Tutos Openclassrooms

Optimisez votre déploiement en créant des conteneurs avec Docker

# Découvrez les conteneurs

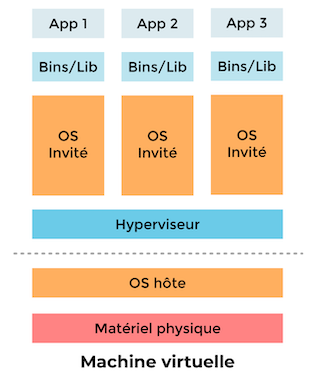
**Dans cette première partie**, nous allons faire connaissance avec les **conteneurs** ainsi qu'avec Docker. Mais avant cela, nous allons revenir sur quelques notions importantes :

* comprendre la notion de **machine virtuelle** ;
* comprendre la notion de **conteneur** ;
* **pourquoi** utiliser des 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.

Or, on avait souvent besoin d'avoir de la puissance supplémentaire pour des périodes de forte charge (fête de Noël, par exemple). Ainsi, vous deviez acheter plus de serveurs pour répondre aux pics d'utilisation. Une solution a ainsi été créée : la ***machine virtuelle***.

## Une machine virtuelle ?



Lorsque vous utilisez une machine virtuelle (VM), vous faites ce qu’on appelle de la ***virtualisation lourde***. En effet, vous recréez un système complet dans le système hôte, pour qu’il ait ses propres ressources.

**L'isolation avec le système hôte est donc totale** ; cependant, cela apporte plusieurs contraintes :

❌une machine virtuelle prend du **temps** à démarrer ;

❌une machine virtuelle **réserve les ressources** (CPU/RAM) sur le système hôte.

Mais cette solution présente aussi de nombreux avantages :

✅une machine virtuelle est totalement **isolée** du système hôte ;

✅les ressources attribuées à une machine virtuelle lui sont totalement **réservées** ;

✅vous pouvez installer **différents OS** (Linux, Windows, BSD, etc.).

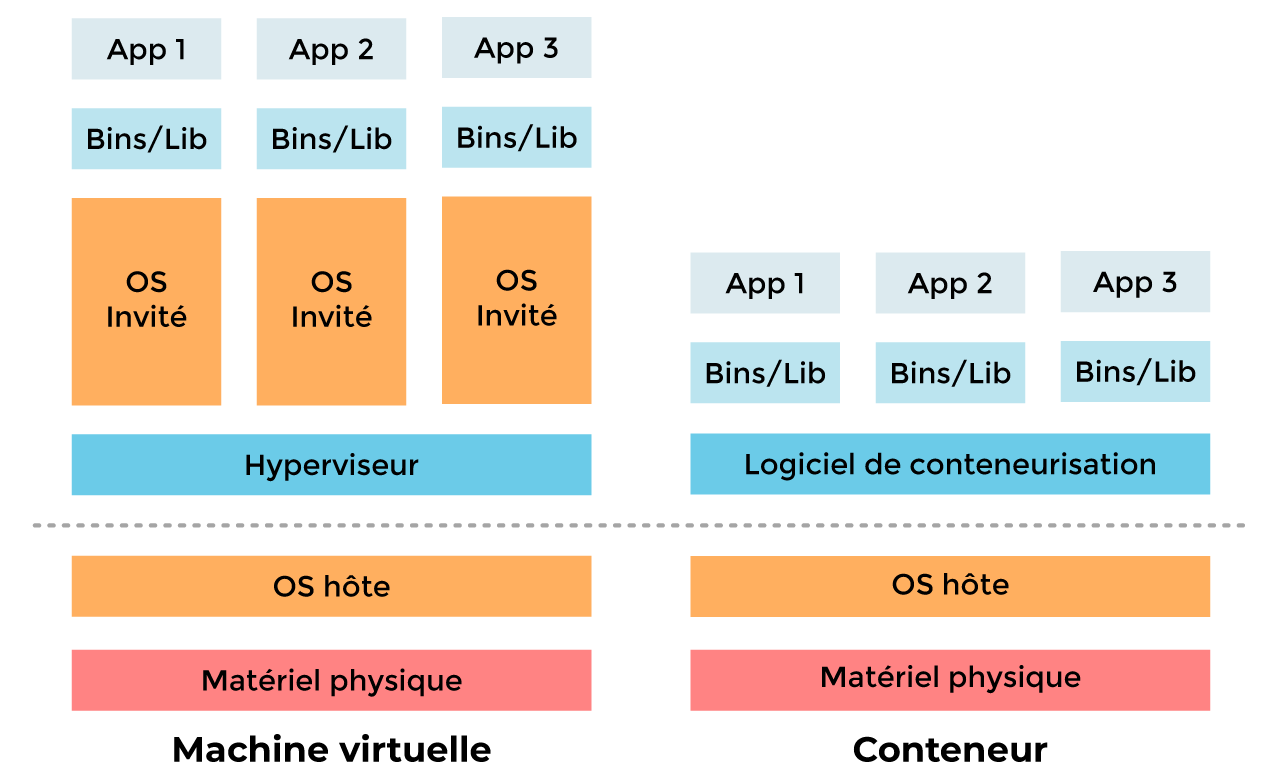
Mais il arrive très souvent que l'application qu'elle fait tourner ne consomme pas l'ensemble des ressources disponibles sur la machine virtuelle. Ainsi est né un nouveau système de virtualisation plus léger : les **conteneurs**.

## Un conteneur ?

Un conteneur Linux est un **processus** ou un ensemble de processus isolés du reste du système, tout en étant **légers**.

Le conteneur permet de faire de la ***virtualisation légère***, c'est-à-dire qu'il ne virtualise pas les ressources, il ne crée qu'une **isolation des processus**. Le conteneur partage donc les ressources avec le système hôte.

Attention, les conteneurs existent depuis plus longtemps que Docker. OpenVZ ou LXC sont des technologies de conteneur qui existent depuis de nombreuses années.



Les conteneurs, au sens d'OpenVZ et LXC, apportent une **isolation importante des processus systèmes** ; cependant, les ressources CPU, RAM et disque sont totalement partagées avec l'ensemble du système. 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.

Voyons ensemble quelques avantages des conteneurs.

**Ne réservez 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.

**Démarrez rapidement vos conteneurs**

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.

**Donnez plus d'autonomie à vos développeurs**

En dehors de la question pécuniaire, il y a aussi la possibilité de faire tourner des conteneurs sur le poste des développeurs, et ainsi de réduire les différences entre la "sainte" production, et l'environnement local sur le poste des développeurs.

## Pourquoi utiliser des conteneurs ?

Les conteneurs permettent de **réduire les coûts**, d'augmenter la **densité de l'infrastructure**, tout en améliorant le cycle de déploiement.

Nous retrouvons les conteneurs dans de nombreux contextes. Par exemple, les outils d'intégration continue (CI) de grandes sociétés utilisent des conteneurs. BlaBlaCar utilise des conteneurs pour créer et isoler rapidement des tests/builds d'application.

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, ils 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.

## En résumé

Vous connaissez maintenant la différence entre conteneur et machine virtuelle ; vous avez ainsi pu voir les différences entre la virtualisation lourde et la virtualisation légère.

Un conteneur doit être léger, il ne faut pas ajouter de contenu superflu dans celui-ci afin de le démarrer rapidement, mais il apporte une isolation moindre. A contrario, les machines virtuelles offrent une très bonne isolation, mais elles sont globalement plus lentes et bien plus lourdes.

Dans le prochain chapitre, nous allons découvrir une solution de conteneur qui s'est imposée comme leader des technologies de conteneurisation, et que vous allez apprendre à utiliser dans ce cours : Docker !

# Découvrez ce qu'est Docker

Docker a été créé pour les besoins d'une société de Platform as a Service (PaaS) appelée **DotCloud**. Finalement, en mars 2013, l'entreprise a créé une nouvelle structure nommée **Docker Inc** et a placé en open source son produit **Docker**.

Docker apporte une notion importante dans le monde du conteneur. Dans la vision Docker, 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. Alors que dans un conteneur LXC ou OpenVZ, nous aurions fait tourner l'ensemble des 3 services dans un seul et unique conteneur.

## Pourquoi utiliser Docker ?

Docker répond à une problématique forte dans le monde du développement.

Prenons un exemple : vous avez développé votre projet de Twitter Lite en local. Tout fonctionne bien, mais au moment de mettre en production, vous vous rendez compte que vous ne savez pas comment **déployer votre projet**. Un autre exemple : 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 !

**Utilisez Docker sur tous vos environnements**

Docker est utilisé par de très nombreuses sociétés pour différents usages. Ainsi, 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.

Plus globalement, vous pouvez utiliser Docker sur l'ensemble de vos environnements.

Attention, cependant, **Docker ne convient pas à tous les usages**. Par exemple, lorsqu'il s'agit de faire persister de grandes quantités de mémoire disque et d'assurer une grande continuité de service. Docker n'est pas vraiment adapté à ce type d'usage.

**Docker, une amélioration de LXC**

Docker a commencé par être une surcouche du conteneur LXC, tout en apportant de nombreux changements pour que les conteneurs soient plus flexibles et portables.

Les conteneurs Docker vous apportent aussi les notions de stateless et d'immutabilité.

**Stateless et immutabilité**

*Stateless vs Stateful*

Dans le monde de Docker, vous allez souvent entendre parler de **stateless et stateful**, deux catégories de conteneurs, et vous devez savoir à quoi correspond chaque catégorie.

Si nous prenons le cas d'une **base de données MySQL**, celle-ci est **stateful** car elle stocke un état. Ainsi, si vous éteignez et rallumez votre base de données, vous la retrouverez dans le même état de fonctionnement.

Stateless est donc l'inverse : l'application ne stocke pas d'état. Vous pouvez prendre le cas du **protocole HTTP**, celui-ci est **stateless**. À chaque nouvelle requête HTTP, les mêmes séries d'actions seront réalisées.

*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.

Si vous êtes sous Windows ou macOS, utilisez Docker Desktop qui va créer pour vous l'ensemble des services nécessaires au bon fonctionnement de Docker.

Si vous êtes sous Linux, prenez la version Community Edition (CE) ; vous utiliserez aussi cette version pour vos serveurs.

La version Docker Enterprise ne ressemble pas du tout aux versions Desktop et CE. Celle-ci répond à des besoins plus poussés des entreprises, et propose une interface de gestion d'infrastructures sous Docker. Cette version est soumise à une licence fournie par Docker Inc.

## En résumé

Dans ce chapitre, vous avez découvert l'outil Docker et ses différences par rapport aux autres conteneurs :

* Les conteneurs Docker sont différents des conteneurs Linux (LXC), car ils sont par définition **immuables** et **stateless** ;
* Docker est utilisé à tous les niveaux de l'infrastructure (CI / Développement / Production) ;
* différence entre Docker Enterprise et Docker CE/Desktop.

# Installez Docker sur votre poste

Nous avons vu dans le chapitre précédent les différentes versions de Docker que nous pourrions être amenés à utiliser. Nous allons voir ensemble deux versions de Docker : Docker for Mac/Docker for Windows, et Docker Community Edition (CE).

## Docker for Mac et Docker for Windows

**Docker Hub**

Le **Docker Hub** est un service fourni par Docker Inc ; vous pouvez le comparer à GitHub, mais spécialisé dans le **stockage d'image pour Docker**.

Pour télécharger et utiliser Docker (et donc pour la suite de ce cours), il vous faudra créer un compte sur le Docker Hub.

**Version Stable et version Edge**

Docker Inc distribue deux versions des clients Docker for Mac et Docker for Windows.

Ainsi, il existe une version Stable qui est mise à jour tous les trimestres, et une version Edge qui dispose des fonctionnalités encore en beta, et qui est disponible tous les mois.

Dans notre cas, nous allons préférer l'utilisation de la version Stable, car celle-ci assure d'avoir un environnement fonctionnel et stable tout le long de votre découverte de Docker.

Maintenant que vous avez un compte Docker Hub fonctionnel, et que nous savons quelle version nous souhaitons utiliser, nous allons pouvoir installer Docker for Windows et Docker for Mac.

**Docker for Windows**

Pour faire fonctionner Docker sous Windows, nous aurions de nombreuses manipulations à faire ; cependant, Docker Inc a créé un outil qui permet d'automatiser l'ensemble des tâches, et ainsi d'avoir rapidement Docker sous Windows.

L'utilitaire créé par Docker Inc va vous créer une machine virtuelle Hyper-V sur votre Windows, et procéder à l'installation de Docker au sein de celui-ci.

Attention, vous devez avoir un Windows Entreprise ou Windows Professional, pour que cette fonctionnalité soit disponible. Dans le cas contraire, il faudra vous tourner vers l'utilisation de **Docker Toolbox**.

Vous devez vous rendre sur cette page (https://hub.docker.com/editions/community/docker-ce-desktop-windows) et télécharger la version "Stable" de Docker for Windows.

Une fois le téléchargement réalisé, vous pourrez lancer l'utilitaire et suivre la procédure d'installation de celui-ci. Attention, vous devrez certainement redémarrer votre PC.

**Installation de docker**

Une fois l'installation terminée, vous allez être invité à connecter votre compte Docker Hub créé précédemment.

Vous avez maintenant un Docker for Windows totalement fonctionnel sur votre PC !

**Docker for Mac**

L'installation de Docker for Mac fonctionne comme pour Docker for Windows. Vous devrez télécharger l'utilitaire à cette adresse (https://hub.docker.com/editions/community/docker-ce-desktop-mac). Référez-vous à la vidéo ci-dessus pour voir une démonstration de cette installation.

## Docker Community Edition

Pour la version **Docker Community Edition**, la création d'un compte sur le Docker Hub n'est pas nécessaire, ni obligatoire. Cependant, par souci d'uniformisation, nous allons là aussi créer un compte sur le Docker Hub et nous y connecter.

Nous partons du postulat que vous utilisez une machine Linux sous Debian, et vous allez procéder à l'installation de Docker CE. Pour voir comment installer Docker sur une autre distribution Linux, reportez-vous à la documentation Docker.

Vous allez commencer par mettre à jour votre système et installer les prérequis pour l'installation de Docker :

sudo apt-get update && sudo apt-get install apt-transport-https ca-certificates curl gnupg2 software-properties-common

Une fois cela réalisé, nous allons ajouter le repository de Docker :

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

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

Nous pouvons maintenant installer Docker sur notre machine :

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

Vous avez maintenant Docker sur votre machine ; cependant, vous n'avez pas les accès nécessaires pour pouvoir parler avec le daemon. Pour y remédier, vous devez utiliser la commande suivante : sudo usermod -aG docker your-user.

Après un redémarrage de votre session utilisateur, vous pourrez communiquer avec le daemon Docker et utiliser Docker normalement.

Comme dit précédemment, pour des raisons d'uniformisation nous allons aussi nous connecter au Docker Hub avec la commande docker login. Vous devrez utiliser votre compte créé précédemment sur le Docker Hub.

## En résumé

Vous êtes maintenant capable d'installer la version de Docker correspondant à votre OS, et de vous connecter au Docker Hub.

# Lancez votre premier conteneur en local

**Dans cette deuxième partie**, je vous propose de prendre en main Docker. Nous allons commencer par découvrir l'**interface en ligne de commande**, qui nous permet de discuter avec le daemon Docker installé précédemment.

D'ici la fin de cette partie, vous serez capable de **lancer et gérer vos conteneurs**. Mais commençons dans ce chapitre par comprendre ce qu'est le **Docker Hub**, puis nous lancerons notre **premier conteneur**.

## Le Docker Hub

Avant de démarrer votre premier conteneur Docker, rappelez-vous quand vous avez créé votre compte sur le Docker Hub pour télécharger votre version de Docker. Celui-ci est aussi **la registry officielle de Docker**.

**Une registry ?**

Une registry est un logiciel 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.

## Démarrez votre premier conteneur Docker

Pour démarrer votre premier conteneur, vous devez utiliser la commande docker run hello-world

➜ docker run hello-world

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/

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

Vous savez maintenant lancer un conteneur, et vous avez compris les actions effectuées par le daemon Docker lors de l'utilisation de la commande docker run.

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.

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. 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.

## Arrêtez votre conteneur Docker

Vous avez créé un conteneur avec l'option --detach, vous aurez donc sûrement besoin de l'arrêter ! Pour cela, faites appel à la commande docker stop ID\_RETOURNÉ\_LORS\_DU\_DOCKER\_RUN.

Maintenant que votre conteneur Docker a été arrêté, vous pouvez le supprimer avec la commande docker rm ID\_RETOURNÉ\_LORS\_DU\_DOCKER\_RUN. Celle-ci va détruire le conteneur et son contenu ; cependant, vous pouvez toujours recréer votre conteneur avec la commande docker run vue plus haut.

## 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 hello-world

Using default tag: latest

latest: Pulling from library/hello-world

Digest: sha256:2557e3c07ed1e38f26e389462d03ed943586f744621577a99efb77324b0fe535

Status: Image is up to date for hello-world:latest

En lançant cette commande, vous téléchargez une image directement depuis le Docker Hub, et vous la stockez en local sur votre ordinateur.

## Affichez l'ensemble des conteneurs existants

Quand vous créez des conteneurs avec l’argument --detach, vous pouvez avoir besoin de savoir si les conteneurs sont toujours actifs ; pour cela, vous devez utiliser la commande docker ps.

➜ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

e2da0910758b nginx "nginx -g 'daemon of…" 5 seconds ago Up 4 seconds 0.0.0.0:8080->80/tcp awesome\_vaughan

Vous pouvez aussi voir l'ensemble des images présentes en local sur votre ordinateur, avec la commande docker images -a.

➜ docker images -a

REPOSITORY TAG IMAGE ID CREATED SIZE

nginx latest 881bd08c0b08 3 weeks ago 109MB

## 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

➜ docker system prune

WARNING! This will remove:

- all stopped containers

- all networks not used by at least one container

- all dangling images

- all dangling build cache

Are you sure you want to continue? [y/N] y

Deleted Containers:

941b8955b4fd8988fefe2aa91c7eb501f2d4f8c56bf4718fea8ed50904104745

a96e73c623fb6530ab41db6a82aca7017d54a99590f0b45eb6bf934ef8e4d3ed

Deleted Images:

deleted: sha256:797a90d1aff81492851a11445989155ace5f87a05379a0fd7342da4c4516663e

deleted: sha256:c5c8911bd17751bd631ad7ed00203ba2dcb79a64316e14ea95a9edeb735ca3ea

Total reclaimed space: 21.08MB

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.

## En résumé

Vous savez maintenant **télécharger**, **démarrer** et **arrêter** des conteneurs ; vous êtes aussi capable de définir une registry Docker et pourquoi on l'utilise.

Pour rappel, voici les points importants du chapitre :

* démarrage d'un conteneur avec un docker run ;
* utilisation des arguments -d et -p lors du démarrage d'un conteneur ;
* récupération d'une image depuis une registry avec la commande docker pull ;
* nettoyage du système avec docker system prune

**Dans le prochain chapitre**, vous allez **créer** votre première image Docker grâce au Dockerfile !

# Créez votre premier Dockerfile

Vous savez maintenant utiliser l'interface de commande de Docker et récupérer des images depuis le Docker Hub. Mais comment créer votre propre image ?

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

Pour cela, nous allons créer un fichier nommé "Dockerfile". Dans ce fichier Dockerfile, vous allez trouverez l'ensemble de la recette décrivant l'image Docker dont vous avez besoin pour votre projet.

À titre de comparaison, vous pouvez voir le Dockerfile comme l'équivalent d'un fichier **package.json** en **Node.js**, ou **composer.json** en **PHP**.

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.

Pour illustrer la création de notre Dockerfile, nous allons utiliser le code de l'application présente sur le repository suivant : **https://github.com/OpenClassrooms-Student-Center/ghost-cms**. L'application possède déjà un Dockerfile fonctionnel, nous allons le recréer. **Pensez bien à vous mettre dans le répertoire racine du projet !**

## 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.

FROM 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.

RUN apt-get update -yq \

&& apt-get install curl gnupg -yq \

&& curl -sL https://deb.nodesource.com/setup\_10.x | bash \

&& apt-get install nodejs -yq \

&& apt-get clean -y

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.

ADD . /app/

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.

WORKDIR /app

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

RUN npm install

Vous auriez pu aussi utiliser deux fois l'instruction ADD, et ainsi permettre à votre image d'ajouter une première fois le fichier package.json ; puis une fois le npm install réalisé, ajouter un second ADD . /app/. Ceci permettrait de réduire l'adhérence entre les dépendances présentes dans le fichier package.json et le code de l'application, ce qui permet d'économiser du temps lors du build de l'image.

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

EXPOSE 2368

VOLUME /app/logs

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.

CMD npm run start

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

FROM debian:9

RUN apt-get update -yq \

&& apt-get install curl gnupg -yq \

&& curl -sL https://deb.nodesource.com/setup\_10.x | bash \

&& apt-get install nodejs -yq \

&& apt-get clean -y

ADD . /app/

WORKDIR /app

RUN npm install

EXPOSE 2368

VOLUME /app/logs

CMD npm run start

## Créez votre fichier .dockerignore

Notre Dockerfile est maintenant prêt à fonctionner ! Cependant, il nous reste encore quelques petites modifications à faire.

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 :

node\_modules

.git

## Profitez de l'optimisation Docker

Une dernière petite chose : quand vous exécutez la commande docker build, **Docker va créer un conteneur pour chaque instruction**, et le résultat sera sauvegardé dans une layer. Le résultat final étant un ensemble de layers qui construisent une image Docker complète.

Mais cela apporte aussi de nombreux avantages. Si une layer ne bouge pas entre deux builds, **Docker ne la reconstruira pas**. Seules les layers situées après une layer qui se reconstruit seront elles aussi reconstruites.

Vous pouvez ainsi créer de nouvelles images très rapidement, sans devoir attendre indéfiniment le build de notre image.

Dans notre cas, si vous ajoutez une dépendance dans le fichier package.json, et que vous relancez un build de votre image, vous verrez qu'il n'y a que les layers situées après le ADD package.json /app/ qui seront reconstruites ; l'installation de Node.js restera en cache.

## Lancez votre conteneur personnalisé !

Vous pouvez maintenant créer votre première image Docker !

docker build -t ocr-docker-build .

L'argument -t permet de donner un nom à votre image Docker. Cela permet de retrouver plus facilement votre image par la suite.

Le . est le répertoire où se trouve le Dockerfile ; dans notre cas, à la racine de notre projet.

Maintenant, vous pouvez lancer votre conteneur avec la commande docker run :

docker run -d -p 2368:2368 ocr-docker-build

Vous retrouvez, dans le dossier logs, les logs de votre application, et vous pourrez y accéder sur le port 2368, soit via l'URL http://127.0.0.1:2368.

## En résumé

Pour créer une image Docker, vous savez utiliser les instructions suivantes :

* FROM qui vous permet de définir l'image **source** ;
* RUN qui vous permet d’exécuter des **commandes** dans votre conteneur ;
* ADD qui vous permet **d'ajouter des fichiers** dans votre conteneur ;
* WORKDIR qui vous permet de définir votre **répertoire de travail** ;
* EXPOSE qui permet de définir les **ports d'écoute** par défaut ;
* VOLUME qui permet de définir les **volumes utilisables** ;
* CMD qui permet de définir la **commande par défaut** lors de l’exécution de vos conteneurs Docker.

# Utilisez des images grâce au partage sur le Docker Hub

Vous avez maintenant créé votre propre image via un Dockerfile, et vous souhaitez partager cette image avec vos collègues. Pour cela, il existe deux solutions :

* soit vous partagez votre fichier Dockerfile à chacun de vos collègues, et vous leur demandez de créer eux-mêmes leur propre image avec un docker build ;
* soit vous envoyez votre image sur votre propre **registry**.

La première solution, bien qu'étant totalement fonctionnelle, force vos collègues à **recréer eux-mêmes chacune des images**, et cela peut prendre beaucoup de temps.

Pour améliorer cela, vous pouvez utiliser une **registry Docker** qui vous permet de partager directement votre image.

## Créez votre image sur le Docker Hub

Vous allez **créer votre image Docker sur le Docker Hub**, pour la rendre accessible de façon publique à l'ensemble de la communauté.

Pour cela, vous devez vous rendre sur le site https://hub.docker.com/ et vous connecter avec le compte que vous avez créé lors de l'installation de Docker.

Puis, cliquez sur le lien *Create Repository*. Vous arrivez alors sur une page où vous devez saisir le **nom de votre image**, ainsi qu'une description.

Pour le reste du cours, nous allons appeler cette image ocr-course-docker. Vous pouvez aussi appeler votre repository de la même façon sur le Docker Hub.

## Envoyez votre image sur le Docker Hub

Dans le chapitre précédent, vous avez créé une image Docker via la commande docker build. Nous allons maintenant la publier sur le Docker Hub. Pour cela, nous devons utiliser plusieurs commandes.

Voici la première commande que vous allez utiliser : docker tag ocr-docker-build:latest YOUR\_USERNAME/ocr-docker-build:latest. Celle-ci va créer un lien entre notre image ocr-docker-build:latest créée précédemment et l'image que nous voulons envoyer sur le Docker Hub YOUR\_USERNAME/ocr-docker-build:latest.

➜ docker tag ocr-docker-build:latest ocr/ocr-docker-build:latest

Si le conteneur que vous utilisez n'a pas de nom, utilisez son *id* de conteneur, que vous pouvez récupérer en retour de la commande docker build.

➜ docker tag id\_du\_conteneur openclassrooms/ocr-docker-build:latest

Vous pouvez maintenant exécuter la dernière commande nécessaire pour envoyer votre image vers le Docker Hub. Pour cela, vous allez exécuter la commande docker push YOUR\_USERNAME/ocr-docker-build:latest.

➜ docker push ocr/ocr-docker-build:latest

The push refers to repository [docker.io/ocr/ocr-docker-build]

00f821099106: Pushed

e9223d0e868c: Pushed

3d99cd871f5d: Pushed

fbb641a8b943: Mounted from library/debian

latest: digest: sha256:a24595e38d46841b4b571cde8bd65d49638bec2dbfc08b8bfa5fc47995a0972e size: 1165

Si vous retournez maintenant sur la page Docker Hub de votre image, vous pourrez voir qu'il existe une première version de celle-ci.

Vous pouvez créer d'autres versions de votre image en remplaçant le :latest par une autre chaîne de caractères. Cependant, il faut bien faire attention, car l'image utilisée par défaut sera toujours l'image :latest.

## Trouvez l'image qui vous correspond

Vous pouvez avoir besoin de rechercher des images dans le Docker Hub, mais il existe deux types d'images différents :

* les images **officielles** ;
* les images **personnelles**.

Docker Inc propose à ce jour 152 images officielles qu'ils maintiennent à jour continuellement, et vous les retrouverez directement sur le Docker Hub à cette adresse (https://hub.docker.com/search/?type=image&image\_filter=official).

Il y a aussi de très nombreuses images personnelles **créées par différents membres de la communauté**, et même par certains éditeurs de solutions open source.

Si vous recherchez une image, vous avez deux façons de procéder :

* la solution du barbu en **ligne de commande** ;
* la solution classique avec **l'interface web**.

Pour la solution du barbu, vous devez utiliser la commande docker search ; par exemple, si on recherche une image Nginx :

docker search nginx

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

nginx Official build of Nginx. 11115 [OK]

jwilder/nginx-proxy Automated Nginx reverse proxy for docker con… 1569 [OK]

richarvey/nginx-php-fpm Container running Nginx + PHP-FPM capable of… 695 [OK]

jrcs/letsencrypt-nginx-proxy-companion LetsEncrypt container to use with nginx as p… 494 [OK]

webdevops/php-nginx Nginx with PHP-FPM 123 [OK]

kitematic/hello-world-nginx A light-weight nginx container that demonstr… 123

zabbix/zabbix-web-nginx-mysql Zabbix frontend based on Nginx web-server wi… 92 [OK]

bitnami/nginx Bitnami nginx Docker Image 64 [OK]

linuxserver/nginx An Nginx container, brought to you by LinuxS… 56

1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5 ubuntu-16-nginx-php-phpmyadmin-mysql-5 49 [OK]

tobi312/rpi-nginx NGINX on Raspberry Pi / armhf 24 [OK]

nginx/nginx-ingress NGINX Ingress Controller for Kubernetes 17

nginxdemos/hello NGINX webserver that serves a simple page co… 13 [OK]

wodby/drupal-nginx Nginx for Drupal container image 12 [OK]

blacklabelops/nginx Dockerized Nginx Reverse Proxy Server. 12 [OK]

schmunk42/nginx-redirect A very simple container to redirect HTTP tra… 11 [OK]

centos/nginx-18-centos7 Platform for running nginx 1.8 or building n… 10

centos/nginx-112-centos7 Platform for running nginx 1.12 or building … 7

nginxinc/nginx-unprivileged Unprivileged NGINX Dockerfiles 4

1science/nginx Nginx Docker images that include Consul Temp… 4 [OK]

mailu/nginx Mailu nginx frontend 3 [OK]

travix/nginx NGinx reverse proxy 2 [OK]

toccoag/openshift-nginx Nginx reverse proxy for Nice running on same… 1 [OK]

ansibleplaybookbundle/nginx-apb An APB to deploy NGINX 0 [OK]

wodby/nginx

Comme vous pouvez le voir, le CLI docker vous **retourne une liste d'images Docker disponibles**.

Vous pouvez aussi faire la même action via le Docker Hub, si nous recherchons la même image sur celui-ci : voici l'URL de la recherche (https://hub.docker.com/search?q=nginx&type=image). Nous retrouverons là aussi une **liste d'images** avec les filtres nécessaires pour **classer les images** comme nous le souhaitons.

## En résumé

Vous savez maintenant envoyer votre image sur le Docker Hub, et vous êtes aussi capable de rechercher des images sur celui-ci.

Retenez ces commandes importantes :

* docker push qui vous permet d'**envoyer vos images locales** sur une registry ;
* docker search qui vous permet de **rechercher une image** sur votre registry.

**Dans la prochaine partie**, nous allons découvrir l'utilisation de Docker Compose, pour orchestrer des conteneurs.