**Tuto Docker - Comprendre Docker**

[Baptiste Donaux](https://www.wanadev.fr/collaborateur/7-baptiste-donaux/)

## Présentation de Docker

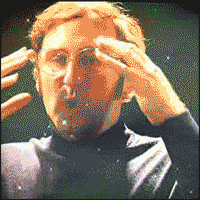
**Docker** est un produit développé par la société du même nom. Initialement développé par un ingénieur français, **Solomon Hykes,** le produit a été dévoilé en mars 2013. Depuis cette date, **Docker** est devenu LE soft à la mode ! Nous allons voir à quoi il sert et comment vous pouvez vous en servir au quotidien.

## Qu'est-ce que Docker ?

**Docker** permet de créer des environnements (appelées containers) de manière à isoler des applications. Jusque là, je suis resté très pragmatique…**Docker** repose sur le kernel **Linux** et sur une fonctionnalité : [les containers](https://linuxcontainers.org/), que vous connaissez peut-être déjà sous le doux nom de [**LXC**](http://fr.wikipedia.org/wiki/LXC). L'idée est de lancer du code (ou d'exécuter une tâche, si vous préférez) dans un environnement isolé. Je m'arrêterai ici pour ce qui est des détails des containers Linux, pour plutôt me concentrer sur les fonctionnalités mises en avant par Docker.Enfin, un troisième composant est requis, **cgroups** qui va avoir pour objectif de gérer les ressources (utilisation de la RAM, CPU entre autres).

Aaah ok ! Donc en fait, Docker utilise simplement des fonctionnalités natives de Linux comme son **noyau**, ses **LXC** et **cgroups** ? Mais du coup, je peux l'utiliser sur Windows ?

Eh bien non, pas encore. Ou du moins pas nativement (ni de façon très fiable), pour l'instant au moins. Sois patient, j'en parle dans les paragraphes suivants.



Docker permet de créer des environnements (appelés containers) de manière à isoler des applications.

## Docker, isoler un environnement ? Comme une VM ?

Oui… et non ! **Docker** et les containers Linux ne se comportent pas de la même manière qu'une **VM**. Une [machine virtuelle](http://fr.wikipedia.org/wiki/Machine_virtuelle) isole tout un système (son OS), et dispose de ses propres ressources.

Dans le cas de  **Docker**, le kernel va partager les ressources du système hôte et interagir avec le(s) container(s). Techniquement, **Docker n'est pas une VM**, pas le moins du monde, mais en terme d'utilisation, **Docker** peut-être apparenté à une **VM**.

Lancer un environnement, et isoler les composants de ce container avec les composants de mon hôte, voilà ce que **Docker** sait faire ! Il le fait d'ailleurs très bien, et reste une alternative bien plus performante que les **VMs** (à utilisation équivalente).

Ok je comprends mieux. Dans une machine virtuelle, tu alloues "physiquement" de la RAM et du stockage, comme 2go de RAM et 20Go. Alors que sur Docker, tout est partagé avec le Linux hôte qui utilise Docker. C'est pour ça que tout est plus rapide, je me trompe ?

## Docker, uniquement sur Linux ? I'm sure !

Tout à l'heure, nous avons abordé la compatibilité entre Windows et Docker. Je vois déjà la tête de certains qui ne sont déjà pas convaincus. Peut-être que vous l'avez déjà essayé ou utilisé via Windows ou MacOS. Pourtant, Docker ne peut travailler qu'avec la présence du Kernel Linux !

Pour malgré tout ne pas exclure les développeurs travaillant sur une autre plateforme qu'une Linux, des VMs (oui cette fois on parle bien de véritables machines virtuelles) ont été mises en place. La plus connue est **boot2docker**. Elle repose sur une **VM VirtualBox**. Cette machine virtuelle contient un Linux qui va alors pouvoir exécuter un **Docker**. Bien évidemment, une instance suffit pour exécuter autant de **Docker** que vous souhaitez. Depuis le dépôt GitHub de [**boot2docker**](https://github.com/boot2docker/boot2docker), il est indiqué celle-ci pèse environ 24 Mo et démarre en 5 secondes à peu près.

De cette manière, tous les développeurs, quelle que soit leur plateforme, sont en mesure de travailler sur une même application, avec un environnement similaire.

Ok, si je résume : Docker ne s'installe que sur Linux, donc pour le faire tourner sur Mac ou Windows, il faut une VM qui l'héberge. C'est pas un peu le serpent qui se mort la queue ?

**Oui, cependant boot2docker** et les autres VMs disponibles ne sont pas destinées à fonctionner en production, mais seulement pour des développeurs souhaitant se placer dans les exactes conditions que les serveurs de production.

Dans le cadre d'une production, l'utilisation d'un environnement Linux (et donc pas d'un Docker) est **FORTEMENT** conseillée.

Attention, **Docker** nécessite une version **3.8 du Kernel** (compatible donc pour une Debian 8 ou un Ubuntu) au minimum. Pour les serveurs qui sont actuellement sur Debian 7 (donc avec un noyau en version **3.2)**, l'équipe en charge du développement Docker a mis en place un [fallback](https://docs.docker.com/installation/debian/#debian-wheezystable-7x-64-bit) qui permet d'installer un noyau plus récent sans devoir passer sur **Debian**8 (aussi appelée Jessie).

### Docker, bientôt disponible nativement sur Windows

Aujourd'hui, **Docker** n'est disponible que pour **Linux** mais un partenariat récent entre **Docker Inc.** et **Microsoft** prévoit le développement de container Windows. Ceux-ci ne seront pas compatibles avec Linux mais répondront au même besoin.

## Docker, cas d'utilisation

Prenons un cas d'utilisation simple et concret.

Une entreprise lyonnaise que j’appellerai à tout hasard **Wanadev**, souhaite développer un projet Symfony comme ils savent le faire. Leur équipe est composée de deux personnes.

* Manu est un vieux de la vieille. Lui, Debian 7, c'est parfait. De plus, il est ISO avec les serveurs de production.
* Baptiste lui est un vrai hippie. Il dispose de la dernière distribution exotique. Lui, **PHP**, c'est la toute dernière version ou rien. Cependant, il peut générer du code qui ne peut pas s’exécuter correctement avec une version plus vieille, comme celle de Manu.

Jusqu'à aujourd'hui, on disait tous, « faisons des tests unitaires ». **Oui**, cela répondait à une bonne partie de nos problèmes (à la condition de faire de bons tests unitaires complets).

En effet, ça pose problème que deux développeurs ne travaillent pas sur les mêmes environnements... Du coup, grâce à Docker, on peut faire en sorte que Manu et Baptiste travaillent sur les mêmes versions Linux sans craindre des problèmes de compatibilité entre leurs codes respectifs ?

Exactement ! Dans ce cas, le plus simple est de mettre en place un **Dockerfile,**document "chef d'orchestre", qui permettra à Manu et à Baptiste de monter une image similaire. En étant malin, ce **Dockerfile**sera calqué sur les éléments présents en production. De ce fait, Manu et Baptiste en plus de travailler sur un environnement identique, seront sur un environnement similaire à celui de la production !

## Docker, déployer vos applications facilement

Dans mon exemple précédent, j'ai mis en avant l'utilisation de **Docker** pour déclarer un unique environnement de développement. Je pense que tous les utilisateurs de **Docker** doivent commencer par ça avant de voir plus grand.

Plus haut, je vous parlais d'une version particulière de **PHP**. Il est évident que vous ne risquez pas d'oublier d'installer **PHP** sur votre serveur de production, mais peut-être que la configuration de votre **PHP-FPM** sera légèrement différente. Peut-être avez-vous besoin d'une librairie supplémentaire comme **Imagick** et de son extension **PHP**. Peut-être avez-vous besoin d'un **Postfix** pour envoyer des mails. Autant de détails que vous pourriez oublier lorsque vous monterez votre serveur de production.

Pour limiter ces erreurs, ne serait-il pas plus simple de déployer le ou les container(s) permettant de faire tourner votre application ? De cette manière, plutôt que de pull un code source et d'effectuer quelques scripts de déploiement, vous pourriez déployer un ensemble cohérent packagé correspondant à une application.

On parle souvent d'application dans le monde du web, une "application" **Symfony**, ne serait pas grand-chose sans un frontal comme **Nginx ou Apache2,** un engine comme **FPM** ou **HHVM**, et une base de donnée.

Si je peux lancer plusieurs Dockers sur ma machine, ça veut dire que je peux multiplier les configurations comme je le souhaite ?

C'est tout à fait ça ! De cette manière, vous pouvez construire des environnements autonomes et isolés les uns les autres. Le bon vieux projet **SPIP** en 5.3 pourrait côtoyer sur le même serveur le dernier projet Symfony avec PHP 5.5. Cool non ?

## Conclusion

Grâce à Docker, multipliez les environnements sur votre machine, sans limiter les performances de votre ordinateur. Les ressources sont partagées avec la machine hôte ! Chaque environnement peut être configuré simplement grâce à son Dockerfile, présent à sa racine.