



# About Me

Benjamin Vouillaume

`benjamin.m.vouillaume@gmail.com`

- > 6 ans d'expérience avec Docker
  - Petites, moyennes et grosses entreprises

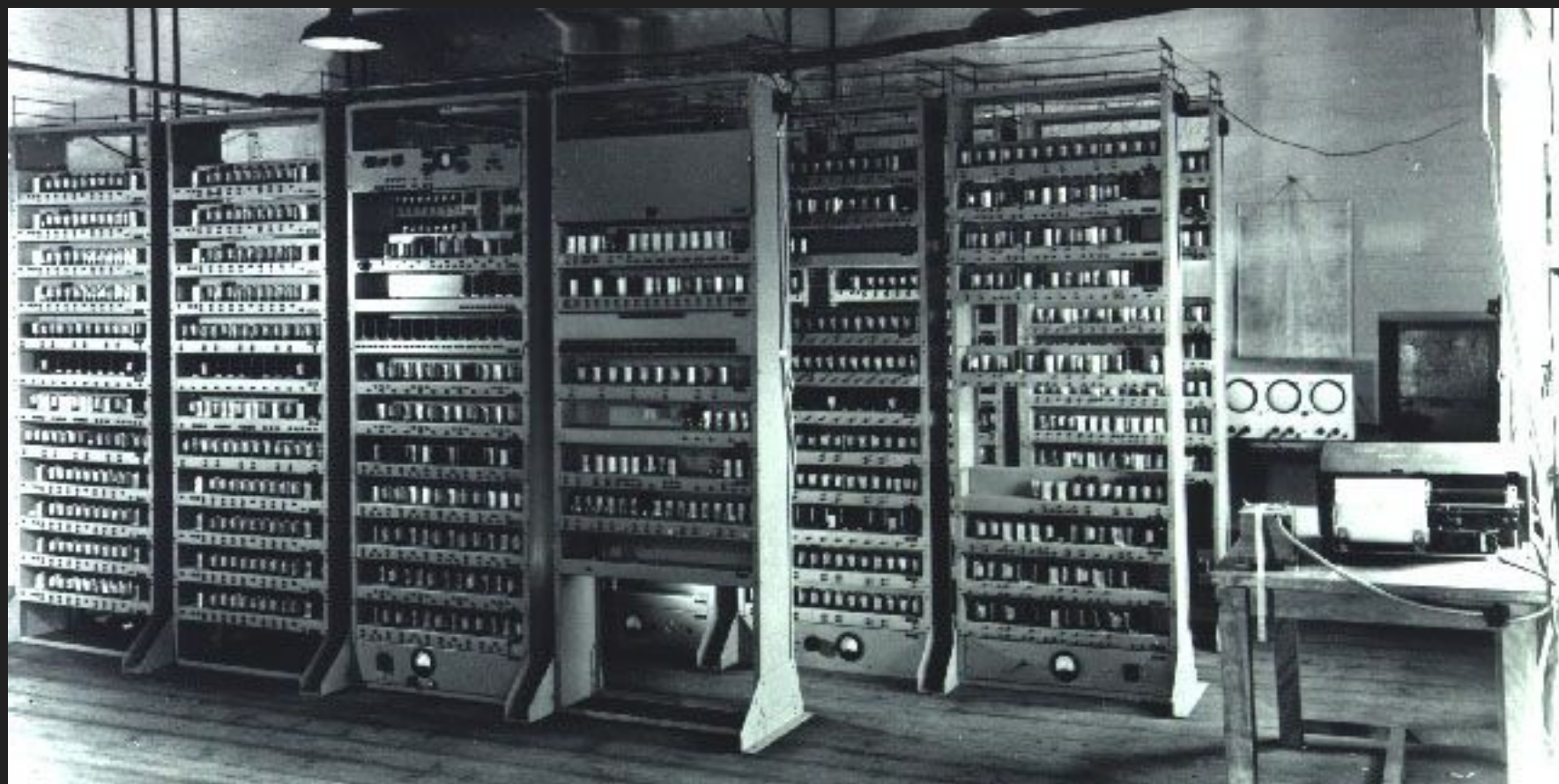
# Introduction

# Agenda

- Théorie de la containerization
  - Focus sur Docker
- Loop
  - Nouveau concept
  - Mise en application
- Objectif
  - Déployer une stack applicative complète avec Docker

# La conteneurisation

# Pourquoi



4GIFs.com







SENORGIF.COM

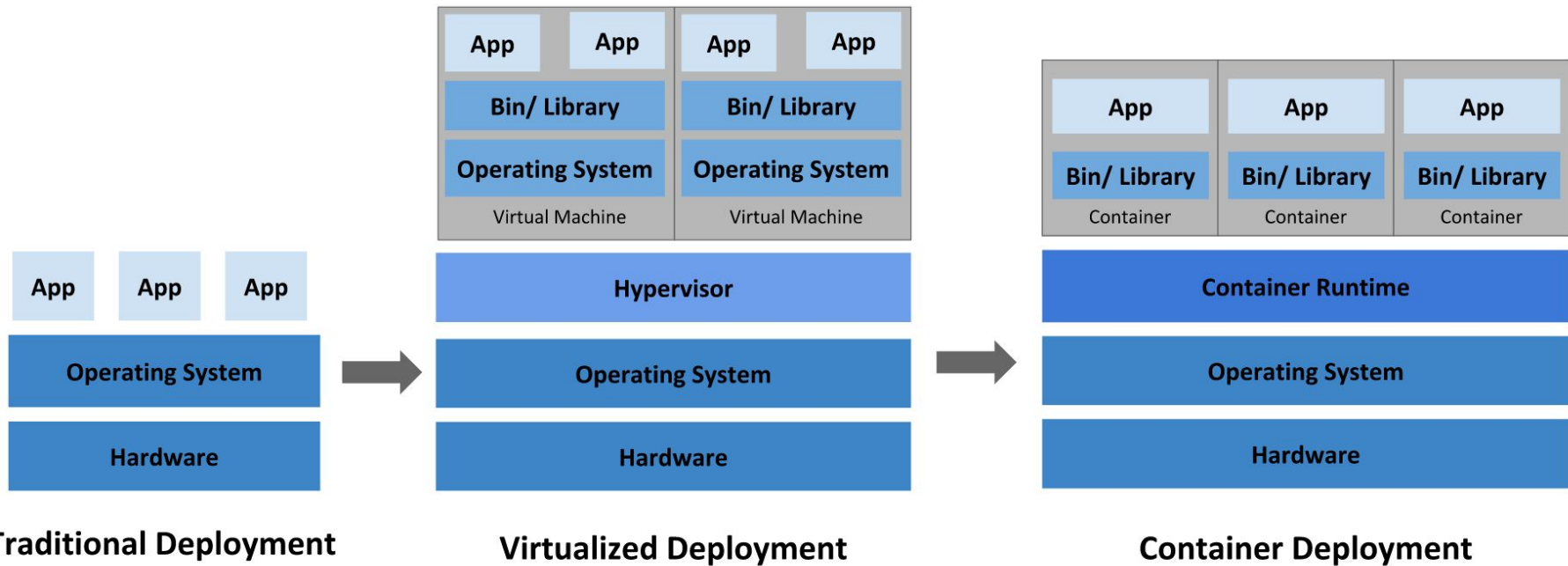












source: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

# La containerization, pourquoi ?

- Réduire, éliminer les problèmes de dépendances  
Pas d'hypothèses sur l'environnement d'exécution !  
L'application containerisée vient avec tous ses dépendances
- Optimiser les ressources utilisées  
Peu d'overhead, démarrage rapide
- Facile à installer, manager et maintenir
- Simplification de l'infrastructure
  - Standardisation

## What's the Diff: VMs vs Containers

VMs	Containers
Heavyweight	Lightweight
Limited performance	Native performance
Each VM runs in its own OS	All containers share the host OS
Hardware-level virtualization	OS virtualization
Startup time in minutes	Startup time in milliseconds
Allocates required memory	Requires less memory space
Fully isolated and hence more secure	Process-level isolation, possibly less secure

Source : <https://www.backblaze.com/blog/vm-vs-containers/>



Comment



# Containerization - Comment

## Isolation des processus

- Seul sur la machine du point de vue du container
- Un processus parmi d'autres du point de vue de la machine

# Containerization - Comment

## Isolation des processus

- Seul sur la machine du point de vue du container
- Un processus parmi d'autres du point de vue de la machine

## Utilisation des namespaces Kernel

- Partitionné, limité, l'utilisation de ressource du kernel
- 7 Namespaces

# La containerization - Comment - Kernel Namespace

- UTS Namespace  
Nom et domaine de la machine
- PID Namespace  
Processus Id de la machine (PID 1)
- Mount Namespace  
Filesystem dédié (chroot)
- Net Namespace  
Stack réseau dédié
- IPC Namespace  
Restreint l'Interprocess Communication

# La containerization - Comment - Kernel Namespace

- User Namespace

User du processus sur la machine (root?)

- Cgroups Namespace

Mesurer, limiter et isoler les ressources physiques (CPU, RAM, Disk I/O, network ...) Features ajouté par Google dans le Kernel

- Est-ce suffisant ?

# La containerization - Comment - Seccomp, AppArmor/SELinux

Dans un système UNIX, tout est fichier

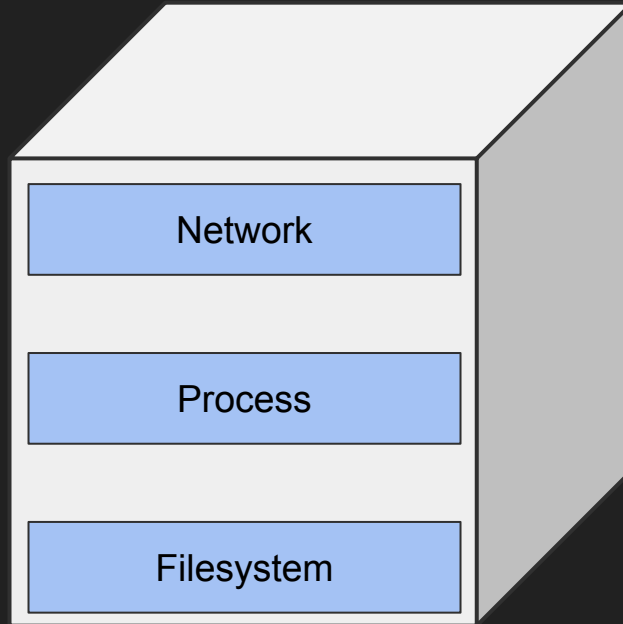
- Les objets Kernel sont des fichiers
- Les Méthodes Kernel sont des fichiers

# La containerization - Comment - Seccomp, AppArmor/SELinux

Dans un système UNIX, tout est fichier

- Les objets Kernel sont des fichiers
- Les Méthodes Kernel sont des fichiers
- Seccomp: Blacklist des appels Kernel
- SELinux/AppArmor: Blacklist des objets Kernel





---

**Kernel UNIX**

# La containerization - Comment

## Le container applicatif

- Le container ne contient qu'un processus à démarrer
- Un process par container (Docker way)

## Le container 'machine'

- Le container contient tout un ensemble de processus à démarrer  
On retrouve le même type de comportement qu'avec une VM, mais dans un container
- Approche plus récente (moins supporté par Docker)

# La containerization - Comment

## Image

- Template de container  
Ensemble constitué d'un filesystem et des configurations permettant de démarrer un container
- C'est l'entité que l'on va construire et distribuer

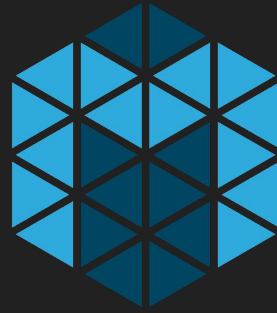
## Container

- C'est une instance en fonctionnement d'une image
- On peut lancer plusieurs containers sur base d'une image
- Les containers ne modifient pas le filesystem de l'image que laquelle ils se basent

# La containerization - Comment

## Container Engine

- L'utilitaire installé sur la machine qui fait le lien avec le Kernel
- Ce qu'on va effectivement installer sur les hosts
- La seule dépendance obligatoire communes pour faire fonctionner nos containers sur nos machines



MESOS



podman



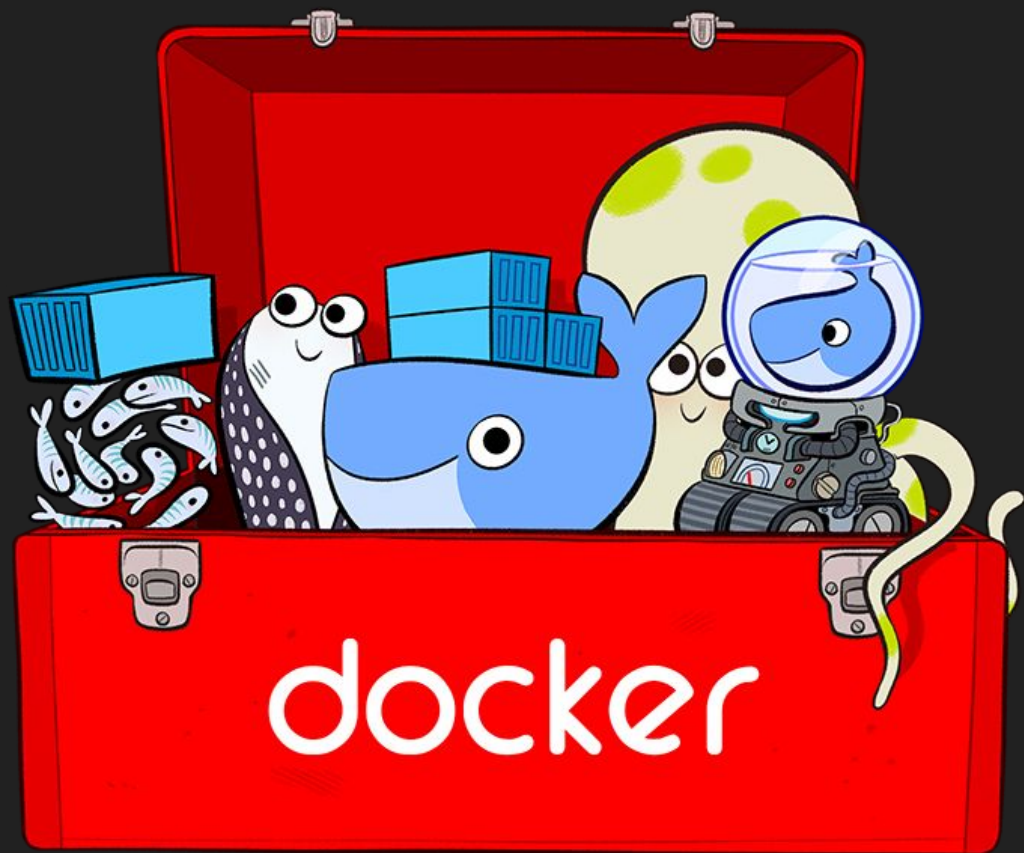


OPEN

CONTAINER  
INITIATIVE

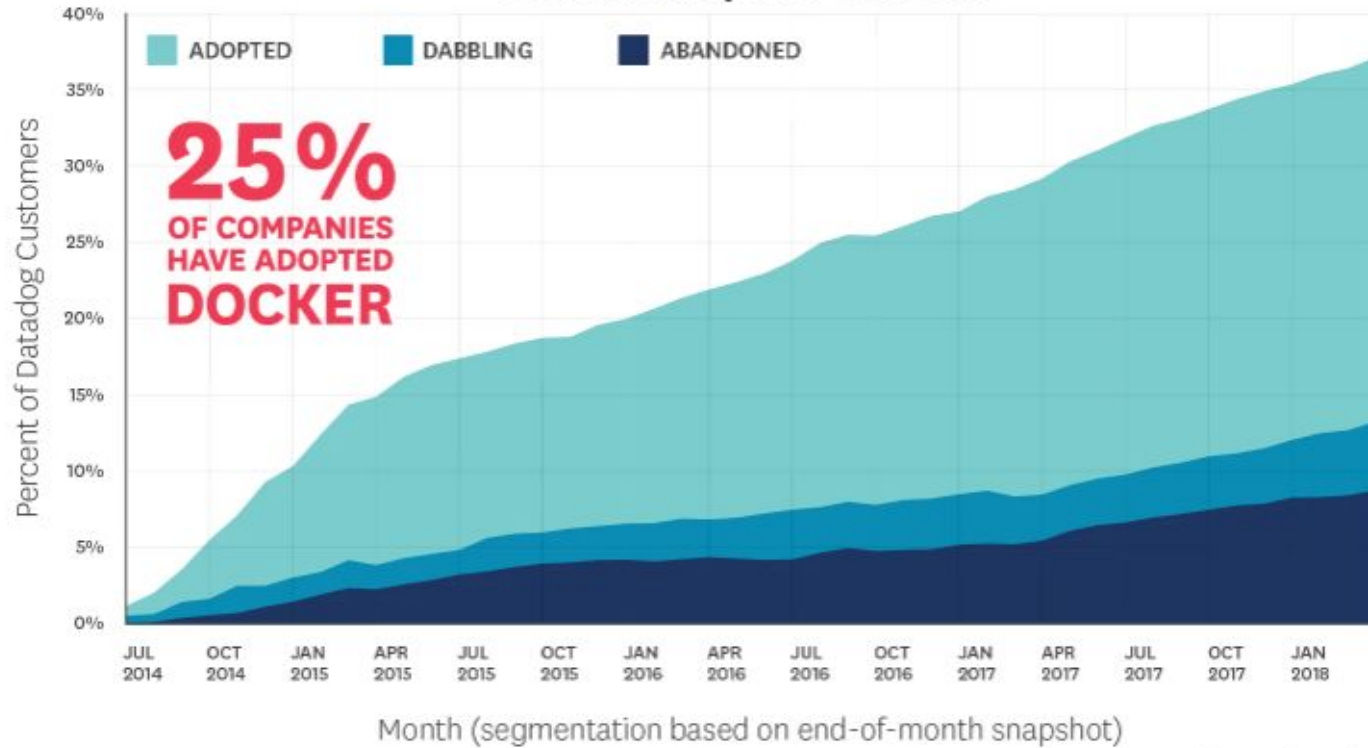






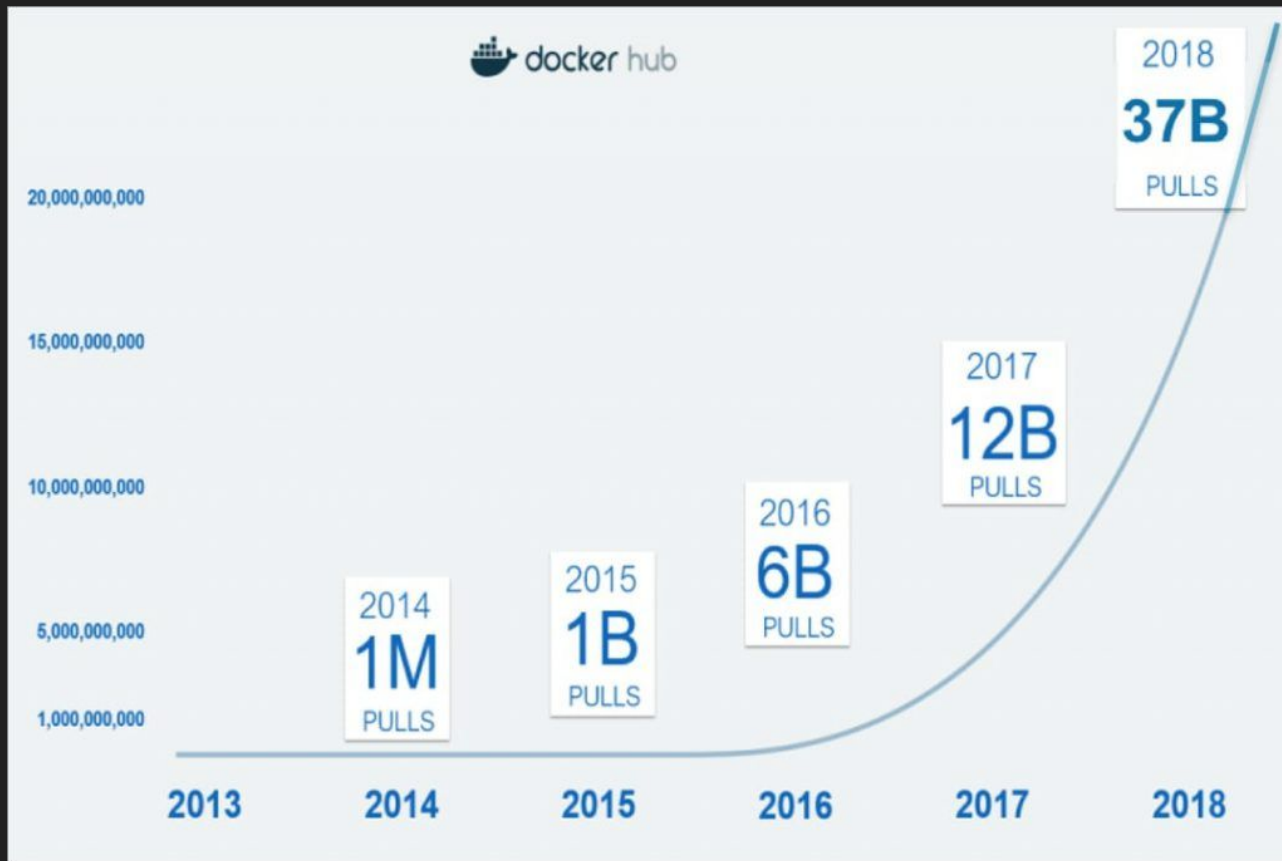


## Docker Adoption Behavior



Source: Datadog

Source: <https://www.datadoghq.com/docker-adoption/>



Source : <https://blog.myagilepartner.fr/index.php/2018/03/24/docker-5-ans-et-toujours-plus-grand/>



## Supported tags and respective Dockerfile links

- `latest` , `centos7` , `7`
- `centos6` , `6`
- `centos7.6.1810` , `7.6.1810`
- `centos7.5.1804` , `7.5.1804`
- `centos7.4.1708` , `7.4.1708`
- `centos7.3.1611` , `7.3.1611`
- `centos7.2.1511` , `7.2.1511`
- `centos7.1.1503` , `7.1.1503`
- `centos7.0.1406` , `7.0.1406`
- `centos6.10` , `6.10`
- `centos6.9` , `6.9`
- `centos6.8` , `6.8`
- `centos6.7` , `6.7`
- `centos6.6` , `6.6`

# Nomenclature

- **Image Officiel**

`<imageName>:<tag>`

`centos:7`

- **Image communautaire**

`<username>/<imageName>:<tag>`

`vbenji/nginx-static-content-config:latest`

# Nomenclature

- **Image Officiel**

`<imageName>:<tag>`

`centos:7`

- **Image communautaire**

`<username>/<imageName>:<tag>`

`vbenji/nginx-static-content-config:latest`

Best Practice: Toujours spécifier la version précise



# Installation de Docker

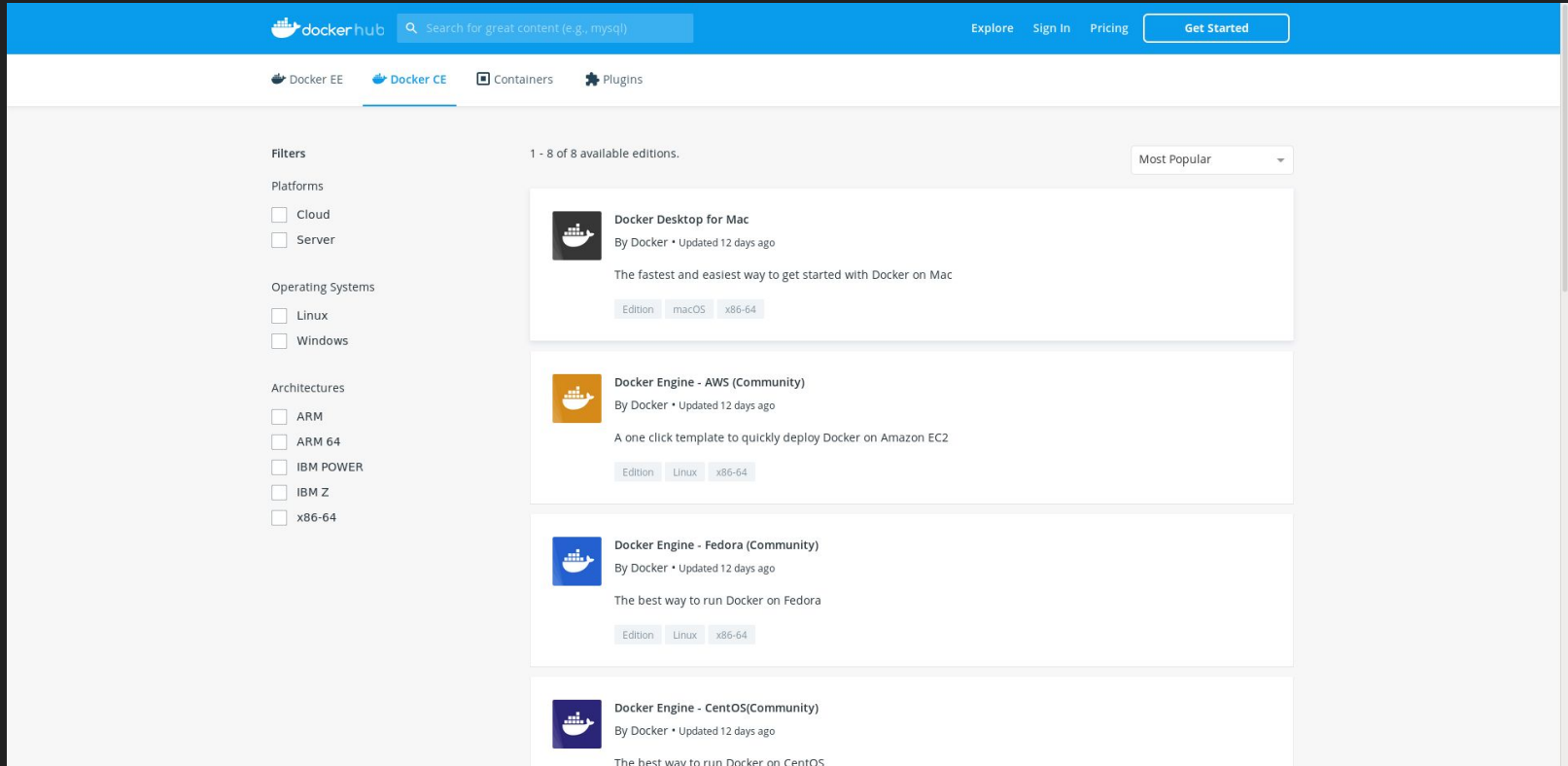
# Installation de Docker

- Docker Engine - Community
- Docker Engine - Enterprise
- Docker Enterprise

# Installation de Docker

- **Docker Engine - Community**
  - Docker Engine - Enterprise
  - Docker Enterprise
- 
- Cloud (AWS & Azure)
  - Linux (Natif) - CentOS, Debian, Fedora, Ubuntu,
  - MacOS (Linux HyperKit VM)
  - Windows 10 (Linux HyperV VM) - Pro, Enterprise, Education

# Installation de Docker - Docker Hub



The screenshot displays the Docker Hub interface. At the top, there is a blue navigation bar with the Docker Hub logo, a search bar, and links for Explore, Sign In, Pricing, and a Get Started button. Below the navigation bar, there are tabs for Docker EE, Docker CE (which is selected), Containers, and Plugins. The main content area shows search results for Docker Desktop for Mac. On the left, there are filters for Platforms (Cloud, Server), Operating Systems (Linux, Windows), and Architectures (ARM, ARM 64, IBM POWER, IBM Z, x86-64). The search results list four items: Docker Desktop for Mac, Docker Engine - AWS (Community), Docker Engine - Fedora (Community), and Docker Engine - CentOS (Community). Each item includes a Docker logo icon, the title, the author (By Docker), the update time (Updated 12 days ago), a description, and a list of available editions (Edition, macOS, x86-64 for the first item; Edition, Linux, x86-64 for the others).

**Filters**

1 - 8 of 8 available editions. Most Popular

**Platforms**

- ☐ Cloud
- ☐ Server

**Operating Systems**

- ☐ Linux
- ☐ Windows

**Architectures**

- ☐ ARM
- ☐ ARM 64
- ☐ IBM POWER
- ☐ IBM Z
- ☐ x86-64

**Docker Desktop for Mac**  
By Docker • Updated 12 days ago  
The fastest and easiest way to get started with Docker on Mac  
Edition macOS x86-64

**Docker Engine - AWS (Community)**  
By Docker • Updated 12 days ago  
A one click template to quickly deploy Docker on Amazon EC2  
Edition Linux x86-64

**Docker Engine - Fedora (Community)**  
By Docker • Updated 12 days ago  
The best way to run Docker on Fedora  
Edition Linux x86-64

**Docker Engine - CentOS (Community)**  
By Docker • Updated 12 days ago  
The best way to run Docker on CentOS

# Installation de Docker

→ ~ docker version

Client: Docker Engine - Community

Version: 19.03.2

API version: 1.40

Go version: go1.12.8

Git commit: 6a30dfc

Built: Thu Aug 29 05:29:33 2019

OS/Arch: linux/amd64

Experimental: false

Server: Docker Engine - Community

Engine:

Version: 19.03.2

API version: 1.40 (minimum version 1.12)

Go version: go1.12.8

Git commit: 6a30dfc

Built: Thu Aug 29 05:28:12 2019

OS/Arch: linux/amd64

Experimental: false

containerd:

Version: 1.2.6

GitCommit:

894b81a4b802e4eb2a91d1ce216b8817763c29fb

runc:

Version: 1.0.0-rc8

GitCommit:

425e105d5a03fabd737a126ad93d62a9eeede87f

docker-init:

Version: 0.18.0

GitCommit: fec3683

→ ~ docker info

# Containers

# Récupération de l'image

```
→ ~ docker pull docker/whalesay
Using default tag: latest
latest: Pulling from docker/whalesay
e190868d63f8: Pull complete
909cd34c6fd7: Pull complete
0b9bfabab7c1: Pull complete
a3ed95caeb02: Pull complete
00bf65475aba: Pull complete
c57b6bcc83e3: Pull complete
8978f6879e2f: Pull complete
8eed3712d2cf: Pull complete
Digest: sha256:178598e51a26abbc958b8a2e48825c90bc22e641de3d31e18aaf55f3258ba93b
Status: Downloaded newer image for docker/whalesay:latest
docker.io/docker/whalesay:latest
```

## Lancer un container sur base de l'image

```
➔ ~ docker run docker/whalesay cowsay Hello INTECH
```

< Hello INTECH >

-----

##

##    ##    ##

$$\frac{1}{2} \frac{d}{dt} \left( \frac{1}{2} \frac{d}{dt} \right)$$

## ## ## ##

=====	=====	=====
=====	=====	=====

[illegible]
$$\sim \sim \sim \left\{ \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \right\} / \equiv \equiv \equiv - \sim \sim \sim$$



# Docker run --help

## **--name**

Donne un nom au container

Autogénéré sinon

## **--rm**

Supprime le container dès qu'il  
s'arrête

## Docker run --help

- Exécuter un container en mode interactif
  - -i se connecte au STDIN du container (--interactive)
  - -t pour avoir un pseudo-terminal (--tty)
- Nécessite d'avoir un interpréteur de commande dans l'image
- Nécessite d'avoir pensé à mettre les commandes utiles dans l'image

```
→ ~ docker run --rm -it docker/whalesay sh
# whoami
root
# exit
→ ~
```

# Containers

```
→ ~ docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
de65f5a0083a	docker/whalesay	"cowsay Hello INTECH"	2 seconds ago	Exited (0)	2 seconds ago
gracious_leavitt					

CONTAINER ID : identifiant du container

IMAGE : identifiant et version de l'image utilisée

COMMAND : commande passée à la création du container

CREATED : date de création du container

STATUS : état du container

PORTS : ports redirigés entre container et hôte

NAMES : noms du container

# Containers

```
→ ~ docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
→ ~ docker container ls -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
de65f5a0083a   docker/whalesay "cowsay Hello INTECH"   20 minutes ago Exited (0)    20 minutes ago          gracious_leavitt
```

- `docker container ls [options]`
  - `-a` renvoie aussi les containers arrêtés (penser à purger régulièrement)
  - `-q` renvoie uniquement les ID (automatisation)
  - `-n=N` affiche les N derniers containers créés

# Container and processus

- La durée de vie d'un container dépend du statut du processus avec le PID 1
  - Tant que ce processus fonctionne, le container fonctionne

Il va donc est important de bien créer nos images pour mettre le processus qui nous importe en PID 1

```
→ ~ docker run -it busybox ps -ef
PID   USER        TIME  COMMAND
    1   root         0:00  ps -ef
→ ~ docker run -it busybox top
```

# Container background execution

```
→ ~ docker run busybox sh -c "while true; do $(echo date); sleep 1; done"  
Sun Sep  8 16:21:24 UTC 2019  
Sun Sep  8 16:21:25 UTC 2019
```

# Container background execution

```
→ ~ docker run busybox sh -c "while true; do $(echo date); sleep 1; done"
Sun Sep  8 16:21:24 UTC 2019
Sun Sep  8 16:21:25 UTC 2019
→ ~ docker run -d busybox sh -c "while true; do $(echo date); sleep 1; done"
44c46e804f38401d392810c7e3ce5c2c414fe74305abd61c34700f7572609352
→ ~ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
44c46e804f38	busybox	"sh -c 'while true; ...'"	18 seconds ago	Up 17 seconds	
zen_jackson					

- Chaque container possède un ID long et court
- On peut identifier un container par ses ID ou son nom

# Container logs

```
→ ~ docker logs zen_jackson  
Sun Sep  8 16:27:05 UTC 2019  
Sun Sep  8 16:27:06 UTC 2019  
Sun Sep  8 16:27:07 UTC 2019
```

- Permet de voir les logs d'un container
  - -f (--follow) pour les avoir en direct
  - --tail N pour n'avoir que les N dernières entrées



# Quelques commandes

```
→ ~ docker stop zen_jackson
→ ~ docker start zen_jackson
→ ~ docker rm -f zen_jackson
→ ~ docker rm gracious_leavitt
```

```
→ ~ docker image ls busybox
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
busybox	latest	19485c79a9bb	3 days ago	1.22MB

```
→ ~ docker image ls postgres
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
postgres	9.6	f5548544c480	3 weeks ago	230MB
postgres	10	47d49eb910f3	2 months ago	230MB

```
→ ~ docker image rm busybox
```

```
→ ~ docker container inspect zen_jackson
```

```
→ ~ docker <object> prune
```

Image

# Faire une image Docker - Méthode dépréciée

```
→ ~ docker run -it centos bash
[root@bcfa4bc48719 /]# touch INTECH
[root@bcfa4bc48719 /]# exit
exit
→ ~ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
bcfa4bc48719	centos	"bash"	28 seconds ago	Exited (0) 19 seconds ago	
priceless_beaver					

```
→ ~ docker commit bcfa4bc48719
sha256:220b02b5022b58b940d66f49fccdf2b771d38546a49e4d6933389a88251363e7
→ ~ docker image ls 220b02b5022b
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
220b02b5022b				

```
→ ~ docker run -it 220b02b5022b bash
[root@13e109e90aba /]# ls
INTECH      bin  etc   lib   media  opt   root  sbin  sys  usr
anaconda-post.log  dev  home  lib64 mnt    proc  run   srv   tmp  var
[root@13e109e90aba /]# exit
exit
→ ~
```

# Dockerfile

# Dockerfile

- Nom du fichier par défaut : **Dockerfile**
- Suite d'instruction pour automatiser la création d'image  
Plus besoin de tout faire à la main et d'enregistrer son image

# Dockerfile

- Nom du fichier par défaut : **Dockerfile**
- Suite d'instruction pour automatiser la création d'image  
Plus besoin de tout faire à la main et d'enregistrer son image

## Avantages :

- Automatisable
- Reproductible
- Maintenable
- Rapide

# Dockerfile

```
FROM alpine
```

```
CMD echo "Hello INTECH"
```

```
→ ~ docker build --tag hello-intech .
```

# Dockerfile

```
→ ~ more Dockerfile
```

```
FROM centos:7
```

```
CMD echo "Hello INTECH"
```

```
→ ~ docker build --tag hello-INTECH .
```

```
Sending build context to Docker daemon 689.5MB
```

```
Step 1/2 : FROM centos:7
```

```
7: Pulling from library/centos
```

```
Digest:
```

```
sha256:307835c385f656ec2e2fec602cf093224173c51119bbeb  
d602c53c3653a3d6eb
```

```
Status: Downloaded newer image for centos:7
```

```
---> 67fa590cfc1c
```

```
Step 2/2 : CMD echo "Hello INTECH"
```

```
---> Running in 70e049fa5feb
```

```
Removing intermediate container 70e049fa5feb
```

```
---> 013bfc688049
```

```
Successfully built 013bfc688049
```

```
Successfully tagged hello-INTECH:latest
```

```
→ ~ docker run hello-INTECH
```

```
Hello INTECH
```

```
→ ~
```



# Dockerfile

→ ~ more Dockerfile

```
FROM fedora
```

```
ENV TERM=xterm
```

```
RUN dnf install sl -y
```

```
CMD /usr/bin/sl
```

→ ~ docker build --tag sl .

→ ~ docker run --rm sl

# Dockerfile - Instruction

- FROM  
Permet de définir l'image 'de base' sur laquelle va se baser notre propre image  
Ou FROM scratch, pour partir d'un filesystem vierge
- RUN  
Exécute une commande lors du build de l'image
- LABEL  
Permet de définir des metadatas pour l'image
- ENV  
Définit une variable d'environnement

## Dockerfile - Instruction

- COPY

Permet d'ajouter un fichier du contexte de build dans l'image

- ADD

Permet d'ajouter un fichier du contexte de build ou depuis une URL dans l'image

Si ce fichier est un tgz, ADD va automatiquement l'extraire

## Dockerfile - Contexte de build

Le dernier paramètre de build, (souvent '.') est ce qu'on appelle le contexte de build. Il s'agit d'un chemin vers un répertoire, qui sera le répertoire avec lequel le build pourra interagir

Il est inutile et contre performant de charger dans le contexte de build des fichiers que l'on n'ajoutera pas dans l'image

Les chemins de COPY et ADD en relatif se baseront sur le chemin de ce contexte.

Fichier d'exclusion '.dockerignore' (semblable à .gitignore)

# Dockerfile - Instruction

- COPY  
Permet d'ajouter un fichier du contexte de build dans l'image
- ADD  
Permet d'ajouter un fichier du contexte de build ou depuis une URL dans l'image  
Si ce fichier est un tgz, ADD va automatiquement l'extraire
- WORKDIR  
Définit un répertoire de travail dans l'image  
Créer ce répertoire s'il n'existe pas  
Impact RUN, CMD et ENTRYPOINT qui suivront

# Dockerfile - Instruction

- ARG  
Définit une variable utilisable dans le Dockerfile et donnée en option de build (--build-arg)
- USER  
Définit le user à utiliser pour RUN, CMD et ENTRYPOINT  
Ne créer pas le user s'il n'existe pas
- VOLUME /foo/bar  
Informe que le container utilise ce répertoire
- EXPOSE 80  
Informe que le container utilise ce port

# Dockerfile - Instruction

- ENTRYPOINT  
Définit la commande par défaut à exécuter dans le container
- CMD  
Définit la commande une commande à exécuter  
Ou les paramètres de l'Entrypoint

Possible de faire l'un, l'autre ou les deux en fonction des usages.

Deux formats:

- Shell: CMD echo INTECH
- Exec: CMD ["echo", "INTECH"] - Best Practice

## Dockerfile - Instruction

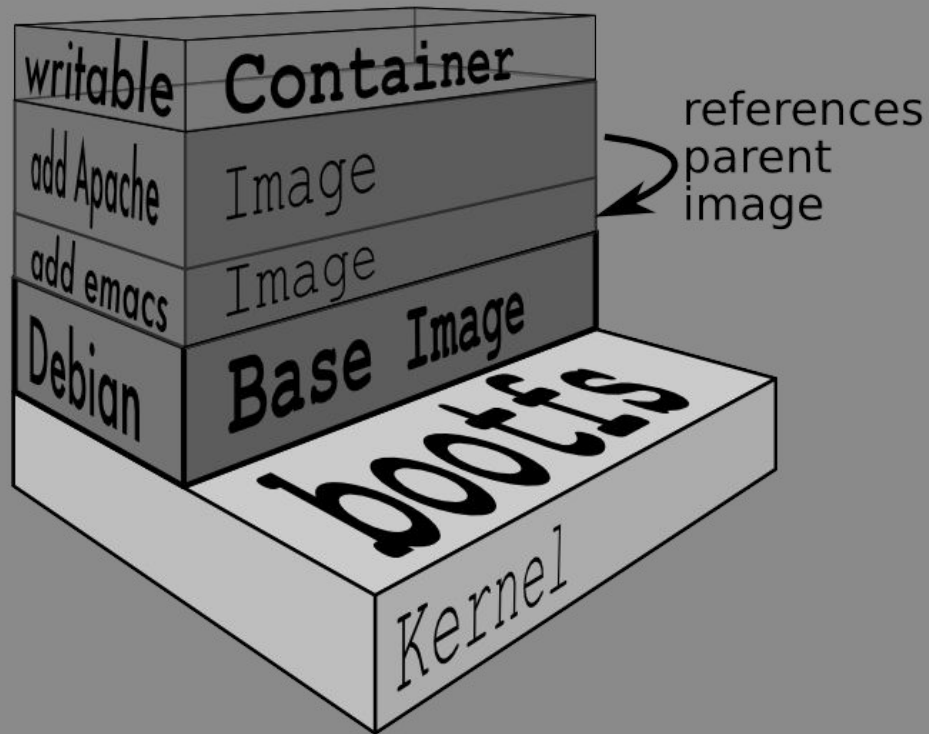
Chaque instruction crée un container, effectue une action dans ce container et commit le résultat pour faire une image qui servira de base à l'instruction suivante, ce sont des layers

Il existe un système de cache pour chaque layer lors des builds, l'ordre des étapes peut donc être important

Il faut placer les instructions stables au début du Dockerfile

Et cumuler les commandes RUN avec &&





# Dockerisation

# Dockerisation - Python

- <https://github.com/damnee562/shopping-cart>
  - python
  - django rest framework

# Dockerisation - Python

1. Quel est notre environnement d'exécution ?

-> Python 2

```
FROM python:2
```

# Dockerisation - Python

2. De quoi avons nous besoin dans notre image ?

-> Notre projet

3. Où le placer ?

-> Répertoire dédié

```
WORKDIR /usr/src/app  
COPY . .
```

# Dockerisation - Python

4 . Que dois-je faire maintenant avec mes sources

-> Télécharger les dépendances

`RUN pip install --no-cache-dir -r requirements.txt`

# Dockerisation - Python

5. Que dois-je faire maintenant avec mes sources et mes dépendances ?

-> Démarrer le serveur d'API

CMD ["python", "manage.py", "runserver"]

# Dockerisation - Python

```
→ backend git:(master) ✕ more Dockerfile
FROM python:2

WORKDIR /usr/src/app

COPY . .

RUN pip install -r requirements.txt

CMD ["python", "manage.py", "runserver"]

→ backend git:(master) ✕ docker build -t shopping-cart:1.0 .
.
.
.
Successfully tagged shopping-cart:1.0
→ backend git:(master) ✕ docker run shopping-cart:1.0
```



# Dockerisation

- Java
  - <https://github.com/spring-projects/spring-petclinic>
- nodejs
  - <https://github.com/jsmuster/angular-node-express-api>
- React
- Angular
  - <https://github.com/jsmuster/angular-node-express>
- ...

# Utilisation direct d'image Docker

# Utilisation direct d'image Docker

- postgres
  - [https://hub.docker.com/\\_/postgres](https://hub.docker.com/_/postgres)
  - 9.6

```
→ ~ docker run -d postgres:9.6
af923acdabb393a5df1f64eb605f76a2ffadd89bc66e35e739fda5ce42236fca
```

```
→ ~ git:(master) ✗ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
af923acdabb3	postgres:9.6	"docker-entrypoint.s..."	4 seconds ago	Up 3 seconds	
5432/tcp	funny_khorana				

```
→ ~ git:(master) ✗ docker rm -f af923acdabb3
af923acdabb3
```

## Configuration nécessaire !

# Variables d'environnement

# Variables d'environnement

- Dockerfile
  - ENV
- Commandline
  - --env / -e

# Variables d'environnement

- Dockerfile
  - ENV
- Commandline
  - --env / -e

## Exemple avec postgres

- POSTGRES\_DB
- POSTGRES\_USER
- POSTGRES\_PASSWORD

```
➔ ~ docker run -d --env POSTGRES_DB=myDatabase --env POSTGRES_USER=databaseUser  
--env POSTGRES_PASSWORD=databasePassword postgres:9.6
```

# Docker exec

# docker exec

Permet de lancer des commandes dans un container en exécution à côté du processus principal

```
→ ~ docker exec condescending_meitner ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
postgres     1      0   0 08:26 ?        00:00:00 postgres
postgres    63      1   0 08:26 ?        00:00:00 postgres: checkpointer process
postgres    64      1   0 08:26 ?        00:00:00 postgres: writer process
postgres    65      1   0 08:26 ?        00:00:00 postgres: wal writer process
postgres    66      1   0 08:26 ?        00:00:00 postgres: autovacuum launcher process
postgres    67      1   0 08:26 ?        00:00:00 postgres: stats collector process
root       114      0   0 08:32 ?        00:00:00 ps -ef
```



# docker exec

Possibilité de lancer ces commandes supplémentaires en interactif comme pour la command docker run

```
→ ~ docker exec -it condescending_meitner bash
root@8a2cbeceb0809:/# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
postgres     1      0  0 08:26 ?           00:00:00 postgres
postgres    63      1  0 08:26 ?           00:00:00 postgres: checkpointer process
postgres    64      1  0 08:26 ?           00:00:00 postgres: writer process
postgres    65      1  0 08:26 ?           00:00:00 postgres: wal writer process
postgres    66      1  0 08:26 ?           00:00:00 postgres: autovacuum launcher process
postgres    67      1  0 08:26 ?           00:00:00 postgres: stats collector process
root        123     0  2 08:34 pts/0       00:00:00 bash
root        130    123  0 08:34 pts/0       00:00:00 ps -ef
root@8a2cbeceb0809:/# exit
exit
→ git
```

# docker exec

## Vérifions notre base de données postgres

```
➔ ~ docker exec -it condescending_meitner bash
root@8a2cbeceb0809:/# psql -U databaseUser -W myDatabase
Password for user databaseUser:
psql (9.6.15)
Type "help" for help.

myDatabase=# \q
root@8a2cbeceb0809:/# exit
exit
➔ ~
```

Comment y accéder ?

# Docker inspect

# docker inspect

```
→ ~ docker inspect 8a2cbeeb0809
[
  {
    "Id": "8a2cbeeb0809537f74870de9e2b9a4758ca0dd0553fbb7d0ee8f465527460fe8",
    "Created": "2019-09-11T08:26:51.879370828Z",
    "Path": "docker-entrypoint.sh",
    "Args": [
      "postgres"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 11221,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2019-09-11T08:26:52.336428713Z",
```

# docker inspect

```
"NetworkSettings": {
  "Bridge": "",
  "SandboxID": "18ff1787dd5b9adb0be12c78bbec68c478141430a45d1c0c63e308e7c5f205c6",
  ...
  "Ports": {
    "5432/tcp": null
  },
  ...
  "Gateway": "172.17.0.1",
  "IPAddress": "172.17.0.2",
  "IPPrefixLen": 16,
  "IPv6Gateway": "",
  "MacAddress": "02:42:ac:11:00:02",
  "Networks": {
    "bridge": {
      "Gateway": "172.17.0.1",
      "IPAddress": "172.17.0.2",
      "IPPrefixLen": 16,
      "MacAddress": "02:42:ac:11:00:02",
      ...
    }
  }
}
```

# PORTS

→ ~ docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
8a2cbecb0809	postgres:9.6	"docker-entrypoint.s..."	24 minutes ago	Up 24 minutes	5432/tcp
condescending_meitner					

# Mapping de ports

# Mapping de ports

- Chaque container possède sa propre stack réseau
  - Avec sa propre adresse IP dans son propre sous réseau



# Mapping de ports

- Chaque container possède sa propre stack réseau
  - Avec sa propre adresse IP dans son propre sous réseau
- Pour accéder au container on peut mapper des ports du container sur des ports du host
- `docker run -p host:container ...`

```
➔ ~ docker run -d -p 5432:5432 --env POSTGRES_DB=myDatabase --env POSTGRES_USER=databaseUser  
--env POSTGRES_PASSWORD=databasePassword postgres:9.6
```

```
➔ ~ docker ps
```

CONTAINER ID NAMES	IMAGE	COMMAND	CREATED	STATUS	PORTS
20762e0a5885	postgres:9.6	"docker-entrypoint.s..."	2 seconds ago	Up 1 second	
0.0.0.0:5432->5432/tcp	recursing_meitner				

```
➔ ~
```

# Mapping de ports

- EXPOSE
  - Information sur le port utilisé par le container
  - Visible avec `docker container ls`
  - Permet un mapping automatique
- `docker run -P ...`

```
→ ~ docker run -d -P --env POSTGRES_DB=myDatabase --env POSTGRES_USER=databaseUser
--env POSTGRES_PASSWORD=databasePassword postgres:9.6
38499d23789258a3e6dc05c96efe288ae06ab842b83dd5d7605a0a1c61500bba
```

```
→ ~ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
38499d237892	postgres:9.6	"docker-entrypoint.s..."	3 seconds ago
Up 2 seconds	0.0.0.0:32768->5432/tcp	gallant_rosalind	

# Mapping de ports

```
➔ ~ docker run -d -P --env POSTGRES_DB=myDatabase --env POSTGRES_USER=databaseUser  
--env POSTGRES_PASSWORD=databasePassword postgres:9.6
```

```
38499d23789258a3e6dc05c96efe288ae06ab842b83dd5d7605a0a1c61500bba
```

```
➔ ~ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
38499d237892	postgres:9.6	"docker-entrypoint.s..."	3 seconds ago
Up 2 seconds	0.0.0.0:32768->5432/tcp	gallant_rosalind	

```
➔ ~ curl localhost:5432
```

```
curl: (52) Empty reply from server
```

# Configuration serveur d'API

# Dockerfile

- EXPOSE 8000

```
Dockerfile > ...  
1  FROM python:2  
2  
3  WORKDIR /usr/src/app  
4  
5  COPY . .  
6  
7  RUN pip install -r requirements.txt  
8  
9  EXPOSE 8000  
10  
11  CMD ["python", "manage.py", "runserver"]  
12  |
```

# Configuration pour postgres

shopping-cart/backend/backend/settings.py

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql_psycopg2',  
        'NAME': '',  
        'USER': '',  
        'PASSWORD': '',  
        'HOST': '',  
        'PORT': ''  
    }  
}
```

# Configuration pour postgres

```
shopping-cart/backend/backend/settings.py
```

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql_psycopg2',  
        'NAME': 'myDatabase',  
        'USER': 'databaseUser',  
        'PASSWORD': 'databasePassword',  
        'HOST': '172.17.0.1',  
        'PORT': '5432'  
    }  
}
```

```
→ ~ ip addr
```

```
...
```

```
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default  
    link/ether 02:42:56:35:1a:19 brd ff:ff:ff:ff:ff:ff  
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
```

# Configuration pour postgres

```
→ backend git:(master) X docker run -d -p 5432:5432 --env POSTGRES_DB=myDatabase --env POSTGRES_USER=databaseUser --env POSTGRES_PASSWORD=databasePassword postgres:9.6
→ backend git:(master) X docker run -p 8000:8000 shopping-cart:1.0
```

```
→ backend git:(master) X curl localhost:8000
curl: (56) Recv failure: Connection reset by peer
→ backend git:(master) X docker exec -it 8f7487c11f70 sh
# curl localhost:8000

<!DOCTYPE html>
<html lang="en">
<head>
```

```
→ backend git:(master) X docker run -p 8000:8000 shopping-cart:1.0
Not Found: /
[11/Sep/2019 11:01:00] "GET / HTTP/1.1" 404 2480
```



# Configuration du server d'API

Par défaut, la commande runserver expose localhost:8000.

Nous ne sommes pas sur localhost

Il faut donc exposer sur le bon network, ou tous les networks  
0.0.0.0:8000

```
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

# Test du serveur d'API

- `http://localhost:8000/api`

Django REST framework

Log in

Api Root

Api Root

OPTIONS GET

The default basic root view for DefaultRouter

GET /api/

HTTP 200 OK

Allow: GET, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "products": "http://localhost:8000/api/products/",
  "orders": "http://localhost:8000/api/orders/",
  "users": "http://localhost:8000/api/users/"
}
```

# Externalisation de la configuration

# Externalisation de la configuration

- Nombreuses méthodes
  - Serveur de configurations
  - Fichiers
  - Stockage clés/valeurs distribuées
  - ...
  - Variables d'environnement

# Externalisation de la configuration

```
databaseName = os.environ['DATABASE_NAME']
databaseUser = os.environ['DATABASE_USER']
databasePassword = os.environ['DATABASE_PASSWORD']
databaseHost = os.environ['DATABASE_HOST']
databasePort = os.environ['DATABASE_PORT']
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': databaseName,
        'USER': databaseUser,
        'PASSWORD': databasePassword,
        'HOST': databaseHost, # host.docker.internal
        'PORT': databasePort
    }
}
```

# Externalisation de la configuration

```
→ backend git:(master) X docker run -e DATABASE_HOST=172.17.0.1 -e DATABASE_PORT=5432 -e  
DATABASE_USER=databaseUser -e DATABASE_PASSWORD=databasePassword -e DATABASE_NAME=myDatabase -p  
8000:8000 shopping-cart:1.0
```

```
→ backend git:(master) X curl http://127.0.0.1:8000/api/
```

# Création des tables dans la database

# Création des tables dans la database

- Manuellement
  - docker exec avec la commande 'python manage.py migrate'
  - docker exec dans la postgre via psql
- > Mauvaise pratique



# Création des tables dans la database

- Manuellement
  - docker exec avec la commande 'python manage.py migrate'
  - docker exec dans la postgre via psql
  - > Mauvaise pratique
- Scripts d'initialisation dans la container postgre
  - répétabilité
  - automatisation
  - > Bonne pratique

# Création des tables dans la database

- Manuellement
  - docker exec avec la commande 'python manage.py migrate'
  - docker exec dans la postgre via psql
  - > Mauvaise pratique
- Scripts d'initialisation dans la container postgre
  - répétabilité
  - automatisation
  - > Bonne pratique
- Association Entrypoint & CMD

# ENTRYPOINT & CMD

Possible d'utiliser ces deux instructions pour effectuer des actions à chaque démarrage

Tout en conservant notre processus applicatif sur le PID1


On passe par un script d'entrypoint

```
ENTRYPOINT [ "./entrypoint.sh" ]
```

```
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

# ENTRYPOINT & CMD

- entrypoint.sh

```
shopping-cart > backend >  entrypoint.sh
1  #!/bin/sh
2
3  python /usr/src/app/manage.py migrate
4
5  exec "$@"
```

- Exécution de l'entrypoint, puis passage au CMD avec association au PID1 via exec "\$@"
- Est exécuté à chaque démarrage du container
- chmod +x entrypoint.sh

# Volume

# Volume

- Simulation d'une perte de notre container postgres
- Exécution d'un nouveau container postgre
- <http://127.0.0.1:8000/api/products/>
- Plus de table products dans la base
  - Plus de données du tout dans la base
- Redémarrage du container Serveur d'API
- <http://127.0.0.1:8000/api/products/>
- Présence de la table products
  - Mais pas des datas

# Volume

- Mapping d'un répertoire du host avec un répertoire du container
  - `-v <hote path>:<container path>:<right>`
  - Chemin absolu
  - Mode read-only `:ro` - Mode read write `:rw`
  - Usage traditionnel avec un montage NFS sur le host

```
→ ~ echo "INTECH" > /tmp/test.txt
→ ~ docker run -it -v /tmp:/host_tmp:ro busybox sh
/ # more /host_tmp/test.txt
INTECH
/ # exit
```

# Volume

- Docker volume
  - Plusieurs Drivers
    - Local: Filesystem du host, mais géré par Docker
    - Autres: Spécifiques en fonction de la solution de stockage



# Volume

- Docker volume
  - Plusieurs Drivers
    - Local: Filesystem du host, mais géré par Docker
    - Autres: Spécifiques en fonction de la solution de stockage
- Volume local
  - -v <nom du volume>:<container path>:<right>

```
→ ~ docker volume create myVolume
myVolume
→ ~ docker volume ls -f name=myVolume
DRIVER      VOLUME NAME
local       myVolume
```

# Volume

```
➔ ~ docker run -it -v myVolume:/volume:rw busybox sh
/ # cd /volume/
/volume # touch INTECH
/volume # ls
INTECH
/volume # exit
➔ ~ docker run -it -v myVolume:/volume:rw python:2 sh
# cd /volume
# ls
INTECH
```

```
➔ ~ docker run -it -v myVolume:/volume:rw busybox
sh
/ # cd /volume/
/volume # ls
INTECH
/volume # exit
➔ ~
```

# Volume

- Postgres
  - Répertoire de stockage: /var/lib/postgresql/data

```
→ ~ docker volume create postgres
postgres
→ ~ docker run --name postgres -d -p 5432:5432 --env POSTGRES_DB=myDatabase --env
POSTGRES_USER=databaseUser --env POSTGRES_PASSWORD=databasePassword --volume
postgres:/var/lib/postgresql/data:rw postgres:9.6

→ ~ docker run --name api-server -d -e DATABASE_HOST=172.17.0.1 -e DATABASE_PORT=5432 -e
DATABASE_USER=databaseUser -e DATABASE_PASSWORD=databasePassword -e DATABASE_NAME=myDatabase -p
8000:8000 shopping-cart:1.0
```

# Network

# Network

- Communication entre la base et le serveur d'API via l'interface réseau du host
  - Peu sécurisé
  - Peu optimisé
- Communication direct entre containers
  - links: communication 1-1 (dépréciée)
  - networks: communication N-Nsous réseaux privés gérés par Docker

# Network

```
→ ~ docker network create myNetwork
```

```
c17bbf0c47c63a0c9ec3ccdd534134140e19550d0f87b8ca1ee5c  
bac220ec3e1
```

```
→ ~ docker network ls
```

NETWORK ID	NAME	DRIVER
SCOPE		
44291f9371dc	bridge	bridge
local		
7877333e1bb4	host	host
local		
c17bbf0c47c6	myNetwork	bridge
local		
ed3943351db0	none	null
local		

```
→ ~ docker run --name c1 -it busybox sh
```

```
/ # ip addr
```

```
...
```

```
/ # ip addr
```

```
→ ~ docker network connect myNetwork c1
```

```
→ ~ docker network disconnect myNetwork c1
```

```
→ ~ docker network connect myNetwork c1
```

```
→ ~ docker network inspect myNetwork
```

# Network

```
→ ~ docker run --name c2 -it --network myNetwork busybox sh    → ~ docker network inspect myNetwork
/ # ip addr
...
/ # ping 172.18.0.2

/ # ping c1
```

- Communication direct entre les deux containers
- Plus accessible via le réseau du host
- Résolveur DNS au sein des sous-réseaux via les noms de containers

# Network

## - Postgres & Serveur d'API

```
→ ~ docker network create backend
```

```
backend
```

```
→ ~ docker run --name postgres -d --env POSTGRES_DB=myDatabase --env POSTGRES_USER=databaseUser --env POSTGRES_PASSWORD=databasePassword --volume postgres:/var/lib/postgresql/data:rw --network backend postgres:9.6
```

```
→ ~ docker run --name api-server -d -e DATABASE_HOST=postgres -e DATABASE_PORT=5432 -e DATABASE_USER=databaseUser -e DATABASE_PASSWORD=databasePassword -e DATABASE_NAME=myDatabase -p 8000:8000 --network backend shopping-cart:1.0
```



# Docker registry

# Docker registry

- Stockage versionné et partage d'image Docker
- Stockage des layers séparément

# Docker registry

- Stockage versionné et partage d'image Docker
- Stockage des layers séparéments
- Implémentation de Docker: Docker Registry
- Autres implémentations:
  - Sonatype Nexus
  - Jfrog Artifactory
  - Portus
- Stockage local
- Mirroring de dépôts externes



# Docker Registry

```
→ ~ docker run -d -p 5000:5000 --name registry registry:2
```

- Transfert via le protocole http
  - https obligatoire par défaut
    - sauf pour localhost
  - Possibilité d'ajouter des 'insecure registry'
    - Adresses de Docker registry en http
    - Nécessite un fichier de configuration et le redémarrage de Docker

# Docker Registry

```
→ ~ sudo more /etc/docker/daemon.json
{
  "insecure-registries" : [ "my-insecure-docker-registry.org:8080" ],
  "registry-mirrors": ["https://my-secure-docker-registry.org"]
}

→ ~ ip addr

→ ~ sudo systemctl daemon-reload
→ ~ sudo systemctl restart docker

→ ~ docker info
...
Insecure Registries:
  127.0.0.0/8
  my-insecure-docker-registry.org:5000
Registry Mirrors:
  https://my-secure-docker-registry.org/
...
```

# Docker Registry - Tag de l'image

- Le nom de l'image doit contenir l'url de la registry de destination
- Il est nécessaire de renommer l'image

```
→ ~ docker image ls shopping-cart
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
shopping-cart	1.0	ab5592262b00	2 days ago	935MB

```
→ ~ docker tag ab5592262b00 localhost:5000/vbenji/shopping-cart:1.0
```

```
→ ~ docker image ls localhost:5000/vbenji/shopping-cart
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost:5000/vbenji/shopping-cart	1.0	ab5592262b00	2 days ago	935MB

# Docker registry - Docker push

```
→ ~ docker push localhost5000/vbenji/shopping-cart:1.0
The push refers to repository [localhost:5000/vbenji/shopping-cart]
6d4fc8689539: Pushed
...
660314270d76: Pushed
1.0: digest: sha256:51a23aeef45f5d964258450a07f21a3c2ce2a7028cb12cc0ff5dfd095cbc05d3 size: 2849
```

# Docker registry - Docker pull

```
→ ~ docker pull <ip-de-mon-voisin>:5000/<nom-de-mon-voisin>/shopping-cart:1.0
1.0: Pulling from <ip-de-mon-voisin>:5000/<nom-de-mon-voisin>/shopping-cart
4ae16bd47783: Already exists
..
42225b875e6d: Pull complete
Digest: sha256:51a23aeef45f5d964258450a07f21a3c2ce2a7028cb12cc0ff5dfd095cbc05d3
Status: Downloaded newer image for <ip-de-mon-voisin>:5000/<nom-de-mon-voisin>/shopping-cart:1.0
```



# Docker Registry - Authentification

- Il est possible de sécuriser une registry

```
Docker login my-secure-docker-registry:5000
```

```
Docker logout my-secure-docker-registry:5000
```

# Docker Registry - Authentication

- Il est possible de sécuriser une registry

```
Docker login my-secure-docker-registry:5000
```

```
Docker logout my-secure-docker-registry:5000
```

- Username & Password
  - Arguments --username et --password
  - Arguments --username et --password-stdin
- Temporairement stocké
  - \$HOME/.docker/config.json

# Docker Registry - Authentication

```
→ DockerCentos more /home/bvouillaume/.docker/config.json
{
  "auths": {
    "localhost:5000": {
      "auth": "SW50ZWNoOkludGVjaA=="
    }
  },
  "HttpHeaders": {
    "User-Agent": "Docker-Client/19.03.2 (linux)"
  }
}
```

- Auth
  - base64(<username>:<password>)

Docker logout <registry> supprime l'entrée correspondante du fichier

# Frontend

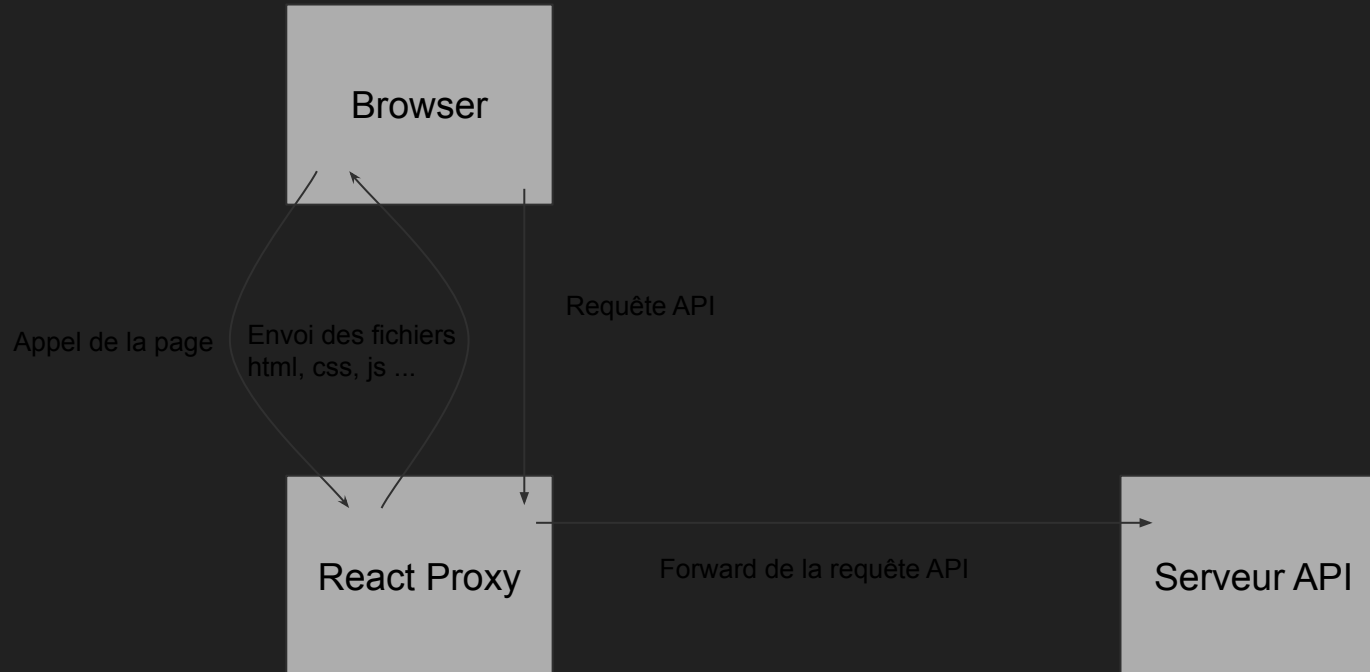
# Frontend - React / Angular

- Plusieurs méthodes
  - Build applicatif dans le build de l'image
  - Build avant le build de l'image
  - Serveur nodejs
  - Serveur de fichiers statiques (httpd, nginx ...)
- Avantages et inconvénients
  - Sécurité
  - Configuration
- Choix structurant dans le développement

# Frontend - React

- Projet React
- Contenu statique mis à disposition par un serveur de fichier
- Exécution du code sur le browser du client (client-side)
- Communication entre le browser et le serveur d'API indirect
  - Le projet react embarque un proxy qui forward les requêtes au backend

# Frontend - React avec proxy



# Frontend - React

- L'utilisation du mode proxy impose la manière de créer l'image Docker
  - Impossible de n'avoir qu'un serveur de fichier statique
  - Serveur NodeJS obligatoire



# Frontend - React

- L'utilisation du mode proxy impose la manière de créer l'image Docker
  - Impossible de n'avoir qu'un serveur de fichier statique
  - Serveur NodeJS obligatoire
- D'autres modes de fonctionnement nécessite de revoir le code
  - Appel direct entre le browser et le serveur API
  - Server side Rendering
  - Frontend hébergé par le backend

# Frontend - React

- Projet React
- Node 10.16
- Utilisation de npm pour la génération de dépendances
  - Supprimer les ^ du package.json
  - Toujours fixer ces dépendances !
  - Changer le script 'watch-css' de 'yarn run' à 'npm run'
- Téléchargement des dépendances
  - npm install
- Démarrage du projet
  - npm run start

# Frontend - React

Dockerfile

```
FROM node:10.16
```

```
WORKDIR /usr/src/app
```

```
COPY . .
```

```
RUN npm install
```

```
EXPOSE 3000
```

```
CMD ["npm", "run", "start"]
```

```
→ ~ docker build -f Dockerfile -t shopping-cart-front:1.0 .
```

# Frontend - React

- Configuration du proxy
  - Package.json
  - "proxy": "<http://172.17.0.1:8000>"

# Frontend - React

- Configuration du proxy
  - Package.json
  - "proxy": "<http://172.17.0.1:8000>"
- Configuration en dure dans le package.json

## Pistes d'améliorations

- Variable d'environnement

# CORS

- backend/backend/settings.py
  - `ALLOWED_HOSTS = ["*"]`
- backend/backend/middleware.py
  - `response['Access-Control-Allow-Origin'] = '*'`
- Externalisation de ces paramètres
  - Variables d'environnements

# Frontend - React

- Avantages du mode proxy
  - Exposition vers l'extérieur nécessaire que pour le serveur nodejs
  - Docker network entre le proxy et le serveur d'API
- Inconvénients
  - Sécurité
  - Performance
  - Complexité

# Frontend - React

- Téléchargement des dépendances à chaque build docker
- Utilisation d'internet lors du build Docker
- Build non répétable
- Possibilité de faire le téléchargement de dépendances et la transpilation du code avant le build Docker
  - CI
  - npm run build
- On n'ajoute au container que ce qui est nécessaire à son bon fonctionnement



# Frontend - React

Dockerfile2

```
FROM node:10.16
```

```
WORKDIR /usr/src/app
```

```
COPY . .
```

```
CMD ["npm", "run", "start-js"]
```

```
→ ~ docker build -f Dockerfile2 -t shopping-cart-front:2.0 .
```

# Frontend - React - Nginx

- Méthode avec Nginx
  - Non fonctionnel à cause du proxy
  - Pour l'exemple
- Nécessite de faire le build avant
  - On ajoute à l'image Docker que le contenu statique
  - On peut ajouter de la configuration nginx
    - Redirection de page inexistante vers index.html (Angular)

# Frontend - React

```
Dockerfile.Nginx
```

```
FROM nginx:1.17
```

```
COPY nginx/ /etc/nginx/
```

```
WORKDIR /usr/share/nginx/html
```

```
COPY ./build/ /usr/share/nginx/html
```

```
→ ~ docker build -f Dockerfile.nginx -t shopping-cart-front:1.0-nginx .
```

# Frontend - React - Nginx

- Avantages du mode Nginx
  - Performance
  - Sécurité
- Inconvénients
  - Client Side
    - Pas d'accès à des variables d'environnements côté serveur
  - Configuration difficile
  - Différences importantes entre les environnements de développements et déployés

# Multi Stage Build

# Multi Stage Build

- Possibilité d'enchaîner plusieurs FROM dans un Dockerfile
- Possibilité de copier des fichiers d'un FROM à l'autre

# Multi Stage Build

- Possibilité d'enchaîner plusieurs FROM dans un Dockerfile
- Possibilité de copier des fichiers d'un FROM à l'autre
- Fonctionnalité introduite par Microsoft
  - Image de base complète > 2 Go
  - Image nano ~ centaines de Mo

# Multi Stage Build

- Possibilité d'enchaîner plusieurs FROM dans un Dockerfile
- Possibilité de copier des fichiers d'un FROM à l'autre
- Fonctionnalité introduite par Microsoft
  - Image de base complète > 2 Go
  - Image nano ~ centaines de Mo
- Décomposer les builds Docker
  - Partie build applicatif
  - Partie run
- Sorte de CI intégrée



# Multi Stage Build

```
Dockerfile2.nginx

FROM node:10.16 AS builder

WORKDIR /usr/src/app

COPY . .

RUN npm install && \
    npm run build

FROM nginx:1.17

#COPY nginx/ /etc/nginx/

WORKDIR /usr/share/nginx/html

COPY --from=builder /usr/src/app/build/ /usr/share/nginx/html
```

# Frontend - Backend - Postgres

# Frontend - Backend - Postgres

## - Postgres & Serveur d'API & Front React

```
→ ~ docker run --name postgres -d --env POSTGRES_DB=myDatabase --env POSTGRES_USER=databaseUser --env POSTGRES_PASSWORD=databasePassword --volume postgres:/var/lib/postgresql/data:rw --network backend postgres:9.6
```

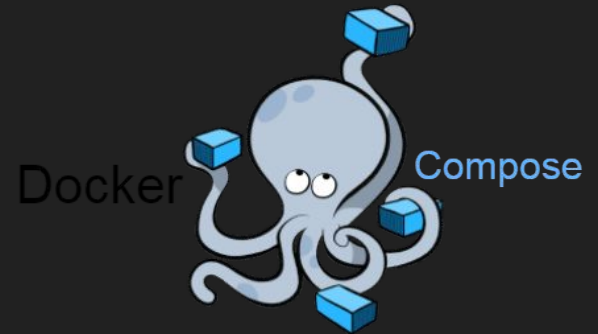
```
→ ~ docker run --name api-server -d -e DATABASE_HOST=postgres -e DATABASE_PORT=5432 -e DATABASE_USER=databaseUser -e DATABASE_PASSWORD=databasePassword -e DATABASE_NAME=myDatabase -p 8000:8000 --network backend shopping-cart:1.0
```

```
→ ~ docker run -p 3000:3000 shopping-cart-front:1.0
```

# Docker Compose

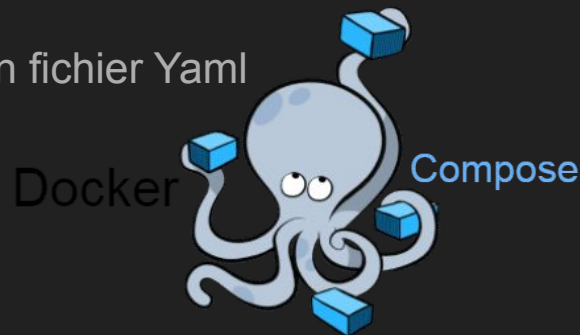
# Docker Compose

- Outil permettant de manipuler plusieurs containers à la fois
  - Commandline
    - docker-compose
    - Proche de celle de Docker



# Docker Compose

- Outil permettant de manipuler plusieurs containers à la fois
  - Commandline
    - docker compose
    - Proche de celle de Docker
  - Fichier de configuration
    - Yaml
    - docker-compose.yml
    - Transcription des commandes docker en fichier Yaml



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

# Docker Compose

```
docker-compose.yml
```

→ ~ docker compose up

```
services:
```

→ ~ docker compose up -d

```
  postgres:
```

```
    image: postgres:9.6
```

→ ~ docker compose logs

```
    container_name: postgres
```

```
    environment:
```

→ ~ docker compose stop

```
      - POSTGRES_DB=myDatabase
```

```
      - POSTGRES_USER=databaseUser
```

→ ~ docker compose start

```
      - POSTGRES_PASSWORD=databasePassword
```

```
    volumes:
```

→ ~ docker compose down

```
      - postgres:/var/lib/postgresql/data:rw
```

```
    networks:
```

```
      - backend
```

```
volumes:
```

```
  postgres:
```

```
networks:
```

```
  backend:
```

# Docker Compose

`docker-compose.yml`

→ `~ docker compose build`

`backend:`

→ `~ docker compose up`

`image: shopping-cart:1.0`

`container_name: api-server`

`depends_on:`

`- postgres`

`environment:`

`- DATABASE_HOST=postgres`

`- DATABASE_PORT=5432`

`- DATABASE_USER=databaseUser`

`- DATABASE_PASSWORD=databasePassword`

`- DATABASE_NAME=myDatabase`

`ports:`

`- 8000:8000`

`networks:`

`- backend`



# Docker Compose

- Depends\_on ordonne les démarrages
- Mais aucune garantie si les services sont bien démarrés ou assez rapidement

# Docker Compose

- Depends\_on ordonne les démarrages
  - Mais aucune garantie si les services sont bien démarrés ou assez rapidement
  - Rendre l'application résiliente en cas d'absence de la base
  - Implémenter un retry
  - ...
- 
- `docker-compose up -d postgres`
  - `docker-compose up -d backend`
  - `docker-compose logs -f`

# Docker Compose

```
docker-compose.yml
```

```
→ ~ docker-compose build
```

```
frontend:
```

```
→ ~ docker-compose up
```

```
  build:
```

```
    context: ./frontend
```

```
    dockerfile: ./Dockerfile2
```

```
  image: shopping-cart-front:1.0
```

```
  ports:
```

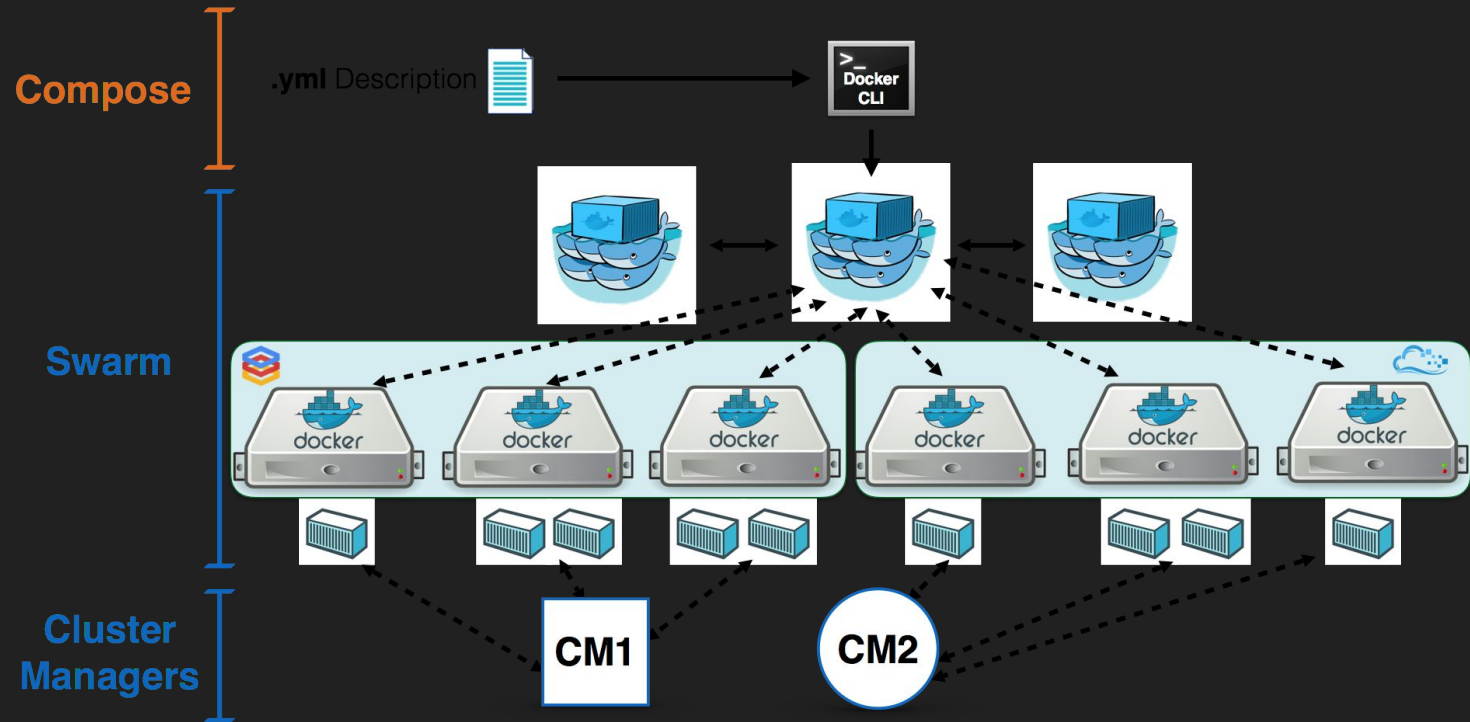
```
  - 3000:3000
```

# Docker Swarm

# Docker Swarm

- On sait gérer des containers sur un host
- Les hosts sont simples et identiques, seul Docker est installé.
- Pourrait-on gérer nos containers sur plusieurs hosts ?
  - C'est de l'orchestration de containers
- Docker Swarm
  - Multi Hosts
  - Gestion des containers centralisée
  - Inclu avec Docker

# Docker Swarm



# Docker Swarm

- 1 - 3 - 5 ... Masters
  - Gestion du cluster
- 1...N nodes
  - Exécutent les containers

# Docker Swarm - Master

```
→ ~ docker swarm init --advertise-addr 192.168.1.12
```

```
Swarm initialized: current node (ylv7twbic2vlvo00t8txfvocu) is now a manager.
```

To add a worker to this swarm, run the following command:

```
docker swarm join --token
```

```
SWMTKN-1-1742nj8oeewz3915kusxdkvomzgy7rja0gsjufh8iu81xpqh0b-5lby6wn7twphj93s6wcjxfbr 192.168.1.12:2377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.



# Docker Swarm - Node

```
[vagrant@centos7 ~]$ docker swarm join --advertise-addr 192.168.1.42 --token SWMTKN-1-1742nj8oezw3915kuzadkvomzgy7rja0ga3ufh8iu8ixpgh0b-5lby6wn7twphj93s6wcjxfbr 192.168.1.12:2377

This node joined a swarm as a worker.
[vagrant@centos7 ~]$
```

```
→ ~ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
ylv7twbic2vlvo00t8txfvocu *	Laptop-Benjamin	Ready	Active	Leader	19.03.2
8slwrtwup7jvbgqmwjc7oc8y	centos7.localdomain	Ready	Active		19.03.2

```
→ ~
```

# Docker Swarm - Service

- Permet de déployer des containers comme service du cluster swarm
  - Placement automatique dans le cluster
  - Possibilité d'augmenter/diminuer le nombre d'instances
  - Gestion semblable aux containers sur un host, sur tout le cluster

# Docker Swarm - Service

```
→ ~ docker service create -p 8080:80 emilevauge/whoami
yg0292675btbwjirc150gq4ag
overall progress: 1 out of 1 tasks
1/1: running [=====>]
verify: Service converged
→ ~ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
408ab09c1821	emilevauge/whoami:latest	"/whoami"	41 seconds ago	Up 40 seconds	80/tcp
gracious_meninsky.1.smum1o9cp4qwr1w4szwqlfaax					

```
→ ~ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
yg0292675btb	gracious_meninsky	replicated	1/1	emilevauge/whoami:latest	

```
*:8080->80/tcp
```

# Docker Swarm - Service

```
→ ~ docker service scale =2
yg0292675btb scaled to 2
overall progress: 2 out of 2 tasks
1/2: running [=====>]
2/2: running [=====>]
verify: Service converged
→ ~ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
yg0292675btb	gracious_meninsky	replicated	2/2	emilevauge/whoami:latest	

```
*:8080->80/tcp
→ ~ docker service ps gracious_meninsky
```

ID	NAME	IMAGE	NODE	DESIRED STATE
CURRENT STATE	ERROR	PORTS		
smum1o9cp4qw	gracious_meninsky.1	emilevauge/whoami:latest	Laptop-Benjamin	Running
Running 2 minutes ago				
1661nj8ysmdt	gracious_meninsky.2	emilevauge/whoami:latest	centos7.localdomain	Running
Running about a minute ago				

# Docker Swarm - Stack

- Permet de déployer un ensemble de service sur base d'un docker-compose
  - Gestion des networks sur tous les hosts
  - Gestion des volumes sur tous les hosts
- Nécessite parfois quelques ajustements du docker-compose
  - Url des services

# Docker Swarm - Stack

```
→ ~ docker stack deploy -c docker-compose.yml INTECH
Ignoring unsupported options: build

Ignoring deprecated options:

container_name: Setting the container name is not supported.

Creating network INTECH_default
Creating network INTECH_backend
Creating service INTECH_frontend
Creating service INTECH_postgres
Creating service INTECH_backend
→ ~
```

# Docker Swarm - Stack

```
→ ~ docker stack deploy -c docker-compose.yml INTECH
Ignoring unsupported options: build

Ignoring deprecated options:

container_name: Setting the container name is not supported.

Creating network INTECH_default
Creating network INTECH_backend
Creating service INTECH_frontend
Creating service INTECH_postgres
Creating service INTECH_backend
→ ~ docker stack ls
NAME                SERVICES          ORCHESTRATOR
INTECH              3                 Swarm
→ ~
```

# Docker Swarm - Stack

```
➔ ~ docker stack services INTECH
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
2ogxmh0f03c0	INTECH_backend	replicated	1/1	shopping-cart:1.0	
*:8000->8000/tcp					
obrru5a013g5	INTECH_frontend	replicated	1/1	shopping-cart-front:1.0	
*:3000->3000/tcp					
rj4xj512j0vx	INTECH_postgres	replicated	1/1	postgres:9.6	

```
➔ ~ docker stack ps INTECH
```

ID	NAME	IMAGE	NODE	DESIRED STATE
CURRENT STATE	ERROR	PORTS		
yte45y5jps2u	INTECH_backend.1	shopping-cart:1.0	Laptop-Benjamin	Running
Running 7 minutes ago				
xel47s1cdk5n	INTECH_postgres.1	postgres:9.6	centos7.localdomain	Running
Running 7 minutes ago				
4adje3nrr54q	INTECH_frontend.1	shopping-cart-front:1.0	Laptop-Benjamin	Running
Running 7 minutes ago				

```
➔ ~
```



# Docker Swarm - Stack

- Possibilité de configurer le déploiement de la stack dans le docker-compose
- Object deploy dans le Yml
  - Interprété par docker stack
  - Non Interprété par docker-compose

# Docker Swarm - Stack

- Possibilité de configurer le déploiement de la stack dans le docker-compose
- Object deploy dans le Yml
  - Interprété par docker stack
  - Non Interprété par docker-compose
- Permet de configurer
  - La réplication
  - Le placement (contrainte, affinité...)
  - Politique de redémarrage
  - Politique d'update, de rollback
  - ...

# Docker Secret

# Docker Secret

- L'activation de Docker Swarm permet d'utiliser les Docker Secret
  - Impossible sans swarm
- Permet de créer, stocker, distribuer des secrets dans le Cluster
  - password
  - Certificats
  - configurations
  - ...
- Un secret est représenté sous la forme du contenu d'un fichier dans le filesystem du container
  - `/run/secrets/<my-secret>`

# Docker secret

```
→ ~ echo "Hello INTECH" | docker secret create INTECH -
uegwj7bmqkbxsxsdkcovbr40

→ ~ docker secret ls


| ID                       | NAME   | DRIVER | CREATED        | UPDATED        |
|--------------------------|--------|--------|----------------|----------------|
| uegwj7bmqkbxsxsdkcovbr40 | INTECH |        | 21 seconds ago | 21 seconds ago |



→ ~ docker service create --name secret-INTECH --secret INTECH nginx
mwcm0tnwh6qavobkgc9hilfj1
overall progress: 1 out of 1 tasks
1/1: running [=====>]
verify: Service converged

→ ~ docker exec -it d3051e37a476 sh
# more /run/secrets/INTECH
Hello INTECH
#
```

# Docker Swarm - Conclusion

- Orchestration de containers
  - Inclu dans Docker
  - Simple
  - Un peu trop simple
- Il manque quelques concepts facilitant l'adoption
  - Pods
  - Réseaux internes / externes
  - Écosystème
- Kubernetes

Questions ?