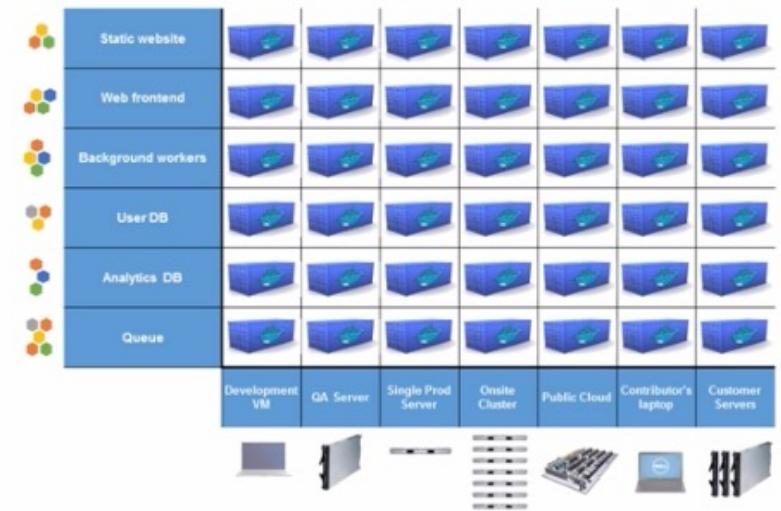


# Kubernetes

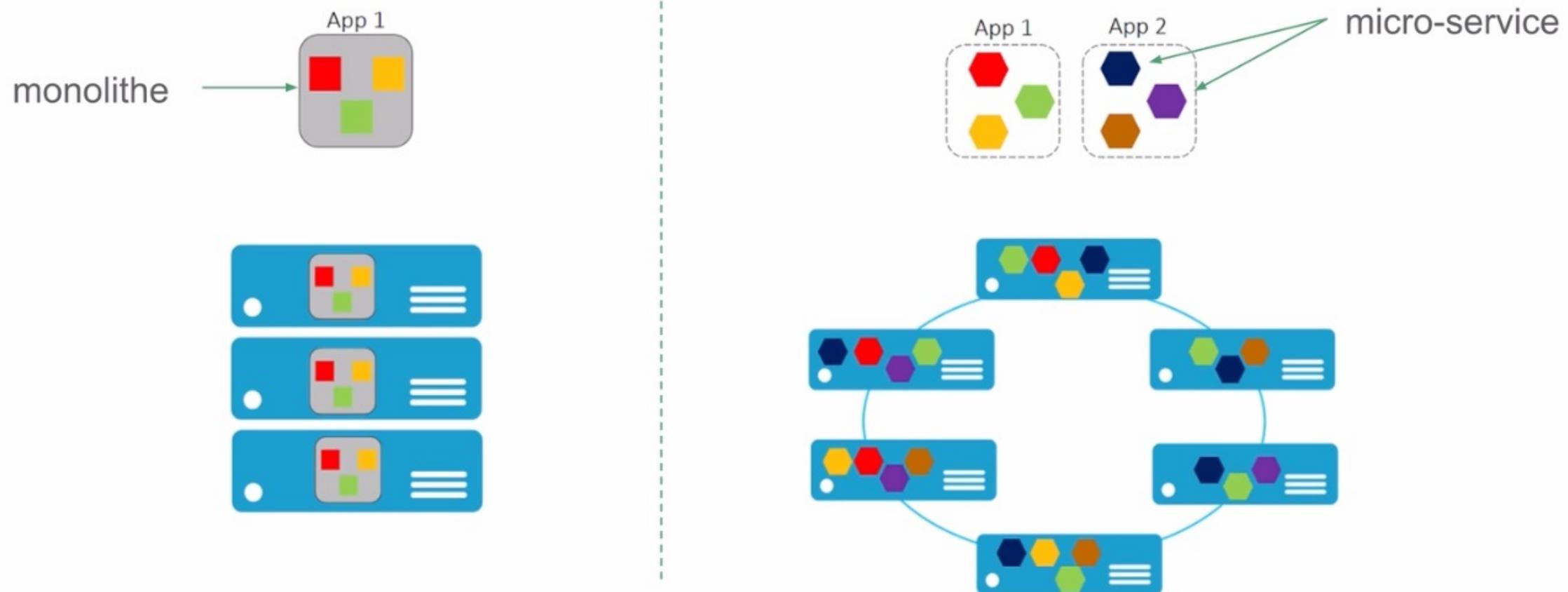
# La plateforme Docker

- Facilite l'utilisation des containers
- Apporte le concept d'image
  - packaging d'une application et de ses dépendances
  - instanciée dans un container
  - déploiement sur une multitude d'environnements
- Nombreuses images disponibles sur le Docker Hub : <https://hub.docker.com>



“Matrix from hell”

# Architecture monolithique vs micro-services



# Architecture micro-services

- Découpage de l'application en multiples services
- Processus indépendant ayant sa propre responsabilité métier
- Plus grande liberté de choix dans le langage
- Équipe dédiée pour chaque service
- Un service peut être mis à jour indépendamment des autres services
- Containers très adaptés pour les micro-services
- Nécessite des interfaces bien définies
- Déplace la complexité dans l'orchestration de l'application globale

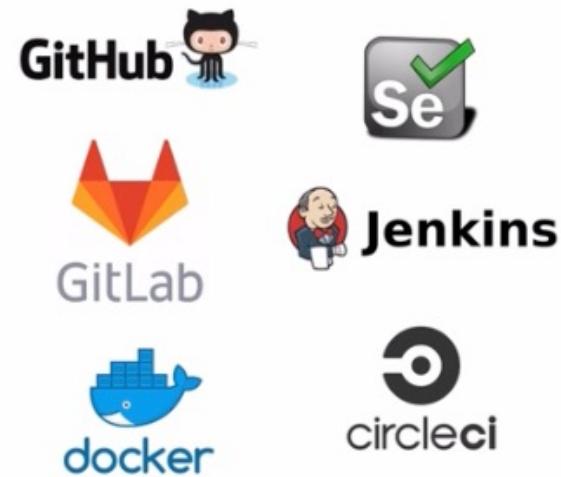
# Application Cloud Native

- Application orientée microservices
- Packagée dans des containers
- Orchestration dynamique
- Nombreux projets portés par la CNCF (Cloud Native Computing Foundation)
  - Kubernetes
  - Prometheus
  - Fluentd
  - ...
- [cncf.io](https://cncf.io)

# DevOps

- Un objectif : minimiser le temps de livraison d'une fonctionnalité
- Déploiements réguliers
- Mise en avant des tests
- Automatisation des processus
  - provisioning / configuration
  - Infrastructure As Code (IaC)
  - Intégration Continue / Déploiement Continu (CI/CD)
  - monitoring
- Une boucle d'amélioration courte

# DevOps : quelques outils et produits



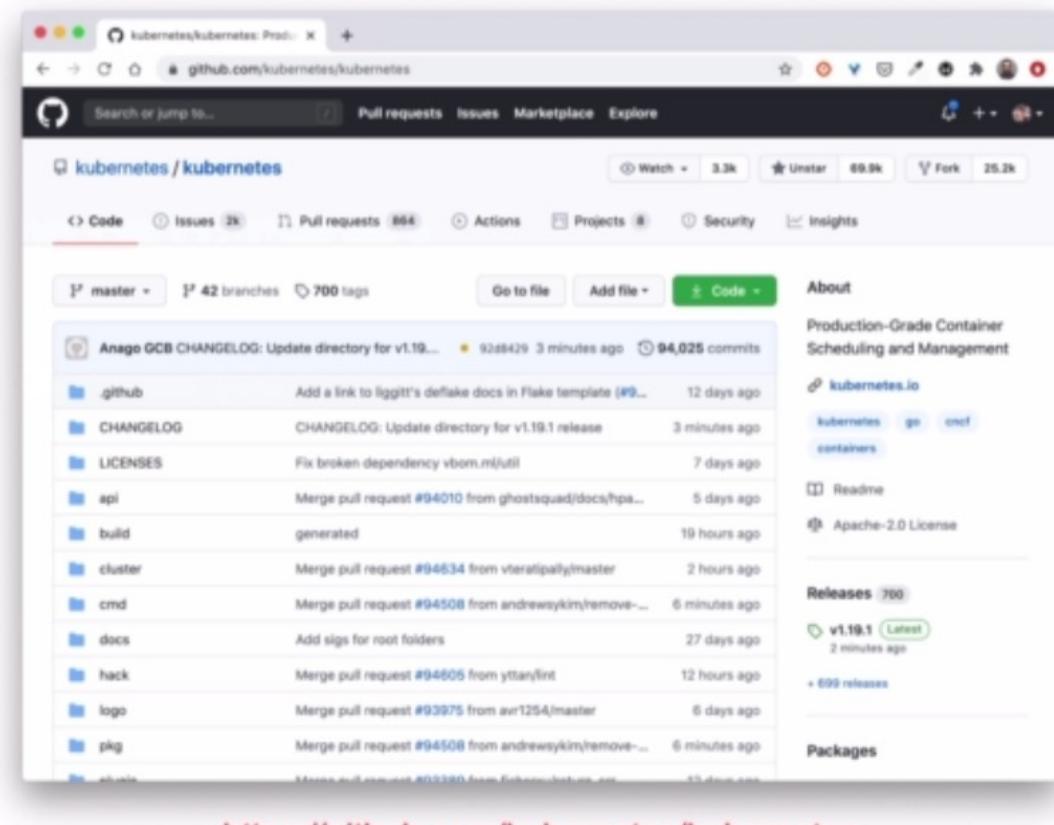
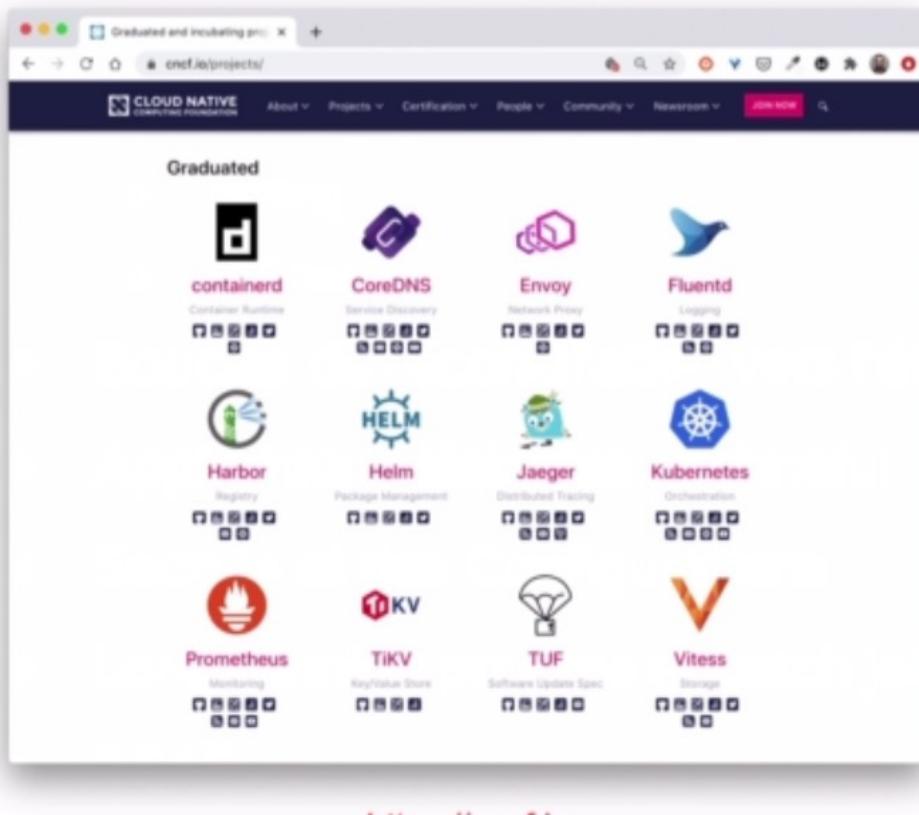
# Historique

- Kubernetes / k8s / kube
- “Homme de barre” / “Pilote” en grec
- Plateforme open source d’orchestration de containers
- Inspirée du système Borg de Google
- v1.0, juillet 2015
- v1.19.0, aout 2020

# Fonctionnalités

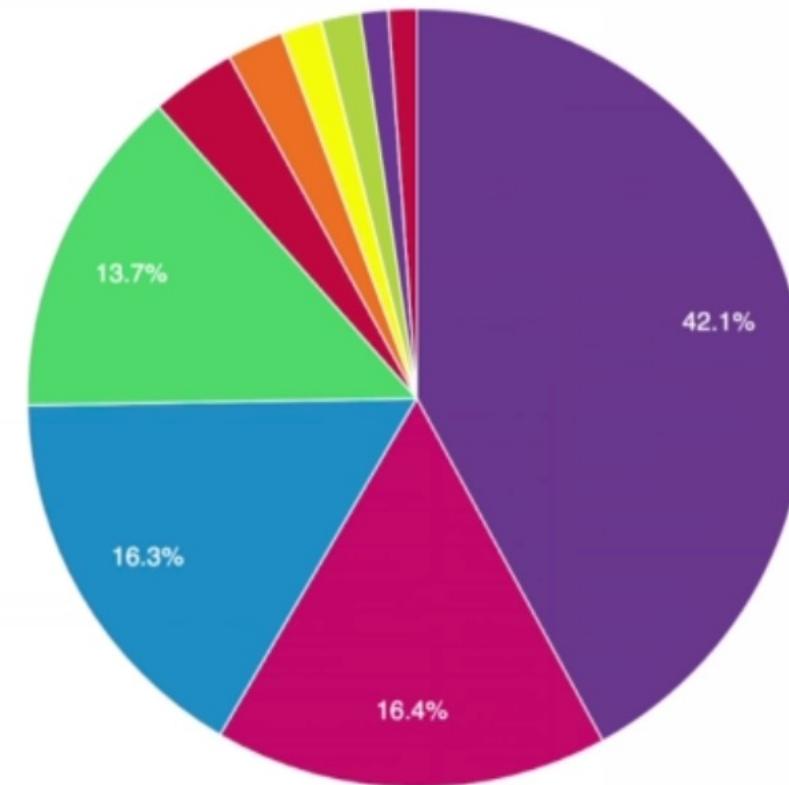
- Gestion d'applications tournant dans des containers
  - déploiement
  - scaling
  - self-healing
- Boucles de réconciliation vers l'état souhaité (contrôleurs)
- Applications stateless et stateful
- Secrets et des Configurations
- Long-running process et batch jobs
- RBAC

# Un projet phare de l'écosystème



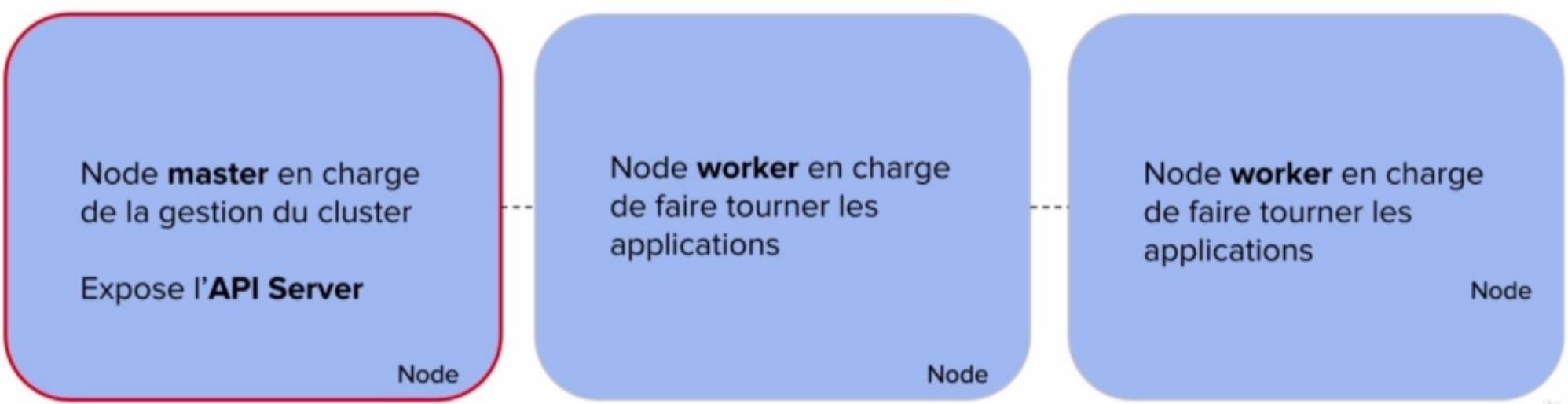
## Les principaux contributeurs (janvier 2020)

#	Company	Commits
1	Google	21202
	*independent	8242
2	Red Hat	8212
3	Huawei	1788
4	ZTE Corporation	1181
5	Microsoft	860
6	VMware	815
7	FathomDB	587
8	IBM	585
9	CoreOS	505



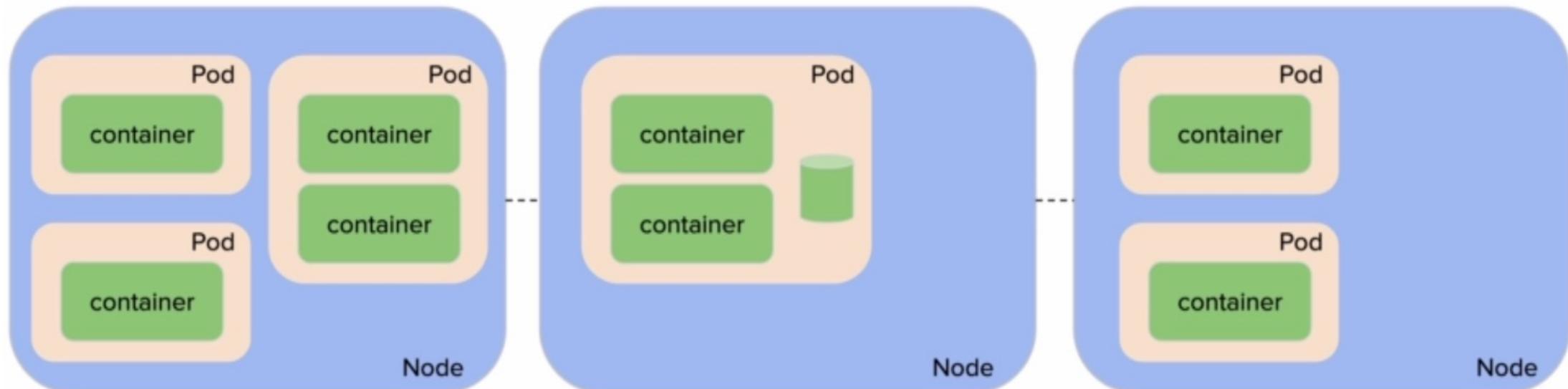
# Cluster

- Ensemble de **nodes** Linux ou Windows (VM / bare metal)
- Nodes **Masters** + nodes **Workers**
- Un Master expose l'**API Server** - point d'entrée pour la gestion du cluster



# Pods

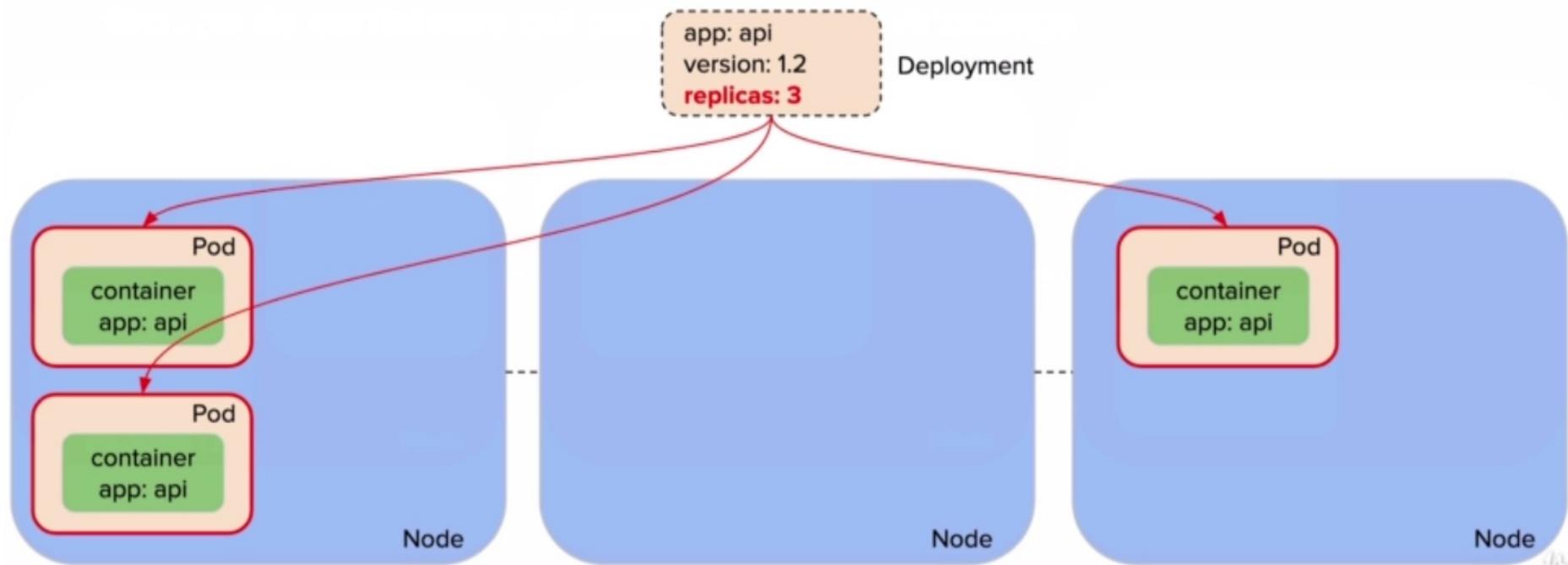
- Plus petite unité applicative qui tourne sur un cluster Kubernetes
- Groupe de **containers** qui partagent réseau/stockage



# Deployment

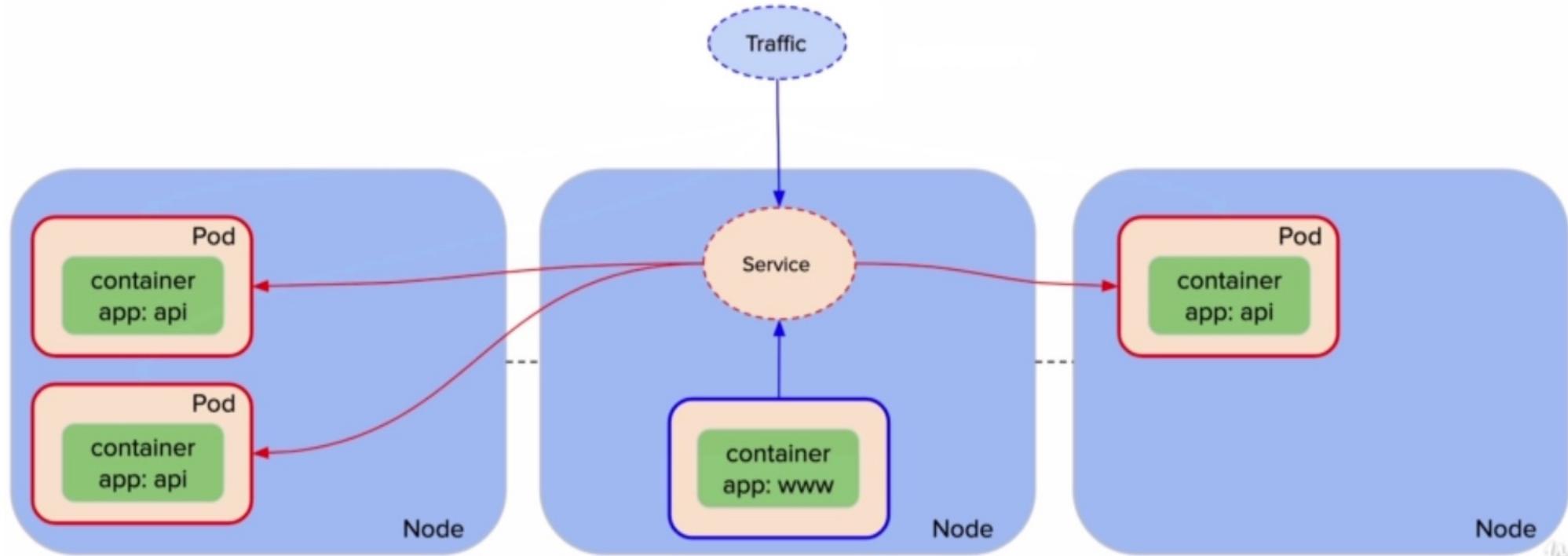
## Deployment

Permet de gérer un ensemble de **Pods** identiques (mise à jour / rollback)



# Service

Expose les applications des **Pods** à l'intérieur ou à l'extérieur du cluster



# Spécification des ressources

- Généralement en **yaml**
- Une structure commune

```
apiVersion: v1
kind: Pod
metadata:
  name: www
  labels:
    app: w3
spec:
  containers:
    - name: www
      image: nginx:1.16
```

Exemple de spécification d'un Pod

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

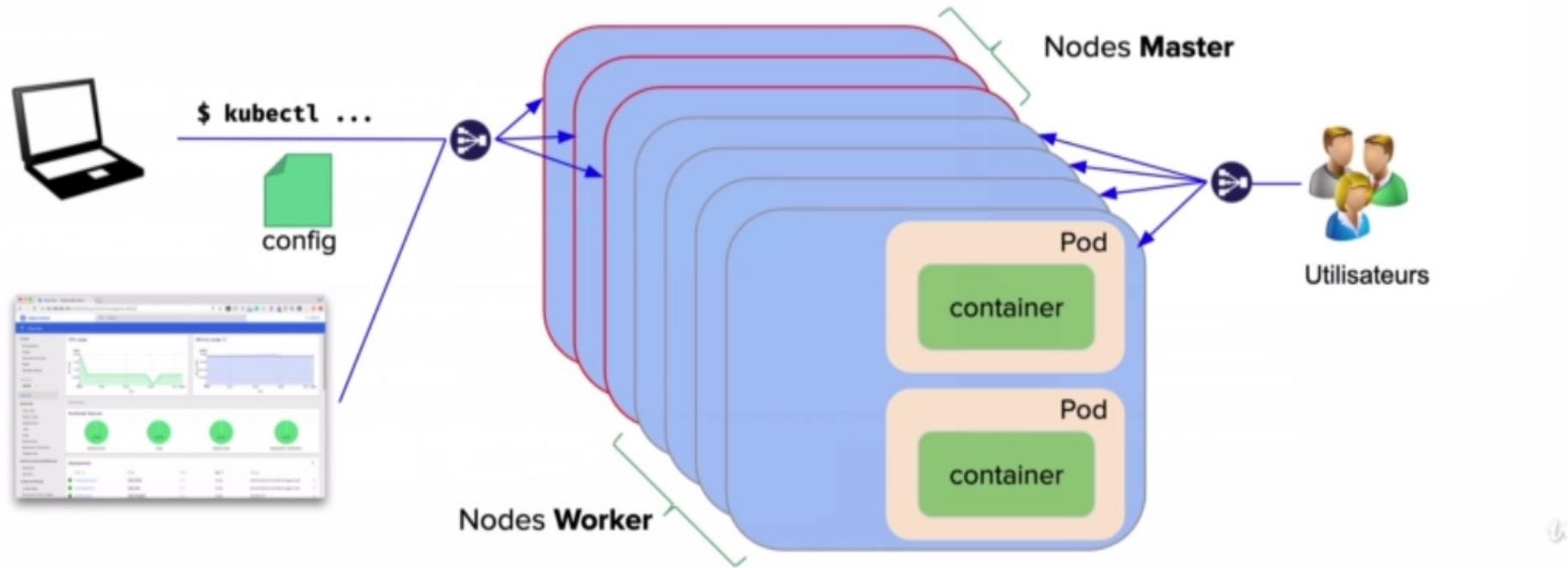
Exemple de spécification d'un Service

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: www
spec:
  replicas: 3
  selector:
    matchLabels:
      app: w3
  template:
    metadata:
      labels:
        app: w3
    spec:
      containers:
        - name: www
          image: nginx:1.16
```

Exemple de spécification d'un Deployment

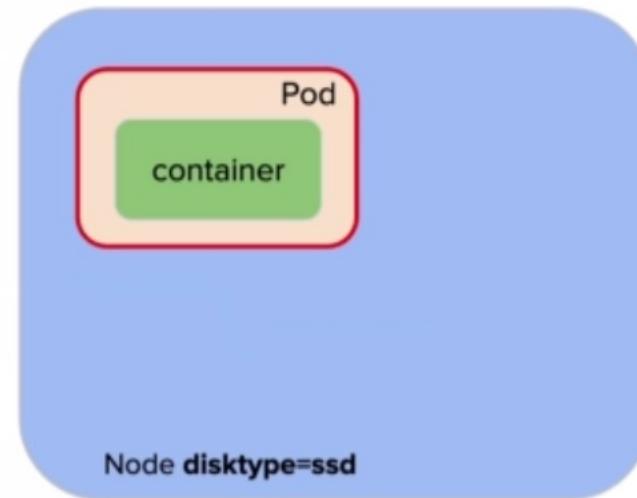
# Gestion du cluster

- Envoi de requêtes HTTPS à l'**API Server**
- Utilisation du binaire **kubectl** ou de l'une des nombreuses interfaces (web / CLI)

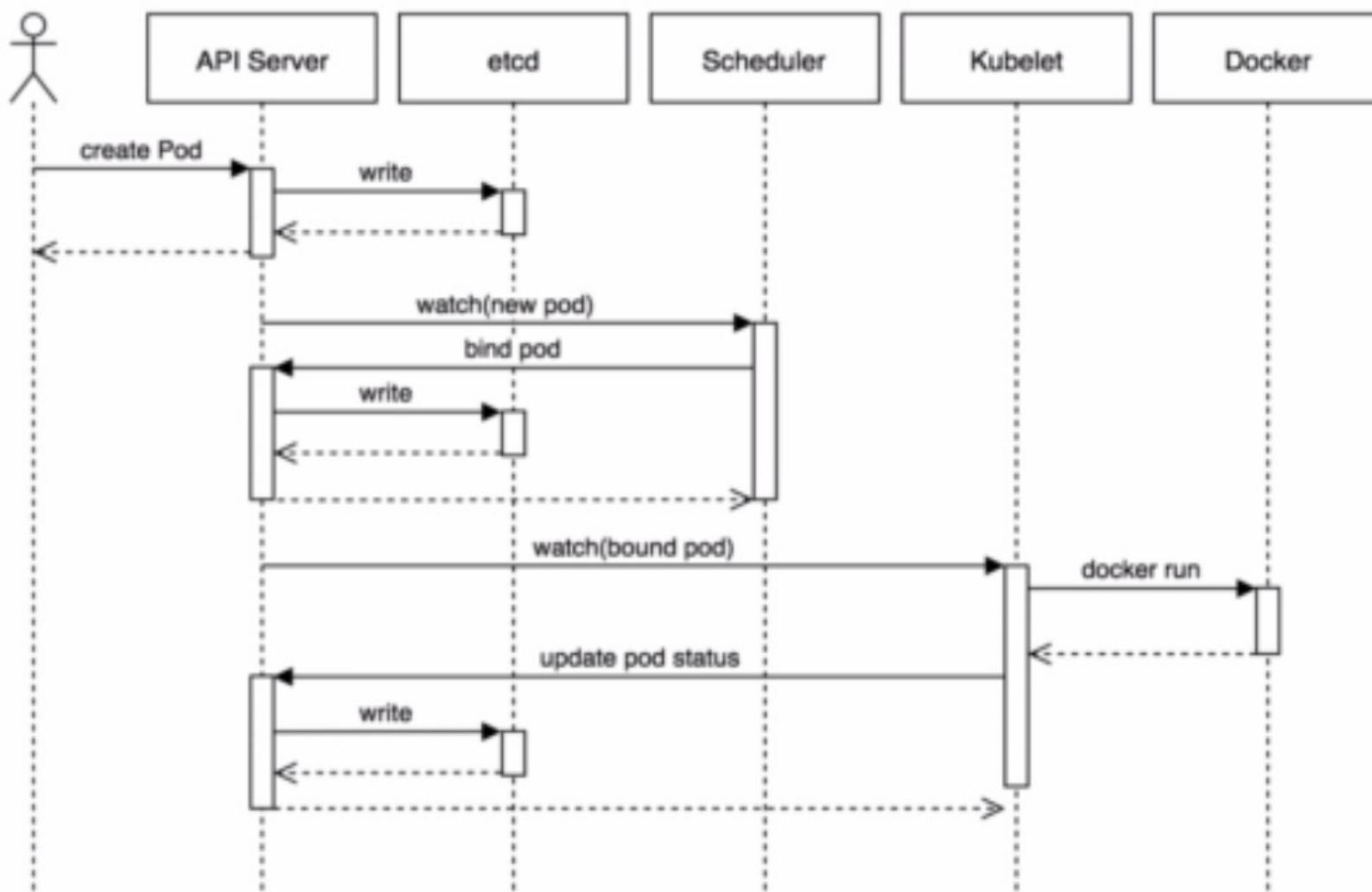


# Labels et Annotations

- Information attachées aux ressources
- Format : key / value
- Labels
  - utilisés pour la sélection d'objets
  - récupération de collections
- Annotations
  - ne sert pas à sélectionner des objets
  - non interne à Kubernetes
  - utilisées par des outils et librairies clientes



# Les processus : workflow de création d'un Pod



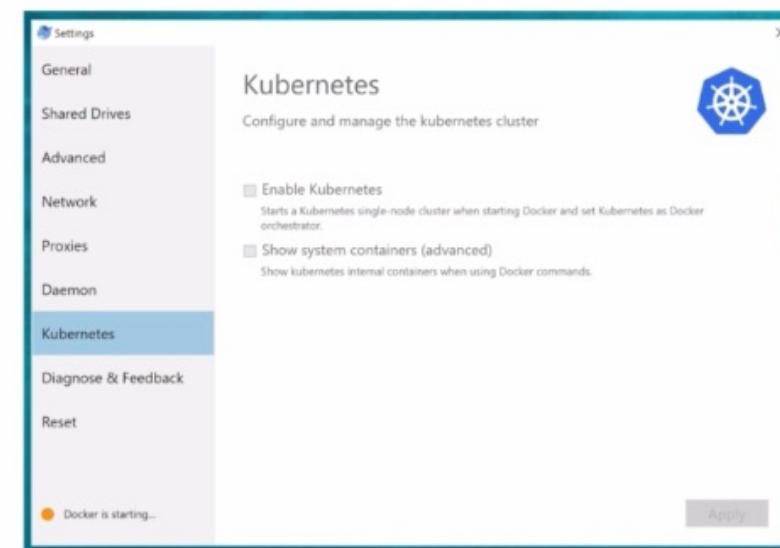
# Minikube

- Tous les composants de Kubernetes dans une seule VM locale
- S'intègre avec différents hyperviseurs
  - HyperKit
  - Hyper-V
  - KVM
  - VirtualBox
  - VMWare
- Nécessite le binaire minikube
  - <https://github.com/kubernetes/minikube/releases>



# Docker Desktop (macOS / Windows)

- Intégration de la version upstream de Kubernetes
- Déploiement sur Swarm ou Kubernetes pendant le développement
- <https://hub.docker.com/editions/community/docker-ce-desktop-windows>
- <https://hub.docker.com/editions/community/docker-ce-desktop-mac>



# Kind (Kubernetes in Docker)



- <https://github.com/kubernetes-sigs/kind>
- Les nodes tournent dans des container Docker
- Cluster HA via un fichier de configuration

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

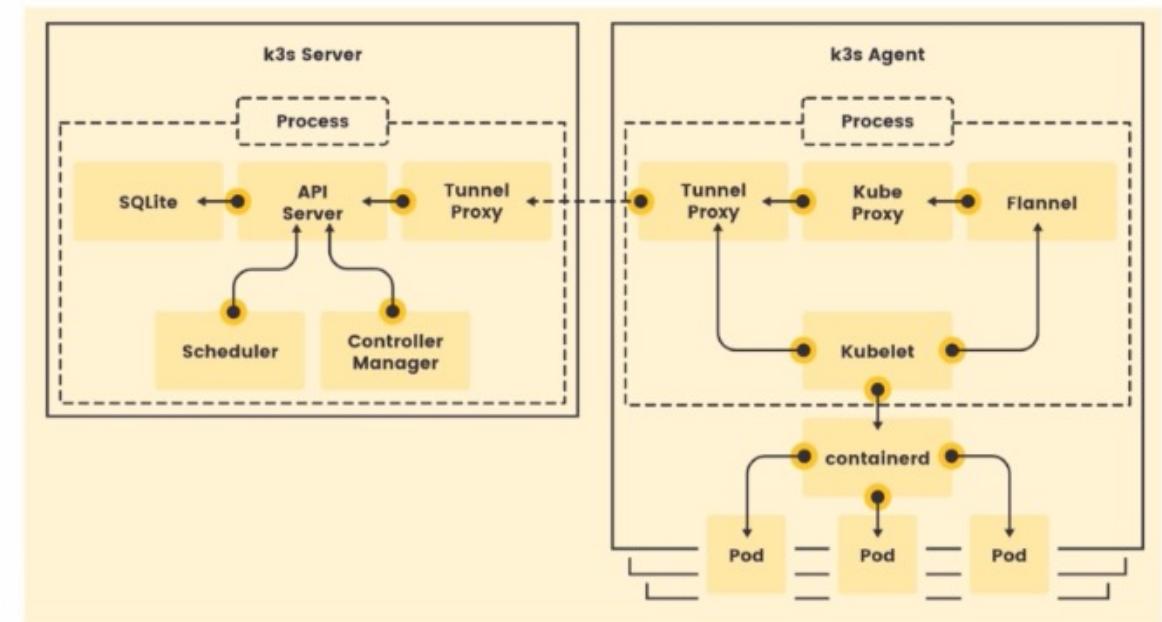
✓ Preparing nodes 🏭
✓ Writing configuration 📜
✓ Starting control-plane 🏪
✓ Installing CNI 🚧
✓ Installing StorageClass 🏫
Set kubectl context to "kind-k8s"
You can now use your cluster with:
kubectl cluster-info --context kind-k8s
Have a nice day! 🎉
```



# K3S

“Kubernetes léger: la distribution Kubernetes certifiée conçue pour l’IoT et l’Edge computing”

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



# 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”

```
$ snap install microk8s --classic  
  
$ microk8s.kubectl get nodes  
NAME      STATUS    ROLES      AGE      VERSION  
microk8s   Ready     <none>    41s      v1.17.0
```



Kelsey Hightower  @kelseyhightower

I think [@Canonical](#) might have assembled the easiest way to provision a single node Kubernetes cluster:

```
$ snap install microk8s --classic  
  
microk8s.io
```

Traduire le Tweet  
1:39 AM · 24 avr. 2019 · Twitter Web App

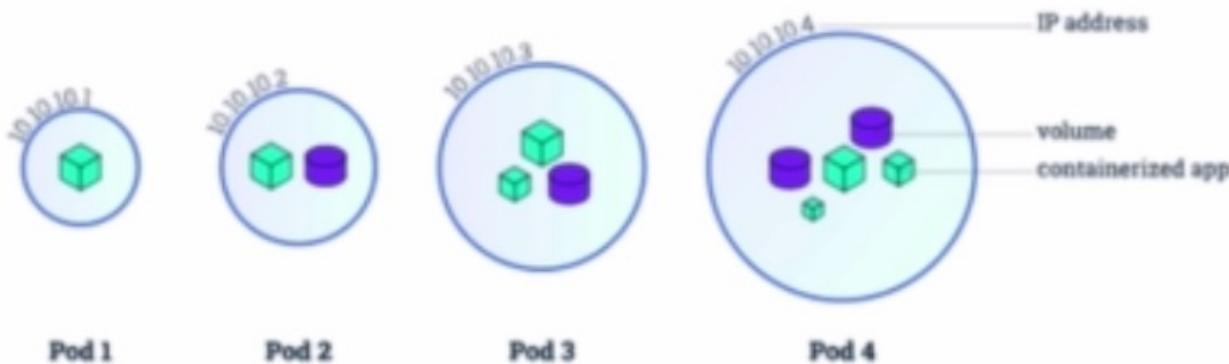
278 Retweets 1,1 k J'aime

Reply Retweet Like Share

# Les Pods

# Présentation

- Groupe de containers tournant dans un même contexte d'isolation
  - Linux namespaces : network, IPC, UTS, ...
- Partagent la stack réseau et le stockage (volumes)
- Adresse IP dédiée, pas de NAT pour la communication entre les Pods



Source: [kubernetes.io](https://kubernetes.io)

# Présentation

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

## Exemple - server http

- Spécification dans un fichier texte yaml (souvent préféré au format json)

```
$ cat nginx-pod.yaml
apiVersion: v1          L'objet Pod est stable et disponible depuis la v1 de l'API
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: www
    image: nginx:1.12.2
```

Spécification d'un Pod dans lequel est lancé un container basé sur l'image nginx

## Exemple - server http

- Spécification dans un fichier texte yaml (souvent préféré au format json)

```
$ cat nginx-pod.yaml
apiVersion: v1
kind: Pod          Spécification du type de l'élément, ici nous définissons un Pod
metadata:
  name: nginx
spec:
  containers:
    - name: www
      image: nginx:1.12.2
```

Spécification d'un Pod dans lequel est lancé un container basé sur l'image nginx

## Exemple - server http

- Spécification dans un fichier texte yaml (souvent préféré au format json)

```
$ cat nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:          Ajout d'un nom, d'autres metadata peuvent être ajoutées (labels,...)
  name: nginx
spec:
  containers:
  - name: www
    image: nginx:1.12.2
```

Spécification d'un Pod dans lequel est lancé un container basé sur l'image nginx

## Exemple - server http

- Spécification dans un fichier texte yaml (souvent préféré au format json)

```
$ cat nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: www
      image: nginx:1.12.2
```

Spécification des containers lancés dans le Pod (un seul ici)  
De nombreux paramètres possibles

Spécification d'un Pod dans lequel est lancé un container basé sur l'image nginx

# Cycle de vie

- Lancement d'un Pod
  - `$ kubectl create -f POD_SPECIFICATION.yaml`
- Liste des Pods
  - `$ kubectl get pod`
  - namespace “default”
- Description d'un Pod
  - `$ kubectl describe pod POD_NAME`
  - `$ kubectl describe po/POD_NAME`

## Cycle de vie

- Logs d'un container 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

```
# Lancement du Pod
$ kubectl create -f nginx-pod.yaml

# Liste des Pods présents
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
www           1/1     Running   0          2m

# Lancement d'une commande dans un Pod
$ kubectl exec www -- nginx -v
nginx version: nginx/1.12.2

# Shell interactif dans un Pod
$ kubectl exec -t -i www -- /bin/bash
root@nginx:/#
```

# Cycle de vie

```
# Détails du Pod
$ kubectl describe po/www
Name:           www
Namespace:      default
Node:          minikube/192.168.99.100
...
Events:
  Type  Reason          Age   From            Message
  ----  ----          ----  ----
  Normal Scheduled       18s   default-scheduler  Successfully assigned nginx to minikube
  Normal SuccessfulMountVolume 18s   kubelet, minikube  MountVolume.SetUp succeeded for volume "default-token-brp4l"
  Normal Pulled          18s   kubelet, minikube  Container image "nginx:1.12.2" already present on machine
  Normal Created         18s   kubelet, minikube  Created container
  Normal Started         18s   kubelet, minikube  Started container
```

# Suppression du Pod

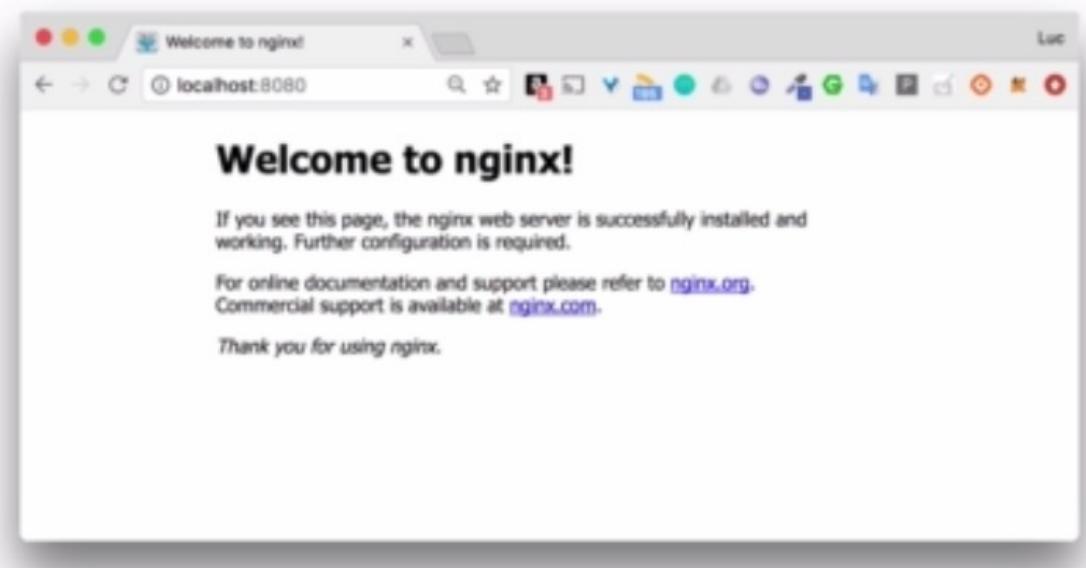
```
$ kubectl delete pod www
pod "www" deleted
```

Ensemble des événements survenus lors du lancement du Pod

# Forward de port

- Commande utilisée pour le développement et debugging
- Permet de publier le port d'un Pod sur la machine hôte
- `$ kubectl port-forward POD_NAME HOST_PORT:CONTAINER_PORT`

```
$ kubectl port-forward www 8080:80
Forwarding from 127.0.0.1:8080 -> 80
```



## Un container particulier

- Présence d'un container **pause**
- Utilisé pour la création et le partage des namespaces avec les autres containers du Pod

```
$ docker ps | grep www
3563ea53ba87 nginx@sha256:547...63e ... k8s_nginx.www_default_b0...e385_0
58845df5b20f gcr.io/google_containers/pause-amd64:3.0 ... k8s_POD.www_default_b0...e385_0
```

## Pod avec plusieurs containers

- Exemple avec Wordpress
- Définition de 2 containers dans le même pod
  - moteur Wordpress
  - base de données MySQL
- Définition d'un volume pour la persistence des données de la base
  - type **emptyDir** : associé au cycle de vie du Pod

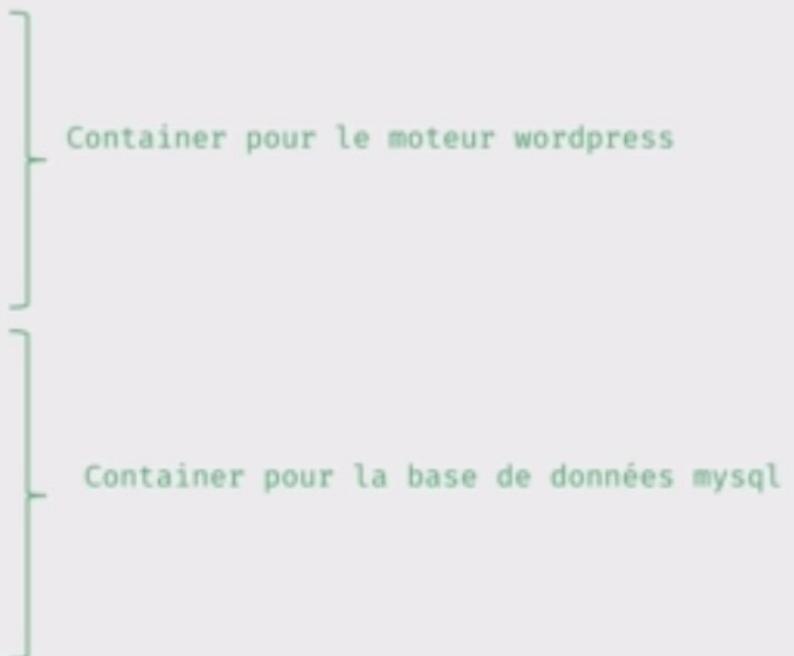
Note: ce n'est pas un setup de production car non scalable

# Pod avec plusieurs containers

```
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: {}
```

# Pod avec plusieurs containers

```
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: {}
```



# Pod avec plusieurs containers

```
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: {}
```

Montage du volume dans le container mysql

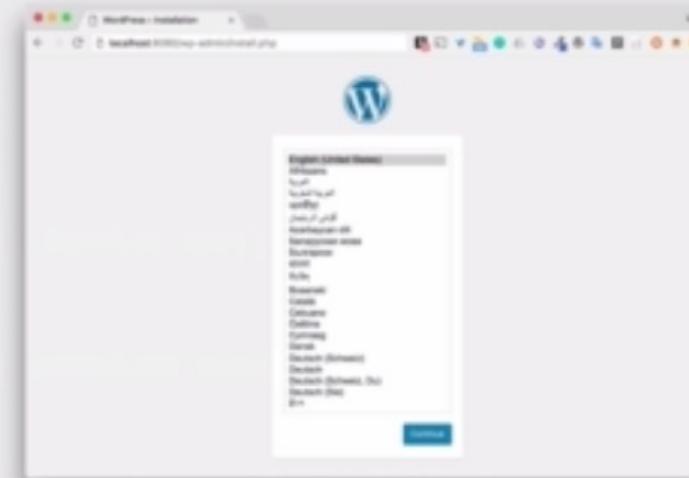
Définition d'un volume: répertoire sur la machine hôte

# Pod avec plusieurs containers

```
# Création du Pod
$ kubectl create -f wordpress-pod.yaml
Pod "wp" created

# Liste des Pod présent
$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
wp        2/2     Running   0          18s

# Exposition du port 80 du container wordpress
$ kubectl port-forward wp 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Handling connection for 8080
```



# Scheduling

# Rôle

- Sélection du node sur lequel un Pod sera déployé
- Etape effectuée par le composant **kube-scheduler**

```
$ kubectl run www --image=nginx:1.16-alpine --restart=Never
pod/www created

$ kubectl describe pod www
...
Events:
Type    Reason     Age   From           Message
----    -----     ---   ----          -----
Normal  Scheduled  12s   default-scheduler  Successfully assigned default/www to pool-ytvhmq
Normal  Pulling    10s   kubelet, pool-ytvhmq  Pulling image "nginx:1.16-alpine"
Normal  Pulled     6s    kubelet, pool-ytvhmq  Successfully pulled image "nginx:1.16-alpine"
Normal  Created    6s    kubelet, pool-ytvhmq  Created container www
Normal  Started    5s    kubelet, pool-ytvhmq  Started container www
```

# nodeSelector

Permet de schéduler un Pod sur un node ayant un label spécifique

```
# Ajout d'un label sur le node1
$ kubectl label nodes node1 disktype:ssd

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

```

```
apiVersion: v1
kind: Pod
metadata:
  name: db
  labels:
    env: prod
spec:
  containers:
  - name: mysql
    image: mysql
    nodeSelector:
      disktype: ssd
```

Ajout d'une contrainte dans la spécification du Pod

mysql-pod.yaml

## nodeAffinity (1/2)

- Permet de schéduler 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
  - requiredDuringSchedulingIgnoredDuringExecution (contrainte "hard")
  - preferredDuringSchedulingIgnoredDuringExecution (contrainte "soft")

## nodeAffinity (2/2)

```
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 schédulé de préférence sur un node dont le label **disktype** a la valeur **ssd**

<https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity>

## podAffinity / podAntiAffinity (1/2)

- Permet de schéduler des Pods en fonction de labels présents sur d'autre Pods
- Différentes règles
  - requiredDuringSchedulingIgnoredDuringExecution (contrainte “hard”)
  - preferredDuringSchedulingIgnoredDuringExecution (contrainte “soft”)
- Utilise le champ **topologyKey** pour la spécification de domaines topologiques
  - hostname
  - region
  - az
  - ...

## podAffinity / podAntiAffinity (2/2)

```
spec:  
  affinity:  
    podAffinity:  
      requiredDuringSchedulingIgnoredDuringExecution:  
        - labelSelector:  
            matchExpressions:  
              - key: security  
                operator: In  
                values:  
                  - S1  
            topologyKey: failure-domain.beta.kubernetes.io/zone  
    podAntiAffinity:  
      preferredDuringSchedulingIgnoredDuringExecution:  
        - podAffinityTerm:  
            labelSelector:  
              matchExpressions:  
                - key: security  
                  operator: In  
                  values:  
                    - S2  
            topologyKey: kubernetes.io/hostname  
...  
...
```

Le Pod devra être placé sur un node qui est dans la même zone de disponibilité d'un Pod 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

```
$ kubectl get node/node1 -o yaml
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: node1
    disktype: ssd
spec:
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
...
...
```

```
apiVersion: v1
kind: Pod
metadata:
  name: fluentd-agent
spec:
  tolerations:
  - key: node-role.kubernetes.io/master
    effect: NoSchedule
  containers:
  - ...

```

fluentd-pod.yaml

# Allocation des ressources

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

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



The code snippet shows a YAML configuration for a Kubernetes Pod named 'db'. It contains one container named 'db' using the 'mysql' image. The container has an environment variable 'MYSQL\_ROOT\_PASSWORD' set to 'password'. It also specifies resource requirements and limits. The 'requests' section sets the minimum required resources to 64 MiB of memory and 250 mili-CPU. The 'limits' section sets the maximum allowed resources to 128 MiB of memory and 500 mili-CPU. Two green curly braces group these sections: one brace groups the 'requests' and 'limits' sections under the heading 'ressources minimales nécessaires' (minimum required resources), and another brace groups them under the heading 'ressources maximales autorisées' (maximum allowed resources).

ressources minimales nécessaires

ressources maximales autorisées

## Création avec l'approche impérative

```
$ kubectl run db --image mongo:4.0
```

*kubectl >= 1.18*

```
$ kubectl run db --generator=run-pod/v1 --image=mongo:4.0
```

ou

```
$ kubectl run db --image=mongo:4.0 --restart=Never
```

*kubectl < 1.18*

# Génération de la spécification d'un Pod

```
$ kubectl run db \
  --image mongo:4.0 \
  --dry-run=client \
  -o yaml
```



```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: db
    name: db
spec:
  containers:
  - image: mongo:4.0
    name: db
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  ...
```

L'option **--dry-run** simule la création d'une ressource

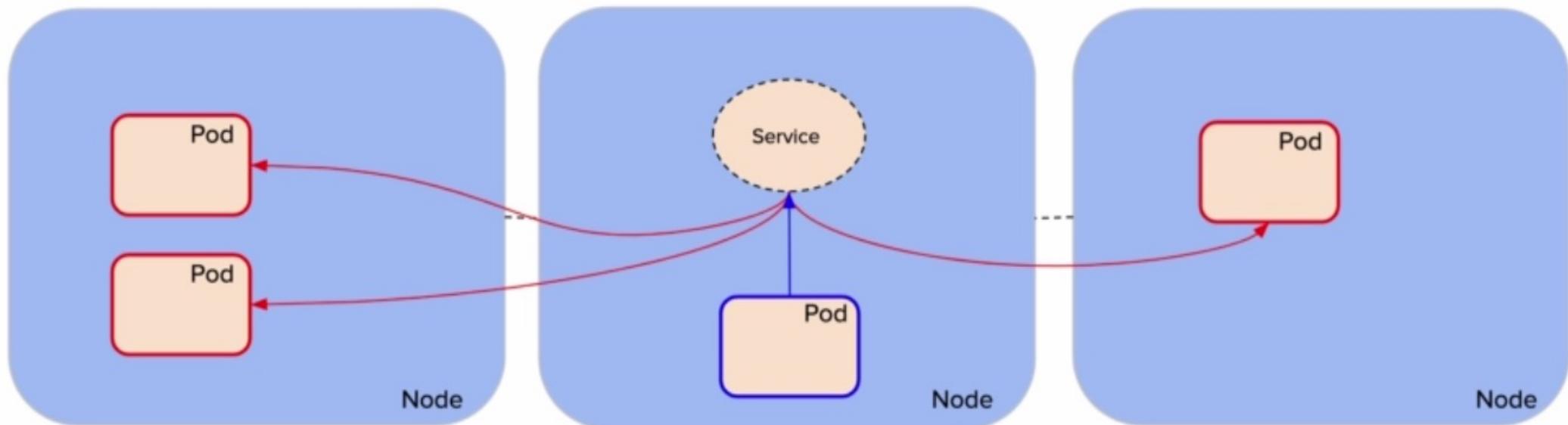
- version < 1.18 ⇒ elle s'utilise sans valeur
- version >= 1.18 ⇒ 2 valeurs possibles
  - \* client: la ressource n'est pas envoyée à l'API Server
  - \* server: traitée par l'API Server mais non persistée

# Les Services

## Rôle

- Expose les Pods d'une application via des règles réseaux
- Utilise des labels pour grouper les Pods
- Adresse IP persistante (VIP : virtual IP address)
- kube-proxy en charge du load balancing sur les Pods
  - userspaces / iptables / **IPVS**
- Différents types
  - ClusterIP (défaut) : exposition à l'intérieur du cluster
  - NodePort : exposition vers l'extérieur
  - LoadBalancer : intégration avec un Cloud Provider
  - ExternalName: associe le service à un nom DNS

## Service de type ClusterIP



# Service de type ClusterIP : exemple

```
apiVersion: v1
kind: Service                                Spécification du type de l'objet
metadata:
  name: vote
spec:
  selector:
    app: vote
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
```

service.yaml

# Service de type ClusterIP : exemple

```
apiVersion: v1
kind: Service
metadata:
  name: vote
    Ajout d'un nom pour identifier le service
spec:
  selector:
    app: vote
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
```

service.yaml

# Service de type ClusterIP : exemple

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

Indique les Pods que le service va exposer (ceux ayant le label "app: vote")

service.yaml

# Service de type ClusterIP : exemple

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

Le service expose le port 80 dans le cluster  
Requêtes envoyées sur le port 80 d'un Pods du groupe

service.yaml

# Service de type ClusterIP : exemple

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

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

pod.yaml



`kubectl apply -f pod.yaml`

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

service.yaml



`kubectl apply -f service.yaml`

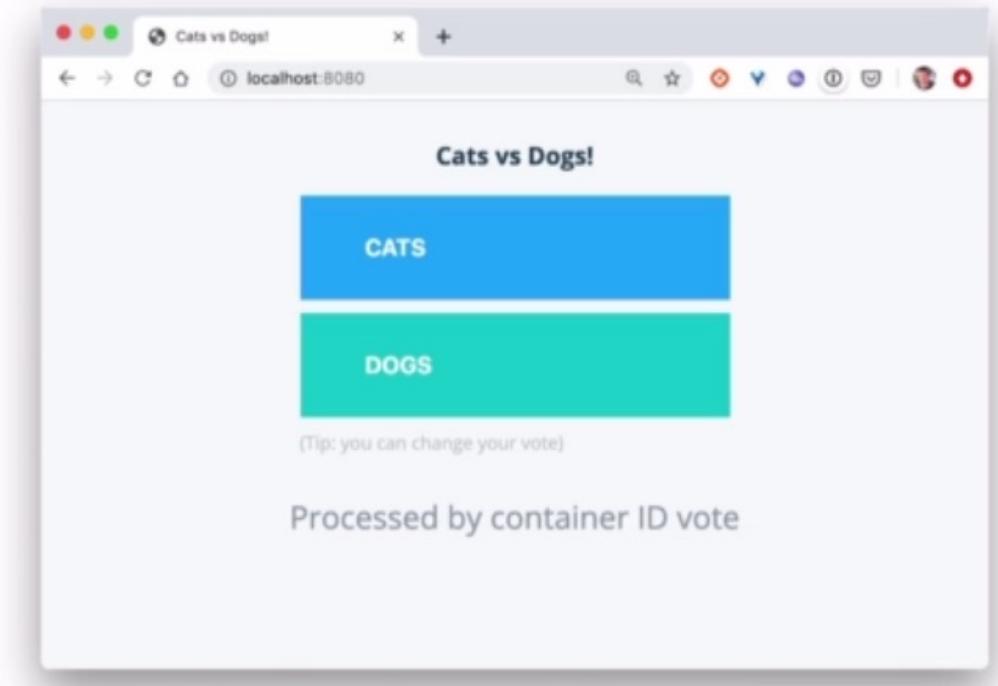
## Service de type ClusterIP : accès depuis un Pod

```
# Lancement d'un Pod
$ kubectl run -ti test --image=alpine

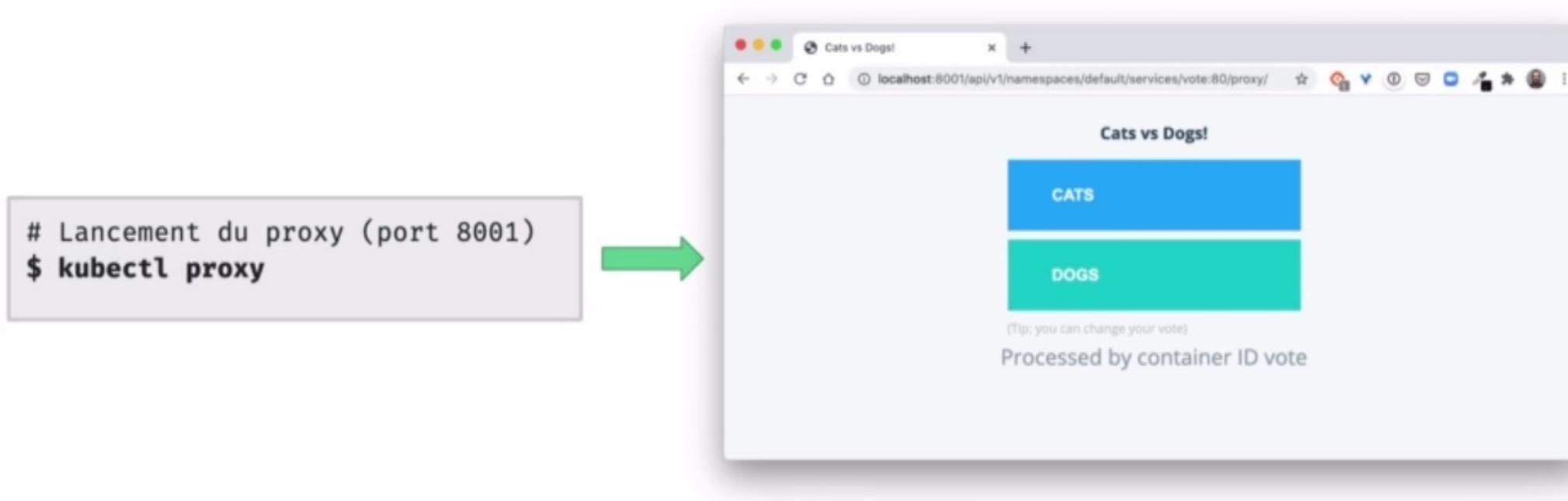
# Accès au service depuis un shell dans ce Pod
/ # apk add -u curl
/ # curl http://vote
<!DOCTYPE html>
<html>
  <head>
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

## Service de type ClusterIP : accès via le port-forward

```
# Accès au service depuis l'extérieur  
$ kubectl port-forward svc/vote 8080:80
```



# Service de type ClusterIP : accès via le proxy

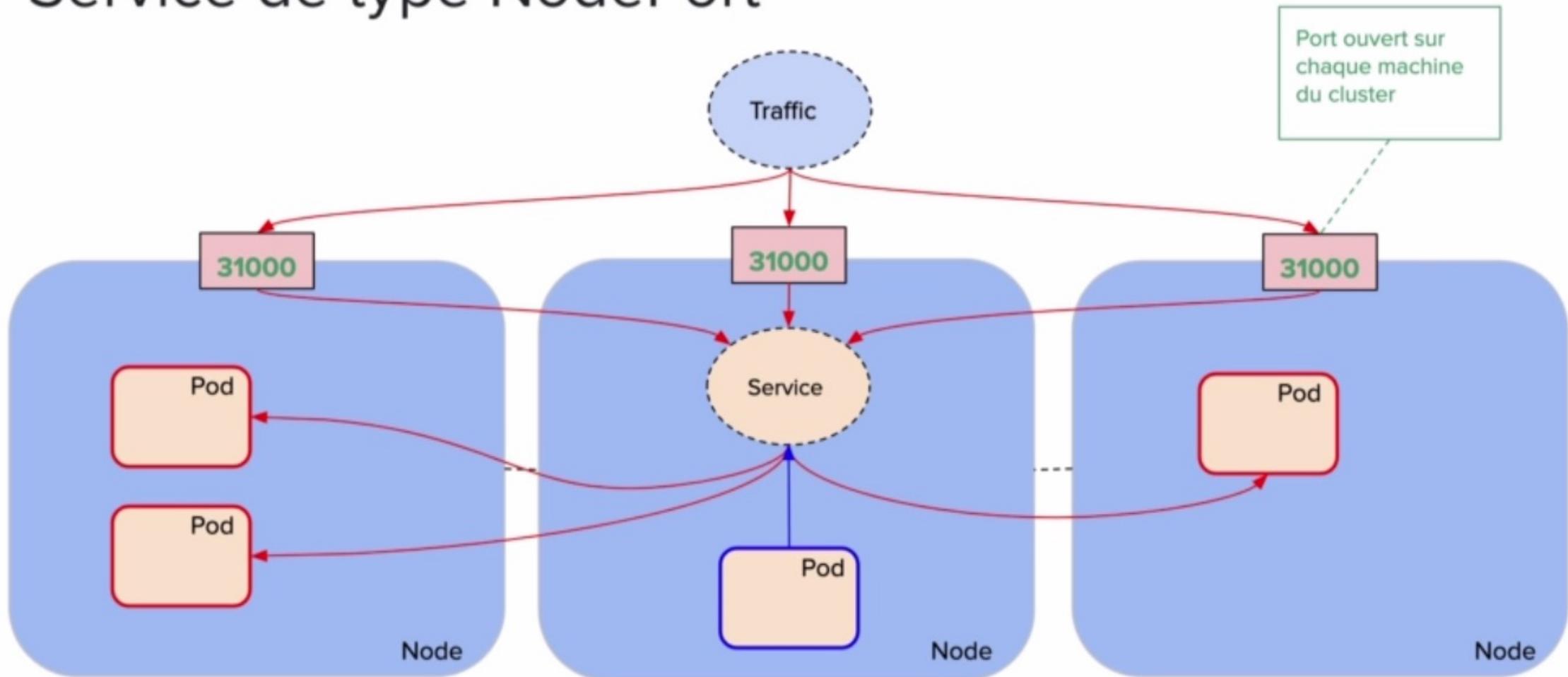


Service accessible au travers du proxy via l'URL:

<http://localhost:8001/api/v1/namespaces/default/services/vote:80/proxy>



# Service de type NodePort



# Service de type NodePort : exemple

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

Service de type NodePort, exposé sur chaque node du cluster

service-np.yaml

# Service de type NodePort : exemple

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

Le service expose le port 80 dans le cluster

service-np.yaml

# Service de type NodePort : exemple

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

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

service-np.yaml

# Service de type NodePort : exemple

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

Service accessible depuis le port 31000 de **chaque node** du cluster

service-np.yaml

## Service de type NodePort : exemple

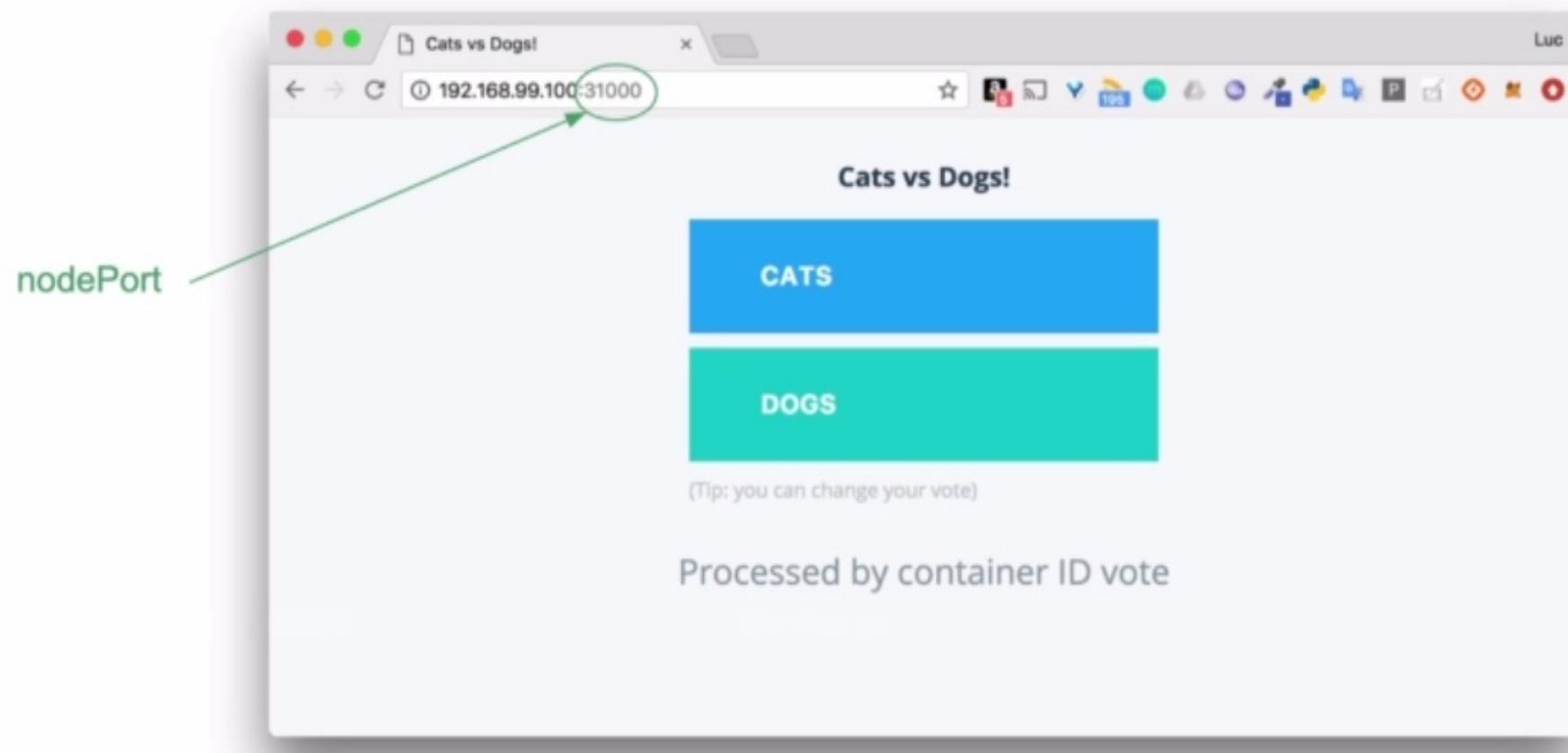
```
apiVersion: v1
kind: Service
metadata:
  name: vote-np
spec:
  selector:
    app: vote
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31000
```

service-np.yaml

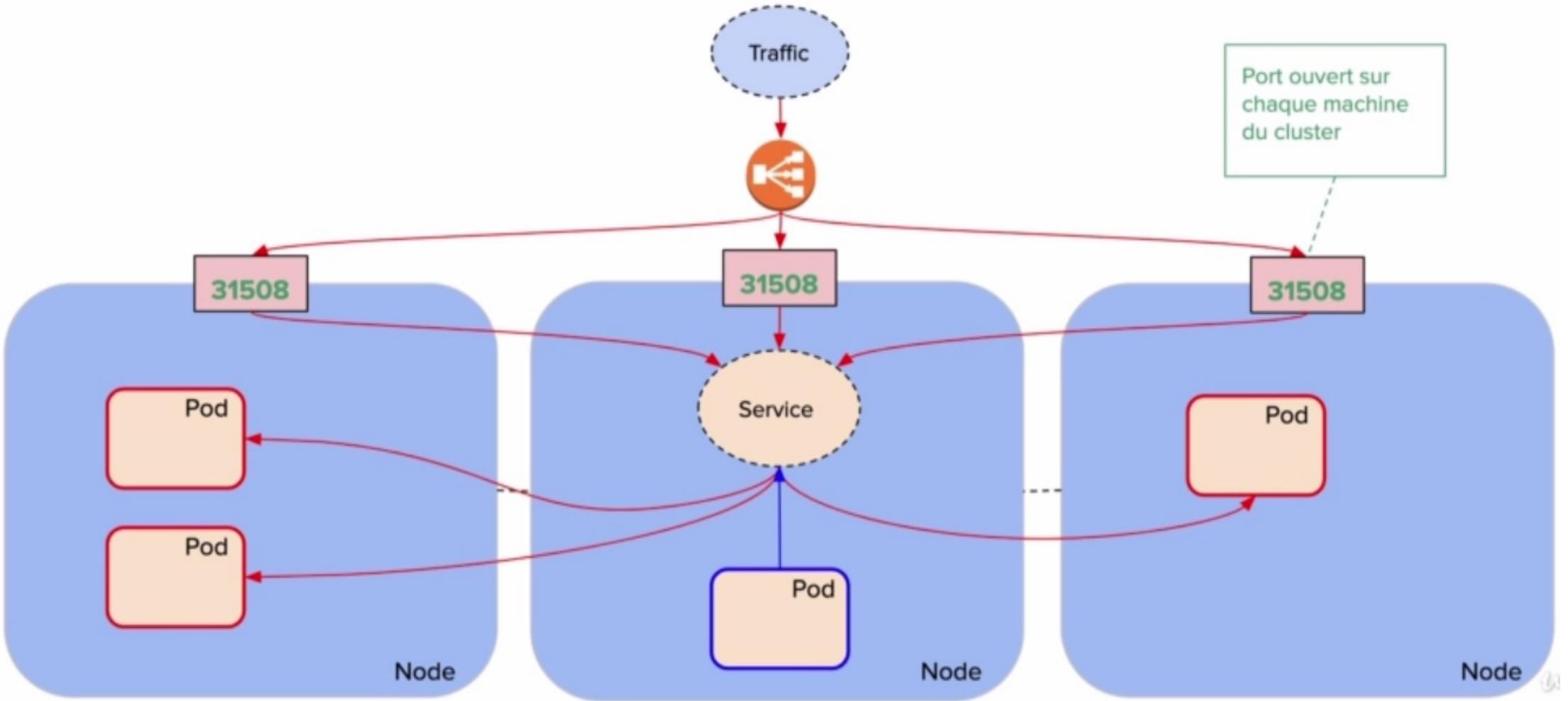


**\$ kubectl apply -f service-np.yaml**

# Service de type NodePort : accès



# Service de type LoadBalancer



# Service de type LoadBalancer : exemple

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

service-lb.yaml

## Service de type LoadBalancer : exemple

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

service-lb.yaml



**\$ kubectl apply -f service-lb.yaml**

# Service de type LoadBalancer : exemple

```
$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
vote-lb	LoadBalancer	10.245.186.209	139.59.203.88	80:31243/TCP	3m4s

DigitalOcean - Control Panel

Search by resource name or IP (Ctrl+R)

CREATE

Load Balancers

Droplets

Kubernetes

Volumes

Databases

Spaces

Images

Networking

Monitoring

API

Marketplace

ACCOUNT

Team

a94716e68b92f1a4dab0c65d43394  
in Training / LDM / Droplets / 139.59.203.88

Droplets

Graphs

Settings

3/3 Healthy Droplets

Name	Status	IP Address	Downtime	Queue	Health Checks
workers-nu10	Healthy	10.0.1.246.146	0s	0	100%
workers-nu11	Healthy	10.0.1.246.251	0s	0	100%
workers-nu12	Healthy	10.0.1.247.53	0s	0	100%

Load balancing basics

Load Balancer overview API docs Tell us what you think

Learn about DigitalOcean Load Balancers, or follow [our 2016 DevOps Report](#). Use the DigitalOcean API to create and manage Load Balancers programmatically. Submit your feedback on Load Balancers.

Création un LoadBalancer sur DigitalOcean

Cats vs Dogs!

CATS

DOGS

(Tip: you can change your vote)

Processed by container ID vote

Le traffic arrive sur le LoadBalancer

# Création avec l'approche impérative : exemple (1/3)

```
# Création d'un Pod  
$ kubectl run whoami --image containous/whoami
```

Rappel: le label `run:NOM_DU_POD` est automatiquement défini dans la spécification du Pod

```
# Exposition via un Service NodePort  
(dry run)  
$ kubectl expose pod whoami \  
  --type=NodePort \  
  --port=8080 \  
  --target-port=80 \  
  --dry-run=client \  
  -o yaml
```

Le label du Pod est utilisé dans le selector du Service



```
apiVersion: v1  
kind: Service  
metadata:  
  labels:  
    run: whoami  
  name: whoami  
spec:  
  ports:  
  - port: 8080  
    protocol: TCP  
    targetPort: 80  
  selector:  
    run: whoami  
  type: NodePort
```

## Création avec l'approche impérative : exemple (2/3)

```
# Création d'un Service NodePort (dry run)
$ kubectl create service \
nodeport \
whoami \
--tcp 8080:80 \
--dry-run=client \
-o yaml
```

Le nom du Service est utilisé dans le selector



```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: whoami
    name: whoami
spec:
  ports:
  - name: 8080-80
    port: 8080
    protocol: TCP
    targetPort: 80
  selector:
    app: whoami
  type: NodePort
```

## Création avec l'approche impérative : exemple (3/3)

```
# Création d'un Pod + Service (dry run)
$ kubectl run db \
  --image=mongo:4.2 \
  --port=27017 \
  --expose \
  --dry-run=client \
  -o yaml
```

2 ressources en une seule commande



```
apiVersion: v1
kind: Service
metadata:
  name: db
spec:
  ports:
    - port: 27017
      protocol: TCP
      targetPort: 27017
  selector:
    run: db

apiVersion: v1
kind: Pod
metadata:
  labels:
    run: db
  name: db
spec:
  containers:
    - image: mongo:4.2
      name: db
      ports:
        - containerPort: 27017
  dnsPolicy: ClusterFirst
  restartPolicy: Always
```

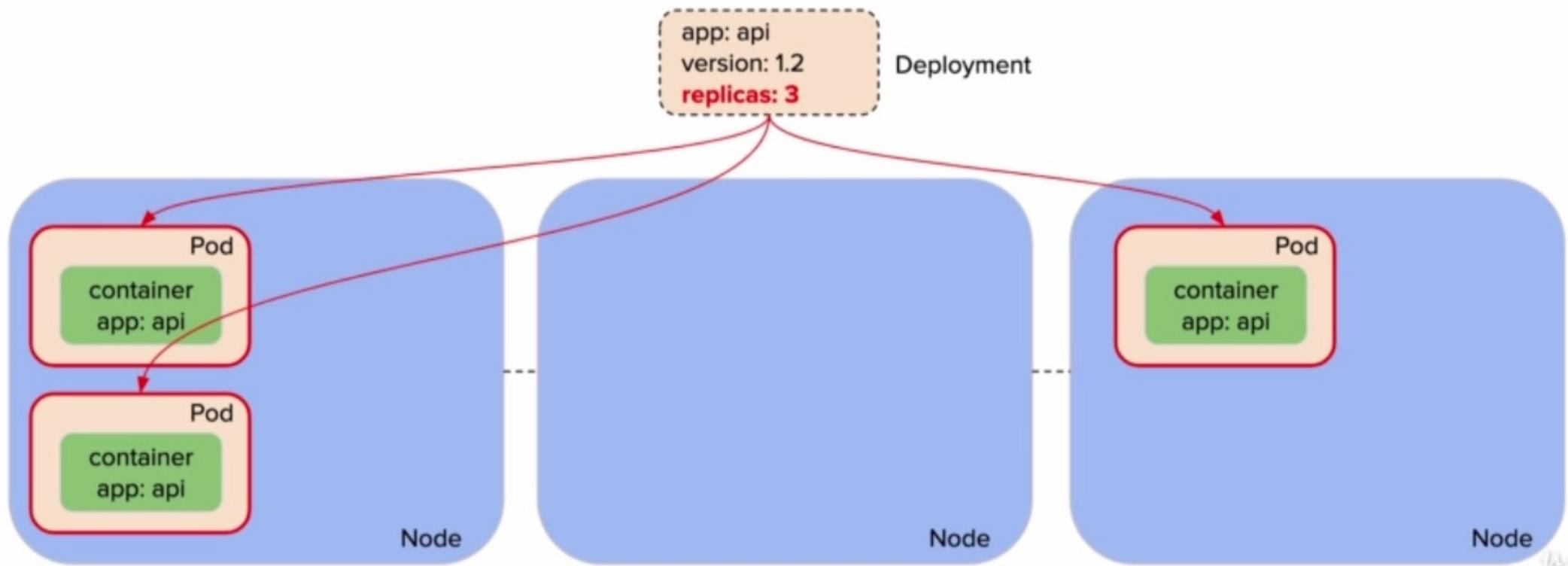
# Les commandes de base

```
# Création d'un Service  
$ kubectl apply -f PATH_SPECIFICATION  
$ kubectl expose ...  
$ kubectl create service ...  
  
# Liste de l'ensemble des services  
$ kubectl get service / kubectl get svc  
  
# Principales informations concernant un service  
$ kubectl get svc/SERVICE_NAME  
  
# Informations détaillées d'un service  
$ kubectl describe svc SERVICE_NAME  
  
# Suppression d'un service  
$ kubectl delete svc/SERVICE_NAME
```

# Les Deployments

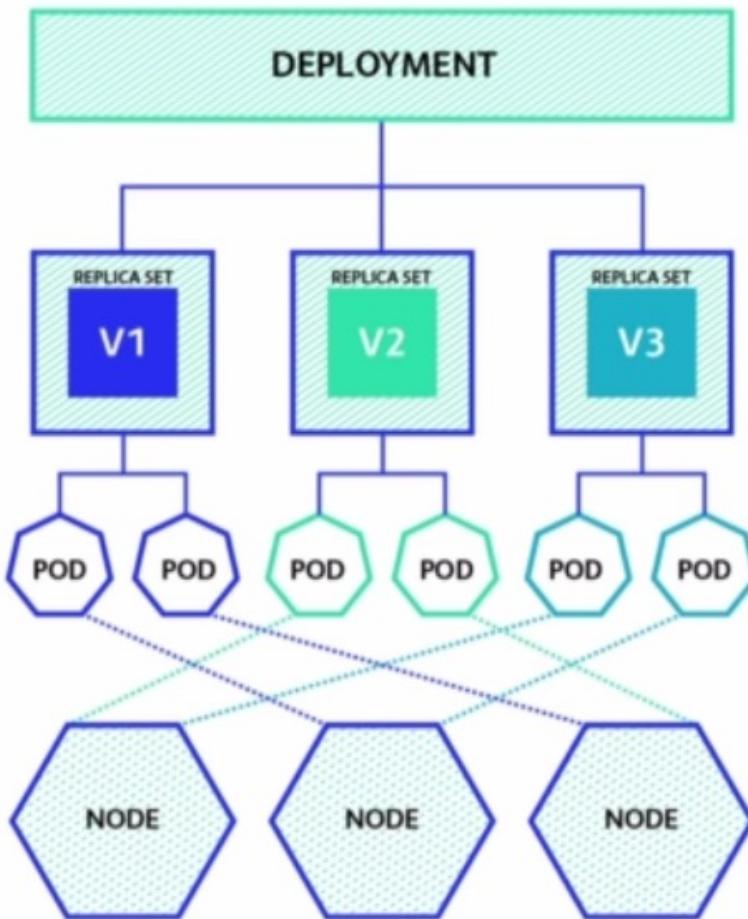
# Rôle

Un **Deployment** permet de gérer un ensemble de **Pods** identiques (mise à jour / rollback)



# Utilisation

- Gestion du cycle de vie de Pods
  - Création / Suppression
  - Scaling
  - Rollout / Rollback
- Différents niveaux d'abstraction
  - Deployment
  - ReplicaSet
  - Pod



<https://thenewstack.io>

# Spécification d'un Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      labels:
        app: vote
    spec:
      containers:
        - name: vote
          image: instavote/vote
          ports:
            - containerPort: 80
```



deploy.yaml

# Spécification d'un Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      labels:
        app: vote
    spec:
      containers:
        - name: vote
          image: instavote/vote
          ports:
            - containerPort: 80
```

deploy.yaml



```
$ kubectl apply -f deploy.yaml
```

# Listes des ressources créées par le Deployment

```
● ○ ● 4. luc@saturn: /tmp (bash)
$ kubectl get deploy,rs,pod
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/vote      3/3     3            3           61s

NAME                     DESIRED   CURRENT   READY   AGE
replicaset.apps/vote-56fb669cc6  3         3         3       61s

NAME          READY   STATUS    RESTARTS   AGE
pod/vote-56fb669cc6-gcm75  1/1     Running   0          61s
pod/vote-56fb669cc6-gprzs  1/1     Running   0          61s
pod/vote-56fb669cc6-rdbhm  1/1     Running   0          61s
$
```

## Création avec l'approche impérative

```
$ kubectl create deploy vote --image instavote/vote
```

- Plusieurs limitations ⇒ il n'est pas possible de :
  - spécifier le nombre de réplicas (1 par défaut)
  - spécifier plusieurs containers
  - ...
- Pratique mais beaucoup moins flexible qu'une spécification yaml

# Génération de la spécification d'un Deployment

```
$ kubectl create deploy vote \
  --image instavote/vote \
  --dry-run=client \
  -o yaml
```

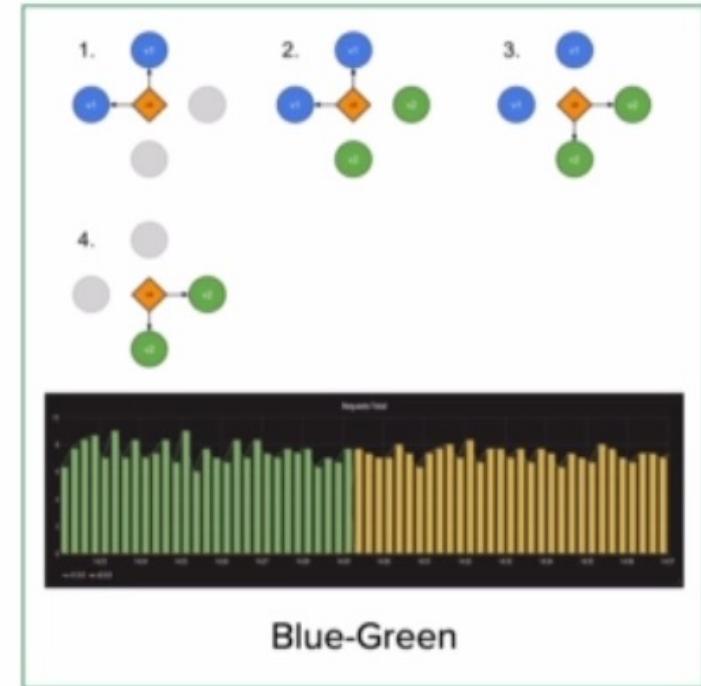
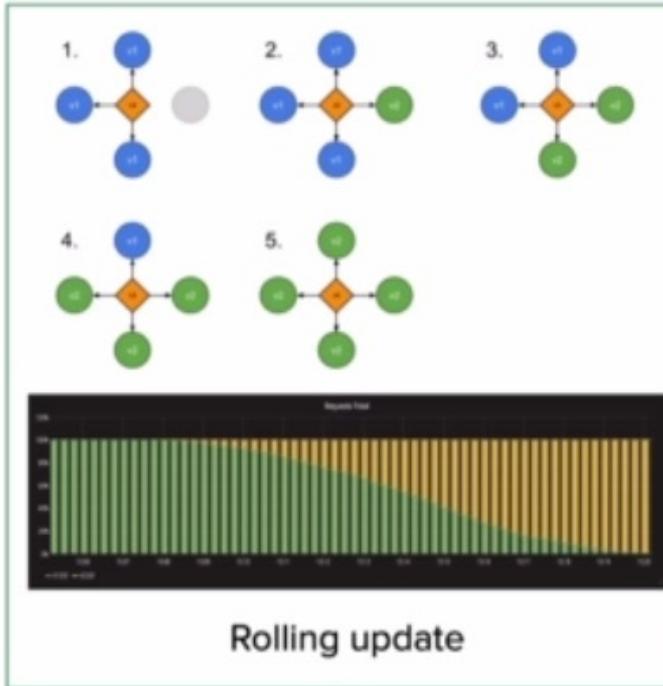
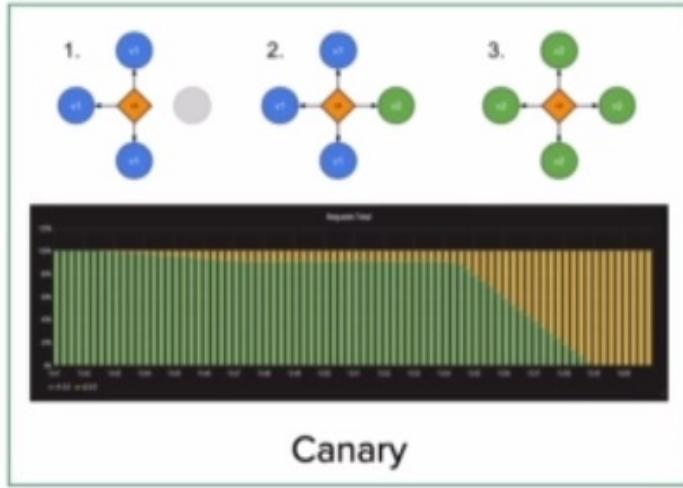


```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: vote
  name: vote
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: vote
    spec:
      containers:
        - image: instavote/vote
          name: vote
...
...
```

L'option **--dry-run** simule la création d'une ressource

- version < 1.18 ⇒ elle s'utilise sans valeur
- version >= 1.18 ⇒ 2 valeurs possibles
  - \* client: la ressource n'est pas envoyée à l'API Server
  - \* server: traitée par l'API Server mais non persistée

# Mise à jour d'une application (général)

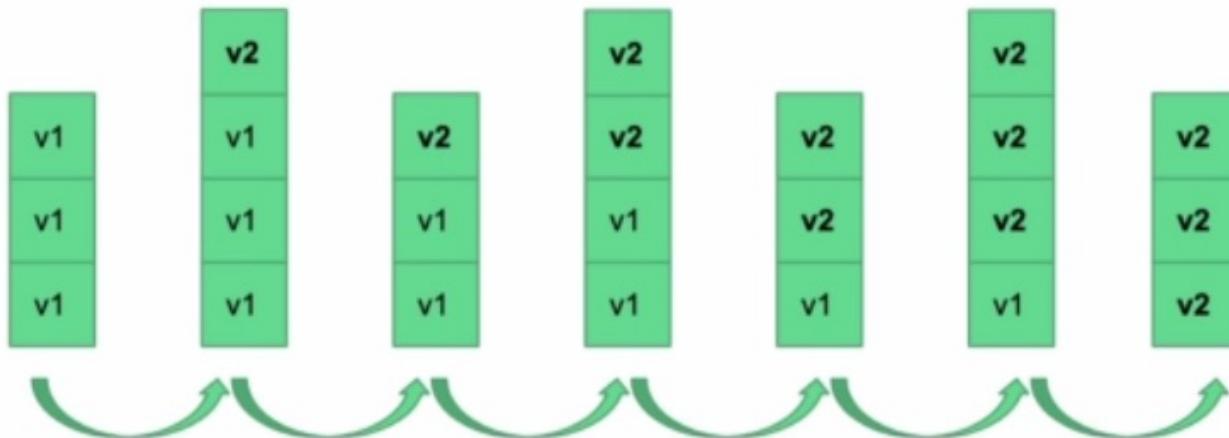


Options disponibles  
dans Kubernetes

Images <https://blog.container-solutions.co>

# Mise à jour d'un Deployment : rolling update

- Mise à jour graduelle de l'ensemble des Pods
- Paramètres pour contrôler la mise à jour
  - maxUnavailable
  - maxSurge



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: www
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    ...
```

# Mise à jour à partir de la spécification

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      labels:
        app: vote
    spec:
      containers:
        - name: vote
          image: instavote/vote:indent
          ports:
            - containerPort: 80
```

deploy.yaml



```
$ kubectl apply -f deploy.yaml
```

## Mises à jour avec l'approche impérative

```
$ kubectl set image deploy/vote vote=instavote/vote:movies --record
```

Note: le flag **--record** est à **false** par défaut, il enregistre la commande dans une annotation de la ressource

# Historique des mises à jour

```
$ kubectl rollout history deploy/vote
deployments "vote-deploy"
REVISION  CHANGE-CAUSE
1          <none>
2          <none>
3          kubectl set image deploy/vote vote=instavote/vote:movies --record=true
```

- CHANGE-CAUSE contient la commande qui a amené à la version correspondante si le flag --record a été spécifié
- 10 révisions par défaut ⇒ modifiable via la propriété ***.spec.revisionHistoryLimit*** du Deployment

# Rollback

Retour vers la révision précédente ou une révision ultérieure

```
$ kubectl rollout undo deploy/vote  
$ kubectl rollout undo deploy/vote --to-revision=X
```

# Forcer la mise à jour

```
$ kubectl rollout restart deploy/www
```

```
$ kubectl get pods -w
```

NAME	READY	STATUS	RESTARTS	AGE
www-567cb66955-6b4nw	1/1	Running	0	73s
www-567cb66955-rznp7	1/1	Running	0	69s
www-5588bfccff-d79g8	0/1	Pending	0	0s
www-5588bfccff-d79g8	0/1	ContainerCreating	0	0s
www-5588bfccff-d79g8	1/1	Running	0	2s
www-567cb66955-rznp7	1/1	Terminating	0	73s
www-5588bfccff-fmrh8	0/1	Pending	0	0s
www-5588bfccff-fmrh8	0/1	ContainerCreating	0	0s
www-5588bfccff-fmrh8	1/1	Running	0	2s
www-567cb66955-6b4nw	1/1	Terminating	0	79s
...				

# Scaling horizontal

Modification du nombre de Pods gérés par un Deployment / ReplicaSet / StatefulSet

```
# Création d'un Deployment  
$ kubectl create deploy www --image=nginx:1.16
```

```
# Augmentation du nombre de répliques  
$ kubectl scale deploy/www --replicas=5
```

# HorizontalPodAutoscaler (1/2)

Modification du nombre de Pods en fonction de l'utilisation du CPU

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: www
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: www
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Création depuis une spécification hpa.yaml

```
$ kubectl autoscale \
  deploy www \
  --min=2 \
  --max=10 \
  --cpu-percent=50
```

Création avec une commande impérative



```
$ kubectl apply -f hpa.yaml
```

## HorizontalPodAutoscaler (2/2)

```
$ kubectl get hpa -w
```

# Les Namespaces

# Présentation

- Scope pour les Pods, Services, Deployments, ...
- Partage d'un cluster
  - équipes / projets / clients
- 3 namespaces par défaut

```
$ kubectl get namespaces
NAME      STATUS   AGE
default   Active   83d
kube-public   Active   83d
kube-system   Active   83d
```

- Ressources créées dans le namespace **default** si non spécifié

# Création

```
# Création du namespaces development (option 1)
$ kubectl create namespace development
namespace "development" created ← En ligne de commande

# Suppression du namespace
$ kubectl delete namespace/development
namespace "development" deleted

# Création du namespace development (option 2)
$ cat development.yaml
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "development",
    "labels": {
      "name": "development"
    }
  }
}
$ kubectl create -f development.yaml
namespace "development" created ← Dans un fichier de spécification
```

# Utilisation

- Pod avec namespace spécifié dans les metadata

```
$ cat nginx-pod-dev.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: development          Ce Pod sera déployé dans le namespace nommé development
spec:
  containers:
    - name: www
      image: nginx:1.12.2
```

# 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 default
$ kubectl get po
No resources found.

# Liste des Pods dans le namespace development
$ kubectl get po --namespace=development
NAME      READY      STATUS      RESTARTS      AGE
nginx    1/1       Running     0            17s

# Liste des Pods dans l'ensemble des namespaces
$ kubectl get po --all-namespaces
NAME      READY      STATUS      RESTARTS      AGE
nginx    1/1       Running     0            17s
...
```

# Utilisation

```
# Création d'un Deployment dans le namespace development
$ kubectl run www --namespace development --replicas 2 --image nginx:1.12.2
```

```
# Liste des Deployments dans le namespace development
```

```
$ kubectl get deploy --namespace development
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
www        2          2          2           2           20s
```

```
# Liste des Pods dans le namespace development
```

```
$ kubectl get po --namespace development
NAME                  READY   STATUS    RESTARTS   AGE
www-6b5dfc4699-8zk92  1/1     Running   0          30s
www-6b5dfc4699-xj5cg  1/1     Running   0          30s
```

# Définition dans le context

```
$ kubectl config view
apiVersion: v1
kind: Config
clusters:
- cluster:
  certificate-authority: /Users/luc/.minikube/ca.crt
  server: https://192.168.99.100:8443
  name: minikube
users:
- name: minikube
  user:
    client-certificate: /Users/luc/.minikube/client.crt
    client-key: /Users/luc/.minikube/client.key
contexts:
- context:
  cluster: minikube
  user: minikube
  name: minikube
current-context: minikube
preferences: {}
```

← Le namespace **default** est utilisé

# Définition dans le context

```
# Vérification du context courant
$ kubectl config current-context
minikube

# Définition du namespace development dans le context courant
$ kubectl config set-context $(kubectl config current-context) --namespace=development
Context "minikube" modified.

# Vérification du changement
$ kubectl config view
...
contexts:
- context:
    cluster: minikube
    namespace: development
    user: minikube
    name: minikube
current-context: minikube
preferences: {}
```

*development* est le namespace utilisé  
dans le context minikube

# Définition dans le context

```
# Création d'un Deployment dans le context modifié  
$ kubectl run w3 nginx:1.12.2
```

```
# Liste des Deployments
```

```
$ kubectl get deploy
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
w3	1	1	1	1	28s
www	2	2	2	2	1d

```
$ kubectl get deploy --namespace development
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
w3	1	1	1	1	37s
www	2	2	2	2	1d

```
$ kubectl get deploy --namespace default
```

```
No resources found.
```

Deployment créé  
dans le namespace  
*development* et non  
*default*