



# docker

Serge NOEL

[Serge.noel@net6a.com](mailto:Serge.noel@net6a.com)

© 2015-2018 Serge NOEL

# Votre formation - Présentation

## Public :

Développeurs, architectes techniques, administrateurs et responsables d'exploitation et de production, chefs de projet.

## Objectifs :

- Installer Docker sous Linux et Windows
- Travailler avec des conteneurs et images
- Construire des images et les publier sur le Docker Hub
- Configurer le réseau et les volumes

# Votre formation - Présentation

## Pré-requis :

- Connaissances systèmes Linux/Windows
- Notions sur les réseaux TCP/IP
- Utilisation de la ligne de commande et du script Shell en environnement Linux / Powershell en environnement Windows
- Windows 10
- Visual studio 2017
- Connaissances basiques en développement

# Votre formation - Formateur

## Son Nom :

- Serge NOEL

## Qualifications :

- Consultant / Chef de projet depuis 1990.

## Compétences pour ce cours :

- Expert Unix depuis 1992, Expert Linux depuis 1998.
- Expert Microsoft depuis 1989
- Développeur Php/ C/ Javascript
- Expert KVM / vmWare / Docker / (OpenStack)

# Votre formation - Vous

- Votre Nom
- Votre fonction
- Vos expériences
- Vos attentes

# **Votre formation - Logistique**

## **Horaires de la formation**

- 9h00-12h30
- 14h00-17h30

## **Pauses**

- 2 x 15 min

**Merci d'éteindre vos téléphones portables**



# Votre formation - Programme

- De la virtualisation à Docker
- Présentation de Docker
- Mise en œuvre en ligne de commande
- Création de conteneur personnalisée
- Mettre en œuvre une application multiconteneurs
- Interfaces d'administration
- Administrer des conteneurs en production
- Orchestration et clusterisation

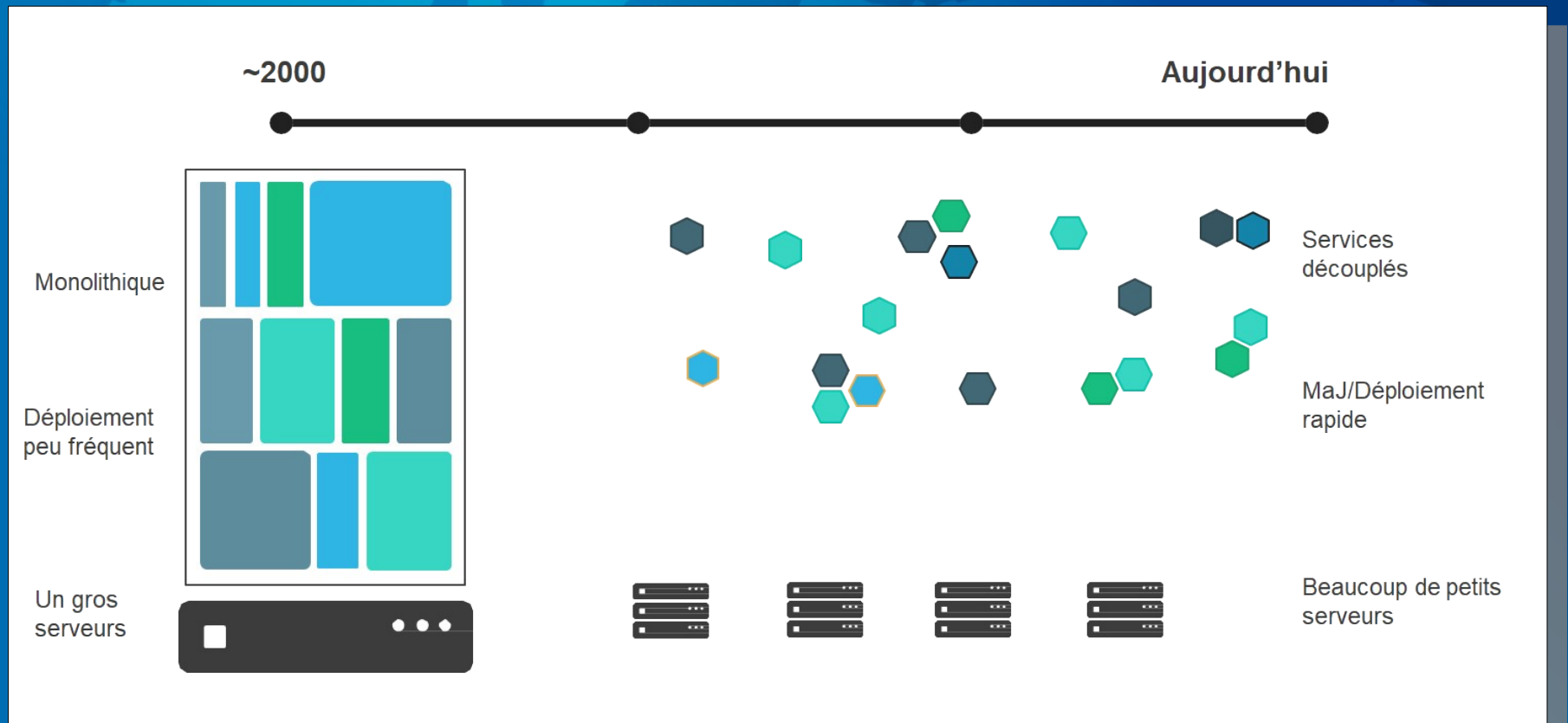
# De la virtualisation à Docker

1. Les différents types de virtualisation.
2. La conteneurisation : LXC, namespaces, control-groups.
3. L'évolution de DotCloud à Docker.
4. Le positionnement de Docker.
5. Docker versus virtualisation.



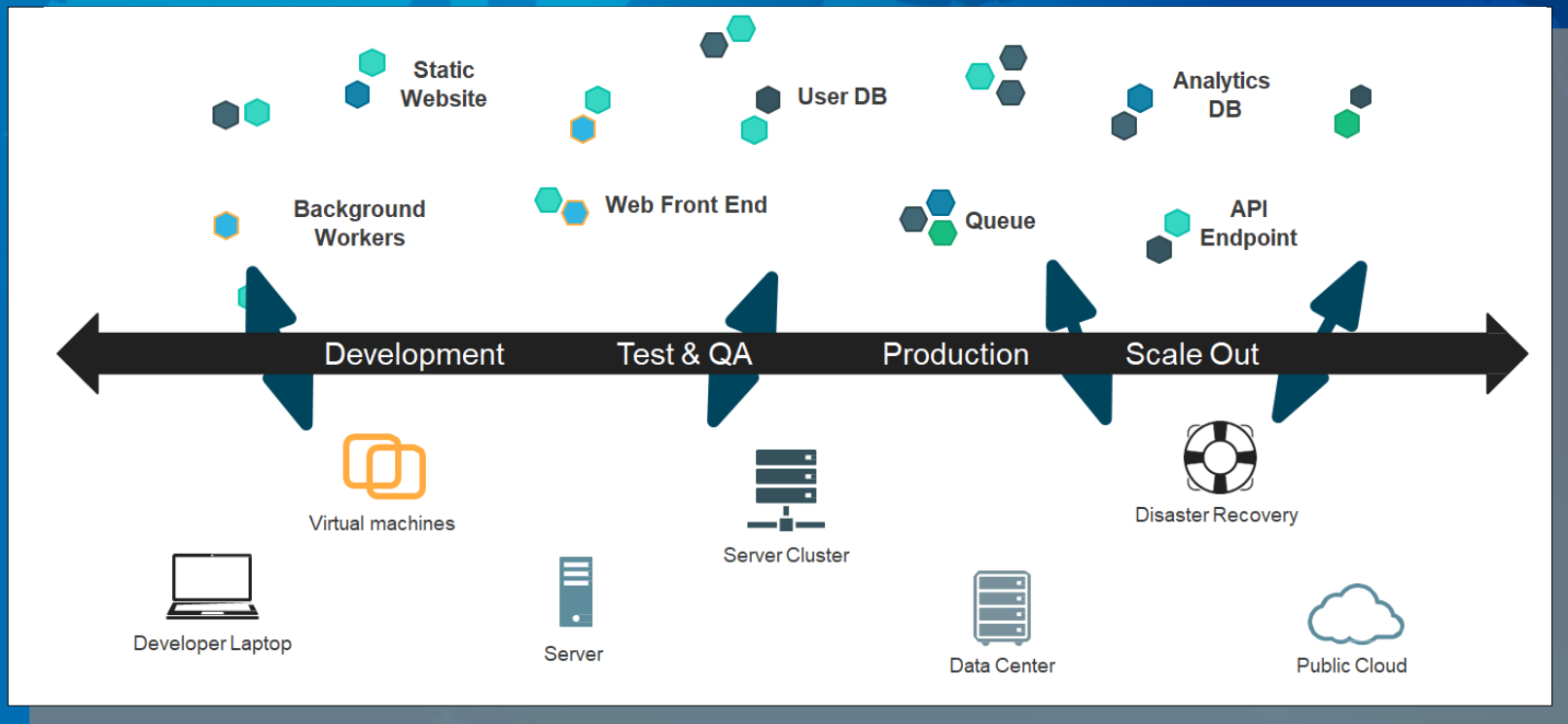
# De la virtualisation à Docker

L'architecture des applications change rapidement



# De la virtualisation à Docker

## Environnements et technologies ultra hétérogènes



# De la virtualisation à Docker

## Analogie – origine du nom

### Cargo Transport Pre-1960

Multiplicity of Goods



Do I worry about  
how goods interact  
(e.g. coffee beans  
next to spices)

Multiplicity of  
methods for  
transporting/storing



Can I transport quickly  
and smoothly  
(e.g. from boat to train  
to truck)

# De la virtualisation à Docker

## Analogie – origine du nom



# De la virtualisation à Docker

## Solution: Des conteneurs Docker

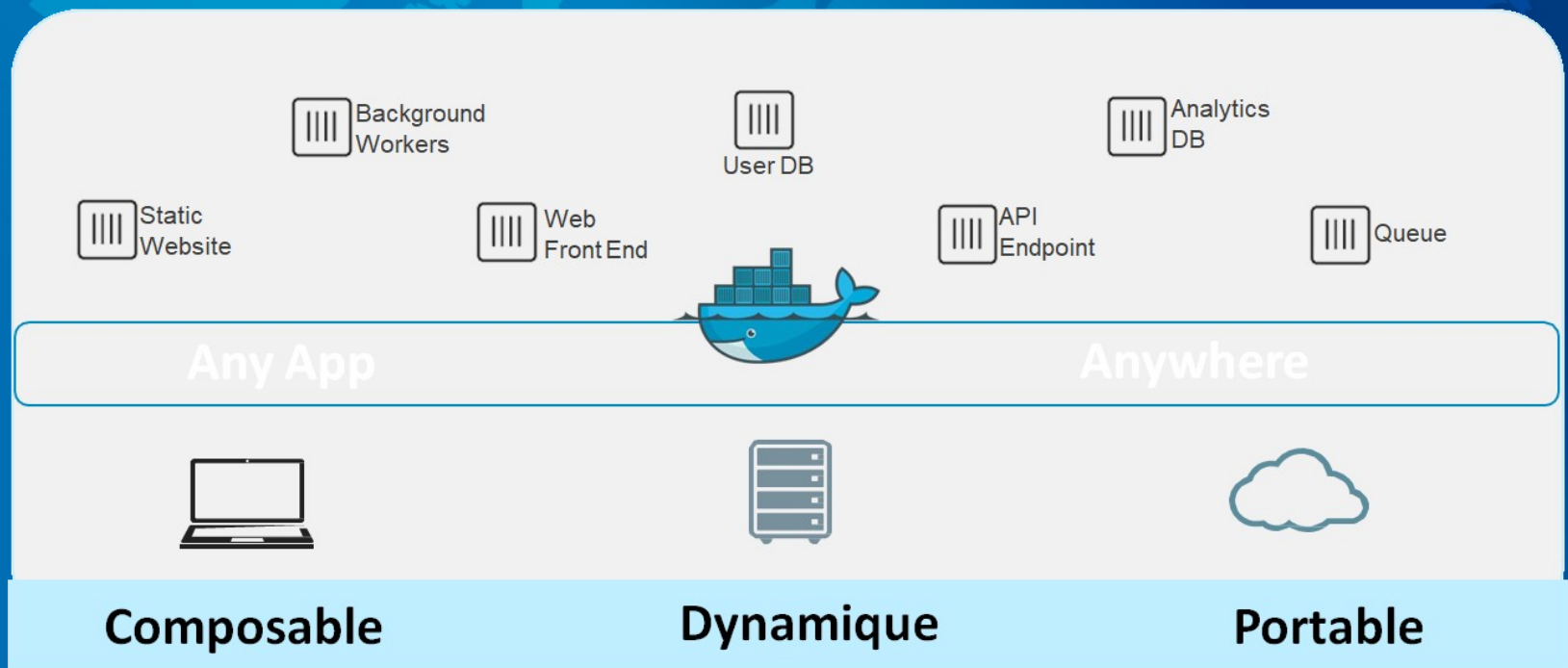


- Package : une application et ses dépendances
- Isoler les applications les unes des autres
- Agnostique sur le contenu
- Création d'un standard pour le format des containers (OCI)
- Facilement portable sur différents environnements
- Flexibilité/modularité des composants
  - Ex. toolkit comme <https://mobyproject.org/>
- Automatisable/Scriptable



# De la virtualisation à Docker

**Solution:** Des conteneurs Docker





# De la virtualisation à Docker

Du point de vue du développeur ...

## **Build once... run anywhere**

- un environnement portable pour l'exécution des apps
- Pas de risque d'oublier des dépendances, packages, ... durant les déploiements
- Chaque application s'exécute dans son propre conteneur : avec ses propres versions des librairies
- Facilite l'automatisation des tests
- élimine les problèmes de compatibilité entre les plateformes
- Coût ressource très bas pour lancer un container. On peut en lancer des dizaines sur un poste développeur (laptop)
- Permet de tester des technologies ou faire des prototypes rapidement et à très bas coût.

# De la virtualisation à Docker

Du point de vue de l'admin sys ...

**Configure once...run anything**

- Rend le workflow plus consistant, prédictible, répétable
- Élimine les inconsistances entre les environnements de dev/test/prod
- Améliore la rapidité et la fiabilité du déploiement continu (continuous deployment)
- Réduction des pb de performances (Ex. avec les VM); réduction des coûts (hébergements cloud, ...)

# De la virtualisation à Docker

## Les différents types de virtualisation.

- VMWare
- Xen (para-virtualisation)
- Hyper-V
- KVM
- Oracle
- Proxmox
- ...

Reproduction d'une machine dans le userspace de l'OS

Implique un OS complet virtualisé : userspace, kernel, ...

# De la virtualisation à Docker

## Les différents types de virtualisation.

**Virtualisation** : noyau en espace utilisateur

- L'OS invité tourne comme un logiciel en espace utilisateur
- Peu performant : un noyau sur un noyau Pas d'isolation ni d'indépendance
- Utile pour les développeurs de noyau (Linux, BSD, ...)

**Exemple** : Qemu

# De la virtualisation à Docker

## Les différents types de virtualisation.

### **Virtualisation** : hyperviseur type 2

- Émule des machines pour les OS invités
- Peu performant : un OS sur une machine émulée dans un OS sur une vraie machine (ou une autre machine émulée)
- Très bonne isolation : une faille dans un invité ne peut mener au système host (sauf si faille dans l'émulateur)
- Permet de virtualiser un OS différent du host
- Utile pour du test, du maquettage ...

Exemples : virtualBox, vmWare Workstation, ...



# De la virtualisation à Docker

## Les différents types de virtualisation.

### **Virtualisation** : hyperviseur type 1

- L'OS laisse un accès presque direct au matériel physique et autorise un système « maître »
- Plutôt performant : accès direct au matériel
- Coûteux : nécessite des machines physiques
- Bonne isolation sauf en cas de faille matérielle
- Utile en datacenter, solution très réputée jusque maintenant

Exemples : KVM, vmWare Esx, Hyper-V, ...



# De la virtualisation à Docker

## Les différents types de virtualisation.

### Virtualisation → **Conteneurs**

- Le système invité partage le noyau du système host
- Très performant : tout passe par le même noyau
- Peu coûteux : s'intègre au host, peut se partager sur plusieurs serveurs
- Peu isolant : une faille peut mener à une prise de privilège (escalation)
- Utile du développement à la production en passant par les tests, solution de plus en plus utilisée

Exemples : openVZ, Lxc, Docker

# De la virtualisation à Docker

## Les différents types de virtualisation.

### **Conteneurs** : multiples technologies

- Pas seulement Docker : terme générique désignant les technologies intégrées au noyau Linux depuis ~10 ans
- Les conteneurs sous Microsoft Windows Server depuis Windows 2016 serveur

# De la virtualisation à Docker

**La conteneurisation : LXC, namespaces, control-groups**

**LXC**, contraction de l'anglais **Linux Containers** est un système de virtualisation, utilisant l'isolation comme méthode de cloisonnement au niveau du système d'exploitation.

Il est utilisé pour faire fonctionner des environnements Linux isolés les uns des autres dans des conteneurs partageant le même noyau et une plus ou moins grande partie du système hôte.

Le conteneur apporte une virtualisation de l'environnement d'exécution (processeur, mémoire vive, réseau, système de fichier...) et non pas de la machine. Pour cette raison, on parle de « conteneur » et non de « machine virtuelle ».

# De la virtualisation à Docker

**La conteneurisation : LXC, namespaces, control-groups**

**LXC** repose sur les fonctionnalités des cgroups du noyau Linux disponibles depuis sa version 2.6.24. Il repose également sur d'autres fonctionnalités de cloisonnement comme le cloisonnement des espaces de nommage du noyau, permettant d'éviter à un système de connaître les ressources utilisées par le système hôte ou un autre conteneur.

Docker est un gestionnaire de conteneurs initialement basé sur LXC.

# De la virtualisation à Docker

**La conteneurisation : LXC, namespaces, control-groups**

Isolation par espace de nommage

- Des ensembles de processus sont séparés de telle façon qu'ils ne puissent pas « voir » les ressources des autres groupes.

Un espace de nommage par identifiant de processus (PID) fournit un ensemble distinct d'identifiants de processus dans chaque espace de nommage.

Sont aussi disponibles des espaces de nommage par mount, UTS (**U**nix **T**imesharing **S**ystem), réseau et SysV IPC.

Les espaces de nom sont créés avec la commande « unshare » ou un appel système, ou en tant que nouveaux marqueurs dans un appel système « clone ».



# De la virtualisation à Docker

**La conteneurisation : LXC, namespaces, control-groups**

L'un des buts de la conception de cgroups a été de fournir une interface unifiée à différents cas d'utilisation, en allant du contrôle de simples processus (comme **nice**) à la virtualisation au niveau du système d'exploitation (comme OpenVZ, Linux-VServer, LXC).



# De la virtualisation à Docker

La conteneurisation : LXC, namespaces, control-groups

Cgroups fournit :

- **Limitation des ressources** : des groupes peuvent être mis en place afin de ne pas dépasser une limite de mémoire
- **Priorisation** : certains groupes peuvent obtenir une plus grande part de ressources processeur ou de bande passante d'entrée-sortie.

# De la virtualisation à Docker

La conteneurisation : LXC, namespaces, control-groups

Cgroups fournit :

- **Comptabilité** : permet de mesurer la quantité de ressources consommées (facturation).
- **Isolation** : séparation par espace de nommage pour les groupes
- **Contrôle** : figer les groupes ou créer un point de sauvegarde et redémarrage

# De la virtualisation à Docker

## L'évolution de DotCloud à Docker

### DotCloud

- Plate-forme Cloud
- Hébergement de services dans des langages différents Performance native
- Créé par un ancien étudiant EPITECH en 2008
- Rejoint un incubateur en 2010, la Silicon Valley en 2011 Open-sourcing en 2013

# De la virtualisation à Docker

## L'évolution de DotCloud à Docker



**Docker** a été développé par **Solomon Hykes** pour un projet interne de **dotCloud** : une société proposant une plate-forme en tant que service.

Docker est une évolution basée sur les technologies propriétaires de dotCloud, elles-mêmes construites sur des projets open source.

Docker a été distribué en tant que projet open source à partir de **mars 2013**.

# De la virtualisation à Docker

## L'évolution de DotCloud à Docker

- 18 novembre 2013, mis en favoris plus de 7 300 fois sur GitHub (14e projet le plus populaire), 900 forks et 200 contributeurs.
- 9 mai 2014, 1 900 forks et 420 contributeurs.
- Octobre 2015, 6 500 forks et 1 100 contributeurs.
- Septembre 2016, plus de 10 000 forks et 1 400 contributeurs.
- Décembre 2017, plus de 13 500 forks et 1 700 contributeurs



# De la virtualisation à Docker

## L'évolution de DotCloud à Docker

- **1979**: Unix V7 introduit le concept de “chroot” i.e. modification du root directory (filesystem) d'un process et de ses enfants (ex. forks). C'est les débuts de tentative d'isolation d'un processus : confinement d'un processus afin de l'empêcher d'accéder à certains fichiers.



- **2000**: FreeBSD Jails : permet à l'admin de partitionner un système FreeBSD en plusieurs systèmes indépendants (appelés “jails”) avec la possibilité d'assigner une adresse IP différente à chacun.



- **2004**: Oracle Solaris Containers





# De la virtualisation à Docker

## L'évolution de DotCloud à Docker

- **2006**: Process Containers. Créer par Google. Fait pour limiter, comptabiliser et isoler l'utilisation des ressources : CPU, RAM, disk IO, network, d'une collection de processus. (cgroups -> noyaux linux 2.6.24)
- **2008**: LXC (**LinuX Containers**) 
- **2013**: Let Me Contain That For You (LMCTFY) mise en open source de la stack de containerisation de Google
- **Docker** utilise LXC au départ pour ensuite créer sa propre implémentation : libcontainer 
- **OCI** <https://www.opencontainers.org/> Open Container Initiative : standards industriel pour container et image

# De la virtualisation à Docker

## Le positionnement de Docker.

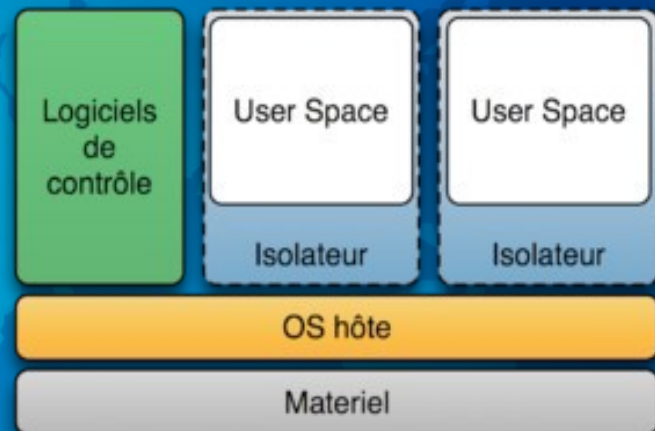
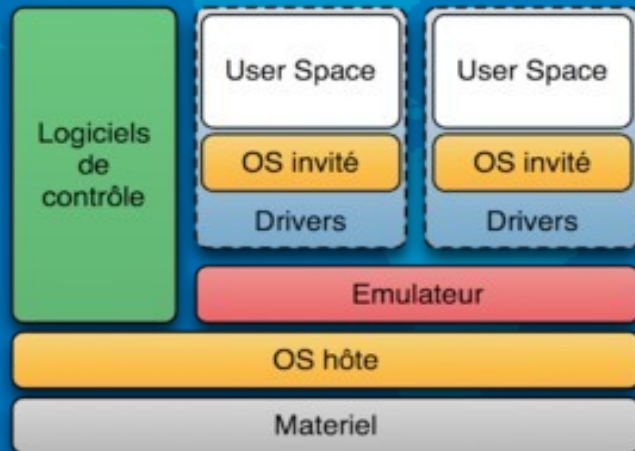
Docker a entrepris de positionner sa suite **Docker Enterprise Edition** auprès des entreprises souhaitant moderniser leurs applications dites Legacy par le biais de conteneurs.

D'un point de vue technologique, cela n'est pas nouveau, la société compte désormais inclure cela dans un programme dédié – Modernising Traditional Applications – et de l'industrialiser avec des partenaires clé : HPE Pointnext (les activités de services restants à HPE), Microsoft et Avanade, et enfin Cisco

# De la virtualisation à Docker

## Docker versus virtualisation

### Comparaison



# De la virtualisation à Docker

## Docker versus virtualisation

- Un PC portable peut faire tourner jusqu'à 100 conteneurs
- 1000 conteneurs sur un serveur
- A l'intérieur des conteneurs, les logiciels tournent aussi vite que si ils étaient lancées sur l'OS hôte.
- Les opérations sur les conteneurs se font dans la seconde





# Présentation de Docker

1. Architecture de Docker.
2. Disponibilité et installation de Docker sur différentes plateformes (Windows, Mac et Linux).
3. Création d'une machine virtuelle pour maquettage.
4. La ligne de commande et l'environnement.

## Travaux pratiques

Créer une machine virtuelle pour réaliser un maquettage.

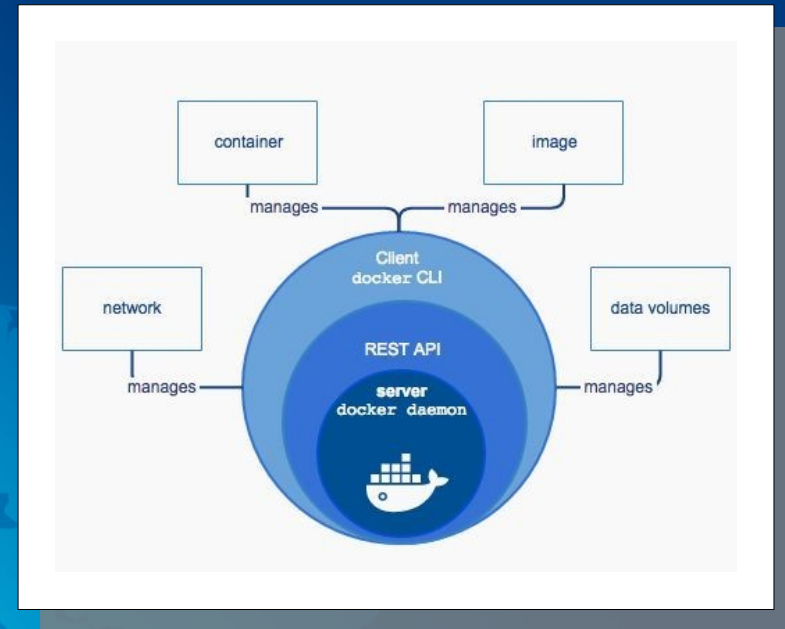


# Présentation de Docker

## Architecture de Docker

- Une partie “**serveur**”  
(unix daemon dockerd)
- une **API REST** pour communiquer avec le serveur.
- Un “**client**” : interface ligne de commande (commande docker)

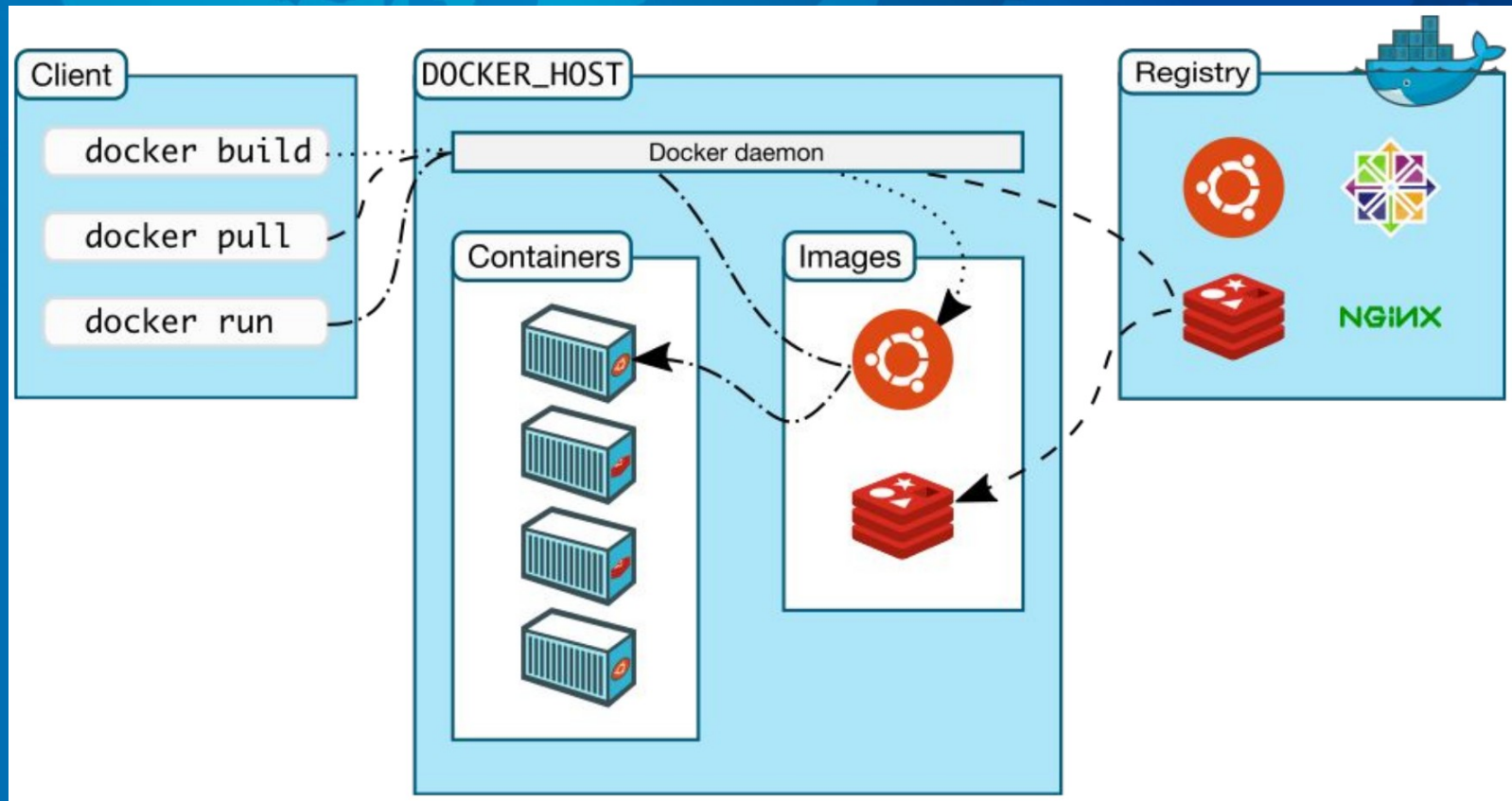
**NB** : beaucoup d'autres applications de l'écosystème docker utilisent l'API REST pour contrôler le daemon dockerd.



# Présentation de Docker

## Architecture de Docker

### Un éco-système



# Présentation de Docker

## Architecture de Docker

### Image

C'est un template de système de fichiers en "lecture seule" (immutable) (on peut voir ça comme une sorte d'archive Tar) avec en plus des méta-informations pour la création d'un container.

Vous pouvez créer vos propres images ou bien utiliser celles présentes dans le hub public.

Pour créer une nouvelle image on utilise un **Dockerfile**, chaque instruction du Dockerfile créer une nouvelle couche (**layer**) dans l'image.

Quand vous modifiez le Dockerfile et recréez l'image, seule la/les couche(s) modifiée(s) sont reconstruites (très léger et performant).

# Présentation de Docker

## Architecture de Docker

### Image

```
root@docker:~# docker history revollat/micro-php-app-docker
```

IMAGE COMMENT	CREATED	CREATED BY	SIZE
74d1fc40c34f	3 years ago	/bin/sh -c #(nop) CMD ["php" "-S" "0.0.0.0:8...	0B
<missing>	3 years ago	/bin/sh -c #(nop) EXPOSE 80/tcp	0B
<missing>	3 years ago	/bin/sh -c #(nop) ADD file:7d46011101766c933...	40.3kB
<missing>	3 years ago	/bin/sh -c apk add --update php-cli && rm -r...	11.3MB
<missing>	3 years ago	/bin/sh -c #(nop) ADD file:b9238a47014404d29...	5.03MB

```
root@docker:~# cat Dockerfile
```

```
FROM alpine:3.1 RUN apk add --update php-cli && rm -rf /var/cache/apk/*
```

```
ADD index.php /var/www/index.php
```

```
EXPOSE 80 CMD ["php", "-S", "0.0.0.0:80", "-t", "/var/www"]
```

revollat/micro-php-app-  
docker:latest  
**16 MiB**  
Layers: 5

ADD file:b9238a47014404d...  
**5 MiB**

RUN apk add --update php-...  
**11 MiB**

ADD file:7d46011101766c9...  
**40 KiB**

EXPOSE 80/tcp  
**0 Bytes**

CMD "php" "-S" "0.0.0.0:80...  
**0 Bytes**



# Présentation de Docker

## Architecture de Docker

### Conteneur (Container)

- C'est une instance exécutable d'une image. (Analogie programmation objet : l'image est à la classe ce que le container est à l'objet)
- En contactant le serveur via l'API ou via le client (commande docker) nous pouvons démarrer, arrêter, déplacer, supprimer des containers.
- Le container exécuté peut être attaché à un réseau, un volume de stockage, on peut aussi prendre un snapshot du container pour créer une nouvelle image.
- On pourra aussi contrôler l'isolation du container (vis-à-vis des autres containers, vis-à-vis de la machine hôte)





# Présentation de Docker

## Architecture de Docker

Un conteneur est défini par les métas informations de son image correspondante et par les éventuelles options passées lors de l'exécution du container.

### Exemple

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

Lorsqu'un container est arrêté et supprimé, les changements d'état qui n'ont pas été sauvegardés dans un volume de stockage persistant sont perdus.

# Présentation de Docker

## Architecture de Docker

“**Container**” (ou Conteneur en français) est juste un terme générique qui désigne sous ce nom unique un ensemble de technologies matures intégré dans le noyau linux depuis une dizaine d'années.

**PS** : une version Microsoft Windows des containers est présente dans les versions de windows serveur 2016 et Windows 10Pro.

# Présentation de Docker

## Architecture de Docker

### Union file systems

Ce sont des systèmes de fichiers qui fonctionnent avec des couches (layers). Docker peut utiliser plusieurs UnionFS, dont AUFS, btrfs, vfs, et DeviceMapper.

Lorsque un conteneur est exécuté l'image correspondante est montée en lecture seule. UnionFS crée une couche en lecture/écriture (une sorte de deltas de modifications par rapport à l'image d'origine).

# Présentation de Docker

## Architecture de Docker

### Union file systems

Plusieurs containers peuvent utiliser la même image de base (mutualisation pour éviter de prendre trop de place sur la machine hôte)

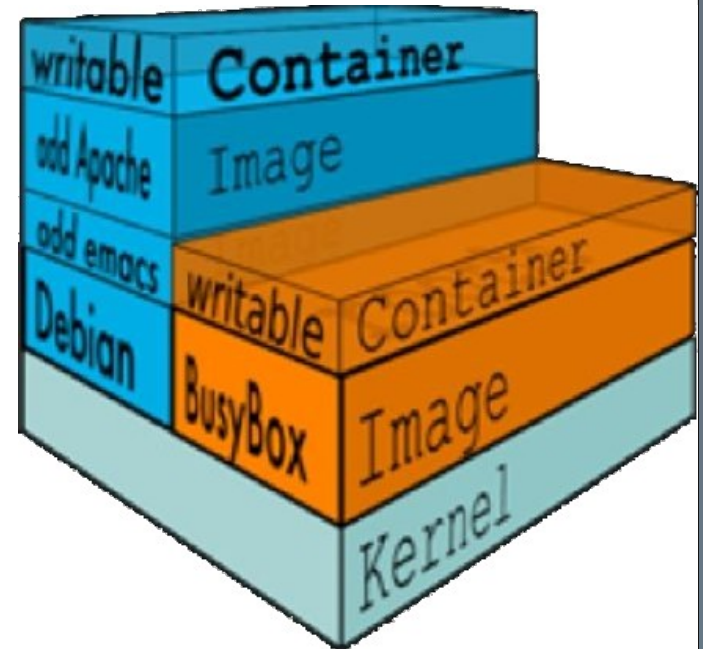
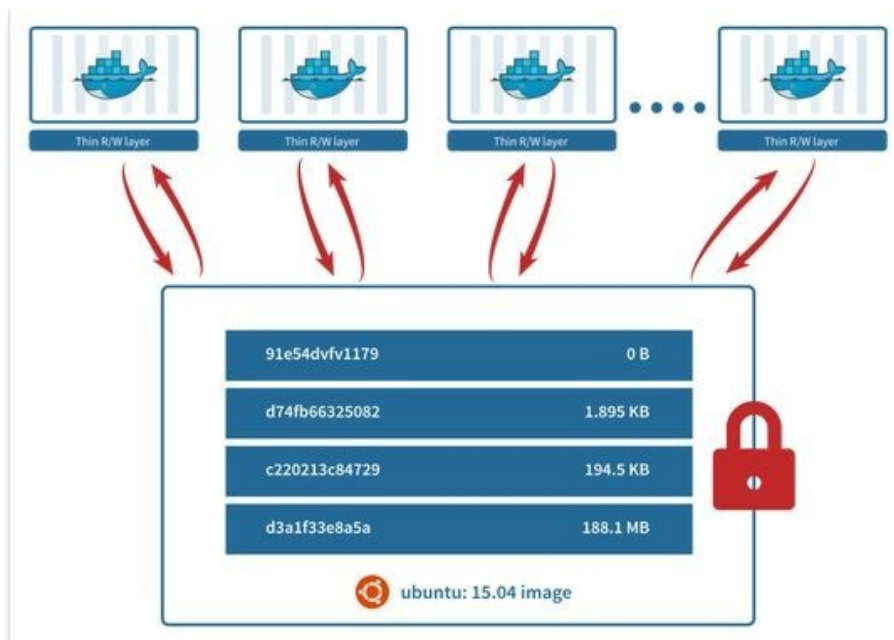
Aller plus loin :

<https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/#the-copy-on-write-cow-strategy>

# Présentation de Docker

## Architecture de Docker

### Union file systems





# Présentation de Docker

## Architecture de Docker

Une architecture modulaire : GRPC, libcontainer, runc, containerd, OCI, ...

Le format des images, comment doit être exécuté un container, etc... a été décrit sous forme de spécifications ouvertes et réunies sous le nom de

OCI : **O**pen **C**ontainer **I**nitiative  
(<https://www.opencontainers.org/>)



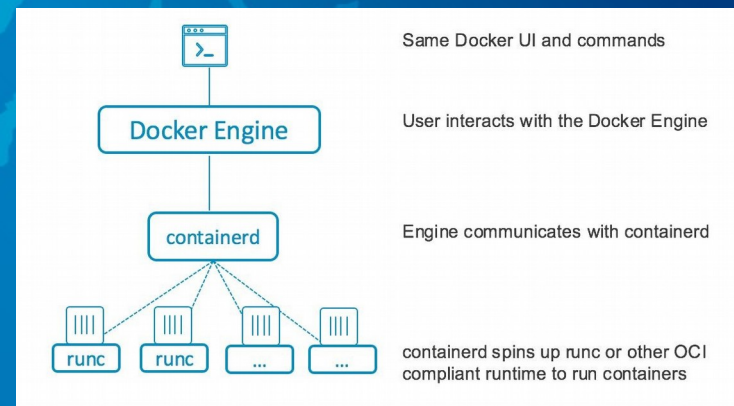
**OPEN** CONTAINER  
INITIATIVE

# Présentation de Docker

## Architecture de Docker

**DotCloud** a été un contributeur majeur de cette standardisation. Le résultat est que le docker engine, auparavant monolithique (implémenté principalement en langage Go), a été découpé en plusieurs modules indépendants et interchangeables.

Ce changement (majeur au niveau de l'architecture bien que transparent pour l'utilisateur) a été introduit dans la version 1.11 de Docker



# Présentation de Docker

## Architecture de Docker



**runC** est une implémentation de Open Containers Runtime specification utilisant **libcontainer** (interface avec les fonctionnalités kernel namespaces, cgroups, capabilities)

**runC** est utilisé par défaut dans docker pour l'exécution des containers. Cette brique peut être remplacée par une autre implémentation (principe "batteries included but swappable").

Grâce à ces changements, depuis la version 1.11 on peut par exemple redémarrer/mettre à jour l'Engine **sans affecter** les containers en cours d'exécution (découplage).

# Présentation de Docker

## Architecture de Docker



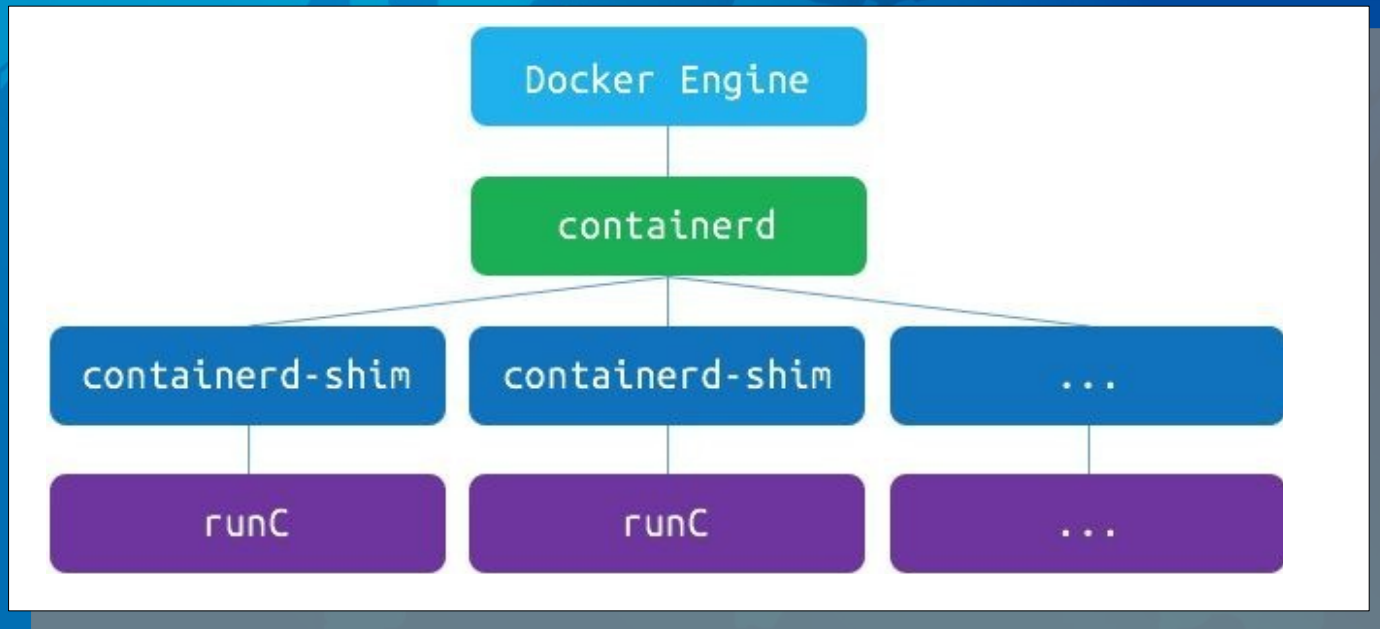
**containerd** est un daemon qui utilise runC (ou une autre implémentation compatible OCI) pour créer, démarrer, arrêter, supprimer les containers.

Le docker Engine gère toujours les images, les build (Dockerfiles), volumes, network, ... et containerd gère les containers.

Engine et Containerd communiquent via GRPC.

# Présentation de Docker

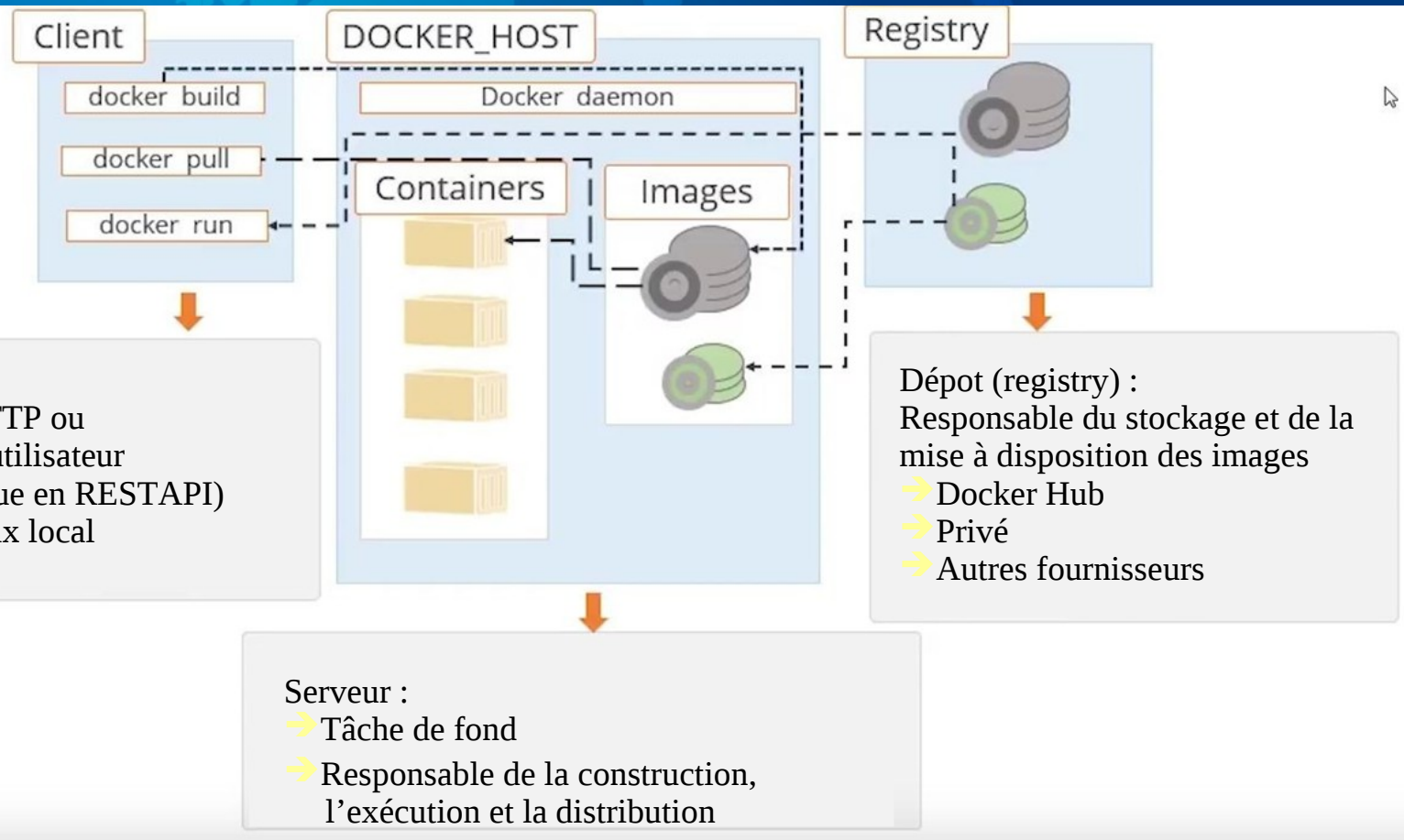
## Architecture de Docker





# Présentation de Docker

## Architecture de Docker - récapitulatif



# Présentation de Docker

## Disponibilité et installation de Docker

Docker pour Windows (Windows 10 Pro /Windows 2016)

Environnement de développement intégré et facile à déployer pour construire, déboguer et tester les applications Docker sur un PC Windows. Docker pour Windows est une application Windows native profondément intégrée avec la virtualisation Hyper-V, la mise en réseau et le système de fichiers, ce qui en fait l'environnement Docker le plus rapide et le plus fiable pour Windows.

# Présentation de Docker

## Disponibilité et installation de Docker

### Docker pour Windows : le paradoxe

- Docker est basé sur les controls groups et les namespaces
- Fort lien à Linux
- Première approche: Boot2docker
- Docker toolbox – juste un changement de nom
- Microsoft est habitué aux technologies propriétaires

# Présentation de Docker

## Disponibilité et installation de Docker

### Docker pour Windows : la suite...

- Microsoft suit l'OCI
- Technologie Drawbridge, puis Windows Jobs Objects (groupement de ressources) et Server Silos (Isolation de file systems, registre windows, ...)
- Première approche: Boot2docker
- Détails sur MSDN magazine (article de Taylor Brown)
- **Collaboration** avec Docker

# Présentation de Docker

## Disponibilité et installation de Docker

### Docker pour Windows : Deux implémentations

#### → Windows 10 Pro

- Conteneurs Windows
- Conteneurs Linux ( MobyLinux + HyperV)

#### → Windows 2016 Server

- Conteneurs Windows uniquement
- Inclusion dans HyperV
- Niveau isolation supplémentaire

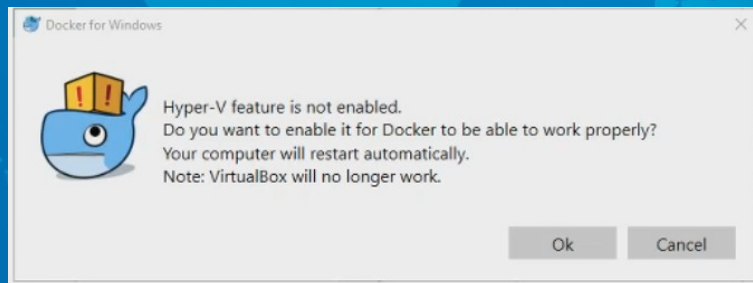


# Présentation de Docker

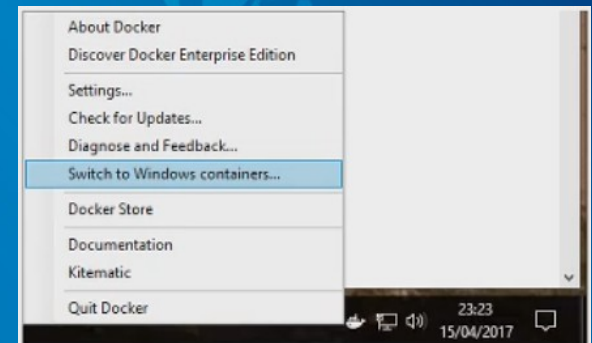
## Disponibilité et installation de Docker

### Docker pour Windows : Fonctionnement

→ Concurrency Vtx



→ Support double mais pas simultané



# Présentation de Docker

## Disponibilité et installation de Docker

Fonctionnement... comme d'habitude

`docker run microsoft/dotnet-samples:dotnet-nanserver`

OFFICIAL REPOSITORY

**mongo** ☆

Last pushed: 5 days ago

Repo info Tags

Short Description

MongoDB document databases provide high availability and easy scalability

Full Description








Supported tags and respective Dockerfile links

- 3.0.14, 3.0 (3.0/Dockerfile)
- 3.0.14-windowsservercore, 3.0-windowsservercore (3.0/windows/windowsservercore/Dockerfile)
- 3.2.12, 3.2 (3.2/Dockerfile)
- 3.2.12-windowsservercore, 3.2-windowsservercore (3.2/windows/windowsservercore/Dockerfile)
- 3.4.3, 3.4, 3, latest (3.4/Dockerfile)
- 3.4.3-windowsservercore, 3.4-windowsservercore, 3-windowsservercore, windowsservercore (3.4/windows/windowsservercore/Dockerfile)
- 3.5.5, 3.5, unstable (3.5/Dockerfile)
- 3.5.5-windowsservercore, 3.5-windowsservercore, unstable-windowsservercore (3.5/windows/windowsservercore/Dockerfile)

microsoft - Docker Hub

https://hub.docker.com/microsoft/

Repos

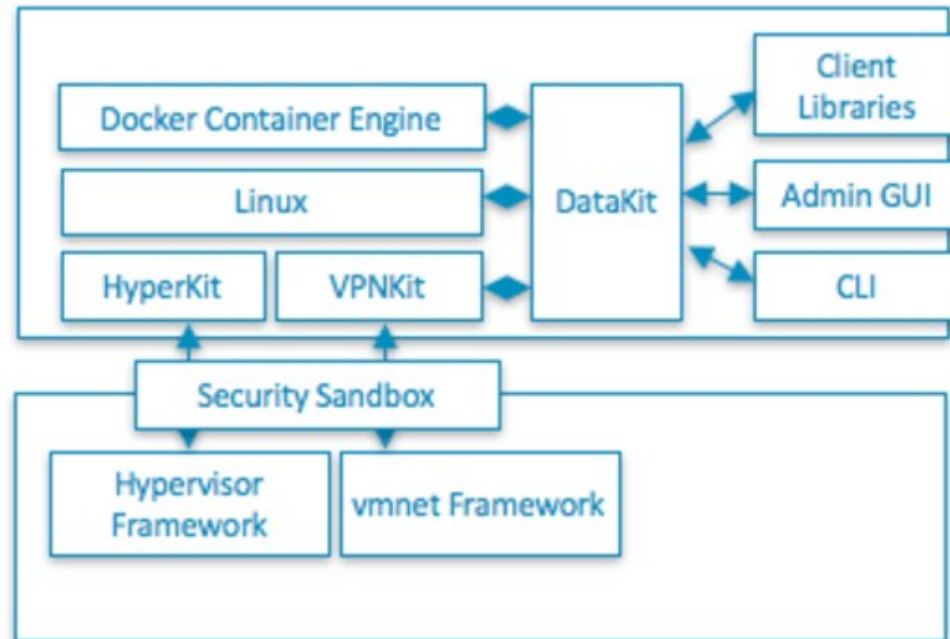
 <b>microsoft/ons</b> public   automated build	29 STARS	10M+ PULLS	> DETAILS
 <b>microsoft/dotnet</b> public   automated build	648 STARS	10M+ PULLS	> DETAILS
 <b>microsoft/vs-agent</b> public	37 STARS	5M+ PULLS	> DETAILS
 <b>microsoft/aspnetcore-build</b> public   automated build	84 STARS	5M+ PULLS	> DETAILS
 <b>microsoft/aspnet-core</b> public   automated build	99 STARS	5M+ PULLS	> DETAILS
 <b>microsoft/aspnet</b> public	105 STARS	5M+ PULLS	> DETAILS
 <b>microsoft/transpiler</b> public	214 STARS	5M+ PULLS	> DETAILS

**microsoft**  
Microsoft

Redmond, WA  
https://www.microsoft.com/en-us  
Joined May 2014

# Présentation de Docker

## Disponibilité et installation de Docker



# Présentation de Docker

## Disponibilité et installation de Docker

Pour les autres versions de Windows :  
Installer **VirtualBox** et **Debian** ou **Boot2docker**



VirtualBox



debian

Via docker-machine

# Présentation de Docker

## Disponibilité et installation de Docker

Et bien sur sous **Linux**



→ Debian



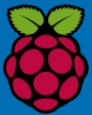
→ Ubuntu



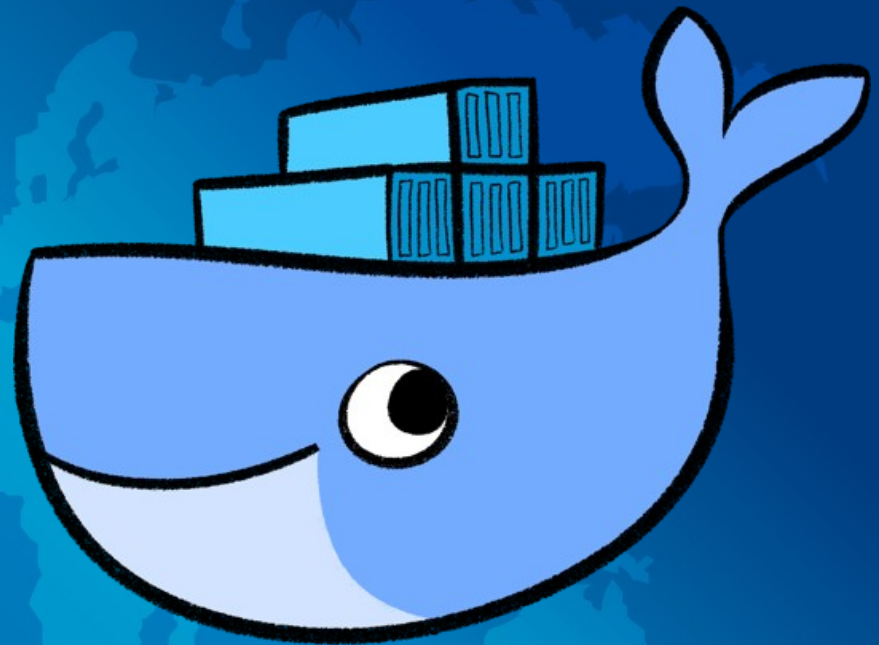
→ Fedora



→ CentOS



→ RaspBerryPi





# Présentation de Docker

**Création d'une machine virtuelle pour maquettage.**

Pour travailler en local, le mieux est de disposer d'un Windows 10 Pro

- ✓ Installer Docker
- ✓ Tester

# Présentation de Docker

**Installer docker sur Windows 2016.**

Installation par PowerShell :

Install-Module -Name DockerMsftProvider

Install-Package -Name Docker -ProviderName DockerMsftProvider

Restart-Computer -force

Installation cloud:

Bouton Deploy to Azure

Images ... -with containers

# Présentation de Docker

## Travaux pratiques

### Exercice 1 :

- Mise en oeuvre de la station Windows
- Installation Docker-ce

# Mise en œuvre en ligne de commande

1. Mise en place d'un premier conteneur.
2. Le Docker hub : ressources centralisées.
3. Mise en commun de stockage interconteneurs.
4. Mise en commun de port TCP interconteneurs.
5. Publication de ports réseau.
6. Le mode interactif.

## Travaux pratiques

Configurer un conteneur en ligne de commande

# Mise en œuvre en ligne de commande

Mise en place d'un premier conteneur.

## Charger et exécuter un container

```
root@docker:~# docker run -it debian:latest /bin/bash
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
ccla78bfd46b: Pull complete
Digest: sha256:de3eac83cd481c04c5d6c7344cd7327625ald8b2540e82a8231b5675cef0ae5f
Status: Downloaded newer image for debian:latest
root@e985c6882e3b:/#
```

En une seule commande on a :

- Télécharger Debian
- Lancer un conteneur
- Exécuter /bin/bash dans ce conteneur

Mais d'où vient ce debian:latest ?



# Mise en œuvre en ligne de commande

## Mise en place d'un premier conteneur.

Les images de base :

**microsoft/windowsservercore**: Windows Server version “réduite”

- Pas de GUI, ...
- Plusieurs Go (4.7 Go avant 10 à 12 !)
- Images SQLServer, .NET legacy seulement

**microsoft/nanoserver** : version fortement allégée

- Uniquement exécution de processus
- Même pas d'accès à distance
- Taille en centaines de Mo (1,19Go avec IIS)
- Images .NET Core, Node.js ou IIS
- NanoServer peut être utilisé comme hôte Docker

# Mise en œuvre en ligne de commande

**Mise en place d'un premier conteneur.**

Les images de base



# Mise en œuvre en ligne de commande

Le Docker hub : ressources centralisées

## Le Docker Hub


Hébergement de l'image dans une registry publique ou privée

- **Registry publique** : certains peuvent écrire, tout le monde peut lire
- **Registry privée** : certains peuvent écrire, certains peuvent lire

[hub.docker.com](https://hub.docker.com)

# Mise en œuvre en ligne de commande

## Le Docker hub : ressources centralisées

 docker hub

[Explore](#) [Help](#) [Sign in](#)

# Docker Hub

Dev-test pipeline automation, 100,000+ free apps, public and private registries



Join us in San Francisco  
dockercon 2018  
register now >>>

### New to Docker?

Create your free Docker ID to get started.

☐ \* I agree to Docker's [Terms of Service](#).

☐ \* I agree to Docker's [Privacy Policy](#) and [Data Processing Terms](#).

☐ I would like to receive email updates from Docker, including its various services and products

[Sign Up](#)

© 2016 Docker Inc.

Transfert des données depuis hub.docker.com...

# Mise en œuvre en ligne de commande






## Le Docker hub : ressources centralisées

Docker Store is the new place to discover public Docker content. [Check it out →](#)

Explore Help [Sign up](#) [Sign in](#)

Repositories (23482)

All


	alpine official	3.7K STARS	10M+ PULLS	<a href="#">&gt; DETAILS</a>
	anapsix/alpine-java public   automated build	311 STARS	5M+ PULLS	<a href="#">&gt; DETAILS</a>
	mhart/alpine-node public	359 STARS	10M+ PULLS	<a href="#">&gt; DETAILS</a>
	tenstartups/alpine public   automated build	5 STARS	10M+ PULLS	<a href="#">&gt; DETAILS</a>
	graze/php-alpine public   automated build	9	1M+	<a href="#">&gt; DETAILS</a>



# Mise en œuvre en ligne de commande

## Le Docker hub : ressources centralisées

alpine is now available in the Docker Store, the new place to discover public Docker content. [Check it out →](#)

  [Explore](#) [Help](#) [Sign up](#) [Sign in](#)

OFFICIAL REPOSITORY


**alpine** ☆

Last pushed: 2 months ago

[Repo Info](#) [Tags](#)

Short Description

A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size!

Docker Pull Command 

```
docker pull alpine
```

Full Description

Supported tags and respective Dockerfile links

- 3.6 ([versions/library-3.6/x86\\_64/Dockerfile](#))
- 3.7, latest ([versions/library-3.7/x86\\_64/Dockerfile](#))
- edge ([versions/library-edge/x86\\_64/Dockerfile](#))
- 3.1 ([versions/library-3.1/Dockerfile](#))
- 3.2 ([versions/library-3.2/Dockerfile](#))
- 3.3 ([versions/library-3.3/Dockerfile](#))
- 3.4 ([versions/library-3.4/Dockerfile](#))

[https://hub.docker.com/\\_/alpine/](https://hub.docker.com/_/alpine/)

# Mise en œuvre en ligne de commande

**Le Docker hub : ressources centralisées**

## Charger l'image :

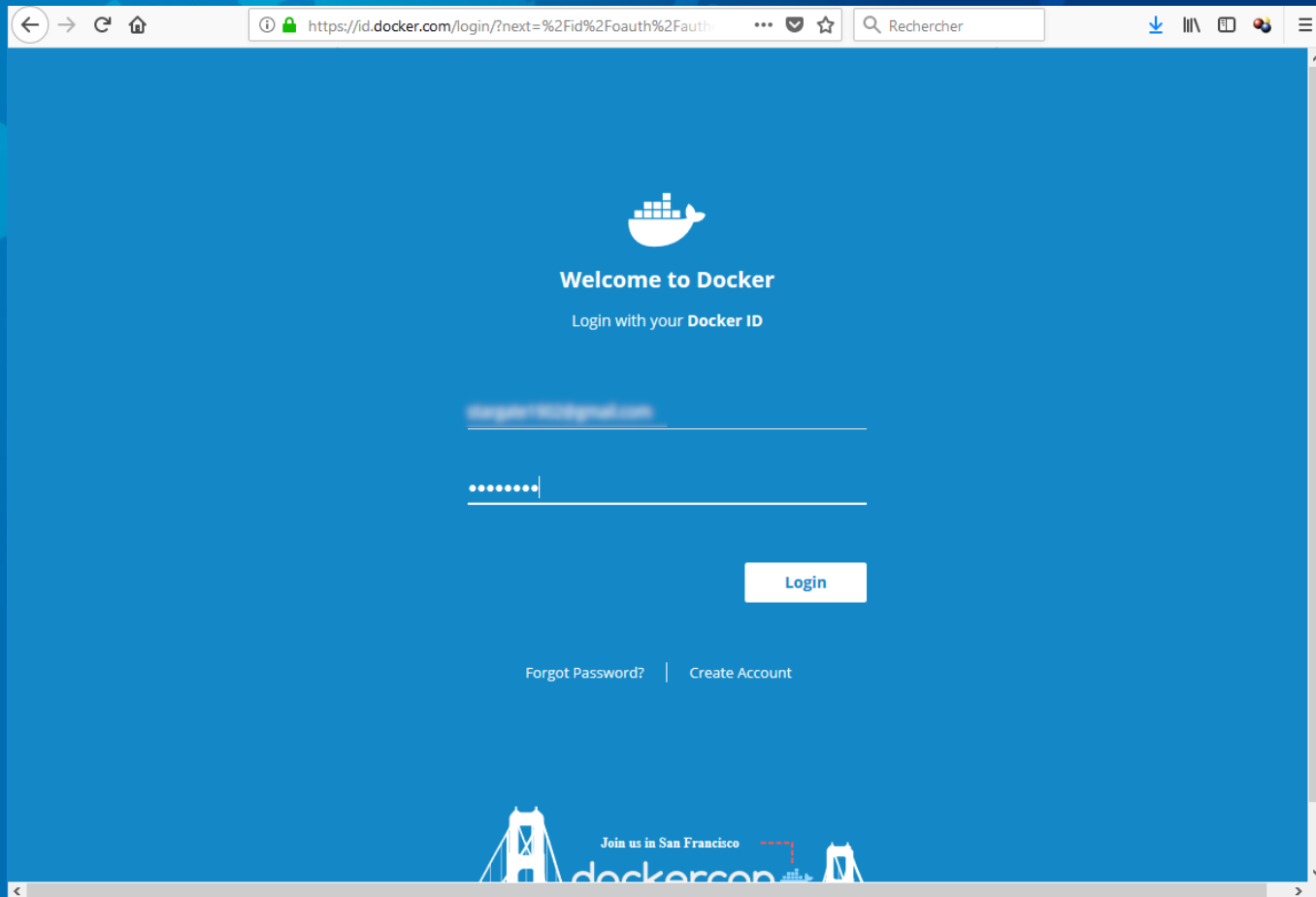
```
@docker:~# docker pull alpine:latest
latest: Pulling from library/alpine
5c916c92: Pull complete
Digest: sha256:7df6db5aa61ae9480f52f0b3a06a140ab98d427f86d8d5de0bedab9b8df6b1c0
Status: Downloaded newer image for alpine:latest
@docker:~#
```

## Exécuter l'image

```
root@docker:~# docker run -it alpine /bin/sh
/#
```

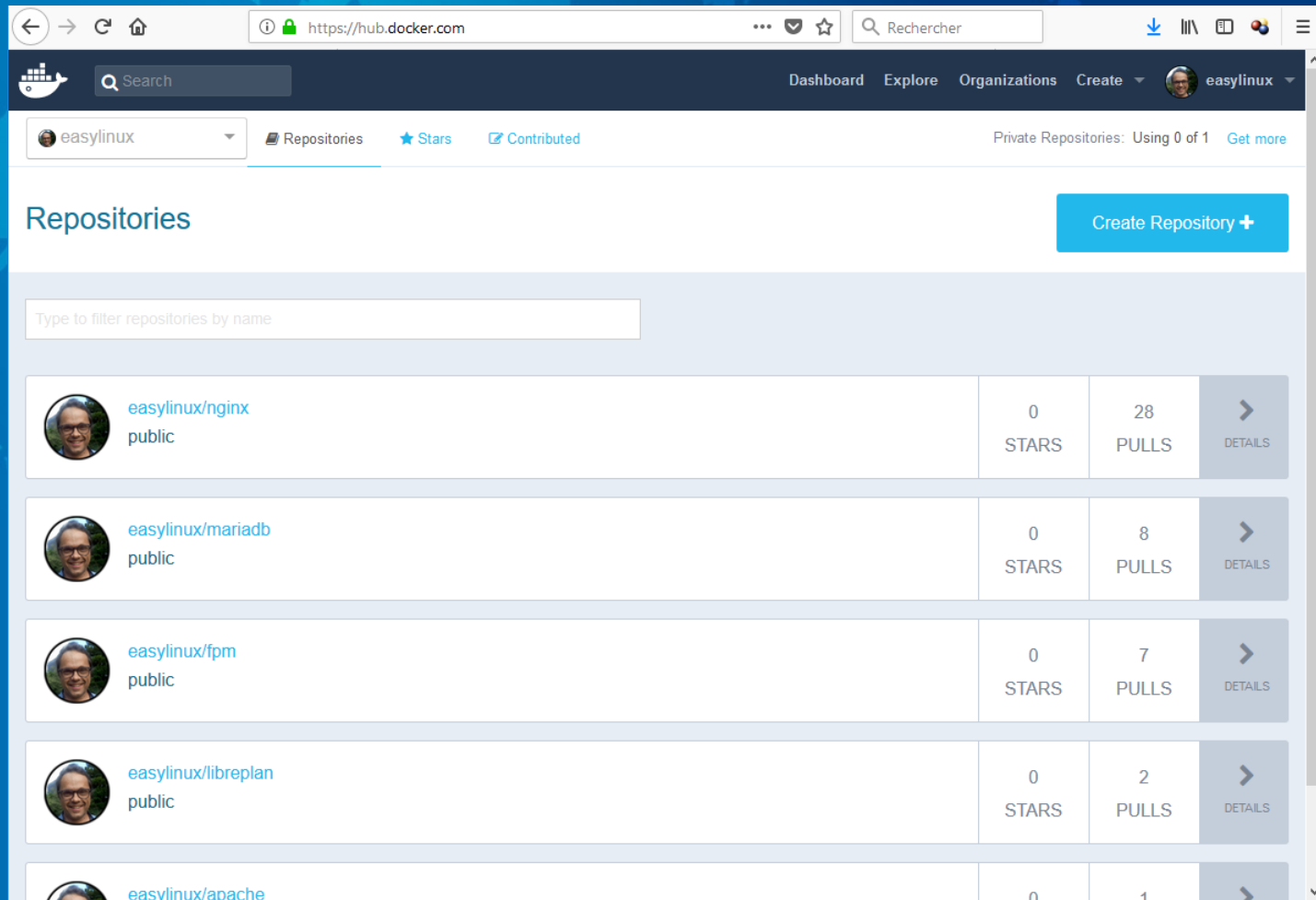
# Mise en œuvre en ligne de commande

## Le Docker hub : ressources centralisées



# Mise en œuvre en ligne de commande

## Le Docker hub : ressources centralisées



The screenshot shows the Docker Hub web interface. The browser address bar displays `https://hub.docker.com`. The user 'easylinux' is logged in, as indicated by the profile icon and name in the top right. The main navigation bar includes links for Dashboard, Explore, Organizations, Create, and a search bar. Below the navigation bar, the 'Repositories' tab is selected for the 'easylinux' user. A 'Create Repository +' button is visible in the top right of the repository list. A search filter box is present with the placeholder text 'Type to filter repositories by name'. The repository list displays the following entries:

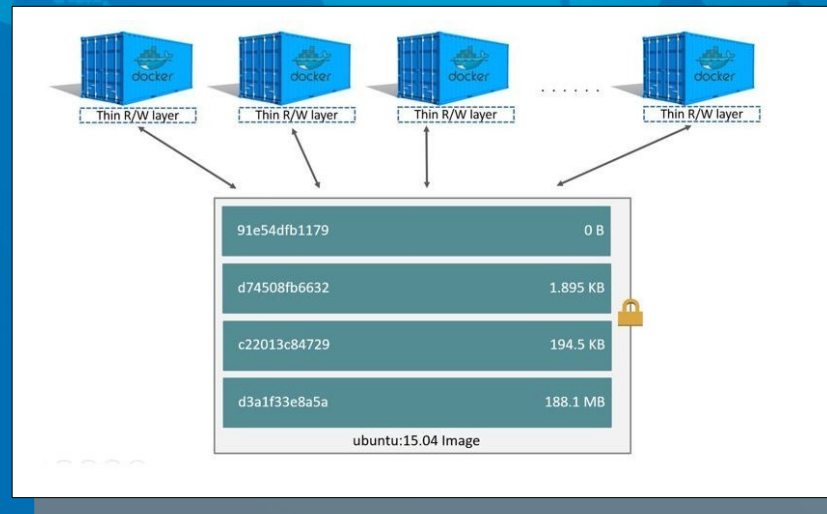
Repository Name	Stars	Pulls	Action
easylinux/nginx public	0	28	DETAILS
easylinux/mariadb public	0	8	DETAILS
easylinux/fpm public	0	7	DETAILS
easylinux/libreplan public	0	2	DETAILS
easylinux/apache	0	1	DETAILS

# Mise en œuvre en ligne de commande

## Mise en commun de stockage interconteneur

On peut écrire des données dans le layer RW d'une image mais :

- Les données ne vont pas persister si on arrête le container.
- L'écriture dans la couche RW nécessite un storage driver (UnionFS, COW) cette couche d'abstraction réduit la performance d'écriture sur le disque comparé à un volume.





# Mise en œuvre en ligne de commande

## Mise en commun de stockage interconteneur

On peut monter des données dans un container à l'aide de volumes

- **Volumes** : les données sont dans un filesystem géré par docker. Les processus de la machine hôte ne modifient pas ces données. Peut être partagé entre plusieurs containers.

Commands:

create	Create a volume
inspect	Display detailed information on one or more volumes
ls	List volumes
prune	Remove all unused local volumes
rm	Remove one or more volumes

Run 'docker volume COMMAND --help' for more information on a command.

```
root@docker:~# docker volume create unVolume
```

```
unVolume
```

```
root@docker:~# docker run -it -v unVolume:/Data alpine /bin/sh
```

```
/ # ls /Data
```

```
/ # █
```

# Mise en œuvre en ligne de commande

## Mise en commun de stockage interconteneur

- **Bind mount** : on monte une partie du FS de la machine hôte. Les processus de la machine hôte peuvent modifier ces données. Utile pour le développement ou partage de config (hôte->container. Ex. montage de /etc/resolv.conf pour le DNS des container)

```
root@docker:~# docker run -it -v /var/log:/Data alpine /bin/sh
/ # ls /Data
alternatives.log    btmp                dpkg.log            kern.log            syslog              wtmp
apt                 daemon.log          faillog             lastlog             unattended-upgrades
auth.log            debug               installer            messages            user.log
```

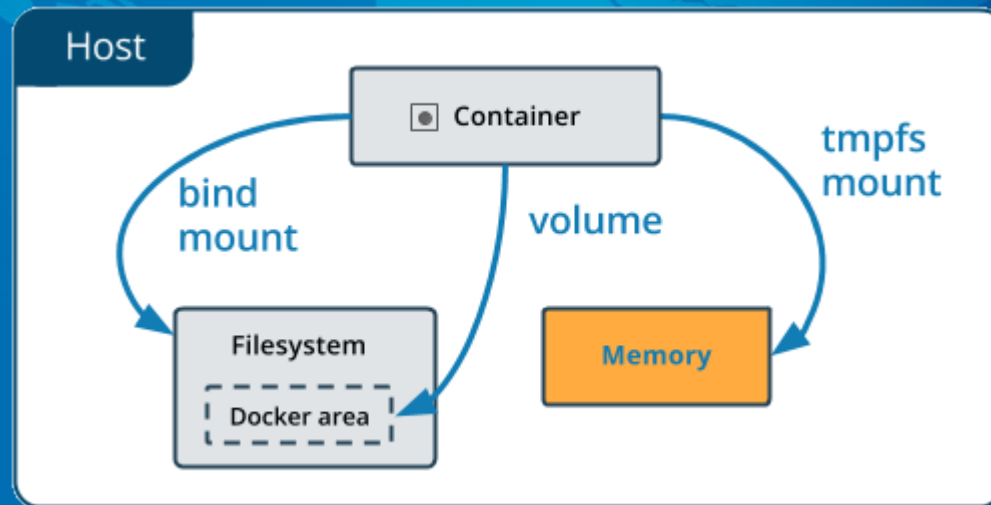
# Mise en œuvre en ligne de commande

## Mise en commun de stockage interconteneur

- **tmpfs** : filesystem dans la RAM de la machine hôte.

```
root@docker:~# docker run -it --mount type=tmpfs,destination=/Data alpine /bin/sh
/ # ls /Data
/ # █
```

## Résumé



# Mise en œuvre en ligne de commande

Mise en commun de port TCP interconteneur

Le réseau sous Docker

# Mise en œuvre en ligne de commande

## Mise en commun de port TCP interconteneur

Briques de bases Linux Networking utilisé par Docker

- **Linux Bridge**

Device couche OSI 2 Ethernet/MAC. Agit comme un “switch virtuel”.

- **Network namespaces**

Stack réseau isolée avec ses propres interfaces, routes, règle firewall, ...

Un namespace est créé pour chaque container, bien qu'il puisse aussi utiliser le namespace d'un autre container ou encore le namespace réseau de l'hôte.



# Mise en œuvre en ligne de commande

## Mise en commun de port TCP interconteneur

### ➤ **Virtual ethernet device (veth)**

Connecte des namespaces réseau. C'est un lien full-duplex qui possède une interface dans chacun des namespaces à connecter.

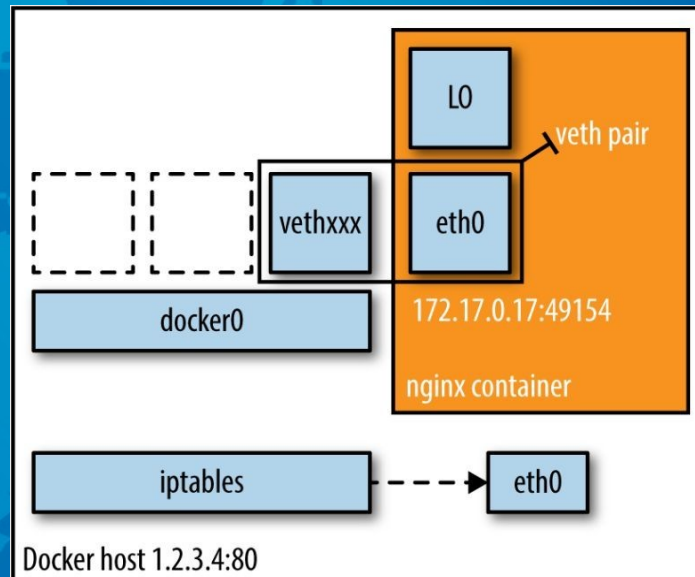
### ➤ **Iptables**

Filtrage de paquet natif du système (interface à netfilter du kernel). Docker l'utilise pour faire du NAT et du port mapping.

# Mise en œuvre en ligne de commande

## Mise en commun de port TCP interconteneur

Un schéma



# Mise en œuvre en ligne de commande

## Mise en commun de port TCP interconteneur

4 type de réseaux pour le single host networking

- **none** : uniquement l'interface loopback dans le container.
- **host** : pour partager la stack réseau de l'hôte dans le container.
- **container** : pour partager la stack réseau d'un autre container.
- **bridge** : Le bridge docker0 (créé par défaut), fournit un réseau dans la machine hôte où les containers qui y sont connectés se voient entre eux.

à la création d'un container, si rien n'est précisé, c'est le bridge qui est utilisé.

# Mise en œuvre en ligne de commande

## Mise en commun de port TCP interconteneur

Utiliser les réseaux :

Commands:

connect	Connect a container to a network
create	Create a network
disconnect	Disconnect a container from a network
inspect	Display detailed information on one or more networks
ls	List networks
prune	Remove all unused networks
rm	Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.

```
root@docker:~# docker network create monReseau
```

```
9aa9f6c5afc8be359c4a6164515240alc33afb8e79a9cc627015410c4c9ac84b
```

```
root@docker:~# ^C
```

```
root@docker:~# docker run -it --network=monReseau alpine /bin/sh
```

```
/ # ip addr
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
```

```
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
    inet 127.0.0.1/8 scope host lo
```

```
        valid_lft forever preferred_lft forever
```

```
19: eth0@if20: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
```

```
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
```

```
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
```

```
        valid_lft forever preferred_lft forever
```

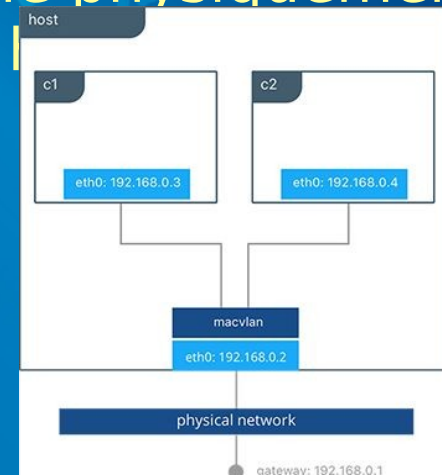
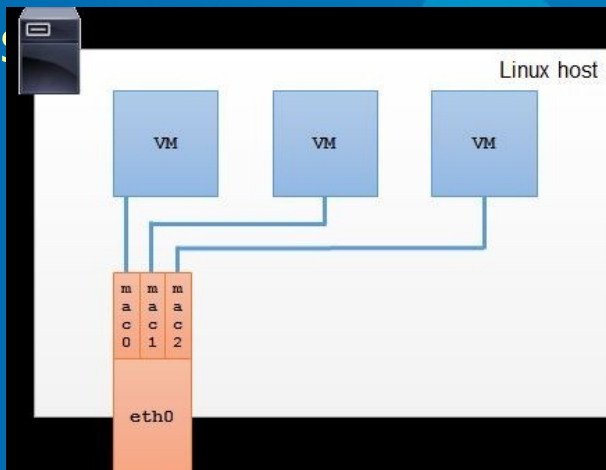
# Mise en œuvre en ligne de commande

## Mise en commun de port TCP interconteneur

### MACVLAN

Permet de configurer plusieurs “sub-interface” couche 2 (Ethernet/MAC) sur une seule interface réseau physique.

Les containers se connectent à ces sub-interfaces pour être directement reliés au réseau physique (comme si le container était directement branché physiquement que la machine hôte).





# Mise en œuvre en ligne de commande

## Mise en commun de port TCP interconteneur

### Exemple :

```
root@docker:~# docker network create -d macvlan --subnet=192.168.0.0/24 --ip-range=192.168.0.128/24 --gateway=192.168.0.254 -o parent=enp0s3 macnet32
2d7597671035699b8b036d047fc43c5ffaf7635a459f162d41ld07ff4bd5bc88
root@docker:~# docker run -it --network=macnet32 alpine /bin/sh
/ # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
21: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:c0:a8:00:01 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.1/24 brd 192.168.0.255 scope global eth0
        valid_lft forever preferred_lft forever
/ #
```

# Mise en œuvre en ligne de commande

## Mise en commun de port TCP interconteneur

### Overlay Network

Facilite le multi-host networking.

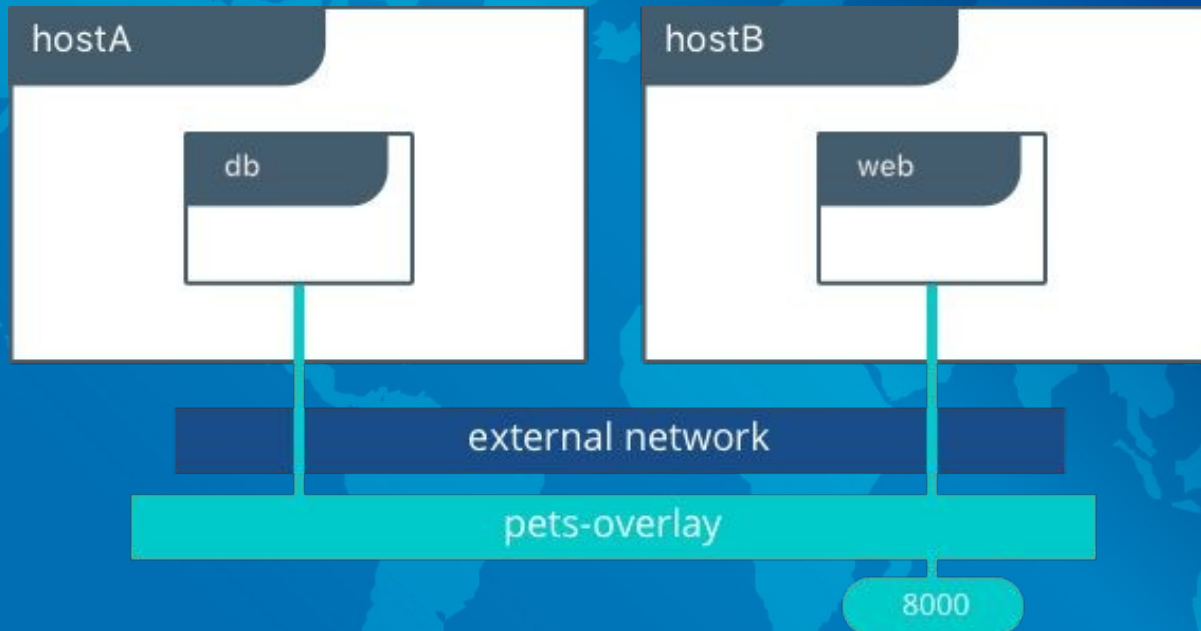
Utilise le standard VXLAN (Virtual Extensible LAN) introduit dans le kernel linux v3.7. Encapsule la couche OSI 2 (MAC/Ethernet) dans des paquet UDP (couche 3) (a.k.a MAC-in-UDP encapsulation)

IANA assigne à VXLAN par défaut le port 4789. Les endpoints du VXLAN se comportent comme un switch.

# Mise en œuvre en ligne de commande

Mise en commun de port TCP interconteneur

## Overlay Network



# Mise en œuvre en ligne de commande

## Publication de ports réseau

### Accès externe

Par défaut les container situés sur le même réseau docker peuvent communiquer sur tous les ports.

Les communications vers l'extérieur sont autorisées explicitement par firewall via Masquerading (flux sortant) et DNAT (flux entrant)

# Mise en œuvre en ligne de commande

## Publication de ports réseau

Pour pouvoir atteindre un conteneur, il faut publier son port (le rendre visible),

**\$ docker run -p <port hôte>:<port conteneur> ...**

```
root@docker:~# docker run -it -p 8000:80 alpine /bin/sh  
/#
```



# Mise en œuvre en ligne de commande

## Le mode interactif

Nous avons utilisé notre premier conteneur Hello World pour tester la technologie de conteneurisation. Nous voulons faire fonctionner un conteneur en mode interactif. La sous-commande `docker` prend une image en entrée et la lance en tant que conteneur. Vous devez passer les drapeaux **-t** et **-i** à la commande **docker run** afin de rendre le conteneur interactif. Le **-i** est la commande qui rend le conteneur interactif en saisissant l'entrée standard (STDIN) du conteneur. Le **-t** attribue un pseudo Terminal (émulateur de terminal) et l'affecte ensuite au conteneur.

```
$ docker run -it alpine /bin/sh
```

# Mise en œuvre en ligne de commande

## `docker run`

- `-d` Detached mode: Lance le conteneur en tâche de fond, affiche l'id
- `-t` Allocate a pseudo-tty
- `-i` Keep STDIN
- `--name` définit le nom du conteneur
- `--cidfile=""` Ecrit ID dans le fichier désigné (automatisation)
- `--restart` `<no|on-failure|always|unless-stopped>`
- `--rm` supprime automatiquement le conteneur en sortie

Référence :

<https://docs.docker.com/engine/reference/run>

# Mise en œuvre en ligne de commande

## Exercice 4 :

- Récupérer un conteneur et le lancer

## Exercice 5 :

- Utiliser un conteneur interactif

## Exercice 6 :

- Lancer un conteneur web

# Création de conteneur personnalisé

1. Produire l'image de l'état d'un conteneur.
2. Qu'est-ce qu'un fichier DockerFile ?
3. Automatiser la création d'une image.
4. Mise en œuvre d'un conteneur.
5. Conteneur hébergeant plusieurs services : supervisor.

## Travaux pratiques

Créer un conteneur personnalisé.

# Création de conteneur personnalisé

## Produire l'image de l'état d'un conteneur

**\$ docker history myApp**

Exemple (à lire de bas en haut)

- Layer 10: Déclaration du port TCP exposé dans le conteneur
- Layer 9 : Définition du script lancé au démarrage du conteneur
- Layer 8 : Installation des dépendances de l'application
- Layer 7 : Définition de /usr/src/app comme répertoire courant
- Layer 6 : Copie du code de l'application dans /usr/src/app
- Layer 5 : chown récursif de /usr/src/app pour myappuser:myappgroup
- Layer 4 : Création d'un utilisateur/groupe myappuser:myappgroup
- Layer 3 : Création du répertoire /usr/src/app
- Layer 2 : Installer dépendances système
- Layer 1 : Image de base: Python 3.5



# Création de conteneur personnalisé

## Qu'est-ce qu'un Dockerfile ?

Les 'Dockerfile' sont des fichiers qui permettent de construire une image Docker adaptée à nos besoins, étape par étape.

Le fichier doit s'appeler Dockerfile et être à la racine de votre projet.

La première chose à faire dans un Dockerfile est de définir de quelle image vous héritez.

**FROM alpine:latest**

**FROM** permet de définir notre image de base, vous pouvez l'utiliser seulement une fois dans un Dockerfile.

# Création de conteneur personnalisé

## Qu'est-ce qu'un Dockerfile ?

Exemple :

```
FROM python:3.5
RUN apt-get update && apt-get install -y libffi-dev
RUN mkdir -p /usr/src/app
RUN groupadd myappgroup \
    && useradd --create-home --home-dir /home/myappuser \
    --uid 4242 -g myappgroup myappuser
RUN chown -R myappuser:myappgroup /usr/src/app
COPY . /usr/src/app
WORKDIR /usr/src/app
RUN pip install requirements.txt
ENTRYPOINT docker-entrypoint.sh
EXPOSE 5000
```

# Création de conteneur personnalisé

## Qu'est-ce qu'un Dockerfile ?

**FROM:** importe récursivement la définition d'images parentes

**LABEL:** définit une valeur (**MAINTAINER**)

**RUN:** lance une commande via /bin/sh

**CMD:** définit la commande par défaut du conteneur

**EXPOSE:** définit les ports (UDP/TCP) écoutés au sein du conteneur

**ENV:** définit une variable d'environnement

**COPY:** copie des fichiers/dossiers dans l'image

# Création de conteneur personnalisé

## Qu'est-ce qu'un Dockerfile ?

**ENTRYPOINT:** définit une commande exécutée au lancement du conteneur

**VOLUME:** espace de données persistant partagé  
entre l'hôte et le conteneur

**USER:** utilisateur au sein du process

**WORKDIR:** répertoire courant du process

Référence : <https://docs.docker.com/engine/reference/builder/>

# Création de conteneur personnalisé

Qu'est-ce qu'un Dockerfile ?

**Attention** à ne pas confondre RUN et CMD :

- **RUN** exécute des commande et créer des commit et nouvelles images
- **CMD** n'exécute rien au moment du build (ça définit dans l'image la commande à exécuter par un container)



# Création de conteneur personnalisé

Qu'est-ce qu'un Dockerfile ?

## ENTRYPOINT vs. CMD

Un Dockerfile doit spécifier au moins un des deux :

- **ENTRYPOINT** doit être utilisé quand on utilise un container comme un exécutable
- **CMD** doit être utilisé pour définir des arguments par défaut pour un ENTRYPOINT ou pour exécuter une commande ad-hoc dans le container.

**CMD** sera surchargé en lançant un container avec des arguments

# Création de conteneur personnalisé

Qu'est-ce qu'un Dockerfile ?

## ONBUILD

Ajouter à l'image un déclencheur (trigger) qui sera déclenché plus tard : lorsque cette image sera utilisée comme base image (FROM) pour la construction (build) d'une nouvelle image.

Cas d'utilisation :

- Une image réutilisable.

# Création de conteneur personnalisé

## Qu'est-ce qu'un Dockerfile ?

### ONBUILD

Lors de la première construction, on ne peut pas ajouter des sources avec ADD car à ce moment nous n'avons pas encore d'application.

...

```
ONBUILD ADD . /app/src
```

```
ONBUILD RUN /usr/local/bin/python-build \  
            --dir /app/src
```

...

Au build de l'image de base des triggers sont ajoutés, mais les instructions ne sont pas exécutés.

# Création de conteneur personnalisé

Qu'est-ce qu'un Dockerfile ?

## ONBUILD

Quand l'image de base est utilisée (FROM) pour un nouveau build.

Le builder vérifie s'il y a des trigger enregistrés et si oui, ils sont exécutés.

Les trigger sont nettoyés de l'image résultante : donc les trigger ne sont pas hérités par les "arrières petits-enfants".

# Création de conteneur personnalisé

## Qu'est-ce qu'un Dockerfile ?

Bonnes pratiques :

- ✓ Un conteneur doit être éphémère
- ✓ Ne pas installer des paquets inutiles
- ✓ Un seul but / application par conteneur
- ✓ Limiter le nombre de couches
- ✓ Trier les lignes multiples (&& \)
- ✓ Limiter la taille !



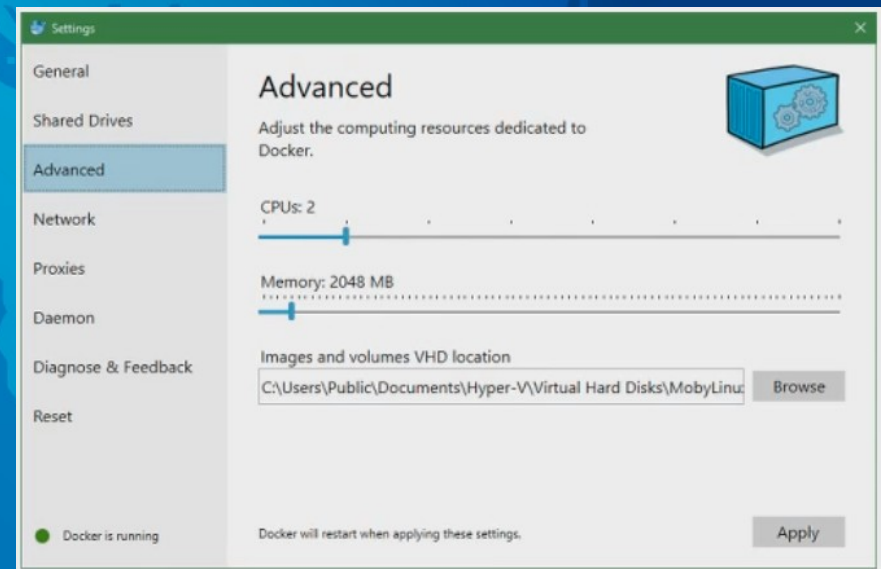
# Création de conteneur personnalisé

## Dockerfile : spécificités Windows

- **WORKDIR** C:\\App
- **VOLUME** C:\\Data
- Possibilité d'utiliser ' à la place de #
- **ADD** web.config C:/App
- **SHELL** ["PowerShell"]
- Par défaut : Cmd.exe

**NB:**

Pas de **FROM SCRATCH**



# Création de conteneur personnalisé

## Automatiser la création d'une image

Construire une image depuis un Dockerfile:

```
docker build -t <target> <rep>
```

Exemple

```
docker build -t easylinux/apache:2.5 .
```

# Création de conteneur personnalisé

## Mise en oeuvre d'un conteneur

L'image créée, nous pouvons la lancer via un docker run :

```
docker run -d easylinux/apache:2.5
```

# Création de conteneur personnalisé

## Conteneur avec plusieurs services : supervisor

Les bonnes pratiques imposent de séparer les services dans des conteneurs séparés. Mais dans certains cas (Nginx/Php), il peut être lourd d'avoir 2 conteneurs.

Solution : **supervisor**

Supervisor est un système qui permet de contrôler un ensemble de processus vivant dans un environnement UNIX. Pour faire simple, Supervisor lance des services et les redémarre s'ils échouent.

# Création de conteneur personnalisé

## Conteneur avec plusieurs services : supervisor

Supervisor lit un fichier de configuration situé dans `/etc/supervisor`.

Chaque fichier finissant en `.conf` et situé dans le chemin `/etc/supervisor/conf.d` représente un processus qui est surveillé par Supervisor.

```
[program:<nom>]
command = <programme>
user = <utilisateur>
directory = <chemin>
autostart = true
autorestart = true
```



# Mise en œuvre conteneur personnalisé

## Exercice 7 :

- Utiliser docker history

## Exercice 8 :

- Créer et lancer un conteneur (dockerfile)

## Exercice 9 :

- Utiliser supervisor

# Mettre en œuvre une application multiconteneur

1. Utilisation Docker Compose.
2. Création d'un fichier yml de configuration.
3. Déployer plusieurs conteneurs simultanément.
4. Lier tous les conteneurs de l'application.

## Travaux pratiques

Mettre en œuvre une application multiconteneur.

# Application multiconteneur

## Utilisation docker-compose

### La problématique

Un bot utilise deux containers Docker :

- un container Redis pour la persistance (dates de publication des posts...)
- un container pour notre application.

Ces containers sont dépendants l'un de l'autre : l'application doit être connectée (**link**, dans la terminologie Docker) au container Redis.

Pour démarrer notre application, il faut donc lancer le container Redis avant le container de l'application.

# Application multiconteneur

## Utilisation docker-compose

### La problématique

La ligne de commande de Docker est assez complexe. Si on veut se souvenir des paramètres adaptés à notre application, on a donc tendance à conserver la ligne de commande dans un ReadMe ou dans des scripts Shell.

```
#!/bin/bash
docker build -t ippontech/rss2twitter:latest app
docker run -p 0.0.0.0:6379:6379 -v
/root/rss2twitter/data:/data -d --name redis redis
redis-server --appendonly yes
docker run --link redis:redis -d --name rss2twitter
ippontech/rss2twitter
```

# Application multiconteneur

## Utilisation docker-compose

### La problématique

Un script ?

- Ça fonctionne mais c'est très manuel et, au final, pas très pratique.

Par exemple, pour arrêter nos containers, il nous faut un autre script.

Et si un container vient à tomber, il faut relancer le bon container.



# Application multiconteneur

## Utilisation docker-compose

### **Solution** : docker-compose

Docker Compose est l'outil "officiel" proposé par Docker.

Pour Docker Compose, un fichier **docker-compose.yml** doit être créé.

Chaque container doit être décrit : image, commande de lancement, volumes, ports...

# Application multiconteneur

## Création d'un fichier yml de configuration

Exemple : `cat docker-compose.yml`

`redis:`

`image: redis`

`command: redis-server --appendonly yes`

`volumes:`

`- /root/rss2twitter/data:/data`

`ports:`

`- "6379:6379"`

`rss2twitter:`

`build: app`

`links:`

`- redis`

# Application multiconteneur

Création d'un fichier yml de configuration

Pour le lancer, un simple **docker-compose up -d** suffit

Référence :

<https://docs.docker.com/compose/reference/>

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

# Application multiconteneur

## Création d'un fichier yml de configuration



```
version: '3'
services:
  webapp:
    image: wordpress
    ports:
      - "8080:80"
    networks:
      - overlay
...

```

Version du fichier

Services à lancer

Nom du service

Nom de l'image

Liste des ports exportés

Réseau(x) à utiliser

# Application multiconteneur

## Création d'un fichier yml de configuration

...

mysql:

image: mysql

volumes:

Volumes à utiliser

- db-data:/var/lib/mysql/data

networks:

- overlay

volumes:

Liste des volumes

db-data:

networks:

Liste des réseaux

overlay:



# Application multi-conteneurs

Déployer plusieurs conteneurs  
simultanément

On a vu dans le fichier .yml que plusieurs services peuvent être définis, ils seront lancés en simultané...

# Application multi-conteneurs

## Lier les conteneurs d'une application

Pour lier les conteneurs :

...

```
dolibarr:
```

```
  image: easylinux/apache
```

```
  links:
```

```
    - dolidb:mysql
```

```
  ports:
```

```
    - "8084:80"
```

```
  volumes:
```

```
    - ./Web/Dolibarr:/var/www/html
```

...

# Application multi-conteneurs

## Exercice 10 :

- Installer docker-compose

## Exercice 11 :

- Créer et lancer un conteneur avec docker-compose

## Exercice 12 :

- Lancer plusieurs conteneurs

## Exercice 13:

- Lancer des conteneurs liés

# Interfaces d'administration

1. L'API Docker et les Webservices.
2. Interface d'administration en mode Web.
3. Docker Registry : construire et utiliser son propre hub.

## Travaux pratiques

Construire et utiliser son propre hub.

# Interfaces d'administration

## L'API docker et les web services

Le démon docker est à l'écoute de commande via un named pipes uniquement par défaut :

### Stop-Service Docker

**Dockerd -unregister-service**

**Dockerd -H npipe:// -H 0.0.0.0:2375 -register-service**

### Start-Service Docker

Ne pas oublier le parefeu :

**Netsh advfirewall add rule name=docker dir=in protocol=tcp  
localport=2375 action=allow**

+ éventuellement le Network Security Group dans Azure



# Interfaces d'administration

## L'API docker et les web services

Le démon docker est à l'écoute de commande via un socket unix :

**unix:///var/run/docker.sock** ou un port IP : 2375

Docker fonctionne via une interface RESTAPI

Référence :

**<https://docs.docker.com/engine/api/v1.37/>**

# Interfaces d'administration

## Interface d'administration en mode web

Une interface très agréable existe pour administrer docker : Portainer de [portainer.io](https://portainer.io)

Installation :

```
docker volume create portainer_data
```

```
docker run -d -p 9000:9000 \
```



```
-v /var/run/docker.sock:/var/run/docker.sock \
```

```
-v portainer_data:/data \ portainer/portainer
```

# Interfaces d'administration

## Interface d'administration en mode web

The screenshot displays the Portainer web interface. On the left is a dark sidebar with the Portainer logo and navigation links. The main content area shows the 'Home' dashboard. At the top right of the main area, the user 'admin' is logged in, with links for 'my account' and 'log out'. Below this is a 'Node info' table showing system details. Further down are four summary cards: Containers (5 total, 2 running, 3 stopped), Images (15 total, 1.5 GB), Volumes (1 total), and Networks (4 total).

**portainer.io**  



CHANGE ENVIRONMENT

**LOCAL**

- Dashboard
- App Templates
- Containers
- Images
- Networks
- Volumes
- Events
- Engine




**PORTAINER SETTINGS**



- User management
- Endpoints
- Registries
- Settings


**Home**  


**Node info**

Name	Asus-Serge
Docker version	17.12.1-ce
CPU	8
Memory	16.7 GB

**5** Containers   

**15** Images  

**1** Volumes 

**4** Networks 

**portainer.io** 1.17.1

# Interfaces d'administration

**Docker registry : utiliser son propre dépôt**

Le Registre est une application côté serveur, évolutive qui stocke et vous permet de distribuer des images Docker.

Il est open-source, sous licence Apache.

# Interfaces d'administration

**Docker registry : utiliser son propre dépôt**

## Pourquoi l'utiliser

Vous devriez utiliser le Registre si vous souhaitez :

- contrôler l'endroit où vos images sont stockées
- posséder entièrement votre système de distribution d'images
- intégrer le stockage et la distribution des images dans votre flux de développement interne.



# Interfaces d'administration

**Docker registry : utiliser son propre dépôt**

## Alternatives

Les utilisateurs à la recherche d'une solution sans maintenance, prête à l'emploi sont encouragés à se diriger vers le **Docker Hub**.

Il fournit un registre hébergé gratuit et des fonctionnalités supplémentaires

# Interfaces d'administration

**Docker registry : utiliser son propre dépôt**

## Installer votre registre

```
docker run -d -d -p 5000:5000 --name registry  
registry registry:latest
```

## Utilisation

```
docker pull alpine
```

```
docker image tag alpine localhost:5000/my-alpine
```

```
docker push localhost:5000/my-alpine
```

```
docker pull localhost:5000/my-alpine
```

# Interface d'administration



## **Exercice 14 :**

- Installer un hub privé

## **Exercice 15 :**

- Utiliser un hub privé

# Administrer des conteneurs en production

1. Automatiser le démarrage des conteneurs au boot.
2. Gérer les ressources affectées aux conteneurs.
3. Gestion des logs des conteneurs.
4. Sauvegardes : quels outils et quelle stratégie ?

## Travaux pratiques

Administrer les conteneurs.

# Administrer des conteneurs en production

## Automatiser le démarrage au boot

Dépend du système d'init de l'OS hôte  
(systemd, sysvinit, ...)

Si une restart policy est active sur un conteneur et qu'il fonctionne lors de l'arrêt, il sera relancé au boot



# Administrer des conteneurs en production

## Gérer les ressources

### Limiter la mémoire

- `-m|--memory` (mémoire max 4M minimum)
- `--memory-reservation` limite soft < -m
- `--memory-swap` swap max
- `--kernel-memory` (mémoire kernel max 4M minimum)
- `--memory-swappiness` 0 à 100
- `--oom-kill-disable` Désactive le OOM killer  
**NB** : uniquement si -m

# Administrer des conteneurs en production

## Gérer les ressources

### Limiter le cpu

- `--cpus` Nb de cpu utilisables
- `--cpu-set-cpus` Cores utilisables
- `--cpu-shares` (1024 par défaut) Plus ou moins

Référence :

[https://docs.docker.com/config/containers/resource\\_constraints](https://docs.docker.com/config/containers/resource_constraints)

# Administrer des conteneurs en production

## Gérer les logs des conteneurs

La visualisation des logs se fait au travers de la commande :

```
docker logs <container>
```

```
docker help logs
```

# Administrer des conteneurs en production

## Sauvegardes : outils et stratégies

- Si vous utilisez les volumes nommés, il faut penser à inclure dans votre système de sauvegarde  
**`/var/lib/docker/volumes/<mesvolumes>`**
- Si vous utilisez les volumes bind, concentrer vos volumes dans un répertoire que vous pourrez inclure dans votre politique de sauvegarde

# Mise en production

## **Exercice 16 :**

- Démarrage automatique de conteneurs

## **Exercice 17 :**

- Affecter / limiter les ressources

## **Exercice 18 :**

- Gestion des logs

## **Exercice 19:**

- Mise en oeuvre sauvegarde



# Orchestration et clusterisation



1. Présentation de Docker Machine.
2. Présentation de l'orchestrateur Swarm.
3. Déploiement d'applications.

# Orchestration et clusterisation

## Présentation de docker machine

Docker Machine est un outil qui vous permet d'installer Docker Engine sur des hôtes virtuels et de gérer les hôtes à l'aide de commandes docker-machine.

Vous pouvez utiliser Machine pour créer des hôtes Docker sur votre Mac ou Windows, sur votre réseau d'entreprise, dans votre centre de données ou sur des fournisseurs de cloud comme Azure, AWS ou Digital Ocean.

# Orchestration et clusterisation

## Présentation de docker machine

En utilisant les commandes docker-machine, vous pouvez démarrer, inspecter, arrêter et redémarrer un hôte géré, mettre à niveau le client et le démon Docker, et configurer un client Docker pour parler à votre hôte.

# Orchestration et clusterisation

## Présentation de docker machine

Machine était le seul moyen d'exécuter Docker sur Mac ou Windows **avant Docker v1.12**.

Depuis Docker est disponible en tant qu'application native.

C'est le meilleur choix pour ces cas d'utilisation.

Les installateurs de Docker pour Mac et Docker pour Windows incluent Docker Machine, ainsi que Docker Compose.

# Orchestration et clusterisation

## Présentation de docker swarm

Docker Swarm fournit :

- Gestion de cluster intégrée
- Conception décentralisée : Docker gère toute spécialisation du noeud au moment de l'exécution.
- Modèle de service déclaratif
- Scalabilité : Pour chaque service, vous pouvez déclarer le nombre de tâches à exécuter.



# Orchestration et clusterisation

## Présentation de docker swarm

Docker Swarm fournit :

- Surveillance : Swarm manager surveille en permanence l'état du cluster
- Réseaux multi-hôtes
- Découverte de services

# Orchestration et clusterisation

## Présentation de docker swarm

Docker Swarm fournit :

- Équilibrage de charge
- Sécurisé par défaut
- Mises à jour régulières

# Orchestration et clusterisation

## Déploiement d'applications

Dans le cas d'utilisation de Swarm, des options supplémentaires s'ajoutent dans docker-compose.yml

**deploy:**

**replicas: 6**

**update\_config:**

**parallelism: 2**

**delay: 10s**

**restart\_policy:**

**condition: on-failure**

# Des questions ?



# docker