Chapitre 4
Rappel des notions techniques de base

1. Virtualisation

La virtualisation est vue aujourd'hui comme le fait de pouvoir émuler les ressources matérielles. L'hyperviseur sert de couche d'abstraction qui permet l'isolation de chaque machine virtuelle de telle sorte qu'un utilisateur connecté à une de ces machines virtuelles n'aura pas conscience qu'il n'est pas sur un vrai système physique.

Pourtant, la notion de virtualisation va bien au-delà de cette vision actuelle. Cette dernière, se concentrant sur l'aspect isolation, en oublie le premier concept qui est le partage des ressources. Historiquement, l'origine de la virtualisation pouvait aisément se confondre avec le début même de l'informatique, après le temps des gros systèmes mono-utilisateurs, fonctionnant par batch. C'est IBM qui introduit, vers la fin des années soixante, le concept de CP/CMS. Ce système de virtualisation très avancé, et dont les concepts ont été repris par les autres systèmes plus récents, demeure à ce jour utilisé dans les séries « z » du matériel IBM.

Le composant CP, pour *Control Program*, met à disposition de l'utilisateur une émulation d'un S/360 dédié. Pour ne pas se retrouver dans des réplications de plusieurs systèmes S/360, le CMS, pour *Consol Monitor System*, s'occupe du partage des ressources et de leur déduplication. En faisant tourner un CMS dans chaque CP, la surcharge sur le système physique est largement diminuée. CP/CMS a permis l'exécution d'applications qui ne sont pas écrites spécialement pour faire du *time sharing*.

Des systèmes multitâches et multi-utilisateurs sont ensuite apparus et ont conquis le marché des serveurs et des ordinateurs en général en exploitant certaines des fonctionnalités de ces techniques de virtualisation. Par exemple, Microsoft, dans son système Windows, utilise un composant essentiel HAL.dll (*Hardware Abstraction Layer*), qui lui permet de tourner indifféremment sur plusieurs matériels. Sans compter les systèmes introduits par des constructeurs comme HP sur ses Superdomes ou IBM pour le partitionnement sur les pSeries.

1.1 Le boom de la virtualisation

Vers la fin des années 1990, le marché de l'informatique est dominé par, d'une part de gros serveurs multitâches et multi-utilisateurs dans les entreprises, et d'autre part des ordinateurs personnels relativement accessibles basés sur de l'architecture x86.

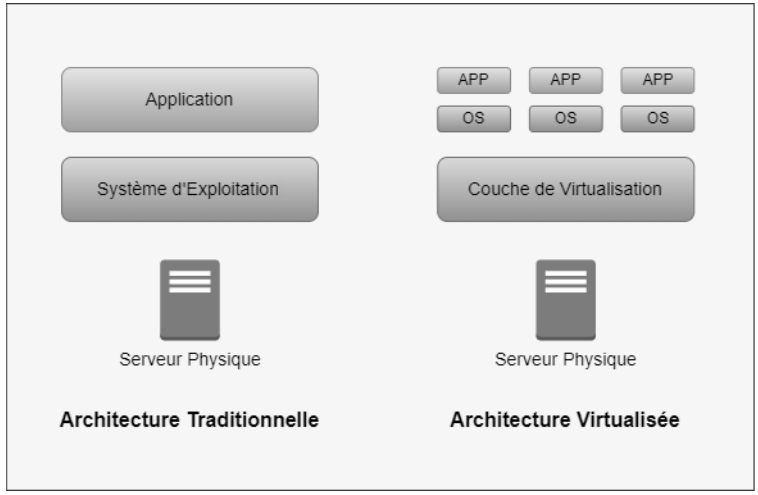
Le constat d'alors est que, la plupart du temps, les ressources investies pour le matériel ne sont pas utilisées. Typiquement, elles ne le sont qu'entre 10 % et 25 %. Mais, malheureusement, ajuster les ressources à ces niveaux d'utilisation introduit la problématique d'absorption des pics d'utilisations. Est arrivé à cette époque l'avènement des architectures distribuées pour optimiser l'utilisation des ressources. La vision idéale, et qui l'est toujours, c'est une attribution de ressources à la demande.

C'est VMware, au début des années 2000, qui lance un premier système de virtualisation avec son ESX 1.0. Le concept est simple, le déploiement l'est relativement aussi et le provisionnement des serveurs a gagné en souplesse. Dans la foulée sont venus Xen Project, puis Hyper-V de Microsoft et Docker, que nous aborderons dans un autre chapitre, pour les acteurs les plus en vue côté entreprise.

La virtualisation a permis une économie substantielle sur les serveurs et, mécaniquement, en espace de Data Center avec des bénéfices engendrés au niveau des consommations énergétiques ainsi que la préservation de l'environnement.

1.2 Fonctionnement de la virtualisation

La virtualisation, dans l'acception comprise actuellement, excluant donc la conteneurisation, fonctionne par abstraction et isolation des ressources matérielles utilisées par la couche logicielle, système d'exploitation comprise, par un hyperviseur. L'hyperviseur reconnaît le matériel sous-jacent, que ce soit le processeur, la mémoire, le stockage et le réseau et en crée des représentations virtuelles. Ces représentations virtuelles sont assez souples pour permettre une relation 1:n entre les ressources physiques et virtualisées. L'hyperviseur fait alors le lien entre ces ressources en assurant la transmission des données et des instructions.

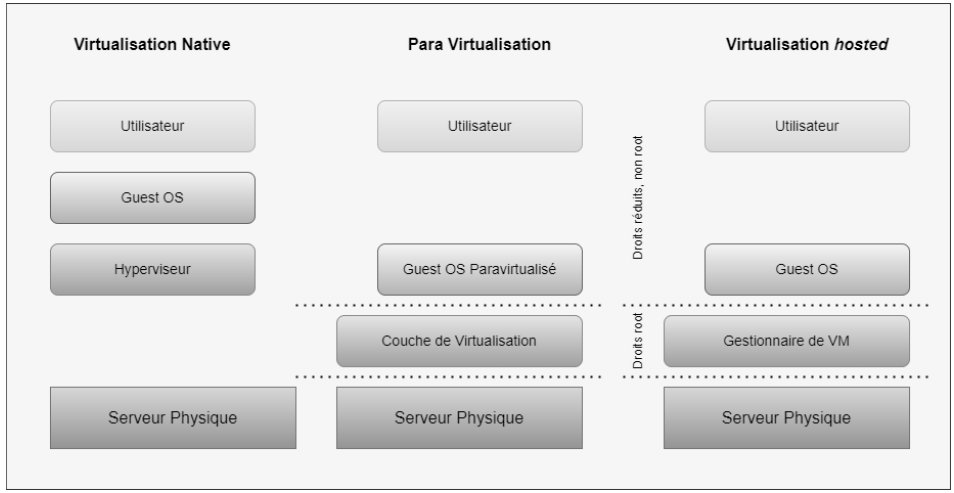


Architecture traditionnelle vs Virtualisation

L'hyperviseur étant responsable des échanges entre le matériel physique et virtualisé, il devient facile de découpler une machine virtuelle de son matériel physique, la machine virtuelle n'interagissant qu'avec l'hyperviseur. Cet état de fait a introduit un concept très difficilement implémentable sur des configurations traditionnelles : la portabilité. Les hyperviseurs actuels sont ainsi capables de migrer des machines virtuelles entre plusieurs serveurs physiques sans interruption de services. C'est un gain substantiel et évident lors d'opérations de migration.

1.3 Les types de virtualisation

Bien qu'on ait souvent en tête la virtualisation VMware, toutes les technologies de virtualisation ne se ressemblent pas. On en distingue plusieurs types : la virtualisation de Type I ou type *natif*, celle de VMware par exemple qui est une virtualisation complète où un micrologiciel faisant office d'hyperviseur gère tout et opacifie le matériel sous-jacent vis-à-vis des machines virtuelles. Il y a aussi la paravirtualisation, où les machines virtuelles ont au moins connaissance qu'elles sont... virtuelles. Et enfin, la virtualisation de Type II ou type *hosted*, qui est directement implémentée dans un système d'exploitation classique, dont une partie fournit le niveau d'isolation. L'abstraction est alors simplifiée par le partage avec le système d'exploitation source des bibliothèques d'instructions spécifiques. À l'instar de CMS du CP/CMS, il y a une mise en commun des ressources des machines virtuelles, résultant à des machines virtuelles légères et très rapides. Solaris Zones est par exemple implémentée de la sorte. Ce type de virtualisation est à l'origine des conteneurs, sujet qui sera vu dans un prochain chapitre. À noter que sont considérées comme virtualisation de Type II, puisque basées sur un OS installé, les virtualisations de la famille VMware Workstation ou Virtual Box.



Les types de virtualisation

1.4 Machine virtuelle

Une machine virtuelle est tout ou partie d'un système d'exploitation qui est installé sur du matériel qui lui est dédié et qui est émulé et géré par un hyperviseur. L'utilisateur qui interagit avec une machine virtuelle est censé avoir la même expérience que sur n'importe quelle machine physique.

Autant il est habituel de classifier les hyperviseurs en *Natif*, *Hosted* ou *Paravirtualisé*, autant cela l'est moins pour les machines virtuelles. Pourtant, elles peuvent être classées en deux catégories :

– Les machines virtuelles de type processus : la machine virtuelle est alors temporaire et n'existe que pendant l'exécution du processus. C'est le cas par exemple de la machine virtuelle Java, JVM, et aussi du *Common Language Runtime* du framework .NET.
– Les machines virtuelles telles que communément connues : celles qui tournent et qui sont managées via des hyperviseurs.

2. Conteneurisation

2.1 À l'origine : cgroups et namespace

Le concept de conteneurisation, même s'il n'est pas explicitement nommé, est présent dès les débuts de l'informatique. Dès que le partage des ressources devenait une problématique, l'isolation de l'utilisation de celles-ci devenait une priorité. De par sa nature même, le cœur du système informatique est séquentiel : un processeur n'exécute qu'une instruction à la fois. Malgré l'introduction du multithreading, ou des architectures multiprocesseurs, cet état de fait demeure et rend contre-intuitif l'exécution isolée et parallélisée des processus, pourtant maintenant largement démocratisée. L'isolation inhérente à la parallélisation a toujours autant été aussi importante que ne peut l'être la parallélisation.

Dans les années 1970, dans le développement des premiers noyaux Unix, **chroot** existait déjà. C'est la capacité de pouvoir créer un sous-système dans le système originel en changeant le système de fichier racine à un autre endroit. Dans les noyaux BSD, un peu plus tard, fût introduite aussi la notion de **jail**, représentant des sous-systèmes entiers étendant **chroot** avec des fonctionnalités réseau et ayant la capacité de se voir assigner des adresses IPs. De même, des technologies d'isolation propriétaires de grands constructeurs de serveurs, Sun Microsystems ou IBM, sont apparues mais ces dernières ont mûri de leur côté sans jamais vraiment interférer dans le cheminement vers ce qui est actuellement la conteneurisation. Le début des années 2000 a marqué une séparation claire d'avec ces concepts d'isolation et une tendance très largement marquée pour le modèle Hyperviseur. Pourtant, quelques technologies de type conteneur ont émergé dont, parmi les plus avancées, celles de Solaris, Linux VServer ou encore Open Virtuzzo. Et c'est en 2013, avec l'apparition de Docker, que le monde de la conteneurisation a connu sa plus grande révolution. En soi, la technologie est connue et stable depuis LXC, aîné de cinq ans et partie intégrante des premières versions de Docker, mais Docker a apporté une touche universelle à la technologie avec un environnement riche et une orientation PaaS autour d'un catalogue d'images et un repository très fourni se détachant des usages jusque-là obscurs de LXC et des autres technologies.

Pourtant, à la base de toutes les technologies de conteneurisation se retrouvent deux concepts : *namespace* et *cgroup*.

2.1.1 Namespace

La notion de *namespace* a été introduite dans le noyau Linux au début des années 2000, rejointe quelques années plus tard par les *cgroups*.

Un *namespace* est un mécanisme de partitionnement implémenté dans le noyau Linux qui permet à un ensemble de processus de voir et d'accéder qu'à un ensemble de ressources. D'autres processus externes au namespace ne peuvent alors pas accéder à ces ressources.

La définition est un peu alambiquée, mais le *namespace* est le premier concept compris de la conteneurisation : un environnement qui s'exécute indépendamment d'autres environnements sur un même système.

À date, le noyau Linux implémente plusieurs types de *namespaces* :

– Le *namespace utilisateur* qui permet d'isoler les utilisateurs et groupes. Les processus et ressources étant assignés à ce *namespace*, ces utilisateurs peuvent avoir le privilège *root* à l'intérieur dudit *namespace* sans pour cela interférer avec le reste du système. C'est par exemple le système qui permettra à un utilisateur sans privilège de faire tourner un démon avec privilège dans un conteneur.

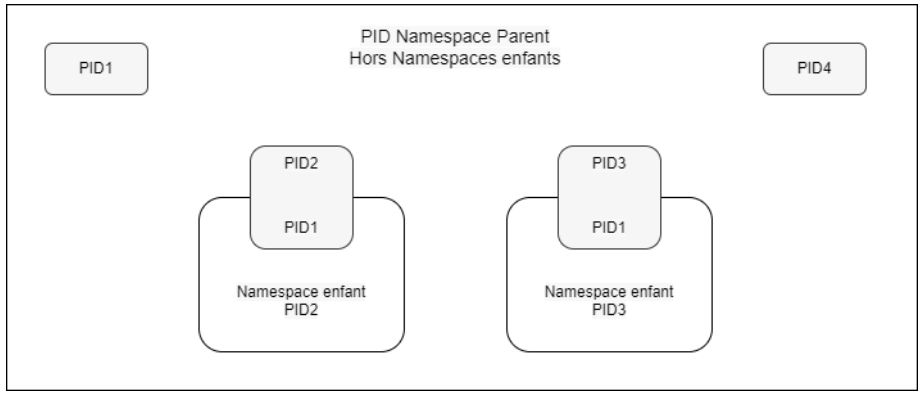
– Le *namespace processus* : c'est le fait d'affecter un ensemble de hiérarchie à l'image du process parent *init* aux processus dans ledit *namespace*. De ce fait, les processus seront organisés tels que le serait un système classique : le premier, parent de tous les autres processus, aura le numéro PID1.

– Le *namespace réseau* qui regroupe tout un univers réseau indépendant avec sa propre table de routage, son ensemble propre de sockets, sa ou ses configurations IPs, ainsi que plusieurs autres objets réseau tels que des routeurs, du DHCP... Cette notion de *namespace réseau* est largement utilisée dans les configurations Linux bridge de la couche Neutron.

– Étant donné que toutes les couches du système d'exploitation ont leur *namespace*, la couche stockage n'est pas en reste avec le *namespace mount* : dans ce *namespace*, des points de montages et systèmes de fichiers peuvent être créés indépendamment sans affecter le système hôte.

– Enfin, afin de pousser jusqu'au bout la logique d'isolation, il existe aussi des *namespaces IPC* sous forme de queues de messages pour les communications interprocessus ainsi que des *namespaces* dits *UTS* pour *Unix Time Sharing*, qui permettent d'émuler un nom d'hôte et même un nom de domaine complètement différent.

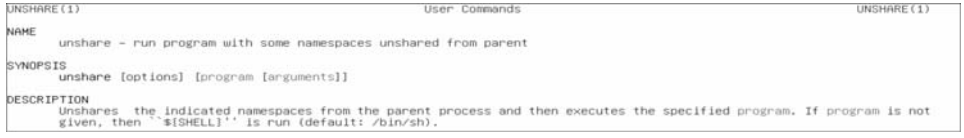
Voici un schéma qui rend compte d'un *namespace* de type processus, avec la description de la relation entre chacun des processus :



Processus et namespaces

Dans l'exemple illustré ici, les processus dans le *namespace* parent, de PID1 à PID4 se voient, accèdent et se partagent des ressources. Par contre, le PID1 dans le *namespace* de PID2 ne verra ni les processus dans le *namespace* parent, par exemple PID4, ni le PID1 qui est dans le *namespace* de PID3.

Le noyau Linux s'est outillé avec la commande **unshare** qui permet de créer tout objet de type *namespaces*.


UNSHARE(1) User Commands UNSHARE(1)

NAME
unshare - run program with some namespaces unshared from parent

SYNOPSIS
unshare [options] [program [arguments]]

DESCRIPTION
Unshares the indicated namespaces from the parent process and then executes the specified program. If program is not given, then ``$[SHELL]'' is run (default: /bin/sh).

Man unshare

Par exemple, voici comment créer un *namespace* de type utilisateur pour devenir root dans l'espace qui est créé :

```
$ unshare --map-root-user --user sh -c whoami
root
```

Cet outil permet de créer un ensemble de *namespaces* complexe pour en faire un environnement entièrement isolé, ayant son propre espace réseau, utilisateur, hiérarchie de processus, etc.