1 - Les conteneurs

1. Les conteneurs

1.1 Histoire des conteneurs

Le principe n'est pas tout jeune, vous avez sans doute déjà entendu parler du principe de **« chroot »** ou de « BSD jail ». Le chroot est un appel système qui a été introduit sur la version 7 d'Unix en 1979. Puis repris en 1982 par BSD, le terme de **« jail »** (emprisonnement) sera introduit en 1991.

Par la suite, FreeBDS étend le concept en ajoutant la commande « jail » au début des années 2000. Deux ans plus tard, Nicolas Boiteux rédigera l'un des premiers articles décrivant une méthodologie simple pour créer un emprisonnement sous Linux en se basant sur la commande « chroot ».

Finalement, la démocratisation du chroot dans les Shell Linux donnera naissance quelques années plus tard (en 2003) aux **premières plateformes de microservices**. Ces dernières offriront les prémices des solutions de SaaS ou de PaaS avec l'apparition d'une facturation à l'usage. On voit ainsi apparaître les prémices du cloud computing mais également des technologies qu'on appellera bien plus tard « containers » ou **« conteneurs »** en français.

Il faut cependant attendre **2008** avec l'arrivée des LXC pour **« LinuX Containers »** qui vont utiliser pour la **première fois** la **terminologie de conteneur**.

Il va finir par gagner en popularité en **2013** suite à son inclusion dans le **kernel Linux 3.8**.

Figure 1 : Logo des conteneurs LXC

Cette même année, une petite société française nommée **dotCloud** sort la **première version open source de son produit**.

Rebaptisé **Docker** par la suite, ce logiciel surcharge le format de conteneur Linux standard (LXC) , grâce à une **API** qui va **isoler les processus**. À ce moment-là, Docker utilise **LXC**, les **cgroups** ainsi que des f**onctionnalités du kernel Linu**x pour fonctionner.
Huit mois après l'arrivée de la première version sur Github, le projet a été mis en favori plus de 7 300 fois, en faisant le 14ème projet le plus populaire de la plateforme, avec plus de 900 forks et 200 contributeurs.

Fin 2017, le projet a été mis en favori plus de **46 000 fois** sur GitHub, avec plus de 13 500 forks et 1 700 contributeurs.

Il a littéralement **explosé** en **quatre ans** pour en faire **un des projets les plus plébiscités de ces dernières années**.

1.2 Principe de conteneur

Revenons à la **commande chroot**.
Comme nous l'avons expliqué précédemment, elle permet d'**isoler l'exécution d'un programme** et ainsi d'éviter qu'un système puisse être intégralement compromis lors de l'exploitation d'une vulnérabilité.
En faisant ainsi, un pirate informatique qui profiterait d'une faille présente sur une application « chrootée » n'aurait accès qu'à un environnement isolé et non plus à l'ensemble du système d'exploitation. Limitant ainsi les dégâts qu'il pourrait causer sur la machine.

Tout comme la virtualisation, la technologie de **conteneur** permet de **faire fonctionner en parallèle plusieurs services sur une seule et même machine hôte**.

Il devient possible d'exécuter facilement des applications 32 bits sur un système 64 bits. Pour cela, il suffit pour cela d'avoir un sous-système qui intègre toutes les bibliothèques logicielles nécessaires compatibles 32 bits.

Contrairement à la virtualisation où l'on vient cloner un PC complet, système d'exploitation compris, les **conteneurs** eux n'utilisent qu**'un seul OS** et **en partage les ressources et les librairies**.

Pour bien comprendre le fonctionnement des conteneurs, il faut revenir à la base des systèmes d'exploitation **GNU/Linux**. J'insiste sur cette appellation, car il existe un abus de langage, en effet, 90% des OS que nous appelons « Linux » sont en réalité des systèmes GNU/Linux.

Pour illustrer mes propos, je reprendrais une phrase de « Linus Torvlad » le Créateur du noyau Linux :
« Malheureusement, un noyau seul ne vous sert à rien. Pour avoir un système qui fonctionne, vous avez besoin d'un interpréteur de commandes, d'un compilateur, d'une bibliothèque, etc. Ce sont des éléments séparés et ils peuvent être sous copyright plus strict (voire plus permissif). La plupart des outils utilisés avec Linux sont des logiciels GNU et sont sous licence copyleft GNU. Ces utilitaires ne sont pas dans la distribution. Contactez-moi (ou contactez GNU) pour plus d'informations. »

Pour avoir un **système d'exploitation fonctionnel**, il vous faut donc l'**écosystème GNU** ainsi que le **noyau Linux**.

Il faut cependant faire attention : **il est tout à fait possible de faire fonctionner le kernel Linux sans logiciel GNU**, c'est le cas d'Android.

Maintenant que toute ambiguïté est levée, retournons à nos conteneurs.

Le **principe** est donc d'**encapsuler une application** mais **pas tout le système d'exploitation** comme on le ferait avec une VM. Ici, on **garde le même noyau Linux** et on **partage les librairies et binaires** déjà présent.

Nous allons ainsi passer d'une application d'une VM qui fait **quelques Giga Octets** à un conteneur qui pèse **quelques centaines de Méga Octets** pour héberger la **même application**.

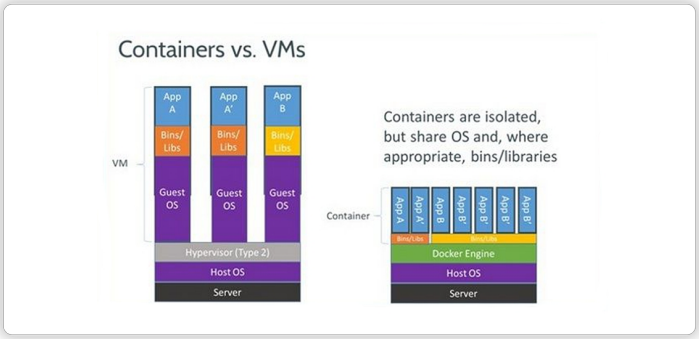

Figure 2 : Machines Virtuelles vs Conteneurs

Comme vous pouvez le deviner, les conteneurs sont **fortement couplés** à l'utilisation du **noyau Linux.** Microsoft travaille d'arrache-pied avec Docker afin de porter le principe sous Windows. Vous trouverez quelques applications compatibles sur un internet, mais il reste encore énormément de chemin à parcourir.

1.3 Les conteneurs et le DevOps

a. Point de vue développeurs

Les **déploiements** entre les différents environnements (DEV, QA, UAT, PROD ...) se trouvent **grandement accélérés** par les containers Docker, car ils sont **très légers**. Les basculer d'un environnement à un autre peut donc se faire en quelques secondes, ce qui n'est pas le cas pour la VM, beaucoup plus lourde.

Du fait de la disparition de l'OS intermédiaire des VM, les **développeurs** bénéficient aussi d'une **pile applicative plus proche de celle de l'environnement de production**, ce qui engendre mécaniquement **moins de mauvaises surprises** lors des **passages en production**.

Dans un **contexte de microservices**, les conteneurs pourront permettre de concevoir une **architecture de test plus agile**, chaque **container** de test pouvant intégrer une **brique de l'application** (base de données, langages, composants...).

Pour tester une nouvelle version d'une brique, il suffira de **mettre à jour le container correspondant**.

Même du côté du **déploiement continu**, la technologie de conteneurs Docker présente un intérêt, car elle permet de **limiter les mises à jour du container** uniquement au **delta entre deux versions**, mais nous verrons ce mécanisme plus en détail sur la fiche « Docker ».

b. En production

C'est grâce au concept d'architecture en **microservices** et aux **containeurs Docker** que l'on peut **isoler chaque composant logiciel** sur **une seule et même machine** de façon **légère** et **robuste**.
Ces containers de composants, du fait de leur légèreté, peuvent eux-mêmes, chacun, reposer sur les ressources machines voulues.

Pour parvenir au même résultat, les outils de virtualisation ont besoin d'un pool de VM inactives provisionnées à l'avance.

Avec Docker, **nul besoin de pool**, puisqu'un container démarre en quelques secondes.

Mais, la promesse des conteneurs Docker va plus loin, car ils sont **facilement portables d'une infrastructure à l'autre**. Il devient donc possible de réaliser du « mirroring » (fonctionnement en miroir actif/actif ou actif/passif) d'application avec équilibrage de charge entre plusieurs clouds.

Il facilite également les plans de reprise et/ou continuité d'activité entre clouds.
Vous pouvez ainsi avoir toute votre production qui tourne sur le cloud provider A et des scripts des déploiements qui vous permettent de redéployer toute la production sur un cloud provider B, si jamais le premier cloud venait à avoir une interruption de service majeur.

Comme vous pouvez le devinez, les conteneurs et principalement l'arrivée de Docker ont permis d'**accélérer l'application des méthodes Agile** au sein des équipes DevOps.

Pour voir la fiche complète et les documents attachés, rendez-vous sur
https://elearning.26academy.com/course/play/5ac4f38a5bdbeb48ce2758cf