



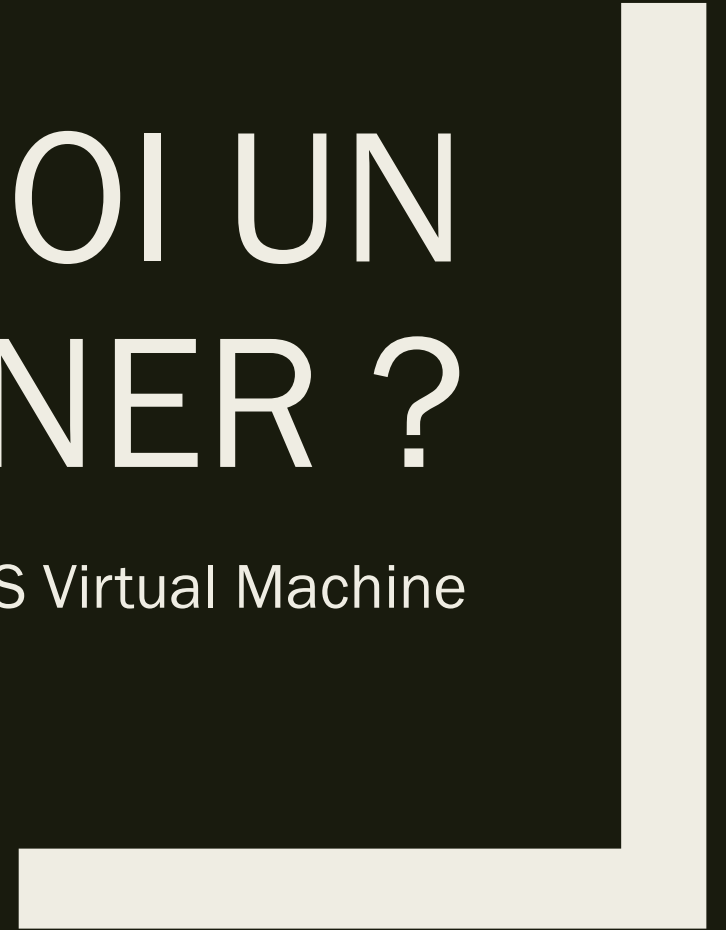
DOCKER

Faciliter le développement et le déploiement

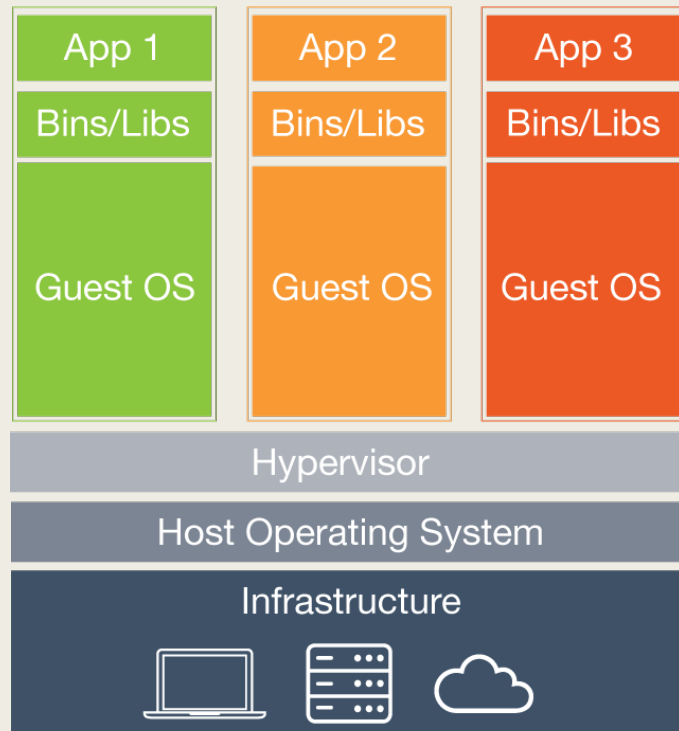


C'EST QUOI UN CONTAINER ?

Container VS Virtual Machine



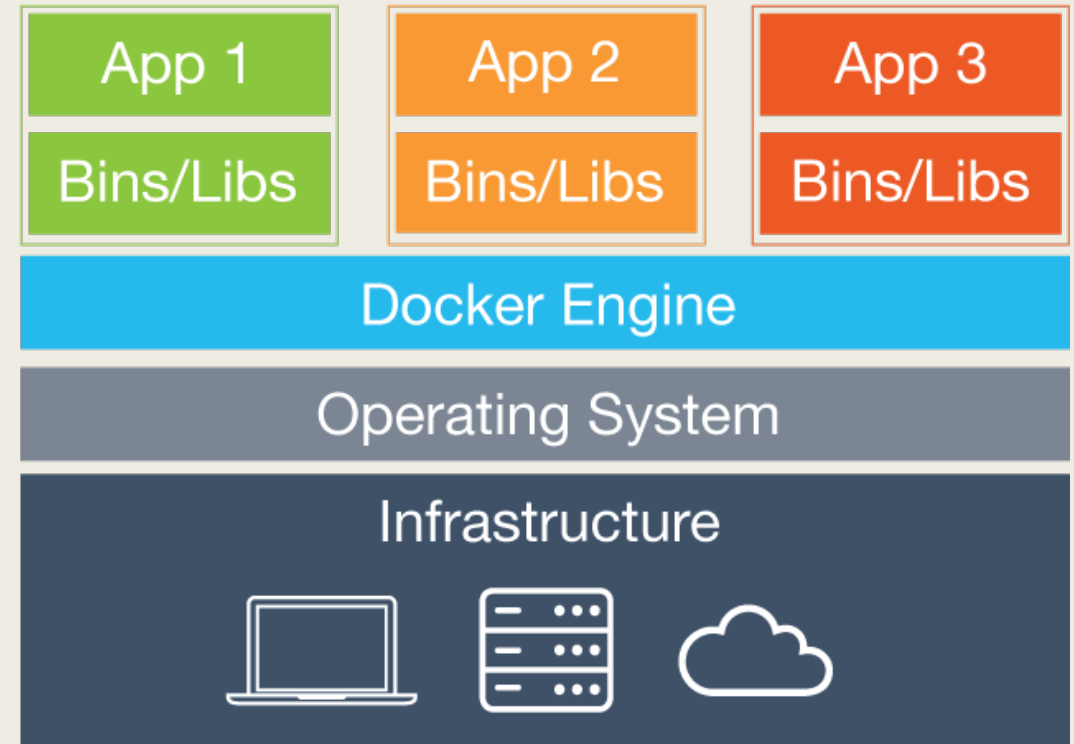
Une machine virtuelle



- Avant de parler de conteneur, on est obligé de parler de machine virtuelle
- Le but d'une machine virtuelle est de recréer totalement un système sur un système hôte : son attribution de ressources, son OS, ses librairies, ses drivers... on parle de virtualisation lourde
- Il arrive souvent que toutes les ressources d'une VM ne soient pas utilisées par une application, si on ajoute la taille totale d'une VM à sauvegarder, la nécessité d'une virtualisation légère se fait sentir

Les conteneurs

- Le but de la virtualisation légère est de faire tourner des environnements linux isolés dans des conteneurs partageant le même noyau
- Contrairement à une VM, un conteneur n'inclut pas d'OS puisqu'il s'appuie sur les fonctionnalités de la machine hôte et chaque conteneur accède à l'OS hôte de façon totalement isolée
- Toujours contrairement à une VM standard, un conteneur ne va pas réserver les ressources, il ne va utiliser que les ressources nécessaires et laisser le reste disponible
- Les conteneurs seront donc bien plus légers et donc plus rapides à sauvegarder et à migrer qu'une VM





Un gros avantage en développement

- Utiliser des containers, plus légers que des VM, permet en production d'en démarrer plusieurs très rapidement pour faire face à des besoins de mise à l'échelle
- Mais ils sont aussi un avantage considérable en développement car ils permettent d'avoir un environnement de travail commun entre tous les membres d'une équipe, quelque soit leur OS.

Fini le « Pourtant ça marche sur ma machine »

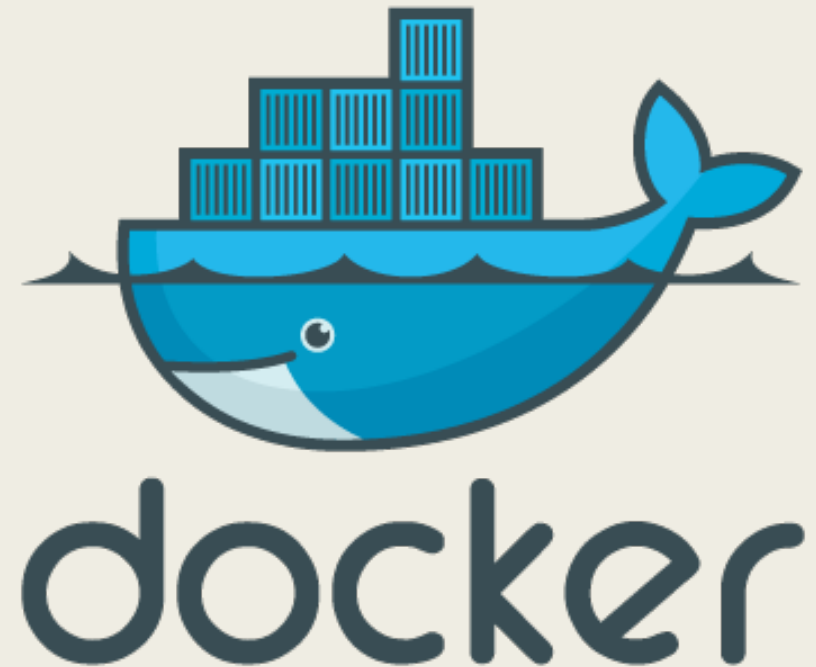
DOCKER

Histoire et installation



Un peu d'histoire

- Docker a initialement été créé pour répondre à des besoins d'une entreprise nommée **DotCloud** avant de créer une structure **Docker Inc** et de placer son produit **Docker** en open source en **2013**
- Docker apporte une notion importante dans le monde du container : un conteneur ne doit faire tourner qu'un seul processus. Donc dans le cas d'un stack LAMP, il faudra créer 3 conteneurs (un Apache, un MySQL et un PHP)
- C'est aujourd'hui le moteur de conteneurisation le plus utilisé
- Docker est écrit en Go



Stateless vs Stateful

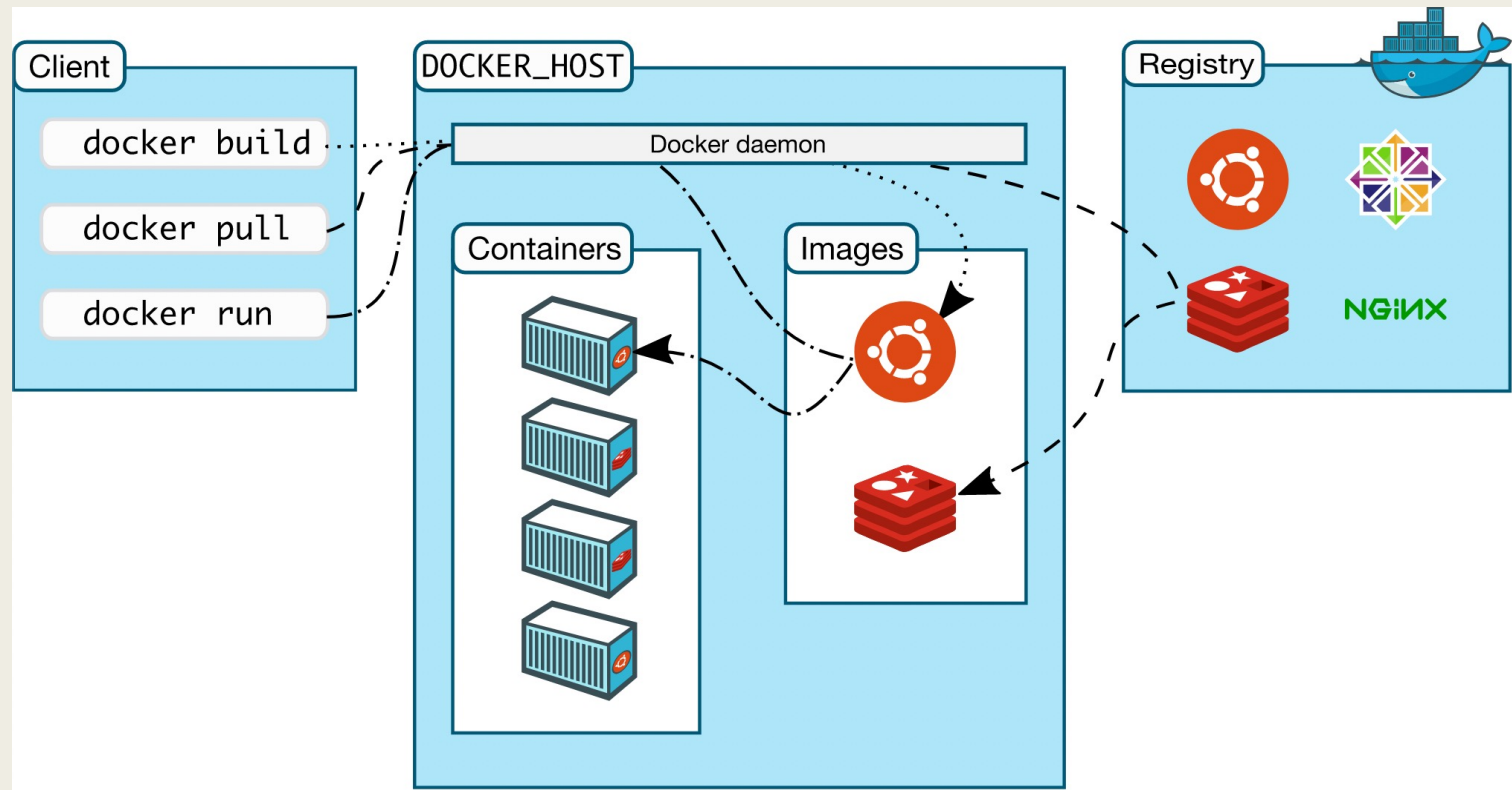
- Dans le monde de Docker, il existe deux catégories de conteneurs : Stateless et Stateful
- Par exemple une base de donnée sera dite **Stateful** car elle stocke un état. Si j'éteins et rallume ma base de données, elle sera dans le même état
- A l'inverse, le protocole HTTP est Stateless, il ne stocke pas d'état et à chaque nouvelle requête, la réponse sera la même

Immutabilité

- La notion d'immutabilité d'un conteneur est très importante.
- Un conteneur ne doit pas stocker de données qui doivent être pérennes car il les perdra
- Si vous souhaitez en local créer une DB dans un conteneur Docker, il faudra créer un volume pour les pérenniser

Architecture

- En terme d'architecture, Docker fonctionne sur le paradigme client – serveur
- Le Client Docker communique avec le Docker Daemon qui fait tourner le Docker Engine
- Le Docker Engine fait tourner les conteneurs et fait office de controleur
- Il est tout à fait possible de faire tourner le Client Docker et le Docker Deamon sur deux machines différentes

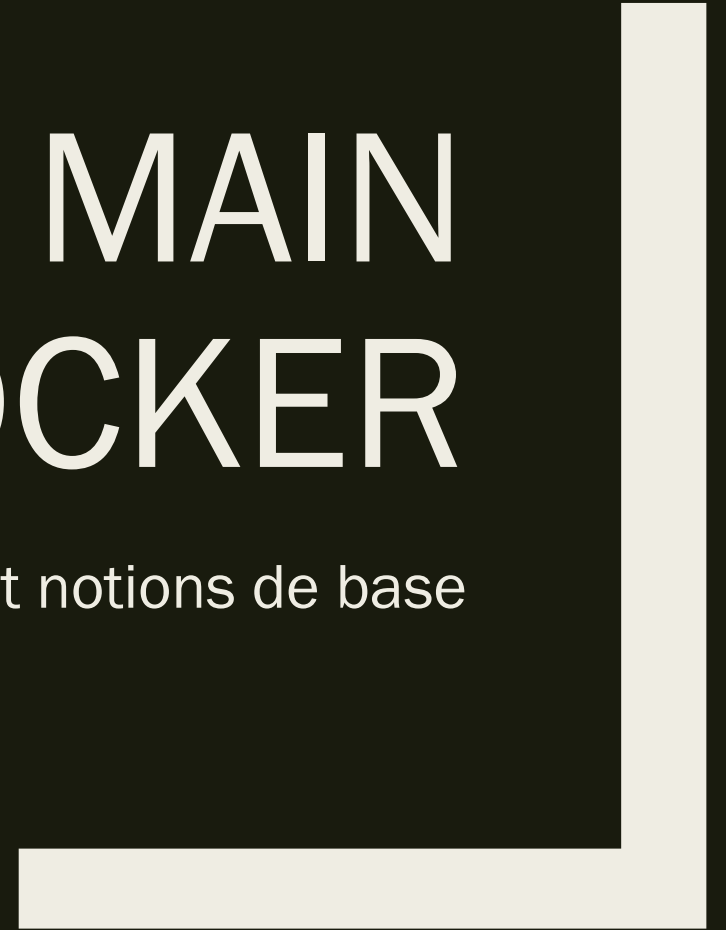


Installation

- Il existe 3 versions de Docker :
 - *Docker Desktop (pour Windows et Mac)*
 - *Docker Community Edition (pour Linux)*
 - *Docker Enterprise (payant et pour Linux)*
- Pour installer Docker Desktop il est nécessaire de se créer un compte Docker Hub
- L'installation de Docker pour Windows est un peu laborieuse et est [entièrement détaillée ici](#)
- L'installation de Docker CE ne nécessite pas de compte Docker Hub et la procédure pour Ubuntu (et ses dérivés) [est disponible ici](#)

PRENDRE EN MAIN DOCKER

Commandes et notions de base



Le concept d'image

- Une image Docker est un Template prêt à l'emploi avec des instructions pour la création d'un conteneur
- Il est tout à fait possible de créer des images nous même, que ce soit from scratch ou à partir d'une image déjà existante

DockerHub

- DockerHub est un peu le GitHub de Docker.
- C'est le repository principal d'images pour Docker. Il permet d'aller chercher des images déjà développées ou alors de stocker ses images pour pouvoir les partager avec son équipe ou avec le reste du monde
- Les images sont séparées en plusieurs catégories :
 - *Les images officielles, ce sont celles qui bénéficient du plus grand niveau de vérification, elles sont stables et c'est un bon point de départ*
 - *Les images de publishers vérifiés : Ce sont des équipes considérées comme étant de confiance.*
 - *Les autres : Là pour le coup, on trouve à boire et à manger*

Lancer un conteneur

- Nous allons lancer notre premier conteneur Hello World !
- Une fois Docker Desktop lancé (si vous êtes PC ou Mac), ouvrez un terminal et tapez

```
docker run hello-world
```

- Le Docker Deamon va essayer de trouver l'image « hello-world » en local. Si il ne la trouve pas, il va aller chercher sur le registry Docker officiel

Hello World !

- Dans notre cas, l'image n'était pas disponible en local
- Docker a donc été chercher la dernière version de l'image avant de la lancer

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
109db8fad215: Pull complete
Digest: sha256:7d91b69e04a9029b99f3585aaaccae2baa80bcf318f4a5d2165a9898cd2dc0a1
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(arm64v8)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

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

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

Quelques options

- On va maintenant lancer un conteneur Nginx `docker run -d -p 8080:80 nginx`
- On commence encore une fois par « docker run » pour lancer un conteneur
- On passe ensuite l'option « -d » pour « detach » ce qui permet de faire tourner le conteneur en tâche de fond sans bloquer le terminal
- L'option suivante est « -p 8080:80 »
Cette option permet de mapper l'utilisation de nos ports. On dit ici de router le trafic du port 8080 de notre machine vers le port 80 (par défaut) du conteneur
Si on va sur <http://localhost:8080/> on va donc avoir la page pas défaut de Nginx
- Enfin on passe le nom de l'image que l'on cherche à exécuter

Plus loin dans les options

- Vous devriez avoir une réponse qui ressemble à ceci :
- L'immense chaîne en dernière ligne correspond à l'ID du container

```
jean-francois@macbook-air-de-jean-francois ~ % docker run -d -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
d10c227306ce: Pull complete
6d7f45405386: Pull complete
1b5147f94884: Pull complete
52ae3a597441: Pull complete
3ce50094efa7: Pull complete
7bf647869efa: Pull complete
Digest: sha256:e4f225847897e0238a4d2a57c849a82ef478c89fa0e1077d5dc57077703d5fec
Status: Downloaded newer image for nginx:latest
2848a77f400c5a51b5d6cfd6cd13ec40a3b8913817fde6c48c4ad6d9e4f9bd47
```

Plus loin dans les options

- Il est aussi possible de voir la liste de tous les containers en cours d'exécution avec la commande « **docker ps** » (l'option « -a » affichera aussi les containers stoppés)

```
jean-francois@macbook-air-de-jean-francois ~ % docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0a9b27ea75f6	nginx	"/docker-entrypoint...."	9 seconds ago	Up 8 seconds	0.0.0.0:8080->80/tcp, :::8080->80/tcp	charming_roentgen

- Comme nous avons utilisé l'option « -d » pour lancer notre conteneur en arrière plan, il va falloir taper « **docker stop <container id>** » pour l'arrêter

```
docker stop 0a9b27ea75f6
```

- Il va être aussi possible de supprimer le conteneur avec la commande « **docker rm <container id>** » ou de le relancer avec « **docker start <container id>** »

```
docker rm 0a9b27ea75f6
```

Explorer un container

- Il est possible d'explorer un container qui est en train de tourner

```
docker exec -ti 06622dcce7b1 sh
```

- L'option « -ti » permet d'avoir le TTY (c'est-à-dire d'afficher la sortie sur la console)
- Ensuite vient l'id du contener
- Et enfin je renseigne l'interpréteur de commandes (ici shell mais je pourrais tout aussi bien avoir bash)

Toujours des commandes utiles

- La commande « `docker images -a` » permet d'afficher toutes les images présentes sur votre machine avec leurs tailles

```
jean-francois@macbook-air-de-jean-francois ~ % docker images -a
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	9fa3786cd6ac	3 hours ago	126MB
hello-world	latest	bc11b176a293	2 months ago	9.14kB

- Du coup, pour supprimer une image : « `docker rmi <image id>` »
ATTENTION : on ne peut pas supprimer une image tant qu'un conteneur l'utilisant existe (même si il est stoppé)

```
docker rmi 9fa3786cd6ac
```

Nettoyer le système

- Vous allez vite vous en rendre compte mais après avoir fait pas mal de tests, les images Docker s'empilent et prennent de la place...
- Pour faire un grand ménage : « docker system prune »

```
[jean-francois@macbook-air-de-jean-francois ~ % docker system prune
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- all dangling build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
fe71578749d4aa466a65ab57c2180dffc6fda39938c00af873dfc3b5576631bf

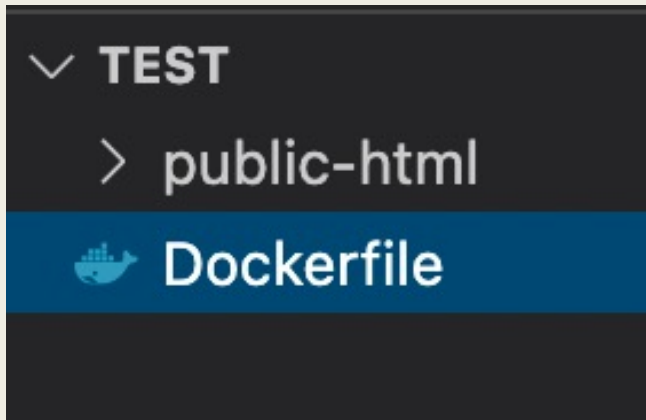
Total reclaimed space: 0B
```

DOCKERFILE

Les recettes de cuisine avec Docker



Créer sa propre image Docker



- Dockerfile est un fichier qui sert de recette à une image Docker
- Elle va décrire tout le processus de création d'une image. Chaque instruction sera un nouveau layer correspondant à une étape de construction
- Attention cependant, plus vous allez ajouter de layers, plus votre image sera lourde et possiblement peu performante
- Pour créer un Dockerfile, rien de plus simple : créez un nouveau dossier de travail et ajoutez à la racine de ce dossier un fichier nommé « Dockerfile » sans extension

Les instructions dans le Dockerfile

- La première chose à faire est de définir l'image qui va nous servir de base grâce à la commande FROM
- Il est possible de partir d'une base existante ou de commencer de rien avec « FROM scratch »
- FROM n'est utilisable qu'une seule fois dans le Dockerfile



Dockerfile

```
1 FROM httpd:2.4
2 # Image Apache officielle
```

ADD / COPY

- Voici deux commandes similaires exécutables dans Dockerfile, elles permettent de copier des fichiers depuis le système hôte vers le conteneur
- ADD est sensiblement plus puissante que COPY puisque elle accepte une URL ou un fichier .tar (une archive qu'elle va extraire automatiquement)
- Néanmoins, selon le guide des bonnes pratiques Docker, il est préférable d'utiliser COPY tant que possible car ADD peut laisser l'utilisateur importer des choses inutiles qui vont gonfler la taille de l'image par inadvertance

```
COPY <src> <dest>
```

```
ADD <src> <dest>
```

ADD / COPY

- Voici un exemple d'utilisation dans un Dockerfile

```
ADD . /app/  
# Ajoute le contenu du dossier actuel (ici la racine)  
# Dans un dossier /app créé pour l'occasion  
  
COPY ./public-html/ /usr/local/apache2/htdocs/  
# Similaire à ADD, va copier le dossier local  
# public-html qui est à la racine de mon dossier de travail  
# vers /usr/local/apache2/htdocs/ du conteneur
```

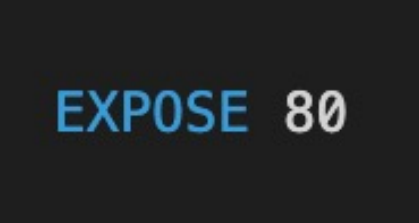
RUN

- On commence à rentrer dans les commandes sérieuses du Dockerfile
- Cette commande permet d'exécuter une liste de commandes dans le terminal pendant la création de l'image
- On peut y faire des choses très simples comme créer un dossier ou un fichier ou bien faire des choses plus complexes comme installer des updates Debian ou encore installer Node.js sur une base Debian
- Il est possible d'exécuter plusieurs RUN

```
WORKDIR /  
# Spécifie le dossier de travail, un peu équivalent à cd  
  
RUN mkdir coucou \  
&& cd coucou \  
&& touch truc.html  
# Simple : crée un dossier, s'y déplace et crée un fichier  
  
RUN apt-get update -yq \  
&& apt-get install curl gnupg -yq \  
&& curl -sL https://deb.nodesource.com/setup_10.x | bash \  
&& apt-get install nodejs -yq \  
&& apt-get clean -y  
# Plus compliqué, met à jour Debian, installe curl  
# puis va chercher Node.js et l'installe
```

EXPOSE

- Permet de définir le port qui sera écouté par votre application
- Il est préférable de toujours utiliser le port par défaut de l'application que vous construisez, par exemple Apache écoute traditionnellement le port 80, MongoDB le 27017...



EXPOSE 80

CMD

- Enfin, une dernière commande : CMD
- Cette commande permet de définir la commande qui sera lancée lors du démarrage de notre conteneur
- Typiquement utile pour lancer une app Node.js dans un conteneur

```
CMD npm run start
```

Construire son image

- On a fabriqué un beau Dockerfile, il ne nous reste plus qu'à en faire une image

```
docker build -t le-nom-de-mon-image .
```

- L'option « -t » permet de donner un nom à votre image
- Enfin, le « . » à la fin permet de spécifier l'emplacement du Dockerfile, ce qui signifie que mon terminal est à l'endroit de mon Dockerfile
- Il ne vous reste plus qu'à « docker run » votre image comme n'importe quelle autre image

Tout le reste

- Il existe en vrai pas mal de commandes pour créer un Dockerfile mais ce sont ici les principales
- [La liste complète peut se trouver ici](#)

Exercice

- Créez un Dockerfile avec une image de PHP 7.4 avec Apache et faite en sorte de copier un fichier index.php depuis votre hôte vers votre container pour qu'il soit lancé quand on démarre le container

Correction

- Notez que de base, l'image de PHP ne contient aucune extension (PDO par exemple n'est pas présent)
- Il est possible de rajouter ces extensions grâce à RUN

Dockerfile

```
1 FROM php:7.4-apache
2
3 # PHP extensions
4 RUN docker-php-ext-install mysqli pdo pdo_mysql
5
6 # Je copie mes fichiers vers mon container
7 COPY ./public-html/ /var/www/html/
8
```

Le problème de l'immuabilité

- Un truc problématique avec Docker, c'est que si on veut modifier un fichier statique dans notre image on est obligé de... la reconstruire, ce qui n'est clairement pas pratique (surtout en dev) !
- Heureusement il existe une solution : les Volumes
- Lors du montage d'un container, nous allons pouvoir spécifier que nous voulons un « pont » entre un dossier de la machine hôte et un dossier du conteneur avec l'option « -v » suivi du dossier de la machine hôte et du dossier du conteneur
- Dans un soucis de portabilité, le chemin du dossier de la machine hôte est à renseigner au moment du montage (tous les hôtes n'auront pas le même OS et donc pas la même arborescence)

```
docker run -d -p 8080:80 -v /Users/jean-francois/desktop/test:/var/www/html test
```

Exercice

- Maintenant que ceci est fait, je vous laisse créer votre container et créer plusieurs pages dedans :
 - *Index.php*
 - *Page1.php*

PLUSIEURS CONTENEURS

Et comment ils font pour parler entre eux



Des micro-services


- Sans rentrer dans les détails de l'architecture micro-services, il va souvent être nécessaire de faire communiquer des conteneurs entre eux
- Prenons un exemple simple :
 - *Un serveur web nginx*
 - *Un script php qui affiche des nombres au hasard (ça pourrait tout autant être une base de données)*
- Dans la philosophie Docker, j'ai ici deux applications et donc deux conteneurs, il va donc falloir que je les fasse communiquer ensemble

Mon serveur web

- Je vais commencer par faire mon premier conteneur : mon serveur web
- Ici rien de très compliqué, je vais récupérer l'image officielle Nginx sur DockerHub
- Je vais devoir ensuite déclarer un Volume qui va faire office de pont. Je vais donc utiliser la commande « VOLUME » dans Dockerfile
- Cette commande ne prend que le dossier « exposé » du conteneur, il sera attribué automatiquement un id et un chemin sur la machine hôte lors du lancement du conteneur à l'autre côté du volume

```
serverweb >  Dockerfile
1  # Je pars d'une image officielle
2  FROM nginx
3
4  # Je précise que ce dossier de mon
5  # conteneur sera "partagé"
6  VOLUME /usr/share/nginx/html
7
```

Le worker

```
worker >  Dockerfile
1  # Je pars de l'image officielle
2  FROM php:7.4-cli
3
4  # J'y copie tous mes fichiers
5  COPY . /usr/src/myapp
6
7  # Enfin, lors de l'exécution du
8  # conteneur, je lance mon script
9  WORKDIR /usr/src/myapp
10 CMD [ "php", "./script.php" ]
11 # sera lu "php ./script.php"
12
```

- De l'autre côté, il va me falloir un script PHP qui fera office de worker
- Je vais donc créer une image à partir de l'image officielle PHP qui inclura mon script

Script.php

```
worker >  script.php > ...
```

```
1  <?php
2
3  while (true) {
4      $file = "/usr/share/nginx/html/index.html";
5      $handle = fopen($file, 'w');
6      $data = random_int(0, 100);
7      fwrite($handle, $data);
8      fclose($handle);
9      sleep(1);
10 }
```

- A l'intérieur de mon script.php, je vais faire quelque chose de simple :
 - *Faire une boucle qui s'exécute toutes les secondes*
 - *Cette boucle va créer un fichier index.html à l'adresse de mon volume dans mon serveur web*
 - *Et un mettre un nombre au hasard pour vérifier qu'il tourne bien*

```
# Je crée mes deux images
```

```
docker build -t worker .
```

```
docker build -t serverweb .
```

```
# Je lance le serveur en lui donnant un nom "site"
```

```
# pour éviter de manipuler les id
```

```
docker run -d -p 3333:80 --name site serverweb
```

```
# Enfin, je lance le worker (en lui donnant un nom aussi)
```

```
# en lui demandant d'exploiter le volume du conteneur "site"
```

```
docker run -d --name worker --volumes-from site worker
```

Lancer tout ça

- Voilà, tout est écrit, il ne nous reste plus qu'à lancer tout ça

DOCKER-COMPOSE

Faire des recettes... avec des recettes

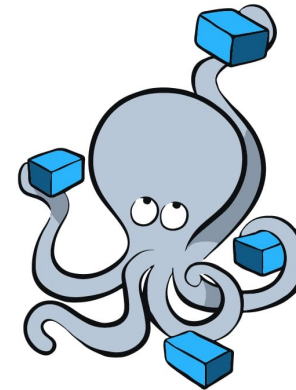


Pourquoi ?

- Comme nous l'avons vu jusque là, il est tout à fait possible de lancer plusieurs conteneurs en même temps et de les faire communiquer ensemble mais c'est tout de suite très laborieux
- C'est là que Docker-compose entre en jeu. Il va nous aider à coordonner tout ça pour faire marcher ensemble plusieurs conteneurs, les galères en moins
- Le rôle de Docker Compose est de coordonner plusieurs images ensemble

Comment ?

- Déjà, bonne nouvelle pour les utilisateurs de Windows ou Mac, Docker Compose est déjà installé avec Docker Desktop
- Pour les utilisateurs de Linux, [je vous renvoie ici pour la procédure](#)
- Ensuite, pour utiliser Docker Compose, il nous suffit de créer un fichier « **docker-compose.yaml** » à la racine de notre dossier projet



docker
Compose

Les commandes de base

```
# Pour construire les images nécessaires sans les lancer  
docker-compose build
```

```
# Pour construire les images et les lancer directement  
docker-compose up
```

```
# La même chose, en mode détaché  
docker-compose up -d
```

```
# Pour stopper tout un stack docker compose  
docker-compose stop
```

```
# Pour détruire l'ensemble des ressources d'un stack  
docker-compose down
```

- Vous ne devriez pas être dépayés avec les commandes de base de Docker Compose, elles sont très similaires à celles de Docker

Les commandes de base (suite)

- Encore d'autres commandes utiles

```
# Pour redémarrer le stack  
docker-compose start
```

```
# Pour connaître l'état des conteneurs  
docker-compose ps
```

```
# Pour supprimer les conteneurs  
docker-compose rm
```

Mon premier docker-compose.yml

- Bon c'est bien beau tout ça mais comment ça marche dans les faits ?
- Un fichier docker-compose commence toujours par la version de Docker Compose que l'on utilise, en ce moment c'est plutôt la version 3
- Ensuite on va déclarer les services que l'on va utiliser, ainsi que le nom qu'on va leur donner et l'image qu'il faut utiliser
- Il ne me reste plus qu'à indiquer les différentes options à appliquer aux images, on trouve toutes les infos directement sur le repository DockerHub de l'image

Un exemple d'un stack WordPress

- Voici l'exemple d'un stack docker-compose pour faire tourner un WordPress
- La clause restart: always va redémarrer le conteneur dès qu'il s'arrête (y compris avec docker-compose stop)

```
1  # J'indique quelle version de docker-compose j'utilise
2  version: '3.1'
3
4  # Je liste ensuite les services que je veux utiliser
5  services:
6      # Le nom du service
7      wordpress:
8          # L'image à utiliser, ici wordpress officiel
9          image: wordpress
10         # La politique de redémarrage en cas d'erreur fatale
11         restart: always
12         # Le mapping de ports, ici le port 8080 de mon hôte
13         # ira vers le port 80 du conteneur
14         ports:
15             - 8080:80
16         # Ensuite les variables d'environnement
17         environment:
18             WORDPRESS_DB_HOST: db
19             WORDPRESS_DB_USER: exampleuser
20             WORDPRESS_DB_PASSWORD: examplepass
21             WORDPRESS_DB_NAME: exampledb
22         # Enfin, pour facilement pérenniser mes données, je vais
23         # lier un volume sur mon hôte à un volume sur mon conteneur
24         # Ca sera plus facile en dev de partager des instances avec leurs
25         # données. Attention cependant, il faudra changer le chemin de fichier
26         volumes:
27             - /Users/jean-francois/desktop/newtest/wordpress:/var/www/html
28
```

Un exemple d'un stack WordPress (suite)

- Il manque aussi une database pour faire fonctionner mon stack WordPress
- Ici j'utilise MariaDB car je suis sur un Mac M1 et l'image MySQL n'est toujours pas compatible

```
29 db:
30   image: mariadb
31   restart: always
32   environment:
33     MYSQL_DATABASE: exampledb
34     MYSQL_USER: exampleuser
35     MYSQL_PASSWORD: examplepass
36     MYSQL_RANDOM_ROOT_PASSWORD: '1'
37   volumes:
38     - /Users/jean-francois/desktop/newtest/datas:/var/lib/mysql
39
```

Docker- compose & Dockerfile

- Les deux fichiers ne sont pas des remplaçants l'un de l'autre, Dockerfile garde quand même quelque chose de très intéressant : ses commandes et particulièrement sa commande RUN
- Imaginons que je veuille faire un stack LAMP avec Apache, MySQL et PHP
- Nous avons remarqué que l'image Docker de PHP n'avait pas toutes les extensions nécessaires comme PDO
- C'est à ce moment que nous allons combiner Dockerfile pour faire une image custom et ensuite docker-compose pour orchestrer le tout

Le Dockerfile

- Commençons par la partie la plus simple : faire un Dockerfile avec une image PHP & Apache à laquelle nous allons rajouter les extensions nécessaires
- Nous l'avons déjà fait tout à l'heure

```
FROM php:7.4-apache
```

```
RUN docker-php-ext-install mysqli pdo pdo_mysql
```

Le docker-compose.yml

- Il faut maintenant dire à notre fichier docker-compose.yml d'utiliser l'image de notre Dockerfile plutôt qu'une image de DockerHub

```
1  version: '3.1'
2
3  services:
4      php:
5          # Pour construire notre image
6          build:
7              # Le chemin relatif pour trouver
8              # le Dockerfile
9              context: .
10             # Et enfin, le nom du Dockerfile
11             dockerfile: Dockerfile
12             # Le reste se passe normalement
13             ports:
14                 - 5555:80
15             volumes:
16                 - ./src:/var/www/html
17
18         db:
19             image: mariadb
20             restart: always
21             environment:
22                 MYSQL_ROOT_PASSWORD: example
23
24         phpmyadmin:
25             image: phpmyadmin
26             restart: always
27             ports:
28                 - 8080:80
29             environment:
30                 - PMA_ARBITRARY=1
```

Lier une DB en PHP

```
// Notez que j'appelle la DB
// avec le nom donné dans docker-compose
$dsn = 'mysql:host=db';
$user = 'root';
$password = 'password';
$pdo = new PDO($dsn, $user, $password);

// Le reste des manipulations est toujours possible
$db = 'demo';
$pdo->exec( statement: "CREATE DATABASE IF NOT EXISTS `{$db}`");
$query = $pdo->query( statement: 'SHOW DATABASES');
print_r($query->fetchAll( mode: PDO::FETCH_ASSOC));
```

- Maintenant que mon stack est créée, je peux le manipuler sans aucun soucis
- En PHP je vais appeler ma DB par le nom que je lui ai donné dans le docker-compose.yaml