

Serge NOEL
serge.noel@easylinux.fr

© 2015-2020 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.

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.
- 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.

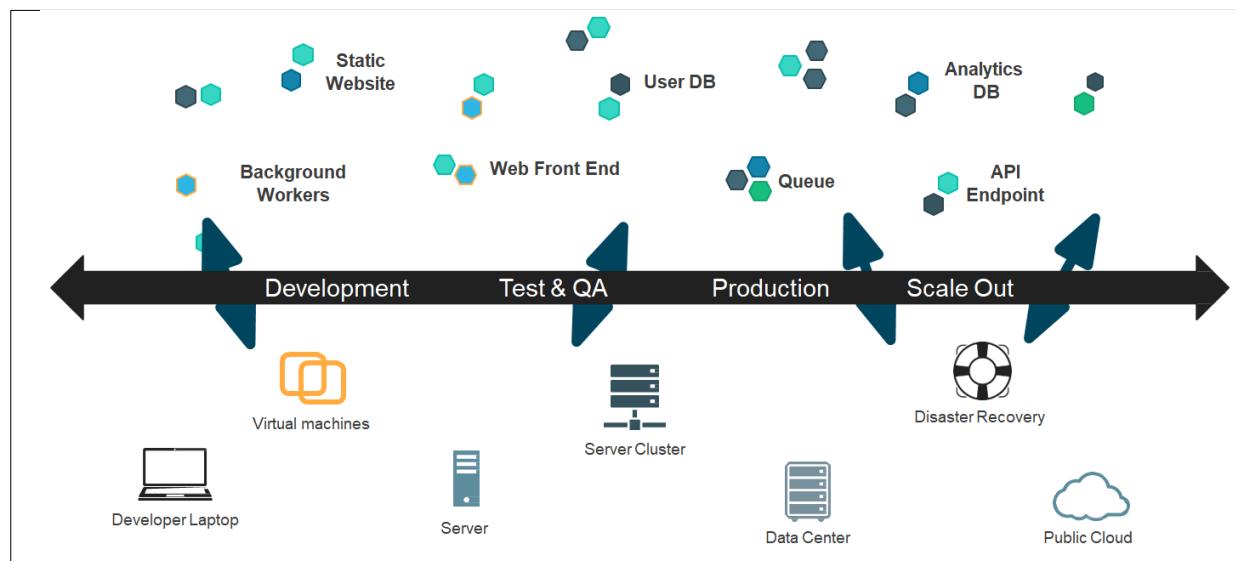
De la virtualisation à Docker

L'architecture des applications change rapidement



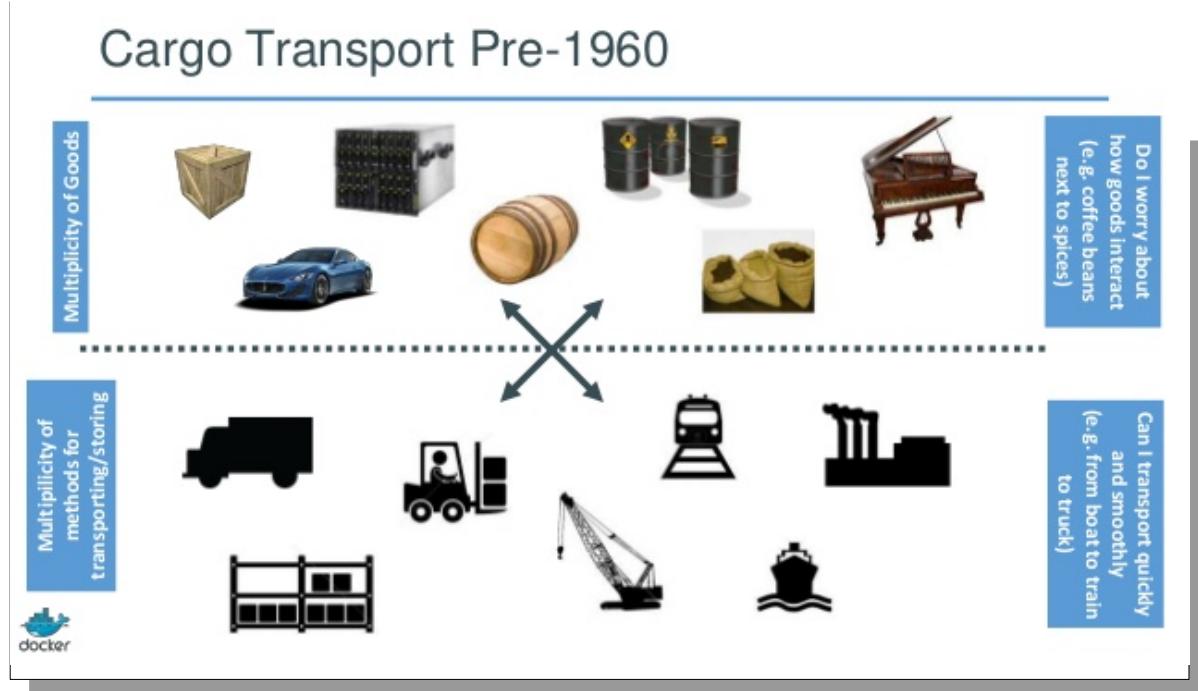
De la virtualisation à Docker

Environnements et technologies ultra hétérogènes



De la virtualisation à Docker

Analogie – origine du nom



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

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 plate-formes
- 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

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

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
- Crée 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

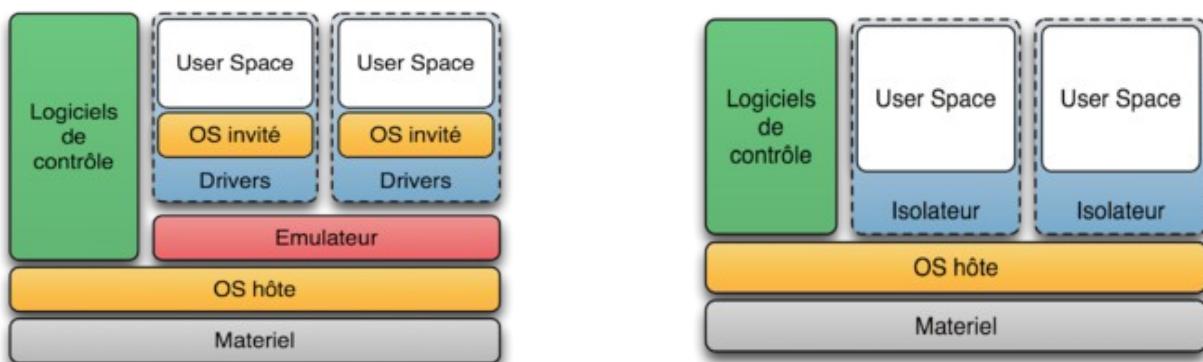
L'évolution de DotCloud à Docker

- 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

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.

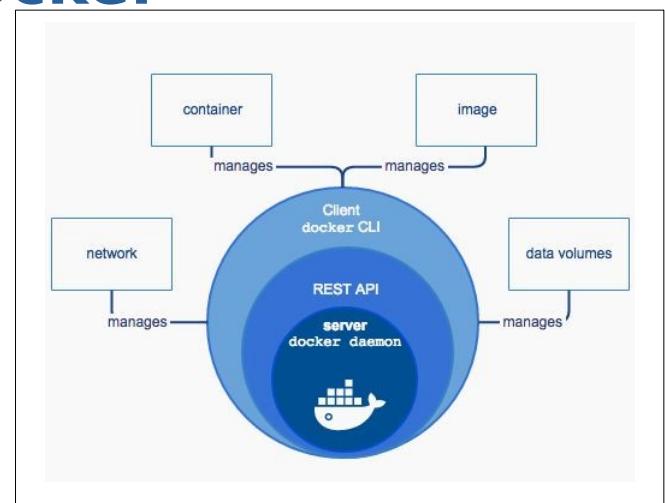
Travaux pratiques

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)

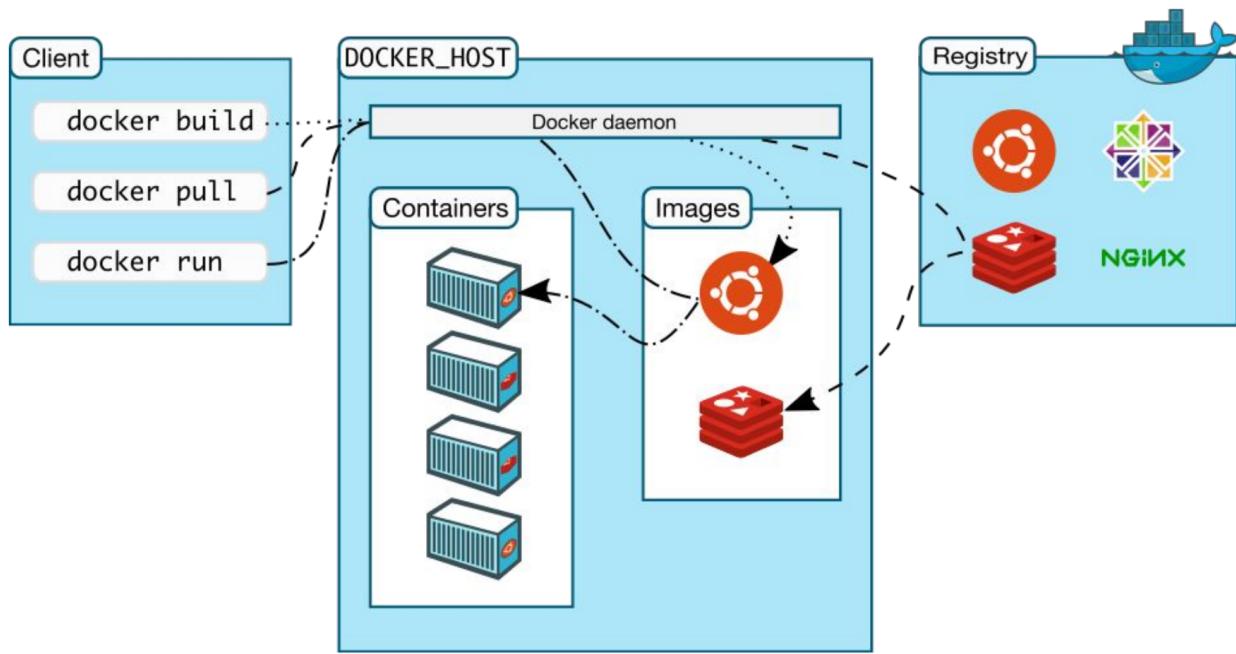


NB : beaucoup d'autres applications de l'écosystème docker utilisent l'API REST pour contrôler le daemon dockerd.

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



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

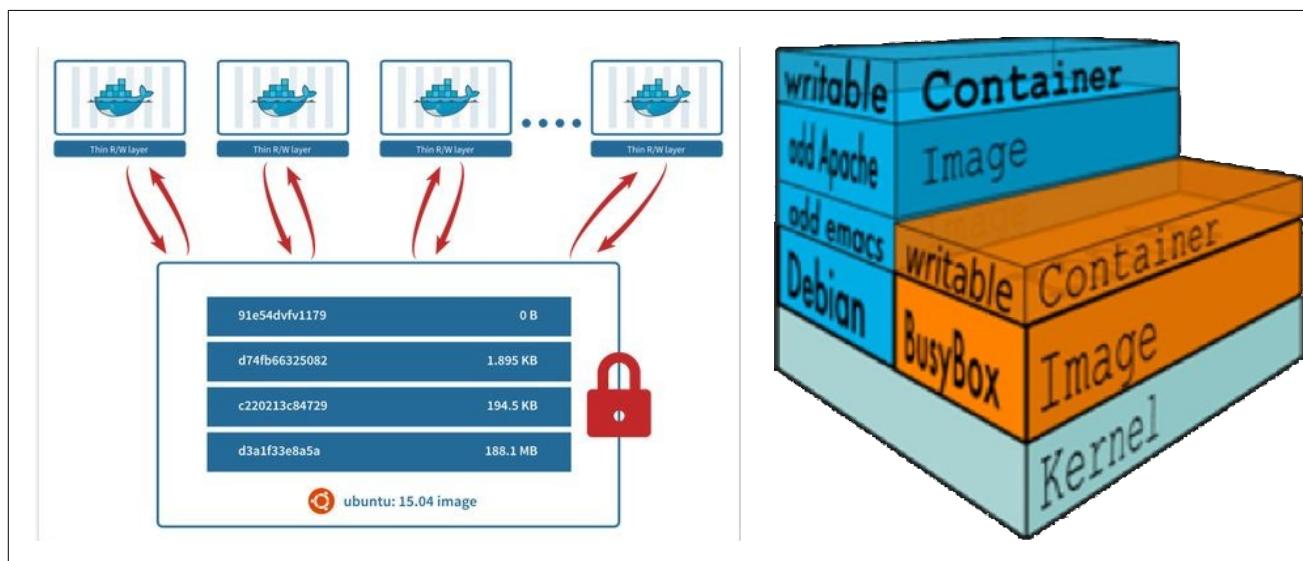
“**Container**” (ou Conteneur en français) est juste un terme générique qui désigne sous ce nom unique un ensemble de technologies matures intégré dans le noyau linux depuis une dizaine d'années.

PS : une version Microsoft Windows des containers est présente dans les versions de windows serveur 2016 et serveur 2019.

Présentation de Docker

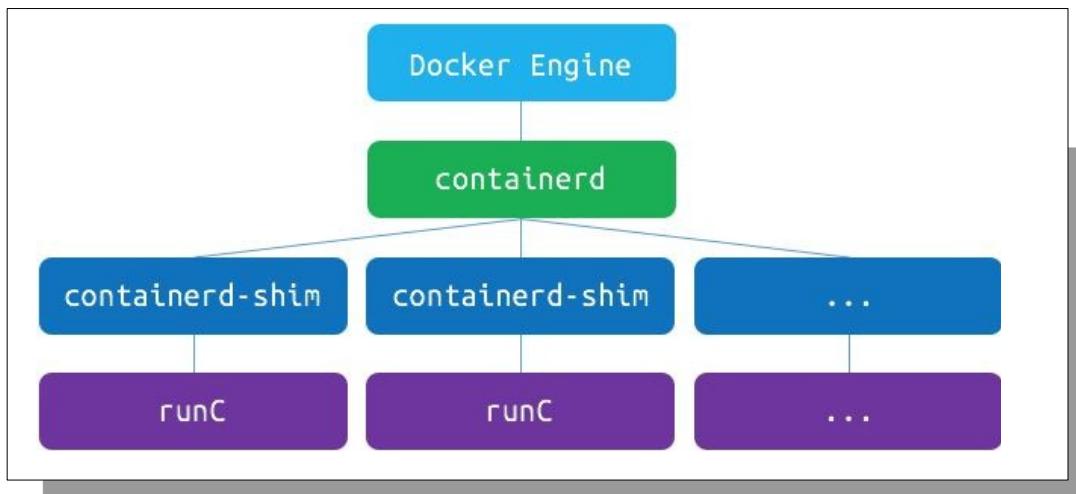
Architecture de Docker

Union file systems



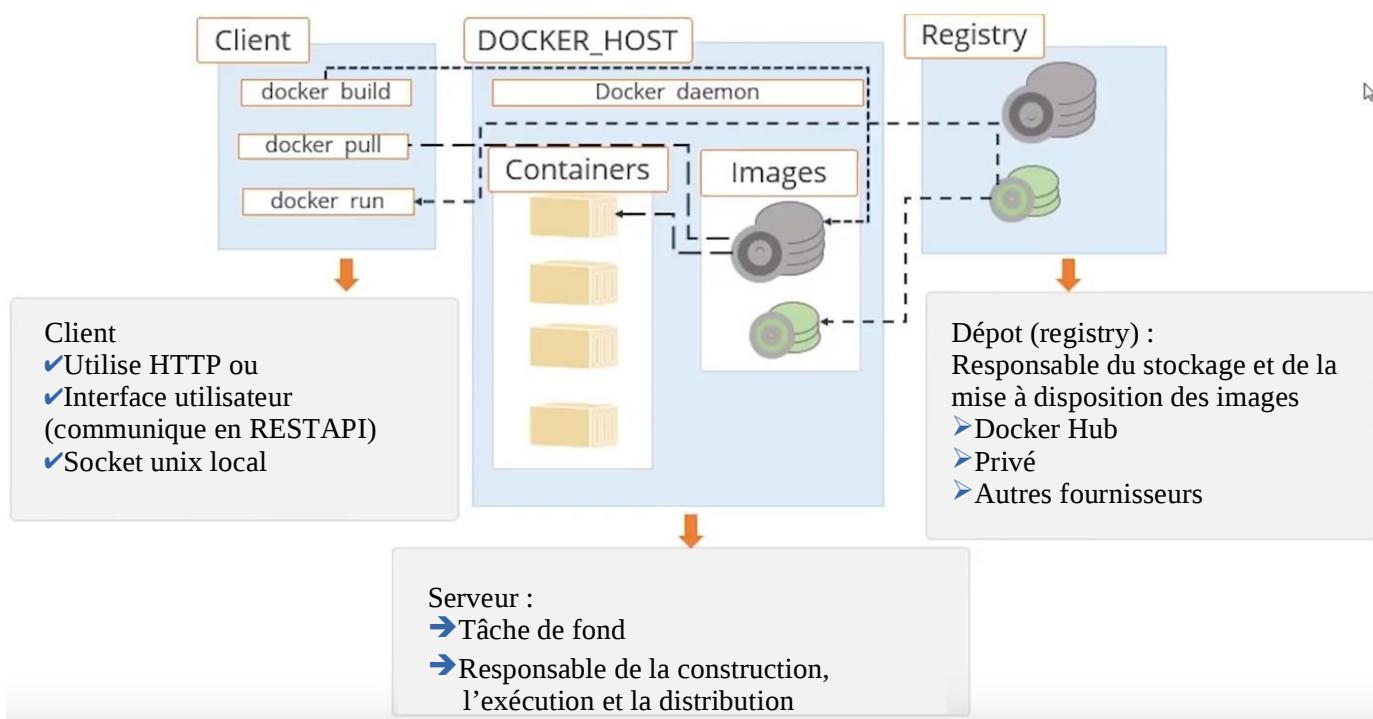
Présentation de Docker

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 / Windows 2019)

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

Pour les autres versions de Windows :

Installer VirtualBox et Debian



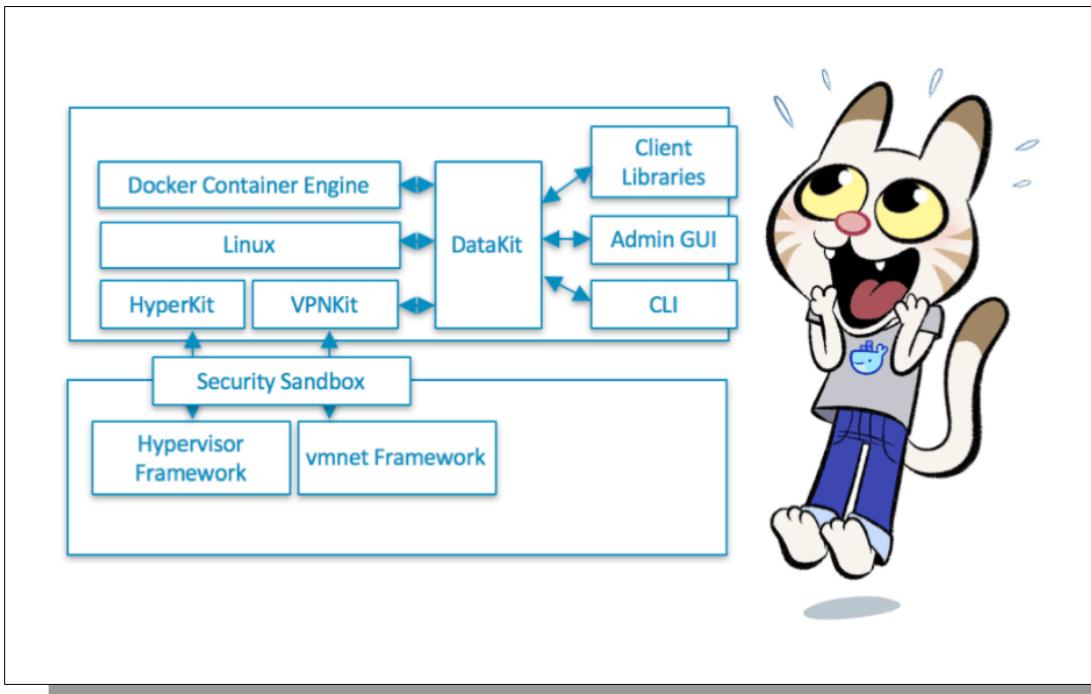
VirtualBox



Présentation de Docker

Disponibilité et installation de Docker

Version Mac



Présentation de Docker

Disponibilité et installation de Docker

Et bien sur sous Linux

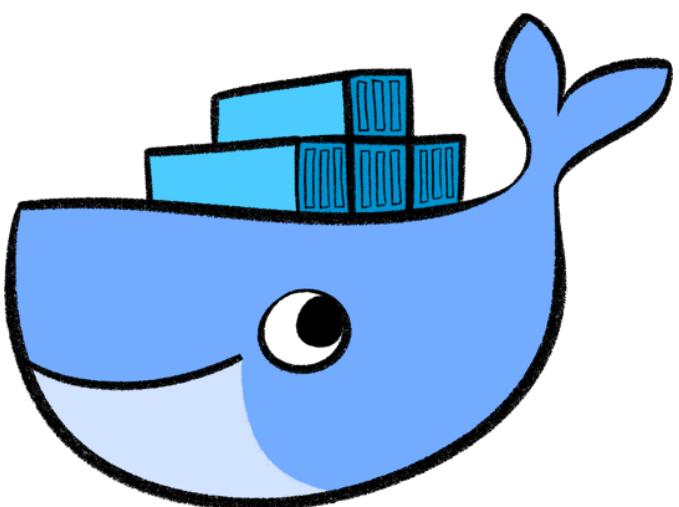
 → Debian

 → 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 Linux Debian

1.Installer virtualBox

Créer une machine en mode **bridge**

2.Installer Debian 9

3.Installer Docker

4.Tester

Présentation de Docker

Travaux pratiques

Exercice 1 :

➤ Mise en oeuvre de la station Windows

Exercice 2 :

➤ Import serveur Debian

Exercice 3 :

➤ 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:de3eac83cd481c04c5d6c7344cd7327625a1d8b2540e82a8231b5675cef0ae5f
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 debian:latest ?

Mise en œuvre en ligne de commande

Le Docker hub : ressources centralisées

Le Docker Hub

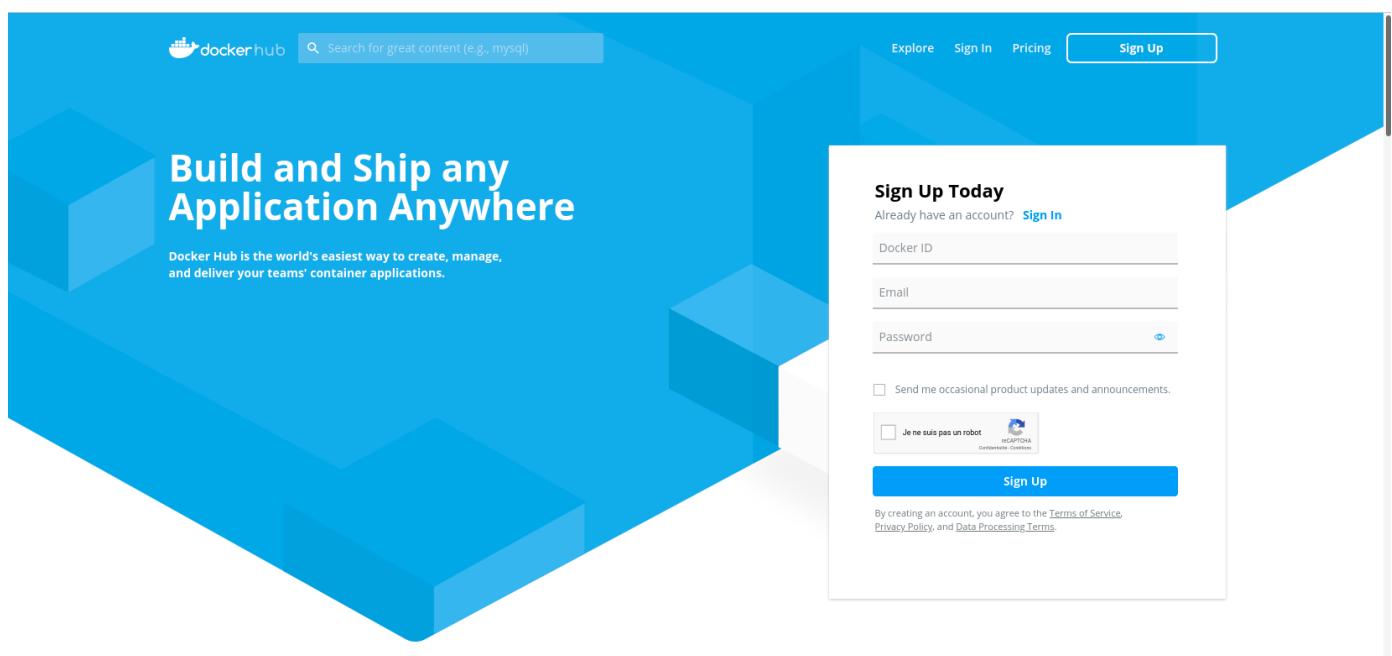
Hébergement de l'image dans une registry publique ou privée

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

hub.docker.com

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 interface for the `alpine/git` repository. At the top, there's a search bar with the text "alpine". The main content area displays the repository details for `alpine/git`, which is a simple git container running in Alpine Linux. It includes a blue cube icon, the repository name, the owner (`alpine`), the last update time (20 days ago), a brief description, and a "Container" button. Below this, there are tabs for "Overview" (which is selected) and "Tags". To the right, there are two boxes: one for the "Docker Pull Command" containing the command `docker pull alpine/git` and another for the "Owner" which is `alpine`.

Ligne de commande

Les volumes

➤ Volume nommé

```
docker container run -v <nom>:<chemin>...
```

➤ Volume bind

```
docker container run -v <chemin>:<chemin> ...
```

➤ Volume tmpfs

```
docker container run --mount type=tmpfs,  
destination=<chemin>
```

Ligne de commande

Gérer les volumes nommés

➤ Créer

```
docker volume create <nom>
```

➤ Lister

```
docker volume ls
```

➤ Information

```
docker volume inspect <nom>
```

➤ Supprimer

```
docker volume rm <nom>
```

Mise en œuvre en ligne de commande

➤ Virtual ethernet device (veth)

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

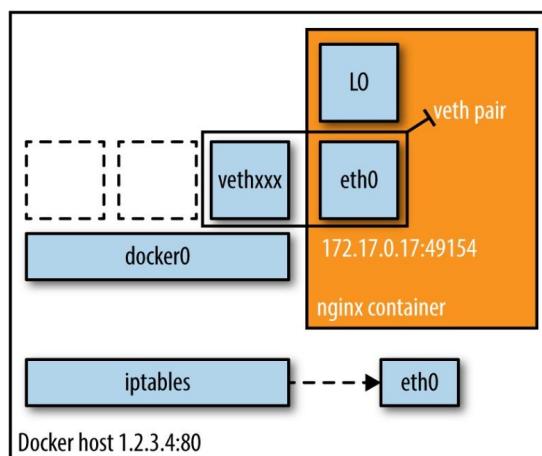
➤ Iptables

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

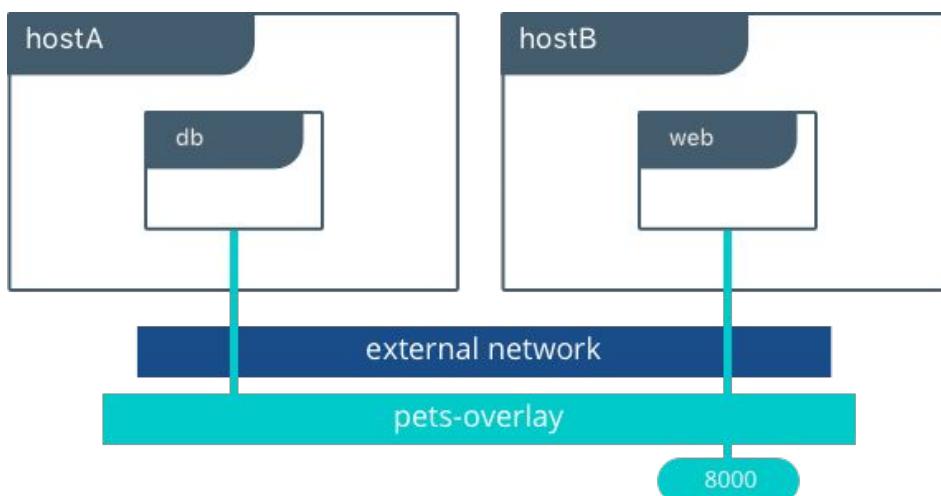
Utiliser les réseaux :

- Créer
`docker network create <nom>`
- Lister
`docker network ls`
- Détail
`docker network inspect <nom>`
- Supprimer
`docker network rm <nom>`

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 container run -p <port hôte>:<port conteneur> ...
```

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 container run -it alpine /bin/sh
```

Mise en œuvre en ligne de commande

`docker container 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

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:3.11
```

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 (**author**)

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 persistent 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 commandes 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 ?

Multi From

Il est possible d'utiliser plusieurs **FROM** dans un Dockerfile.

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

Automatiser la création d'une image

Construire une image depuis un Dockerfile:

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

Exemple

```
docker image build -t 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 programme 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.

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

```
Version: '3'
Services:
  redis:
    image: redis
    command: redis-server --appendonly yes
    volumes:
      - /root/rss2twitter/data:/data
    ports:
      - "6379:6379"

  rss2twitter:
    build: app
    depends_on:
      - 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 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 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

Installation :

```
docker volume create portainer_data
docker container 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 shows the Portainer.io dashboard for a Docker host named "Asus-Serge". The top navigation bar includes "Dashboard", "Home", "Node info", "admin", "my account", and "log out". The left sidebar has sections for LOCAL (Dashboard, App Templates, Containers, Images, Networks, Volumes, Events, Engine) and PORTAINER SETTINGS (User management, Endpoints, Registries, Settings). The main content area displays "Node info" with details: Name (Asus-Serge), Docker version (17.12.1-ce), CPU (8), and Memory (16.7 GB). Below this are four summary cards: "Containers" (5 total, 2 running, 3 stopped), "Images" (15 total, 1.5 GB), "Volumes" (1 total), and "Networks" (4 total).

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

Utilisation

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

Interface d'administration

Exercice 14 :

- Installer un hub privé

Exercice 15 :

- Utiliser un hub privé

Administre 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
Administre les conteneurs.

Administre 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

Administre 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

Administre 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

Administre des conteneurs en production

Gérer les logs des conteneurs

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

```
docker container logs <container>
```

Administre 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 l'orchestrateur Swarm.
2. Déploiement d'applications.

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 le fichier .yml

deploy:

```
replicas: 6
update_config:
    parallelism: 2
    delay: 10s
restart_policy:
    condition: on-failure
```

Des questions ?

