CLEAN:

(draft here: <https://docs.google.com/document/d/1ouTLbz0cI1rpX02s7N5ueQAOoKvJOD_UHdoR37p38_Q/edit> )

La conteneurisation est née d'un besoin répondant aux préoccupation des équipes de développement, mais de l'utilisation croissantes de ces derniers et de la multiplication des conteneurs par services (notion de micro-services) , d'autres problèmes ont vu le jour : la gestions des différents conteneurs, entre déploiement, évolutivité, mise en réseaux...

Effectivement, les conteneurs résolvent d'eux-mêmes de nombreux problèmes liés au processus de développement, tels que les dépendances et la compatibilité. De ce fait, la réflexion se porte maintenant sur la manière dont les conteneurs seront déployés. Comment faire évoluer les services déployés selon le besoin, comment s'assurer de la bonne utilisation des ressources fournies, et enfin, comment donner l'accès aux services hébergés.

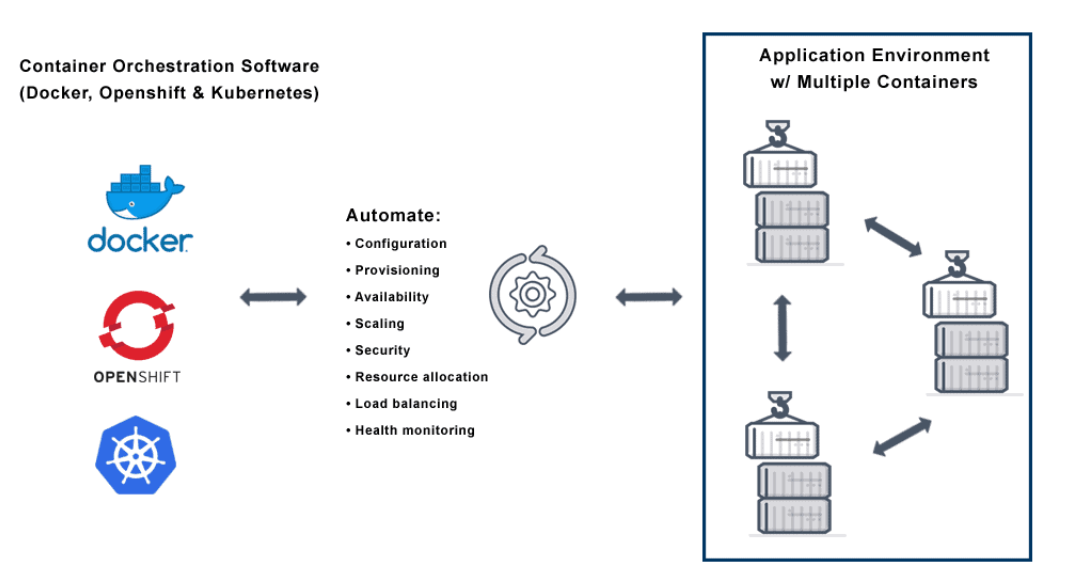
* Orchestration : (9)
  + Définition

“L'orchestration des conteneurs permet d'automatiser le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs. ”

On peut dire aussi que l’orchestration est l’automatisation et la coordination des tâches, d'une manière à optimiser les performances, gérer les concurrences, et minimiser les problèmes et le temps de production.

* + Principe de fonctionnement:

L’orchestration sert à gérer plusieures tâches, en créant un workflow, c'est-à-dire décrire comment les tâches seront exécutés et par qui. Sur un système basé sur l'orchestration, un administrateur décrit l'état désiré de système (performance, nombre des tâches exécutés, ressources utilisées…), à travers un ensemble des règles et spécifications. L’orchestrateur, en se basant sur ces paramètres, ajuste l'état des systèmes automatiquement, ses tâches incluent: la gestion des ressources physiques, l'ordonnancement des tâches, la mise à l'échelle… ce qui implique que l’orchestration nécessite une communication continue entre l'orchestrateur et les environnements d'exécution des tâches.



* + Avantages et inconvénients
  + Outils
    - Etude comparative

|  | Kubernetes | DockerSwarm | Apache Mesos |
| --- | --- | --- | --- |
| Développé par | Google | Docker, Inc. | Apache Software Foundation |
| Open Source or not | oui | oui | oui |
| supported containers | Docker, Rkt | Docker | Docker, Rkt |
| max number of nodes | 5000 noeud de travail (théorique) | 7 noeud maître | 10,000 noeud de travail (théorique) |
| découverte de services | implémentée | implémentée | implémentée |
| Mise à l'échelle automatique | n’est pas implémentée | implémentée | implémentée |

Ainsi, bien que les conteneurs n'aient besoin que d'un runtime pour s'exécuter, de nombreuses organisations préfèrent les lancer avec un orchestrateur de conteneurs. Parmi ceux-ci, Kubernetes est de loin le plus populaire.

Un orchestrateur de conteneurs gère les aspects opérationnels d'exécution d’applications dans des conteneurs - Comme la découverte de services, le redémarrage des conteneurs en cas d’arrêt, l'allocation des ressources systèmes et la gestion des groupes de conteneurs qui doivent être étroitement liés, pour n'en nommer que quelques-uns.

* Kubernetes

# **Définition:**

Kubernetes est une solution d'orchestration de conteneurs développée par Google et publiée en 2014.

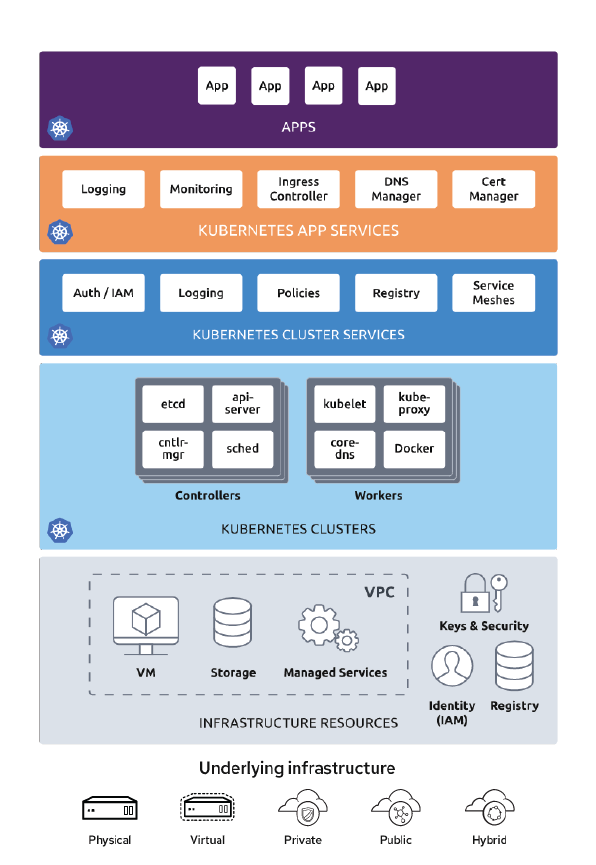
Selon le site officiel de Kubernetes:

“Kubernetes est une plate-forme open-source extensible et portable pour la gestion de charges de travail (workloads) et de services conteneurisés. Elle favorise à la fois l'écriture de configuration déclarative (declarative configuration) et l'automatisation. C'est un large écosystème en rapide expansion. Les services, le support et les outils Kubernetes sont largement disponibles.”

Autrement dit, Kubernetes est un logiciel utilisé pour automatiser les opérations de gestion des conteneurs, notamment la création, le déploiement et la mise à l'échelle. Cet outil permet de coordonner un grand nombre de conteneurs déployés sur plusieurs machines qui travaillent ensemble, d’une manière efficace.

# **kubernetes infrastructure**

* Une infrastructure kubernetes peut être divisée en 5 parties:
* Partie physique: il s’agit de l’infrastructure physique sur lequel le cluster est déployé, elle peut être purement physique, virtuelle (en site ou sur le cloud), publique, privée ou hybride.
* Le cluster Kubernetes: c’est la première couche d’abstraction sur l’infrastructure physique, c’est une vue logique, des machines sur lesquelles kubernetes est déployé.
* Les services liés au cluster Kubernetes: c’est des services (logicielles) qui permettent de monitorer, gérer et administrer le cluster Kubernetes.
* Les services liés au applications: c’est des services (logicielles) qui permettent de monitorer, gérer et administrer les applications qui s'exécutent sur le cluster.
* Les applications: les services et les applications qui s'exécutent sur le cluster.

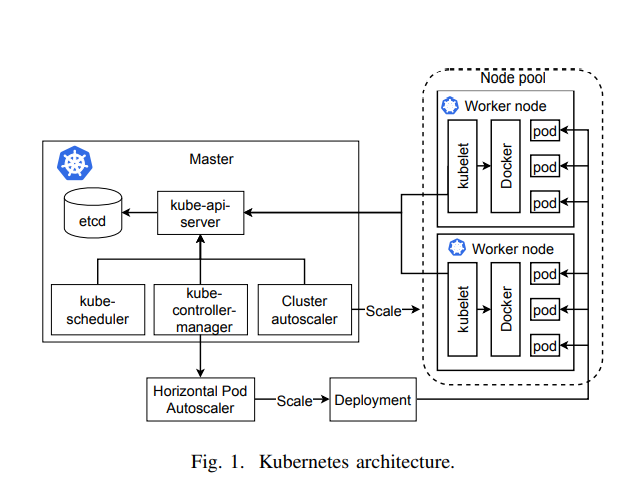


# **Principe de fonctionnement**

# **Architecture:**

Un cluster Kubernetes se compose de 2 parties importantes: une partie plan de contrôle, constituée d'un ou plusieurs nœuds maîtres, et d’une partie responsable de l'exécution des conteneurs constitué de un ou plusieurs nœuds de travail -workers-. La communication entre ces deux parties se fait à travers un API REST, géré par le kube-api-server.

* Un nœud étant responsable de l'exécution d’un conteneur, il comprend un container runtime, et un ensemble de PODs, géré par un Kubelet qui est toujours en communication avec le plan de contrôle.
* Le plan de contrôle contient plusieurs composantes responsables de maintenir ou d’atteindre un état défini par l’administrateur de cluster.



# **Modules:**

* + **Pod**: c’est la plus petite unité d'exécution qui peut être créée et déployée sur un cluster kubernetes. Un pod peut contenir un ou plusieurs conteneurs (cas des applications qui sont fortement couplées), qui seront toujours placés et ordonnancés ensemble, et qui partagent le même espace de stockage, la même adresse IP et la même plage de ports. Un pod est muni également d’un ensemble d’options qui contrôlent comment ses conteneurs doivent s'exécuter.
  + **Cluster** : L’ensemble des machines, physiques ou virtuelles, appelées nœuds, sur lesquelles Kubernetes est déployé.
  + **Nœuds**: Une unique machine du cluster, qui peut être soit physiques ou virtuelles. Un noeud peut se voir attribuer des rôles :
    - Noeud de travail (Worker node): c’est un nœud responsable de l'exécution de conteneurs, contrôlé par l'orchestrateur de cluster. Chaque nœud contient un ou plusieurs unités logiques appelées pod, et les services suivants qui permettent leur exécution: Container Runtime, Kubelet et Kube-proxy.
    - Nœud maître (Master Node): c’est le nœud responsable de la gestion de cluster, i.e l'orchestrateur de l'ensemble des nœuds de travail. Un cluster peut avoir plusieurs nœuds maîtres, pour garantir la haute disponibilité, dans ce cas il y aura, parmi les nœuds maîtres, un maître principal suivi par les autres.
  + **Objet Kubernetes**: se sont des entités persistantes sur le cluster qui servent à décrire son état. Il sont souvent stockés suivant le format *.yaml*
  + **ETCD**: une base de données clé-valeur consistante et hautement disponible utilisée comme mémoire de sauvegarde pour toutes les données du cluster.
  + **Container Runtime**: c’est le logiciel qui permet de gérer et d'exécuter des conteneurs, par exemple: Docker Engine.
  + **Kubelet**: c’est un “agent” qui s'exécute sur chaque nœud de travail pour gérer l'ensemble des pods et des conteneurs et assurer leur santé (bon état), en communiquant avec le nœud maître a travers l’API de kubernetes.
  + **Kube-proxy**: c’est un proxy qui s'exécute sur chaque nœud de travail. Sa responsabilité est d’assurer communication vers les Pods depuis l'intérieur ou à l'extérieur du cluster. Il intervient aussi dans l'équilibrage de charge.
  + **kube-apiserver**: kubernetes utilise l’API REST pour les communications internes et externes. Le serveur API constitue l’interface de communication au sein de cluster qui fournit les opérations REST est définie les objets qui peuvent être échangés entre les nœuds ou avec l'extérieur.

## **kube-scheduler**: c’est l'ordonnanceur de kubernetes qui s'exécute sur le nœud maître. Il est responsable d’affecter un pod au nœud qui convient. Le choix est pris selon les besoins de pod et même de conteneurs au sein de pod.

* + **kube-controller-manager**: c’est un logiciel toujours en cours d'exécution (daemon) qui contient le loop de contrôle qui régule l'état du système. Il surveille l'état de système à travers le kube-apiserver, et il essaye de maintenir ou d'atteindre l’état désiré de cluster.
  + **Deployment:** c’est un objet Kubernetes qui sert à décrire un état désiré de déploiement, c'est-à-dire le nombre des PODs dans le cluster. Ces objets sont utilisés par des contrôleurs qui travaillent à assurer cet état.
  + **Cluster-Autoscale:** c’est un composant de kubernetes qui contrôle la taille de Cluster, en créant ou supprimant des nouveaux pods ou nœuds selon le besoin.
  + **Horizontal-Pod-Autoscaler:** c’est un composant de kubernetes qui contrôle le nombre des PODs dans le cluster selon la charge de travail: une mise à l'échelle horizontale.
  + **Vertical-Pod-Autoscaler:** c’est un composant de kubernetes qui contrôle les ressources physique (CPU et mémoire) de cluster selon la charge de travail: mise à l'échelle verticale.

# **Fonctionnalités:**

Kubernetes offre plusieurs fonctionnalités, parmi les plus importantes on retrouve:

**Configuration déclarative**: kubernetes permet de décrire un état désiré du système, à travers des fichiers de configuration. Après cette déclaration, le logiciel va toujours essayer de maintenir et/ou d’atteindre cet état.

**Réparation et scalabilité automatique**: pour s’assurer que l'état actuel de cluster est conforme au état désiré déclaré, Kubernetes se charge de créer, redémarrer, remplacer, ou détruire des conteneurs d’une manière automatique selon les besoins.

**infrastructure immuable:** kubernetes permet de mettre en place une infrastructure qui reste toujours inchangée pour une application donnée, mais plutôt destructible. Par exemple, on peut remplacer un conteneur ayant certaines caractéristiques (dépendances, ressources…) par un autre complètement différent. Le but de l’immuabilité est de réduire la complexité et le risque aux pannes, et d'améliorer la sécurité et facilite la maintenance.

**Allocation des ressources**: ayant les informations sur les ressources en mémoire et en CPU nécessaire pour un conteneur sur le cluster, Kubernetes peut l'associer à une machine d’une manière automatique et efficace.

**Découverte de services**: Kubernetes permet d'annoncer certaines applications comme étant des services réseaux accessibles de l’extérieur, en attribuant des adresses IP aux pods et un nom DNS unique pour un ensemble de pods. Et lorsqu'une requête est reçue de l'extérieur, il assure un équilibrage du travail, et la redirection de requêtes.

**Gestion de cycle de vie:** Kubernetes facilite la gestion de vie des services et des applications, notamment le déploiement, la maintenance, la mise à l'échelle et l'extensibilité.

**Stockage:** Kubernetes permet d’ajouter plusieurs supports et solutions de sauvegarde de fichiers, comme le stockage local, le cloud…

**Sécurité:** Kubernetes offre la possibilité d'intégrer les processus DevSecOps avec la gestion des conteneurs tout au long de leur cycle de vie.

# **Avantages et inconvénients**

**Advantages**:

* **Gain de temps**: l'autonomisation des différentes opérations de gestion de conteneurs permet de gagner du temps, et d'améliorer les performances.
* **Scalabilité**: les clusters kubernetes sont évolutifs, c'est-à-dire que l’on peut ajouter des nodes, et des conteneurs à la demande d’une manière facile, rapide et efficace.
* **économisation**: Kubernetes permet également d'économiser de l'argent et de l'énergie, parce qu'il offre une infrastructure évolutive où le minimum de ressources physiques est nécessaire.
* **Abstraction d’infrastructure**: Kubernetes offre une abstraction de l'infrastructure physique, soit le nombre de machines physiques, leurs caractéristiques, systèmes d'exploitation…ne sont connus ni par l'équipe de développement ni par les utilisateurs.
* **Haut disponibilité**: Kubernetes facilite la mise en place des clusters hautement disponibles, grâce à ses fonctionnalités d'équilibrage de charge, et de réparation et scalabilité automatique, qui peuvent même être déployés sur des régions distantes dans le monde.
* **Flexibilité et Conformité**: Kubernetes permet à la fois de gérer une infrastructure qui est flexible (scalable, extensible et indépendante de placement géographique), et répondant aux besoins et exigences de QoS (sécurité…)ou autres(ressources disponibles…) grâce aux configurations déclaratives.

**Inconvénients**:

* **Complexité**: la migration vers Kubernetes (installation et de configuration) est souvent difficile surtout si la taille du système existant est importante. D’autres part, si le système est de taille réduite, l’utilisation d’une solution aussi complexe et sophistiquée que Kubernetes ne représente pas le choix le plus judicieux.
* **Approche centralisée**: comme toute solution d'orchestration, le cluster kubernetes dépend d’un point central de coordination. Ce qui devient un point de défaillance unique(SPOF), c'est-à-dire que s’il tombe en panne, le système dans son entièreté ne fonctionnera plus.
* **Trafic Réseau:** Kubernetes nécessite une communication permanente entre l'orchestrateur et les nœuds de cluster, ce qui peut encombrer le réseau et diminuer les performances.