

# 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 et Service
- 4 Kubernetes Architecture Concept
  - Architecture concept
  - Kubernetes Node
  - Kubernetes Master
  - General Overview
- 5 Kubernetes scheduler work
  - Basic Algorithm
- 6 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.
- **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.

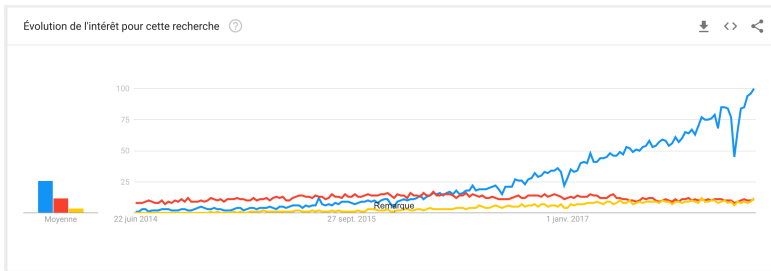
Dernière version : 1.9.3 (sortie il y a 3 jours)

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



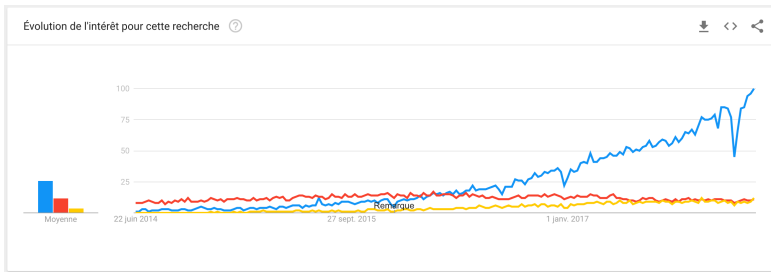
# Popularité

## Évolution des recherches entre Kubernetes, Mesos, Docker Swarm



# Popularité

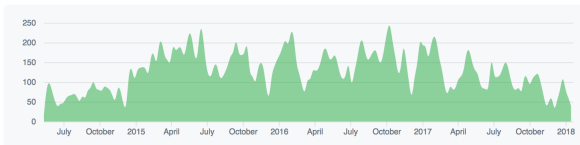
## Évolution des recherches entre Kubernetes, Mesos, Docker Swarm



Une communauté très active :

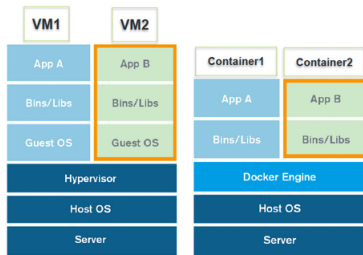
- Actuellement 61000 commits avec plus de 1500 contributeurs

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

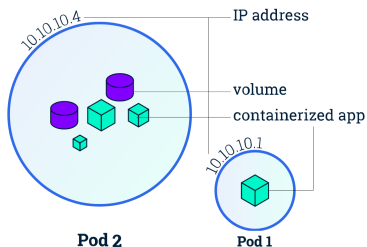


FIGURE – Les Pod dans Kubernetes

## Caractéristiques du Pod

- Unité de base de l'ordonnancement.

# Pods

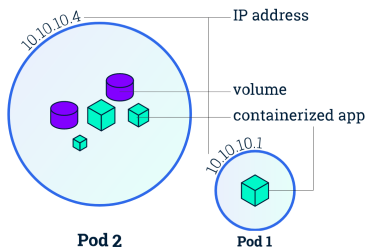


FIGURE – Les Pod dans Kubernetes

## Caractéristiques du Pod

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

# Pods

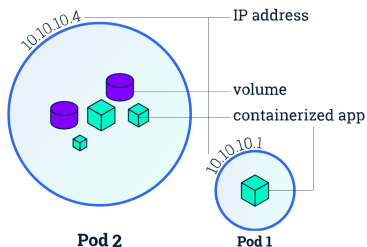


FIGURE – Les Pod dans Kubernetes

## Caractéristiques du Pod

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

# Pods

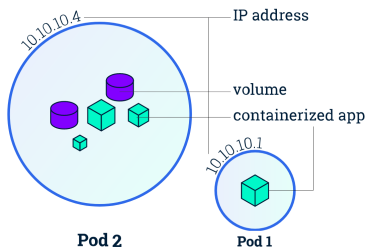


FIGURE – Les Pod dans Kubernetes

## 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

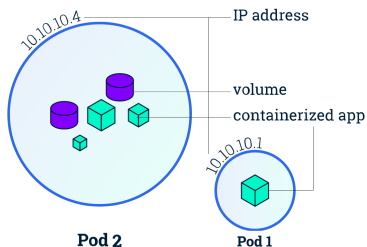


FIGURE — Les Pod dans Kubernetes

## 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

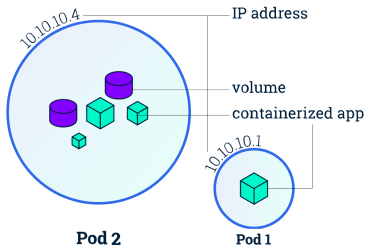


FIGURE – Les Pod dans Kubernetes

## 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

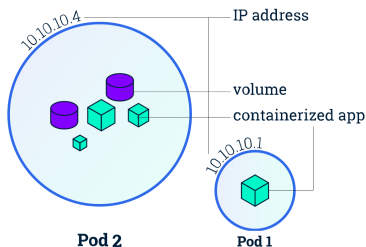


FIGURE – Les Pod dans Kubernetes

## 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

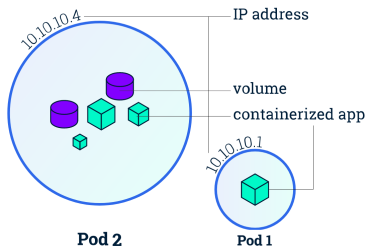


FIGURE – Les Pod dans Kubernetes

## 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

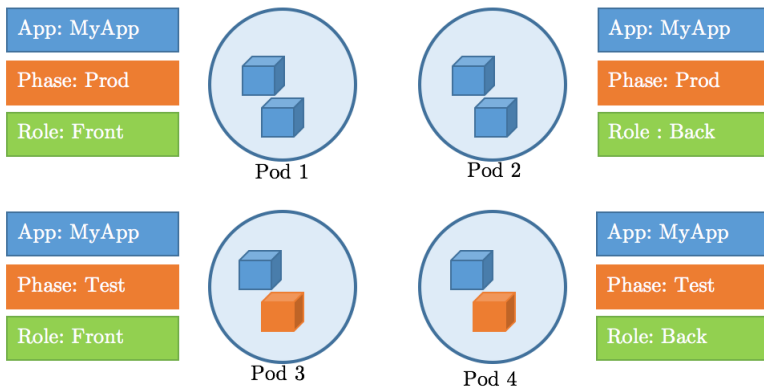


FIGURE – Exemple avec différents selectors

# Exemple d'utilisation

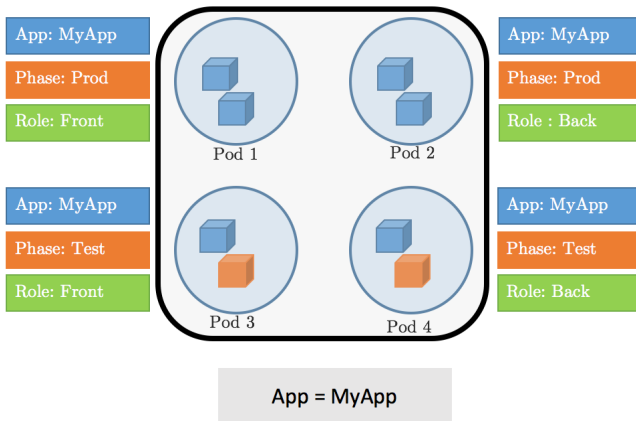


FIGURE – Exemple avec différents selectors

# Exemple d'utilisation

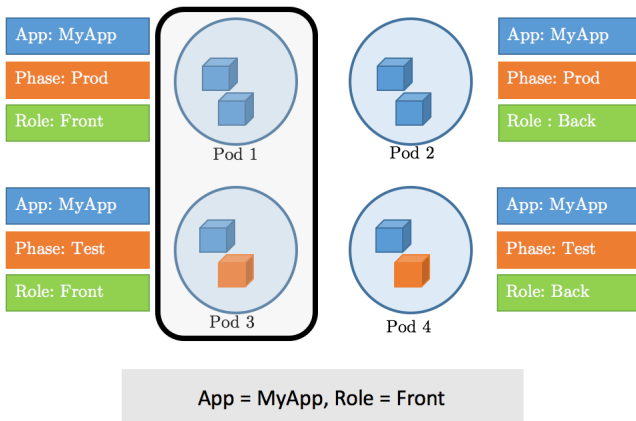


FIGURE – Exemple avec différents selectors

# Exemple d'utilisation

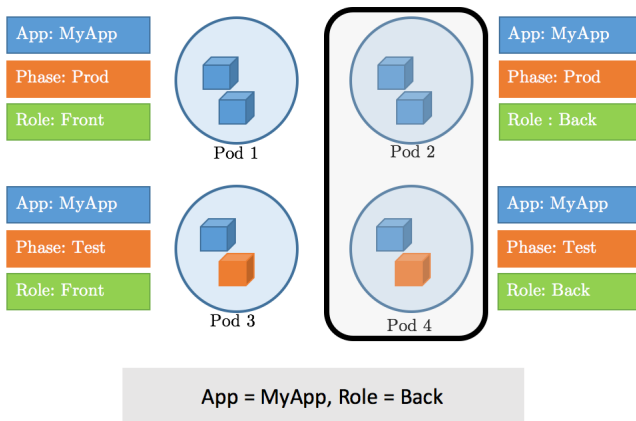


FIGURE – Exemple avec différents selectors

# Exemple d'utilisation

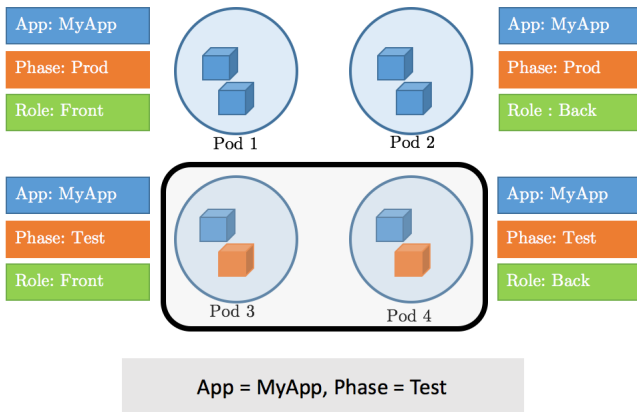


FIGURE – Exemple avec différents selectors

# 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

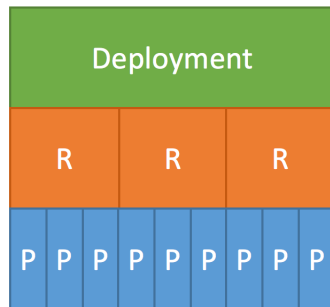


FIGURE – Relation entre Objet Deployment, ReplicaSet et Pod

# Exemple du Rolling-Update

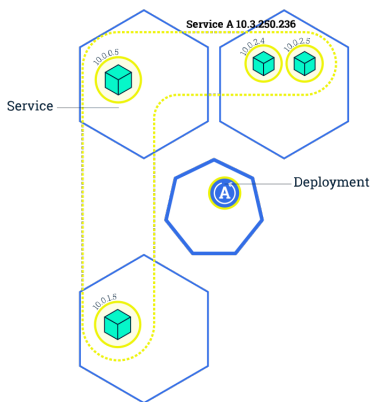


FIGURE – Exemple du Rolling Update tiré de la documentation Kubernetes

# Exemple du Rolling-Update

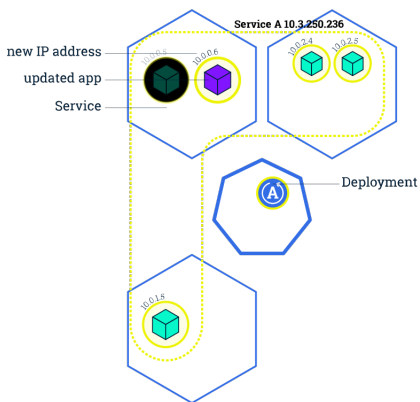


FIGURE – Exemple du Rolling Update tiré de la documentation Kubernetes

# Exemple du Rolling-Update

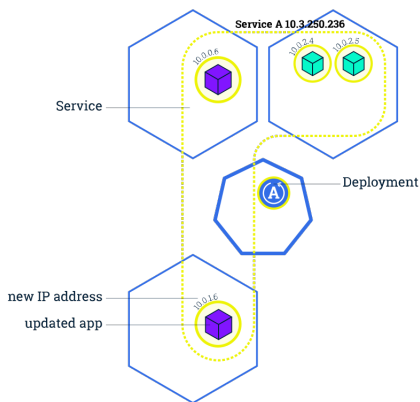


FIGURE – Exemple du Rolling Update tiré de la documentation Kubernetes

# Exemple du Rolling-Update

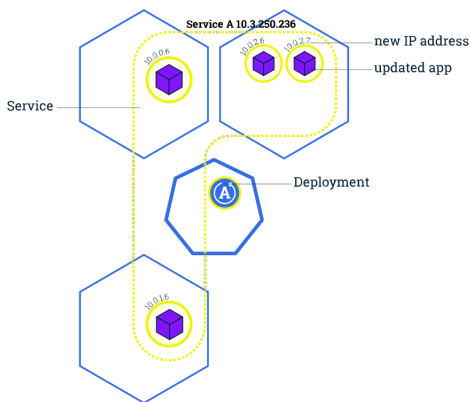


FIGURE – Exemple du Rolling Update tiré de la documentation Kubernetes

# Kubernetes Node i.e Worker Node

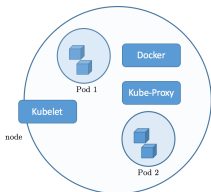


FIGURE – Exemple de node

## 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

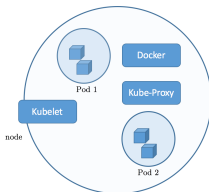


FIGURE – Exemple de node

## 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

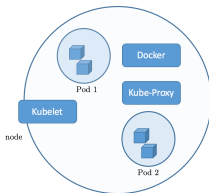


FIGURE – Exemple de node

## 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

The master node is responsible for the management of Kubernetes cluster. This is the entry point of all administrative tasks. The master node is the one taking care of orchestrating the worker nodes, where the actual services are running.

Let's dive into each of the components of the master node. API server

The API server is the entry points for all the REST commands used to control the cluster. It processes the REST requests, validates them, and executes the bound business logic. The result state has to be persisted somewhere, and that brings us to the next component of the master node. etcd storage

etcd is a simple, distributed, consistent key-value store. It's mainly used for shared configuration and service discovery. It provides a REST API for CRUD operations as well as an interface to register watchers on specific nodes, which enables a reliable way to notify the rest of the cluster about configuration changes.

An example of data stored by Kubernetes in etcd is jobs being scheduled, created and deployed, pod/service details and state, namespaces and replication information, etc. scheduler

The deployment of configured pods and services onto the nodes happens thanks to the scheduler component. The scheduler has the information regarding resources available on the members of the cluster, as well as the ones required for the configured service to run and hence is able to decide where to deploy a specific service.

controller-manager

Optionally you can run different kinds of controllers inside the master node.

controller-manager is a daemon embedding those. A controller uses apiserver to watch

the shared state of the cluster and makes corrective changes to the current state to

# Kubernetes Overview

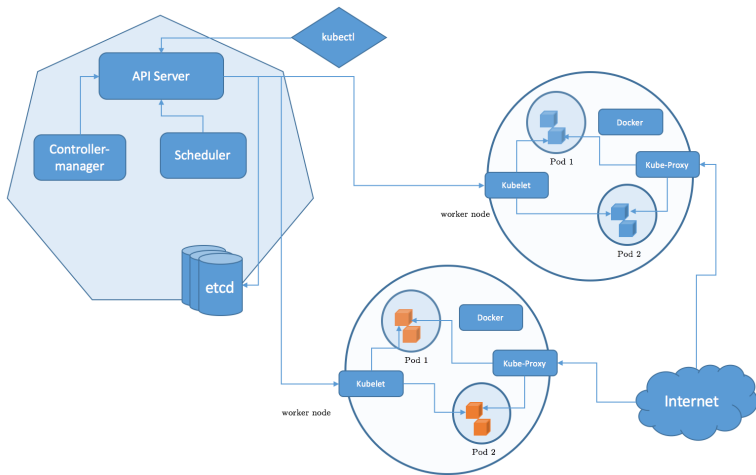


FIGURE – Overview

# Algo

# Conclusion