

Kubernetes

N. Salleron B. Affes

Lundi 12 Février 2018

Sommaire

- 1 Introduction
 - Introduction
- 2 Docker
 - Some few things about Docker
- 3 Kubernetes Core Concept
 - Pods
 - Label et Selector
 - Réplication et Mise à jour continue
 - Deployment et Service
- 4 Kubernetes Architecture Concept
 - Architecture concept
 - Kubernetes Node
 - Kubernetes Master
 - General Overview
- 5 Focus sur le Scheduler de Kubernetes
 - Algorithme du scheduler
- 6 Conclusion
 - Conclusion
 - Références

Historique

Une longue émergence

■ Borg

- Démarrage en 2004.
- Développé en interne.
- Manager de containers.
- Objectif : réduction des coups en partageant machines et applications.
- **Inconvéniant : notion de travail, gestion des ports**
- **Non open-source.**

Historique

Une longue émergence

■ Borg

- Démarrage en 2004.
- Développé en interne.
- Manager de containers.
- Objectif : réduction des coups en partageant machines et applications.
- **Inconvéniant : notion de travail, gestion des ports**
- **Non open-source.**

■ Omega

- Fils de Borg.
- Amélioration de l'écosystème apporté par Borg.
- **Non open-source.**

Historique

Une longue émergence

- Borg
 - Démarrage en 2004.
 - Développé en interne.
 - Manager de containers.
 - Objectif : réduction des coups en partageant machines et applications.
 - **Inconvéniant : notion de travail, gestion des ports**
 - **Non open-source.**
- Omega
 - Fils de Borg.
 - Amélioration de l'écosystème apporté par Borg.
 - **Non open-source.**
- Kubernetes
 - Adaptable à plusieurs infrastructure cloud.
 - **Open-source.**

Introduction

Nom venant du Grec, crée par 3 ingénieurs de chez Google en 2014.

- *Orchestrateur* - Gestionnaire de conteneur.
- Exécute et manages des containers.
- Propose une API permettant la gestion de plusieurs clouds (Google, Microsoft, Amazon, et pleins d'autres).
- 100% Open Source écrit en Go.



FIGURE – Logo de Kubernetes

Il permet de se focus sur les applications et non sur le déploiement. Google exécute 2 milliards de conteneurs par semaine avec ces systèmes.

Introduction

Nom venant du Grec, crée par 3 ingénieurs de chez Google en 2014.

- *Orchestrateur* - Gestionnaire de conteneur.
- Exécute et manages des containers.
- Propose une API permettant la gestion de plusieurs clouds (Google, Microsoft, Amazon, et pleins d'autres).
- 100% Open Source écrit en Go.



FIGURE – Logo de Kubernetes

Il permet de se focus sur les applications et non sur le déploiement. Google exécute 2 milliards de conteneurs par semaine avec ces systèmes.

Dernière version : 1.9.3 (sortie il y a 3 jours)

Introduction

Nom venant du Grec, crée par 3 ingénieurs de chez Google en 2014.

- *Orchestrateur* - Gestionnaire de conteneur.
- Exécute et manages des containers.
- Propose une API permettant la gestion de plusieurs clouds (Google, Microsoft, Amazon, et pleins d'autres).
- 100% Open Source écrit en Go.



FIGURE – Logo de Kubernetes

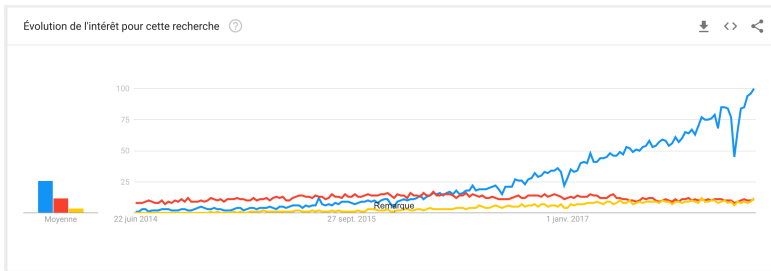
Il permet de se focus sur les applications et non sur le déploiement. Google exécute 2 milliards de conteneurs par semaine avec ces systèmes.

Dernière version : 1.9.3 (sortie il y a 3 jours)

"manage application, not machines" - Tim Hockin

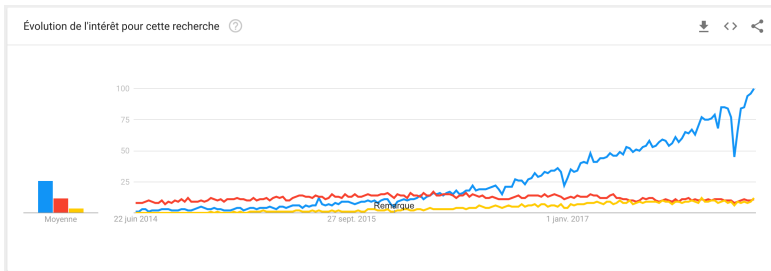
Popularité

Évolution des recherches entre Kubernetes, Mesos, Docker Swarm



Popularité

Évolution des recherches entre Kubernetes, Mesos, Docker Swarm



Une communauté très active :

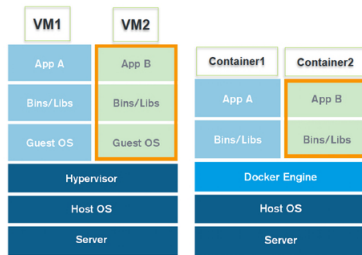
- Actuellement 61000 commits avec plus de 1500 contributeurs

Contributions to master, excluding merge commits





Docker est un conteneur léger, permettant de l'isolation entre les processus.



- Retire le coût de la virtualisation (pas de gestion hardware)
- Retire le coût d'exécution de plusieurs OS.

Docker se base sur deux technologies du noyau :

- CGroups
- Namespace

Docker se base sur deux technologies du noyau :

- CGroups
- Namespace

Control Groups

Feature kernel qui permet de contrôler, limité et isoler l'usage des ressources pour un processus ou une collection de processus.

Docker se base sur deux technologies du noyau :

- CGroups
- Namespace

Control Groups

Feature kernel qui permet de contrôler, limité et isoler l'usage des ressources pour un processus ou une collection de processus.

CGroups Isolation

- Quantitative Isolation : Les CGroups ne peuvent pas avoir plus de pages que la limite imposé.
- Qualitative Isolation : Les CGroups doivent accéder à leur mémoire comme si elles étaient seules sur la machine.

Docker se base sur deux technologies du noyau :

- CGroups
- Namespace

Control Groups

Feature kernel qui permet de contrôler, limité et isoler l'usage des ressources pour un processus ou une collection de processus.

CGroups Isolation

- Quantitative Isolation : Les CGroups ne peuvent pas avoir plus de pages que la limite imposé.
- Qualitative Isolation : Les CGroups doivent accéder à leur mémoire comme si elles étaient seules sur la machine.

Namespace

Feature linux qui permet de créer une vue local pour les ressources d'un systèmes. Les ressources en dehors du namespace ne sont pas visible.

Kubernetes



kubernetes

Pods

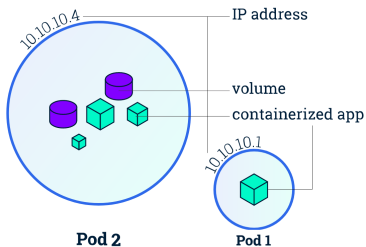


FIGURE – Les Pod dans Kubernetes

Caractéristiques du Pod

- Unité de base de l'ordonnancement.

Pods

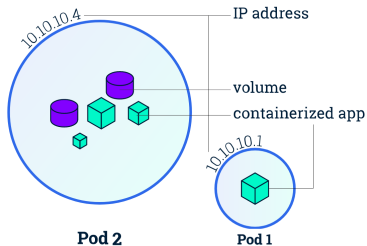


FIGURE – Les Pod dans Kubernetes

Caractéristiques du Pod

- Unité de base de l'ordonnancement.
- Vue abstraite de composants conteneurisés.

Pods

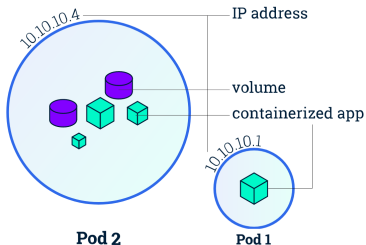


FIGURE – Les Pod dans Kubernetes

Caractéristiques du Pod

- Unité de base de l'ordonnancement.
- Vue abstraite de composants conteneurisés.
- Il peut regrouper 1 ou * conteneurs.
=> Couplage fort.

Pods

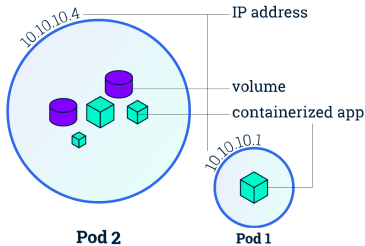


FIGURE – Les Pod dans Kubernetes

Caractéristiques du Pod

- Unité de base de l'ordonnancement.
- Vue abstraite de composants conteneurisés.
- Il peut regrouper 1 ou * conteneurs.
=> Couplage fort.
- Chaque pod possède une adresse IP unique (limité au cluster).

Pods

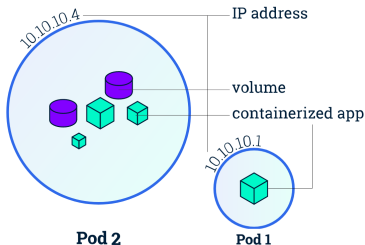


FIGURE – Les Pod dans Kubernetes

Caractéristiques du Pod

- Unité de base de l'ordonnancement.
- Vue abstraite de composants conteneurisés.
- Il peut regrouper 1 ou * conteneurs.
=> Couplage fort.
- Chaque pod possède une adresse IP unique (limité au cluster).
- Un Pod peut définir un volume. Il a la même durée de vie que le Pod.

Pods

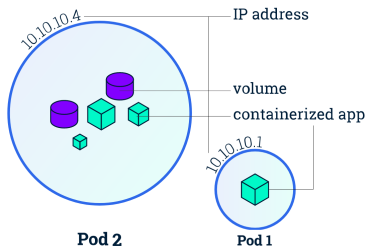


FIGURE – Les Pod dans Kubernetes

Caractéristiques du Pod

- Unité de base de l'ordonnancement.
- Vue abstraite de composants conteneurisés.
- Il peut regrouper 1 ou * conteneurs.
=> Couplage fort.
- Chaque pod possède une adresse IP unique (limité au cluster).
- Un Pod peut définir un volume. Il a la même durée de vie que le Pod.

Bénéfices du pod :

- Plusieurs conteneurs dans 1 Pod
=> Processus qui ont besoin d'interroger un autre processus avec une faible latence.

Pods

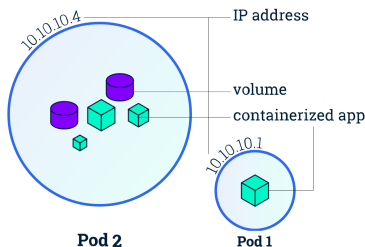


FIGURE – Les Pod dans Kubernetes

Caractéristiques du Pod

- Unité de base de l'ordonnancement.
- Vue abstraite de composants conteneurisés.
- Il peut regrouper 1 ou * conteneurs.
=> Couplage fort.
- Chaque pod possède une adresse IP unique (limité au cluster).
- Un Pod peut définir un volume. Il a la même durée de vie que le Pod.

Bénéfices du pod :

- Plusieurs conteneurs dans 1 Pod
=> Processus qui ont besoin d'interroger un autre processus avec une faible latence.
- Utilisable sous plusieurs environnements (fichier de configuration indépendant de la plateforme)

Pods

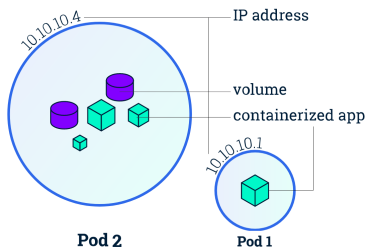


FIGURE – Les Pod dans Kubernetes

Caractéristiques du Pod

- Unité de base de l'ordonnancement.
- Vue abstraite de composants conteneurisés.
- Il peut regrouper 1 ou * conteneurs.
=> Couplage fort.
- Chaque pod possède une adresse IP unique (limité au cluster).
- Un Pod peut définir un volume. Il a la même durée de vie que le Pod.

Bénéfices du pod :

- Plusieurs conteneurs dans 1 Pod
=> Processus qui ont besoin d'interroger un autre processus avec une faible latence.
- Utilisable sous plusieurs environnements (fichier de configuration indépendant de la plateforme)
- Mortel : un container peut mourir.

Label et Selector

Label

- Méta-données arbitraire attaché à un objet.
- Forme (K :V)
- Représente généralement une identité.

Selector

- Permet de sélectionner plusieurs objets.
- API supporte deux types de selector :
 - equality-based : "=", "==", "!="
 - set-based : "in", "notin", "exists"

```
"labels": {  
  "key1" : "value1",  
  "key2" : "value2"  
}
```

FIGURE – Exemple K :V format JSON

```
environment = production  
tier != frontend
```

FIGURE – Selector "equality-based"

```
environment in (production, qa)  
tier notin (frontend, backend)
```

FIGURE – Selector "set-based"

Exemple d'utilisation

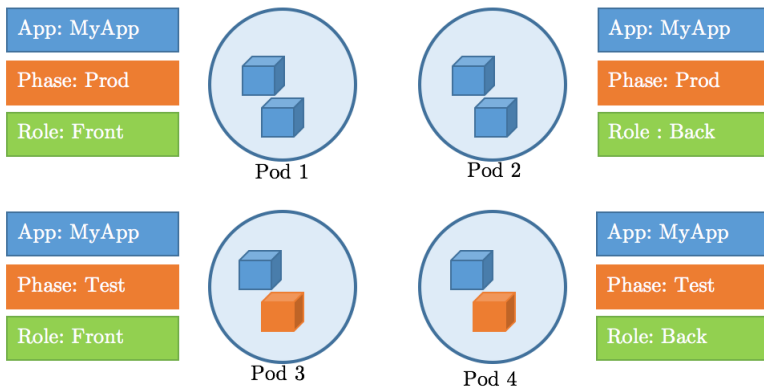


FIGURE – Exemple avec différents selectors

Exemple d'utilisation

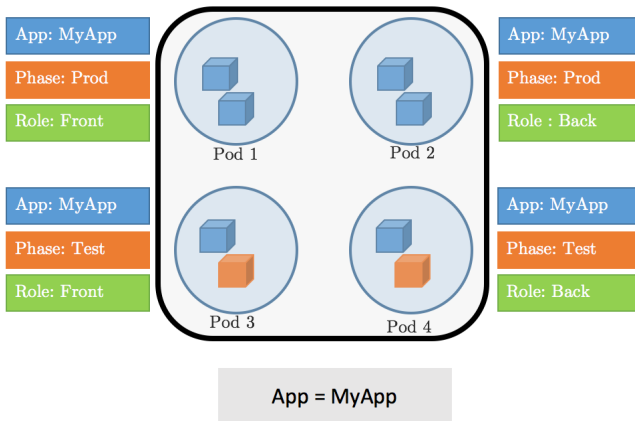


FIGURE – Exemple avec différents selectors

Exemple d'utilisation

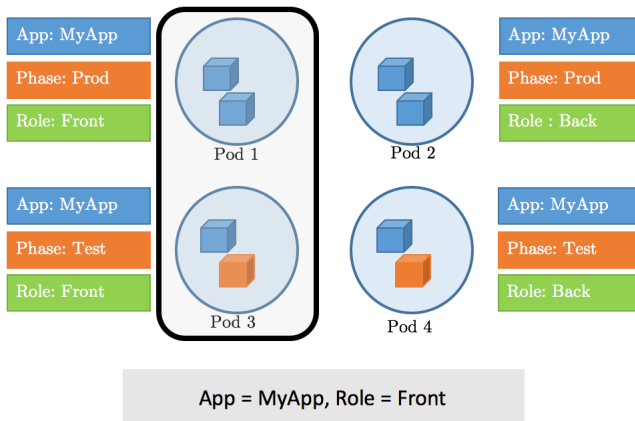


FIGURE – Exemple avec différents selectors

Exemple d'utilisation

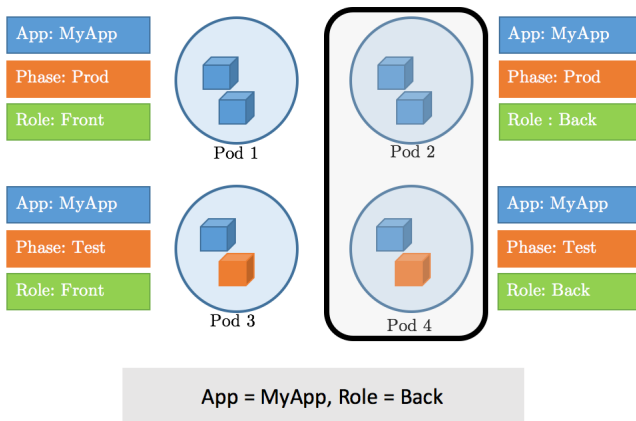


FIGURE – Exemple avec différents selectors

Exemple d'utilisation

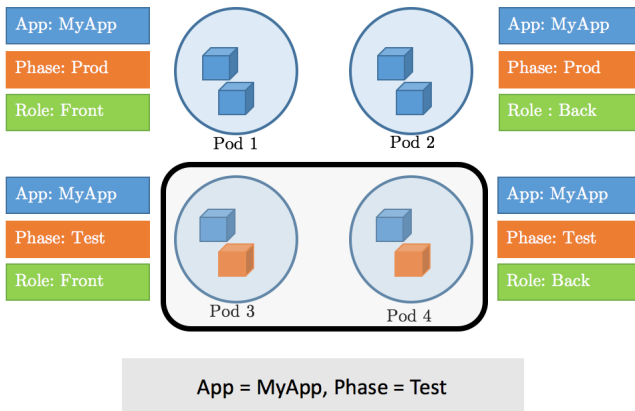


FIGURE – Exemple avec différents selectors

Les ReplicaSet

Objectifs

- Gère l'unité basique dans Kubernetes, le Pod.
- Il s'assure que le nombre de Pod voulu est présent.
 - Groupe les Pods via des Selectors
 - Si $n < \text{LIMIT}$: start Pod
 - Si $n > \text{LIMIT}$: kill Pod
- Les Pods répliqués n'ont pas d'identité propre.
- Ce sont des consommables.
- Peut-être utilisé pour une mise à l'échelle horizontale automatisé.

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

FIGURE – Mise à l'échelle horizontale automatique

Attention

Les ReplicaSet sont déconseiller pour faire de la mise à jour continue.

=> A utiliser seulement pour des applications n'ayant pas besoin de mise à jour

Deployment et Service

Deployment

- Possède et gère 1 ou plusieurs Replica Sets.
- Permet la mise à jour continue avec 3 paramètres :
 - minReadySeconds
 - maxSurge
 - maxUnavailable
- Permet également le rollback après deployment.

Service

- Une abstraction d'un ensemble logique de pod et d'une politique d'accès aux Pods.
- Adapté aux micro-services
- Load-balancing

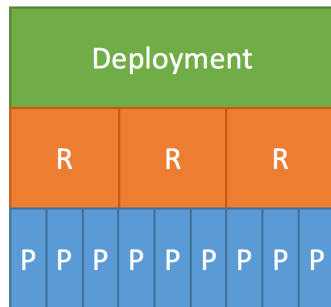


FIGURE – Relation entre Objet Deployment, ReplicaSet et Pod

Exemple du Rolling-Update

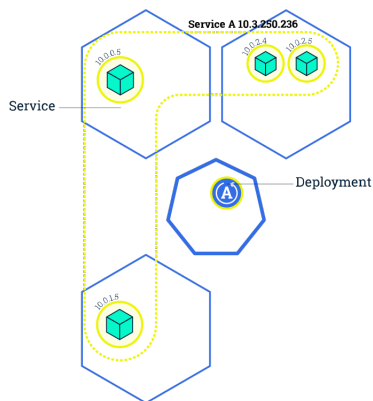


FIGURE – Exemple du Rolling Update tiré de la documentation Kubernetes

Exemple du Rolling-Update

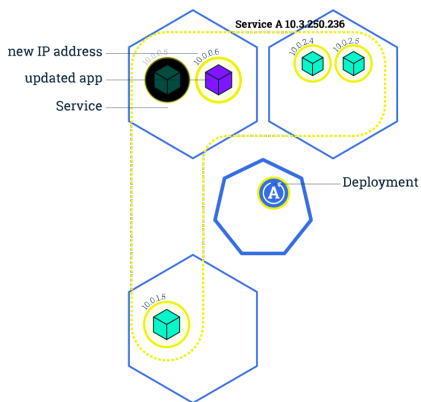


FIGURE – Exemple du Rolling Update tiré de la documentation Kubernetes

Exemple du Rolling-Update

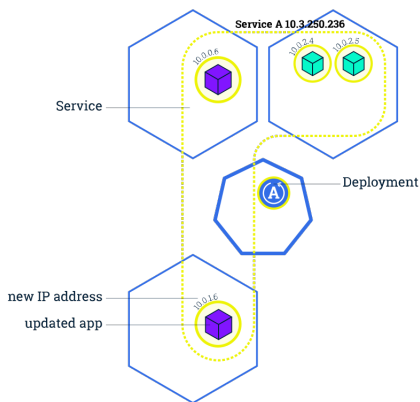


FIGURE – Exemple du Rolling Update tiré de la documentation Kubernetes

Exemple du Rolling-Update

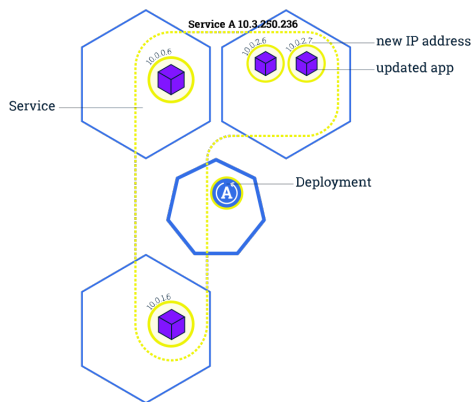


FIGURE – Exemple du Rolling Update tiré de la documentation Kubernetes

Kubernetes Node i.e Worker Node

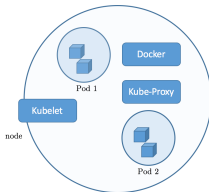


FIGURE – Exemple de node

Rôle du node

- Les pods sont exécuté dans des nodes.
- Il contient les services de gestion et de communications entre les containers.
- Il assigne les ressources au containers qui sont ordonnancé par le master node.

Kubernetes Node i.e Worker Node

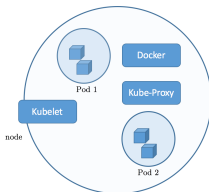


FIGURE – Exemple de node

Rôle du node

- Les pods sont exécuté dans des nodes.
- Il contient les services de gestion et de communications entre les containers.
- Il assigne les ressources au containers qui sont ordonnancé par le master node.

Kubelet

- Il est responsable de l'exécution (start/stop/maintenance)
- Il surveille l'état d'un pod, communique l'état du node au master node.
- Interaction avec un moteur Docker sous-jacent pour démarrer des conteneurs.

Kubernetes Node i.e Worker Node

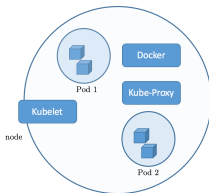


FIGURE – Exemple de node

Rôle du node

- Les pods sont exécuté dans des nodes.
- Il contient les services de gestion et de communications entre les containers.
- Il assigne les ressources au containers qui sont ordonnancé par le master node.

Kubelet

- Il est responsable de l'exécution (start/stop/maintenance)
- Il surveille l'état d'un pod, communique l'état du node au master node.
- Interaction avec un moteur Docker sous-jacent pour démarrer des conteneurs.

Kube-proxy

- Routage du trafic vers le conteneur grâce à l'IP/Port et *répartiteur de charge*.

Kubernetes Master

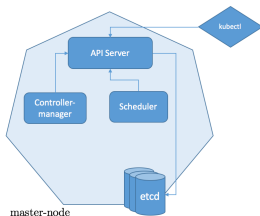


FIGURE – Exemple de master node

Master Node

Le master node est responsable du management du cluster Kubernetes.

- C'est le point d'entrée de toutes les tâches administratives.
- Il est le responsable de *l'orchestration* des worker-nodes
- Il est composé de plusieurs sous éléments.

Kubernetes Master

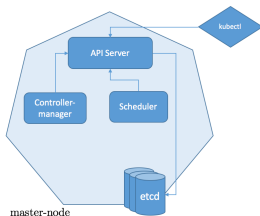


FIGURE – Exemple de master node

Master Node

Le master node est responsable du management du cluster Kubernetes.

- C'est le point d'entrée de toutes les tâches administratives.
- Il est le responsable de *l'orchestration* des worker-nodes
- Il est composé de plusieurs sous éléments.

Master Node - l'API Server

- C'est le point d'entrée pour toutes les commandes utilisé pour contrôler le cluster.
- Il récupère les commandes (REST), les valide, et les exécute.
- Le résultat de ces commandes est conservé dans *etcd*.

Kubernetes Master

Master Node - etcd

- etcd est utilisé pour partager la configuration et la découverte de service.
- API pour des opérations CRUD et permet aux nœuds de s'enregistrer

Kubernetes Master

Master Node - etcd

- etcd est utilisé pour partager la configuration et la découverte de service.
- API pour des opérations CRUD et permet aux nœuds de s'enregistrer

Master Node - kube-scheduler

Le scheduler permet le deployment de pods configurer et de services sur les nodes.

- Il possède les informations des ressources disponibles.
- Il possède également les informations requises pour l'exécution du service.
- Il choisit l'endroit du déploiement.

Kubernetes Master

Master Node - etcd

- etcd est utilisé pour partager la configuration et la découverte de service.
- API pour des opérations CRUD et permet aux nœuds de s'enregistrer

Master Node - kube-scheduler

Le scheduler permet le deployment de pods configurer et de services sur les nodes.

- Il possède les informations des ressources disponibles.
- Il possède également les informations requises pour l'exécution du service.
- Il choisit l'endroit du déploiement.

Master Node - kube-controller-manager

C'est un composant qui possède plusieurs controllers.

- Node Controller : Responsable de la bonne gestion des noeuds.
- Replication Controller : Responsable du maintien du bon nombre de pods pour chaque ReplicatSet objet du système.
- Endpoints Controller : Service qui associe service et pods.

Kubernetes Overview

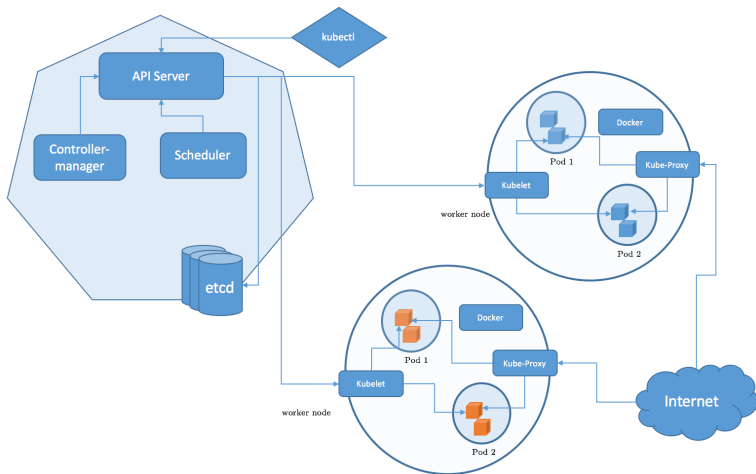


FIGURE – Overview

Comment Kubernetes Scheduler récupère-t-il un Pod ?

D'après le commit 5871b50 de <https://github.com/kubernetes/kubernetes/> il y a 3 grandes étapes :

```
// Etape 1 : lancement d'une goroutine sched.scheduleOne  
go wait.Until(sched.scheduleOne, 0, sched.config.StopEverything)
```

Comment Kubernetes Scheduler récupère-t-il un Pod ?

D'après le commit 5871b50 de <https://github.com/kubernetes/kubernetes/> il y a 3 grandes étapes :

// Etape 1 : lancement d'une goroutine `sched.scheduleOne`

`go` `wait.Until(sched.scheduleOne, 0, sched.config.StopEverything)`

`func (sched *Scheduler) scheduleOne() {` // Etape 2 : Récupération du prochain pod

`pod := sched.config.getNextPod()`

`// Autres éléments du scheduler`

`// Call de la fonction schedule`

`}`

Comment Kubernetes Scheduler récupère-t-il un Pod ?

D'après le commit 5871b50 de <https://github.com/kubernetes/kubernetes/> il y a 3 grandes étapes :

// Etape 1 : lancement d'une goroutine `sched.scheduleOne`

go wait.`Until`(`sched.scheduleOne`, 0, `sched.config.StopEverything`)

`func` (`sched *Scheduler`) `scheduleOne`() { // Etape 2 : Récupération du prochain pod

 pod := `sched.config.getNextPod`()

 // Autres éléments du scheduler

 // Call de la fonction `schedule`

}

`func` (`c *configFactory`) `getNextPod`() *`v1.Pod` {

 pod, err := `c.podQueue.Pop`() // Etape 3 : On retire le pod d'une file de Pod

 if err == nil {

 return pod

 }

 return nil

}

Comment Kubernetes Scheduler récupère-t-il un Pod ?

D'après le commit 5871b50 de <https://github.com/kubernetes/kubernetes/> il y a 3 grandes étapes :

// Etape 1 : lancement d'une goroutine `sched.scheduleOne`

```
go wait.Until(sched.scheduleOne, 0, sched.config.StopEverything)
```

```
func (sched *Scheduler) scheduleOne() { // Etape 2 : Récupération du prochain pod
```

```
    pod := sched.config.getNextPod()
```

```
    // Autres éléments du scheduler
```

```
    // Call de la fonction schedule
```

```
}
```

```
func (c *configFactory) getNextPod() *v1.Pod {
```

```
    pod, err := c.podQueue.Pop() // Etape 3 : On retire le pod d'une file de Pod
```

```
    if err == nil {
```

```
        return pod
```

```
    }
```

```
    return nil
```

```
} // Les pods sont ajoutés à la queue par un EventHandler
```

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

- 2 ensemble de fonctions, exécuté indépendamment l'un de l'autre.
 - Predicates : Détermine quel node est adéquat pour recevoir le Pod.
 - Priorities : Donne un score pour chaque node restant de la première phase.
- Le scheduler choisit le node avec le plus haut score.

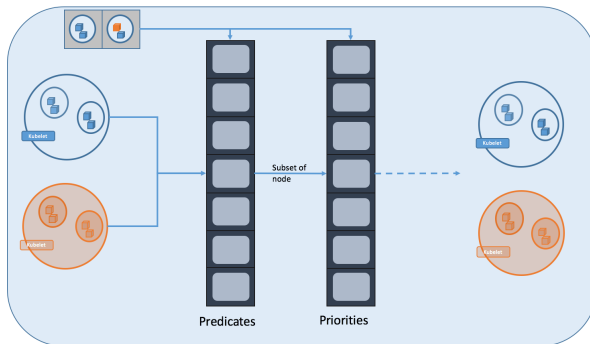


FIGURE — Overview du Kubernetes Scheduler

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Dans le code :

```
func (sched *Scheduler) scheduleOne() {
    // Call de la fonction schedule
    suggestedHost, err := sched.schedule(pod)
    if err != nil {
        // schedule() may have failed because the pod would not fit on any host
        return
    }
    // Bind suggestedHost et Pod
}
```

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Dans le code :

```
func (sched *Scheduler) scheduleOne() {  
    // Call de la fonction schedule  
    suggestedHost, err := sched.schedule(pod)  
    if err != nil {  
        // schedule() may have failed because the pod would not fit on any host  
        return  
    }  
    // Bind suggestedHost et Pod  
}  
  
func (sched *Scheduler) schedule(pod *v1.Pod) (string, error) {  
    host, err := sched.config.Algorithm.Schedule(pod, sched.config.NodeLister)  
    if err != nil {  
        //Traitement d'erreur  
        return "", err  
    }  
    return host, err  
}
```

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Le scheduler est configurable au startup de kubernetes.

Les fonctions de la phase **Predicates**

- Prevent overcommit : Le node donné remplit-il les exigences en terme de ressource du pod ?

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Le scheduler est configurable au startup de kubernetes.

Les fonctions de la phase **Predicates**

- Prevent overcommit : Le node donné remplit-il les exigences en terme de ressource du pod ?
- Prevent co-scheduling : On ne veut pas que notre Pod (Type A) se retrouve avec un autre Pod (Type B) via PodAntiAffinity.

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Le scheduler est configurable au startup de kubernetes.

Les fonctions de la phase **Predicates**

- Prevent overcommit : Le node donné remplit-il les exigences en terme de ressource du pod ?
- Prevent co-scheduling : On ne veut pas que notre Pod (Type A) se retrouve avec un autre Pod (Type B) via PodAntiAffinity.
- Force co-scheduling : On veut que notre Pod (Type A) se retrouve avec un autre Pod (Type B) via PodAffinity.

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Le scheduler est configurable au startup de kubernetes.

Les fonctions de la phase **Predicates**

- Prevent overcommit : Le node donné remplit-il les exigences en terme de ressource du pod ?
- Prevent co-scheduling : On ne veut pas que notre Pod (Type A) se retrouve avec un autre Pod (Type B) via PodAntiAffinity.
- Force co-scheduling : On veut que notre Pod (Type A) se retrouve avec un autre Pod (Type B) via PodAffinity.
- Dedicated-machines : On reserve certains node à certain type de pod
- Others...

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Les priorités : répond au problème du nombre de node restant après la première phase (ce dernier pouvant être très grand)

Les fonctions de la phase **Priorities**

- Best fit : Minimise la mémoire perdu à un instant t.

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Les priorités : répond au problème du nombre de node restant après la première phase (ce dernier pouvant être très grand)

Les fonctions de la phase **Priorities**

- Best fit : Minimise la mémoire perdu à un instant t.
- Worst fit : Maximise la mémoire perdu à un instant t.

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Les priorités : répond au problème du nombre de node restant après la première phase (ce dernier pouvant être très grand)

Les fonctions de la phase **Priorities**

- Best fit : Minimise la mémoire perdu à un instant t.
- Worst fit : Maximise la mémoire perdu à un instant t.
- CalculateSpreadPriority : Répartition des pods entre hôtes en prenant en compte les pods appartenant au même service ou au même contrôleur de réplication.
=> commit e273d89 du 14 Mar 2017.

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Les priorités : répond au problème du nombre de node restant après la première phase (ce dernier pouvant être très grand)

Les fonctions de la phase **Priorities**

- Best fit : Minimise la mémoire perdu à un instant t.
- Worst fit : Maximise la mémoire perdu à un instant t.
- CalculateSpreadPriority : Répartition des pods entre hôtes en prenant en compte les pods appartenant au même service ou au même contrôleur de réplication.
=> commit e273d89 du 14 Mar 2017.
- NodeAffinityPriority : Préférence pour que un pod soit sur un node.

Comment Kubernetes Scheduler détermine-t-il le bon node pour ce pod ?

Les priorités : répond au problème du nombre de node restant après la première phase (ce dernier pouvant être très grand)

Les fonctions de la phase **Priorities**

- Best fit : Minimise la mémoire perdu à un instant t.
- Worst fit : Maximise la mémoire perdu à un instant t.
- CalculateSpreadPriority : Répartition des pods entre hôtes en prenant en compte les pods appartenant au même service ou au même contrôleur de réplication.
=> commit e273d89 du 14 Mar 2017.
- NodeAffinityPriority : Préférence pour que un pod soit sur un node.
- PodAffinityPriority/PodAntiAffinityPriority : Préférence pour qu'un pod soit sur le même node (ou non) qu'un autre pod.

Une fois l'ensemble des fonctions exécutés, le scheduler prend celui avec le plus haut score.

Conclusion

- Kubernetes est un projet open-source, conçu à l'origine par Google et offert à la Cloud Native Computing Foundation.
- Il se base sur **Docker** pour la gestion des conteneurs.
- Il possède certains *Core Concept* comme :
 - Pods
 - ReplicatSet
 - Deployment
 - Service
- Il possède un node master et 1 ou plusieurs worker node.
- Son scheduler est configurable et se compose de deux faces
 - **Predicates** : qui retournera un subset de nodes correspondant aux prédicats
 - **Priorities** : qui assignera un score à chaque node. Le node ayant la priorité la plus haute sera choisit pour accueillir le pod.

Références

- Kubernetes Documentation. 2018. Nodes | Kubernetes. [ONLINE] Available at : <https://kubernetes.io/docs/concepts/architecture/nodes/>. [Accessed 10 February 2018].
- Kubernetes Documentation. 2018. Service | Kubernetes. [ONLINE] Available at : <https://kubernetes.io/docs/concepts/services-networking/service/>. [Accessed 10 February 2018].
- Kubernetes Documentation. 2018. Pods | Kubernetes. [ONLINE] Available at : <https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/>. [Accessed 10 February 2018].
- Kubernetes Documentation. 2018. Concepts | Kubernetes. [ONLINE] Available at : <https://kubernetes.io/docs/concepts/>. [Accessed 10 February 2018].
- acmqueue, BRENDAN BURNS, BRIAN GRANT, DAVID OPPENHEIMER, ERIC BREWER, AND JOHN WILKES, GOOGLE INC., 2016. Borg, Omega, and Kubernetes. System evolution, [Online]. 14/1, 70. Available at : <https://queue.acm.org/detail.cfm?id=2898444> [Accessed 10 February 2018].

Références

- Quora - Jorg Brown. 2015. What is Borg at Google ?. [ONLINE] Available at : <https://www.quora.com/What-is-Borg-at-Google>. [Accessed 10 February 2018].
- The Morning Paper - Adrian Colyer. 2015. Large-scale cluster management at Google with Borg. [ONLINE] Available at : <https://blog.acolyer.org/2015/05/07/large-scale-cluster-management-at-google-with-borg/>. [Accessed 10 February 2018].
- Carla Schroder. 2017. What Makes Up a Kubernetes Cluster ?. [ONLINE] Available at : <https://www.linux.com/news/learn/chapter/intro-to-kubernetes/2017/4/what-makes-kubernetes-cluster>. [Accessed 10 February 2018].
- Fabrice Jammes - Cloud Native Computing Foundation. 2017. Kubernetes et les micro-services. [ONLINE] Available at : https://indico.in2p3.fr/event/16962/attachments/46065/57449/Kube_webinaire_RI3_201801.pdf. [Accessed 10 February 2018].

Références

- Nune Isabekyan. 2016. Introduction to Kubernetes Architecture. [ONLINE] Available at : <https://x-team.com/blog/introduction-kubernetes-architecture/>. [Accessed 10 February 2018].
- Julia Evans. 2017. A few things I've learned about Kubernetes. [ONLINE] Available at : <https://jvns.ca/blog/2017/06/04/learning-about-kubernetes/>. [Accessed 10 February 2018].
- Julia Evans. 2018. How does the Kubernetes scheduler work ?. [ONLINE] Available at : <https://jvns.ca/blog/2017/07/27/how-does-the-kubernetes-scheduler-work/>. [Accessed 10 February 2018].
- YouTube. (2018). Kubernetes Scheduling Features or How Can I Make the System Do What I Want ? [I] - Marek Grabowski & Wojciech Tyczynski. [Online Video]. 17 April 2017. Available from : <https://www.youtube.com/watch?v=bbPcb2JuJPw&t=601s>. [Accessed : 10 February 2018].

Références

- YouTube. (2018). Container clusters with Kubernetes -Tim Hockin. [Online Video]. 21 April 2015. Available from : <https://www.youtube.com/watch?v=KIdK5WGZUms>. [Accessed : 10 February 2018].
- YouTube. (2018). Dockers Containers & Kubernetes - Brian Dorsey. [Online Video]. 7 June 2015. Available from : <https://www.youtube.com/watch?v=BaHp8YXIsD0>. [Accessed : 10 February 2018].

Merci

Merci pour votre attention