

# Activité : découvrir Docker

Pour cette découverte de Docker, vous pouvez :

- **installer** Docker sur votre ordinateur ou
- utiliser l'environnement de découverte appelé **Play with Docker (PWD)** : <https://labs.play-with-docker.com/>

## Un premier conteneur

### Lancement sans option d'un conteneur

Cette commande lance le classique Hello world ! :

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Already exists
Digest: sha256:305243c734571da2d100c8c8b3c3167a098cab6049c9a5b066b6021a60fcb966
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/>

Ce qu'il faut retenir de l'exécution de cette commande :

- La première ligne indique que l'image du conteneur **'hello-world:latest'** n'est pas disponible **localement** c'est-à-dire sur votre ordinateur. De plus, comme aucune version du conteneur n'est précisée, c'est la dernière version disponible, la **latest** qui est recherchée.
- L'image recherchée est alors **tirée (pull)** de Docker Hub depuis Internet.
- Une explication des étapes réalisées est indiquée : le client Docker, la session de terminal de votre ordinateur, contacte le **daemon** Docker de votre ordinateur (le service Docker) qui tire l'image du conteneur depuis Docker Hub, **crée un conteneur, exécute** ce qui est prévu dans l'image du conteneur (la production du texte à afficher) et envoie cette sortie texte vers le client Docker votre terminal.
- Le conteneur est ensuite **arrêté automatiquement** car le traitement prévu s'est effectué, l'affichage du texte et rien d'autre.

## Afficher les images Docker présentes sur la machine

```
$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
hello-world latest d2c94e258dcb 19 months ago 13.3kB
```

Vous pouvez vérifier la présence de l'image qui a été tirée (pull) de Docker Hub et stockée sur votre serveur. Docker gère un **cache** des images sur la machine. Lors du prochain démarrage d'une instance de conteneur sur la même image il n'y aura pas de nouveau téléchargement sauf si elle a été modifiée entre-temps.

### INFORMATION

Pour en savoir plus sur une image il suffit de faire une recherche sur Docker Hub. Pour le conteneur hello-world : [https://hub.docker.com/\\_/hello-world](https://hub.docker.com/_/hello-world)

### Explication des tags

- Le **TAG** associé à l'image hello-world est latest et est le marqueur de version de l'image.
- **IMAGE ID** est l'identifiant unique de l'image.
- **CREATED** est la date de création de l'image publiée sur le site Docker Hub.
- **SIZE** est sa taille virtuelle, c'est-à-dire de la couche logicielle téléchargée.

## Lister les conteneurs actifs

L'image du conteneur hello-world est présente sur votre serveur mais après le lancement d'un conteneur basé sur cette image, vous avez retrouvé l'invite de commandes de votre serveur. Le conteneur lancé s'est ensuite arrêté automatiquement. Mais existe-il toujours ?

Pour constater que le conteneur lancé est bien arrêté vous pouvez visualiser les conteneurs actifs :

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
```

Le conteneur hello-world n'est effectivement **plus actif**.

## Lister tous les conteneurs y compris ceux qui sont inactifs

Relancez le conteneur hello-world. Vous pouvez à nouveau vérifier que celui n'est pas actif après l'affiche du texte prévu.

Visualisez maintenant tous les conteneurs créés, qu'ils soient actifs ou non :

```
btssio@ubuntudocker:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
9bbd425c86f4   hello-world "/hello" 2 seconds Exited (0) 1 s ago   vibrant_goldberg
ae50ecb3a66d   hello-world "/hello" 49 seconds Exited (0) 47 s ago   quirky_elbakyan
btssio@ubuntudocker:~$
```

Les 2 conteneurs basés sur la même image hello-world, sont bien présents mais arrêtés sans erreur normale (code de sortie 0) comme le montre le champ STATUS : Exited (0). Le premier conteneur s'est arrêté il y a 1 seconde, le 2ème il y a 47 secondes. Chaque conteneur : \* possède un identifiant de conteneur CONTAINER ID, \* et un nom, NAMES, qui a été généré par le démon Docker. La colonne COMMAND nous renseigne sur le processus qui avait été lancé, à savoir un script hello.

==== Comprendre le système de couche logicielle des images Docker =====

==== Récupérer une image dans sa dernière version =====

```
<code shell> $ docker pull ubuntu
```

Using default tag: latest latest: Pulling from library/ubuntu de44b265507a: Pull complete Digest: sha256:80dd3c3b9c6cecb9f1667e9290b3bc61b78c2678c02cbdae5f0fea92cc6734ab Status: Downloaded newer image for ubuntu:latest </code>

Le TAG est latest par défaut si vous ne précisez pas la version du conteneur voulu. Cette image ubuntu est une image officielle ; elle n'est pas préfixée et vous pouvez avoir des informations sur Docker hub. Lien : <https://hub.docker.com/>



# ubuntu ☆

## Docker Official Images

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

↓ 10M+

- Container
- Linux
- x86-64
- ARM
- ARM 64
- 386
- PowerPC 64 LE
- IBM Z
- Base Images
- Operating Systems
- Official Image

Linux - x86-64 ( latest )

Copy and paste to pull this image

```
docker pull ubuntu
```

[View Available Tags](#)

Cette image est basée sur quatre couches logicielles : <code shell> ... f476d66f5408: Pull complete 8882c27f669e: Pull complete

d9af21273955: Pull complete f5029279ec12: Pull complete ... </code>

### INFORMATION

Le chargement de ce conteneur **ubuntu** et des couches logicielles nécessaires a lieu **une fois pour toutes**. Tous les lancements de conteneur suivant se baseront sur cette image, y compris - et c'est une des grands avantages de l'architecture par couches - toutes les images qui se trouvent sur Docker Hub et qui ont été construites à partir de cette image ubuntu. Et il existe sur Docker Hub de nombreuses images basées sur Ubuntu qui est une distribution de base fréquemment utilisée pour les images Docker.

Lancez un conteneur Ubuntu. Comme précédemment, le conteneur est ensuite arrêté mais existe bien. La colonne COMMAND indique que le processus qui a été lancé est cette fois-ci un shell Bash. <code shell> btssio@ubuntudocker:~\$ docker run ubuntu btssio@ubuntudocker:~\$ docker ps -a CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES b5ec671c6d11 ubuntu "/bin/bash" 21 s ago Exited ... laughing\_ride ... </code> Vous pouvez prendre connaissance de la configuration du conteneur sur Docker Hub en consultant son fichier **Dockerfile**.

### IMPORTANT

Le fichier **Dockerfile** associé à une image Docker, **décrit exactement** comment l'image a été **construite**. Quand vous recherchez sur Docker Hub une image qui doit répondre à votre besoin et qui a été publiée par un membre de la communauté, ce fichier vous permet de savoir ce que contient l'image afin de vous permettre de faire un choix éclairé parmi la profusion d'images possibles. Une bonne pratique consiste donc :

- à **choisir une image officielle** si celle-ci répond à votre besoin,
- et, si cela n'est pas le cas, à choisir une image de la communauté en regardant si elle est souvent utilisée (sa **popularité**), sa **documentation** mais aussi son fichier **Dockerfile** pour savoir exactement ce qu'elle contient.

Sur Docker Hub plusieurs Dockerfile sont indiqués pour l'image officielle ubuntu :

## Supported tags and respective Dockerfile links

- [18.04](#) , [bionic-20210827](#) , [bionic](#)
- [20.04](#) , [focal-20210827](#) , [focal](#) , [latest](#)
- [21.04](#) , [hirsute-20210825](#) , [hirsute](#) , [rolling](#)
- [21.10](#) , [impish-20210827](#) , [impish](#) , [devel](#)
- [14.04](#) , [trusty-20191217](#) , [trusty](#)
- [16.04](#) , [xenial-20210804](#) , [xenial](#)

Télécharger l'image Docker ubuntu :bionic-20210827 <code shell> btssio@ubuntudocker:~\$ docker pull ubuntu:bionic-20210827 bionic-20210827: Pulling from library/ubuntu e4ca327ec0e7: Pull complete Digest: sha256:9bc830af2bef73276515a29aa896eedfa7bdf4bdb5c1063b4c457a4bbb8cd79 Status: Downloaded newer image for ubuntu:bionic-20210827 docker.io/library/ubuntu:bionic-20210827 btssio@ubuntudocker:~\$ docker images REPOSITORY TAG IMAGE ID CREATED SIZE Ubuntu bionic-20210827 d131e0fa2585 11 days ago 102MB ubuntu latest d131e0fa2585 11 days ago 102MB btssio@ubuntudocker:~\$ </code> Il s'agit exactement de la même image comme le montre le champ IMAGE ID. Vous pouvez donc télécharger une version plus ancienne d'Ubuntu si votre projet porte sur une version bien précise de cet OS et faire cohabiter ensuite des conteneurs utilisant des versions différentes de système d'exploitation ou de paquets logiciels. Le champ SIZE indique la taille virtuelle de l'image, c'est à dire la taille de la couche téléchargée qui est plus important que celle du conteneur hello-world car l'image ubuntu ne s'appuie pas sur une couche préexistante. Pour connaître comment cette image a été construite, cliquez sur le lien du Dockerfile : <code shell> FROM scratch ADD ubuntu-bionic-core-cloudimg-amd64-root.tar.gz / ... # overwrite this with 'CMD []' in a dependent Dockerfile CMD ["/bin/bash"] </code> La première ligne - FROM scratch - indique que cette image ne se base pas sur une image préexistante. On part de la version de base d'Ubuntu. La dernière ligne - CMD ["/bin/bash"] - indique le processus qui doit être lancé par le conteneur et qui est ici un shell Bash. Sans rentrer dans le détail du contenu du fichier Dockerfile, les différentes actions décrites dans ce fichier se sont traduites par le téléchargement des quatre couches de l'image ubuntu sur votre serveur Ubuntu.

===== Utiliser les conteneurs ===== Le conteneur ubuntu lancé prévoit bien l'exécution d'un processus shell mais il s'est ensuite automatiquement arrêté. En effet, comme rien n'a été précisé lors du lancement pour que vous puissiez utiliser ce shell, le conteneur a considéré que l'action à faire, lancer un shell et rien d'autre, est terminée. Il s'arrête donc. ===== Lancer un conteneur en mode interactif ===== Pour pouvoir réellement interagir et utiliser le conteneur ubuntu grâce à ce processus de shell Bash qui est lancé, il faut : \* lancer le

conteneur en mode interactif pour lui associer une console TTY : paramètre `-i`, \* ouvrir une console pour que vous puissiez saisir des commandes : paramètre `-t`. `<code shell> btssio@ubuntudocker:~$ docker run -i -t ubuntu root@394beb25ab78:/# ls bin dev home lib64 mnt proc run srv tmp var boot etc lib media opt root sbin sys usr root@394beb25ab78:/# cd /root root@394beb25ab78:/# touch docker.txt root@394beb25ab78:/# ls docker.txt root@394beb25ab78:/# </code>` Le lancement du conteneur est toujours aussi rapide et le résultat de la commande est différent puisque vous avez le **shell du conteneur** avec comme nom l'**identifiant** du nouveau conteneur lancé. Vous êtes identifiés comme **root** sur la VM nommée **394beb25ab78** qui est l'identifiant du nouveau conteneur lancé et qui a servi à attribuer un nom à la VM. La commande **ls** montre l'arborescence de la VM et non celle du serveur Ubuntu. Un fichier a été créé dans le dossier **/root**. Tapez **Exit** pour revenir à la machine hôte en sortant du shell et donc du conteneur qui ne faisant que tourner cet unique processus s'arrête.

### IMPORTANT

Toutes les **modifications effectuées** dans le conteneur lancé, comme la création de ce fichier `docker.txt`, ne modifie pas l'image ubuntu qui a servi à sa création. Les modifications ne sont faites que dans une **nouvelle couche** ajoutée par Docker pour gérer les modifications du système de fichiers afin de n'avoir aucun impact sur la couche inférieure, la couche ubuntu. En lançant un nouveau conteneur, vous consterez que le dossier `root` ne contient pas de fichier `docker.txt`.

Pour lancer un conteneur arrêté avec son identifiant ou son nom : `<code shell> btssio@ubuntudocker:~$ docker start IDouson_nom </code>` `==== Autre commande pour visualiser les conteneurs actifs ==== <code shell> btssio@ubuntudocker:~$ docker container ls </code>` `==== Lancer un conteneur en mode détaché ====` Lors du lancement d'un conteneur, on perd l'accès à la console du serveur Ubuntu. Cette commande permet de lancer le conteneur et de retrouver le shell du serveur sans arrêter le conteneur. `<code shell> btssio@ubuntudocker:~$ docker run -i -t -d ubuntu e70a11313849c4c2ce8beebe72935368b909759ab5eb4e2c2f1bcf715c1a2bd1 btssio@ubuntudocker:~$ docker ps CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES e70a11313849 ubuntu "/bin/bash" 3 seconds ago Up 2 s gallant_kilby </code>` On peut aussi écrire les paramètres de la façon suivante : `<code shell> btssio@ubuntudocker:~$ docker run -itd ubuntu </code>` `==== Se connecter à un conteneur lancé ====` Après le lancement d'un conteneur en mode détaché, il peut être nécessaire de pouvoir accéder à la console du conteneur et donc au shell. Voici deux manières de procéder. \* S'attacher à la console du conteneur : **ATTENTION**, en quittant le shell, cela arrête le processus shell et donc le conteneur : `<code shell> btssio@ubuntudocker:~$ docker attach gallant_kilby root@e70a11313849:/# </code>` \* Pour quitter le conteneur mais en le laissant s'exécuter en arrière-plan : `<code> Ctrl + p + q. </code>` \* Pour lancer un processus shell (commande `ls` par exemple) dans un conteneur en cours d'exécution : `<code shell> btssio@ubuntudocker:~$ docker exec -it gallantkilby ls bin boot dev ... btssio@ubuntudocker:~$ </code>` `==== Obtenir la liste des modifications d'un conteneur par rapport à son image de lancement ==== <code shell> btssio@ubuntudocker:~$ docker diff 394beb25ab78 C /root A /root/.bashhistory A /root/docker.txt btssio@ubuntudocker:~$ </code>` Cette commande utilise la syntaxe `diff/patch` en Linux : \* `D` pour les parties d'arborescence supprimées, \* `C` pour les parties d'arborescence créées : le dossier `/root` \* `A` pour les ajouts : le fichier `.bash_history` est créé automatiquement pour stocker l'historique des actions effectuées dans le shell, et le fichier `docker.txt` que vous avez créé manuellement. `==== Conteneurs et variables d'environnement ====` \* Modification du point d'entrée par défaut (action par défaut au lancement du conteneur) : Lancer un conteneur basée sur l'image Ubuntu qui liste les variables d'environnement présentes dans l'image. `<code shell> $ docker run ubuntu env PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin HOSTNAME=9a123737965f HOME=/root </code>` \* Injecter une variable d'environnement au démarrage d'un conteneur en faisant remonter la variable `LOGNAME` dans un conteneur. `<code shell> $ docker run -i -t -e=LOGNAME ubuntu root@41716d7ee125:/# env HOSTNAME=41716d7ee125 PWD=/ LOGNAME=smb101 HOME=/root LSCOLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=3 0;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:.tar=01;31:.tgz=01;31:.arc=01;31:.arj=01;31:.taz=01;31:.lha=01;31:.lz4=01;31:.lz h=01;31:.lzma=01;31:.tlz=01;31:.txz=01;31:.tzo=01;31:.t7z=01;31:.zip=01;31:.z=01;31:.dz=01;31:.gz=01;31:.lrz=01;31:.lzo=0 1;31:.xz=01;31:.zst=01;31:.tzt=01;31:.bz2=01;31:.bz=01;31:.tbz=01;31:.tbz2=01;31:.tz=01;31:.deb=01;31:.rpm=01;31:.jar=01;31:.war= 01;31:.ear=01;31:.sar=01;31:.rar=01;31:.alz=01;31:.ace=01;31:.zoo=01;31:.cpio=01;31;.7z=01;31;.rz=01;31;.cab=01;31;.wim=01;31;.sw m=01;31;.dwm=01;31;.esd=01;31;.jpg=01;35;.jpeg=01;35;.mjpg=01;35;.mjpeg=01;35;.gif=01;35;.bmp=01;35;.pbm=01;35;.pgm=01;35;. ppm=01;35;.tga=01;35;.xbm=01;35;.xpm=01;35;.tif=01;35;.tiff=01;35;.png=01;35;.svg=01;35;.svgz=01;35;.mng=01;35;.pcx=01;35;.mo v=01;35;.mpg=01;35;.mpeg=01;35;.m2v=01;35;.mkv=01;35;.webm=01;35;.ogm=01;35;.mp4=01;35;.m4v=01;35;.mp4v=01;35;.vob=01; 35;.qt=01;35;.nuv=01;35;.wmv=01;35;.asf=01;35;.rm=01;35;.rmvb=01;35;.flc=01;35;.avi=01;35;.fli=01;35;.flv=01;35;.gl=01;35;.dl=01;3 5;.xcf=01;35;.xwd=01;35;.yuv=01;35;.cgm=01;35;.emf=01;35;.ogv=01;35;.ogx=01;35;.aac=00;36;.au=00;36;.flac=00;36;.m4a=00;36;.mi d=00;36;.midi=00;36;.mka=00;36;.mp3=00;36;.mpc=00;36;.ogg=00;36;.ra=00;36;.wav=00;36;.oga=00;36;.opus=00;36;.spx=00;36;.xsp f=00;36: TERM=xterm SHLVL=1 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin =/usr/bin/env </code>` \* Création d'une variable d'environnement qui n'existe pas sur l'hôte pour spécifier au conteneur l'adresse IP `192.168.1.200` d'un service Web nécessaire pour la réalisation de sa tâche `<code shell> $ docker run -e=IPAPIWEB=192.168.1.200 ubuntu env PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin HOSTNAME=2352eb733166 IPAPIWEB=192.168.1.200 HOME=/root </code>` \* Modification du hostname au démarrage du conteneur `<code shell> $ docker run -it -h idefix ubuntu root@idefix:/# </code>` `====` Quelques autres commandes de Docker `====` `==== Supprimer un conteneur === <code shell> docker rm [identifiant ou nom du conteneur] </code>` `====` `==== Faire le ménage ===` La commande `system` et sa sous-commande `prune` permettant de réaliser le ménage dans les conteneurs arrêtés, les images orphelines et d'autres ressources a priori non utilisées, sans supprimer les images. `<code shell> docker system prune </code>` `====` `==== Supprimer une image ===` Les conteneurs qui utilisent cette image doivent au préalable avoir été supprimés. Le paramètre `-f` force cependant la suppression si cela n'est pas le cas. `<code shell> docker rmi [identifiant ou nom de l'image] </code>` `====` `==== Lancer un conteneur avec un nom donné === <code shell> docker run -name=[nom fourni] [image] </code>` `====` `==== Lancer un conteneur en mode détaché ===` Lors du lancement d'un conteneur, on perd l'accès à la console du serveur Ubuntu. Cette commande permet de lancer le conteneur et de retrouver le shell du serveur sans arrêter le conteneur. `<code shell> btssio@ubuntudocker:~$ docker run -i -t -d ubuntu e70a11313849c4c2ce8beebe72935368b909759ab5eb4e2c2f1bcf715c1a2bd1 btssio@ubuntudocker:~$ docker ps CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES e70a11313849 ubuntu "/bin/bash" 3 seconds ago Up 2 s gallant_kilby </code>` `====` `==== Se connecter à un conteneur lancé ===` Pour accéder ensuite à ce conteneur on utilise la commande `attach` avec l'identifiant ou le nom du conteneur : `<code shell> btssio@ubuntudocker:~/Documents/lab_exercice$ docker attach e70a11313849 root@e70a11313849:/# </code>` `====` `==== Arrêter un conteneur === <code shell> docker stop [conteneur] </code>` `====` `==== Démarrer un`

conteneur arrêté == <code shell> docker start [conteneur] </code>

Docker propose également un **guide de découverte de Docker (en anglais)** avec son ordinateur personnel ou bien en ligne dans un lab (Cloud) : <https://www.docker.com/101-tutorial>

Autres ressources :

- <https://devopssec.fr/article/cours-complet-apprendre-technologie-docker>

===== Memento Docker =====

**Memento Docker :**

<https://www.digitalocean.com/community/tutorials/how-to-remove-docker-images-containers-and-volumes-fr>

===== Retour Accueil Docker ===== \* [Docker](#)

From:

/ - **Les cours du BTS SIO**

Permanent link:

</doku.php/reseau/docker/utiliserdocker?rev=1733691595>

Last update: **2024/12/08 21:59**

