TIW - Cloud computing

Docker

Fabien Rico (fabien.rico@univ-lyon1.fr)
Jean Patrick Gelas

Univ. Claude Bernard Lyon 1

séance 2

Qu'est-ce que c'est

C'est :

- un moyen de faire fonctionner plusieurs processus dans un environnement isolé et/ou limité ;
- un moyen d'installer et de configurer des logiciels dans cet environnement ;
- un moyen de gérer différentes images et de les distribuer.

Ce n'est pas :

- de la virtualisation au sens général : moins de souplesse ;
- une simple exécution de programme : séparation du système hôte.

1 Introduction

2 Fonctionnement
- Séparation
- Contrôle des ressources
- Système de fichiers
- Résultats

3 Les commandes
- Images
- Gestions des conteneurs

Virtualisation niveau système

C'est le fait, pour un système d'exploitation, de pouvoir exécuter des processus dans un environnement isolé (cf wikipedia).

Rappel

- mémoire virtuelle ;
- isolation des processus ;
- mode utilisateur/mode noyau ;
- appels systèmes.

Pour séparer ou limiter les processus, les appels système peuvent :

- gérer des espaces de nom ;
- refuser certaines opérations ;
- appliquer des quotas.

Exemple historique : chroot

Définition (chroot)

`chroot` (*Change Root*) est une commande qui permet de changer le répertoire racine du système pour un processus et tous ses processus fils.

Normalement

$ ls /

1. le programme demande à ouvrir /
2. le programme obtient un descripteur de /
3. le programme liste /

Le résultat est :

bin boot dev etc home lib lib64 media ...

Dans un chroot

$ chroot /tmp/plop/ ls /

1. le programme demande à ouvrir /
2. le programme obtient un descripteur de /tmp/toto/
3. le programme liste /tmp/toto

Le résultat est :

bin lib lib64

Description d'un conteneur

Un conteneur docker peut être vu comme une extension du `chroot`. Il est composé :

- d'une description ;
- d'un ou plusieurs répertoire(s) de l'hôte qui forment son système de fichiers ;
- d'interfaces réseau virtuelles ;
- éventuellement des processus en train de s'exécuter dans l'environnement.

S'il n'y a pas de processus, le conteneur est éteint, mais peut être relancé.
S'il est en fonctionnement on peut lui attacher un nouveau processus.
On peut sauvegarder un conteneur sous la forme d'une *image* qui permettra de créer d'autre conteneur.

Espace de nom

`chroot` est une commande unix qui permet de sécuriser certaines applications car le système ajoute automatiquement un préfix à tous les fichiers manipulés. Cela apporte :

- une limitation des actions ;
- un cloisonnement des applications.

On peut généraliser cela à d'autres objets du système (utilisateur, processus, ...). C'est ce qu'on fait via les *Espace de noms*

Définition (Espace de nom)

Un espace de nom est un préfixe que l'on ajoute aux noms de certains objets du système. Il permet de séparer des processus qui ont un espace de nom différent.

Si un processus demande à accéder à un objet, l'espace de nom est automatiquement ajouté à la requête. Par exemple, si le processus est dans l'espace de nom `toto`
`open("truc")` devient `open ("toto::truc")`
Il ne peux accéder qu'aux objets du même espace.

Espace de nom (suite)

Sous linux, les *espaces de noms* permettent une séparation :

- `mnt` des points de montage (des disques) ;
- `pid` des numéros de processus ;
- `net` des outils de communication réseau ;
- `ipc` des outils de communication inter processus ;
- `user` des utilisateurs ;
- `uts` du nom de la machine et de son domaine ;
- `cgroups` des groupes de contrôle de processus.

Contrôle des ressources : les *cgroups*

Définition (cgroups)

Les *control groups (cgroups)* sont une fonctionnalité du noyau linux qui permet de rassembler des processus dans un groupe et de :

- limiter leur accès aux ressources ;
- comptabiliser l'utilisation de ces ressources ;
- appliquer des priorités ;
- geler, créer des points de sauvegarde ou restaurer des groupes.

Cela permet d'appliquer à un groupe de processus ce que l'ont fait déjà sur une machine virtuelle classique.

Gestion du stockage : le problème

Le système de fichiers du conteneur est séparé de l'hôte. *Il est vide !*

- Pour fonctionner, un logiciel a besoin
  - de dialoguer avec le noyau du système ;
  - d'utiliser des logiciels standard ;
  - d'utiliser des bibliothèques de fonctions.
- Seul le noyau est déjà présent dans un docker : *il faut installer le reste*.

Par exemple, un conteneur basé sur ubuntu 12.01 avec le serveur apache doit avoir dans son système de fichiers :

- les outils de base : `apt`, `bash`, `vi` ...
- les configurations minimales : `/etc/`
- les bibliothèques essentielles : `ld`, `libc`, ...
- le serveur apache.

Si un serveur contient 30 conteneurs basés sur le même modèle, combient de fois faut-il chaque outil ?

Gestion du stockage : *Copy on Write*

On peut « *empiler* » les conteneurs.

Définition (COW)

Le *Copy-on-Write* est la capacité de maintenir 2 copies d'un ensemble de données

- en gardant une seule copie de ce qui est commun ;
- en dupliquant uniquement ce qui est modifié.

Grâce au COW on peut :

- mutualiser les systèmes de fichiers basés sur le même modèle ;
- créer rapidement des conteneurs qui spécialisent un conteneur existant.

Que permet docker ?

- Créer des images avec un mini système d'exploitation
  $\Rightarrow$ un disque de machine virtuel ?
- Créer un conteneur basé sur ce disque auquel on attribue des ressources
  $\Rightarrow$ une instance de VM ?
- Allumer, éteindre, sauvegarder le conteneur
  $\Rightarrow$ même opération que sur les VMs ?

Le comportement est très proche de celui des hyperviseurs. Mais c'est de la virtualisation au niveau du système c'est à dire que ce n'est que de la gestion de processus différents dans un seul système d'exploitation.

Différence avec la virtualisation

Comme ce ne sont que des processus dans un seul système :

- le partage de ressources est facilité ;
- il y a moins de surcout que la virtualisation ;
- il n'y pas de gestion du matériel ;
- il y a moins de sécurité.

A quoi cela sert ?

Les éléments

- images : le template, les données de l'application prêtes à l'emploi ;
- conteneur : instance qui fonctionne ou qui peut être démarrée ;
- volumes : répertoire qu'on peut partager entre conteneur ou avec l'hôte ;
- dépot : le docker hub ou tout le monde peut déposer une image de docker à télécharger et tester.

Hub et images

`https://hub.docker.com/` contient les images proposées par les autres :

- beaucoup de choix ;
- un système de notation (les étoiles) ;
- possibilité de télécharger des versions (les *tags*).

Quelques commandes :

- `docker search` *`mot clef`* chercher une image ;
- `docker pull` *`nom:TAGdeVersion`* télécharger une image ;
- `docker images` lister les images présentent localement ;
- `docker rmi` effacer une image locale.

Récupération, dépot d'image

Les images sont sur un serveur *registry* par défaut le *Docker Hub*. On peut télécharger ou déposer des images sur ces serveurs avec les commandes `pull` et `push`

`docker pull [`*`registry/`*`]`*`repository`*`[:`*`tag`*`]`

- Le *registry* est le serveur qui est contacté (par défaut hub.docker.com :443).
- Le *repository* est la famille d'images que vous voulez obtenir (par exemple debian, ubuntu, nginx,...).
- Le *tag* est la version (par défaut *latest*).

Par exemple

```
$ docker pull debian:8.5
```

permet d'obtenir un docker basé sur la version 8.5 de la débian.

```
$ docker pull debian
```

permet d'obtenir un docker basé sur la dernière version de la débian.

Construction d'image

On peut recréer une image à partir d'un docker

`docker commit` *`NomDuConteneur NomImage:Tag`*

Cette image peut ensuite être

- utilisée pour relancer un conteneur
  `docker run --name` *`NomNouveau NomImage:Tag command`*
- sauvegardée sous la forme d'une archive
  `docker save -o` *`fichier.tar NomImage:Tag`*
- envoyée sur un serveur
  `docker push` *`NomImage:Tag`*

Dockerfile

On peut construire une nouvelle image à partir d'un fichier de description appelé `Dockerfile`.

`docker build -t` *`nom:tag RepertoireDuDockerfile`*

A partir du fichier de description, la commande :

- télécharge une image de base ;
- applique une série de commandes qui modifie l'image ;
- crée une image docker utilisable
  - avec une description de l'environnement
  - avec une commande à exécuter au lancement

La construction via `Dockerfile` utilise le COW pour construire les différentes images générées.

Exemple de Dockerfile

```
FROM ubuntu:vivid
MAINTAINER Fabien Rico <fabien.rico@univ-lyon1.fr>

RUN pt-get update \
    apt-get -y install  apache2 && apt-get clean

COPY serveur.conf /etc/apache2/sites-enabled/
    /etc/apache2/sites-enabled/000-default.conf

WORKDIR /var/www/html

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2

EXPOSE 80

CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

image de base

installation d'apache

ajout d'un fichier

répertoire de travail

variables d'environnement

port exposé

commande à exécuter

Création du conteneur

C'est au moment de la création que l'on peut donner la configuration du docker

`docker create --name` *nom* [*options*] *NomImage* [*commande*]

- `-p` *HostPort*`:`*DockerPort* : mapping de port entre l'hote et le conteneur
- `--link` *NomAutreDocker* : ajoute un lien avec un autre docker (c'est à dire fixe des variables d'environnement permettant de contacter le docker
- `-v` *VolumeHote*`:`*VolumeCont* crée un volume partagé entre l'hôte et le conteneur.
- `--name` *nom* nom du conteneur
- *NomImage* le nom de l'image à exécuter
- *commande* la commande à exécuter dans le docker (par défaut celle du Dockerfile)

Gestion du conteneur

- Lancer un conteneur `docker start` *`nomDuConteneur`*
  le conteneur s'exécute en lançant la commande prévue dans le *create* ou définie dans le Dockerfile.
- stopper un contenu `docker stop` *`nom`*
- Lister les conteneurs exécutés `docker ps`
- Lister les conteneurs existant `docker ps -a`
- exécuter une commande dans un docker existant
  `docker exec -it` *`nomDuConteneur commande`*
- Supprimer un conteneur `docker rm` *`nom`*
- Créer et lancer un conteneur
  docker run ... avec les même options que `docker create`.