



Formation Kubernetes
Architecture des Clusters

Popa Ionut

07/03/2022

Table des matières

1	Architecture Kubernetes	3
2	Description d'un cluster Kubernetes	3
2.1	Configuration du cluster	4
3	Les composantes applicatives du Nœud Master	5
3.1	kube-api-server	5
3.2	etcd	6
3.3	kube-scheduler	6
3.4	kube-controller	6
4	Les composantes applicatives du Nœud Worker	7
4.1	kubelet	7
4.2	Container Runtime (CRE)	7
4.3	kube-proxy	8

1 Architecture Kubernetes

Ce document fournit des informations sur les différentes composantes d'infrastructure qu'on peut retrouver dans un cluster Kubernetes. Le document n'a pas l'ambition d'être un guide exhaustif de tous les briques d'un cluster mais seulement un document facilitant la compréhension du rôle de chaque composante dans un cluster.

Au cours de ce documents nous allons resumer les notions suivantes :

- Description d'un cluster Kubernetes
- Les différents types de Noeuds
- Les composantes applicatives d'un cluster.
- Les objets Kubernetes

Chaque notion abordée sera accompagnée des exemples et commandes kubectl.

2 Description d'un cluster Kubernetes

Un cluster Kubernetes est un ensemble de machines virtuelles ou physiques (servers) permettant d'exécuter des applications conteneurisées.

Si vous exécutez Kubernetes, vous exécutez un cluster. Au minimum, un cluster contient un Noeud (server) de type control-plane (connu également sous le nom de Master) et une ou plusieurs machines worker.

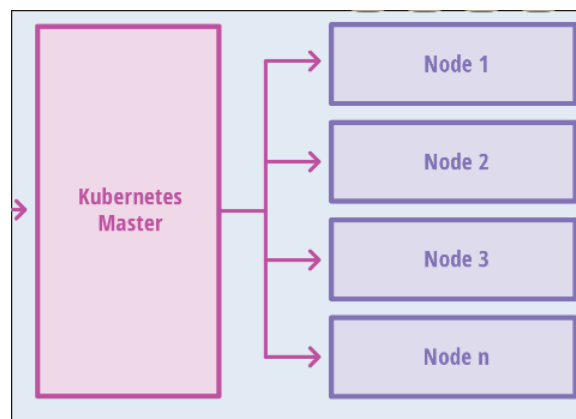


Figure 1 – Les neuds d'un cluster Kubernetes

Le Control-Plane est responsable du maintien de l'état souhaité du cluster, par exemple les applications en cours d'exécution et les images de conteneur qu'elles utilisent.

Les nœuds Worker exécutent les applications et les charges applicatives (scripts, cronjobs etc..).

Le cluster est au cœur de l'avantage clé de Kubernetes : la possibilité de planifier et d'exécuter des conteneurs sur un groupe de machines, qu'elles soient physiques ou virtuelles, sur site ou dans le cloud.

Les conteneurs Kubernetes ne sont pas liés à des machines individuelles. Au contraire, ils sont abstraits à travers le cluster.

```
# Obtenir l'ensemble de Noeuds dans mon cluster.
$ kubectl get nodes -o wide
```

2.1 Configuration du cluster

Un cluster Kubernetes a un état souhaité, qui définit quelles applications (scripts, jobs, services etc..) doivent être exécutées, ainsi que les images de conteneur qu'elles doivent utiliser, les ressources qui doivent être mises à leur disposition et d'autres détails de configuration.

Un état souhaité est défini par des fichiers de configuration, qui sont des fichiers en format JSON ou YAML déclarant le type d'application à exécuter et le nombre de répliques nécessaires pour faire fonctionner un stack applicatif.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Figure 2 – Fichier de configuration type en format YAML

L'état souhaité du cluster est communiqué via les fichiers de configuration à l'API Kubernetes.

La configuration peut être envoyée via la ligne de commande (en utilisant kubectl) ou en utilisant l'API (via des applications web, Kubernetes Dashboard) pour interagir avec le cluster afin de définir ou et comment modifier l'état souhaité.

Kubernetes gèrera automatiquement votre cluster en fonction de l'état souhaité. À titre d'exemple, supposons que vous déployez une application avec un état souhaité de 3 replicas, ce qui signifie que 3 répliques de l'application doivent être en cours d'exécution.

Si l'un de ces répliques tombe en panne, Kubernetes constatera que seules deux répliques sont en cours d'exécution et en ajoutera une autre pour satisfaire l'état souhaité.

3 Les composantes applicatives du Nœud Master

Nous allons expliquer le rôle des différentes composantes qu'on peut retrouver sur les nœuds de type Master.

Les nœuds Master forment le control-plane du cluster, celui-ci représente le “cerveau” central de notre infrastructure.

Tout changement d'état de notre cluster passe par un Nœud Master avant d'être appliqué sur les nœuds workers.

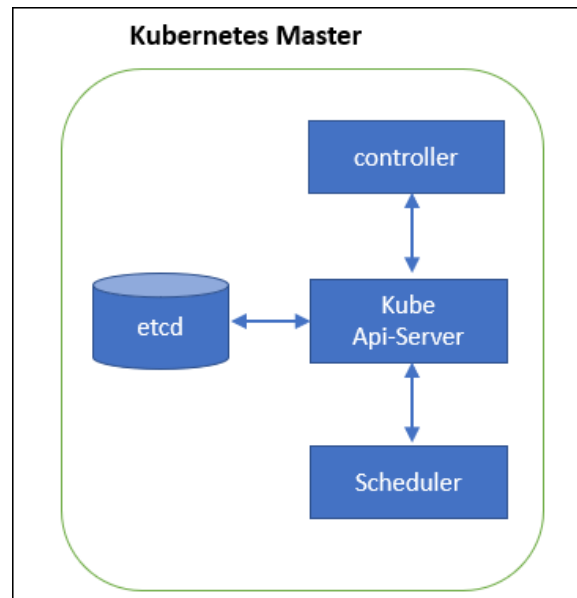


Figure 3 – Composantes principales d'un Nœud Master

3.1 kube-api-server

Fichier de configuration : `/etc/kubernetes/manifests/kube-apiserver.yaml`

Namespace : kube-system

Le serveur API sert d'interface pour la communication avec le plan de contrôle (ensemble de un ou plusieurs Nœuds Master constituant un cluster).

Il est chargé d'exposer l'API (Application Programming Interface) Kubernetes, ce qui garantit que le plan de contrôle peut traiter les demandes externes et internes.

Le serveur API accepte les demandes, détermine si elles sont valides et les exécute. Les ressources externes peuvent accéder directement à l'API via des appels REST, tandis que les composants internes du cluster y accèdent généralement à l'aide d'outils en ligne de commande comme `kubectl` ou `kubernetes-dashboard` qui expose une interface web pour l'interaction avec le cluster.

Comme vu précédemment la définition de l'état du cluster se fait à partir d'un fichier de configuration en format YAML.

Cette demande sera envoyée vers le api serve sous la forme d'une recette REST.

L'api-server se chargera de vérifier l'identité de l'utilisateur, les droits que celui-ci possède au sein du cluster et validera la requête.

Si la requête faite par l'utilisateur passe tous les vérifications faites par l'api-server ceux-ci seront enregistrées dans la base de données etcd.

3.2 etcd

Fichier de configuration : **/etc/kubernetes/manifests/etcd.yaml**

Namespace : kube-system

ETCD est une base de données type clé valeur dans laquelle l'état de notre cluster est stocké, tout changement transmis vers l'api-server doit être écrit dans la base de données avant qu'il soit appliqué par les autres composants Kubernetes.

Sachant que la base de données contient l'état de notre cluster, un backup de celle-ci pourrait être utilisé pour rétablir le cluster à un état précédent.

Attention : Les données liées à nos applications déployées dans le cluster ne sont pas stockées dans la base de données etcd, seulement l'état de nos composants est stocké (pods, services, deployments etc..)

3.3 kube-scheduler

Fichier de configuration : **/etc/kubernetes/manifests/kube-scheduler.yaml**

Namespace : kube-system

Dans le cadre d'un déploiement d'application dans un cluster Kubernetes le cluster doit décider sur quel Nœud de type Worker mettre les objets (Pods) qui concernent notre déploiement.

Pour faire cela Kubernetes utilise un composant appelé kube-scheduler.

Celle-ci se charge d'identifier le Nœud qui respecte les demandes de l'administrateur (**filtering** sur : zones, type cpu, type RAM etc..) ensuite parmi tous les nœuds choisis comme potentielles hôtes, il identifie celui qui est le moins chargé (**scoring**) (pour assurer que les ressources demandées par l'application soient présentes).

Après avoir identifié le/les nœud(s) qui porteront la charge applicative, ceux-ci seront inscrits dans la base de données etcd en tant que cible du déploiement.

Maintenant le kube-controller pourra lancer le déploiement des applications.

3.4 kube-controller

Fichier de configuration : **/etc/kubernetes/manifests/kube-controller-manager.yaml**

Namespace : kube-system

La composante kube-controller surveille la base de données etcd, dans le cas où il identifie des objets qui doivent être déployés sur un nœud mais qui ne le sont pas encore il envoie les instructions aux Nœuds Workers pour créer les objets demandés.

Dans le cas où une instance de notre application tombe, le contrôleur observe la différence entre l'état courant et l'état cible, il va demander au composant responsable de la création des objets sur les Nœuds de les créer pour arriver à l'état souhaité.

4 Les composantes applicatives du Nœud Worker

Nous allons expliquer le rôle des différentes composantes qu'on peut retrouver sur les nœuds de type Worker.

Comme nous l'avons mentionné précédemment les workers sont chargés d'accueillir les applications métier qu'on déploie dans notre cluster.

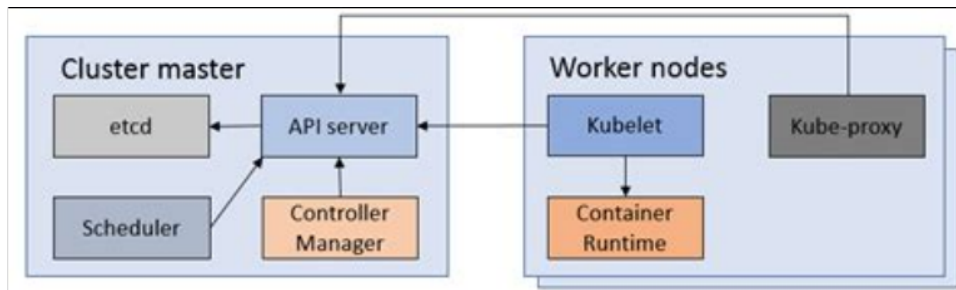


Figure 4 – Composantes de Nœuds Worker

4.1 kubelet

Fichier de configuration : **/etc/kubernetes/kubelet.conf**

Namespace : pas de namespace, directement déployé sur la machine.

Quand le controller-manager envoie des directives vers les workers celles-ci sont reçues par kubelet.

Cette composante est responsable de la communication entre le Master et le Worker.

Les changements de l'état au niveau du Nœud sont transmis par le kubelet vers l'api-server qui lui a son tour les écrit dans la base de données etcd.

Les autres composantes se chargeront de surveiller les instances de etcd et actionner pour arriver à l'état souhaité par l'administrateur.

kubelet n'est pas déployé par le cluster Kubernetes lors de la création du cluster, il est installé directement sur le Nœud en tant qu'application (via des package manager comme apt, dkpg, rpm ou pacman).

Dans le cas où le kubelet n'arrive plus à envoyer des informations au control-plane pendant plus de 5 minutes (par défaut) le nœud est considéré comme mort et les applications seront transférées sur les autres nœuds.

Kubelet reçoit les

4.2 Container Runtime (CRE)

Fichier de configuration : **Dépend du CRE installé**

Namespace : pas de namespace, directement déployé sur la machine.

CRE ou Container Runtime Engine est la composante responsable de la création et du management de conteneurs.

Kubelet applique les instructions envoyées par le controller-manager en s'appuyant sur le CRE pour les opérations comme :

- Création de conteneur
- Suppression de conteneur
- Configuration des paramètres réseau, environnement et montage des volumes disque dans le conteneur.

Le CRE communique seulement avec le kubelet.

4.3 kube-proxy

Fichier de configuration : **/var/lib/kube-proxy/config.conf**

Namespace : kube-system

Cette composante est présente sur tous les Noeuds et elle est responsable de la création des règles de flux pour autoriser ou interdire la communication entre les conteneurs.

Le kube-proxy va assurer la communication entre les conteneurs et redirige le trafic entre les répliquas en cas de besoin.