# TP 1: Commandes de base de Docker

Pratique: Les commandes de base

Au programme vous allez manipuler ici les commandes de bases utilisées pour la gestion des containers.

Nous commençons par vérifier la version du client et celle du daemon

```
$ docker version
Client:
 Version:      17.09.0-ce
 API version:  1.32
 Go version:   go1.8.3
 Git commit:   afdb6d4
 Built:        Tue Sep 26 22:40:09 2017
 OS/Arch:      darwin/amd64

Server:
 Version:      17.09.0-ce
 API version:  1.32 (minimum version 1.12)
 Go version:   go1.8.3
 Git commit:   afdb6d4
 Built:        Tue Sep 26 22:45:38 2017
 OS/Arch:      linux/amd64
 Experimental: true
```

Lancement de containers
Serveur web
Lançons un container basé sur l'image nginx.

```
$ docker container run nginx
```

Vous devriez obtenir le résultat suivant:

Arrêtez le container en faisant un CTRL-C dans le terminal.

Nous allons utiliser des options supplémentaires afin:

- de lancer le serveur web en tâche de fond (option -d)
- de publier, sur le port 8080 de l'hôte, le port 80 exposé dans l'image nginx (option -p 8080:80)

```
$ docker container run -d -p 8080:80 nginx
```

L'identifiant du container est retourné. Le serveur web est disponible sur le port 8080 de la machine hôte. Un curl sur http://localhost:8080 permet d'obtenir la page index.html servie par défaut par nginx.

```
$ curl localhost:8080
```

## Shell intéractif

Lançons un shell interactif dans un container basé sur l'image ubuntu, pour cela nous utilisons les options suivantes:

- l'option -t crée un pseudo terminal dans notre container
- l'option -i permet de laisser l'entrée standard ouvert

```
$ docker container run -ti ubuntu
```

Une fois l'image ubuntu récupérée, un shell est lancé dans le container

```
/ #
```

Note: nous n'avons pas spécifié de commande à la suite du nom de l'image (alpine) car par défaut la commande sh est lancée.

Lançons une commande dans notre container afin de connaitre la version de la distribution ubuntu que nous utilisons.

```
# lsb_release -a
```

Nous pouvons maintenant sortir du container.

```
# exit
```

## Lancement d'un process dans un container existant

Généralement un seul processus tourne dans un container, son PID est 1. Il est parfois utile, par exemple pour des fins de debugging, de lancer un autre processus dans un container toujours en execution, c'est ce que nous allons illustrer dans cette partie.

Nous commençons par lancer, en background, un container basé sur l'image mongo:3.4 (MongoDB est l'une des bases NoSQL les plus connues / utilisées). Nous utilisons également l'option --name afin de spécifier un nom pour notre container.

```
$ docker container run -d --name mongo mongo:3.4
```

L'image mongo:3.4 est téléchargée depuis le Docker Hub, le container est créé et son identifiant est renvoyé.

La commande docker container exec permet de lancer un processus dans un container existant. Nous allons lancer un shell intéractif dans le container mongo que nous venons de créer. Pour cela, nous pouvons fournir l'ID ou bien le nom du container à la commande exec.

```
$ docker container exec -ti mongo bash
```

Nous avons ainsi access à un shell bash à l'intérieur du container mongo. Regardons quels sont les processus en cours dans ce container.

```
# ps ax
```

Le résultat de cette commande est semblabe à celui ci-dessous. Le process de PID 1 est mongod, le daemon mongo. C'est la commande qui est lancée par défaut lorsque l'on créé un container à partir d'une image mongo.

```
  PID TTY      STAT   TIME COMMAND
```

```
    1 ?        Ssl    0:17 mongod
   35 ?        Ss     0:00 bash
   42 ?        R+     0:00 ps ax
```
Nous pouvons également voir que le process bash, dans lequel nous sommes, est également listé.

```
# exit
```

## Inspection d'un container

La commande inspect permet d'obtenir une vue détaillé d'un container (état, paramètres de configuration, …). Nous commençons par lancer un container basé sur nginx.

```
$ docker container run --name www -d nginx
```
Nous utilisons la command inspect pour obtenir les détails de ce container.

```
$ docker container inspect www
```
Cette commande génère beaucoup d'information. Souvent, nous n'aurons besoin que d'un sous-ensemble de ces informations, parfois même d'un seul champ. Pour cela nous pouvons utiliser les Go templates afin de récupérer seulement l'information dont nous avons besoin.

Les 2 commandes suivantes utilisent le format Go template pour récupérer le hostname et l'adresse IP du container.

La clé Hostname est disponible dans Config.

```
$ docker container inspect --format "{{ .Config.Hostname }}" www
```
La clé IPAdress est disponible dans NetworkSettings.

```
$ docker container inspect --format "{{ .NetworkSettings.IPAddress }}" www
```

pour plus de details sur l'tulisation de –format consulter les sites suivants :
https://docs.docker.com/engine/reference/commandline/inspect/
https://buildvirtual.net/how-to-use-docker-inspect/

## Autres commandes pour la gestion des containers

La liste des commandes disponibles pour la gestion des containers est obtenue avec

```
$ docker container --help
```

## Comprendre la couche associée au container

La couche d'un container, est la couche read-write créé lorsqu'un container est lancé. C'est la couche dans laquelle tous les changements effectués dans le container sont sauvegardés. Cette couche est supprimée avec le container et ne doit donc pas être utilisée comme un stockage persistant.

Nous allons illustrer cela en commençant par lancer un shell intéractif dans un container basé sur l'image ubuntu.

```
$ docker container run -ti ubuntu
```
figlet est un package qui prend un texte en entrée et le formatte de façon marrante. Par défaut ce package n'est pas disponible dans l'image ubuntu.

# figlet
La commande devrait donner le résultat suivant.

bash: figlet: command not found
Nous installons figlet dans le container.

# apt-get update -y
# apt-get install figlet
Vérifions que le binaire fonctionne

# figlet Holla
Ce qui devrait donner le résultat suivant

```
 _           _ _
| |__   ___ | | | __ _
| '_ \ / _ \| | |/ _` |
| | | | (_) | | | (_| |
|_| |_|\___/|_|_|\__,_|
```

Nous pouvons maintenant sortir du container

# exit
Nous lançons un nouveau container basé sur ubuntu.

$ docker container run -ti ubuntu
Est ce que le mackage figlet est présent ?

# figlet
Nous obtenons alors l'erreur suivante

bash: figlet: command not found
Comment expliquez-vous ce résultat ?

Chaque container lancé à partir de l'image ubuntu est différent des autres. Le second container est différent de celui dans lequel figlet a été installé. Chacun correspond à une instance de l'image ubuntu et a sa propre couche, ajoutée au dessus des couches de l'image, et dans laquelle tous les changements effectué dans le container sont sauvés.

Sortons du container

# exit
Nous pouvons lister les containers en exécution

$ docker container ls
ainsi que l'ensemble des containers (en exécution ou non) sur la machine hôte

$ docker container ls -a
Depuis cette liste, récuperez l'ID du container dans lequel le package figlet a été installé (ce devrait être le second container de la liste) et redémarrez le avec la commande start.

$ docker container start CONTAINER_ID
Lancez un shell intéractif dans ce container en utilisant la commande exec.

```
$ docker container exec -ti CONTAINER_ID bash
```
Vérifez que figlet est présent dans ce container.

```
# figlet still there !
```
Nous pouvons maintenant sortir de ce container une nouvelle fois.

```
# exit
```

Nettoyage
Nous listons l'ensembles des containers créés sur la machine.

```
$ docker container ls -a
```
Si nous ajoutons l'option -q, nous obtenons seulement les IDs des containers.

```
$ docker container ls -aq
```
Pour supprimer tous les containers, nous pouvons utiliser les commandes rm et ls -aq conjointement. Nous ajoutons l'option -f afin de forcer la suppression des containers encore en exécution. Il faudrait sinon arrêter les containers et les supprimer.

```
$ docker container rm -f $(docker container ls -aq)
```
Tous les containers ont été supprimés.

```
$ docker container ls -a
```

En résumé
Nous avons commencé à jouer avec les containers et à comprendre la couche read-write qui est créée et associée à chaque container. Nous avons également vu les commandes les plus utilisées pour la gestion du cycle de vie des containers (run, exec, ls, rm, inspect).

## Exercice 1

### Création d'un container basé sur ubuntu

1. Lancez un container basé sur ubuntu en lui fournissant la command 'echo hello'
2. Quelles sont les étapes effectuées par le docker daemon ?
3. Lancez un container basé sur ubuntu sans lui spécifier de commande. Qu'observez-vous ?

## Exercice 2

### Création d'un container en mode interactif

1. Lancez un container basé sur ubuntu en mode interactif sans lui spécifier de commande
2. Que s'est-il passé ?
3. Quel est la commande par défaut d'un container basé sur ubuntu ?
4. Naviguez dans le système de fichiers
5. Utilisez le gestionnaire de package d'alpine (apt) pour ajouter un package
   - apt update
   - apt install curl

## Exercice 3

### Création de containers en foreground (premier plan) / background

1. Lancez un container basé sur ubuntu en lui spécifiant la commande ping 8.8.8.8
2. Arrêtez le container avec CTRL-C. Le container est t-il toujours en cours d'exécution ?
   **Note:** utiliser la command docker ps .
3. Lancez un container en mode interactif en lui spécifiant la command ping 8.8.8.8
4. Arrêtez le container avec CTRL-P CTRL-Q. Le container est t-il toujours en cours d'exécution ?
5. Lancez un container en background, toujours en lui spécifiant la command ping 8.8.8.8. Le container est t-il toujours en cours d'exécution ?

## Exercice 4

### Création d'un container en exposant un port sur la machine hôte

1. Lancez un container basé sur nginx et publiez le port 80 du container sur le port 8080 de l'hôte
2. Vérifiez depuis votre navigateur que la page par défaut de nginx est servie sur http://localhost:8080
3. Lancez un second container en publiant le même port
4. Qu'observez-vous ?

## Exercice 5

### Liste des containers sur le système

1. Listez les containers en cours d'exécution
2. Est ce que tous les containers que vous avez créés sont listés ?
3. Ajouter l'option -a pour voir également les containers qui ont été stoppés

4. Ajoutez l’option -q pour ne lister que les IDs des containers (en cours d’exécution ou stoppés)

## Exercice 6

### Inspection d’un container

1. Lancez, en background, un nouveau container basé sur nginx en publiant le port 80 du container sur le port 3000 de la machine host. Notez l'identifiant du container retourné par la commande précédente.
2. Inspectez le container en utilisant son identifiant
3. En utilisant le format Go template, récupérez le nom et l’IP du container
4. Manipuler les Go template pour récupérer d'autres information
   - Le templating Go est très riche, voici quelques exemples supplémentaires: https://docs.docker.com/config/formatting/

## Exercice 7

### Lancement d’un processus dans un container existant

1. Lancez un container en background, basé sur l’image ubuntu. Spécifiez la commande ping 8.8.8.8 et le nom ping avec l’option --name
2. Observez les logs du container en utilisant l’ID retourné par la commande précédente ou bien le nom du container
3. Quittez la commande de logs avec CTRL-C
4. Lancez un shell sh, en mode interactif, dans le container précédent
5. Listez les processus du container
6. Qu'observez vous par rapport aux identifiants des processus ?

## Exercice 8

### Arrêt et suppression de tous les containers existant

1. Listez tous les containers (actifs et inactifs)
2. Stoppez tous les containers encore actifs en fournissant la liste des IDs à la commande stop
3. Vérifiez qu’il n’y a plus de containers actifs
4. Listez les containers arrêtés
5. Supprimez tous les containers
6. Vérifiez qu’il n’y a plus de containers