

# Kubernetes



**Khalid EL BAAMRANI**

**ENSA de Marrakech**

1

## Références

Les ressources suivantes ont été utilisées pour préparer ces slides:

- Introduction to Kubernetes, Vikram, IoT Application Dev
- Kubernetes v1.15 07/2019, Bob Killen and Jeffrey Sica
- Certified Kubernetes Administrator (CKA) Course, kodekloud

<https://github.com/kodekloudhub/certified-kubernetes-administrator-course/tree/master?tab=readme-ov-file>

# Orchestration des conteneurs

- L'orchestration de conteneurs automatise le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs à travers le cluster.
- Les entreprises qui doivent déployer et gérer des centaines ou des milliers de conteneurs et d'hôtes Linux peuvent bénéficier de l'orchestration des conteneurs.
- L'orchestration de conteneurs est utilisée pour automatiser les tâches suivantes à grande échelle :
  - Configuration et planification des conteneurs
  - Approvisionnement et déploiement de conteneurs
  - Redondance et disponibilité des conteneurs
  - Augmenter ou supprimer des conteneurs pour répartir uniformément la charge des applications sur l'infrastructure hôte
  - Mouvement de conteneurs d'un hôte à un autre en cas de pénurie de ressources d'un hôte ou en cas d'arrêt d'un hôte
  - Allocation des ressources entre les conteneurs
  - Exposition externe des services exécutés dans un conteneur avec le monde extérieur
  - Équilibrage de charge de la découverte de services entre les conteneurs
  - Surveillance de la santé des conteneurs et des hôtes



3

## Kubernetes

- **Kubernetes**, également connu sous le nom de **K8s**, est un outil open source de gestion de conteneurs.
  - K8s est une abréviation de Kubernetes. Au lieu d'utiliser le mot entier, nous remplaçons simplement 'ubernete' par le chiffre 8.
- Il fournit un environnement d'exécution de conteneur, une orchestration de conteneurs, des mécanismes d'auto-réparation, la découverte de services, l'équilibrage de charge et la mise à l'échelle des conteneurs.
- Initialement développé par Google, pour gérer des applications conteneurisées dans un environnement en cluster, mais ensuite donné à la CNCF
- Il est écrit en Golang
- Il s'agit d'une plate-forme conçue pour gérer complètement le cycle de vie des applications et services conteneurisés à l'aide de méthodes offrant la prévisibilité, l'évolutivité et la haute disponibilité.

4

# Kubernetes vs Swarm

Features	Kubernetes	Docker Swarm
<b>Installation &amp; Cluster Configuration</b>	Installation is complicated; but once setup, the cluster is very strong	Installation is very simple; but cluster is not very strong
<b>GUI</b>	GUI is the Kubernetes Dashboard	There is no GUI
<b>Scalability</b>	Highly scalable & scales fast	Highly scalable & scales 5x faster than Kubernetes
<b>Auto-Scaling</b>	Kubernetes can do auto-scaling	Docker Swarm cannot do auto-scaling
<b>Load Balancing</b>	Manual intervention needed for load balancing traffic between different containers in different Pods	Docker Swarm does auto load balancing of traffic between containers in the cluster
<b>Rolling Updates &amp; Rollbacks</b>	Can deploy Rolling updates & does automatic Rollbacks	Can deploy Rolling updates, but not automatic Rollbacks
<b>Data Volumes</b>	Can share storage volumes only with other containers in same Pod	Can share storage volumes with any other container
<b>Logging &amp; Monitoring</b>	In-built tools for logging & monitoring	3rd party tools like ELK should be used for logging & monitoring

5

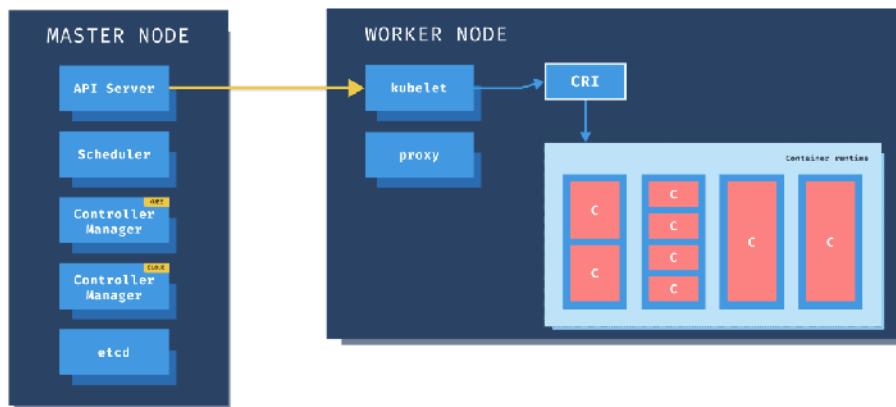
## Cluster Kubernetes

Un cluster Kubernetes est un ensemble de machines physiques ou virtuelles et d'autres ressources d'infrastructure nécessaires à l'exécution de vos applications conteneurisées. Chaque machine d'un cluster Kubernetes est appelée un **node**.

Il existe deux types de nœuds dans chaque cluster Kubernetes :

**Master node(s):** héberge les composants du plan de contrôle Kubernetes et gère le cluster

**Worker node(s):** exécute vos applications conteneurisées

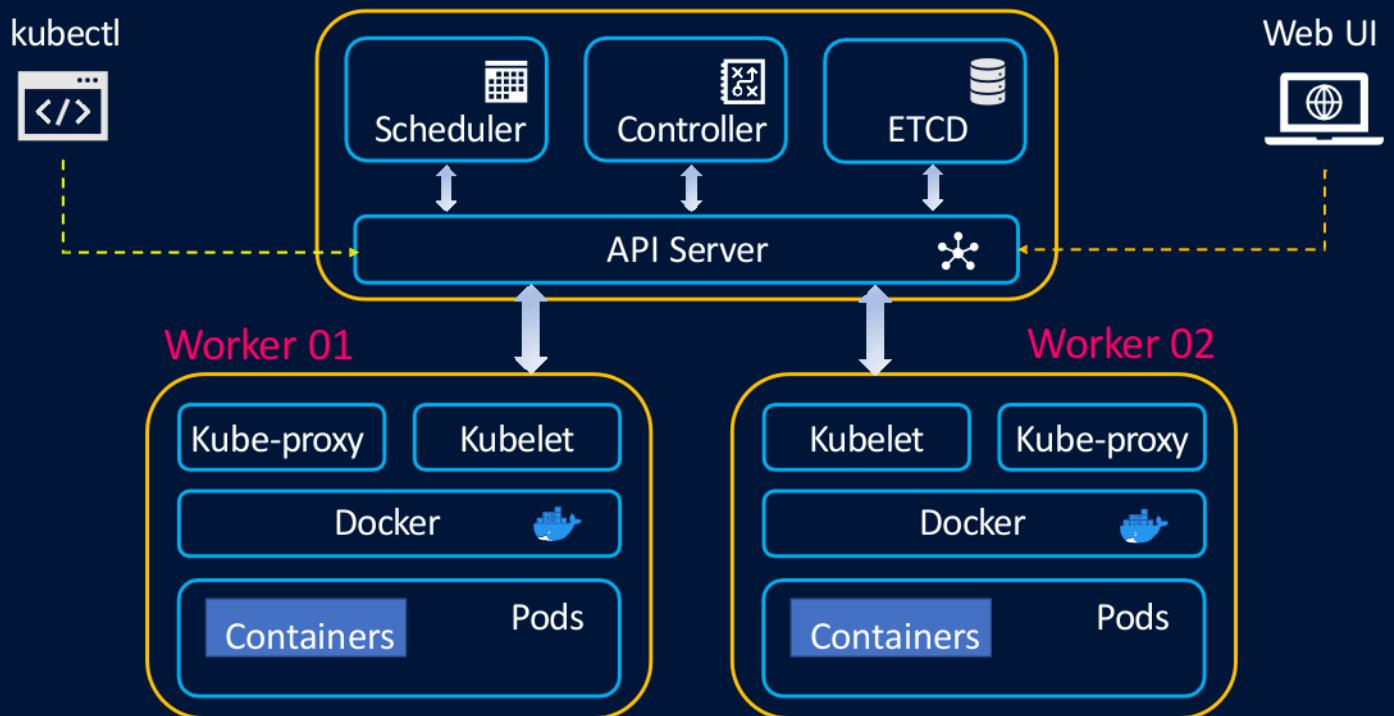


www.technicloudnative.com

6

# Architecture de Kubernetes

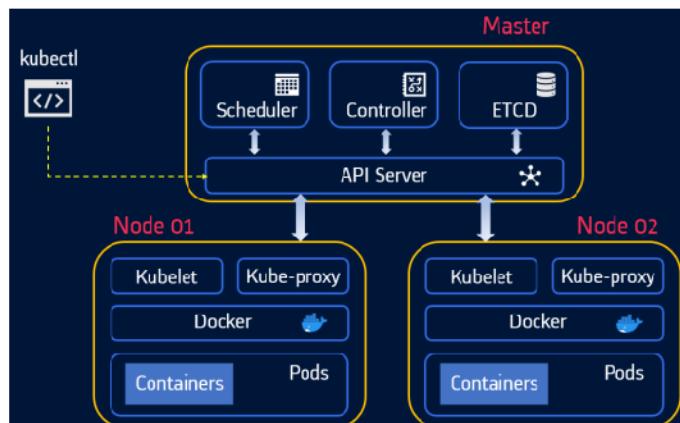
Master



## Architecture de Kubernetes

### Kubernetes Master

- Le maître est responsable de la gestion du cluster.
- Vous pouvez accéder au nœud maître via CLI, GUI ou API
- Le maître surveille les nœuds du cluster et il est responsable de l'orchestration des conteneurs sur les worker nodes
- Pour atteindre la tolérance aux pannes, il peut y avoir plusieurs master nodes dans le cluster.
- C'est le point d'accès à partir duquel les administrateurs et autres utilisateurs interagissent avec le cluster pour gérer la planification et le déploiement des conteneurs.
- Il comporte quatre composants : **ETCD** **Scheduler** **Controller** et **API Server**

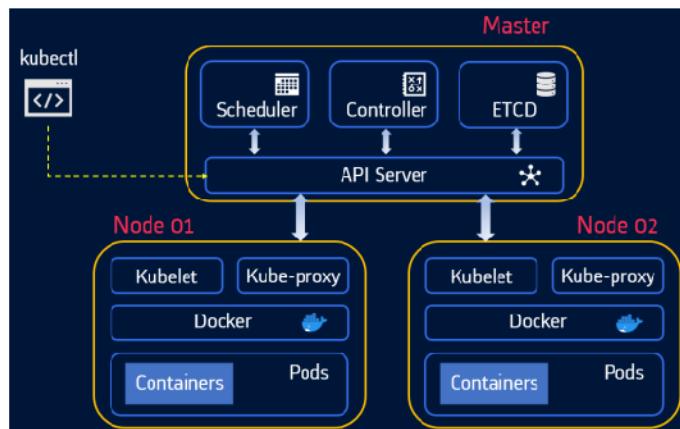


# Architecture de Kubernetes

## Kubernetes Master

### ETCD

- ETCD est une base de données clé-valeur distribuée et fiable utilisée par Kubernetes pour stocker toutes les données utilisées pour gérer le cluster.
- Lorsque vous avez plusieurs nœuds et plusieurs masters dans votre cluster, etcd stocke toutes ces informations sur tous les nœuds du cluster de manière distribuée.



### Scheduler

- Il est responsable de la répartition du travail ou des conteneurs sur plusieurs nœuds.
- Il recherche les conteneurs nouvellement créés et les affecte aux nœuds.

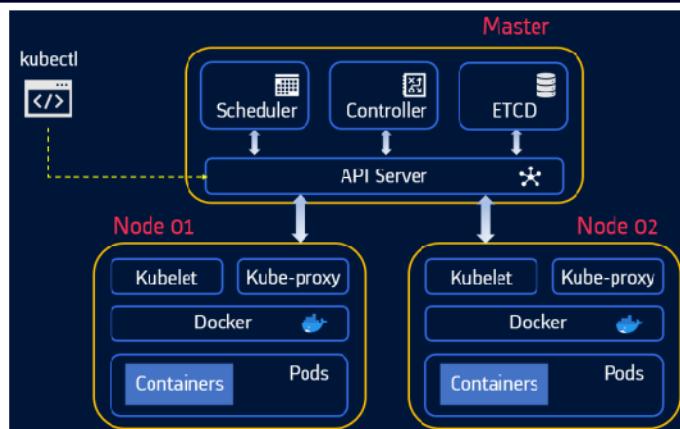
9

# Architecture de Kubernetes

## Kubernetes Master

### API server manager

- Les maîtres communiquent avec le reste du cluster via le kube-apiserver, le principal point d'accès au plan de contrôle.
- Il valide et exécute les commandes REST de l'utilisateur
- kube-apiserver s'assure également que les configurations dans etcd correspondent aux configurations des conteneurs déployés dans le cluster.



### Controller manager

- Les contrôleurs sont le cerveau derrière l'orchestration.
- Ils sont chargés de réagir lorsque des nœuds ou des conteneurs tombent en panne. Les contrôleurs prennent la décision de faire démarrer de nouveaux conteneurs dans de tels cas.
- Le controller exécute des boucles de contrôle qui gèrent l'état du cluster en vérifiant si les déploiements, répliques et nœuds requis sont en cours d'exécution dans le cluster.

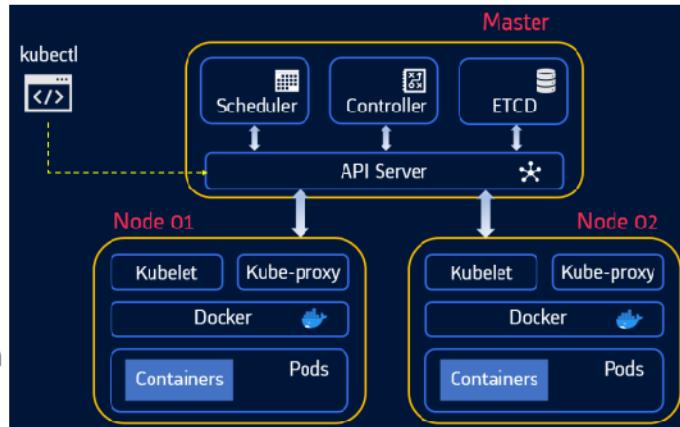
10

# Architecture de Kubernetes

## Kubernetes Master

### Kubectl

- C'est l'utilitaire de ligne de commande grâce auquel nous pouvons interagir avec le cluster k8s
- Utilise les API fournies par le serveur API pour interagir avec le cluster.
- Utilisé pour déployer et gérer des applications sur un cluster Kubernetes



- `kubectl run nginx` used to deploy an application on the cluster.
- `kubectl cluster-info` used to view information about the cluster and the
- `kubectl get nodes` used to list all the nodes part of the cluster.

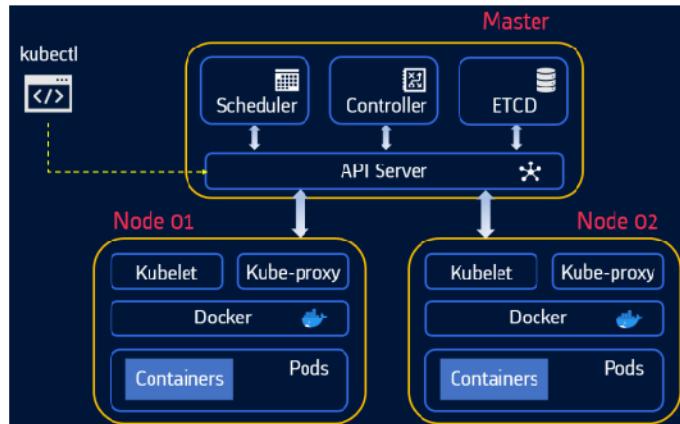
11

# Architecture de Kubernetes

## Kubernetes Worker

### Kubelet

- Les worker nodes ont l'agent kubelet qui est chargé d'interagir avec le maître pour fournir des informations sur l'état du worker node
- Réaliser les actions demandées par le master sur les worker nodes.



### Kube proxy

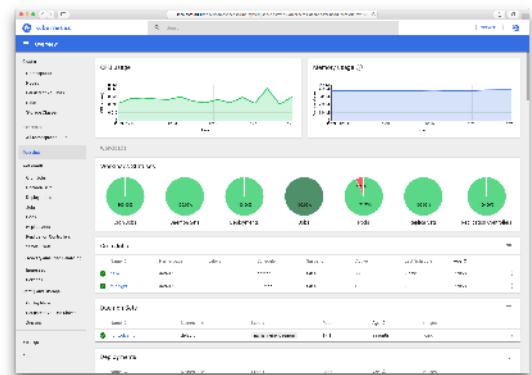
- Il est chargé de garantir que le trafic réseau est correctement acheminé vers les services internes et externes selon les besoins et il est basé sur les règles définies par les politiques réseau dans kube-controller-manager

12

# Architecture de Kubernetes

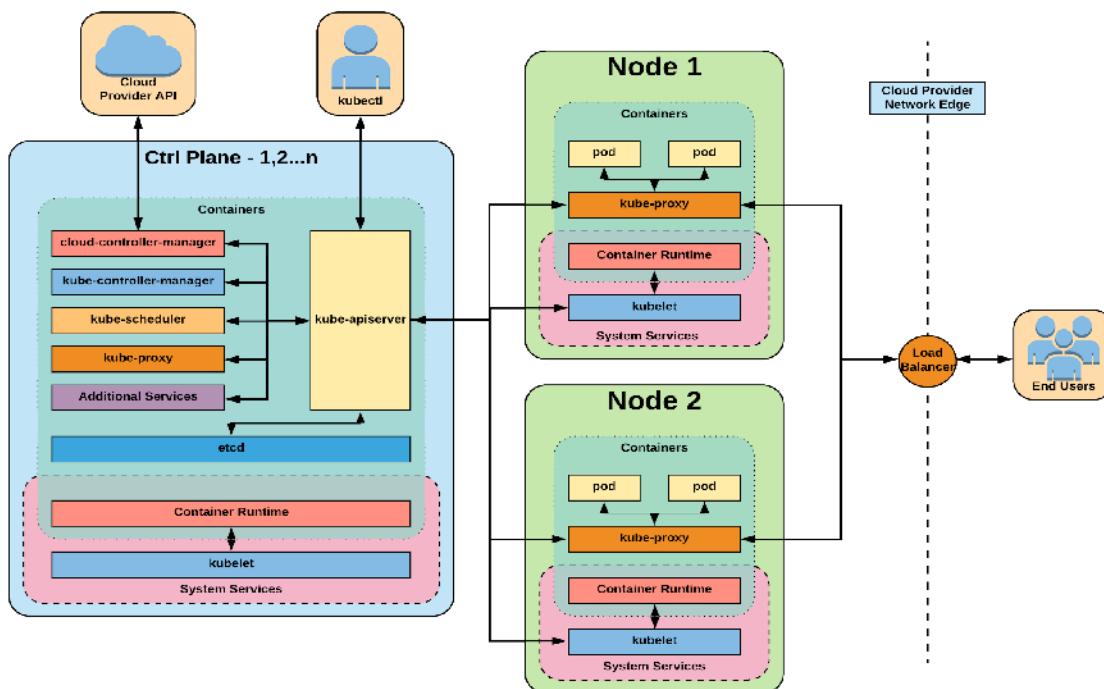
## Optional Services:

- Cloud-controller-manager
  - Daemon that provides cloud-provider specific knowledge and integration capability into the core control loop of Kubernetes.
- Cluster DNS
  - Provides Cluster Wide DNS for Kubernetes Services.
- Kube Dashboard
  - A limited, general purpose web front end for the Kubernetes Cluster.



13

# Architecture de Kubernetes



14

# Installation

15

## Certified Kubernetes Distributions

- Cloud Managed: **EKS** by AWS, **AKS** by Microsoft and **GKE** by google
- Self Managed: **OpenShift** by Redhat and Docker Enterprise
- Local dev/test: Micro K8s by Canonical, **Minikube**
- Vanilla Kubernetes: The core Kubernetes project(baremetal), **Kubeadm**
- Special builds: **K3s** by Rancher, a light weight K8s distribution for Edge devices

Online Emulator: <https://labs.play-with-k8s.com/>

<https://www.cncf.io/certification/software-conformance/>

16

# Installation de Kubernetes dans PC

- Minikube
- Kubeadm
- K3S
- Play-with-k8s



minikube



kubeadm



K3S

17

## Minikube

- Minikube est un outil facilitant l'exécution locale de Kubernetes.
- Minikube exécute un **cluster Kubernetes à nœud unique** dans une machine virtuelle (VM) ou un ordinateur portable pour les utilisateurs qui souhaitent essayer Kubernetes ou le développer au quotidien.

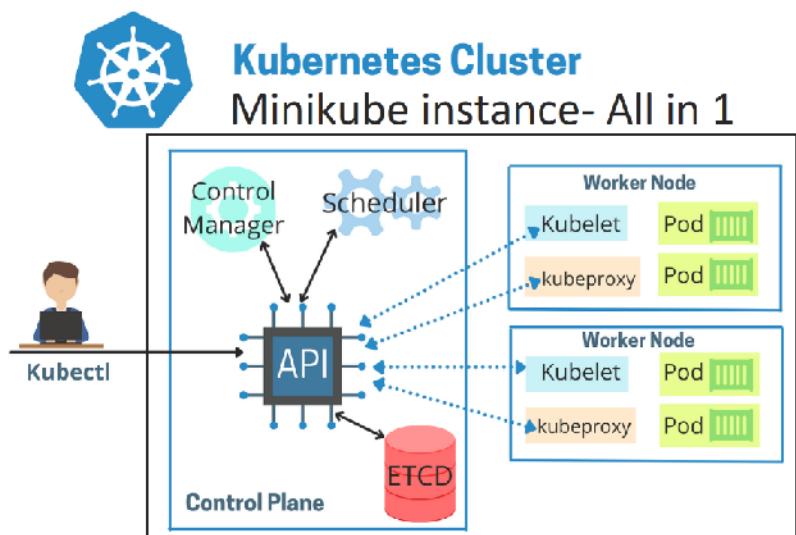


minikube

18

# Caractéristiques de minikube

- Prend en charge la dernière version de Kubernetes
- Multiplateforme (Linux, macOS, Windows)
- Déployer en tant que VM, conteneur ou baremetal
- Plusieurs environnements d'exécution de conteneurs (CRI-O, docker)
- API direct pour un chargement et une création d'images ultra-rapides
- Fonctionnalités avancées telles que LoadBalancer, les montages de système de fichiers et la politique réseau
- Addons pour des applications Kubernetes faciles à installer



19

## Installation de Minikube

- Install kubectl
  - sudo apt-get update && sudo apt-get install -y apt-transport-https
  - curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
  - echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list
  - sudo apt-get update
  - sudo apt-get install -y kubectl
  - kubectl version --client
- Installer un hyperviseur
  - KVM, Virtualbox ...
- Install minikube
  - curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube\_latest\_amd64.deb
  - sudo dpkg -i minikube\_latest\_amd64.deb
- Start your cluster
  - minikube start --nodes=3
  - minikube status

```
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

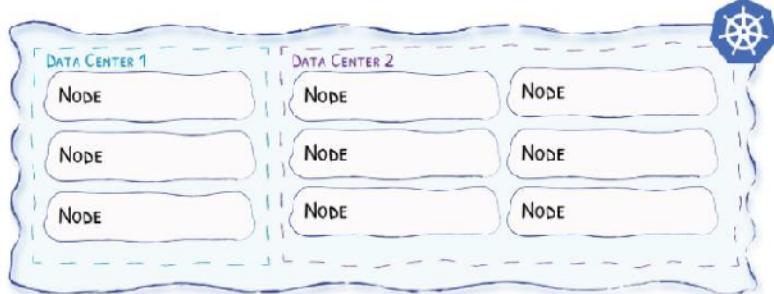
20

# Vérification de l'installation

## kubectl version

```
root@k3s-master:/home/osboxes# kubectl version -o yaml
clientVersion:
  buildDate: "2020-05-07T00:18:01Z"
  compiler: gc
  gitCommit: 608e444a03fe3a705849fc8d9ad9e35f1dffec286
  gitTreeState: clean
  gitVersion: v1.18.2+k3s1
  goVersion: go1.13.8
  major: "1"
  minor: "10"
  platform: linux/amd64
serverVersion:
  buildDate: "2020-05-07T00:10:01Z"
  compiler: gc
  gitCommit: 608e444a03fe3a705849fc8d9ad9e35f1dffec286
  gitTreeState: clean
  gitVersion: v1.18.2+k3s1
  goVersion: go1.13.0
  major: "1"
  minor: "10"
  platform: linux/amd64
```

NODES ARE  
THE MACHINES  
IN YOUR  
CLUSTER.  
  
THEY ARE  
AVAILABLE  
RESOURCES  
THAT CAN  
COME AND GO.



Nodes are machines in the cluster

## kubectl get nodes -o wide

```
root@k3s-master:/home/osboxes# kubectl get nodes -o wide
NAME      STATUS   ROLES   AGE    VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE        KERNEL-VERSION   CONTAINER-RUNTIME
k3s-slave01  Ready   <none>  7d4h   v1.18.2+k3s1  192.168.0.113  <none>       Ubuntu 18.04.3 LTS  5.0.0-23-generic  containerd://1.3.3-k3s2
k3s-master   Ready   master   7d4h   v1.18.2+k3s1  192.168.0.102  <none>       Ubuntu 18.04.3 LTS  5.0.0-23-generic  containerd://1.3.3-k3s2
k3s-slave02  Ready   <none>  7d4h   v1.18.2+k3s1  192.168.0.104  <none>       Ubuntu 18.04.3 LTS  5.0.0-23-generic  containerd://1.3.3-k3s2
root@k3s-master:/home/osboxes#
```

## kubectl cluster-info

```
root@k8s-master:/home/osboxes# kubectl cluster-info
Kubernetes master is running at https://192.168.50.2:6443
KubeDNS is running at https://192.168.50.2:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

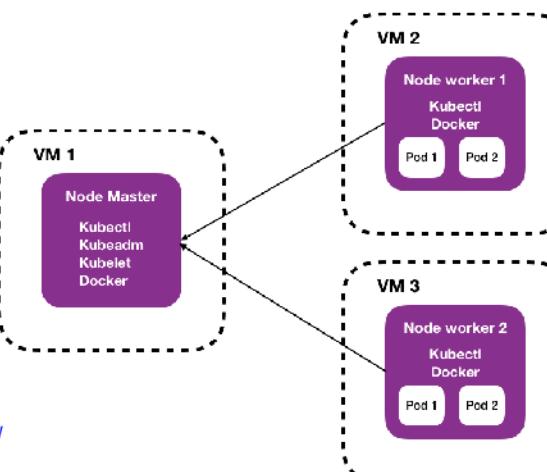
21

# Kubeadm



- Kubeadm est conçu pour être un moyen simple pour les nouveaux utilisateurs de commencer à essayer Kubernetes, pour la première fois éventuellement.
- C'est un moyen pour les utilisateurs avancés de tester leur application en même temps qu'un cluster facilement, et aussi être une brique de base dans un autre écosystème et/ou un outil d'installation avec une plus grande portée.
- Pré-requis :
  - Une ou plusieurs machines exécutant un système d'exploitation compatible deb/rpm, par exemple Ubuntu ou CentOS
  - 2 Go ou plus de RAM par machine.
  - 2 processeurs ou plus sur le master
  - Connectivité réseau entre toutes les machines du cluster,
- Installation : suivre les étapes sur le lien ci-dessous

<https://kubernetes.io/fr/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>



22

# Concepts fondamentaux

Kubernetes comporte plusieurs éléments de base qui constituent la base de ses composants de niveau supérieur.

## Namespaces

**Pods**

**Labels**

**Services**

**Selectors**

**Deployment**

23

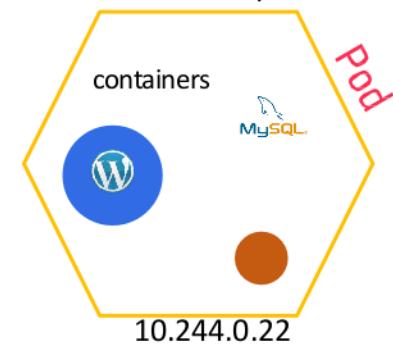
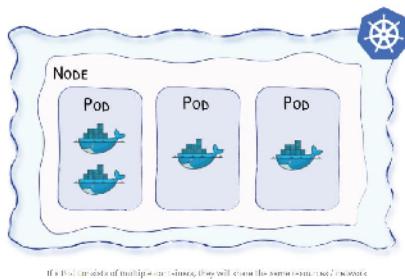
## Kubernetes Pods

24

# Pods

- Kubernetes n'exécute pas directement les conteneurs ; au lieu de cela, il enveloppe **un ou plusieurs conteneurs** dans une structure de niveau supérieur appelée **pod**
- Il s'agit également de la plus petite unité déployable pouvant être créée, planifiée et gérée sur un cluster Kubernetes.
- Chaque pod reçoit une adresse IP unique au sein du cluster.
- Les pods peuvent également contenir plusieurs conteneurs
- Puisque les pods sont scale up and down en tant qu'une unité, tous les conteneurs d'un pod doivent évoluer ensemble, quels que soient leurs besoins individuels. Cela peut conduire à un gaspillage de ressources.

Ex: nginx, mysql, wordpress..

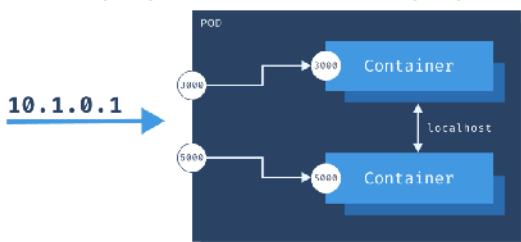


25

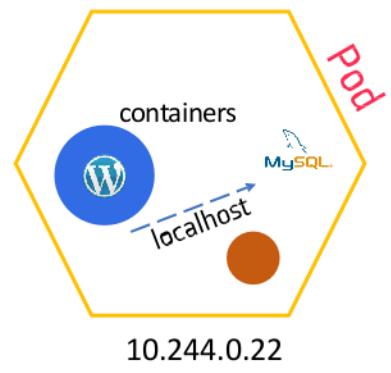
# Pods

- Tous les conteneurs du même pod partageront les mêmes volumes de stockage et ressources réseau et communiqueront à l'aide de localhost
- K8s utilise **YAML** pour décrire l'état souhaité des conteneurs dans un pod. Ceci est également appelé **Pod Spec**. Ces objets sont transmis au kubelet via le serveur API.
- Les pods sont utilisés comme unité de réPLICATION dans Kubernetes. Si votre application devient trop populaire et qu'une seule instance de pod ne peut pas supporter la charge, Kubernetes peut être configuré pour déployer de nouvelles répliques de votre pod sur le cluster si nécessaire.

Inside cluster



Using the example from the above figure, you could run curl 10.1.0.1:3000 to communicate to the one container and curl 10.1.0.1:5000 to communicate to the other container from other pods. However, if you wanted to talk between containers - for example, calling the top container from the bottom one, you could use http://localhost:3000.



26

# Kubernetes

## Imperative vs Declarative commands

- L'API Kubernetes définit de nombreux objets/ressources, tels que des namespaces, des pods, des déploiements, des services, des secrets, des config map, etc.
- Il existe deux manières de base de déployer des objets dans Kubernetes : de manière **impérative** et **déclarative**.

### impérative

- Implique l'utilisation de l'une des commandes basées sur les verbes comme `kubectl run`, `kubectl create`, `kubectl expose`, `kubectl delete`, `kubectl scale` et `kubectl edit`
- Convient pour les tests et l'expérimentation interactive

### déclarative

- Les objets sont écrits dans des fichiers **YAML** et déployés à l'aide `kubectl create` ou `kubectl apply`
- Idéal pour les environnements de production

27

# Pods

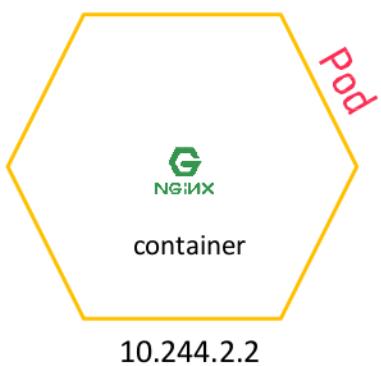
## Creating Pods: Imperative way

exposes port 80 of container `kubectl run test --image nginx --port 80`

```
root@k-master:/home/osboxes# kubectl get pods -o wide
NAME    READY   STATUS    RESTARTS   AGE      IP           NODE     NOMINATED NODE   READINESS GATES
test    1/1     Running   0          9m13s   10.244.2.2   k-slave02   <none>   <none>
root@k-master:/home/osboxes#
```

display extended information of pod `kubectl describe pod test`

```
root@k-master:/home/osboxes# k describe pod test
Name:         test
Namespace:    default
Priority:    0
Node:        k-slave02/192.168.0.107
Start Time:  Mon, 18 May 2020 13:52:09 -0400
Labels:       run=test
Annotations:  <none>
Status:      Running
IP:          10.244.2.2
IPs:
```



28

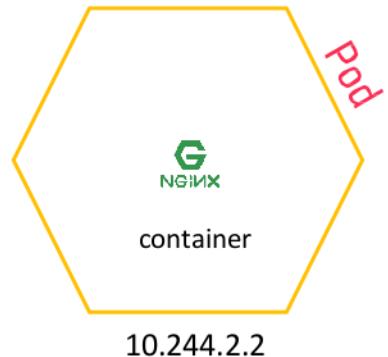
# Pods

## Creating Pods: Imperative way

curl <ip-of-pod>

```
root@k-master:/home/osboxes# curl 10.244.2.2
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
}
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

```



29

# Pods

## Creating Pods : Imperative way

kubectl run <pod-name> --image <image-name>

kubectl run nginx --image nginx --dry-run=client

dry-run n'exécute pas la commande mais montrera les modifications que la commande apporterait au cluster

```
root@k-master:/home/osboxes# kubectl run nginx --image nginx --dry-run=client
pod/nginx created (dry run)
```

kubectl run nginx --image nginx --dry-run=client -o yaml

```
root@k-master:/home/osboxes# kubectl run nginx --image nginx --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
    name: nginx
  name: nginx
  spec:
    containers:
    - image: nginx
      name: nginx
      resources: {}
    dnsPolicy: ClusterFirst
    restartPolicy: Always
status: {}
```

affiche le résultat de la commande en YAML. Raccourci pour créer un yaml déclaratif à partir de commandes impératives

# Kubernetes

## Manifest /Spec file

Fichiers de configuration d'objet K8s écrits en YAML ou JSON

Ils décrivent l'état souhaité de votre application. Un fichier peut inclure une ou plusieurs descriptions d'objets (manifestes).

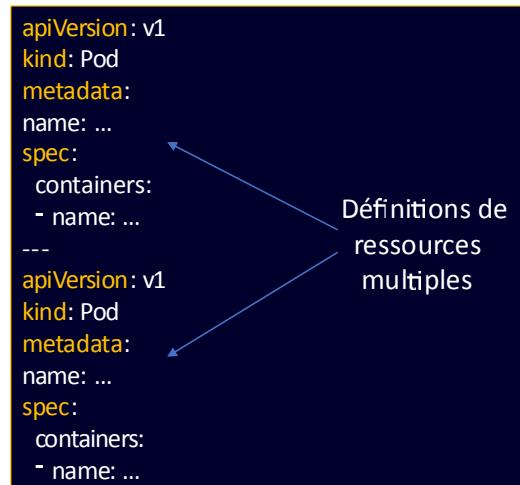
### # manifest file template

**apiVersion** : version de l'API Kubernetes utilisée pour créer l'objet

**kind** : type d'objet en cours de création

**metadata** : Données permettant d'identifier l'objet de manière unique, y compris un nom et un namespace facultatif

**spec** : configuration qui définit ce que l'on souhaite pour l'objet



31

## Pods

### Creating Pods: Declarative way

- ◆ **name** - The name of the container
- ◆ **image** - The container image
- ◆ **ports** - array of ports to expose. Can be granted a friendly name and protocol may be specified
- ◆ **env** - array of environment variables
- ◆ **command** - Entrypoint array (equiv to Docker **ENTRYPOINT**)
- ◆ **args** - Arguments to pass to the command (equiv to Docker **CMD**)

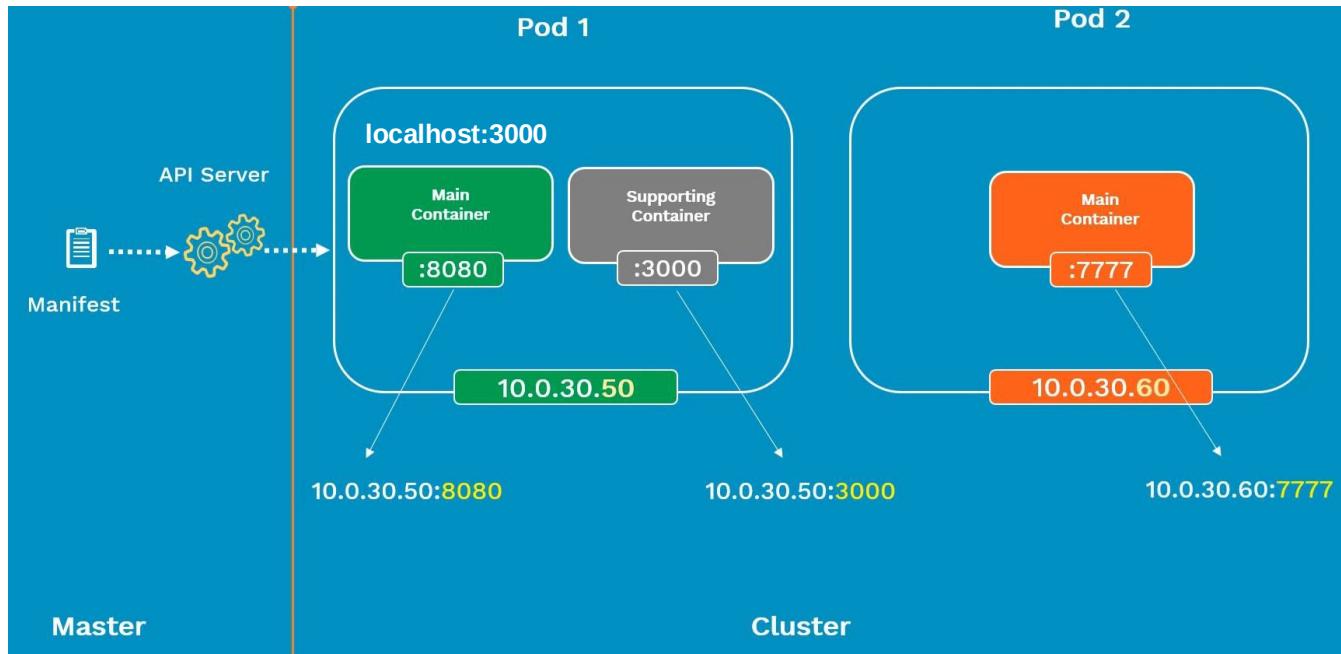
- Create a pod
  - **kubectl create –f pod-definition.yml**
- if manifest file is changed/updated after deployment and need to re-deploy the pod again
  - **kubectl apply –f pod-definition.yml**
- delete pod :
  - **kubectl delete pod <pod-name>**

```
# pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
    name: nginx
    image: nginx:stable-alpine
    ports:
      - containerPort: 80
        name: http
        protocol: TCP
    env:
      - name: MYVAR
        value: isAwesome
    command: ["/bin/sh", "-c"]
    args: ["echo ${MYVAR}"]
```

32

# Pods

## Pod Networking

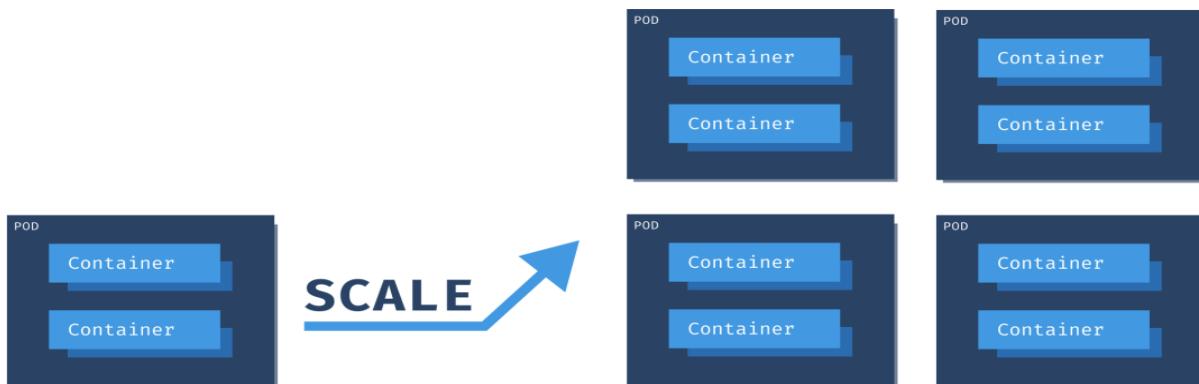


33

# Pods

## Scaling Pods

- Tous les conteneurs du pod sont mis à l'échelle ensemble (scale up or down).
- Vous ne pouvez pas mettre à l'échelle des conteneurs individuels dans les pods. Le pod est l'unité d'échelle des K8.
- La méthode recommandée est de n'avoir qu'un seul conteneur par pod. Les pods multi-conteneurs sont très rares.



34

# Pods

## Scaling Pods

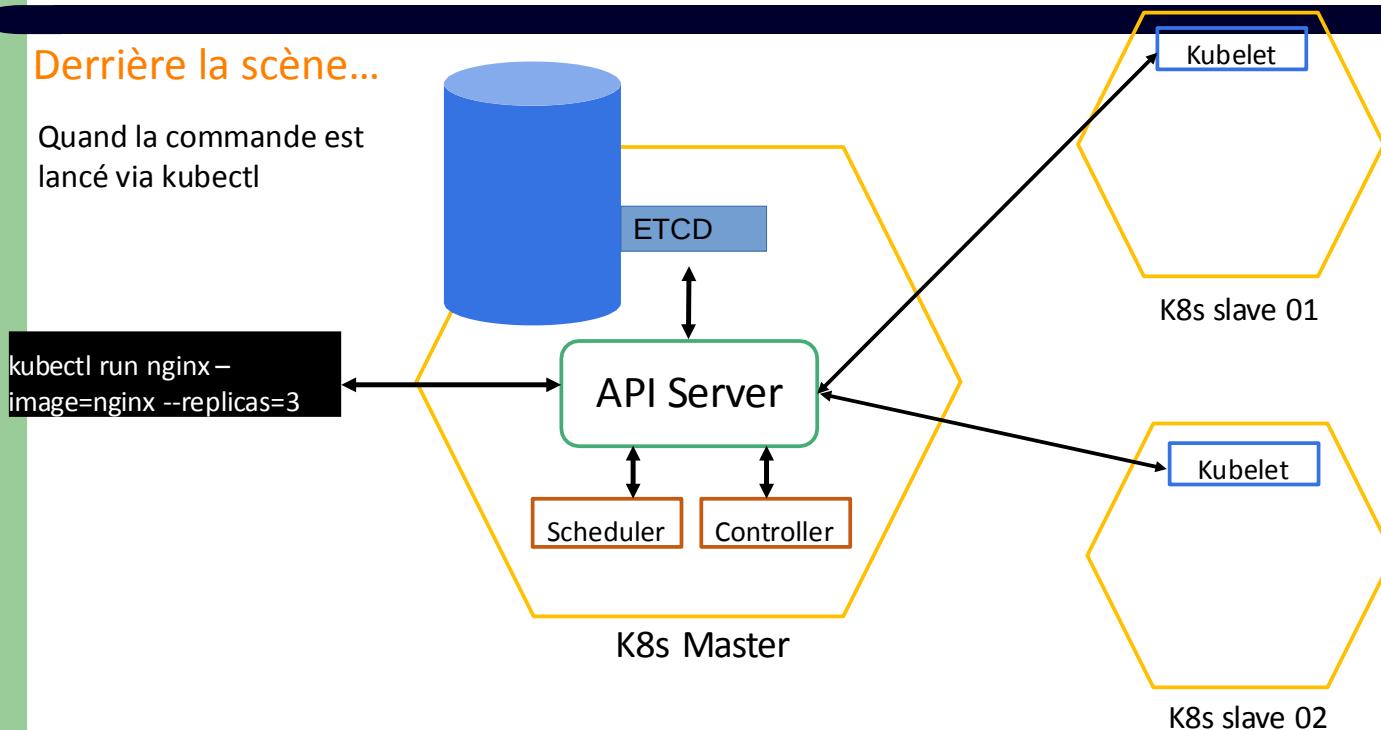
- Un seul pod peut ne pas suffire à gérer le trafic utilisateur. De plus, si ce seul pod tombe en panne à cause d'une panne kubernetes ne relancera pas automatiquement ce pod.
- Afin d'éviter cela, nous aimerais que plusieurs pod s'exécutent en même temps à l'intérieur du cluster.
- Kubernetes prend en charge différents contrôleurs (Replicacontroller et ReplicaSet) pour gérer plusieurs instances d'un pod. Ex : 3 répliques du serveur web nginx
- Replication Controller garantit la haute disponibilité en remplaçant les pods défectueux/morts par de nouveaux pour garantir que les répliques voulu s'exécutent toujours au sein d'un cluster.
- Alors, cela signifie-t-il que vous ne pouvez pas utiliser un contrôleur de réPLICATION si vous prévoyez d'avoir un seul pod ?
  - Non! Même si vous disposez d'un seul pod, le contrôleur de réPLICATION peut vous aider en lançant automatiquement un nouveau pod lorsque celui tombe en panne.
- Une autre raison pour laquelle nous avons besoin d'un contrôleur de réPLICATION est de créer plusieurs pod pour partager la charge entre eux.
- **Le contrôleur de réPLIQUE est obsolète et remplacé par Replicaset**

35

# Pods

## Derrière la scène...

Quand la commande est lancé via kubectl



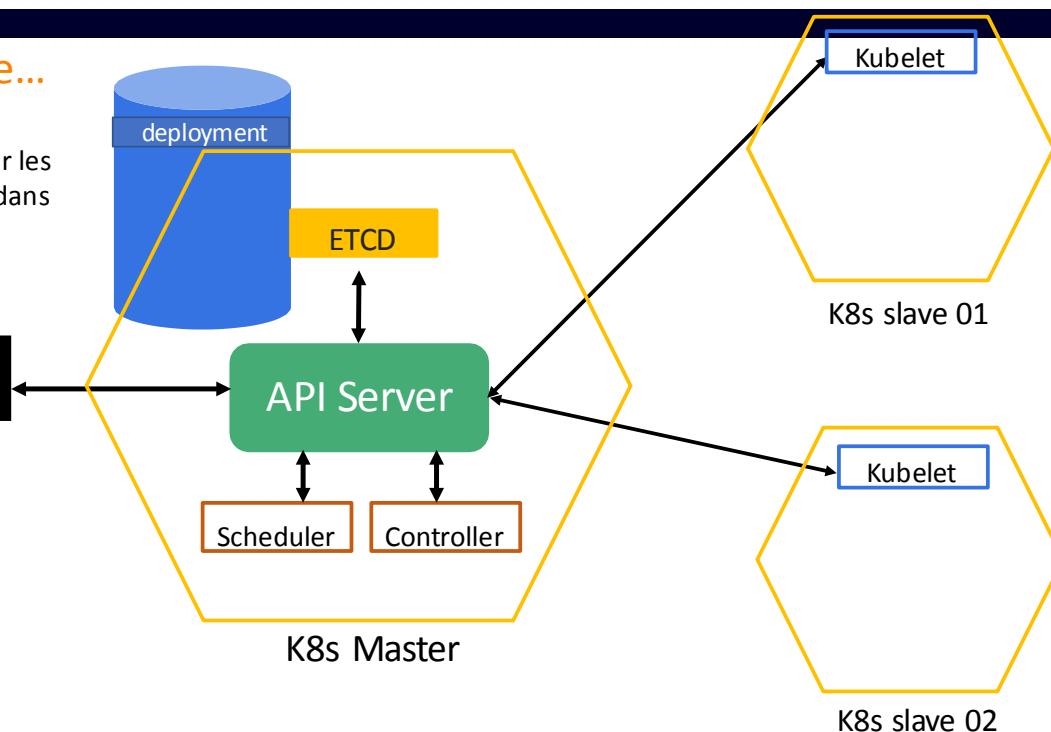
36

# Pods

## Derrière la scène...

Le serveur API met à jour les détails du déploiement dans ETCD

```
$ kubectl run nginx --image=nginx --replicas=3
```



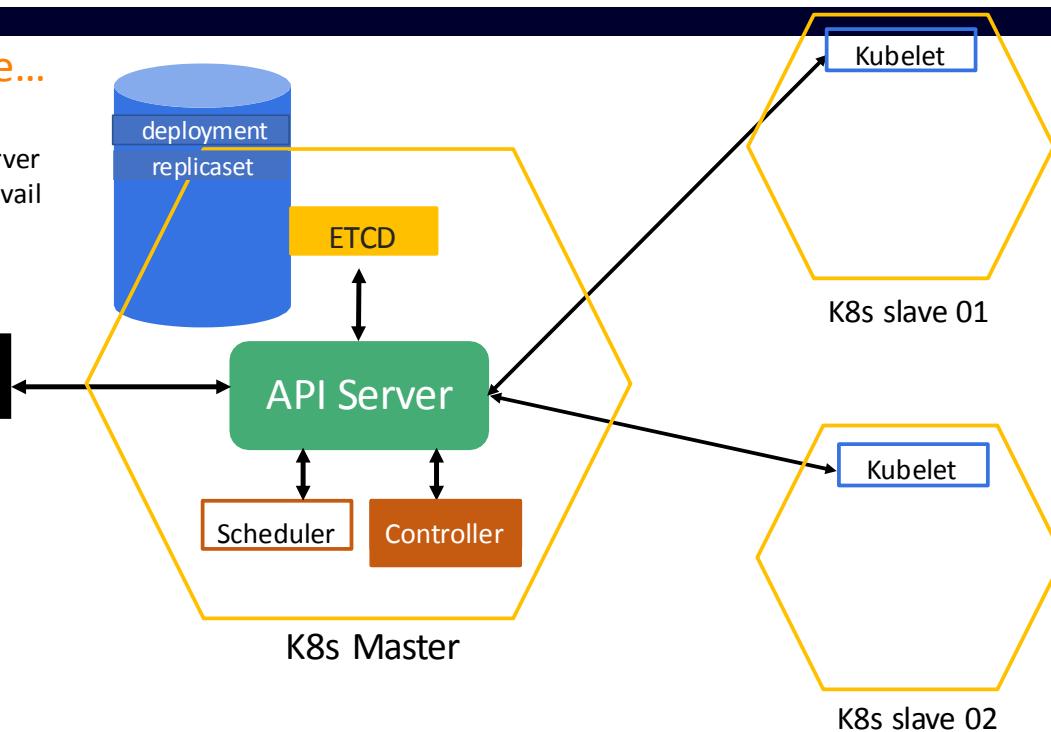
37

# Pods

## Derrière la scène...

Le contrôleur via API Server identifie la charge de travail et crée un ReplicaSet

```
$ kubectl run nginx --image=nginx --replicas=3
```



38

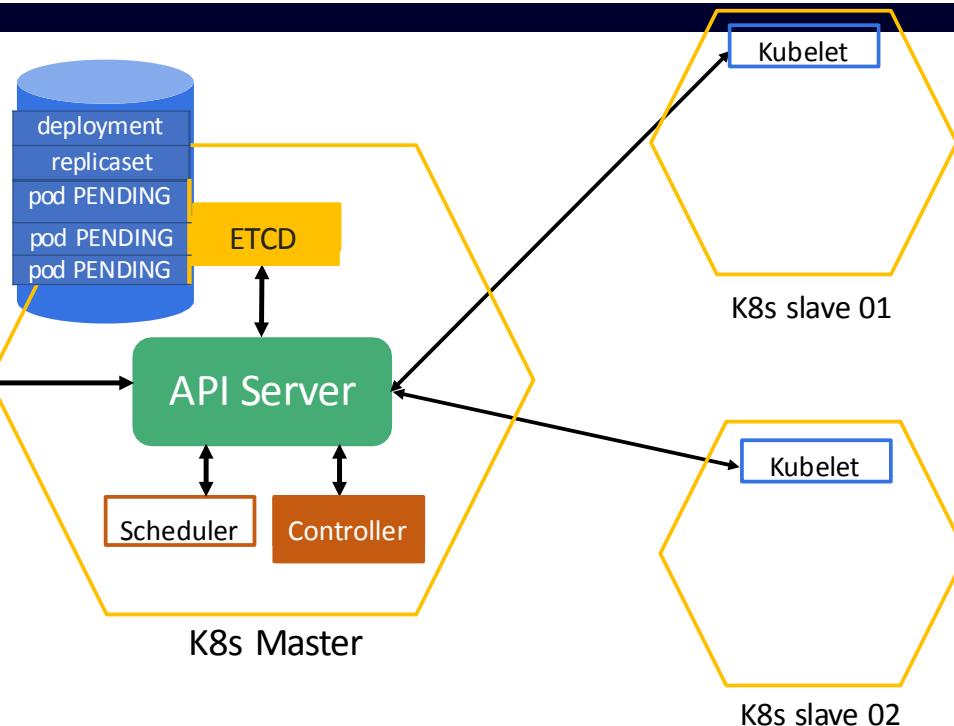
# Pods

## Derrière la scène...

ReplicaSet crée le nombre requis de pods et met à jour l'ETCD.

Notez l'état des pods. Ils sont toujours en attente

```
$ kubectl run nginx --image=nginx --replicas=3
```



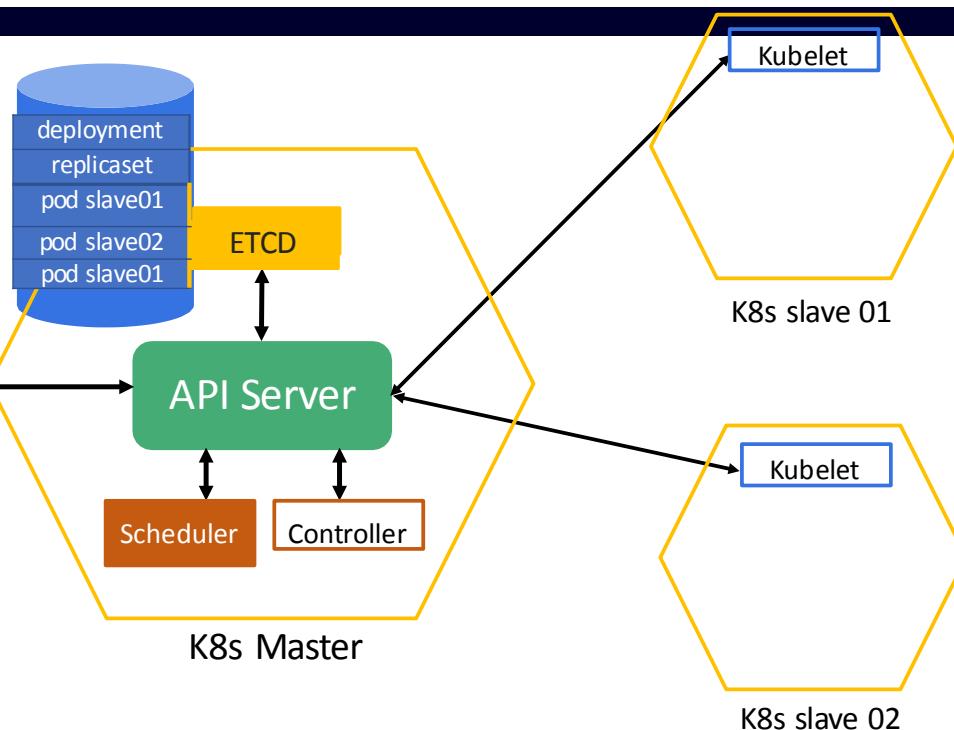
39

# Pods

## Derrière la scène...

Le scheduler identifie la charge de travail via API-Server et décide des nœuds sur lesquels le pod doit être planifié.

```
$ kubectl run nginx --image=nginx --replicas=3
```



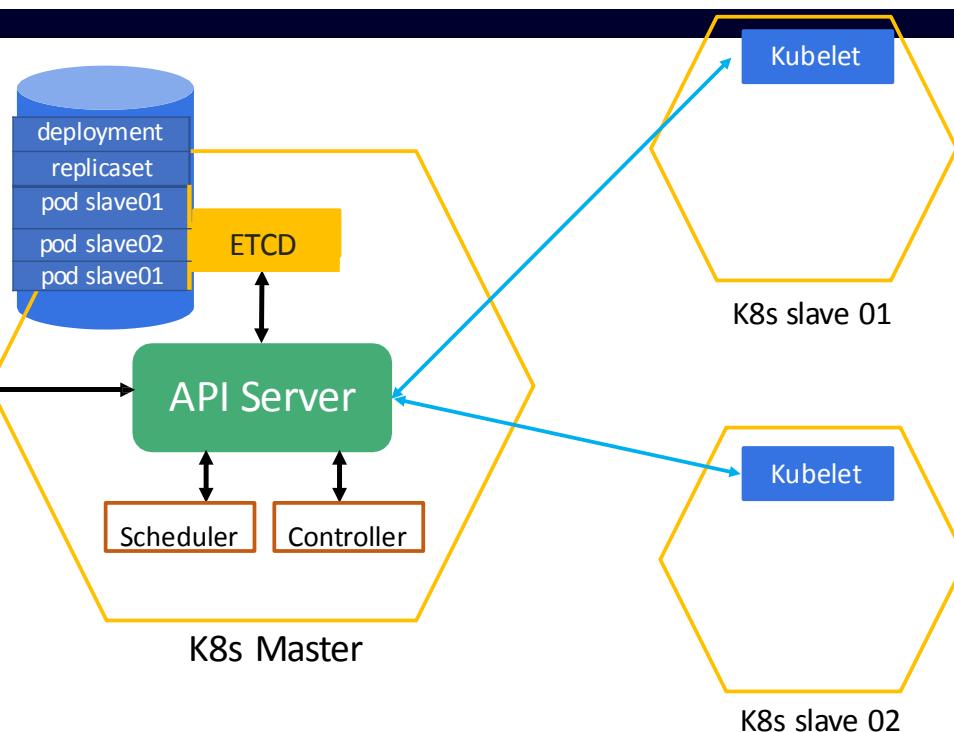
40

# Pods

## Derrière la scène...

Kubelet identifie sa charge de travail via API-Server et comprend qu'il doit déployer certains pods sur son nœud

```
$ kubectl run nginx --image=nginx --replicas=3
```



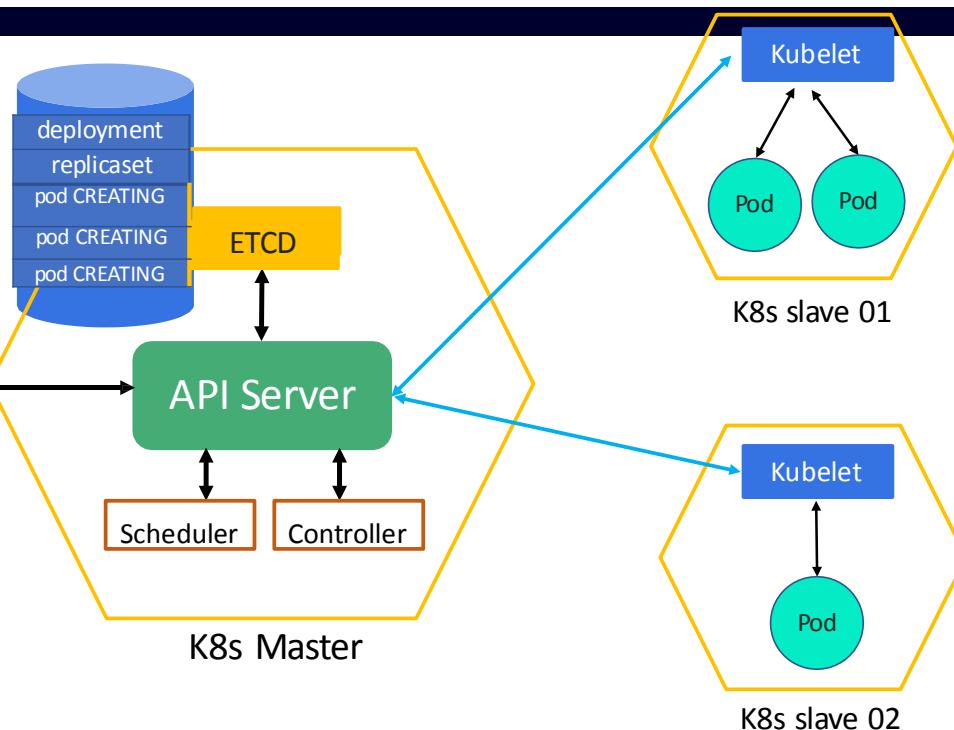
41

# Pods

## Derrière la scène...

Kubelet demande au démon Docker de créer les pods. En même temps, il met à jour le statut en tant que « Pods CREATING » dans ETCD via le serveur API.

```
$ kubectl run nginx --image=nginx --replicas=3
```

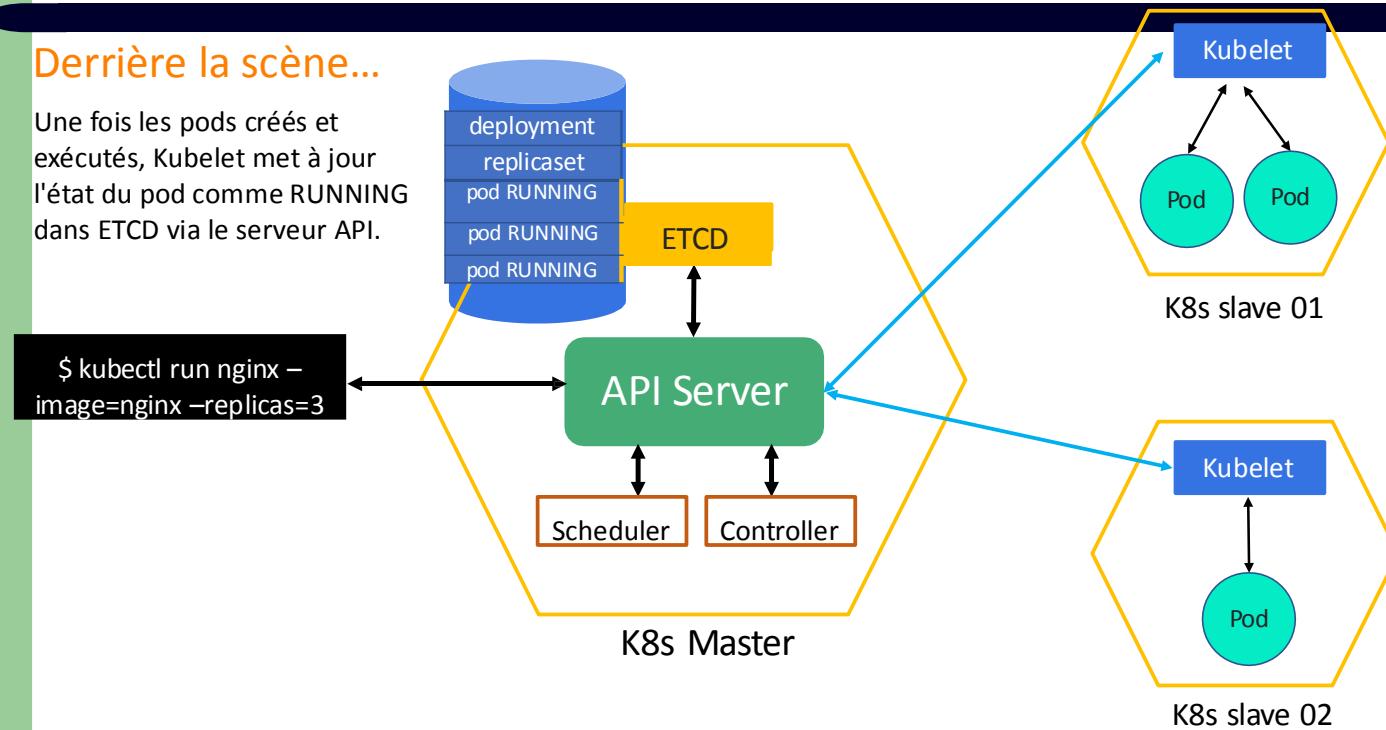


42

# Pods

## Derrière la scène...

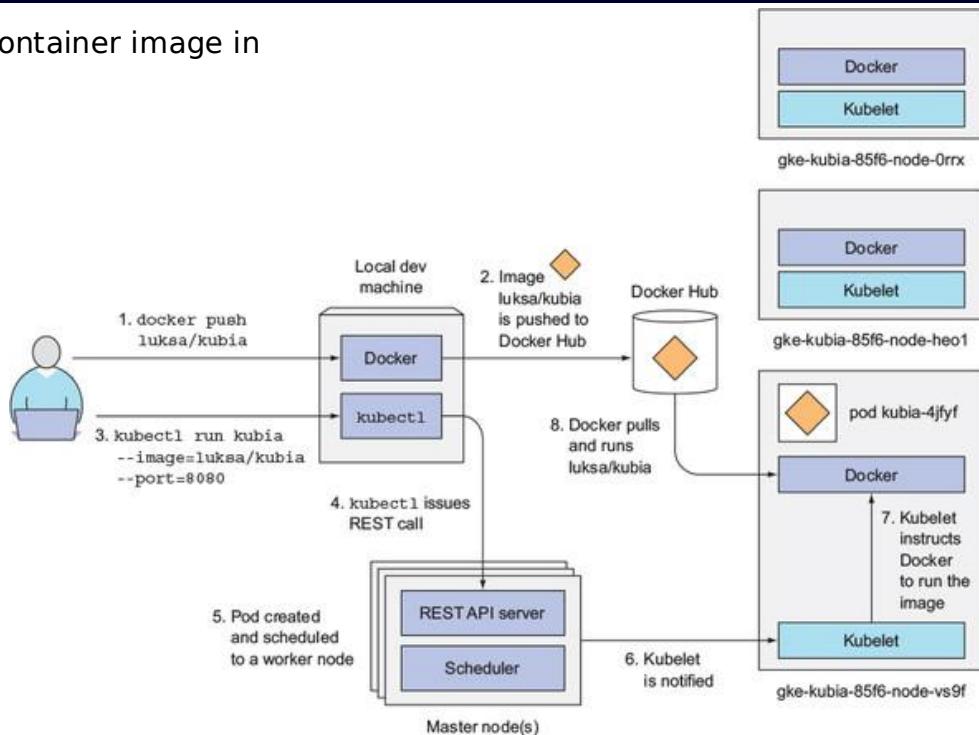
Une fois les pods créés et exécutés, Kubelet met à jour l'état du pod comme RUNNING dans ETCD via le serveur API.



43

# Pods

Running the container image in Kubernetes



44

# Pods

## Interaction with pods

```
kubectl run -i --tty busybox --image=busybox -- sh          # Run pod as interactive shell  
kubectl run nginx --it --image=nginx -- bash                # Run pod nginx  
kubectl run nginx --image=nginx --dry-run -o yaml > pod.yaml # Run pod nginx and write its spec into a file  
                                                               called pod.yaml  
kubectl attach my-pod -i                                     # Attach to Running Container  
kubectl exec my-pod -- ls /                                    # Run command in existing pod (1 container case)  
kubectl exec my-pod -c                                         # Run command in existing pod (multi-container  
                                                               case)  
  
root@k8s-master:/home/osboxes# kubectl run nginx -it --image nginx -- bash  
If you don't see a command prompt, try pressing enter.  
root@nginx:/# █
```

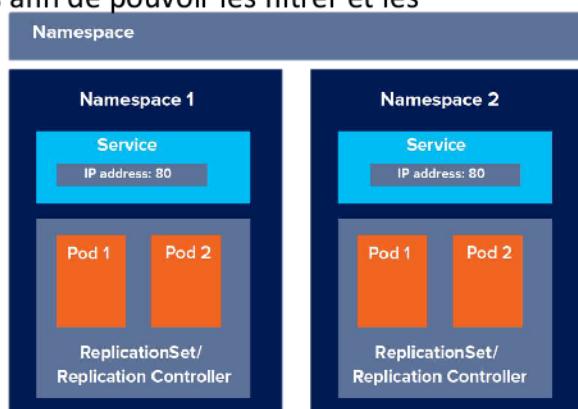
45

# Kubernetes Namespaces

46

# Namespaces

- Kubernetes prend en charge plusieurs clusters virtuels présents sur le même cluster physique. Ces clusters virtuels sont appelés namespaces (espaces de nommage en français).
- Les clusters Kubernetes peuvent gérer simultanément un grand nombre de charges de travail car les organisations choisissent souvent de déployer des projets créés par des équipes distinctes sur des clusters partagés.
- Avec plusieurs déploiements dans un seul cluster, il y a de fortes chances que les déploiements supprimés appartiennent à des projets différents
- Ainsi, les namespaces vous permettent de regrouper des objets afin de pouvoir les filtrer et les contrôler en tant qu'unité/groupe.
- Les namespaces fournissent une portée pour les noms. Les noms des ressources doivent être uniques au sein d'un namespace, mais pas entre les namespaces.
- Ainsi, chaque namespace Kubernetes fournit la portée des noms Kubernetes qu'il contient ; ce qui signifie qu'en utilisant la combinaison d'un nom d'objet et d'un namespace chaque objet obtient une identité unique à travers le cluster



# Namespaces

Par défaut, un cluster Kubernetes est créé avec le trois espaces de noms suivants :

- **Default:** il s'agit de namespace par défaut pour les utilisateurs. Par défaut, toutes les ressources créées dans le cluster Kubernetes sont créées dans cet espace de noms
- **Kube-system:** il s'agit de l'espace de noms pour les objets créés par Kubernetes. Toute modification apportée aux objets dans cet espace de noms causerait des dommages irréparables au cluster lui-même
- **kube-public:** ce namespace est créé automatiquement et est visible par tous les utilisateurs . Ce namespace est principalement réservé à l'utilisation du cluster, au cas où certaines ressources devraient être disponibles publiquement dans l'ensemble du cluster.

# Namespaces

kubectl get namespaces

```
root@k8s-master:/home/osboxes# kubectl get ns
NAME      STATUS  AGE
default   Active  68m
kube-node-lease  Active  68m
kube-public  Active  68m
kube-system  Active  68m
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: prod
  labels:
    app: MyBigWebApp
```

kubectl get all -n kube-system (lists available objects under a specific namespace)

```
root@k8s-master:/home/osboxes# kubectl get all -n kube-system
NAME          READY  STATUS    RESTARTS  AGE
pod/coredns-66bff467f8-bm6kr  1/1   Running  0          68m
pod/coredns-66bff467f8-hj9ll  1/1   Running  0          68m
pod/etcd-k8s-master          1/1   Running  0          68m
pod/kube-apiserver-k8s-master 1/1   Running  0          68m
pod/kube-controller-manager-k8s-master 1/1   Running  0          68m
pod/kube-flannel-ds-amd64-bc5vg 1/1   Running  0          67m
pod/kube-flannel-ds-amd64-cg48x 1/1   Running  0          68m
pod/kube-flannel-ds-amd64-xz8qn 1/1   Running  0          67m
pod/kube-proxy-rq77v          1/1   Running  0          68m
pod/kube-proxy-tl99m          1/1   Running  0          67m
pod/kube-proxy-w/bqz          1/1   Running  0          67m
pod/kube-scheduler-k8s-master  1/1   Running  0          68m

NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)           AGE
service/kube-dns ClusterIP  10.96.0.10  <none>        53/UDP,53/TCP,9153/TCP 68m
```

```
$ kubectl get ns --show-labels
NAME      STATUS  AGE  LABELS
default   Active  11h  <none>
kube-public  Active  11h  <none>
kube-system  Active  11h  <none>
prod      Active  6s   app=MyBigWebApp
```

kubectl get all --all-namespaces (lists available objects under all available namespaces)

49

# Namespaces

kubectl create ns dev # Namespace for Developer team

kubectl create ns qa # Namespace for qualite team

kubectl create ns production # Namespace for Production team

Deploy objects in a namespace

kubectl run nginx --image=nginx -n dev

kubectl get pod/nginx -n dev

kubectl apply --namespace=dev -f pod.yaml

```
root@k8s-master:/home/osboxes# kubectl run --image=nginx nginx -n dev
pod/nginx created
root@k8s-master:/home/osboxes# k get pods -n dev
NAME      READY  STATUS    RESTARTS  AGE
nginx    1/1   Running  0          7m37s
root@k8s-master:/home/osboxes#
```

Delete a namespace

kubectl delete ns production

50

# Labels & Selectors

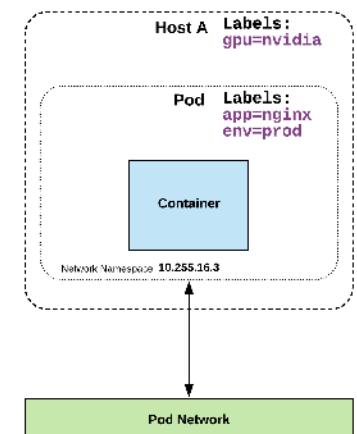
## Labels and Selectors

### Labels

- Labels (étiquettes) sont des paires clé/valeur attachées à des objets, tels que les pods, les services, les déploiements et même les nœuds
- Permettent d'identifier, décrire et regrouper des ensembles d'objets ou de ressources associés
- Exemple labels:
  - "release" : "stable", "release" : "canary"
  - "environment" : "dev", "environment" : "qa", "environment" : "production"
  - "tier" : "frontend", "tier" : "backend", "tier" : "cache"
  - "partition" : "customerA", "partition" : "customerB"
  - "track" : "daily", "track" : "weekly"

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
  labels:
    app: nginx
    env: prod
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
  
```



# Labels and Selectors

## Labels

- Si les étiquettes ne sont pas mentionnées lors du déploiement d'objets k8s à l'aide de commandes impératives, l'étiquette est automatiquement définie comme `app: <object-name>`

```
kubectl run --image nginx nginx  
kubectl get pods --show-labels
```

```
root@k8s-master:/home/osboxes# kubectl get pods --show-labels  
NAME READY STATUS RESTARTS AGE LABELS  
nginx 1/1 Running 0 90s run=nginx
```

## Adding Labels

```
kubectl label pod nginx environment=dev
```

```
root@k8s-master:/home/osboxes# kubectl label pod nginx environment=dev  
pod/nginx labeled  
root@k8s-master:/home/osboxes# kubectl get pods --show-labels  
NAME READY STATUS RESTARTS AGE LABELS  
nginx 1/1 Running 0 6m45s environment=dev,run=nginx  
root@k8s-master:/home/osboxes# █
```

53

# Labels and Selectors

## Selectors

- Les sélecteurs permettent de filtrer les objets en fonction des étiquettes (labels)
- L'API prend en charge deux types de sélecteurs : basés sur l'égalité (**equality-based**) et basés sur des ensembles (**set-based**).
- Un sélecteur d'étiquettes peut être composé de plusieurs exigences séparées par des virgules

## Equality-based Selector

- Les exigences basées sur l'égalité ou l'inégalité (**equality**- or **inequality-based**) permettent un filtrage par clés et valeurs d'étiquette.
- Trois types d'opérateurs sont admis `=,==,! =`

```
kubectl get pods -l environment =dev  
kubectl get pods -l environment=dev, tier=backend
```

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: pod-label-example  
labels:  
  app: nginx  
  env: prod  
spec:  
  containers:  
    - name: nginx  
      image: nginx:stable-alpine  
      ports:  
        - containerPort: 80  
nodeSelector:  
  gpu: nvidia
```

Utilisé par les contrôleurs de réPLICATION et services

54

# Labels and Selectors

## Set-based Selector

- permet de filtrer les clés en fonction d'un ensemble de valeurs.
- Trois types d'opérateurs sont pris en charge `in`, `notin` et `exists` (uniquement l'identifiant de la clé).

```
kubectl get pods -l 'environment in (production, qa)'
```

```
kubectl get pods -l 'environment in (production),tier in (frontend)'
```

```
selector:
  matchLabels:
    component: redis
  matchExpressions:
    - {key: tier, operator: In, values: [cache]}
    - {key: environment, operator: NotIn, values: [dev]}
```

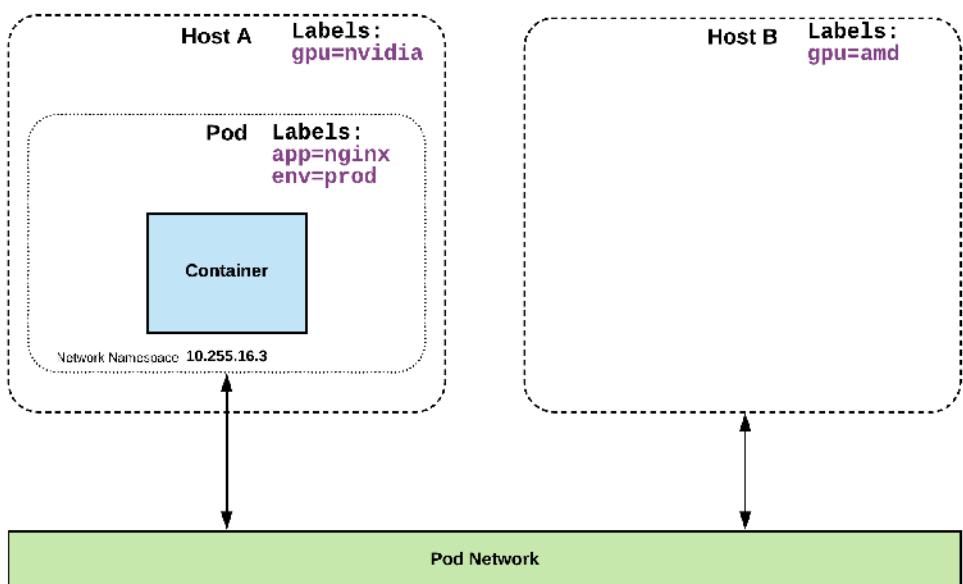
Utilisé par ReplicaSets, Deployments, DaemonSets

55

# Labels and Selectors

## Selector Example:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
labels:
  app: nginx
  env: prod
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
nodeSelector:
  gpu: nvidia
```



56

# Labels and Selectors

Selector Types:

**Equality based:** `=,==, or !=`

```
selector:  
matchLabels:  
  gpu: nvidia
```

**Set-based:** `in,notin, and exist.`

```
selector:  
matchExpressions:  
  - key: gpu  
    operator: in  
    values: ["nvidia"]
```

Pour plus de détails voir :

[https://wangwei1237.github.io/Kubernetes-in-Action-Second-Edition/docs/Filtering\\_Objects\\_With\\_Label\\_Selectors.html](https://wangwei1237.github.io/Kubernetes-in-Action-Second-Edition/docs/Filtering_Objects_With_Label_Selectors.html)

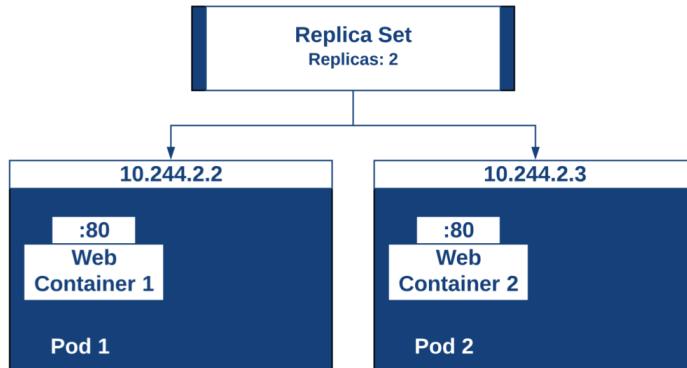
57

## ReplicaSet

58

# ReplicaSet

- ReplicaSets sont des API de niveau supérieur qui permet d'exécuter facilement plusieurs instances d'un pod.
- ReplicaSets garantit que le nombre exact de pods (répliques) est toujours en cours d'exécution dans le cluster en remplaçant tous les pods défaillants par de nouveaux.



59

# ReplicaSet

## Pod vs ReplicaSet

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: webapp
spec:
  containers:
    - name: nginx-container
      image: nginx
      ports:
        - containerPort: 80
```

pod.yml

- **Replicas** : contrôle le nombre de répliques
- **Selector** : le sélecteur d'étiquettes du ReplicaSet générera toutes les instances de pod qu'il cible

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: webapp
    type: front-end
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      name: nginx-pod
      labels:
        app: webapp
    spec:
      containers:
        - name: nginx-container
          image: nginx
          ports:
            - containerPort: 80
```

*Actually definition of POD itself*

replicaset.yml

60

# ReplicaSet

## ReplicaSet Manifest file

Number of pods (replicas)

Which pods to watch?

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: hello
  labels:
    app: hello
spec:
  replicas: 5
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
        - name: hello-container
          image: busybox
          command: [ ... ]
```

Pod template to use

61

# ReplicaSet

kubectl create -f replica-set.yml

```
root@k-master:/home/osboxes# kubectl create -f replica-set.yml
replicaset.apps/nginx-replicaset created
root@k-master:/home/osboxes#
```

kubectl get rs -o wide

```
root@k-master:/home/osboxes# kubectl get rs -o wide
NAME      DESIRED   CURRENT   READY   AGE     CONTAINERS   IMAGES   SELECTOR
nginx-replicaset   3         3         3       76s   nginx-container   nginx   app=webapp
root@k-master:/home/osboxes#
```

kubectl get pods -o wide

```
root@k-master:/home/osboxes# kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE     IP           NODE
nginx-replicaset-67nb9  1/1    Running   0          93s   10.244.1.9   k-slave01
nginx-replicaset-g85dz  1/1    Running   0          93s   10.244.2.4   k-slave02
nginx-replicaset-l9cpz  1/1    Running   0          93s   10.244.1.8   k-slave01
```

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: webapp
    type: front
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      name: nginx-pod
      labels:
        app: webapp
    spec:
      containers:
        - name: nginx-container
          image: nginx
          ports:
            - containerPort: 80
```

Source : [https://medium.com/@saurabh\\_adhau/cracking-the-kubernetes-code-a-closer-look-at-deployment-and-replica-sets-be9c1d6183ae](https://medium.com/@saurabh_adhau/cracking-the-kubernetes-code-a-closer-look-at-deployment-and-replica-sets-be9c1d6183ae)

62

# ReplicaSet

## Commands

- `kubectl edit replicaset <replicaset-name>` ⇒ edit a replicaset; like image, replicas
- `kubectl delete replicaset <replicaset-name>` ⇒ delete a replicaset; like image, replicas
- `kubectl delete --f replica-set.yml`
- `kubectl get all` ⇒ get pods, replicaset, deployments, services all in one shot
- `kubectl replace -f replicaset-definition.yml` ⇒ replaces the pods with updated definition file
- `kubectl scale --replicas=6 --f replicaset-definition.yml` ⇒ scale using definition file
- `kubectl scale --replicas=6 replicaset <replicaset-name>` ⇒ using name of replicaset

63

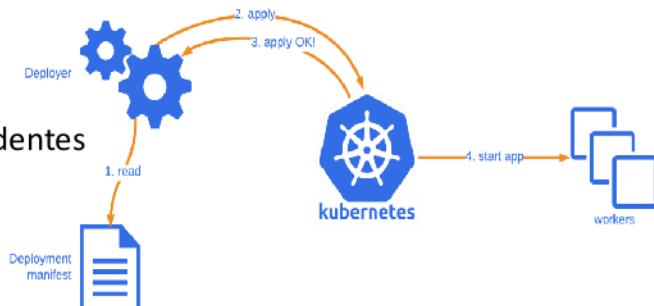
# Kubernetes Deployments

64

# Déploiement

- Un déploiement fournit des mises à jour déclaratives pour les pods et les ReplicaSets.
- Vous décrivez un état souhaité dans un déploiement et le contrôleur modifie l'état réel vers l'état souhaité à un rythme contrôlé.
- Cela semble similaire aux ReplicaSets mais avec des fonctions avancées
- Le déploiement est la méthode recommandée pour déployer un pod ou un ReplicaSet
- Par défaut, Kubernetes effectue les déploiements selon une stratégie de mise à jour continue.
- Principales fonctionnalités du déploiement :
  - Déployez facilement un ReplicaSet
  - Mises à jour continue (rolling update) des pods
  - Revenir (rollback) aux versions de déploiement précédentes
  - Mise à l'échelle (scale up/down) du déploiement
  - Suspension et reprise du déploiement

<https://kubernetes.io/fr/docs/concepts/workloads/controllers/deployment/>



65

# Déploiement

## Stratégie de déploiement

- Chaque fois que nous créons un nouveau déploiement, K8s déclenche un Rollout.
- Rollout est le processus de déploiement ou de mise à niveau progressif de vos conteneurs d'applications.
- Pour chaque déploiement/mise à niveau, un historique des versions sera créé, ce qui permettra de revenir à la version courante en cas d'échec de la mise à jour.
- Dans Kubernetes, il existe différentes manières de publier des mises à jour pour une application.
  - Recreate
  - RollingUpdate
  - Blue/green
  - Canary

```
spec: replicas: 10
strategy:
  type: Recreate
```

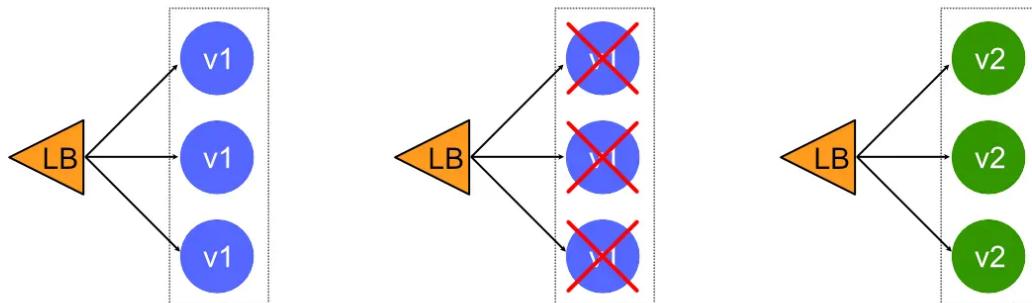
```
spec:
  replicas: 10
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 2
    maxUnavailable: 0
```

66

# Déploiement

## Stratégie de déploiement

- **Recreate:** mettra fin à toutes les instances en cours d'exécution, puis les recréera avec la version la plus récente. L'application subit des temps d'arrêt.



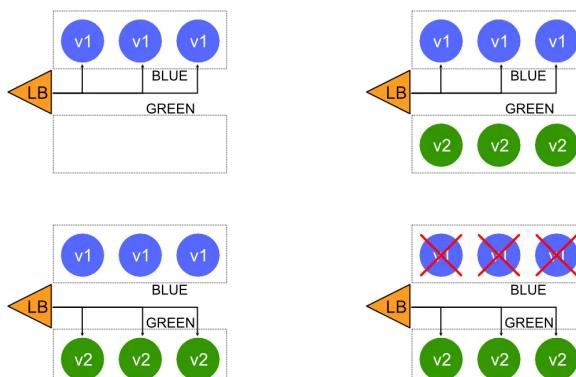
<https://ludovicwyffels.dev/k8s-deployment-strategies/>

67

# Déploiement

## Stratégie de déploiement

- **Blue/green** déployer la version “green” de l’application en parallèle de la version “blue”. Après avoir vérifié que la nouvelle version réponde aux exigences on switch à la nouvelle version et on arrête l’ancienne



68

# Déploiement

## Stratégie de déploiement

- **RollingUpdate** : 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. *C'est la stratégie par défaut dans K8s. Aucun temps d'arrêt de l'application n'est requis.*
  - Par défaut, le déploiement garantit que seulement 25 % de vos pods sont indisponibles lors d'une mise à jour et ne met pas à jour plus de 25 % des pods à un moment donné.
  - Il ne tue pas les anciens pods jusqu'à ce qu'un nombre suffisant de nouveaux pods apparaissent
  - Il ne crée pas de nouveaux pods tant qu'un nombre suffisant d'anciens pods n'ont pas été tués.

69

# Déploiement

## Rolling Update

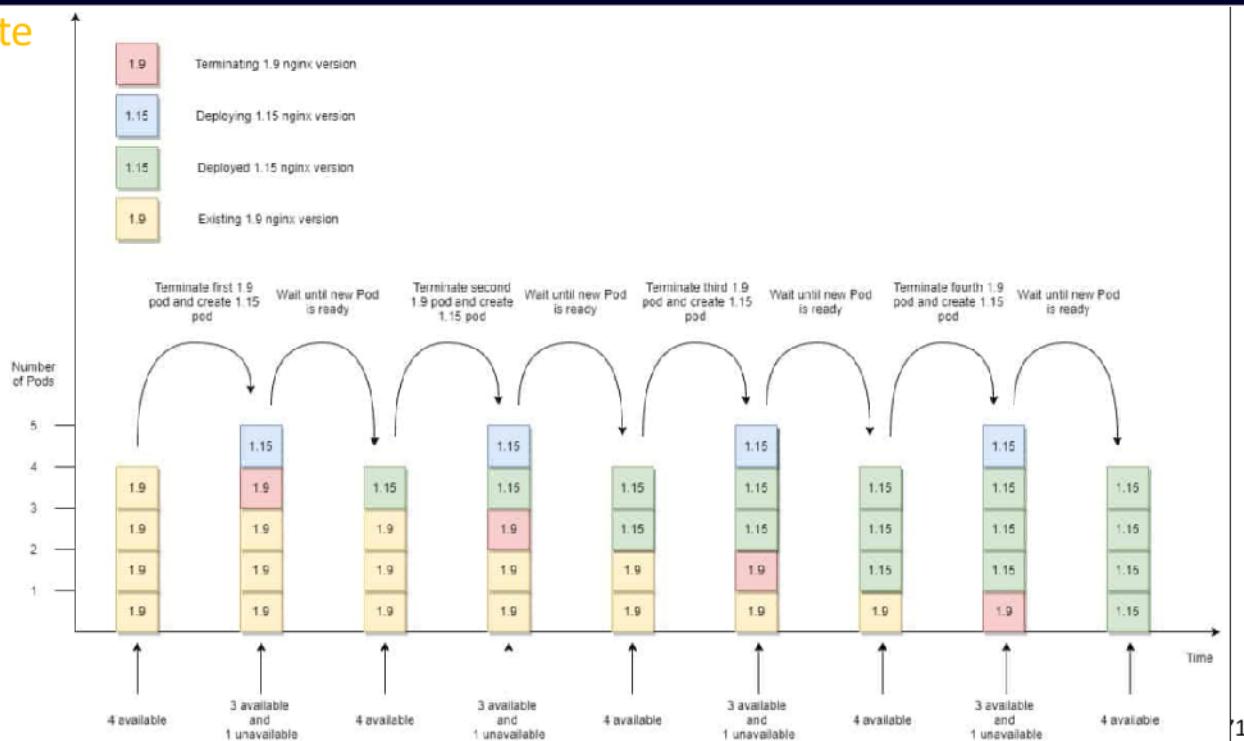
- Il existe deux paramètres que vous pouvez modifier pour contrôler le processus : **maxUnavailable** et **maxSurge**. Les deux ont les valeurs par défaut définies à 25 %
- **maxUnavailable** spécifie le nombre maximal de pods qui peuvent être indisponibles pendant le processus de mise à jour. La valeur peut être un nombre absolu ou un pourcentage des pods souhaités (par défaut 25%).
  - Par exemple, lorsque cette valeur est définie sur 40%, l'ancien ReplicaSet peut être réduit à 60% des pods. Une fois que les nouveaux pods sont prêts, l'ancien ReplicaSet peut être réduit davantage, suivi d'une augmentation du nouveau ReplicaSet, garantissant que le nombre total de pods disponibles à tout moment pendant la mise à jour est d'au moins 60% des pods souhaités.
- **MaxSurge** spécifie le nombre maximal de pods pouvant être créés sur le nombre de pods souhaité.
  - Par exemple, lorsque cette valeur est définie sur 40%, le nouveau ReplicaSet peut être mis à l'échelle de sorte que le nombre total d'anciens et de nouveaux pods ne dépasse pas 140% des pods souhaités. Une fois que les anciens pods ont été détruits, le nouveau ReplicaSet peut être augmenté davantage, garantissant que le nombre total de pods en cours d'exécution à tout moment pendant la mise à jour est au maximum de 140% des pods souhaités.

70

# Déploiement

## Rolling Update

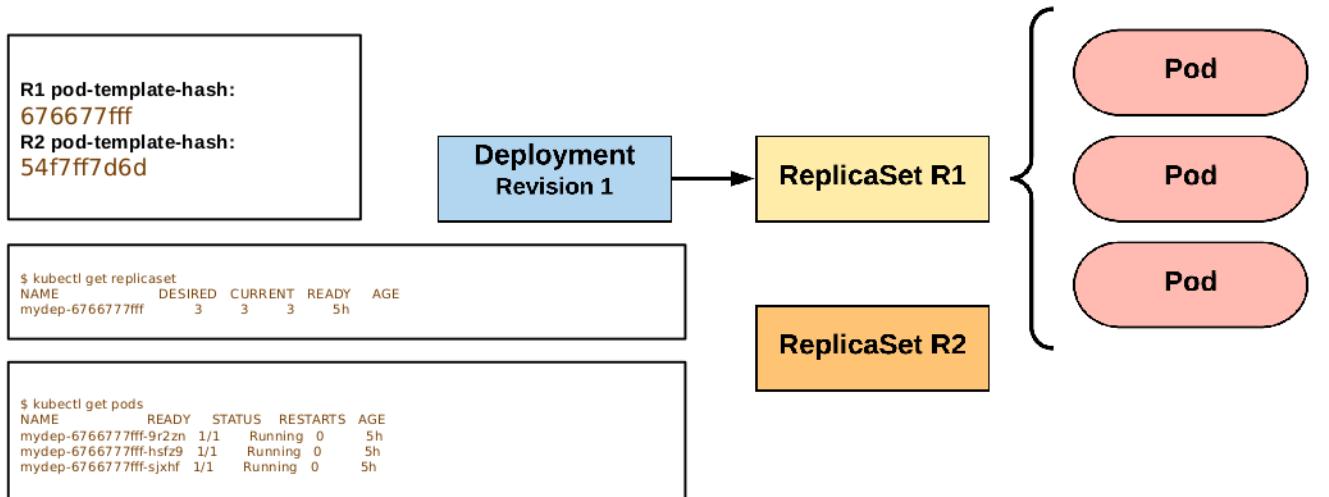
Example 1 :



# Déploiement

## Rolling Update : Example 2

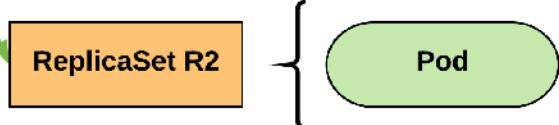
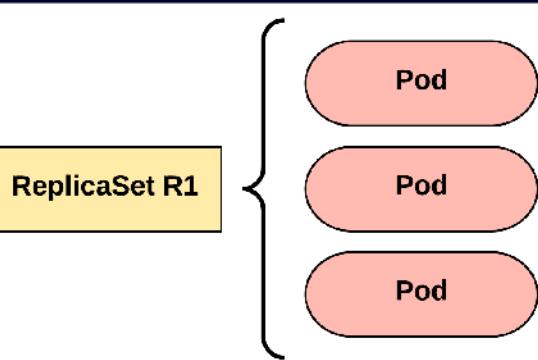
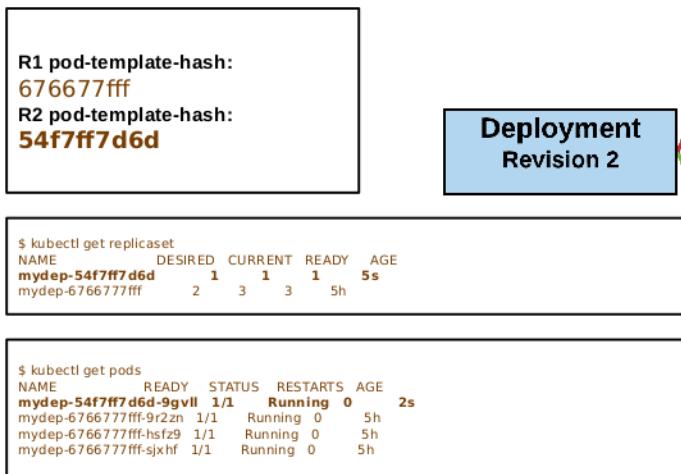
Updating pod template generates a new **ReplicaSet** revision.



# Déploiement

## Rolling Update : Example 2

New **ReplicaSet** is initially scaled up based on [maxSurge](#).

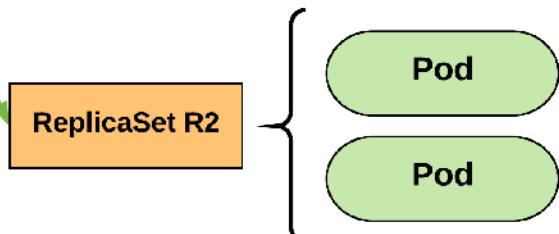
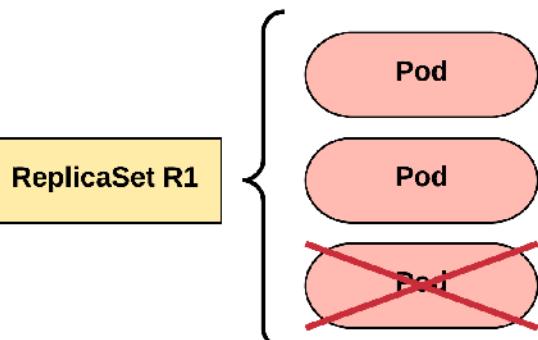
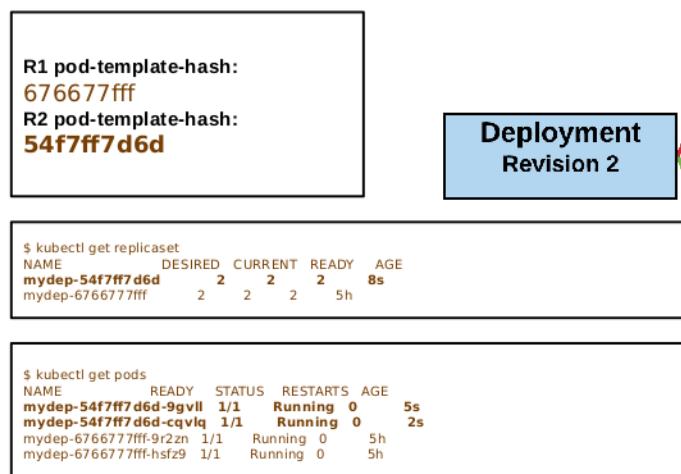


73

# Déploiement

## Rolling Update : Example 2

Phase out of old Pods managed by [maxSurge](#) and [maxUnavailable](#).

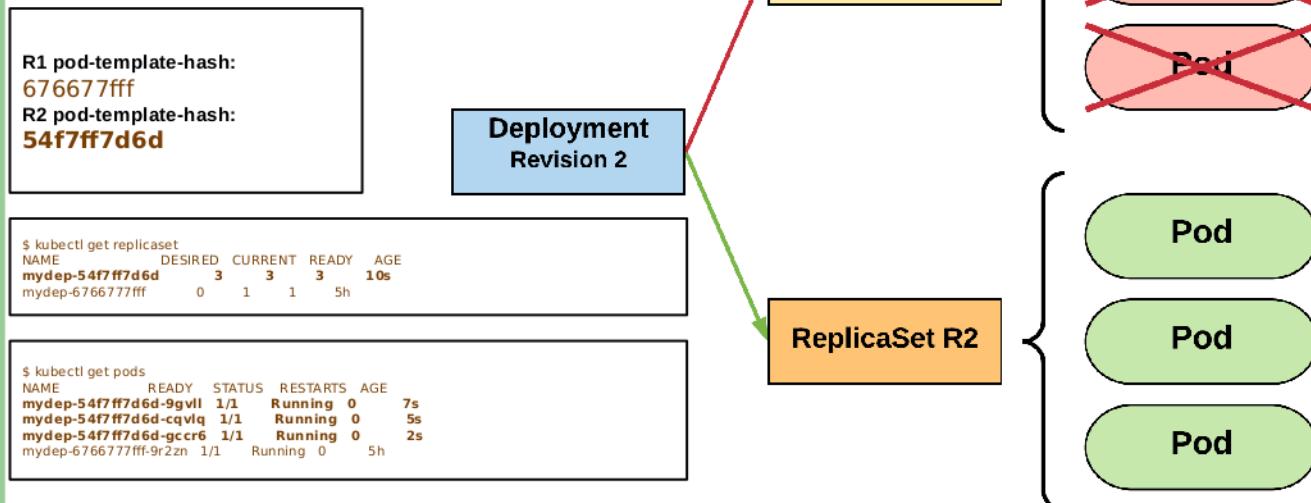


74

# Déploiement

## Rolling Update : Example 2

Phase out of old Pods managed by [maxSurge](#) and [maxUnavailable](#).

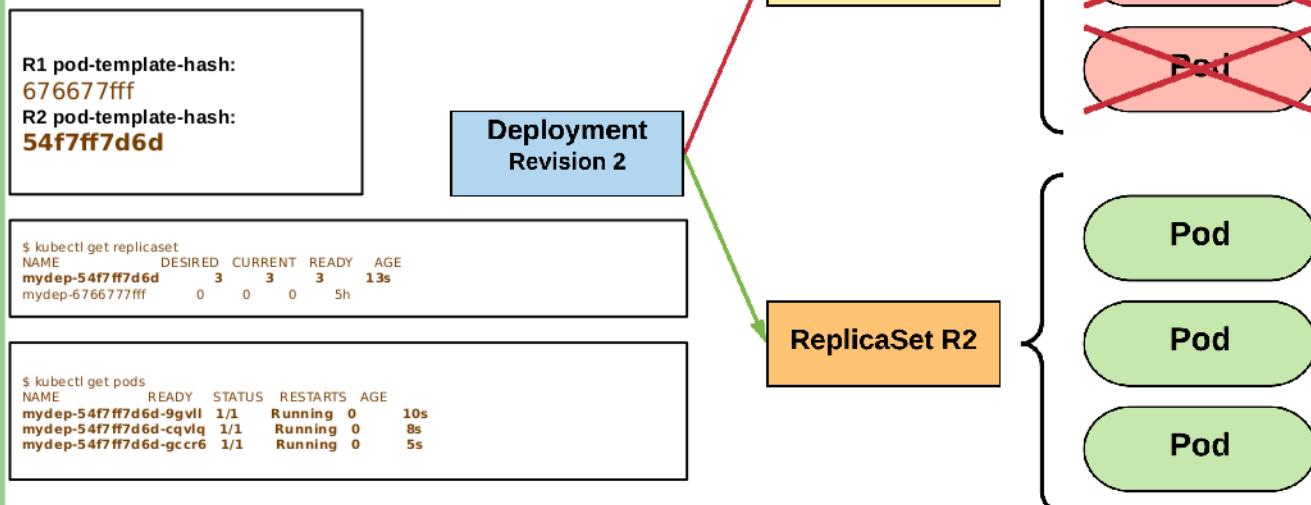


75

# Déploiement

## Rolling Update : Example 2

Phase out of old Pods managed by [maxSurge](#) and [maxUnavailable](#).

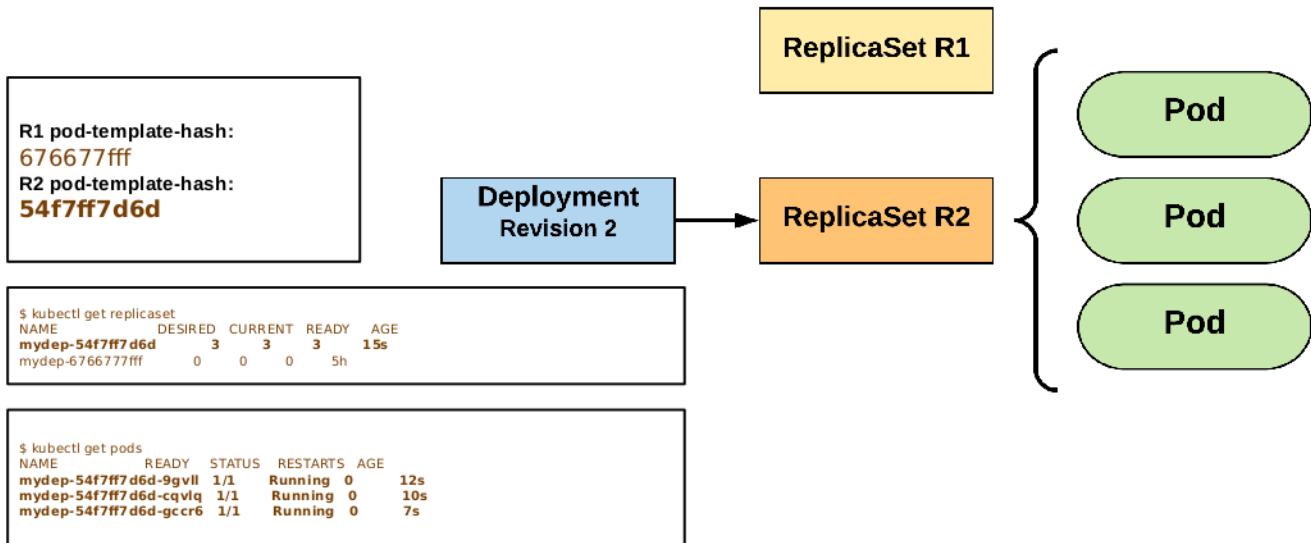


76

# Déploiement

## Rolling Update : Example 2

Updated to new deployment revision completed.

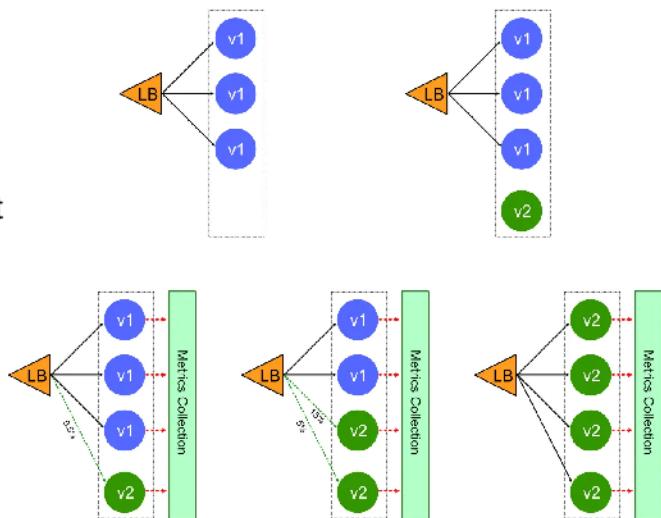


77

# Déploiement

- **Canary :** laissez le consommateur faire le test avant de déployer

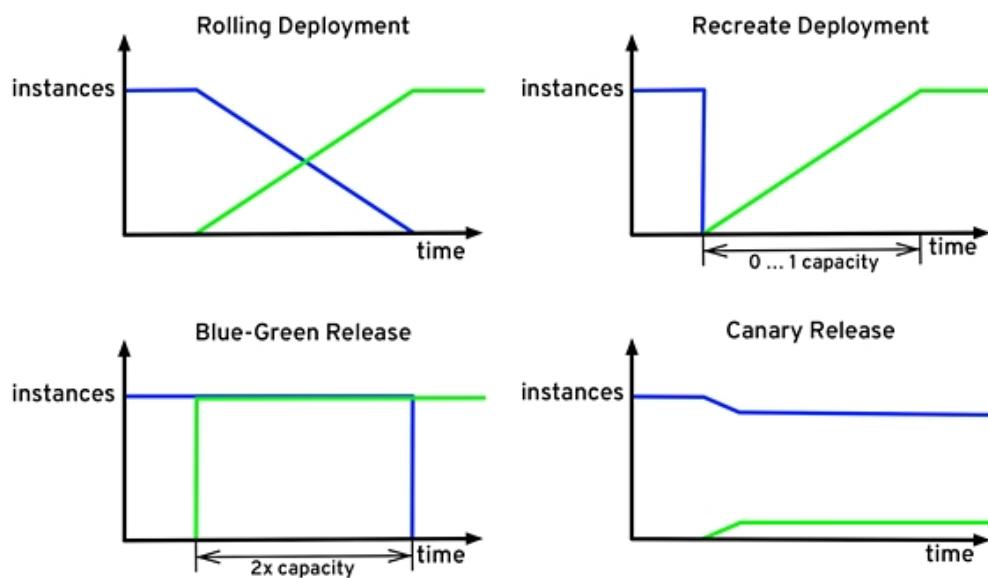
- Dans cette stratégie, la nouvelle version est déployée auprès d'un petit sous-ensemble d'utilisateurs avant d'être déployée auprès de l'ensemble des utilisateurs.
- En fonction des retours et des performances, il est soit progressivement déployé sur le reste, soit annulé.
- Cette approche limite l'impact de tout nouveau bug et permet d'effectuer des tests réels à petite échelle avant un déploiement à grande échelle.
- Il est utile pour les applications où les commentaires des utilisateurs ou les données de performances réelles sont cruciaux pour la validation.



78

# Déploiement

## Stratégie de déploiement



79

# Déploiement

- `kubectl create -f deployment.yaml`

```
root@k-master:/home/osboxes# kubectl create -f deployment.yaml
deployment.apps/nginx-deployment created
root@k-master:/home/osboxes#
```

- `kubectl get deployments`

```
root@k-master:/home/osboxes# kubectl get deployments -o wide
NAME           READY   UP-TO-DATE   AVAILABLE   AGE      CONTAINERS
nginx-deployment  5/5     5           5          117s    nginx-container
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx-container
          image: nginx
          ports:
            - containerPort: 80
```

80

# Déploiement

- `kubectl describe deployment <deployment-name>`

```
root@k-master:/home/osboxes# kubectl describe deployment nginx-deployment
Name:           nginx-deployment
Namespace:      default
CreationTimestamp:  Tue, 19 May 2020 03:40:19 -0400
Labels:          app=nginx
Annotations:    deployment.kubernetes.io/revision: 1
                  kubernetes.io/change-cause: kubectl create --filename=deployment.yaml
Selector:        app=nginx
Replicas:       5 desired | 5 updated | 5 total | 5 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
    nginx-container:
      Image:  nginx
      Port:   80/TCP
      Host Port:  80/TCP
      Environment: <none>
      Mounts:  <none>
      Volumes: <none>
  Conditions:
    Type     Status  Reason
    ----  -----
    Available  True    MinimumReplicasAvailable
    Progressing  True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  nginx-deployment-96577bc6d (5/5 replicas created)
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx-container
          image: nginx
          ports:
            - containerPort: 80
```

81

# Déploiement

- `kubectl get pods -o wide`

```
root@k-master:/home/osboxes# kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE     IP          NODE
TES
nginx-deployment-96577bc6d-2hkpk  1/1    Running   0          3m34s  10.244.2.20  k-slave02
nginx-deployment-96577bc6d-h5gdv  1/1    Running   0          3m34s  10.244.2.19  k-slave02
nginx-deployment-96577bc6d-nqtn6  1/1    Running   0          3m34s  10.244.2.22  k-slave02
nginx-deployment-96577bc6d-pd4cg  1/1    Running   0          3m34s  10.244.2.21  k-slave02
nginx-deployment-96577bc6d-tphhh  1/1    Running   0          3m34s  10.244.2.23  k-slave02
root@k-master:/home/osboxes#
```

- `kubectl edit deployment <deployment -name>` ⇒ perform live edit of deployment
- `kubectl scale deployment <deployment -name> --replicas2`
- `kubectl apply -f deployment.yaml` ⇒ redeploy a modified yaml file; Ex: replicas changed to 5, image to nginx:1.18

82

# Déploiement

- `kubectl rollout status deployment <deployment -name>`

```
root@k-master:/home/osboxes# k rollout status deployment.apps/nginx-deployment
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 2 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 4 of 5 updated replicas are available...
deployment "nginx-deployment" successfully rolled out
```

surveillez l'état du déploiement jusqu'à ce qu'il soit terminé.

- `kubectl rollout history deployment <deployment -name>`

```
root@k-master:/home/osboxes# k rollout history deployment.apps/nginx-deployment
deployment.apps/nginx-deployment
REVISION  CHANGE-CAUSE
1          kubectl create --filename=deployment.yml --record=true
2          kubectl create --filename=deployment.yml --record=true
```

Afficher les révisions et configurations de déploiement précédentes.

83

# Déploiement

- `kubectl rollout undo deployment <deployment -name>` ⇒ Revenir à un déploiement précédent.

```
root@k-master:/home/osboxes# kubectl rollout undo deployment.apps/nginx-deployment
deployment.apps/nginx-deployment rolled back
root@k-master:/home/osboxes# k rollout history deployment.apps/nginx-deployment
deployment.apps/nginx-deployment
REVISION  CHANGE-CAUSE
2          kubectl create --filename=deployment.yml --record=true
3 ←       kubectl create --filename=deployment.yml --record=true
```

- `kubectl rollout undo deployment <deployment -name> --to-revision=1`
- `kubectl rollout pause deployment <deployment -name>` ⇒ Marquez la ressource fournie en pause.
- `kubectl rollout resume deployment <deployment -name>` ⇒ Reprendre une ressource en pause
- `kubectl delete -f <deployment-yaml-file>` ⇒ supprime le déploiement et les dépendances associées
- `kubectl delete all --all` ⇒ supprime pods, replicaset, deployments and services in current namespace

84

# Logs

```
kubectl logs my-pod                                # dump pod logs (stdout)
kubectl logs -l name=myLabel                         # dump pod logs, with label name=myLabel (stdout)
kubectl logs my-pod -c my-container                  # dump pod container logs (stdout, multi-container case)
kubectl logs -l name=myLabel -c my-container          # dump pod logs, with label name=myLabel (stdout)
kubectl logs -f my-pod                               # stream pod logs (stdout)
kubectl logs -f my-pod -c my-container               # stream pod container logs (stdout, multi-container case)
kubectl logs -f -l name=myLabel --all-containers    # stream all pods logs with label name=myLabel (stdout)
kubectl logs my-pod -f --tail=1                      # stream last line of pod logs
kubectl logs deploy/<deployment-name>              # dump deployment logs
```

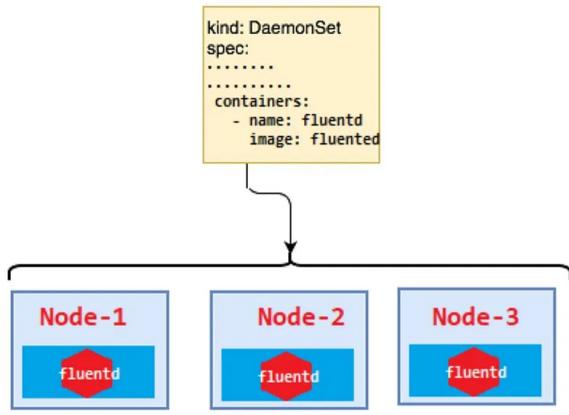
85

# DaemonSet

86

# DaemonSet

- Garantit que la copie d'un pod défini dans la configuration est toujours disponible sur chaque nœud Worker d'un cluster.
- Quand un nouveau nœud est ajouté à un cluster, le DaemonSet crée automatiquement le pod sur ce nœud.
- De même, lorsqu'un nœud est supprimé, le pod qui s'exécute sur le nœud est également supprimé et n'est pas replanifié sur un autre nœud .
- Cela vous permet de vous assurer qu'une application spécifique est déployée sur tous les nœuds du cluster.
- Par exemple, vous pouvez les utiliser afin de déployer des pods pour effectuer des tâches de maintenance et prendre en charge des services sur chaque nœud :
  - Exécutez un démon de collection de journaux, tel que Logstash et Fluentd.
  - Exécutez un démon de surveillance de nœud, tel que Prometheus.
  - Exécutez un démon de stockage de cluster, tel que glusterd ou ceph.



87

# DaemonSet

```
$ minikube start --nodes=3
$ kubectl get nodes
NAME      STATUS   ROLES      AGE VERSION
minikube   Ready    control-plane   62s v1.27.4
minikube-m02 Ready    <none>     45s v1.27.4
minikube-m03 Ready    <none>     31s v1.27.4

$ kubectl apply -f fluentd.yaml
daemonset.apps/fluentd created

$ kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP          NODE
fluentd-jn24d 1/1    Running   0          2m10s  10.244.1.2  minikube-m02
fluentd-pzmjh 1/1    Running   0          2m10s  10.244.2.2  minikube-m03
fluentd-zcq57  1/1    Running   0          2m10s  10.244.0.3  minikube

$ kubectl get daemonset
NAME   DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR  AGE
fluentd  3         3        3      3           <none>     3m55s
```

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd
spec:
  selector:
    matchLabels:
      name: fluentd
  template:
    metadata:
      labels:
        name: fluentd
    spec:
      containers:
        - name: fluentd
          image: fluentd/fluentd
```

fluentd.yaml

# Scheduling

89

## Scheduling

- Les utilisateurs de Kubernetes n'ont normalement pas besoin de choisir un nœud sur lequel leurs pods doivent être planifiés
- Au lieu de cela, la sélection du ou des nœuds appropriés est automatiquement gérée par le Scheduler Kubernetes.
- La sélection automatique des nœuds empêche les utilisateurs de sélectionner des nœuds en mauvais état ou des nœuds manquant de ressources.
- Cependant, une planification manuelle ( manual scheduling) est parfois nécessaire pour garantir que certains pods sont planifiés uniquement sur des nœuds dotés de matériel spécialisé comme des stockages SSD, ou pour colocaliser des services qui communiquent fréquemment (zones de disponibilité), ou pour dédier un ensemble de nœuds à un ensemble particulier d'utilisateurs.
- Kubernetes propose plusieurs façons de planifier manuellement les pods. Dans tous les cas, l'approche recommandée consiste à utiliser des sélecteurs d'étiquettes pour effectuer la sélection.
- Les options de planification manuelle incluent :
  1. nodeName
  2. nodeSelector
  3. Node affinity
  4. Taints and Tolerations

<https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/>

90

# Scheduling

## nodeName

- nodeName est un champ de PodSpec
- nodeName est la forme la plus simple de contrainte de sélection de nœud, mais en raison de ses limitations, elle n'est généralement pas utilisée
- Lorsque le scheduler ne trouve aucune propriété nodeName, il l'ajoute automatiquement et attribue le pod à n'importe quel noeud disponible
- Attribuez manuellement un pod à un nœud en écrivant la propriété nodeName avec le nom de nœud souhaité.
- Nous pouvons également programmer des pods sur Master par cette méthode
- Certaines des limitations liées à l'utilisation de nodeName pour sélectionner des nœuds sont :
  - Si le nœud nommé n'existe pas, le pod ne sera pas exécuté et, dans certains cas, pourra être automatiquement supprimé.
  - Si le nœud nommé ne dispose pas des ressources nécessaires pour accueillir le pod, le pod échouera et sa raison indiquera pourquoi, par exemple OutOfMemory ou OutOfCPU
  - Les noms de nœuds dans les environnements cloud ne sont pas toujours prévisibles ou stables

91

# Scheduling

## nodeName

nodeName.yml

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP
node/k8s-master	Ready	master	5d	v1.18.3	192.168.0.108	<none>
node/k8s-slave01	Ready	<none>	5d	v1.18.3	192.168.0.110	<none>
node/k8s-slave02	Ready	<none>	5d	v1.18.3	192.168.0.109	<none>

kubectl apply -f nodeName.yml

```
root@k8s-master:/home/osboxes/demo_kubernetes# kubectl apply -f nodeName.yml
pod/nginx created
root@k8s-master:/home/osboxes/demo_kubernetes# kubectl get pods -o wide
NAME      READY   STATUS    RESTARTS   AGE     IP           NODE   NOMINATED-NODE   READINESS   GATES
nginx    1/1     Running   0          44s    10.244.0.26   k8s-master   <none>        <none>
root@k8s-master:/home/osboxes/demo_kubernetes#
```

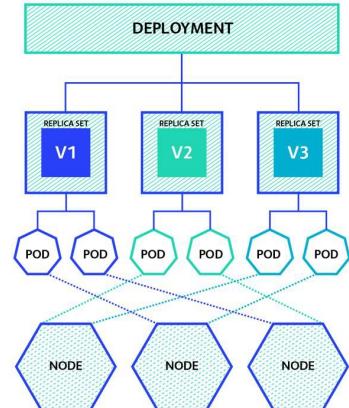
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
nodeName: k8s-master
```

92

# Scheduling

## nodeSelector

- nodeSelector est un champ de PodSpec
- Il s'agit de la forme recommandée la plus simple de contrainte de sélection de nœud.
- Il utilise des étiquettes (paires clé-valeur) pour sélectionner les nœuds correspondants sur lesquels les pods peuvent être planifiés.
- L'inconvénient de nodeSelector est qu'il utilise des préférences strictes, c'est-à-dire que si les nœuds correspondants ne sont pas disponibles, les pods restent en attente (**pending state**)



## vérifier les étiquettes par défaut des nœuds

kubectl describe node <node-name>

```
root@k8s-master:/home/osboxes/demo_kubernetes# k describe node k8s-master | grep -i label
Labels:           beta.kubernetes.io/arch=amd64
root@k8s-master:/home/osboxes/demo_kubernetes#
```

93

# Scheduling

## nodeSelector

### Add labels to nodes

kubectl label nodes <node-name> <label-key>=<label-value>

kubectl label nodes k8s-slave01 environment=dev

```
root@k8s-master:/home/osboxes/demo_kubernetes# kubectl describe node k8s-slave01 | grep -i labels -A 5
Labels:           beta.kubernetes.io/arch=amd64
                  beta.kubernetes.io/os=linux
                  environment=dev
                  kubernetes.io/arch=amd64
                  kubernetes.io/hostname=k8s-slave01
                  kubernetes.io/os=linux
root@k8s-master:/home/osboxes/demo_kubernetes#
```

delete a label: **kubectl label nodes <node-name> <labelname>- add - at the end of the command**

Ex : **kubectl label nodes k8s-slave01 environment=dev-**

94

# Scheduling

## nodeSelector

```
kubectl apply -f nodeSelector.yml  
kubectl get pods -o wide --show-labels  
kubectl describe pod <pod-name>
```

```
root@k8s-master:/home/osboxes/demo_kubernetes# kubectl apply -f nodeSelector.yml  
pod/nginx created  
root@k8s-master:/home/osboxes/demo_kubernetes# kubectl get pods -o wide --show-labels  
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES LABELS  
nginx 1/1 Running 0 20s 10.244.1.113 k8s-slave01 <none> <none>  
root@k8s-master:/home/osboxes/demo_kubernetes# kubectl describe pod nginx | grep -i selector -A 5  
Node-Selectors: environment=dev  
Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s  
node.kubernetes.io/unreachable:NoExecute for 300s  
Events:  
Type Reason Age From Message  
---- ---- -- -- --  
root@k8s-master:/home/osboxes/demo_kubernetes#
```

nodeSelector.yml

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: nginx  
  labels:  
    env: test  
spec:  
  containers:  
  - name: nginx  
    image: nginx  
  nodeSelector:  
    environment: dev
```

95

# Scheduling

## nodeAffinity

- L'affinité du nœud est spécifiée sous forme du champ `nodeAffinity` dans `PodSpec`
- L'affinité de nœud est conceptuellement similaire à `nodeSelector`, elle vous permet de planifier manuellement des pods en fonction des étiquettes du nœud. Mais il présente quelques améliorations clés :
  - L'implémentation de `nodeAffinity` est plus expressive. Le langage propose davantage de règles de correspondance en plus des correspondances exactes créées avec une opération AND logique dans `nodeSelector`
  - Les règles sont des préférences souples plutôt que des exigences strictes. Ainsi, si le planificateur ne parvient pas à trouver un nœud avec des étiquettes correspondantes, le pod sera toujours planifié sur d'autres nœuds.
- Il existe actuellement deux types de règles d'affinité de nœud :
  - `RequiredDuringSchedulingIgnoredDuringExecution` : exigence stricte comme `nodeSelector`. Si aucune d'étiquette de nœud correspondante alors pas de planification de pod
  - `preferredDuringSchedulingIgnoredDuringExecution` : exigence soft. Si aucune étiquette de nœud correspondante, alors le pod est planifié sur d'autres nœuds.
- La partie `IgnoredDuringExecution` indique que si les étiquettes d'un nœud changent au moment de l'exécution de telle sorte que les règles d'affinité sur un pod ne sont plus respectées, le pod continuera à s'exécuter sur le nœud.

96

# Scheduling

## nodeAffinity

```
kubectl apply -f nodeAffinity.yml
```

- Le pod est planifié sur le nœud portant l'étiquette `environment=production`
- Si aucun des nœuds ne possède cette étiquette, le pod reste en état d'attente (`pending` state).
- Pour éviter cela, utilisez l'affinité `preferredDuringSchedulingIgnoredDuringExecution`

```
nodeAffinity.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: environment
                operator: In
                values:
                  - prod
      containers:
        - name: nginx-container
          image: nginx
```

<https://kubernetes.io/docs/tasks/configure-pod-container/assign-pods-nodes-using-node-affinity/>  
<https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/>

97

# Scheduling

## Taints and Tolerations

- Les Taints permettent l'inverse d'une affinité: elles permettent à un nœud de refuser un ensemble de pods.
- C'est simple, ajouter une Taint à un nœud revient à appliquer une mauvaise odeur sur ce nœud. Seuls les Pods qui tolèrent cette mauvaise odeur pourront être exécutés sur le nœud.
- Tout comme les Labels, une ou plusieurs Taints peuvent être appliquées à un nœud; cela signifie que le nœud ne doit accepter aucun pod qui ne tolère pas l'ensemble de ces Taints.
- Les Taints sont appliquées aux nœuds (verrouillage)
- Les tolerances sont appliquées aux pods (clés)
- En bref, le pod doit tolérer la taint du nœud afin de pouvoir y fonctionner. C'est comme avoir une clé correcte avec un padlock pour déverrouiller le nœud et y entrer
- Les taints et les tolerances fonctionnent ensemble pour garantir que les pods ne sont pas planifiés sur des nœuds inappropriés.



98

# Scheduling

## Taints and Tolerations

- Par défaut, le nœud maître est verrouillé Vous ne pouvez donc déployer aucun pod sur Master
- Pour vérifier les teintes appliquées sur n'importe quel nœud, utilisez
  - `kubectl describe node <node-name>`

### Appliquer taint aux nœuds

```
kubectl taint nodes <node-name> key=value:<taint-effect>
```

- la clé et la valeur de Taint peuvent être n'importe quelle chaîne arbitraire
  - Ex : `kubectl taint nodes node1 key1=value1:NoSchedule`
- L'effet de taint doit être l'un des effets de Taint pris en charge, tels que
  1. **NoSchedule** : L'exécution des pods non tolérants à la Taint ne sera pas planifiée sur ce nœud. C'est une **contrainte forte**.
  2. **PreferNoSchedule** : Le Scheduler évitera de placer un Pod qui ne tolère pas la Taint sur le nœud, mais ce n'est pas obligatoire. C'est une **contrainte douce**
  3. **NoExecute** : Le pod non tolérants à la Taint sera expulsé du nœud (s'il est déjà en cours d'exécution sur le nœud) ou ne sera pas planifié sur le nœud (s'il n'est pas encore exécuté sur le nœud). C'est une **contrainte forte**.

99

# Scheduling

## Taints and Tolerations

### Apply taint to nodes

```
kubectl taint nodes k8s-slave01 env=stag:NoSchedule
```

```
root@k8s-master:/home/osboxes/demo_kubernetes# kubectl taint nodes k8s-slave01 env=stag:NoSchedule
node/k8s-slave01 tainted
root@k8s-master:/home/osboxes/demo_kubernetes#
```

Dans le cas ci-dessus, le nœud k8s-slave01 est marqué avec l'étiquette env=stag et l'effet de teinte est **NoSchedule**. Seuls les pods correspondant à cette teinte seront planifiés sur ce nœud.

### Check taints on nodes

```
kubectl describe node k8s-slave01 | grep -i taint
```

```
root@k8s-master:/home/osboxes/demo_kubernetes# kubectl describe node k8s-slave01 | grep -i taint
Taints:           env=stag:NoSchedule
root@k8s-master:/home/osboxes/demo_kubernetes#
```

### To untaint a node

```
kubectl taint nodes k8s-slave01 env=stag:NoSchedule- add - at the end of the command
```

100

# Scheduling

## Taints and Tolerations

### Apply tolerations to pods

```
kubectl apply -f taint_toleration.yml
```

```
kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS	GATES
nginx-deployment-547c5b4448-jg8v8	1/1	Running	0	32s	10.244.1.117	k8s-slave01	<none>	<none>	<none>
nginx-deployment-547c5b4448-k59ef	1/1	Running	0	32s	10.244.1.118	k8s-slave01	<none>	<none>	<none>
nginx-deployment-547c5b4448-ngl88	1/1	Running	0	32s	10.244.2.109	k8s-slave02	<none>	<none>	<none>

- Ici, les pods sont planifiés sur les deux nœuds slaves.
- Seul slave01 est tainté ici et les tolérances correspondantes sont ajoutées aux pods. Les pods sont donc également programmés sur slave-01.
- Si nous supprimons les tolérances des pods, elles seront planifiées sur slave-02 uniquement car slave01 est tainté

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
    spec:
      containers:
        - name: nginx-container
          image: nginx
          tolerations:
            - key: "env"
              operator: "Equal"
              value: "stag"
              effect: "NoSchedule"
```

101

# Kubernetes Volumes

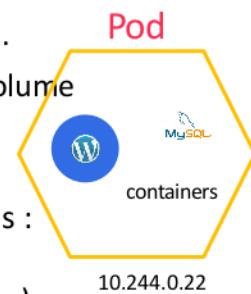
102

# Volumes

- Par défaut, les données du conteneur sont stockées dans leur propre système de fichiers
- Mais lorsqu'ils sont détruits, les données qu'ils contiennent sont supprimées.
- De plus, lors de l'exécution de plusieurs conteneurs dans un pod, il est souvent nécessaire de partager des fichiers entre ces conteneurs.
- Afin de conserver les données au-delà du cycle de vie du pod, Kubernetes fournit des volumes
- Un volume peut être considéré comme un répertoire accessible aux conteneurs d'un pod.
- Le support sur lequel repose un volume et son contenu sont déterminés par le type de volume

## Types de volumes Kubernetes

- Il existe différents types de volumes que vous pouvez utiliser dans un pod Kubernetes :
  - ◆ **Node-local memory** `emptyDir` and `hostPath`
  - ◆ **Cloud volumes** (e.g., `awsElasticBlockStore`, `gcePersistentDisk`, and `azureDiskVolume`)
  - ◆ **File-sharing volumes**, such as Network File System (NFS)
  - ◆ **Distributed-file systems** (e.g., CephFS and GlusterFS)
  - ◆ **Special volume types** such as `PersistentVolumeClaim`, `secret`, `configmap` and `gitRepo`

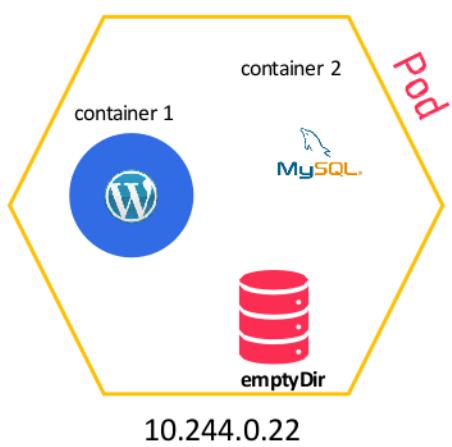


103

# Volumes

## emptyDir

- Le volume `emptyDir` est créé pour la première fois lorsqu'un pod est attribué à un nœud
- Il est initialement vide et a la même durée de vie qu'un pod
- Les volumes `emptyDir` sont stockés sur le support sur lequel repose le nœud : il peut s'agir d'un disque, d'un SSD, d'un stockage réseau ou d'une RAM.
- Les conteneurs du Pod peuvent tous lire et écrire les mêmes fichiers dans le volume `emptyDir`
- Ce volume peut être monté sur des chemins identiques ou différents dans chaque conteneur
- Lorsqu'un **pod est supprimé** d'un nœud pour une raison quelconque, les **données du emptyDir sont supprimées définitivement**
- Principalement utilisé pour stocker le cache ou les données temporaires à traiter



104

# Volumes

## emptyDir

```
kubectl apply -f emptyDir-demo.yml
```

```
kubectl exec -it pod/emptydir-pod -c container-2 -- cat /cache/date.txt  
kubectl logs pod/emptydir-pod -c container-2
```

```
root@k8s-master:/home/osboxes/demo_kubernetes# k exec -it pod/emptydir-pod -c container-2 -- cat /cache/date.txt  
Tue Jun 2 00:10:17 UTC 2020  
root@k8s-master:/home/osboxes/demo_kubernetes# k logs pod/emptydir-pod -c container-2  
Tue Jun 2 08:10:17 UTC 2020  
root@k8s-master:/home/osboxes/demo_kubernetes#
```

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: emptydir-pod  
  labels:  
    app: busybox  
    purpose: emptydir-demo  
spec:  
  volumes:  
  - name: cache-volume  
    emptyDir: {}  
  containers:  
  - name: container-1  
    image: busybox  
    command: ["/bin/sh", "-c"]  
    args: ["date >> /cache/date.txt; sleep 1000"]  
    volumeMounts:  
    - mountPath: /cache  
      name: cache-volume  
  - name: container-2  
    image: busybox  
    command: ["/bin/sh", "-c"]  
    args: ["cat /cache/date.txt; sleep 1000"]  
    volumeMounts:  
    - mountPath: /cache  
      name: cache-volume
```

105

# Volumes

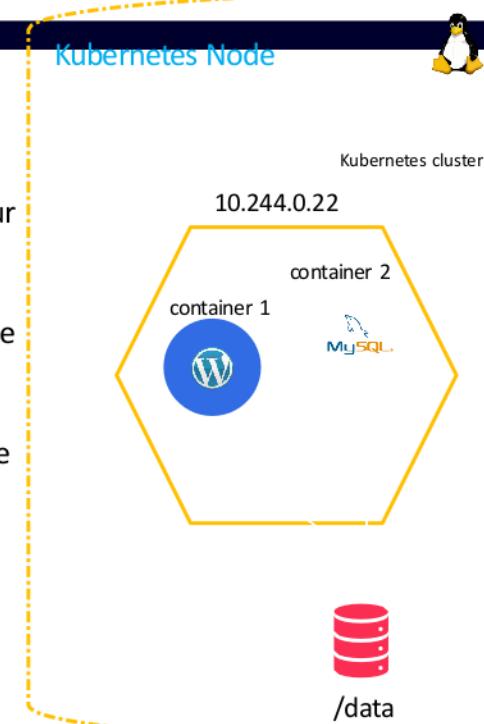
## hostPath

- Ce type de volume monte un fichier ou un répertoire du système de fichiers du nœud hôte dans votre pod
- Le répertoire hostPath fait référence au répertoire créé sur le nœud sur lequel le pod est exécuté
- **Utilisez-le avec prudence**, car lorsque les pods sont planifiés sur plusieurs nœuds, chaque nœud obtient son propre volume de stockage hostPath. Ceux-ci peuvent ne pas être synchronisés les uns avec les autres et différents pods peuvent utiliser des données différentes.
- Supposons que le pod avec la configuration hostPath est déployé sur le nœud Worker 2. Ensuite, host fait référence au nœud Worker 2. Ainsi, tout emplacement hostPath mentionné dans le fichier manifeste fait référence au nœud Worker 2 uniquement.
- Lorsque le nœud devient instable, alors les pods peuvent ne pas parvenir à accéder au répertoire hostPath et finissent par être arrêtés

Kubernetes Node



Kubernetes cluster



<https://kubernetes.io/docs/concepts/storage/volumes/#hostpath>

106

# Volumes

## hostPath

- En plus de la propriété requise path, un utilisateur peut optionnellement spécifier un type pour un volume hostPath.
- Les valeurs supportées pour le champ type sont les suivantes :
  - "" : Une chaîne de caractères vide (par défaut) sert à la rétrocompatibilité, ce qui signifie qu'aucune vérification ne sera effectuée avant de monter le volume hostPath.
  - DirectoryOrCreate : Si rien n'existe au chemin fourni, un dossier vide y sera créé au besoin avec les permissions définies à 0755, avec le même groupe et la même possession que Kubelet.
  - Directory : Un dossier doit exister au chemin fourni
  - FileOrCreate : Si rien n'existe au chemin fourni, un fichier vide y sera créé au besoin avec les permissions définies à 0644, avec le même groupe et la même possession que Kubelet.
  - File: Un fichier doit exister au chemin fourni

<https://kubernetes.io/docs/concepts/storage/volumes/#hostpath>

Kubernetes Node



Kubernetes cluster

10.244.0.22



MySQL



/data

107

# Volumes

## hostPath

kubectl apply -f hostPath-demo.yml

```
kubectl logs pod/hostpath-pod -c container-1  
kubectl exec -it pod/hostpath-pod -c container-1 -- ls /cache
```

```
root@k8s-master:/home/osboxes/demo_kubernetes# kubectl logs pod/hostpath-pod -c container-1  
1.txt  
2.txt  
3.txt  
root@k8s-master:/home/osboxes/demo_kubernetes# kubectl exec -it pod/hostpath-pod -c container-1 -- ls /cache  
1.txt 2.txt 3.txt  
root@k8s-master:/home/osboxes/demo_kubernetes#
```

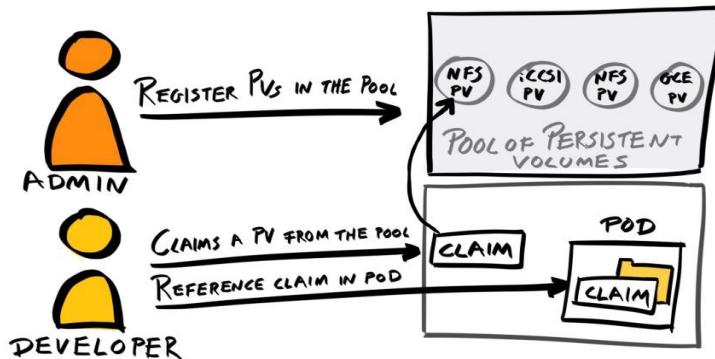
```
apiVersion: v1  
kind: Pod  
metadata:  
  name: hostpath-pod  
spec:  
  volumes:  
    - name: hostpath-volume  
      hostPath:  
        path: /data  
      type: DirectoryOrCreate  
  containers:  
    - name: container-1  
      image: busybox  
      command: ["/bin/sh","-c"]  
      args: ["ls /cache; sleep 1000"]  
      volumeMounts:  
        - mountPath: /cache  
          name: hostPath-volume
```

108

# Volumes

## Persistent Volume et Persistent Volume Claim

- La gestion du stockage est un problème distinct au sein d'un cluster. Vous ne pouvez pas compter sur emptyDir ou hostPath pour les données persistantes.
- Pour surmonter ce problème, le **PersistentVolume** fournit une API aux utilisateurs et aux administrateurs qui extrait les détails de la façon dont le stockage est fourni et de la façon dont il est consommé. Pour ce faire, K8s propose deux API : **PersistentVolume** et **PersistentVolumeClaim**



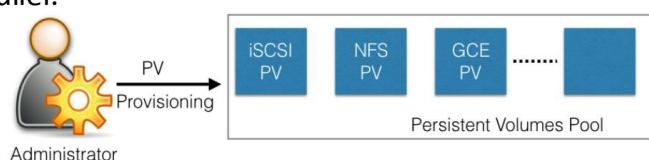
109

# Volumes

## Persistent Volume et Persistent Volume Claim

### Persistent volume (PV)

- Un PersistentVolume (PV) est un élément de stockage dans le cluster qui a été provisionné par un administrateur ou provisionné dynamiquement à l'aide de classes de stockage (approvisionneurs et paramètres prédéfinis pour créer un volume persistant).
- L'administrateur crée un pool de PV parmi lesquels les utilisateurs peuvent choisir
- Il s'agit d'une ressource à l'échelle du cluster utilisée pour stocker/conserver les données au-delà de la durée de vie d'un pod.
- PV n'est pas soutenu par un stockage connecté localement sur un nœud de travail, mais par un système de stockage en réseau tel que le stockage des fournisseurs de cloud ou NFS ou un système de fichiers distribué comme Ceph ou GlusterFS.
- Les volumes persistants fournissent un système de fichiers qui peut être monté sur le cluster, sans être associé à un nœud particulier.



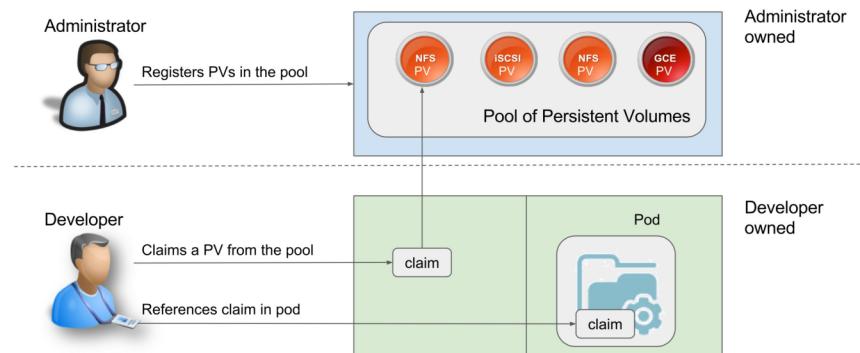
110

# Volumes

## Persistent Volume and Persistent Volume Claim

### Persistent Volume Claim (PVC)

- Pour utiliser un PV, l'utilisateur doit d'abord le demander à l'aide d'un PVC
- PVC demande un PV avec la spécification souhaitée (taille, vitesse, etc.) à Kubernetes, puis le lie à une ressource (pod, déploiement...) en tant que montage de volume
- Les demandes doivent être créées dans le même espace de noms que celui dans lequel le pod est créé.



<https://www.learnitguide.net/2020/03/kubernetes-persistent-volumes-and-claims.html>

111

# Kubernetes Services

112

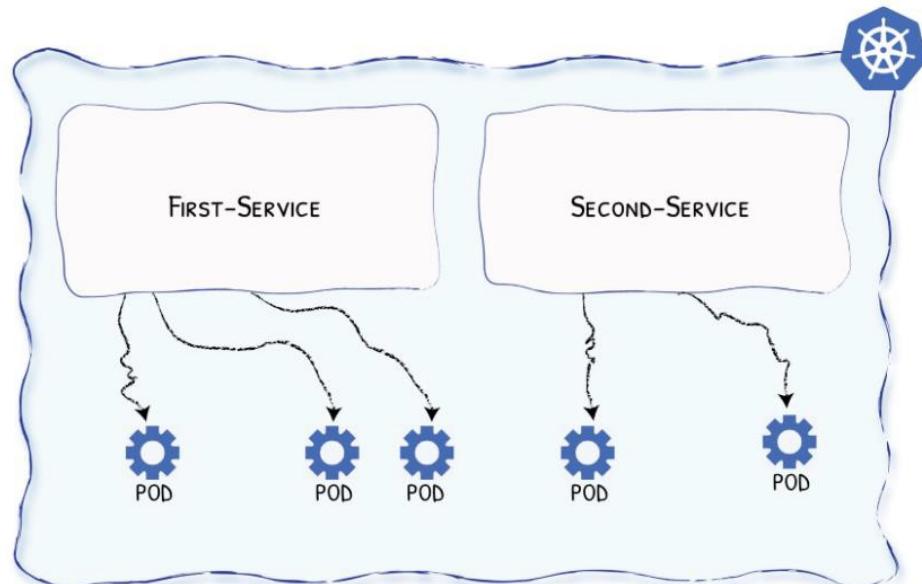
# Services

Le cycle de vie des pods est irrégulier ; ils sont arrêtés et créés par Kubernetes.

Alors, comment pouvez-vous envoyer une demande à votre application si vous ne pouvez pas savoir avec certitude où elle se trouve ?

La réponse réside dans les **services**.

Les services sont liés aux pods à l'aide **d'étiquettes** de pod et fournissent un point final stable permettant aux utilisateurs d'accéder à l'application.

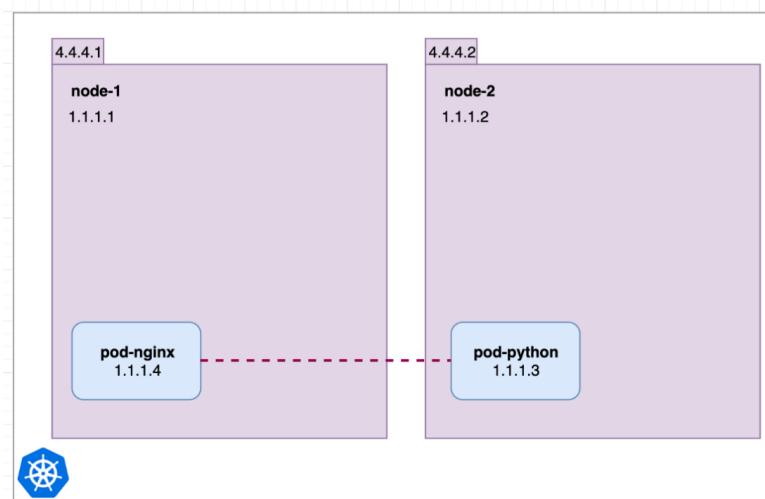


Lorsque vous demandez votre application, vous ne vous souciez pas de son emplacement ni du pod qui répond à la demande.

113

# Services

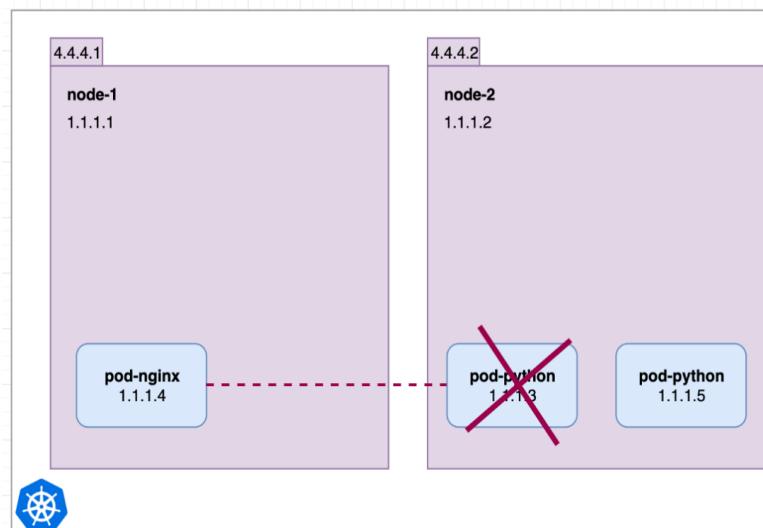
- Supposons nous avons deux nœuds qui ont des adresses IP externes (4.4.4.1, 4.4.4.2) et internes (1.1.1.1, 1.1.1.2).
- Imaginez 2 pods sur 2 nœuds distincts node-1 et node-2 avec leur adresse IP interne
- pod-nginx peut pinguer et se connecter à pod-python en utilisant son IP interne 1.1.1.3.



114

# Services

- Imaginons maintenant que le pod-python meurt et qu'un nouveau soit créé avec une nouvelle IP 1.1.1.5.
- Désormais, pod-nginx ne peut pas atteindre pod-python sur 1.1.1.3 car son adresse IP est modifiée en 1.1.1.5.



- Comment supprimer cette dépendance ?
- pour éviter ce problème de dépendance avec les IP, Kubernetes utilise les Services.
- Ils permettent de localiser un pod en utilisant les **étiquettes** sans passer par son adresse IP.

115

# Services

- Les services connectent les pods à travers le cluster pour permettre la mise en réseau entre eux
- Kubernetes ne considère pas les pods comme des instances uniques et de longue durée ; si un pod rencontre un problème et perdu, Kubernetes le remplacer afin que l'application ne subisse aucun temps d'arrêt
- Les services garantissent que même après la mort d'un pod (back-end) en raison d'une panne, les pods nouvellement créés seront atteints par ses pods de dépendance (front-end) via les services.
- Dans ce cas, les applications front-end trouvent toujours les applications back-end via un simple service (en utilisant le nom du service ou l'adresse IP), quel que soit leur emplacement dans le cluster.
- **Les services pointent directement vers les pods à l'aide d'étiquettes.** Les services ne pointent pas vers des déploiements ou des ReplicaSets. Ainsi, tous les pods portant le même label sont attachés au même service.
- 3 types: **ClusterIP**, **NodePort** et **LoadBalancer**

116

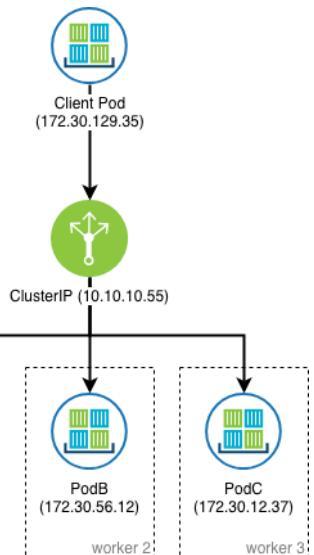
# Services

## ClutserIP

- Le service ClusterIP est le service Kubernetes par défaut.
- Il vous offre un service au sein de votre cluster auquel d'autres applications de votre cluster peuvent accéder
- Il restreint l'accès à l'application au sein du cluster lui-même et aucun accès externe
- Utile lorsqu'une application front-end souhaite communiquer avec le back-end
- Chaque service ClusterIP obtient une adresse IP unique à l'intérieur du cluster
- Contrairement aux pods, un service n'est pas planifié sur un nœud spécifique. Il s'étend sur tout le cluster

Les services pointent directement vers les pods à l'aide d'étiquettes !!!

<https://kubernetes.io/docs/concepts/services-networking/service/>

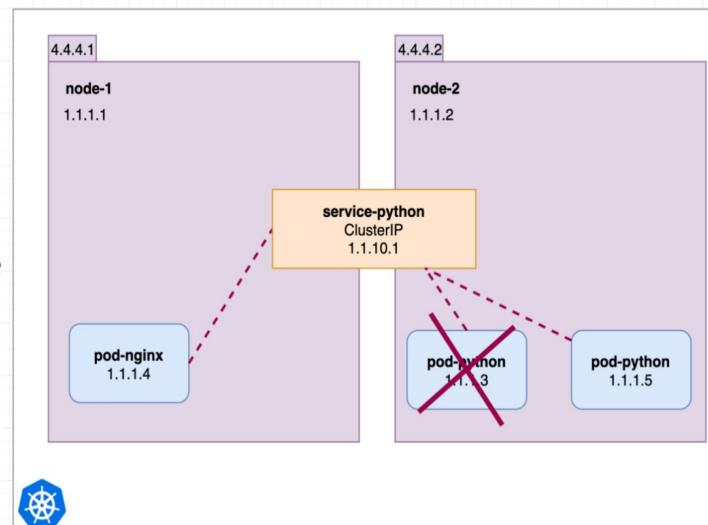


117

# Services

## ClutserIP

- En utilisant le service ClusterIP, Pod-nginx peut toujours se connecter en toute sécurité à pod-python en utilisant l'IP du service 1.1.10.1 ou le nom DNS du service (service-python)
- Même si le pod python est supprimé et recréé à nouveau, le pod nginx peut toujours atteindre le pod python en utilisant le service, mais pas directement avec l'adresse IP du pod python.



118

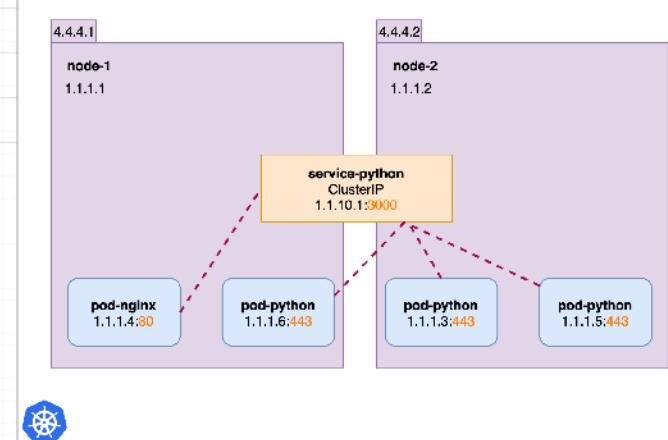
# Services

## ClusterIP

- Nous étendons l'exemple en dupliquant 3 fois le pod python et nous affichons maintenant les ports des adresses IP internes de tous les pods et services.
- Tous les pods du cluster peuvent atteindre les pods python sur leur port 443 via `http://service-python:3000` ou `http://1.1.10.1:3000` (l'utilisation de l'adresse IP de cluster est déconseillé). Le service-python ClusterIP distribue les requêtes sur la base d'une approche aléatoire ou round-robin.
- Le fichier yaml de ce service :

```
kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
service-python	ClusterIP	1.1.10.1	<none>	3000/TCP	54s	run=python



```
apiVersion: v1
kind: Service
metadata:
  name: service-python
spec:
  selector:
    run: pod-python
  type: ClusterIP
  ports:
    - port: 3000
      protocol: TCP
      targetPort: 443
```

facultatif

119

# Services

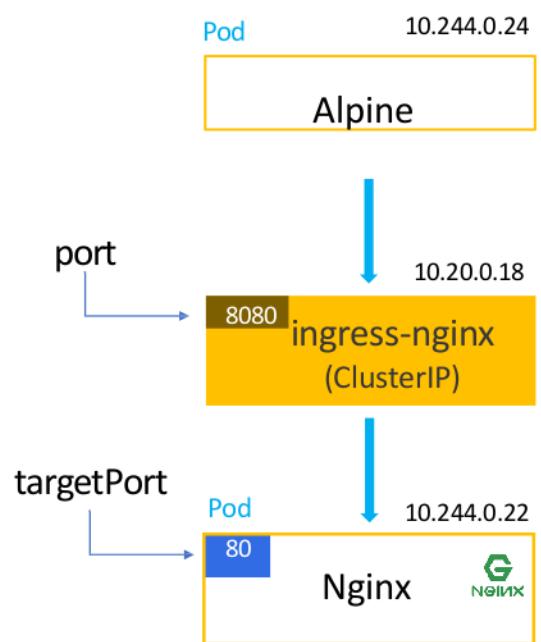
## ClusterIP

clusterservice.yml

```
apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx
spec:
  type: ClusterIP
  ports:
    - name: http
      port: 8080
      targetPort: 80
      protocol: TCP
  selector:
    app: nginx-backend
```

pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: backend-pod
  labels:
    app: nginx-backend
spec:
  containers:
    - name: nginx-container
      image: nginx
      ports:
        - containerPort: 80
```



120

# Services

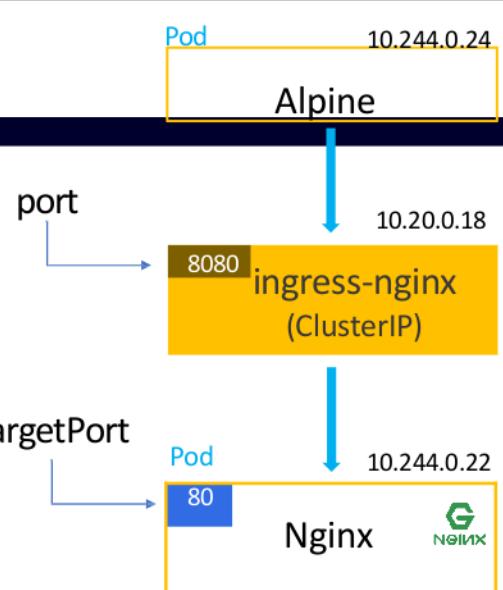
## ClusterIP

clusterservice.yml

```
apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx
spec:
  type: ClusterIP
  ports:
    - name: http
      port: 8080
      targetPort: 80
      protocol: TCP
  selector:
    app: nginx-backend
```

pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: backend-pod
  labels:
    app: nginx-backend
spec:
  containers:
    - name: nginx-container
      image: nginx
      ports:
        - containerPort: 80
```



kubectl create -f clusterservice.yml  
kubectl create -f pod.yml

root@alpine: # curl ingress-nginx

check the endpoints:

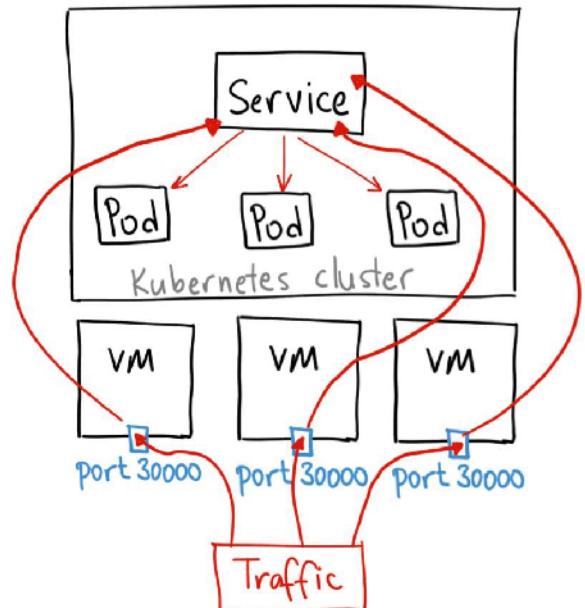
kubectl describe svc/<svc-name>

121

# Services

## NodePort

- NodePort ouvre un port spécifique sur tous les nœuds du cluster et transmet tout le trafic reçu sur ce port aux services internes
- Utile lorsque les pods frontend doivent être exposés en dehors du cluster pour que les utilisateurs puissent y accéder
- NodePort est construit sur le service ClusterIP en exposant le service ClusterIP en dehors du cluster
- NodePort doit être dans la plage de ports 30000-32767
- Si vous ne spécifiez pas ce port, un port aléatoire sera attribué. Il est recommandé de laisser K8s attribuer automatiquement ce port

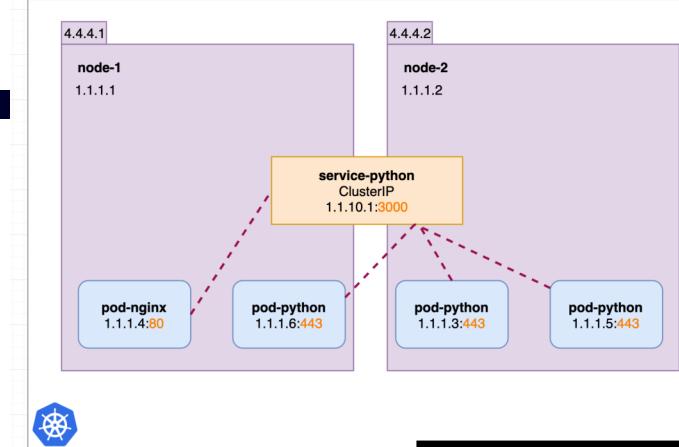


124

# Services

## NodePort

- Supposons maintenant que nous voudrons rendre le service ClusterIP disponible de l'extérieur
- Pour cela nous le convertissons en service NodePort avec seulement deux simples modifications yaml :



```
kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service-python	NodePort	1.1.10.1	<none>	3000:30080/TCP	3m12s

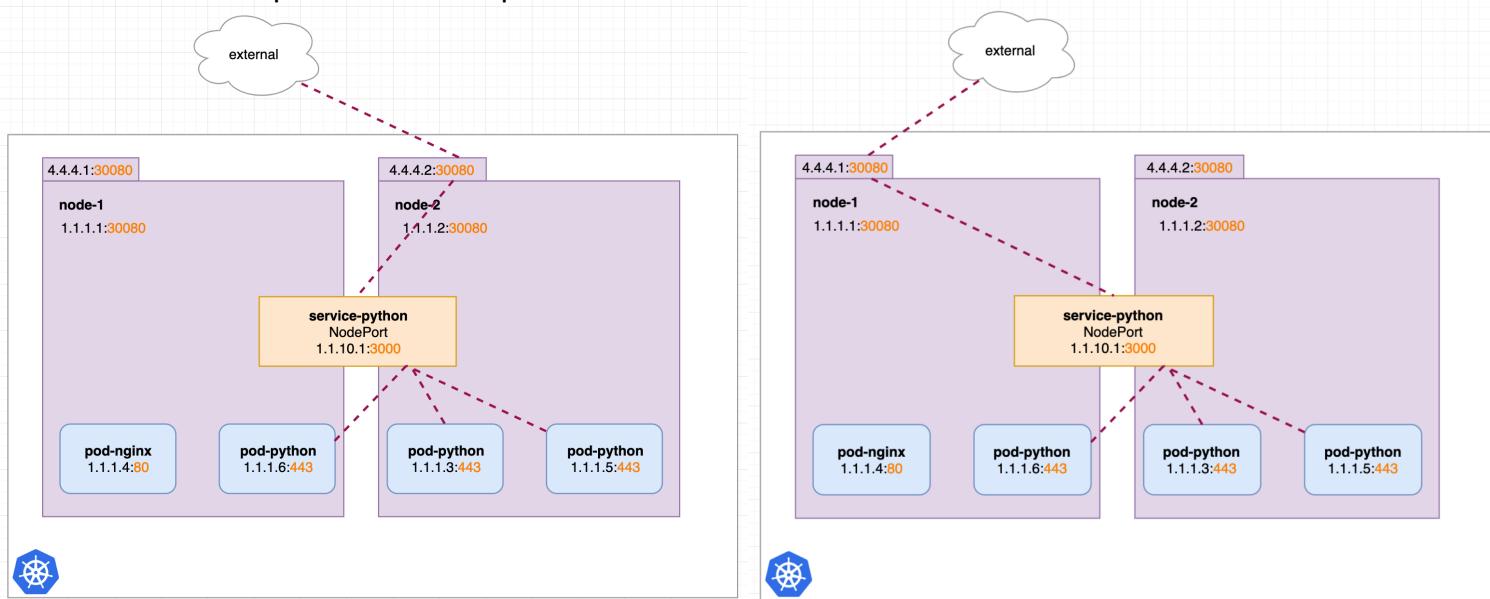
le service NodePort agit toujours comme le service ClusterIP auparavant. Il est utile d'imaginer qu'un service NodePort crée un service ClusterIP, même s'il n'y a plus d'objet ClusterIP supplémentaire.

```
apiVersion: v1
kind: Service
metadata:
  name: service-python
spec:
  selector:
    run: pod-python
  type: NodePort
  ports:
    - port: 3000
      protocol: TCP
      targetPort: 443
      nodePort: 30080
```

# Services

## NodePort

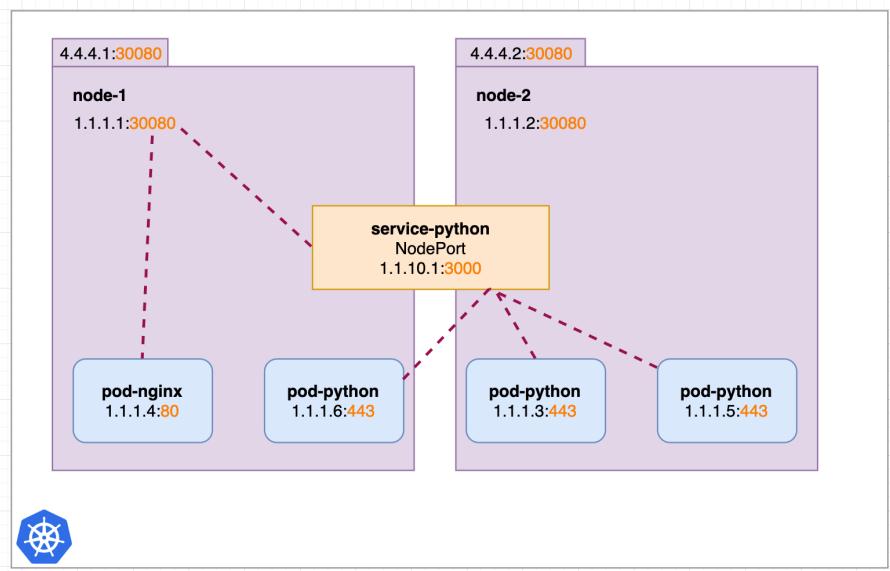
Maintenant notre service-python interne sera désormais accessible à partir de l'adresse IP interne et externe de chaque nœud sur le port 30080.



# Services

## NodePort

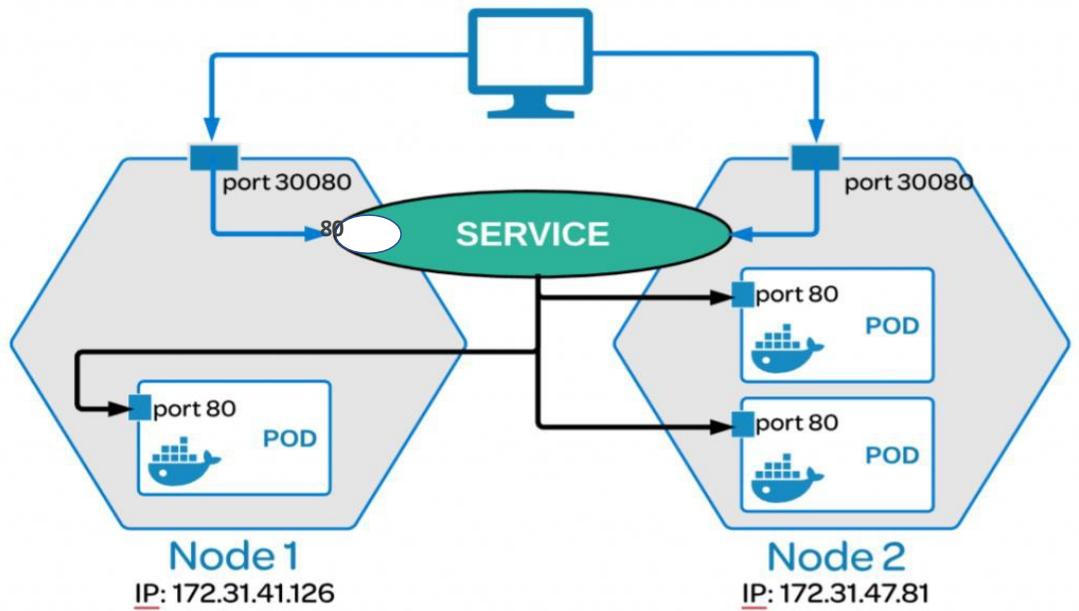
Un pod à l'intérieur du cluster peut également se connecter à une adresse IP de nœud interne sur le port 30080.



127

## NodePort

```
spec:  
  type: NodePort  
  ports:  
    - port: 80  
      targetPort: 80  
      nodePort: 30080
```



128

# Services

## NodePort

nodeport-service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: nodeport-nginx-service
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30001
      protocol: TCP
  selector:
    app: nginx-frontend
```

pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-frontend
  labels:
    app: nginx-frontend
spec:
  containers:
    - name: nginx-container
      image: nginx
      ports:
        - containerPort: 80
```

NodePort

192.168.0.2

port

30001

10.105.32.217

targetPort

80

ingress-nginx  
(NodePort)

Node 03

Nginx



10.244.1.66

Master/Worker

kubectl create -f nodeportservice.yml

kubectl create -f pod.yml

132

# Services

## Demo: NodePort

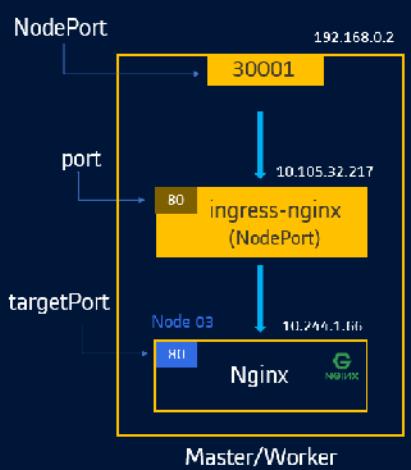
kubectl create -f nodeport-service.yml

kubectl create -f pod.yml

```
root@k-master:/home/osboxes# kubectl create -f nodeport-service.yml
service/nodeport-nginx-service created
pod/nginx-frontend created
root@k-master:/home/osboxes#
```

kubectl get services

```
root@k-master:/home/osboxes# kubectl get services
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes     ClusterIP  10.96.0.1   <none>        443/TCP   13m
nodeport-nginx-service  NodePort   10.105.32.217  <none>        80:30001/TCP 2m3s
root@k-master:/home/osboxes# kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
nginx-frontend 1/1     Running   0          2m10s
root@k-master:/home/osboxes#
```



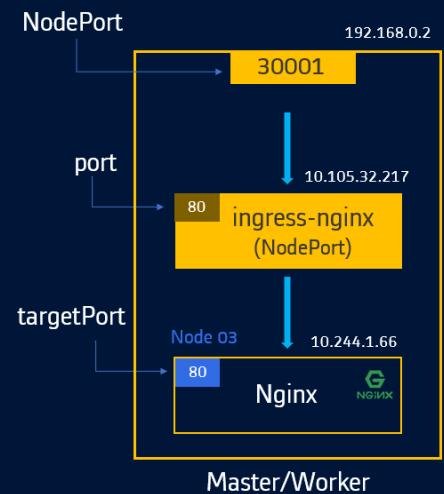
133

# Services

## Demo: NodePort

kubectl describe service <service-name>

```
root@k-master:/home/osboxes# kubectl describe service nodeport-nginx-service
Name:           nodeport-nginx-service
Namespace:      default
Labels:         <none>
Annotations:   <none>
Selector:       app=nginx-frontend
Type:          NodePort
IP:            10.105.32.217
Port:          <unset> 80/TCP
TargetPort:    80/TCP
NodePort:     <unset> 30001/TCP
Endpoints:    10.244.1.66:80
Session Affinity: None
External Traffic Policy: Cluster
Events:        <none>
root@k-master:/home/osboxes#
```



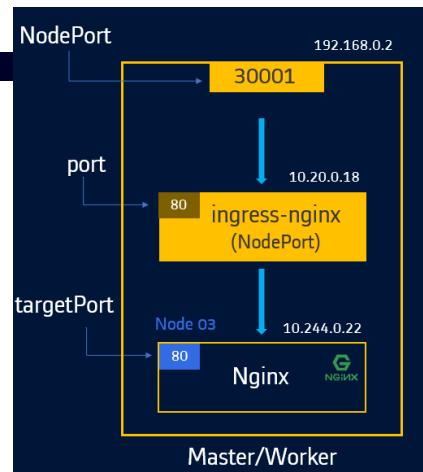
134

# Services

## Demo: NodePort

kubectl get nodes -o wide

```
root@k-master:/home/osboxes# kubectl get nodes -o wide
NAME      STATUS  ROLES   AGE   VERSION  INTERNAL-IP      EXTERNAL-IP
k-master  Ready   master   25h   v1.18.2  192.168.0.107  <none>
k-slave01 Ready   <none>  25h   v1.18.2  192.168.0.104  <none>
k-slave02 NotReady <none>  25h   v1.18.2  192.168.0.108  <none>
root@k-master:/home/osboxes#
```



← → ⌂ ⓘ Not secure | 192.168.0.107:30001

## Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org). Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

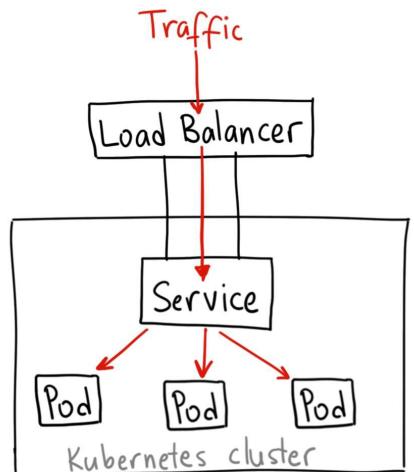
G 192.168.0.107:30001

135

# Services

## Load Balancer

- Un service LoadBalancer est le moyen standard d'exposer un service Kubernetes à Internet
- Sur GKE (Google Kubernetes Engine), cela lancera un équilibrEUR de charge réseau qui vous donnera une adresse IP unique qui transmettra tout le trafic externe à votre service.
- Tout le trafic sur le port que vous spécifiez sera redirigé vers le service
- Il n'y a pas de filtrage, pas de routage, etc. Cela signifie que vous pouvez y envoyer presque n'importe quel type de trafic, comme HTTP, TCP, UDP ou WebSocket.
- Caractéristiques de LoadBalancer :
  - Chaque service exposé aura sa propre adresse IP
  - Il devient très coûteux d'avoir une adresse IP externe pour chacun des services(application)



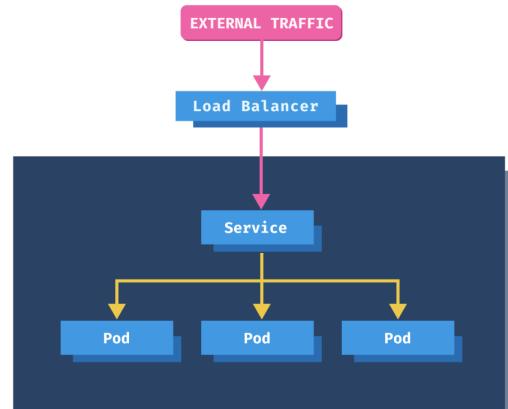
<https://rancher.com/blog/2018/2018-06-08-load-balancing-user-apps-with-rancher/>

137

# Services

## Load Balancer

- Sur Google Cloud, AWS ou Azure, un type de service LoadBalancer dans le fichier manifeste du service exécutera immédiatement un Cloud Load Balancer qui attribue une adresse IP externe (IP publique) à votre application.
- Mais pour les clusters K8 sur site ou baremetal, cette fonctionnalité n'est pas disponible
- L'utilisation du type de service comme LoadBalancer sur baremetal n'attribuera aucune adresse IP externe et la ressource de service restera pour toujours en état d'attente (pending).



```
spec:  
  type: LoadBalancer  
  selector:  
    app: hello  
  ports:  
    - port: 80  
      targetPort: 8080  
      protocol: TCP
```

```
kubectl --kubeconfig=[full path to cluster config file] get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	192.0.2.1	<none>	443/TCP	2h
sample-load-balancer	LoadBalancer	192.0.2.167	<pending>	80:32490/TCP	6s

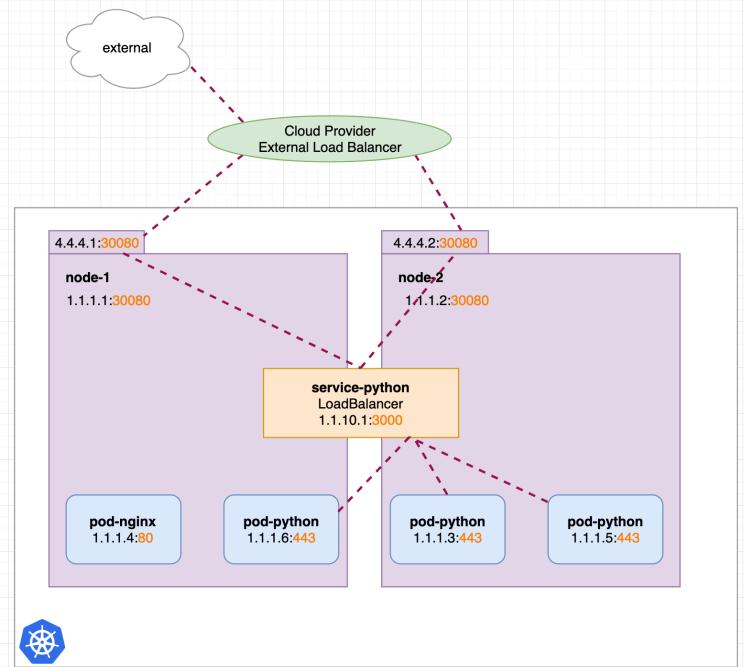
<https://collabnix.com/3-node-kubernetes-cluster-on-bare-metal-system-in-5-minutes/>

138

# Services

## Load Balancer

- Le service LoadBalancer permet d'avoir une seule IP qui distribue les requêtes (en utilisant une méthode comme le round robin) à toutes les IP de nos nœuds. Il est donc construit sur un service NodePort
- Tout ce qu'un service LoadBalancer fait, c'est créer un service NodePort.
- De plus, il envoie un message au fournisseur qui héberge le cluster Kubernetes demandant la configuration d'un équilibreur de charge pointant vers toutes les adresses IP des nœuds externes et un nodePort spécifique.
- Si le fournisseur ne prend pas en charge le message de demande, alors rien ne se passe et LoadBalancer serait égal à un service NodePort.

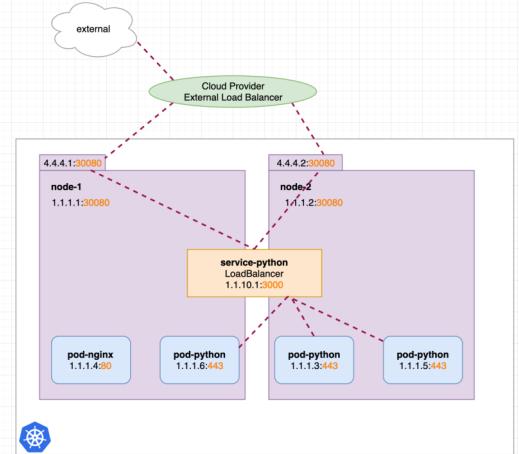


139

# Services

## Load Balancer

- Le service LoadBalancer crée un service NodePort qui crée un service ClusterIP.
- Le yaml modifié pour LoadBalancer par opposition au NodePort avant est simplement :



```
kubectl get svc
```

	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
NodePort	service-python	NodePort	1.1.10.1	<none>	3000:30080/TCP	3m12s
LoadBalancer	service-python	LoadBalancer	1.1.10.1	34.89.144.24	3000:30080/TCP	26m

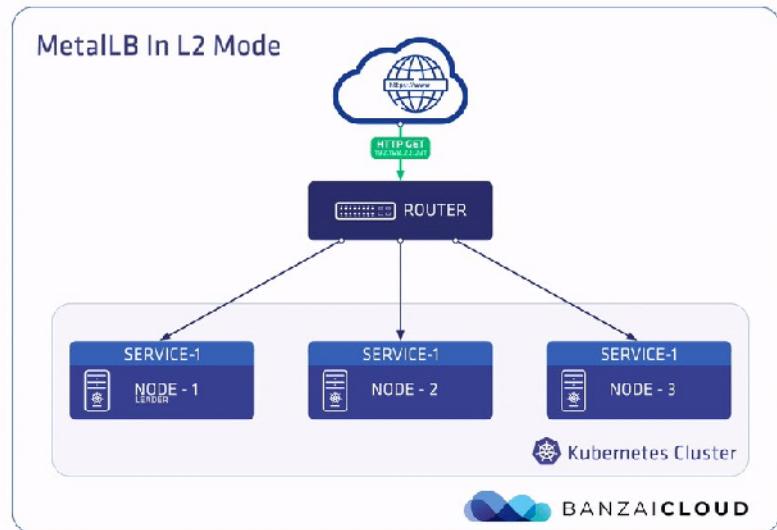
- La commande montre simplement l'ajout de EXTERNAL-IP
- Le service LoadBalancer ouvre toujours le port 30080 sur les IP internes et externes des nœuds comme auparavant. Et il agit toujours comme un service ClusterIP.

```
apiVersion: v1
kind: Service
metadata:
  name: service-python
spec:
  selector:
    run: pod-python
  type: LoadBalancer
  ports:
  - port: 3000
    protocol: TCP
    targetPort: 443
    nodePort: 30080
```

# Services

## MetalLB Load Balancer

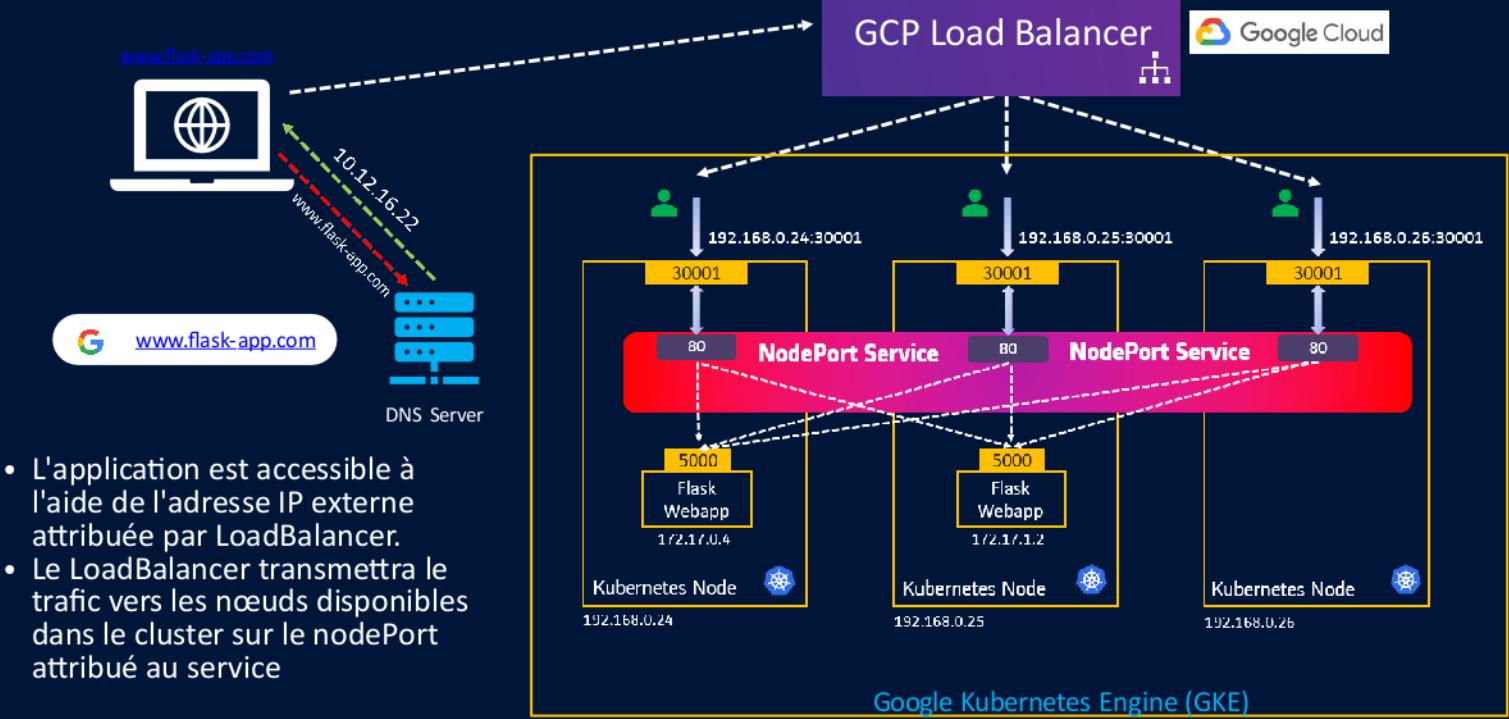
- MetalLB est une implémentation de loadbalancer pour les clusters Kubernetes baremetal.
- Il vous permet de créer des services Kubernetes de type "LoadBalancer" dans des clusters baremetal/on-prem qui ne fonctionnent pas sur des fournisseurs de cloud comme AWS, GCP, Azure et DigitalOcean.



<https://metallb.universe.tf/>

141

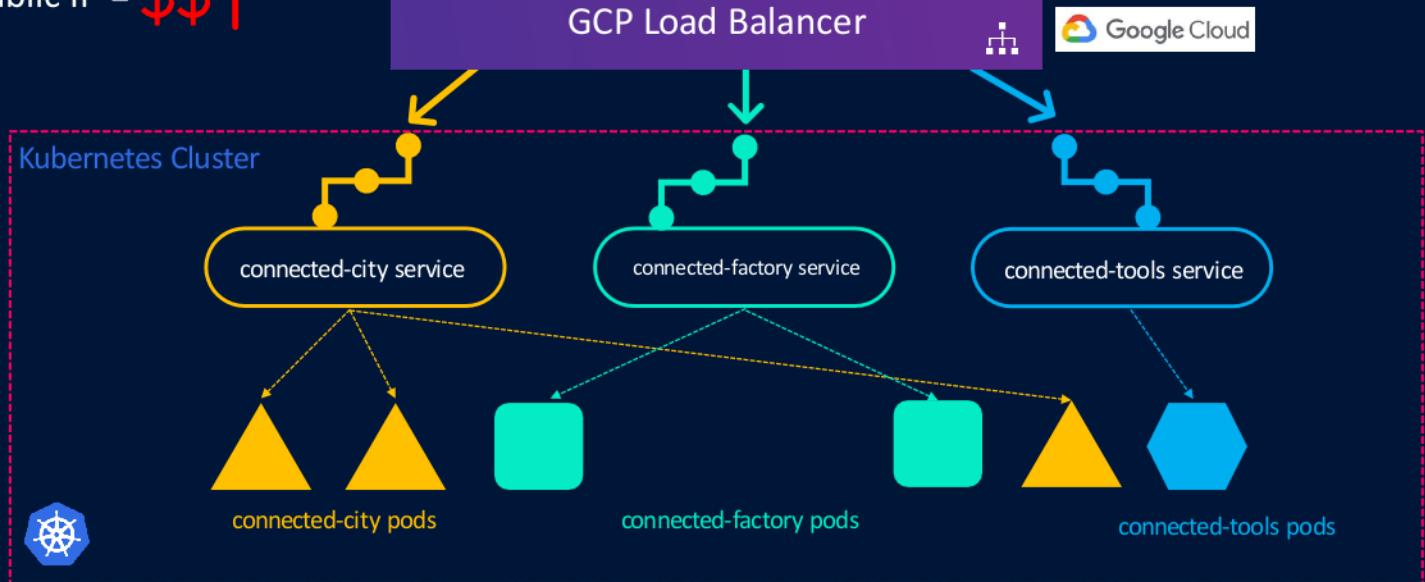
## GCP LoadBalancer



## GCP LoadBalancer

Cons

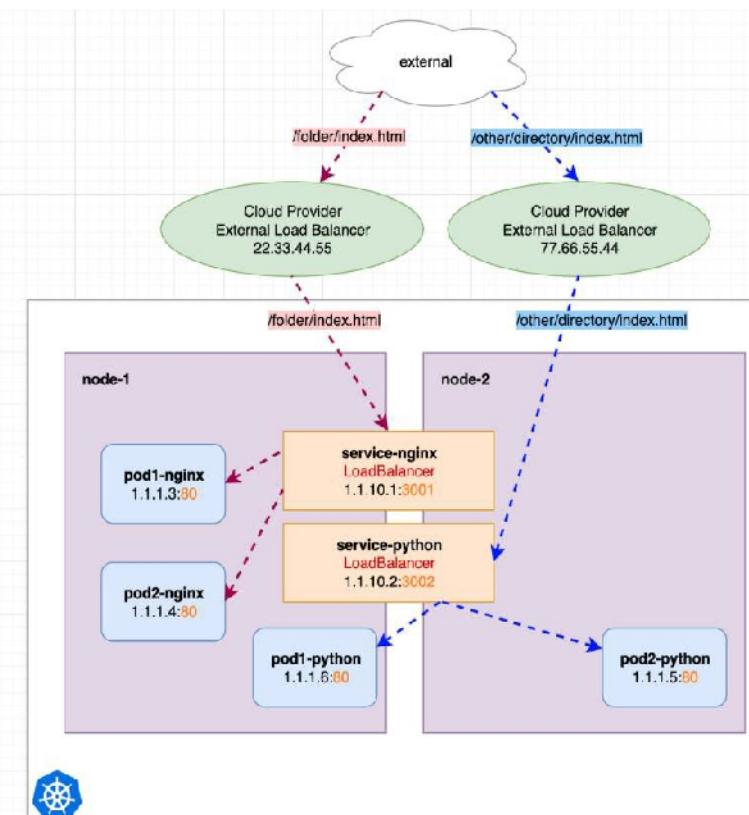
↑ Public IP = \$\$\$↑



## Services

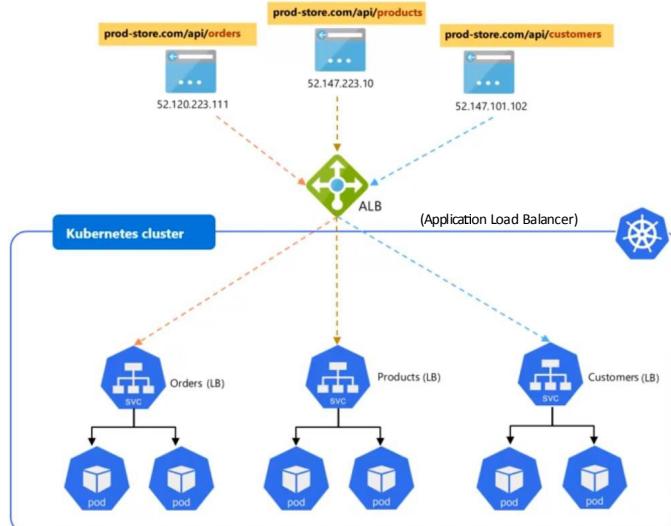
### Cloud LoadBalancer: Inconvénients

- Chaque service exposé aura sa propre adresse IP
- Il devient très coûteux d'avoir une IP externe pour chacun des services (applications)
- Nous avons deux LoadBalancers, chacun ayant sa propre IP. Si nous envoyons une demande à :
  - 22.33.44.55, elle est redirigée vers service-nginx.
  - 77.66.55.44, elle est redirigée vers service-python.
- Cela fonctionne très bien ! Mais les adresses IP sont rares et la tarification de LoadBalancer dépend des fournisseurs de cloud. Imaginez maintenant que nous n'ayons pas seulement deux mais bien plus de services internes pour lesquels nous aimerais créer des LoadBalancers, les coûts augmenteraient.
- Existe-t-il une autre solution qui nous permettrait d'utiliser un seul LoadBalancer (avec une seule IP) tout en atteignant directement nos deux services internes ?

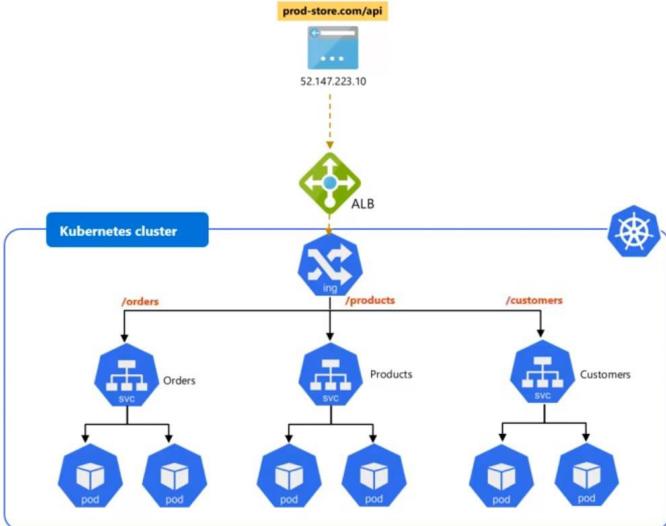


# Services

## LoadBalancer Vs Ingress



- Les IP publiques coûtent cher
- LB ne peut gérer que des IP limitées



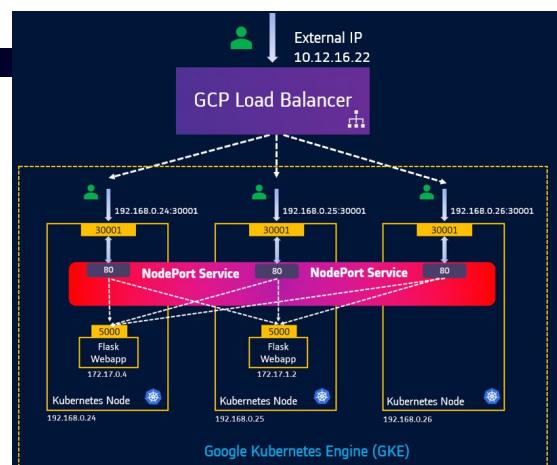
- Ingress agit comme LoadBalancer interne
- Achemine le trafic en fonction du chemin de l'URL
- Toutes les applications n'auront besoin que d'une seule adresse IP publique

146

# Services

## Ingress Resource(rules)

- Avec les LoadBalancers cloud, nous devons payer pour chacun des services exposés en utilisant LoadBalancer comme type de service. À mesure que le nombre de services augmente, la complexité de la gestion des SSL, de la mise à l'échelle, de l'authentification, etc. augmente également
- **Ingress** nous permet de gérer tout ce qui précède au sein du cluster Kubernetes avec un fichier de définition, qui accompagne le reste de vos fichiers de déploiement d'application.
- **Ingress controller** peut effectuer des configurations d'équilibrage de charge, d'authentification, SSL et de routage basées sur URL/chemin
- Ingress aide les utilisateurs à accéder à l'application à l'aide d'un **seule URL accessible de l'extérieur**, que vous pouvez configurer pour acheminer vers différents services au sein de votre cluster en fonction du chemin de l'URL



<https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0>  
<https://cloud.google.com/kubernetes-engine/docs/concepts/ingress>

147

# Services

## Ingress Controller

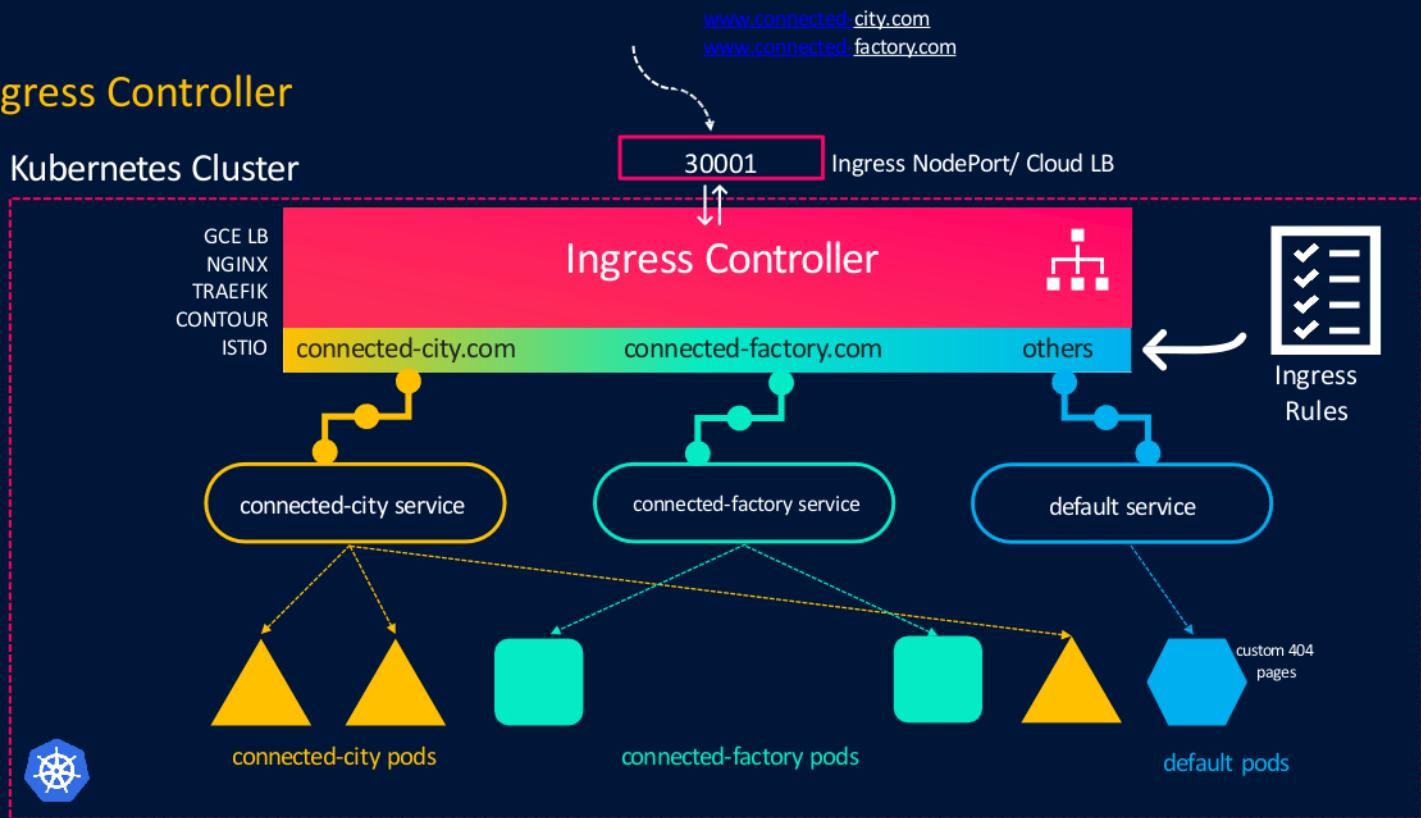
- Ingress resources ne peuvent rien faire par elles-mêmes. Nous avons besoin d'un **Ingress controller** pour que les ressources Ingress fonctionnent
- **Ingress controller implémente les règles définies par ingress resources**
- Ingress controllers ne sont pas fournis avec Kubernetes, ils doivent être déployés séparément
- Kubernetes prend actuellement en charge et maintient les ingress controllers **GCE** et **nginx ingress controllers**
- Ingress controllers populaires incluent Traefik, HAProxy ingress, istio, Ambassador, etc.,
- Ingress controllers doivent être exposés en dehors du cluster à l'aide de NodePort ou d'un Cloud Native LoadBalancer.
- Ingress est le plus utile si vous souhaitez exposer plusieurs services sous la même adresse IP
- Ingress controllers peut effectuer des configurations d'équilibrage de charge, d'authentification, SSL et de routage basées sur URL/chemin

<https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/>

149

## Ingress Controller

### Kubernetes Cluster



# Services

## Nginx Ingress Controller

- Ingress-nginx est un contrôleur Ingress pour Kubernetes utilisant NGINX comme proxy inverse et équilibrer de charge
- Officiellement maintenu par la communauté Kubernetes
- Achemine les requêtes vers les services en fonction de l'hôte ou du chemin de la requête, en centralisant un certain nombre de services dans un seul point d'entrée.  
Ex: www.mysite.com

## Deploy Nginx Ingress Controller

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-0.32.0/deploy/static/provider/baremetal/deploy.yaml
```

<https://github.com/kubernetes/ingress-nginx/blob/master/docs/deploy/index.md#bare-metal>

151

# Dashboard

- Default login access to dashboard is by using token or kubeconfig file. This can be bypassed for internal testing but not recommended in production
- Uses NodePort to expose the dashboard outside the Kubernetes cluster
- Change the service to ClusterIP and use it in conjunction with ingress resources to make it accessible through a DNS name(similar to previous demos)

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
```

Name	Namespace	Labels	Pods	Created	Images
dashboard	default	app: dashboard	1 / 1	3 minutes ago	alpine

<https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/>

<https://devblogs.microsoft.com/premier-developer/bypassing-authentication-for-the-local-kubernetes-cluster-dashboard/>

152

**FIN**