

# DOCKER, A QUOI ÇA SERT ?

AUTEUR : DAVID NEGRIER  
JUN 2015

**TheCodingMachine™**  
TCM://

# A QUI S'ADRESSE CE LIVRE BLANC ?

L'un des enjeux principaux de Docker est de rapprocher les équipes de développement de celles de la production. Docker permet, en effet, d'aisément installer une application, mais également de la modifier rapidement.

Ce livre blanc s'adresse donc à la fois aux opérationnels et aux développeurs mais aussi à tous ceux qui organisent la DSI ou les différents projets.

## CONTEXTE

Créée il y a tout juste 2 ans, Docker est la startup par excellence. 135 millions de dollars levés, évaluée en avril 2015 à plus de 1 milliard de dollars de capitalisation boursière, la société créée par des français fait le buzz dans la Silicon Valley. Elle attire l'attention des plus gros : Google, Amazon, IBM, Microsoft, tout le monde suit de près son évolution. Alors... simple bulle ou vraie révolution ? Et puis d'abord, c'est quoi Docker ?

# L'HEBERGEMENT AVANT DOCKER

Docker modifie en profondeur la manière dont on héberge les applications en créant des containers. Les containers sont des sortes de boîtes noires dans lesquelles différentes applications vont s'exécuter. La notion n'est pas vraiment nouvelle en informatique. En effet, dans d'autres domaines, on parle souvent de sandbox (ou bac à sable) pour isoler un processus du reste de la machine. Mais comme souvent le diable se cache dans les détails, avant d'aller plus en avant dans cette notion de containers, voyons où elle se positionne par rapport aux autres technologies d'hébergement déjà existantes.

Vous avez un site web à mettre en ligne. Vous allez donc vous adresser à un hébergeur. Jusqu'en 2013 et l'apparition de Docker, les hébergeurs vous offraient généralement ces options :

**UN HEBERGEMENT MUTUALISE** : c'est le domaine des « petits » sites webs, qui ne contiennent que du contenu, des sites amateurs ou des forums type PHPbb. Ce type d'hébergement -spécifique au monde PHP- a tendance à disparaître.

**UN SERVEUR DEDIE** : vous louez un serveur dans les locaux de l'hébergeur. Libre à vous d'installer ce que vous voulez dessus, vous êtes maître de la machine. En France, des sociétés comme OVH ou 1&1 sont spécialisées dans ce domaine avec des offres très compétitives.

**UN SERVEUR VIRTUEL** : vous louez une machine « virtuelle », dans le « cloud ». En réalité, votre machine virtuelle s'exécute sur un serveur bien réel (l'hôte), chez un hébergeur. Du point de vue de l'utilisateur cependant, le serveur virtuel se comporte comme un serveur dédié. Le serveur hôte alloue une partie de ses ressources (CPU, mémoire, réseau et disque) au serveur virtuel. Typiquement, un hôte sera une machine extrêmement puissante (64 ou 128 Go de RAM, 24 ou 48 cœurs, etc.) qui hébergera de nombreuses « petites » machines virtuelles. La technologie de virtualisation utilisée simule une machine complète. Cela signifie qu'il est possible de faire fonctionner une machine virtuelle Linux sur un serveur hôte Windows, ou inversement !

Les serveurs virtuels ont déclenché toute la folie marketing autour du « cloud », et ce n'est pas surprenant. Ils offrent en effet de nombreux avantages ! Une machine virtuelle est facile à administrer. On peut par exemple augmenter sa RAM d'un simple

clic. On peut la faire passer d'un serveur à un autre par une simple copie d'un fichier, etc.

Pour terminer ce tour d'horizon des solutions d'hébergement « pre-Docker », rajoutons une nuance à propos des machines virtuelles. Nous allons faire la distinction entre 2 types d'hébergements :

**LE SERVEUR VIRTUEL PUBLIC** est une offre dans laquelle le serveur que vous commandez fonctionne sur un hôte dont l'hébergeur est responsable. C'est par exemple le cas avec Amazon Web Services (AWS pour les intimes). Vous commandez un serveur virtuel, vous ne savez rien de l'hôte sur lequel il tourne. D'autres serveurs virtuels d'autres clients fonctionnent sur le même hôte. En fait, Amazon pourrait très bien décider de changer votre serveur d'hôte, cela ne vous concerne pas.

**AVEC UN SERVEUR VIRTUEL PRIVE**, au contraire, vous possédez toute l'architecture, donc tous les hôtes. Vous choisissez comment répartir les machines virtuelles sur chacun des hôtes à votre guise. Evidemment, le coût est sensiblement différent ! Des technologies ouvertes comme par exemple OpenStack permettent de créer un cloud privé. Leur installation est cependant fort complexe (n'espérez pas mettre en place OpenStack sans passer préalablement plusieurs semaines à apprendre le système !)

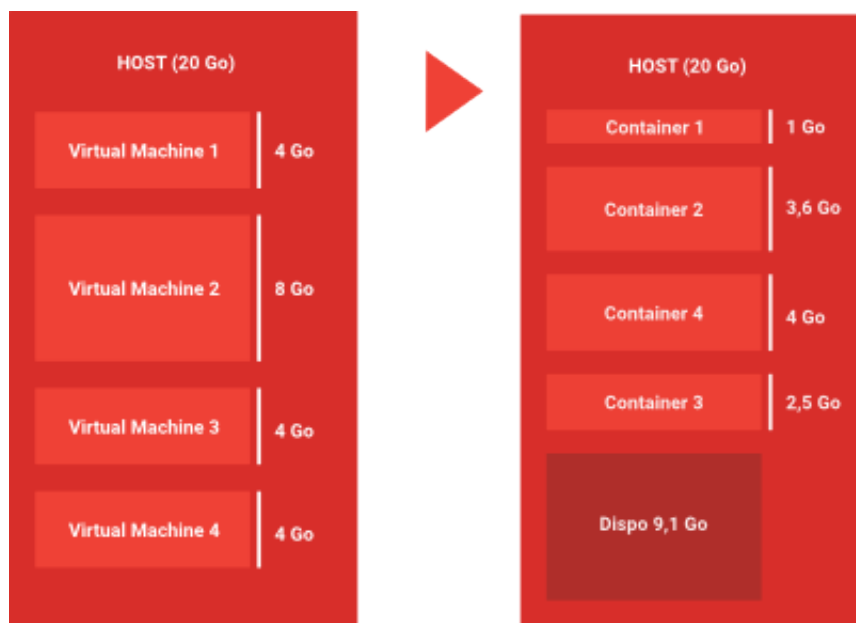
# L'ARRIVEE DES CONTAINERS

En 2013, l'arrivée des containers secoue le monde Linux. Pour être parfaitement exact, la notion de container existe déjà depuis 2008, mais n'a été jusque-là utilisée qu'à la marge.

Un conteneur est un espace bac à sable dans lequel s'exécute une application. Cela ressemble énormément à une machine virtuelle, à quelques détails près :

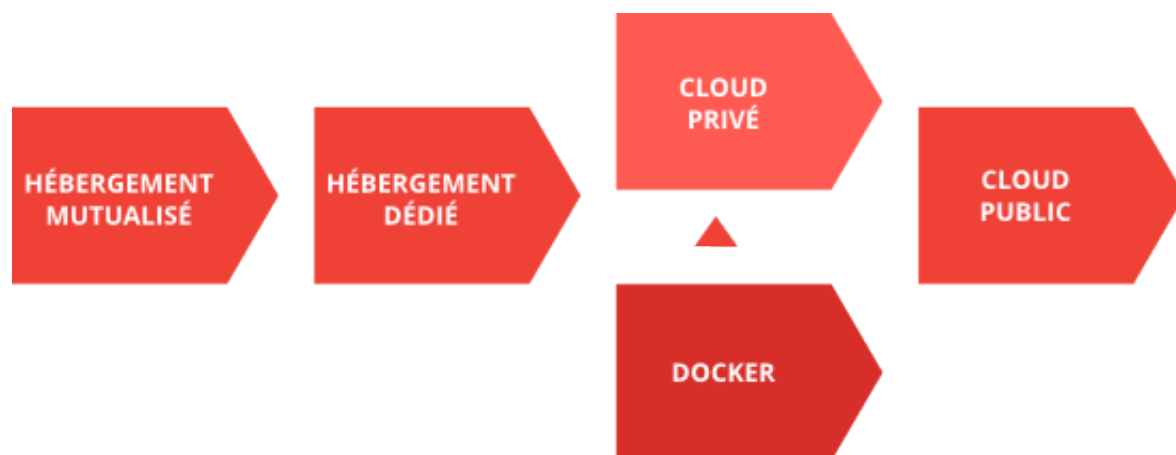
- Tous les conteneurs partagent le même noyau Linux. Cela signifie qu'un conteneur fonctionne forcément sous Linux. Il partage avec le système hôte la RAM, les CPU et l'espace disque.
- Les conteneurs sont donc beaucoup plus légers qu'une machine virtuelle (ils n'embarquent pas l'OS).
- Ils contiennent néanmoins tous les fichiers d'un système d'exploitation. Comme une machine virtuelle, un conteneur dispose de ses propres utilisateurs, de son adresse IP, et évidemment de ses applications.

On comprend mieux l'intérêt de partager le noyau avec les autres containers sur le schéma ci-dessous. Dans le cas de la virtualisation, les ressources doivent être allouées « en dur » aux machines virtuelles. Une machine virtuelle doit être dimensionnée pour tenir les pics de charges, donc on va lui allouer la quantité de RAM maximale dont elle pourrait avoir besoin. Dans le cas des containers, chaque container ne consomme que la RAM dont il a besoin à un instant « *t* », laissant de la place pour plus de containers sur la même machine.



Pour faire simple, avec un même serveur, on gère plus d'applications, donc on fait des économies ! Succès assuré, surtout auprès de gros consommateurs de serveurs comme Google !

Cet avantage est aussi un inconvénient potentiel. Puisque les containers partagent le même noyau, un container pourrait s'attribuer toutes les ressources d'un serveur. Même s'il existe des techniques pour éviter cela, on comprend que les containers ne soient pas aussi efficaces que les machines virtuelles dans le cadre d'un cloud public. Les containers sont donc en fait naturellement en concurrence avec les offres de cloud privé, type OpenStack.



Evidemment, les containers seront installés sur un serveur hôte, qui sera la plupart du temps un serveur dédié, mais qui pourrait aussi être un serveur virtuel !

À ce niveau de lecture de ce livre blanc, vous devez déjà avoir une idée de la réponse à la question « Est-ce que Docker est fait pour mon entreprise ? ». Si vous avez ressenti le besoin de disposer d'un cloud privé, si vous multipliez les serveurs pour de petites applications ou si au contraire vous avez de nombreuses applications hébergées sur un seul serveur et que vous souhaitez les stocker dans des environnements bien séparés, il est probable que la réponse soit oui.

# DOCKER, UN OUTIL POUR LES DEVELOPPEURS

Cela ne vous aura pas échappé, jusqu'ici, nous avons surtout parlé hébergement. Mais ici, chez TheCodingMachine, notre métier, c'est plutôt le développement web. Alors pourquoi écrire un livre blanc sur une technologie plutôt utilisée par nos partenaires hébergeurs ? Il doit bien y avoir quelque chose de particulier pour les développeurs, non ?

Tout juste !

Vous avez sans doute déjà entendu parler de mouvement « DevOps », qui vise à réduire la friction entre les équipes de développement (dévs) et les équipes opérationnelles en charge de la maintenance des serveurs (ops). Et bien Docker s'inscrit précisément dans ce mouvement.

## LE DOCKERFILE

Afin de mettre en place un container Docker, il faut indiquer à Docker la liste des commandes à exécuter pour installer les applications que l'on souhaite mettre dans le container. Cette liste de commandes est saisie dans un fichier nommé Dockerfile. Et qui rédige ce Dockerfile ? Le développeur !

Docker déclenche donc un transfert de responsabilité des « ops » vers les « devs ». Avant Docker, le choix des applications installées (base de données, serveur web, etc.) était à la main des « ops ». Souvent, les développeurs n'avaient pas la main sur la machine de production pour installer l'application. Avec Docker, le « dev » fournit un container dans lequel il a configuré lui-même les applications.

Est-ce une bonne chose ?

Oui, pour plusieurs raisons :

- Le développeur a souvent une très bonne connaissance des applications qu'il utilise et sait les configurer très précisément.
- Cette configuration est de plus stockée avec le code (donc versionnée dans un dépôt de code type GIT).
- Le fichier Dockerfile est « autodocumenté » et remplace avantageusement une documentation d'installation qui pourrait ne pas être toujours mise à jour.

- Les différents environnements (dév, tests, préprod, prod...) sont strictement identiques.

Attention cependant ! Le métier d'hébergeur impose des responsabilités que les développeurs n'ont pas l'habitude de porter :

- Le suivi des failles de sécurité ;
- La mise à jour régulière des paquets utilisés.

Ces 2 points ne sont pas aujourd'hui intégrés nativement dans les images Docker et demandent une attention particulière !

Enfin, il est important de noter que le format du Dockerfile est absolument trivial. Il n'est donc pas très compliqué de créer son propre container.

Vous ne me croyez pas ? Mettons les mains dedans alors ! Voici un exemple d'un Dockerfile qui crée un container contenant le serveur web Apache.

## DOCKERFILE

```
FROM ubuntu

RUN apt-get update && apt-get -y install apache2 && apt-get clean

ADD default.conf /etc/apache2/sites-available/default.conf

EXPOSE 80

CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

La première ligne (FROM) nous donne le point de départ pour la construction de notre container. Sous Docker, nous ne partons jamais de zéro, mais toujours d'un container que l'on étend. Dans notre cas, nous étendons « ubuntu », qui est un container possédant les fichiers du système d'exploitation éponyme. D'où vient ce « ubuntu » ? Du repository principal Docker. Nous parlerons de ce repository tout à l'heure.

La deuxième ligne (RUN) permet d'exécuter des instructions dans le container. Ici, les habitués des systèmes Debian / Ubuntu auront reconnu la commande déclenchant l'installation d'un serveur Apache.

La troisième ligne (ADD) permet de rajouter des fichiers à partir de notre serveur dans le container. Il s'agit généralement de fichiers de configuration spécifiques.



La quatrième ligne (EXPOSE) permet d'ouvrir le port 80. Il s'agit du port réseau utilisé par le protocole HTTP. Nous devons systématiquement préciser à Docker quels sont les ports qui sont accessibles depuis l'extérieur du container pour des raisons de sécurité.

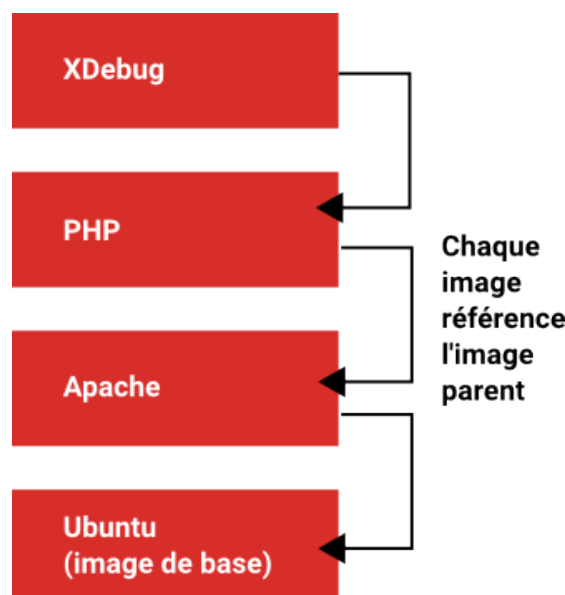
La dernière ligne (CMD) contient l'instruction qui sera lancée par défaut au démarrage du container. Ici, nous démarrons le serveur Apache.

## LE SYSTEME DE FICHIER DE DOCKER : UNION FILE SYSTEM

Il est très facile de créer et démarrer de nouveaux containers avec Docker. Mais quid de l'espace disque ? Puisque chaque container dispose d'un système de fichier d'un OS complet, ça risque de prendre de la place non ?

Et bien pas tant que ça. Les développeurs de Docker ont pensé à tout. Reprenons notre exemple précédent. Notre container « Apache » est basé sur un container « ubuntu ». Si on démarre 2 containers « Apache », on pourrait s'attendre à ce que les fichiers de Apache et de Ubuntu soient dupliqués. Mais au lieu de cela, seuls les fichiers modifiés dans un container et pas dans l'autre seront stockés en plus, grâce à l'Union File System, un système de fichier qui sait ne stocker que les deltas.

Ainsi, chaque Dockerfile créé en fait un layer sur lequel vont s'appuyer d'autres containers, qui seront eux-mêmes les layers d'autres containers...



## LA REGISTRY DOCKER

Puisque chaque container est basé sur un container parent, lorsque vous souhaitez construire votre propre container, la première question va être :  
de quelle image partir ?

Mais avant tout, qu'est-ce qu'une image ? Une image est créée à partir de fichiers de configuration Dockerfile qui proposent une description exacte de ce qui doit être installé sur le système. Les images contiennent tout ce que l'on souhaite y installer (un script à lancer, une base de données MySQL...) mais ne sont pas actives. C'est le container qui a le pouvoir d'interagir avec les applications présentes dans l'image (faire tourner un serveur par exemple).

Une des grandes forces de Docker, c'est justement d'offrir une « registry » (<http://registry.hub.docker.com/>). Il s'agit d'un site référençant un très grand nombre d'images (plusieurs dizaines de milliers de containers !)

N'importe qui peut se créer un compte dans la registry et venir déposer ses propres images (qui seront publiques).

Attention cependant, la plupart de ces images sont de très mauvaise qualité, ou non maintenues ! Il faut donc apprendre à faire le tri. Voici nos conseils :

- Utiliser en priorité des images officielles si elles existent. Celles-ci peuvent être soit :
  - Développées par les équipes de Docker (par exemple l'image ubuntu)
  - Développées par les équipes des projets eux-mêmes (par exemple l'image Mysql)
- Si l'application que vous utilisez ne dispose pas d'une image officielle, il est probable que quelqu'un en ait déjà développé une. Avant de vous précipiter dessus, vérifiez :
  - Le nombre de téléchargements et « d'étoiles », qui donnent une indication de la popularité de l'image
  - Les différents « tags », qui permettent de voir si l'image est supportée dans le temps par son auteur
- Si vous n'avez pas une grande confiance en une image, par exemple parce qu'elle semble non maintenue, il vaut mieux récupérer le code du Dockerfile de l'image et repartir de ce code pour votre projet (i.e. faire un fork).

Note : les images publiées sur Docker sont par défaut publiques. Docker offre néanmoins la possibilité d'héberger des images privées contre un abonnement

mensuel. D'ailleurs, le business-model de Docker semble être basé sur cet hébergement d'images privées.

Chez TheCodingMachine, on est un peu circonspects vis-à-vis de cette stratégie commerciale... on peut très bien utiliser Docker sans publier d'images privées dans la registry ! Nous n'avons pas besoin de publier nos Dockerfile pour les utiliser. Docker est très utile dans sa version gratuite et les services offerts par l'abonnement nous semblent limités. Bref, on voit mal Docker devenir une cash-machine juste avec ces abonnements. L'intérêt que portent les grosses entreprises à Docker et les tours de table successifs semblent être plus liés aux économies que permet de faire Docker (limiter le nombre de serveurs), plutôt qu'aux potentiels bénéfices à venir sur l'activité abonnement de Docker.

# ORGANISER LES CONTAINERS

Si vous allez voir la registry Docker, vous allez trouver de très nombreuses images. Par exemple, vous trouverez des images de serveurs Apache+PHP pour le serveur web, des images MySQL pour la base de données, des images Elasticsearch pour la recherche full-text, des images Postfix pour le serveur de mails...

Mais votre application web à vous, elle a besoin de tous ces composants. Vous vous mettez donc à la recherche d'un container Apache + PHP + MySQL + Elasticsearch + Postfix... Stop ! Vous partez dans la mauvaise direction !

Il y a un point important dont nous n'avons pas encore parlé ! Dans la philosophie de Docker, chaque container est censé faire une tâche et une seule. Donc votre application devrait réellement être composée de 4 containers :

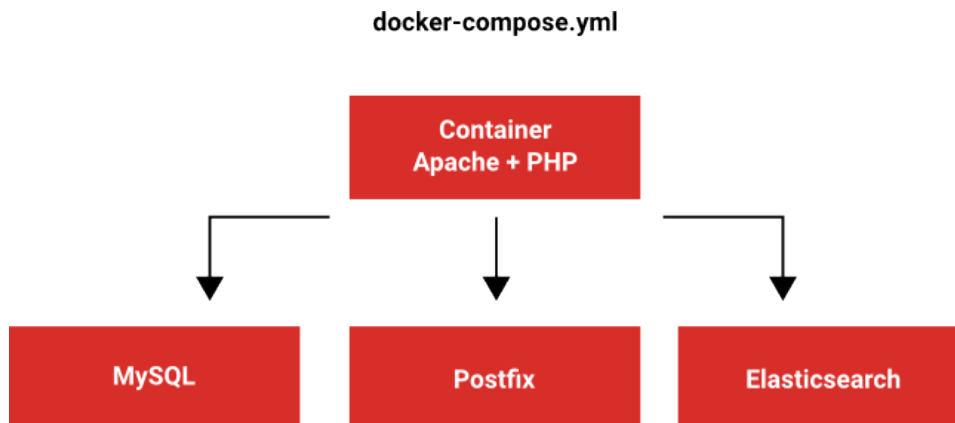
- Les containers MySQL, ElasticSearch et Postfix, que vous utiliserez « tels-quels »
- Le container Apache+PHP, que vous étendrez via un Dockerfile pour ajouter le code PHP de votre application à l'intérieur.

Il faut imaginer chaque container comme une sorte de boîte noire, gérant un micro-service, et qui peut être connectée à d'autres containers.

Et là, vous vous dites : « 4 containers pour une simple application web ? Mais cela ajoute de la complexité et de la maintenance ! Dans le cas d'une machine virtuelle, je pouvais mettre toutes mes applications au même endroit. Avec Docker, j'ai 4 éléments différents à administrer ! »

Effectivement ! Et pour gérer cette complexité, Docker propose des outils d'orchestration. Nous allons ici parler du plus connu : Docker-compose.

Avec Docker-compose, le devops écrit un simple fichier de configuration (au format YAML) qui décrit l'ensemble des containers composant une application, ainsi que la manière dont ces containers sont interconnectés que ce soit un accès réseau ouvert, un partage de système de fichier, etc.



Docker-compose est la glue qui rassemble les différents containers de votre application.

Cependant, comme souvent avec Docker, on détecte dans Docker-compose des défauts de jeunesse.

Voici notre expérience. Lorsque nous avons commencé à travailler avec Docker-compose (début 2015), celui-ci s'appelait FIG. C'était un projet communautaire. Le projet a été depuis repris officiellement par l'équipe de Docker, et le format du fichier a légèrement changé, forçant une petite adaptation.

En mars 2015, l'outil ne permettait pas de distinguer certains paramètres par environnement. On avait alors un fichier de configuration Docker-compose par environnement, ce qui occasionnait de la redondance donc de la double maintenance. Un mois plus tard, une fonctionnalité était ajoutée pour gérer le support de ces différents environnements...

Docker-compose est encore plus jeune que Docker et cela se ressent, avec un rythme de sortie de nouvelles versions assez important.

Egalement, Docker-compose dispose d'une limitation importante : il ne gère les environnements que sur une machine unique. Impossible donc d'utiliser Docker-compose pour une architecture distribuée sur plusieurs machines (load-balancing ou haute disponibilité).

Pour ce genre de besoins, on se tournera vers l'un des nombreux concurrents de Docker-compose :

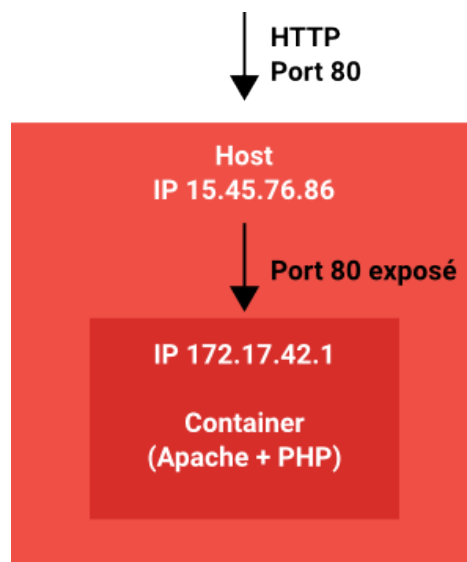
- Helios, développé par Spotify
- Machine et Swarm, fournis par Docker directement
- Shipyard
- Kubernetes, fourni par Google
- OpenStack, qui propose dans sa dernière version une gestion des containers en plus des machines virtuelles
- ...

Il y a plus d'une trentaine de solutions disponibles et concurrentes !

L'orchestration de containers Docker est un sujet en pleine ébullition, et une consolidation devrait avoir lieu dans les prochains mois. Ainsi, il est difficile à l'heure où nous écrivons ces lignes de savoir quels outils prendront le dessus et quels outils seront abandonnés. Nous ne pouvons que vous recommander la prudence si vous devez choisir un de ces outils.

# HEBERGER PLUSIEURS APPLICATIONS WEB SUR UN MEME HOTE

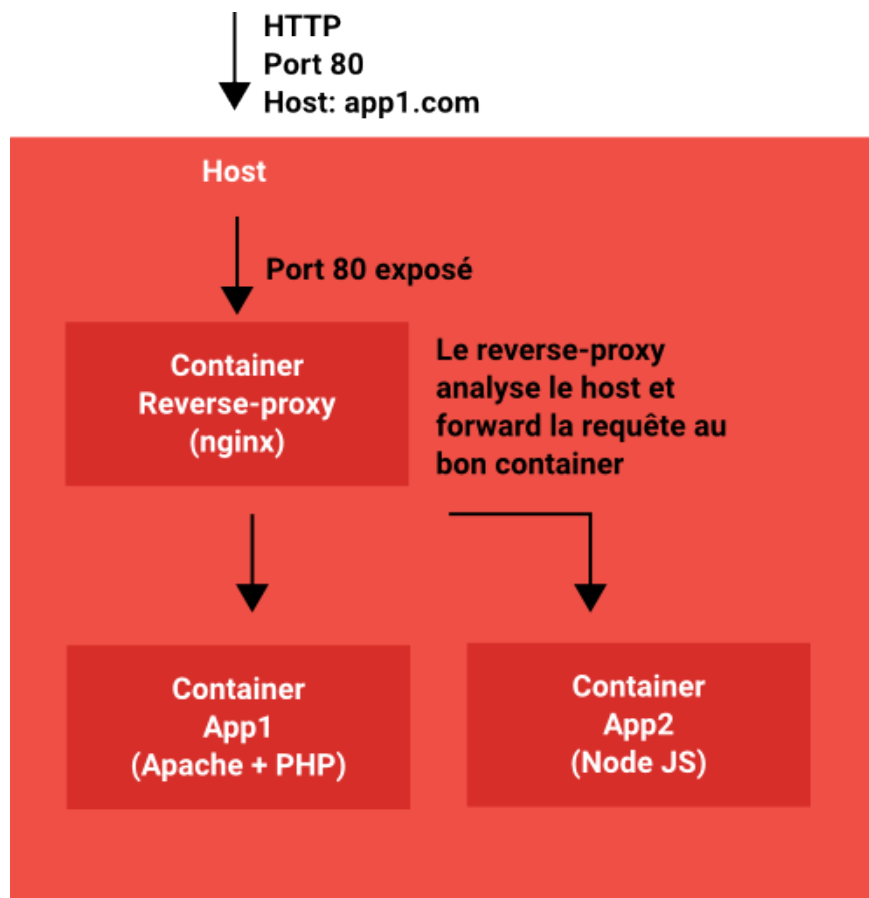
L'intérêt principal de Docker est de permettre d'isoler votre application web dans un container. Comme nous l'avons expliqué, un container dispose de sa propre adresse IP interne (commençant par 172.17.XXX.XXX). Le container n'est pas accessible depuis l'extérieur, sauf si un port est exposé. Dans le cas d'une application web, on exposera le port 80 (port par défaut pour le protocole HTTP).



Mais que se passe-t-il si vous voulez disposer de plusieurs containers (donc plusieurs applications web) sur le même hôte ?

Par exemple, vous avez peut-être 2 containers PHP et un container NodeJS qui souhaiteraient répondre sur le port 80. Hélas, un port ne peut être occupé que par un seul container !

Pour résoudre ce problème, Docker propose de passer par un « reverse-proxy ». Derrière ce nom barbare se cache un processus tout simple. C'est un autre container (généralement utilisant le serveur web Nginx), qui va écouter sur le port 80. En fonction du nom de domaine demandé, la requête sera automatiquement transférée au container concerné.



Point important à signaler, la configuration du « reverse-proxy » est extrêmement simple à mettre en place. Il suffit de rajouter une simple variable d'environnement dans les containers web contenant le nom de domaine pour que le « reverse-proxy » comprenne qu'il doit transférer les appels web du domaine au container.



# COMPATIBILITE OSX ET WINDOWS

La technologie des containers n'est disponible que sous Linux. Si la plupart des serveurs fonctionnent sous Linux, les développeurs, eux, peuvent utiliser une variété de systèmes d'exploitation.

Aujourd'hui, lors de la phase de développement, les développeurs utilisant Linux sont nettement avantagés par rapport à leurs homologues sous Mac ou Windows car ces derniers ne peuvent pas utiliser directement Docker. Il y a néanmoins une solution de contournement :

Boot2Docker fournie par Docker, est une machine virtuelle qui permet de démarrer un Linux contenant Docker sous n'importe quel système d'exploitation.

Hélas, dans la pratique, l'utilisation d'une machine virtuelle est fastidieuse. Par exemple, à l'heure où nous écrivons ces lignes, la machine virtuelle Boot2Docker n'est pas compatible avec Docker-compose, l'outil d'orchestration. Egalement, les écarts de droit entre la gestion des fichiers Windows et Linux peuvent causer des problèmes.

Heureusement, la technologie avance à grands pas. Si la solution n'est pas mature pour l'instant, gageons qu'elle le sera bientôt.

## DERNIERE MINUTE !

Nous allons publier ce livre blanc et une nouvelle vient juste d'être annoncée... Docker, CoreOS, Google, Amazon et Microsoft (entre autres) ont décidé de travailler ensemble pour créer un standard unique de déploiement de containers: the *Open Container Project*. Il s'agira d'un standard permettant l'interopérabilité entre des containers Docker (le standard *de-facto* jusque là) et d'autres containers. Quand on vous dit que les choses bougent vite dans le monde des containers.

# CONCLUSION

Alors Docker, simple mode ou vraie révolution ?

Révolution, assurément. Docker est une technologie très jeune, et pourtant déjà très ancrée auprès des devops. Il est clair que les avantages offerts par la technologie et sa facilité d'approche vont rendre l'utilisation des containers incontournable dans le futur. Mais doit-on adopter Docker tout de suite ?

Et bien tout dépend de votre besoin. La solution est neuve et bouge encore beaucoup :

- Suivre les évolutions de Docker peut être couteux.
- Egalement, la qualité assez moyenne des images existantes dans le repository impose un coût d'adaptation assez fort. La promesse de Docker est de fournir des images de containers comme des « boîtes noires » qui marchent du premier coup, mais force est d'avouer que pour l'instant, il est fréquent de devoir modifier ces images.
- Les bonnes pratiques d'utilisation commencent tout juste à émerger et ne sont pas encore universellement partagées, tant du point de vue développeur que du point de vue opérationnel.
- Les outils d'orchestration sont trop nombreux pour savoir lequel choisir pour une architecture complexe, avec un vrai risque de choisir un outil qui va périr.

Cependant, il ne s'agit que de défauts de jeunesse et d'ici quelques années, l'environnement sera plus stable et plus mûr.

La preuve ? Les levées de fonds successives de Docker assurent que la technologie va évoluer. Si Docker est une technologie qui ne fonctionne aujourd'hui que sous Linux, Microsoft est en train de travailler à une compatibilité Windows, preuve de l'intérêt des géants de l'informatique pour les containers.

Et c'est compréhensible ! À la clef, nous parlons d'économies importantes en termes de matériel et d'une automatisation de processus d'installation complexes.

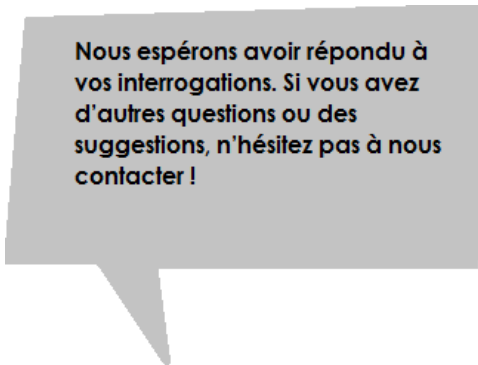
Alors nul ne peut lire l'avenir, et peut-être que Docker tombera aussi vite qu'elle est montée, mais une chose est sûre : la technologie des containers que Docker a réussi à démocratiser est là pour rester.

## A PROPOS DE THECODINGMACHINE

TheCodingMachine accompagne ses clients sur des missions de conseil technologique et sur des projets de développement d'applications Web.

Nous sommes spécialisés dans le développement de sites Internet, d'extranets (évidemment), d'intranets, d'applications Web métiers en PHP et en JavaScript.

Fondée en 2005, TheCodingMachine a piloté plus de 300 projets. Nous travaillons aussi bien pour des grands comptes privés et publics, pour des PME-PMI que pour des startups. Nous avons investi dès notre création dans la R&D, ce qui nous permet par exemple d'être à la pointe des technologies web (PHP, Node.JS, AngularJS, streaming video etc.).



**Nous espérons avoir répondu à vos interrogations. Si vous avez d'autres questions ou des suggestions, n'hésitez pas à nous contacter !**

[www.thecodingmachine.com](http://www.thecodingmachine.com)

Tél : 01 71 18 39 73

[contact@thecodingmachine.com](mailto:contact@thecodingmachine.com)

4, rue de la Michodière – 75002 PARIS