Warning: ce qui est écrit en italique est ma propre interprétation, ce n'est peut-être pas toujours exact.

Introduction à Kubernetes

Kubernetes (abrégé K8s) est un mot grec qui veut dire “gouvernail”. *On peut faire l'analogie avec le conteneur de marchandise qui a révolutionné la logistique en créant un standard: on peut dire que Docker a également établi un standard dans le packaging d'application, et Kubernetes quand à lui pourrait être comparé à un port automatisé (par exemple via la 5G en Chine https://www.youtube.com/watch?v=GhIdUWpTKvE).*

Kubernetes permet de faire du calcul distribué sur un cluster. C'est une sorte de linux réparti sur plusieurs serveurs, dans lequel les conteneurs jouent le rôle de processus: tout comme l'OS gère la répartition des ressources entre les différents processus pour éxecuter des application en parallèle, Kubernetes gère la répartition des ressources entre les conteneurs. En plus de cela, des sondes sont intégrées dans k8s pour surveiller en permanence l'état de santé des conteneurs: si ces derniers disfonctionnent, ils sont tout simplement détruits et recrées pour aboutir à l'état souhaité par l'administrateur, état qu'il a défini dans des fichiers de configuration. Le rôle de l'administrateur kubernetes est différent de celui d'administrateur de vm, qui doit lui gérer lui-même l'état de santé des vm, le stockage et l'infrastructure réseau.

1. Concepts

1.1 Applications monolithiques et microservices

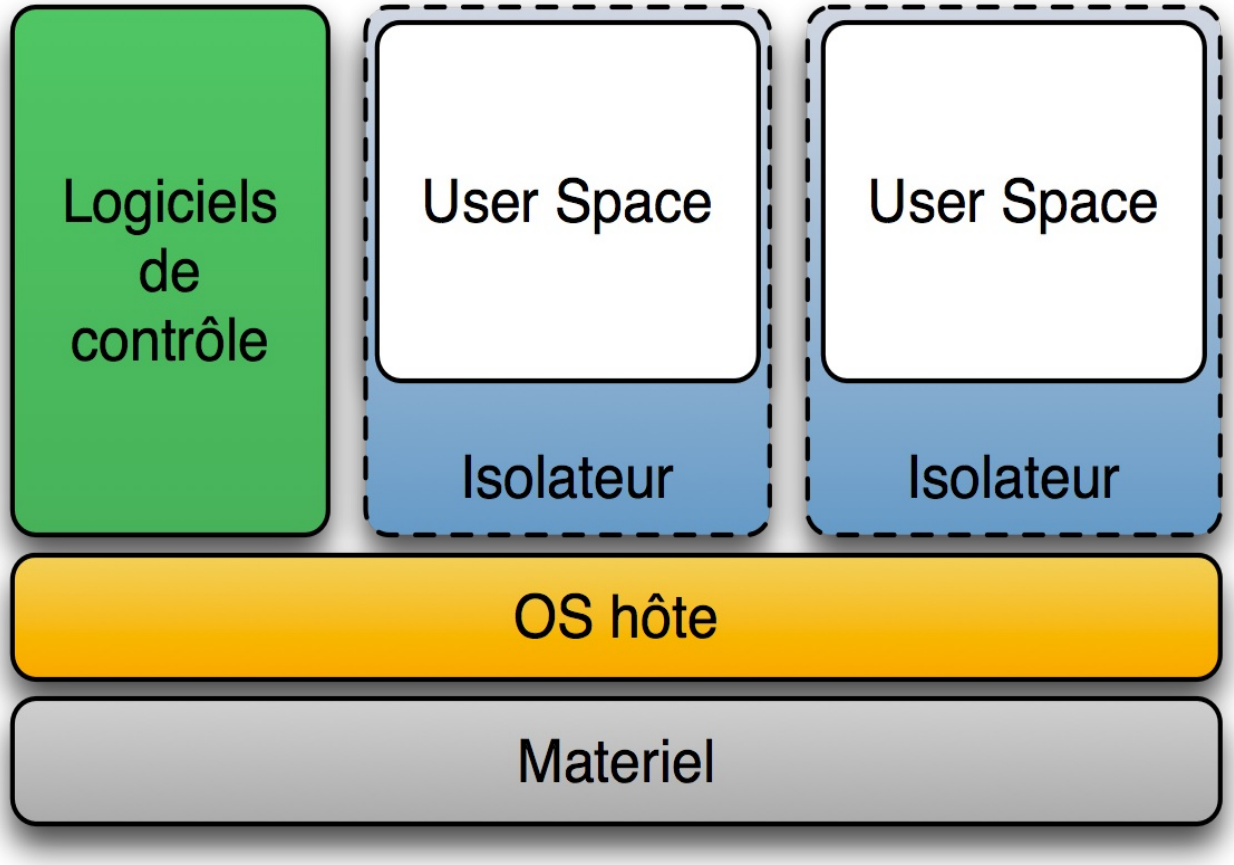
On peut faire le parallèle entre la programmation orientée objet et les microservices: les deux concepts sont basés sur la **modularité** et le minimum de couplage entre les parties d'une application. Un système basé sur un mainframe “legacy” (typiquement dans le secteur bancaire) développé dans des langages plus anciens(tels que Cobol) est un exemple d'application monolithique. Les applications modernes développées pour le cloud (typiquement les GAFAM, Netflix, Uber, toutes les grosses entreprises qui ont besoin de haute disponibilité et de résilience) sont des microservices. Pour simplifier, on pourrait prendre l'image d'un immense rocher versus des petits galets: avec le temps, les nouvelles fonctionnalités et améliorations ont ajouté de la complexité au code du monolithe, rendant le développement plus difficile et augmentant les durées des mises à jour, alors qu'il est beaucoup plus facile de collecter des cailloux dans un sceau et de les transporter là où c'est nécessaire. Cependant, l'architecture distribuée des microservices est plus complexe.

- modularité
- fléxibilité

1.2 Les conteneurs

Un point de départ usuel pour expliquer les conteneurs est de les comparer aux machines virtuelles. Il y a différentes techniques de virtualisation mais qui peuvent être complémentaires: typiquement faire tourner des conteneurs dans un parc de machines virtuelles.

- Isolateurs(conteneurs)



Avantages des conteneurs:

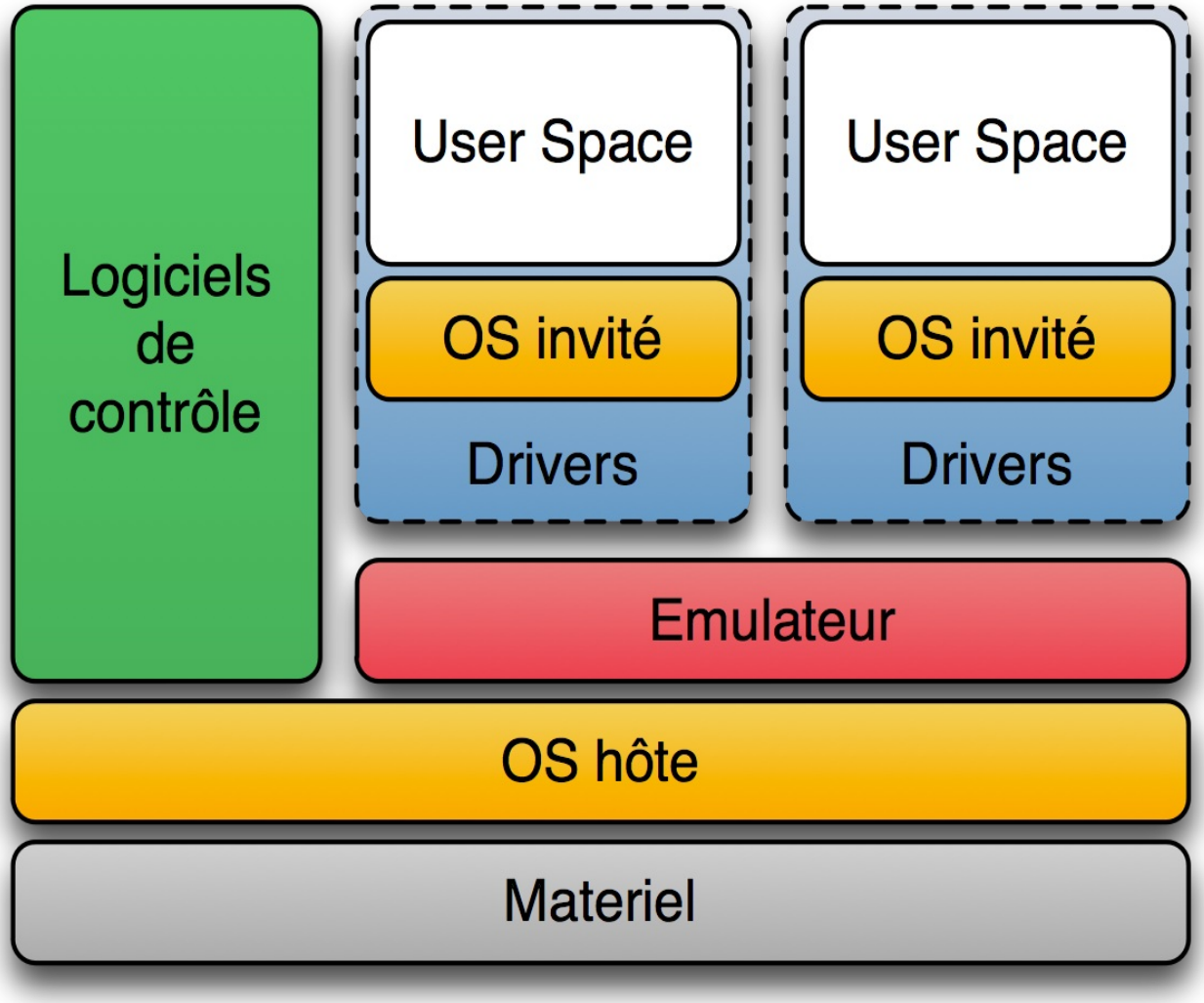
une très forte densité: on peut déployer beaucoup plus de conteneurs que de machines virtuelles. Par exemple j'ai pû déployer 200 réplications d'un conteneur docker basé sur une image nginx, en 10 secondes, pourtant sur un laptop peu puissant (core i3 dualcore et 6 Go de RAM).

performances: très peu d'overhead (temps passé par le système à ne rien faire d'autre que de se gérer lui-même). empreinte réduite: un conteneur est vraiment comme une *enveloppe* autour de l'application. Par exemple une application de test hello-world de taille 1,8Mio peut être packagée dans un conteneur de taille 1,9Mio! démo: `shell admin@ubuntu:~$`

Outre la densité, on améliore le temps de *cold start*, et, plus important encore on réduit la surface d'attaque au strict minimum (il y a moins de vulénrabilités potentielles dans un binaire que dans un Linux, même un Linux minimaliste).

Cependant, on ne peut pas vraiment parler de virtualisation de système d'exploitation. exemples: chroot, BSD Jail, OpenVZ, Docker (qui s'appuie sur LXC - Linux Container - et les namespaces, fonctionnalités du noyau Linux) La principale différence entre un conteneur et une machine virtuelle est que le conteneur **utilise le noyau de l'hôte**, ils sont donc très légers et très faciles et rapides à déployer/détruire/redéployer.

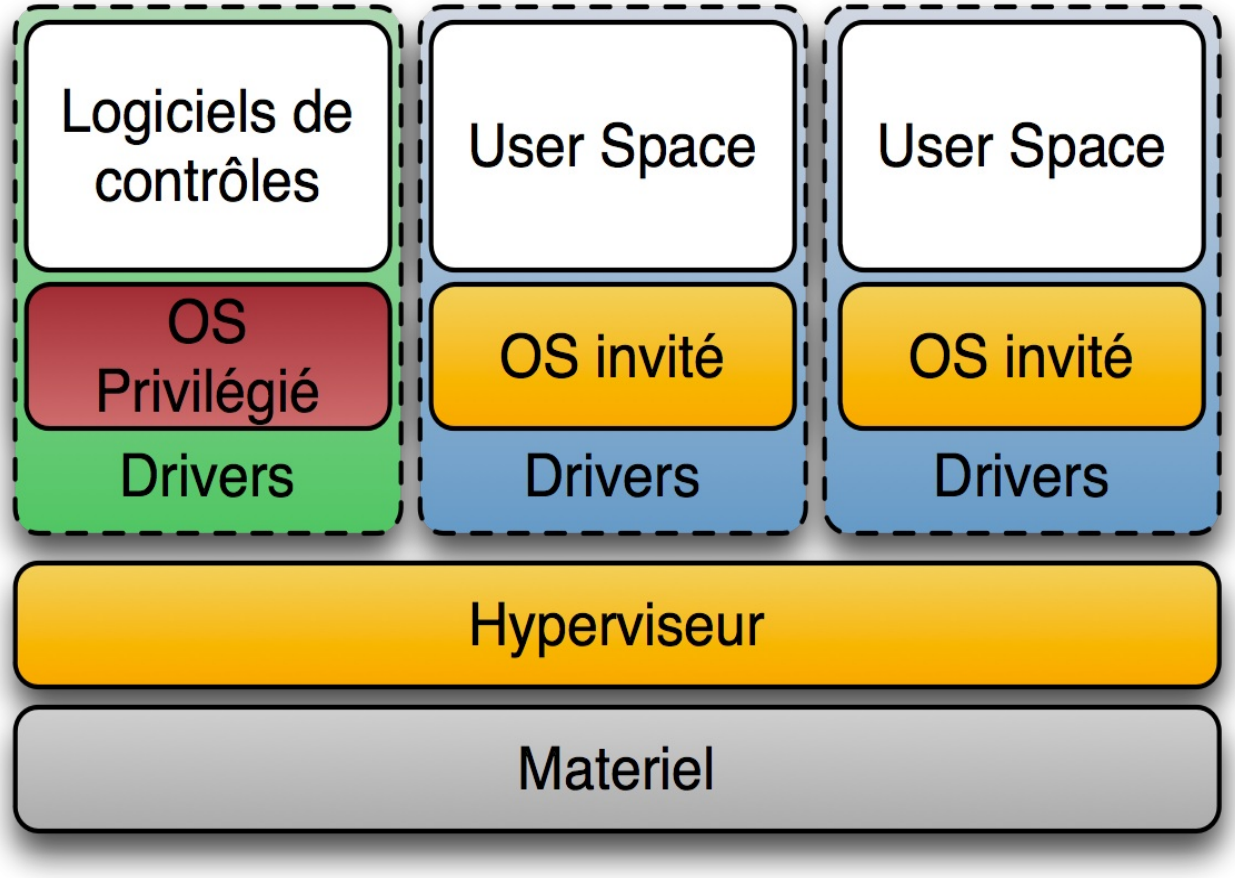
- Les hyperviseurs de type 2



Utile pour les développeurs et les testeurs, en formation ou pour découvrir de nouveaux OS. Cette solution isole bien les OS invités mais les performances sont bien moindres que dans le cas des hyperviseurs de type 1, car dans le type 2, l'hyperviseur est un programme lourd qui tourne sur l'OS de l'hôte.

Exemples: VMware Fusion/Workstation, VirtualBox, QEMU, ...

- Les hyperviseurs de type 1



C'est la solution adoptée en entreprise. L'hyperviseur de type 1 est un noyau système (Linux) très léger (juste avec une busybox) et optimisé pour gérer les accès des noyaux d'OS invités à l'architecture sous-jacente. Inconvénient: plus onéreux. Exemples: VMware ESXi/vSphere, Hyper-V, KVM, Xen, ...

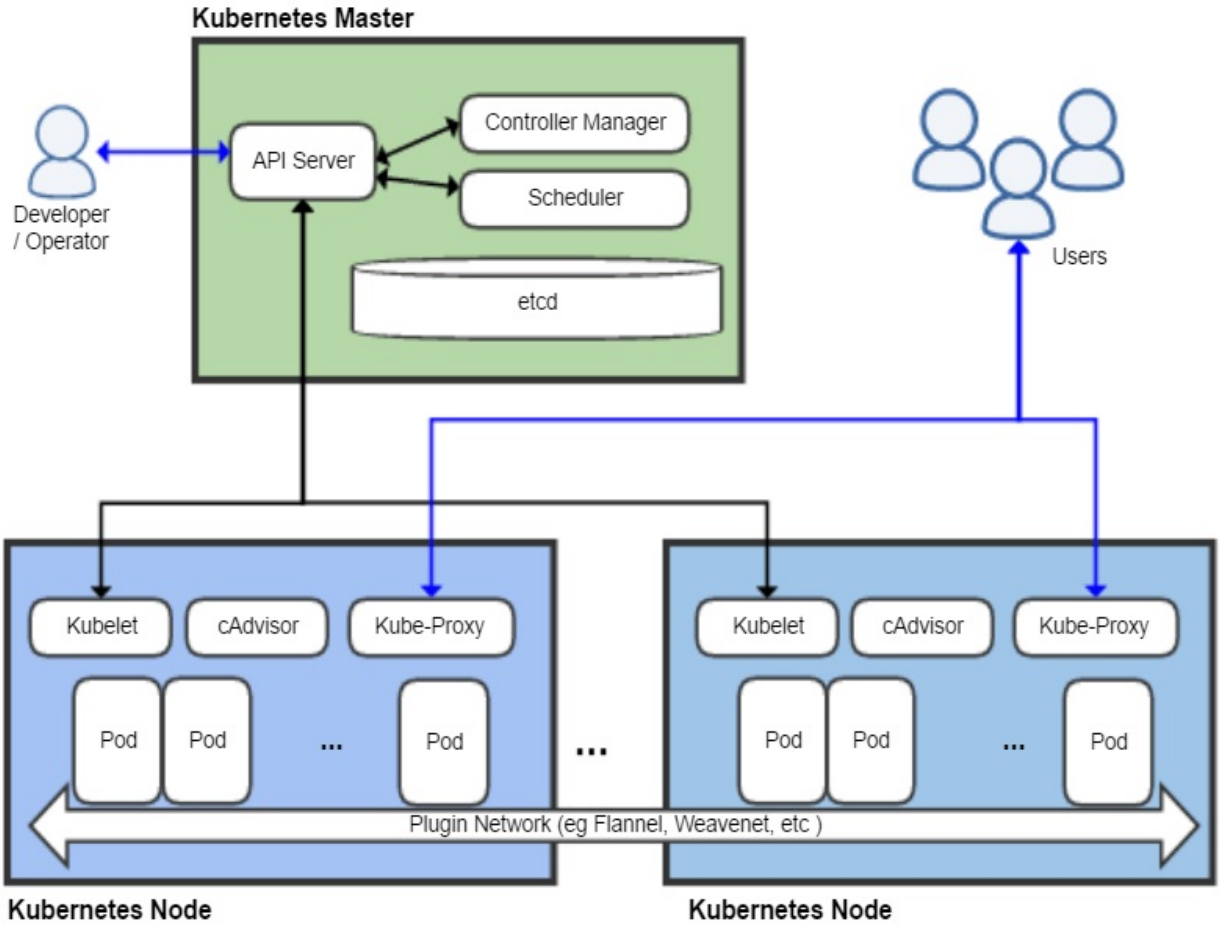
1.3 Orchestration des conteneurs

La gestion d'un petit nombre de conteneurs (pour les développeurs et les testeurs) n'a pas besoin d'un orchestrateur, mais dès qu'on passe dans des environnements de production il faut une Infrastructure As A Service comme K8s, ou Plateform As A Service (OpenShift, Rancher, ...).

- Docker swarm La solution d'orchestration de conteneurs de la société Docker est efficace et simple à mettre en place, mais elle a ses limites. Si un conteneur tombe en panne par exemple, c'est l'administrateur qui doit s'occuper manuellement de corriger le problème, et pendant ce temps, l'application est indisponible. Par contre Kubernetes garantit la disponibilité en s'occupant lui-même de corriger ce problème. Ce n'est qu'un exemple de ce que K8s fait de mieux que ses concurrents (Docker Swarm, Mesos,...), on pourrait citer aussi la gestion de l'infrastructure réseau.
- Kubernetes À l'origine, Kubernetes vient du projet Borg développé par Google depuis 2005, qu'ils ont ensuite rendu opensource en 2015 sous le nom de Kubernetes, et qui est depuis développpé par de nombrexu acteurs de l'opensource (RedHat, VMWare, ...). K8s est écrit en langage Go, langage développé par Ken Thompson le créateur d'Unix. *Go est un langage efficace pour la programmation parallèle, et donc bien adapté à un projet tel que K8s.*

2 Architecture

2.1 Architecture de Kubernetes



2.2 Les différents types d'installation du cluster

3 Utilisation

3.1 Création d'un cluster local avec minikube

3.2 Les blocs de base de K8s (nodes, pods, label, services, volumes, namespaces, ...)

3.3 Gestion du cluster Kubernetes