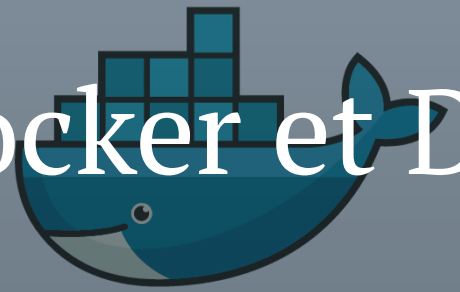
Images Docker et Dockerfiles



Plan de cette partie

Objectifs :

- Théorie
  - Comprendre le concept d'image Docker
  - Architecture de Docker
  - Manipulation des images
  - Création et partage d'images
- Pratique
  - Création d'images personnalisées
  - Gestion de la bibliothèque d'images

Table des matières :

Exemple d’utilisation

Pour un administrateur système

- Un développeur fournit une application qui affiche le contenu de la requête HTTP sous forme d'une image docker (`containous/whoami`)
- Pour la mettre en production :

```
docker run -p 8080:80 containous/whoami
```

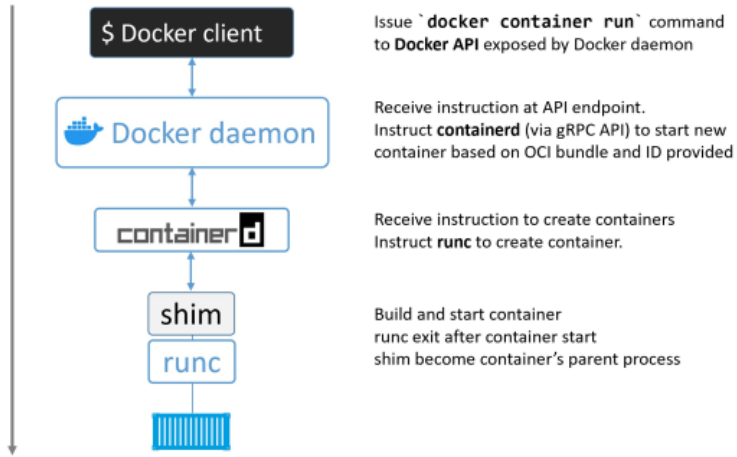
Configuration avancée

- Configuration derrière un virtualhost `http://whoami.mycompany.com/`
- Ajout d'une authentification OAuth2 (via Google)
- Prérequis : DNS correctement paramétré (ligne dans `/etc/hosts`)

Architecture de Docker

Vue d’ensemble

- Un démon avec communication HTTP via un socket unix (ou autre si configuré)



API Docker

Exemples d’appels API :

```
# Liste des images
curl --unix-socket /var/run/docker.sock http://localhost/images/json

# Surveillance des événements
curl --unix-socket /var/run/docker.sock http://localhost/events

# Création et démarrage d'un conteneur
curl -XPOST --unix-socket /var/run/docker.sock -d '{"Image":"nginx"}' \
     -H 'Content-Type: application/json' http://localhost/containers/create
curl -XPOST --unix-socket /var/run/docker.sock http://localhost/containers/IDICI/start
```

Documentation : Docker Engine API v1.41

Images : concept, création et manipulation

Concept d’image

- Une image est à un conteneur ce qu’un exécutable est à un processus
- Au lancement d’un conteneur on "instancie une image" :
  - Création des namespaces
  - Montage des fichiers en read-only
  - Ajout d’une couche read-write (thin layer)
  - Attribution des limitations
  - Lancement du processus

Note : La taille de la "thin layer" est affichée par `docker ps -s`

OCI - Open Container Initiative

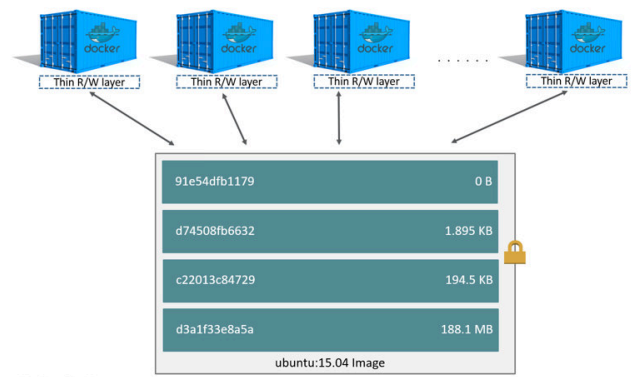
- Format standard et ouvert de description d'image
- Documentation relativement compliquée, mais en gros, une image est :
  - Des données (système de fichiers)
  - Des méta-données (processus, auteur, tags, etc.)

Pour voir les méta-données :

```
docker image inspect ubuntu
```

Système de couches (Layers)

- Remarquez que les données sont stockées en "couches" (Layers).
- Ce sont les différents éléments qui ont étés téléchargés en parallèle lorsque vous avez fait docker run.
- C’est *le truc malin* qui va permettre de rendre la manipulation d’images très rapide.



Gestion des images

- Téléchargement automatique avec docker run / create /
- Plusieurs conteneur peuvent utiliser la même image.
- Commandes de gestion :

```
# Lister les images
docker image ls
docker images

# Supprimer une image
docker image rm

# Télécharger une image
docker image pull
```

Création d'images

- Utilisation d'un Dockerfile
- Construction avec `docker build .`
- Chaque instruction crée une nouvelle couche
- **Attention** : Le contexte de build inclut tout le répertoire courant

Exercices pratiques

À vous !

Annexe : OCI - Aller plus loin

Une image est un empilement de couches ('Layers')

- Manifest : https://docs.docker.com/registry/spec/manifest-v2-2/
  - Un système de fichiers, décrit par des couches
- un système de fichier

```
docker pull redis
docker create --name tempredis redis
docker export tempredis | tar -C redis/rootfs -xf -
docker rm tempredis
```

- un fichier de configuration

```
runc spec
```

- On peut lancer notre conteneur :

```
runc run nombidon
```

API docker registry

id=$(curl --silent "https://auth.docker.io/token?scope=repository:library/ubuntu:pull&service=registry.docker.io" | jq -r '.token') curl --silent --header "Accept: application/vnd.docker.distribution.manifest.v2+json" --header "Authorization: Bearer $id" "https://registry-1.docker.io/v2/ubuntu/manifests/latest"