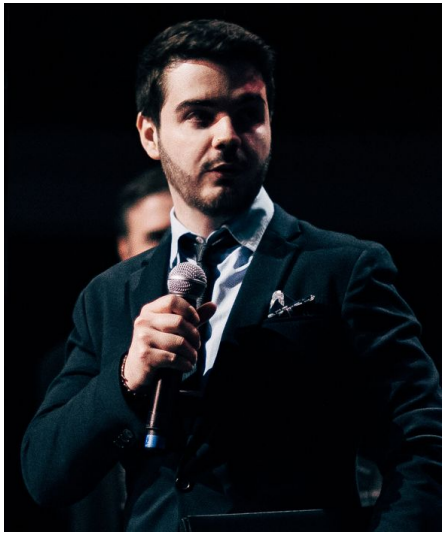


Samuel ANTUNES
Consultant Ingénieur DevSecOps
OCTO Technology
Email : contact@samuelantunes.fr

ICEBREAKER

shorturl.at/imMW9

AGENDA

1. **Les bases de Docker**
   a. Introduction: l'avant Docker
   b. Qu'est-ce que Docker
   c. Architecture et Concepts
2. **Docker en Pratique**
   a. Les images
   b. Les conteneurs
   c. Les volumes
   d. Les networks
   e. Création d'images (Dockerfile) et les registres
   f. Docker-compose
3. **Docker en Prod**
   a. Doable or not

“ Les bases de Docker ”

INTRODUCTION - CONTEXTE D'ARRIVÉE EN 2013

- De nombreuses problématiques liées aux applications
  - La portabilité des applications
  - La distribution des applications
  - Le besoin de décorréler applications et infrastructure
  - La rationalisation des infrastructures

- La montée en puissance
  - Des solutions de PaaS
  - De la philosophie DevOps

INTRODUCTION - LA PORTABILITÉ LOGICIELLE

Comment assurer le déploiement homogène d'une application sur tous ses environnements ?





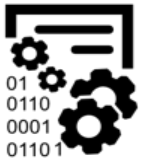


	Env de développement	Assurance Qualité (QA)	Serveur de Production	Cluster de machines	Cloud Public	Ordinateur Personnel	Serveur du Client
Site web statique	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Jobs en Arrière Plan	?	?	?	?	?	?	?
Base de données	?	?	?	?	?	?	?
Analytics	?	?	?	?	?	?	?
Files de messages	?	?	?	?	?	?	?

INTRODUCTION - LA DISTRIBUTION LOGICIELLE

Comment distribuer un logiciel de façon simple et efficace ?

Les différentes méthodes de distribution logicielle



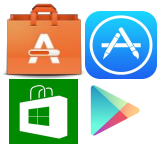
Binaire

Paquet

Installeur

Dépôts de paquets

Virtual Appliance



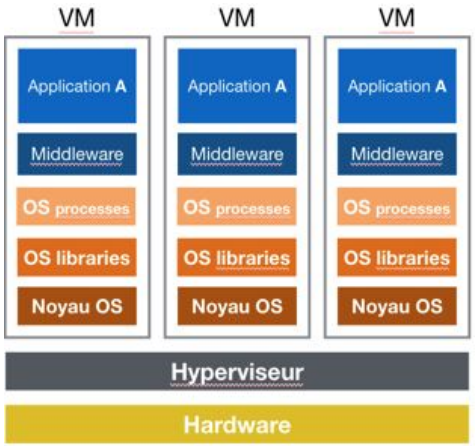
Application Store

INTRODUCTION - LA RATIONALISATION D'INFRA

Comment optimiser l'utilisation des ressources ?
Comment décorréler application et infrastructure ?

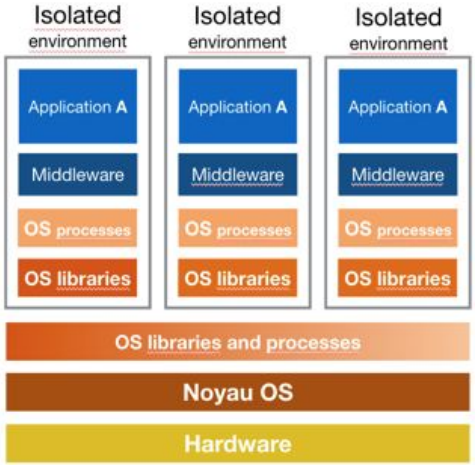
Les 2 technologies de virtualisation des systèmes

Virtualisation



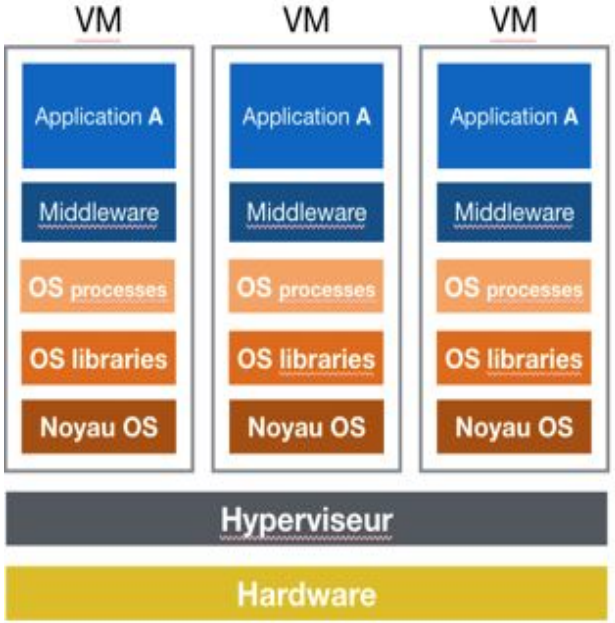
VS

Isolation



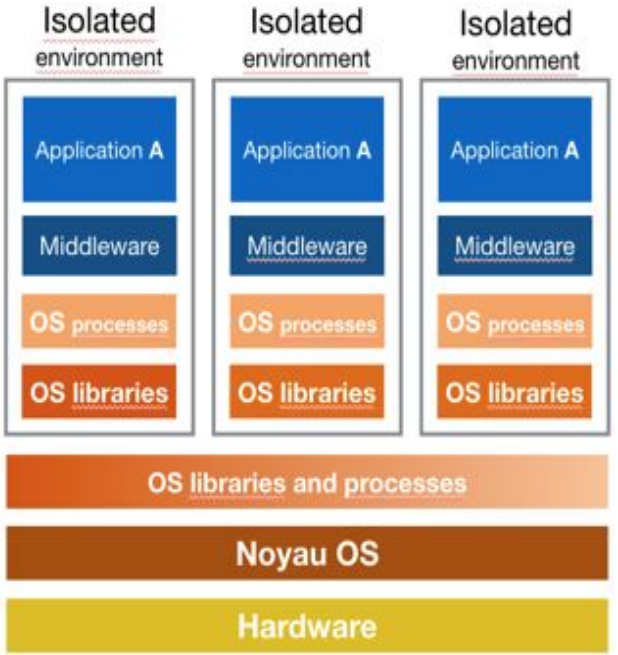
INTRODUCTION - LA RATIONALISATION D'INFRA

VMWare

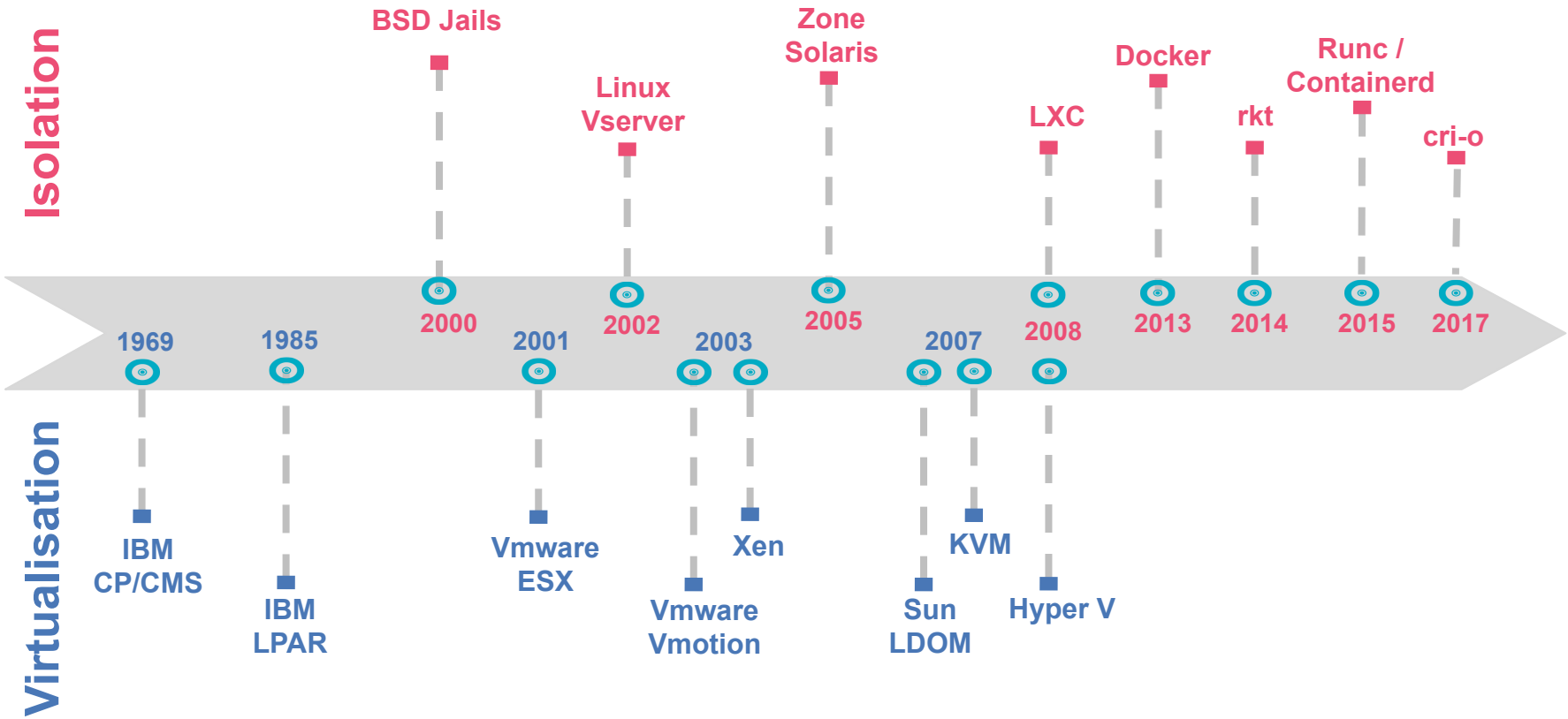


VS

LXC

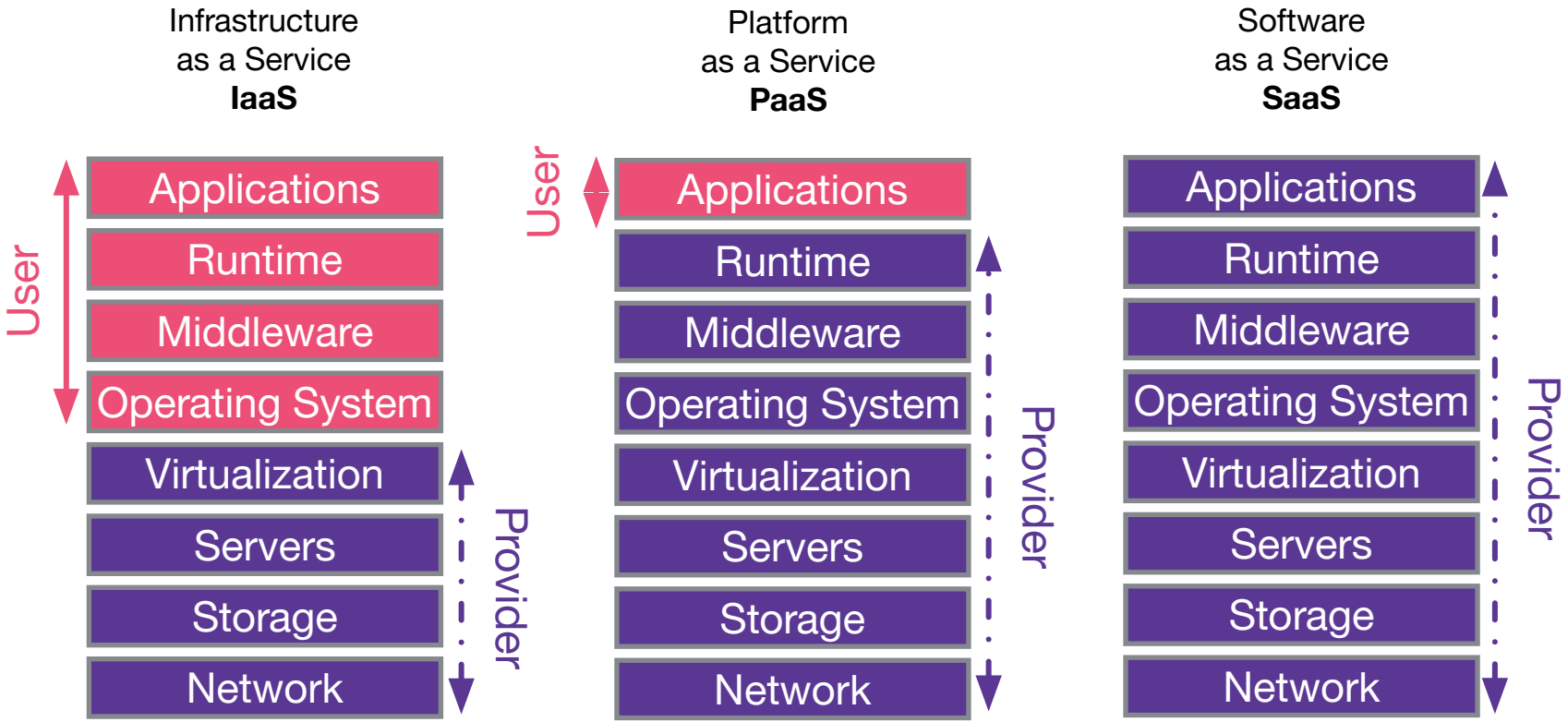


INTRODUCTION - DES TECHNOS PAS SI VIEILLES



INTRODUCTION - LE PAAS

Schéma des différents niveaux de services Cloud



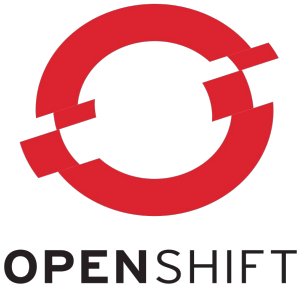
INTRODUCTION - LES CHALLENGES DU PAAS

- Les PaaS concentrent toutes les problématiques indiquées et doivent en plus :
  - Déployer rapidement des nouvelles applications
  - Assurer une élasticité rapide
  - Isoler les applications entre elles

➜ Besoin d'une sur-couche légère d'isolation et d'abstraction

… des conteneurs !!

Les principaux PaaS et leur technologie de conteneurs



LXC containers

INTRODUCTION - LA PHILOSOPHIE DEVOPS

« DevOps est un ensemble de pratiques qui visent à réduire le Time to Market et améliorer la Qualité en optimisant la coopération entre les Développeurs et la Production »

Docker (homonymie)

Cette page d'homonymie répertorie les différents sujets et articles partageant un même nom.

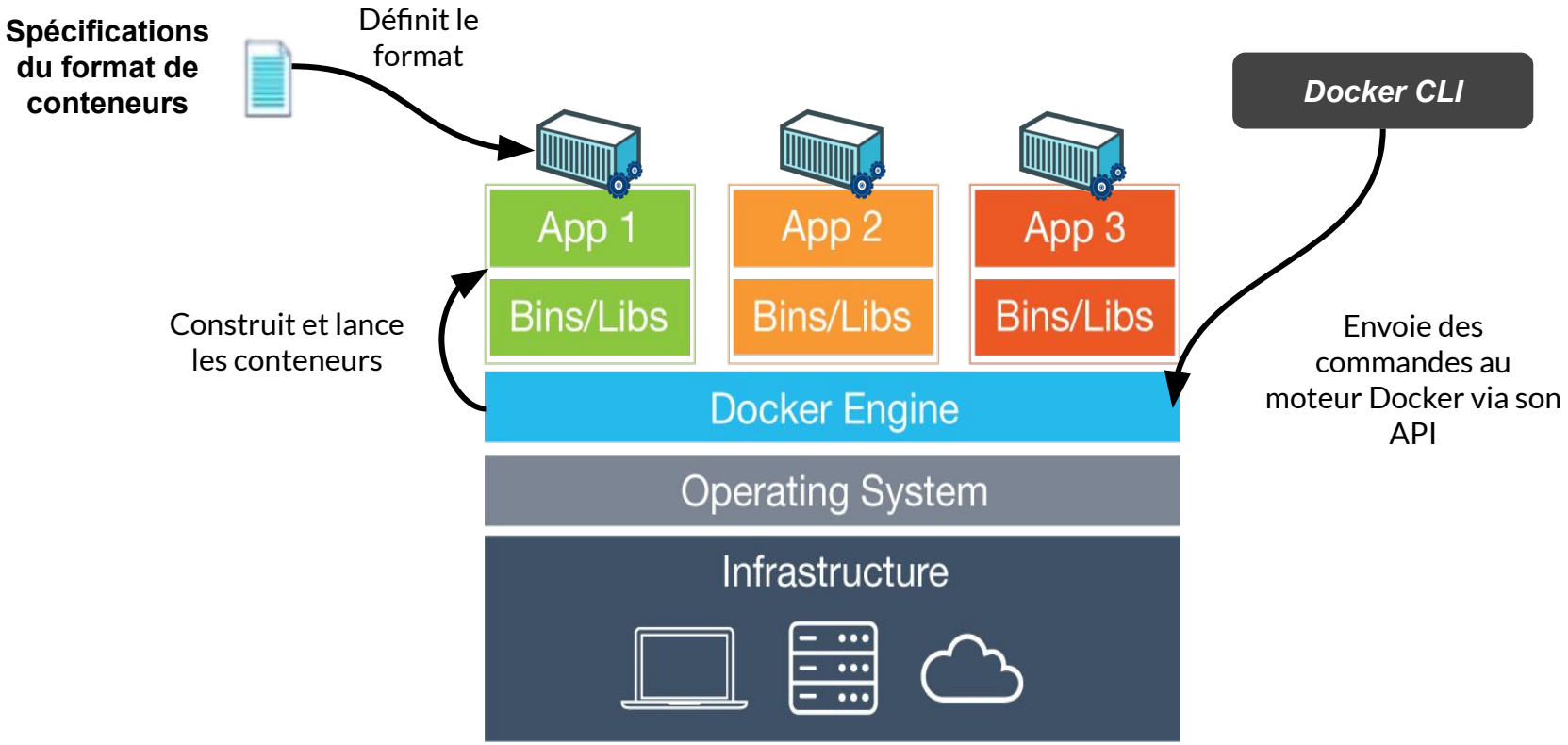
Informatique

- **Docker Inc**, la compagnie qui développe la plateforme Docker.
- **Docker Engine**, le logiciel qui construit et fait tourner les conteneurs.
- **Docker**, le format de conteneur.
- **Docker CLI**, l'outil en ligne de commande pour piloter les conteneurs.
- **Docker Platform**, l'ensemble des logiciels de Docker Inc permettant de gérer les conteneurs.

QU'EST-CE DOCKER ? UNE DÉFINITION

« Une **technologie** permettant de
standardiser le **packaging** et l'**opération** des **applications** »

QU'EST-CE DOCKER ? ENGINE, CLI, CONTENEURS



QU'EST-CE DOCKER ? DOCKER INC

- Société créée initialement en 2008 à San Francisco sous le nom de DotCloud, pour offrir un service de PaaS
- **Renommée en Docker Inc fin 2013** pour se concentrer autour du projet Docker, puis revend la partie PaaS mi-2014
- **Mène depuis 2014 une politique d'acquisition** des solutions qui émergent de la communauté (orchard, kitematic socketplane, tutum…)
- **Se positionne comme le leader du projet communautaire Docker** de développement d'une plateforme ouverte pour les applications distribuées
- Docker Inc modifie son système de packaging du **Docker Engine** avec l'introduction de **Moby** en 2017
- En 2017, Docker Inc lance le **Modernize Traditional Applications** (MTA) en s'associant avec des **éditeurs traditionnels**
- En 2018, Docker EE 2.0 intègre désormais **Kubernetes** et rends possible le déploiement des **stacks** et des fichiers **compose** au travers de Swarm ou Kubernetes

Des caractéristiques uniques

PO — **PORTABLE**

DI — **DISPOSABLE**

LI — **LIVE**

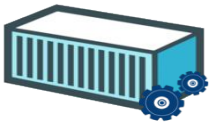
SO — **SOCIAL**

TP1

```
git clone https://3AIW:rBrPi4tgoyEHXjLivp9y@gitlab.com/santunes-formations/docker.git
```

LES ARCHITECTURES ET CONCEPTS DOCKER

L'image
Une arborescence de fichiers contenant tous les éléments requis pour faire tourner une application

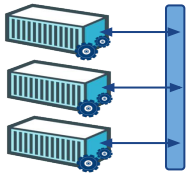


Le conteneur
Une instanciation d'une image en cours d'exécution
sur un système hôte

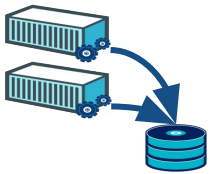
Le registre
Un service centralisé de stockage et distribution d'images

Le montage de répertoires
Un espace de stockage indépendant de l'image utilisable pour les données persistantes

Le *Dockerfile*
Un fichier contenant les instructions permettant de construire une image

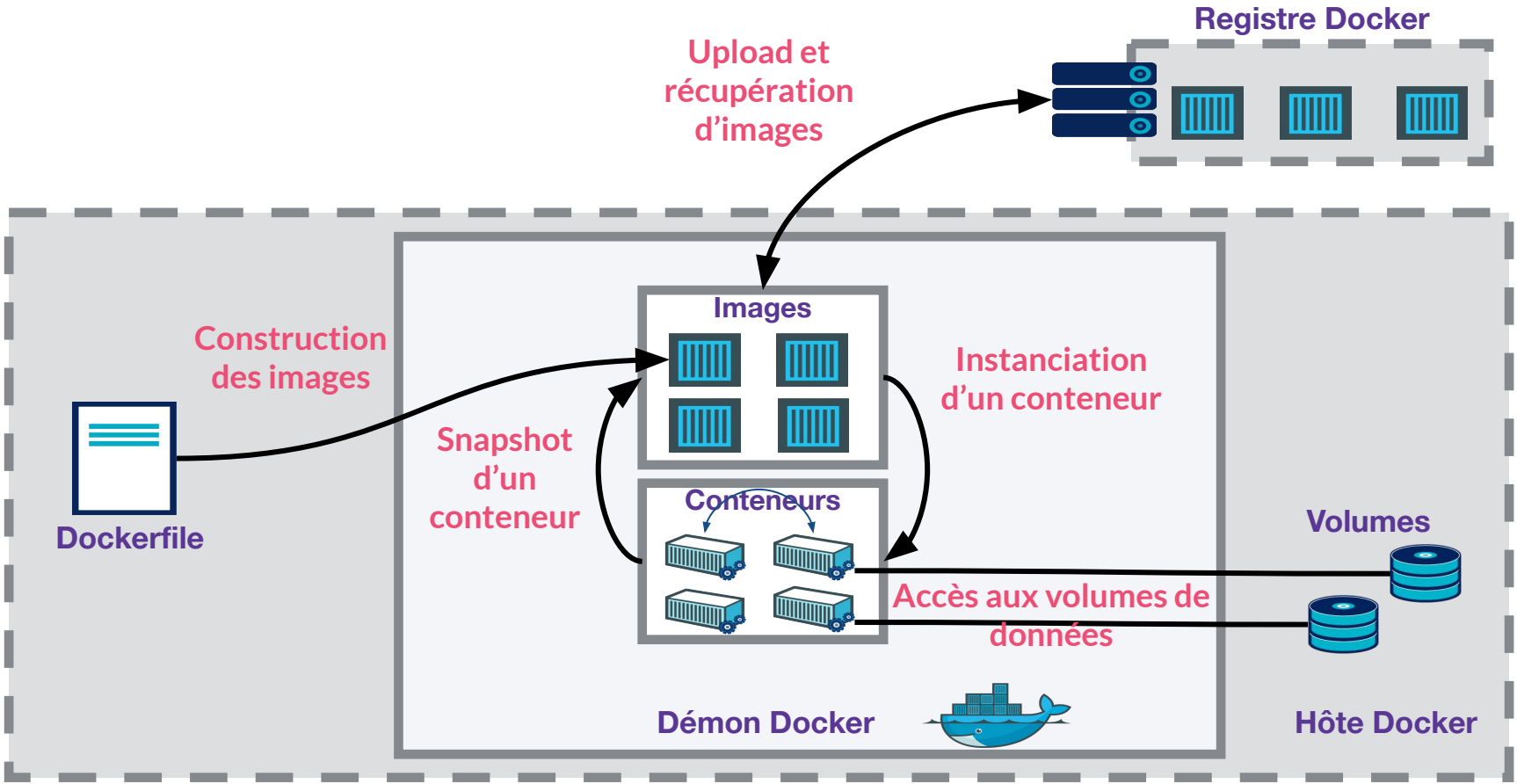


Les networks (docker network)
Permet la communication de conteneurs dans un ou plusieurs réseaux sur une ou plusieurs machines hôtes



Les volumes (docker volume)
Un espace de stockage indépendant de l'image utilisable pour les données persistantes.

LES ARCHITECTURES ET CONCEPTS DOCKER - FONCTIONNEMENT



“ Docker en pratique ”

Une image Docker c'est

- **Un système de fichiers auto-suffisant** contenant *a minima* librairies et binaires de base (libc, libresolv, bash…)

- **Un identifiant unique** assigné à l'image à sa création

- **Des métadonnées** pour préciser la façon d'instancier l'image
  - le processus à exécuter à l'instanciation de l'image,
  - les variables d'environnement à positionner
  - l'utilisateur qui va lancer l'application
  - la configuration réseau (ports exposés, réseau...),
  - les volumes de données à connecter,
  - ...

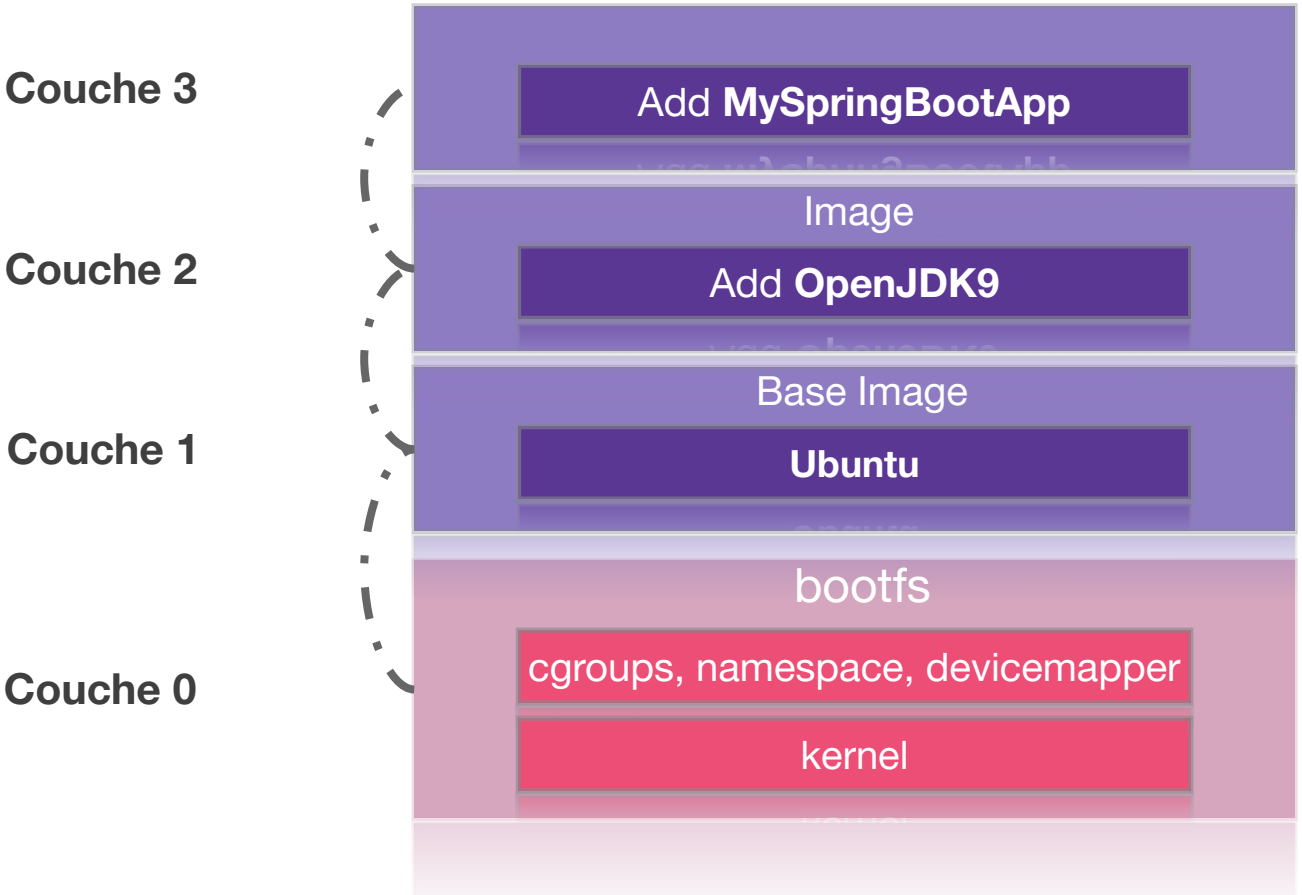
LE MILLE-FEUILLES DES IMAGES

- Docker utilise un systèmes de fichiers avec un **système de couches** pour les images de conteneurs

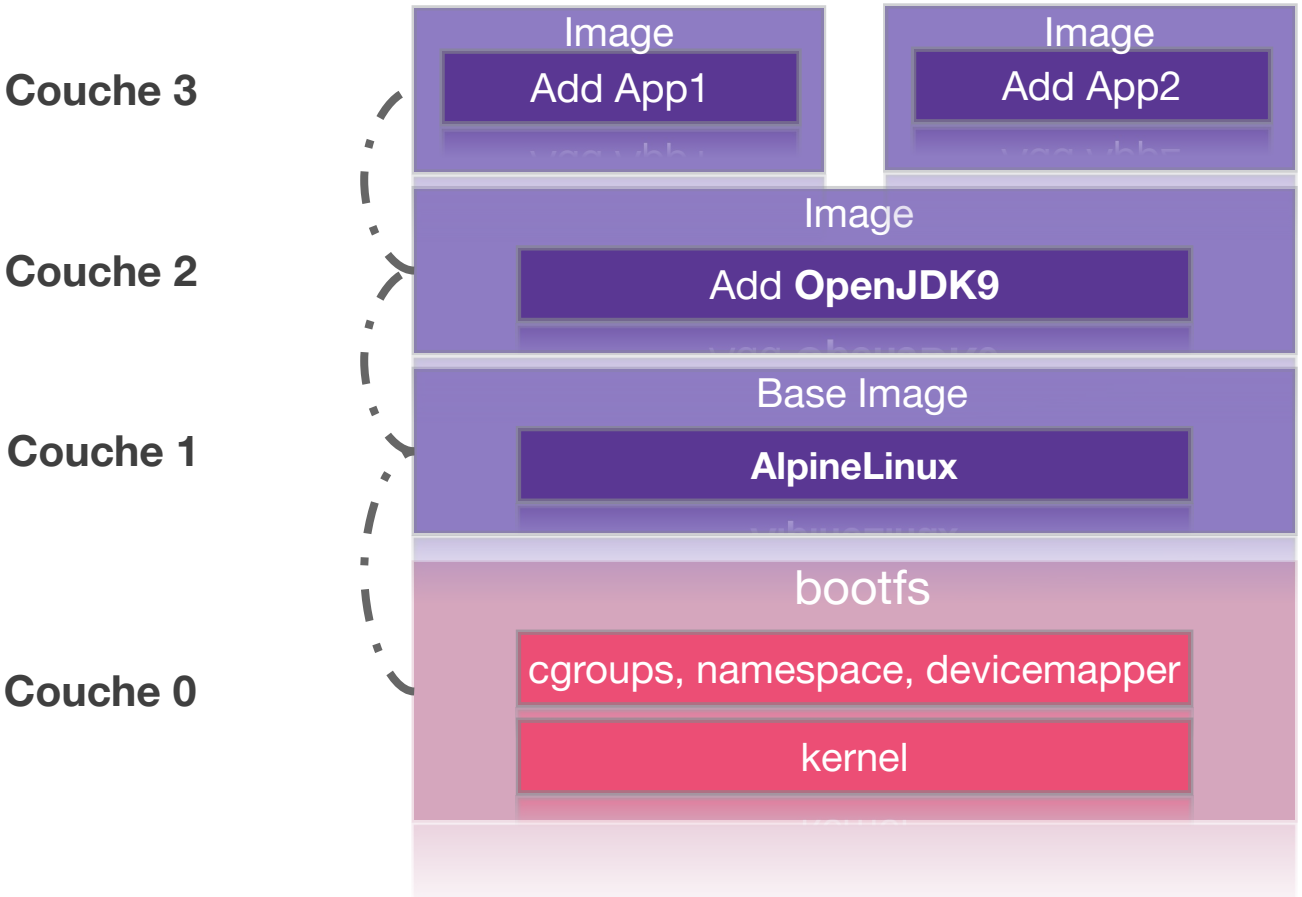
- **Principe**
  - Un ensemble de couches partagées en lecture seule
  - Unifiées par le système pour simuler un unique système de fichiers à plat pour le conteneur

- **Avantages**
  - **Évite la perte d'espace** avec n x 500Mo par OS Ubuntu dans des VMs
  - Permet la récupération et le démarrage rapide des conteneurs

LE MILLE-FEUILLES DES IMAGES



LE MILLE-FEUILLES DES IMAGES



LES IMAGES - L'ESSENTIEL DES COMMANDES

Lister les images locales

docker image ls

Télécharger une image à partir d'un registre

docker image pull

docker image rm

Supprimer une image

docker image tag

Labelliser une image

docker image history

Lister les couches d'une image

TP2

```
git clone https://3AIW:rBrPi4tgoyEHXjLivp9y@gitlab.com/santunes-formations/docker.git
```

LES CONTENEURS

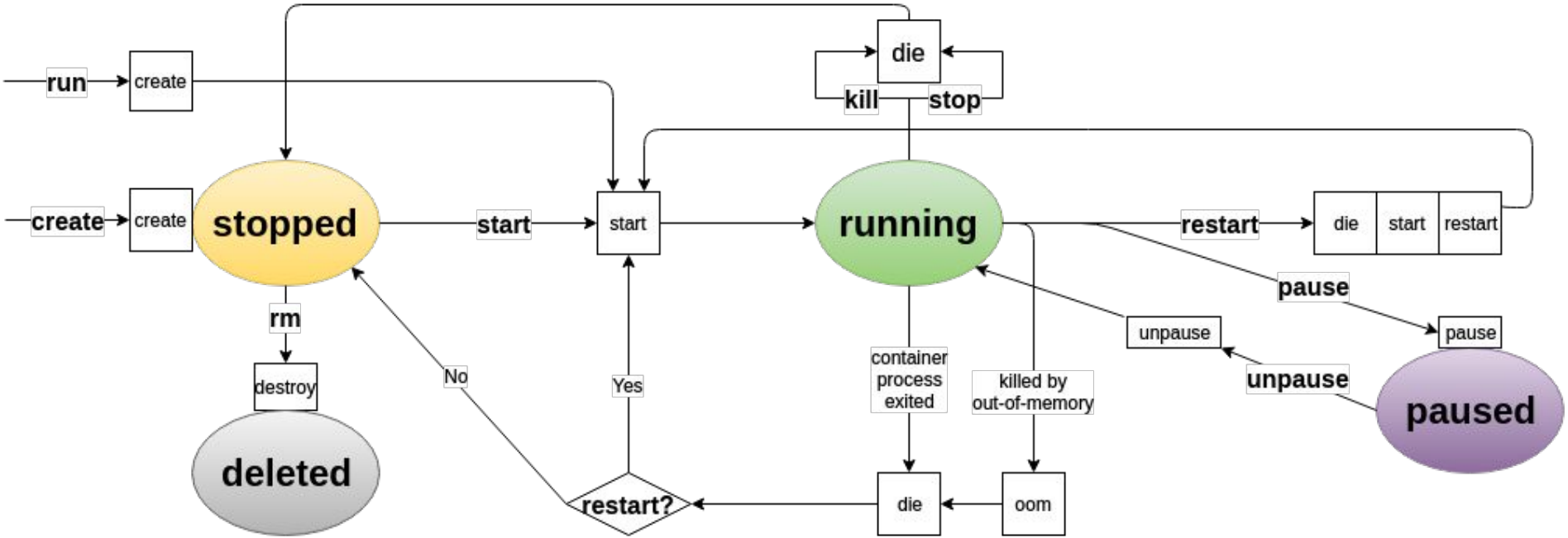
- Le conteneur est **la brique de base de Docker**

- Il est toujours **instancié à partir d'une image**

- Un conteneur **ne vit que pour les processus** qu'il contient

- **Si ces processus s'arrêtent**
  - Le conteneur est stoppé
  - Le contenu modifié subsiste tant que le conteneur n'est pas détruit

LES CONTENEURS - CYCLE DE VIE



LES CONTENEURS - L'ESSENTIEL DES COMMANDES

Créer un conteneur et lancer une commande

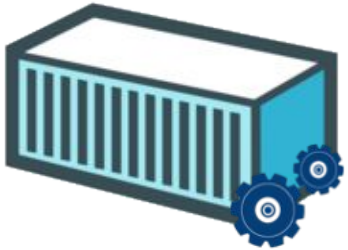
docker container run

Démarrer un conteneur

docker container start

docker container attach

Se connecter à la console du conteneur



Supprimer un conteneur

docker container rm

docker container restart

Redémarrer le conteneur

docker container ls

docker container logs

Accéder aux logs produits

Lister les conteneurs

docker container stop

Stopper un conteneur

LES CONTENEURS - DES COMMANDES AVANCÉES

Créer un conteneur

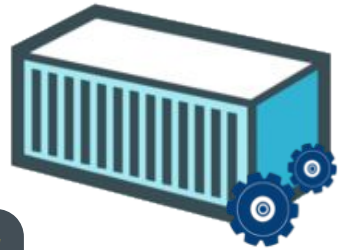
docker container create

Lister les processus

docker container top

docker container exec

Exécuter une commande dans le conteneur



docker container diff

Affiche les différences dans le système de fichiers du conteneur

docker container kill

Tue un conteneur

docker container inspect

Affiche des informations sur un conteneur

TP3

```
git clone https://3AIW:rBrPi4tgoyEHXjLivp9y@gitlab.com/santunes-formations/docker.git
```

LES VOLUMES

- Les conteneurs sont **légers et éphémères** et ne sont pas faits pour enregistrer des données de matières persistantes

- Les données doivent être stockées en dehors de l'image du conteneur dans des **volumes de données dédiés** à cet usage

- **2 techniques historiques pour accéder à des volumes** de données
  - l'accès au système de fichiers de l'hôte,
  - le volume lié au conteneur

LES VOLUMES

Objectifs :

- **Gérer des applications stateful** : pour les bases de données et les applications à architecture traditionnelle

- **Conserver l'indépendance avec les hôtes** : les données ne doivent pas être liées à un hôte et doivent suivre le déplacement des conteneurs qui y sont associés

- **Pouvoir conserver des données indépendamment de l'application** : pour gérer le cycle de vie de la donnée

- **Faciliter les tâches opérationnelles** : snapshot, sauvegarde, restauration, copie…

LES VOLUMES - LES COMMANDES ESSENTIELLES

Créer un volume de données

docker volume create

Supprimer un volume

docker volume rm

Obtenir les informations sur un volume

docker volume inspect

docker volume ls

Lister les volumes existants

LES VOLUMES - EXEMPLES

Exemples sur 2 cas d'utilisation:

- Création d'un volume de données pour MySQL

```
$ docker volume create --name mysql_data
$ docker container run -d -v mysql_data:/var/lib/mysql mysql
```

- Sauvegarde puis restauration des données de MySQL

```
$ docker volume create --name mysql_backup
$ docker container run -v mysql_data:/var/lib/mysql -v mysql_backup:/backups mysql
tar cjvf /backups/backup.tar.bz2 /var/lib/mysql
```

```
$ docker container run -v mysql_data:/var/lib/mysql -v mysql_backup:/backups mysql bash
-c 'cd /var/lib/mysql && tar xvjf /backups/backup.tar.bz2'
```

TP4

```
git clone https://3AIW:rBrPi4tgoyEHXjLivp9y@gitlab.com/santunes-formations/docker.git
```

LES NETWORKS

- Les networks peuvent être des réseaux privés permettant d'isoler des conteneurs entre eux.

- Il existe plusieurs drivers de réseaux permettant de réaliser des actions particulières :
  - Le driver Bridge
  - Le driver none
  - Le driver Host
  - Le driver overlay
  - Le driver macvlan

- On peut ouvrir des ports entre notre machine hôte et notre conteneur grâce à l'argument "*-p <PORT_HOTE>:<PORT_CONTENEUR>*"

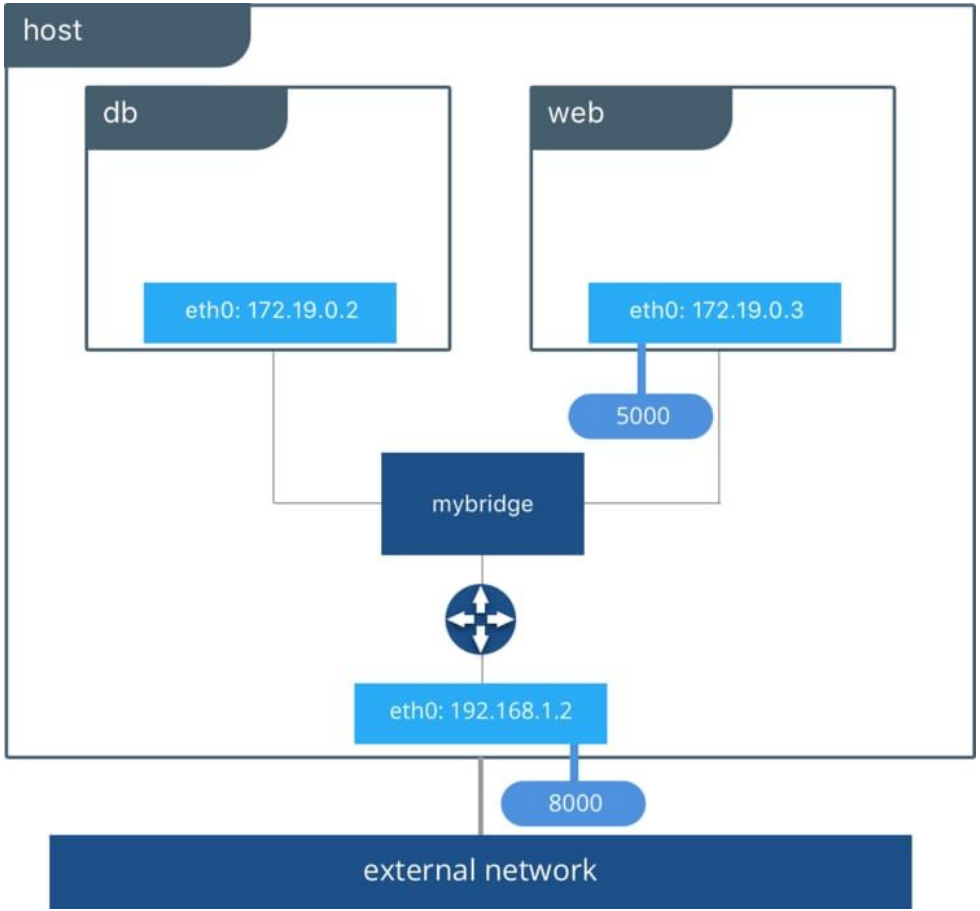
LES NETWORKS - BRIDGE

- A l’installation de Docker, un réseau nommé bridge connecté à l'interface réseau docker0 est créé et lié à chaque conteneur par défaut.

- Le réseau bridge est le type de réseau le plus couramment utilisé.

- Les conteneurs qui utilisent ce driver, ne peuvent communiquer qu'entre eux, cependant ils ne sont pas accessibles depuis l'extérieur.

LES NETWORKS - BRIDGE



LES NETWORKS - LES AUTRES DRIVERS

- **None** : Réseau qui isole complètement un conteneur (ni entrées, ni sorties)

- **Host** : Le conteneur utilise l'interface réseau de son hôte (prendra son IP le rendant disponible à l'extérieur)

- **Overlay** : Permet de créer un lien réseau partagé entre plusieurs hôtes. Docker gère de manière transparente le routage.

- **Macvlan** : Ce type de réseau permet d'attribuer une adresse mac à un conteneur le faisant ainsi apparaître comme un périphérique physique

LES NETWORKS - LES COMMANDES ESSENTIELLES

Créer un nouveau network

docker network create

Supprimer un network

docker network rm

Connecter un conteneur à un network existant

docker network connect

docker network ls

Lister les networks existants

docker network disconnect

Déconnecter un conteneur d'un network

TP5

```
git clone https://3AIW:rBrPi4tgoyEHXjLivp9y@gitlab.com/santunes-formations/docker.git
```

DOCKERFILE

- Fichier contenant des suites d'instructions Docker et de commandes à exécuter pour construire un conteneur

- Permet par exemple d'installer tous les paquets requis par une application (Apache, Java, …)

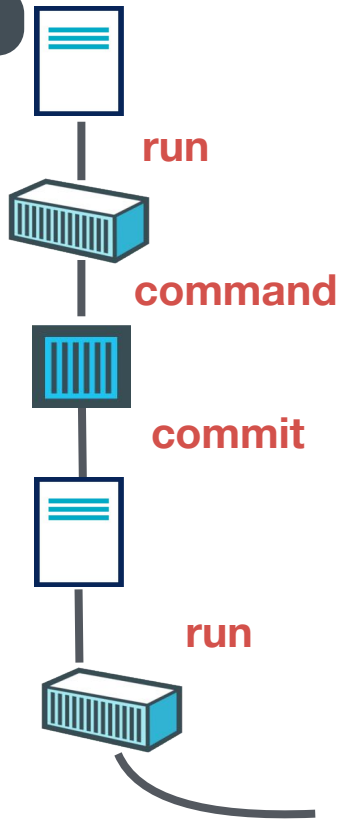
Exemple de fichier *Dockerfile*

```
FROM ubuntu

LABEL maintainer="Sam LE BG"

# Update the repository and install nginx
RUN apt-get update

RUN apt-get install -y nginx

# Copy a configuration file from the current directory
ADD nginx.conf /etc/nginx/

EXPOSE 80

CMD ["nginx"]
```

DOCKERFILE - LE WORKFLOW

```
$ docker image  build -t username/myimage .
```

- Docker lance l'exécution d'un conteneur à partir d'une image
- Une instruction est exécutée et change le contenu en termes de fichiers
- Docker lance l'exécution d'un docker commit pour sauvegarder les changements de la nouvelle couche dans une image
- Docker lance un nouveau conteneur à partir de cette nouvelle image
- La prochaine instruction est exécutée… et ainsi de suite
- L'image finale est taggée "*username/myimage*"



DOCKERFILE - L'INSTRUCTION RUN

Exécute une commande puis crée une nouvelle image à partir des changements

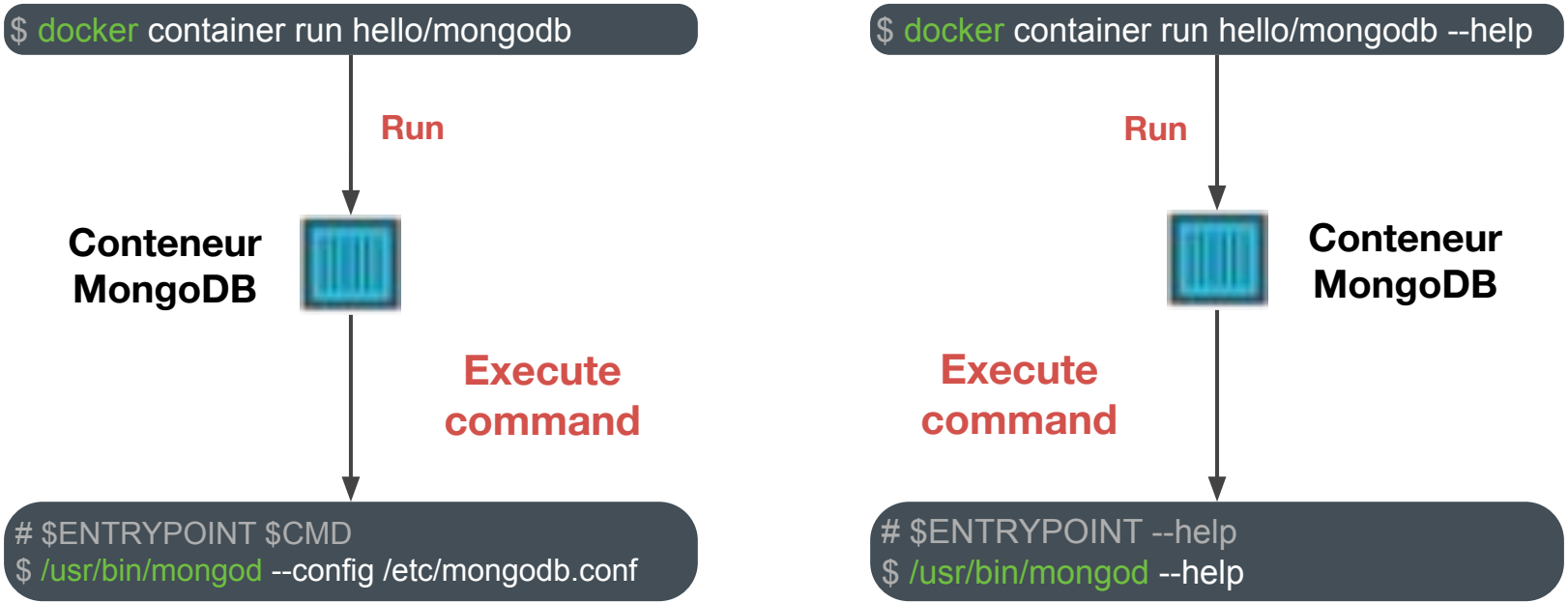
```
RUN apt-get -y install apt-utils
```

Faire une étape de purge n'a pas de sens, il vaut mieux tout inliner dans une même commande :

```
RUN apt-get -y update && apt-get install -y vim
   && rm -rf /var/lib/apt/lists/*
```

DOCKERFILE - L'INSTRUCTION ENTRYPOINT ET CMD ENSEMBLE

```
ENTRYPOINT ["/usr/bin/mongod"] Exécute un container comme un exécutable,
                               potentiellement avec des arguments

CMD ["--config", "/etc/mongodb.conf"]
```



DOCKERFILE - L'INSTRUCTION WORKDIR

Définie le répertoire courant de travail dans le Dockerfile
Effectue un **syscall** pour changer de dossier

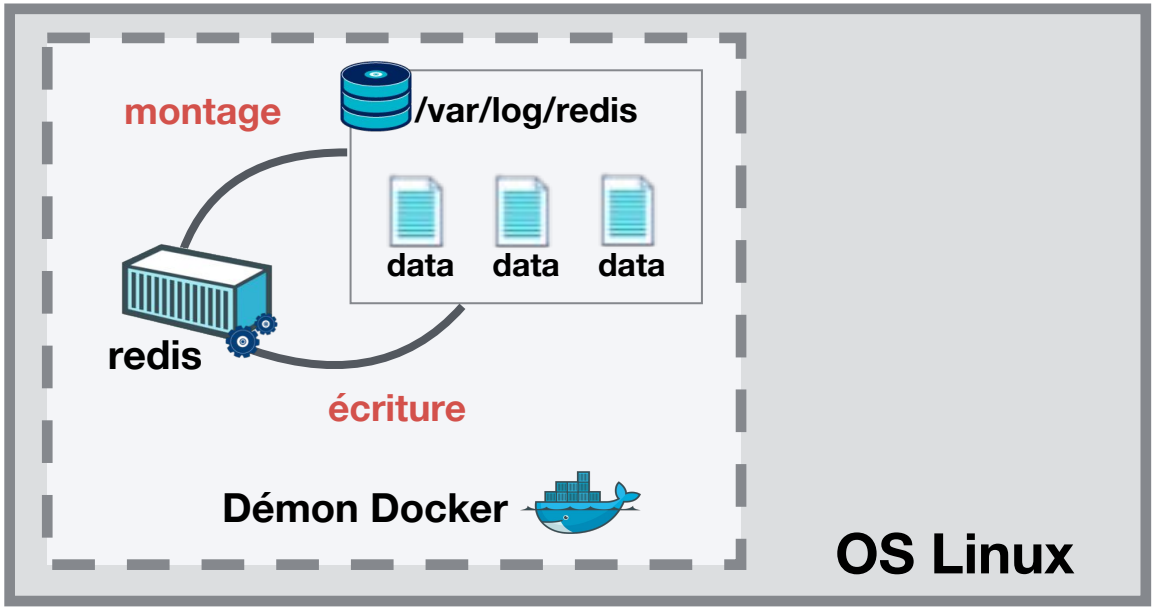
```
WORKDIR /var/www/html
```

Toutes les commandes postérieur à un **WORKDIR** auront lieu dans le répertoire pointé

DOCKERFILE - L'INSTRUCTION VOLUME

Déclare un volume secondaire au sein du conteneur pour sauvegarder des données (même effet que **docker container run -v <path> …**)

```
VOLUME ["/var/log/redis"]
```



DOCKERFILE - L'INSTRUCTION ENV

Place une variable d'environnement pour toutes les commandes **RUN** qui suivent et pour la commande qui sera lancée par le conteneur

```
ENV JAVA_HOME /usr/share/java
```

DOCKERFILE - L'INSTRUCTION USER

Spécifie le **user** à utiliser pour démarrer un conteneur d'application par CMD ou ENTRYPOINT

défaut = **root**

```
USER mongo # Not Kubernetes friendly
```

ou

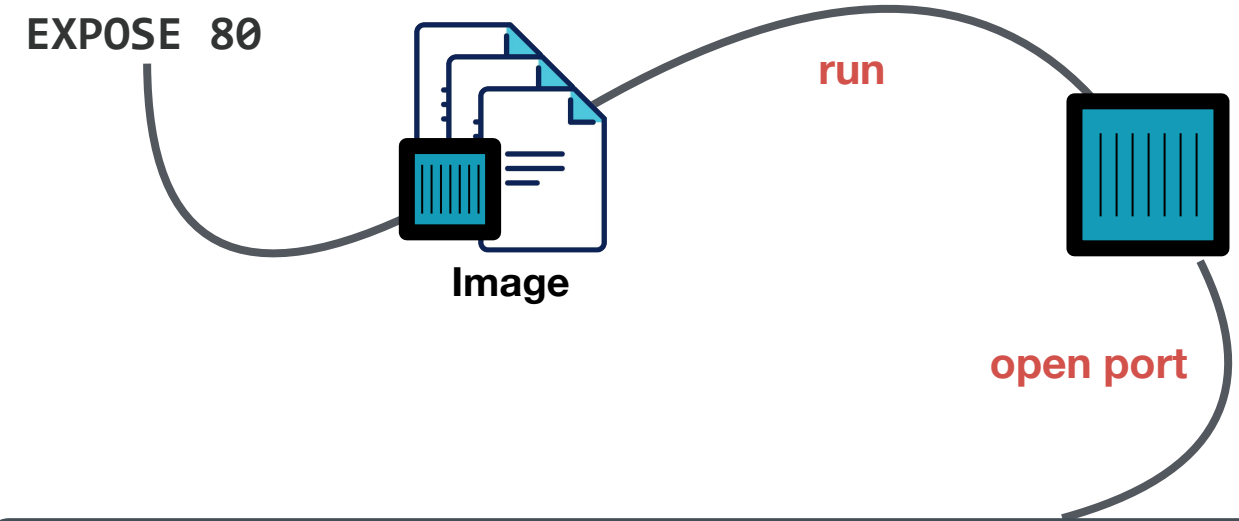
```
USER 1000 # More Kubernetes friendly
```

DOCKERFILE - L'INSTRUCTION EXPOSE

Documente le fait que l'application écoute sur ce port

Pour pouvoir l'exposer réellement, il faut le préciser au lancement du container :

docker run -d nginx -p 8080:80




```
$ iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 80 -j DNAT --to 192.168.1.2:8080 ACCEPT
```

DOCKERFILE - L'INSTRUCTION COPY ET ADD

COPY : Copie un fichier ou un dossier de l'Hôte dans le conteneur au chemin spécifié

```
COPY [--chown=<user>:<group>] <src>... <dest>
```

ADD : Ajoute un fichier ou un dossier dans le conteneur au chemin spécifié.
Le fichier source peut-être une URL.
Décompresse également les archives

```
ADD [--chown=<user>:<group>] <src>... <dest>
```

DOCKERFILE - LES IMAGES DE BASE

- Toutes les distributions historiques fournissent des images de base : Debian, CentOS, Ubuntu…

- Ces images peuvent être considérées comme trop riches et complexes dans le cadre de la conteneurisation

- Des images encore plus minimalistes sont apparues. Exemple :

DOCKERFILE - LE MULTISTAGE

- Depuis la version **17.05** de Docker Engine

- Objectif : faire des images finales les plus **légères** possibles et avec le **moins de failles** possible

- Principe : utiliser plusieurs images pour la construction d'une image définitive
  - Des images **intermédiaires** embarquent les **outils de build** (compilation, packaging, test, minification…)
  - L'image **finale** est allégée car ne contient que le **strict nécessaire** à l'**exécution** de l'application

DOCKERFILE - LE MULTISTAGE EN EXEMPLE

Image : > 700Mo

```
FROM golang:1.11 as builder
WORKDIR /go/src/
RUN go get -d -v golang.org/x/net/html
COPY app.go    .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .
```

Image : < 40Mo

```
FROM alpine:3.8
RUN apk --no-cache add ca-certificates
WORKDIR /
COPY --from=builder /go/src/app .
USER 1000
CMD ["./app"]
```

UN MOT SUR LES REGISTRES

- **Un référentiel centralisé** qui stocke et rend accessible toutes les images avec leur différentes couches

- **Accessible via une API REST** formalisée et connue de tous les clients Docker

- **Permet le partage d'images** avec communauté ou au sein d'une entreprise

- Plusieurs implémentations existantes
  - ***Docker Hub*** : registre public et gratuit
  - ***Docker Store*** : registre d'images payantes
  - ***Docker Trusted Registry*** : version entreprise on-premise
  - ***Docker Registry*** : implémentation open-source
  - ***Docker Registry chez les Cloud providers*** : Instanciation à la demande de Registries privées ou publiques (ex: AWS ECR Repository, GCP, Azure)
  - ***Nexus***, ***Artifactory*** peuvent également offrir le service de registre Docker
  - ***Gitlab*** (une registry Docker pour chaque répo de code)
  - ***Portus***, ***Harbor***

LE DOCKERHUB

- **Service de registre de Docker Inc. en mode SaaS**

- **Ouvert aux entreprises et aux utilisateurs**
  (plan gratuit pour un unique dépôt privé (image en plusieurs versions) et pour un nombre illimité de dépôts publics)

- **Utilisé pour la distribution des images officielles**
  (ubuntu, nodejs, ruby, gitlab…)

TP6

```
git clone https://3AIW:rBrPi4tgoyEHXjLivp9y@gitlab.com/santunes-formations/docker.git
```