Introduction à Docker

Dans ce TP nous allons voir l'installation de Docker et les premiers exemples d'utilisations de votre nouvel environnement.

Installation de Docker sous Windows

Pour installer Docker rien de plus simple, il suffit d'utiliser l'installeur officiel en le téléchargent sur le site de Docker.

Lancer Docker

Lancer Docker en tant qu'administrateur (et autoriser Hyper-V si celui-ci vous le demande).

Premier test

Maintenant qu'il est installé sur votre poste, rien de plus simple. Dans une console, entrez la commande suivante :

```
docker version
```

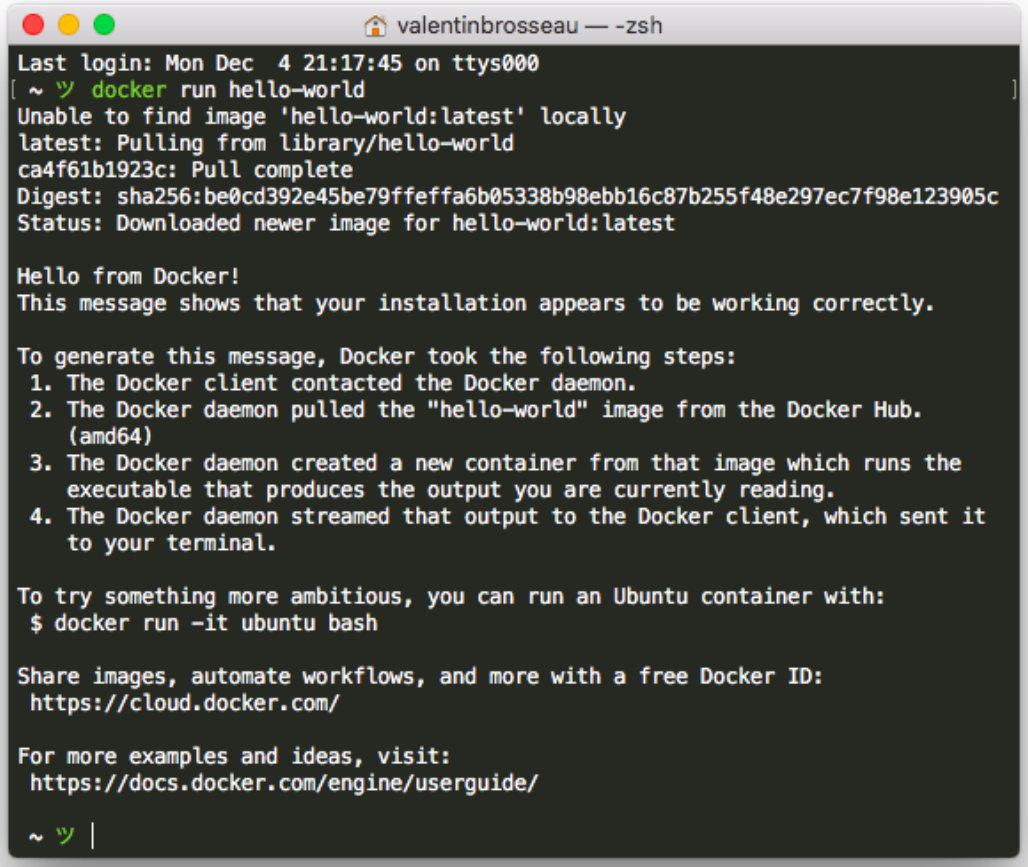
Vous devez voir la version de Docker.

- Que constatez-vous ?
- Pourquoi est-ce important d'utiliser une version récente ?

Second test

Maintenant que nous savons que Docker est correctement installé, testons avec une « machine » fournie par Docker. L'image se nomme `hello-world`, celle-ci ne fait qu'afficher un message de bienvenue. De nouveau, dans la console, entrez la commande suivante :

```
docker run hello-world
```



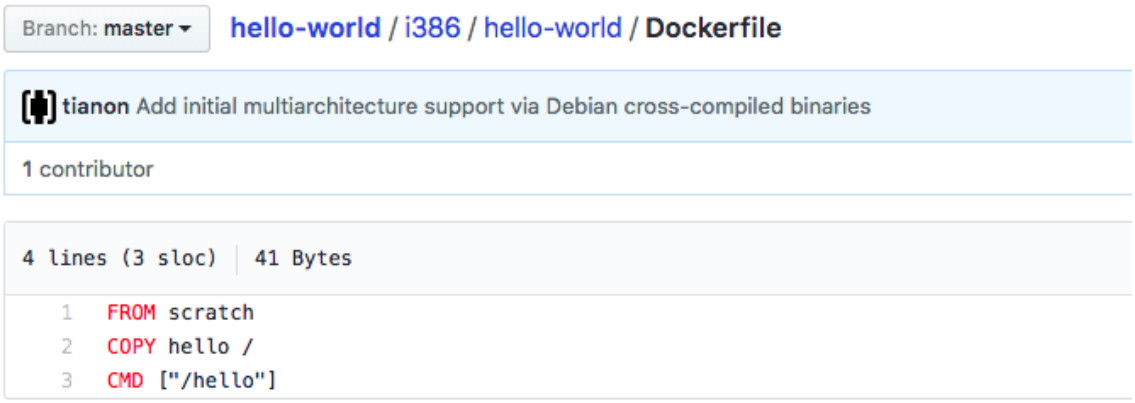
- Que constatez-vous ?
- Comment être certain que l'image n'a rien fait d'anormal sur notre machine ?

Les sources de l'image hello world

Maintenant que nous avons lancé notre première « vrai » machine, intéressons-nous à son fonctionnement. Vous avez du constater le terme « Pulling From » c'est l'image ou les morceaux d'image utile au fonctionnement de votre service. Celui-ci est téléchargé directement depuis le « Docker Hub » (il est également possible d'avoir un Hub privé). Vous pouvez voir « les sources » de l'image en question : ici, comme beaucoup de projets libres l'image est disponible sur Github.

Surprise ! On retrouve des plateformes (amd64, i686, armXX). Et c'est normal, Docker est multiplate-forme et dans le cas de notre exemple l'exécutable « hello » est codé en C il est donc logique de retrouver l'exécutable pour les différentes plateformes ou l'image doit fonctionner.

Maintenant que nous avons vu le projet, entrons plus en détail, allons voir la définition de notre image le fichier Dockerfile.



Peu d'informations, 3 lignes :

- `FROM scratch` (Image de base, celle utilisée comme référence elle aussi possède un DockerFile).
- `COPY hello /` (ajoute le fichier hello à la racine de votre « machine »).
- `CMD ["/hello"]` (Commande lancée au démarrage de votre image).

PS On créera un DockerFile dans le 2nd TP.

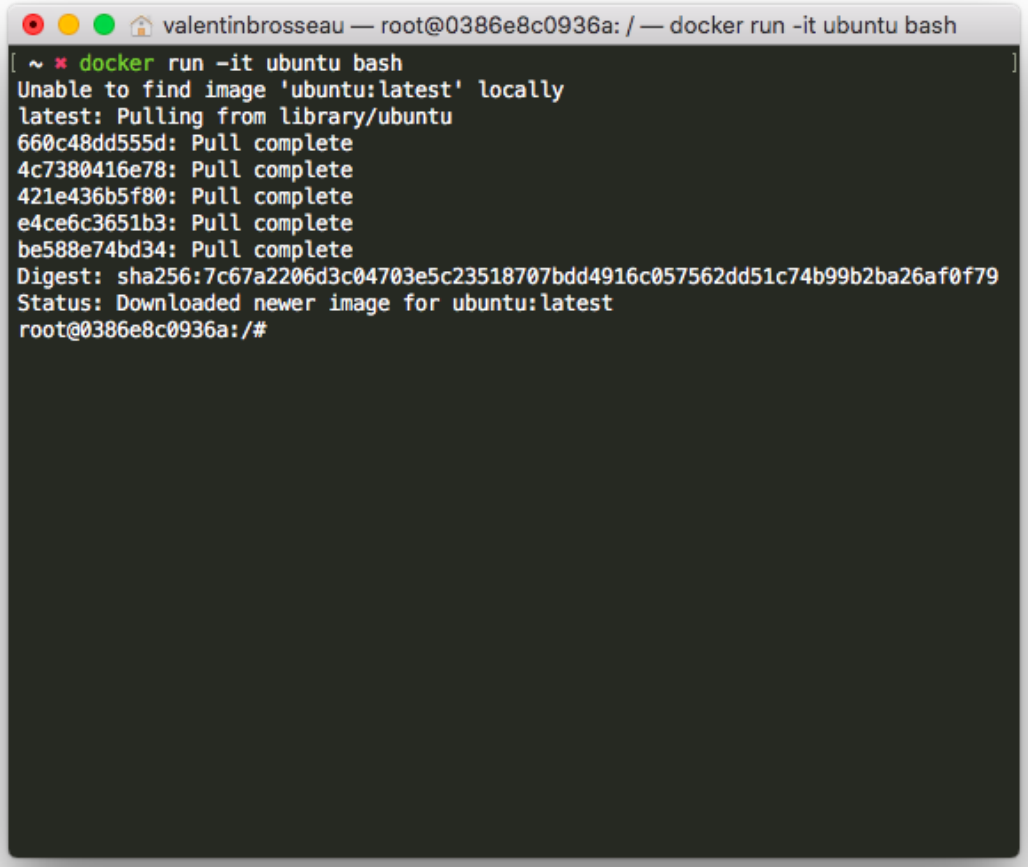
Pour aller plus loin

Bon, un texte à l'écran c'est bien… Mais si on lançait un système entier ? Ubuntu par exemple. Pour ça rien de plus simple, dans la console lancer :

bash

```
docker run -it ubuntu bash
```

1



Et voilà, vous avez un Linux complètement opérationnel en quelques minutes sur votre poste Windows. Pratique ! Même si ce n'est pas vraiment le but premier de Docker c'est cool.

Utiliser un peu le shell de votre « nouveau Linux », exemple de commande :

- `uname -a` : Affiche la version du noyau.
- `whoami` : Qui suis-je ? (normalement root, d'ailleurs est-ce normal ?).
- `top` : Affiche les processus en cours.
- `ls /`

Questions :

- D'ailleurs, pourquoi le top ne retourne-t-il que deux processus ?
- Aucune trace des fichiers de votre machine… normale de base rien n'est accessible.

Créer un fichier dans la machine

Créer un fichier vide avec la commande :

```
touch fichier_test
```

Vérifier avec un `ls` que le fichier est bien présent. Vous pouvez quitter votre l'image en saisissant `exit` dans le terminal. Relancer de nouveau l'image avec la commande :

```
docker run -it ubuntu bash
```

Faites à nouveau un `ls`, que constatez-vous ? Et bien oui, le fichier n'est plus présent… C'est normal, tous les fichiers créés dans l'image sont non persistants (c'est-à-dire qu'ils sont supprimés à chaque fois que l'image s'arrête).

Avoir accès aux fichiers de votre machine

Bon, c'est bien, mais si l'on donnait accès à un stockage persistant à notre image. Sur votre machine le Stockage persistant c'est votre disque dur (HDD, SSD, etc.). Avec docker (comme sous Linux d'ailleurs) on parle de monter « un volume », une fois monté ce volume sera accessible comme un dossier (ou un fichier, on y reviendra).

Monter un dossier

Pour monter un volume il suffit d'ajouter un `-v` à la commande de lancement, exemple pour avoir le dossier courant :

⚠⚠ Attention ! L'accès est en lecture ET en écriture sur **VOTRE MACHINE** donc attention.

Sous Windows :

```
docker run -v %cd%:/mnt/ -it ubuntu bash # Monte le dossier courant dans le /mnt du Docker.
```

Sous Unix :

```
docker run -v $(pwd):/mnt/ -it ubuntu bash # Monte le dossier courant dans le /mnt du Docker.
```

Lancer la commande, `ls /mnt` vous devriez voir vos fichiers.

Monter un fichier

Comme je le disais dans l'introduction avec docker il est possible de rendre accessible le dossier, mais également les fichiers. Pour les fichiers la commande est la même sauf qu'au lieu de spécifier un dossier on spécifie le chemin d'un fichier. Exemple :

Windows :

```
docker run -v %cd%/mon_fichier:/mnt/mon_fichier -it ubuntu bash
```

Unix :

```
docker run -v $(pwd)/mon_fichier:/mnt/mon_fichier -it ubuntu bash
```

Le fichier est maintenant accessible dans votre image Docker. Il est également possible de limiter l'accès à votre fichier en le montant par exemple en « RO » en français « lecture seulement » :

```
docker run -v $(pwd)/mon_fichier:/mnt/mon_fichier:ro -it ubuntu bash
```

Et c'est la que l'on voit la puissance, on verra qu'il sera possible par la suite de créer de vrai « stack » applicative qui définiront l'ensemble des paramètres de notre environnement (cloisonné) et gérant finement les droits d'accès à la configuration par exemple ! Un régal !

Compilation Cordova

Je pense que vous vous souvenez que paramétrer l'environnement de compilation Android sur Windows n'était pas vraiment simple (JDK, SDK, etc.), avec Docker c'est beaucoup plus simple ! Vraiment beaucoup plus simple. Pour compiler un projet Cordova avec Docker j'ai préparé une petite image Docker qui possède l'ensemble des éléments nécessaires, pour l'utiliser :

Unix (Linux, Osx)

```
cd mon_projet_cordova/
docker run -it --entrypoint /bin/bash -v $(pwd):/Sources/ c4software/cordova-
light:latest
# Vous pouvez maintenant lancer, par exemple : cordova build
```

Windows

```
cd mon_projet_cordova/
docker run -it --entrypoint /bin/bash -v %cd%:/Sources/ c4software/cordova-light:latest
# Vous pouvez maintenant lancer, par exemple : cordova build
```