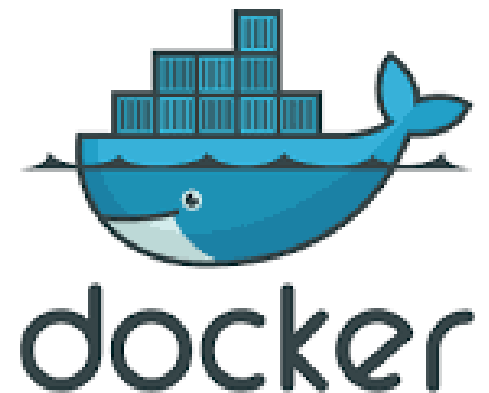




Docker



Docker

Introduction aux conteneurs

Créer ses premiers conteneurs

Les images

Le réseau

La persistance

Concepts avancés

Orchestration

Docker

Introduction

Introduction

Un conteneur Linux est un ensemble de processus qui sont isolés du reste du système.

Un conteneur s'exécute à partir d'une image distincte qui fournit tous les fichiers nécessaires à la prise en charge des processus qu'il contient.

En fournissant une image qui contient toutes les dépendances d'une application, le conteneur assure la portabilité et la cohérence de l'application entre les divers environnements (développement, test puis production).



Introduction

Les conteneurs fournissent un environnement isolé sur un système hôte, semblable à un chroot sous Linux ou une jail sous BSD, mais en proposant plus de fonctionnalités en matière d'isolation et de configuration.

Ces fonctionnalités sont dépendantes du système hôte et notamment du kernel.

Introduction

Vous travaillez sur un ordinateur portable dont l'environnement présente une configuration spécifique.

D'autres développeurs peuvent travailler sur des machines qui présentent des configurations légèrement différentes.

L'application que vous développez repose sur cette configuration et dépend de fichiers spécifiques.

Introduction

En parallèle, votre entreprise exploite des environnements de test et de production qui sont standardisés sur la base de configurations qui leur sont propres et de l'ensemble des fichiers associés.

Vous souhaitez émuler ces environnements autant que possible localement, mais sans avoir à payer les coûts liés à la recreation des environnements de serveur.

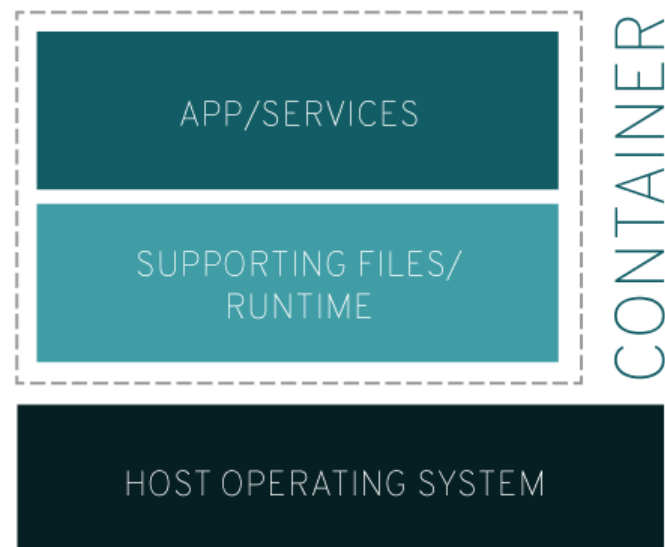
Introduction

Comment faire ?

il vous suffit d'utiliser des conteneurs.

Le conteneur qui accueille votre application contient toutes les configurations (et les fichiers) nécessaires.

Vous pouvez ainsi le déplacer entre les environnements de développement, de test et de production, sans aucun effet secondaire.



Introduction

C'est un exemple simple, mais les conteneurs Linux peuvent aussi être utilisés pour résoudre des problèmes liés à des situations qui nécessitent un haut niveau de portabilité, de configurabilité et d'isolation.

Quelle que soit l'infrastructure (sur site, dans le cloud ou hybride), les conteneurs répondent à la demande.

Bien sûr, le choix de la plateforme de conteneurs est aussi important que les conteneurs eux-mêmes.

Introduction

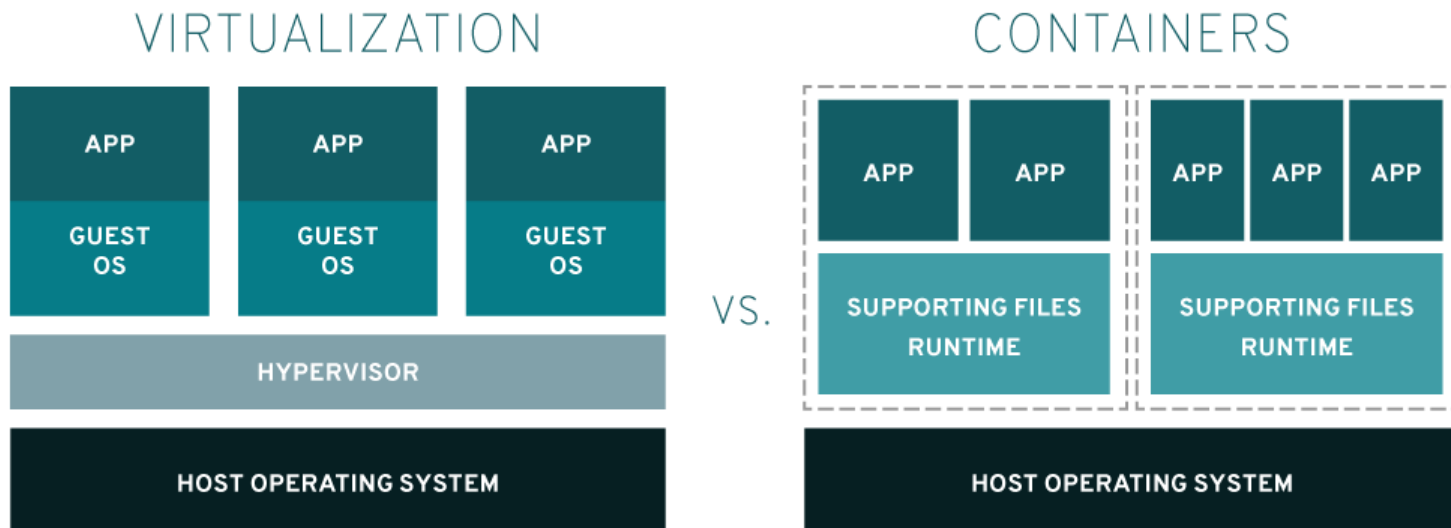
Est-ce de la virtualisation ?

Oui et non. Voyons pourquoi avec ces deux définitions simples :

- 1/ La virtualisation permet à de nombreux systèmes d'exploitation de s'exécuter simultanément sur un seul système.
- 2/ Les conteneurs partagent le même noyau de système d'exploitation et isolent les processus de l'application du reste du système.

Introduction

Virtualisation vs Containers



Introduction

Qu'est-ce que cela signifie ?

Virtualisation ?

Tout d'abord, l'exécution de plusieurs systèmes d'exploitation sur un hyperviseur (le logiciel qui permet la virtualisation) n'est pas une solution aussi légère que les conteneurs.

Lorsque vous disposez de ressources limitées, aux fonctionnalités limitées, il est nécessaire que vos applications soient légères et puissent être déployées de manière dense.

Introduction

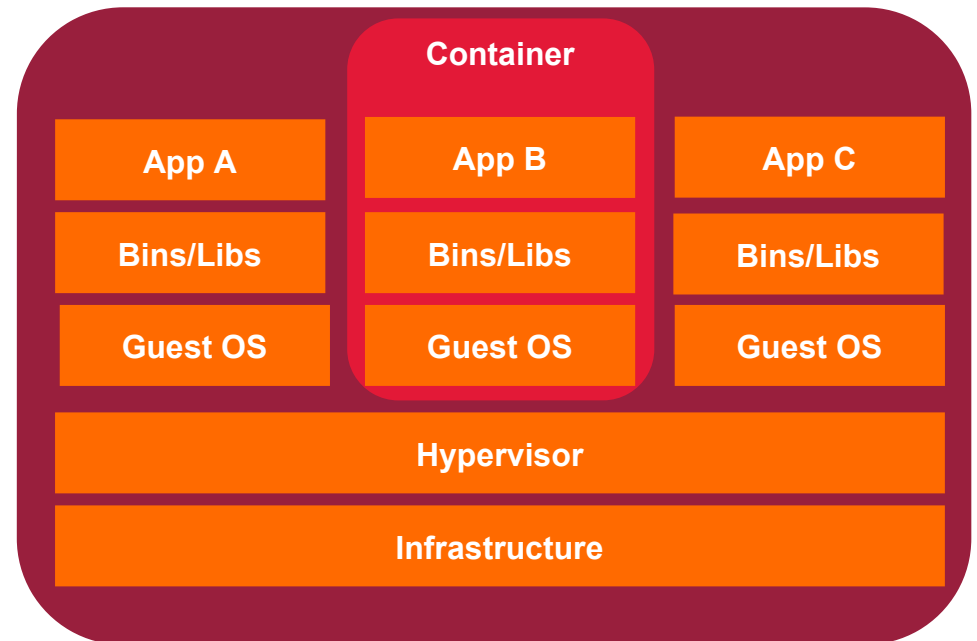
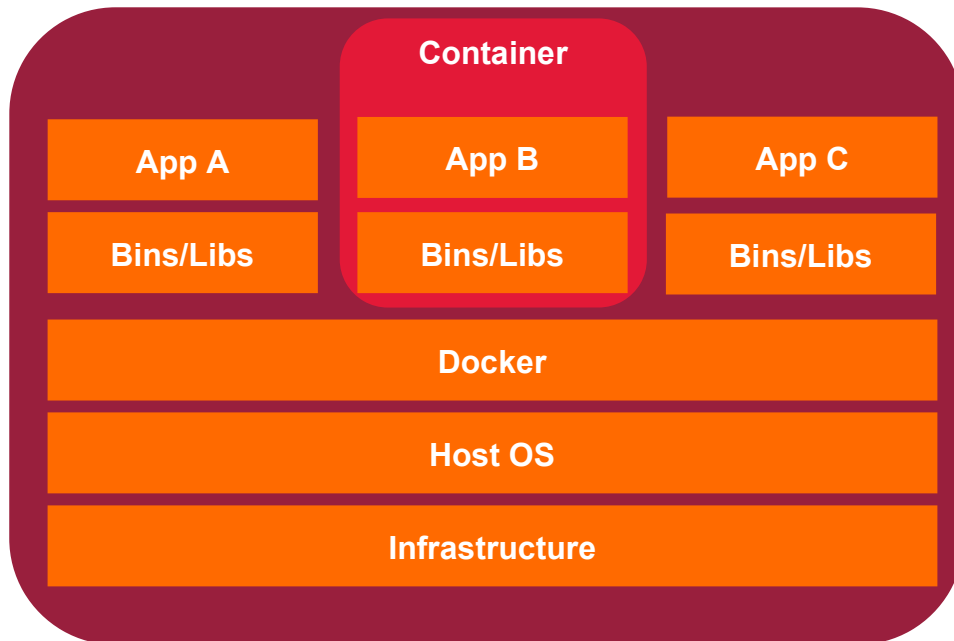
Qu'est-ce que cela signifie ?

Conteneurs ?

Les conteneurs Linux s'exécutent sur cet unique système d'exploitation, qu'ils partagent entre eux.

Vos applications et services restent ainsi légers et s'exécutent rapidement en parallèle.

Introduction



Introduction

Avantages ?

Rapidité

- Pas d'OS à booter – les apps sont dispos en quelques secondes
- Le lancement de docker est fait à partir d'une simple commande
- Vous pouvez créer et détruire des containers à la demande

Profitabilité

- Moins de dépendances entre les couches applicatives
- Maintenance simplifiée
- Rapidité de déploiement
- Portabilité entre les machines

Efficacité

- Moins de partage d'OS
- Les containers sont légers
- Gestion des versions
- Orientation architecture Micro-Services

Introduction

Histoire

La technologie que nous appelons aujourd'hui « conteneurs » est apparue en 2000 sous le nom de jail FreeBSD et permettait à l'époque de partitionner un système FreeBSD en plusieurs sous-systèmes ou « jails » (prisons en français).

Les systèmes jails étaient développés comme des environnements sécurisés qu'un administrateur système pouvait partager avec plusieurs utilisateurs opérant à l'intérieur ou à l'extérieur d'une entreprise.

Introduction

Histoire

L'objectif était de créer des processus dans un environnement chrooted modifié (dans lequel l'accès au système de fichiers, au réseau et aux utilisateurs est virtualisé) et de les y « emprisonner » afin qu'ils ne compromettent pas l'ensemble du système.

La mise en œuvre des environnements jails était cependant limitée et des méthodes ont finalement été trouvées pour en échapper.

Introduction

Histoire

En 2001, Jacques Gélinas a créé le projet VServer, rendant possible la mise en œuvre d'un environnement isolé sur un serveur Linux.

D'après M. Gélinas, ce projet visait à exécuter plusieurs serveurs Linux généralistes sur une seule machine avec un haut niveau d'indépendance et de sécurité.

Une fois cette base posée pour l'exploitation de plusieurs espaces utilisateurs contrôlés, toutes les pièces se sont mises en place pour former les conteneurs Linux que l'on connaît aujourd'hui.

Introduction

Histoire

Très rapidement, d'autres technologies se sont greffées aux conteneurs pour concrétiser cette approche de l'isolement.

La fonction du noyau cGroups (groupes de contrôle) permet de contrôler et de limiter l'utilisation des ressources pour un processus ou un groupe de processus.

Introduction

Histoire

Le système d'initialisation systemd permet de définir l'espace utilisateur et de gérer les processus associés.

Il est utilisé par la fonction cgroups pour offrir un niveau de contrôle plus élevé sur ces processus isolés.

Ces deux technologies, qui permettent de mieux contrôler Linux, ont servi de base pour pouvoir exécuter des environnements bien qu'ils soient séparés.

Introduction

Histoire

Les progrès en matière d'espaces de noms utilisateur ont ensuite permis de faire avancer la technologie des conteneurs.

Les espaces de noms utilisateur « permettent de mettre en correspondance des identifiants d'utilisateurs et de groupes au sein d'un espace de noms.

Introduction

Histoire

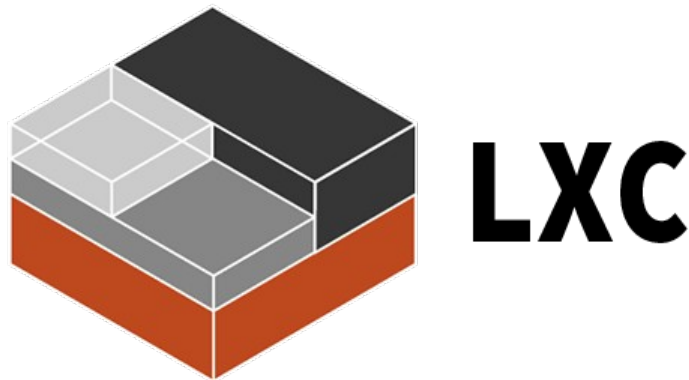
Dans le contexte des conteneurs, cela signifie que des utilisateurs et des groupes peuvent avoir des privilèges pour effectuer certaines opérations au sein du conteneur, mais que ces privilèges ne leur sont pas accordés en dehors du conteneur ».

Le concept est similaire à celui d'un environnement jail, mais avec un niveau de sécurité plus élevé, atteint grâce une isolation plus marquée des processus, au lieu d'un environnement modifié.

Introduction

Histoire

Le projet LXC a ensuite enrichi la technologie d'éléments indispensables (outils, modèles, bibliothèques et liaisons de langage) qui ont permis d'améliorer l'expérience des utilisateurs de conteneurs.



Introduction

Histoire de Docker

En 2008, la technologie de conteneurs Docker a fait son apparition (via dotCloud).

Elle combine le travail du projet LXC à des outils améliorés pour les développeurs, qui augmentent le niveau de convivialité des conteneurs.

La technologie Open Source Docker est actuellement le projet le plus connu et utilisé pour déployer et gérer des conteneurs Linux...

Introduction

Histoire de Docker

- 1/ Le logiciel « Docker » est une technologie de conteneurisation qui permet la création et l'utilisation de conteneurs Linux.
 - 2/ La communauté Open Source Docker travaille à l'amélioration de cette technologie disponible gratuitement pour tout le monde.
 - 3/ L'entreprise Docker Inc. s'appuie sur le travail de la communauté Docker, sécurise sa technologie et partage ses avancées avec tous les utilisateurs.
- Elle prend ensuite en charge les technologies améliorées et sécurisées pour ses clients professionnels.

Introduction

Technologie Docker

La technologie Docker utilise le noyau Linux et des fonctions de ce noyau, telles que les groupes de contrôle **cgroups** et les **espaces de noms**, pour séparer les processus afin qu'ils puissent s'exécuter de façon indépendante.

Cette indépendance reflète l'objectif des conteneurs : exécuter plusieurs processus et applications séparément les uns des autres afin d'optimiser l'utilisation de votre infrastructure tout en bénéficiant du même niveau de sécurité que celui des systèmes distincts.

Introduction

Technologie Docker

Les outils de conteneurs, y compris Docker, sont associés à un modèle de déploiement basé sur une image.

Il est ainsi plus simple de partager une application ou un ensemble de services, avec toutes leurs dépendances, entre plusieurs environnements.

Docker permet aussi d'automatiser le déploiement des applications (ou d'ensembles de processus combinés qui forment une application) au sein d'un environnement de conteneurs.

Introduction

Technologie Docker

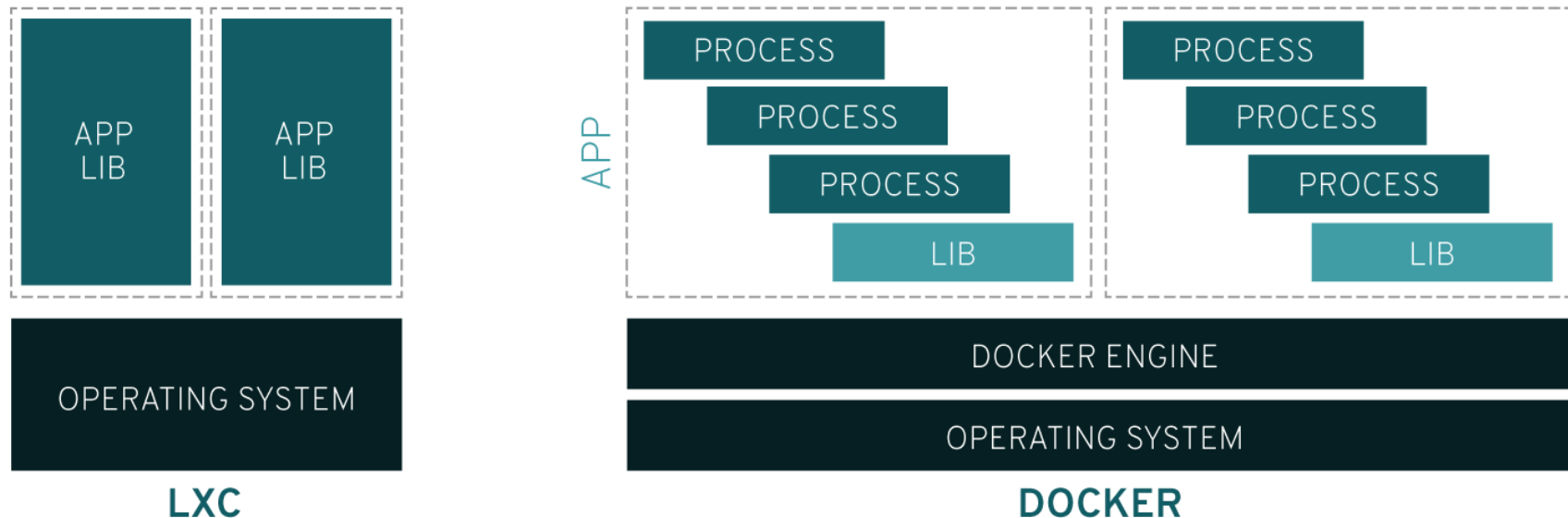
Ces outils conçus sur des conteneurs Linux (d'où leur convivialité et leur singularité) offrent aux utilisateurs un accès sans précédent aux applications, la capacité d'accélérer le déploiement, ainsi qu'un contrôle des versions et de l'attribution des versions.

Introduction

Technologie Docker vs Lxc

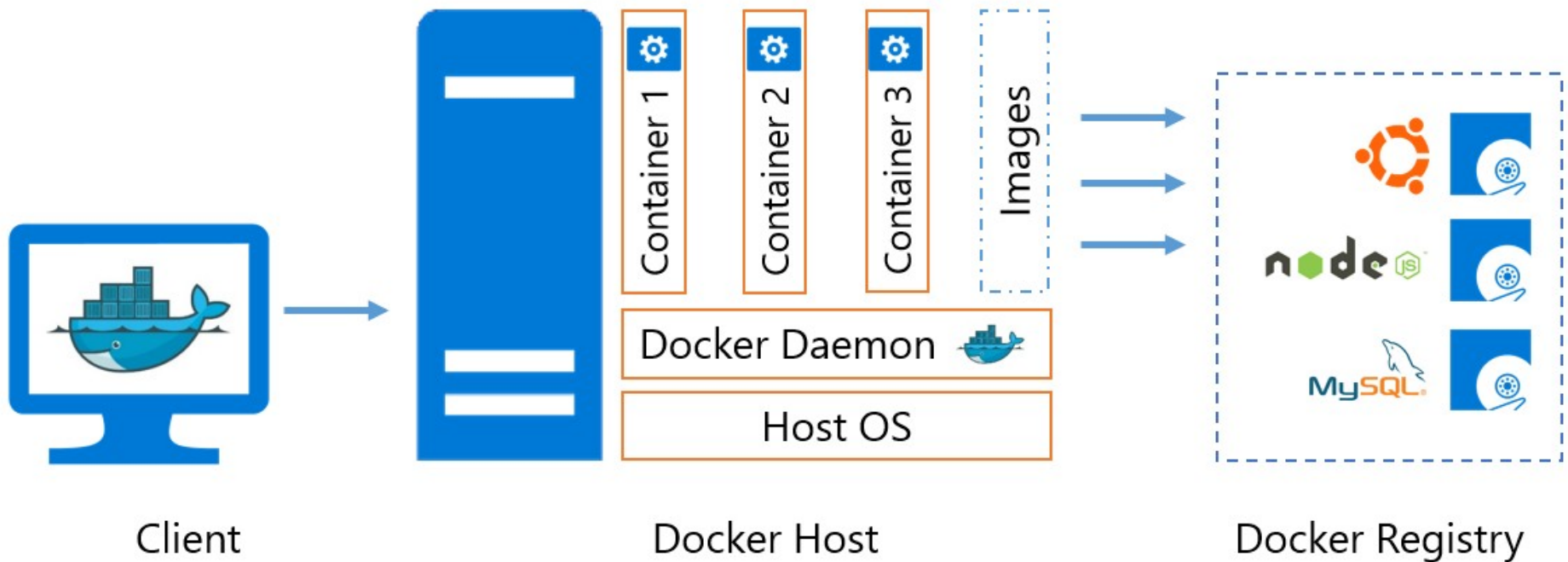
La technologie Docker permet non seulement d'exécuter des conteneurs, mais aussi de simplifier la conception et la fabrication des conteneurs, l'envoi d'images, le contrôle des versions d'image, etc.

Traditional Linux containers vs. Docker



Introduction

Architecture de Docker



Introduction

Docker, les différentes versions...

Capabilities	Community Edition	Enterprise Edition Basic	Enterprise Edition Standard	Enterprise Edition Advanced
Container engine and built in orchestration, networking, security	✓	✓	✓	✓
Certified infrastructure, plugins and ISV containers		✓	✓	✓
Image management			✓	✓
Container app management			✓	✓
Image security scanning				✓

Source:

<https://docs.docker.com/engine/installation/>

Introduction

Modularité

L'approche de Docker en matière de conteneurisation repose sur la décomposition des applications : c'est-à-dire la capacité de réparer ou de mettre à jour une partie d'une application sans devoir désactiver l'ensemble de cette dernière.

En plus de cette approche basée sur les microservices, Docker vous permet de partager des processus entre différentes applications quasiment comme vous le feriez avec une architecture orientée services (SOA).

Introduction

Couches et contrôle des versions d'image

Chaque fichier image Docker est composé d'une série de couches (layers).

Ces couches sont assemblées dans une image unique.

Chaque modification de l'image engendre la création d'une couche.

Chaque fois qu'un utilisateur exécute une commande, comme run ou copy, une nouvelle couche se crée.

Docker réutilise ces couches pour la construction de nouveaux conteneurs, accélérant ainsi le processus de construction.

Introduction

Couches et contrôle des versions d'image

Les modifications intermédiaires sont partagées entre les images, ce qui optimise la vitesse, la taille et l'efficacité.

Qui dit superposition de couches, dit contrôle des versions.

À chaque changement, un journal des modifications est mis à jour afin de vous offrir un contrôle total des images de votre conteneur.

Introduction

Restauration

La fonction la plus intéressante de la superposition de couches est sans doute la restauration.

Chaque image est composée de couches.

Aussi, si l'itération actuelle d'une image ne vous convient pas, vous pouvez restaurer la version précédente.

Cette fonction favorise le développement agile et vous aide à mettre en œuvre les pratiques d'intégration et de distribution continues au niveau des outils.

Introduction

Déploiement rapide

Avant, il fallait plusieurs jours pour mettre en place du nouveau matériel, l'exécuter, l'approvisionner et le rendre disponible.

C'était un processus complexe et fastidieux.

Aujourd'hui, avec les conteneurs Docker, vous pouvez effectuer tout cela en quelques secondes seulement.

En créant un conteneur pour chaque processus, vous pouvez rapidement partager les processus similaires avec les nouvelles applications.

Introduction

Déploiement rapide

De plus, comme vous n'avez pas besoin de redémarrer le système d'exploitation pour ajouter ou déplacer un conteneur, le délai de déploiement s'en trouve encore réduit.

Et ce n'est pas tout. La vitesse du déploiement est telle que vous pouvez vous permettre de créer et de détruire facilement et à moindre coût les données de vos conteneurs, sans aucun problème.

Introduction

Conclusion...

Pour résumer, la technologie Docker propose une approche plus granulaire (détaillée), contrôlable et basée sur des microservices, qui place l'efficacité au cœur de ses objectifs.

Introduction

Composants et éléments de Docker

Image de conteneur

Une image de conteneur contient :

le paramétrage des ressources systèmes :

- système de fichier

- interfaçage réseau

les fichiers qui seront disponibles dans l'environnement virtuel d'exécution au lancement du conteneur.

Introduction

Composants et éléments de Docker

Un conteneur s'exécute à partir d'une image.

Une image peut se baser sur une autre image parente afin d'en réutiliser le contenu et son paramétrage.

Par exemple, pour un conteneur fonctionnant sur Ubuntu, l'image parente de l'image de ce conteneur sera l'image Ubuntu fourni sur le repository officiel de Docker.

Introduction

Fichier Dockerfile

Dockerfile est un fichier texte qui décrit comment construire une image de conteneur Docker.

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

Introduction

Registry

Le registry de Docker permet d'avoir un référentiel d'images de conteneurs pour l'entreprise.

Le registry permet de versionner et de partager des images de conteneurs afin de démarrer des conteneurs à partir de ces images sur des environnements différents : développement, intégration, qualification, production.

Introduction

Registry

Docker a un registry public nommé Docker Hub Registry contenant les images officielles ainsi que les images fournies par la communauté.

De plus, Docker permet de créer son propre registry privé pour indexer ses images privées.

Pour cela, il propose l'image registry pour démarrer rapidement son propre registry dans un conteneur docker.

Introduction

Démon

Le démon linux Docker gère les images et l'exécution des conteneurs :

- construction des images

- envoi et réception des images au registry Docker

- export et import d'image depuis un fichier Tar compressé

- démarrage d'un conteneur à partir d'une image

- arrêt d'un conteneur

- destruction d'une image ou d'un conteneur

Le client Docker [docker](#) en ligne de commandes Linux permet de dialoguer avec le démon Docker pour gérer les conteneurs.

Docker

Installation

Installation

Installation de Docker...

Faire une mise à jour de Debian

```
apt update && apt dist-upgrade
```

```
reboot
```

Installation

Installation de Docker...

Installer les prérequis...

```
apt install apt-transport-https ca-certificates curl gnupg2 software-properties-common
```

Installation

Installation de Docker...

Ajouter la clé GPG

```
curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add -
```


Installation

Installation de Docker...

Vérifier sa version de Linux

```
lsb_release -cs
```

Puis ajouter le dépôt

```
add-apt-repository \  
"deb [arch=amd64] https://download.docker.com/linux/debian \  
$(lsb_release -cs) \  
stable"
```

Installation

Installation de Docker...

Mettre à jour les dépôts...

```
apt update
```

Puis installation...

```
apt install docker-ce
```

Tester son installation

```
docker run hello-world
```

Installation (RedHat)

Installation de Docker sur RedHat / CentOS...

Passer en ligne de commande

```
systemctl set-default multi-user.target
```

Désactiver SE Linux

```
sestatus
```

```
nano /etc/sysconfig/selinux
```

```
SELINUX=disabled
```

```
reboot
```

```
sestatus
```

Installation (RedHat)

Installation de Docker sur RedHat / CentOS...

Ajouter les prérequis...

```
yum install -y yum-utils
```

Ajouter les dépôts...

```
yum-config-manager \
```

```
--add-repo \
```

```
https://download.docker.com/linux/centos/docker-ce.repo
```

Installation (RedHat)

Installation de Docker sur RedHat / CentOS...

Installer Docker

```
yum install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Installer une version spécifique de Docker

```
yum list docker-ce --showduplicates | sort -r
```

```
yum install docker-ce-<VERSION_STRING> docker-ce-cli-  
<VERSION_STRING> containerd.io docker-compose-plugin
```

Installation (RedHat)

Installation de Docker sur RedHat / CentOS...

Démarrer Docker

```
systemctl start docker (systemctl enable docker)
```

Tester Docker

```
docker run hello-world
```

Installation

Installation de Docker...

Tester son installation

```
docker run -d -p 8080:80 docker/getting-started
```

-p 8080:80 TCP port 80 in the container to port 8080 on the Docker host

<http://localhost:8080>

Installation

Installation de Docker...

Pour utiliser Docker en utilisateur normal

`usermod -aG docker stagiaire`

Se déconnecter et se reconnecter...

Installation

Suppression de Docker...

Désinstaller Docker

```
apt-get purge docker-ce
```

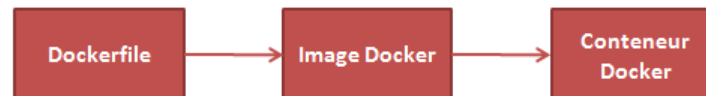
```
rm -rf /var/lib/docker
```

Installation

Cycle de vie

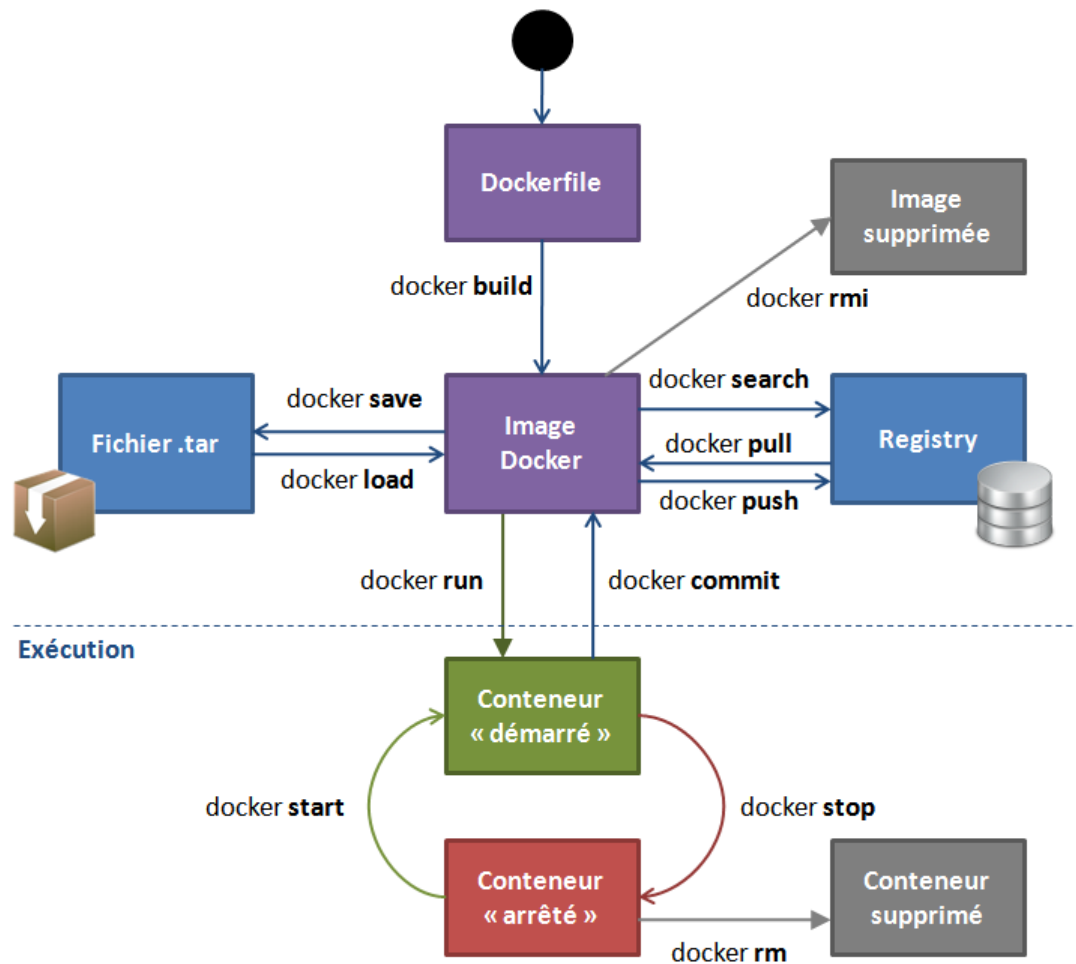
Avec Docker, on part d'un fichier Dockerfile pour construire une image de conteneur.

Cette image servira à démarrer des conteneurs qui utiliseront le contenu de cette image.



Installation

Cycle de vie



Installation

Execution d'une image

La commande `docker run <image>` indique au démon Docker de récupérer l'image en question et de démarrer un conteneur sur cette image.

Exemple :

```
docker run ubuntu
```

```
docker attach ID
```

Installation

Commandes de base...

<code>docker images</code>	liste des images en local
<code>docker ps</code>	liste des conteneurs démarrés
<code>docker ps -a</code>	liste tous les conteneurs
<code>docker logs</code>	attache la sortie console à celle du conteneur
<code>docker inspect</code>	description détaillée du conteneur
<code>docker info</code>	résumé sur l'état et le nombre d'images...
<code>docker version</code>	version de Docker

Installation

Accéder au shell du conteneur démarré

```
docker run -it ubuntu
```

Installation

Supprimer le conteneur...

```
docker ps -a
```

La colonne **CONTAINER ID** contient l'identifiant du conteneur

```
docker rm 5ade4de4bd80 (exemple)
```

Vérifier que le conteneur n'existe plus...

```
docker ps -a
```

Installation

Supprimer l'image...

`docker images`

La colonne **IMAGE ID** contient l'indentifiant de l'image

`docker rmi ubuntu (exemple)`

Vérifier que l'image n'existe plus...

`docker images`

Installation

Commandes groupées...

Arrêter tous les conteneurs

```
docker ps|sed "1 d"|awk '{print $1}'|xargs docker stop
```

Arrêter les conteneurs d'une même image

```
docker ps|sed "1 d"|grep "ubuntu"|awk '{print $1}'|xargs docker stop
```

Installation

Supprimer

Supprimer tous les conteneurs

```
docker ps -a|sed "1 d"|awk '{print $1}'|xargs docker rm
```

Supprimer les conteneurs d'une même image

```
docker ps -a|sed "1 d"|grep "ubuntu"|awk '{print $1}'|xargs docker rm
```

Installation

Supprimer

Supprimer les conteneurs arrêtés

```
docker ps -a|sed "1 d"|grep "Exit"|awk '{print $1}'|xargs docker rm
```

Supprimer toutes les images

```
docker images|sed "1 d"|awk '{print $3}'|sort|uniq|xargs docker rmi
```

Supprimer toutes les images selon leur nom

```
docker images|sed "1 d"|grep "ubuntu"|awk '{print $3}'|sort|uniq|xargs docker rmi
```

Installation

Exécuter en tâche de fond...

```
docker run -d image (-dit ...)
```

- le conteneur tourne en tâche de fond via le démon de Docker
- aucune sortie console n'est affichée

Afficher les conteneurs en cours d'exécution...

```
docker ps -a
```

```
ps aux
```

Installation

Arrêter le conteneur...

Lister les conteneurs

```
docker ps
```

Arrêter le conteneur

```
docker stop 024404fc5d95
```

Relancer le conteneur

```
docker start 024404fc5d95
```

Docker

Images

Images

Votre système d'exploitation est composé de 2 choses : un système de fichiers, et des processus.

Une image Docker représente le système de fichiers, sans les processus.

Elle contient tout ce que vous avez décidé d'y installer (Java, une base de donnée, un script que vous allez lancer, etc...), mais est dans un état inerte.

Images

Les images sont créées à partir de fichiers de configuration, nommés "Dockerfile", qui décrivent exactement ce qui doit être installé sur le système.

Un conteneur est l'exécution d'une image : il possède la copie du système de fichiers de l'image, ainsi que la capacité de lancer des processus.

En fait, c'est un système d'exploitation, avec lequel vous pouvez interagir.

Dans ce conteneur, vous allez donc pouvoir interagir avec les applications installées dans l'image, exécuter des scripts, faire tourner un serveur...

Images

Résumé...

1/ le "Dockerfile" est votre fichier source

2/ l'image est le fichier compilé

3/ le container est une instance de votre classe

Images

Dockerfile...

Le fichier Dockerfile décrit comment une image est construite.

La commande `docker build` construit une image à partir de ce fichier Dockerfile.

Images

Syntaxe...

MAINTAINER <name>

Auteur de l'image

FROM <image>

nom de l'image de base existante qui sert de base à l'image à créer afin de ne pas partir de rien de tout mais du contenu d'une image existante

pour une image fonctionnant sur Ubuntu, l'image de base peut être Ubuntu

Images

Syntaxe...

ENV <key> <value>

Définir une variable d'environnement

ADD <src> <dest>

Ajoute un fichier local dans l'image

VOLUME ["/<dir>"]

Répertoire monté au démarrage du conteneur qui est externe au conteneur.

Images

Syntaxe...

USER <user>

Indique l'utilisateur Linux utilisé lors du démarrage du conteneur

EXPOSE <port>

Indique le port réseau exposé par le conteneur

Ceci est utilisé pour définir sur quels ports réseau communiquent les conteneurs entre eux ou vers l'extérieur de la machine hôte de Docker

Images

Syntaxe...

`RUN <command>`

`RUN ["executable", "param1", "param2"]`

Exécute la commande sur l'image

Images

Syntaxe...

`CMD <command>`

`CMD ["executable", "param1", "param2"]`

Il ne doit y avoir qu'un seul CMD dans le fichier Dockerfile

Il s'agit de la commande qui est lancée à chaque démarrage d'un conteneur basé sur cette image

La commande n'est lancée qu'à l'exécution du conteneur et non durant la construction de l'image

Cette commande peut être redéfinie dans la commande de lancement du conteneur

Images

Voici un exemple simple de fichier Dockerfile :

```
# Se base sur l'image ubuntu version 20.04
```

```
FROM ubuntu:20.04
```

```
# Mettre a jour les packages linux
```

```
RUN apt-get update
```

```
# Lance le shell Bash au lancement du conteneur
```

```
CMD echo "Hello !"
```


Images

Exemple

La ligne **FROM** (obligatoire) indique que cette image se base sur l'image officielle **ubuntu** en version 20.04 disponible sur le registry public de Docker.

La commande suivante **RUN apt-get update** lance la mise à jour des packages Linux.

La dernière ligne **CMD** définit la commande qui sera lancée au démarrage du conteneur (et non lors de la création de l'image) qui est ici de lancer un shell Bash.

Images

Créer un répertoire `image1` dans `/opt`

```
mkdir /opt/image1
```

```
cd /opt
```

Créer le fichier Dockerfile

```
curl -o image1/Dockerfile
```

```
https://raw.githubusercontent.com/darkw0lf/docker/master/Dockerfile
```

Vérifier le contenu du fichier Dockerfile

```
cat image1/Dockerfile
```

Images

Créer l'image à partir du fichier Dockerfile

```
docker build -t image1 image1
```

Lance la commande de création de l'image en indiquant le nom du répertoire contenant le fichier...

Images

L'option `-t image1` n'est pas obligatoire : elle permet de nommer et de tagguer l'image dans le registry afin de la retrouver facilement

Docker télécharge l'image de base Ubuntu 20.04

Il lance ensuite la commande `RUN` pour mettre à jour les packages Linux dans l'image

Docker stocke l'image en local `(/opt/image1)`

Images

Dans la sortie console de la commande `docker build`, la ligne suivante indique l'identifiant de l'image qui vient d'être créée :

Successfully built <identifiant>

Successfully built e2e2638ca29b

L'identifiant de l'image, exemple : `231fc2fcbf5a`, est à utiliser dans les commandes docker pour référencer l'image

Images

Lancer la commande pour visualiser les images disponibles en local :

```
docker images
```

Le nom du tag **image1** permet de repérer l'image qui vient d'être créée

L'image de base **ubuntu** déclarée dans le **FROM** du fichier Dockerfile a aussi été téléchargée et visible dans la liste des images du registry local de Docker

La colonne **IMAGE ID** contient l'identifiant de l'image.

Images

Démarrer un conteneur à partir de l'image

Lancer la commande suivante pour démarrer un conteneur à partir de l'image que nous venons de créer en indiquant l'identifiant de l'image qui a été récupéré via la commande `docker images` (colonne **IMAGE ID**) :

```
docker run <identifiant>
```

```
docker run image1
```

Images

La sortie console de cette commande affiche "Hello !" :

il s'agit de la commande echo "Hello !" définie par la ligne **CMD** à la fin du fichier Dockerfile et qui a été exécuté par Docker au démarrage du conteneur

Cette commande ne lance pas de processus qui tourne en permanence ou en tâche de fond, c'est pourquoi Docker arrête l'exécution du conteneur.

Lancer la commande suivante pour voir la liste des processus en cours d'exécution :

docker ps

Images

Le conteneur qui vient d'être exécuté n'est pas visible

Lancer la commande suivante pour voir la liste de tous les processus, même ceux arrêtés :

```
docker ps -a
```

Le conteneur qui vient d'être exécuté est indiqué comme arrêté...

La colonne **CONTAINER ID** contient l'identifiant du conteneur

La colonne **IMAGE** contient l'identifiant de l'image

Images

Pour qu'un conteneur continue de s'exécuter, il faut qu'il y ait au moins un processus qui tourne en tâche de fond dans celui-ci.

Si aucun processus ne tourne dans le conteneur, ce conteneur est alors arrêté automatiquement par Docker.

TP

Arrêter tous les conteneurs...

Supprimer tous les conteneurs

Supprimer toutes les images

Recréer l'image (**image1**) et démarrer un conteneur

Remplacer la dernière ligne du Dockerfile par :

CMD while true; do ps -aux; sleep 2; done

Supprimer la ligne qui effectue la mise à jour des packages Linux :

RUN apt-get update

TP

Lancer la création de l'image

Récupérer l'identifiant de cette image via la commande :

`docker images`

Lancer l'exécution du conteneur à partir de cette image...

Que se passe t'il ?

TP (solution)

Arrêter tous les conteneurs...

```
docker ps|sed "1 d"|awk '{print $1}'|xargs docker stop
```

Supprimer tous les conteneurs

```
docker ps -a|sed "1 d"|awk '{print $1}'|xargs docker rm
```

Supprimer toutes les images

```
docker images|sed "1 d"|awk '{print $3}'|sort|uniq|xargs docker rmi
```

Recréer l'image et démarrer un conteneur

Remplacer la dernière ligne du Dockerfile par :

```
CMD while true; do ps -aux; sleep 2; done
```

Supprimer la ligne qui effectue la mise à jour des packages Linux...

TP (solution)

Lancer la création de l'image

```
docker build -t image1 image1
```

Récupérer l'identifiant de cette image via la commande :

```
docker images
```

Lancer l'exécution du conteneur à partir de cette image...

```
docker run <identifiant de l'image>
```

La commande `ps -aux` est lancée toutes les 2 secondes sur le conteneur.

Faire **CTRL + C** pour arrêter l'exécution du conteneur...

En tâche de fond :

```
docker run -d <identifiant>
```

Images

Commandes utiles...

Récupérer une image dans une version

```
docker pull ubuntu:trusty
```

Récupérer une image Ubuntu dans sa dernière version

```
docker pull ubuntu:latest
```

Images

Commandes utiles...

Export d'une image sous forme d'archive

```
docker save -o fichier.tar nom_image
```

```
docker save ubuntu > ubuntu.tar (plus simple)
```

Import d'une archive d'image

```
docker load -i fichier.tar
```


Images

Nginx (serveur web)

Récupérer l'image Nginx

```
docker pull nginx
```

Lancer le container sur le port 80 et ayant pour tag "docker-nginx"

```
docker run --name docker-nginx -p 8080:80 nginx
```

-p 8080:80 TCP port 80 in the container to port 8080 on the Docker host

Le container est créé et est lancé, pour l'arrêter faire CTRL + C

```
docker start 4b59731bb892
```

Images

Nginx (serveur web)

Lancer le container sur le port 80 et ayant pour tag "docker-nginx"

```
docker run --restart unless-stopped --name docker-nginx -p 8080:80  
nginx
```

Ou **always**

-p 8080:80 TCP port 80 in the container to port 8080 on the Docker host

Le container tourne toujours après le reboot du host...

```
docker ps
```

<https://docs.docker.com/config/containers/start-containers-automatically/>

Images

Nexcloud (cloud)

Récupérer l'image Nextcloud

```
docker pull nextcloud
```

Lancer le container nextcloud

```
docker run -d -p 8080:80 nextcloud
```

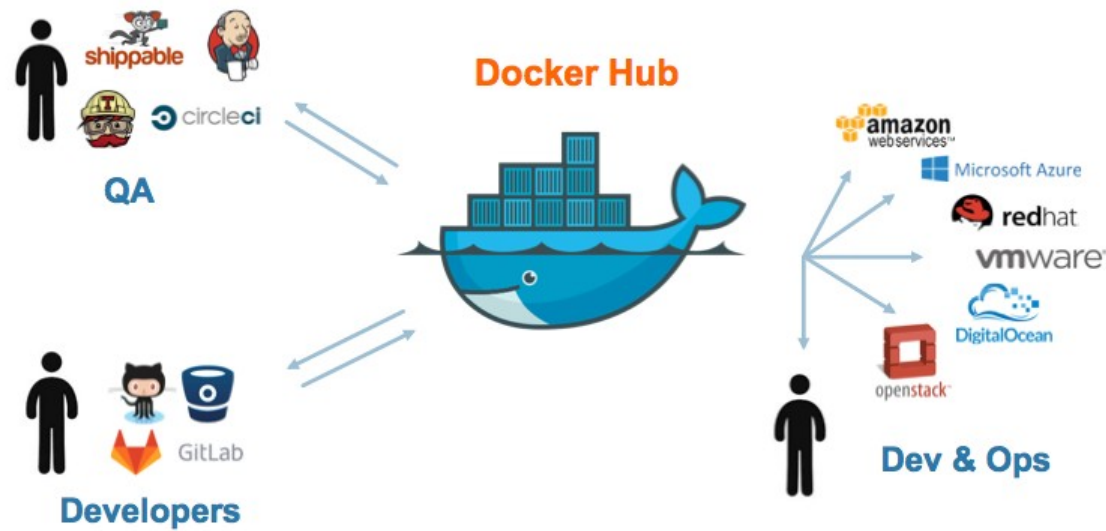
-p 8080:80 TCP port 80 in the container to port 8080 on the Docker host

Le container est créé et est lancé, pour l'arrêter faire **CTRL + C**

...

Images

Docker Hub



Images

Docker Hub

Créer un compte sur :

<https://hub.docker.com/>

Puis se connecter sur son compte en cli :

`docker login --username=username`

Images

Docker Hub

Lister les ID des images locales...

```
docker images
```

Nommer votre image...

```
docker tag bb38976d03cf username/ubuntu:firsttry
```

Envoyer votre image...

```
docker push username/ubuntu
```


Images

PUBLIC REPOSITORY

pascalsene/linux ☆

Last pushed: a few seconds ago

Repo InfoTagsCollaboratorsWebhooksSettings

Tag Name	Compressed Size	Last Updated	
ubuntu	32 MB	a few seconds ago	

Images

Docker Hub

Chercher une image sur Ubuntu

```
docker search ubuntu
```

Récupérer une image d'un repository

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

Envoyer une image sur un repository

```
docker push [OPTIONS] NAME[:TAG]
```


Images

Docker Hub

Chercher une image sur Ubuntu par note et officielle...

```
docker search --filter "is-official=true" --filter "stars=3" busybox
```

```
docker search ubuntu | grep 20.04
```

Images

Docker Hub

Chercher par le tag...

```
wget https://raw.githubusercontent.com/darkw0lf/docker/master/dockertags
```

```
chmod +x dockertags
```

```
./dockertags ubuntu
```

Registry

Créer son propre registry...

Installation...

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

Registry

Créer son propre registry...

Mettre un tag sur une image locale...

```
docker tag feb5d9fea6a5 localhost:5000/toto (ID de l'image locale...)
```

Registry

Créer son propre registry...

Envoyer l'image...

```
docker push localhost:5000/toto
```

Puis récupérer l'image...

```
docker pull localhost:5000/toto
```

Docker

Réseau

Réseau

Docker réseau

Lors de l'installation de Docker, trois réseaux sont créés automatiquement.

`docker network ls`

```
CONTAINER ID      IMAGE      COMMAND      CREATED
[root@formation:/home/stagiaire# docker network ls
NETWORK ID        NAME        DRIVER        SCOPE
865787247c67     bridge     bridge        local
036715f6faff     host       host          local
085e26088d28     none       null          local
[root@formation:/home/stagiaire#
```

Un réseau de type bridge est créé...

Réseau

Le réseau Bridge est présent sur tous les hôtes Docker.

Lors de la création d'un conteneur, si l'on ne spécifie pas un réseau particulier, les conteneurs sont connectés au Bridge `docker0`.

La commande `ifconfig` ou `ip a` fournit les informations sur le réseau ponté (bridge).

Réseau

La commande `docker network inspect bridge`, retourne les informations concernant ce réseau...

Réseau

Exercice...

Créer deux containers avec l'image Ubuntu...

```
docker run -itd --name=container1 ubuntu
```

```
docker run -itd --name=container2 ubuntu
```

Réseau

Exercice...

Informations du réseau avec :

`docker network inspect bridge`

Les conteneurs sont connectés au **Bridge** par défaut `docker0` et peuvent communiquer entre eux par adresse IP, les conteneurs se trouvent alors sur le même réseau.

Réseau

Réseaux définis par l'utilisateur

Il est recommandé d'utiliser des réseaux Bridgés définis par l'utilisateur pour contrôler quels conteneurs peuvent communiquer entre eux, et ainsi permettre la résolution DNS des noms d'hôtes des conteneurs avec les adresses IP.

Réseau

Réseaux définis par l'utilisateur

Docker fournit des pilotes pour créer ces réseaux.

Réseau Bridgé, réseau overlay (Multihost) ou un réseau MACVLAN (Overlay et MacVlan).

On peut également créer des plugins réseau ou utiliser un réseau distant.

La librairie réseau de Docker , libnetwork peut agir comme un proxy pour des plugins distant, remote driver.

Réseau

Réseaux définis par l'utilisateur

Le réseau ponté ou Bridgé est le réseau par défaut lors de la création de nouveaux réseaux.

On peut créer autant de réseaux que l'on souhaite et on peut connecter un conteneur à zéro ou plus de ces réseaux à tout moment.

On peut également, connecter et déconnecter des conteneurs en cours d'exécution des réseaux sans redémarrer ces derniers.

Réseau

Réseaux bridgés...

Un réseau Bridgé est le type de réseau le plus utilisé dans Docker.

Les réseaux Bridgés créés par les utilisateurs sont semblables au réseau bridge par défaut créé à l'installation de Docker **Docker0**.

De nouvelles fonctionnalités sont ajoutées, la gestion du DNS par exemple.

Lors de la création de nouveau réseau, une nouvelle interface est créée.

Cette interface est pontée...

Réseau

Réseaux bridgés...

```
apt install bridge-utils
```

```
brctl show
```

Voici un résultat ressemblant à :

```
docker0          8000.0242fbdce780    no          veth46fbe31
```


Réseau

Réseaux bridgés...

Création d'un réseau bridgé Formation

```
docker network create --driver bridge formation
```

```
docker network inspect formation
```

```
docker network ls
```

Réseau

Réseaux bridgés...

Connecter des containers...

```
docker run --network formation -itd --name container3 ubuntu
```

```
docker network inspect formation
```

Réseau

Réseaux bridgés...

Connexion d'un container au réseau **formation**

```
docker run --network formation -itd --name container4 ubuntu
```

```
docker network connect formation container2
```

```
docker network inspect formation
```

Réseau

Serveur DNS

Docker exécute un serveur DNS intégré qui fournit une résolution de noms aux conteneurs connectés au réseau créé par les utilisateurs, de sorte que ces conteneurs peuvent résoudre les noms de d'hôtes en adresses IP.

Si le serveur DNS intégré est incapable de résoudre la demande, il sera transmis à tous les serveurs DNS externes configurés pour le conteneur.

Pour faciliter cela lorsque le conteneur est créé, seul le serveur DNS intégré **127.0.0.11** est renseigné dans le fichier `resolv.conf` du conteneur.

Réseau

Serveur DNS

Testons le serveur DNS intégré...

`docker network create --driver bridge formation` (existe déjà normalement !)

`docker network inspect formation`

`docker attach container4` (permet de se connecter au container...)

`root@3342d2e4c660:/#`

`apt-get update`

`apt-get install dnsutils`

Réseau

Serveur DNS

Testons le serveur DNS intégré...

```
root@3342d2e4c660:/#
```

```
dig
```

```
dig container3 (yes il répond !)
```

```
cat /etc/resolv.conf
```

```
nameserver 127.0.0.11
```

```
options ndots:0
```

Réseau

Serveur DNS

Testons le serveur DNS intégré...

```
docker exec container4 cat /etc/resolv.conf
```

```
nameserver 127.0.0.11
```

```
options ndots:0
```

Réseau

Ports et Ip...

Testons le numéro de port...

Permet de tester la connexion du container

`docker port ID`

`80/tcp -> 0.0.0.0:80`

Réseau

Driver réseau Bridge

Le pilote réseau par défaut. Si vous ne spécifiez pas de pilote, il s'agit du type de réseau que vous créez.

Les réseaux en pont sont généralement utilisés lorsque vos applications s'exécutent dans des conteneurs autonomes devant communiquer.

Réseau

Driver réseau Host

Pour les conteneurs autonomes, supprimez l'isolation réseau entre le conteneur et l'hôte Docker et utilisez directement la mise en réseau de l'hôte.

Host est uniquement disponible pour les services Swarm sur Docker 17.06 et supérieur.

Réseau

Driver réseau Overlay

Les réseaux superposés (overlay) connectent plusieurs démons Docker ensemble et permettent aux services swarm de communiquer entre eux.

Vous pouvez également utiliser des réseaux superposés pour faciliter la communication entre un service Swarm et un conteneur autonome ou entre deux conteneurs autonomes sur différents démons Docker.

Cette stratégie élimine le besoin de routage au niveau du système d'exploitation entre ces conteneurs

Réseau

Driver réseau Macvlan

Les réseaux Macvlan vous permettent d'attribuer une adresse MAC à un conteneur, le faisant apparaître comme un périphérique physique sur votre réseau.

Le démon Docker route le trafic vers les conteneurs en fonction de leurs adresses MAC.

Le pilote macvlan est parfois le meilleur choix lorsque vous utilisez des applications héritées qui s'attendent à être directement connectées au réseau physique, plutôt que routées via la pile réseau de l'hôte Docker.

Réseau

Driver réseau None

Pour ce conteneur, désactivez tous les réseaux.

Habituellement utilisé avec un pilote réseau personnalisé.

Aucun n'est disponible pour les services Swarm...

Docker

Persistence

Persistence

Ajouter un volume de données

La gestion des volumes se fait avec l'option `-v` (pour `--volume`) des commandes `docker run` ou `docker create`.

Évidemment, plusieurs `-v` peuvent être placés les uns après les autres en fonction du nombre de volumes souhaités.

Persistence

Ajouter un volume de données

Ajoutons un volume à l'image de MariaDB (a récupérer)

par exemple :

```
docker run --name mariadb-test --volume /var/lib/mysql -e  
MYSQL_ROOT_PASSWORD=123456 -d mariadb
```


Persistance

Ajouter un volume de données

Ici le volume `/var/lib/mysql` a été créé dans mon container.

Il est par défaut accessible en lecture / écriture mais il est possible de restreindre ces accès en utilisant `/var/lib/mysql:ro` pour **read-only**.

```
docker run --name mariadb-test --volume /var/lib/mysql:ro -d mariadb
```

```
-v myvolmysql:/app
```

Persistence

Où trouver ce volume sur mon hôte ?

Toujours grâce à la commande `inspect` :

```
docker inspect mariadb-test
```

Où « `Source` » indique le dossier sur l'hôte et « `Destination` » le point de montage dans le container.

« `RW` » indiquant si le volume est accessible en lecture / écriture.

Persistence

Où trouver ce volume sur mon hôte ?

Les volumes sont toujours conservés, que les containers soient arrêtés ou non.

En revanche le volume (et les données) est supprimé une fois que le dernier container y faisant référence l'est aussi.

Persistence

Copie du host vers un container et vice versa...

```
docker cp /etc/resolv.conf container:/opt/resolv.conf
```

```
docker cp container:/etc/resolv.conf /root/resolv.conf
```

Le nom du container c'est par exemple : 8e5b3b8707e6

```
docker cp /etc/resolv.conf 8e5b3b8707e6:/opt/resolv.conf
```

Persistence

Lister l'ID du container avec la commande docker ps :

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
72ca2488b353	my_image		X hours ago	Up X hours		container

Copie un fichier depuis le host vers le container :

```
docker cp foo.txt 72ca2488b353:/foo.txt
```

Copie un fichier depuis le host Docker vers le container :

```
docker cp 72ca2488b353:/foo.txt foo.txt
```

Persistence

Images en mode persistant avec un bash Linux...

Bash Linux avec la distribution R et partage de ses données...

```
docker run -it -v ~/Downloads:/Down r-base bash
```

Persistence

Images en mode persistant avec un bash Ubuntu...

Idem mais avec Ubuntu...

```
docker pull ubuntu
```

```
docker run -it -v ~/Downloads:/Down ubuntu bash
```

Docker

Ecosystème

Docker Compose

Docker Compose

Docker Compose est un outil qui permet de décrire (dans un fichier YAML) et gérer (en ligne de commande) plusieurs conteneurs comme un ensemble de services inter-connectés.

Il permet d'installer une stack (suite de containers) avec toutes ses dépendances...



Docker Compose

Docker Compose

Installation :

```
apt-get install docker-compose
```

Docker Compose

Dockerfile

Wordpress...

wget

<https://raw.githubusercontent.com/darkw0lf/docker/master/docker-compose.yml>

Docker Compose

Docker Compose

Lancer ce build...

`docker-compose up -d`

Docker Compose

Docker Compose

Autre stack...

https://raw.githubusercontent.com/darkw0lf/docker/master/compose_laravel.txt

Docker Compose

Docker Compose

Pour démarrer :

```
docker-compose up -d
```

Pour arrêter :

```
docker-compose down
```

Docker Compose

Docker Compose

Exécute le projet :

```
docker-compose run -d
```

Liste les conteneurs du projet :

```
docker-compose ps
```

Docker Compose

Docker Compose

Arrête les conteneurs du projet :

```
docker-compose stop
```

Supprime les conteneurs du projet :

```
docker-compose down --volumes
```


Docker Machine

Docker Machine

Docker Machine est un outil permettant de déployer des hôtes docker sur différentes plateformes...

Nous travaillons sur des machines GNU/Linux, il existe des possibilités pour faire tourner **docker-machine** sous MacOS et Windows.

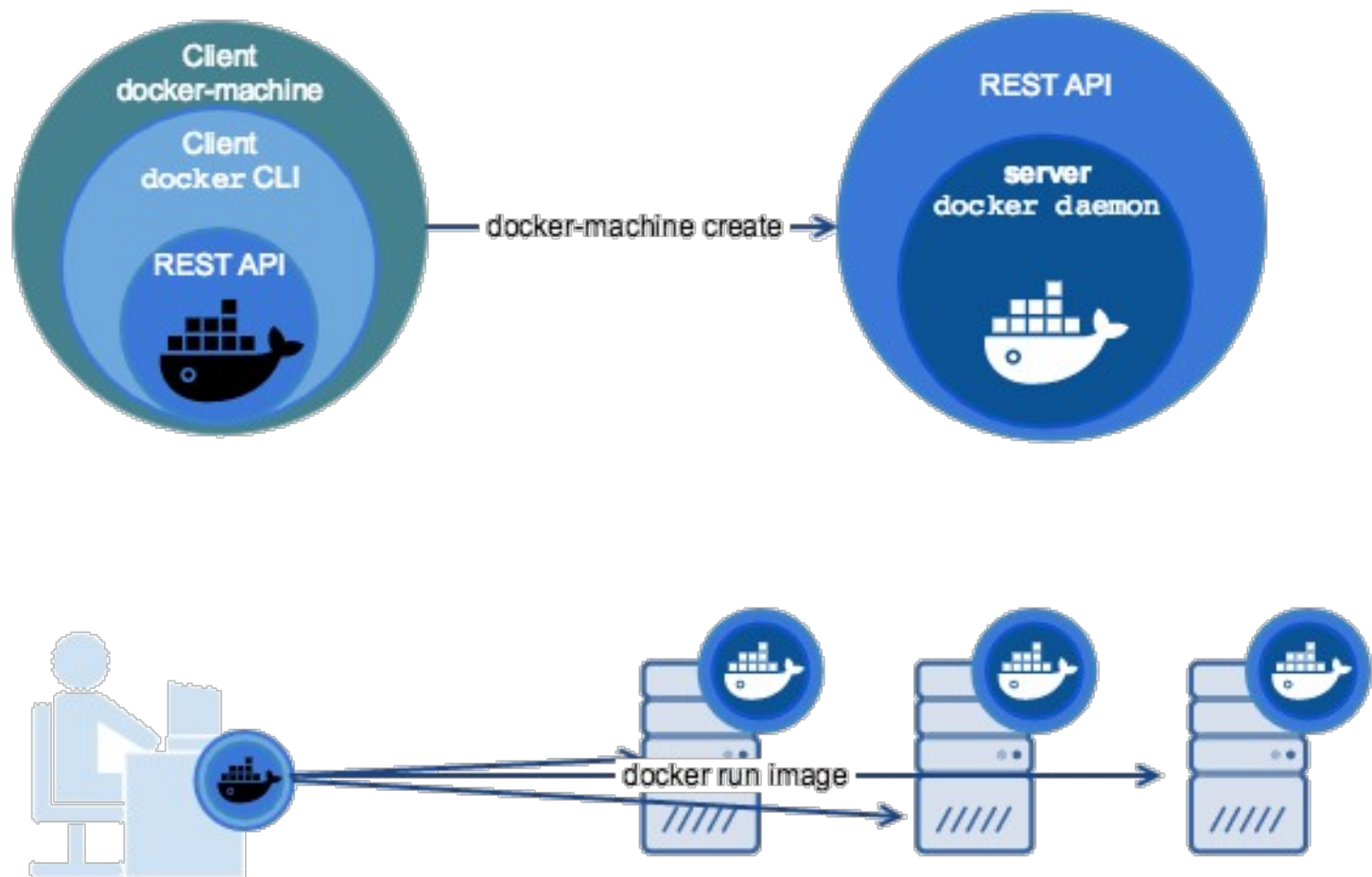


Docker Machine on Mac



Docker Machine on Windows

Docker Machine



Docker Machine

Installation...

```
base=https://github.com/docker/machine/releases/download/v0.16.0
curl -L $base/docker-machine-$(uname -s)-$(uname -m)
>/tmp/docker-machine

mv /tmp/docker-machine /usr/local/bin/docker-machine

chmod +x /usr/local/bin/docker-machine

docker-machine -v
```

Docker Machine

Installation...

<https://docs.docker.com/machine/install-machine/>

<https://devopssec.fr/article/deployer-gerer-vos-hotes-docker-avec-docker-machine>

Docker Machine

Installer Virtualbox sur son Linux par les dépôts !

```
echo "deb http://download.virtualbox.org/virtualbox/debian buster contrib" |  
sudo tee /etc/apt/sources.list.d/virtualbox.list
```

```
wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc -O- |  
sudo apt-key add -
```

```
wget -q https://www.virtualbox.org/download/oracle_vbox.asc -O- | sudo  
apt-key add -
```

```
apt-get update
```

```
apt-get install virtualbox-6.0
```

Docker Machine

Ou installer Virtualbox sur son Linux en direct !

```
wget
```

```
https://download.virtualbox.org/virtualbox/6.1.16/virtualbox-6.1\_6.1.16-140961~Debian~stretch\_amd64.deb
```

```
dpkg -i virtualbox-6.1_6.1.16-140961~Debian~stretch_amd64.deb
```

```
apt install -f
```

Docker Machine

Drivers...

Le Driver correspond à la plateforme sur laquelle vous allez pouvoir déployer vos machines. La liste est assez intéressante et diversifiée :

<https://docs.docker.com/machine/drivers/>

Docker Machine

Création d'une machine Virtualbox...

Installer Virtualbox sur son Linux !

```
docker-machine create --driver virtualbox vbox-test
```

Soyons patients !

Activer la virtualisation dans le BIOS...

Si erreur de certificats...

```
rm -rf .docker/machine/certs
```


Docker Machine

On se connecte sur la machine...

```
docker-machine ssh vbox-test (facultatif...)
```

```
Boot2Docker version 1.11.0, build HEAD : 32ee7e9 - Wed Apr 13  
20:06:49 UTC 2022
```

```
Docker version 1.11.0, build 4dc5990
```

```
docker@vbox-test:~$
```

Docker Machine

Pour lancer un conteneur sur cette nouvelle machine :

Test...

```
docker-machine ls
```

```
docker-machine env vbox-test
```

```
eval $(docker-machine env vbox-test)
```

```
docker-machine active
```

Doit répondre : `vbox-test`

```
docker run -d -p 8000:80 --name vbox-test-httpd httpd
```

Docker Machine

Quelques commandes utiles...

`docker-machine stop machine`

`docker-machine restart machine`

`docker-machine start machine`

`docker-machine rm machine (!!!)`

Docker Machine

Generic...

```
docker-machine create --driver generic --generic-ip-address=IP --  
generic-ssh-user=root mac03-gen
```

<https://docs.docker.com/machine/drivers/generic/>

Docker Machine

Hyper-V...

```
docker-machine create --driver hyperv vm
```

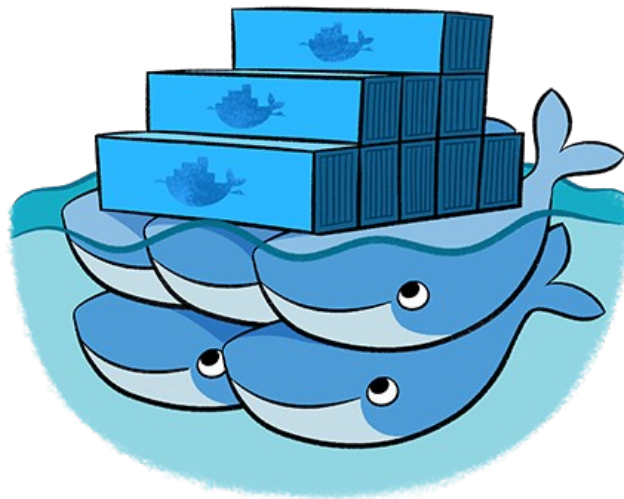
<https://docs.docker.com/machine/drivers/hyper-v/>

Swarm

Un Swarm est un groupe de machines exécutant le moteur Docker et faisant partie du même cluster.

Docker swarm vous permet de lancer des commandes Docker auxquelles vous êtes habitué sur un cluster depuis une machine maître nommée manager/leader Swarm.

Quand des machines rejoignent un Swarm, elles sont appelés nœuds.



Swarm

Les managers Swarm sont les seules machines du Swarm qui peuvent exécuter des commandes Docker ou autoriser d'autres machines à se joindre au Swarm en tant que workers.

Les workers ne sont là que pour fournir de la capacité et n'ont pas le pouvoir d'ordonner à une autre machine ce qu'elle peut ou ne peut pas faire.

Swarm

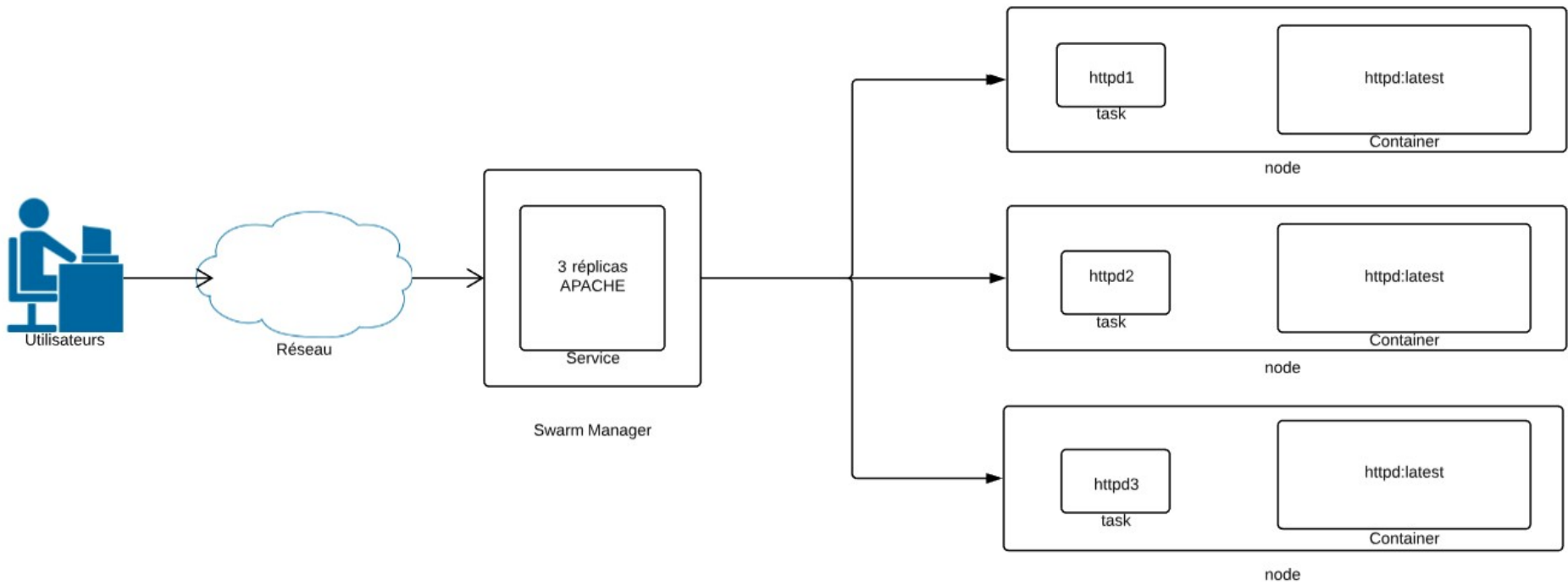
Jusqu'à présent, vous utilisiez Docker en mode hôte unique sur votre ordinateur local.

Mais Docker peut également être basculé en mode swarm permettant ainsi l'utilisation des commandes liées au Swarm.

L'activation du mode Swarm sur hôte Docker fait instantanément de la machine actuelle un manager Swarm.

À partir de ce moment, Docker exécute les commandes que vous exécutez sur le Swarm que vous gérez, plutôt que sur la seule machine en cours.

Swarm



Swarm

Création du Cluster Swarm

```
docker swarm init --advertise-addr <ip du host>
```

Suivez les instructions en faisant un copier-coller de la commande sur chaque node (host) qui servira de worker...

Vous pouvez vérifier l'état de votre cluster Swarm grâce à cette commande (à exécuter depuis le manager)

```
docker node ls
```

Swarm

Création d'un service

Avec swarm nous allons beaucoup parler de service.

Un service est, pour ainsi dire, le point d'entrée de votre application.

Un service est composé de 1-N container et apporte une dimension de clustering, de fault tolérance et de haute disponibilité à votre environnement docker.

- Dans l'exemple ci-dessous, nous allons créer un service httpd (apache) nommé "monserveurweb"

```
docker service create --name monserveurweb --publish 80:80 httpd
```

Swarm

Vérifiez ensuite l'état de votre service

```
docker service ls
```

Vous pourrez constater que si vous tapez `http://<ip de votre manager ou d'un node>` depuis votre navigateur que le serveur apache hébergé dans le container répond bien...

Swarm

Scalabilité

En faisant un `docker service ls`, vous pouvez voir que le nombre de replica est à 1.

Cela signifie qu'un seul container a été provisionné et déployé dans votre cluster.

Il est possible d'augmenter le nombre de replica via cette commande

```
docker service update --replicas <nombre de replica> <nom de votre service>
```

```
docker service scale <nom de votre service>=<nombre de replicas>
```

Swarm

Scalabilité

```
docker service ps <id de votre service>
```

Swarm

Scalabilité

Ajouter un manager

Vous l'aurez compris, le manager est un élément critique de votre cluster Swarm.

Il est très fortement conseillé d'en avoir plusieurs afin de limiter au maximum le risque d'incidents critiques.

Pour créer un nouveau manager (qui sera un manager en standby):

- Récupérez le token:

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

Swarm

Scalabilité

Copiez-collez la commande qui vous est indiquée sur le host qui vous servira de manager

En faisant un "docker node ls" vous pouvez voir votre nouveau node avec le statut "reachable".

Ce qui voudra dire qu'il prendra le relais en cas de perte du manager.

`docker node ls`

Swarm

Scalabilité

Attention toutefois: afin que l'algorithme raft fonctionne bien (ce qui permet l'élection d'un nouveau manager en cas de perte de celui-ci), il est préconisé que le nombre de manager désigné soit impair...

Swarm

Fonctionnalités

Exemple de commande pour fixer des quota de ressources

```
docker service update --limit-cpu=.5 --reserve-cpu=.75 --limit-memory=128m --reserve-memory=256 monserveurweb
```

Swarm

Fonctionnalités

Cette fonctionnalité vous permettra de "tagger" vos nodes et vos services. Cela vous sera très utile; par exemple, si vous voulez que certains services se déploient sur certain nodes

Affecter un label à un node

```
docker node update --label-add nodelabel=<nom du label> <id de votre node>
```

Swarm

Fonctionnalités

Cette fonctionnalité vous permettra de "tagger" vos nodes et vos services. Cela vous sera très utile; par exemple, si vous voulez que certains services se déploient sur certain nodes

Créer un service qui sera déployé sur un node en fonction de son label

```
docker service create --name <nom du service> --constraints  
'node.labels.nodelabel=='<nom du label>' <nom de l'image>
```

Swarm

Fonctionnalités

Supprimer un node

Sur votre node tapez la commande suivante

```
docker swarm leave
```

Swarm

Fonctionnalités

Supprimer un node

Depuis le manager tapez la commande suivante

```
docker node rm <id de votre node>
```

Swarm

Fonctionnalités

Swarm et docker-compose

Il est possible de créer un service depuis un yaml docker-compose grâce à cette commande

```
docker stack deploy --compose-file <fichiercompose.yml> <nom service>
```

Swarm

Docker Swarm

Il existe un outil sympathique qui permet de visualiser le cluster docker swarm ainsi créé et de visualiser les containers qui sont déployés dans le cluster...

```
docker run -it -d -p 5000:8080 -v /var/run/docker.sock:/var/run/docker.sock  
dockersamples/visualizer
```


Swarm (autre exemple)

Docker Swarm

Executer cette commande :

```
docker service create --name helloworld alpine:3.6 ping docker.com
```

Vous obtiendrez l'adresse ip du leader. Maintenant ouvrez votre navigateur préféré avec l'url suivant : [http://\[ip leader\]:5000](http://[ip leader]:5000)

<https://www.it-wars.com/posts/virtualisation/docker-swarm-par-lexemple/>

<https://journaldunadminlinux.fr/tutoriel-installer-et-utiliser-dockerswarm/>

Docker

Concepts avancés

Concepts avancés

Créer une partition séparée pour Docker

L'ensemble des données de docker sont stocké dans [/var/lib/docker](#).

C'est le répertoire par défaut que docker utilise pour stocker toutes ses images et ses containers.

Dès les premiers jours d'utilisation, on se rend vite compte que Docker prend rapidement de la place.

Concepts avancés

Créer une partition séparée pour Docker

L'inconvénient de ce répertoire par défaut, c'est qu'il est situé sous le répertoire racine / et que si Docker remplit ce répertoire, votre répertoire racine sera également plein, ce qui va rendre votre système hôte inutilisable.

Une image mal intentionnée pourrait volontairement se mettre à occuper tout l'espace disponible pour votre système hôte.

Même sans un esprit malveillant, Docker peut en tout état de cause occuper tout l'espace disponible après seulement quelques pull d'images.

Concepts avancés

Créer une partition séparée pour Docker

Il faut créer une partition physique séparée pour le répertoire [/var/lib/docker](#) dès l'installation de votre système hôte.

Si votre système est déjà installé, créez une partition logique avec LVM (Logical Volume Manager).

Ces 2 solutions permettent de définir un quota à ne pas dépasser et ne mettre pas en péril votre partition racine.

Concepts avancés

Maintenez votre système hôte à jour

Que ce soit votre système linux, le kernel ou Docker Engine, assurez-vous que votre système soit bien à jour.

Mettez à jour votre noyau linux et utilisez la version recommandée, évitez les versions instables.

Assurez vous de mettre régulièrement à jour Docker dès qu'une version stable est sortie.

Concepts avancés

Interdire les communications entre les containers

Par défaut, la communication entre tous les containers est possible sans forcément utiliser la fonction réseau.

Une mauvaise image pourrait donc faire du sniffing et voir tout ce qui se passe sur le sous-réseau Docker de votre système hôte.

C'est particulièrement dangereux car la plupart du temps, il n'y a pas de connexion sécurisé entre vos containers que vous considérez comme "isolé" sur le sous-réseau [docker0](#)

Concepts avancés

Interdire les communications entre les containers

La bonne pratique est d'interdire ce comportement par défaut.

Seul les containers liés entre eux par la fonction link de docker seront capable de communiquer entre eux.

Cette solution est disponible nativement avec Docker.

Il suffit de passer le paramètre `-icc=false` au daemon.

Sous debian et ubuntu, cette opération se fait dans le fichier `/etc/default/docker` en modifiant la variable `DOCKER_OPTS`

```
DOCKER_OPTS="-icc=false"
```


Concepts avancés

N'utilisez pas n'importe quel registry

La plupart des images sont téléchargeables depuis le registry Docker Hub.

D'une manière générale, préférez toujours les images "official" proposées et validées par la société Docker.

Si ce n'est pas le cas, assurez que l'image a suffisamment été téléchargée et à priori "testés" par les autres utilisateurs.

N'hésitez pas non plus à aller vérifier le Dockerfile sur github afin de s'assurer que l'image correspond à ce qu'elle est censé être.

Concepts avancés

Créer un utilisateur dans votre Dockerfile

La plupart de vos applications ne nécessite pas d'être lancé en root, c'est même très rarement le cas.

Par conséquent, il n'est pas nécessaire d'utiliser le user root dans vos containers.

La création d'un utilisateur dans votre image peut se faire directement depuis le Dockerfile avec les 2 instructions suivantes :

```
RUN useradd -d /home/myappuser -m -s /bin/bash myappuser
```

```
USER myappuser
```

Concepts avancés

Ne lancer pas vos containers en root

Cela rejoint le point précédent, mais si votre image ne nécessite l'utilisateur root, il est préférable de ne pas lancer le container avec l'utilisateur root...

```
docker run -u <Username or ID> <Run args> <Container Image  
Name or ID> <Command>
```

Concepts avancés

Ne mapper que les ports utiles

Docker propose un mapping entre les ports déclarés par le Dockerfile et ceux ouvert sur votre système hôte.

L'option `-P` (en majuscule) permet lancer un container en exposant tous les ports déclarés dans votre Dockerfile.

N'utiliser jamais l'option `-P` (en majuscule), préférez l'option `-p` (en minuscule) qui permet mapper les ports un par un.

`docker run -p 80:80 apache`

le premier port 80 correspond à celui de la machine hôte et le second au port interne du conteneur

Concepts avancés

Ne mapper que les ports utiles

De plus, si vous n'avez pas besoin de rendre public l'adresse IP, vous pouvez spécifier sur quel interface réseau vous souhaitez écouter

```
docker run -p 192.168.1.100:9200:9200 ubuntu
```

Cette commande permet de n'ouvrir le port 9200 que pour votre réseau local. Une connexion depuis une machine hors réseau local (internet) sera refusée.

Concepts avancés

Permission sur les fichiers

De la même manière qu'on doit vérifier les permissions du répertoire `/var/www` pour un serveur apache, il n'y a pas de raison de ne pas vérifier les permissions pour un container faisant tourner un serveur apache.

Concepts avancés

Permission sur les fichiers

Les bonnes pratiques de sécurité Linux en terme de permissions doivent s'appliquer à toutes vos images et volumes.

Docker est certes très facile à prendre en main, mais il faut pour autant comprendre les notions de base d'un système unix pour assurer à minima la sécurité d'un container tournant sous Linux grâce aux permissions et aux groupes utilisateurs.

Sécurité

Sécurisé Docker avec TLS

Créer les certificats...

```
mkdir -p /etc/docker/ssl
```

```
mkdir -p ~/.docker
```


Sécurité

Sécurisé Docker avec TLS

Créer les certificats...

```
openssl genrsa -out ~/.docker/ca-key.pem 2048
```

```
openssl req -x509 -new -nodes -key ~/.docker/ca-key.pem \  
-days 10000 -out ~/.docker/ca.pem -subj '/CN=docker-CA'
```

```
ls ~/.docker/
```

```
ca-key.pem  ca.pem
```

```
cp ~/.docker/ca.pem /etc/docker/ssl
```

Sécurité

Sécurisé Docker avec TLS

Créer une configuration pour le client...

```
vi ~/.docker/openssl.cnf
```

Sécurité

Sécurisé Docker avec TLS

[req]

req_extensions = v3_req

distinguished_name = req_distinguished_name

[req_distinguished_name]

[v3_req]

basicConstraints = CA:FALSE

keyUsage = nonRepudiation, digitalSignature, keyEncipherment

extendedKeyUsage = serverAuth, clientAuth

Sécurité

Sécurisé Docker avec TLS

Idem pour le server...

`vi /etc/docker/ssl/openssl.cnf`

Sécurité

Sécurisé Docker avec TLS

[req]

req_extensions = v3_req

distinguished_name = req_distinguished_name

[req_distinguished_name]

[v3_req]

basicConstraints = CA:FALSE

keyUsage = nonRepudiation, digitalSignature, keyEncipherment

extendedKeyUsage = serverAuth, clientAuth

subjectAltName = @alt_names

Sécurité

Sécurisé Docker avec TLS

[alt_names]

DNS.1 = docker.local

IP.1 = 172.17.8.101

IP.2 = 127.0.0.1

Sécurité

Sécurisé Docker avec TLS

Création du certificat pour le client

```
openssl genrsa -out ~/.docker/key.pem 2048
```

```
openssl req -new -key ~/.docker/key.pem -out ~/.docker/cert.csr \  
-subj '/CN=docker-client' -config ~/.docker/openssl.cnf
```

```
openssl x509 -req -in ~/.docker/cert.csr -CA ~/.docker/ca.pem \  
-CAkey ~/.docker/ca-key.pem -CAcreateserial \  
-out ~/.docker/cert.pem -days 365 -extensions v3_req \  
-extfile ~/.docker/openssl.cnf
```

Sécurité

Sécurisé Docker avec TLS

Création du certificat pour le serveur

```
openssl genrsa -out /etc/docker/ssl/key.pem 2048
```

```
openssl req -new -key /etc/docker/ssl/key.pem \
```

```
-out /etc/docker/ssl/cert.csr \
```

```
-subj '/CN=docker-server' -config /etc/docker/ssl/openssl.cnf
```

```
openssl x509 -req -in /etc/docker/ssl/cert.csr -CA ~/.docker/ca.pem \
```

```
-CAkey ~/.docker/ca-key.pem -CAcreateserial \
```

```
-out /etc/docker/ssl/cert.pem -days 365 -extensions v3_req \
```

```
-extfile /etc/docker/ssl/openssl.cnf
```


Sécurité

Sécurisé Docker avec TLS

Modification de la conf du serveur...

`vi /etc/systemd/system/multi-user.target.wants/docker.service`

Sécurité

Sécurisé Docker avec TLS

Remplacer la ligne :

ExecStart=/usr/bin/dockerd

Par :

ExecStart=/usr/bin/dockerd -H tcp://localhost:2376 --tls --tlskey /etc/docker/ssl/key.pem --tlscert /etc/docker/ssl/cert.pem

systemctl daemon-reload

systemctl restart docker

Sécurité

Sécurisé Docker avec TLS

Tester la connexion...

```
docker -H tcp://127.0.0.1:2376 info
```

Sécurité

Sécurisé Docker avec TLS

Ajout de variables pour le client...

```
export DOCKER_HOST=tcp://127.0.0.1:2376
```

```
export DOCKER_TLS_VERIFY=1
```

```
export DOCKER_CERT_PATH=~/.docker
```

```
docker info
```

Docker

Orchestration

Portainer

Qu'est-ce Portainer ?

C'est un projet open-source qui permet de gérer et d'orchestrer des containers Docker.

Portainer est une interface Web (WebUI) open source qui permet de créer, modifier, redémarrer, surveiller... des conteneurs Docker.

Le petit plus de l'outil, c'est également de pouvoir interconnecter plusieurs serveurs utilisant Docker (via Portainer Agent), afin de contrôler/surveiller les conteneurs répartis sur plusieurs serveurs depuis la même interface et cela très simplement.

Portainer

Installer Portainer

Récupérer la dernière version...

```
docker pull portainer/portainer
```



Portainer

Portainer

Créer un volume...

```
docker volume create portainer_data
```

Lancer le container...

```
docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --  
restart=always -v /var/run/docker.sock:/var/run/docker.sock -v  
portainer_data:/data portainer/portainer
```


Rancher



Qu'est-ce Rancher ?

C'est un projet open-source qui permet de gérer et d'orchestrer des containers Docker.

Le projet propose Rancher OS, qui est une distribution Linux minimaliste qui fait tourner l'OS entier.

Et il y a Rancher UI, une interface qui permet de gérer graphiquement toute notre infrastructure Docker sur nos serveurs.

Les deux projets sont indépendants dans le sens où vous pouvez utiliser Rancher OS sans utiliser Rancher UI et vice versa.

Rancher

Installation

Il faut travailler avec deux serveurs

Un pour héberger Rancher et un autre pour les containers...

Récupérer la dernière version stable

```
docker pull rancher/server
```

Lancer Rancher

```
docker run -d --restart=always -p 8080:8080 rancher/server
```

Rancher

Installation

Idem en mode sécurisé

Lancer Rancher

```
docker run -d --restart=always -p 80:80 -p 443:443 rancher/server
```

--restart=always : redémarre si le container est arrêté

Outils

Docker Desktop

DockStation

Visual Studio Code

...

Divers

Liens...

<https://fr.jeffprod.com/blog/2015/lamp-sous-docker/>

<https://hub.docker.com/>

<https://rancher.com/docs/>

<https://www.guillaume-leduc.fr/docker-comme-solution-de-virtualisation-les-volumes.html>

<https://www.it-wars.com/posts/virtualisation/docker-swarm-par-exemple/>

<https://journal dun admin linux.fr/tutoriel-installer-et-utiliser-dockerswarm/>

<https://www.supinfo.com/articles/single/3037-comment-configurer-docker-swarm>

Docker

Des questions ?