**Optimisez votre déploiement en créant des conteneurs avec Docker**

# Découvrez les conteneurs

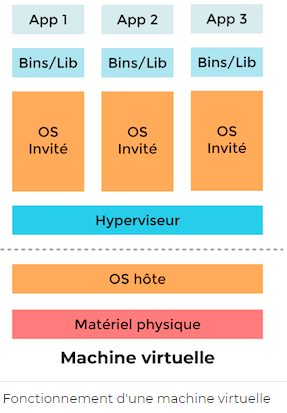
Historiquement, quand nous avions besoin de serveurs, nous achetions des serveurs physiques avec une quantité définie de CPU, de mémoire RAM ou de stockage sur le disque.

## Une machine virtuelle

C’est la possibilité de mettre plusieurs OS totalement distincts sur une seule machine physique.

Chaque machine virtuelle réserve les ressources qu’elle a besoin du système hôte.

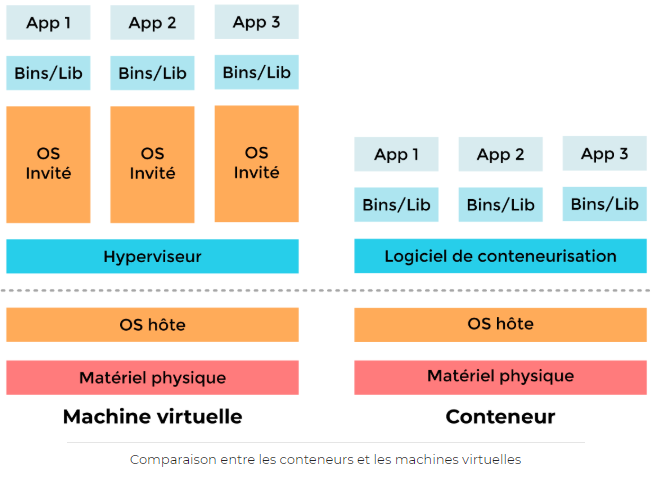
On parle de virtualisation ***lourde***



Mais il arrive très souvent que l'application qu'elle fait tourner ne consomme pas l'ensemble des ressources disponibles sur la machine virtuelle. Ainsi est né un nouveau système de virtualisation plus léger : les **conteneurs**.

## Un conteneur

C’est de la virtualisation de processus et non plus de ressources, on parle de virtualisation légère. On isole les processus .Le conteneur partage donc les ressources avec le système hôte.



 Les conteneurs partagent entre eux le kernel Linux ; ainsi, il n'est pas possible de faire fonctionner un système Windows ou BSD dans celui-ci.

Avantages des conteneurs :

* Ne réserve que les ressources nécessaires :

Une autre différence importante avec les machines virtuelles est qu'un conteneur **ne réserve pas** la quantité de CPU, RAM et disque attribuée auprès du système hôte. Ainsi, nous pouvons allouer 16 Go de RAM à notre conteneur, mais si celui-ci n'utilise que 2 Go, le reste ne sera pas verrouillé.

* Démarre plus rapide

Les conteneurs n'ayant pas besoin d'une virtualisation des ressources mais seulement d'une isolation, ils peuvent **démarrer beaucoup plus rapidement** et plus fréquemment qu'une machine virtuelle sur nos serveurs hôtes, et ainsi réduire encore un peu les frais de l'infrastructure.

## Pourquoi utiliser des conteneurs ?

Les conteneurs étant capables de démarrer très rapidement, ils sont souvent utilisés en production pour ajouter des ressources disponibles et ainsi répondre à des besoins de mise à l'échelle, ou de scalabilité. Mais ils répondent aussi à des besoins de préproduction ; en étant légers et rapides au démarrage, il permettent de créer des environnements dynamiques et ainsi de répondre à des besoins métier.

Attention, bien que le fonctionnement des conteneurs et des machines virtuelles soient très proches, **l'un ne remplace pas l'autre** ! Pour faire tourner un conteneur, il faut toujours une machine (virtuelle ou non). Ce sont donc deux outils complémentaires à utiliser en connaissance de cause

# Découvrez ce qu’est Docker

Docker est un outil permettant d’isoler des applications dans des conteneurs virtuels

**Docker** is a tool used to create, deploy, and run applications by using containers

Il faut voir Docker, en gros, comme un outil pour rendre portable votre application, sans les contraintes d’une machine virtuelle.

Docker est une plateforme qui va vous permettre d'exécuter votre code à l'intérieur d'un conteneur indépendamment de la machine sur laquelle vous êtes

Un conteneur ne doit faire tourner qu'**un seul processus**. Ainsi, dans le cas d'une stack LAMP (Linux, Apache, MySQL, PHP), nous devons créer **3 conteneurs différents**, un pour Apache, un pour MySQL et un dernier pour PHP.

## Pourquoi utiliser Docker ?

Vous êtes dans une équipe de 10 personnes et chacun utilise un OS différent (Ubuntu, macOS, Windows, CentOS, etc.). Comment faire pour avoir un **environnement unifié et fonctionnel** chez l'ensemble des développeurs ?

Docker répond à ces problématiques en créant des conteneurs. Grâce à Docker, vous n'aurez plus de problème de différence d'environnement, et votre code marchera partout !

un des premiers usages de Docker se trouve dans **la création d'environnements locaux**. Il est plus simple d'utiliser Docker en local quand on travaille avec de nombreuses versions différentes des logiciels, et ainsi ne pas avoir des problèmes de compatibilité entre elles.

On retrouve aussi Docker dans les domaines de la CI (*Continous Integration*, ou **Intégration Continue**) et de la CD (*Continous Delivery*, ou **Livraison Continue**). Cela permet à la CI/CD de créer rapidement des espaces isolés pour faire tourner vos tests.

##### Un conteneur est immuable

**L'immutabilité** d'un conteneur est aussi importante. Un conteneur ne doit pas stocker de données qui doivent être pérennes, car il les perdra (à moins que vous les ayez pérennisées). Mais si vous souhaitez en local mettre une base de données dans un conteneur Docker, vous devez créer un volume pour que celui-ci puisse stocker les données de façon pérenne.

## Choisissez votre version de Docker

Docker Inc distribue 3 versions de Docker différentes :

* Docker Community Edition (Linux seulement) ;
* Docker Desktop (Mac ou Windows) ;
* Docker Enterprise (Linux seulement).

**Docker Desktop** et **Docker Community Edition** (CE) sont deux versions de Docker **gratuites**. Avec les deux solutions, vous aurez un Docker **fonctionnel** sur votre ordinateur.

# Installez Docker sous Linux Ubuntu

Install using the repository

Set up the repository

1.Update the apt package index ( pour mettre à jour nos dépôts)

$ sudo apt-get update

2.Install packages to allow apt to use a repository over HTTPS:

Sudo curlapt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common

3. Add Docker’s official GPG key:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

4. Set up the **stable** repository.

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

Install Docker CE

5. Update the apt package index.

sudo apt-get update

6. Install the *latest version* of Docker Engine - Community and containerd

sudo apt-get install docker-ce docker-ce-cli containerd.io

To install a specific version of Docker CE, list the available versions in the repo, then select and install

7.Run docker

systemctl enable docker

systemctl start docker

8. Verify that Docker CE is installed correctly by running the hello-world image.

docker run hello-world

9. **Exécution de la commande Docker sans Sudo (optionnel)**

Par défaut, la commande docker ne peut être exécutée que par l'utilisateur **root** ou par un utilisateur du groupe **docker**, créé automatiquement lors du processus d'installation de Docker. Si vous essayez d'exécuter la commande docker sans la préfixer avec sudo ou sans faire partie du groupe **docker**, vous obtiendrez une sortie de données comme celle-ci:

Output

docker: Cannot connect to the Docker daemon. Is the docker daemon running on this host?.

See 'docker run --help'.

Si vous voulez éviter de taper sudo chaque fois que vous exécutez la commande docker, ajoutez votre nom d'utilisateur au groupe docker:

* sudo usermod -aG docker your-user

Pour appliquer la nouvelle appartenance à un groupe, déconnectez-vous du serveur et reconnectez-vous, ou tapez ce qui suit:

* su your-user

Vous serez invité à entrer le mot de passe de votre utilisateur pour continuer.

Pour résumer il faut :

1. Update
2. Installer les prérequis
3. Ajout de la clé GPG
4. Ajout du repository Docker aux sources APT
5. Faire un Update
6. Installer la dernière version de Docker

# Docker Images

Une image Docker représente un template en lecture seule.

Cette image contient le code qui sera exécuté, notamment les éventuelles définitions des bibliothèques et des dépendances dont votre code a besoin. Un conteneur Docker est une image Docker instanciée (c.-à-d. en cours d'exécution).

Les images sont créées à partir de fichiers de configuration, nommés “Dockerfile”, qui décrivent exactement ce qui doit être installé sur le système. Un conteneur est l’exécution d’une image : il possède la copie du système de fichiers de l’image, ainsi que la capacité de lancer des processus. En gros, c’est un OS, avec lequel vous pouvez interagir.

# Lancez votre premier conteneur en local

Un conteneur Docker est une image Docker instanciée (c.-à-d. en cours d'exécution)

## Le Docker Hub

C'est la registry officielle de Docker :

Une **registry** est un service proposé par Docker qui permet de partager des images à d'autres personnes. C’est un composant majeur dans l’écosystème Docker, car il permet :

* à des **développeurs** de **distribuer des images** prêtes à l’emploi et de les **versionner** avec un système de tags ;
* à des **outils** d’intégration en continu de jouer une suite de **tests**, sans avoir besoin d’autre chose que de Docker ;
* à des **systèmes automatisés** de **déployer ces applications** sur vos environnement de développement et de production.

## [Docker Hub](https://hub.docker.com) is a service provided by Docker for finding and sharing container images with your team. It provides the following major features:

## [Repositories](https://docs.docker.com/docker-hub/repos/): Push and pull container images.

## [Teams & Organizations](https://docs.docker.com/docker-hub/orgs/): Manage access to private repositories of container images.

## [Official Images](https://docs.docker.com/docker-hub/official_images/): Pull and use high-quality container images provided by Docker.

## [Publisher Images](https://docs.docker.com/docker-hub/publish/customer_faq/): Pull and use high- quality container images provided by external vendors. Certified images also include support and guarantee compatibility with Docker Enterprise.

## [Builds](https://docs.docker.com/docker-hub/builds/): Automatically build container images from GitHub and Bitbucket and push them to Docker Hub.

## [Webhooks](https://docs.docker.com/docker-hub/webhooks/): Trigger actions after a successful push to a repository to integrate Docker Hub with other services.

## Démarrez votre premier conteneur

Il faut taper la commande suivante dans le terminal : docker run hello-world

Quand vous utilisez cette commande, le **daemon Docker** va chercher si l'image hello-world est **disponible en local**. Dans le cas contraire, il va la **récupérer sur la registry Docker officielle**.

Dans notre cas, le conteneur a démarré, puis affiché du contenu, et il a fini par s'arrêter. Si vous souhaitez **que votre conteneur reste allumé jusqu’à l'arrêt du service qu'il contient**, vous devez ajouter l’argument --detach (-d). Celui-ci permet de ne pas rester attaché au conteneur, et donc de pouvoir lancer plusieurs conteneurs. Nous allons voir dans la section suivante comment utiliser l’argument *-d*.

## Démarrez un serveur Nginx avec un conteneur Docker

Maintenant, nous allons aller plus loin avec celui-ci. Nous allons lancer un conteneur qui démarre un serveur Nginx en utilisant deux options : docker run -d -p 8080:80 nginx.

Dans cette commande, nous avons utilisé deux options :

* -d pour **détacher le conteneur** du processus principal de la console. Il vous permet de continuer à utiliser la console pendant que votre conteneur tourne sur un autre processus ;
* -p pour définir **l'utilisation de ports**. Dans notre cas, nous lui avons demandé de transférer le trafic **du port 8080 vers le port 80 du conteneur**. Ainsi, en vous rendant sur l'adresse  http://127.0.0.1:8080, vous aurez la page par défaut de Nginx.

Pour attribuer un nom au conteneur on l’ajoute avec l’option –name ( ce sont deux barres - ) ex :

docker run -d -p 8080 :80 --name webserver nginx

Vous pourriez aussi avoir besoin de "rentrer" dans votre conteneur Docker pour pouvoir y effectuer des actions. Pour cela, vous devez utiliser la commande docker exec -ti ID\_RETOURNÉ\_LORS\_DU\_DOCKER\_RUN bash ou docker exec -ti NOM\_DU\_CONTENEUR bash.

Par exemple : docker exec -ti 75d7679910a2 bash ou docker exec -ti webserver bash

Dans cette commande, l'argument -ti permet d'avoir un shell bash pleinement opérationnel. Une fois que vous êtes dans votre conteneur, vous pouvez vous rendre, via la commande cd /usr/share/nginx/html, dans le répertoire où se trouve le fichier index.html, pour modifier son contenu et voir le résultat en direct à l'adresse http://127.0.0.1:8080

**docker run -d -p 8080:80 alexwhen/docker-2048**

## Arrêtez votre conteneur Docker

docker stop ID ou NOM.

Maintenant qu'il a été arrêté, vous pouvez le supprimer avec la commande :

docker rm ID ou NOM. Celle-ci va détruire le conteneur et son contenu.

## Récupérez une image du docker Hub

Vous pouvez aussi avoir besoin de **récupérer des images sur le Docker Hub** sans pour autant lancer de conteneur. Pour cela, vous avez besoin de lancer la commande suivante:

docker pull NOM\_DE\_LIMAGE ex : docker pull ubuntu

## Affichez l'ensemble des conteneurs existants

docker ps

Vous pouvez aussi voir l'ensemble des images présentes en local sur votre ordinateur, avec la commande:

 docker images -a.

## Comment nettoyer mon système

Après avoir fait de nombreux tests sur votre ordinateur, vous pouvez avoir besoin de faire un peu de ménage. Pour cela, vous pouvez supprimer l'ensemble des ressources manuelles dans Docker.

Ou vous pouvez laisser faire Docker pour qu'il fasse lui-même le ménage. Voici la commande que vous devez utiliser pour faire le ménage :docker system prune

Celle-ci va supprimer les données suivantes :

* l'ensemble des **conteneurs** Docker qui ne sont pas en status *running* ;
* l'ensemble des **réseaux** créés par Docker qui ne sont pas utilisés par au moins un conteneur ;
* l'ensemble des **images** Docker non utilisées ;
* l'ensemble des **caches** utilisés pour la création d'images Docker.

# Créez votre premier Dockerfile

Le Dockerfile est un fichier qui contient toutes les instructions pour créer une image

Nous allons créer ensemble une image Docker, dans laquelle nous allons installer **Node.js**, ainsi que les différentes **dépendances de notre projet**.

Chaque instruction que nous allons donner dans notre Dockerfile va créer une nouvelle layer correspondant à chaque étape de la construction de l'image, ou de la recette.

Si nous restons dans l'analogie de la cuisine, le Dockerfile permet de connaître notre recette pour faire une pièce montée. Alors, chaque argument de celle-ci crée un nouvel étage sur la pièce montée, nommé layer. Notre but étant de limiter le nombre d'étages, pour que votre pièce montée soit la plus légère et performante possible.

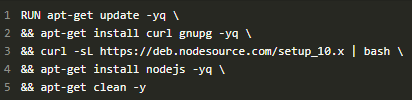
## Créez les instructions dans votre Dockerfile

La première chose que vous devez faire est de créer un fichier nommé "Dockerfile", puis de définir dans celui-ci **l'image que vous allez utiliser comme** base, grâce à l'instruction FROM. Dans notre cas, nous allons utiliser une image de base Debian 9.



L'instruction **FROM** n'est utilisable qu'une seule fois dans un Dockerfile.

Ensuite, utilisez l'instruction **RUN** pour exécuter une commande dans votre conteneur.



Limitez au maximum le nombre d'instructions RUN, afin de limiter le nombre de layers créées, et donc de réduire la taille de notre image Docker.

Puis, utilisez l'instruction  **ADD**  afin de copier ou de télécharger des fichiers dans l'image. Dans notre cas, nous l'utilisons pour ajouter les **sources de notre application** locale dans le dossier /app/ de l'image.



Utilisez ensuite l'instruction **WORKDIR** qui permet de modifier le répertoire courant. La commande est équivalente à une commande **cd** en ligne de commande. L'ensemble des commandes qui suivront seront toutes exécutées depuis le répertoire défini.



Puis, l'instruction **RUN** suivante permet d'installer le package du projet Node.js.



Maintenant que le code source et les dépendances sont bien présents dans votre conteneur, nous devons indiquer à notre image quelques dernières informations.



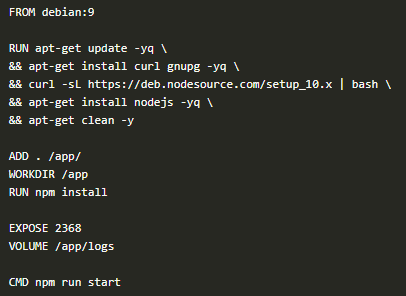
L'instruction EXPOSE permet d'indiquer le port sur lequel votre application écoute. L'instruction VOLUME permet d'indiquer quel répertoire vous voulez partager avec votre host.

Les instructions EXPOSE et VOLUME ne sont pas nécessaires au bon fonctionnement de notre image Docker. Cependant, les ajouter permet une meilleure compréhension pour l’utilisateur des ports d'écoute attendus, ainsi que des volumes partageables.

Nous allons conclure par l'instruction qui doit toujours être présente, et la placer en dernière ligne pour plus de compréhension : **CMD**. Celle-ci permet à notre conteneur de savoir quelle commande il doit exécuter lors de son démarrage.



**En résumé, voici notre Dockerfile une fois terminé :**



## Créez votre fichier .dockerignore

Sur un projet Git, nous utilisons un fichier .gitignore ; sur Docker il existe le même type de fichier. Celui-ci permet de ne pas copier certains fichiers et/ou dossiers dans notre conteneur lors de l’exécution de l'instruction ADD.

À la racine de votre projet (soit à côté de votre fichier **Dockerfile**), vous devez créer un fichier .dockerignore qui contiendra les lignes suivantes :



## Profitez de l'optimisation Docker

Lancez votre conteneur personnalisé !

g