Introduction à Docker

Badre Belabbess

Atos Integration

20 Février 2017

Sommaire

Définition

<+-> 

C'est une technologie de Container Linux !

Run everywhere... ou presque :)

- Sans adhérence à la version du noyau
- Sans adhérence à la **distribution Linux** sous-jacente

Si ça fonctionne sur le hôte, ça peut fonctionner dans un container

À travers les âges

- 18/01/2013 : 1er commit
- 01/02/2013 : 1ere démo en ligne
- 21/03/2013 : 1ere démo à Pycon US
- 26/03/2013 : Ouverture du dépôt GitHub
- 03/06/2013 : Version 0.4
- 25/06/2013 : Rejoint la fondation Linux
- 13/03/2014 : Version 0.9 (libcontainer remplace LXC)
- 02/04/2016 : Version 1.10

En quelques chiffres : Github, ce mois-ci

March 4, 2016 – April 4, 2016

Period: **1 month**

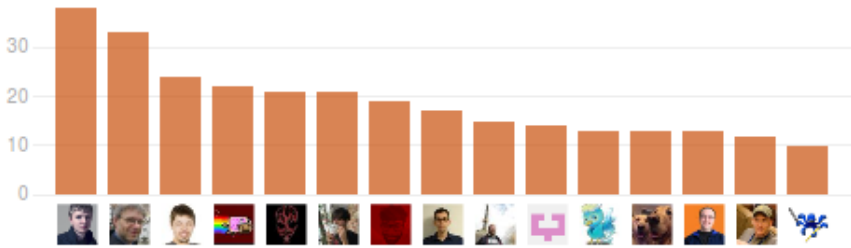
Overview

399 Active Pull Requests

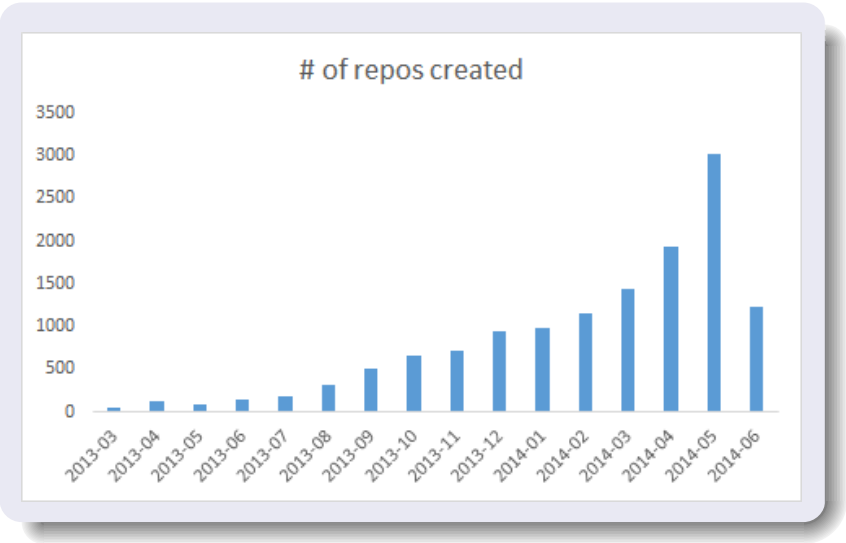
457 Active Issues

343 Merged Pull Requests	**56** Proposed Pull Requests	**290** Closed Issues	**167** New Issues

Excluding merges, **87 authors** have pushed **375 commits** to master and **493 commits** to all branches. On master, **965 files** have changed and there have been **32,932 additions** and **23,070 deletions**.



En quelques chiffres : Docker Hub (Juin 2014)

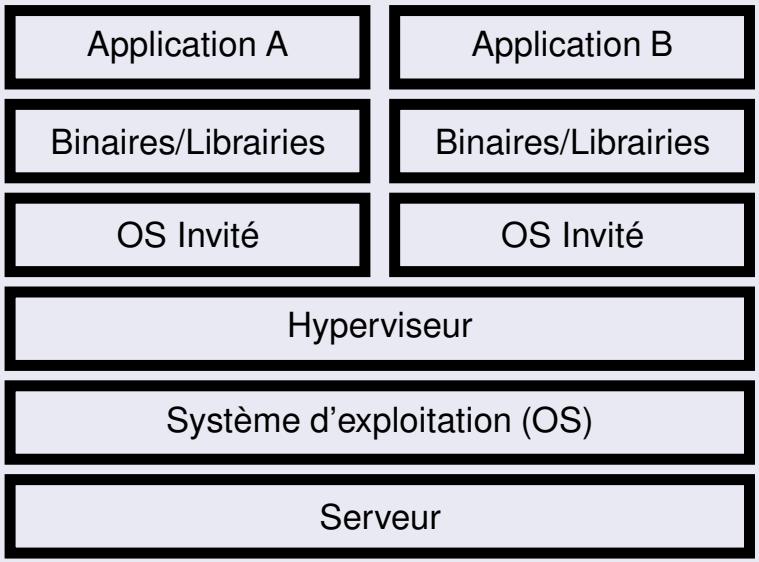


- 101477 images
- 13475 dépôts
- 5091 utilisateurs

Comparaison Machine Virtuelle / Container Docker

Virtualisation classique

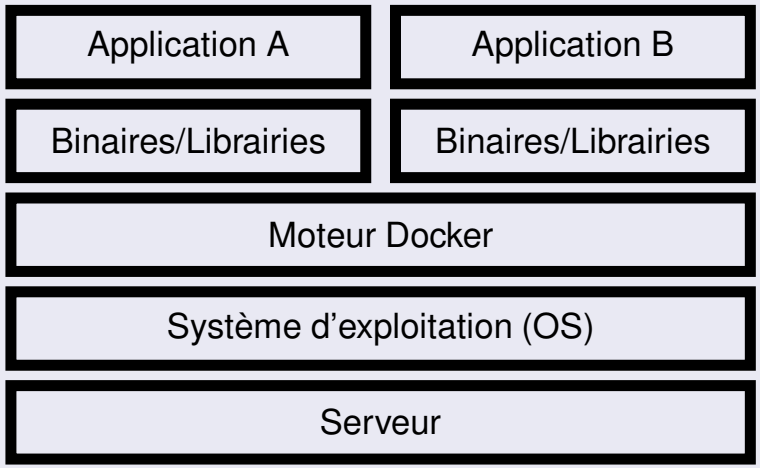
- OS complet par VM
- Ressources importants
- Process de boot complet



Machine Virtuelle

Virtualisation noyau

- N containers par noyau
- Tout virtualisé
- Démarrage d'un process



Container Docker

La virtualisation au niveau noyau

C'est une technologie aussi vieille que le monde !

- 1982 : chroot
- 1998 : FreeBSD Jail
- 2001 : Linux VServer, Virtuozzo
- 2005 : OpenVZ, Solaris Containers (Zones)
- 2007 : AIX WPARS, HP-UX Containers
- 2008 : Linux LXC
- 2013 : Docker

La virtualisation noyau par Docker

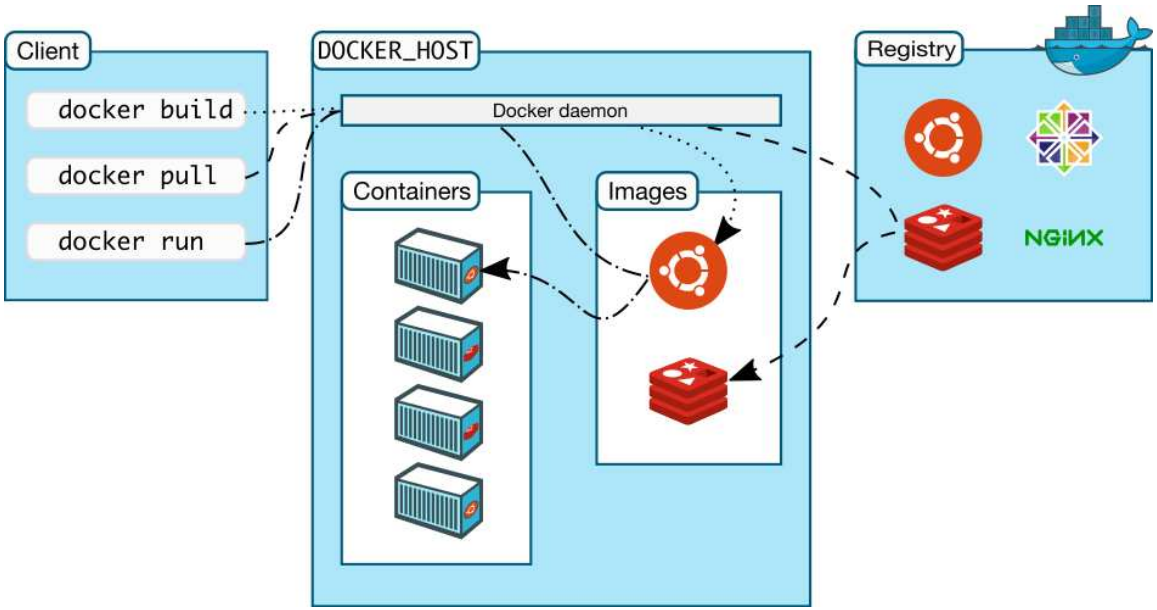
Docker apporte nativement :

- un environnement d'administration des containers
- un DSL [1] pour la spécification des «Dockerfiles»
- la gestion en version des containers
- un registry pour stocker les images
- une API REST

Docker est une solution permettant d'exécuter un ou plusieurs logiciels dans des environnements séparés (containers) pouvant communiquer entre eux.

1. Domain specific language

Architecture Docker



Docker s'appuie sur les technologies du Noyau pour fonctionner :

- les espaces de nommage
- les groupes de contrôle
- les capacités noyau
- le stockage

Namespace

Définition

Ce sont les espaces de nommages du noyau Linux qui assure aux containers Docker leur isolation.

Docker en utilise principalement 5 :

- `mnt` Gestion des points de montages
- `uts` Isolation du noyau et des identifieurs de version
- `ipc` Gestion des accès aux ressources inter-process
- `pid` Isolation des processus
- `net` Accès aux ressources réseau

Control Groups

- Gestion des tâches
  - Partitionnement en groupes hiérarchiques
  - Démarrage / Arrêt
- Contrôle des ressources systèmes
  - Limitation CPU / Mémoire
  - Limitation des I/O
  - Priorisation des tâches
  - Contrôle d'accès
- Gestion de tâches

Capabilities

Définition

C'est un des modules de sécurité du noyau Linux

Permet de limiter les privilèges de `root`

- `CAP_SETPCAP`
- `CAP_SYS_MODULE`
- `CAP_SYS_RAWIO`
- `CAP_SYS_PACCT`
- `CAP_SYS_NICE`
- `CAP_SYS_RESOURCE`
- `CAP_SYS_TIME`
- `CAP_SYS_TTY_CONFIG`
- `CAP_AUDIT_WRITE`
- `CAP_AUDIT_CONTROL`
- `CAP_MAC_OVERRIDE`
- `CAP_MAC_ADMIN`
- `CAP_SYSLOG`
- `CAP_NET_ADMIN`
- `CAP_SYS_ADMIN`

Backends de Stockage

aufs backend historique de Docker. Empilement de répertoires pour en faire une vue unifiée

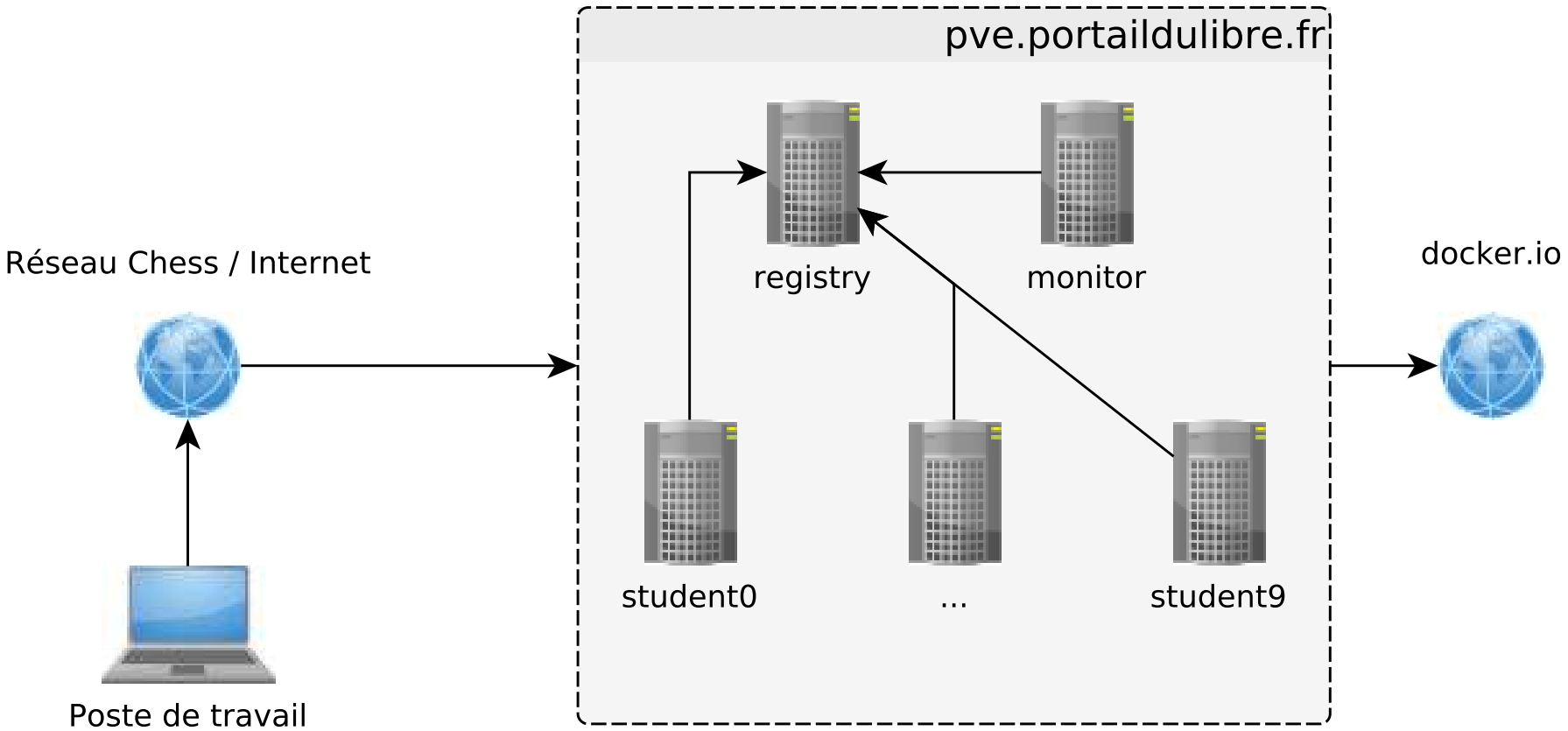
devicemapper stockage sur périphérique virtuel avec des fonctionnalités d'allocation fine, COW, les snapshots. . . A été développé par RedHat

overlay UFS moderne intégré au noyau (> 3.18). Design plus simple mais meilleurs performances par rapport à `aufs`

btrfs Système de fichier avec gestion native des fonctionnalités de `devicemapper`

Sommaire

L'infrastructure



Installation et configuration de Docker

Attention

Sur votre VM, c'est déjà fait !

Installation sur une CentOS à jour

```
# yum -y update
# yum -y install docker python-docker-scripts
```

Configuration du backend de stockage

```
# vim /etc/sysconfig/docker-storage-setup
# docker-storage-setup
```

Configuration du moteur Docker

```
# vim /etc/sysconfig/docker
```

Configuration de Docker derrière un proxy

```
# vim /etc/systemd/system/docker.service.d/http-proxy.conf
```

Paramétrage de vos VMs

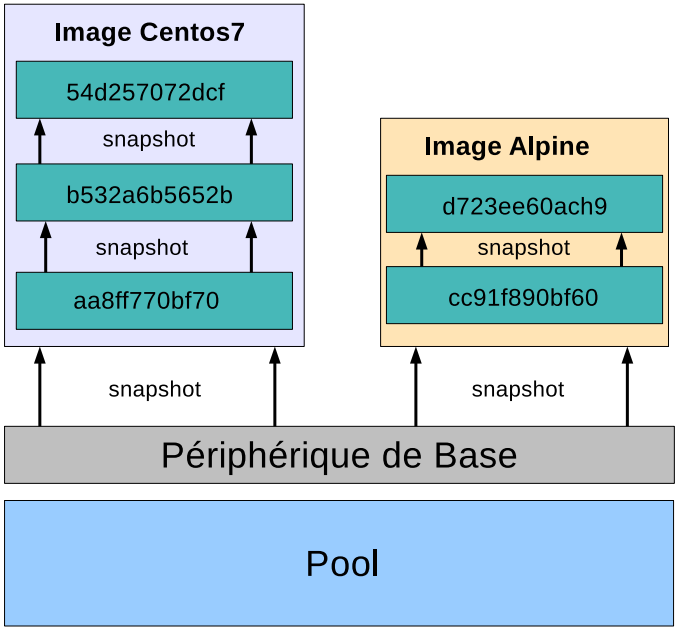
Le stockage

/etc/sysconfig/docker-storage-setup

```
STORAGE_DRIVER=devicemapper
DEVS=/dev/vdc
VG=storage
DATA_SIZE=60%FREE
MIN_DATA_SIZE=2G
CHUNK_SIZE=512K
GROWPART=false
AUTO_EXTEND_POOL=yes
POOL_AUTOEXTEND_THRESHOLD=60
POOL_AUTOEXTEND_PERCENT=20
```

Paramétrage de vos VMs

Le stockage `devicemapper`



Paramétrage de vos VMs

Les paramètres du moteur

/etc/sysconfig/docker

```
OPTIONS='--selinux-enabled --tlsverify \
         --tlscacert=/etc/docker/certs.d/CA.crt \
         --tlscert=/etc/docker/certs.d/formation.docker.crt \
         --tlskey=/etc/docker/certs.d/formation.docker.key \
         -H tcp://0.0.0.0:2376 -H unix:///var/run/docker.sock'

DOCKER_CERT_PATH=/etc/docker

ADD_REGISTRY='--add-registry registry.formation.docker'
```

Paramétrage de vos VMs

Le proxy

/etc/systemd/system/docker.service.d/http-proxy.conf

```
[Service]
Environment="HTTP_PROXY=http://proxy-fr.glb.my-it-solutions.net:84/" \
            "NO_PROXY=localhost,127.0.0.1,registry.myatos.net"
```

Attention !

- Le proxy n'est pas nécessaire sur votre VM
- Le répertoire docker.service.d doit être créé
- Je n'ai pas testé cette configuration proxy :)

Démarrage du service

Docker est contrôlé par Systemd

Activation au boot

```
# systemctl enable docker.service
```

Démarrage du service

```
# systemctl start docker.service
```

Vérification du service

```
# systemctl status docker.service
```

```
docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since mar. 2016-04-05 17:05:12 CEST; 13h ago
     Docs: http://docs.docker.com
 Main PID: 2182 (sh)
   Memory: 20.7M
   CGroup: /system.slice/docker.service
           |-2182 /bin/sh -c /usr/bin/docker daemon $OPTIONS            $DOCKER_STO...
           |-2183 /usr/bin/docker daemon --selinux-enabled --tlsverify --tlscacert=...
           |-2184 /usr/bin/forward-journald -tag docker
```

Commandes de bases

Une commande pour les gouverner tous !

```
docker [COMMAND] help
```

Commandes de bases

Manipuler les images

Lister

```
docker images [OPTIONS] [REPOSITORY[:TAG]]
```

Construire

```
docker build [OPTIONS] PATH | URL | -
```

Supprimer

```
docker rmi [OPTIONS] IMAGE [IMAGE...]
```

Commandes de bases

Manipuler les images

Lister

```
docker ps [OPTIONS]
```

Démarrer à partir d'une image

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Supprimer

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

Construire une image

Le «Dockerfile»

Permet de décrire la fabrication de l'image

```
FROM alpine:latest

ENTRYPOINT sh
```

Utilise un DSL permettant de décrire le contenu de l'image

- FROM
- MAINTAINER
- CMD
- LABEL
- EXPOSE
- ENV
- RUN
- ADD
- COPY
- ENTRYPOINT
- VOLUME
- USER
- ARG
- ONBUILD
- STOPSIGNAL

Construire une image

L'image de base

Définition

Une image qui n'a pas de Parent est une **image de base**. En général, une image de base ne contient que les librairies de base (`glibc`,...) d'une distribution Linux.

Sa construction est outillée :

Scripts Docker `https://github.com/docker/docker/tree/master/contrib`

Supermin5 `http://people.redhat.com/~rjones/supermin/` (pour Redhat/CentOS)

Debootstrap `https://wiki.debian.org/fr/Debootstrap` (Pour Debian/Ubuntu)

Lab 1 : Construire une image de base avec supermin5

- Écrire un Dockerfile
- Construire son image
- Démarrer un container basé sur cette image

Avant d'aller plus loin...

Quelques commandes utiles pour :

Démarrer un container en mode test

```
docker run -rm -it [IMAGE] [COMMANDE]
```

Nettoyer les containers

Tous `docker rm $(docker ps -aq)`

À l'arrêt `docker rm $(docker ps -aq -f status=exited)`

Exemples de «Dockerisation»

cowsay

```
$ fortune informatique | cowsay
 _______________________________________
/ Les logiciels, c'est comme le sexe ; \
| C'est meilleur quand c'est libre.     |
\ -+- Linus Torvalds -+-                /
 ---------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

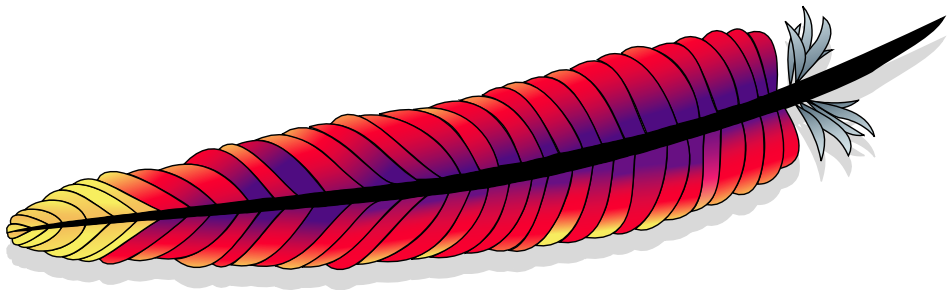
Lab 2 : Le container `cowsay` en 3 étapes

1. Une image qui contient `cowsay`
2. Une vache perroquet (répète ce qu'on dit)
3. Une vache qui parle toute seule :) [a]

a. Hint : On utilisera `fortune`

Exemples de «Dockerisation»

Apache `httpd`



Lab 3 : A vous de jouer ! ! !

Rappels et astuces :

- Un container s'arrête dès que l'`ENTRYPOINT` termine son exécution
- S'inspirer du lancement du serveur `httpd` au premier plan

Pour les plus téméraires : partir de l'image `alpine:monitor`

- La distribution `alpine` dispose du gestionnaire de paquets apk
- Observer l'empreinte disque des deux images

Sommaire

API Docker

Docker expose une API `HTTP Rest` qui permet :

- de lister les images

```
$ echo -e "GET /images/json HTTP/1.0\r\n" | \
nc -U /var/run/docker.sock
$ ./curl.sh https://$HOSTNAME:2376/images/json | jq
```

- rechercher une image dans les dépôts d'images

```
$ ./curl.sh -XGET https://$HOSTNAME:2376/images/search?term=alpine \
| jq '.[] | .name'
```

- d'obtenir des images depuis les dépôts d'images

```
$ ./curl.sh -xPOST \
https://$HOSTNAME:2376/images/create?fromImage=alpine\&tag=3.3
```

- de lister les containers

```
$ ./curl.sh -XGET https://$HOSTNAME:2376/containers/json
```