

# Kubernetes

manage application, not machines

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 object
  - Services
- 4 Kubernetes Architecture Concept
  - Architecture concept
  - Kubernetes Master
  - Kubernetes Node
- 5 Conclusion
  - Conclusion

# Historique

## Une longue émergence

### ■ Borg

- Démarrage en 2004.
- Développé en interne.
- Manager de containers.
- Objectif : réduction des coups en partageant machines et applications.
- **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.
- **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.
- **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

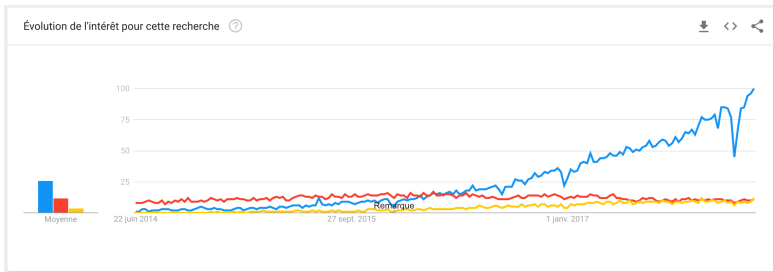
Il permet de se focus sur les applications et non sur le déploiement. Google exécute 2 milliards de conteneurs par semaine avec ces systèmes.

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

*"manage application, not machines" - Tim Hockin*

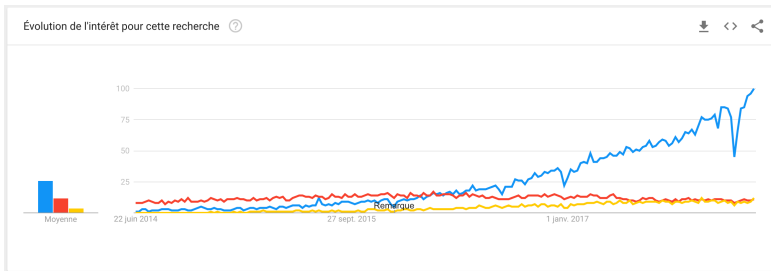
# Popularité

## 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

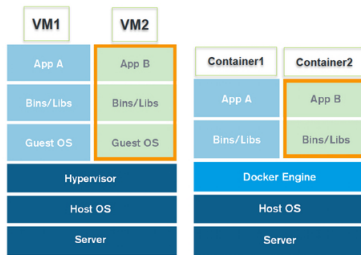
Contributions to master, excluding merge commits







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

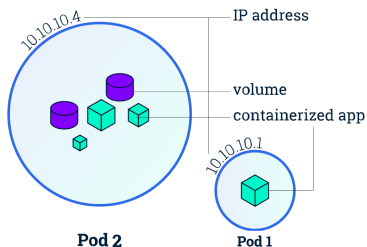


FIGURE – Les Pod dans Kubernetes

## Caractéristiques du Pod

- Unité de base de l'ordonnancement.



# Pods

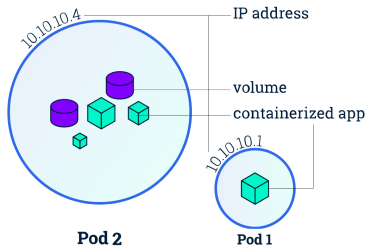


FIGURE – Les Pod dans Kubernetes

## Caractéristiques du Pod

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

# Pods

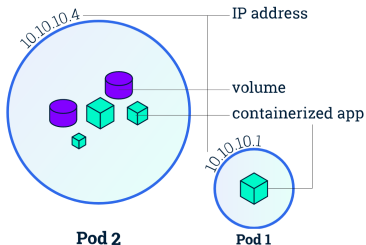


FIGURE – Les Pod dans Kubernetes

## Caractéristiques du Pod

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

# Pods

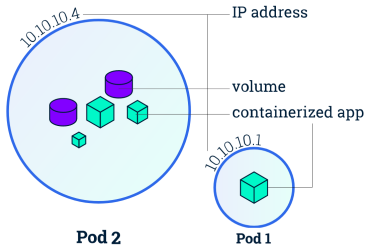


FIGURE – Les Pod dans Kubernetes

## 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

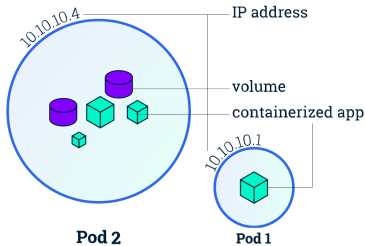


FIGURE – Les Pod dans Kubernetes

## 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

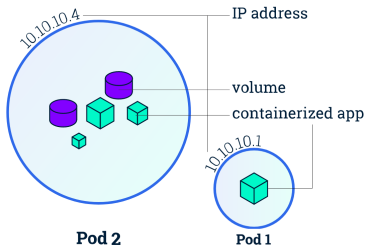


FIGURE – Les Pod dans Kubernetes

## 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

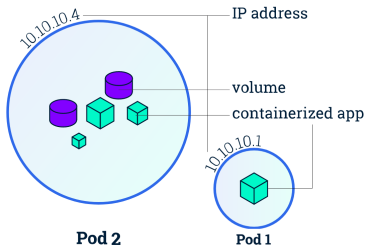


FIGURE – Les Pod dans Kubernetes

## 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

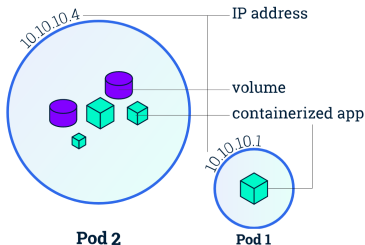


FIGURE – Les Pod dans Kubernetes

## 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.

# Pods

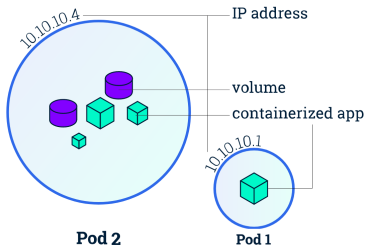


FIGURE – Les Pod dans Kubernetes

## 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

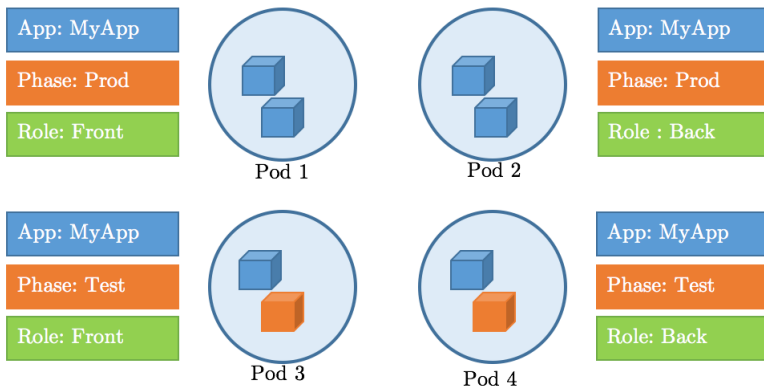


FIGURE – Exemple avec différents selectors

# Exemple d'utilisation

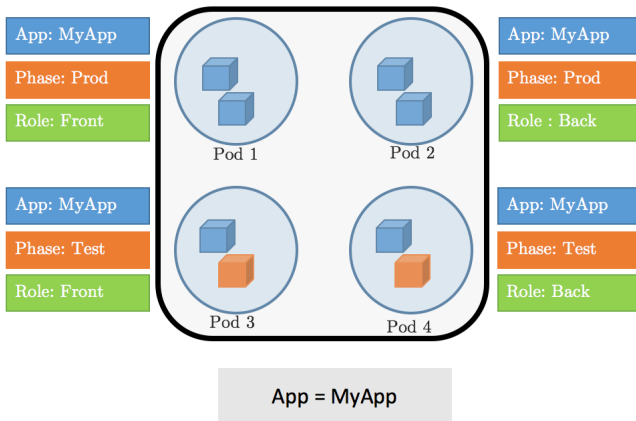


FIGURE – Exemple avec différents selectors

# Exemple d'utilisation

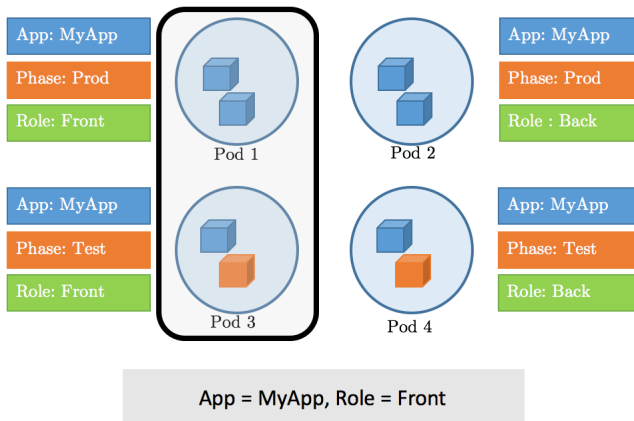


FIGURE – Exemple avec différents selectors

# Exemple d'utilisation

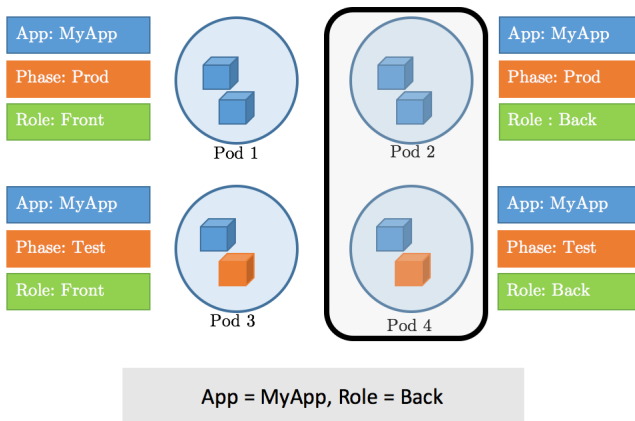


FIGURE – Exemple avec différents selectors

# Exemple d'utilisation

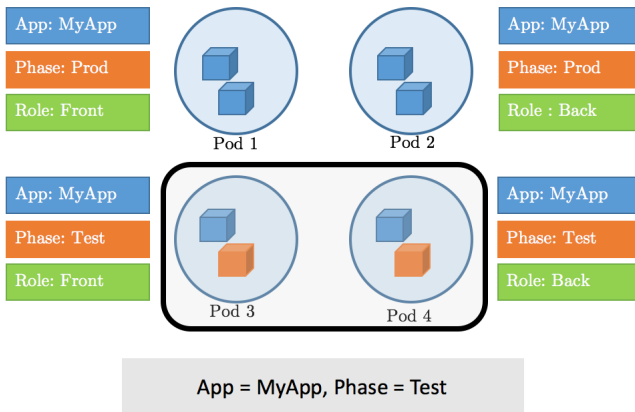


FIGURE – Exemple avec différents selectors

# Les ReplicatSet

## 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 ReplicatSet 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

## Un objet Deployment

- Possède et gère 1 ou plusieurs Replica Sets.
- Permet la mise à jour continue avec 3 paramètres :
  - minReadySeconds : Bootup Time de l'application
  - maxSurge : Indique le nombre de pod supplémentaire pendant le processus d'exécution.
  - maxUnavailable : Nombre de pod pouvant être indisponible pendant le processus de mise à jour.
- Permet également le rollback après deployment.

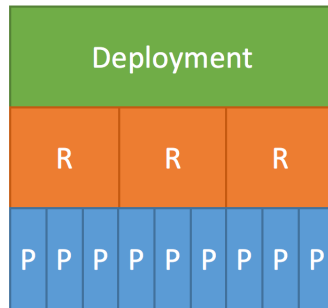


FIGURE – Relation entre Objet Deployment, ReplicaSet et Pod



# Services

# Kubernetes Master

# Kubernetes Node

# Conclusion