Une première prise en main de docker

Benoit DARTIES

***Avant propos** : Nous utilisons ici un système d'exploitation Ubuntu 16.04. Les commandes utilisent un shell bash et l'installation s'appuie sur le système de gestion de paquetages propre à ubuntu (aptitude). Ce TP s'adapte très facilement sur d'autres distributions UNIX, Windows ou Mac OS X. Les zones de code en vert correspondent aux commandes saisies, et celles en gris au résultat affiché à l'écran.*

1 Installation de docker

Après avoir démarré sur notre système, nous récupérons l'application docker. L'installation se fait très simplement en installant les packages **docker** et **docker.io**. La majorité des commandes devra être exécutée avec des droits administrateur, c.-à-d. en les précédant de **`sudo`**. Nous installons donc docker avec la commande suivante :

```
$ sudo apt-get install docker docker.io
```

Nous pouvons vérifier que l'installation s'est correctement déroulée en affichant la version de docker :

```
$ sudo docker version
```

2 Récupération d'une image d'un container applicatif

Même si elle n'est pas présente initialement, une image peut être automatiquement téléchargée lors de la création d'un container applicatif. Pour des raisons pédagogiques, nous effectuerons plutôt l'installation pas à pas en débutant d'abord par la récupération des images.

De base Docker n'est livré avec aucune image de container. Nous pouvons vérifier que cette liste est vide avec la sous-commande **`images`**

```
$ sudo docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

Nous allons tout d'abord récupérer des images de containers disponibles sur internet.

Nous allons sur **`hub.docker.com`** qui est la plate-forme officielle de dépôt des images docker :

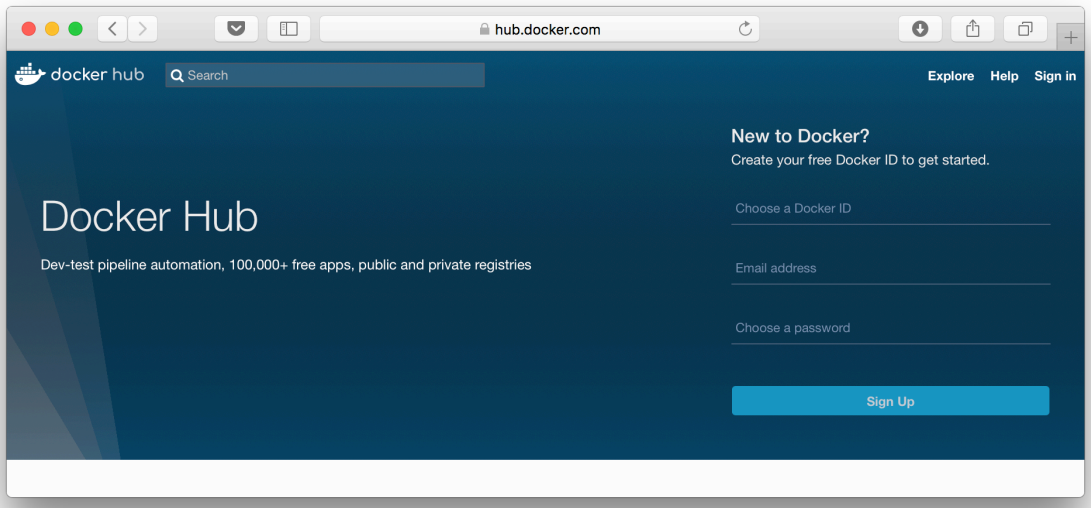

Figure 1 - Ecran d'accueil de la plateforme Docker Hub

Dans un premier temps, nous allons installer toute une distribution complète ubuntu en tant que container applicatif pour illustrer le fonctionnement. Par la suite nous prendrons des applications plus ciblées. Nous recherchons **`ubuntu`** dans la barre de recherche. Le résultat de cette recherche est présenté en Figure 2.

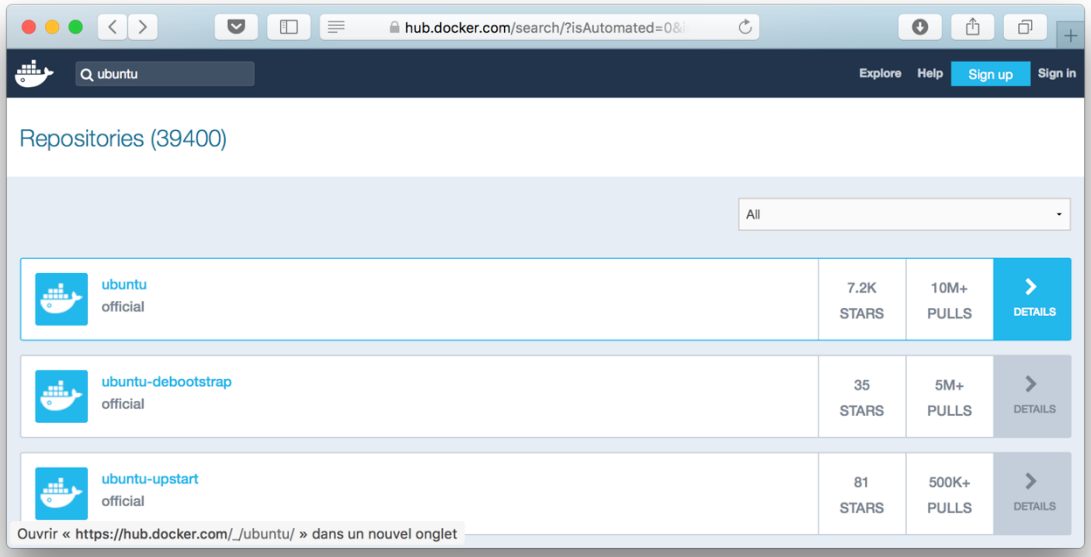

Figure 2 - Résultat de la recherche 'ubuntu' sur hub.docker.com

Nous sélectionnons le premier résultat et en affichons les détails. Nous parcourons la description de l'image, qui contient des informations supplémentaires sur son utilisation, ou les différentes versions mises à disposition, le changelog, etc (Figure 3). L'élément le plus important de cette page est situé en haut à droite de cette dernière, *Docker Pull Command*, qui indique la commande à exécuter pour installer cette image dans notre distribution Docker.). Notez également la présence d'un onglet « tag », dont l'utilité est expliquée un peu après.

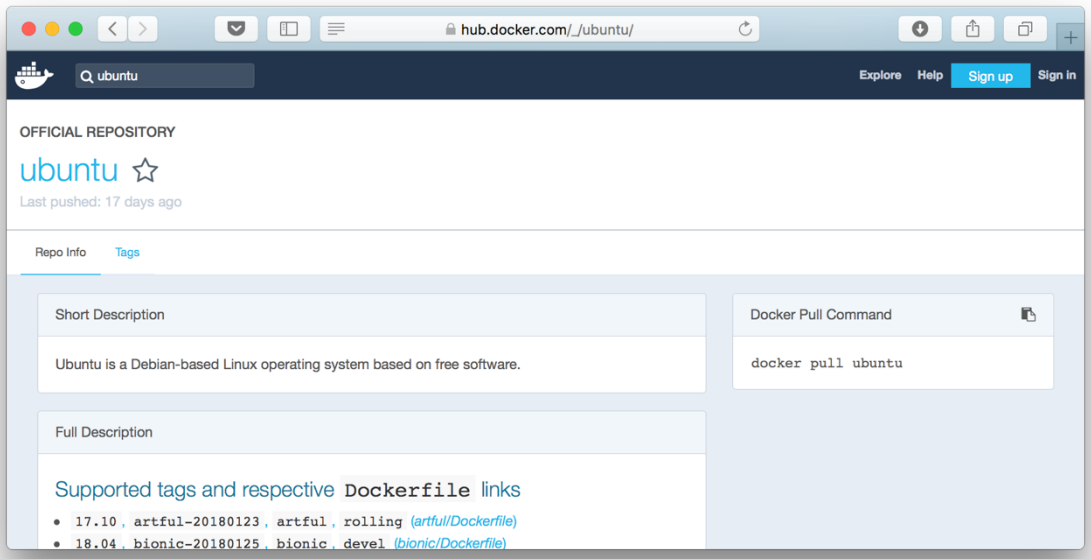

Figure 3 - détail de l'image ubuntu de hub.docker.com

Dans notre terminal nous installons donc l'image ubuntu en tapant la commande suivante :

```
$ sudo docker pull Ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
1be7f2b886e8: Pull complete
6fbc4a21b806: Pull complete
c71a6f8e1378: Pull complete
4be3072e5a37: Pull complete
06c6d2f59700: Pull complete
Digest:
sha256:e27e9d7f7f28d67aa9e2d7540bdc2b33254b452ee8e60f388875e5b7d9b2b696
Status: Downloaded newer image for ubuntu:latest
```

Nous vérifions que l'image est désormais installée :

```
$ sudo docker images
REPOSITORY        TAG           IMAGE ID        CREATED             SIZE
Ubuntu            latest        0458a4468cbc    2 weeks ago         112 MB
```

Par défaut, la dernière version de l'image est installée. En utilisation un peu plus avancée, il est possible de sélectionner d'autres versions de l'image. Pour cela il suffit d'ajouter le tag correspondant à la version avec la notation *nomImage* :*nomTag* . Par exemple pour la version 14.04 d'ubuntu, il faudrait installer l'image ubuntu :14.04. La liste des tags disponibles pour une image donnée est disponible dans l'onglet

3 Création et manipulation de container applicatifs

3.1 Créer, lancer, stopper un container

A partir de l'image **ubuntu**, nous créons désormais un nouveau container applicatif avec la commande create. Nous devons ici utiliser cette commande avec l'option **–t** , pour créer et associer un terminal virtuel à ce container. Nous allons également préciser un nom à ce container avec l'option **––name nom**. Si cette option n'est pas précisée, un nom aléatoire sera alloué au container.

```
$ sudo docker create -t --name maVMubuntu ubuntu
b2d473c54d41f518dfc30c593cbf9c6f79e911da7805923cee387685976a584b
```

Nous récupérons en sortie l'identifiant unique (UUID) du container applicatif. Ce container est désormais créé mais pas lancé. Il est possible de visualiser l'ensemble des containers créés ainsi que leur statut avec la commande **docker ps**. Sans option, cette commande ne liste que les containers actifs. Avec l'option **–a** (all) de la commande liste l'ensemble des containers, actifs ou non.

```
$ sudo docker ps
CONTAINER ID  IMAGE   COMMAND     CREATED       STATUS        PORTS   NAMES
$ sudo docker ps -a
CONTAINER ID  IMAGE   COMMAND     CREATED       STATUS        PORTS   NAMES
b2d473c54d41  ubuntu "/bin/bash"  2 minutes ago Created           maVMubuntu
```

Notre container applicatif existe bien, mais n'est pas lancé. Nous le lançons avec la commande **docker start.**

```
$ sudo docker start maVMubuntu
maVMubuntu
```

puis nous vérifions son exécution avec la commande **docker ps** :

```
benoit@nostromo:~$ sudo docker ps
CONTAINER ID   IMAGE    COMMAND      CREATED        STATUS        PORTS    NAMES
b2d473c54d41   ubuntu   "/bin/bash" 5 minutes ago Up 11 seconds     maVMubuntu
```

Le statut **Up** indique que le container fonctionne bien. Pour stopper un container, nous utiliserions la commande **docker stop**, auquel quoi le statut passerait en **Exited**. Vérifiez ceci puis relancez le container.

3.2 Exécution de premières commandes dans un container applicatif

Avant tout, il faut savoir que les containers peuvent être manipulés de nombreuses façons différentes, mais que certains containers réagissent différemment à certaines commandes selon qu'ils contiennent juste une application, ou tout un système d'exploitation. Dans le cas d'un container ubuntu, certaines commandes pourront être déstabilisantes de prime abord, et d'autre ne pas marcher comme on l'aurait pensé. Ce comportement est normal mais peut être appréhendé avec l'expérience.

Jusqu'à présent notre container tourne, mais nous n'avons aucune interaction sur ce container. Simplement nous pouvons le lancer, ou le stopper. Nous allons voir quelques commandes utiles pour manipuler ce container.

Commençons par quelque chose de simple : notre container tourne avec ubuntu. Nous allons lui demander d'exécuter une commande, par exemple nous retourner son nom d'hôte. Pour envoyer une commande à notre container, nous utilisons la commande **docker exec** en précisant le nom du container à qui envoyer la commande, suivi de la commande à exécuter. Le container va alors exécuter la commande et nous renvoyer son résultat. Sous ubuntu, récupérer le nom d'hôte s'effectue avec la commande **hostname**. Nous tapons donc la commande suivante (le container doit être actif) :

```
$ sudo docker exec maVMubuntu hostname
b2d473c54d41
```

Nous constatons que le nom d'hôte du container a été initialisé avec son identifiant docker. Nous pouvons tester quelques commandes usuelles d'ubuntu en manipulations complémentaires.

```
$ sudo docker exec maVMubuntu date
Mon Feb 12 14:42:50 UTC 2018
$ sudo docker exec maVMubuntu uname -sov
Linux #32~16.04.2-Ubuntu SMP Thu Jul 20 10:19:48 UTC 2017 GNU/Linux
```

Une finalité intéressante est de lancer un shell bash sur cette machine, afin d'en avoir un contrôle total. L'exécution d'un shell est cependant plus contrainte : d'une part il faut disposer d'un terminal virtuel tty pour intéragir avec ce programme. Ceci est réalisable en ajoutant l'option **–t** à notre ligne de commande. D'autre part, il faut maintenir ouvert la liaison avec l'entrée standard. Ceci est réalisable en ajoutant l'option **–i** à notre ligne de commande.

Nous tapons donc la commande suivante et observons le résultat :

```
 $ sudo docker exec -t -i maVMubuntu /bin/bash
root@b2d473c54d41:/#
```

Nous avons désormais un prompt shell sur notre container virtuel ubuntu. Notez que le prompt affiche l'identité du container, signe que nous interagissons avec ce dernier. Nous pouvons interagir avec le système, naviguer dans l'arborescence, et surtout installer les packages de notre choix. Pour quitter le container, il suffit de fermer le shell avec la commande **exit**. Le shell se ferme et nous revenons sur notre terminal d'appel (symbolisé par le prompt **$**.

```
root@b2d473c54d41:/# exit
exit
$
```

3.3 Interaction via un serveur openssh

L'image ubuntu que nous avons installée est minimale et dispose de très peu d'outils. Nous allons lui installer un serveur openssh, pour pouvoir nous y connecter par la suite sans passer par l'exécution de la commande **`docker exec`**. Nous exécutons donc les actions suivantes :

- Connexion au container

```
$ sudo docker exec -t -i maVMubuntu /bin/bash
root@b2d473c54d41:/#
```

- Mise en place d'un mot de passe pour le compte **`root`**

```
root@b2d473c54d41:/# passwd root
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

- Mise à jour des dépôts pour aptitude sur la machine container

```
root@b2d473c54d41:/# apt-get update
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
...
Reading package lists... Done
```

- Installation d'**`openssh-server`** :

```
root@b2d473c54d41:/# apt-get install openssh-server
Reading package lists... Done
...
After this operation, 55.1 MB of additional disk space will be used.
Do you want to continue? [Y/n]
(taper Y et valider)
...
done.
Processing triggers for systemd (229-4ubuntu21) ...
```

- Lancement d'**`openssh-server`** via le gestionnaire de services :

```
root@b2d473c54d41:/# service ssh start
* Starting OpenBSD Secure Shell server sshd
```

Le containeur fait désormais tourner un serveur ssh.

Par défaut, le login en utilisant le compte **`root`** est désactivé en ssh. Nous pouvons soit installer un éditeur et modifier le fichier de configuration du serveur ssh (fichier **`/etc/ssh/sshd_config`**) et relancer le serveur , soit créer un compte utilisateur standard, lui définir un mot de passe, et une fois authentifié nous passerons en mode administrateur en utilisant le mot de passe du **`root`**. Nous choisissons cette seconde solution dans la suite.

Nous exécutons les commandes suivantes pour créer un utilisateur **`benoit`** :

```
root@b2d473c54d41:/# useradd benoit
root@b2d473c54d41:/# passwd benoit
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Nous sortons enfin du container, et retournons au système d'exploitation :

```
root@b2d473c54d41:/# exit
```

Nous testons ensuite la connexion ssh : lorsque le container a été lancé, il lui a été attribué une adresse IP. Il est possible de voir toute la configuration du container avec la commande **`docker inspect`** suivi du nom du container. Si nous tapons la commande suivante :

```
$ sudo docker inspect maVMubuntu
```

Nous obtenons l'ensemble de la configuration de la machine au format JSON. En inspectant cette configuration nous y trouvons les informations de configuration réseau, ici l'adresse IP **`172.17.0.2`** :

```
...
"NetworkSettings": {
      ...
      "Gateway": "172.17.0.1",
      "IPAddress": "172.17.0.2",
      "MacAddress": "02:42:ac:11:00:02",
      ...
}
...
```

Nous pouvons filtrer ce résultat avec la commande suivante pour récupérer directement l'adresse IP :

```
$ sudo docker inspect --format '{{ .NetworkSettings.IPAddress }}' maVMubuntu
```

Nous pouvons alors vérifier que la connexion vers le serveur ssh du container est opérationnelle en nous authentifiant avec le compte standard que nous avons créé :

```
$ ssh 172.17.0.2 -l benoit
benoit@172.17.0.2's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-28-generic x86_64)
...
$
```

Nous passons enfin en mode administrateur dans le container avec la commande su. Nous retrouvons alors notre accès administrateur :

```
$ su
Password:
root@b2d473c54d41:/#
```

Attention : stopper un container puis le relancer équivaut à stopper la machine et la relancer. Si le serveur ssh n'a pas été configuré pour être démarré automatiquement, il sera inactif par la suite.

Une fois les manipulations faites, nous stoppons, puis supprimons ce container avec les commandes **`docker stop`** et **`docker rm`** depuis le système d'exploitation hôte :

```
$ sudo docker stop maVMubuntu
maVMubuntu
benoit@nostromo:~$ sudo docker rm maVMubuntu
maVMubuntu
```

Ces premières commandes introductives s'avèrent nécessaires pour comprendre et manipuler les éléments de base de docker, mais elles ne permettent pas d'illustrer la puissance de de docker. Dans la suite nous allons pousser notre découverte de docker en introduisant la notion de volumes.

4 Manipulation des containers de stockage

Une bonne pratique consiste à créer des volumes, c'est à dire des containers dédiés au stockage. Ils permettent de séparer l'exécution d'un processus d'un coté et le stockage de l'autre. Il est même possible d'isoler les éléments de stockage entre eux, par exemple pour séparer les données de la configuration.

4.1 Création d'un volume de stockage

Nous allons tout d'abord créer un volume docker. Nous utilisons la sous-commande **`volume`** :

```
docker volume create --name HDDubuntuVM
```

Avec cette commande, nous venons de créer un containeur de stockage nommé HDDubuntuVM. On peut utiliser les sous commandes **`ls`** et **`inspect`** pour lister l'ensemble des volumes, ou obtenir des informations complémentaires, comme le chemin physique d'un volume.

Listons l'ensemble des volumes docker disponibles. Nous y retrouvons notre volume :

```
docker volume ls
DRIVER              VOLUME NAME
local               HDDubuntuVM
```

Affichons ensuite des informations détaillées sur le volume **`HDDubuntuVM`** avec la sous-commande **`inspect`** de **`docker volume`** (cette commande existe aussi pour les containers applicatifs) :

```
$ sudo docker volume inspect HDDubuntuVM
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/HDDubuntuVM/_data",
        "Name": "HDDubuntuVM",
        "Options": {},
        "Scope": "local"
    }
]
```

Ce container est associé à un répertoire sur notre système, dans notre exemple, il s'agit de **`/var/lib/docker/volumes/HDDubuntuVM/_data`** . Nous reviendrons plus tard sur ce répertoire.

4.2 Association d'un volume de stockage à un container applicatif.

Nous allons ensuite créer un nouveau container applicatif et cette fois-ci le lier avec notre volume de stockage. Ceci se réalise en précisant lors de la création l'existence d'un lien entre d'une part un volume de stockage et un point de montage dans le container applicatif, qui s'exprime avec l'option **`-v`** suivi de ***`nomVolume:pointMontage`*** . Dans notre exemple, nous allons créer un nouveau container applicatif maVMubuntu2 basé sur **`ubuntu`**, en liant le volume de stockage **`HDDubuntuVM`** existant à un répertoire **`/data`** du container applicatif.

- Création du container maVMubuntu2

```
$ sudo docker create -t -v HDDubuntuVM:/data/test  --name maVMubuntu2 ubuntu
e7c686cff89c3733b91eebc4d368f82e03331ec720514c4962b151270a4a612f
```

- Lancement du container et attachement d'un shell utilisateur

```
$ sudo docker start maVMubuntu2
$ sudo docker exec -t -i maVMubuntu2 bash
root@e7c686cff89c:/#
```

- Vérification de l'existence du répertoire **/data** et inspection :

```
root@e7c686cff89c:/# ls
bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
root@e7c686cff89c:/# cd /data
root@e7c686cff89c:/data# ls
root@e7c686cff89c:/data#
```

Le répertoire est présent. Vide, mais présent. La suite devient plus intéressante …

4.3 Les volumes comme passerelle entre hôte et containers applicatifs.

Toujours dans notre container applicatif **maVMubuntu2**, créons un fichier nommé **test.txt** et ajoutons-y une ligne de contenu :

```
root@e7c686cff89c:/data# touch test.txt
root@e7c686cff89c:/data# echo "bonjour, ceci est un fichier test" > test.txt
root@e7c686cff89c:/data# ls
test.txt
root@e7c686cff89c:/data# cat test.txt
bonjour, ceci est un fichier test
```

Quittons enfin notre container applicatif.

```
root@e7c686cff89c:/data# exit
```

Rappelons que le container tourne toujours en arrière-plan. De retour sur le système hôte, listons le contenu du répertoire qui était associé à notre volume, récupérable avec **docker volume inspect**, dans notre exemple **/var/lib/docker/volumes/HDDubuntuVM/_data** (notons qu'il nous faudra les droits administrateur pour naviguer dedans).

```
$ sudo ls /var/lib/docker/volumes/HDDubuntuVM/_data
test.txt
$ sudo cat /var/lib/docker/volumes/HDDubuntuVM/_data/test.txt
bonjour, ceci est un fichier test
```

Nous pouvons voir que le fichier créé au sein du container est présent, il s'agit de notre volume monté en tant que répertoire **/data** dans **maVMubuntu2**. Nous pourrions ajouter quelques fichiers depuis le système hôte pour tester et vérifier que ces fichiers sont bien présents dans le répertoire **/data** de **maVMubuntu2**.

Nous pouvons faire mieux ! Depuis le système hôte, nous créons un nouveau container applicatif **maVMubuntu2** et nous lions notre volume **HDDubuntuVM** à ce dernier en utilisant un point de montage, qui peut être identique ou non, ici pour marquer la différence nous utiliserons **/donnees/ext** .

```
$ sudo docker create -t -v HDDubuntuVM:/donnees/ext  --name maVMubuntu3 ubuntu
12bc8848701c003a0b419f2800278d767176d11ba17a4f0f1dbecda01bac8a18
```

Nous listons enfin le contenu du répertoire **/donnes/ext** (inutile d'utiliser un shell, nous pouvons exécuter directement les commandes **ls** ou **cat** sans passer par **bash**)

```
$ sudo docker exec -t -i maVMubuntu3 ls /donnees/ext
test.txt
$ sudo docker exec -t -i maVMubuntu3 cat /donnees/ext/test.txt
bonjour, ceci est un fichier test
```

Nous avons donc un volume de stockage **HDDubuntuVM**, ici associé à deux containers applicatifs **maVMubuntu2** et **maVMubuntu3**, tous deux utilisant l'image **ubuntu**, monté sur deux points de montages différents (/data sur **maVMubuntu2**, et **/donnees/ext** sur **maVMubuntu3)** et associé à un répertoire de notre hôte, ici **/var/lib/docker/volumes/HDDubuntuVM/_data**.

Les possibilités de docker se révèlent peu à peu : il est ainsi possible d'avoir différents containers applicatifs partageant certains fichiers en commun, et d'autres non.

4.4 Exemple : des volumes sur répertoires existants pour partage de configuration

Il est possible de lier nos volumes avec des répertoires déjà existants de notre container applicatif, et/ou de les partager avec d'autres systèmes,

Dans l'exemple suivant, nous allons disposer de deux containers applicatifs **maVMubuntu4** et **maVMubuntu5**, et de trois volumes **VMconfig**, **dataShared**, **dataVM4** et **dataVM5**.

Le volume **VMconfig** sera initialement lié au répertoire **/etc** de **maVMubuntu4**, - **/etc** est le répertoire de configuration global d'un système UNIX - et nous pourrons directement interagir sur ce dernier depuis notre hôte. Puis nous l'utiliserons pour répliquer la configuration de **maVMubuntu4** vers **maVMubuntu5** en le montant sur le répertoire **/etc** de **maVMubuntu5**. Chaque container applicatif disposera de son propre répertoire de données personnelles (**dataVM4** et **dataVM5**), ainsi que d'un répertoire de partage de données **dataShared**. Ces répertoires seront montés sur des points de montage au nom explicite.

- Création des volumes

```
benoit@nostromo:~$ sudo docker volume create VMconfig
benoit@nostromo:~$ sudo docker volume create dataVM4
benoit@nostromo:~$ sudo docker volume create dataVM5
benoit@nostromo:~$ sudo docker volume create dataShared
```

- Affichage du chemin de **VMconfig** sur l'hôte et vérification du contenu qui doit être vide

```
$ sudo docker inspect --format '{{ .Mountpoint }}'  VMconfig
/var/lib/docker/volumes/VMconfig/_data
$ sudo ls /var/lib/docker/volumes/VMconfig/_data
$
```

- Création du container **maVMubuntu4** et montage des répertoires

```
$ sudo docker create -t -v VMconfig:/etc -v dataVM4:/data/local -v
dataShared:/data/shared --name maVMubuntu4 ubuntu
505f9e380fcaac1e19909917764dca614339794270fd03628d8a9c6974cf0a79
```

- Vérification du contenu de **VMconfig**,rempli avec les fichiers de configuration de **/etc**

```
$ sudo ls /var/lib/docker/volumes/VMconfig/_data/
adduser.conf default      init    legal         opt        rc5.d     subgid
alternatives deluser.conf init.d  libaudit.conf os-release rc6.d     subuid
...
```

- Création du container **maVMubuntu5** et montage des répertoires

```
$ sudo docker create -t -v VMconfig:/etc -v dataVM5:/data/local -v
dataShared:/data/shared --name maVMubuntu5 ubuntu
34f74c48dc3a7d0b8557affb747a8e31e1dce48af1038418d305796cbff59e60
```

- Lancement des containers

```
$ sudo docker start maVMubuntu4
$ sudo docker start maVMubuntu5
```

5 Déploiement d'une application dédiée, un serveur mySQL

Jusqu'à présent, nous avons utilisé une image **`ubuntu`** pour illustrer le fonctionnement global de docker dans un environnement pédagogique. Mais la philosophie docker a comme objectif principal de faire seulement un serveur particulier par container, et non tout un OS. L'exécution du container est alors limitée à une utilisation minimale et optimale. Dans cette ultime section, nous allons mettre en place un container dédié à l'exécution d'un serveur mySQL.

nous introduisons également de nouvelles commandes et options :

- **docker run** permet de créer un container depuis une image et de le lancer (**create + start**)
- **-e** permet de passer des variables d'environnement depuis la ligne de commande. Chaque container possède ses propres variables d'environnement. Pour connaître la liste des varaibles utilisables il faut consulter la documentation
- **-p** permet de faire de la translation de ports. Dit autrement, nous allons pouvoir lier un port de notre machine avec un port du container.
- **-d** permet de passer une exécution en mode daemon

De plus, si l'image d'un container n'est pas présente dans le système, docker se charge de regarder si elle existe sur hub.docker.com et de la télécharger.

Connaissant le nom de l'image du serveur à déployer, ici **`mysql`**, nous pouvons, en une seule ligne, lancer ce dernier en tant que container.

```
$ sudo docker run --name VMmysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=test
  -d mysql
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
f49cf87b52c1: Pull complete
Digest:
sha256:7cdb08f30a54d109ddded59525937592cb6852ff635a546626a8960d9ec34c30
Status: Downloaded newer image for mysql:latest
cae1ada93b41f24524d9c07720b5260f61ba5a9089c79355f5efd7e7b120764d
```

Nous vérifions que le serveur **`mysql`** est actif avec **`docker ps`** : (présentation modifiée)

```
$ sudo docker ps
CONTAINER ID : cae1ada93b41
IMAGE       mysql
...
PORTS       0.0.0.0:3306->3306/tcp
NAMES       VMmysql
```

Notez que grâce à l'utilisation de **`–p 3306:3306`**, tout ce qui arrive sur notre machine sur le port **`3306`** en TCP (port par défaut en SQL) est redirigé sur le port **`3306`** tcp de notre container.Depuis notre système hôte, nous pouvons récupérer un client mysql et tester la connexion en utilisant le login **`root`** et le mot de passe entré lors de la création du container.

- Installation d'un client mysql

```
$ sudo apt install mysql-client-core-5.7
```

- Connexion au serveur mysql depuis le système hôte

```
mysql -h 127.0.0.1 -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
mysql>
```

- Connexion au serveur mysql via le container

```
$ sudo docker exec -ti VMmysql mysql -p
```