

# Système d'exploitation

## Introduction à Docker

Juan Angel Lorenzo del Castillo

ING1 Informatique - Mathématique appliquée  
2022-2023

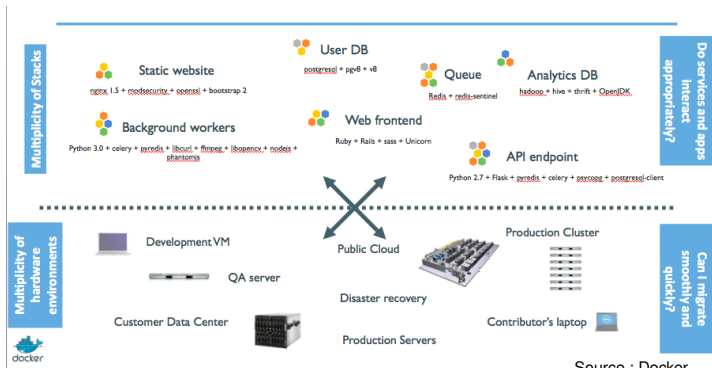


# Plan

- 1 Introduction
- 2 Docker
- 3 Installation et configuration de Docker
- 4 Création et gestion des conteneurs

# Pourquoi les conteneurs ?

Le problème de déploiement dans l'industrie du logiciel



## Avant

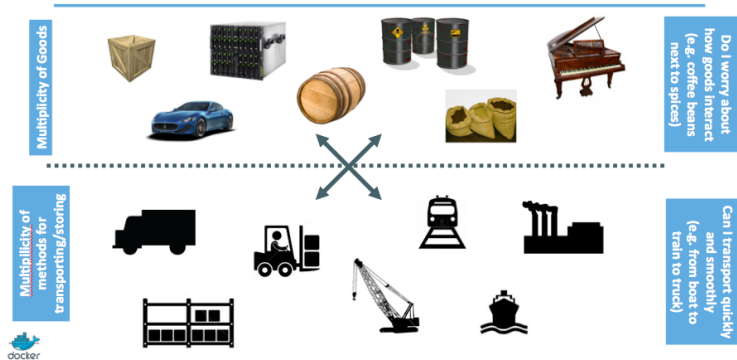
- applications monolithiques
- long cycles de développement
- un seul environnement
- passage à l'échelle lent

## Maintenant

- services découplés
- améliorations rapides et itératives
- plusieurs environnements
- passage à l'échelle horizontale rapide

# Pourquoi les conteneurs ?

Une analogie (de l'histoire ancienne)



Source : Docker

# Pourquoi les conteneurs ?

## Conteneurs maritimes intermodaux



Source : Docker

# Pourquoi les conteneurs ?

Cela a engendré un écosystème de conteneurs maritimes



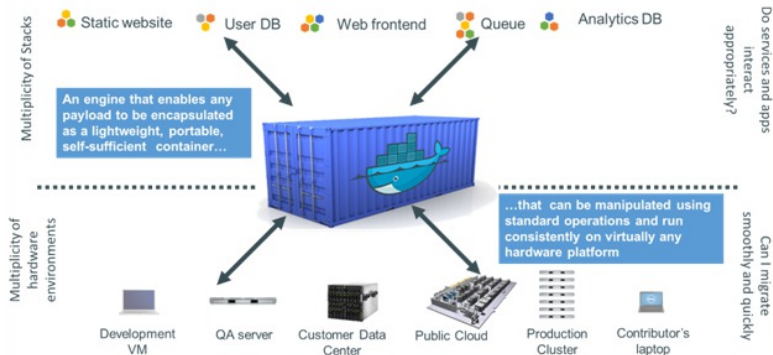
- 90% of all cargo now shipped in a standard container
  - Order of magnitude reduction in cost and time to load and unload ships
  - Massive reduction in losses due to theft or damage
  - Huge reduction in freight cost as percent of final goods (from >25% to <3%)
- massive globalization
- 5000 ships deliver 200M containers per year



Source : Docker

# Pourquoi les conteneurs ?

## Un système de conteneurs pour les applications



Source : Docker

# Comment fonctionne un conteneur

Permet d'isoler l'exécution des applications dans des contextes d'exécution.

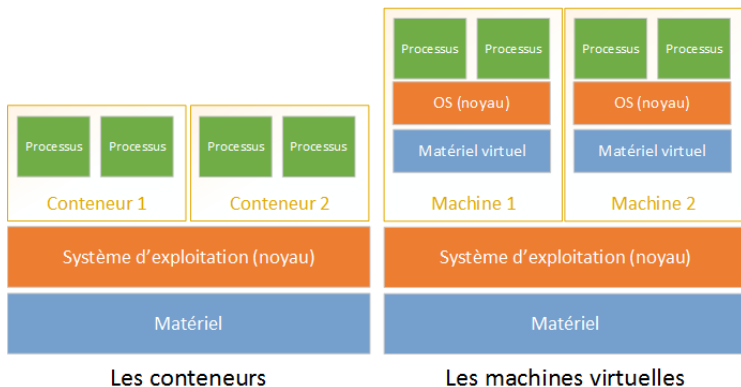
## Similaire à la virtualisation, mais sans virtualisation :

- Agnostique sur le contenu et le transporteur.
- Isolation et automatisation.
- Principe d'infrastructure consistante et répétable.
- Peu d'overhead par rapport à une VM.
- En gros, un super chroot (*chroot on steroids*).
- L'un des points forts de Solaris depuis plusieurs années.
- Technologie utilisée chez Google avec son scheduler **Borg** depuis longtemps.

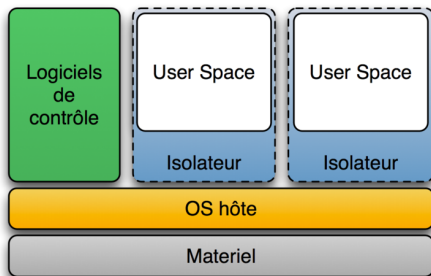
Inspiré par les slides de Johan Moreau (IRCAD/IHU), JDev 2015.



# Différences entre une VM et un conteneur



# Chroot on steroids

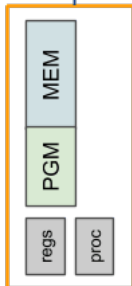


- Avec son propre space de processus
- Avec sa propre interface de réseau
- Sans son propre `/sbin/init`
- Processus isolés
- Kernel partagé avec le host
- Permettent la création de plusieurs environnements similaires, avec des versions du logiciel et configurations identiques

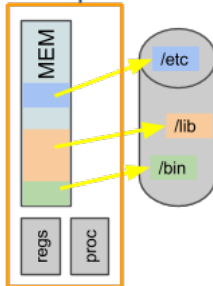
# Différences entre processus et conteneur

## Containers vs. Processes

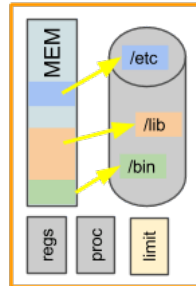
textbook process



real process



container

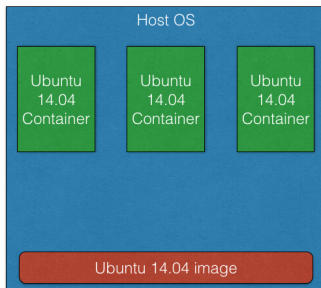


Source : rightscale.com

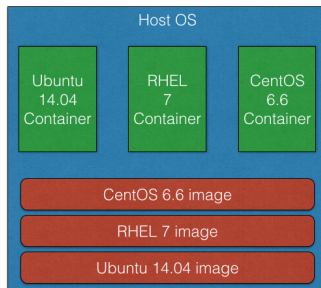
# Cas d'utilisation des conteneurs

## 1 Conteneur de Système d'exploitation

- ▶ Environnement virtuel qui partage le kernel avec le SE du host
- ▶ Mais isolé dans l'espace d'utilisateur
- ▶ On exécute plusieurs processus et services
- ▶ Pratique pour exécuter différentes distributions en utilisant des images (modèles)



Identical OS containers



Different flavoured OS containers

# Cas d'utilisation des conteneurs

## 2 Conteneur d'application

- ▶ Un seul service ou application par conteneur
- ▶ **Microservices** : Décomposer une application grande dans plusieurs services petits
- ▶ Au lieu de mettre à jour toute une application, on met à jour les services concernés

# Plan

- 1 Introduction
- 2 Docker**
- 3 Installation et configuration de Docker
- 4 Création et gestion des conteneurs

# Contributions de Docker

## Avant Docker

- Pas de format d'échange standardisé (les conteneurs n'étaient pas portables).
- Les conteneurs sont difficiles à utiliser par les développeurs.
- Dependency hell : "*Ça marche dans ma machine.*"
- Pas de composants réutilisables, APIs ou des outils.
- *Analogie*: Les conteneurs maritimes ne sont pas que des caisses en acier. Ce sont des boîtes en acier avec une taille standard, les mêmes crochets et trous.

## Après Docker

- Format de conteneur standardisé.
- Rend les conteneurs faciles à utiliser par les développeurs.
- Expédition des images de conteneurs avec toutes leurs dépendances.
- Accent sur les composants réutilisables, les API, un écosystème d'outils standards.
- Amélioration par rapport aux outils ad-hoc, internes, spécifiques.

# C'est quoi Docker ?

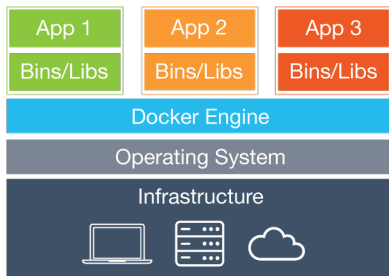
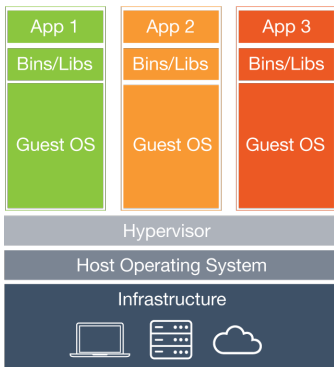
Docker Engine + Docker Hub = Docker Platform

- **Docker Engine** : Exécute les conteneurs
  - ▶ Écrit en Go
  - ▶ API REST
  - ▶ Construction des images
  - ▶ Partage d'images en utilisant des registres
- **Docker Hub** : Facilite la migration
  - ▶ Registres (repos) publiques
  - ▶ Registres privés
  - ▶ Construction automatique du logiciel



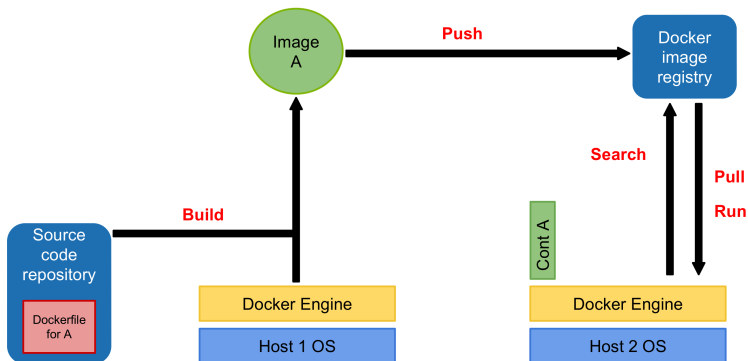


# C'est quoi Docker ?



Source : Docker

# Docker pour la gestion du logiciel



Source : Treeptik

# Plan

- 1 Introduction
- 2 Docker
- 3 Installation et configuration de Docker**
- 4 Création et gestion des conteneurs

# Installation de Docker

- Par défaut, Docker est déjà installé sur votre ordinateur portable CY-Tech. À vérifier avec :

```
$ docker version # il faudra peut-être le préfixer avec sudo
```

si vous obtenez une réponse sans erreur, il n'y a rien d'autre à faire.

- Sinon, pour installer Docker sur votre portable Linux ou sur une machine virtuelle, il y a deux alternatives :
  - ▶ Installation depuis les paquets des distros

```
$ sudo yum install docker # Red Hat et dérivés  
$ sudo apt-get install docker.io # Debian et dérivés
```

- ▶ Utiliser le script d'installation de Docker. Disponible pour Ubuntu, Debian, Fedora et Gentoo :

```
$ curl -s https://get.docker.com/ | sudo sh
```

# Configuration de Docker

- Le moteur Docker exécute un client et un serveur.
- L'utilisateur Docker est `root` équivalent
- Il fournit un accès de niveau racine à l'hôte.
- Si votre utilisateur n'est pas dans le groupe Docker, vous devrez préfixer chaque commande avec `sudo`; par exemple. `sudo docker version`.
- Pour éviter cela, ajoutez votre utilisateur au groupe Docker :

- ▶ Ajouter le groupe `docker`

```
$ sudo groupadd docker
```

- ▶ Ajouter votre utilisateur au groupe `docker`

```
$ sudo gpasswd -a $USER docker
```

- ▶ Redémarrer le *daemon* `docker`

```
$ sudo service docker restart
```

# Plan

- 1 Introduction
- 2 Docker
- 3 Installation et configuration de Docker
- 4 Création et gestion des conteneurs**

# Création d'un conteneur

```
$ docker run -i -t ubuntu /bin/bash
```

- **run** : lance un nouveau conteneur
- **-i -t** : demande une terminal en mode interactif dans une pseudo-terminal
- **ubuntu** : l'image à utiliser pour ce conteneur. Si elle n'est pas disponible sur votre ordinateur, elle sera téléchargée du *Docker Hub*.
- **/bin/bash** : exécute bash dans le conteneur

```
$ docker run -i -t ubuntu /bin/bash
root@0bc82356b52d9 :/# cat /etc/issue
Ubuntu 14.04.2 LTS
root@0bc82356b52d9 :/# exit
```

# Création d'un conteneur

- Tapez `exit`. Votre conteneur est maintenant dans un état *stopped*. Il existe toujours sur le disque, mais toutes les ressources ont été libérées.
  - ▶ À partir de la ligne de commande de l'hôte, répertoriez les conteneurs actifs avec `docker ps`. Ensuite, exécutez `docker ps -a`. Pourquoi la sortie diffère-t-elle ?
  - ▶ Si vous démarrez un nouveau conteneur avec `docker run`, nous démarrons un tout nouveau conteneur à partir de l'image *ubuntu*.
- Pour revenir à notre conteneur arrêté, depuis notre hôte nous devons :
  - 1 Trouver l'ID du conteneur :

```
$ docker ps -a
CONTAINER ID        IMAGE
0bc82356b52d9      ubuntu
```

- 2 Démarrer le conteneur

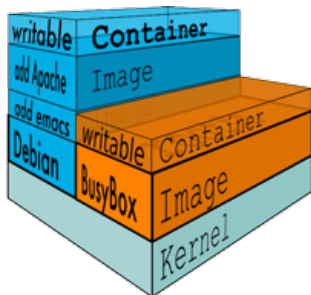
```
$ docker start 0bc82356b52d9
```

- 3 Attachez le conteneur

```
$ docker attach 0bc82356b52d9
```



# Images



- Collection de **fichiers** de seule lecture
- **Images base** sur lesquelles on construit les autres images
- Images en **couches**, conceptuellement empilées les unes sur les autres.
- Chaque couche peut ajouter, modifier et supprimer des fichiers. C'est un *différentiel* de la couche précédente.
- Les images peuvent partager des couches pour optimiser l'utilisation du disque, les temps de transfert, l'utilisation de la mémoire, etc.
- **Conteneur** : Ensemble de processus et fichiers modifiés (lecture-écriture)

# Management des images

- Pour lister toutes nos images locales (images stockées dans notre hôte Docker) :

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
sameersbn/skype	1.0.1-3	4c06c9bb3da0	5 months ago	411.5 MB
ubuntu	latest	c73a085dc378	10 months ago	127.1 MB
jpetazzo/clock	latest	12068b93616f	2 years ago	2.433 MB

- Pour rechercher des images sur un registre distant (nous ne pouvons pas lister toutes les images, nous recherchons par mots-clés)

```
$ docker search wordpress
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
wordpress	The WordPress rich ...	1826	[OK]	
bitnami/wordpress	Bitnami Docker Ima...	42		[OK]
...				

- ▶ Les images *officielles* sont celles de l'espace de noms racine.
  - ▶ Les images *automated* sont construites automatiquement par le Docker Hub (avec la recette de construction toujours disponible).
- Pour supprimer un conteneur ou une image :
  - ▶ `docker rm containerID` supprime un conteneur, **pas** une image.
  - ▶ `docker rmi imageID` supprime une image.