

Architecture et composants Kubernetes

Comprendre le fonctionnement interne de Kubernetes

Vue d'ensemble de l'architecture



Architecture client-serveur

Kubernetes utilise une architecture distribuée avec une séparation claire entre le plan de contrôle et les nœuds de travail.



Plan de contrôle (Control Plane)

Anciennement appelé "Master", il gère l'état global du cluster et prend les décisions centralisées (planification, mise à l'échelle, etc.).



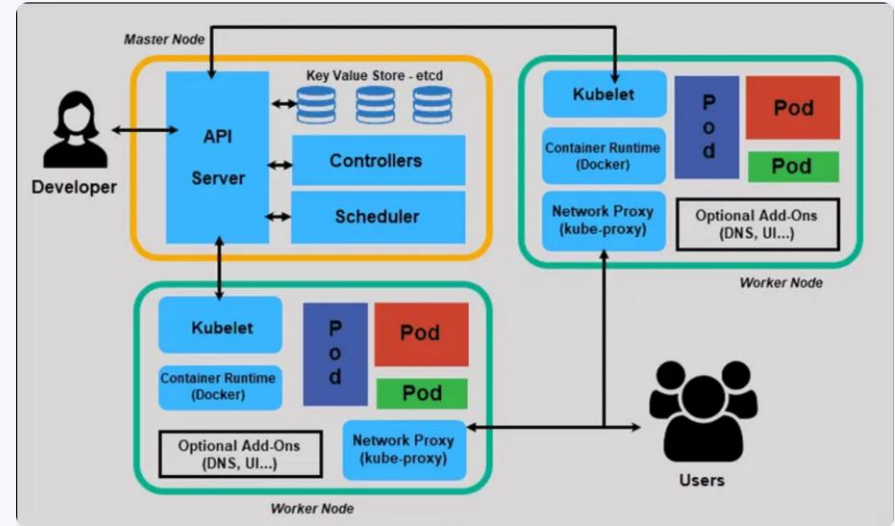
Nœuds de travail (Worker Nodes)

Machines qui exécutent les applications conteneurisées et les services nécessaires pour gérer les conteneurs.



Communication

Tous les composants communiquent via l'API Server, qui sert de passerelle centrale pour toutes les opérations.



Master Node (Plan de contrôle)

Le plan de contrôle est le cerveau de Kubernetes qui maintient l'état souhaité du cluster et prend toutes les décisions globales. Il est généralement déployé de manière hautement disponible sur plusieurs machines.



API Server (kube-apiserver)

Point d'entrée central pour toutes les requêtes REST



etcd

Base de données clé-valeur stockant l'état du cluster



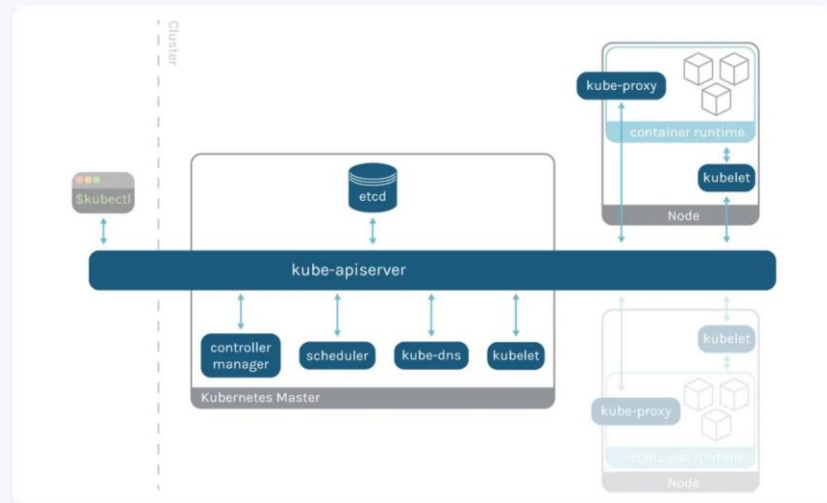
Controller Manager

Exécute les processus de contrôle régulant l'état du cluster



Scheduler

Attribue les pods aux nœuds selon les ressources disponibles



API Server



Rôle central

- ✓ Point d'entrée unique pour toutes les requêtes REST vers le cluster
- ✓ Valide et configure les données pour les objets API
- ✓ Seul composant qui communique directement avec etcd



Sécurité

- 🔒 Authentification : certificats TLS, tokens, comptes de service
- 👤 Autorisation : RBAC, ABAC, Node, Webhook



Flux de communication

Exemple : Création d'un pod

- 1 L'utilisateur envoie une requête de création de pod via kubectl
- 2 L'API Server authentifie et autorise la requête
- 3 Les contrôleurs d'admission valident la requête
- 4 L'API Server enregistre le pod dans etcd
- 5 Le Scheduler détecte le nouveau pod et sélectionne un nœud
- 6 Le kubelet du nœud démarre le pod

Distribution typique des requêtes API (%)






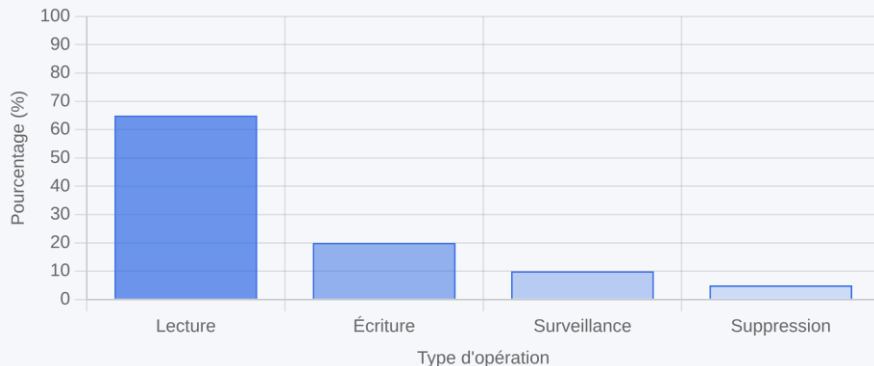
etcd

Qu'est-ce que etcd ?

etcd est une base de données clé-valeur distribuée, cohérente et hautement disponible qui sert de stockage principal pour toutes les données du cluster Kubernetes.

Caractéristiques principales

-  Utilise l'algorithme de consensus Raft pour garantir la cohérence des données
-  Haute disponibilité avec réplication sur plusieurs nœuds
-  Performances optimisées pour les lectures fréquentes



Données stockées dans etcd



Objets Kubernetes

Pods, Services, Deployments, ConfigMaps, Secrets et autres objets API



État du cluster

Informations sur les nœuds, leur disponibilité et leurs ressources



Configuration du cluster

Paramètres de configuration et politiques du cluster



Informations réseau

Configuration réseau, routes et règles de routage



Points importants



Sauvegarde régulière essentielle pour la récupération du cluster



Seul l'API Server communique directement avec etcd

Controller Manager

Le Controller Manager est un composant du plan de contrôle qui exécute les processus de contrôle. Ces contrôleurs observent l'état du cluster via l'API Server et travaillent pour faire correspondre l'état actuel à l'état souhaité.



Principaux contrôleurs



Node Controller

Surveille l'état des nœuds et réagit lorsqu'un nœud devient inaccessible



Replication Controller

Maintient le nombre correct de pods pour chaque objet ReplicaSet



Endpoints Controller

Remplit les objets Endpoints (associe Services et Pods)



Service Account & Token Controllers

Crée des comptes par défaut et des jetons d'accès API



Job Controller

Surveille les objets Job et crée des pods pour exécuter les tâches

Cycle de réconciliation

- 1 Observer l'état actuel du cluster via l'API Server
- 2 Comparer l'état actuel avec l'état souhaité
- 3 Effectuer des actions pour rapprocher l'état actuel de l'état souhaité
- 4 Mettre à jour l'état via l'API Server
- 5 Répéter le cycle en continu

Scheduler



Rôle du Scheduler

Le Scheduler est responsable de l'attribution des pods nouvellement créés aux nœuds du cluster. Il surveille l'API Server pour détecter les pods sans nœud assigné et sélectionne le nœud optimal pour chaque pod.



Processus de sélection des nœuds

- 1 **Filtrage** : Élimine les nœuds qui ne répondent pas aux exigences du pod (ressources, contraintes, etc.)
- 2 **Notation** : Attribue un score à chaque nœud éligible selon plusieurs critères
- 3 **Sélection** : Choisit le nœud avec le score le plus élevé
- 4 **Liaison** : Informe l'API Server de la décision d'affectation



Contraintes et affinités



nodeSelector : Contrainte simple basée sur les labels des nœuds



Affinité de nœud : Préférence pour certains nœuds avec expressions complexes



Anti-affinité : Évite de placer des pods sur les mêmes nœuds



Taints et tolérances : Permet aux nœuds de repousser certains pods

Worker Node

Les nœuds de travail (Worker Nodes) sont les machines qui exécutent les applications conteneurisées. Ils sont gérés par le plan de contrôle et contiennent tous les services nécessaires pour exécuter des pods.

Composants principaux



kubelet

Agent principal qui s'assure que les conteneurs fonctionnent dans un pod



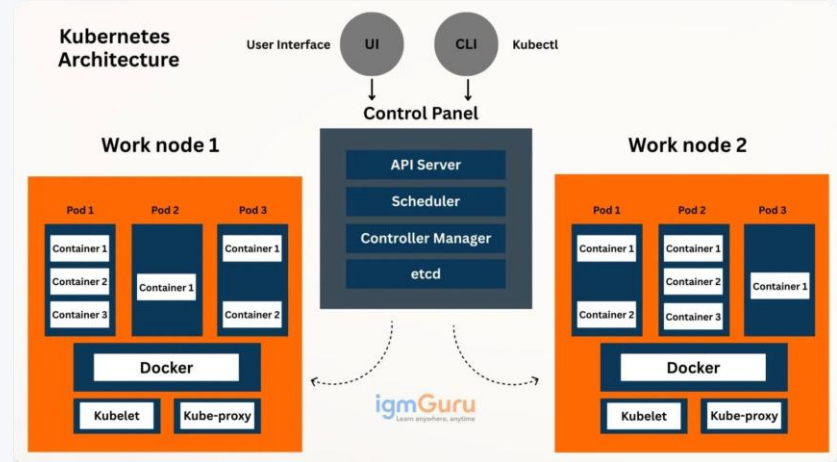
kube-proxy

Maintient les règles réseau et permet la communication entre les services



Container Runtime

Logiciel responsable de l'exécution des conteneurs (Docker, containerd, CRI-O)



Responsabilités

- ✓ Exécution des charges de travail (pods et conteneurs)
- ✓ Communication avec le plan de contrôle via l'API Server
- ✓ Gestion du réseau pour les pods et les services

Container Runtime et kubelet



Container Runtime

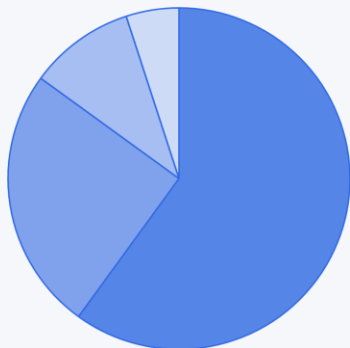
Le Container Runtime est le logiciel responsable de l'exécution des conteneurs. Il interagit avec le système d'exploitation pour isoler les processus, gérer les systèmes de fichiers et configurer les réseaux des conteneurs.

✓ **Rôles principaux** : Télécharger les images, créer et exécuter les conteneurs, allouer les ressources

🔌 **CRI** (Container Runtime Interface) : Interface standardisée permettant à Kubernetes de communiquer avec différents runtimes

Utilisation des Container Runtimes (estimation)

■ containerd ■ CRI-O ■ Docker (via cri-dockerd) ■ Autres



kubelet

Le kubelet est l'agent principal qui s'exécute sur chaque nœud. Il s'assure que les conteneurs décrits dans les PodSpecs sont en cours d'exécution et en bonne santé.

📋 **Gestion des pods** : Création, surveillance et suppression des pods sur le nœud

💓 **Vérifications de santé** : Exécution des sondes de disponibilité et de vivacité

🔄 **Reporting** : Communication avec l'API Server pour signaler l'état des pods

Interaction kubelet - Container Runtime

- 1 Le kubelet reçoit une PodSpec de l'API Server
- 2 Le kubelet traduit la PodSpec en instructions pour le Container Runtime via CRI
- 3 Le Container Runtime télécharge les images nécessaires
- 4 Le Container Runtime crée et démarre les conteneurs
- 5 Le kubelet surveille l'état des conteneurs et rapporte à l'API Server

kube-proxy et communication



kube-proxy

kube-proxy est un composant réseau qui s'exécute sur chaque nœud du cluster. Il maintient les règles réseau et permet la communication avec les pods depuis l'intérieur ou l'extérieur du cluster.



Modes de fonctionnement



Mode userspace

Premier mode historique, moins performant



Mode iptables

Mode par défaut, utilise les règles iptables du noyau Linux



Mode IPVS

Mode plus récent et performant pour les grands clusters



Types de communication



Container-to-Container

Les conteneurs dans le même pod partagent le même espace réseau



Pod-to-Pod

Chaque pod a sa propre adresse IP et peut communiquer directement



Pod-to-Service

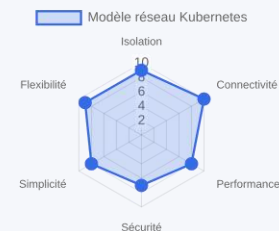
Les services fournissent une adresse IP stable pour accéder aux pods



External-to-Service

Accès externe via NodePort, LoadBalancer ou Ingress

Caractéristiques du modèle réseau



Cloud Controller Manager



Qu'est-ce que le Cloud Controller Manager ?

Le Cloud Controller Manager (CCM) est un composant du plan de contrôle qui intègre Kubernetes avec les API des fournisseurs cloud. Il permet de séparer les composants qui interagissent avec la plateforme cloud des composants qui interagissent uniquement avec le cluster.



Contrôleurs spécifiques au cloud



Node Controller

Vérifie auprès du fournisseur cloud si un nœud a été supprimé dans le cloud lorsqu'il ne répond plus



Route Controller

Configure les routes réseau dans l'infrastructure cloud pour permettre la communication entre les pods sur différents nœuds



Service Controller

Crée, met à jour et supprime les équilibres de charge cloud lorsque des services de type LoadBalancer sont créés ou modifiés



Avantages de l'architecture CCM

- ✓ **Modularité** : Sépare les préoccupations entre la gestion du cluster et l'intégration cloud
- ✓ **Flexibilité** : Permet d'utiliser différents fournisseurs cloud ou de passer d'un fournisseur à un autre
- ✓ **Maintenance** : Facilite la mise à jour de Kubernetes sans affecter les intégrations cloud
- ✓ **Innovation** : Permet aux fournisseurs cloud de développer leurs fonctionnalités à leur propre rythme



Fournisseurs cloud supportés

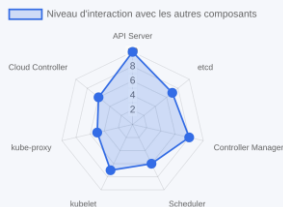


Conclusion

✓ Points clés à retenir

- ✓ **Architecture décentralisée** : Séparation claire entre le plan de contrôle et les nœuds de travail pour une meilleure scalabilité
- ✓ **API Server central** : Point d'entrée unique pour toutes les opérations, garantissant cohérence et sécurité
- ✓ **État déclaratif** : Les contrôleurs travaillent en continu pour faire correspondre l'état actuel à l'état souhaité
- ✓ **Modèle réseau robuste** : Communication fluide entre tous les composants du cluster
- ✓ **Extensibilité** : Architecture modulaire permettant l'intégration avec différentes infrastructures cloud

Niveau d'interaction entre les composants



☰ Ressources pour approfondir



Documentation officielle Kubernetes

kubernetes.io/docs/concepts/architecture/



Kubernetes: Up and Running

Livre de référence par Brendan Burns, Joe Beda et Kelsey Hightower



Certifications CNCF

CKA (Certified Kubernetes Administrator), CKAD (Certified Kubernetes Application Developer)



Kubernetes The Hard Way

Guide pratique pour comprendre chaque composant en profondeur



Communauté Kubernetes

Slack, forums, meetups et KubeCon pour échanger avec d'autres utilisateurs

Merci pour votre attention !