



Automatiser l'Exploitation des Conteneurs avec **Kubernetes**





PLAN

Plan

- **Présentation**
- **Les concepts de base**
- **Un Cluster Kubernetes**
- **L'outil – kubectl**
- **Le POD**
- **Le Deployment**
- **Le Namespace**
- **Le Service**



PRÉSENTATION

Présentation

- Kubernetes / k8s / Kube
- « Homme de barre » / « Pilote » en grec
- Plateforme open source d'orchestration de conteneurs
- Inspirée du système Borg de Google
- v1.0, Juin 2014
- V1.21, Avril 2021

Fonctionnalités

- Gestion d'applications tournant dans des conteneurs
 - Déploiement
 - Montée en charge (Scaling) et haute disponibilité
- Boucle de réconciliation vers l'état souhaité
- Sauvegarde et restauration



Les concepts de base

Cluster

- Ensemble de nodes Linux ou windows (VM/bare metal)
- $(2*n+1)$ nodes **Masters** + m nodes **Workers**

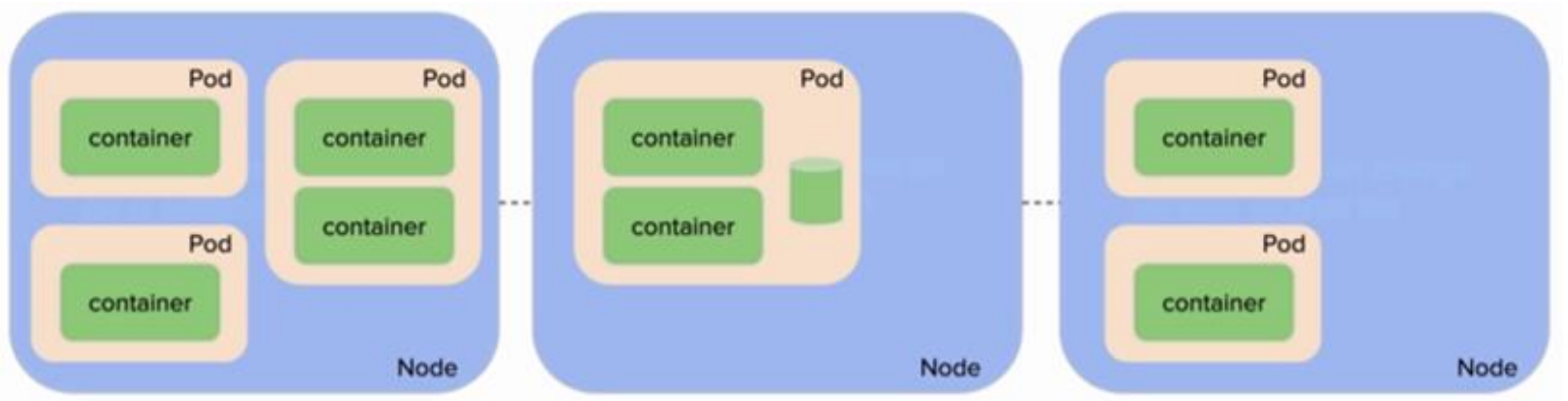
Node **master** en
charge de la gestion
du cluster

Node **worker** en
charge de faire
tourner les
applications

Node **worker** en
charge de faire
tourner les
applications

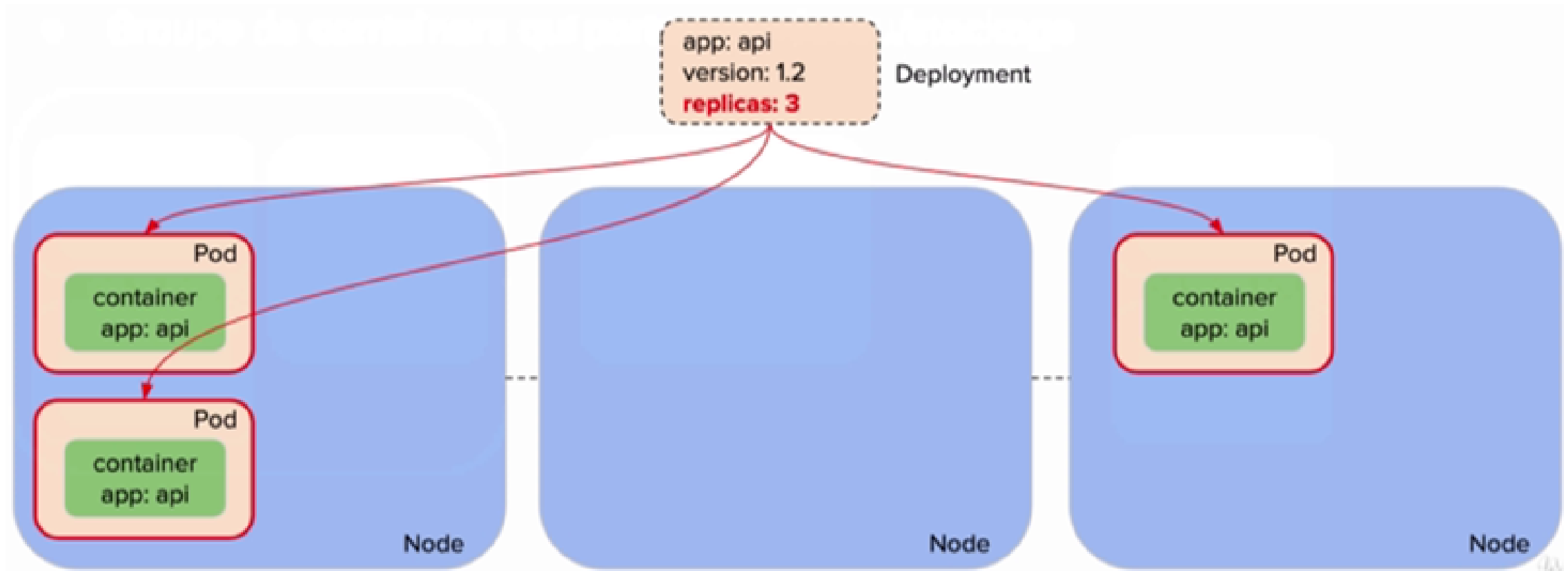
Pods

- Plus petite unité applicative qui tourne sur un cluster Kubernetes
- Groupe de conteneurs qui partagent une stack réseau/stockage (généralement une application/pod)



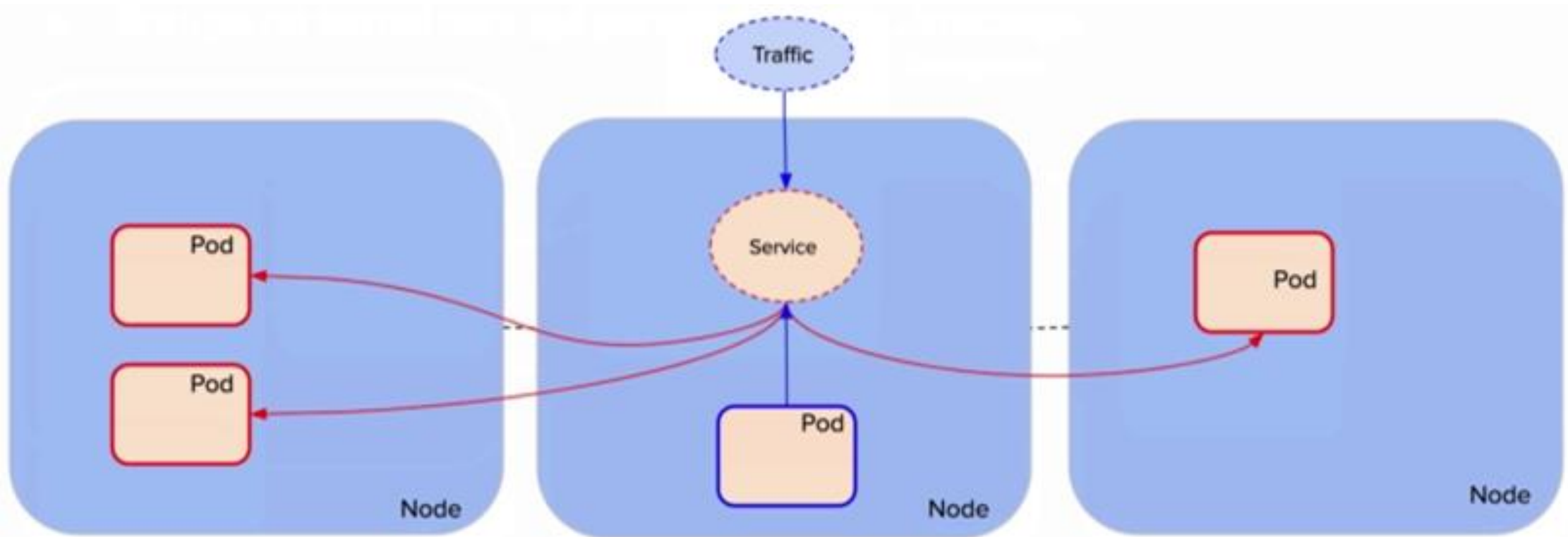
Deployment

- Permet de gérer un ensemble de Pods identique (mise à jour / Rollback)



Service

- Expose les application des Pods à l'intérieur ou à l'extérieur du cluster



Spécification des ressources

- Fichier **yaml**
- Une structure commune
 - **apiVersion** : dépend de la maturité du composant (v1, apps/v1, apps/v1beta1, ...)
 - **kind**: Pod, Service, Deployment, ...)
 - **metadata** : ajout de nom, label, annotations, timestamp, ...
 - **spec** : spécification / description du composant

```
apiVersion: v1
kind: Pod
metadata:
  name: www
  labels:
    app: w3
spec:
  containers:
    - name: www
      image: nginx:1.12.2
```

Exemple de spécification d'un Pod

```
apiVersion: v1
kind: Service
metadata:
  name: www
spec:
  selector:
    app: w3
  ports:
    - port: 80
      targetPort: 80
```

Exemple de spécification d'un Service

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: www
spec:
  replicas: 3
  selector:
    matchLabels:
      app: w3
  template:
    ...
```

Exemple de spécification d'un Deployment

Labels et annotation (Selectors)

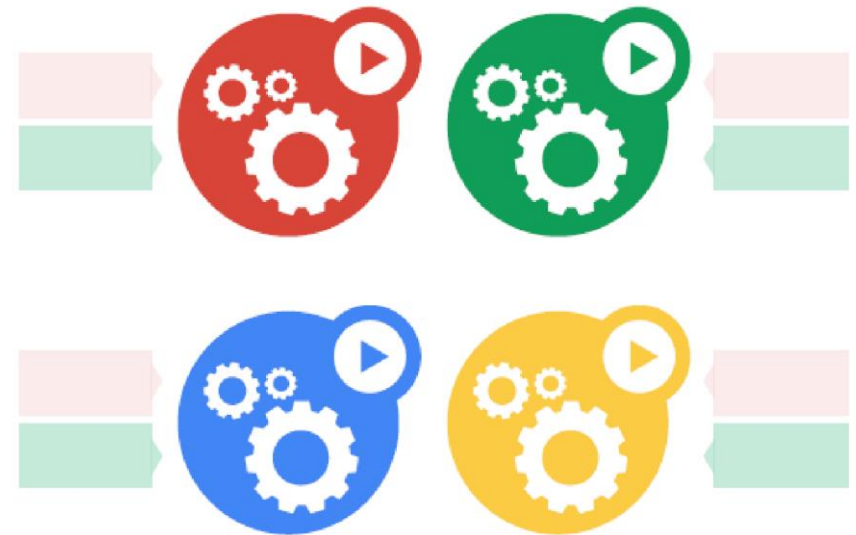
- Informations attachées aux ressources
- Format : key / value
- **Labels**
 - Utilisés pour la sélection de objets
 - Récupération de collections
- **Annotations**
 - Non interne à kubernetes
 - Utilisées par des outils et librairies clientes

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: www
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  labels:
    www
...

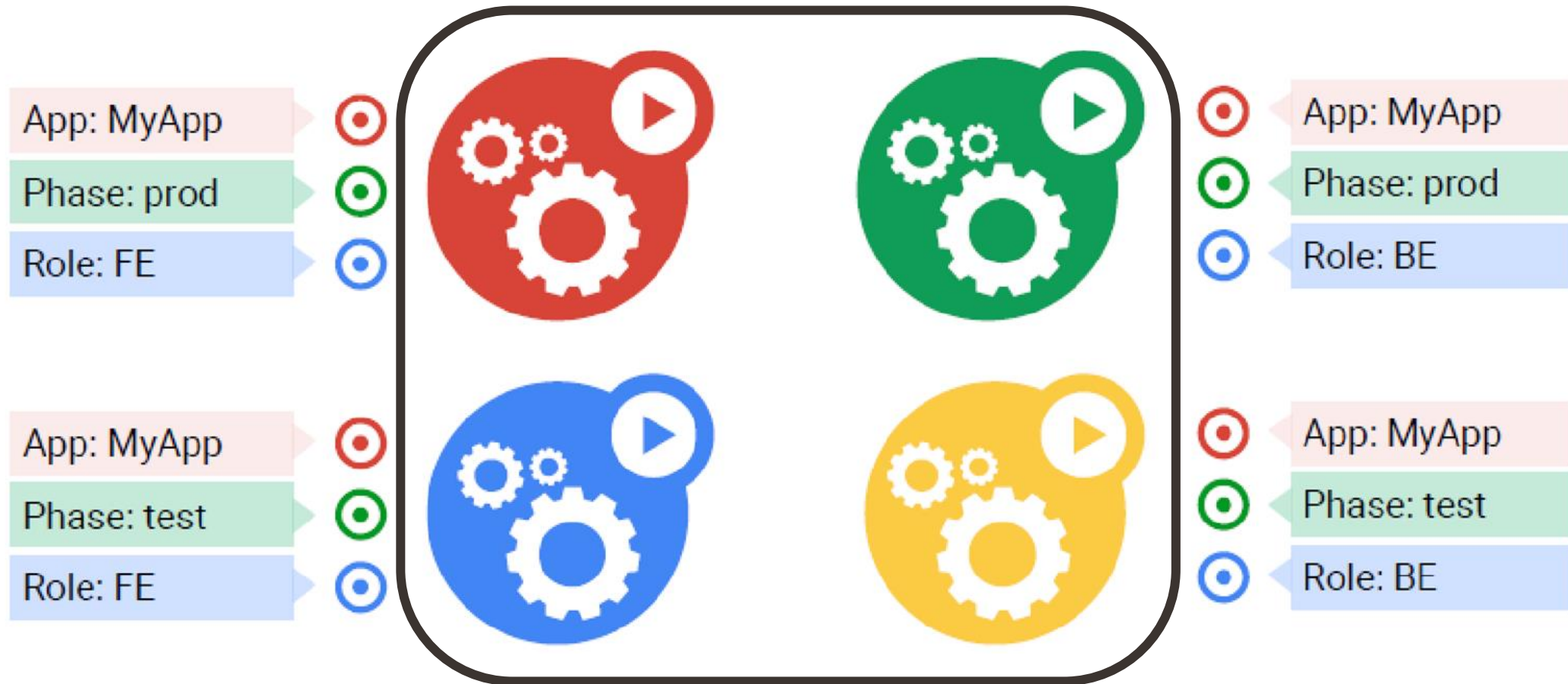
```

Labels et annotation (Selectors)

- Informations attachées aux ressources
- Format : key / value
- **Labels**
 - Utilisés pour la sélection de objets
 - Récupération de collections
- **Annotations**
 - Non interne à kubernetes
 - Utilisées par des outils et librairies clientes

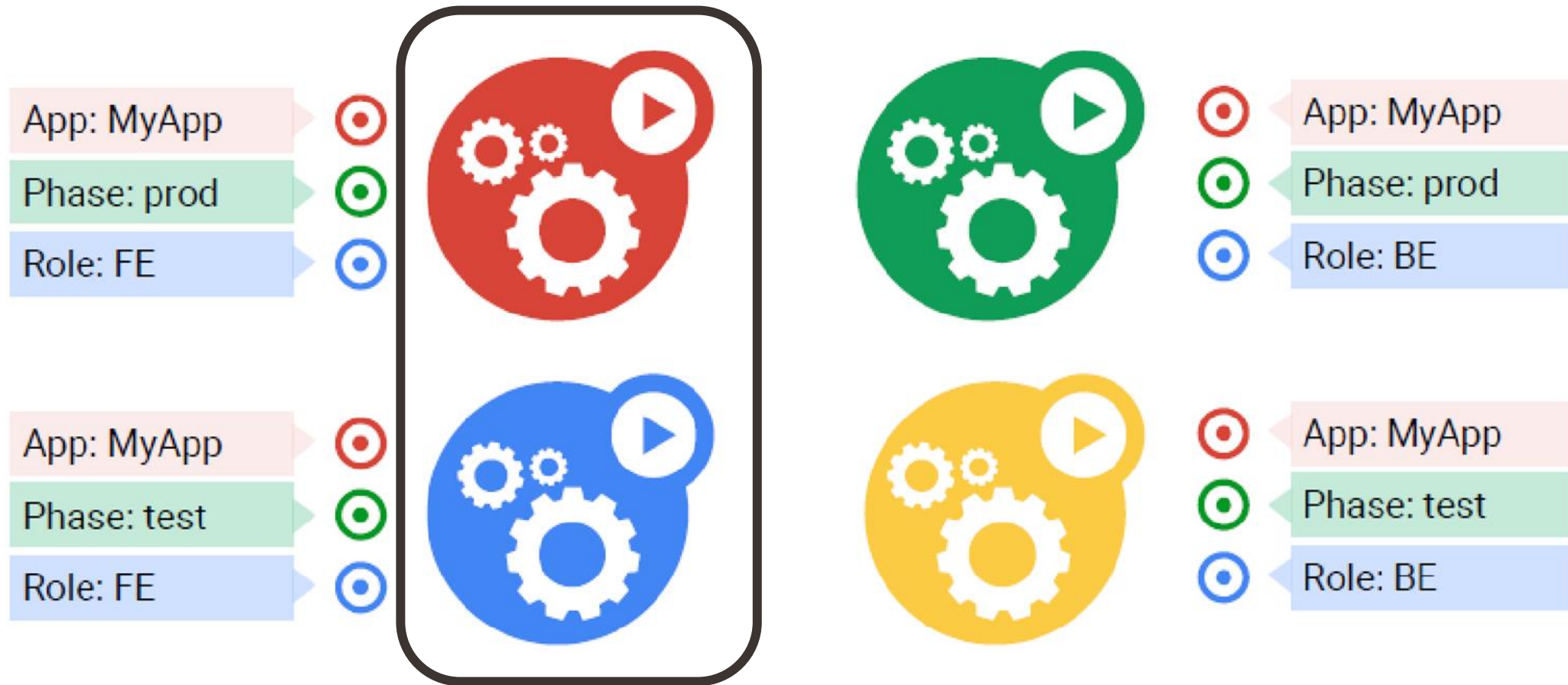


Labels et annotation (Selectors)



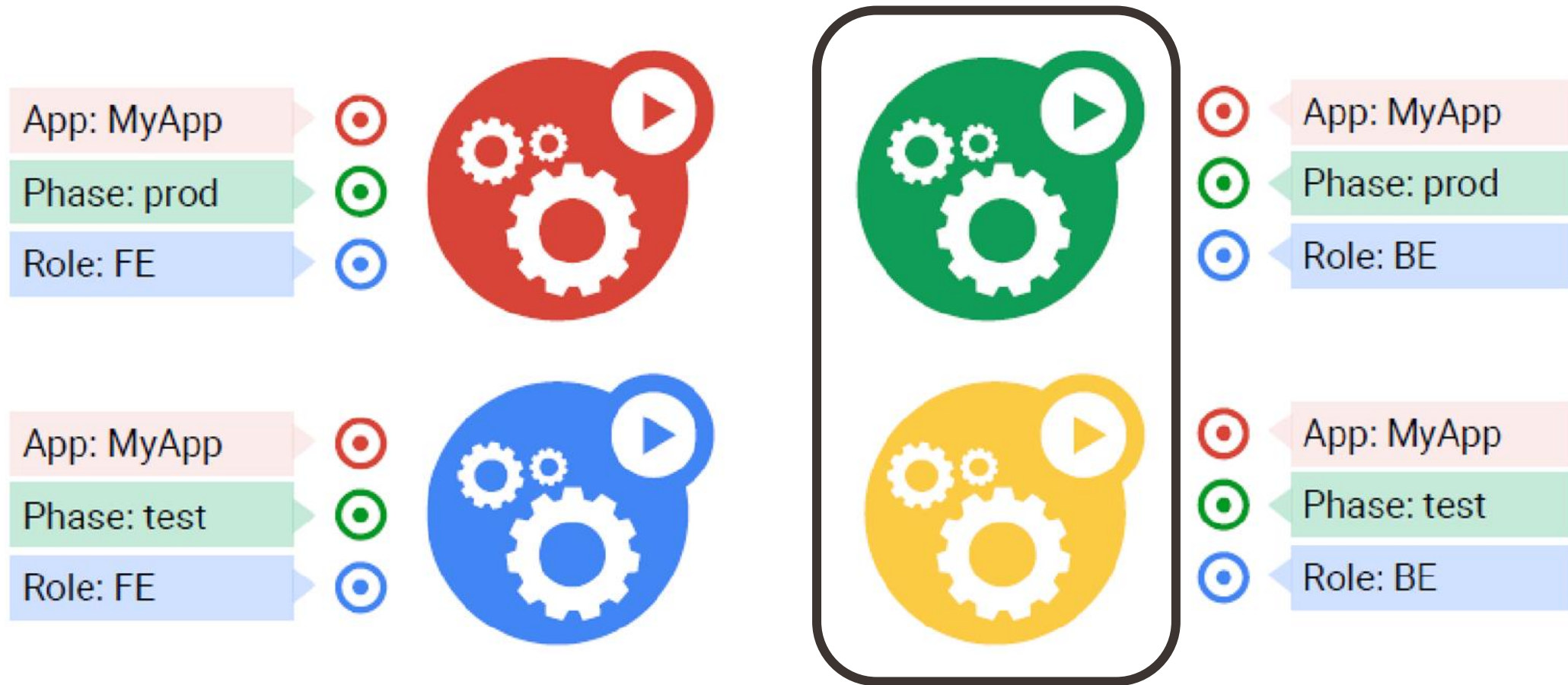
App = MyApp

Labels et annotation (Selectors)



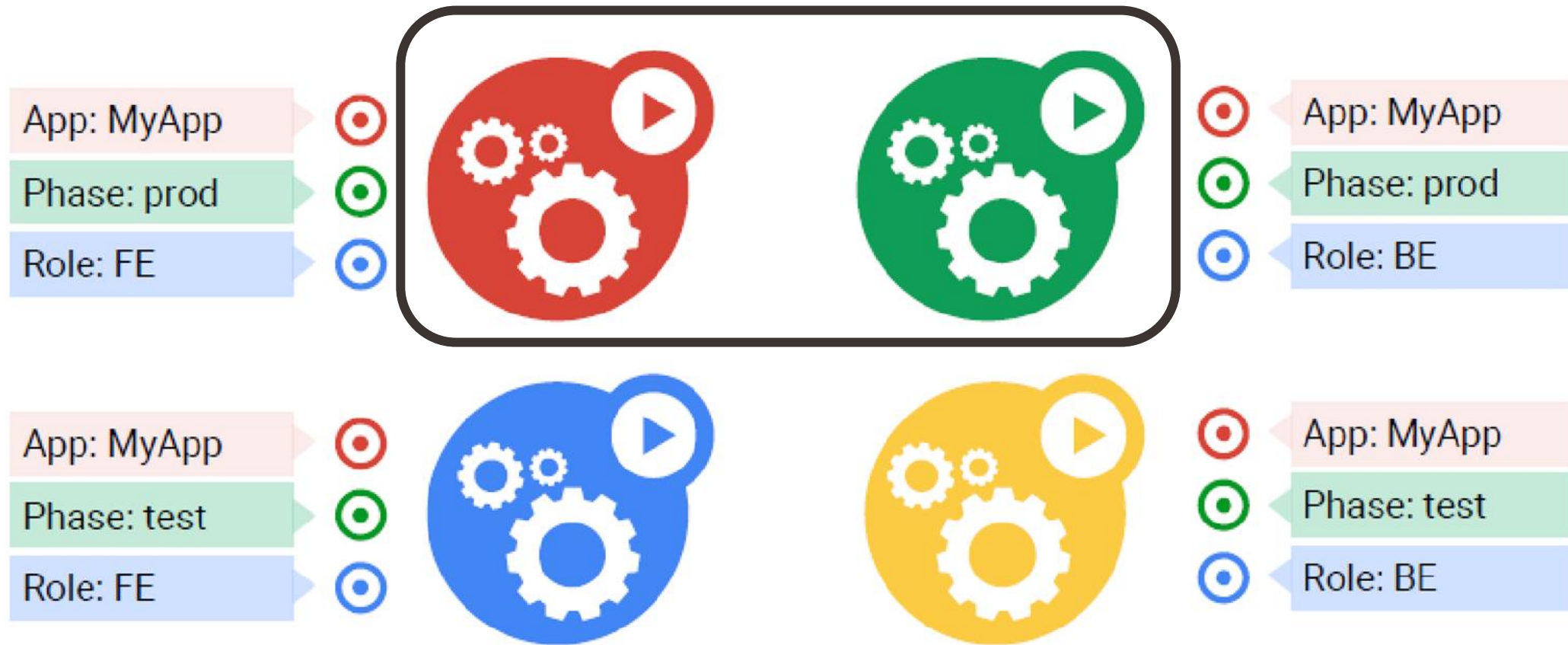
App = MyApp, Role = FE

Labels et annotation (Selectors)



App = MyApp, Role = BE

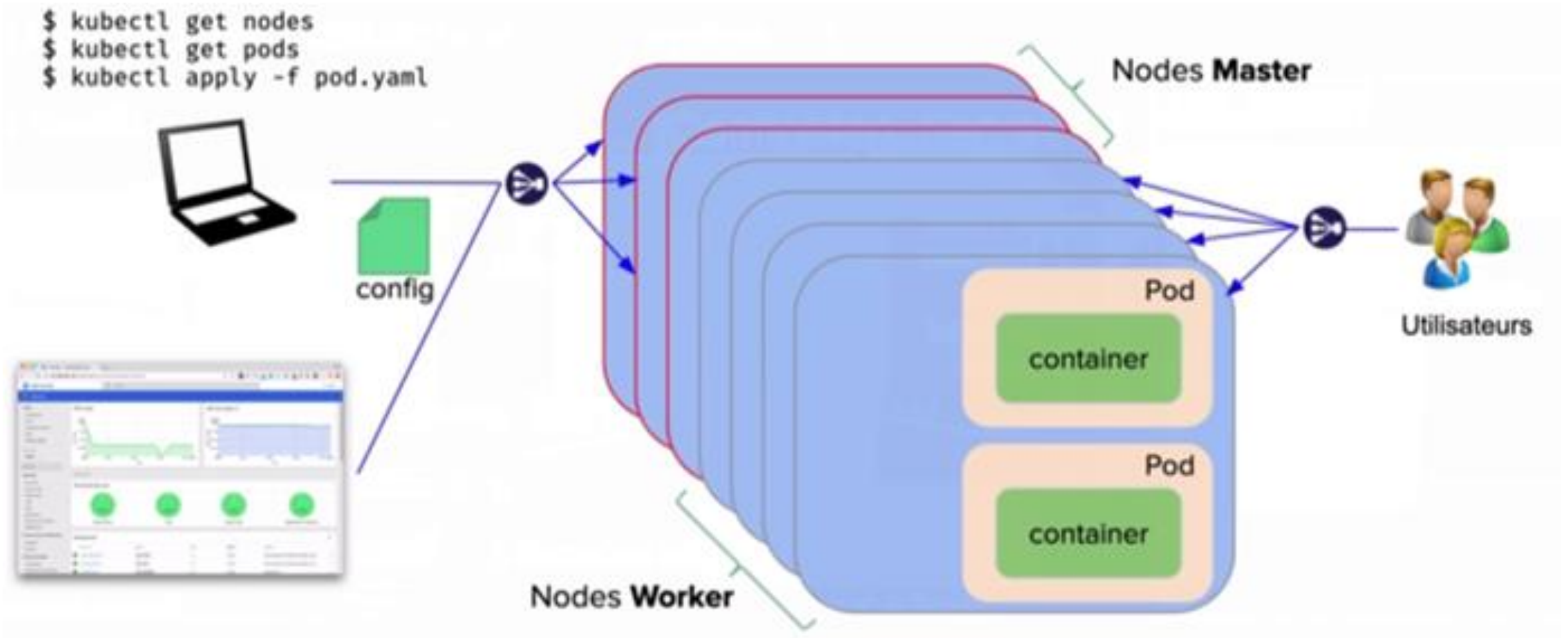
Labels et annotation (Selectors)



App = MyApp, Phase = prod

Communication avec le cluster

- En utilisant le binaire kubectl ou l'interface web



Les différents types de nodes

```
[root@master-node ~]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master-node	Ready	control-plane,master	30h	v1.20.1
worker-node1	Ready	<none>	29h	v1.20.1
worker-node2	Ready	<none>	29h	v1.20.1

```
[root@master-node ~]#
```

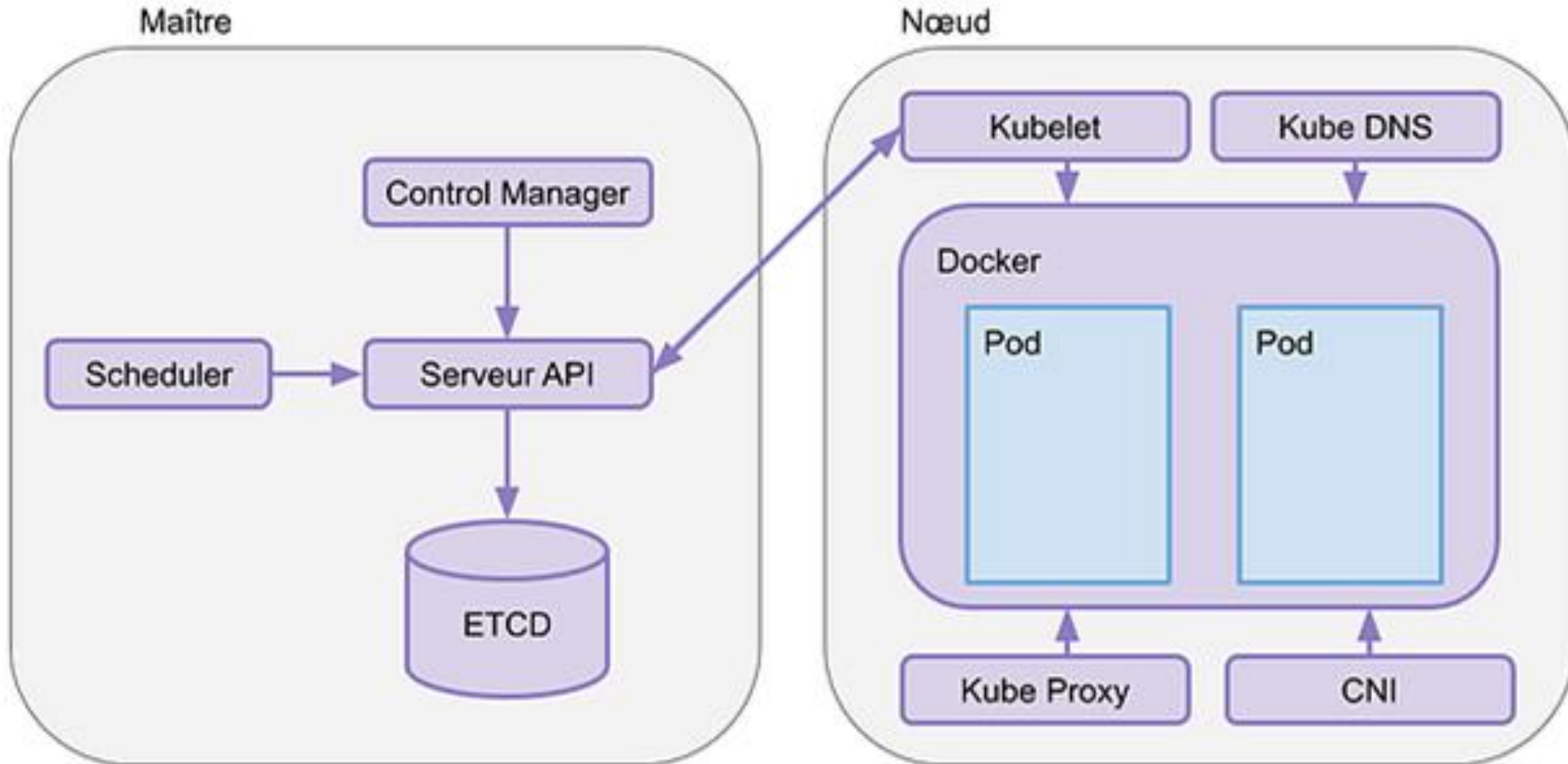
Master

- Responsable de la gestion du cluster ("Control plane")
- Expose l'API Server
- Schedule les Pods sur les nodes cluster

Worker / Nodes

- Node sur lequel sont lancés les Pods applicatifs
- Communique avec le Master
- Fournit les ressources aux Pods

Architecture Kubernetes



Architecture des nœuds

```
[root@master-node ~]# kubectl get pod -n kube-system
```

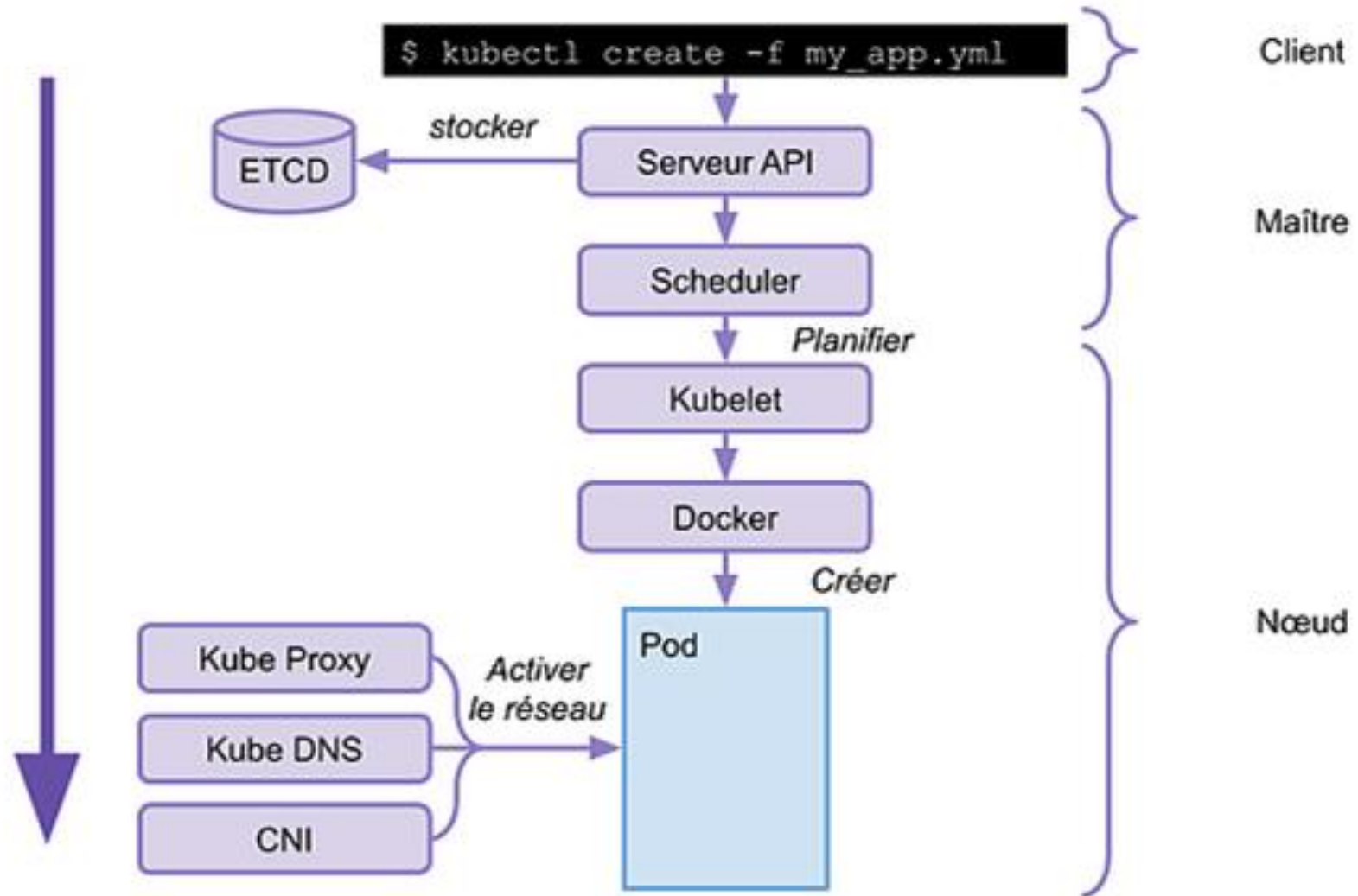
NAME	READY	STATUS	RESTARTS	AGE
coredns-74ff55c5b-8qx94	1/1	Running	0	42m
coredns-74ff55c5b-wkpg8	1/1	Running	0	42m
etcd-master-node	1/1	Running	0	42m
kube-apiserver-master-node	1/1	Running	0	42m
kube-controller-manager-master-node	1/1	Running	0	42m
kube-flannel-ds-j4fw6	1/1	Running	0	23s
kube-flannel-ds-k2kkk	1/1	Running	0	23s
kube-flannel-ds-nnt4x	1/1	Running	0	23s
kube-proxy-mlkvm	1/1	Running	0	42m
kube-proxy-xjjxt	1/1	Running	0	42m
kube-proxy-zwnht	1/1	Running	0	41m
kube-scheduler-master-node	1/1	Running	0	42m

Processus tournant sur les masters

Processus tournant sur chacun des nodes (masters ou workers)

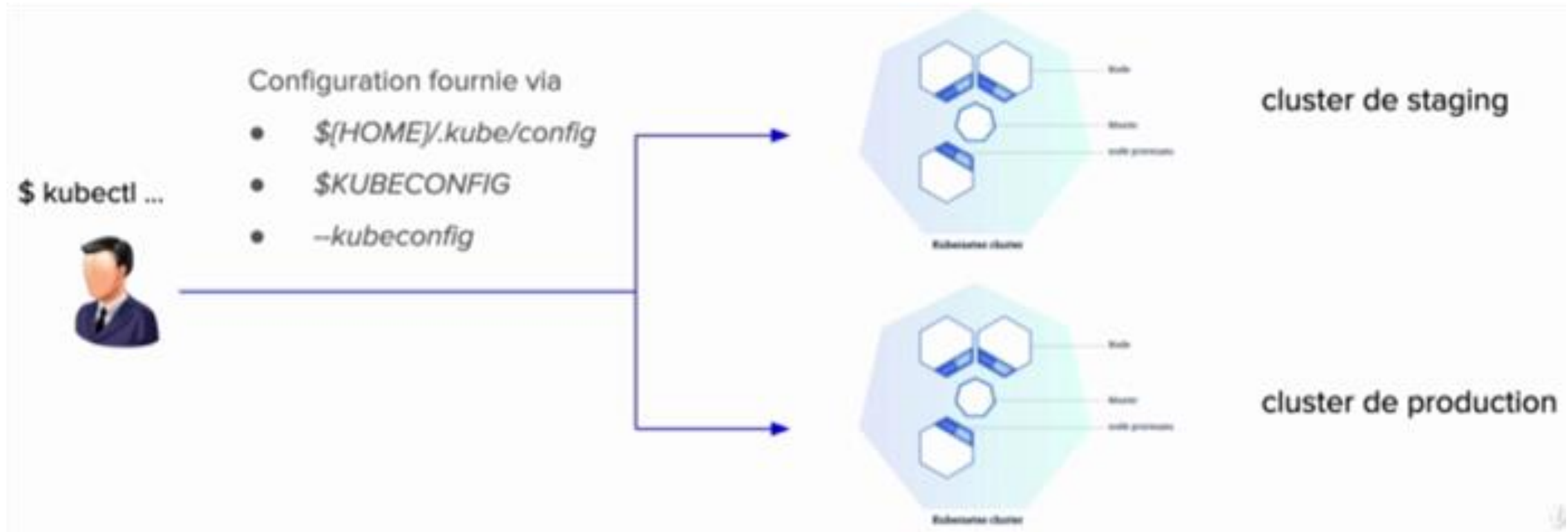
Processus répliqués sur le cluster (exp : Serveur DNS add-on)

Cycle de vie de déploiement d'une application



Contexte d'utilisation

- Définie le cluster auquel s'adresse et avec quel utilisateur





UN CLUSTER KUBERNETES

Minkube

- Tous les composants de Kubernetes dans une **seule VM locale**
- S'intègre avec différents hyperviseurs
 - Hyperkit (MacOS)
 - Hyper-V (Windows)
 - KVM (Linux)
 - VirtualBox
 - VMWare
- Nécessite le binaire minikube



Kind (Kubernetes in Docker)

- Les nodes tournent dans des conteneurs Docker
- Cluster HA via un fichier de configuration
- Il faut installer Docker et télécharger le binaire Kind

```
$ kind create cluster
Creating cluster "k8s" ...
  ✓ Ensuring node image (kindest/node:v1.16.3)

  ✓ Preparing nodes 📦
  ✓ Writing configuration 📄
  ✓ Starting control-plane 🚦
  ✓ Installing CNI 🌐
  ✓ Installing StorageClass 🗄️
Set kubectl context to "kind-k8s"
You can now use your cluster with:
kubectl cluster-info --context kind-k8s
Have a nice day! 🌞
```



MicroK8s

- Un seul paquet de k8s pour 42 versions de Linux.
- Conçu pour les développeurs et idéal pour les périphériques, l'IoT et les appliances

The logo for MicroK8s, featuring the text "MicroK8s" in white on an orange rectangular background.

MicroK8s

```
$ snap install microk8s --classic
```

```
$ microk8s.kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
microk8s	Ready	<none>	41s	v1.17.0

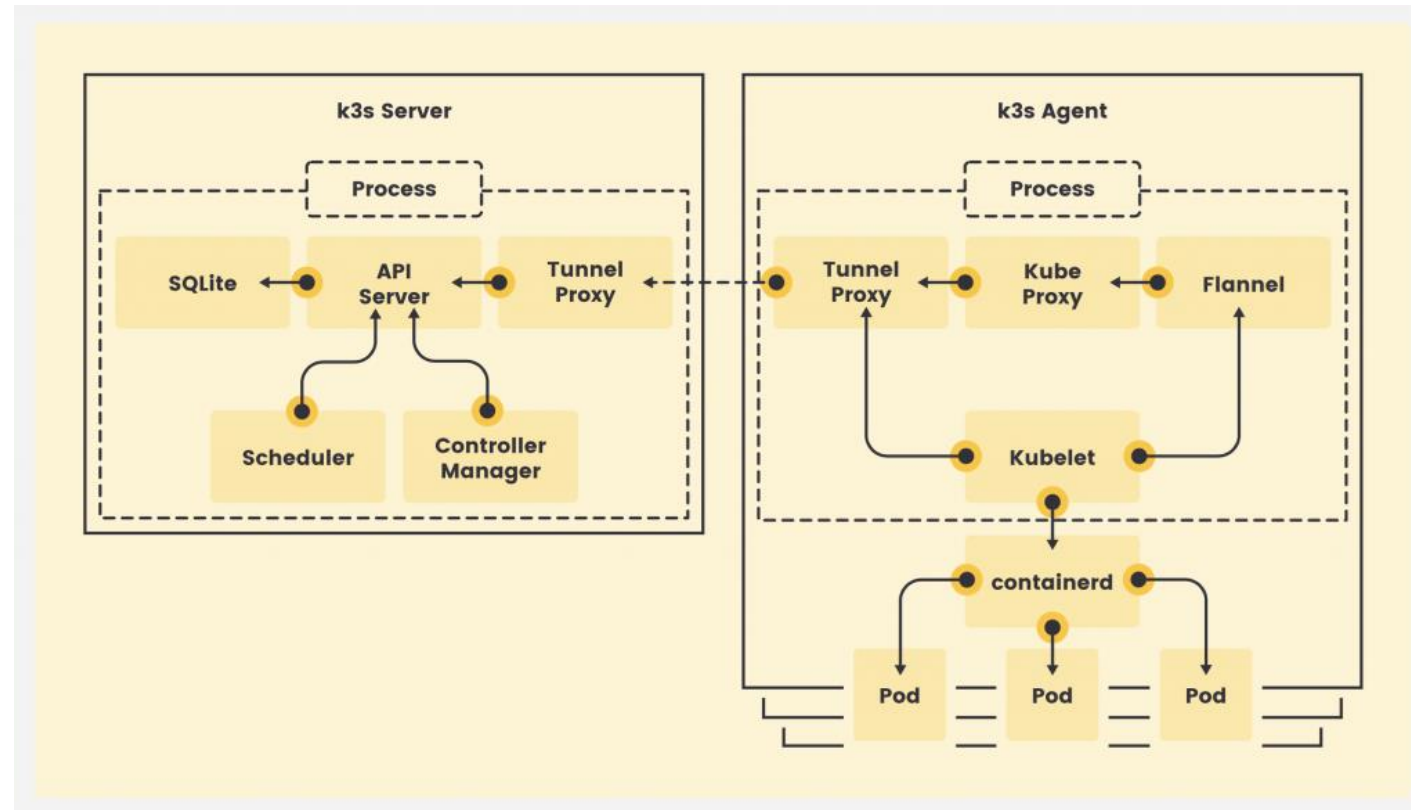


K3S

- Kubernetes léger : la distribution Kubernetes certifiée conçu pour l'IoT

```
$ curl -sfL https://get.k3s.io | sh -  
  
$ kubectl get nodes  
NAME    STATUS    ROLES    AGE    VERSION  
k3s     Ready     master   2m40s   v1.16.3-k3s.2
```

K3s: Lightweight Kubernetes



Cluster Kubernetes managé

- GKE – Google Kubernetes Engine
- AKS – Azure Container service
- EKS – Amazon Elastic Container Service
- DigitalOcean

Cluster Kubernetes managé : GKE

- Création depuis l'interface <https://console.cloud.google.com> ou bien le binaire gcloud

Google Cloud Platform | lj-k8s-training

Créer un cluster Kubernetes

Modèles de cluster
Sélectionnez un modèle avec des paramètres préconfigurés ou personnalisez un modèle selon vos besoins

- ☐ Cloner un cluster existant
Sélectionnez l'un de vos clusters existants pour remplir les champs
- ☒ **Cluster standard**
Intégration continue, serveurs Web, backends. Le meilleur choix pour bénéficier d'une personnalisation plus complète ou si vous ne savez pas quoi choisir.
- ☐ Votre premier cluster
Tester Kubernetes Engine, déployer votre première application. Un choix abordable pour commencer.
- ☐ Applications nécessitant une utilisation intensive des processeurs
L'exploration du Web ou toute autre tâche qui nécessite davantage de processeurs.
- ☐ Applications qui utilisent beaucoup de mémoire
Bases de données, analyses, outils comme Hadoop, Spark, ETL ou toute autre tâche nécessitant davantage de mémoire.

Nom
mykube

Type d'emplacement
☒ Zonale
☐ Régional

Zone
europe-west3-a

Version maître
1.11.8-gke.6 (par défaut)

Pools de nœuds
Les pools de nœuds sont des groupes d'instances distincts qui exécutent Kubernetes dans un cluster. Vous pouvez ajouter des pools de nœuds dans différentes zones pour définir une disponibilité plus élevée, ou ajouter des pools de nœuds de différents types de machines. Pour ajouter un pool de nœuds, cliquez sur "Modifier". En savoir plus

default-pool

Nombre de nœuds
3

Type de machine
Cliquez sur "Personnaliser" pour sélectionner des cœurs, la mémoire et des GPU
2 vCPU 7,5 Go de mémoire Personnaliser

Mettez à jour votre compte pour créer des instances possédant jusqu'à 96 cœurs

Créer Réinitialiser Requête REST ou ligne de commande équivalente

```
# gcloud init
```

```
# gcloud container clusters create kube-demo  
--cluster-cersion=latest --num-nodes 3
```

<https://cloud.google.com/sdk/install>

Cluster Kubernetes managé: AKS

- Création depuis l'interface web <https://portal.azure.com> ou le binaire az

Accueil > Nouveau > Créer un cluster Kubernetes

Créer un cluster Kubernetes

De base Mise à l'échelle Authentification Mise en réseau Supervision Étiquettes Vérifier + créer

Azure Kubernetes Service (AKS) gère votre environnement Kubernetes hébergé pour accélérer et faciliter le déploiement et la gestion des applications en conteneur, sans avoir à maîtriser l'orchestration de conteneurs. Il permet également d'éviter les opérations et la maintenance au quotidien grâce au provisionnement, à la mise à niveau et à la mise à l'échelle des ressources à la demande, sans mettre vos applications hors connexion. [En savoir plus sur Azure Kubernetes Service](#)

DÉTAILS DU PROJET

Sélectionnez un abonnement pour gérer les coûts et les ressources déployées. Utilisez les groupes de ressources comme des dossiers pour organiser et gérer toutes vos ressources.

* Abonnement ⓘ Paiement à l'utilisation

* Groupe de ressources ⓘ Sélectionner l'élément existant...
[Créer nouveau](#)

DÉTAILS DU CLUSTER

* Nom du cluster Kubernetes ⓘ myKube ✓

* Région ⓘ (Europe) Europe Ouest

* Version de Kubernetes ⓘ 1.12.7 (par défaut)

* Préfixe du nom DNS ⓘ myKube-dns ✓

POOL DE NOEUDS PRINCIPAL

Nombre et taille des nœuds dans le pool de nœuds principal de votre cluster. Pour les charges de travail de production, au moins 3 nœuds sont recommandés pour assurer la résilience. Pour les charges de travail de test ou de développement, un seul nœud est nécessaire. Vous ne pouvez pas changer la taille de nœud après la création du cluster, mais vous pouvez changer le nombre de nœuds dans votre cluster après sa création. Si vous voulez ajouter des pools de nœuds, vous devez activer la fonctionnalité « X » sous l'onglet « Mise à l'échelle » qui vous permet d'ajouter plusieurs pools de nœuds après la création du cluster. [En savoir plus sur les pools de nœuds dans Azure Kubernetes Service](#)

* Taille de nœud ⓘ Standard DS2 v2

[Vérifier + créer](#) [Précédent](#) [Suivant : Mise à l'échelle >](#)

Création d'un groupe

```
# az group create --name kubegroup --location westeurope
```

Création du cluster

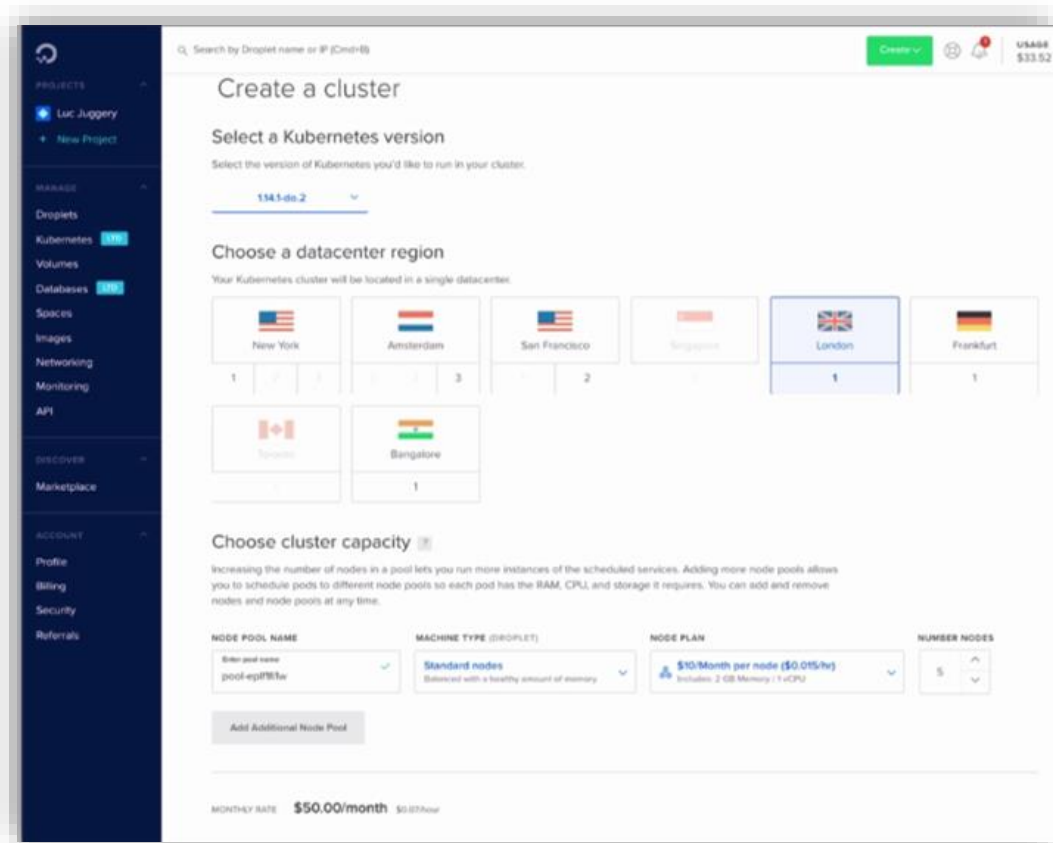
```
# az aks create --name kubedemo --resource-group kubegroup --node-count 3 --generate-ssh-keys
```

Mise à jour de la configuration de kubectl

```
# az aks get-credentials --resource-group kubegroup --name kubedemo
```


Cluster Kubernetes managé: DigitalOcean

- Création depuis l'interface web <https://digitalocean.com> ou le binaire doctl



```
# doctl k8s cluster create mykube \  
--region lon1 \  
--version 1.14.1-do.2 \  
--node-pool="name=worker;size=s-2vcpu-4gb;count=5" \  
--update-kubeconfig=true
```

Cluster Kubernetes local

- Kubeadm : <https://kubernetes.io/docs/reference/setup-tools/kubeadm/>
- Kops : <https://github.com/kubernetes/kops>
- Kubespray : <https://kubespray.io/>
- Rancher : <https://rancher.com/>
- Pharos : <https://k8spharos.dev/>
- Docker EE (deploy Swarm et Kubernetes)

LAB 1 : Installer Un Cluster Kubernetes





L'outil - kubectl

Utilisation

- Binaire utilisé pour communiquer avec l'API Server, permet de contrôler les clusters Kubernetes
- Envoi des requêtes HTTP
- Fonctionnalités du kubectl :
 - Gestion du cluster
 - Gestion des ressources
 - Troubleshooting
- **<https://kubernetes.io/docs/tasks/tools/install-kubectl/>**

Gestion de ressources : Impérative vs Déclarative

Impérative

- Gestion des ressources en ligne de commande
 - Approche simple, pour aller vite
 - Pas de suivi de changements dans un VCS
 - Nombreuses commandes
- ☐ Pour le développement

```
# kubectl run --generator=run-pod/v1 --  
image=nginx:1.16 www
```

```
# kubectl expose www --port=80
```

Déclarative

- Fichier de configuration pour chaque ressource
 - Nécessite une connaissance des ressources
 - Suivi des changements dans VCS
 - Commandes apply
 - Analyse automatique des différences
- ☐ Pour la production

```
# kubectl create -f wordpress-pod.yaml
```

Les ressources disponibles

kubectl api-resources

```
[root@master-node pods]# kubectl api-resources
NAME                SHORTNAMES  APIVERSION  NAMESPACE  KIND
bindings            v1          v1          true        Binding
componentstatuses   cs          v1          false       ComponentStatus
configmaps          cm          v1          true        ConfigMap
endpoints           ep          v1          true        Endpoints
events              ev          v1          true        Event
limitranges         limits      v1          true        LimitRange
namespaces          ns          v1          false       Namespace
nodes               no          v1          false       Node
persistentvolumeclaims pvc         v1          true        PersistentVolumeClaim
persistentvolumes   pv          v1          false       PersistentVolume
pods                po          v1          true        Pod
podtemplates        v1          v1          true        PodTemplate
replicationcontrollers rc          v1          true        ReplicationController
resourcequotas      quota       v1          true        ResourceQuota
secrets             v1          v1          true        Secret
serviceaccounts     sa          v1          true        ServiceAccount
services            svc         v1          true        Service
mutatingwebhookconfigurations admissionregistration.k8s.io/v1 false       MutatingWebhookConfiguration
validatingwebhookconfigurations admissionregistration.k8s.io/v1 false       ValidatingWebhookConfiguration
customresourcedefinitions crd,crds    apiextensions.k8s.io/v1 false       CustomResourceDefinition
apiservices         apiregistration.k8s.io/v1 false       APIService
controllerrevisions apps/v1      v1          true        ControllerRevision
daemonsets          ds          apps/v1     v1          true        DaemonSet
deployments          deploy      apps/v1     v1          true        Deployment
replicasets         rs          apps/v1     v1          true        ReplicaSet
statefulsets        sts         apps/v1     v1          true        StatefulSet
tokenreviews         authentication.k8s.io/v1 false       TokenReview
localsubjectaccessreviews authorization.k8s.io/v1 true        LocalSubjectAccessReview
selfsubjectaccessreviews authorization.k8s.io/v1 false       SelfSubjectAccessReview
selfsubjectrulesreviews authorization.k8s.io/v1 false       SelfSubjectRulesReview
subjectaccessreviews authorization.k8s.io/v1 false       SubjectAccessReview
horizontalpodautoscalers hpa         autoscaling/v1 true        HorizontalPodAutoscaler
```

Documentation des ressources

kubectl explain ressource

```
[root@master-node pods]# kubectl explain pod.spec.containers.command
```

```
KIND:      Pod
```

```
VERSION:   v1
```

```
FIELD:     command <[]string>
```

DESCRIPTION:

Entrypoint array. Not executed within a shell. The docker image's ENTRYPOINT is used if this is not provided. Variable references `$(VAR_NAME)` are expanded using the container's environment. If a variable cannot be resolved, the reference in the input string will be unchanged. The `$(VAR_NAME)` syntax can be escaped with a double `$$`, ie: `$$$(VAR_NAME)`. Escaped references will never be expanded, regardless of whether the variable exists or not. Cannot be updated. More info:

<https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/#running-a-command-in-a-shell>

Informations sur l'état des ressources

```
# kubectl get pods
```

```
# kubectl get po/www -o yaml
```

```
# kubectl description po/www
```

custom-columns

- Personnalisation de l'affichage
- Utilise l'approche jsonPath

```
[root@master-node pods]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
wp	2/2	Running	2	5h27m
www	1/1	Running	0	3h13m

```
[root@master-node pods]# kubectl get pods -o custom-columns=\n    'Name:metadata.name,Image:spec.containers[*].image'
```

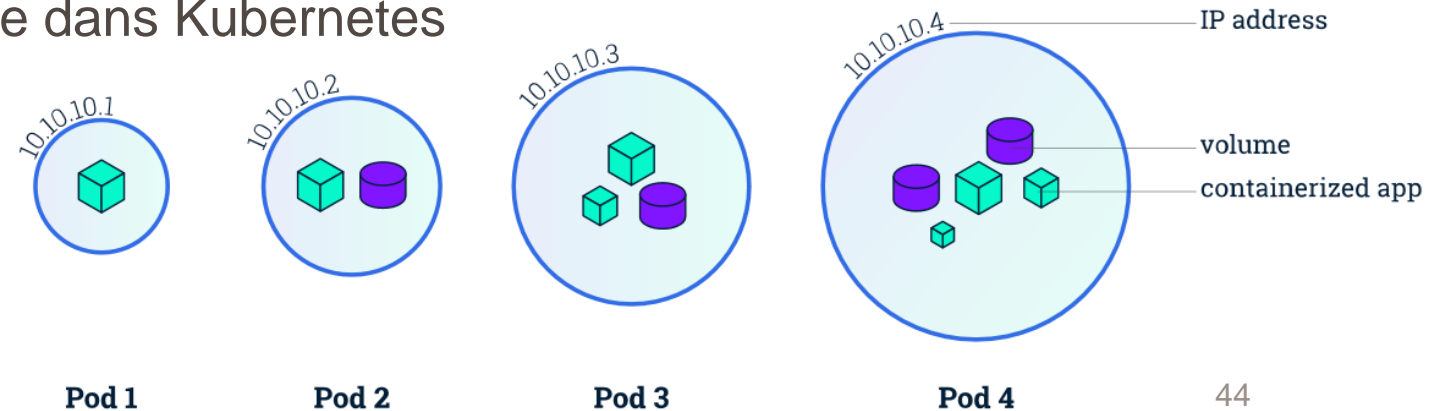
Name	Image
wp	wordpress:4.9-apache,mysql:5.7
www	nginx:1.16-alpine



LE POD

POD : Présentation

- Groupe de conteneurs tournant dans un même contexte d'isolation
 - Linux namespaces : network, IPC, UTS, ...
- Les conteneurs partagent certaines spécifications du POD :
 - La stack IP (network namespace). Adresse IP dédiée, pas de NAT pour la communication entre les Pods
 - Inter-process communication (PID namespace)
 - Volumes
- C'est la plus petite unité orchestrable dans Kubernetes



POD : Présentation

- Application découpée en plusieurs spécifications de Pods
- Chaque spécification correspond à un service métier (microservice)
- Scaling horizontal via le nombre de réplica d'un Pod
 - Création de nouveau Pod basé sur la même spécification

POD : Exemple - server http

- Spécification dans un fichier texte yaml (souvent préféré au format json)
- Exemple : Spécification d'un Pod dans lequel est lancé un container basé sur l'image nginx

```
# nginx-pod.yaml
```

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
  name: nginx
```

```
spec:
```

```
  containers:
```

```
    - name: www
```

```
      image: nginx:1.12.2
```

← L'objet Pod est stable et disponible depuis la version 1 de l'API.

← Spécification du type de l'élément, ici c'est un Pod

← Ajout d'un nom, d'autres metadata peuvent être ajoutées (labels, etc.)

← Spécification des conteneurs lancés dans le Pod (un seul ici). De nombreux paramètres possibles.

Cycle de vie

- Lancement d'un Pod

```
# kubectl create -f SPECIFICATION_POD.yaml
```

- Liste des Pods

```
# kubectl get pod
```

➡ Par défaut lister les pods du namespace « **default** »

- Description d'un Pod

```
# kubectl describe pod POD_NAME | po/POD_NAME
```

Cycle de vie

- Logs d'un conteneur d'un Pod

```
# kubectl logs POD_NAME [-c CONTAINER_NAME]
```

- Lancement d'une commande dans un Pod existant

```
# kubectl exec POD_NAME [-c CONTAINER_NAME] -- COMMAND
```

- Suppression d'un Pod

```
# kubectl delete pod POD_NAME
```


Cycle de vie

```
[root@master-node pods]# kubectl create -f nginx-pod.yaml
pod/nginx created
[root@master-node pods]# kubectl get pod
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   0           4s
[root@master-node pods]#
[root@master-node pods]# kubectl exec nginx -- nginx -v
nginx version: nginx/1.12.2
[root@master-node pods]#
[root@master-node pods]# kubectl exec -t -i nginx -- /bin/bash
root@nginx:/#
```

- ← Lancement du Pod
- ← Liste des Pods présents
- ← Lancement d'une commande dans un Pod
- ← Shell interactif dans un Pod

Cycle de vie

➡ Description d'un Pod

```
[root@master-node pods]# kubectl describe po/nginx
```

```
Name:          nginx
Namespace:     default
```

```
...
```

```
Events:
```

Type	Reason	Age	From	Message
----	-----	----	----	-----
Normal	Scheduled	6m50s	default-scheduler	Successfully assigned default/nginx to worker-node2
Normal	Pulled	6m49s	kubelet	Container image "nginx:1.12.2" already present on machine
Normal	Created	6m49s	kubelet	Created container www
Normal	Started	6m49s	kubelet	Started container www

➡ Suppression d'un Pod

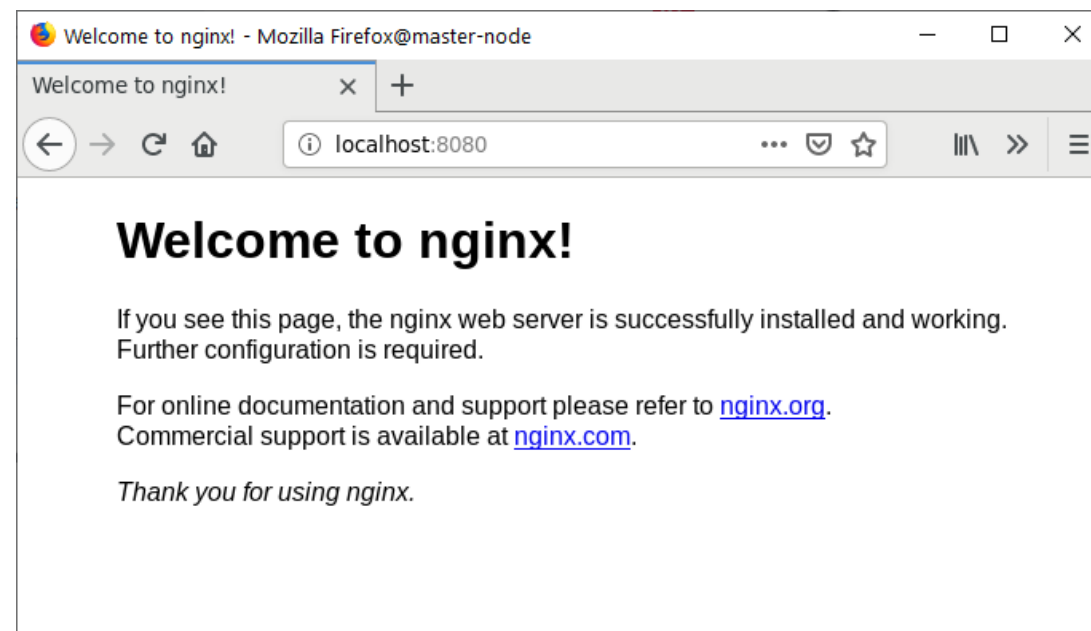
```
[root@master-node pods]# kubectl delete pod nginx
pod "nginx" deleted
```

Forward de port

- Commande utilisée pour le développement et le debugging
- Permet de publier le port d'un Pod sur la machine hôte

```
# kubectl port-forward POD_NAME Host_Port:Container_Port
```

```
[root@master-node pods]# kubectl port-forward nginx 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
Handling connection for 8080
```



Pod avec plusieurs conteneurs

- Exemple avec Wordpress
- Définition de 2 conteneurs dans le même pod
 - Application wordpress
 - Base de données MySQL
- Définition d'un volume pour la persistance des données de la base
 - Type **emptyDir** : associé au cycle de vie du Pod

Remarque : Ce n'est pas un setup de production car non scalable

Pod avec plusieurs conteneurs

```
apiVersion: v1
kind: Pod
metadata:
  name: wp
spec:
  containers:
  - image: wordpress:4.9-apache
    name: wordpress
    env:
      - name: WORDPRESS_DB_PASSWORD
        value: mysqlpwd
      - name: WORDPRESS_DB_HOST
        value: 127.0.0.1
  - image: mysql:5.7
    name: mysql
    env:
      - name: MYSQL_ROOT_PASSWORD
        value: mysqlpwd
    volumeMounts:
      - name: data
        mountPath: /var/lib/mysql
  volumes:
    - name: data
      emptyDir: {}
```

Conteneur pour l'application Wordpress

Conteneur pour la base Mysql

Montage du volume dans le conteneur Mysql

Montage d'un volume : répertoire sur la machine hôte

nodeSelector

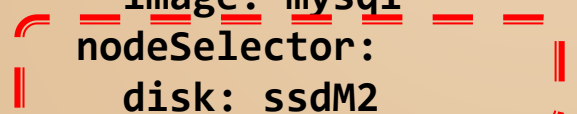
- Permet de scheduler un Pod sur un node ayant un label spécifique
- C'est la forme recommandée la plus simple pour spécifier une contrainte de sélection de nœud.

Ajout d'un label sur le node worker-node1

```
# kubectl label nodes worker-node1 disktype=ssd
node/worker-node1 labeled

# kubectl get node/worker-node1 -o yaml
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/arch: amd64
    kubernetes.io/hostname: worker-node1
    kubernetes.io/os: linux
    disktype: ssd
...
```

```
apiVersion: v1                                #mysql-pod.yaml
kind: Pod
metadata:
  name: db
  labels:
    env: prod
spec:
  containers:
    - name: mysql
      image: mysql
      nodeSelector:
        disk: ssdM2
```



Ajout d'une contrainte dans la spécification du Pod

nodeAffinity

- Permet de scheduler des Pods sur certains nodes seulement
- Plus granulaire que **nodeSelector**
- Autorise les opérateurs In, NotIn, Exists, doesNotExist, Gt, Lt
- Se base sur les labels existant sur les nodes
- Différentes règles
 - **required**DuringSchedulingIgnoredDuringExecution (Contrainte "**hard**")
 - **preferred**DuringSchedulingIgnoredDuringExecution (Contrainte "**Soft**")

nodeAffinity

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: kubernetes.io/e2e-az-name
                operator: In
                values:
                  - e2e-az1
                  - e2e-az2
      preferredDuringSchedulingIgnoredDuringExecution:
        - preference:
            matchExpressions:
              - key: disktype
                operator: In
                values:
                  - ssd
```

Le Pod devra être placé sur un node dont la valeur du label `kubernetes.io/e2e-az-name` est `e2e-az1` Ou `e2e-az2`

Parmi les nodes sélectionnés, le Pod sera schedulé de préférence sur un node dont la label `disktype` a la valeur `ssd`

podAffinity / podAntiAffinity

- Permet de scheduler des Pods en fonction de labels présents sur d'autres Pods
- Différentes règles
 - **requiredDuringSchedulingIgnoredDuringExecution** (Contrainte "hard")
 - **preferredDuringSchedulingIgnoredDuringExecution** (Contrainte "Soft")
- Utilise le champ **topologyKey** pour la spécification de domaines topologiques
 - Hostname
 - Region
 - ...

podAffinity / podAntiAffinity

```
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: security
                operator: In
                values:
                  - S1
            topologyKey: topology.kubernetes.io/zone
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
                - key: security
                  operator: In
                  values:
                    - S2
            topologyKey: topology.kubernetes.io/zone
```

Le Pod devra être placé sur un node qui est dans la même zone de disponibilité d'un Pod **topology.kubernetes.io/zone** dont la valeur du label **security** est **S1**

De préférence le Pod ne devra pas être placé sur un node sur lequel tourne un Pod dont le label **security** a la valeur **S2**

Taints et Tolerations

- Un Pod doit tolérer la taint d'un node pour pouvoir être exécuté sur celui-ci

Ajout d'un taint sur le node worker-node1

```
# kubectl taint nodes master-node node-role=master:NoSchedule
node/worker-node1 tainted
```

```
# kubectl get node/master-node -o yaml
apiVersion: v1
kind: Node
...
spec:
  podCIDR: 10.244.1.0/24
  podCIDRs:
  - 10.244.1.0/24
  taints:
  - effect: NoSchedule
    key: node-role
    value: master
...
```

```
apiVersion: v1
kind: Pod
metadata:
  name: fluend-agent
spec:
  tolerations:
  - key: "node-role"
    operator: "Equal"
    value: "master"
    effect: "NoSchedule"
  containers
  - ...
```

Allocation des ressources

- Consommation de la RAM et du CPU de chaque container d'un Pod

Ressources **minimales** nécessaires

Ressources **maximales** autorisées

```
apiVersion: v1
kind: Pod
metadata:
  name: db
spec:
  containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "mot de passe"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

LAB 2 – Manipulation des Pods

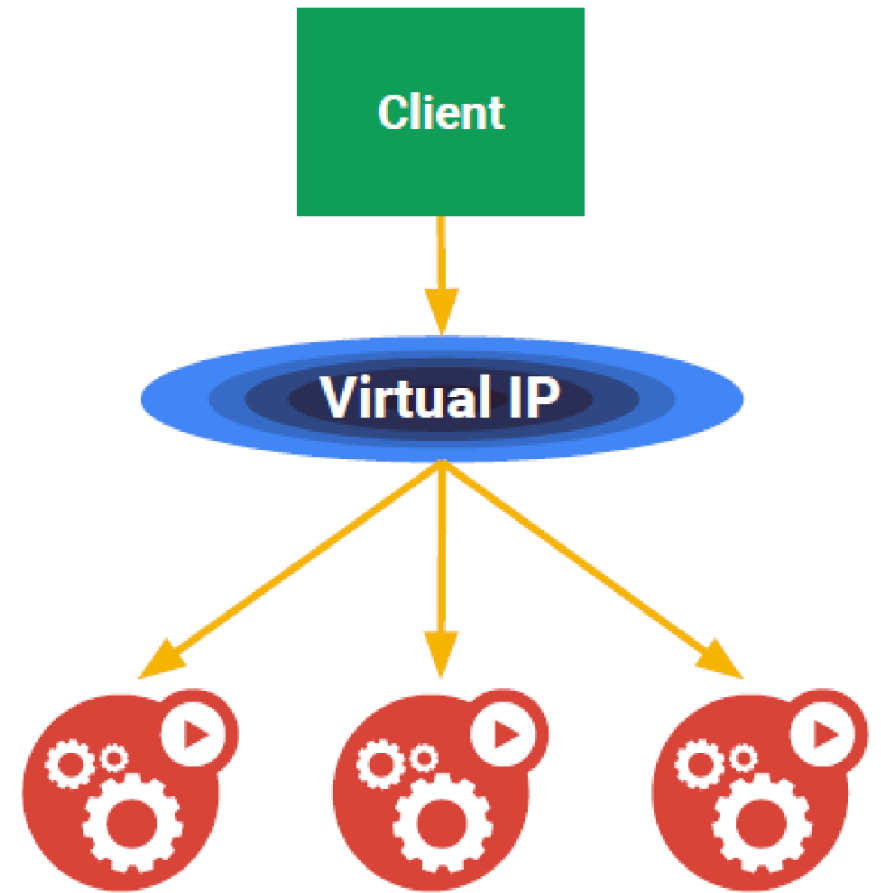




Les Services

Le service

- Abstraction définissant un ensemble logique de Pods
- Groupement basé sur l'utilisation de labels/selector pour des pods qui fonctionnent ensemble
- Rendre un ensemble de PODs accessibles depuis l'extérieur
- En charge de la répartition de la charge entre les Pods sous-jacents
- Adresse IP sera allouer pour chaque service



Différents types des services

- **ClusterIP** (par défaut) - Expose le service sur une adresse IP interne dans le cluster. Ce type rend le service uniquement accessible **à partir du cluster**.
 - Via SERVICE_NAME
 - Via SERVICE_NAME.NAMESPACE
- **NodePort** - Expose le service sur le même port de chaque nœud sélectionné dans le cluster à l'aide de NAT. Rend un service accessible **depuis l'extérieur du cluster** à l'aide de **<NodeIP>:<NodePort>**.
- **LoadBalancer**: intégration avec un Cloud Provider. crée une répartition de charge externe dans le cloud actuel et attribue une adresse IP externe fixe au service.
AWS, GCE, Azure, ...

Cycle de vie d'un service

- Lancement d'un service

```
# kubectl create -f SERVICE_SPECIFICATION.yaml
```

- Description d'un Service

```
# kubectl describe svc SERVICE_NAME
```

- Suppression d'un Service

```
# kubectl delete svc SERVICE_NAME
```

Spécification d'un service

Exemple de type **ClusterIP** (par défaut)

vote-service-clusterIP.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 80
```

- ← Service est stable depuis la version 1 de l'API
- ← Spécification du type de l'objet
- ← Ajout d'un nom pour identifier le service
- ← Indique les Pods que le service va regrouper (les Pods ayant le label « app:vote »)
- ← Type par défaut, d'autres Pods accèdent au service par son nom
- ← Le service expose le port 80 dans le cluster
targetPort : c'est le port utilisé pour renvoyer les requête sur le Pod de regroupement

Spécification d'un service

Chaque requête reçue par le service est envoyée sur l'un des Pods ayant le label spécifié

Vote-app.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: vote
  labels:
    app: vote
spec:
  containers:
    - name: vote
      image: instavote/vote
      ports:
        - containerPort: 80
```

vote-service-clusterIP.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 80
```

Spécification d'un service

Exemple de type **NodePort**

vote-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 31000
```

← Service de type NodePort, exposé sur chaque node du cluster

← Le service expose le port 80 dans le cluster

← Requêtes renvoyées sur le port 80 d'un des Pods du groupe

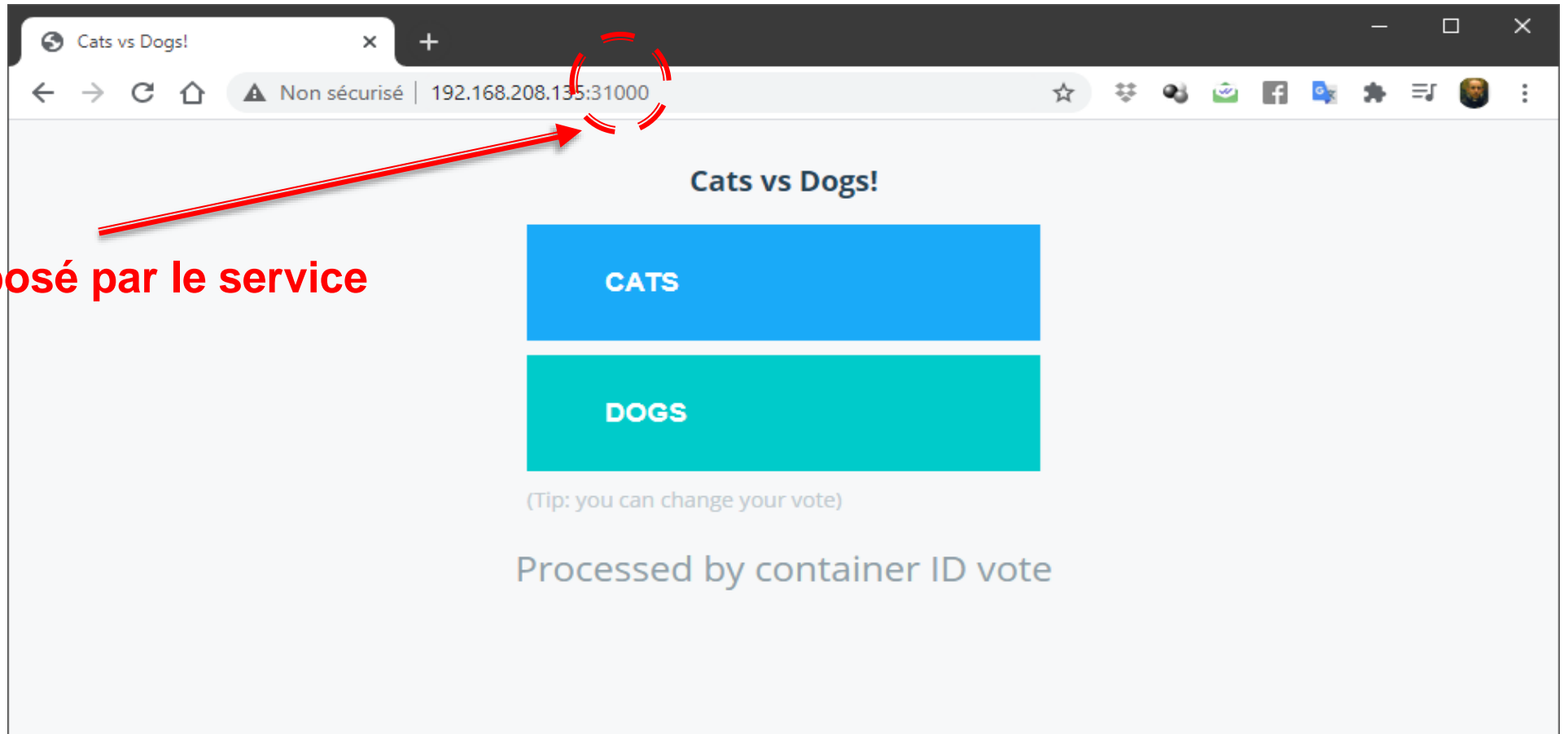
← Service accessible depuis le port 31000 de chaque node

Cycle de vie d'un service

- Lancement d'un service

```
# kubectl create -f vote-service.yaml
```

Port 31000 exposé par le service



LAB 3 - Manipulation des Services

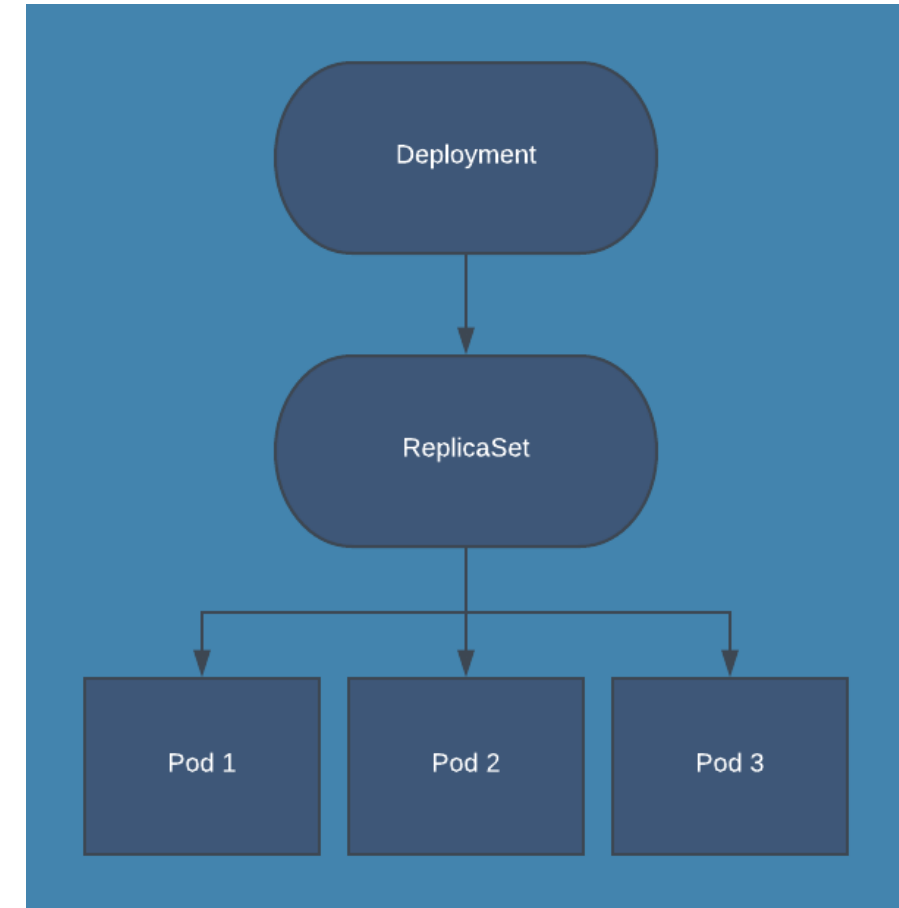




Les Objets Deployment

Les Objets Deployment

- Différents niveaux d'abstraction:
 - Deployment
 - ReplicaSet
 - Pod
- Un Deployment gère des ReplicaSet
- ReplicaSet:
 - Gère un ensemble de Pods de même spécification
 - Assure que les Pods sont actifs
- Pod généralement créés via un Déploiement



Spécification d'un Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote-deploy
```

Deployment

spec:

```
  replicas: 3
  selector:
    matchLabels:
      app: vote
```

ReplicatSet

template:

```
  metadata:
    labels:
      app: vote
```

spec:

```
  containers:
    - name: vote
      image: instavote/vote
      ports:
        - containerPort: 80
```

Pod

Spécification d'un Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: vote-deploy
```

```
spec:
```

```
  replicas: 3
```

```
  selector:
```

```
    matchLabels:
```

```
      app: vote
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: vote
```

```
    spec:
```

```
      containers:
```

```
        - name: vote
```

```
          image: instavote/vote
```

```
          ports:
```

```
            - containerPort: 80
```

← Version de l'API dans laquelle l'objet Deployment est défini

← "Deployment" est le type de la ressource considérée

← Le nom "vote-deploy" est donné au Deployment

← Définition de la façon dont seront lancés les Pods

← 3 réplicas seront créés pour ce Deployment

← Détermine les Pods qui seront managés par ce Deployment, Seuls les Pods ayant le label **app:vote** seront considérés

← Spécification des Pods, correspond à la clé spec utilisée lors de la définition d'un Pod

Spécification d'un Deployment

Lancement du Deployment

```
[root@master-node deployment]# kubectl create -f vote-deployment.yaml
deployment.apps/vote-deploy created
```

Liste des Deployments

```
[root@master-node deployment]# kubectl get deploy
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
vote-deploy	1/3	3	1	15s

Un ReplicatSet est créé et associé au Deployment

Liste des ReplicatSet

```
[root@master-node deployment]# kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
vote-deploy-7c4d84d659	3	3	3	2m15s

Le ReplicatSet gère les 3 Pods (réplicas) définis dans la spécification du Deployment

Liste des Pods

```
[root@master-node deployment]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
vote-deploy-7c4d84d659-7j94x	1/1	Running	0	2m33s
vote-deploy-7c4d84d659-hmtwf	1/1	Running	0	2m33s
vote-deploy-7c4d84d659-qlt9f	1/1	Running	0	2m33s

Stratégies de mise à jour d'un Deployment

- Recreate
 - Supprime l'ancienne version et crée la nouvelle
- Rolling update
 - Release graduelle de la nouvelle version
- Blue/green
 - Ancienne et nouvelle version déployées ensemble
 - Mise à jour du trafic via le Service exposant l'application

Mise à jour d'un Deployment: rolling update

Création d'un nouveau Deployment à partir d'un fichier de spécification

```
# kubectl create -f vote-deployment.yaml --record=true  
deployment.apps/vote-deploy created
```

Enregistrement des changements effectués dans chaque révision

1^{ère} mise à jour de l'image du conteneur vote défini dans la spécification

```
# kubectl set image deployment/vote-deploy vote=instavote:indent  
deployment.apps/vote-deploy image updated
```

2^{ième} mise à jour de l'image

```
# kubectl set image deployment/vote-deploy vote=instavote:movies  
deployment.apps/vote-deploy image updated
```

Liste des ReplicaSet pour ce Deployment

```
# kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
vote-deploy-55fc97b7cc	3	3	3	14s
vote-deploy-74d89ffb8b	0	0	0	49s
vote-deploy-7c4d84d659	0	0	0	4m29s

Un nouveau ReplicaSet a été créé pour chaque version de l'application

Mise à jour d'un Deployment: historique

- **--record=true** permet d'enregistrer les changements de chaque révision
- Facilite la sélection de la révision pour un **rollback**

```
# kubectl rollout history deploy/vote-deploy
deployment.apps/vote-deploy
REVISION  CHANGE-CAUSE
1          kubectl create --filename=vote-deployment.yaml --record=true
2          kubectl create --filename=vote-deployment.yaml --record=true
3          kubectl create --filename=vote-deployment.yaml --record=true
```

```
# kubectl describe deploy/vote-deploy
Name:                vote-deploy
...

Annotations:         deployment.kubernetes.io/revision: 3
                    kubernetes.io/change-cause: kubectl create --filename=vote-deployment.yaml --
record=true.
...
```

Mise à jour d'un Deployment: rollback

- Rollback vers la révision précédente ou une révision ultérieure
 - `# kubectl rollout undo ...`
 - `# kubectl rollout undo ... --to-revision=X`

```
# kubectl rollout undo deploy/vote-deploy  
deployment.apps/vote-deploy rolled back
```

```
# kubectl rollout undo deploy/vote-deploy --to-revision=1  
deployment.apps/vote-deploy rolled back
```

LAB 4 - Manipulation des Deployments





Les Objets - Namespace

Les Objets - Namespace

- Isoler des ressources : les Pods, Service, Deployments, ...
- Partager un cluster (approche multi-tenant)
 - Équipe / Projets / Clients
- 4 namespaces par défaut

```
# kubectl get namespaces
NAME                STATUS    AGE
default             Active    26h
kube-node-lease     Active    26h
kube-public         Active    26h
kube-system         Active    26h
```

- Ressources créés dans le namespace default si non spécifié

Les Objets – Namespace : Création

- Création du namespace development (Option 1)

```
# kubectl create namespace development
namespace/development created
```

- Suppression du namespace development

```
# kubectl delete namespaces development
namespace "development" deleted
```

- Création du namespace depuis un fichier de spécification (option 2)

```
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "development",
    "labels": {
      "name": "development"
    }
  }
}
```

```
# kubectl create -f development.yaml
namespace/development created
```

Les Objets – Namespace : Utilisation

- Pod avec namespace spécifié dans les metadata

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: development
spec:
  containers:
  - name: www
    image: nginx:1.14
```

← Ce Pod sera déployé dans le namespace development

Les Objets – Namespace : Utilisation

- Lancement d'un Pod dans le namespace development

```
# kubectl create -f nginx-pod-dev.yaml  
pod/nginx created
```

- Liste des Pods dans le namespace development

```
# kubectl get po --namespace=development  
NAME      READY   STATUS    RESTARTS   AGE  
nginx     1/1     Running   0           62s
```

- Liste des Pods dans l'ensemble des namespaces

```
# kubectl get po --all-namespaces
```

Les Objets – Namespace : Le contexte

```
# kubectl config view
```

- Le namespace par défaut dans kubernetes est : **default**
- Pour changer le namespace default

```
# kubectl config set-context $(kubectl config current-context) --namespace=development
```

Les Objets – Namespace : Le contexte

- Création d'un pod dans le contexte modifié

```
# kubectl run w3 --image=nginx:1.14
```

- Liste des Pods du namespace **development**

```
[root@master-node ~]# kubectl get pod --namespace development
```

NAME	READY	STATUS	RESTARTS	AGE
nginx	0/1	Pending	0	25m
w3	0/1	Pending	0	56s

- Liste des Pods du namespace **default**

```
[root@master-node ~]# kubectl get pod --namespace default
```

No resources found.

Lab 5 - Manipulation des Namespaces



Vos Questions

