



## KUBERNETES



# Table des matières

- Introduction
  - Comprendre les containers
  - Rappel Docker & Containerisation
- L'architecture
  - Big Picture
  - Terminologie
  - Object Management Model
- Installation
  - Docker Desktop, K8S et WSL2
  - Kind
  - Dashboard
  - KubeConfig

# Table des matières

- Pods
  - Introduction
  - Création d'un pod
  - Les commandes
  - Conteneurs et pods
  - Scheduling
  - Les affinités
  - Noeud
  - Pods
  
- Documentation
  - Outils
  - Installation sous linux/Mac
  - Utilisation
  
- Déploiement
  - Overview
  - specifications
  - Test
  - Mise à jour

# Table des matières

- Namespaces
  - - Introduction
  - - Contextes
- Configuration des applications
  - - ConfigMap
  - 
  - Volumes
    - - définition

# Introduction

Kubernetes

Nom : Kubernetes = navigateur en Grec ancien (ou K8S)

Créateur : Google

Première version : 2015

Open-Source : donné au CNCF en 2014

Langage : Ecrit en Go/Goland

Repository :

<https://github.com/kubernetes/kubernetes>

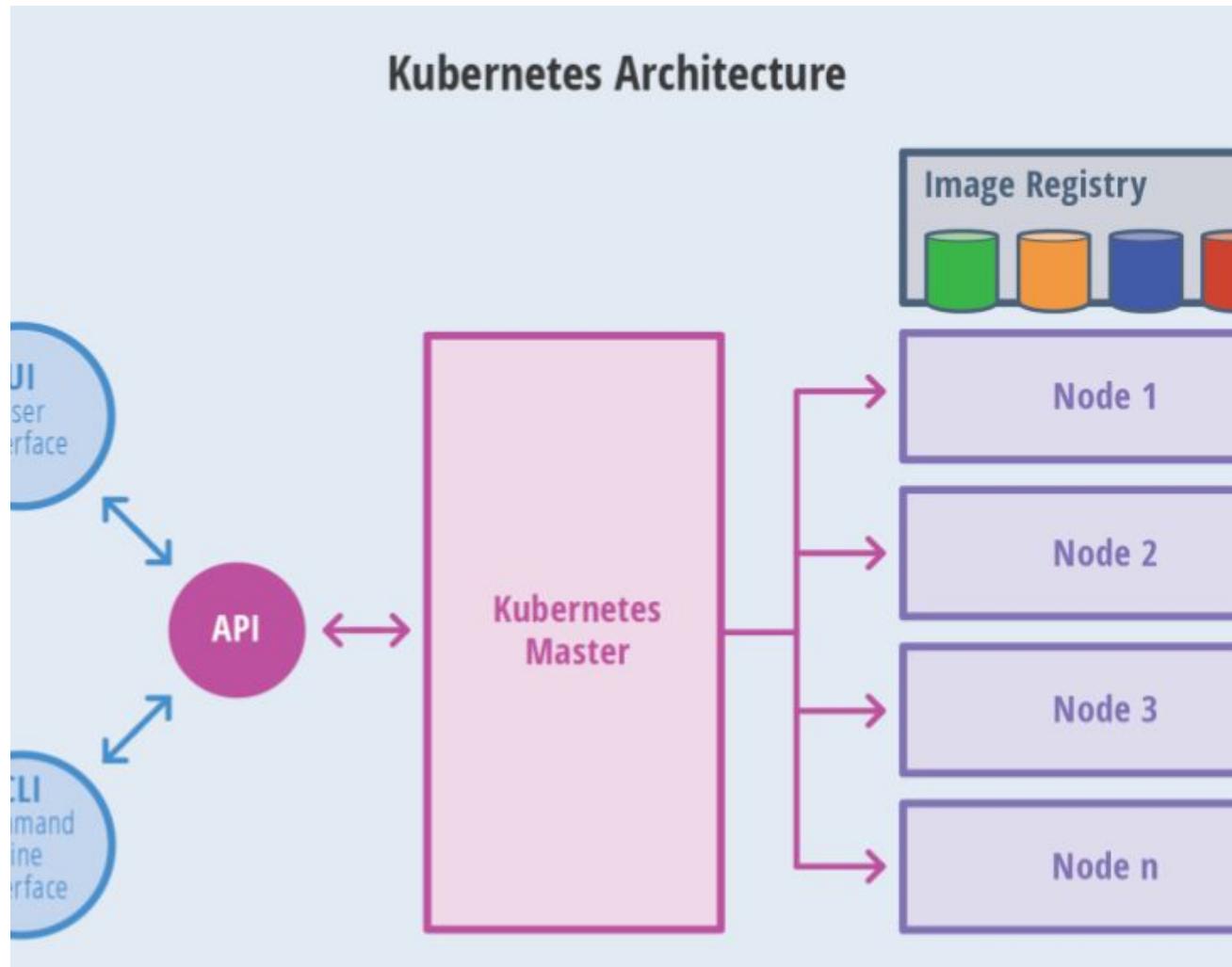
Interactive Tutorial :

<https://kubernetes.io/docs/tutorials/kubernetes-basics/>



Kubernetes est une plate-forme open-source extensible et portable pour la gestion de charges de travail (workloads) et de services conteneurisés. Elle favorise à la fois l'écriture de configuration déclarative (declarative configuration) et l'automatisation.

Son but est d'orchestrer les ressources entre containers : stockage, réseau, ...



# Container & Docker (rappel)

Kubernetes

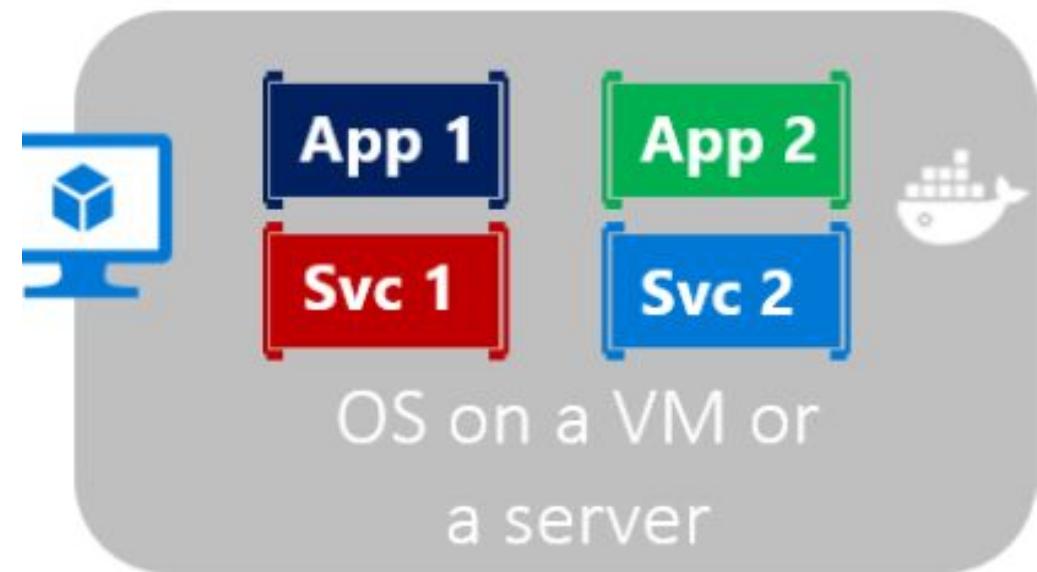
# Container

Container & Docker (rappel)

## Containerization

- Approche pour le développement de logiciel ou de services
- Les dépendances et la configuration sont « empaquetées » ensemble dans une Image
- Les « containers » agissent comme un standard de déploiement d'unité logicielle pouvant contenir différents codes et dépendances
- Les logiciels « containerisés » permettent aux développeurs de déployer sur différents environnements avec peu voire aucune modification.
- Un « container » isole les applications contenues de l'OS et des autres applications
- L'application « containerisée » s'exécute dans un Host exécutant par-dessus l'OS (Windows/Linux) avec un coût en termes de ressources moindre qu'une machine virtuelle.
- Facilite la « scalabilité » par l'ajout de nouveaux containers

## Docker Host

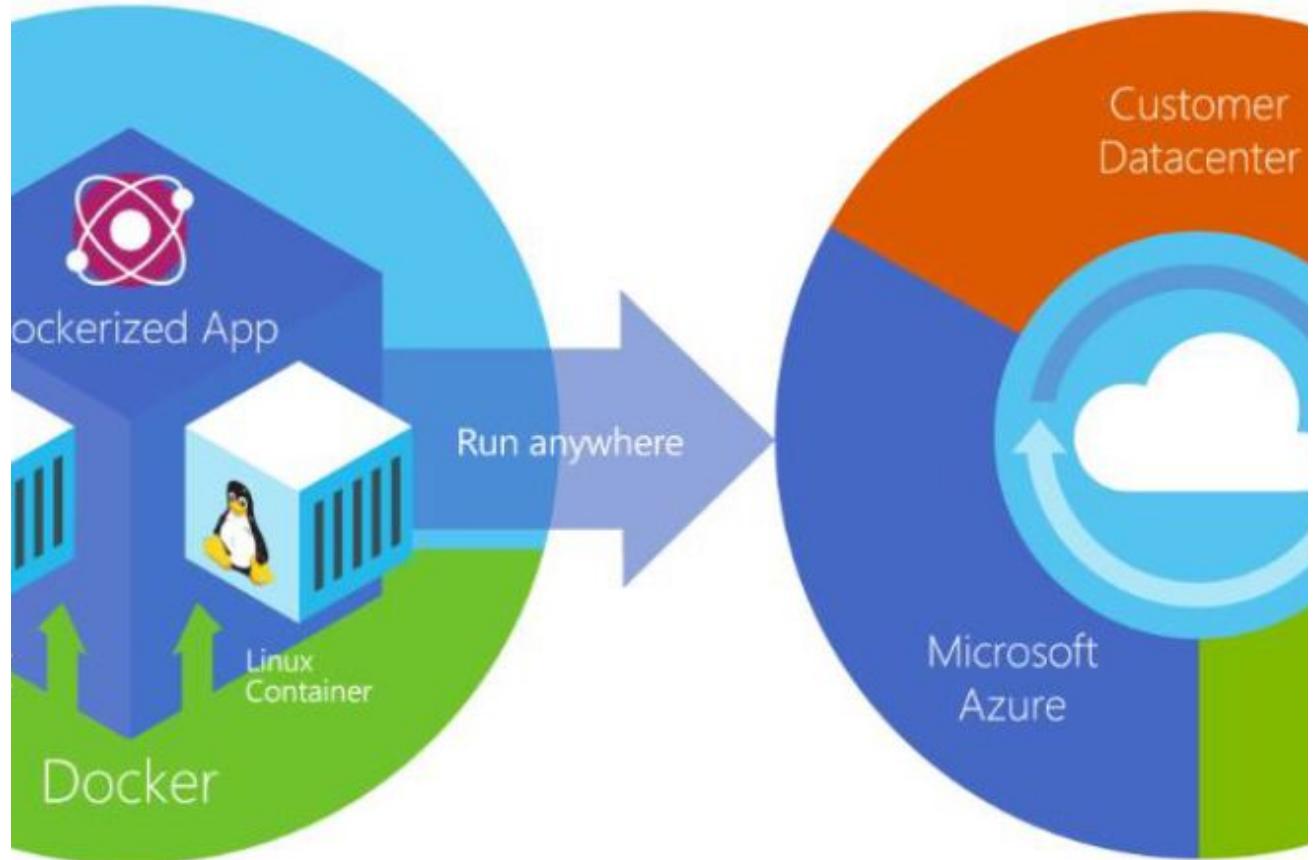


# Docker

Container & Docker (rappel)

Docker est un projet Open-Source pour automatiser le déploiement d'applications portables dans un container auto-suffisant qui peut s'exécuter aussi bien dans le cloud que dans un environnement local.

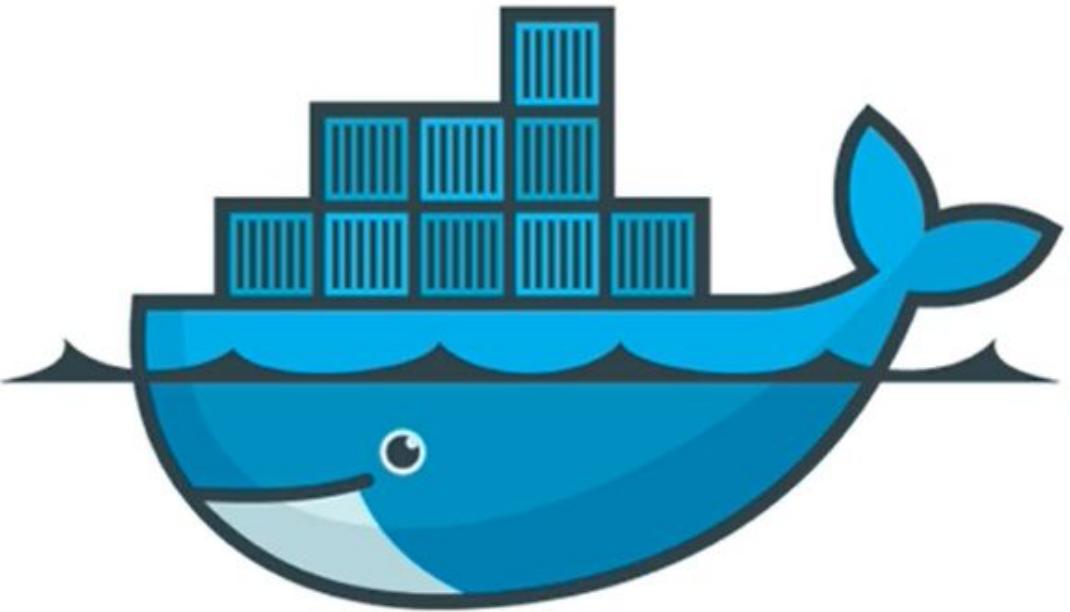
Mais Docker c'est aussi une entreprise qui promotionne, maintient, fait évoluer...cette technologie en travaillant en collaboration avec les fournisseurs de Cloud (AWS, Azure,...) et d'OS (Linux, Windows,...)



# Docker?

En quelques mots :

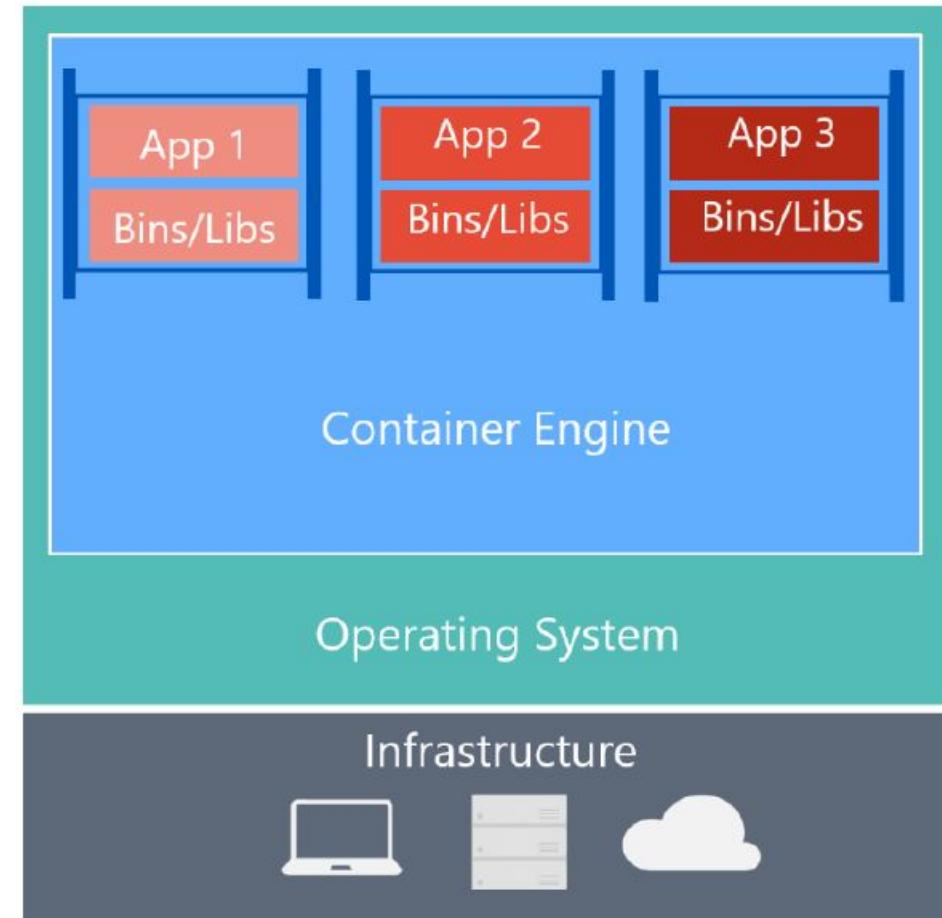
- Plateforme ouverte, sécurisée et légère
- Simplifie la construction, l'exécution et le déploiement des applications
- S'exécute nativement sur Linux ou Windows Server
- S'exécute sur les machines pour le développement sous Windows/Mac par l'intermédiaire de machines virtuelles ou WSL2



Il a la charge

- d'**Exécuter les containers**
- d'assurer l'**Isolation** des containers
- de gérer **les ressources** : mémoire et cpu

Grâce à lui nous pouvons donc , sans changer notre application située dans le container, attribuer plus de ressources lorsque nous passons en prod, que lorsque nous sommes en test



# Docker Client

- Son but est de nous connecter au Docker Engine et lui transmettre des commandes, des actions à effectuer
- Il est installé sur la même machine que le Docker engine ou sur un poste de travail

Deux clients existent : ligne de commande et GUI

Ligne de commande	GUI
  	 

## Docker Image :

Template en lecture-seule permettant de construire un container.  
Contient les librairies et les exécutables nécessaires pour exécuter une application.

**Exemple :** WindowsCore avec IIS + .net Core + application

Une image seule n'a pas de sens car finalement elle ne contient que des définitions d'environnement et de la configuration.

Les images sont distribuées par des registry (serveur d'image)

Par défaut, le serveur d'image est : hub.docker.com

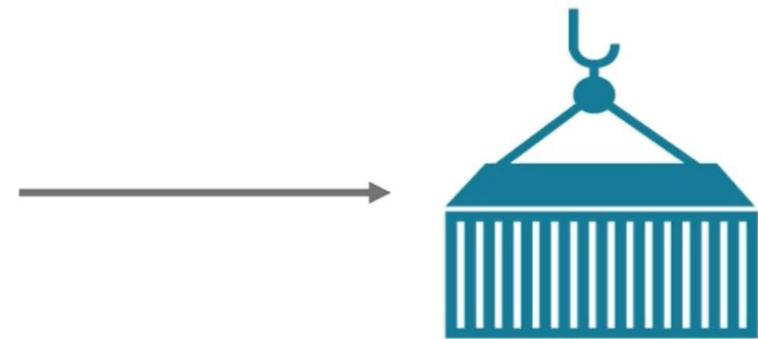
## Docker Container

Un container est un environnement isolé, sécurisé créé grâce à une image. C'est le container qui est l'unité d'exécution des environnements (OS, Framework,...) et du code des applications.

Il peut être démarré, stoppé, supprimé, déplacé,...facilement et rapidement.



**Docker Image**



**Docker Container**

# Container & Docker (rappel)

## Avantages

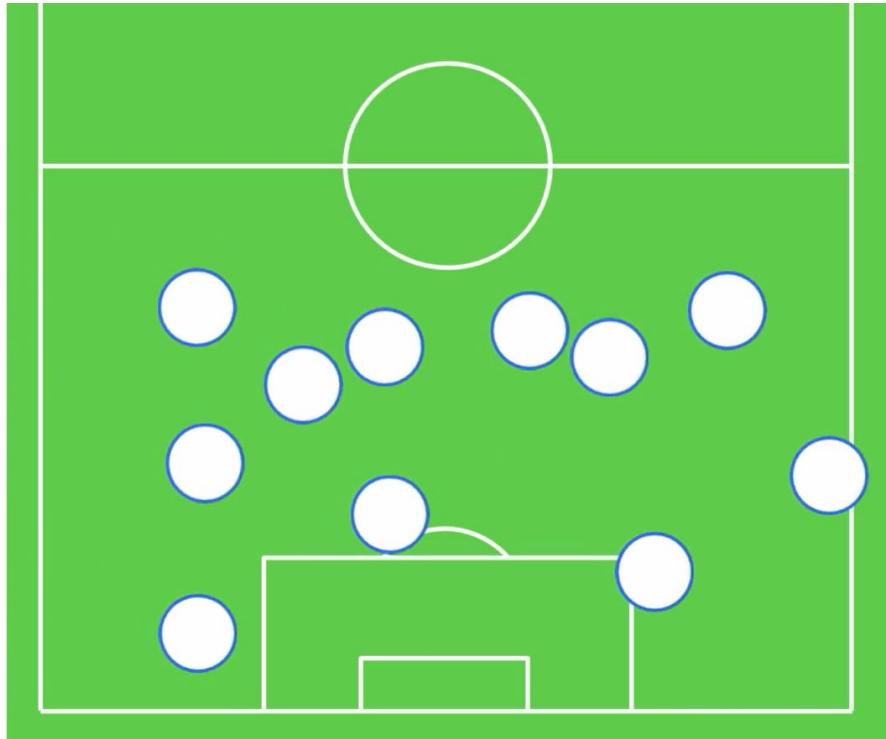
- Application portable puisqu'indépendante du hardware
- Format standard pour la distribution des applications
- Déploiement « facile »
- Facilitation et automatisation des tests d'intégration
- ...

# Architecture

Kubernetes

K8s est un orchestrateur qui se charge d'organiser, diriger,... les applications.

## Sans Coach/Manager



Chaque application connaît son propre « rôle » et son but. Mais pas de cohésion d'équipe pour arriver au but commun.

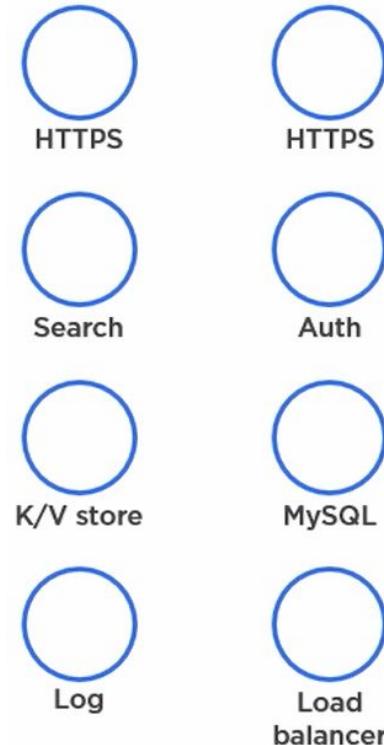
## Avec Coach/Manager



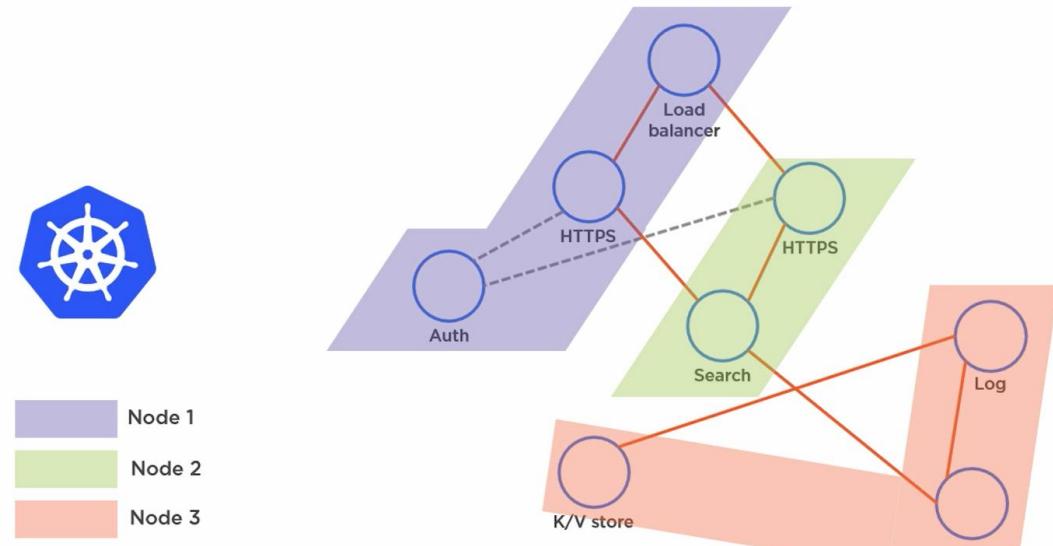
Désormais chaque rôle est connu et le manager organise la communication, l'interaction entre chaque app afin de travailler ensemble de manière efficiente

Dans K8S, chaque « joueur » devient un Pods qui « héberge » une app avec ses caractéristiques et fonctionnalités.  
Le tout organisé sous forme de nœuds(Nodes) et un groupe de Nœud = 1 cluster

## Sans K8s



## Avec K8s



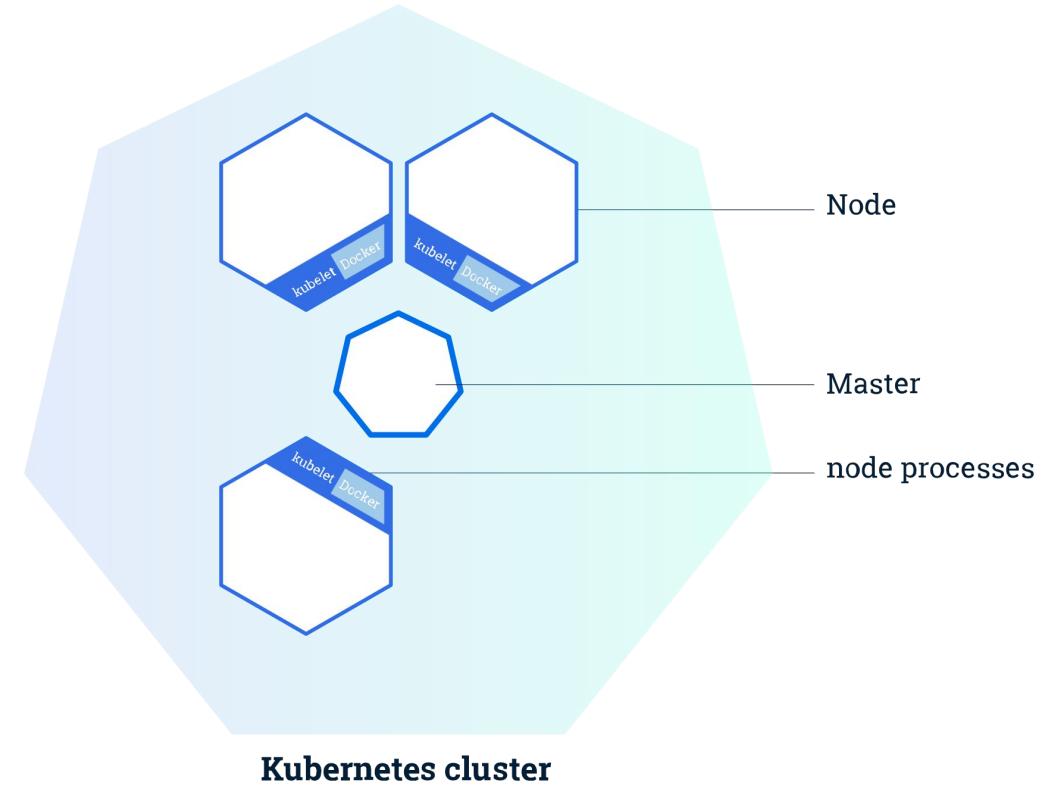
# Terminologie

## Cluster Kubernetes

Un cluster est un ensemble de nœuds Windows ou Linux.

Il est constitué de deux types de ressources:

- Le maître coordonne le cluster.
- Les nœuds sont les serveurs qui exécutent des applications.

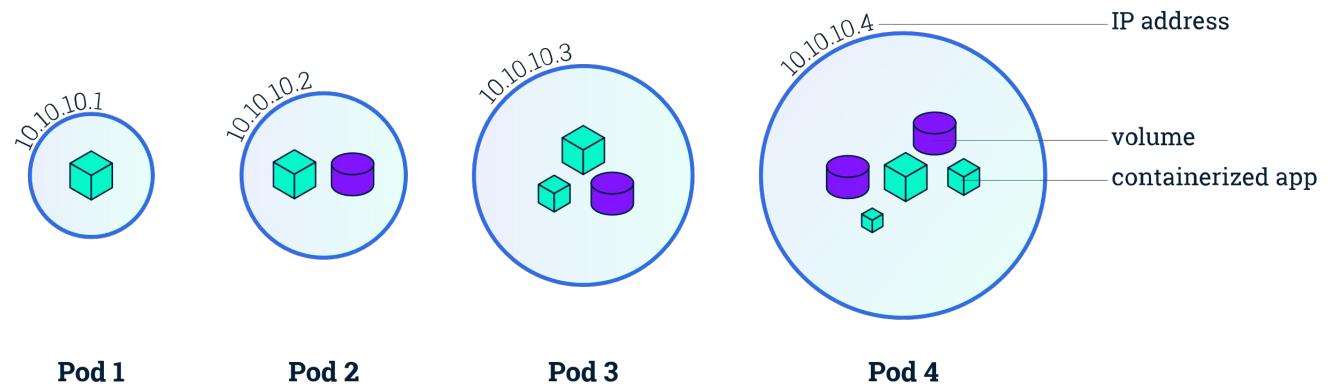


# Terminologie

## Pod

Les pods sont les plus petites unités informatiques déployables que vous pouvez créer et gérer dans Kubernetes.

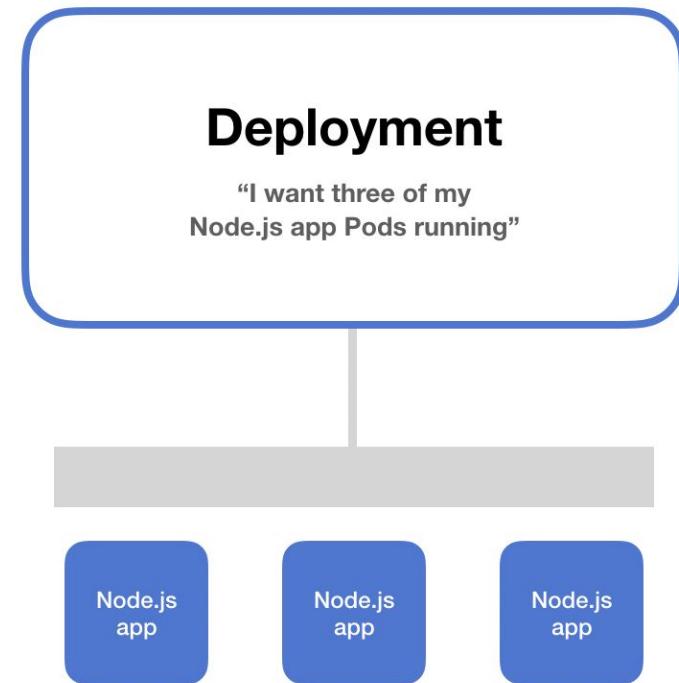
Un pod est un groupe d'un ou plusieurs conteneurs, avec un stockage et des ressources réseau partagés, et une spécification sur la manière d'exécuter les conteneurs.



# Terminologie

## Deployment

Objet permet de gérer un Pod de manière unique ou de le répliquer ou de le mettre à jour

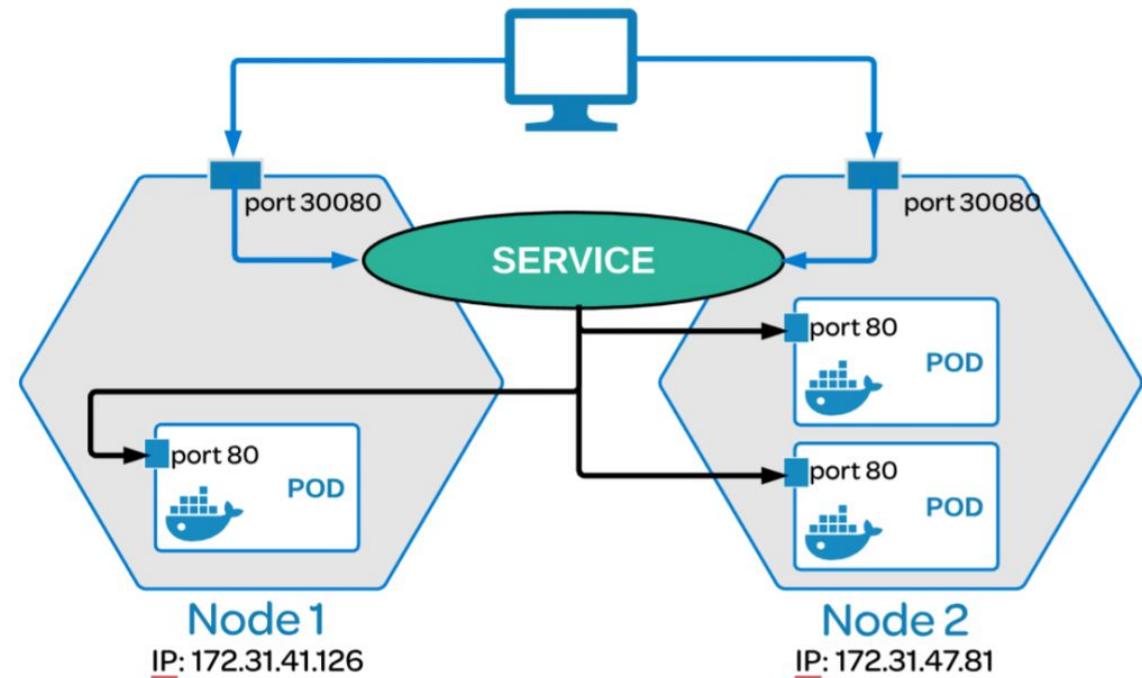


# Terminologie

## Services

Est une manière abstraite d'exposer une application exécutée sur un ensemble de pods en tant que service réseau.

Il se charge donc d'exposer les ports d'un Pod à l'intérieur ou à l'extérieur de celui-ci





**Master**



Architecture

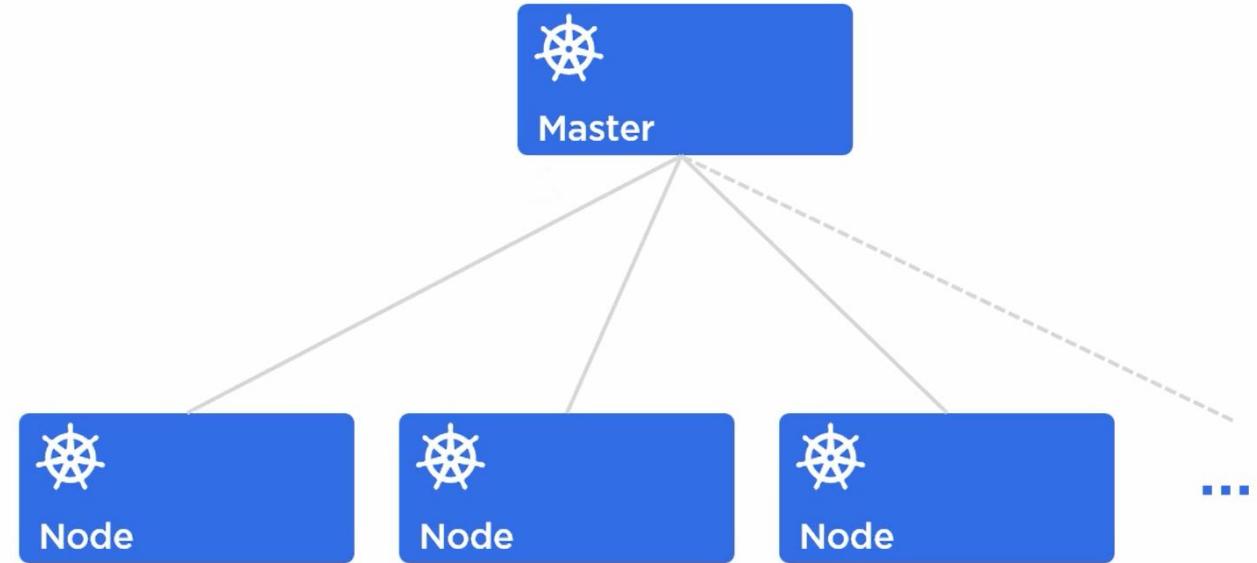
## Master (Kubernetes Control Plane)

Il contrôle l'ensemble du cluster afin de permettre la gestion du cycle de vie des autres nœuds.

Ses principaux composants sont :

- Kube-Apiserver
- Etcd (Cluster Store)
- Kube-Controller-Manager
- Kube-Scheduler

Il n'a pas vocation à contenir nos Pods car ceux-ci seront répartis, hébergés dans les nœuds enfants.



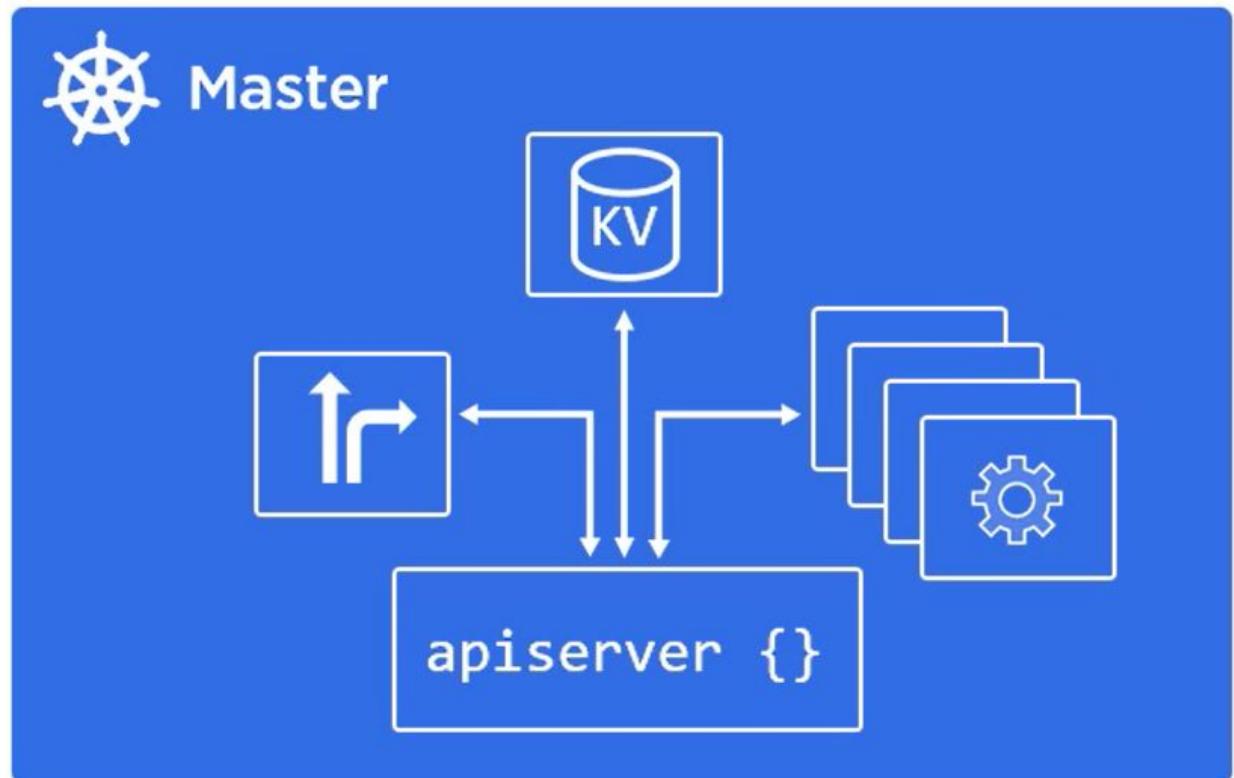
## Kube-apiserver

Il s'agit d'un Api RESTful permettant de contrôler et gérer le cluster.

Pour cela nous définirons des manifests.

Ces fichiers *manifest* peuvent être définit en yaml ou json et permettent , entre autres, de créer et organiser les ressources...

```
---  
apiVersion: v1  
kind: Pod  
metadata:  
  name: rss-site  
  labels:  
    app: web  
spec:  
  containers:  
    - name: front-end  
      image: nginx  
      ports:  
        - containerPort: 80  
    - name: rss-reader  
      image: nickchase/rss-php-nginx:v1  
      ports:  
        - containerPort: 88
```

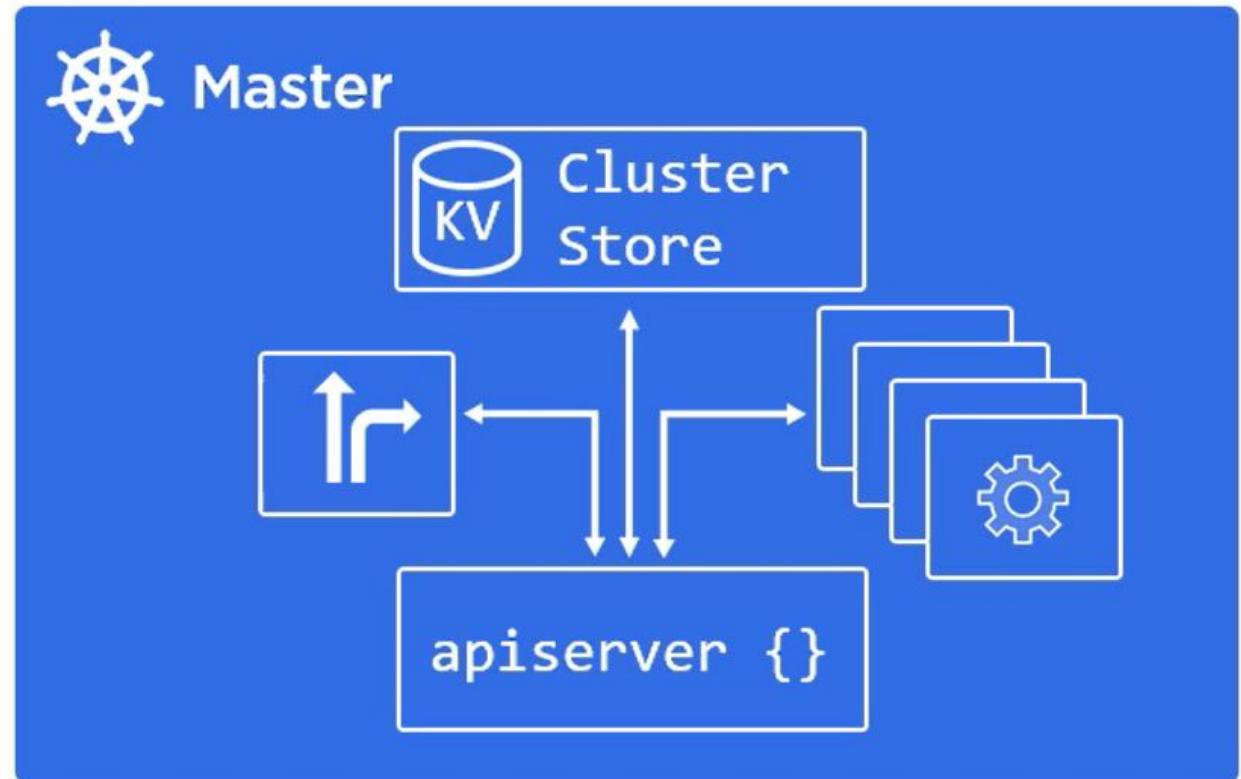


## etcd (Cluster Store)

Base de données distribuée NoSQL de type Key-Value assurant la consistance des données.

Il s'agit donc de notre stockage permanent de notre configuration du cluster et de son état.

Il est donc très **très** important : Pas de Cluster Store Pas de Cluster.



## Kube-Controller-Manager

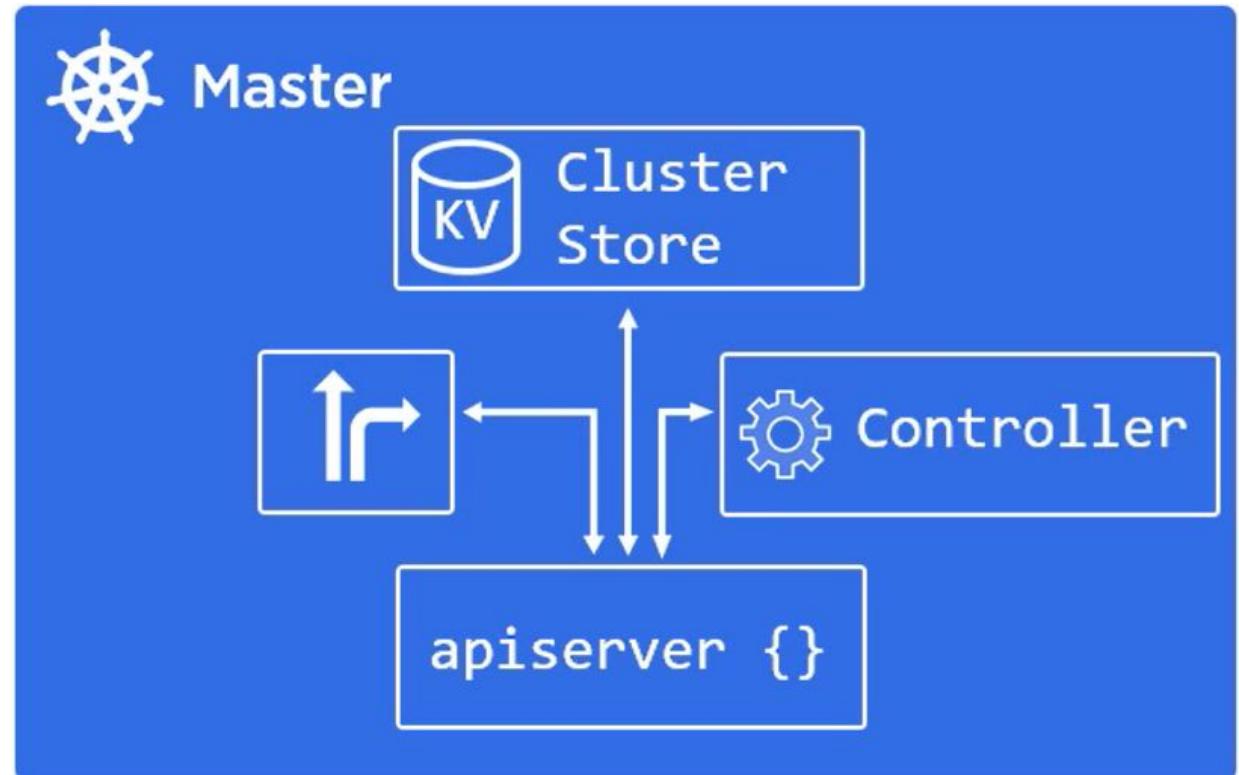
c'est un démon qui intègre les boucles de contrôle principales fournies avec Kubernetes.

Il surveille l'état du cluster partagé par l'api-server et effectue les modifications nécessaires aux changements d'états demandés.

Il permet de contrôler :

- les nœuds
- Les réplicas
- Les end-points
- La gestion des comptes d'accès au niveau du système
- ...

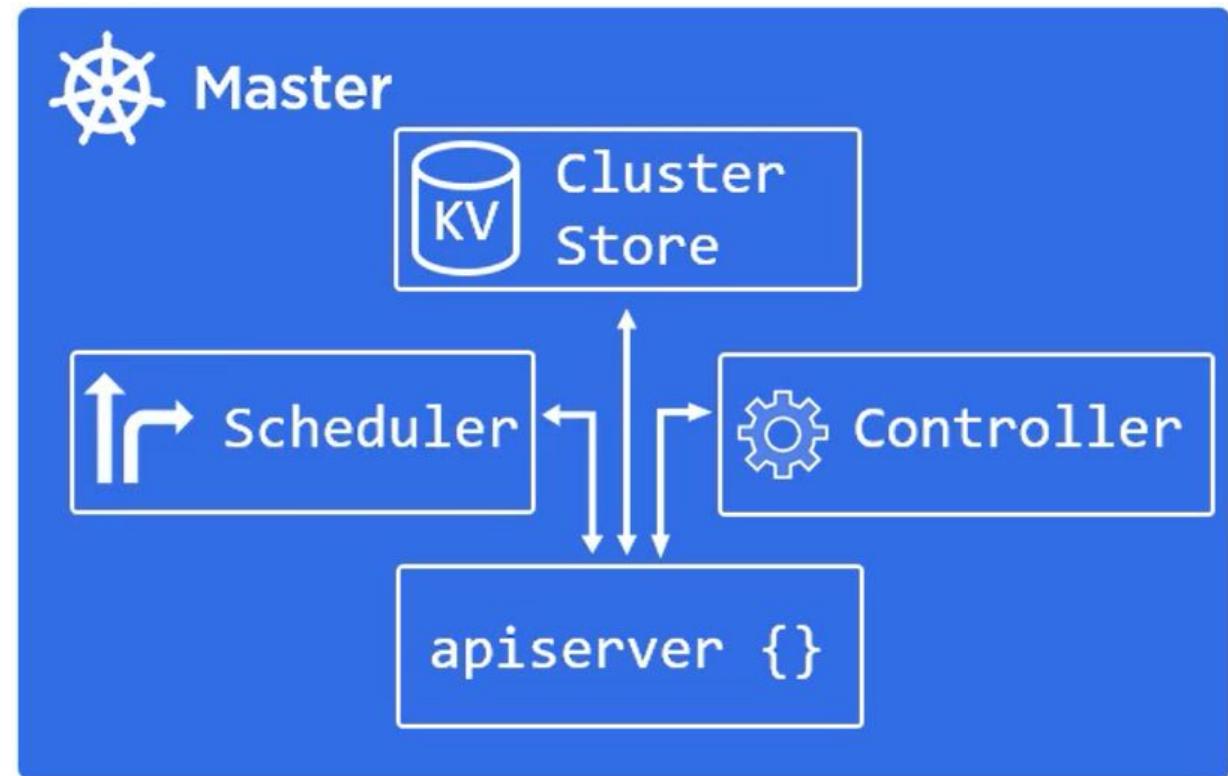
Toute la gestion du cluster se fait via ce composant.



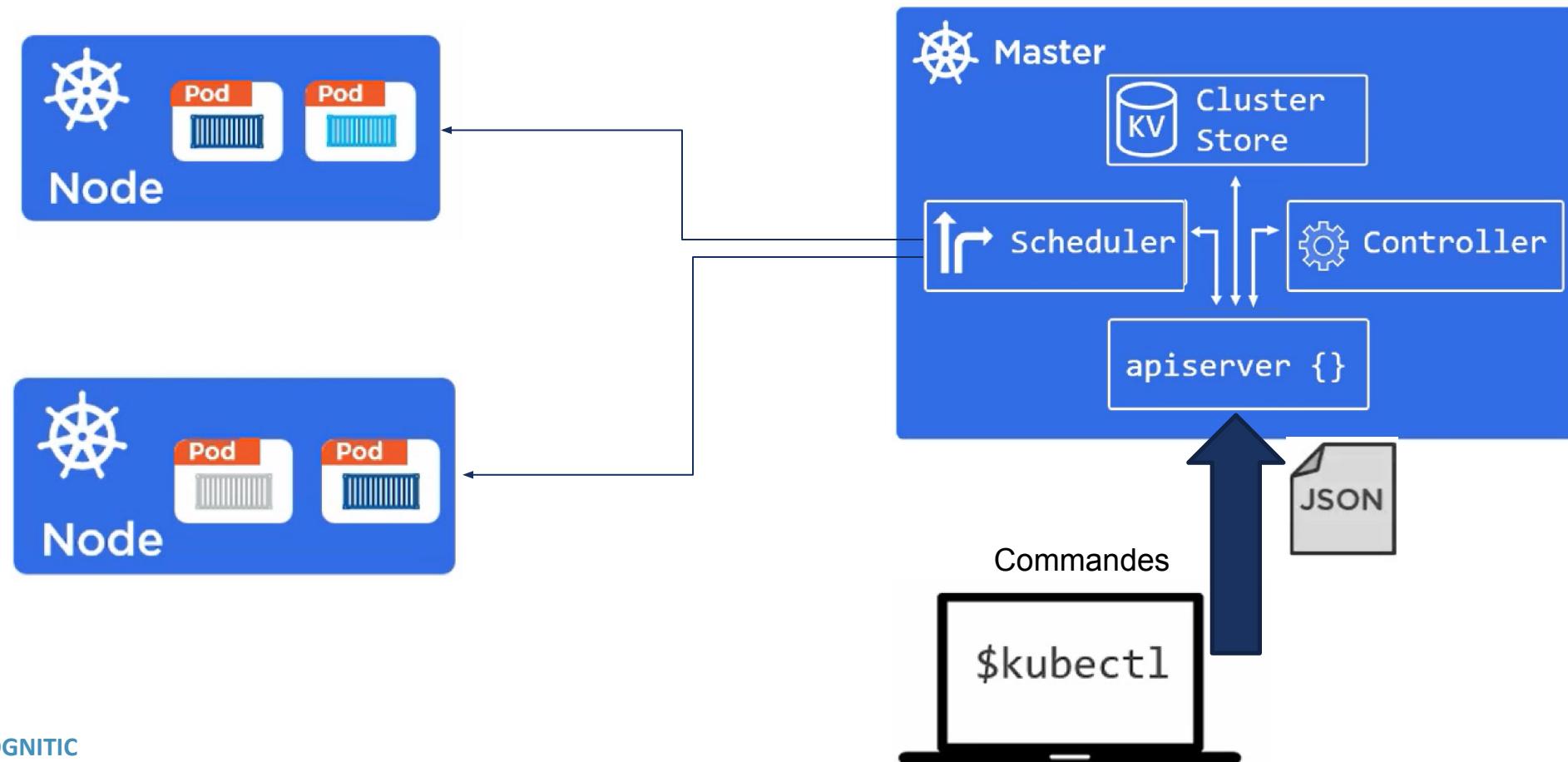
## Kube-Scheduler

Il audit les clusters afin de vérifier si de nouveaux Pods apparaissent et réparti les charges de travail sur les nodes managés suivant :

- Les contraintes
- Les affinités
- Les ressources
- ...



# Architecture





Nodes



Architecture

# Nodes : Kubernetes Workers

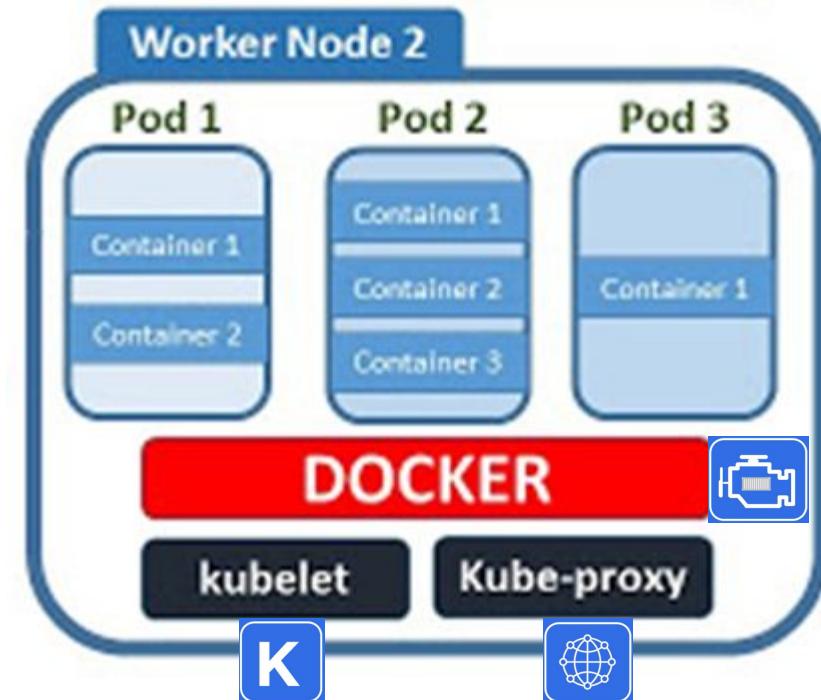
Un nœud est une machine de travail dans Kubernetes, connue auparavant sous le nom de *Minions*.

Un nœud peut être une machine virtuelle ou une machine physique, selon le cluster.

Chaque nœud contient les services nécessaires à l'exécution de *Pods* et est géré par les composants du master.

Un nœud contient les services suivants :

- 1 Kubectl
- 1 Container Runtime
- 1 Kube-Proxy



# Nodes : Kubernetes Workers



**Kublet** est l'agent principal de Kubernetes, sans lui pas de nœud.

Ses responsabilités sont :

- Enregistrer le nœud dans le cluster
- Ecouter les demandes faites via le kube-apiserver
- Instancier les pods
- Informer le master du changement d'état des pods, du nœud,...

Exemple :

Si un Pods crash, le Kublet informe le master qui prendra la décision de le relancer ou pas.

- Exposer un endpoint sur le port 10255 pour permettre d'inspecter le nœud via les commandes `/spec`, `/healtz`, `/pods`

# Nodes : Kubernetes Workers



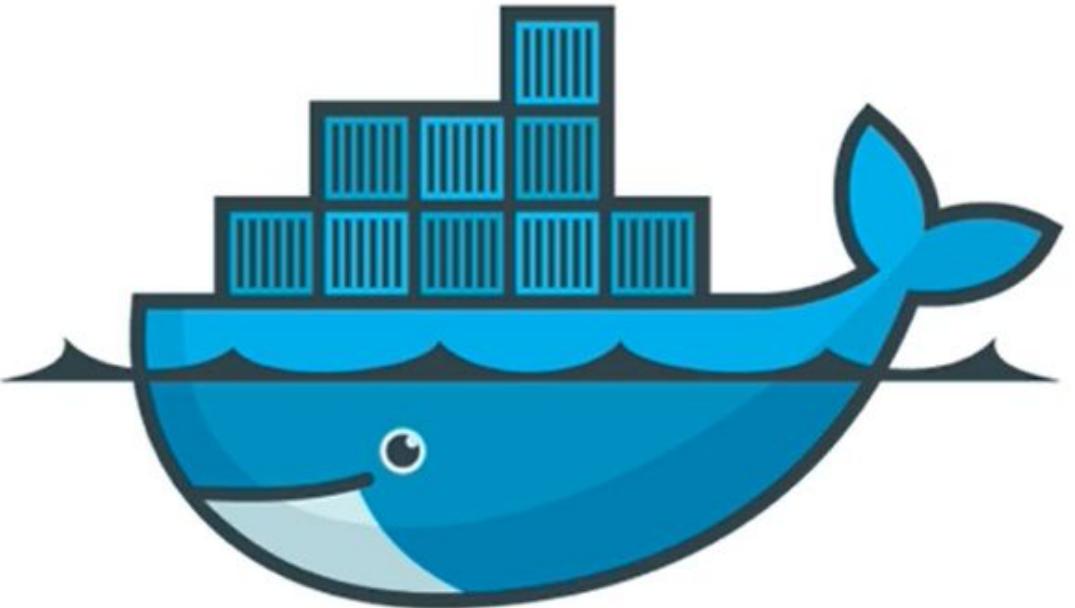
Container Engine

C'est notre gestionnaire de container (par défaut Docker).

Ses responsabilités sont :

- Faire un pull des images
- Lancer/arrêter les containers
- ...

(Cfr le rappel de docker)



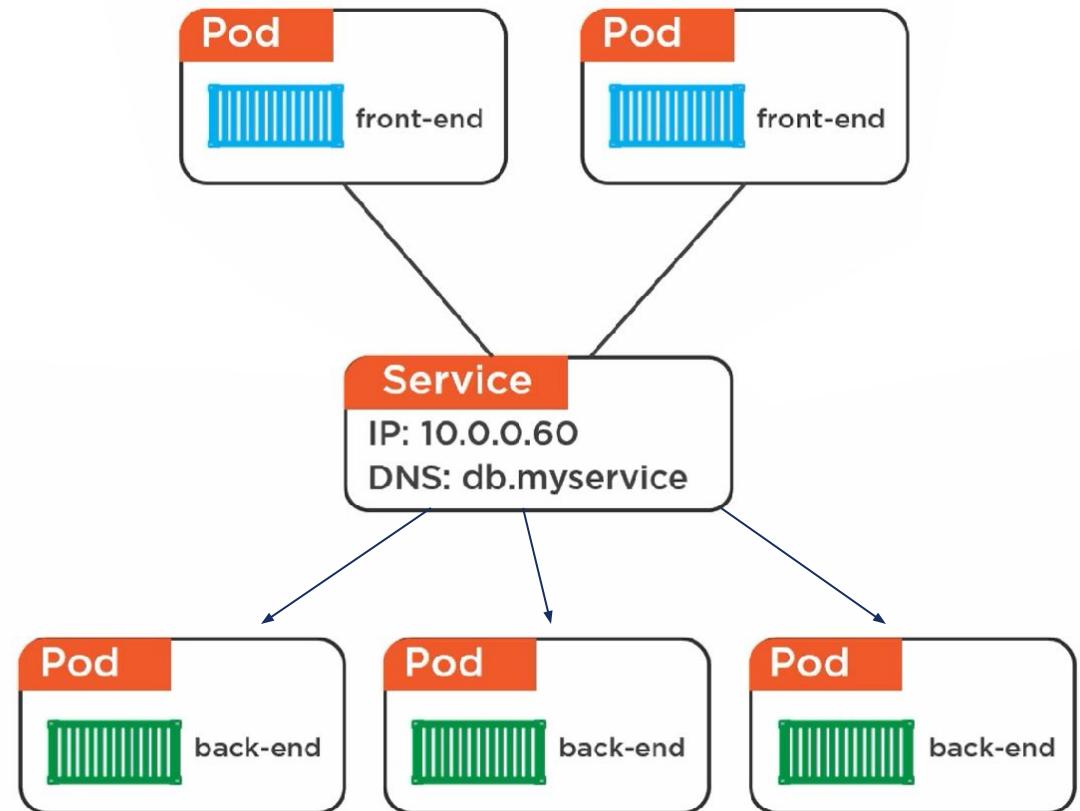
# Nodes : Kubernetes Workers



Kube-Proxy est la partie réseau du nœud .

Ses responsabilités sont :

- Attribuer une ip *unique* pour ses Pods
- Effectuer la distribution des charges entre les pods via un service



# Object Management Model

Kubernetes

# Object Management

Avant de nous lancer dans les charges de travail Kubernetes (pods, contrôleurs, etc.), il est important de comprendre le modèle déclaratif de Kubernetes.

## Pour faire simple :

Nous utilisons une spécification (Yaml ou Json) décrivant l'état désiré de nos nœuds. K8S va donc continuellement observer les nœuds afin de coller au plus près de cet état.

## Exemple :

Je demande 3 pods avec nginx.

Au départ, j'ai 3 pods dans 3 nœuds.

Un nœud tombe!

K8s va donc « réinstancier » un pod nginx sur 1 des 2 nœuds restant afin de rester dans l'état souhaité.



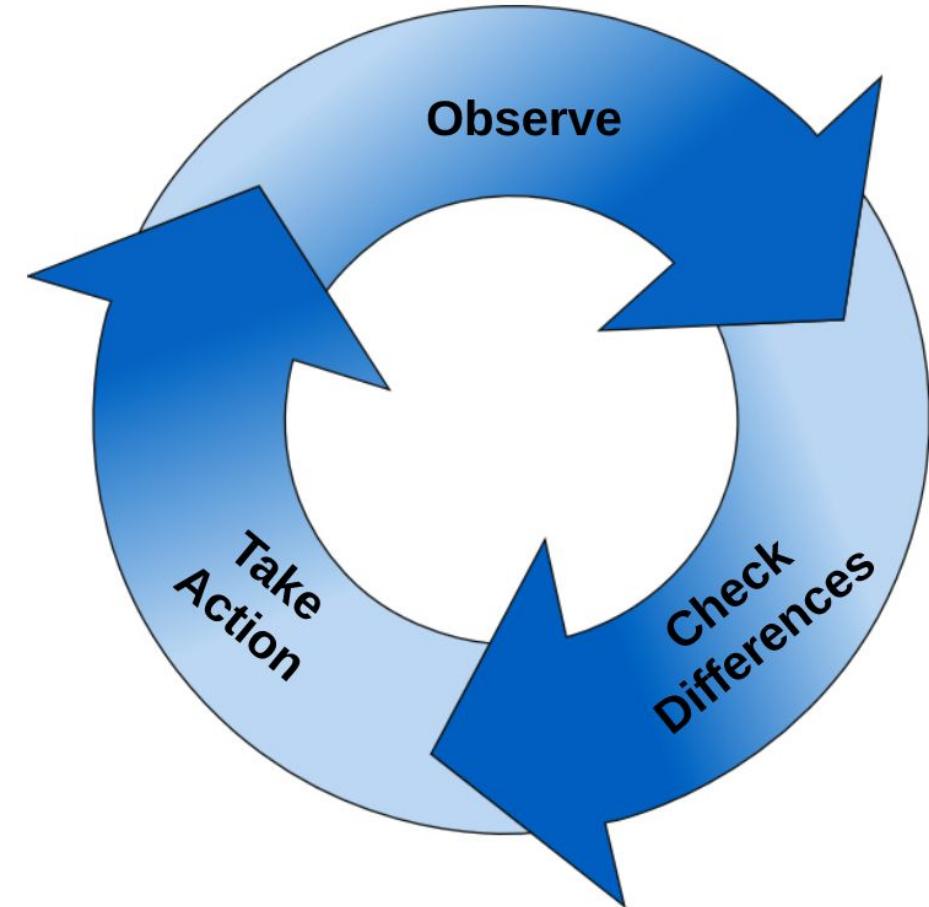
# Object Management

Le mécanisme qui permet à K8S de tendre vers ce « desired state » : le Control Loop.

- Observe : Quel est le « desired state » demandé?
- Check : Quels sont les états actuels des objets et quels sont les différences avec l'état souhaité?
- Take Action : Change l'état actuel vers l'état désiré
- Repeat : on relance l'observation.

K8s a plusieurs « control loop » qui s'exécutent en simultané dans :

- Le controller de déploiement : vérifie si de nouveau déploiement sont demandés
- Le controller de Réplicaset : vérifie si de nouveaux réplicas sont demandés et instancie les pods nécessaires
- ...



# INSTALLATION

Kubernetes

# Installation

Il existe plusieurs types d'installation :

- Installation Locale - Orienté développeur
- Installation Hébergée (On-Premise) – Orienté Production
- Installation Cloud (Cé en main) (<https://kubernetes.io/fr/docs/setup/pick-right-solution/#solutions-cl%C3%A9s-en-main>)

Kubernetes peut fonctionner sur des plateformes variées: sur votre PC portable, sur des VMs d'un fournisseur de cloud, ou un rack de serveurs bare-metal. L'effort demandé pour configurer un cluster varie de l'exécution d'une simple commande à la création de votre propre cluster personnalisé

# Solutions Locales

- Minikube est une méthode pour créer un cluster Kubernetes local à noeud unique pour le développement et le test.
- Docker Desktop est une application facile à installer pour votre environnement Mac ou Windows qui vous permet de commencer à coder et déployer votre code dans des conteneurs en quelques minutes sur un nœud unique Kubernetes.
- Minishift installe la version communautaire de la plate-forme d'entreprise OpenShift de Kubernetes pour le développement local et les tests. Il offre notamment une VM tout-en-un (minishift start) pour Windows, macOS et Linux.
- MicroK8s fournit une commande unique d'installation de la dernière version de Kubernetes sur une machine locale pour le développement et les tests. L'installation est rapide (~30 sec)
- Kubeadm-dind est un cluster Kubernetes multi-nœuds (tandis que minikube est un nœud unique) qui ne nécessite qu'un docker-engine.
- Ubuntu sur LXD supporte un déploiement de 9 instances sur votre machine locale

# Solutions Hébergées

Lorsque vous êtes prêts à augmenter le nombre de machines et souhaitez bénéficier de la haute disponibilité, une solution hébergée est la plus simple à déployer et à maintenir.

Il en existe énormément et le but ce cours n'est pas de les comparer mais vous avez notamment:

- [Amazon Elastic Container Service for Kubernetes](#) offre un service managé de Kubernetes.
- [Azure Kubernetes Service](#) offre des clusters Kubernetes managés.
- [DigitalOcean Kubernetes](#) offre un service managé de Kubernetes.
- [Google Kubernetes Engine](#) offre des clusters Kubernetes managés
- ...

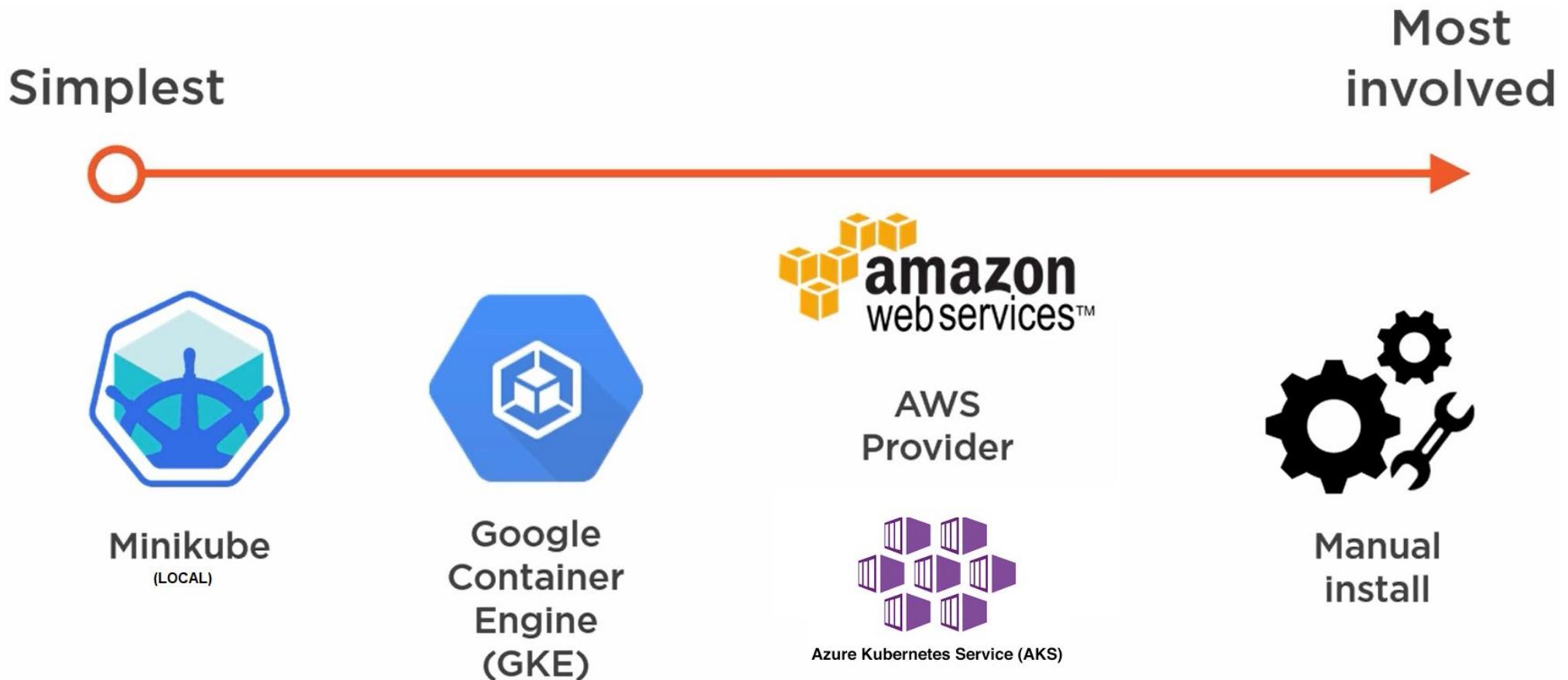
# Solutions Cloud (Clé en main)

Ces solutions vous permettent de créer des clusters Kubernetes sur une gamme de fournisseurs de Cloud IaaS avec seulement quelques commandes. Ces solutions sont activement développées et bénéficient du soutien actif de la communauté.

En voici quelques-unes :

- Azure
- AWS
- Google Computed Engine
- Docker Enterprise
- Kublr
- Oracle Container Engine for K8s
- VMWare Cloud PKS

# Installation





**Windows**



**Installation**

# Windows

**!!Attention!!! K8s évolue très très vite ce qui fait qu'il peut y avoir des différences dans le processus d'installation**

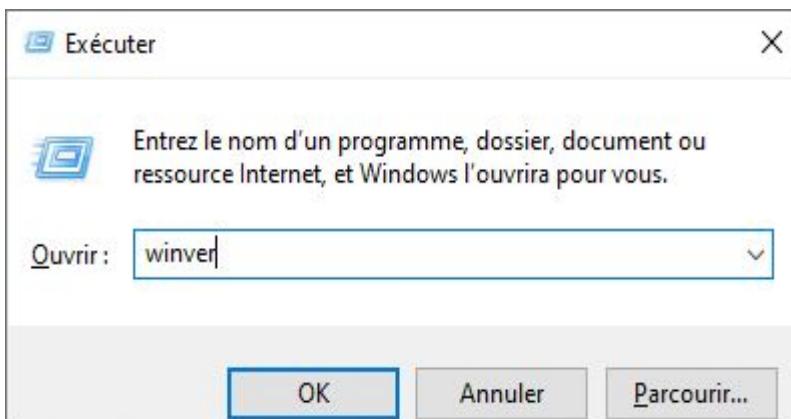
## Prérequis :

- OS : Windows 10
  - X64 : version 1903 min avec le build 18362 min
  - ARM64 : version 2004 min avec le build 19041 min
- WSL2 Activé
- 1 distro WSL2 installée (exemple : UBUNTU)
- Docker Desktop for Windows 2.2.0.4 minimum

# Windows

## ■ Vérifier la version de Windows

Vous pouvez vérifier votre version de Windows en appuyant simultanément sur la touche **Windows + r** et ensuite tapez **winver**



*Si votre machine ne propose pas la bonne version □*

<https://support.microsoft.com/fr-fr/windows/obtenir-la-mise-%C3%A0-jour-windows-10-d-octobre-2020-7d20e88c-0568-483a-37bc-c3885390d212>

# Activer WSL2 (Windows Sous-système Linux)

En 2016, Microsoft ouvrait la possibilité d'exploiter des distributions Linux sous Windows, Ubuntu en tête. C'était alors sous forme d'un accès limité à Bash via un terminal spécifique, reposant sur une brique nommée sous-système Linux (WSL).

La version 2 permet d'exploiter un environnement linux complet virtualisé sous Windows alors que la version 1 était beaucoup plus limitée.

## Comparaison des fonctionnalités

Fonctionnalité	WSL 1	WSL 2
Intégration entre Windows et Linux	✓	✓
Temps de démarrage courts	✓	✓
Faible encombrement des ressources	✓	✓
Machine virtuelle managée	✗	✓
Noyau Linux complet	✗	✓
Compatibilité complète des appels système	✗	✓
S'exécute avec VMware et VirtualBox	✓	✗
Performances dans les systèmes de fichiers du système d'exploitation	✓	✗

# Activer WSL2

Installer WSL2 sous Windows 10 :

1. Activer WSL2

Dans une fenêtre PowerShell en mode Administrateur , tapez la commande suivante :

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

2. Activer la fonctionnalité « Virtual Machine Platform »

Dans cette même fenêtre PowerShell

Version 2004

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

Version 1903-1909

```
Enable-WindowsOptionalFeature -Online -FeatureName VirtualMachinePlatform -NoRestart
```

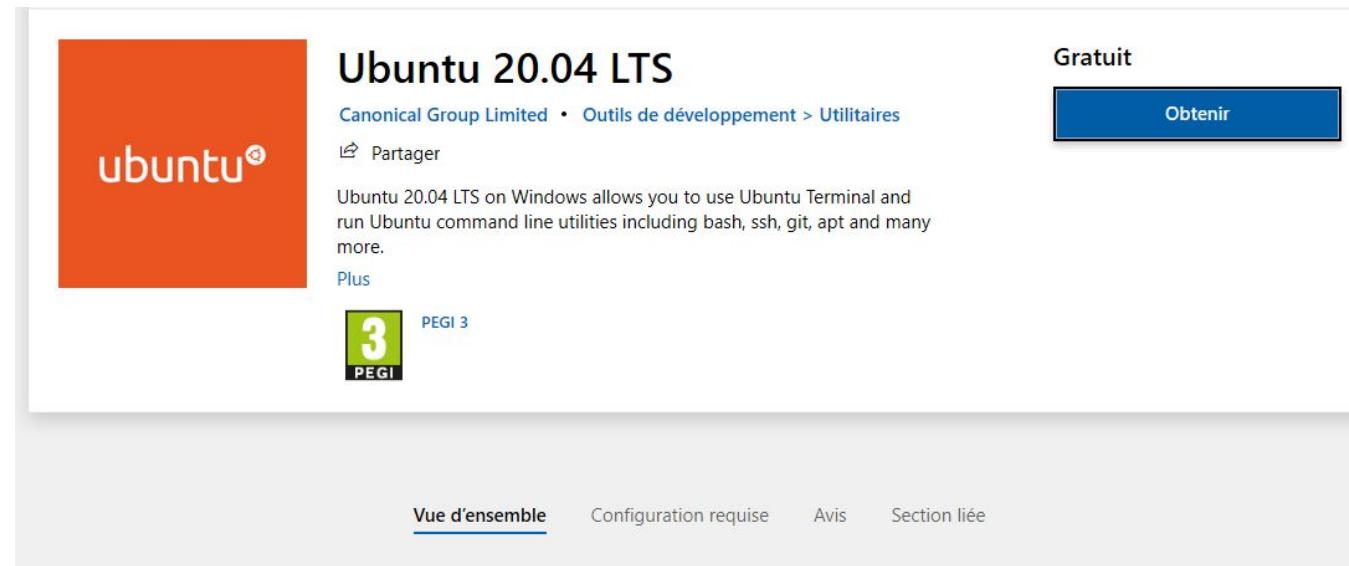
3. Définir WSL2 comme version par défaut

```
wsl --set-default-version 2
```

# Activer WSL2

## 4. Installer une distro Linux

Ouvrez le Microsoft store  et téléchargez, installez Ubuntu

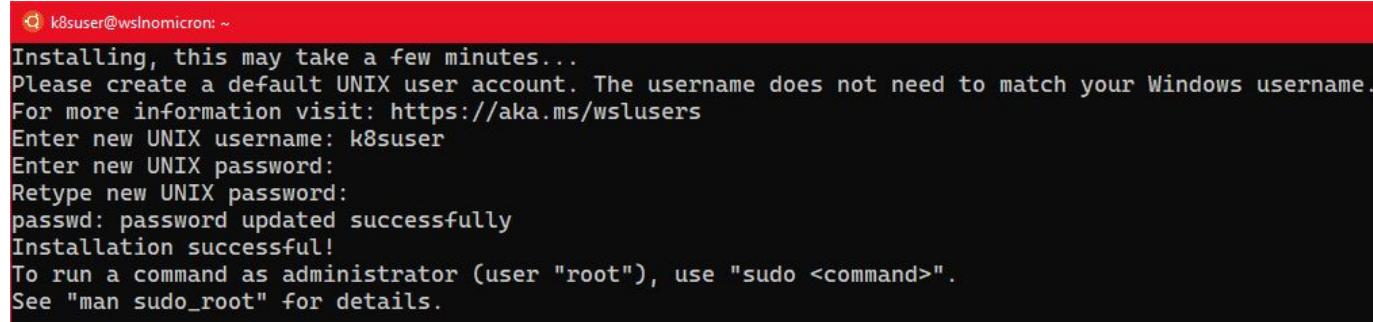


Remarque : D'autres distributions linux existent (*Open SUSE, Pengwin, Fedora Remix, Alpine Linux*)

# Activer WSL2

## ■ Configuration :

Vous devez ensuite lancer la distro Ubuntu pour créer un utilisateur et un password.



```
k8suser@wslnomicroon: ~
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: k8suser
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

## ■ Update -Upgrade :

Il est important d'avoir une version à jour. Il est donc recommandé de lancer les commandes suivantes sur notre distro

*# Update the repositories and list of the packages available*

sudo apt update

*# Update the system based on the packages installed > the "-y" will approve the change automatically*

sudo apt upgrade -y

# WSL2

Vous pouvez ensuite vérifier si votre WSL2 est correctement activé via la ligne de commande suivante : **wsl -l -v**

```
Administrator : Windows PowerShell
PS C:\WINDOWS\system32> wsl -l -v
  NAME          STATE      VERSION
* docker-desktop-data  Running     2
  Ubuntu-20.04    Running     2
  docker-desktop   Running     2
PS C:\WINDOWS\system32>
```

Afin de s'assurer que la version par défaut , nous pouvons exécuter la commande suivante : **wsl --set-default-version 2**

Et si nous voulons « upgrader » un wsl1 vers Wsl2, nous pouvons utiliser la commande suivante : **wsl.exe --set-version DISTRO\_NAME 2**

# Docker Desktop

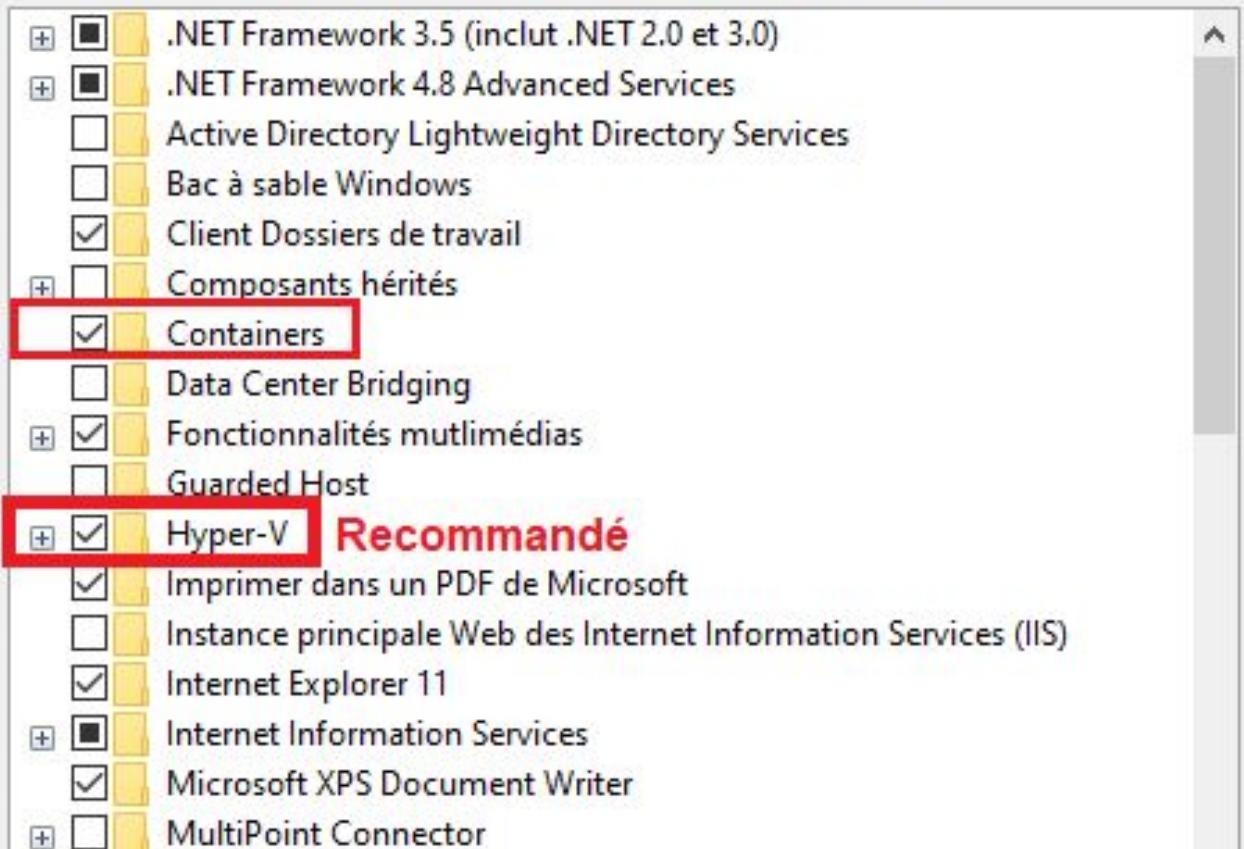
L'installateur est disponible pour différents OS :

- Windows :  
<https://hub.docker.com/editions/community/docker-ce-desktop-windows/>
- Mac :  
<https://hub.docker.com/editions/community/docker-ce-desktop-mac/>

Sous Windows, vous devez avoir la fonctionnalité *Hyper-V* ou *container* activé... Si vous n'avez pas l'*Hyper-V* disponible, l'installation est plus « complexe »

(<https://poweruser.blog/docker-on-windows-10-without-hyper-v-a529897ed1cc>)  
sinon vous devez simplement suivre le wizard step by step.

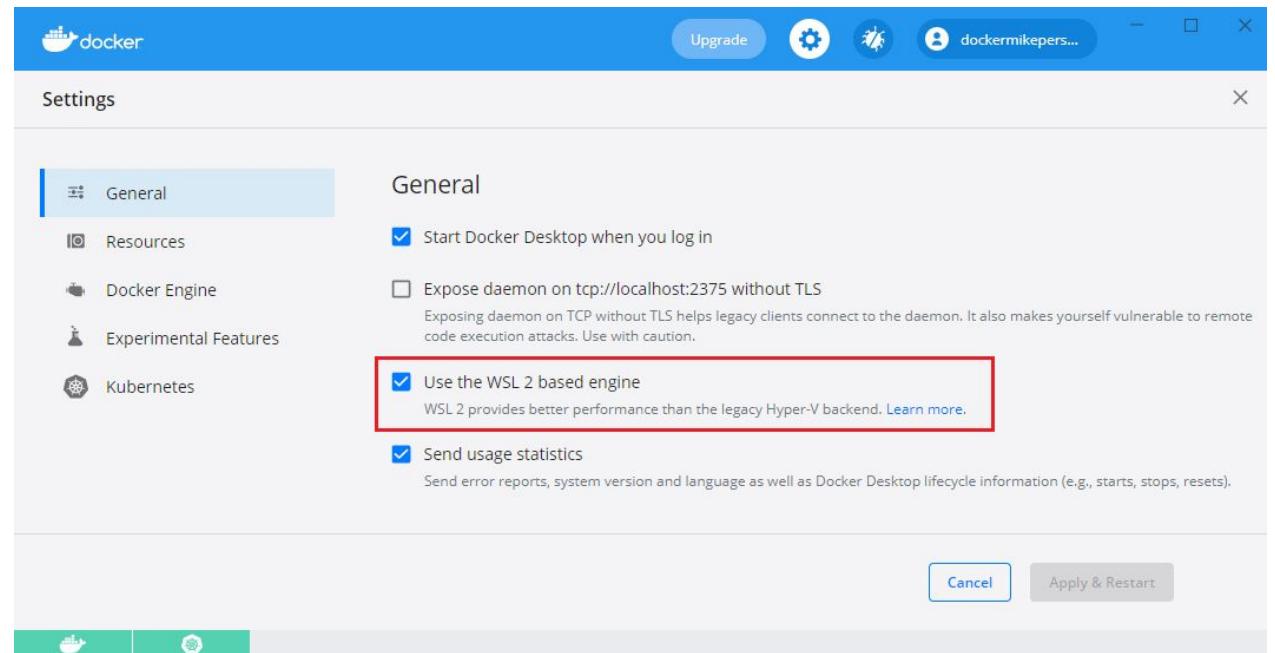
Pour activer une fonctionnalité, activez la case à cocher correspondante. Pour désactiver une fonctionnalité, désactivez la case à cocher correspondante. Une case à cocher pleine signifie qu'une partie de la fonctionnalité est activée.



# Activation WSL2 sur Docker Desktop

Une fois l'installation de Docker Desktop effectuée, nous pouvons configurer celui-ci pour utiliser WSL2 à la place d'une machine virtuelle Hyper-V.

Il faut se rendre dans les settings de *Docker Desktop* et ensuite dans l'onglet général



# Choix de la distro sous Docker Desktop

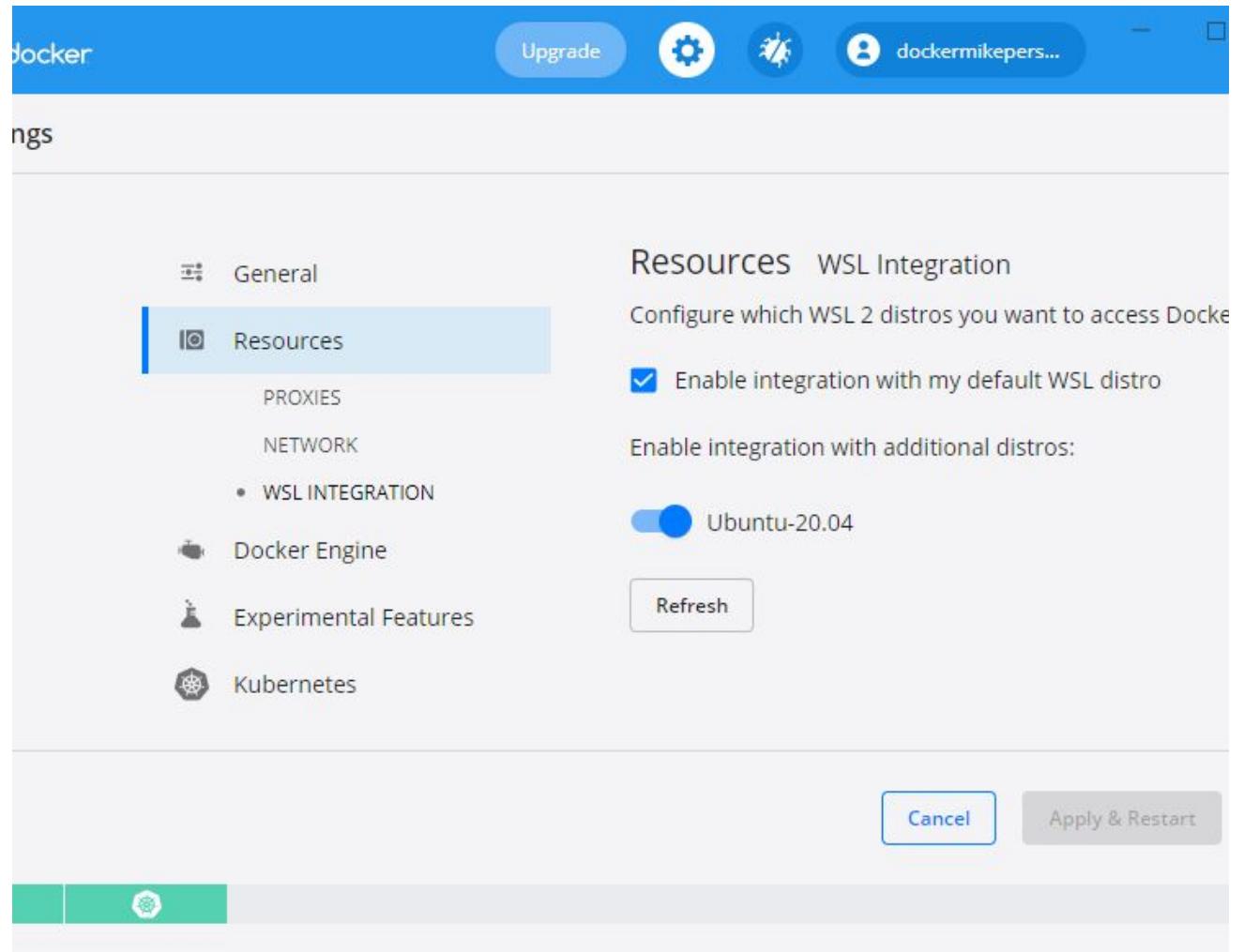
Enfin, nous sélectionnons la distro Ubuntu installée précédemment dans l'onglet ressource  WSL Integration.

## Remarque :

Si votre distro n'apparaît pas, il se peut qu'elle soit en Wsl1...

Dans ce cas, changez là vers WSL2 en tapant la commande suivante dans une console PS en admin :

**wsl.exe --set-version DISTRO\_NAME 2**



# Vérification de notre environnement

Nous pouvons retourner dans notre distro (Ubuntu) et lancer la commande suivante qui nous permet de vérifier que Docker est désormais présent pour notre distro :

*docker version*

```
mike@MIKEW10: ~
mike@MIKEW10:~$ docker version
Client: Docker Engine - Community
  Cloud integration: 1.0.7
  Version:          20.10.2
  API version:     1.41
  Go version:      go1.13.15
  Git commit:      2291f61
  Built:           Mon Dec 28 16:17:34 2020
  OS/Arch:         linux/amd64
  Context:         default
  Experimental:   true

Server: Docker Engine - Community
  Engine:
    Version:          20.10.2
    API version:     1.41 (minimum version 1.12)
    Go version:      go1.13.15
    Git commit:      8891c58
    Built:           Mon Dec 28 16:15:28 2020
    OS/Arch:         linux/amd64
    Experimental:   false
  containerd:
    Version:          1.4.3
    GitCommit:        269548fa27e0089a8b8278fc4fc781d7f65a939b
  runc:
    Version:          1.0.0-rc92
    GitCommit:        ff819c7e9184c13b7c2607fe6c30ae19403a7aff
  docker-init:
    Version:          0.19.0
    GitCommit:        de40ad0
mike@MIKEW10:~$
```

# Dernière étape

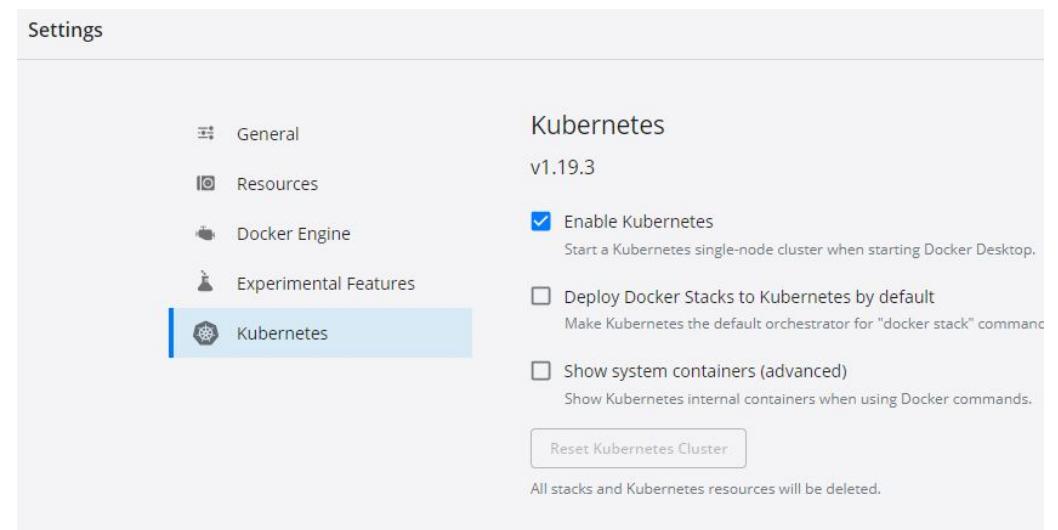
Docker est disponible sous notre distro Mais en ce qui concerne K8S, nous n'avons pas encore de server :



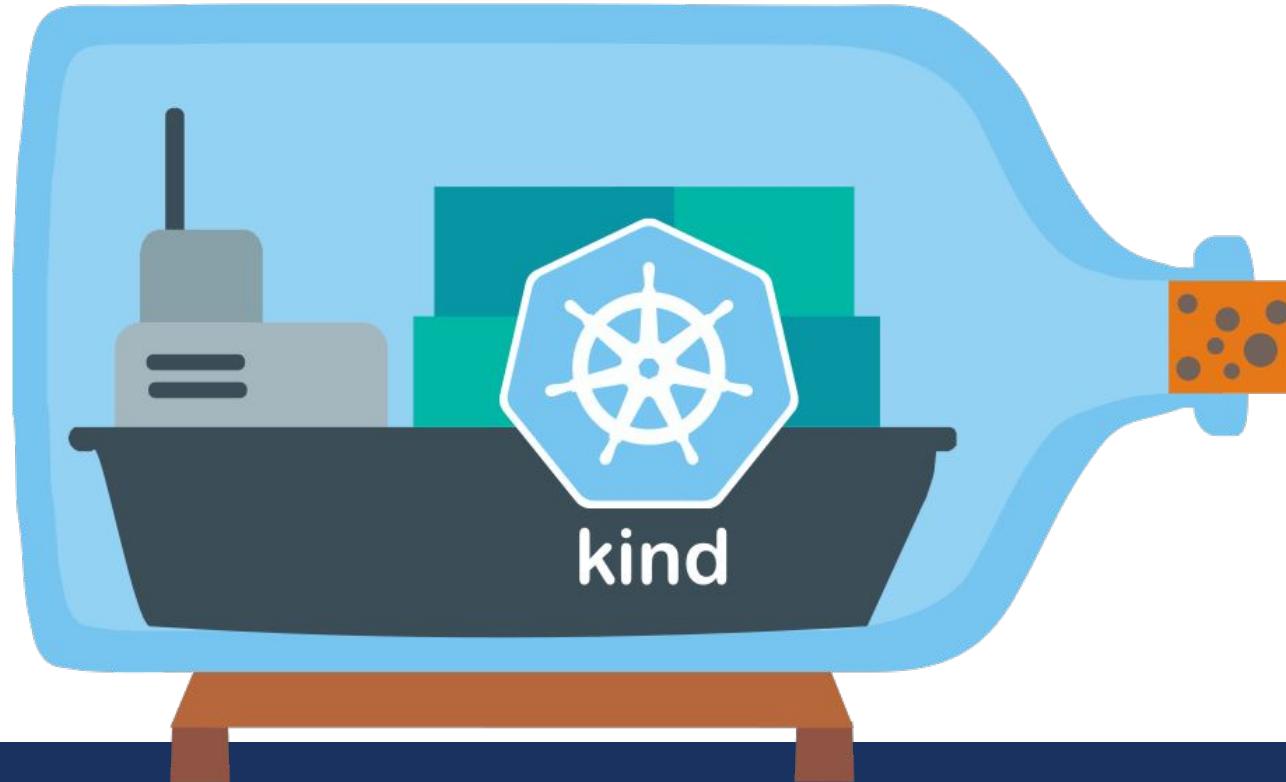
```
mike@MIKEW10:~$ kubectl version
Client Version: version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.3", GitCommit:"1e11e4a2108024935ecfc2912226cedea
d99df", GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z", GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd
64"}
Error from server (InternalError): an error on the server ("") has prevented the request from succeeding
mike@MIKEW10:~$
```

Heureusement, Docker Desktop comprend un serveur Kubernetes qui s'exécute localement dans votre instance Docker.

En activant l'intégration K8S dans docker desktop, nous aurons donc un environnement fonctionnel.



# Installation Multi Nodes avec Kind (Local K8S)



COGNITIC

# Pourquoi Kind (Local K8S) ?

Utiliser Kind (**Kubernetes in Docker**) plutôt que de déployer une architecture complète localement présente plusieurs avantages lorsqu'il s'agit de montrer et d'expliquer Kubernetes :

- **Léger et facile à mettre en place** : Kind permet de créer rapidement des clusters **Kubernetes légers** et isolés sur une seule machine. Il utilise Docker pour exécuter les différents composants de Kubernetes, ce qui signifie qu'il n'est pas nécessaire de configurer une infrastructure complexe avec plusieurs nœuds physiques.
- Rapidité et simplicité : Avec Kind, il est possible de créer un cluster Kubernetes en quelques minutes, ce qui facilite les démonstrations et les explications. Il suffit de quelques commandes pour démarrer un cluster prêt à l'emploi.
- Isolation des environnements : Kind permet de créer des clusters Kubernetes isolés les uns des autres, ce qui est idéal pour effectuer des démonstrations ou des tests sans risquer d'interférer avec d'autres environnements de développement ou de production.
- Portable : Les clusters créés avec Kind peuvent être facilement déployés et exécutés sur différentes machines, ce qui facilite le partage et la collaboration. Il est également possible de les exporter sous forme de fichiers YAML pour les inclure dans des scripts ou des configurations.

En résumé, l'utilisation de Kind pour montrer et expliquer Kubernetes offre une approche légère, rapide et portable, permettant de se concentrer sur les concepts et les fonctionnalités de Kubernetes sans se soucier de la configuration d'une architecture complexe. C'est un outil pratique pour les démonstrations, les tests et l'apprentissage de Kubernetes.

# Multi-Nodes Kubernetes

Si un single-node server peut suffire pour en local, nous avons dans certains cas besoin d'avoir plusieurs nœuds dans K8s.

Pour cela, nous pouvons installer *Kind*(<https://kind.sigs.k8s.io/>).

Kind a été créé au départ pour tester K8S lui-même mais est souvent utilisé pour un environnement local de test.

- Sur notre Ubuntu (WSL distro), nous lançons la commande suivante :

- # For AMD64 / x86\_64
  - [ \$(uname -m) = x86\_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.24.0/kind-linux-amd64
  - # For ARM64
  - [ \$(uname -m) = aarch64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.24.0/kind-linux-arm64
  - chmod +x ./kind
  - sudo mv ./kind /usr/local/bin/kind

- Et nous créons notre premier Cluster

- kind create cluster --name wslkind4Cours*

```
root@DESKTOP-BJ6JDQ3:/home/evengyl# kind create cluster --name wslkind4cours
Creating cluster "wslkind4cours" ...
✓ Ensuring node image (kindest/node:v1.27.1)
✓ Preparing nodes 🐄
✓ Writing configuration 📜
✓ Starting control-plane 🕹️
✓ Installing CNI ✨
✓ Installing StorageClass 💾
Set kubectl context to "kind-wslkind4cours"
You can now use your cluster with:

kubectl cluster-info --context kind-wslkind4cours

Have a question, bug, or feature request? Let us know! https://
root@DESKTOP-BJ6JDQ3:/home/evengyl#
```

# Création du cluster terminée

Nous pouvons constater la création du cluster réussie, les différents node de contrôle ayant déjà été expliqués, nous ne nous attarderons pas dessus pour le moment.

Containers <a href="#">Give feedback</a>							
Container CPU usage				Container memory usage			
12.11% / 1600% (16 CPUs available)				1.18GB / 22.76GB			
Search				Show charts			
<input type="checkbox"/> Only show running containers				<input checked="" type="checkbox"/>			
<input type="checkbox"/>		Name	Image	Status	Port(s)	CPU (%)	Mem
<input type="checkbox"/>		<a href="#">portainer</a> e9eb874615fd	<a href="#">portainer/pce2.21.1</a>	Running	8000:8000 ↗ <a href="#">Show all ports (2)</a>	0%	15.7 23.3
<input type="checkbox"/>		<a href="#">kind-worker</a> a6580ef456f2	<a href="#">kindest/noc</a>	Running		1.68%	163 23.3
<input type="checkbox"/>		<a href="#">kind-worker3</a> b113c8e2c119	<a href="#">kindest/noc</a>	Running		1.14%	163 23.3
<input type="checkbox"/>		<a href="#">kind-worker2</a> 7bd884868624	<a href="#">kindest/noc</a>	Running		0.97%	163 23.3
<input type="checkbox"/>		<a href="#">kind-control-plane</a> 219ee8d37d99	<a href="#">kindest/noc</a>	Running	30000:30000 ↗ <a href="#">Show all ports (9)</a>	8.71%	702. 23.3

# Multi-Nodes Kubernetes (configuration)

Si l'installation est correctement effectuée, nous avons désormais un fichier de config dans notre dossier **\$HOME** (voir annexe).

Le fichier **\$HOME/.kube/config** est un fichier de configuration utilisé par **kubectl** pour spécifier les paramètres de **connexion** au **cluster Kubernetes**, tels que l'adresse du **serveur API**, les informations **d'authentification** et les options de **contexte**.

```
evengyl@DESKTOP-BJ6JDQ3: ~ + 
root@DESKTOP-BJ6JDQ3:/home/evengyl# ls $HOME/.kube
cache  config
```

Nous pouvons obtenir des informations sur notre cluster via la commande : *kubectl cluster-info*

```
root@DESKTOP-BJ6JDQ3:/home/evengyl# kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:44873
CoreDNS is running at https://127.0.0.1:44873/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

# Multi-Nodes Kubernetes (configuration)

Et, cerise sur le gâteau, comme nous utilisons Docker desktop, la configuration réseau est prévue pour que nous puissions utiliser directement l'adresse proposée dans notre navigateur.

```
{  
    "kind": "Status",  
    "apiVersion": "v1",  
    "metadata": {},  
    "status": "Failure",  
    "message": "forbidden: User \\"system:anonymous\\" cannot get path \"/\\\"",  
    "reason": "Forbidden",  
    "details": {},  
    "code": 403  
}
```

# Multi-Nodes Kubernetes (configuration) KIND

Nous obtenons un message nous signalant un souci d'accès.

C'est tout à fait normal car l'accès à l'API utilise l'autorisation **RBAC**  
**(Rôle-Based Access Control)**.

Nous allons voir par la suite comment utiliser RVAC pour accéder au Dashboard K8S

Pour les curieux :

<https://kubernetes.io/docs/reference/access-authn-authz/rbac/>

```
{  
  "kind": "Status",  
  "apiVersion": "v1",  
  "metadata": {},  
  "status": "Failure",  
  "message": "forbidden: user \"\" cannot get resource \"Nodes\" at group \"\" in version \"v1\"",  
  "reason": "Forbidden",  
  "details": {},  
  "code": 403  
}
```

Pour plus d'information sur la création de cluster via Kind :

<https://kind.sigs.k8s.io/docs/user/configuration/#getting-started>

# Dashboard

Même si nous pouvons obtenir les informations sur le cluster via l'API RESTful en json ou via la ligne de commande, il est plus confortable d'avoir une interface utilisateur qui nous présentera l'information d'une façon plus user-friendly : **un dashboard**.

0 ) Installer Helm pour faciliter le déploiement : <https://helm.sh/docs/intro/install/>

1) Ajout du repository Helm pour le dashboard

```
helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/
```

2) Installation du dashboard via Helm

```
helm upgrade --install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard --create-namespace  
--namespace kubernetes-dashboard
```

# Multi-Nodes Kubernetes (Dashboard)

## 2. Vérification de la création de la ressource

a. `kubectl get all -n kubernetes-dashboard`

mike@LenoMike:~/Final\$ kubectl get all -n kubernetes-dashboard					
NAME	READY	STATUS	RESTARTS	AGE	
pod/kubernetes-dashboard-api-8544788d69-kgfr2	1/1	Running	0	59s	
pod/kubernetes-dashboard-auth-7fcf56748b-vg5j2	1/1	Running	0	59s	
pod/kubernetes-dashboard-kong-57d45c4f69-72kkb	1/1	Running	0	59s	
pod/kubernetes-dashboard-metrics-scraper-6b6f6f5d5c-cq9sg	1/1	Running	0	59s	
pod/kubernetes-dashboard-web-75cccd6488-5vplx	1/1	Running	0	59s	
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes-dashboard-api	ClusterIP	10.96.242.239	<none>	8000/TCP	59s
service/kubernetes-dashboard-auth	ClusterIP	10.96.239.101	<none>	8000/TCP	59s
service/kubernetes-dashboard-kong-manager	NodePort	10.96.128.223	<none>	8002:31389/TCP,8445:32585/TCP	59s
service/kubernetes-dashboard-kong-proxy	ClusterIP	10.96.73.252	<none>	443/TCP	59s
service/kubernetes-dashboard-metrics-scraper	ClusterIP	10.96.197.126	<none>	8000/TCP	59s
service/kubernetes-dashboard-web	ClusterIP	10.96.161.12	<none>	8000/TCP	59s
NAME	READY	UP-TO-DATE	AVAILABLE	AGE	
deployment.apps/kubernetes-dashboard-api	1/1	1	1	59s	
deployment.apps/kubernetes-dashboard-auth	1/1	1	1	59s	
deployment.apps/kubernetes-dashboard-kong	1/1	1	1	59s	
deployment.apps/kubernetes-dashboard-metrics-scraper	1/1	1	1	59s	
deployment.apps/kubernetes-dashboard-web	1/1	1	1	59s	
NAME	DESIRED	CURRENT	READY	AGE	
replicaset.apps/kubernetes-dashboard-api-8544788d69	1	1	1	59s	
replicaset.apps/kubernetes-dashboard-auth-7fcf56748b	1	1	1	59s	
replicaset.apps/kubernetes-dashboard-kong-57d45c4f69	1	1	1	59s	
replicaset.apps/kubernetes-dashboard-metrics-scraper-6b6f6f5d5c	1	1	1	59s	
replicaset.apps/kubernetes-dashboard-web-75cccd6488	1	1	1	59s	

# Multi-Nodes Kubernetes (Dashboard)

3. Nous créons port forward permettant d'atteindre le dashboard

```
kubectl -n kubernetes-dashboard port-forward svc/kubernetes-dashboard-kong-proxy 8443:443
```

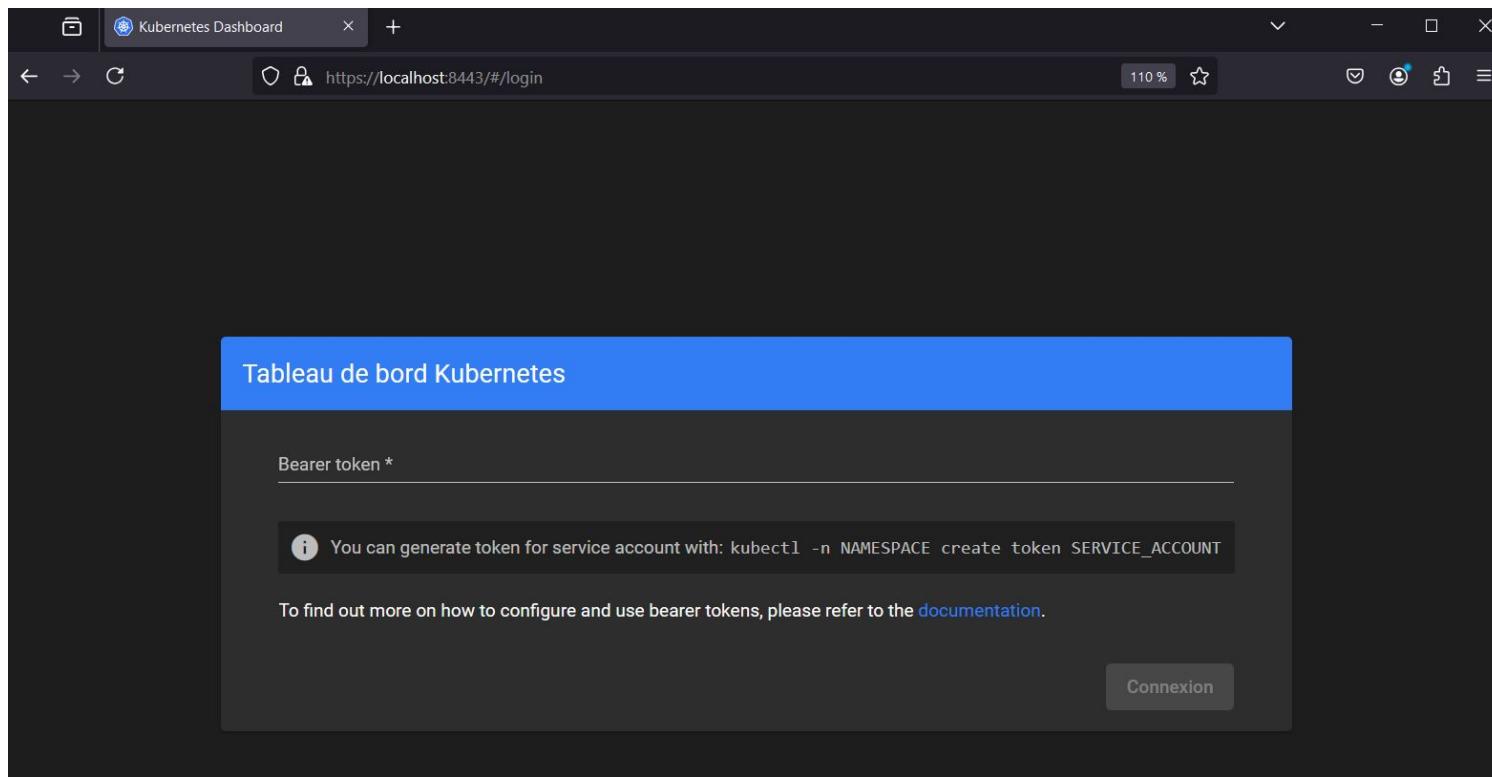
```
mike@LenoMike:~/Final$ kubectl -n kubernetes-dashboard port-forward svc/kubernetes-dashboard-kong-proxy 8443:443
Forwarding from 127.0.0.1:8443 -> 8443
Forwarding from [::1]:8443 -> 8443
```

C'est une solution de dépannage car le fait de faire un port-forward condamne notre console.

Nous verrons plus tard comment mettre en place un Ingress controller pour pouvoir atteindre notre Dashboard à l'extérieur de notre Cluster

# Multi-Nodes Kubernetes (Dashboard)

4. Nous pouvons entrer l'adresse vers le dashboard dans notre navigateur :
  - a. <https://localhost:8443/#/login>



# Multi-Nodes Kubernetes - KubeConfig

## Configuration des accès

Pour permettre l'accès au cluster, nous devons modifier notre « *kubeconfig* ».

Nous créons donc un fichier de configuration pour l'admin-user de la dashboard :

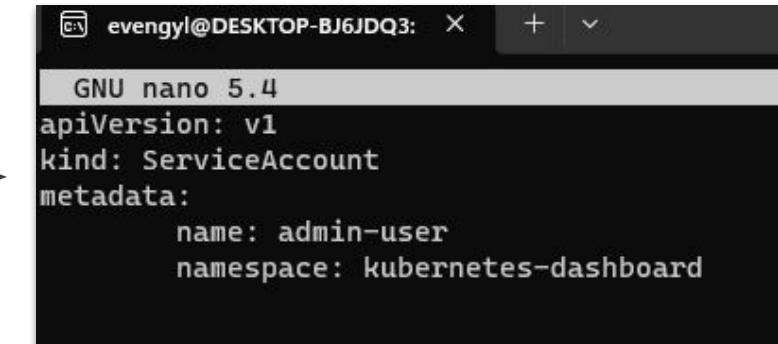
- `nano $HOME/.kube/admin-user.yaml`

Un fichier de configuration décrit les clusters, les utilisateurs et les contextes.

Nous allons configurer l'accès au dashboard, en commençant par créer un compte de service pour le namespace dashboard via la commande kubectl

Le compte de service doit-être en **minuscule** et commencer et terminer par une lettre. Les seuls caractères spéciaux permis sont \_ et -

Remarque : Un fichier utilisé pour configurer l'accès à un cluster est appelé *kubeconfig*. C'est une manière générique de se référer aux fichiers de configuration. Cela ne signifie pas nécessairement qu'il existe un fichier nommé *kubeconfig*.



```
GNU nano 5.4
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
```

```
root@DESKTOP-BJ6JDQ3:~/ .kube# kubectl apply -f admin-user.yaml
serviceaccount/admin-user created
root@DESKTOP-BJ6JDQ3:~/ .kube#
```

Une fois terminé nous pouvons appliquer la configuration au cluster avec kubectl

COGNITIC

- `kubectl apply -f admin-user.yaml`

# Multi-Nodes Kubernetes - KubeConfig

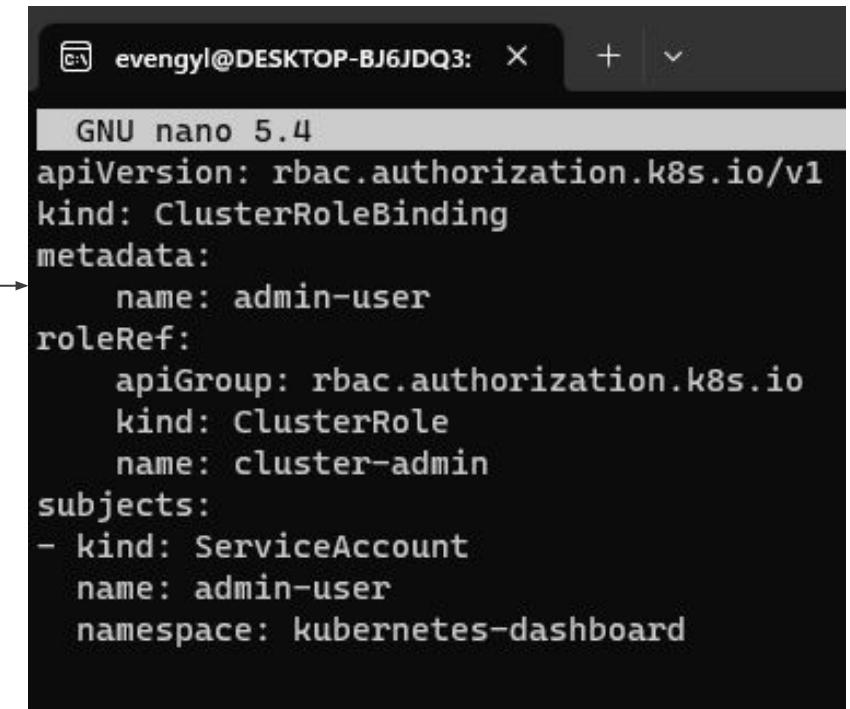
Dans la plupart des cas, après avoir provisionné le cluster à l'aide de kops, **kubeadm** ou de tout autre outil populaire, l'administrateur de cluster **ClusterRole** existe déjà dans le cluster. Nous pouvons l'utiliser et créer simplement un **Cluster Rôle Binding** pour notre **ServiceAccount**. S'il n'existe pas, vous devez d'abord créer ce rôle et accorder manuellement les priviléges requis.

Il nous faut maintenant lier le rôle **ClusterAdmin** au compte de service comme il n'existait pas, ensuite on applique avec kubectl apply

- *kubectl apply -f apply-role.yaml*



```
root@DESKTOP-BJ6JDQ3:~/.kube# kubectl apply -f apply-role.yaml
clusterrolebinding.rbac.authorization.k8s.io/admin-user created
root@DESKTOP-BJ6JDQ3:~/.kube#
```



```
evengyl@DESKTOP-BJ6JDQ3: ~ + v
GNU nano 5.4
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

# Multi-Nodes Kubernetes - KubeConfig

Récupération du Token d'accès :

- Nous pouvons désormais récupérer le Token qui nous permettra de nous authentifier sur le dashboard en tant qu'Admin.
- `kubectl -n kubernetes-dashboard create token admin-user`

*(vous pouvez utiliser `-duration 24h` pour définir un token 24h par exemple)*

```
evengyl@DESKTOP-BJ6JDQ3: ~/.kube# kubectl -n kubernetes-dashboard create token admin-user
eyJhbGciOiJSUzI1NiIsImtpZCI6Ik1IYXI1TDN3b0lKT0Y1YmE0c29kTFRzY3JDOS1PaFh1TVB2SkhyTxdkRzQifQ.eyJhdWQiOlsiaHR0cHM6Ly9rd
WJlcmt5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIxubG9jYWwiXSwiZXhwIjoxNjg0NTgxMDQwLCJpYXQiOjE2ODQ1Nzc0NDAsImlzcyI6Imh0dHBzOi8va
3ViZXJuZXRLcy5kZWZhdx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRLcy5pbvI6eyJuYW1lc3BhY2UiOjJrdWJlcmt5ldGVzLWRhc2hib2FyZ
CIsInNlcnZpY2VhY2NvdW50Ijp7Im5hbWUiOjJhZG1pbii1c2VyIiwidWlkIjoiYzFmODYwZDUtY2EzOC00YjI1LWIwMGYtZDg4NTBlZGU0NjEyIn19L
CJuYmYiOjE2ODQ1Nzc0NDAsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlcmt5ldGVzLWRhc2hib2FyZDphZG1pbii1c2VyIn0.L0z2ZEaKT
0jh54MHMGabkd9zoXwKUIPzAdsA02zJVa4q164JLM0wvo-GBYfTw_oIo0b_K8irWRXIWfoRNbg5WAlW3Sf52SX0ouObJYZ9idCE1SMmkzgw7tapPLn82
9U4C_G3Qq9wBiJ00CDWo9M9fZzk7vi8jZgRarG8nr5odHm87GevkDq4hZ0iWRZVAHLm5zPHx6Qe5Tz6KH4YJTtJPn8EesxiEcc346PTloxF6ImuQsHVTk
loFoQ_i7LjugH19dN021MvCVOY0DQmdKLEM-_OJ71K1tHGS2scDV_8gOUZvN_kza61pHz1uPQ0pxcN_uAb7DhqQizadlc9RANXpxA
root@DESKTOP-BJ6JDQ3: ~/.kube#
```

# Multi-Nodes Kubernetes - KubeConfig

Application de notre jeton pour accéder à la dashboard, n'oublions pas que le proxy de kubectl doit être activé !

```
evengyl@DESKTOP-BJ6JDQ3: ~$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

### Kubernetes Dashboard

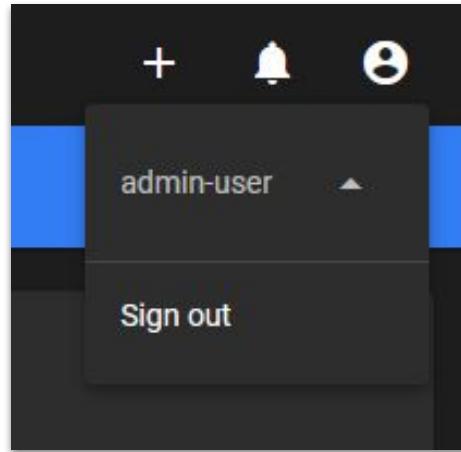
Jeton  
Chaque compte de service a un Secret associé avec un jeton porteur (Bearer Token) valide qui peut être utilisé pour se connecter au Dashboard. Pour en savoir plus sur la façon de configurer et utiliser des jetons porteurs, veuillez vous référer à la section [Authentification](#).

Kubeconfig  
Veuillez sélectionner le fichier kubeconfig que vous avez créé pour accéder au cluster. Pour en savoir plus sur la façon de configurer et utiliser un fichier kubeconfig, veuillez vous référer à la section [Configurer l'accès à plusieurs clusters](#).

Saisissez un jeton \*

Connexion

# Multi-Nodes Kubernetes - Dashboard access

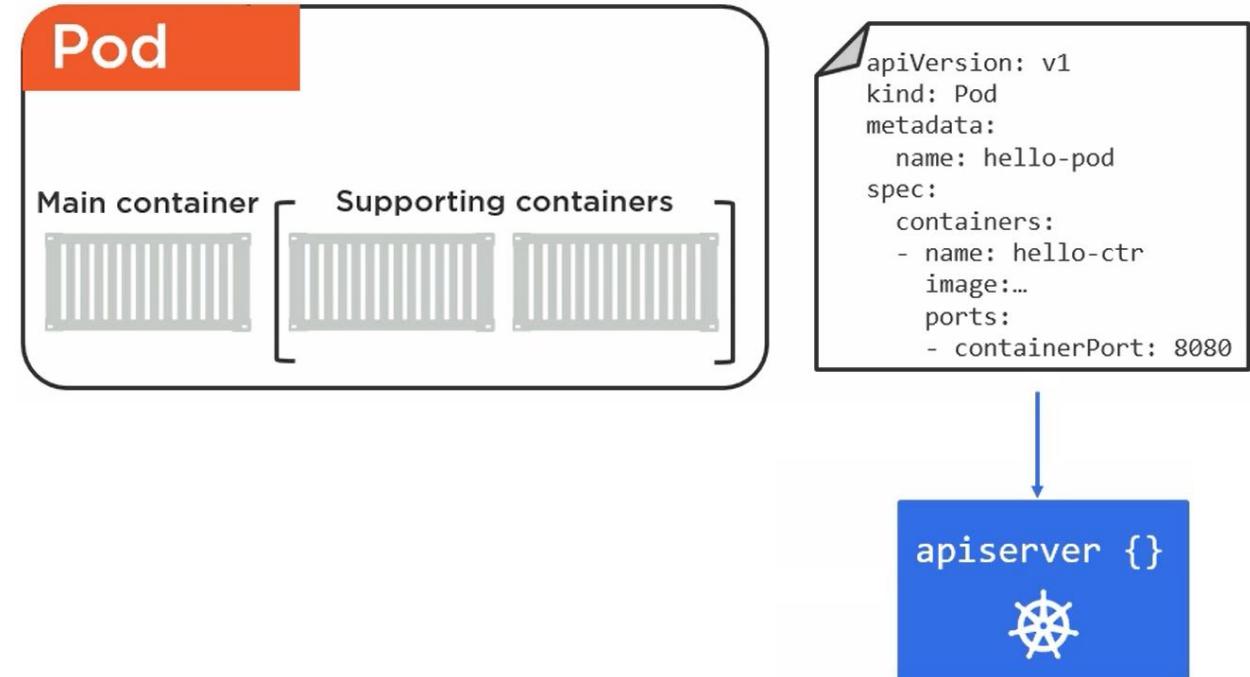
A screenshot of a web browser displaying the Kubernetes Dashboard at the URL "localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard/proxy/#/workloads?namespace=default". The page has a dark theme. The top navigation bar includes a back button, forward button, refresh button, and a search bar with the placeholder "Recherche". The main content area has a blue header bar with the text "Charges de travail". On the left, there is a sidebar with buttons for "Workloads" (highlighted), "Cron Jobs", and "Daemon Sets". The main content area displays the message "Il n'y a rien à afficher ici" (There is nothing to display here) and a sub-message: "You can [deploy a containerized app](#), select other namespace or [take the Dashboard Tour](#) to learn more.".

# Pods

Kubernetes

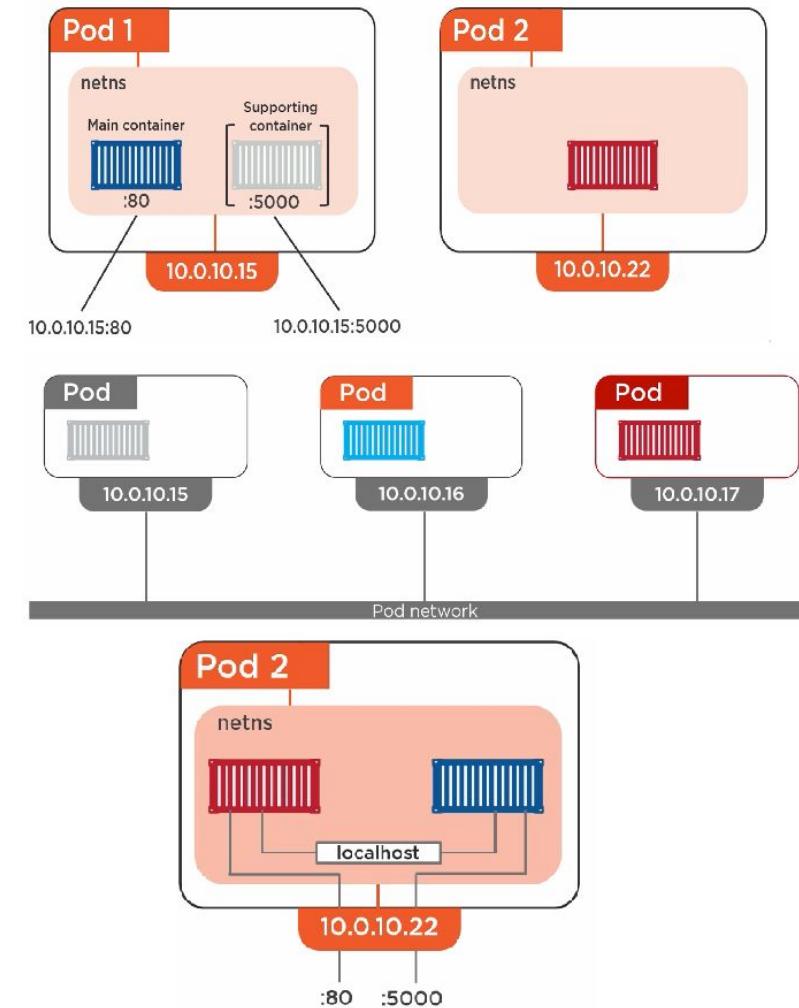
# Pods

- Les pods sont les plus petites unités atomiques de K8S.
- Ils contiennent 1,ou plusieurs container(s).
- Ils sont définis par un fichier *manifest* qui est envoyé via l'api-server afin que K8S « instancie » celui-ci



# Pods

- Chaque Pod a sa propre IP. Les containers présents dans les pods ne peuvent donc pas utiliser les mêmes ports.
- Ils sont dans le même *netnamespace*
- Les containers discutent via *Localhost* entre eux
- Les pods dans un même cluster ont des IP dans le même range ce qui permet la communication inter-pods (*podnetwork*)



# Pods – Cycle de vie

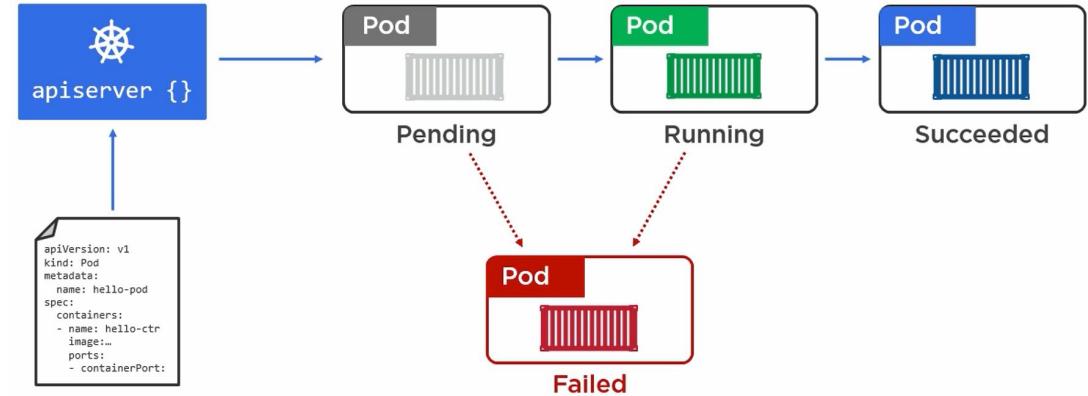
Après avoir envoyé le *manifest* à l'*ApiServer*,

Le pods est « *schedulé* » et passe par les phases suivantes :

1. **Pending** : Le Pod a été accepté par Kubernetes, mais une ou plusieurs images de conteneurs n'ont pas encore été créées
2. **Running** : Le pod a été affecté à un nœud et tous les conteneurs ont été créés. Au moins un conteneur est toujours en cours d'exécution, ou est en train de démarrer ou redémarrer.
3. **Succeed** : Tous les conteneurs du pod ont terminé avec succès et ne seront pas redémarrés.

Deux autres phases :

1. **Failed** : Tous les conteneurs d'un pod ont terminé, et au moins un conteneur a terminé en échec
2. **Unknown** : Pour quelque raison l'état du pod ne peut pas être obtenu



## Remarque :

Un Pod est « mortel »: lorsqu'il tombe en failed, nous n'avons pas le moyen pour le « ressusciter » si ce n'est effectuer un nouveau déploiement

# Pods

## ■ Exemple de création d'un pods nginx

### 1) Création du fichier manifest : vi web.yml

- Kind : Pod
- Nom du pod : web
- Nom du container : web
- Nœud : kind-worker
- Image : nginx:latest

### 2) Création du pod

**kubectl apply -f web.yml**

```
mike@MIKEW10:~/CoursePod$ cat web.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: web
spec:
  nodeName: kind-worker
  containers:
  - name: nginx
    image: registry.hub.docker.com/library/nginx:latest
    ports:
    - containerPort: 80
mike@MIKEW10:~/CoursePod$ kubectl apply -f web.yml
pod/nginx created
mike@MIKEW10:~/CoursePod$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
nginx    1/1     Running   0          11s
mike@MIKEW10:~/CoursePod$
```

# Pods

## 3) (bonus) Port forwarding

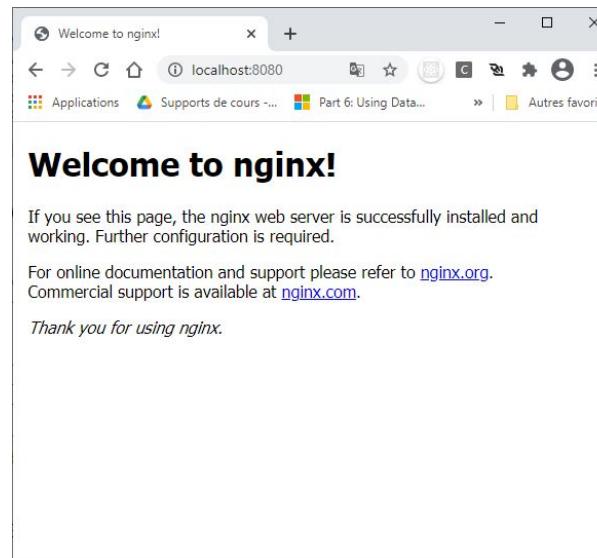
Nous pouvons mettre en place un port forwarding pour tester notre serveur web en dehors de notre environnement K8S

`kubectl port-forward nonPod portexterne:portPod`

Remarque :

**cette commande ne s'utilise que pour nos tests,  
en production, nous verrons comment créer  
proprement un déploiement**

```
mike@MIKEW10:~/CoursePod
mike@MIKEW10:~/CoursePod$ kubectl port-forward nginx 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```



# Pods

## Suppression d'un pod

trois étapes sont nécessaires :

### 1. **kubectl drain <nodename>**

Permet d'éjecter tous les utilisateurs du node

### 2. **Kubectl cordon <nodename>**

Evite qu'un nouveau pod soit planifié pendant notre opération

### 3. **Kubectl delete pod/ <podname>**

Supprime définitivement le pod

```
mike@MIKEW10:~$ kubectl get nodes
NAME                  STATUS   ROLES      AGE     VERSION
wslkind4cours-control-plane   Ready    master   20h    v1.17.0
mike@MIKEW10:~$ kubectl get pods -o wide | grep <nodename>
-bash: syntax error near unexpected token `newline'
mike@MIKEW10:~$ kubectl get pods -o wide | grep <nodename>
-bash: syntax error near unexpected token `newline'
mike@MIKEW10:~$ kubectl get pods -o wide | grep nginx
nginx   1/1   Running   0          29m   10.244.0.7   wslkind4cours-control-plane
mike@MIKEW10:~$ kubectl cordon wslkind4cours-control-plane
node/wslkind4cours-control-plane cordoned
mike@MIKEW10:~$ kubectl delete pod nginx
pod "nginx" deleted
mike@MIKEW10:~$ kubectl get pods -o wide | grep nginx
No resources found in default namespace.
mike@MIKEW10:~$ kubectl get nodes
NAME                  STATUS   ROLES      AGE     VERSION
wslkind4cours-control-plane   Ready,SchedulingDisabled   master   20h    v1.17.0
mike@MIKEW10:~$ kubectl uncordon wslkind4cours-control-plane
node/wslkind4cours-control-plane uncordoned
mike@MIKEW10:~$ kubectl get nodes
NAME                  STATUS   ROLES      AGE     VERSION
wslkind4cours-control-plane   Ready    master   20h    v1.17.0
mike@MIKEW10:~$ kubectl get pods -o wide | grep nginx
No resources found in default namespace.
mike@MIKEW10:~$
```

# Pods

Nous pouvons ensuite vérifier si notre pods est créé correctement via les commandes suivantes :

- **Kubectl get pods** : nous renvoie la liste des pods de notre cluster
- **kubectl describe po/nginx**: permet d'obtenir les informations sur notre pod (IP, nœud, status, container, volumes, events,...)

```
mike@MIKEW10: ~/CoursePod
mike@MIKEW10:~/CoursePod$ kubectl get pods
NAME     READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          2m44s
```

## Remarque :

Docker n'est plus le « container runtime »  
par défaut , il a été remplacé par containerd.

## Plus d'infos :

<https://www.docker.com/blog/what-is-containerd-runtime/>

```
mike@MIKEW10: ~/CoursePod
mike@MIKEW10:~/CoursePod$ kubectl describe po/nginx
Name:           nginx
Namespace:      default
Priority:       0
Node:          wslkind4cours-control-plane/172.17.0.2
Start Time:    Tue, 16 Feb 2021 11:07:39 +0100
Labels:         app=web
Annotations:   <none>
Status:        Running
IP:            10.244.0.7
IPs:
  IP:  10.244.0.7
Containers:
  nginx:
    Container ID:  containerd://72b5a1ec34ee25c41094ea04f0f4653b4f48bf1c7e4b738d730e0a6df389d874
    Image:          registry.hub.docker.com/library/nginx:latest
    Image ID:      registry.hub.docker.com/library/nginx@sha256:8e10056422503824ebbe99f37c26a90fe705419426
```

# Pods

## ■ Exemple de création d'un second pods avec ouverture d'un Shell

### 1) Création du fichier manifest : vi inspect.yml

- Kind : Pod
- Nom du pod : inspect
- Nom du container : inspect
- Nœud : kind-worker2
- Image : alpine:latest

### 2) Création du pod

**kubectl apply -f inspect.yml**

Le statut du pods *inspect* est *Completed*. Nous ne pouvons donc plus interagir avec lui. Pour éviter cela, nous allons lui transmettre une commande. cette commande aura pour but de nous laisser le temps d'ouvrir un Shell sur ce pod

```
mike@MIKEW10:~/CoursePod$ cat inspect.yml
apiVersion: v1
kind: Pod
metadata:
  name: inspect
  labels:
    version: v1
    zone: dev
spec:
  nodeName: kind-worker2
  containers:
  - name: inspect
    image: alpine:latest
```

```
mike@MIKEW10:~/CoursePod$ kubectl apply -f inspect.yml
pod/inspect created
mike@MIKEW10:~/CoursePod$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
inspect   0/1     Completed  1          12s
nginx     1/1     Running   0          9m8s
mike@MIKEW10:~/CoursePod$
```

# PODS

## ■ Les commandes dans le fichier de spécification

Il est possible de transmettre une commande et des arguments au container lors de la création du Pods via la spécification *command*.

### Remarque :

La commande et les arguments que vous définissez dans le fichier de configuration remplacent la commande et les arguments par défaut fournis par l'image du conteneur

## ■ Exemple :

```
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
```

# PODS

- 1) Suppression du Pod inspect
- 2) Modification de notre spécification
  - Ajout d'une commande sleep de 3600 secondes
- 3) Création de notre Pods

Désormais notre pod est en état *Running* et nous pouvons interagir avec lui.

Nous pouvons, par exemple, lancer un Shell dans le conteneur *alpine* de notre pod via la commande **exec**

```
kubectl exec -t -i inspect -- sh
```

**-t** : pour terminal

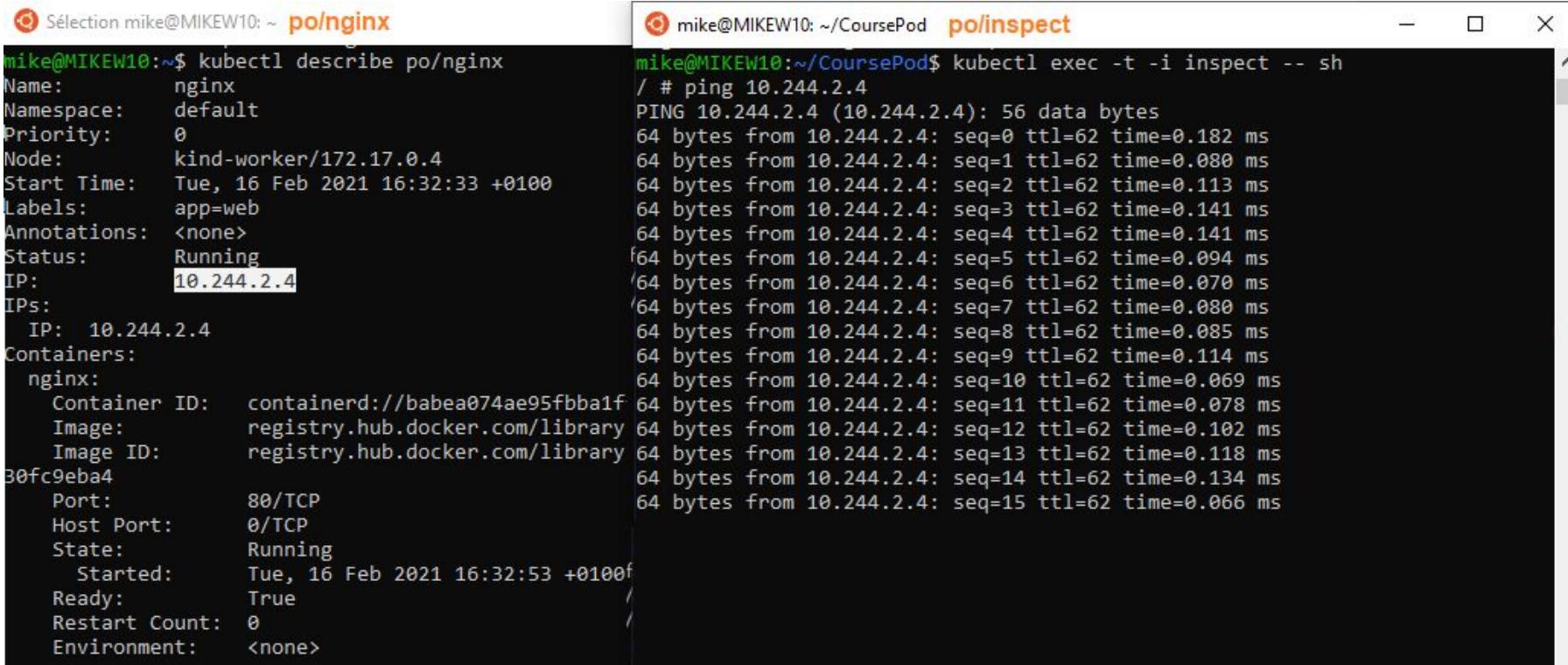
**-i**: interactive

**-- sh** : notre shell

```
mike@MIKEW10:~/CoursePod$ kubectl get pods
NAME      READY   STATUS            RESTARTS   AGE
nginx    0/1     ContainerCreating   0          20s
mike@MIKEW10:~/CoursePod$ cat inspectOK.yml
apiVersion: v1
kind: Pod
metadata:
  name: inspect
  labels:
    version: v1
    zone: dev
spec:
  nodeName: kind-worker2
  containers:
  - name: inspect
    image: alpine:latest
    command: ["sleep","3600"]
mike@MIKEW10:~/CoursePod$ kubectl apply -f inspectOK.yml
pod/inspect created
mike@MIKEW10:~/CoursePod$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
inspect  1/1     Running   0          11s
nginx    1/1     Running   0          67s
mike@MIKEW10:~/CoursePod$
```

# Pods

Nous pouvons ensuite utiliser le Shell de notre pods *inspect* afin de vérifier que nos Pods se voient et peuvent communiquer.



The image shows two terminal windows side-by-side. The left window, titled 'Sélection mike@MIKEW10: ~ po/nginxx', displays the output of the command 'kubectl describe po/nginxx'. It provides detailed information about the Pod, including its name (nginx), namespace (default), node (kind-worker/172.17.0.4), start time (Tue, 16 Feb 2021 16:32:33 +0100), labels (app=web), annotations (<none>), status (Running), and IP address (10.244.2.4). It also lists the container (nginx) with its container ID (containerd://babea074ae95fbba1f), image (registry.hub.docker.com/library/nginx), and port mapping (Port: 80/TCP, Host Port: 0/TCP, State: Running, Started: Tue, 16 Feb 2021 16:32:53 +0100, Ready: True, Restart Count: 0, Environment: <none>).

The right window, titled 'mike@MIKEW10: ~/CoursePod po/inspect', shows the output of 'kubectl exec -t -i inspect -- sh'. It runs a ping command from the 'inspect' shell to the IP 10.244.2.4. The output shows multiple ping requests being sent and received, with sequence numbers (seq=0 to seq=15), TTL values (ttl=62), and round-trip times (time) ranging from 0.066 ms to 0.182 ms.

```
mike@MIKEW10:~/CoursePod$ kubectl exec -t -i inspect -- sh
/ # ping 10.244.2.4
PING 10.244.2.4 (10.244.2.4): 56 data bytes
64 bytes from 10.244.2.4: seq=0 ttl=62 time=0.182 ms
64 bytes from 10.244.2.4: seq=1 ttl=62 time=0.080 ms
64 bytes from 10.244.2.4: seq=2 ttl=62 time=0.113 ms
64 bytes from 10.244.2.4: seq=3 ttl=62 time=0.141 ms
64 bytes from 10.244.2.4: seq=4 ttl=62 time=0.141 ms
64 bytes from 10.244.2.4: seq=5 ttl=62 time=0.094 ms
64 bytes from 10.244.2.4: seq=6 ttl=62 time=0.070 ms
64 bytes from 10.244.2.4: seq=7 ttl=62 time=0.080 ms
64 bytes from 10.244.2.4: seq=8 ttl=62 time=0.085 ms
64 bytes from 10.244.2.4: seq=9 ttl=62 time=0.114 ms
64 bytes from 10.244.2.4: seq=10 ttl=62 time=0.069 ms
64 bytes from 10.244.2.4: seq=11 ttl=62 time=0.078 ms
64 bytes from 10.244.2.4: seq=12 ttl=62 time=0.102 ms
64 bytes from 10.244.2.4: seq=13 ttl=62 time=0.118 ms
64 bytes from 10.244.2.4: seq=14 ttl=62 time=0.134 ms
64 bytes from 10.244.2.4: seq=15 ttl=62 time=0.066 ms
```

# Command Line - Summary

```
# Afficher la liste des pods
kubectl get pods [En option <POD NAME>]
  -o : format de sortie
    wide : afficher l'ip du pod et le nœud qui l'héberge
    yaml : afficher encore plus d'informations sur un pod sous format YAML
    json : afficher encore plus d'informations sur un pod sous format JSON
  --template : récupérer des informations précises de la sortie de la commande
    Récupérer l'IP d'un pod : kubectl get pod <POD NAME> --template={{.status.podIP}}
# Créer un Pod
kubectl create -f <filename.yaml>
# Supprimer un pod
kubectl delete pods <POD NAME>
# Appliquer des nouveaux changements à votre Pod sans le détruire
kubectl apply -f <filename.yaml>
```

# Command Line - Summary

```
# Afficher les détails d'un pod
kubectl describe pods <POD NAME>
# Exécuter une commande d'un conteneur de votre pod
kubectl exec <POD NAME> -c <CONTAINER NAME> <COMMAND>
  -t : Allouer un pseudo TTY
  -i : Garder un STDIN ouvert
# Afficher les logs d'un conteneur dans un pod
kubectl logs <POD NAME> -c <CONTAINER NAME>
  -f : suivre en permanence les logs du conteneur
  --tail : nombre de lignes les plus récentes à afficher
  --since=1h : afficher tous les logs du conteneur au cours de la dernière heure
  --timestamps : afficher la date et l'heure de réception des logs d'un conteneur
```

# Exercice

Créer les scripts nécessaires pour la mise en place de l'architecture suivante sous K8S :

- 1 cluster
  - 3 nodes
    - Controller-plane(Master)
    - Worker
    - Worker2

Vous devez faire en sorte d'obtenir 1 pod dans le noeud worker2 avec le server du projet

<https://github.com/alex-arriaga/node-js-restful-api-example>

Prévoyez également un container dans 1 pod dans le nœud Worker, ce container devra héberger un client consommant le service et permettre d'y accéder via votre navigateur (hors K8S).

Cette application devra être accessible sur l'adresse <http://127.0.0.1:8080>

# Exercice

Remarque :

Pour faire fonctionner notre application, nous sommes obligés d'utiliser un port-forward afin que l'application cliente puisse accéder à notre api et également être accessible en dehors de notre K8s.

Ce n'est pas viable à long terme.

# Services

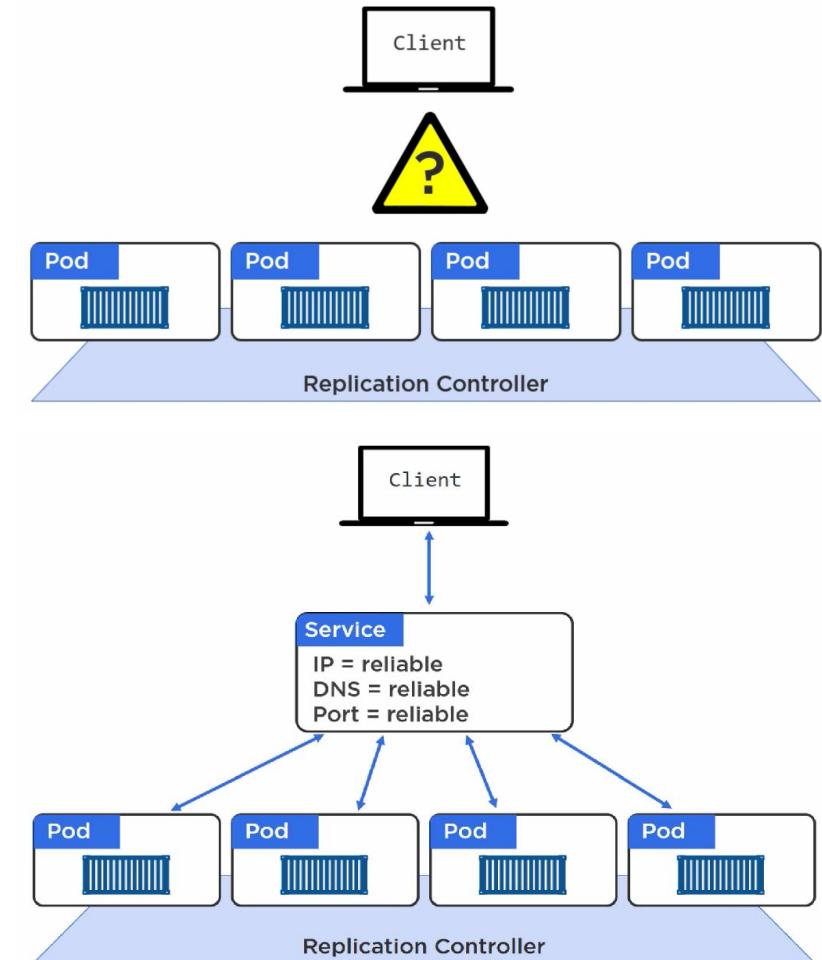
Kubernetes

# Services

Comment communiquer avec nos pods à partir d'un client extérieur?

## □ Via un Service

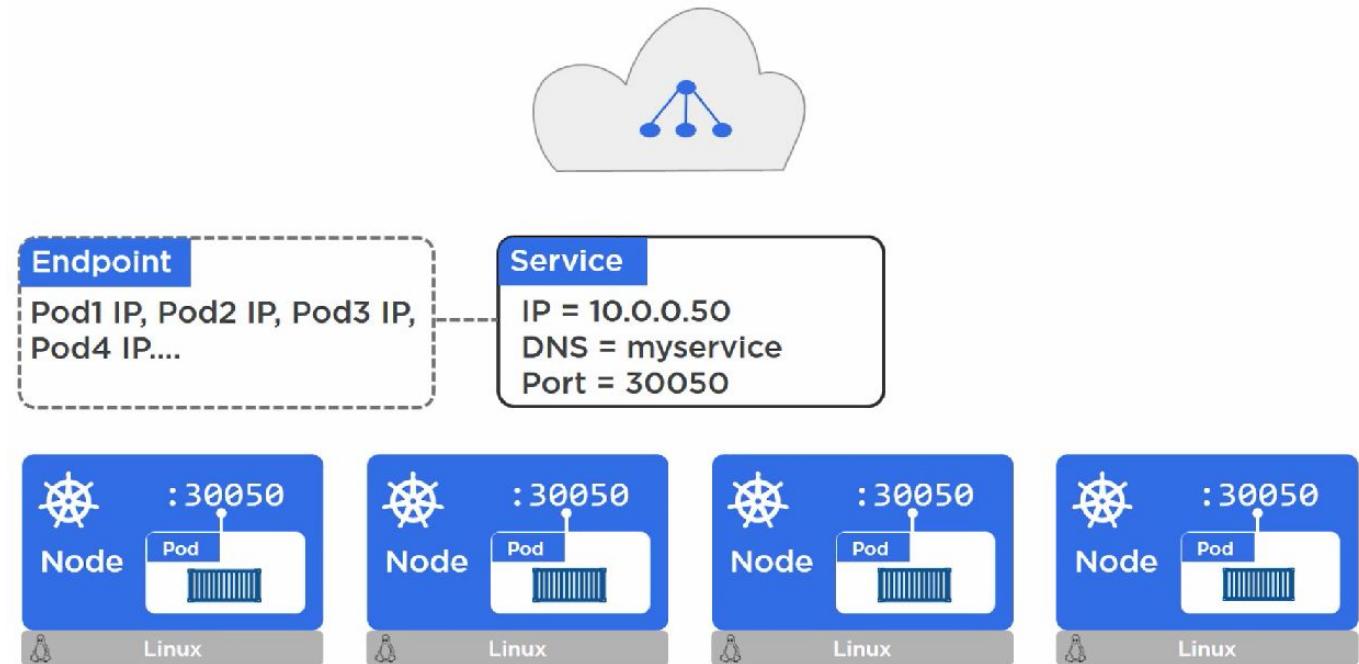
- C'est un ensemble logique de pods
- Il fournit une seule adresse IP et un nom DNS permettant d'accéder aux pods
- C'est un objet REST dont la définition est postée sur l'APIServer
- Il fournit un accès interne et externe aux pods



# Services

Le service maintient l'accès aux Pods en exposant une IP unique et les ports nécessaires à la communication avec les containers se trouvant dans les Pods.

Si un nouveau Pod est créé, ou si un Pod est « détruit », il met à jour sa liste de « endpoint » pour refléter l'état actuel des différents pods associés.



# Services

Comment le service peut-il reconnaître les pods ? Effectuer les opérations de load-balancing ? Rediriger les demandes vers ceux-ci ?

Grâces aux **Labels**.

Ces labels sont spécifiés dans le fichier yaml de configuration du pod.

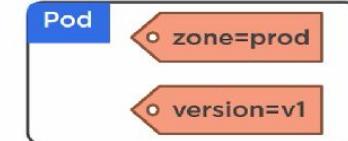
Grâce à eux, le service est capable de reconnaître les pods faisant un même travail.

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-pod
  labels:
    zone: prod
    version: v1
spec:
  containers:
    - name: hello-ctr
      image:
        nigelpoulton/pluralsight-docker-ci:latest
      ports:
        - containerPort: 8080
```

Service

Label selector:

zone=prod     version=v1



# Services

- Commandes Kubectl
  - kubectl apply -f service.yml  
Permet de créer le service à partir du fichier de description yaml
  - kubectl describe svc servicename  
Permet d'obtenir les informations sur le service
  - kubectl delete svc/servicename  
Permet de supprimer un service

# Services

## Définition d'un service

Dans notre fichier yml, nous définissons le type sur service

```
kind: Service
```

Nous lui donnons un nom pour le retrouver sur notre dashboard et l'auditer

```
metadata:  
  name: my-service
```

Dans les specs, nous définissons le selector( le label des pods qui devront être managé par le service)

```
selector:  
  app: MyApp
```

Mais également les informations sur la connectivité

- Port : exposé dans le cluster
- targetPort : port exposé par le container

```
ports:  
  - protocol: TCP  
    port: 80  
    targetPort: 9376
```

```
apiVersion: v1  
kind: Service  
metadata:  
  name: my-service  
spec:  
  selector:  
    app: MyApp  
  ports:  
    - protocol: TCP  
      port: 80  
      targetPort: 9376
```

# Services

## ■ Cas particulier : Service sans selectors

Les services abritent le plus souvent l'accès aux pods Kubernetes, mais ils peuvent également abstraire d'autres types de backends.

### Exemple :

- Un cluster de base de données externe en production, mais nos propres bases de données en dev.
- Pointer votre service vers un service dans un autre Namespace ou sur un autre cluster
- ...

Dans ce cas, nous devons créer manuellement notre endpoint car comme le service n'a pas de selector, il ne peut atteindre un pod.

# Services

Je mappe le port externe 80  
Vers le port 9376 du endpoint qui sera créé

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Déclaration d'un endpoint appelé **my-service**  
sur l'adresse ip **192.0.2.42**  
et le port **9376 (interne au container)**

```
apiVersion: v1
kind: Endpoints
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 192.0.2.42
    ports:
      - port: 9376
```

# Services

## ■ Les Types de services

Nous avons également la possibilité de définir le type de service qui nous intéresse :

- ClusterIP: Expose le service sur une IP interne au cluster. Il y a une résolution via le nom du service.(Défaut – Cfr slide précédent)
- NodePort: Expose le service sur l'IP de chaque nœud sur un port statique (le NodePort). Un service ClusterIP, vers lequel le service NodePort est automatiquement créé. Vous pourrez contacter le service NodePort, depuis l'extérieur du cluster, en demandant <NodeIP>: <NodePort>( Mapping)
- LoadBalancer: Expose le service en externe à l'aide de l'équilibrEUR de charge d'un fournisseur de cloud. Les services NodePort et ClusterIP, vers lesquels les itinéraires de l'équilibrEUR de charge externe, sont automatiquement créés.

# Services

- Exemple de service NodePort

```
$ more web-service.yml
apiVersion: v1
kind: Service
metadata:
  name: web-service
spec:
  selector:
    app: web
  type: NodePort ← Type du service
  ports:
    - port: 80 ← Port exposé dans le cluster
      targetPort: 80 ← Port cible d'un pod du groupe (Container)
      nodePort: 30000 ← Service accessible à l'extérieur du cluster
```

Attention : le nodeport doit être compris entre 30000 et 32767

# Services

Remarque :

Docker-desktop ne permet pas nativement avec kind l'accès aux ports prévus par les services

Nous devons dès lors prévoir une configuration lors de la création de notre cluster pour ajouter un mapping.

Il suffit de répéter la section  
-containerPort:....

Pour mapper d'autres ports sur notre control-plane(master)

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
    extraPortMappings:
      - containerPort: 30000
        hostPort: 30000
        protocol: TCP
      - role: worker
      - role: worker
```

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
    extraPortMappings:
      - containerPort: 30000
        hostPort: 30000
        protocol: TCP
      - containerPort: 30001
        hostPort: 30001
        protocol: TCP
      - role: worker
      - role: worker
```

# Command Line - Summary

```
# Afficher la liste des Services
kubectl get service [En option <SERVICE NAME>]
# Créer un Service depuis un template
kubectl create -f <template.yaml>
# Créer un Service sans template
kubectl expose deployment <DEPLOYMENT NAME>
  --name : nom du service
  --type : type du service
  --protocol : protocole à utiliser (TCP/UDP)
  --port : port utilisé par le service
  --target-port : port utilisé Target par le Pod
  --selector='clé=valeur': le sélecteur utilisé par service
# Supprimer un Service
kubectl delete service <SERVICE NAME>
```

# Command Line - Summary

```
# Appliquer des nouveaux changements à votre Service sans le détruire  
kubectl apply -f <template.yaml>  
# Modifier et appliquer les changements de votre Service instantanément sans le détruire  
kubectl edit service <SERVICE NAME>  
# Afficher les détails d'un Service kubectl describe service <SERVICE NAME>
```

# ReplicationController & ReplicaSets

Kubernetes



## ReplicationController



ReplicationController & ReplicaSets

# ReplicationController

Un **replicationController** est un objet K8S de haut niveau nous permettant de contrôler un ensemble d' « instances » de pods.

Si nous avons besoin d'avoir plusieurs **pods** basé sur la même **description** pour effectuer du LoadBalancing par exemple, nous pouvons utiliser un **replicationController**.

Nous définissons le nombre de Replicas désirés et grâce au *Reconciliation Loop*, le controller va garantir ce nombre de pods disponible :

- Si il y 'en a trop, il arrête les pods superflus
- Si il n'y en pas le nombre désiré, il crée les pods supplémentaires

```
#ajout d'un pod avec à partir de l'image Heroes-back-end avec replicas de 5
apiVersion: v1
kind: ReplicationController
metadata:
| name: heroes-back-end-rc
spec:
| replicas: 5
| selector:
| | app: heroes-back-end
| template:
| | metadata:
| | | labels:
| | | | app: heroes-back-end
| | spec:
| | | containers:
| | | | - name: heroes-back-end-pod
| | | | image: registry.gitlab.com/cogcoursedevops/kubernetes_heroes_app:latest
| | | | ports:
| | | | | - containerPort: 5000
```

# ReplicationController

Si j'applique ce fichier, j'obtiens donc 5 répliques de mon Pod contenant un container basé sur mon Image

```
mike@MIKEW10:~/CoursePod$ kubectl apply -f HeroesBackEndWithReplica.yml
replicationcontroller/heroes-back-end-rc created
mike@MIKEW10:~/CoursePod$ kubectl get pods
NAME           READY   STATUS            RESTARTS   AGE
heroes-back-end-rc-bt522  0/1    ContainerCreating  0          8s
heroes-back-end-rc-ks96s  0/1    ContainerCreating  0          8s
heroes-back-end-rc-psqkp  0/1    ContainerCreating  0          8s
heroes-back-end-rc-r5jrp  0/1    ContainerCreating  0          8s
heroes-back-end-rc-xcdkr  0/1    ContainerCreating  0          8s
mike@MIKEW10:~/CoursePod$
```

Le nom de base est *heroes-back-end-rc* suivit d'une suite de caractères définit par le *ReplicationController*

Si je modifie le nombre de répliques désirés et que je réapplique le fichier, le *Reconciliation Loop* va modifier le nombre de pods en vie pour arrivé à l'état désiré...

De même si un Pod tombe (erreur interne, ...), automatiquement, le Reconciliation Loop va en remettre un en route.

# Replication Controller

```
mike@MIKEW10:~/CoursePod$ kubectl describe rc
Name:           heroes-back-end-rc
Namespace:      default
Selector:       app=heroes-back-end
Labels:         app=heroes-back-end
Annotations:    <none>
Replicas:      5 current / 5 desired
Pods Status:   5 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=heroes-back-end
  Containers:
    heroes-back-end-pod:
      Image:      registry.gitlab.com/cogcoursedevops/kubernetes_heroes_app:latest
      Port:       5000/TCP
      Host Port:  0/TCP
      Environment: <none>
      Mounts:     <none>
      Volumes:    <none>
  Events:
    Type      Reason     Age      From               Message
    ----      ----     --      ----              -----
    Normal    SuccessfulCreate  5m46s   replication-controller  Created pod: heroes-back-end-rc-ks96s
    Normal    SuccessfulCreate  5m46s   replication-controller  Created pod: heroes-back-end-rc-xcdkr
    Normal    SuccessfulCreate  5m46s   replication-controller  Created pod: heroes-back-end-rc-psqkp
    Normal    SuccessfulCreate  5m45s   replication-controller  Created pod: heroes-back-end-rc-r5jrp
    Normal    SuccessfulCreate  5m45s   replication-controller  Created pod: heroes-back-end-rc-bt522
mike@MIKEW10:~/CoursePod$
```

Si je supprime le ReplicationController, je supprime tous les pods associés

```
mike@MIKEW10:~/CoursePod$ kubectl delete rc heroes-back-end-rc
replicationcontroller "heroes-back-end-rc" deleted
mike@MIKEW10:~/CoursePod$ kubectl get pods
NAME          READY   STATUS        RESTARTS   AGE
heroes-back-end-rc-bt522  1/1    Terminating   0          7m55s
heroes-back-end-rc-ks96s  1/1    Terminating   0          7m55s
heroes-back-end-rc-psqkp  1/1    Terminating   0          7m55s
heroes-back-end-rc-r5jrp  1/1    Terminating   0          7m55s
heroes-back-end-rc-xcdkr  1/1    Terminating   0          7m55s
```



## Replicaset



ReplicationController & ReplicaSets

# ReplicaSets

Les ReplicaSets sont des abstractions de plus haut niveau chargées de garantir qu'un nombre spécifié de copies exactes d'un pod donné sont en cours d'exécution...

## Quelle différence avec le ReplicationController ?

Pas grand-chose. Leur but est le même mais le ReplicaSet se base sur un champ *OwnerReference* pour savoir quels pods doivent être répliqué.

Concrètement , on définit un Replicaset via deux concepts : un selector et un pod template

```
apiVersion: app/v1
kind : ReplicaSet
metadata:
  name: heroes-back-end
  labels:
    app: backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: heroes-nodejs-backend
          image: registry.gitlab.com/cogcoursedevops/kubernetes_heroes_app:latest |
```

ReplicaSet est disponible dans la version app/v1 de K8S

Ces champs doivent matcher. Le selector permet d'identifier les pods qui doivent faire partie du ReplicaSet

# ReplicaSet

Si j'applique le fichier, j'obtiens alors 3 répliques de mon pods.

```
mike@MIKEW10: ~/Replicaset
mike@MIKEW10:~/Replicaset$ kubectl apply -f HeroesReplicaSet.yml
replicaset.apps/heroes-front-end created
mike@MIKEW10:~/Replicaset$ kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
heroes-front-end-k7ftk  1/1     Running   0          4m3s
heroes-front-end-lkr2k  1/1     Running   0          4m3s
heroes-front-end-x25m6  1/1     Running   0          4m3s
mike@MIKEW10:~/Replicaset$
```

```
mike@MIKEW10: ~/Replicaset
mike@MIKEW10:~/Replicaset$ kubectl get rs
NAME      DESIRED   CURRENT   READY   AGE
heroes-front-end  3         3         3       5m3s
mike@MIKEW10:~/Replicaset$
```

```
mike@MIKEW10: ~/Replicaset
mike@MIKEW10:~/Replicaset$ kubectl describe rs/heroes-front-end
Name:           heroes-front-end
Namespace:      default
Selector:       app=backend
Labels:         app=backend
Annotations:    <none>
Replicas:       3 current / 3 desired
Pods Status:   3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:        app=backend
  Containers:
    heroes-nodejs-backend:
      Image:        registry.gitlab.com/cogcoursesdevops/kubernetes_heroes_app:latest
      Port:         <none>
      Host Port:   <none>
      Environment: <none>
      Mounts:       <none>
      Volumes:     <none>
  Events:
    Type     Reason            Age   From               Message
    ----     ----            --   --                --
    Normal   SuccessfulCreate  8m6s  replicaset-controller  Created pod: heroes-front-end-lkr2k
    Normal   SuccessfulCreate  8m6s  replicaset-controller  Created pod: heroes-front-end-k7ftk
    Normal   SuccessfulCreate  8m6s  replicaset-controller  Created pod: heroes-front-end-x25m6
mike@MIKEW10:~/Replicaset$
```

# ReplicaSet

Si nous regardons le yaml des pods, nous verrons également que le *ownerReferences* contient les informations du ReplicaSet créé.

```
mike@MIKEW10: ~/Replicaset
mike@MIKEW10:~/Replicaset$ kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
heroes-front-end-k7ftk  1/1     Running   0          10m
heroes-front-end-lkr2k  1/1     Running   0          10m
heroes-front-end-x25m6  1/1     Running   0          10m
mike@MIKEW10:~/Replicaset$ kubectl get pods heroes-front-end-k7ftk -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2021-02-22T21:35:28Z"
  generateName: heroes-front-end-
  labels:
    app: backend
    name: heroes-front-end-k7ftk
    namespace: default
  ownerReferences:
    - apiVersion: apps/v1
      blockOwnerDeletion: true
      controller: true
      kind: ReplicaSet
      name: heroes-front-end
      uid: 50b9bb43-686f-45cb-a250-923b8638c4b5
  resourceVersion: "10610"
  selfLink: /api/v1/namespaces/default/pods/heroes-front-end-k7ftk
  uid: c5d52dbb-931c-47fc-86aa-00d0dfdaa45e
spec:
  containers:
    - image: registry.gitlab.com/cogcoursedevops/kubernetes_heroes_app:latest
      imagePullPolicy: Always
      name: heroes-nodejs-backend
```

# ReplicaSet

## Acquisition de pods

Imaginons qu'avant de créer le réplicaset, nous ayons déjà créé un pod avec le label ciblé par le selector.

## Kubectl apply -f HeroesAlone.yml

```
mike@MIKEW10:~/Replicaset$ kubectl get pods
No resources found in default namespace.
mike@MIKEW10:~/Replicaset$ kubectl apply -f HeroesBackEndAlone.yml
pod/heroes-alone created
mike@MIKEW10:~/Replicaset$ kubectl get pods
NAME      READY   STATUS        RESTARTS   AGE
heroes-alone  0/1   ContainerCreating   0          3s
mike@MIKEW10:~/Replicaset$
```

```
#ajout d'un pod avec à partir de l'image Heroes-back-end
apiVersion: v1
kind: Pod
metadata:
  name: heroes-alone
  labels:
    app: backend
spec:
  nodeName: kind-worker
  containers:
  - name: heroes-back-end-container
    image: registry.gitlab.com/cogcoursedevops/kubernetes_heroes_app:latest
    resources:
      limits:
        memory: 512Mi
        cpu: "1"
      requests:
        memory: 256Mi
        cpu: "0.2"
    ports:
    - containerPort: 5000
```

# ReplicaSet

Si maintenant, j'applique le fichier de définition de notre ReplicaSet, celui-ci va juste instancier les pods manquant et acquérir celui que nous venons de mettre en place.

De même, il suffit de changer le label de notre pods pour qu'il soit détaché du ReplicaSet et qu'un nouveau le remplace.

```
mike@MIKEW10: ~/Replicaset
mike@MIKEW10:~/Replicaset$ kubectl apply -f HeroesReplicaSet.yml
replicaset.apps/heroes-front-end created
mike@MIKEW10:~/Replicaset$ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
heroes-alone          1/1     Running   0          2m19s
heroes-front-end-gj87m 1/1     Running   0          11s
heroes-front-end-hbmkd 1/1     Running   0          11s
mike@MIKEW10:~/Replicaset$
```

```
mike@MIKEW10: ~/Replicaset
mike@MIKEW10:~/Replicaset$ kubectl apply -f HeroesBackAloneDetach.yml
pod/heroes-alone configured
mike@MIKEW10:~/Replicaset$ kubectl describe po/heroes-alone | grep Labels
Labels:      app=backend-alone
mike@MIKEW10:~/Replicaset$ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
heroes-alone          1/1     Running   0          6m46s
heroes-front-end-8dpbf 1/1     Running   0          58s
heroes-front-end-gj87m 1/1     Running   0          4m38s
heroes-front-end-hbmkd 1/1     Running   0          4m38s
mike@MIKEW10:~/Replicaset$
```

# Commande Line - Summary

```
# Afficher la liste des Replicasets
kubectl get replicaset [En option <REPLICASET NAME>]
    --template : récupérer des informations précises de la sortie de la commande
        Ex: Récupérer le nombre de réplique : kubectl get rs <REPLICASET NAME> --template={{.status.replicas}}
# Créer un ReplicaSet
kubectl create -f <template.yaml>
# Supprimer un ReplicaSet
kubectl delete replicaset <REPLICASET NAME>
# Appliquer des nouveaux changements à votre ReplicaSet sans le détruire
kubectl apply -f <template.yaml>
# Modifier et appliquer les changements de votre ReplicaSet instantanément sans le détruire
kubectl edit rs <REPLICASET NAME>
# Afficher les détails d'un replicaset
kubectl describe replicaset <REPLICASET NAME>
```

# Exercice

- Mettez en place l'architecture suivante :
  - Un pod sur Kind-worker contenant un container avec Drupal
  - Un pod sur Kind-worker2 contant un container avec Mysql 5.7
  - Créer un secret pour stocker le mot de passe de Mysql

```
#utiliser echo -n 'votre password' | base64 pour générer la valeur à mettre dans le fichier yaml suivant)
apiVersion: v1
kind: Secret
metadata:
  name: mysql-pass
type: Opaque
data:
  password: YWRtaW4=
```

- Installer Drupal via votre navigateur grâce à un port-forward
- Exposer votre site Drupal sur le port 30001
- (Bonus) créer un replicaset pour mysql

# Deployment

Kubernetes

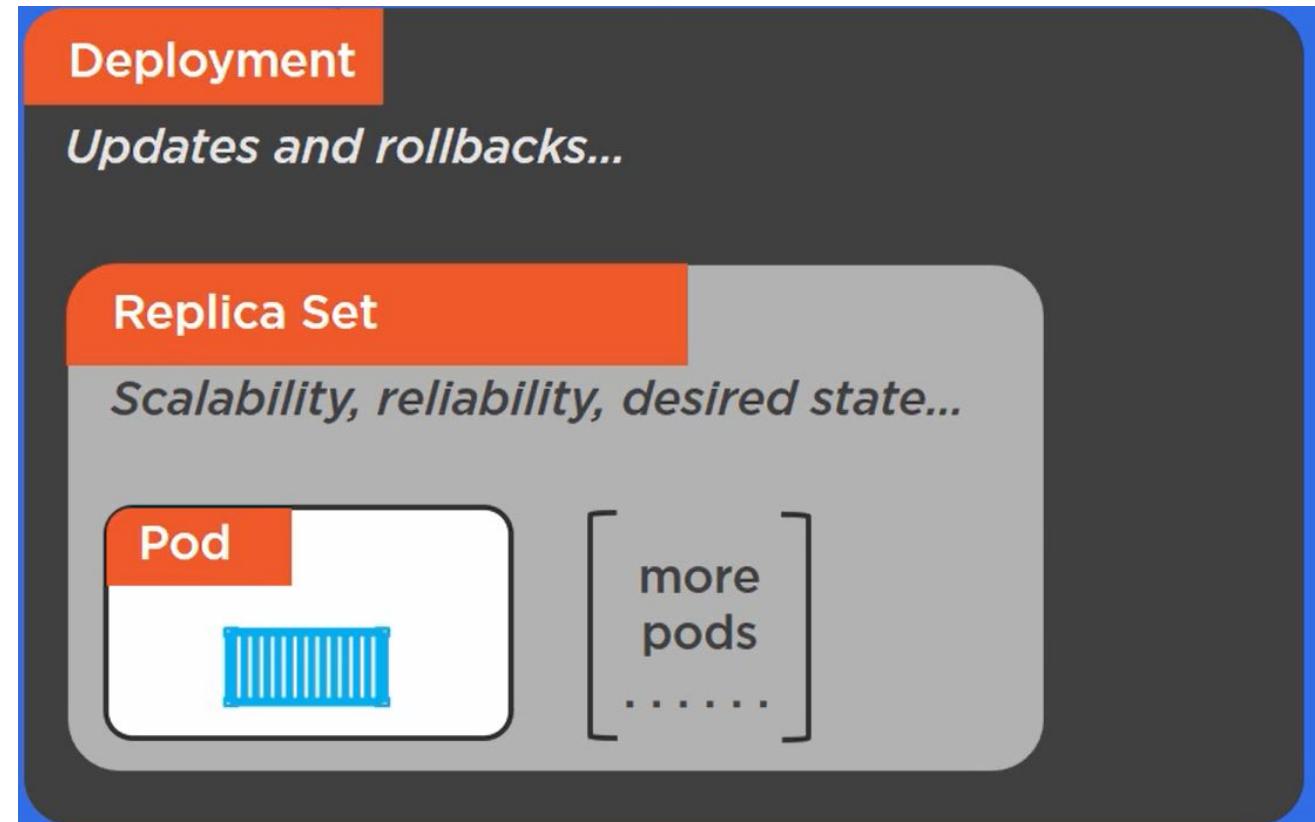
# Deployment

Un *Deployment* (déploiement en français) fournit des mises à jour déclaratives pour Pods et Replicaset.

Vous décrivez un *état désiré* dans un déploiement et le contrôleur de déploiement change l'état réel à l'état souhaité à un rythme contrôlé.

Vous pouvez définir des Deployments pour créer de nouveaux ReplicaSets, ou pour supprimer des déploiements existants et adopter toutes leurs ressources avec de nouveaux déploiements.

Vous l'avez compris c'est notre sésame pour mettre en place notre environnement de développement sous K8S.



# Deployment

Le but « opérationnel » du *Deployment* est de nous aider à maintenir nos Pods de manière simple.

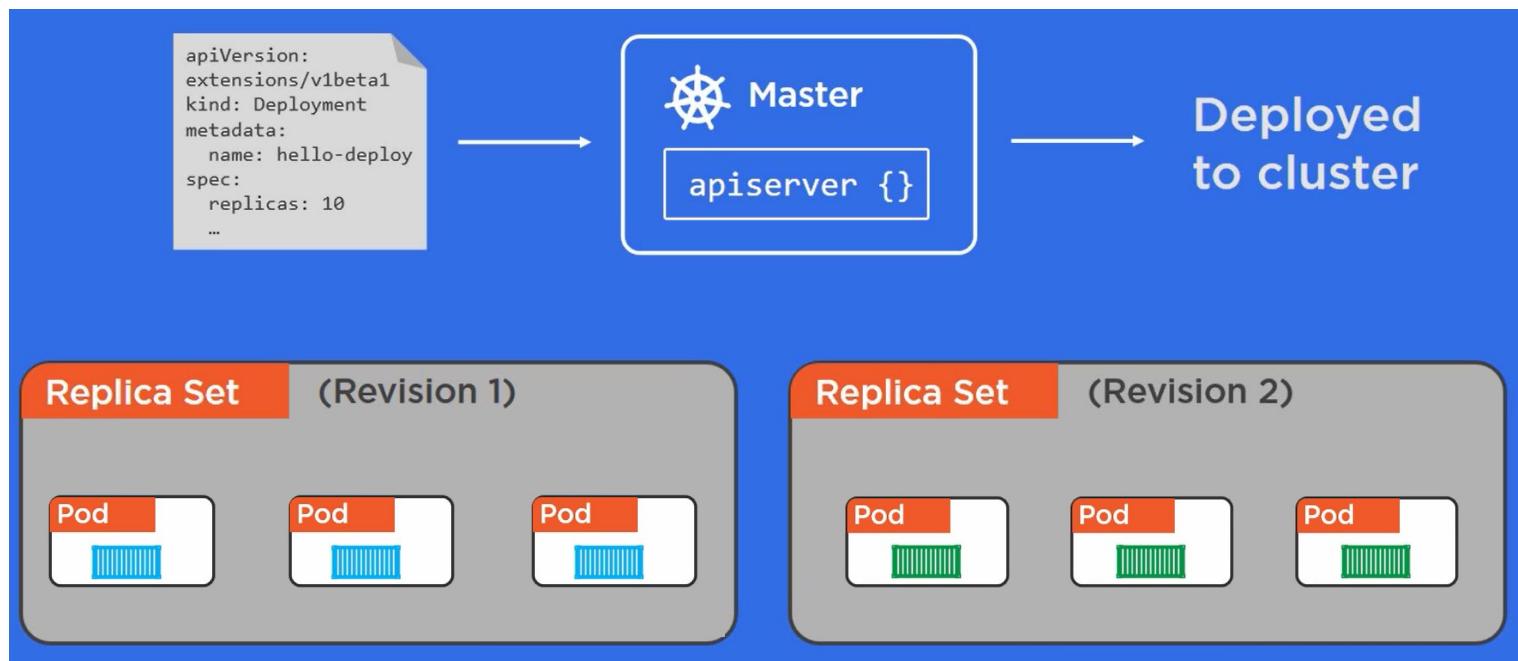
En effet, si nous désirons mettre à jour un Replicaset sans le deployment, nous devons créer un nouveau réplicaSet et utiliser la commande **kubectl rolling-update – f updated-rc.yml**



# Deployment

Via un deployment, c'est le controller recevant le fichier yaml via l'api server qui se charge de l'Update et du Rollback...

En effet, lorsqu'une mise à jour est effectuée, le principe est le même qu'avec les ReplicaSet **MAIS** les anciennes versions des pods sont « désactivés » et non supprimées. Le rollback est donc possible.



# Deployment

En bref

1. Un déploiement peut planifier plusieurs pods. Un pod en tant qu'unité ne peut pas évoluer par lui-même.
2. Un déploiement représente un objectif unique avec un groupe de POD.
3. Un seul POD peut avoir plusieurs conteneurs et ces conteneurs à l'intérieur d'un seul POD partagent la même adresse IP et peuvent se parler en utilisant l'adresse localhost.
4. Pour accéder à un Déploiement avec un ou plusieurs POD, vous avez besoin d'un point de endpoint Kubernetes Service mappé au déploiement à l'aide d'étiquettes et de sélecteurs.

# Manifest -Yaml

## 1. apiVersion

Version de K8s.. Il est possible d'avoir la liste des objets via `http://localhost:8001` en utilisant un kubectl proxy.

## 2. Kind

Type d'objet/ressource à créer qui changent suivant la version (apiversion). Exemple pour la version 1.16 □

<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.16/>

## 3. Metadata

Données identifiant l'objet. Les valeurs possibles sont :

**1.Labels** : Paires clé-valeur principalement utilisées pour grouper et catégoriser l'objet de déploiement

**2.Nom:** Il représente le nom du déploiement à créer.

**3.Espace:** Nom de l'espace de noms dans lequel vous souhaitez créer le déploiement.

**4.Annotation:** paires clé-valeur comme labels , cependant, utilisé pour différents cas d'utilisation. Vous pouvez ajouter n'importe quelle information aux annotations. Par exemple, vous pouvez avoir une annotation comme "monitoring" : "true et les sources externes pourront trouver tous les objets avec cela(exemple pour Prometheus =>

<https://medium.com/@akilblanchard09/monitoring-a-kubernetes-cluster-using-prometheus-and-grafana-8e0f21805ea9>

# Manifest -Yaml

## 4. Spec

Etat souhaité et caractéristiques de l'objet. Exemple, pour un déploiement : nombre de réplicas, nom de l'image, etc..

Il y a 3 champs importants :

- Replicas : nombre de pods en cours d'exécution
- Selector : Labels pour les pods à déployer et manager
- Template : Il a ses propres données et spec (contient l'image à containeriser)



## Création d'un Déploiement



Deployment

# Deployment

Comme pour les nodes, pods, services, replicaSet,... nous allons créer un fichier yml contenant les informations nécessaires au déploiement.

```
apiVersion : app/v1
kind : Deployment
metadata :
  name: heroes-deploy
```

Cette partie descriptive permet de définir un déploiement appelé ***heroes-deploy***

La façon la plus simple de créer un déploiement est d'utiliser Kubectl qui permet, en spécifiant un nom et une image, de créer le fichier yaml.

```
kubectl create deployment heroes-deploy -o yaml --image registry.gitlab.com/cogcoursedevops/kubernetes_heroes_app:latest
```

(vous pouvez retrouver la syntaxe : <https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#-em-deployment-em->)

# Deployment

Nous allons plutôt créer notre fichier de manière simple :

Nous créons ensuite le déploiement via **kubectl create -f ...**

```
mike@MIKEW10: ~/Deployements$ cat heroesDeploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: heroes-deploy
spec:
  replicas: 5
  selector:
    matchLabels:
      app: heroes-backend
  template:
    metadata:
      labels:
        app: heroes-backend
    spec:
      containers:
        - name: heroes-backend-pod
          image: registry.gitlab.com/cogcoursesdevops/kubernetes_heroes_app:latest
          resources:
            limits:
              memory: 512Mi
              cpu: "1"
            requests:
              memory: 256Mi
              cpu: "0.2"
          ports:
            - containerPort: 5000
mike@MIKEW10:~/Deployements$ kubectl create -f heroesDeploy.yml
deployment.apps/heroes-deploy created
mike@MIKEW10:~/Deployements$
```

# Deployment

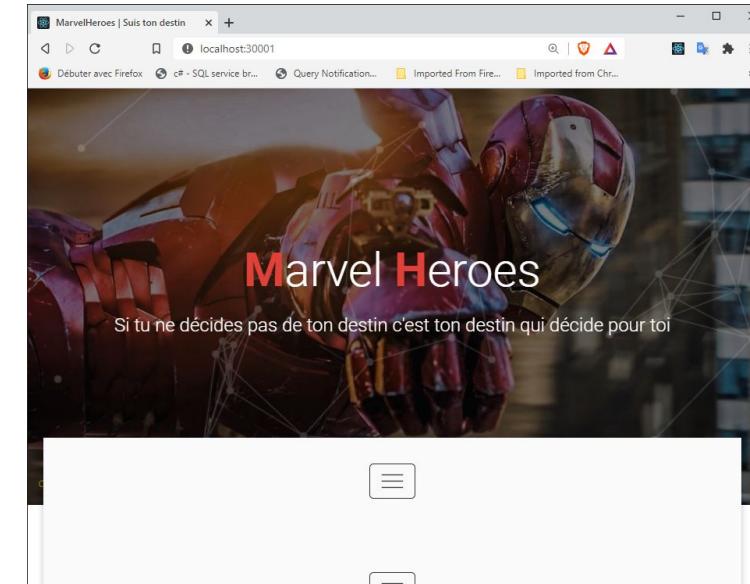
Nous pouvons ensuite vérifier la bonne mise en place via la commande **kubectl get deploy -o wide**

```
mike@MIKEW10:~$ kubectl get deploy -o wide
NAME           READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES
heroes-deploy   5/5     5            5           5h18m   heroes-backend-pod   registry.gitlab.com/cogcoursedevops/kubernetes_heroes_app:latest   app=heroes-backend
```

Nous avons bien nos 5 répliques qui sont disponibles..

En ajoutant un service, nous pouvons donc maintenant accéder de l'extérieur à notre déploiement

```
apiVersion: v1
kind: Service
metadata:
  name: deploy-heroes-backend-svc
spec:
  selector:
    app: heroes-backend
  type: NodePort
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
      nodePort: 30001
```





## Mise à jour, Historique et RollBack



Deployment

# Mise à jour, Historique et Rollback

L'avantage du déploiement c'est que nous pouvons facilement l'éditer et le faire ainsi évoluer suivant nos besoins.

Par exemple, nous pourrions avoir besoin de changer une image de notre container pour une version plus à jour ou différente.

Il suffit d'éditer le fichier de déploiement :

```
① Sélection mike@MIKEW10: ~/CoursDeploy
apiVersion: apps/v1
kind: Deployment
metadata:
  name: heroes-deploy
spec:
  replicas: 5
  selector:
    matchLabels:
      app: heroes-backend
  template:
    metadata:
      labels:
        app: heroes-backend
    spec:
      containers:
        - name: heroes-backend-pod
          image: registry.gitlab.com/cogcoursesdevops/kubernetes heroes app:k8s
          resources:
            limits:
              memory: 512Mi
              cpu: "1"
            requests:
              memory: 256Mi
              cpu: "0.2"
          ports:
            - containerPort: 5000
```

# Mise à jour, Historique et Rollback

Nous devons encore choisir une stratégie de mise à jour :

- **Recreate**
- **RollingUpdate**
- **Blue/Green (non vu dans ce cours)**
- **Canary (non vu dans ce cours)**
- **A/B testing (non vu dans ce cours)**

Pour les stratégies non vues : <https://ludovicwyffels.dev/k8s-deployment-strategies/>

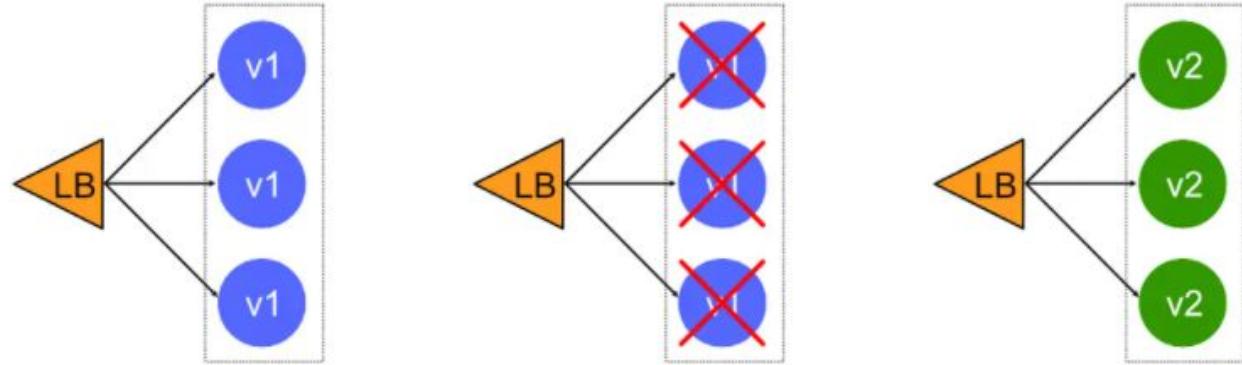
# Mise à jour, Historique et Rollback

## Recreate (Dev)

Un déploiement défini avec une stratégie de type **recreate** mettra fin à toutes les instances en cours d'exécution, puis les recréera avec la version la plus récente.

Dans notre fichier yaml :

```
spec:  
  replicas: 5  
  strategy:  
    type: Recreate
```



### Avantages:

État d'application entièrement renouvelé

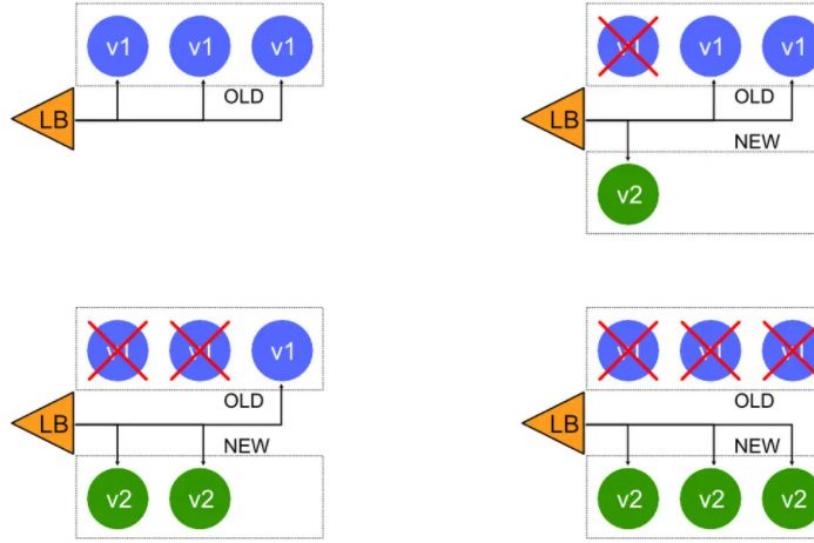
### Inconvénients:

Temps d'arrêt qui dépend à la fois de la durée d'arrêt et du démarrage de l'application

# Mise à jour, Historique et Rollback

## RollingUpdate (progressif)

Ce déploiement met à jour les pods de façon progressive. Un **ReplicaSet** secondaire est créé avec la nouvelle version de l'application, puis le nombre de répliques de l'ancienne version est réduit et la nouvelle version est augmentée jusqu'à ce que le nombre correct de répliques soit atteint..



### Avantages:

la version est lentement publiée sur toutes les instances

### Inconvénients:

le déploiement/la restauration peut prendre du temps

la prise en charge de plusieurs API est difficile

# Mise à jour, Historique et Rollback

Dans notre fichier yaml □

## Remarque:

Si vous déclenchez un déploiement alors qu'un déploiement existant est en cours, L'existant sera mis en pause et passera à une nouvelle version en remplaçant le déploiement Initial

## !!Attention!!

Si vous utilisez dans votre déploiement, une image avec le tag:latest, vous devrez détruire votre déploiement pour le recréer avec un autre tag

```
1  apiVersion: v1
2  kind: Deployment
3  metadata:
4    name: heroes-deploy
5  spec:
6    replicas: 10
7    minReadySeconds: 10
8    strategy:
9      rollingUpdate:
10     maxUnavailable: 2
11     maxSurge: 1
12     type: RollingUpdate
```

**maxUnavailable : nombre max. de Pod pouvant être "non disponible" durant la mise à jour**

**maxSurge : nombre max. de Pod pouvant être créés en plus du nombre actuel de répliques**

# Historique

Chaque mise à jour est sauvegardée dans un historique que nous pouvons consulter via la commande **kubectl rollout history [notre déploiement]**

```
Exemple : deployments "nginx-deployment"
REVISION  CHANGE-CAUSE
1  kubectl create -f nginx-pod.yaml
2  kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
```

Nous pouvons donc facilement revenir à une version précédente via la commande  
**kubectl rollout undo [notre déploiement] –to-revision=[num révision]**

```
Exemple :
kubectl rollout undo deployment/nginx-deployment --to-revision=1

kubectl describe deploy nginx-deployment | grep Image
```

# Commande Line - Summary

```
# Afficher la liste des Deployments
kubectl get deployment [En option <DEPLOYMENT NAME>] -o wide : récupérer en plus, le nom de l'image et le
sélecteur
# Créer un Deployment
kubectl create -f <template.yaml>
# Supprimer un Deployment
kubectl delete deployment <DEPLOYMENT NAME>
# Appliquer des nouveaux changements à votre Deployment sans le détruire
kubectl apply -f <template.yaml>
# Modifier et appliquer les changements de votre Deployment instantanément sans le détruire
kubectl edit deployment <DEPLOYMENT NAME>
# Afficher les détails d'un Deployment
kubectl describe deployment <DEPLOYMENT NAME>
# Mettre à jour l'image des Pods de votre Deployment
kubectl set image deployment/<DEPLOYMENT NAME> <CONTAINER NAME>=<NEW IMAGE NAME>
```

# Commande Line - Summary

```
# Afficher le statuts du rolling update de votre Deployment  
kubectl rollout status deployment/<DEPLOYMENT NAME>  
# Afficher l'historique des révisions de votre Deployment  
kubectl rollout history deployment/<DEPLOYMENT NAME>  
# Revenir à une version précédente  
kubectl rollout undo deployment/nginx-deployment --to-revision=<REVISION NUMBER>
```

# Exercice

- Mettre en place un déploiement pour notre application client/server « simpleApi »
- Ce déploiement devra prévoir 5 réplicas pour la partie serveur
- La partie cliente devra être accessible de l'extérieur (navigateur ) via l'adresse <http://127.0.0.1:30001>
- Prévoyez ensuite une version 2 de la partie serveur en y ajoutant 5 events dans le fichier json
  - Créez l'image docker
  - Uploader l'image docker sur votre dockerhub
- Effectuer un update de votre déploiement pour augmenter le nombre de réplicas(10) et changer l'image du container
- Lister l'historique
- Effectuez un rollout sur la version initiale

**Remarque:** Vérifiez chaque étape via les commandes kubectl appropriées.

# Stockages

Volumes et Volumes persistants

# Volumes

Les pods sont éphémères et « Isolés » !!!

Les Volumes répondent à deux besoins :

- Si un conteneur plante     Perte des données, fichiers,... Je dois pouvoir garantir la récupération des fichiers
- Si je dois partager des fichiers entre pods ? Je dois pouvoir définir un espace de stockage commun pour l'échange

## Ephemeral resource



# Volumes

## DOCKER

« Les volumes sont des répertoires qui sont stockés en dehors du système de fichiers du conteneur et qui contiennent des données réutilisables et partageables qui peuvent persister même lorsque les conteneurs sont arrêtés. Ces données peuvent être réutilisées par le même service lors du redéploiement ou partagées avec d'autres services. »

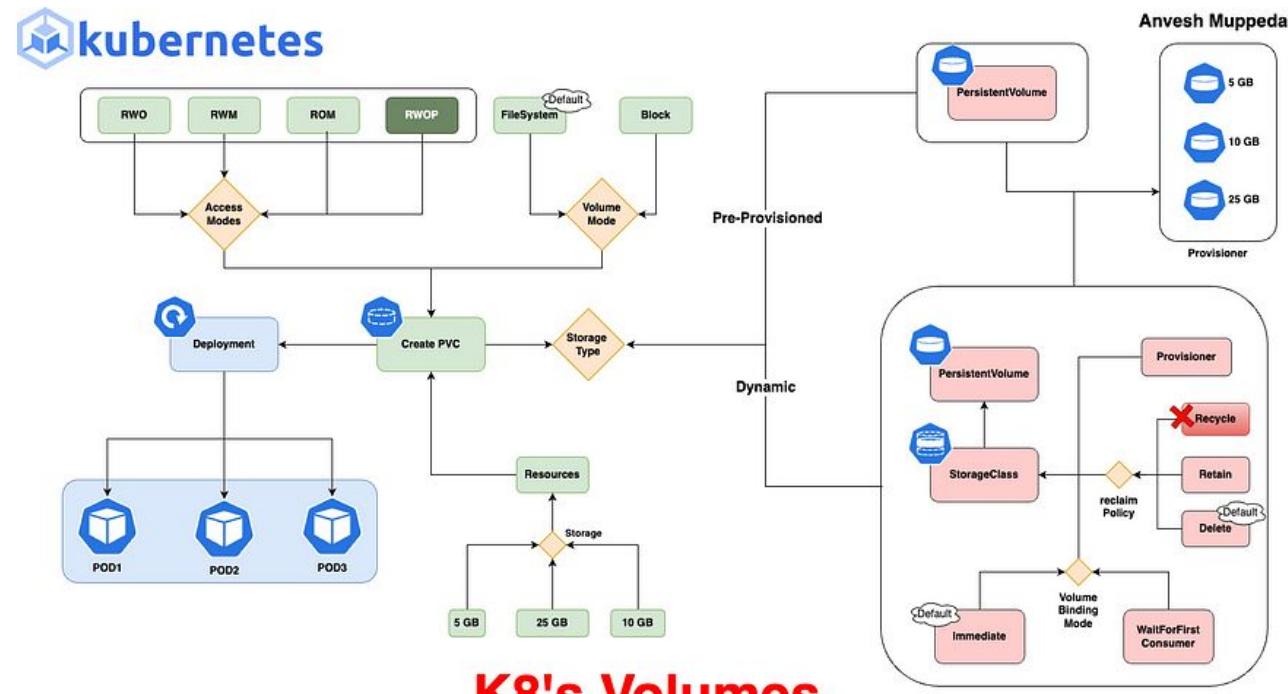
## Kubernetes

« À la base, un volume est juste un répertoire, éventuellement avec des données, qui est accessible aux conteneurs dans un pod. La façon dont ce répertoire est créé, le support qui le soutient et son contenu sont déterminés par le type de volume particulier utilisé. »

# Volumes

K8S propose un large choix de volumes (<https://kubernetes.io/docs/concepts/storage/volumes/>) répondant aux besoins spécifiques des hébergement de nos cluster( Azure, AWS, local,...)

Nous ne ferons pas le tour de tous les types de volumes dans ce cours mais nous allons voir ensemble les concepts et des exemples pour certains types



# Volumes

## ■ Volumes Persistants (PV)

**Définition:** Les volumes persistants sont des ressources de stockage provisionnées dans un cluster Kubernetes, gérées par l'administrateur du cluster. Ils fournissent une couche d'abstraction sur les périphériques de stockage physiques, permettant aux utilisateurs d'accéder au stockage sans avoir besoin de connaître les détails de l'infrastructure sous-jacente.

### Options Disponibles:

- **Capacity:** Spécifie la taille du volume.
- **Access Mode:** Définit comment le volume peut être consulté. Ci-dessous les options disponibles.
  1. **ReadWriteOnce:** Peut être monté en lecture-écriture par un seul nœud.
  2. **ReadOnlyMany:** Peut être monté en lecture seule par plusieurs nœuds.
  3. **ReadWriteMany:** Peut être monté en lecture-écriture par plusieurs nœuds.
- **Storage Class Name :** Associe le PV à une Classe de Stockage spécifique.
- **Volume Mode:** Détermine si le volume est monté en tant que filesystem ou block dispositif.
- **Persistent Volume Reclaim Policy:** Détermine ce qui arrive aux ressources PV's une fois publiées.
- **mount options:** Options supplémentaires à passer à la commande mount.

# Volume

## ■ Exemple

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-local-pv
spec:
  capacity:
    storage: 500Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: my-local-storage
  local:
    path: /mnt/disk/vol1
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - node1
```

# Volume

## ■ Persistent Volume Claims (PVC)

**Définition:** Les réclamations de volume persistantes sont des demandes de stockage effectuées par des utilisateurs ou des applications s'exécutant dans le cluster. Ils permettent aux utilisateurs de consommer du stockage sans avoir besoin de connaître les spécificités de sa mise en service.

### Options Disponibles:

- **Capacity:** Spécifie la taille minimale du volume.
- **Access Modes:** Demande le mode d'accès requis.
  1. **ReadWriteOnce:** Peut être monté en lecture-écriture par un seul noeud.
  2. **ReadOnlyMany:** Peut être monté en lecture seule par plusieurs nœuds.
  3. **ReadWriteMany:** Peut être monté en lecture-écriture par plusieurs nœuds.
- **Storage Class Name:** Demande une Classe de Stockage spécifique.
- **Volume Mode:** Détermine si le volume est monté en tant que filesystem ou block. Filesystem est le mode par défaut utilisé lorsque le paramètre volumeMode est omis.

# Volume

## ■ Exemple

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-local-pv
spec:
  capacity:
    storage: 500Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: my-local-storage
  local:
    path: /mnt/disk/vol1
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - node1
```

# Volumes

- Pour plus d'informations : <https://medium.com/@muppedaanvesh/a-hand-on-guide-to-kubernetes-volumes-%EF%B8%8F-b59d4d4e347f>

# Ingress

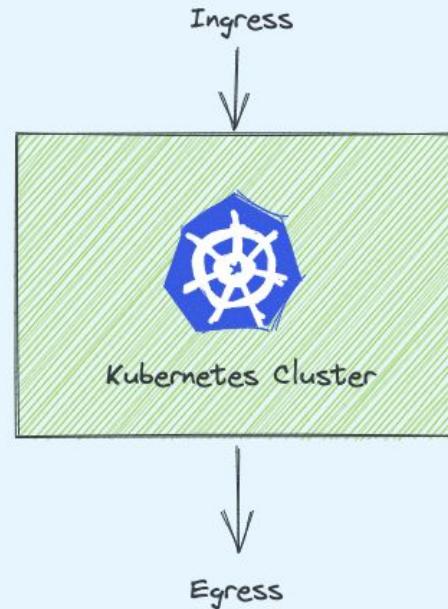
Traffic entrant

# Introduction à Kubernetes Ingress

- Le sens littéral: Ingress fait référence à l'acte d'entrer.
- Il en est de même dans le monde de Kubernetes :  
Ingress signifie que le trafic entre dans le cluster  
Egress est le trafic qui sort du cluster.

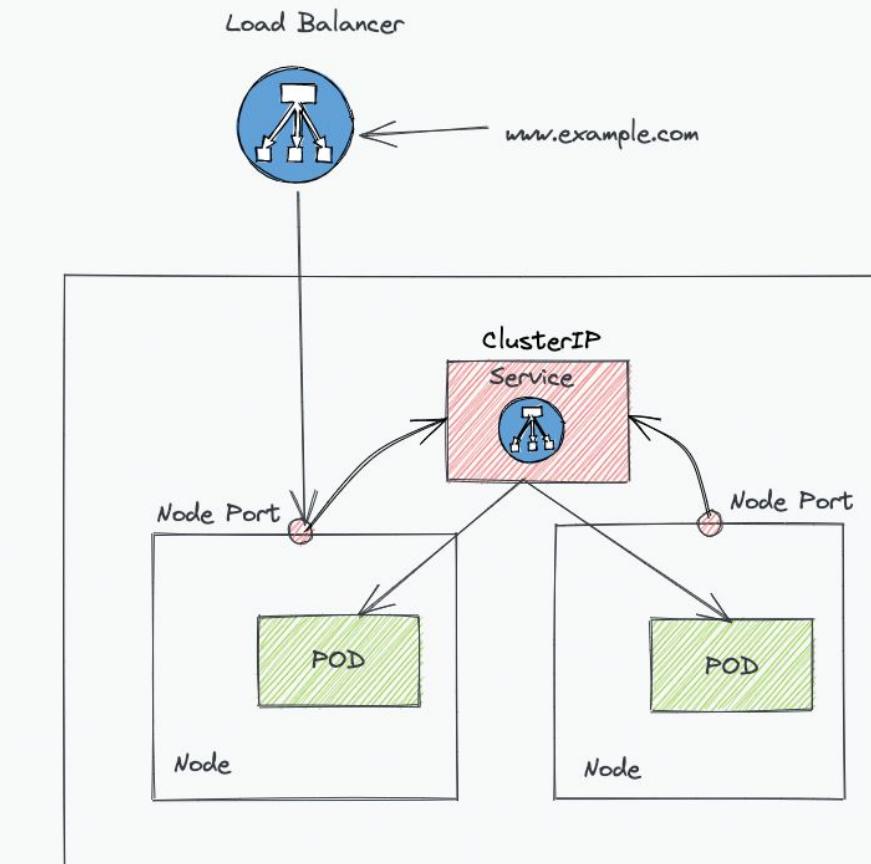
Ingress est natif à Kubernetes comme les pods, les déploiements, etc.

En utilisant Ingress , vous pouvez **maintenir les configurations de routage DNS**.



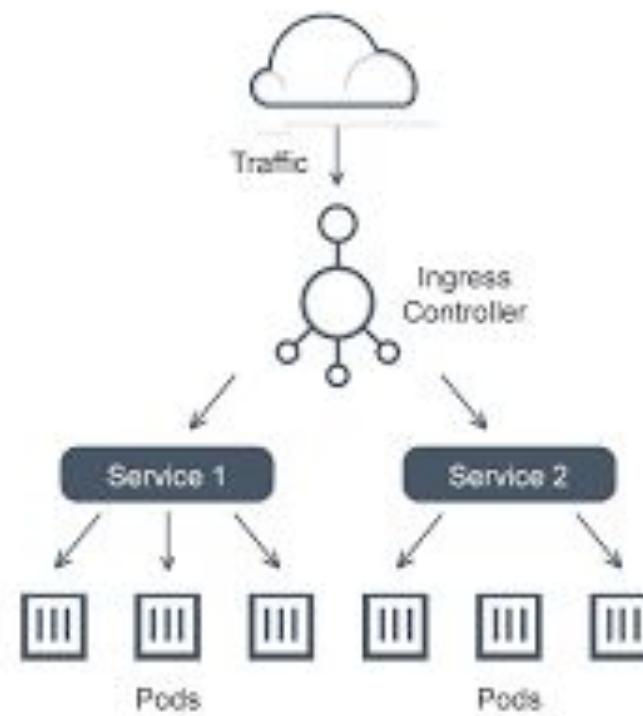
# Introduction à Kubernetes Ingress

Sans Ingress, nous devons utiliser un service de Type **Loadbalancer** au déploiement



# Introduction à Kubernetes Ingress

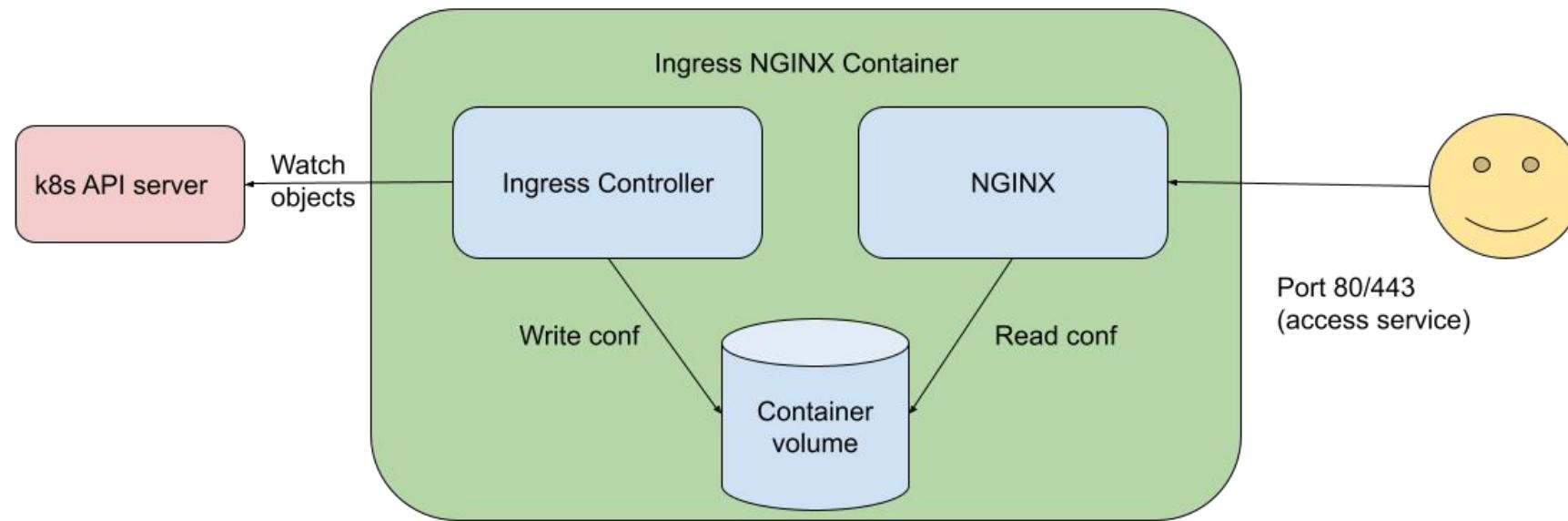
Avec Ingress, nous avons une couche proxy (Ingress Controller) entre le **LoadBalancer** et le endpoint du service K8S.



# Comment fonctionne Kubernetes Ingress ?

Deux concepts clés permettent de comprendre le fonctionnement de Ingress :

- Kubernetes Ingress Ressource : Stocker les règles DNS du cluster
- Kubernetes Ingress Controller : Responsable du « routing » grâce aux règles DNS



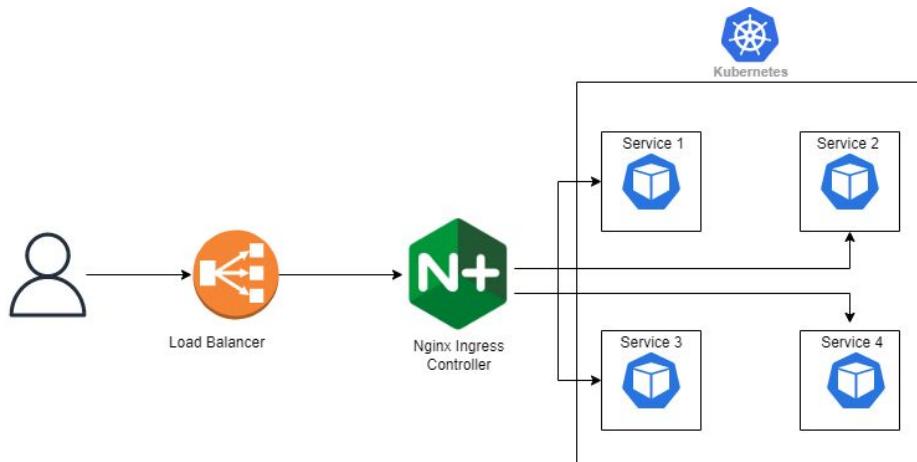
# Ingress Ressource

La ressource Kubernetes Ingress est une ressource native kubernetes où vous spécifiez les règles de routage DNS.

Cela signifie que vous mappez le trafic DNS externe aux points de terminaison internes du service Kubernetes.

Remarque :

- Une entrée nécessite un controller Ingress pour acheminer le trafic
- Le trafic externe **n'atteint pas** l'API d'Ingress mais le endpoint du Ingress Controller configuré directement avec le **LoadBalancer**



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dev-ingress
  namespace: dev
spec:
  rules:
    - host: dev.apps.cognitic.be
      http:
        paths:
          - backend:
              serviceName: hello-service
              servicePort: 80
```

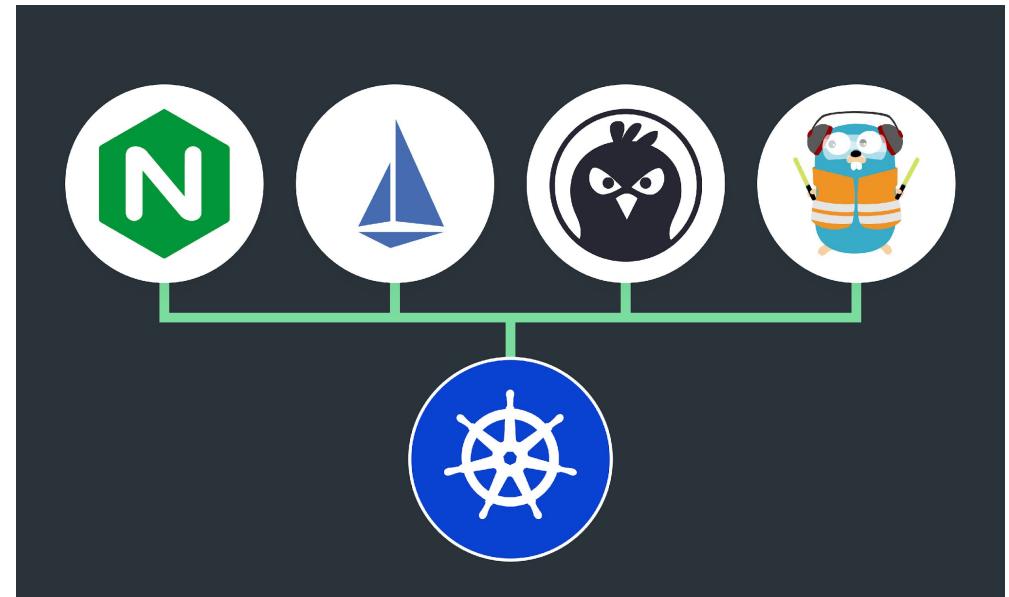
# Ingress Controller

!!! Non natif à K8S : Il faut donc le configurer !!!

Il s'agit en fait d'un reverse proxy déployer via un objet **deployment** exposé en tant que service **LoadBalancer**.

Il existe plusieurs projets open-source et commerciaux de Ingress Controller :

- NGINX (<https://docs.nginx.com/nginx-ingress-controller/>)
- Skipper (<https://opensource.zalando.com/skipper/>)
- HAProxy (<https://www.haproxy.com/documentation/kubernetes-ingress/>)
- Traefik (<https://doc.traefik.io/traefik/providers/kubernetes-ingress/>)
- ...



# Fonctionnement du contrôleur Nginx

1. `nginx.conf` à l'intérieur du pod de contrôleur Nginx est un modèle de LUA permettant de discuter avec Api Ingress et obtenir les dernières valeurs pour le routage du trafic en temps réel. Le template est disponible sur :

<https://github.com/kubernetes/ingress-nginx/blob/main/rootfs/etc/nginx/template/nginx.tmpl>

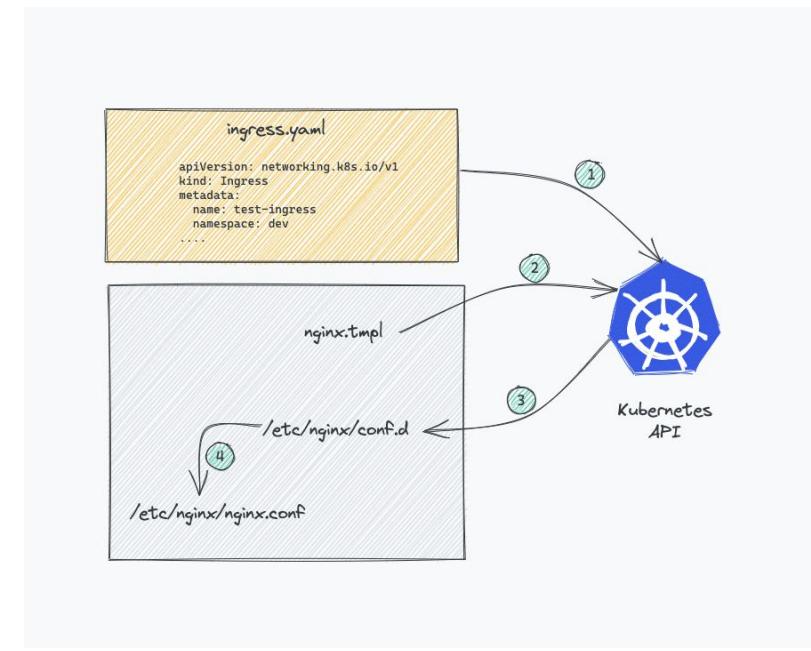
2. Le contrôleur Nginx échange avec l'api Ingress pour vérifier s'il existe une règle créée pour le routage du trafic.

3. S'il trouve des règles d'entrée, le contrôleur Nginx génère une configuration de routage à l'emplacement `/etc/nginx/conf.d` dans chaque pod nginx.

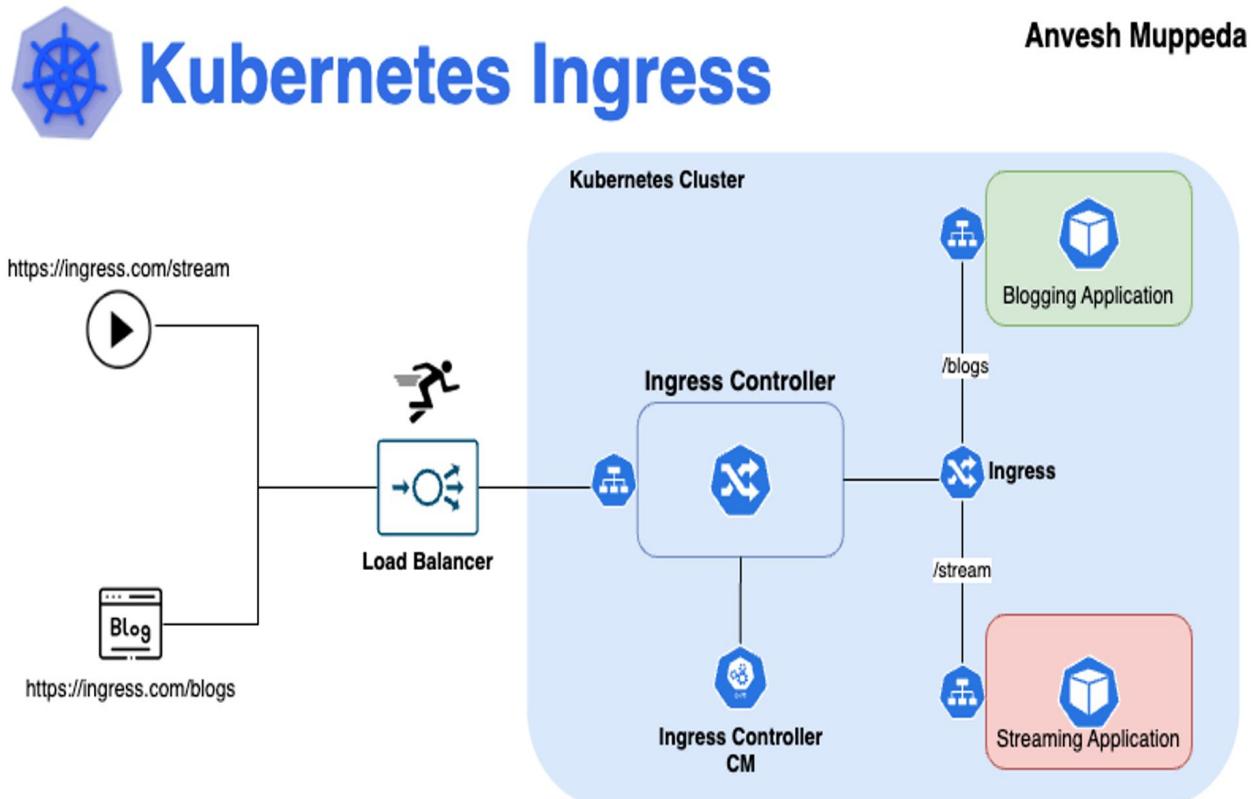
4. Pour chaque ressource d'entrée que vous créez, Nginx génère une configuration interne `/etc/nginx/conf.d`.

5. Le fichier principal `/etc/nginx/nginx.conf` contient toutes les configurations de `etc/nginx/conf.d`.

6. Si vous mettez à jour l'objet Ingress , la configuration Nginx est mise à jour à nouveau et il effectue un rechargeement de la configuration.



# Fonctionnement du contrôleur Nginx



# Installation et Configuration Nginx Ingress

Nous allons utiliser la version Community : <https://github.com/kubernetes/ingress-nginx>

Nous devons savoir déjà mis en place :

- Un cluster K8S
- Kubectl
- Un accès administrateur
- Un domaine valide (optional)



# Installation et Configuration Nginx Ingress

- Création d'un espace de nom pour notre Ingress
  - ≥ `kubectl create ns ingress-nginx`
- Création des accès (ServiceAccount, Roles,...)
  - Service Account
  - Rôle
  - RolesBinding
  - ClusterRole
  - ClusterRoleBinding

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  name: ingress-nginx-admission
  namespace: ingress-nginx
```

# Installation et Configuration Nginx Ingress

- Création d'un espace de nom pour notre Ingress
  - ≥ `kubectl create ns ingress-nginx`
- Création des accès (ServiceAccount, Roles,...)
  - Service Account
  - Rôle
  - RolesBinding
  - ClusterRole
  - ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  annotations:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  name: ingress-nginx-admission
  namespace: ingress-nginx
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  verbs:
  - get
  - create
```

# Installation et Configuration Nginx Ingress

- Création d'un espace de nom pour notre Ingress
  - ≥ `kubectl create ns ingress-nginx`
- Création des accès (ServiceAccount, Roles,...)
  - Service Account
  - Rôle
  - RolesBinding
  - ClusterRole
  - ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    name: ingress-nginx-admission
    namespace: ingress-nginx
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role
    name: ingress-nginx-admission
  subjects:
  - kind: ServiceAccount
    name: ingress-nginx-admission
    namespace: ingress-nginx
```

# Installation et Configuration Nginx Ingress

- Création d'un espace de nom pour notre Ingress
  - ≥ `kubectl create ns ingress-nginx`
- Création des accès (ServiceAccount, Roles,...)
  - Service Account
  - Rôle
  - RolesBinding
  - ClusterRole
  - ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    name: ingress-nginx-admission
  rules:
    - apiGroups:
        - admissionregistration.k8s.io
      resources:
        - validatingwebhookconfigurations
      verbs:
        - get
        - update
```

# Installation et Configuration Nginx Ingress

- Création d'un espace de nom pour notre Ingress
  - ≥ `kubectl create ns ingress-nginx`
- Création des accès (ServiceAccount, Roles,...)
  - Service Account
  - Rôle
  - RolesBinding
  - ClusterRole
  - ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    name: ingress-nginx-admission
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: ingress-nginx-admission
  subjects:
  - kind: ServiceAccount
    name: ingress-nginx-admission
    namespace: ingress-nginx
```

# Installation et Configuration Nginx Ingress

## ■ Création d'un « validating Webhook »

Le webhook est une requête http qui est exécuté suite à un événement et qui envoie ensuite les données nécessaires à la destination prévue...

Le webhook suivant joute les objets entrants à la liste des entrées, génère la configuration et appelle nginx pour s'assurer que la configuration n'a pas d'erreurs de syntaxe.

```
---
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  name: ingress-nginx-admission
(Ficher yaml coupé pour la présentation)
```

Doc : <https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#webhook-configuration>

```
webhooks:
- admissionReviewVersions:
  - v1
clientConfig:
  service:
    name: ingress-nginx-controller-admission
    namespace: ingress-nginx
    path: /networking/v1/ingresses
failurePolicy: Fail
matchPolicy: Equivalent
name: validate.nginx.ingress.kubernetes.io
rules:
- apiGroups:
  - networking.k8s.io
  apiVersions:
  - v1
  operations:
  - CREATE
  - UPDATE
  resources:
  - ingresses
  sideEffects: None
```

# Installation et Configuration Nginx Ingress

## ■ WebHook Certificate

Comme le webhook de validation/admission ne fonctionne qu'en Https, nous avons besoin d'un certificat  
Pour cela, nous pouvons utiliser un objet de type job afin de créer le CA et patcher le webhook...

Nous créons donc un manifest qui utilisera kube-webhook-certgen pour générer le certificat et patcher le webhook

(cfr 3\_CA\_jobs.yml)

```
mike@LenoMike:~/k8s/kubernetes_heroes_app/Materiel/WslK8S/09_Ingress/SetupeNginx/manifests$ kubectl get jobs -n ingress-nginx
NAME                      COMPLETIONS   DURATION   AGE
ingress-nginx-admission-create  1/1          7s         3m3s
ingress-nginx-admission-patch   1/1          2m52s     3m3s

mike@LenoMike:~/k8s/kubernetes_heroes_app/Materiel/WslK8S/09_Ingress/SetupeNginx/manifests$ kubectl describe ValidatingWebhookConfiguration ingress-nginx-admission
Name:           ingress-nginx-admission
Namespace:      default
Labels:         app.kubernetes.io/component=admission-webhook
                app.kubernetes.io/instance=ingress-nginx
                app.kubernetes.io/name=ingress-nginx
Annotations:    <none>
API Version:   admissionregistration.k8s.io/v1
Kind:          ValidatingWebhookConfiguration
Metadata:
  Creation Timestamp:  2024-09-18T22:22:09Z
  Generation:        2
  Resource Version:  12292
  UID:              8fee2e6b-d5af-4cd5-835e-e84319f7453e
Webhooks:
  Admission Review Versions:
    v1
  Client Config:
    Ca Bundle: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJkakNDQVJ5Z0F3SUJBZ0lSQuxiV2R1WWF3NHJEdFU3c1pBwmcweVL3Q2dZSUtvIk16ajBFQXdJd0R6RU4KTUFzR0ExVUVDaE1FYm1sc01UQWdGdzB5TkRBNU1UZ3lNakUxTXpkYUdBOHlNEkwTUreUSUX1NVF6TjFvdwpeekVOTUFzR0ExVUVdaE1FYm1sc01UQlpNQk1HQnIxRLNNNDLBz0VHQnRxR1NNNDLBd0VIQTBJQUJPSEFHNTFDcmpeDNDbw5UV21xcXgrZLd1dUVvVnVyMwtUjUrMwixVUU2Qk9nc21DN1LLZnZkTHFtQ2ZzOTJuU0NGcwFMlgKdnJzMTFadXfseFo5cVpaLz6QlZNQTRQTFVZER3RUId1FFQxdJQ0JEQVRcz05wsFNRUREQUtcZ2dyQmdFRgpCUwNEqVRBUEJnTLZIuk1CQWY4RUJUQURBUUgvTUiwR0ExVWRZ1FXQkJTQkpRUnJNNTNRdwNURWxXVFYrZmFlCl14Q05qekFLQmdncWhrak9QUVFEQWdOSUFEQkZBaUVBNTJBVUpstVNYNFB1MG95d05qaW5NRHE5S1Z2L0dP21MKbTRNSitIqlUyRwtDSUNy1hWZU16bCsrN1NjVWRKSmxTsNHBmcnVsV3F6L2pmWEhzcGhXcStwdmEKLS0tLS1FTkQg00VSVEIGSUNBVEmLS0tLQo=
  Services:
```

# Installation et Configuration Nginx Ingress

- Création du Rôle Ingress Controller et son service account  
(cfr 4\_nginx-service-account.yaml)

```
mike@LenoMike:~/k8s/kubernetes_heroes_app/Materiel/WslK8S/09_Ingress/SetupeNginx/manifests$ kubectl apply -f 4_nginx-service-account.yaml
serviceaccount/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
```

- Création de ConfigMap

Nous pouvons grâce à cela customiser les paramètres Nginx tels que le header par exemple... Voir  
<https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/> pour les détails de configurations

```
apiVersion: v1
data:
  allow-snippet-annotations: "true" #permet à l'utilisateur d'ajouter des directives à exécuter !!! uniquement si vous pouvez faire confiance aux utilisateurs
kind: ConfigMap
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  name: ingress-nginx-controller
  namespace: ingress-nginx
```

# Installation et Configuration Nginx Ingress

- Création des services pour Ingress Controller et Adminission Controller

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  name: ingress-nginx-controller-admission
  namespace: ingress-nginx
spec:
  ports:
    - appProtocol: https
      name: https-webhook
      port: 443
      targetPort: webhook
  selector:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  type: ClusterIP
```

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  externalTrafficPolicy: Local
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  ports:
    - appProtocol: http
      name: http
      port: 80
      protocol: TCP
      targetPort: http
    - appProtocol: https
      name: https
      port: 443
      protocol: TCP
      targetPort: https
  selector:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  type: LoadBalancer
```

# Installation et Configuration Nginx Ingress

- Nous devons maintenant créer une IngressClass.

Chaque IngressClass peut spécifier des configurations personnalisées qui influencent la manière dont le trafic est routé. Par exemple, un Ingress peut être configuré pour utiliser un proxy inverse différent, gérer des fonctionnalités comme le SSL, ou définir des stratégies de routage différentes en fonction du contrôleur.

Nous allons donc créer une IngressClass nommée nginx qui associera tous les objets Ingress qui l'utilisent au contrôle ingress nginx.

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    name: nginx
spec:
  controller: k8s.io/ingress-nginx
```

# Installation et Configuration Nginx Ingress

- And the last but not least : nous pouvons créer le déploiement du controller Ingress ☐ 8\_IngressControllerDeployment.yaml (trop volumineux pour être affiché ici)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  minReadySeconds: 0
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app.kubernetes.io/component: controller
      app.kubernetes.io/instance: ingress-nginx
      app.kubernetes.io/name: ingress-nginx
  template:
    metadata:
      labels:
        app.kubernetes.io/component: controller
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/name: ingress-nginx
    spec:
      containers:
        - args:
            - /nginx-ingress-controller
            - --publish-service=$(POD_NAMESPACE)/ingress-nginx-controller
            - --election-id=ingress-controller-leader
            - --controller-class=k8s.io/ingress-nginx
            - --configmap=$(POD_NAMESPACE)/ingress-nginx-controller
            - --validating-webhook=:8443
            - --validating-webhook-certificate=/usr/local/certificates/cert
            - --validating-webhook-key=/usr/local/certificates/key
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: LD_PRELOAD
              value: /usr/lib/x86_64-linux-gnu/libjansson.so
```

# Kind - Ingress

Ingress

# Ingress pour Kind

- Si vous désirez utiliser ingress avec Kind sur votre machine locale, voici la marche à suivre :

- 1) Création d'un cluster avec extra-port qui mappe les ports 80 et 443
- 2) Application du Deployment proposé par kind :  
<https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml>
- 3) Attendre que les processus soient lancés

```
#attendre que les processus soit ready
kubectl wait --namespace ingress-nginx \
    --for=condition=ready pod \
    --selector=app.kubernetes.io/component=controller \
    --timeout=90s
```

- 4) Créer un pod, un service et surtout un objet ingress pour définir les règles d'accès

# Ingress pour Kind

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
  - http:
      paths:
      - pathType: Prefix
        path: /foo(/|$(."))
        backend:
          service:
            name: foo-service
            port:
              number: 8080
      - pathType: Prefix
        path: /bar(/|$(."))
        backend:
          service:
            name: bar-service
            port:
              number: 8080
```

## Pour aller plus loin Helm

- <https://helm.sh/docs/intro/install/>
- <https://www.devopsroles.com/devops-ci-cd-pipeline-tutorial-part-1/>
- <https://medium.com/@sampath5898ch/ci-cd-with-jenkins-pipeline-nodejs-into-k8s-part-1-fd816c63b733>

## Annexes - Vocabulaires - etc...

- **GPG** : Les clés GPG (GNU Privacy Guard) sont utilisées pour chiffrer et signer électroniquement des données. Elles sont largement utilisées pour garantir l'authenticité, l'intégrité et la confidentialité des informations échangées, notamment dans le cadre de la sécurité des e-mails et des logiciels. Les clés GPG sont composées d'une paire de clés, une clé privée utilisée pour signer les données, et une clé publique utilisée pour vérifier les signatures et chiffrer les messages.
- **Kubectl** : est l'outil en ligne de commande (CLI) principal pour interagir avec les clusters Kubernetes. Il permet de gérer les déploiements, les services, les pods, les réplicas, etc.
- **Kubelet** : est l'agent qui s'exécute sur chaque nœud du cluster Kubernetes. Il est responsable de l'exécution des conteneurs dans les pods et de la gestion des ressources du nœud.
- **Kubeadm** : est une commande de ligne de commande qui facilite le processus de configuration d'un cluster Kubernetes. Il permet de créer un cluster, d'initialiser le contrôleur maître, de rejoindre les nœuds au cluster et de gérer les mises à jour du cluster.
- **Kubernetes CNI (Container Networking Interface)** : est une spécification et un ensemble d'outils pour la gestion des réseaux dans les clusters Kubernetes. Il fournit une interface standardisée pour la configuration et la gestion des réseaux entre les pods et les nœuds dans un cluster Kubernetes. Le CNI permet de gérer les aspects de la connectivité réseau des conteneurs, notamment l'allocation des adresses IP, la configuration des routes, la création de ponts réseau, etc.

## Annexes - Vocabulaires - etc...

- **Flannel** : Il agit en tant que réseau sous-jacent pour les communications entre les différents pods et services dans le cluster. Flannel utilise le protocole VXLAN (Virtual Extensible LAN) pour créer un réseau superposé (overlay network) qui relie les nœuds du cluster. Chaque nœud obtient une adresse IP virtuelle à l'intérieur du réseau Flannel, ce qui lui permet de communiquer avec les autres nœuds et les pods qui y sont exécutés. Lorsque les pods sont déployés sur un nœud, Flannel configure une interface virtuelle appelée "flannel.1" pour chaque pod. Les paquets réseau émis par les pods sont encapsulés dans des paquets VXLAN avec des en-têtes spécifiques, puis envoyés via le réseau Flannel vers le nœud de destination. Flannel utilise un backend réseau pour configurer les règles de routage et de pontage nécessaires. Il prend en charge plusieurs backends, notamment VXLAN, UDP, Host-GW (gateway hôte) et d'autres plugins de réseau spécifiques à la plateforme. En résumé, Flannel permet de fournir un réseau virtuel overlay pour les clusters Kubernetes, en connectant les nœuds et les pods ensemble de manière transparente. Il facilite la communication entre les différents composants du cluster et assure une connectivité réseau fiable et sécurisée pour les applications déployées dans le cluster.
- **\$HOME** : Sur Debian (et dans d'autres distributions Linux), la variable d'environnement \$HOME fait référence au répertoire personnel de l'utilisateur actuel. Elle représente le chemin absolu vers le répertoire dans lequel les fichiers et les configurations spécifiques à cet utilisateur sont stockés.

## Annexes - Vocabulaires - etc...

- **sudo chown \$(id -u):\$(id -g)** : Cette ligne de commande "sudo chown \$(id -u):\$(id -g)" est utilisée pour changer le propriétaire et le groupe d'un fichier ou d'un répertoire en utilisant la commande "chown" sous le privilège superutilisateur ("sudo").
  - "**\$(id -u)**" est une sous-commande qui récupère l'identifiant utilisateur de l'utilisateur actuel. L'option "-u" de la commande "id" est utilisée pour afficher uniquement l'identifiant utilisateur.
  - "**\$(id -g)**" est une sous-commande qui récupère l'identifiant de groupe de l'utilisateur actuel. L'option "-g" de la commande "id" est utilisée pour afficher uniquement l'identifiant de groupe.
    - En exécutant cette ligne de commande avec "sudo", les privilèges de superutilisateur sont utilisés pour appliquer le changement de propriétaire et de groupe sur le fichier ou le répertoire spécifié. Cela peut être utile dans des cas où les permissions nécessitent des privilèges élevés pour être modifiées.

# Ressources

- Kubectl CheatSheet : <https://v1-16.docs.kubernetes.io/docs/reference/kubectl/cheatsheet/>
- Commandes kubectl : <https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands>

## Ressources utilisées pour ce cours

- <https://www.pluralsight.com/>
- <https://docs.docker.com/docker-for-windows>
- <https://www.linkedin.com/learning/>
- <https://medium.com/faun/kubernetes-architecture-85ad2999882a>
- <https://kubernetes.io/docs/>
- <https://kind.sigs.k8s.io/docs/user/quick-start/#installation>
- <https://birthday.play-with-docker.com/>