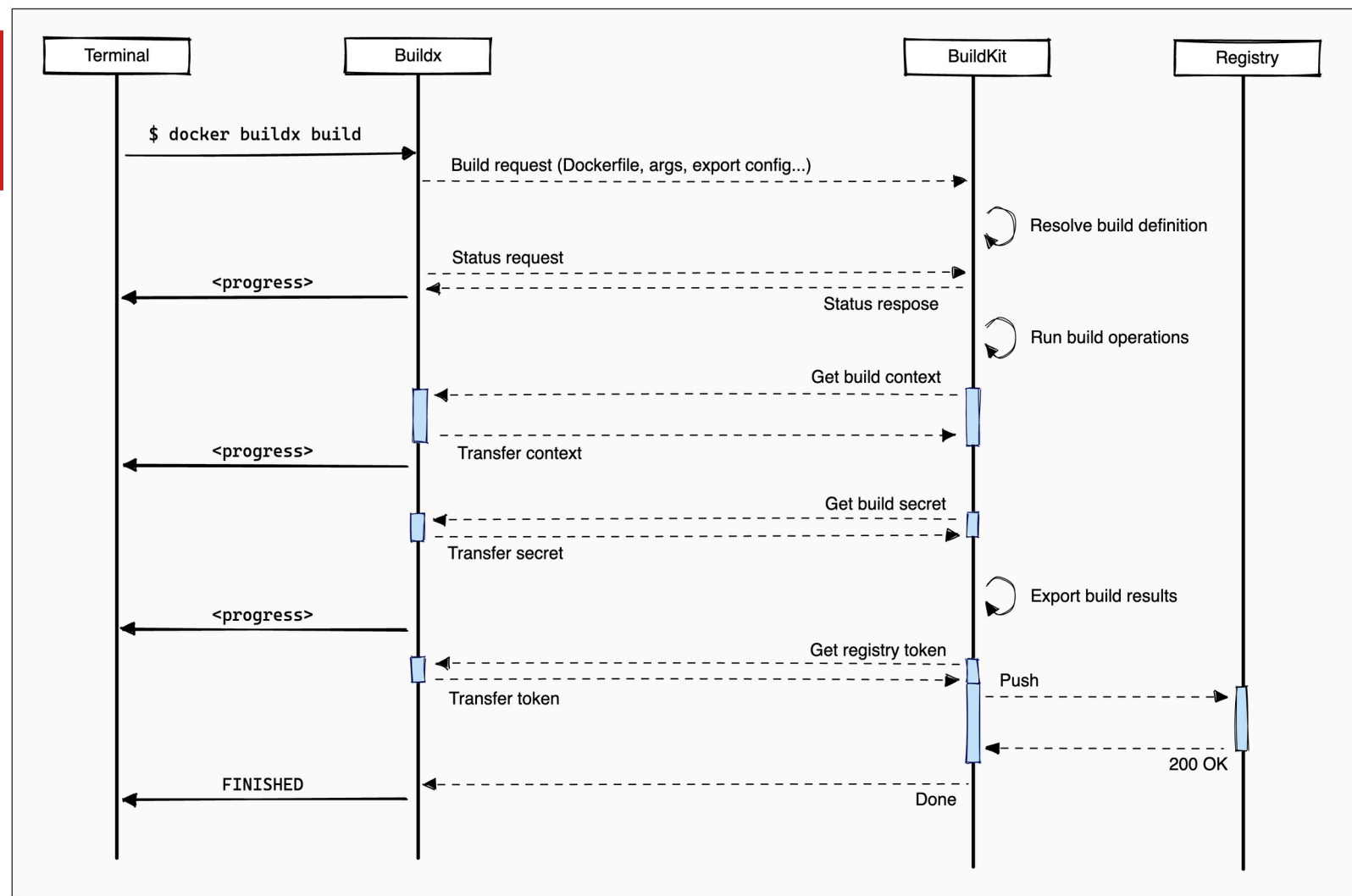
BuildKit, or buildkitd, is the daemon process that executes the build workloads.

Docker CLI - Docker Build architecture

A build execution starts with the invocation of a docker build command. Buildx interprets your build command and sends a build request to the BuildKit backend. The build request includes:

- The Dockerfile
- Build arguments
- Export options
- Caching options

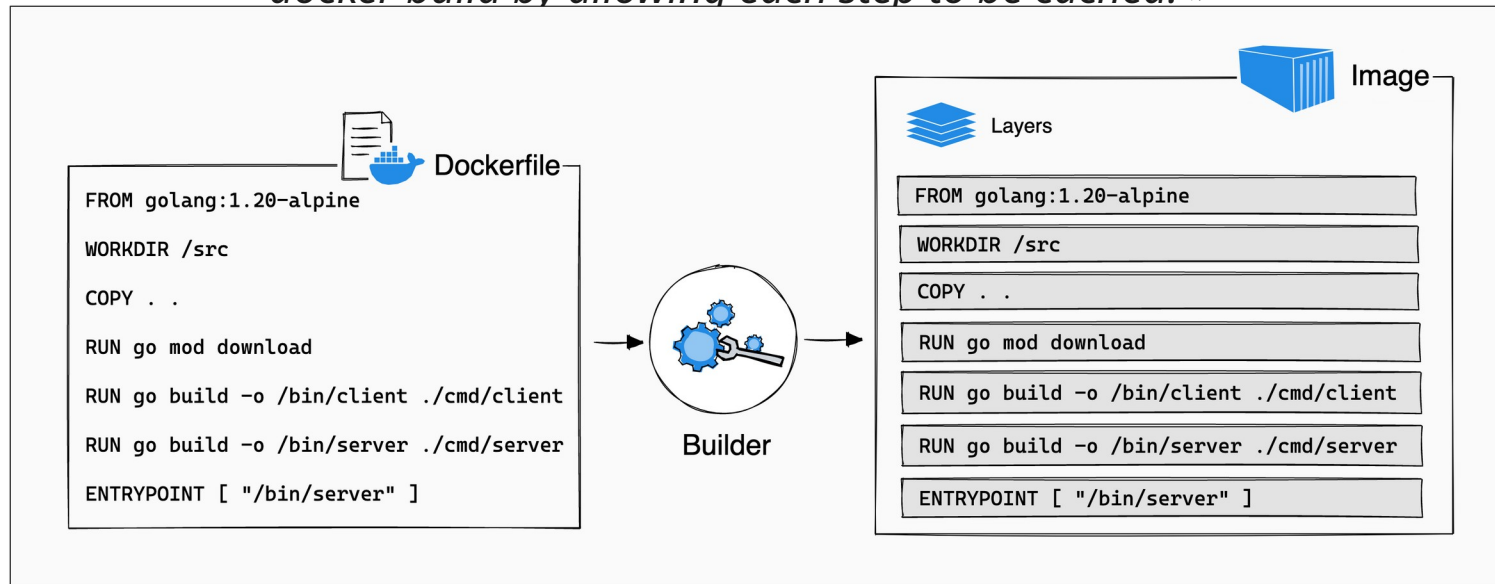
Docker BUILD



Docker CLI – image layers

Une image est composée de couches (FS) successives

« Docker images have intermediate layers that increase reusability, decrease disk usage, and speed up docker build by allowing each step to be cached. »



Docker CLI – image layers

Une image est composée de couches (FS) successives

```
f1s@cygnus:~$ docker pull python
Using default tag: latest
latest: Pulling from library/python
8457fd5474e7: Already exists
13baa2029dde: Downloading 23.66MB/24.05MB
325c5bf4c2f2: Download complete
7e18a660069f: Downloading 42.38MB/211.1MB
98a59f0ffede: Download complete
72c7f17f2221: Downloading 9.898MB/22.51MB
2f40b346325a: Waiting
f3f08e04e337: Waiting
```

Docker CLI – image layers

Gestion du cache, ordre des instructions Dockerfile

Lors d'un build, le builder tente de réutiliser les couches des précédents builds
=> **Cache**

Si une couche a été **modifiée** depuis le précédent build, le builder reconstruit la couche **ainsi que toutes les couches sous-jacentes**.

L'ordre des instructions dans un Dockerfile a donc une grande importance.

Docker CLI – image layers

Si on modifie le code copié dans l'exemple précédent.



Layers

Cache?

```
FROM golang:1.20-alpine
```



```
WORKDIR /src
```



```
COPY . .
```



```
RUN go mod download
```



```
RUN go build -o /bin/client ./cmd/client
```



```
RUN go build -o /bin/server ./cmd/server
```

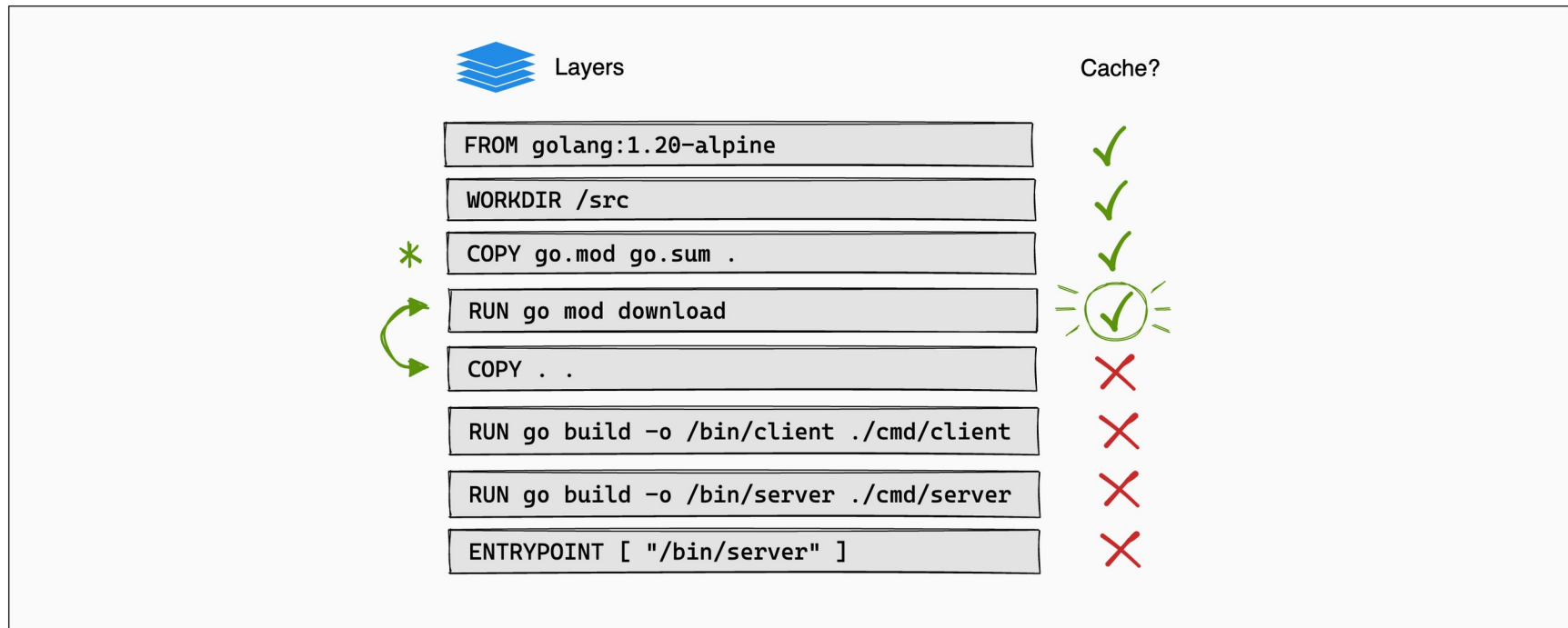


```
ENTRYPOINT [ "/bin/server" ]
```



Docker CLI – image layers

Pour ne pas avoir à re-télécharger les modules go à chaque build.



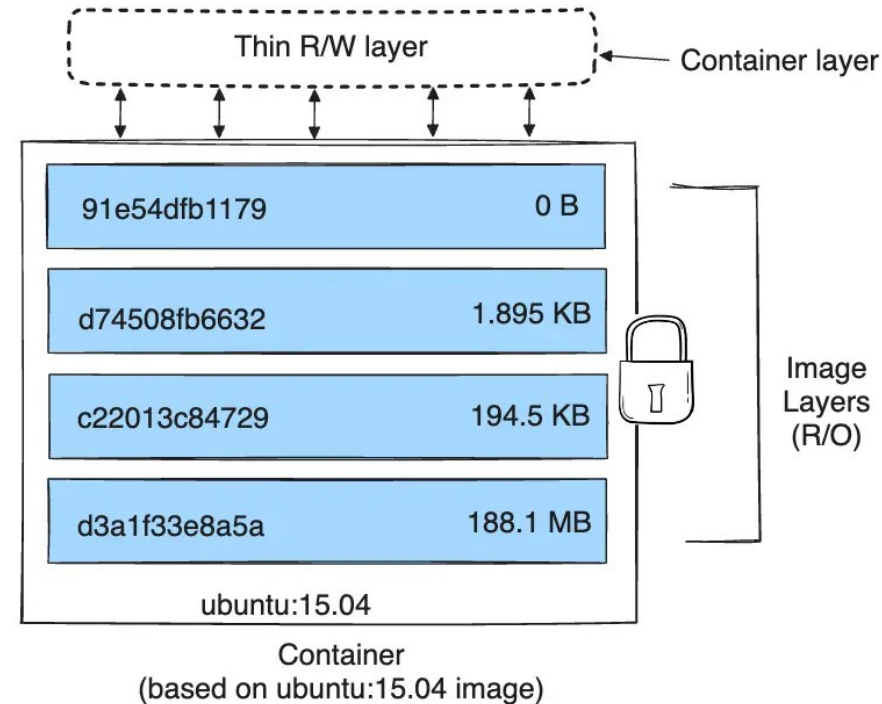
Docker CLI – image layers

Pour résumer,

Une image = Immuable (R/O)

Une image = Union des filesystem des différentes couches qui la composent.

Un conteneur = Image + une couche R/W dans laquelle toutes les modifications du FS réalisées lors de l'exécution du conteneur sont enregistrées.



Docker CLI - Dockerfile

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

Docker peut **créer des images** automatiquement en lisant les instructions d'un **Dockerfile**.

Un Dockerfile est un document texte qui contient toutes les **commandes** qu'un utilisateur peut utiliser pour construire une image.

Format d'un Dockerfile

Comment

INSTRUCTION arguments

INSTRUCTION arguments

My Dockerfile

FROM ubuntu

CMD ["top", "-b"]

Docker CLI – Dockerfile - Commandes (non-exhaustif)

FROM	Sélection d'une image de base
RUN	Exécute une commande lors de la construction
CMD	Commande exécutée lors de l'instanciation (1 seule) ou paramètres pour ENTRYPOINT si déclaré
ENV	Permet de définir des variables d'environnement (build + run)
COPY	Copie de fichiers dans l'image
ADD	Idem, mais peut récupérer des ressources via une URL
ENTRYPOINT	Point d'entrée lors du RUN de l'image
VOLUME	Créer un volume anonyme* lors de l'instanciation
WORKDIR	Répertoire de travail pour les instructions du Dockerfile (default /)
ARGS	Permet de définir des variables pour le build (--build-args)
EXPOSE	Informe Docker que le conteneur va écouter un port spécifique

Docker CLI – CMD vs ENTRYPOINT

CMD. Sets default parameters that can be overridden from the Docker Command Line Interface (CLI) when a container is running.

ENTRYPOINT. Default parameters that cannot be overridden when Docker Containers run with CLI parameters.

Docker CLI – CMD vs ENTRYPOINT

image t1

FROM ubuntu
ENTRYPOINT ["top", "-b"]

docker run t1

CONTAINER ID	IMAGE	COMMAND
6e9aa0a076a5	t1	"top -b"

image t2

FROM ubuntu
CMD ["top", "-b"]

docker run t2

CONTAINER ID	IMAGE	COMMAND
c7581e8e1931	t2	"top -b"

image t3

FROM ubuntu
ENTRYPOINT ["top", "-b"]
CMD ["-c"]

docker run t3

CONTAINER ID	IMAGE	COMMAND
e8d364e5e69d	t3	"top -b -c"

docker run t3 -u root

CONTAINER ID	IMAGE	COMMAND
21415f86912b	t3	"top -b -u root"

Docker CLI – Dockerfile – exemples (1)

Pull debian base image

FROM debian:latest

Install nginx (foreground mode)

RUN apt-get update && \

apt-get install --no-install-recommends -y nginx; \

echo "daemon off;" >> /etc/nginx/nginx.conf

Copy app source code

COPY myapp /var/www/html

Expose HTTP

EXPOSE 80

Start nginx

CMD ["/usr/sbin/nginx"]

```
f1s@docker:~/myapp$ tree
.
├── Dockerfile
└── myapp
    ├── background.jpg
    ├── index.html
    ├── style.css
    └── README.md

2 directories, 5 files
```

Docker CLI – Dockerfile – exemples (1)

Build : `docker build -t "nginxapp" .`

```
fls@docker:~/myapp$ docker build -t "nginxapp" . --no-cache
[+] Building 6.8s (8/8) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile            0.0s
=> => transferring dockerfile: 389B                             0.0s
=> [internal] load .dockerignore                               0.0s
=> => transferring context: 2B                                    0.0s
=> [internal] load metadata for docker.io/library/debian:latest 0.4s
=> CACHED [1/3] FROM docker.io/library/debian:latest@sha256:fab22df37377621693c68650b06680c0d8f7c6bf816ec92637944778 0.0s
=> [2/3] RUN apt-get update && apt-get install --no-install-recommends -y nginx; rm -rf /var/lib/apt/lists/*; echo 5.9s
=> [internal] load build context                                0.0s
=> => transferring context: 139B                                  0.0s
=> [3/3] COPY myapp /var/www/html/webapp                       0.1s
=> exporting to image                                           0.2s
=> => exporting layers                                           0.2s
=> => writing image sha256:8cb102f47bc4eeb08e529e920ee0dc7636b3ceadbe85d94ef3a570601f0ece63 0.0s
=> => naming to docker.io/library/nginxapp                      0.0s
```

Docker CLI – Dockerfile – exemples (1)

Résultat du build : L'image construite par le Dockerfile

```
fls@docker:~/myapp$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginxapp	latest	8cb102f47bc4	About a minute ago	134MB

• Layers

Cmp	Size	Command
	116 MB	FROM dc4f6460a303d43
	17 MB	RUN /bin/sh -c apt-get update && apt-get install --no-install-recommends -y nginx; rm -rf
	95 kB	COPY myapp /var/www/html/webapp # buildkit

Docker CLI - run

docker run --help

Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Create and run a new container from an image

```
fls@docker:~/myapp$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:88ec0acaa3ec199d3b7eaf73588f4518c25f9d34f58ce9a0df68429c5af48e8d
Status: Downloaded newer image for hello-world:latest

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

Docker CLI – RUN

<https://docs.docker.com/engine/reference/commandline/run/>

Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

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

```
-d : mode détaché  
--name : nom du conteneur  
-e : définir des variables d'environnement  
-e : définir des variables d'environnement  
-e : définir des variables d'environnement  
-p : mapping de ports hôte:conteneur  
-p : mapping de ports hôte:conteneur  
-v : bind mount de volume  
--restart : action en cas de reboot  
IMAGE
```


Docker CLI – Dockerfile – exemples (1)

Création d'un conteneur avec l'image créée

```
fls@docker:~/myapp$ docker run -d -p 8777:80 nginxapp  
75aef826eb09c4cf579a33435793d90964a82f1a47a8609a4ffac6ccce8e7f88
```

```
fls@docker:~/myapp$ curl http://localhost:8777/webapp/  
<html>  
<head>  
  <title>My Web App</title>  
  <link href="style.css" rel="stylesheet" type="text/css">  
</head>  
<body>  
  <div id="container">  
      
  </div>  
</body>  
</html>
```

Docker CLI – EXEC

<https://docs.docker.com/engine/reference/commandline/exec/>

Usage: `docker exec [OPTIONS] CONTAINER COMMAND [ARG...]`

Permet d'exécuter une commande dans un conteneur en cours d'exécution

Exemples

```
docker exec -d mycontainer touch /tmp/text.txt
```

```
docker exec -it mycontainer /bin/bash
```

Docker CLI – COMMIT

<https://docs.docker.com/engine/reference/commandline/commit/>

Usage: `docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]`

Permet de créer une image à partir d'un conteneur

Exemple

```
docker commit mycontainer mycontainerimg:1.2
```

Docker CLI – Dockerfile – exemples (2)

```
FROM debian:stable-slim

RUN apt-get update -y && apt-get install -q -y apache2

RUN apt-get install -q -y mariadb-server


COPY database.sql /

COPY *.php /var/www/html/


RUN apt-get install -q -y php-mysql php && \
    rm -f /var/www/html/index.html && apt-get autoclean -y


EXPOSE 80

WORKDIR /var/www/html

ENTRYPOINT service mariadb start && mysql < /database.sql && apache2ctl -D FOREGROUND
```

```
fls@docker:~/phpcrud$ tree
.
├── config.php
├── create.php
├── database.sql
├── delete.php
├── Dockerfile
├── error.php
├── index.php
├── README.md
├── read.php
├── update.php
└──

1 directory, 10 files
```

Docker CLI – Dockerfile – exemples (2)

database.sql

```
CREATE DATABASE IF NOT EXISTS phpcrud;  
CREATE USER 'user'@'%' IDENTIFIED BY 'pass';  
GRANT ALL PRIVILEGES ON phpcrud.* TO 'user'@'%';  
use phpcrud;  
  
CREATE TABLE IF NOT EXISTS students (  
  id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
  nom VARCHAR(100) NOT NULL,  
  ecole VARCHAR(255) NOT NULL,  
  age INT(10) NOT NULL  
);
```

```
fls@docker:~/phpcrud$ tree  
.  
├── config.php  
├── create.php  
├── database.sql  
├── delete.php  
├── Dockerfile  
├── error.php  
├── index.php  
├── README.md  
├── read.php  
└── update.php  
  
1 directory, 10 files
```

Source de l'application d'origine : <https://github.com/Letecode/phpcrud>

Docker CLI – Dockerfile – exemples (2)

config.php

```
<?php

/* Database connexion */

define('DB_SERVER', 'localhost');

define('DB_USERNAME', 'user');

define('DB_PASSWORD', 'pass');

define('DB_NAME', 'phpcrud');

/* Connexion à la base de données */

$link = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD, DB_NAME);

// verifier connection

if($link === false){

    die("ERROR: Could not connect. " . mysqli_connect_error());

}

?>
```

```
fls@docker:~/phpcrud$ tree
.
├── config.php
├── create.php
├── database.sql
├── delete.php
├── Dockerfile
├── error.php
├── index.php
├── README.md
├── read.php
└── update.php

1 directory, 10 files
```

Docker CLI – Dockerfile – exemples (2)

BUILD

```
fls@docker:~/phpcrud$ docker build -t "mycrudimg" . --no-cache
[+] Building 59.5s (12/12) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 460B                               0.0s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                     0.0s
=> [internal] load metadata for docker.io/library/debian:stable-slim 0.4s
=> CACHED [1/7] FROM docker.io/library/debian:stable-slim@sha256:1529b15b786397b4695b16 0.0s
=> [internal] load build context                                  0.0s
=> => transferring context: 240B                                   0.0s
=> [2/7] RUN apt-get update -y && apt-get install -q -y apache2 17.5s
=> [3/7] RUN apt-get install -q -y mariadb-server               14.8s
=> [4/7] COPY database.sql /                                     0.3s
=> [5/7] RUN apt-get install -q -y php-mysql php && rm -f /var/www/html/in 18.8s
=> [6/7] COPY *.php /var/www/html/                              0.2s
=> [7/7] WORKDIR /var/www/html                                  0.1s
=> exporting to image                                           6.9s
=> => exporting layers                                           6.8s
=> => writing image sha256:bc977c34473ce35ec7022fa00f1ed1f64305e95b9fb6d396a113b64df7d1 0.0s
=> => naming to docker.io/library/mycrudimg                     0.0s
```

Docker CLI – Dockerfile – exemples (2)

RUN

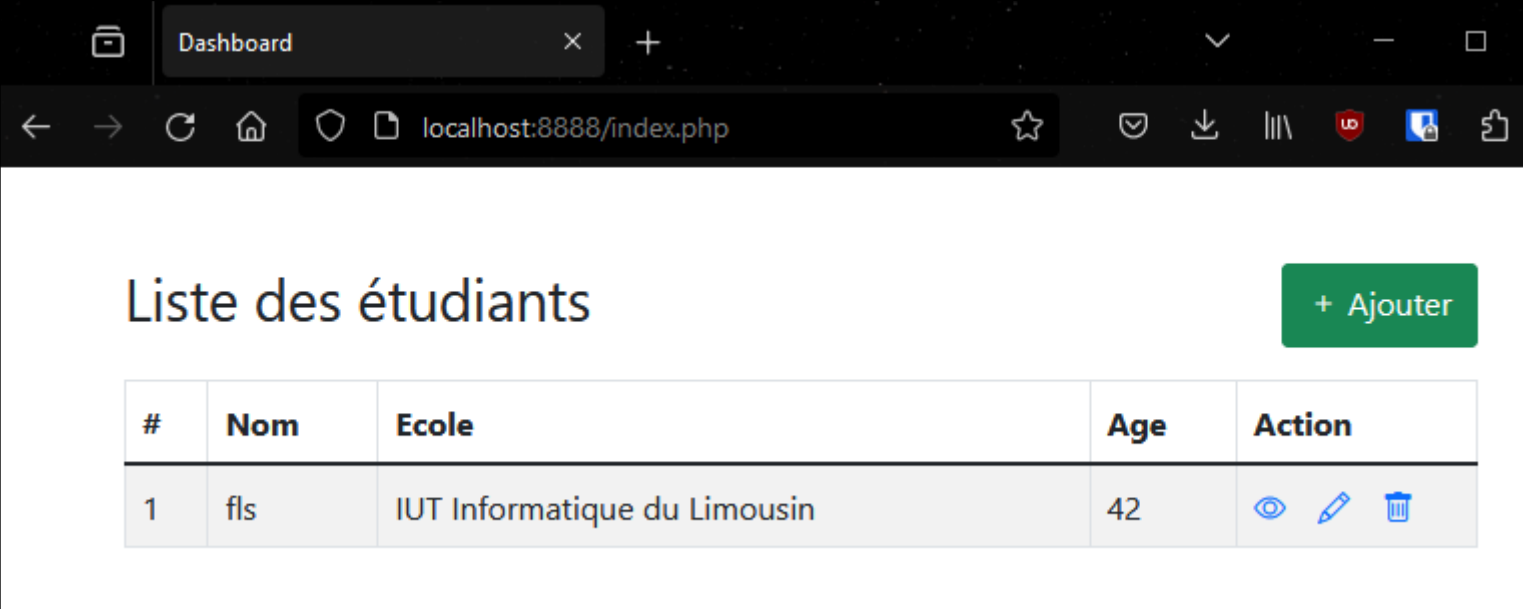
`docker run -d --name "mycrudapp" -p 8888:80 mycrudimg`

```
fls@docker:~/phpcrud$ docker run -d --name "mycrudapp" -p 8888:80 mycrudimg
43b8439dc846b41a844ce008c0be2aaafc80ed60f66fc5036e311eb204b4ff864
```

```
fls@docker:~/phpcrud$ docker ps --format 'table {{ .ID }}\t{{.Image}}\t{{ .Ports }}'
CONTAINER ID    IMAGE                PORTS
43b8439dc846    mycrudimg            0.0.0.0:8888->80/tcp, :::8888->80/tcp
```


Docker CLI – Dockerfile – exemples (2)

TEST






Dashboard

localhost:8888/index.php

Liste des étudiants

+ Ajouter

#	Nom	Ecole	Age	Action
1	fls	IUT Informatique du Limousin	42	  

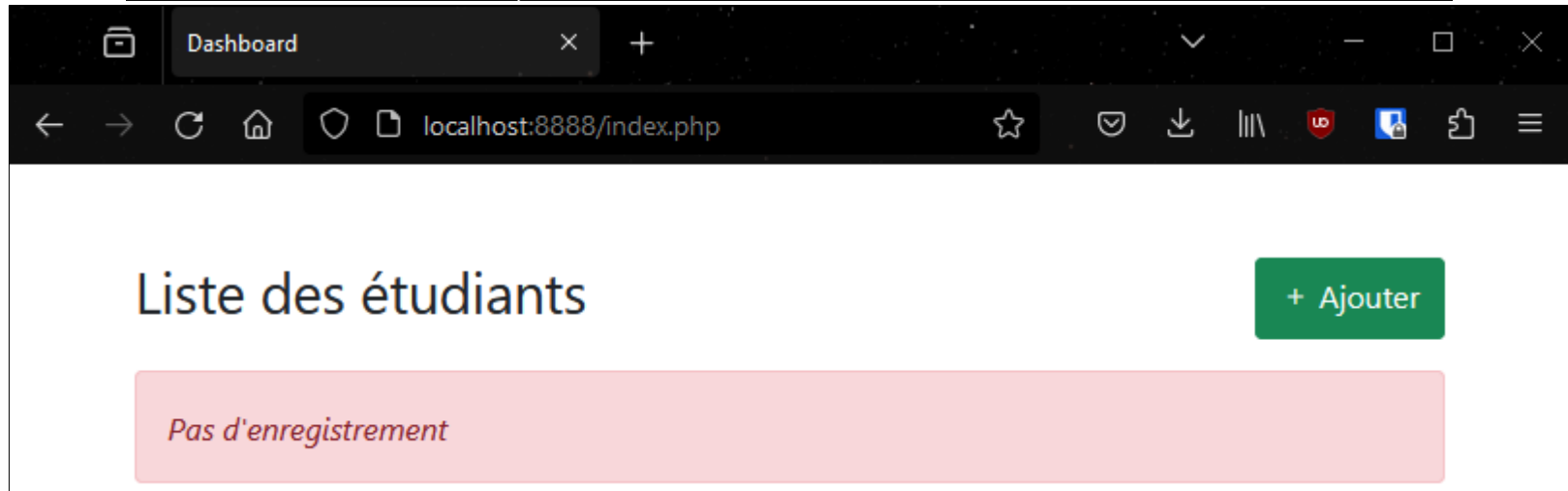
Docker CLI – Dockerfile – exemples (2)

Persistance des données ?

```
docker rm -f 43b8439dc846
```

```
docker run -d --name "mycrudapp" -p 8888:80 mycrudimg
```

```
fls@docker:~/phpcrud$ docker run -d --name "mycrudapp" -p 8888:80 mycrudimg  
20553a26ad1c78b0cf66ef2f6cb094bd498cbd2930d96be3e6b7f81f52898380
```



Docker CLI – Dockerfile – exemples (2)

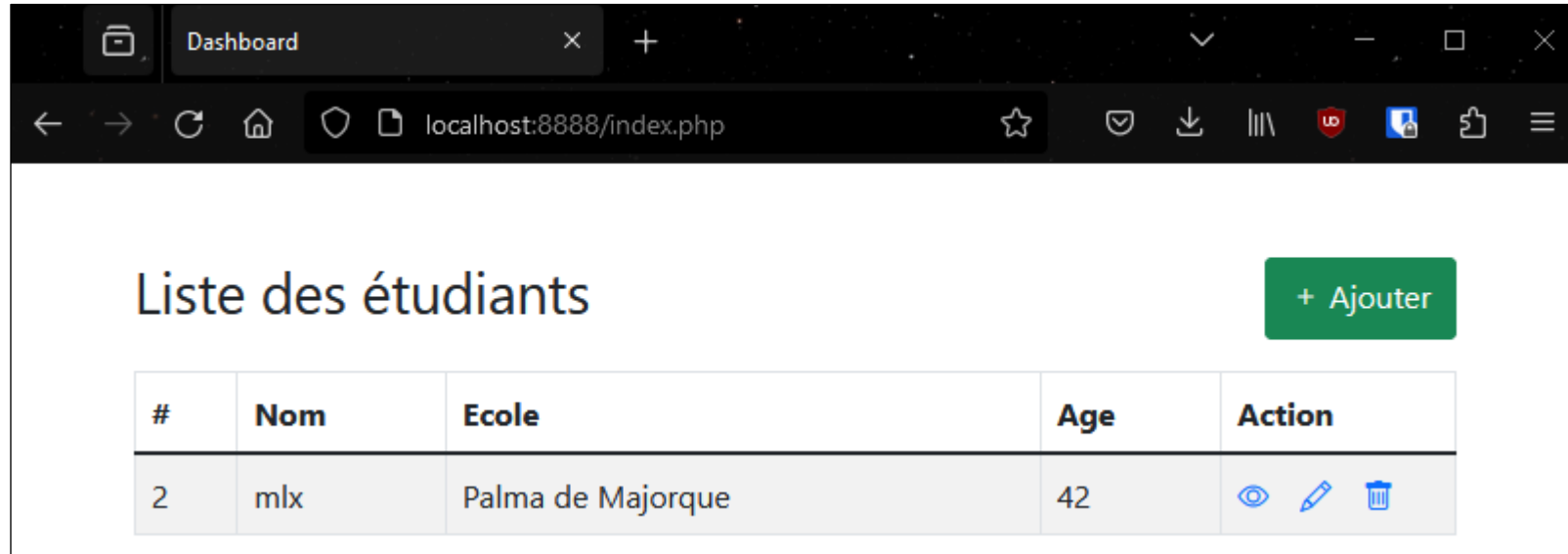
Persistance des données ?

`docker run -d --name "mycrudapp" -v dbdata:/var/lib/mysql -p 8888:80 mycrudimg`

```
fls@docker:~/phpcrud$ docker run -d --name "mycrudapp" -v dbdata:/var/lib/mysql -p 8888:80 mycrudimg
07ec5a0a6411970e9dac26cbabdec9de2c88806cca39247e0eeaa7fa75eccbda
fls@docker:~/phpcrud$ docker volume inspect dbdata
[
  {
    "CreatedAt": "2023-11-05T18:11:53+01:00",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/dbdata/_data",
    "Name": "dbdata",
    "Options": null,
    "Scope": "local"
  }
]
```

Docker CLI – Dockerfile – exemples (2)

TEST






Dashboard

localhost:8888/index.php

Liste des étudiants

+ Ajouter

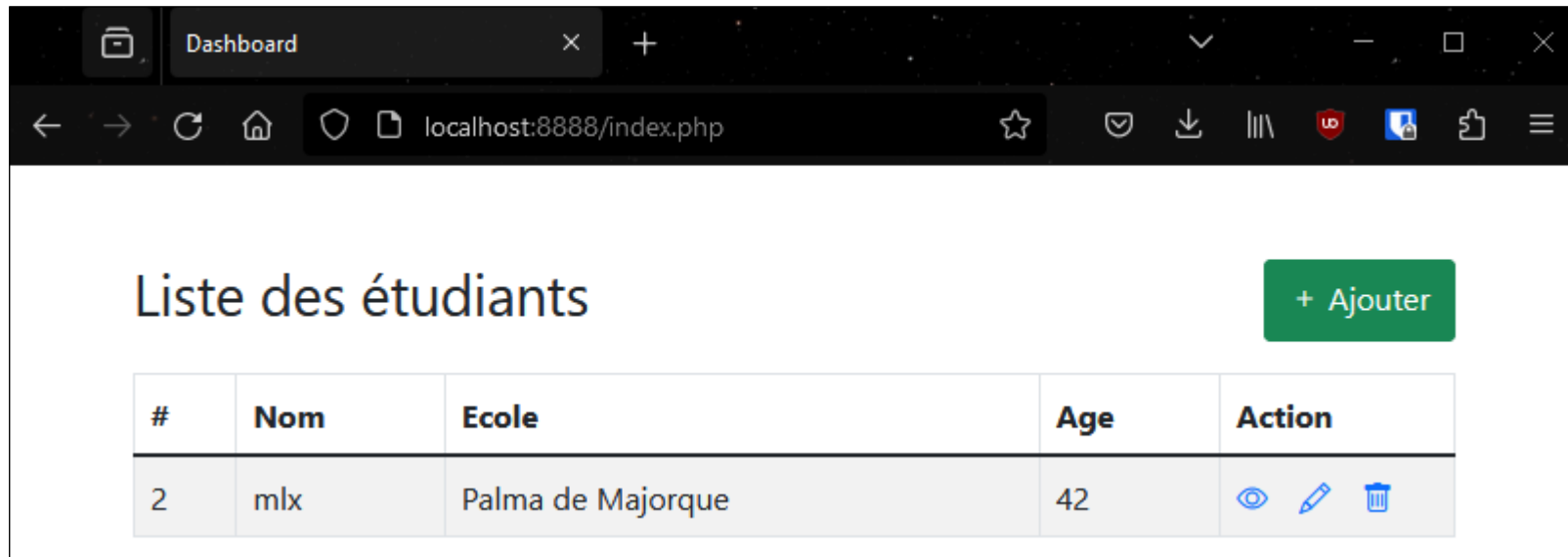
#	Nom	Ecole	Age	Action
2	mlx	Palma de Majorque	42	  

Docker CLI – Dockerfile – exemples (2)

Persistence des données ?

`docker run -d --name "mycrudapp" -v dbdata:/var/lib/mysql -p 8888:80 mycrudimg`

```
fls@docker:~/phpcrud$ docker run -d --name "mycrudapp" -p 8888:80 mycrudimg
20553a26ad1c78b0cf66ef2f6cb094bd498cbd2930d96be3e6b7f81f52898380
```






Dashboard

localhost:8888/index.php

Liste des étudiants

+ Ajouter

#	Nom	Ecole	Age	Action
2	mlx	Palma de Majorque	42	  

Docker CLI – Docker compose

Usage: docker compose [OPTIONS] COMMAND

Define and run multi-container applications with Docker.

Commandes usuelles

build	Build or rebuild services
up	Create and start containers
down	Stop and remove containers, networks
ps	List containers
logs	View output from containers

Docker CLI – Docker compose

config format	Parse, resolve and render compose file in canonical format
cp	Copy files between a service container and the local fs
create	Creates containers for a service.
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.
ls	List running compose projects
pause	Pause services
port	Print the public port for a port binding.

Docker CLI – Docker compose

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.
start	Start services
stop	Stop services
top	Display the running processes
unpause	Unpause services
version	Show the Docker Compose version information
wait	Block until the first service container stops

Docker CLI – Docker compose file

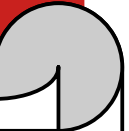
Docker compose permet de définir et lancer une application multi-conteneurs. Ex : Application Web + PHP + Database + Redis

Pour une application comme celle-ci, il faut donc construire 4 conteneurs pour héberger les différents **services**.

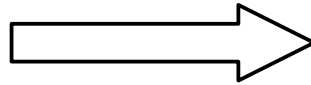
Il serait possible de lancer les conteneurs « à la main » avec docker run, mais il est beaucoup plus aisé et pratique d'utiliser docker compose.

Un **fichier Docker compose** est écrit au format **YAML** et définit les différents **services** d'une application, leur build/image, leurs interactions et tous les éléments de configuration nécessaires (env, ports, network, volumes, etc...)

Docker CLI – Dockerfile vs Docker compose file



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



```
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 #optional  
    restart: unless-stopped
```

Docker CLI – Docker compose file

Docker compose File Reference

<https://docs.docker.com/compose/compose-file/compose-file-v3/>

Un premier pas

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

Docker CLI – Docker compose example (1)

Run a **nextcloud** instance : https://hub.docker.com/_/nextcloud/

version: '2'

services:

app:

image: nextcloud

restart: always

ports:

- 8080:80

links:

- db

volumes:

- nextcloud:/var/www/html

environment:

- MYSQL_PASSWORD=passw23

- MYSQL_DATABASE=nextcloud

- MYSQL_USER=nextcloud

- MYSQL_HOST=db

db:

image: mariadb:10.6

restart: always

command: --transaction-isolation=READ-COMMITTED --log-bin=binlog --binlog-format=ROW

volumes:

- db:/var/lib/mysql

environment:

- MYSQL_ROOT_PASSWORD=password

- MYSQL_PASSWORD=passw23

- MYSQL_DATABASE=nextcloud

- MYSQL_USER=nextcloud

volumes:

nextcloud:

db:

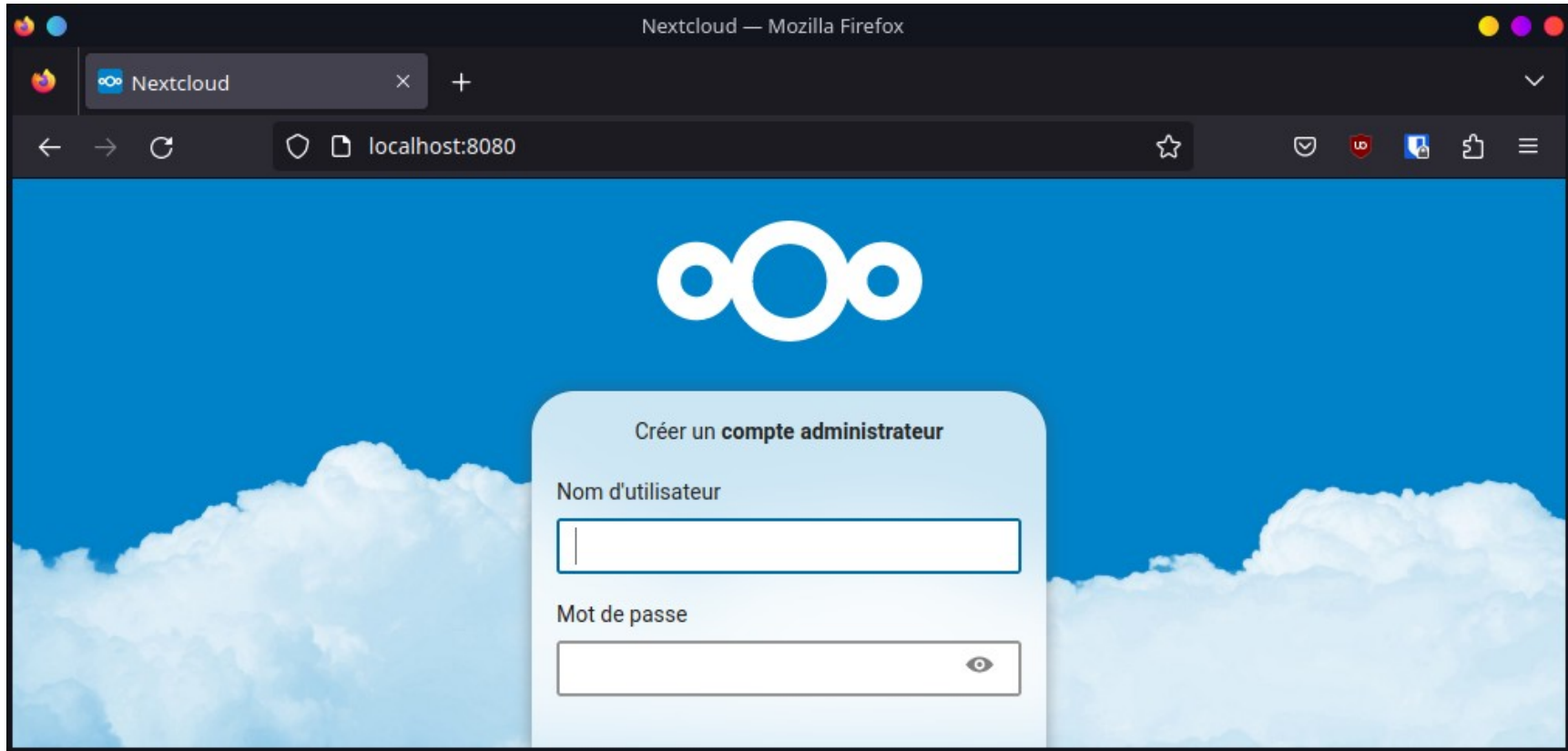
Docker CLI – Docker compose example (1)

```
fls@cygnus:~/ex3$ ls
docker-compose.yml
fls@cygnus:~/ex3$ docker compose up -d
[+] Running 3/3
 ✓ Network ex3_default      Created           0.1s
 ✓ Container ex3-db-1       Started        0.0s
 ✓ Container ex3-app-1      Started        0.0s
fls@cygnus:~/ex3$ docker compose ps
```

NAME	IMAGE	COMMAND	SERVICE	CREATED	STATUS	PORTS
ex3-app-1	nextcloud	"/entrypoint.sh apache2-foreground"	app	4 seconds ago	Up 2 seconds	0.0.0.0:8080->80/tcp
ex3-db-1	mariadb:10.6	"docker-entrypoint.sh --transaction-isolation=READ-COMMITTED --log-bin=binlog --binlog-format=ROW"	db	4 seconds ago	Up 3 seconds	3306/tcp

```
fls@cygnus:~/ex3$
```

Docker CLI – Docker compose example (1)



Docker CLI – Docker compose file

Il est tout à fait possible de combiner Dockerfile et Docker compose, pour lancer une application multi-conteneurs avec un build d'image pour chaque service.

Pour simplement build les images d'un fichier docker compose :

docker compose build

```
Version: "3.2"
services:
  web:
    build:
      dockerfile: Dockerfile
      context: php/
    image: mywebimg:1.0
    restart: unless-stopped
    volumes:
      - "./src:/var/www/html/"
    ports:
      - 8000:80
```

Docker CLI – Docker compose example (2)

L'application PHPCRUD en TP