Docker - An open platform to build, ship and run any app, anywhere

Titouan FREVILLE

3 juillet 2016

Résumé

Cet article présente la technologie Docker et son utilisation pour la création d'outils et d'espaces de dévelopement uniformiser. Docker a pour objectif la création d'image de programme (d'unité complète et légère permettant de faire tourner un programme particulier) et son déploiement en cloud.

1 Qu'est ce que docker ?

1.1 Système de container

Docker est un système de containerisation de programme. Nous pouvons voir cela comme une machine virtuelle, en regardant de très loin. En effet, de même qu'une macine virtuelle, un container va faire tourner un système avec sa propre configuration isolé de notre environement local de travail. Quel différence donc entre machine virtuelle et container, et quel intérêt à utiliser l'un plutôt que l'autre.

1.2 Rappel sur les machines virtuelles (VMs)

Tout d'abord, rappelons ce qu'est une VM.

Une VM, c'est un "ordinateur" intégralement virtualiser. C'est à dire que à l'aide d'un logiciel, nous allons pouvoir simuler le foncitonnement complet d'une machine. Pour cela, le logiciel va devoir créer une simulation de toutes les couches nécessaires au lancement du système d'exploitation choisie. Nous aurons alors les couches suivantes entre notre environement et l'environement virtuel :

— Environement virtuel
— Système d'exploitation virtuel
— Noyeau et coeur du système virtuel
— Environement local

Ainsi, nous avons une plateforme simulant un environement vierge totalement distinct de notre environement local. Il est possible de passer des documents de l'un un l'autre via un système de dossier partager et les capactiés de la machine virtuel sont liées à celle de l'environement local et définis préalablement par l'utilisateur.

1.3 Les Containers

Voyons maintenant ce qu'est un container.

Un container, à l'égal d'une VM a pour objectif d'isoler un système fonctionnel. Cependant, la ou une VM recréer un système complet, un docker va ce contenter de recréer l'architecture necessaire au lancement et fonctionnement d'un programme. Docker est un système de container basé sur les distribution Linux (debian, ubuntu principalement) sur les quels ils tournent nativement. Et son application sur des environement Linux qui nous intéresse ici. En effet, Docker propose des outils pour tourner sur Windows ou Mac OS mais ceux si sont basé sur Docker machine, qui est une VM linux créer par docker pour faire docker (donc extrèmement légère comparée à une VM standart développer pour créer un système complet). Le container, que soit nativement sous Linux ou depuis Docker machine, va donc se lancer en utilisant les couches système de l'environement local. Aussi, la ou vous ne pourrez trouver des informations relative à la VM que si vous le demander, vous les retrouverer à la base de votre système linux (dans un répertoir /etc/docker). Aussi, seul les utilitaires indispensable doivent être présent dans un container (par défault, seul les Binary et les Library sont présente dans les container). De même que pour les VMs l'exposition de fichier doit être spécifier (ici, par des volumes qui peuvent être des répertoires dans un autre container ou des répertoir du système local). Nous allons donc avoir le système de couche :

- Environement virtuel
- Docker Engine
- Environement local

1.4 Images

Nous pouvons donc voir à quel point la structure de Container et proche mais distincte de la structure de VM. Les containers sont plus léger que les VMs mais moins complet. Aussi, on utilisera les container pour faire tourner des processus unique (un processus = un container) la ou une VM pourra supporter un serveur complet par exemple. Aussi, les containers sont des éléments préconfigurer, et ils est peut recommander de les configurer au lancement du container (bien que ce ne soit possible), mais plutôt d'utiliser des variables d'environement si des paramètres doivent être adapté en cours

d'activité du container. Il est donc préférable (aujourd'hui car Docker a pour but de supprimer ces problèmes) de laisser les activités à forte configuration (ex Base de données) ou nécessitant de fort paramètres de sécurité à des VMs standart ou des serveurs dédiés.

Maintenant, comment partager ces containers ?
Dans le cas des VMs, il est possible de les exporters via le logiciel utiliser ou d'en donner l'accés en résaux, via d'autre logiciel. Nous avons alors accés à l'image de la VM qui nous permettra de la lancer tel que configurer. Il existe un procéder similaire pour Docker, qui a été conçus pour le partage et le déploiement facile des containers. Aussi, le container représente l'unité fonctionnel lancé. Mais la première unité créer, appelé image, va être l'équivalent de l'image d'une machine virtuel. C'est à dire une capture des processus et du système de fichier nécessaire au bon fonctionnement de la machine ou, dans notre cas, du container. Aussi, en utilisant Docker, nous créeons des images ayant un nom et pouvant avoir un Tag (ex : docker_init :exemple docker_init est le nom de l'image, exemple son tag). Si nous observons une image de plus près, nous verrons qu'elle est composer de plusieur couches (des layers) chacune représentant un processus actif ou un système de fichier copier. Aussi, des images ayant le même nom mais des tags différent peuvent partager leur sous images (par exemple : deux images hello_world :Supinfo et hello_world :Whale, ayant pour but d'afficher le message en tag, vont pouvoir partager les layers représentant le système d'exploitation et le script copier pour afficher le message. Par contre, le layer correspondant à la commande exécuter par défault lors du lancement de l'image seront différent.) Ce système permet de partager sur des plateforme adapté (ex : Dockerhub (public), Portus (privé pour entreprise), etc ...) des images adapté aux besoins et prenant le minimum d'espace, un layer n'étant stocker qu'une seule fois puis référencer en fonction des tags.

Conclusion

Nous avons donc comparer deux méthodes pour virtualiser des applications. D'un côté les machines virtuelles, qui recrée intégralement un contexte permettant d'exécuter les applications. De l'autre, le système d'image mis en place par Docker, qui va utiliser une grande part du système local, et simuler uniquement les exétuables et Librairie propre au programme que l'on cherche à lancer. Les machines virtuelles sont des outils très puissant, mais aussi très couteux en ressources, du fait de la virtualisation complète d'un système. A l'opposé, les images ont été pensé pour être le plus légère, et il est très fortement conseillé de maintenir cet état de fait dans vos propres images, en incluant uniquement ce qui est nécessaire au processus à lancer (processus unique pour chaque image). Les deux technologie sont donc, aujourd'hui plus complémentaire que rivale. Les images Docker vont être

très pratique dans l'optique d'un déploiement en cloud de votre application et son maintient. Etant très légère, elle se déploient très rapidement, et sont instantanément opérationnel. Les machines virtuelles, elles seront plus utiles pour mettre en place des serveurs ou des sytèmes lourd en configuration nécessitant plusieurs processus simultanée. Il est évidemment possible de lier des container Docker entre à l'exécutione, mais leur lien n'est pas la même que s'ils étaient lancer sur un même environement, notemment au niveau du partage des processus et données.

2 Pourquoi utiliser docker ?

L'intérêt d'un container plutôt qu'une VM est plutôt apparent. Il tendent à remplacer les machines virtuelle en procurant une alternative plus légère et plus simple à mettre en place dans de nombreux cas. En effet, configurer un container et y faire tourner votre programme revient à copier les fichiers nécessaire à l'interrieur du container, se basé sur la bonne image (une image est un container non lancer, nous reviendrons plus en détails sur cette notions plus tard.) et lancer la bonne commande.