

DÉPLOIEMENT D'UN SERVICE AVEC DOCKER

ACTIVITÉ 1 – INSTALLATION DE DOCKER ET PREMIER PAS

Le document « récapitulatif des commandes Docker » vient en complément.

Liens :

- Lien officiel de l'installation : <https://docs.docker.com/engine/install/debian/>
- Lien vers une formation sur Docker complète mais très simple :
- <https://blog.microlinux.fr/formation-docker/>

L'objectif est ici de :

- se familiariser avec le fonctionnement de Docker ;
- comprendre les notions d'images et de conteneurs.

Nous allons déployer une distribution légère d'Ubuntu et rendre accessible le service SSH.



Vous allez découvrir un certain nombre de commandes (il en existe beaucoup d'autres), il est nécessaire de ne pas se contenter de les saisir, mais de les comprendre. Vous serez bientôt en autonomie.

SOMMAIRE

A. Installation de Docker.....	2
B. Vérification du bon fonctionnement de Docker.....	3
C. Premier pas avec Docker.....	4
1. Gérer les images.....	4
1.1 Interroger le « registre Docker » (Docker registry) en ligne de commandes.....	4
1.2 Télécharger une image.....	6
1.3 Connaître les images disponibles sur le système.....	6
1.4 Supprimer une image.....	6
2. Gérer les conteneurs.....	7
2.1 Lancer un conteneur.....	7
2.2 Lister les conteneurs en cours d'exécution (actifs).....	7
2.3 Lister tous les conteneurs qu'ils soient actifs ou non.....	7
2.4 Supprimer un conteneur.....	8
2.5 Lancer un conteneur en « programmant » sa suppression.....	9
2.6 Relancer un conteneur.....	9
2.7 Lancer un conteneur en interactif.....	9
2.8 Pour voir les processus lancés dans un conteneur.....	10
3. Créer d'une nouvelle image à partir d'un conteneur.....	11
3.1 Sauvegarder une image en local.....	12
3.2 Restaurer depuis une image locale.....	12
4. Rendre accessible un service.....	12

Vous devez faire toutes les manipulations, notamment celles précédées par :



A. 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 update
apt install ca-certificates curl gnupg
```

Importation de la clé du dépôt docker

```
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
chmod a+r /etc/apt/keyrings/docker.gpg
```

Cette série de commandes télécharge (commande curl), déchiffre et décompresse (l'option --dearmor) le fichier GPG de clé, puis le stocke dans le répertoire /etc/apt/keyrings. Ensuite, elle ajoute des permissions de lecture à tous les utilisateurs sur ce fichier GPG.

Dans les commandes ci-après, des variables sont introduites, ce qui fait qu'elles sont valables pour n'importe quelle distribution sur Linux.

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

```
echo \
"deb [arch="$(dpkg --print-architecture)"
signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \
"$(. /etc/os-release && echo "$VERSION_CODENAME)" stable" | \
tee /etc/apt/sources.list.d/docker.list > /dev/null
apt update
```

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

La variable `$(dpkg --print-architecture)` renvoie l'architecture du processeur (ici amd64).

Installation de Docker

```
apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin
```

Docker est démarré (et est configuré pour démarrer automatiquement au lancement de la machine).

```
systemctl status docker
```

```
docker.service - Docker Application Container Engine
Loaded: loaded (/lib/systemd/system/docker.service; enabled; preset: enabled)
Active: active (running) since Tue 2023-08-08 16:26:27 CEST; 48s ago
TriggeredBy: ● docker.socket
Docs: https://docs.docker.com
Main PID: 143 (dockerd)
...
```

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

```
gpasswd -a votre_login docker
```

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

B. VÉRIFICATION DU BON FONCTIONNEMENT DE DOCKER

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

```
user@servDockerAR:~$ docker version
```

Client: Docker Engine - Community	Server: Docker Engine - Community
Version: 24.0.5	Engine: 24.0.5
API version: 1.43	Version: 24.0.5
Go version: go1.20.6	API version: 1.43
Git commit: ced0996	Go version: go1.20.6
Built: Fri Jul 21 20:35:35 2023	Git commit: a61e2b4
OS/Arch: linux/amd64	Built: Fri Jul 21 20:35:35 2023
Context: default	OS/Arch: linux/amd64
	Experimental: false

Lancement d'un conteneur de test qui sert uniquement à afficher "Hello world" (l'option `--rm` supprime le conteneur juste après l'affichage).

Le lancement d'un conteneur Docker se réalise avec la commande « `docker run` ». 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.

```
user@servDockerAR:~$ docker run --rm hello-world
```

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:926fac19d22aa2d60f1a276b66a20eb765fbee2db5dbdaafeb456ad8ce81598
Status: Downloaded newer image for hello-world:latest
```

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

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
`$ docker run -it ubuntu bash`

Share images, automate workflows, and more with a free Docker ID:
<https://hub.docker.com/>

For more examples and ideas, visit: <https://docs.docker.com/get-started/>

 Cette commande renvoie également une série d'informations qui sont expliquées dans les activités suivantes.

C. PREMIER PAS AVEC DOCKER

1. GÉRER LES 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.

Le « **Docker registry** » (**Hub ou registre 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 images officielles (images créées par l'équipe de Docker), des images certifiées par l'équipe de Docker 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.

1.1 Interroger le « registre Docker » (Docker registry) en ligne de commandes

```
docker search <mot clé>
```

```
user@servDockerAR:~$ docker search ubuntu
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys...	16255	[OK]	
...				
ubuntu-upstart	DEPRECATED, as is Upstart (find other proces...	115	[OK]	
ubuntu/nginx	Nginx, a high-performance reverse proxy & we...	97		
ubuntu/cortex	Cortex provides storage for Prometheus. Long...	4		

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 ubuntu) ne sont pas précédés par le nom de l'utilisateur qui l'a créée (comme ubuntu/nginx).

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 par l'équipe de Docker.

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

L'équivalent de la recherche de la page précédente via le Web (<https://hub.docker.com>) donne l'image ci-dessous :

The screenshot shows the Docker Hub search results for 'ubuntu'. The search bar at the top contains 'ubuntu'. The results show two images: 'ubuntu' (DOCKER OFFICIAL IMAGE) and 'websphere-liberty' (DOCKER OFFICIAL IMAGE). The 'ubuntu' image has 1B+ pulls and 10K+ stars. The 'websphere-liberty' image has 10M+ pulls and 296 stars. A blue arrow points to the 'ubuntu' image with the text 'Le nombre de fois où l'image a été téléchargée'.

Un clic sur la « carte » permet d'avoir des informations sur les images (plus ou moins détaillées). On pourra par exemple trouver les versions des applications ou la commande typique pour lancer un conteneur.

The screenshot shows the Docker Hub page for the 'ubuntu' image. The page includes a 'Quick reference' section with information about the image's maintenance and where to get help. It also features a 'Supported tags and respective Dockerfile links' section with a list of tags and their corresponding Dockerfile links. A blue arrow points to the 'latest' tag in the list. The 'Recent Tags' section shows a list of recent tags. The 'About Official Images' section provides information about the Docker Official Images and why they are used.

Docker permet d'ajouter un tag sur une image. Par défaut c'est la dernière image (latest) qui est téléchargée. On voit ici que « latest » correspond à la version 22.04 (jammy).

La commande « `docker pull ubuntu` » est équivalente à « `docker pull ubuntu:latest` » qui est également équivalente à la commande « `docker pull ubuntu:22.04` ».

Si vous voulez lancer une autre version (par exemple la version 23.04), la commande est `docker pull ubuntu:23.04` ou `docker pull ubuntu:lunar`, etc.

1.2 Télécharger une image

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

```
user@servDockerAR:~$ docker pull ubuntu
```

```
Using default tag: latest
latest: Pulling from library/ubuntu
3153aa388d02: Pull complete
Digest: sha256:0bced47fffa3361afa981854fcabcd4577cd43cebbb808cea2b1f33a3dd7f508
Status: Downloaded newer image for ubuntu:latest
docker.io/library/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. Par exemple, pour récupérer la version *lunar*, il faut utiliser la commande suivante :

```
user@servDockerAR:~$ docker pull ubuntu:lunar
```

```
lunar: Pulling from library/ubuntu
03b3dca73f87: Pull complete
Digest: sha256:7a520eeb6c18bc6d32a21bb7edcf673a7830813c169645d51c949cecb62387d0
Status: Downloaded newer image for ubuntu:lunar
docker.io/library/ubuntu:lunar
```

1.3 Connaître les images disponibles sur le système

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

```
docker images
```

```
user@servDockerAR:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	lunar	24881e92d648	8 days ago	70.3MB
ubuntu	latest	5a81c4b8502e	5 weeks ago	77.8MB
hello-world	latest	9c7a54a9a43c	3 months ago	13.3kB

1.4 Supprimer une image

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



Il n'est pas possible de supprimer une image si un conteneur est associé même si ce dernier n'est plus en cours d'exécution (voir partie suivante), comme cela est le cas pour hello-world : dans ce cas, il faut utiliser l'option « f ».

```
user@servDockerAR:~$ docker rmi hello-world
```

```
Error response from daemon: conflict: unable to remove repository reference
"hello-world" (must force) - container be7839d9440e is using its referenced
image 9c7a54a9a43c
```


```
user@servDockerAR:~$ docker rmi hello-world -f
```

```
Untagged: hello-world:latest
Untagged: hello-
world@sha256:926fac19d22aa2d60f1a276b66a20eb765fbee2db5dbdaafeb456ad8ce81598
Deleted:
sha256:9c7a54a9a43cca047013b82af109fe963fde787f63f9e016fdc3384500c2823d
```



L'option -f est à utiliser avec précaution.

2. GÉRER LES 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.

2.1 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 point 2.7) ;
- **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 point 4) ;
- **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 à partir de l'image `ubuntu` :

```
user@servDockerAR:~$ docker run ubuntu
```

 Mais rien ne semble s'être passé !

2.2 Lister les conteneurs en cours d'exécution (actifs)

```
docker ps
```

```
user@servDockerAR:~$ 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`.

2.3 Lister tous les conteneurs qu'ils soient actifs ou non

```
docker ps -a
```

```
user@servDockerAR:~$ 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 un 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 plus loin).

```
user@servDockerAR:~$ docker run ubuntu
```

```
user@servDockerAR:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	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) :

```
user@servDockerAR:~$ docker run ubuntu cat /etc/lsb-release
```

```
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=22.04
DISTRIB_CODENAME=jammy
DISTRIB_DESCRIPTION="Ubuntu 22.04.2 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 (avec dans la colonne « `COMMAND` » « `cat /etc/lsb-release` » ;

```
user@servDockerAR:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	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.

2.4 Supprimer un conteneur

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

```
user@servDockerAR:~$ docker rm jolly_wilson (le nom du conteneur est bien sûr à adapter)
```

```
user@servDockerAR:~$ docker ps -a (ne fait plus apparaître le conteneur supprimé)
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	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. Si un conteneur est actif, il faut soit le stopper avant soit utiliser l'option -f : `docker rm -f <conteneur>`

2.5 Lancer un conteneur en « programmant » sa suppression

Si, par exemple, le conteneur a pour seule 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, comme nous l'avons fait avec le conteneur « hello-world » : cela peut se faire avec l'option « --rm » :

```
user@servDockerAR:~$ 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.

2.6 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.

```
user@servDockerAR:~$ docker start -a gracious_alle
```

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



Avant de continuer, nous allons supprimer le dernier conteneur non actif créé :
user@servDockerAR:~\$ `docker rm sad_jennings`

2.7 Lancer un conteneur en interactif


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

```
root@1228a3c367ca:/#
```

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 (1228a3c367ca). **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**.

 user@servDockerAR:~\$ docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
1228a3c367ca	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 et services** que l'on veut.

 root@1228a3c367ca:/#apt update

```
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
...
Fetched 26.3 MB in 40s (651 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
4 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

 root@1228a3c367ca :/#apt dist-upgrade


 root@1228a3c367ca:/#apt install openssh-server

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

 root@1228a3c367ca:/#adduser user

Vous pouvez mettre « ubuntuUser » en mot de passe (histoire de vous en rappeler).

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 les mises à jour, le paquet `openssh-server` et ses dépendances ainsi que le nouvel utilisateur).

Pour le réactiver et y accéder de nouveau en interactif :

1. `docker start servUbuntu`
2. `docker attach servUbuntu` (pour accéder au pseudo-terminal d'un conteneur démarré)

 user@servDockerAR:~\$ docker start servUbuntu

2.8 Pour voir les processus lancés dans un conteneur

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

 user@servDockerAR:~\$ 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.

```
user@servDockerAR:~$ docker attach servUbuntu
```

```
root@1228a3c367ca:/# service ssh start
* Starting OpenBSD Secure Shell server sshd
```

Dans une autre console

```
user@servDockerAR:~$ 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

 Mais comment rendre ce service accessible ?

Pour rendre le service disponible (ici ssh) et donc accéder au conteneur via ssh, il est nécessaire de lancer un nouveau conteneur avec un certain nombre d'options (voir 1.5), mais nous avons vu que si on relançait le conteneur avec l'image dont on dispose, on repartait forcément de « zéro ». **Il faut donc pouvoir lancer un conteneur à partir de l'image modifiée !** Il faut donc pour cela créer une nouvelle image intégrant toutes les modifications opérées.

3. CRÉER 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 mais dans l'activité 3) consiste à écrire un *Dockerfile* contenant une liste de commandes permettant de créer l'image.

La deuxième utilise la commande « commit » :

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

ou

```
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.

Dans la commande qui suit (le nom préfixé – ici aporaf –) doit être adapté.

```
user@servDockerAR:~$ docker commit servUbuntu aporaf/ubuntu:ssh
```


```
sha256:d6770360588fbb5a1dd773aa235406dc10d942e251f0963f27379e451a3354fe
```

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

Nous constatons qu'une nouvelle image est disponible :

```
user@servDockerAR:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
aporaf/ubuntu	ssh	d6770360588f	47 seconds ago	230MB
ubuntu	lunar	24881e92d648	9 days ago	70.3MB
ubuntu	latest	5a81c4b8502e	6 weeks ago	77.8MB
hello-world	latest	9c7a54a9a43c	3 months ago	13.3kB

 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 ou avoir saisi la commande `exit` si on est encore en mode interactif).

3.1 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 (voir activité suivante) :

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


 user@servDockerAR:~\$ docker save aporaf/ubuntu:ssh > serv_ubuntu-ssh.tar

3.2 Restaurer depuis une image locale

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

Par exemple :

 user@servDockerAR:~\$ docker load -i serv_ubuntu-ssh.tar

 Dans les manipulations qui suivent, nous utiliserons la nouvelle image créée : `aporaf/ubuntu :ssh` (vous devez bien sûr utiliser votre propre nom d'image)

 Vous pouvez également supprimer tous les conteneurs.

4. RENDRE ACCESSIBLE UN SERVICE

Pour rendre un service accessible, **le conteneur doit être lancé en arrière plan**. Docker utilise la technique 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 :

 user@servDockerAR:~\$ 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> <nom_image> COMMANDE
```


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

- si l'adresse IP réelle de l'hôte n'est pas indiquée, le système n'écouterait que sur *localhost* ;

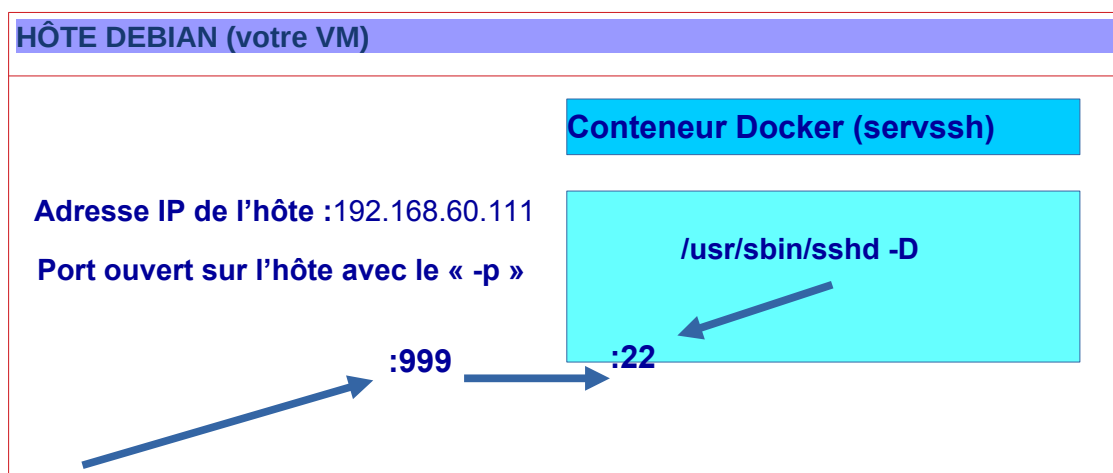
- si le port de l'hôte à mapper n'est pas indiqué, Docker en choisira un automatiquement.

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.

 `user@servDockerAR:~$ docker run -d -p 192.168.60.111:999:22 --name servssh aporaf/ubuntu:ssh /usr/sbin/sshd -D`

6e5414e11795030fe0d2be13577adb12be9908a5922e9a1e75a9ac6b549e5803



Pour accéder au docker en ssh, à partir de n'importe quel poste

—(apollonie@apoXPS13kali)-[~]

```
└─$ ssh user@192.168.60.111 -p 999
...
Warning: Permanently added '[192.168.60.111]:999' (ED25519) to the list of
known hosts.
user@192.168.60.111's password:
...
user@6e5414e11795:~$
```

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

...	PORTS	NAMES
...	0.0.0.0:999→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

`docker logs servssh`

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).