

Exploitation de Docker

Activité 1 : Installation et première exploitation de Docker

L'objectif est ici de :

- se familiariser avec le fonctionnement de Docker, les notions d'images et de conteneurs ;
- se familiariser avec les commandes Docker ;
- commencer à préparer l'image que l'on va déployer sur le cloud pour les administrateurs réseaux ;
- commencer à préparer l'image qui va permettre de monter une plateforme de test complète.

Vous disposez de la documentation suivante :

- **document 1** : Les composants de Docker
- **document 2** : Installation de Docker
- **document 3** : Gestion basique des images
- **document 4** : Gestion basique des conteneurs
- **document 5** : Lancement d'un conteneur en interactif
- **document 6** : Création d'une nouvelle image à partir d'un conteneur
- **document 7** : Comment rendre accessible un service (conteneur en arrière plan)
- **document 8** : Récapitulatif des commandes de base

Travail à faire



Installez Docker.



Avant de continuer, il est conseillé de réaliser toutes les manipulations détaillées dans la documentation. *Vous supprimerez ensuite tous les conteneurs créés et les images.*



Téléchargez le docker Debian dans sa version *strech*.



Démarrez le conteneur en interactif avec un shell en lui donnant le nom « servWebStatique ».



Mettez à jour les paquets du conteneur.



Mettez à jour la date du système (dpkg-reconfigure tzdata).



Installez le serveur Web apache2.



Créez une nouvelle image (vos_initiales/debian:strech-apache2) à partir du conteneur modifié.



Lancez 2 conteneurs (servweb1 et servweb2) à partir de cette nouvelle image permettant d'accéder à la page Web par défaut d'Apache (le premier mappé sur le port 8001 et le deuxième mappé sur le port 8002).



Testez l'accès à chacun des services Web.

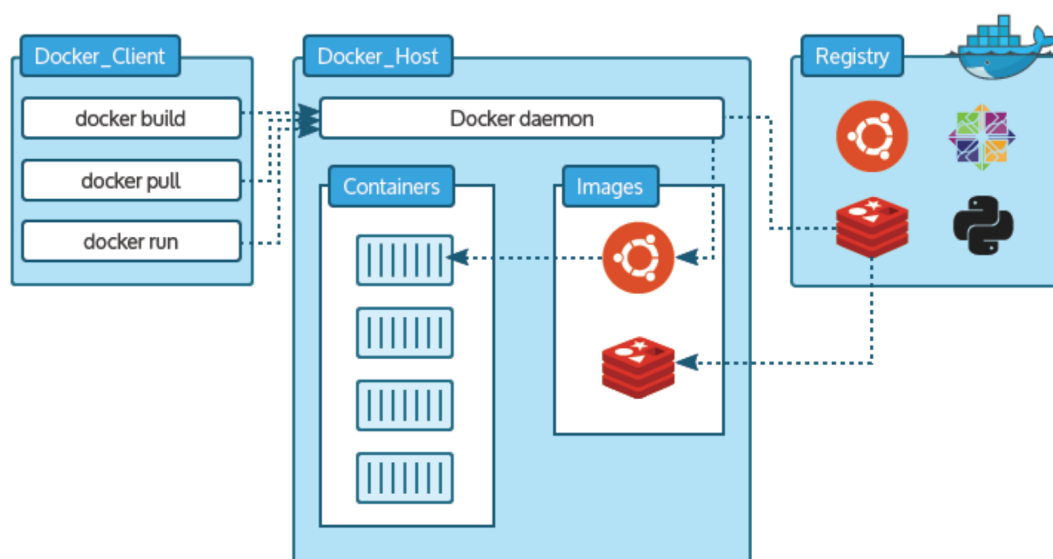
Document 1 – Les composants de Docker

Docker utilise une architecture client/serveur. Il est composé d'un serveur qui gère les images, les conteneurs et leurs journaux respectifs et d'un client qui communique avec le serveur via une API.

Une image est l'entité de base de Docker. Elle peut être comparée à une image de machine virtuelle en lecture seule. Elle contient tout ce que le créateur a décidé d'installer (apache2, php, des pages html et php, des scripts, etc).

Un conteneur est une image en cours d'exécution : lorsqu'un conteneur est lancé à partir d'une image, Docker va monter le système de fichiers de l'image en lecture seule et une couche accessible en lecture/écriture. Dans ce conteneur, il sera donc possible d'interagir avec les applications installées dans l'image, d'exécuter des scripts, de modifier des fichiers, d'installer d'autres applications, etc.

Le schéma ci-dessous (ainsi que les explications qui suivent) extraite d'un l'article « Comprendre Docker & son écosystème » du 5 décembre 2016 de Yahya Al Dhuraibi (<https://www.scalair.fr/blog/docker>) résume bien les principaux composants de Docker et leurs interactions :



Le Docker Registry permet de stocker et distribuer des images. Par défaut, le registry est le « Docker Hub », qui détient des milliers d'images publiques. Les conteneurs Docker sont créés en utilisant ces images de base.

Le Docker Host représente la machine physique ou la machine virtuelle dans laquelle Docker Daemon et les conteneurs sont déployés : c'est ce que nous appellerons « hôte » dans la suite de cette production.

Le Docker Daemon est à la fois responsable de la création, du démarrage et du monitoring des conteneurs, mais aussi de la construction et du stockage des images. Le système d'exploitation a la charge de démarrer le Docker Daemon.

Le Docker Client communique avec le Docker Daemon via une API REST. Il propose un ensemble de commandes pour :

- exécuter et gérer les conteneurs ;
- décrire, créer, publier des images dans un *registry* ;
- Interroger un *registry* et gérer les versions des images dans un *registry*.

La combinaison des Docker Clients et des Docker Daemons est appelée **Docker Engine**.

Document 2 – Installation de Docker

D2.1 - Installation de Docker

L'installation de Docker se réalise en quelques étapes très simples. Sur Linux, il suffit d'ajouter le dépôt Docker dans les sources de dépôts du système.

Installation des paquets nécessaires à l'utilisation du dépôt *docker* en https

```
apt-get install apt-transport-https ca-certificates curl gnupg2 software-properties-common
```

Importation de la clé du dépôt docker

```
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg | apt-key add -
```

La variable « `$(. /etc/os-release; echo "$ID")` » renvoie la distribution. Attention à l'espace entre « `/` » et le « `.` » et de même entre « `add` » et « `-` ».

Intégration du dépôt docker dans le *source.list* et mise à jour des dépôts

```
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")  
$(lsb_release -cs) stable"  
  
apt update
```

Installation de Docker

```
apt install docker-ce
```

Configuration de docker pour démarrer automatiquement

```
systemctl enable docker
```

Enfin, pour ne pas avoir besoin de faire tourner docker en administrateur, il est possible de **modifier les droits de l'utilisateur en l'ajoutant au groupe docker**.

```
addgroup votre_login docker
```

Il est nécessaire de se reconnecter avec cet utilisateur pour continuer à travailler avec Docker.

D2.2 - Vérification de l'installation de Docker

! Pour utiliser Docker en étant connecté avec l'utilisateur « non root », il peut être nécessaire de relancer la machine.

Toutes les commandes Docker commencent par le mot clé « docker ».

```
docker version
```

Client:		Server:	
Version:	18.03.0-ce	Engine:	
API version:	1.37	Version:	18.03.0-ce
Go version:	go1.9.4	API version:	1.37 (minimum version 1.12)
Git commit:	0520e24	Go version:	go1.9.4
Built:	Wed Mar 21 23:10:06 2018	Git commit:	0520e24
OS/Arch:	linux/amd64	Built:	Wed Mar 21 23:08:35 2018
Experimental:	false	OS/Arch:	linux/amd64
Orchestrator:	swarm	Experimental:	false

Lancement d'un docker de test qui sert uniquement à afficher “Hello world”.

Le lancement d'un conteneur Docker se réalise avec la commande « docker run » (voir document 4). Dans la commande suivante, Docker devrait vous afficher qu'il ne trouve pas l'image Docker du nom de hello-world mais il la télécharge automatiquement sur le *Docker hub*.

```
docker run hello-world
```

```
...
Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```



Hors le « Hello from Docker », cette commande renvoie une série d'informations qui sont expliquées dans les documents suivants.

Document 3 – Gestion basique des images

Il n'est pas inutile de rappeler que :

- **une image est l'entité de base de Docker.** Elle peut être comparée à une image de machine virtuelle en lecture seule. Elle contient tout ce que le créateur a décidé d'installer (apache2, php, des pages html et php, des scripts, etc) ;
- **un conteneur est la version exécutée d'une image Docker.** Il possède la copie du système de fichiers de l'image, ainsi que la capacité de lancer des processus. Lorsqu'un conteneur est lancé à partir d'une image, Docker va monter le système de fichiers de l'image en lecture seule et monter une couche accessible en lecture/écriture. Dans ce conteneur, il sera donc possible d'interagir avec les applications installées dans l'image, d'exécuter des scripts, de modifier des fichiers, d'installer d'autres applications, etc.



Lorsqu'il est apporté des modifications à une image, seule la différence est prise en compte et placée dans la couche supérieure d'exécution du conteneur. Les modifications peuvent être sauvegardées sous la forme d'une image qui sera juste composée de la couche de différences par le moteur de Docker, identifiée par un identifiant unique. Ainsi, cette seule version pourra être envoyée vers le dépôt public de Docker (ou vers un dépôt privé), sans avoir à envoyer toutes les couches dans leur intégralité.

Le **"Docker registry" (Hub officiel)** contient un très grand nombre d'images permettant d'utiliser sous forme de conteneur la plupart des outils utilisés aujourd'hui : apache, php, mysql, mariadb, haproxy, debian, ubuntu, odoo, etc.



Docker Hub comporte des dépôts officiels, mais aussi des images créées par n'importe quel utilisateur. Lorsqu'on récupère une image, il faut toujours en vérifier son auteur, des logiciels malveillants pouvant être installés.

Interroger le dépôt officiel en ligne de commandes sans passer par le site web

```
docker search <mot clé>
```

docker search ubuntu

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys...	7603		[OK]
rastasheep/ubuntu-sshd	Dockerized SSH service, built on top of offi...	140		[OK]
ubuntu-upstart	Upstart is an event-based replacement for th...	86		[OK]
eclipse/ubuntu_jdk8	Ubuntu, JDK8, Maven 3, git, curl, nmap, mc, ...	5		[OK]

La commande renvoie beaucoup de résultats (limités par défaut à 25) et classés par ordre décroissant de popularité. Seuls 4 résultats apparaissent ci-dessus.

NAME indique le nom de l'image. Les noms des images créées par l'équipe de Docker (comme httpd) ne sont pas précédés par le nom de l'utilisateur qui l'a créée (comme eboraas/apache-php).

DESCRIPTION : décrit l'image et donne des informations sur son contenu.

STARS indique le nombre de fois où l'image a été mise en favori par les utilisateurs, elle donne ainsi une idée sur sa popularité.

OFFICIAL indique si l'image a été produite par une source officielle reconnue par l'équipe de Docker.

AUTOMATED indique si l'image a été créée automatiquement depuis un dépôt GitHub ou Bitbucket.

Il est possible d'affiner cette commande. Comme pour n'importe quelle autre commande, les options sont décrites avec la commande :

```
docker search --help
```

...

Options:



```
-f, --filter filter  Filter output based on conditions provided
--format string     Pretty-print search using a Go template
--limit int         Max number of search results (default 25)
--no-trunc          Don't truncate output
```

L'équivalent de la recherche de la page précédente via le Web donne l'image ci-dessous :




Repositories (42442)

Le nombre de fois où
l'image a été téléchargée

All		
 ubuntu official	7.6K STARS	10M+ PULLS
 ubuntu-debootstrap official	38 STARS	5M+ PULLS

Un clic sur « Détails » permet d'avoir des informations sur les images (plus ou moins détaillées).

OFFICIAL REPOSITORY

 ubuntu ☆

Last pushed: 4 days ago

Repo Info Tags

Short Description

Ubuntu is a Debian-based Linux operating system based on free software.

Full Description

Docker Pull Command

```
docker pull ubuntu
```

On pourra par exemple trouver les
versions des applications ou les
commandes pour lancer un conteneur.

Tag Name	Compressed Size	Last Updated
xenial	43 MB	4 days ago
xenial-20180417	43 MB	4 days ago
16.04	43 MB	4 days ago

Docker permet d'ajouter un *tag* à une image. Par défaut c'est la dernière image (latest) qui est téléchargée.

Télécharger une image

```
docker pull <nom image>
```

docker pull ubuntu

```
Using default tag: latest
latest: Pulling from library/ubuntu
a48c500ed24e: Pull complete
1e1de00ff7e1: Pull complete
0330ca45a200: Pull complete
471db38bcfbf: Pull complete
0b4aba487617: Pull complete
Digest: sha256:c8c275751219dadad8fa56b3ac41ca6cb22219ff117ca98fe82b42f24e1ba64e
Status: Downloaded newer image for ubuntu:latest
```

Docker télécharge les différentes couches qui forment l'image.

Pour récupérer une autre version de l'image, il faut associer le nom du tag au nom de l'image. Dans l'exemple précédent, pour récupérer la version *xenial*, il faut utiliser la commande suivante : **docker pull ubuntu:xenial**.

```
xenial: Pulling from library/ubuntu
297061f60c36: Pull complete
e9ccef17b516: Pull complete
dbc33716854d: Pull complete
8fe36b178d25: Pull complete
686596545a94: Pull complete
Digest: sha256:1dfb94f13f5c181756b2ed7f174825029aca902c78d0490590b1aaa203abc052
Status: Downloaded newer image for ubuntu:xenial
```

Connaître les images disponibles sur le système

La liste des images téléchargées localement s'obtient avec la commande :

```
docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	xenial	0b1edfbffd27	6 days ago	113MB
ubuntu	latest	452a96d81c30	6 days ago	79.6MB
hello-world	latest	e38bc07ac18e	3 weeks ago	1.85kB

Supprimer une image

```
docker rmi <nom image>
```



- Il n'est pas possible de supprimer une image si un conteneur associé est exécuté (voir partie suivante) ;
- une image peut être référencée dans plusieurs référentiels, comme cela est le cas pour hello-world : dans ce cas, il faut utiliser l'option « f ».

docker rmi hello-world -f

```
Untagged: hello-world:latest
Untagged: hello-world
@sha256:f5233545e43561214ca4891fd1157e1c3c563316ed8e237750d59bde73361e77
Deleted: sha256:e38bc07ac18ee64e6d59cf2eafcd99cec2364dfe129fe0af75f1b0194e0c96
Deleted: sha256:2b8cbd0846c5aeaa7265323e7cf085779eaf244ccbdd982c4931aef9be0d2faf
```

Document 4 – Gestion basique des conteneurs



On rappelle ici que l'image est un fichier en lecture seule, contenant lui-même un système de fichiers avec tout le nécessaire pour faire fonctionner la ou les applications. Le conteneur est le processus qui utilise l'image pour faire tourner la ou les applications.

Lancer un conteneur

```
docker run [OPTIONS] <nom image> [COMMANDE]
```

Selon ce que l'on veut faire, il est possible de lancer un conteneur :

- **en interactif**, si l'on veut agir sur le conteneur pour par exemple le mettre à jour ou installer un paquetage (voir **document 5**) ;
- **en arrière plan**, si l'on veut, par exemple, lancer un service installé dans ce conteneur : le conteneur doit dans ce cas là rester actif : **c'est l'usage le plus courant** (voir **document 7**) ;
- **en arrière plan**, mais uniquement pour lancer une commande : le conteneur existe toujours mais n'est plus actif : **il s'arrête dès que la ou les commandes passées sont exécutées**.



Si le conteneur n'était pas déjà téléchargé, la commande *run* le fait automatiquement (sans requérir un *pull* spécifique)

La commande suivante lance un conteneur (sans options et commandes spécifiques à exécuter dans le conteneur) à partir de l'image *ubuntu* :

docker run ubuntu



Mais rien ne semble s'être passé !

Lister les conteneurs en cours d'exécution (actifs)

```
docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

Il n'y a **aucun conteneur actif** malgré le fait qu'on en ait créé un avec la commande *run*.

Lister tous les conteneurs qu'ils soient actifs ou non

```
docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
25366713428d	ubuntu	"/bin/bash"	7 seconds ago	Exited (0) 5 seconds ago		sad_jennings

CONTAINER ID : ID unique du conteneur.

IMAGE : image utilisée pour le conteneur.

COMMAND : commande exécutée ⇒ le processus lancé ici est */bin/bash* (shell Bash) (il s'agit du processus lancé par défaut prévu par les concepteurs de cette image).

CREATED : temps depuis la création du conteneur.

STATUS : état du conteneur (en cours ou arrêté) ici *exited* avec un code retour 0 (sans erreur) ⇒ le processus s'est arrêté de manière normale.

PORTS : liste des ports d'écoute (ici aucun), nous verrons cela plus loin.

NAMES : nom du conteneur aléatoire (*sad_jennings*), il est possible de définir un nom personnalisé avec le paramètre *--name*.



Si on lance de nouveau le conteneur avec la même commande « *docker run* », ce n'est pas le même conteneur qui est exécuté mais un nouveau conteneur qui est créé. Pour relancer un conteneur, Il faut utiliser la commande « *docker start* » (voir ci-après).

docker run ubuntu

docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
25366713428d	ubuntu	"/bin/bash"	7 seconds ago	Exited (0) 5 seconds ago		sad_jennings
e85b97b93c8a	ubuntu	"/bin/bash"	9 seconds ago	Exited (0) 7 seconds ago		jolly_wilson

Pour lancer un conteneur en lui faisant exécuter une commande spécifique (et non seulement celle prévue par défaut) :

docker run ubuntu cat /etc/lsb-release

```
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04 LTS"
```

La commande *run* a lancé un conteneur à partir de l'image ubuntu et a exécuté dessus la commande *cat* pour afficher le fichier de version d'ubuntu.

Nous pouvons voir qu'un troisième conteneur non actif existe :

docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
25366713428d	ubuntu	"/bin/bash"	7 seconds ago	Exited (0) 5 seconds ago		sad_jennings
e85b97b93c8a	ubuntu	"/bin/bash"	9 seconds ago	Exited (0) 7 seconds ago		jolly_wilson
fe1a6bd9fb64	ubuntu	"cat /etc/lsb-release"	7 seconds ago	Exited (0) 5 seconds ago		gracious_alle



Nous pouvons nous retrouver très facilement avec de multiples conteneurs totalement inutiles qu'il faut supprimer.

Supprimer un conteneur

```
docker rm <id conteneur> ou docker rm <nom du conteneur>
```

Par exemple : **docker rm jolly_wilson**

docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
25366713428d	ubuntu	"/bin/bash"	7 seconds ago	Exited (0) 5 seconds ago		sad_jennings
fe1a6bd9fb64	ubuntu	"cat /etc/lsb-release"	7 seconds ago	Exited (0) 5 seconds ago		gracious_alle



Seul un conteneur avec un statut « exited » peut être supprimé.

Lancer un conteneur en « programmant » sa suppression

Si, par exemple, le conteneur a pour seul vocation de lancer une commande, il est inutile de le conserver sur la machine hôte et il peut être supprimé dès qu'il a rempli sa fonction : cela peut se faire avec l'option « --rm » :

docker run --rm ubuntu cat /etc/lsb-release

```
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04 LTS"
```

Aucun nouveau conteneur n'apparaît avec la commande **docker ps -a**



Tant qu'un conteneur n'est pas supprimé, son environnement est sauvegardé et il peut être relancé.

Relancer un conteneur

```
docker start <id conteneur> ou docker start <nom du conteneur>
```



Par défaut, le conteneur est relancé en arrière plan, donc rien ne s'affiche, pour le lancer au premier plan, il faut utiliser l'option -a : **docker start -a gracious_alle**.

Document 5 - Lancement d'un conteneur en interactif

```
docker run -- name servUbuntu -it ubuntu
```

```
root@f37761aae4b0:/#
```

Le paramètre --name (facultatif) permet de nommer le conteneur pour plus de confort dans sa manipulation future.

Le paramètre « -i » permet d'activer le mode interactif, qui récupère l'entrée standard (clavier) et redirige tous les messages sur la sortie standard (écran).

Le paramètre « -t » permet d'avoir un pseudo-terminal pour exécuter des commandes dans le conteneur une fois lancé.



Nous pouvons remarquer que nous sommes dans un shell spécifique au conteneur car le prompt contient son identifiant (f37761aae4b0). Depuis une autre console, la commande listant les conteneurs montre que ce conteneur est en cours d'exécution (car nous sommes connectés en mode interactif) : **status à UP**.

docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f37761aae4b0	ubuntu	"/bin/bash"	10 minutes ago	Up 10 minutes		servUbuntu



Il est donc possible d'interagir avec le système pour le mettre à jour par exemple et y installer les applications que l'on veut.

```
root@f37761aae4b0:/#apt update
Get:1 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
...
Fetched 24.9 MB in 8s (3041 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
```

```
root@f37761aae4b0:/#apt install openssh-server
```

On crée également un utilisateur pour pouvoir se connecter au service ultérieurement.

```
root@f37761aae4b0:/#adduser user
```

Quand le travail est terminé, on sort du conteneur par la commande `exit` qui sort du shell courant, et donc du conteneur puisque celui-ci n'a pour but que de faire tourner cet unique processus : le statut du dernier conteneur passe donc à « exited ».



Ce qui a été fait est sauvegardé tant que le conteneur n'est pas supprimé ⇒ même s'il est arrêté, le conteneur contient tous les fichiers ajoutés (typiquement ici le paquet openssh-server et le nouvel utilisateur)

Pour le réactiver et y accéder de nouveau en interactif :

1. `docker start servUbuntu`
2. `docker attach servUbuntu`

Pour voir les modifications apportées au conteneur (la commande diff)

```
docker diff servUbuntu
```

```
C /sbin
A /sbin/runlevel
...
```

A pour ajout, C pour modification et D pour suppression

Pour voir les processus lancés dans le conteneur

```
docker top servUbuntu
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	12541	12521	0	14:26	pts/0	00:00:00	/bin/bash

Nous pouvons constater que le service « ssh » n'est pas lancé.

```
root@f37761aae4b0:/# service ssh start
* Starting OpenBSD Secure Shell server sshd
```

docker top servUbuntu

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	12541	12521	0	14:26	pts/0	00:00:00	/bin/bash
root	21797	12541	0	14:36	?	00:00:00	/usr/sbin/sshd

Document 6 - Création d'une nouvelle image à partir d'un conteneur

Il existe deux méthodes pour créer des images Docker. La première (que nous n'aborderons pas ici) consiste à écrire un *dockerfile* contenant une liste de commandes permettant de créer l'image.

La deuxième utilise la commande « commit » :

```
docker commit <nom conteneur> <nom image>
```



Le seul paramètre obligatoire ici est le nom ou l'identifiant du container.

La commande ci-dessus n'utilise pas les nombreuses options disponibles.

Le nom de la future image est libre mais nous pouvons utiliser les règles d'usage en préfixant le nom et en y associant un tag pour identifier la modification plus simplement parmi les autres.

docker commit servUbuntu ar/ubuntu:ssh

```
sha256:0dbdf689c2f5438993494a1f1dbc610f5b930c4882a653a6d94912006be0f63a
```

La commande produit un hash permettant au système d'identifier de manière unique la nouvelle image.

Nous constatons qu'une nouvelle image est disponible :

docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ar/ubuntu	ssh	0dbdf689c2f5	About a minute ago	234MB
ubuntu	xenial	0b1edfbffd27	7 days ago	113MB
ubuntu	latest	452a96d81c30	7 days ago	79.6MB



C'est cette nouvelle image (et non plus l'image téléchargée du Hub) que nous allons exploiter à l'avenir (Il est désormais possible de lancer plusieurs serveurs SSH isolés en rendant ce service accessible (voir paragraphe suivant) ⇒ Nous pouvons donc supprimer maintenant le conteneur (après l'avoir arrêté).

Sauvegarder une image en local

Il peut être utile de sauvegarder une image localement à des fins d'exploitation sur un autre PC (en attendant de publier notre image sur le Hub officiel ou un autre Hub) :

```
docker save <image> > <nom_fichier.tar>
```

Par exemple : **docker save ar/ubuntu:ssh > serv_ubuntu-ssh.tar**

Restaurer depuis un conteneur en local

```
docker load -i <nom_fichier.tar>
```

Par exemple : **docker load -i serv_ubuntu-ssh.tar**

Document 7 - Comment rendre accessible un service (conteneur en arrière plan)

Pour rendre un service accessible, le **conteneur doit être lancé en arrière plan**. Docker utilise la technique classique du **mappage de port** : un port de l'hôte va être redirigé vers un port du container.

Docker dispose pour cela d'une interface réseau sur la machine hôte :

ip a

```
...
docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
        link/ether 02:42:f6:de:8b:5b brd ff:ff:ff:ff:ff:ff
        inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
...
```

Une commande possible est la suivante :

```
docker run -d -p <IP:port-hôte:port-conteneur> --name <nom conteneur> <image> COMMANDE
```

Le paramètre permettant le mappage est **-p <IP:port-hôte:port-conteneur>** :

- si l'adresse IP réelle de l'hôte n'est pas indiquée, le système n'écouterait que sur *localhost* ;
- si le port de l'hôte à mapper n'est pas indiqué, Docker en choisirait un automatiquement.

Le principe est ensuite de lancer le service en même temps que le conteneur sachant que :

- le service doit être lancé en premier plan (par exemple `/usr/sbin/sshd -D` pour le service ssh)
- le conteneur sera lancé en arrière plan (sans laisser de console ouverte) en mode « détaché » : c'est l'**option « -d »** qui permet cela.

```
docker run -d -p 192.168.0.120 :22222:22 --name servssh ar/ubuntu:ssh /usr/sbin/sshd -D
055fec843da0d633a6b1f3777c1f3b192f0e31ee2d02ba962f81e155a052f3a3
```

HÔTE DEBIAN

Adresse IP de l'hôte : 192.168.0.120
Port ouvert sur l'hôte avec le « -p »

Conteneur Docker (servssh)

`/usr/sbin/sshd -D`

:22222 → :22



Pour accéder au docker en ssh, à partir de n'importe quel poste :
ssh user@192.168.0.120 -p 22222

```
The authenticity of host '[192.168.0.120]:22222 ([192.168.0.120]:22222)' can't be established.
ECDSA key fingerprint is SHA256:zQ1fPQmICgv7LU/vI/Cz+dOulm2JmfwPI5tD9XbZtCA.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:22222' (ECDSA) to the list of known hosts.
user@192.168.0.120's password:
...
```

Nous pouvons constater via la commande « `docker ps` » que l'attribut PORT est maintenant rempli :

...	PORTS	NAMES
...	0.0.0.0:22222→22/tcp	servssh



L'option « -p » peut être multiplié autant de fois que nécessaire (si par exemple il y a plusieurs services à exposer dans le conteneur).

Afficher les logs du conteneur

Il s'agit d'une commande utile notamment si la création du conteneur se passe mal et si l'accès en ssh n'est pas possible (cette commande ne renvoie rien dans ce cas si tout se passe bien).

```
docker logs servssh
```

Document 8 – Récapitulatif des commandes de base

La commande docker --help permet de voir les nombreuses commandes disponibles.

Obtenir des informations sur le système Docker installé

docker info

Chercher une image sur le Hub officiel

docker search <mot_clé>

Télécharger une image depuis le Hub officiel

docker pull <nom image>

nom_image peut comporter un tag comme debian:jessie

Lister les images disponibles

docker images

Supprimer une image

docker rmi <nom_image>

Créer un conteneur

docker run [OPTIONS] <nom image> [COMMANDE]

Quelques paramètres de la commande run utilisés dans cette activité :

-t : fourni un terminal au docker.

-i : permet d'écrire dans le conteneur (couplé à -t).

-d : exécute le conteneur en arrière plan.

-p : permet de mapper un port sur le conteneur vers un port sur l'hôte

--name : Donne un nom au conteneur

--rm : supprime un conteneur dès qu'il a rempli sa fonction (utile, par exemple, si le conteneur a pour seule vocation de lancer une commande)



Dans ce qui suit <conteneur> peut être remplacé soit par l'id du conteneur soit par le nom donné au conteneur.

Arrêter un conteneur

docker stop <conteneur> ou docker kill <conteneur>

Réactiver un conteneur et y accéder en interactif :

- docker start <conteneur>
- docker attach

Supprimer un conteneur

docker rm <conteneur>

Le conteneur doit avoir été stoppé sinon il faut utiliser l'option « -f » : docker rm -f <conteneur>

Lister les conteneurs démarrés

docker ps

-a pour afficher tous les conteneurs

-q pour n'afficher que les « id »

Supprimer rapidement tous les conteneurs actifs

docker rm -f \$(docker ps -aq)

Mettre en pause un conteneur

docker pause <conteneur>

Sortir de la pause un conteneur

`docker unpause <conteneur>`

Afficher les processus en cours d'un conteneur

`docker top <conteneur>`

Afficher les statistiques d'un ou des conteneurs (CPU, mémoire, etc)

`docker stats [<conteneur>]`

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
4cfc7c35645d	servWeb	0.04%	11.21MiB / 512MiB	2.19%	23.6kB / 21.8kB	606kB / 0B	8
d8d73eb9780e	servSQL	0.19%	89.83MiB / 512MiB	17.55%	9.64kB / 14.1kB	54.2MB / 142MB	30

Exécuter une commande dans un conteneur existant

`docker exec [OPTIONS] <conteneur> COMMAND [ARG...]`

Docker exec est utilisé pour lancer des commandes dans un conteneur qui tourne en mode détaché. Par exemple, exécuter un bash en attachant le container nommé test : `docker exec -it <nom conteneur> bash`

Afficher les logs d'un conteneur

`docker logs <conteneur>`

Voir en temps réel les évènements sur un conteneur

`docker events <conteneur>`

Connaître la configuration et les éléments d'un conteneur

`docker inspect <conteneur>`

Créer une image à partir d'un conteneur

`docker commit <conteneur> <nom future image>`

Sauvegarder une image en local

`docker save <image> > <nom_fichier.tar>`

Restaurer depuis un conteneur en local

`docker load -i <nom_fichier.tar>`

Voir les différences apportées par rapport à une image d'origine :

`docker diff <conteneur>`

Renommer un conteneur

`docker rename <ancien nom conteneur> <nouveau nom conteneur>`

Pour faire rapidement le ménage (comme la suppression des conteneurs non actifs), voir ici : <http://damiengustave.fr/nettoyer-docker/>