FONCTIONNEMENT ET MANIPULATION DES CONTENEURS DOCKER

Différence entre image et conteneur

Pour rappel lorsque vous utilisez des fonctionnalités permettant une **isolation du processus** (comme les namespaces et les cgroups), on appelle cela des conteneurs. Dans ces conteneurs on peut retrouver généralement un ou plusieurs programme(s) avec toute leurs dépendances de manière à les maintenir isolées du système hôte sur lequel elles s'exécutent.

Nous avons aussi vu que sur docker **un conteneur est une instance d'exécution d'une image** plus précisément un conteneur est ce que l'image devient en mémoire lorsqu'elle est exécutée (c'est-à-dire une image avec un état ou un processus utilisateur).

Quelques commandes

Créer un conteneur

Pour créer une instance de notre image, ou autrement dit **créer un conteneur**, alors nous utiliserons une commande que nous avions déjà vue sur les chapitres précédents, soit :

```
docker run [OPTIONS] <IMAGE_NAME ou ID>
```

Nous allons pour le moment créer un conteneur basé sur l'image hello-world, et pour faire les choses dans les règles de l'art, nous allons d'abord commencer par

télécharger notre image depuis le Docker Hub Registry, et ensuite on va exécuter notre image afin de créer notre conteneur.

Étape 1 : Téléchargement de l'image hello-world :

```
docker pull hello-world:latest
```

Étape 2 : Instanciation de l'image hello-world :

```
docker run hello-world:latest
```

Avouons-le, cette image n'est pas vraiment utile, car elle n'est prévue que pour afficher un petit message d'information et en plus de ça elle se ferme directement après. Téléchargeons une image plus utile, comme par exemple l'image Ubuntu, et pourquoi pas la manipuler avec l'interpréteur de commandes bash et de télécharger dessus des outils comme l'outil git.

Normalement si on suit la logique vu précédemment, on devrait exécuter les commandes suivantes :

Étape 1 : Téléchargement de l'image ubuntu :

```
docker pull ubuntu:latest
```

Étape 2 : Instanciation de l'image ubuntu :

```
docker run ubuntu:latest
```

Vous : *"Ah mais attend mais moi quand je lance mon image Ubuntu, mon conteneur se quitte directement après, est-ce que les commandes sont bonnes ? !"*

La réponse est oui, vos commandes sont bien les bonnes mais ce n'est pas le cas pour vos options, car oui cette commande `docker run` possède beaucoup d'options consultables soit depuis la doc officielle, soit depuis commande `docker run --help`.

Comme vous, pouvez le constater il existe beaucoup d'options, mais rassurez-vous, car vous n'avez pas besoin de tous les connaître, voici pour le moment les options qui nous intéressent pour ce chapitre :

- `-t` : **Allouer un pseudo TTY** (terminal virtuel)
- `-i` : **Garder un STDIN ouvert** (l'entrée standard plus précisément l'entrée clavier)
- `-d` : **Exécuter le conteneur en arrière-plan** et afficher l'ID du conteneur

Dans notre cas nous avons besoin d'une Tty (option `-t`) et du mode interactif (option `-i`) pour interagir avec notre conteneur basé sur l'image Ubuntu. Tentons alors l'exécution de ces deux options :

```
docker run -ti ubuntu:latest
```

héhé, vous êtes maintenant à l'intérieur de votre conteneur ubuntu avec un jolie shell .

Profitons-en pour télécharger l'outil git, mais juste avant n'oubliez pas de récupérer les listes de paquets depuis les sources afin que votre conteneur sache quels sont les paquets disponibles en utilisant la commande `apt-get update`, ce qui nous donnera la commande suivante :

```
apt-get update -y && apt-get install -y git
```

Maintenant, si j'essaie de vérifier mon installation git alors vous constaterez que je n'ai aucune erreur :

```
git --version
```

Résultat :

```
git version 2.17.1
```

Je profite un peu de cette partie pour vous montrer **la puissance des conteneurs**. Je vais détruire mon conteneur Ubuntu avec la commande `rm -rf /*`. Si vous souhaitez faire comme moi, alors **ASSUREZ-VOUS BIEN AVANT QUE VOUS ETES DANS UN CONTENEUR**. Après avoir lancé cette commande je peux vous affirmer d'ores et déjà dire que j'ai bien détruit mon conteneur, preuve à l'appui je ne peux même plus lancer la commande `ls` :

```
root@7cfb553ebcc2:/# ls
bash: /bin/ls: No such file or directory
```

Vous : *"Mais tu es fou de lancer cette commande destructrice !"*

Ne vous inquiétez pas, je vais réparer mon conteneur en même pas 1 seconde. Je vais d'abord commencer par **quitter mon conteneur** avec la commande `exit` et ensuite je vais demander à mon moteur Docker la phrase suivante : *"abra cadabra Docker crée-moi un nouveau conteneur"* :

```
docker run -ti ubuntu:latest
```

Maintenant si je relance ma commande `ls` :

```
root@7cfb553ebcc2:/# ls
        bin   etc   lib64  opt   run   sys  var
    boot  home  media  proc  sbin  tmp
    dev   lib   mnt    root  srv   usry
```

Tout est en ordre chef ! Mon conteneur est de nouveau fonctionnel et devinez quoi cette réparation ne m'a même pas pris 1 seconde à faire (oui je tape vite au clavier). Cette puissance de pouvoir **créer rapidement des conteneurs** à la volée avec une latence quasi inexistante est possible, car nous avions préalablement téléchargé notre image Ubuntu sur notre machine locale. Et comme notre image contient déjà

son **propre environnement d'exécution**, alors elle est déjà prête à l'emploi et elle n'attend plus qu'à être exécutée !

Par contre dans mon nouveau conteneur, si je lance la commande `git --version`, alors je vais obtenir une belle erreur m'indiquant que git est inexistant dans mon conteneur.

```
bash: git: command not found
```

Pourquoi ? Car **les données et les fichiers dans un conteneur sont éphémères** !

Vous : *"Quoi ? Il existe bien un moyen pour sauvegarder mes données ??? Dit moi oui please please ???"*

Bien sûr que oui, sinon les conteneurs ne serviraient pas à grand-chose. Il existe deux façons pour **stocker les données d'un conteneur**, soit on transforme notre conteneur en image (cependant ça reste une méthode non recommandée, mais je vous montrerai à la fin de ce chapitre comment l'utiliser), soit on utilise le système de volume, nous verrons cette notion dans un autre chapitre dédié aux volumes.

Pour le moment je souhaite vous montrer l'utilité de l'option `-d` de la commande `docker run`. Pour mieux comprendre cette option je vais utiliser l'image officielle d'Apache. Bon, maintenant vous connaissez la routine, on télécharge l'image et on l'exécute, mais juste avant j'aimerais bien vous montrer trois autres options utiles de la commande `docker run` :

- `--name` : **Attribuer un nom au conteneur**
- `--expose` : **Exposer un port ou une plage de ports** (on demande au firewall du conteneur de nous ouvrir un port ou une plage de port)

- `-p ou --publish` : **Mapper un port** déjà exposé, plus simplement ça permet de **faire une redirection de port**

Par défaut l'image apache expose déjà le port 80 donc pas besoin de lancer l'option `--expose`, pour notre exemple nous allons mapper le port 80 vers le 8080 et nommer notre conteneur en "monServeurWeb", ce qui nous donnera comme commande :

```
docker pull httpd:latest
```

```
docker run --name monServeurWeb -p 8080:80 httpd:latest
```

Une fois votre image exécutée, visitez http://localhost:8080 et vous verrez la phrase **"It works"**!

Seulement le seul souci, c'est que sur notre terminal, on aperçoit les logs d'Apache qui tournent en boucle :

```
hatim@localhost ~ » docker run --name monServeurWeb -p 8080:80 httpd:latest
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'Ser
verName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'Ser
verName' directive globally to suppress this message
[Wed Jun 12 20:14:30.194632 2019] [mpm_event:notice] [pid 1:tid 140201429643328] AH00489: Apache/2.4.39 (Unix) config
ured -- resuming normal operations
[Wed Jun 12 20:14:30.194815 2019] [core:notice] [pid 1:tid 140201429643328] AH00094: Command line: 'httpd -D FOREGROU
ND'
172.17.0.1 - - [12/Jun/2019:20:14:33 +0000] "GET / HTTP/1.1" 304 -
172.17.0.1 - - [12/Jun/2019:20:14:34 +0000] "GET / HTTP/1.1" 304 -
172.17.0.1 - - [12/Jun/2019:20:14:34 +0000] "GET / HTTP/1.1" 304 -
172.17.0.1 - - [12/Jun/2019:20:14:34 +0000] "GET / HTTP/1.1" 304 -
172.17.0.1 - - [12/Jun/2019:20:14:34 +0000] "GET / HTTP/1.1" 304 -
172.17.0.1 - - [12/Jun/2019:20:14:35 +0000] "GET / HTTP/1.1" 304 -
172.17.0.1 - - [12/Jun/2019:20:14:35 +0000] "GET / HTTP/1.1" 304 -
```

Ce qui serait intéressant c'est de laisser notre conteneur tourner en arrière-plan avec l'option `-d`

```
docker run --name monServeurWeb -d -p 8080:80 httpd:latest
```

Cependant, si vous relancez la commande, vous obtenez l'erreur suivante :

```
docker: Error response from daemon: Conflict.
The container name "/monServeurWeb" is already in use by container "832d83bf810a28a68
You have to remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.
```

Pour faire simple, il nous indique qu'il n'est pas possible de démarrer deux conteneurs sous le même nom (ici "monServeurWeb"). Dans ce cas vous avez deux solutions, soit vous choisissez la facilité en renommant votre conteneur, mais ce n'est pas vraiment très propre comme solution, en plus moi, ce que je souhaite c'est de garder le nom "monServeurWeb" pour mon conteneur. Soit la deuxième solution, qui est de supprimer mon conteneur, et d'en recréer un nouveau avec le bon nom. Ça tombe bien car c'est l'occasion de vous montrer d'autres **commandes utiles pour manipuler les conteneurs Docker**.

Afficher la liste des conteneurs

Pour supprimer notre conteneur, il faut d'abord l'identifier et par la suite récolter soit son id, soit son nom. Ça sera l'occasion de vous dévoiler une commande que vous utiliserez beaucoup, cette commande vous permettra de lister les conteneurs disponibles sur votre machine.

```
docker container ls
```

ou

```
docker ps
```

Résultat :

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STA
55e5ccfc1357        httpd:latest        "httpd-foreground"       19 seconds ago      Up
```

Voici l'explication des différentes colonnes :

- `CONTAINER ID` : id du conteneur
- `IMAGE` : L'image sur laquelle c'est basé le conteneur
- `COMMAND` : Dernière commande lancée lors de l'exécution de votre image (ici la commande `httpd-foreground` permet de lancer le service apache en premier plan)
- `CREATED` : date de création de votre conteneur
- `STATUS` : statut de votre conteneur, voici une liste des **différents états d'un conteneur** :
    - `created` : conteneur créé mais non démarré (cet état est possible avec la commande `docker create`)
    - `restarting` : conteneur en cours de redémarrage
    - `running` : conteneur en cours d'exécution
    - `paused` : conteneur stoppé manuellement (cet état est possible avec la commande `docker pause`)
    - `exited` : conteneur qui a été exécuté puis terminé
    - `dead` : conteneur que le service docker n'a pas réussi à arrêter correctement (généralement en raison d'un périphérique occupé ou d'une ressource utilisée par le conteneur)
- `PORTS` : les ports utilisés par votre conteneur
- `NAMES` : nom de votre conteneur

Par défaut la commande `docker ps` ou `docker container ls` ne vous affichera que les conteneurs en état running, pour pouvoir afficher les conteneurs peu importe leur

état, alors il faut utiliser l'option `-a` ou `--all`.

Supprimer un conteneur

Maintenant que nous avons pu récupérer l'id ou le nom du conteneur, on est capable de **supprimer notre conteneur** avec la commande suivante :

```
docker rm <CONTAINER NAME ou ID>
```

Soit :

```
docker rm monServeurWeb
```

Une fois mon conteneur supprimé, je peux enfin créer mon conteneur Apache avec l'option `-d`

```
docker run --name monServeurWeb -d -p 8080:80 httpd:latest
```

Vous ne voyez plus les logs d'apache et votre conteneur tourne en arrière-plan, preuve à l'appui, avec la commande `docker ps` :

```
CONTAINER ID        IMAGE               COMMAND              CREATED             STA
f3f971625432        httpd:latest        "httpd-foreground"   7 minutes ago       Up
```

Exécuter une commande dans un conteneur

Il existe une commande `docker exec` qui permet de lancer n'importe quelle commande dans un conteneur déjà en cours d'exécution. Nous allons l'utilisée pour récupérer notre interpréteur de commande `/bin/bash`, ce qui aura pour but de se connecter directement à notre conteneur Apache.

```
docker exec -ti monServeurWeb /bin/bash
```

Vous êtes maintenant dans votre conteneur. Pour s'amuser un peu, on va changer le message de la page d'accueil :

```
echo "<h1>Docker c'est vraiment cool</h1>" > /usr/local/apache2/htdocs/index.html
```

Revisitez la page, http://localhost:8080 et vous verrez votre nouveau message.

> **Information**
>
> Pour **quitter votre conteneur sans le détruire**, utilisez le raccourcis suivant : `Ctrl + P + Q`

Afficher les logs d'un conteneur

Dès fois, vous aurez besoin de déboguer votre conteneur en regardant les **sorties/erreurs d'un conteneur**.

Il existe pour cela la commande `docker logs` qui vient avec deux options très utiles :

- `-f` : **suivre en permanence les logs du conteneur** (correspond à tail -f)
- `-t` : **afficher la date et l'heure de réception des logs d'un conteneur**

```
docker logs -ft monServeurWeb
```

si vous visitez votre page, alors vous verrez des logs s'afficher successivement sur votre terminal (`Ctrl + C` pour quitter vos logs)

Transformer votre conteneur en image

Comme promis voici, la commande qui permet de **transformer un conteneur en image**, afin de stocker nos données

> **Attention**
>
> Je me répète mais c'est important, cette commande n'est pas vraiment recommandée, pour stocker vos données. Il faut pour ça utiliser les volumes, que nous verrons dans un autre chapitre.

Voici les étapes que nous allons suivre :

- Exécuter notre conteneur basé sur l'image officielle Ubuntu
- Installer l'outil git
- Mettre du texte dans un nouveau fichier
- Transformer notre conteneur en image
- Relancer notre un nouveau conteneur basé sur cette nouvelle image

```
docker run -ti --name monUbuntu  ubuntu:latest
```

```
apt-get update -y && apt-get install -y git
```

```
echo "ceci est un fichier qui contient des donnes de test" > test.txt && cat test.txt
```

`Ctrl + P + Q`

Maintenant, c'est l'étape ou je vais créer mon image depuis mon nouveau conteneur. Voici son prototype :

```
docker commit <CONTAINER NAME or ID> <NEW IMAGENAME>
```

Ce qui nous donnera :

```
docker commit monUbuntu ubuntugit
```

> **Information**
>
> Vous pouvez voir, votre nouvelle image avec la commande `docker images`

Voilà, maintenant, on va lancer notre conteneur basé sur cette nouvelle image :

```
docker run -ti --name ubuntugit_container  ubuntugit
```

À présent, je vais vérifier si les données ont bien été stockées sur ce nouveau conteneur.

```
cat test.txt
```

Résultat :

```
ceci est un fichier qui contient des donnes de test
```

```
git --version
```

Résultat :

```
git version 2.17.1
```

Notre outil git et notre fichier sont bien présents dans notre nouveau conteneur .

Conclusion

Vous avez appris, à manipuler des conteneurs docker. Dans le futur chapitre, nous allons apprendre à créer notre propre image Docker avec le fameux **Dockerfile**.

Comme pour chaque fin de chapitre, je vous mets un **cheat sheet** reprenant les commandes qu'on a pu voir dans ce chapitre.

```
## Exécuter une image Docker
docker run <CONTAINER_ID ou CONTAINER_NAME>
    -t ou --tty : Allouer un pseudo TTY
    --interactive ou -i : Garder un STDIN ouvert
    --detach ou -d : Exécuter le conteneur en arrière-plan
    --name : Attribuer un nom au conteneur
    --expose: Exposer un port ou une plage de ports
    -p ou --publish : Mapper un port  "<PORT_CIBLE:PORT_SOURCE>"
    --rm : Supprimer automatiquement le conteneur quand on le quitte

## Lister des conteneurs en état running Docker
docker container ls
# ou
docker ps
    -a ou --all : Afficher tous les conteneurs peut-importe leur état

## Supprimer un conteneur Docker
docker rm <CONTAINER_ID ou CONTAINER_NAME>
    -f ou --force : forcer la suppression

## Supprimer tous les conteneurs Docker
docker rm -f $(docker ps -aq)

## Exécuter une commande dans un conteneur Docker
docker exec <CONTAINER_ID ou CONTAINER_NAME> <COMMAND_NAME>
    -t ou --tty : Allouer un pseudo TTY
    -i ou --interactive : Garder un STDIN ouvert
    -d ou --detach : lancer la commande en arrière plan

## sorties/erreurs d'un conteneur
docker logs <CONTAINER_ID ou CONTAINER_NAME>
    -f : suivre en permanence les logs du conteneur
    -t : afficher la date et l'heure de la réception de la ligne de log
    --tail <NOMBRE DE LIGNE> = nombre de lignes à afficher à partir de la fin (par dé


## Transformer un conteneur en image
docker commit <CONTAINER_NAME ou CONTAINER_ID> <NEW IMAGENAME>
    -a ou --author <string> : Nom de l'auteur (ex "John Hannibal Smith <hannibal@a-te
    -m ou --message <string> : Message du commit
```