Étude de l'Architecture de Kubernetes

Conception et Architecture des Systèmes d'Infonuagique (8INF876)

MAMOU Antoine - BIDET Arno

9 novembre 2025

Table des matières

1 Introduction

Kubernetes (ou **K8s**) est un système d'orchestration de conteneurs open-source, créé par Google et maintenant maintenu par la *Cloud Native Computing Foundation (CNCF)*. Son rôle principal est de gérer, déployer et faire évoluer des applications conteneurisées sur un **cluster de serveurs**, c'est-à-dire un ensemble de machines physiques ou virtuelles interconnectées qui fonctionnent comme un système unique.

Kubernetes automatise :

— le déploiement des conteneurs,

— la mise à l'échelle (*scaling*),

— le réseau entre conteneurs,

— la tolérance aux pannes,

— les mises à jour continues.

2 Architecture

Kubernetes est organisé autour de deux types principaux de composants :

2.1 Le Plan de contrôle (Control Plane)

Le plan de contrôle représente le **cerveau du cluster**. Il gère :

— l'état global du système,

— la planification des conteneurs,

— la surveillance des nœuds,

— et la communication entre les services.

Les principaux composants du plan de contrôle sont présentés dans le tableau suivant :

Composant	**Rôle**
API Server (kube-apiserver)	Point d'entrée du cluster. Toutes les commandes (`kubectl`) ou les composants communiquent avec Kubernetes via cette API REST.
etcd	Base de données clé-valeur hautement disponible contenant l'état du cluster (configurations, secrets, noms de Pods, etc.).
Controller Manager	Surveille l'état du cluster et exécute des boucles de contrôle pour s'assurer que l'état réel correspond à l'état désiré (par exemple, redémarrer un Pod supprimé).
Scheduler	Attribue les Pods aux nœuds disponibles selon les ressources, la proximité et les contraintes.
Cloud Controller Manager (optionnel)	Intègre Kubernetes avec les API des fournisseurs Cloud (*Cloud Provider API*) comme AWS, Azure ou GCP, afin de permettre la gestion automatique du réseau, du stockage, etc.

2.2 Les Nœuds (Nodes)

Les **nœuds** exécutent réellement les conteneurs. Chaque nœud contient les composants suivants :

Composant	**Rôle**
Kubelet	Agent local qui communique avec le plan de contrôle. Il veille à ce que les conteneurs soient bien en cours d'exécution.
Kube-proxy	Gère le réseau au niveau du nœud : équilibre de charge et routage du trafic entre Pods et Services.
Container Runtime	Moteur qui exécute les conteneurs (par exemple : Docker, containerd, CRI-O, etc.).

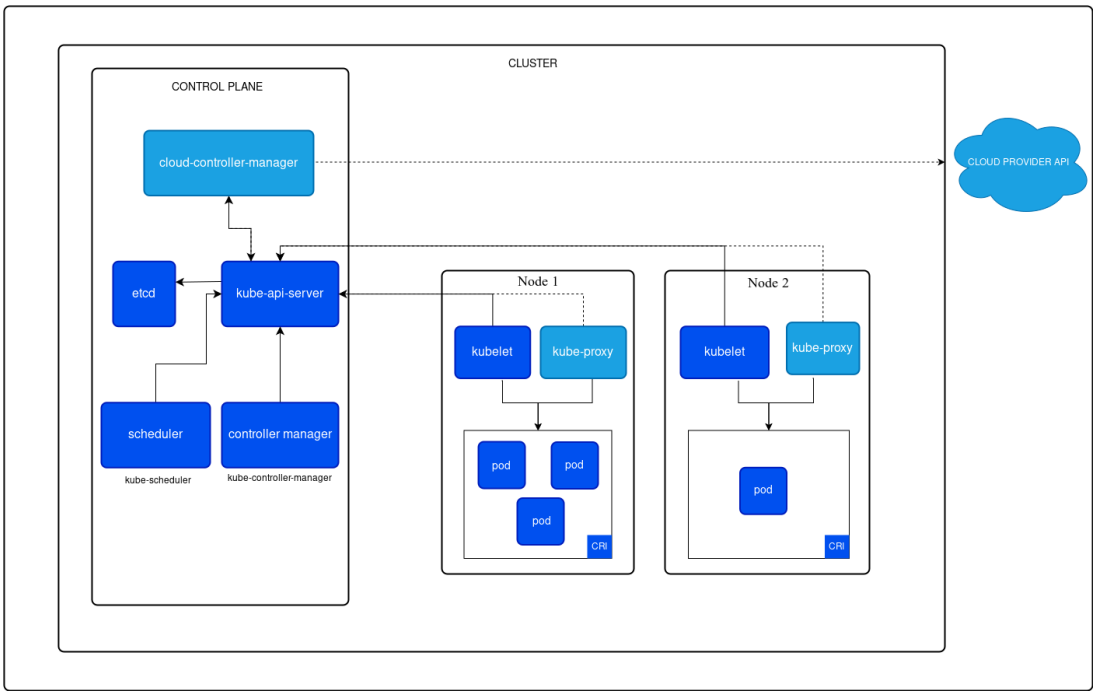

FIGURE 1 – Architecture générale d'un cluster Kubernetes

3 Composants logiques (Objets Kubernetes)

Kubernetes ne gère pas directement les conteneurs, mais des **objets** représentant leur état et leur configuration. Les principaux objets sont décrits ci-dessous :

Objet	Description
Pod	Plus petite unité déployable : un ou plusieurs conteneurs partageant le même réseau et stockage.
ReplicaSet	Garantit qu'un nombre défini de Pods est toujours actif.
Deployment	Gère les mises à jour et la montée en charge des ReplicaSets.
Service	Point d'accès stable (adresse IP fixe) pour exposer un ou plusieurs Pods.
Ingress	Gère l'accès externe HTTP/HTTPS vers les Services.
ConfigMap & Secret	Gèrent la configuration et les données sensibles séparément du code.
PersistentVolume (PV) & PersistentVolumeClaim (PVC)	Gèrent le stockage persistant pour les conteneurs.

4 Virtualisation et Isolation

La **virtualisation** est une technologie qui permet de faire fonctionner plusieurs environnements d'exécution isolés sur une même machine physique. Traditionnellement, cette virtualisation est réalisée au niveau matériel à l'aide d'un *hyperviseur*, qui crée et gère des **machines virtuelles (VM)**. Chaque VM embarque son propre système d'exploitation, ses bibliothèques et ses applications, ce qui offre une isolation forte mais au prix d'une consommation importante de ressources.

Kubernetes, quant à lui, ne repose pas sur une virtualisation matérielle classique. Il utilise une forme de **virtualisation légère au niveau du système d'exploitation**, rendue possible par les technologies de conteneurisation. Dans ce modèle, tous les conteneurs partagent le même noyau du système hôte, mais restent isolés les uns des autres grâce à des mécanismes spécifiques du noyau Linux.

Les principales technologies utilisées sont les suivantes :

— **Namespaces** : permettent d'isoler les espaces de noms des processus, du réseau, du système de fichiers et d'autres ressources. Chaque conteneur possède ainsi sa propre vue du système.

— **cgroups (control groups)** : assurent la limitation et la gestion des ressources matérielles (CPU, mémoire, disque) attribuées à chaque conteneur.

— **UnionFS / OverlayFS** : offrent une gestion efficace et modulaire des systèmes de fichiers des conteneurs, en superposant plusieurs couches pour réduire l'utilisation d'espace disque.

Grâce à cette approche, Kubernetes peut exécuter des centaines de conteneurs sur un seul hôte physique, tout en garantissant l'isolation, la sécurité et la performance. Cette forme de virtualisation légère constitue un compromis idéal entre efficacité et flexibilité, bien plus économe que l'utilisation de multiples machines virtuelles.

5 Avantages et Inconvénients de Kubernetes

Kubernetes présente de nombreux avantages dans la gestion et l'orchestration de conteneurs à grande échelle, mais il comporte également certaines limites liées à sa complexité et à ses besoins en ressources.

Avantages

— **Automatisation du déploiement et de la gestion** : Kubernetes permet de déployer, mettre à jour et redémarrer automatiquement les applications conteneurisées sans interruption de service.

— **Haute disponibilité et tolérance aux pannes** : En cas de défaillance d'un nœud, Kubernetes redéploie automatiquement les conteneurs sur d'autres nœuds du cluster.

— **Scalabilité horizontale** : Il est possible d'ajuster dynamiquement le nombre de Pods en fonction de la charge grâce à l'*Horizontal Pod Autoscaler*.

— **Portabilité** : Les applications peuvent être déployées sur n'importe quel environnement (cloud, local, hybride) sans dépendance à une infrastructure spécifique.

— **Écosystème riche et extensible** : Kubernetes s'intègre avec de nombreux outils (Prometheus, Helm, Istio, etc.) pour la supervision, la sécurité et la gestion des configurations.

— **Gestion des mises à jour continues (Rolling updates)** : Les déploiements se font progressivement, sans interruption du service, avec la possibilité de revenir en arrière (*rollback*) en cas d'échec.

Inconvénients

— **Complexité de mise en œuvre** : L'installation, la configuration et la maintenance d'un cluster Kubernetes nécessitent une expertise technique avancée.

— **Courbe d'apprentissage élevée** : Le grand nombre de concepts (Pods, Services, Deployments, etc.) peut rendre la prise en main difficile pour les débutants.

— **Consommation de ressources** : Les composants du plan de contrôle (API Server, etcd, Controller Manager) exigent une infrastructure importante.

— **Debugging plus complexe** : En raison de la nature distribuée du système, le diagnostic des erreurs peut être long et complexe.

— **Surcoût opérationnel** : La maintenance d'un cluster Kubernetes complet (monitoring, sécurité, mise à jour) implique souvent des coûts humains et matériels supplémentaires.

6 Comparaison entre Kubernetes, Docker et les Machines Virtuelles (VM)

Dans cette section, nous comparons Kubernetes avec les conteneurs Docker “simples” et les Machines Virtuelles (VM) selon trois critères essentiels : la maintenance, la scalabilité et le type d’interaction.

Comparaison selon différents critères

Critère	**Kubernetes**	**Conteneurs Docker seuls**	**Machines Virtuelles(VM)**
Maintenance	Centralisée via le plan de contrôle ; automatisation des déploiements et des mises à jour ; supervision intégrée.	Nécessite une gestion manuelle ou via des scripts Docker Compose ; pas d’orchestration native.	Maintenance lourde : chaque VM nécessite un système d’exploitation complet et des mises à jour individuelles.
Scalabilité	Excellente : montée ou descente automatique en charge grâce à l’autoscaling et la distribution multi-nœuds.	Limitée : nécessite la création manuelle de nouveaux conteneurs ou la gestion par des outils externes.	Faible à moyenne : le déploiement de nouvelles VMs est long et coûteux en ressources.
Type d’interaction	*Event-driven* : réagit aux événements (état des Pods, charge CPU, pannes) pour adapter le cluster automatiquement.	Basé sur des commandes manuelles ou des scripts ; peu d’automatisation native.	Interaction principalement manuelle via un hyperviseur ou des outils de gestion ; pas de réaction automatique aux événements.

Analyse

Kubernetes se distingue par son haut niveau d’automatisation et de résilience, ce qui en fait un outil de référence pour les environnements cloud modernes. Contrairement à Docker utilisé seul, Kubernetes gère automatiquement la communication, la mise à l’échelle et le redéploiement des conteneurs. Face aux machines virtuelles, il offre une utilisation bien plus efficace des ressources, une réactivité accrue et une maintenance simplifiée grâce à son architecture orientée événements.

En revanche, cette puissance s’accompagne d’une complexité de mise en œuvre et d’administration plus élevée, justifiant son usage principalement dans les environnements de production à grande échelle.