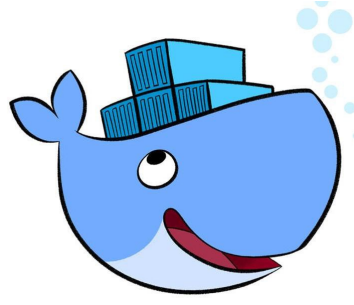
PRÉSENTATION DOCKER



CATI DIISCICO

Julien Cufi

julien.cufi@inrae.fr

04/02/2020

PRÉSENTATION DOCKER - CATI DIISCICO

1. Introduction et principes
2. Cas d'utilisation
3. Retours d'expérience

INTRODUCTION ET PRINCIPES

Qu'est-ce que c'est ?

Docker est une technologie permettant d'exécuter une application dans un environnement isolé, comprenant l'application mais également l'ensemble des dépendances nécessaires à son fonctionnement.

INTRODUCTION ET PRINCIPES

Quelques mots de vocabulaire...

- L'environnement dans lequel s'exécute l'application est appelé un ***conteneur Docker***.
- La matrice servant à définir ce qui est présent dans le conteneur est appelée une ***image Docker***.
- Un catalogue public d'images accessible sur le web permet de les mutualiser, ce catalogue s'appelle le ***DockerHub***.

INTRODUCTION ET PRINCIPES

Qui l'a crée et pourquoi ?

- Docker est un projet OpenSource sous licence Apache 2.0 crée en 2013 par Solomon Hykes.
- Le projet est supporté par la communauté et par l'entreprise Docker Inc.
- Crée a l'origine pour la société dotCloud (PaaS)

INTRODUCTION ET PRINCIPES

Comment ça marche ?

A l'origine Docker s'appuie sur LXC (LinuX container) et les fonctionnalités d'isolation du noyau Linux (cgroup, namespaces, ...) pour fournir un environnement d'exécution étanche.

⇒ Problème de portabilité pour Docker Inc.

INTRODUCTION ET PRINCIPES

- Remplacement de LXC par libcontainer
- Modularisation de la partie serveur
  - création d'outils spécifiques pour la création de conteneur (runc), et la gestion du cycle de vie (condainerd), ...
  - implémentation de référence des spécifications émises par Open Container Initiative...

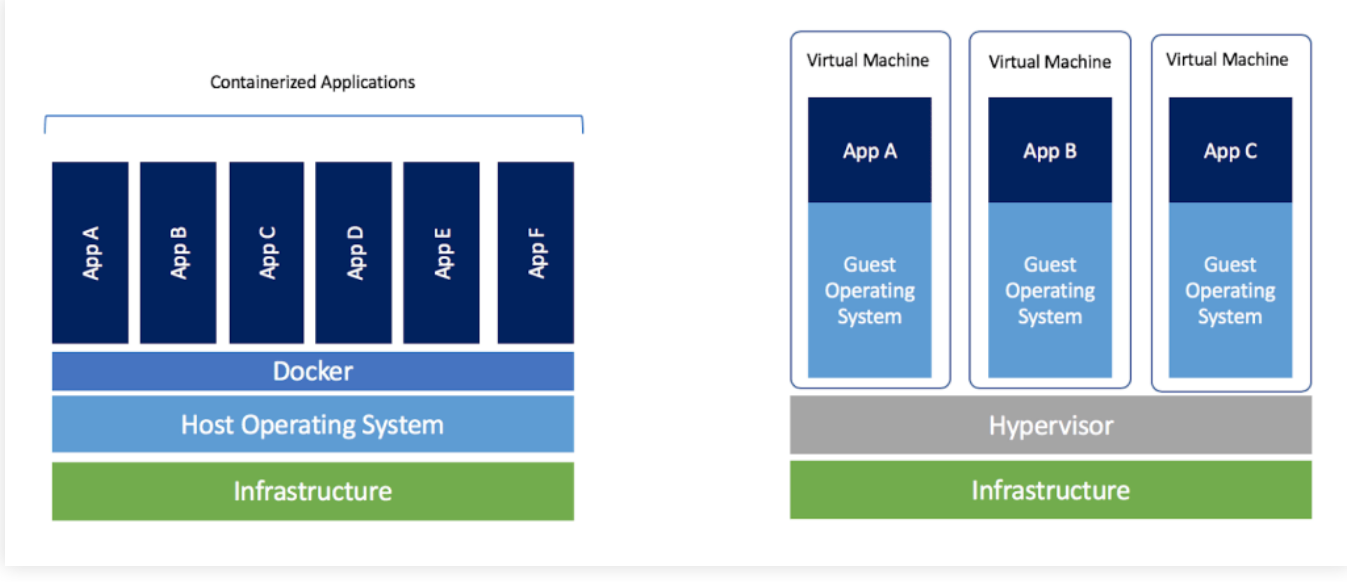
INTRODUCTION ET PRINCIPES

Open Container Initiative

- Projet crée en 2015 et supporté par la Fondation Linux
- Objectif : la normalisation des conteneurs
  - environnement d'exécution des conteneurs
  - images
- Crée par Docker Inc. avec quelques petites sociétés...

Je fais déjà ça avec mes machines virtuelles !

Dans une machine virtuelle, un hyperviseur simule une machine (*ie.* toute la partie hardware) et chaque machine virtuelle à son propre système d'exploitation ⇒ Surcoût



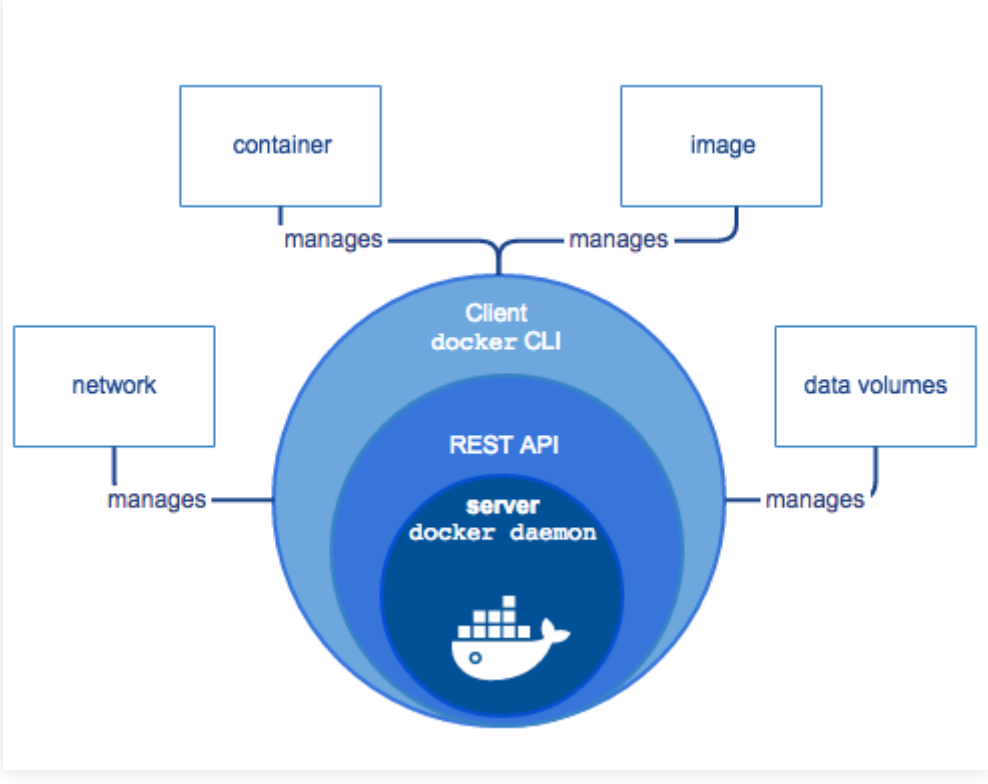
Container Model VS VM Model

INTRODUCTION ET PRINCIPES

Comment je l'utilise ?

- Docker est disponible sur Linux / Windows / Mac (VM)
  ⇒ requiert une version de windows avec Hyperviseur
- Deux versions EE et CE (≠ niveaux de support)
- L'utilisation se fera au travers de lignes de commandes

INTRODUCTION ET PRINCIPES



Docker s'appuie sur une architecture client/serveur, un client en ligne de commande envoie des instructions (via une API REST) au serveur (daemon docker / docker engine).

passons à la pratique...

CAS D'UTILISATION

Deux cas d'utilisation

- Pour un utilisateur qui souhaite tirer partie de Docker pour installer un logiciel et le tester
- Pour un développeur qui souhaite diffuser un logiciel

CAS D'UTILISATION

Quelques commandes de base

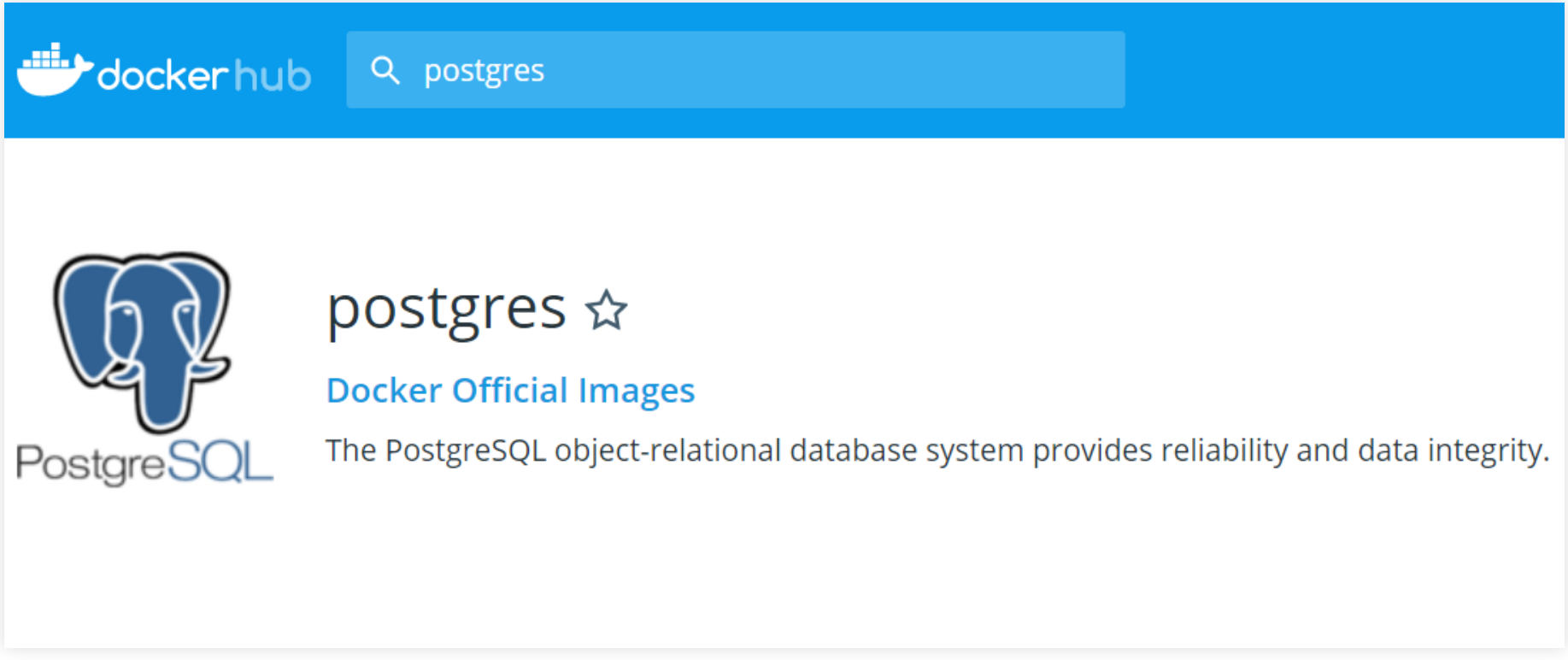
```
# Télécharger une image
$ docker image pull <image>

# Démarrer un conteneur
$ docker container run <image>

# Lister les conteneurs démarrés
$ docker container ps

# Stopper un conteneur
$ docker container stop <nom conteneur>

# Supprimer un conteneur
$ docker container rm <nom conteneur>
```

DOCKER : 1ER CAS D'UTILISATION

Je souhaite démarrer une base postgres v12 pour effectuer quelques tests.

- Recherche d'une image existante sur DockerHub (https://hub.docker.com/)
- Le DockerHub contient
  - des images officielles : vérifiées par Docker et à jour
  - des images non-officielles : le Far West

DOCKER : 1ER CAS D'UTILISATION



Capture du site DockerHub

DOCKER : 1ER CAS D'UTILISATION

Démarrage d'un conteneur basé sur l'image postgres:12

```
$ docker container run -it postgres:12
Unable to find image 'postgres:12' locally
12: Pulling from library/postgres
8ec398bc0356: Downloading [====>]  11.72MB/27.09MB
65a7b8e7c8f7: Download complete
b7a5676ed96c: Download complete
```

Le client demande le démarrage d'un conteneur, le serveur ne connaissant pas l'image il interroge le DockerHub et la télécharge.

DOCKER : 1ER CAS D'UTILISATION

Une fois l'image téléchargée le conteneur est démarré, la base est prête à être utilisée

```
PostgreSQL init process complete; ready for start up.
listening on IPv4 address "0.0.0.0", port 5432
database system is ready to accept connections
```

⇒ Youpi ?

DOCKER : 1ER CAS D'UTILISATION

Quelques particularités sur les conteneurs

- Isolé de l'hôte *par défaut*
  - Pas de communication réseau
  - Pas de partage de données
- Monoprocessus
- Doit être considéré comme éphémère
  - Le conteneur s'arrête lorsque le processus s'arrête
  - Données stockées dans le conteneur de manière temporaire

DOCKER : 1ER CAS D'UTILISATION

On recommence

```
# Création d'un volume
$docker volume create pgdata
pgdata
# Lancement du conteneur
$docker container run -it
                      -p 5432:5432
                      -v pgdata:/var/lib/postgresql/data postgres:12
```

-p 5432:5432

⇒ Association de port hôte/conteneur

-v pgdata:/var/lib/postgresql/data postgres:12

⇒ Montage d'un volume partagé

DOCKER : 1ER CAS D'UTILISATION

Pour les curieux

```
$ docker volume inspect pgdata
[
    {
        "CreatedAt": "2019-02-26T17:12:59+01:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/pgdata/_data",
        "Name": "pgdata",
        "Options": {},
        "Scope": "local"
    }
]
$ ls /var/lib/docker/volumes/pgdata/_data
postgresql.conf base     pg_commit_ts  pg_ident.conf  pg_notify
...
```

Remarque : la suppression du conteneur n'entraine pas la suppression du volume.

DOCKER : 2EME CAS D'UTILISATION

J'ai implémenté un algorithme, je souhaite le mettre à disposition.

Objectif : Faciliter la reproductibilité des résultats en minimisant les étapes d'installation* du logiciel

⇒ Nécessite de créer une image Docker propre à son logiciel

* mais il y en aura toujours

DOCKER : CRÉATION D'IMAGE 1/4

- Une image docker est un fichier texte nommé DockerFile respectant un langage propre à Docker
- Une image peut "hériter" d'une image existante pour l'étendre
- Le DockerFile contient l'ensemble des instructions nécessaires à l'installation du logiciel
- Il est nécessaire de lancer l'assemblage de l'image a partir du fichier DockerFile
- L'image réalisée peut rester en local, *i.e* sans être partagée sur le DockerHub

DOCKER : CRÉATION D'IMAGE 2/4

Exemple de fichier Dockerfile :

```
FROM ubuntu:latest
RUN apt-get update && \
    apt-get install -y cowsay
ENTRYPOINT ["/usr/games/cowsay"]
```

- FROM : Indique de quelle image existante l'on hérite
- RUN : Permet de lancer des commandes d'installation
- ENTRYPOINT : Définit le point d'entrée du conteneur

DOCKER : CRÉATION D'IMAGE 3/4

Assemblage de l'image :

```
$ docker image build -t cow .
Step 1/3 : FROM ubuntu:latest
 ---> dd6f76d9cc90
Step 2/3 : RUN apt-get update &&     apt-get install -y cowsay
 ---> Running in 9a0c163a5579
...
Step 3/3 : ENTRYPOINT ["/usr/games/cowsay"]
 ---> Running in c17aa839e8a8
Removing intermediate container c17aa839e8a8
 ---> 000e5e657c8d
Successfully built 000e5e657c8d
Successfully tagged cow:latest
```

DOCKER : CRÉATION D'IMAGE 4/4

Utilisation de l'image

```
$ docker container run cow "Je suis une vache"
 ______________________
< Je suis une vache >
 ----------------------
        \    ^__^
         \   (oo)\_______
             (__)\       )\/\
                 ||----w |
                 ||     ||
```

DOCKER : 2EME CAS D'UTILISATION

J'ai implémenté un algorithme, je souhaite le mettre à disposition.

Cette implémentation requiert :

- un jeu de données de test
- une version de java, de maven / ant

DOCKER : 2EME CAS D'UTILISATION

```
# Le fichier Dockerfile
FROM maven:3.6.2-jdk-8
RUN mkdir -p /app/results && \
    mkdir -p /app/src && \
    mkdir -p /app/data
COPY data /app/data
COPY src /app/src/
COPY pom.xml build.xml /app/
RUN gzip -d /app/data/FoodOnAgroPortalImport2.nq.gz && \
    chmod -R 755 /app
WORKDIR /app
ENTRYPOINT ["mvn", "package", "exec:java", ..."]
VOLUME [ "/app/results"]
```

```
# Installation du logiciel
$ docker image build -t align-tool .
# Lancement du logiciel sous linux
$ docker container run --rm -it -v ${pwd}/results:/app/results align-
# sous windows
$ docker container run --rm -it -v %cd%/results:/app/results align-to
```

RETOUR D'EXPÉRIENCE

Notre besoin :

- Mise en place de deux plateformes (test et production) avec nos applications
- Automatiser l'installation
  - applications web JAVA, Ruby, Python
  - base de données relationnelles, sémantiques, NoSQL
  - serveur de calcul R
- Mutualiser les installations
- Gérer "proprement" les différentes versions des dépendances (ex: JAVA)

RETOUR D'EXPÉRIENCE

- Complexité
  - Unix : lu, parlé, écrit
  - Problèmes liés au fait que docker soit monoprocessus
  - Impossibilité de charger des modules dans le kernel (modprobe)
  - Beaucoup de commandes ...
- Projet en constante évolution

RETOUR D'EXPÉRIENCE

- Bonnes pratiques (communes)
  - Bon sens : Ne pas récupérer aveuglement des images sur le DockerHub
  - Ne pas monter la racine / dans le conteneur
  - Groupe et utilisateurs dédiés
  - Ne pas surcharger le conteneur avec des paquets inutiles
  - Logiciel à maintenir à jour

RETOUR D'EXPÉRIENCE

- Sécurité
  - Nécessiterait une présentation dédiée !
  - Ne nous affranchit pas du respect des règles de sécurité en vigueur
  - Docker requiert des droits élevés : les conteneurs sont exécutés par root

    ⇒ Directive USER dans le DockerFile, configuration du serveur dockeremap

Merci de votre attention !

Des questions ?

LIENS

- Documentation Docker : https://docs.docker.com/
- Docker security bench : https://github.com/docker/docker-bench-security
- Lien vers projet Docker OpenSource : https://github.com/moby
- Lien RedHat "VM vs container" : https://www.redhat.com/fr/topics/containers/whats-a-linux-container
- Lien vers Open Container Initiative : https://www.opencontainers.org/

ANNEXE AUFS

```
$ docker image pull mongo
Using default tag: latest
latest: Pulling from library/mongo
5c939e3a4d10: Pull complete
c63719cdbe7a: Pull complete
...
Digest: sha256:a1681be5c90348e576966
Status: Downloaded newer image for mongo:latest
```

- Une image est décomposée en couches
- L'union des différentes couches produit la couche finale sur laquelle l'application s'exécutera
- AUFS (Advanced multi layered Unification FileSystem)
- Docker utilise un drive spécifique pour écrire dans ce FS ⇒ Eviter les IO sur AUFS : peu performant mieux vaut privilégier l'écriture dans des volumes

ANNEXE DOCKER COMPOSE

- En pratique l'utilisateur de Docker a besoin :
  - de passer des paramètres au conteneur (plusieurs volumes, ports, nom, réseau...)
  - de démarrer plusieurs conteneurs avec des relations de dépendances en même temps

⇒ Difficilement réalisable en ligne de commande...

Solution : docker-compose

ANNEXE DOCKER COMPOSE

```
version: "3.0"
services:
  centos-efactor:
    container_name: tomcat-efactor
    image: centos-efactor:latest
    networks:
      - ico-network
    depends_on:
      - centos-rserve
  centos-rserve:
    image: centos-rserve:latest
    volumes:
      - rserve:/usr/local/efactor/rserve
    networks:
      - ico-network
```

Extrait d'un fichier docker-compose.yml

```
# Lancement des conteneurs
$ docker-compose up
```

ANNEXE DOCKER COMPOSE