

## CEPH

### C'est quoi ?

Ceph est une solution de stockage distribuée et open-source développée pour mettre en place un stockage hautement évolutif, redondant et performant pour des systèmes de fichiers tels que les blocs de données et les objets. L'installation de CEPH peut se faire sur de nombreux appareils qui va d'un rack de serveurs avec 86 disques à un simple Raspberry Pi. Il n'y pas de limite théorique de stockage pour un cluster. Par exemple, il y a 7 ans, le CERN a testé 11 000 disques sur un cluster CEPH.

### Historique

#### 2004

À la base, Ceph a été développé par Sage Weil dans le cadre de sa thèse de doctorat à l'Université de Californie. L'objectif principal était de créer un système de stockage distribué permettant d'avoir des performances élevées, une haute disponibilité et une forte évolutivité.

#### 2010

Le client CEPH pour les systèmes Block et Fichier est désormais disponible dans le noyau Linux. CEPH devient ainsi bien plus accessible. Il suffit juste d'installer un simple paquet pour pouvoir l'utiliser.

#### 2012

Sage Weil fonde Inktank Storage pour offrir un support commercial et de la formation sur CEPH. Le projet se professionnalise et le rend viable pour les entreprises qui cherchent un support commercial.

#### 2014

Red Hat, une entreprise majeure dans l'open-source, rachète Inktank Storage pour 175 millions de dollars.

#### 2018

Il y a la création de la fondation CEPH. Elle dépend de la fondation Linux. Parmi les différents membres qui la composent, on y trouve des fournisseurs de stockage : Samsung, Intel, Western Digital, des fournisseurs de cloud : Digital Ocean, OVH Cloud et des entreprises de support/conseil : Red Hat ...

## Les composants

### *CEPH Monitor*

Le CEPH Monitor surveille l'état général du serveur en gardant en mémoire différents maps (monitor, manager, OSD, MDS, CRUSH). Ces maps permettent aux daemons CEPH de se coordonner entre eux. Son nom peut porter à confusion : Il ne fait pas de monitoring. Cette Tâche revient aux OSD. Le Monitor gère toute authentification entre le client et les daemons via un système de token. En générale, il est conseillé d'avoir 3-5 monitors par cluster.

### *CEPH Manager*

Le CEPH Manager permet de surveiller/mesurer des statistiques (utilisation, performance, charge système) en temps réel des daemons. Les informations sont exposées/gérées par des modules python hébergées dans les managers. Il y a prise en charge des APIs REST. Un CEPH

Dashboard y est mis à disposition. Des plug-ins sont disponibles. Par exemple, un plug-in permet de prédire quand les disques ne vont plus fonctionner en se basant sur les informations recueillies. Pour garantir une haute disponibilité, il est utile d'avoir au moins deux managers.

### *CEPH Object Storage Daemon*

Il gère directement l'entièreté du stockage des disques. Il fait du monitoring : Il surveille si les autres fonctionnent bien et si dans le cas où un OSD dysfonctionne, les autres autour vont l'expulser du cluster. Pour avoir une certaine résilience des données, il effectue de la réplication ou de l'effacement codé au choix de l'utilisateur. Normalement, on attribue un OSD par disque.

La réplication est similaire à du RAID 1. L'entièreté des données d'un disque est répliquée dans les autres. Cela demande un coût en stockage élevé. La vitesse est limitée au disque le plus lent. Cela garde une bonne performance et une certaine simplicité. Attention, à avoir au moins 3 disques sinon avec seulement 2 disques, quand CEPH va détecter une incohérence dans les données, il ne pourra pas déterminer quel disque est juste et il va juste choisir arbitrairement un.

L'effacement codé ressemble à du RAID 6. Chaque donnée est divisée en  $(n \text{ disques} - 1)$  parties en plus du checksum qui sont réparties dans les disques. Grâce au checksum et aux différentes parties de la données réparties à travers les disques, si un disque tombe en panne, alors il est simple de recréer la partie perdue à partir du reste. Malheureusement, il y a un coût en performance parce qu'il est nécessaire de faire une requête pour chaque disque pour reconstituer la donnée souhaitée. Cette approche est plus complexe que la réplication.

### *RADOS*

Le RADOS est composé d'au moins un OSD et un Monitor. C'est le minimum pour avoir un cluster fonctionnel. Une fois qu'on a le RADOS, on peut utiliser les bibliothèques Librados qui permettent d'accéder aux données CEPH avec différents langages (Python, Java, Rust, C ...). On peut désormais faire du Block qui donnent accès aux fonctionnalités suivantes :

- Snapshots (Copie d'un instant T d'un disque),
- Copy on Write (Exemple : Lancer 100 VM Ubuntu avec une seule image stockée)
- Mirroring (Synchronisation de toutes les écritures d'un cluster CEPH avec un autre)

### *CEPH Metadata Serveur*

Si on souhaite avoir un système de fichier (CephFS), il est nécessaire d'avoir un serveur de métadonnées. Avec beaucoup de RAM, le serveur va stocker un maximum de métadonnées. Côté client, des données seront mis en cache pour éviter de refaire des requêtes coûteuses si l'état du cluster n'a pas changé.

### *RADOS Gateway*

Si on veut de l'object storage (AWS S3), alors il faut un RADOS Gateway. Il permet d'agréger les clusters entre eux pour faire du multi zone. Par exemple, on éparpille des clusters à travers le monde pour qu'ils soient au plus proche des clients. Les clusters sont tous synchronisés. On souhaite juste distribuer du contenu aux plus proches des clients. Le daemon met à disposition un Gateway RESTful entre les applications et les clusters.

## Fonctionnement

### *Pool – Groupe de placement*

On ne peut pas stocker directement un objet dans le cluster. D'abord, il est nécessaire de créer un pool. Ce dernier représente un cas d'utilisation prévu. Puis, au sein de ce pool, on y définit des groupes de placement. En générale, il y en a une centaine par OSD. L'objet que l'on souhaite stocker est placé dans ce groupe de placement. Ces groupes permettent d'alléger la gestion des objets à l'OSD. Selon l'algorithme CRUSH, en fonction de la donnée voulue et de l'architecture du cluster, il va déterminer le bon OSD à communiquer directement pour y récupérer la donnée souhaitée.

### *CSI*

Le Container Storage Interface est une norme qui permet aux systèmes de stockage comme Ceph, Amazon EBS de s'intégrer de manière uniforme avec les orchestrateurs de conteneurs Kubernetes, Docker Swarm, ...

Parmi les fonctionnalités proposées

- Le provisionnement dynamique permet de créer automatiquement du stockage quand les applications en ont besoin.
- Monter et démonter des volumes sur les nœuds où les pods sont déployés
- Prendre des Snapshots des volumes
- Redimensionner en augmentant la taille des volumes existants
- Supprimer les volumes non utilisées pour libérer des ressources

## Hardware

### *Processeur CPU*

- **CEPH Métadonnée Serveur**, Le serveur est particulièrement gourmand en puissance du processeur. Ce dernier doit être à haute fréquence, mais ne nécessite pas un grand nombre de cœur pour satisfaire le serveur.
- **Monitor et Manager**, Ils ne nécessitent pas spécialement d'un CPU puissant. Un modeste leur suffit.
- **OSD**, Les nœuds OSD nécessitent une puissance de traitement suffisante pour exécuter le service RADOS, calculer le placement des données avec CRUSH et répliquer les données. Avec les disques NVMe, Ceph peut utiliser efficacement plusieurs cœurs par OSD.

### *Mémoire RAM*

- **Monitor et Manager**, 64 Go peut suffire pour un cluster moyen, sinon 128 Go pour des clusters plus grand
- **OSD**, La configuration par défaut de `osd_memory_target` est de 4 Go. Il est utile de prévoir une marge en plus pour l'OS et les tâches administratives, en allouant 8 Go par OSD utilisant BlueStore.
- **CEPH Métadonnée Serveur**, Il est recommandé de garder 1 Go. La consommation de mémoire dépend de la taille du cache configuré.

### *Stockage*

- **Disques Durs HDD**, Il est recommandé d'avoir un disque dédié pour l'OS et un pour chaque OSD.

- **Disques SSD**, Leur utilisation pour le Monitor, Manager et les pools va améliorer considérablement les performances. Faire attention de bien aligner les partitions des disques pour éviter des ralentissements au niveau des transferts de données.
- Concernant le système de fichier CephFS, la séparation du stockage des métadonnées à celle du contenu des fichiers améliore les performances.

## Configuration

### *Backend - BlueStore*

BlueStore est le backend de stockage par défaut de Ceph OSD, développé pour remplacer l'ancien backend Filestore. Il permet d'améliorer les performances, la fiabilité et l'efficacité du stockage des objets dans CEPH. Il possède plusieurs caractéristiques notamment le mécanisme de 'copy-on-write clone' et la vérification des données et métadonnées avec des checksums.

### *Sources, sections de configuration*

Les services/démons CEPH récupèrent leurs configurations à partir de différentes sources : Valeurs par défaut, Base de données de configuration, Fichiers de configuration locaux, Variables d'environnement, Arguments de ligne de commande et surcharges en temps réel définies par l'administrateur.

Les options configurables sont regroupées en sections pour spécifier leur portée : global (tous), mon (monitor), mgr (manager), osd (OSD), mds (serveur métadonnées) et clients (CephFS, Block, RGW).

La commande 'ceph config' permet de configurer le cluster.

### *Réseau*

Par défaut, CEPH utilise un seul réseau public pour toutes les communications. Cependant, pour avoir de meilleures performances dans les grands clusters, il est conseillé de configurer un réseau supplémentaire, appelé "cluster network", réservé aux opérations internes comme la réplication et la récupération des données. Cela permet d'isoler le trafic interne du cluster du trafic client.

### *Protocole de communication - Messenger V2*

Le protocole Messenger V2 permet aux démons et clients CEPH de communiquer entre eux de manière sécurisée et flexible. Il possède quelques caractéristiques comme une amélioration de l'encapsulation des authentifications (intégration future de Kerberos) et une meilleure annonce/négociation des fonctionnalités au début de la communication. Dans la V2, les démons écoutent sur le port 3300.

Le mode CRC est utilisé par défaut. Il fournit une authentification lors du début de la communication et réalise une vérification d'intégrité via CRC32C. Pour se protéger d'attaque 'man-in-the-middle', utiliser en plus le mode sécurisé permettra de chiffrer les données qui y transitent.

### *Protocole d'authentification – CephX*

Le protocole CephX est le mécanisme d'authentification cryptographique activé par défaut pour sécuriser les communications entre les clients et les services du cluster. Le monitor CEPH remplit le rôle d'autorité d'authentification en possédant une base de données de secrets. Un token est remis après authentification auprès du monitor qui permettra de réaliser des actions avec les services CEPH. La désactivation de CephX permet d'économiser en puissance de calcul, mais est vivement déconseillé.

## Outils de déploiement

### Cephadm

Cephadm est un outil de déploiement natif qui permet de déployer directement sur des machines virtuelles ou physiques sans avoir Kubernetes. La gestion des services sur plusieurs nœuds passe par SSH et les containers (Docker). Il permet d'automatiser le provisionnement et mise à jour du cluster. Il est idéal pour un déploiement CEPH natif sans Kubernetes. La perte et récupération d'un nœud se repose sur CEPH lui-même pour détecter la perte et restaurer les données. Sur des forums, le déploiement avec Cephadm est plus simple et direct que celui avec Rook. Cependant, la récupération d'un nœud défaillant est plus difficile.

## ROOK

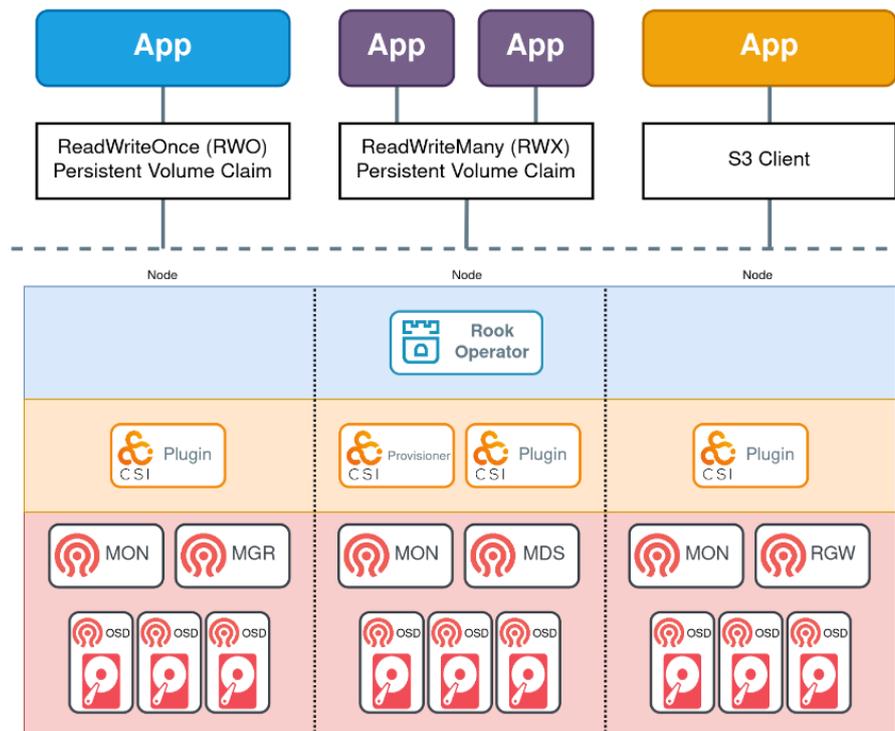
### C'est quoi ?

Rook est un orchestrateur de stockage cloud natif qui permet de gérer et déployer CEPH sur Kubernetes de manière automatisée et simplifiée. Il agit comme un opérateur de stockage pour Kubernetes. Il permet de simplifier le déploiement, la configuration de CEPH, détecter / récupérer des défaillances et gérer des volumes persistants sur Kubernetes.

L'opérateur a été déclaré comme étant stable depuis décembre 2018 avec la version 0.9. Cela résulte en une solution prête pour la production depuis quelques années. Le projet Rook est reconnu comme 'Graduated' par la Cloud Native Computing Foundation (CNCF) attestant un haut niveau de maturité, croissance et d'adoption.

## Architecture

## Rook Architecture

*Opérateur Rook - rook-ceph-operator*

L'architecture de Rook tourne autour du concept d'Opérateur Kubernetes, un contrôleur qui automatise la gestion de ressources complexes à partir de simples définitions déclaratives. L'opérateur est déployé sous la forme d'un Pod nommé 'rook-ceph-operator'. Similaire au CEPH monitor, il surveille en continu l'état du cluster mais via les Custom Resource Definitions (CRDs) comme CephCluster, CephBlockPool, ou CephFilesystem.

Après avoir défini les paramètres d'un cluster CEPH à travers des scripts de configurations Kubernetes, l'opérateur va automatiquement déployer une série de pods Ceph essentiels (comme les moniteurs, les OSDs, les managers, etc.). Ces composants sont gérés à travers des objets Kubernetes comme les Deployments, StatefulSets, ou ReplicaSets, selon le type de service. Par exemple, les OSDs sont souvent lancés dans des Deployment, un par disque détecté, tandis que les moniteurs sont gérés en StatefulSet pour garantir leur stabilité d'identité réseau. L'opérateur adapte également dynamiquement le nombre de réplicas de certains services en fonction de l'état du cluster

*CSI Driver - csi-rbdplugin, csi-cephfsplugin (DaemonSet + Deployment)*

Le Container Storage Interface permet à Kubernetes de créer, attacher, détacher ou supprimer des volumes Ceph à la demande via des demandes de stockages persistants (volumes). Il est composé de plusieurs pods : certains sont déployés sur chaque nœud (DaemonSet), d'autres sont des contrôleurs (Deployment). Il agit comme le pont entre Kubernetes et le cluster Ceph

pour les opérations de volume. Les Pods exécutant les plugins `csi-rbdplugin` gèrent les volumes blocs et les Pods exécutant les plugins `csi-cephfsplugin` gèrent les volumes en mode fichier partagé

#### *rook-discover (DaemonSet)*

C'est un DaemonSet déployé automatiquement par l'opérateur Rook pour scanner les disques disponibles sur chaque nœud du cluster Kubernetes. Déployé sur chaque nœud, il détecte tous les périphériques de stockage physiques (comme des disques durs ou SSD) présents sur les nœuds du cluster. Ces informations sont utilisées par l'opérateur Rook pour savoir quels disques peuvent être utilisés par CEPH, notamment pour les OSD.

#### *Ceph Monitors - rook-ceph-mon (StatefulSet)*

Les moniteurs CEPH sont déployés et répliqués dans des Pods Kubernetes grâce au StatefulSet. Chaque moniteur garde son identité et ses données pour garantir la cohérence du cluster.

#### *Ceph Managers - rook-ceph-mgr (Deployment)*

Avec un script de déploiement Kubernetes, les mangagers CEPH sont déployés et répliqués grâce au ReplicaSet Kubernetes. Les managers apportent des fonctions de gestion supplémentaire : collecte de métriques, interface web (dashboard), intégration Prometheus, ...

#### *Ceph Object Storage Daemons - rook-ceph-osd (Deployment ou DaemonSet)*

Grâce à un Déploiement ou DaemonSet Kubernetes, un pod exécutant un Object Storage Daemon CEPH est déployé pour chaque disque. Un OSD est responsable de la réplication, de la récupération et du placement des objets sur son disque respectif.

#### *Ceph Metadata Servers - rook-ceph-mds (Deployment ou StatefulSet)*

Requis quand CephFS, le système de fichier distribué est activé. Ils gèrent les **métadonnées** (noms de fichiers, hiérarchie, droits), tandis que les OSDs gèrent le contenu des fichiers.

#### *Ceph RADOS Gateway - rook-ceph-rgw (Deployment)*

Optionnel, Le RADOS Gateway permet d'exposer Ceph comme un service de stockage objet compatible S3. Il est déployé via un script de déploiement Kubernetes.

#### *Ceph RADOS Gateway Service - rgw-service (ClusterIP ou LoadBalancer)*

Ce service Kubernetes fournit un point d'accès S3-compatible si on a activé le stockage objet.

#### *Intertafec web - dashboard-service (ClusterIP ou NodePort ou LoadBalancer)*

Ce service kubernetes permet d'exposer l'interface web. On peut ouvrir un port externe pour y accéder depuis une machine externe au cluster

## Custom Ressources Definitions

Les CRDs (Custom Resource Definitions) sont au centre du fonctionnement de Rook : ce sont les objets personnalisés que nous déclarons pour que l'opérateur Rook déploie et configure Ceph selon nos besoins.

#### *CephCluster*

Ce CRD est le point de départ de tout déploiement Ceph avec Rook. Il représente un cluster Ceph complet, et indique à l'opérateur Rook comment l'installer, où placer les données, combien de composants déployer, et comment configurer le réseau.

#### **Paramètres utiles :**

- `spec.mon.count` – nombre de moniteurs (souvent 3)
- `spec.dataDirHostPath` – chemin local sur chaque nœud pour stocker les données Ceph
- `spec.storage.useAllNodes` – activer l'utilisation automatique de tous les nœuds
- `spec.storage.useAllDevices` – utiliser tous les disques disponibles
- `spec.resources` – ressources CPU/mémoire allouées à chaque composant Ceph
- `spec.dashboard.enabled` – active ou non le dashboard Ceph
- `spec.network.hostNetwork` – active le mode réseau "host" pour les pods Ceph

### *CephBlockPool*

Ce CRD permet de créer un pool RADOS Block Device dans Ceph, utilisé ensuite par Kubernetes via le CSI `csi-rbdplugin` pour fournir du stockage en mode bloc aux pods. Ce pool est ensuite lié à un `StorageClass`.

#### **Paramètres utiles:**

- `spec.replicated.size` – nombre de répliquions
- `spec.compressionMode` – mode de compression des données
- `spec.mirroring.enabled` – option de mirroring pour Ceph Block

### *CephFilesystem*

Ce CRD permet de créer un CephFS, système de fichiers accessible depuis plusieurs Pods en lecture/écriture. Il est utilisé via le CSI CephFS et repose sur des pools pour stocker les fichiers et les métadonnées.

#### **Paramètres utiles:**

- `spec.metadataPool.replicated.size` – nombre de répliquions des pools de métadonnées
- `spec.dataPools[].replicated.size` – nombre de répliquions des fichiers utilisateurs
- `spec.metadataServer.activeCount` – nombre de MDS actifs
- `spec.mirroring.enabled` – option de mirroring pour CephFS

### *CephObjectStore*

Ce CRD permet de déployer le composant RADOS Gateway, qui fournit une interface de stockage objet compatible S3. Il permet ensuite de créer des buckets et d'y stocker des objets.

#### **Paramètres clés :**

- `spec.gateway.instances` – nombre de pods RGW à déployer
- `spec.gateway.port` – port HTTP d'accès à l'API S3
- `spec.gateway.securePort` – port HTTPS (optionnel)
- `spec.metadataPool.replicated.size` – nombre de répliquions des métadonnées S3
- `spec.dataPool.replicated.size` – nombre de répliquions des objets stockés

### *CephObjectStoreUser*

Ce CRD permet de créer un utilisateur S3 lié à un objet `CephObjectStore` ci-dessus. Il génère une paire `accessKey` / `secretKey` stockée dans un `Secret` Kubernetes.

## Guide pour appliquer/gérer un CRD Rook

- Écrire un objet personnalisé (par exemple, `CephCluster`) dans un fichier `yaml`.

- **Appliquer le CRD pour créer et/ou mettre à jour le cluster** avec la commande suivante :  
`kubectl apply -f <fichier.yaml>`
- **Modifier un objet CRD existant directement dans le cluster**  
`kubectl edit cephcluster -n <namespace> <name object ceph>`  
Cela ouvre l'objet yaml dans un éditeur. Nous pouvons modifier des champs (ex : `dashboard.enabled`),. Rook détecte les changements et les applique automatiquement.
- **Vérifier l'état du CRD**  
`kubectl get cephcluster -n <namespace>`
- **Supprimer un objet CRD**  
`kubectl delete cephcluster -n <namespace> <name object ceph>`

## Différences avec Cephadm

La principale différence entre Cephadm et Rook réside dans l'environnement d'exécution. Cephadm installe CEPH de manière native alors que Rook transforme CEPH en un service Kubernetes. La gestion des services se fait avec la commande 'ceph orch' dans Cephadm. Dans Rook, les CRD permettent de gérer services CEPH qui sont dans pods Kubernetes.

Avec Cephadm, la détection d'une panne est réalisée par le monitor CEPH. La recréation des OSD et l'ajout d'un nouveau nœud doit se faire manuellement avec les commandes dédiées. Avec Rook, la détection de la panne se fait via le Node Controller. La recréation des OSD, ainsi que l'ajout d'un nouveau nœud sont automatisés.

Le choix se repose essentiellement entre un contrôle élargi malgré une configuration plus manuelle et une automatisation maximale des services dans un environnement Kubernetes déjà en place mais un contrôle légèrement amoindri.

# Kubernetes

## Comparaisons

### Kubernetes (K8s)

K8s est la plateforme d'orchestration de conteneurs la plus populaire, connue pour sa scalabilité et flexibilité. Elle est adaptée aux déploiements sur site et dans le cloud, et convient aux applications conteneurisées à grande échelle. K8s donne accès à des fonctionnalités avancées telles que la mise à l'échelle automatique, l'auto-guérison et les mises à jour progressives. Il est idéal pour les environnements de production complexes.

Dans le cas d'un déploiement sur des Raspberry Pi 3, K8s est trop lourd. Il demande beaucoup trop de ressources en RAM et CPU. Son installation et sa maintenance sont plus complexes par rapport aux deux autres version Kubernetes.

### K3s

Développé par Rancher Labs, K3s est une version allégée de Kubernetes, conçue pour les environnements aux ressources limitées, tels que l'edge computing et l'IOT. K3s est facile à déployer et à gérer. Il garde l'essentiel des fonctionnalités de K8s tout en réduisant la complexité et les exigences matérielles. Il est optimisé pour ARM. Il supporte les Custom Resource

Definitions nécessaires au fonctionnement de Rook. K3s a été certifié par la CNCF comme projet 'Sandbox' et cela assure une certaine fiabilité et sécurité.

Probablement le meilleur choix dans le cadre d'une utilisation avec un Raspberry Pi 3 : Il est très léger (moins de 100 MO). Il a été optimisé pour l'architecture ARM. Il est simple à installer et gérer. Il possède moins de dépendances que K8s. Il correspond bien à une utilisation pour un petit cluster. La plupart des exemples de déploiements de CEPH dans des Raspberry trouvés sur le web ont choisi K3s.

## K0s

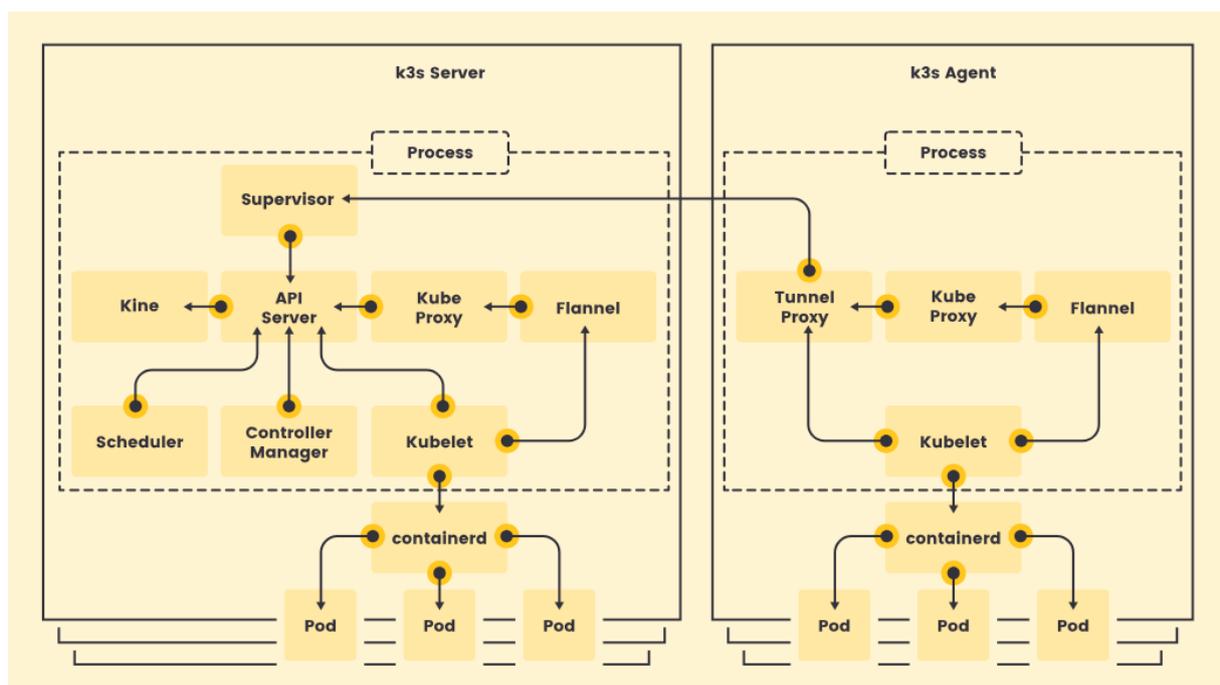
Développée par Google, K0s est une distribution Kubernetes légère et sécurisée, adaptée aux environnements aux ressources limitées. Elle possède une architecture monolithique composé d'un seul binaire. Cela facilite le déploiement et réduit la consommation de ressources. Cette plateforme conteneur native a été prévu pour lancer des applications conteneurisées dans un environnement de calcul distribué.

K0s constitue comme une alternative viable à K3s. Il n'a pas de dépendances système du fait de son architecture. Il supporte l'architecture ARM du Raspberry Pi 3. Il est facile à mettre à jour et sécuriser. Sa taille est plus légère que celle de K3s. Dans sa documentation officielle, on y trouve un guide sur l'installation de CEPH avec Rook et K0s.

## Choix final – K3s

K3s reste l'option la plus populaire en ce qui concerne le déploiement dans un environnement aux ressources limitées. Malgré qu'il soit un peu moins léger par rapport à K0s, son adoption est plus répandue. Par conséquent, il a été bien éprouvé dans ce cas d'utilisation et dispose d'une communauté active avec un support fiable. Il est aussi maintenu par Rancher.

## K3s



## Noeuds

### Serveur

Le noeud serveur est responsable de la gestion du cluster. Il contient les composants critiques et nécessaires pour gérer le cluster. Parmi eux, il y a le Scheduler (Attribue un pod à un noeud), le Controller Manager (Supervise les ressources), la base de données (etcd), le CNI (Flannel qui gère le réseau des Pods). Un serveur peut exécuter des Pods et leurs conteneurs, sauf si le flag '-disable-agent' y est défini. Au moins 3 serveurs sont requis pour avoir un cluster à Haute Disponibilité.

### Agent

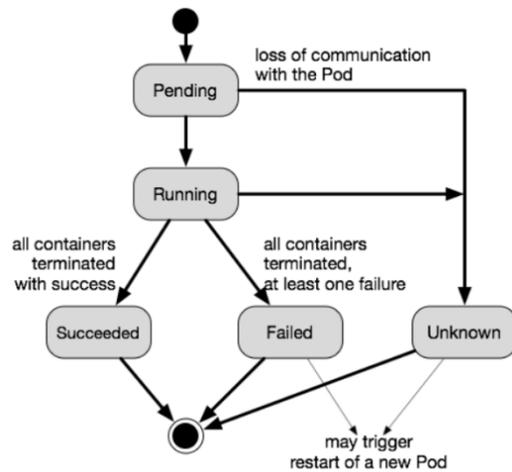
Le noeud agent est une machine qui s'occupe exclusivement de l'exécution des Pods et leurs conteneurs. Ils suivent les ordres donnés par le(s) serveur(s).

## Objets Kubernetes natif

### Pod

Un Pod est la plus petite unité de calcul que nous pouvons créer et gérer dans Kubernetes. Un Pod peut s'agir d'un ou plusieurs conteneurs disposant de configurations de réseau partagées, d'un stockage partagé et d'une spécification pour savoir comment démarrer le(s) conteneur(s). À travers un contexte partagé, le contenu des Pods sont colocalisés et coplanifiés. Le contexte partagé d'un pod est constitué d'un set d'espaces de nom et de cgroups Linux pour une isolation du Pod. L'utilisation la plus commune des Pods consiste à un Pod par conteneur. Cependant, il est possible d'avoir plusieurs conteneurs pour un même Pod dans certains cas.

Concernant le cycle de vie d'un Pod, s'il meurt, alors un autre sera démarré pour le remplacer.



### Service (ClusterIP, NodePort ou LoadBalancer)

Un service permet de fournir un nom d'hôte ou une adresse IP pour accéder à un service implémenté dans un ou plusieurs Pods où ces derniers peuvent changer au fil du temps, notamment leur adresse IP. En d'autres termes, un service permet d'exposer un groupe de Pods sur le réseau. Les services peuvent être de différents types, en fonction du niveau d'exposition réseau souhaité.

Le type par défaut, ClusterIP, crée un service interne uniquement accessible depuis le cluster. C'est le cas par exemple des moniteurs Ceph ou du service interne vers le gestionnaire Ceph (MGR). Ces services ne sont pas accessibles depuis l'extérieur, ce qui convient parfaitement à la plupart des communications internes.

Si l'on veut accéder à un service depuis l'extérieur du cluster, Kubernetes propose deux autres types de services : NodePort et LoadBalancer. Le service de type NodePort ouvre un port fixe sur chaque nœud du cluster et cela donne un accès externe via l'adresse IP du nœud et le port assigné. C'est utilisé pour des interfaces comme le dashboard Ceph.

Le type LoadBalancer est utilisé dans les environnements cloud : Kubernetes demande au fournisseur cloud de provisionner un équilibreur de charge avec une IP publique. Cela permet d'accéder à des interfaces comme l'API S3 ou le dashboard de monitoring.

## ReplicaSet

Un ReplicaSet garantit qu'un certain nombre fixe de pods sont en cours d'exécution à tout moment. Il gère de la réplication sans-état : les Pods sont interchangeables. Si un pod tombe, il en recrée un autre, sans se soucier de ses informations sur son nom et identité. Le Déploiement ci-dessous en génère généralement.

## StatefulSet

Similaire au ReplicaSet, le StatefulSet gère de la réplication avec état : l'identité (nom et volume) de chaque Pod persiste. Après un redémarrage, les Pods concernés retrouvent leurs noms et volumes. Leurs déploiements et arrêts se font dans un ordre précis. Dans le contexte de Rook/CEPH, les moniteurs et OSD sont déployés avec un StatefulSet.

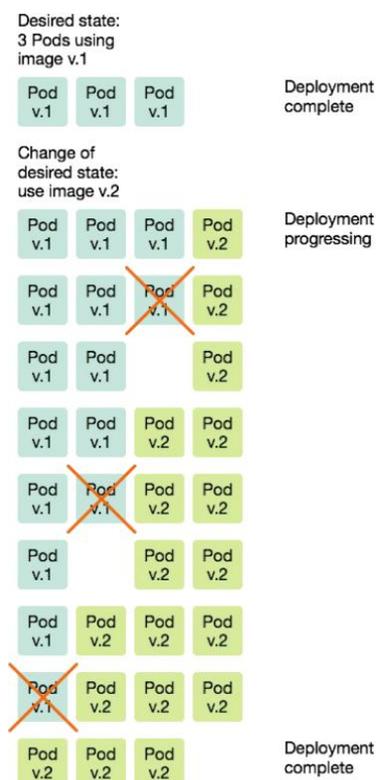
## DaemonSet

Le DaemonSet s'assure qu'un pod tourne sur chaque nœud (ou certains nœuds sélectionnés). Le rook-discover utilise un DaemonSet pour détecter les disques sur chaque nœud du cluster.

## Déploiement

Un déploiement permet de gérer un groupe de Pods qui y exécute une application. En fonction d'un état souhaité défini dans le déploiement, le contrôleur va mettre à jour l'état actuel de l'application vers celui désiré. Il est possible de définir le nombre de répliqués des Pods concernées par le déploiement.

Pour éviter une interruption de l'accès à l'application durant une mise à jour, Kubernetes va effectuer un roulement des mises à jour : Un nouveau groupe de réplicas est créé. Puis, tant qu'il reste un Pod pas encore mis à jour, un Pod avec l'état souhaité est ajouté et un Pod avec l'ancien état est supprimé juste après.



## StorageClass

Un StorageClass représente un modèle de stockage dynamique que Kubernetes peut utiliser pour créer automatiquement des volumes persistants lorsqu'un pod en demande un à travers un PersistentVolumeClaim.

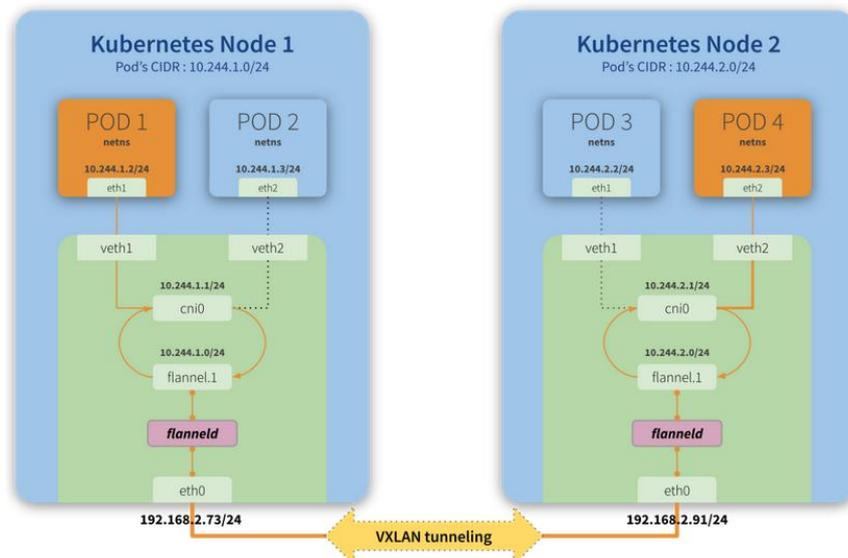
## Composants dans un nœud K3s

### Containerd

Containerd est un moteur de conteneurs léger et rapide utilisé dans Docker et Kubernetes. Il a été conçu pour superviser et exécuter les conteneurs. Exclusivement dédié à la gestion des conteneurs, il ne permet pas de créer des images, une fonctionnalité disponible dans Docker.

Containerd supporte nativement Kubernetes. Au lancement d'un Pod, le composant Kubelet va demander à Containerd de créer un nouveau conteneur. Containerd télécharge l'image et l'ex

## Flannel



Flannel est un CNI (Container Network Interface) qui permet de créer un sous-réseau plus petit pour les Pods au sein d'un nœud Kubernetes. Il attribue une adresse IP privée pour chaque Pod. Par défaut, il utilise un CIDR 10.244.0.0/16 qu'il va allouer en de plus petit sous-réseau avec le CIDR 10.244.X.0/24 dans chacun des nœuds.

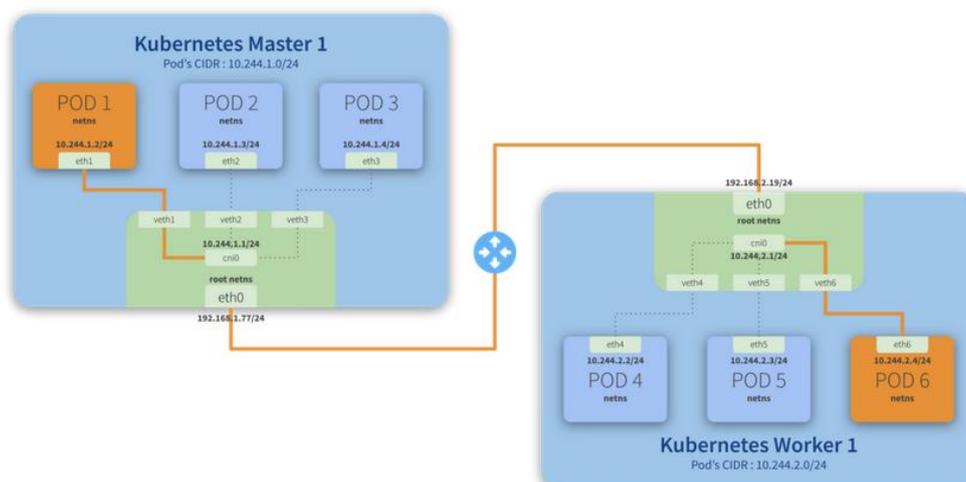
### Flanneld

Flanneld est un daemon responsable de veiller à ce que les routes entre les nœuds restent à jour et disponibles pour les communications entre eux.

### VXLAN

VXLAN est un protocole qui permet de créer un tunnel réseau pour transporter les paquets entre les nœuds. VXLAN encapsule les paquets dans UDP pour le faire transiter dans le sous-réseau.

## Kube-proxy



Sur chaque noeud, le Kube-proxy gère le routage des requêtes entre les pods Kubernetes. La redirection des requêtes se fait grâce à la configuration des iptables qui permet de déterminer le Pod cible à atteindre.

## API Server

L'API serveur s'occupe de l'attribution des adresses IP aux Services Kubernetes. Il expose l'API REST Kubernetes pour toutes interactions externes avec le cluster. Il stocke l'état du cluster dans une base de données comme etcd, SQLite, ...

## Kine

Kine permet de traduire l'API etcd pour un autre moteur de base de données comme SQLite, Postgres, MySQL, MariaDB et NATS. Il n'est pas réservé à K3s et peut être utilisé dans d'autres versions Kubernetes. Il traduit les appels etcdTX vers ceux l'API souhaité (Create, Update, Delete). Dans notre cas, en vue de créer un cluster HA, nous n'envisageons pas d'utiliser une base de données externe (SQL). L'option par défaut dans ce cas de figure est le stockage dans un système etcd.

## Controller Manager

Le Controller Manager est un processus qui exécute les contrôleurs Kubernetes. Il y a plusieurs types de contrôleurs Kubernetes: Le Node controller est responsable de notifier et alerter quand un noeud est tombé, le Job controller surveille les tâches à exécuter dans un Pod à créer, l'EndpointSlice controller répertorie les objets du même nom qui permettent de relier les services aux Pods associés et etc.

## Scheduler

Le Scheduler s'occupe de l'attribution des nouveaux pods au sein d'un nœud choisi par lui. La décision du choix d'un nœud dépend de plusieurs facteurs : ressources demandées, contraintes hardware/software, spécifications d'affinité entre les Pods, proximités avec la source de données, interférences entre workloads (séparer les Pods qui peuvent se gêner mutuellement à cause d'une utilisation intensive du CPU, RAM et réseau) et contraintes de délais (priorité aux Pods qui doivent s'exécuter rapidement).

## Kubelet

Kubelet est un agent qui vérifie le bon fonctionnement des Pods en exécution dans le noeud. Il vérifie selon les spécifications des Pods définies si leur état actuel le respecte.

## Etcd

Etcd est un stockage clé/valeur utilisé par Kubernetes pour y mettre les données du cluster. Chaque nœud maître récupère les configurations du cluster dans son etcd embarqué. Les configurations sont synchronisées entre les nœuds. Parmi les nœuds maîtres, un a le rôle de leader qui lui permet d'écrire dans le stockage etcd. Les autres ne peuvent que synchroniser et lire dans leur stockage respectif.

## Sources

CEPH ROOK

<https://docs.ceph.com/en/reef/start/>

<https://rook.io/docs/rook/latest-release/Getting-Started/intro/>

[https://www.youtube.com/watch?v=WVsv3Ca\\_Y70](https://www.youtube.com/watch?v=WVsv3Ca_Y70)

*Kubernetes*

<https://k3s.io/>

<https://k0sproject.io/>

<https://www.nops.io/blog/k0s-vs-k3s-vs-k8s/>

[https://www.reddit.com/r/kubernetes/comments/1i5n75f/anyone\\_using\\_k3smicrok8sk0s\\_in\\_production/](https://www.reddit.com/r/kubernetes/comments/1i5n75f/anyone_using_k3smicrok8sk0s_in_production/)