

Introduction à Kubernetes

Au sommaire

- 1 Des concepts utiles**
- 2 La plateforme**
- 3 Exemples d'utilisation**
- 4 Les objets de Kubernetes**
- 5 Application satetful**
- 6 Utilisateurs et droits d'accès**
- 7 Helm**

Des concepts utiles

Containers

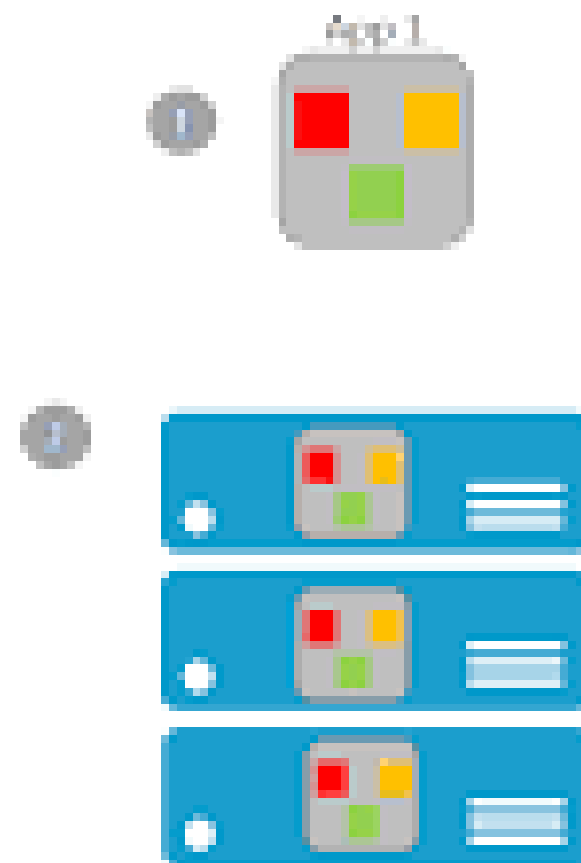
- C'est un processus qui utilise les primitives du noyau linux
- Visible sur la machine hôte
- Avec une vision limitée du système
- Avec une limite dans les ressources qu'il peut utiliser
- Combinaison de 2 primitives Linux
 - Namespace
 - Control groups (ou cgroups)

La plateforme Docker

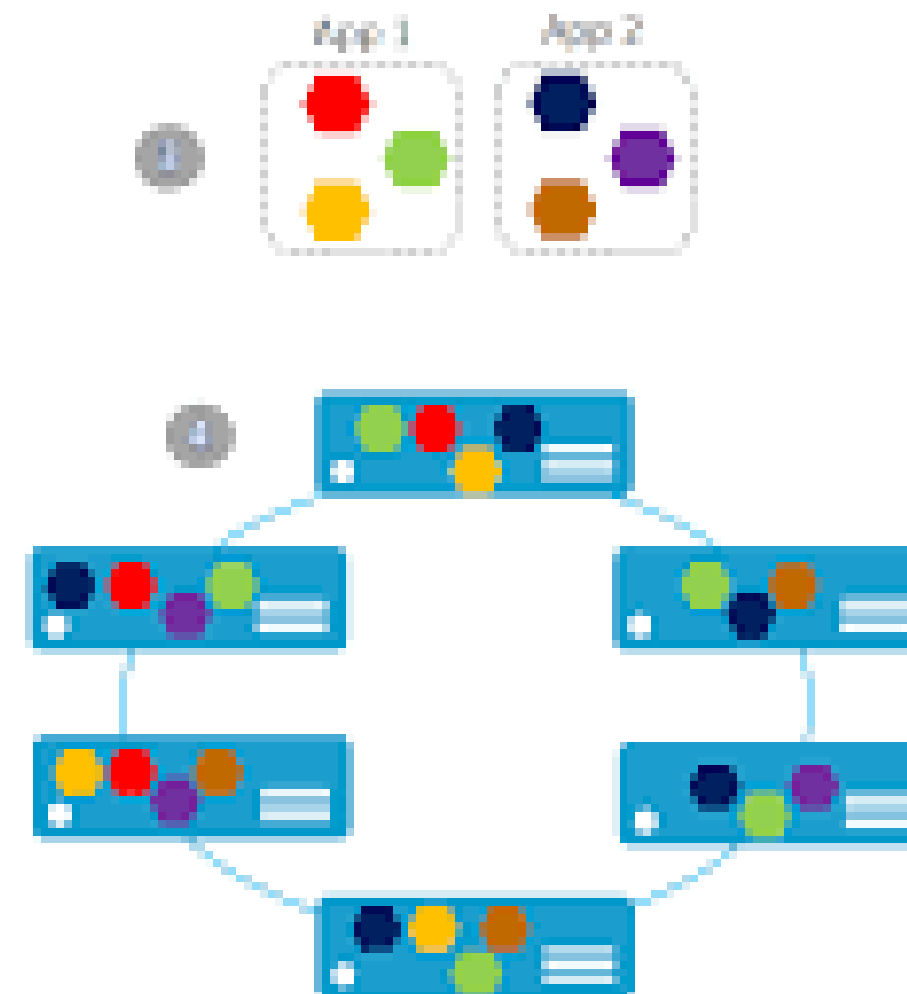
- Facilite l'utilisation des containers
- Apporte le concept d'image
 - Packaging d'une application et de ses dépendances
 - Instanciée dans un container
 - Déploiement sur une multitude d'environnement.
- Nombreuses images disponibles sur le Docker Hub :
<https://hub.docker.com>

Architecture monolithique vs micro-services

Monolithic application approach



Microservices application approach



Avantages et inconvénients des micro-services

Avantages

- Découpage de l'application en multiples services
- Processus indépendant ayant sa propre responsabilité métier
- Plus grande liberté de choix dans le langage
- Equipe dédiée pour chaque service
- Un service peut être mis à jour indépendamment des autres services
- Containers très adaptés pour les micro services

Inconvénients

- Nécessite des interfaces bien définies
- Déplace la complexité dans l'orchestration de l'application globale.

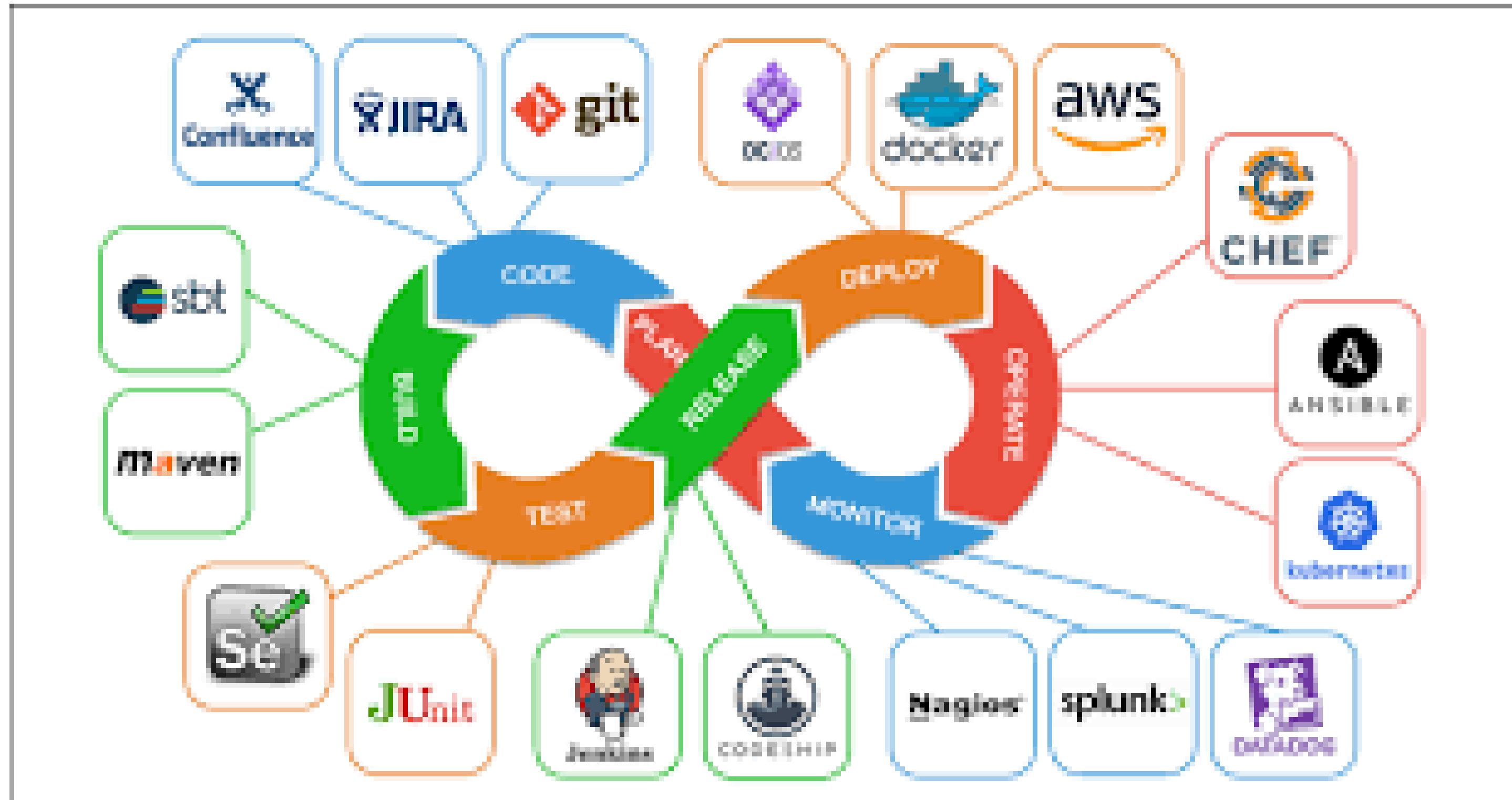
Application Cloud Native

- Application orientée micro-services
- Packagée dans des containers
- Orchestration dynamique
- Nombreux projets portés par la CNCF (Cloud Native Computing Foundation)
 - Kubernetes
 - Prometheus
 - Fluentd
 - ...
- cncf.io

Devops

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

Devops



Le projet

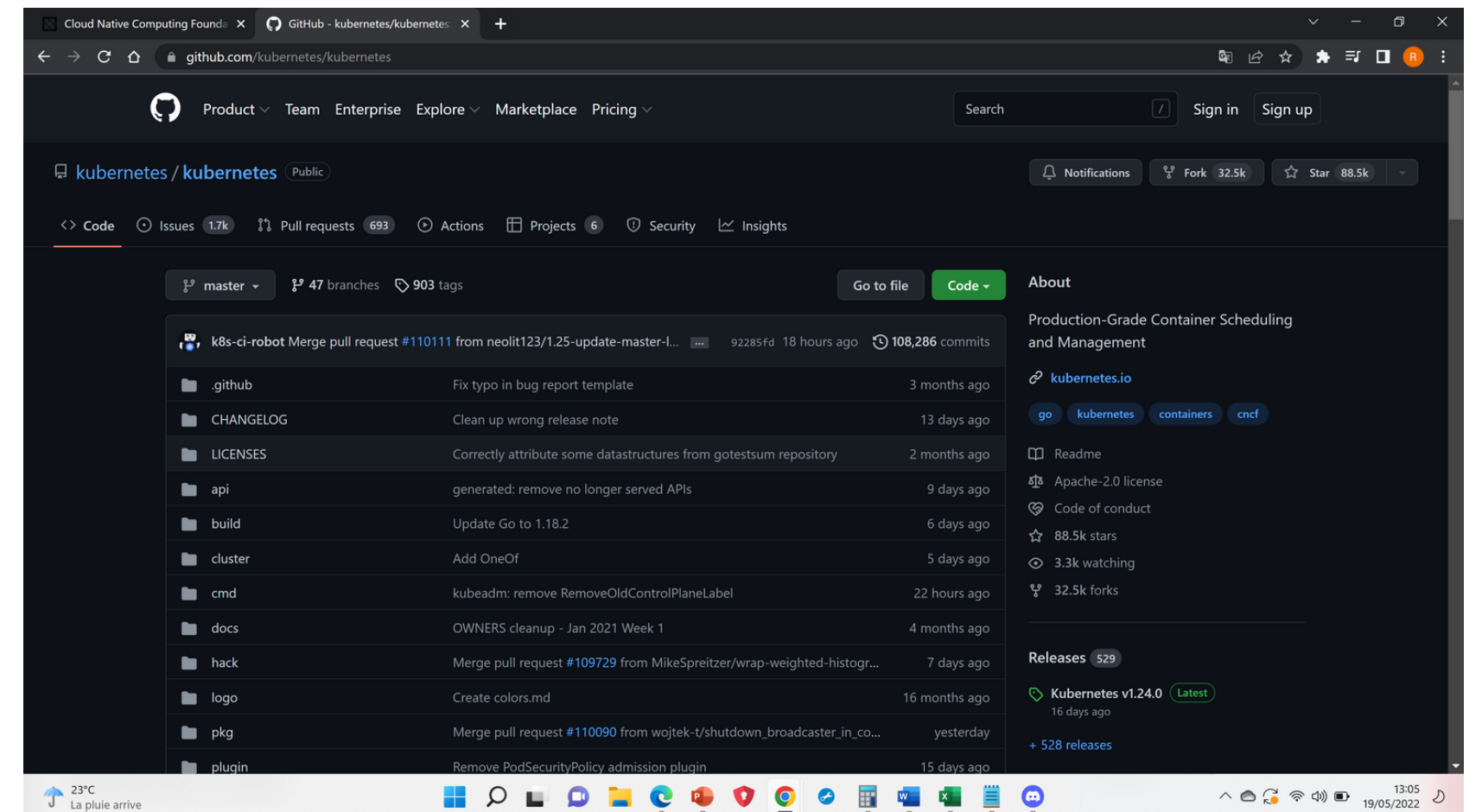
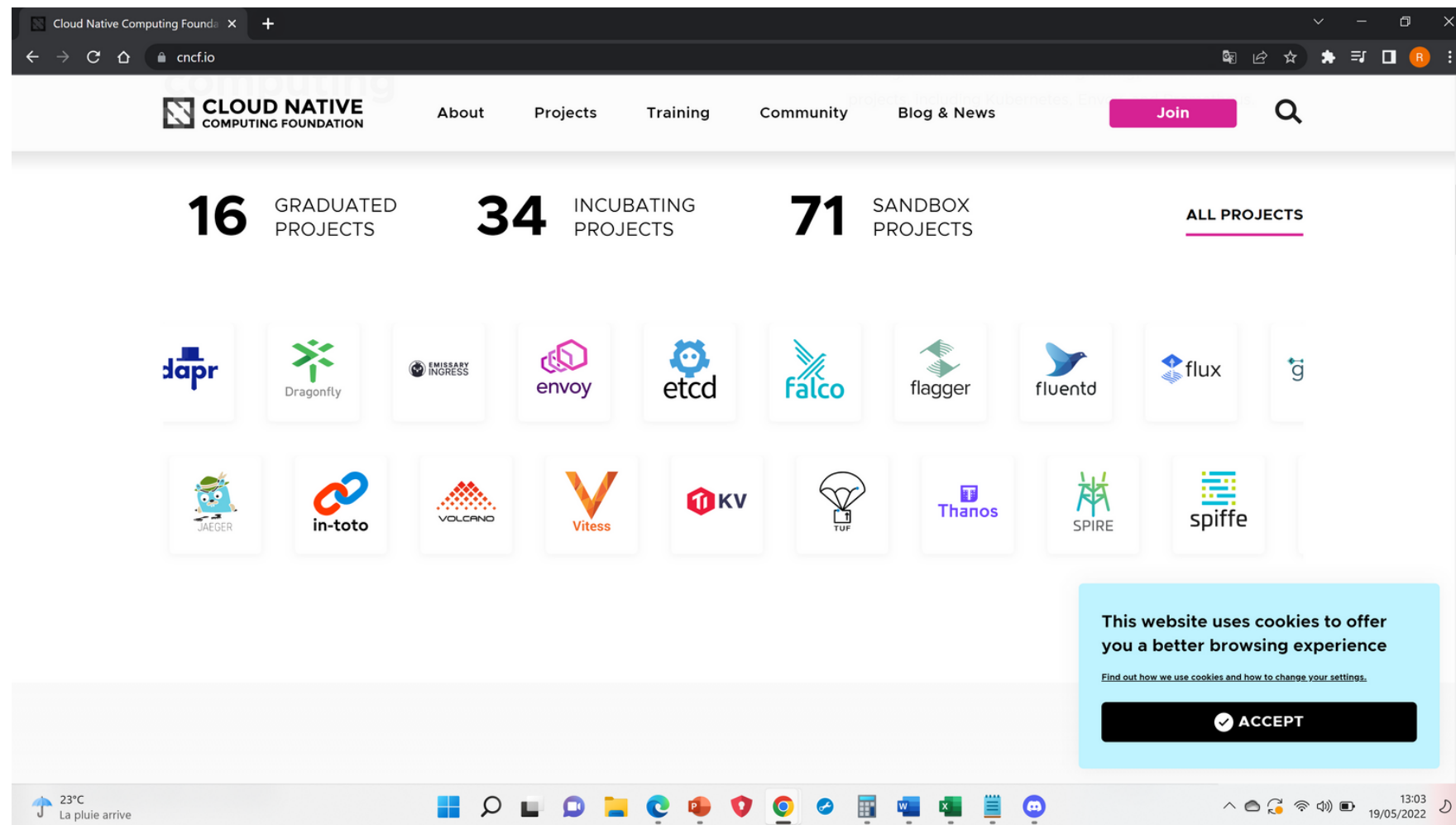
Historique

- 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 (4 versions mineure par an)

Fonctionnement

- Gestion d'applications tournant dans des containers
 - déploiement
 - scaling (montée en charge)
 - self-healing (mise à jour)
- Boucles de réconciliation vers l'état souhaité (contrôleurs)
- Plateforme open source d'orchestration de containers
- Application stateless et stateful
- Secrets et des Configurations
- Long-running process et batch jobs
- RBAC

Un projet phare de l'écosystème



Les concepts de base

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

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

Expose l'**API Server**

Node

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

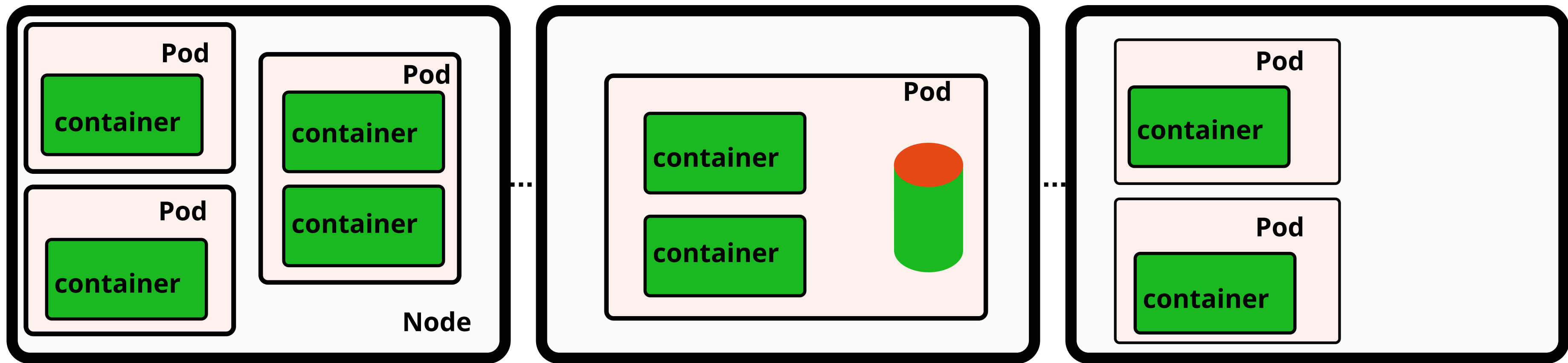
Node

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

Node

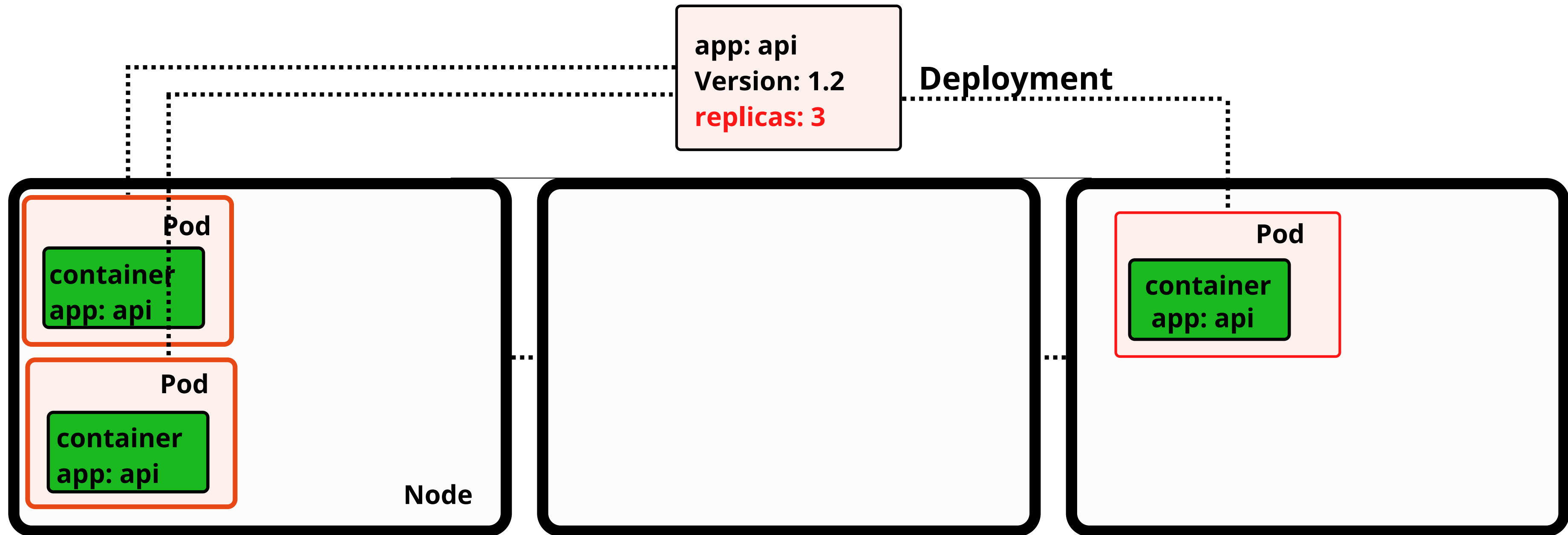
Pods

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



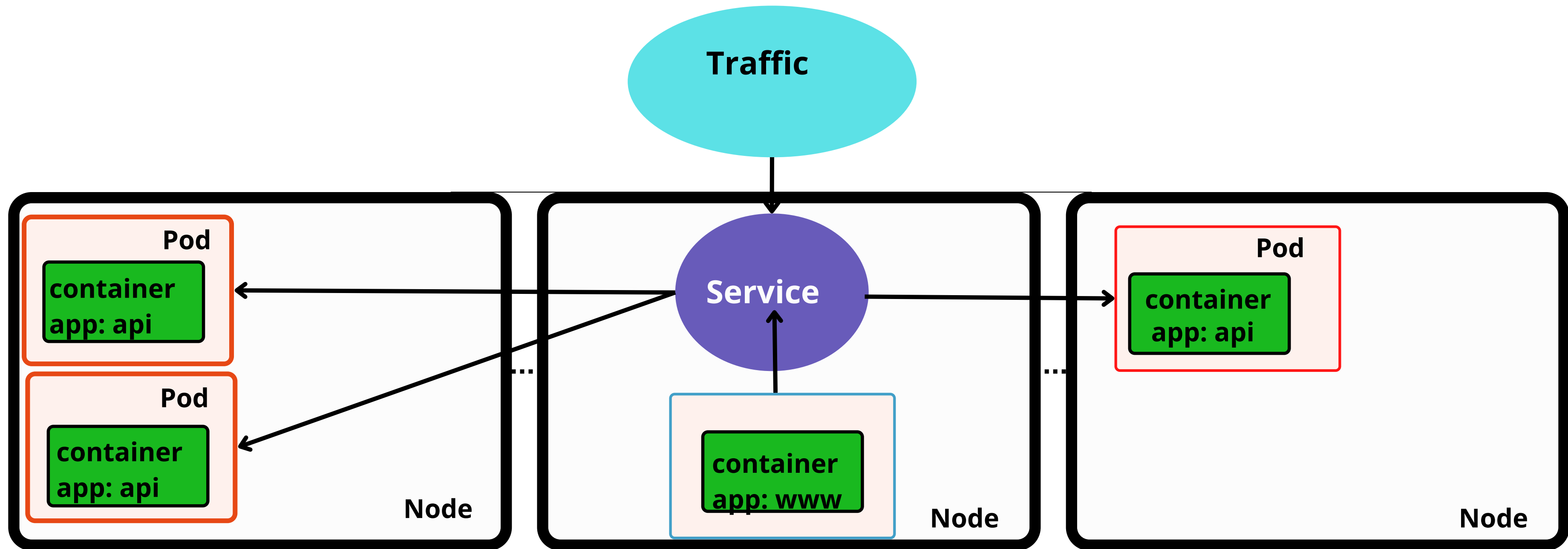
Deployment

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



Gestion du cluster

- Pour communiquer avec Kubernetes, on va utiliser le binaire kubectl qui s'installe indépendamment du cluster ou on peut utiliser une des nombreuses interfaces qui existe.
- Ces outils permettent d'envoyer des requêtes à l'API server exposé par Kubernetes pour gérer le cluster.

Katacoda

Katacoda

Gestion du cluster

- Katacoda : <https://katacoda.com>
- Plateforme d'apprentissage de technologies "Cloud native" créée par Ben Hall
- <https://katacoda.com/courses/kubernetes/playground>

"

Architecture

Gestion du cluster

kubectl get nodes

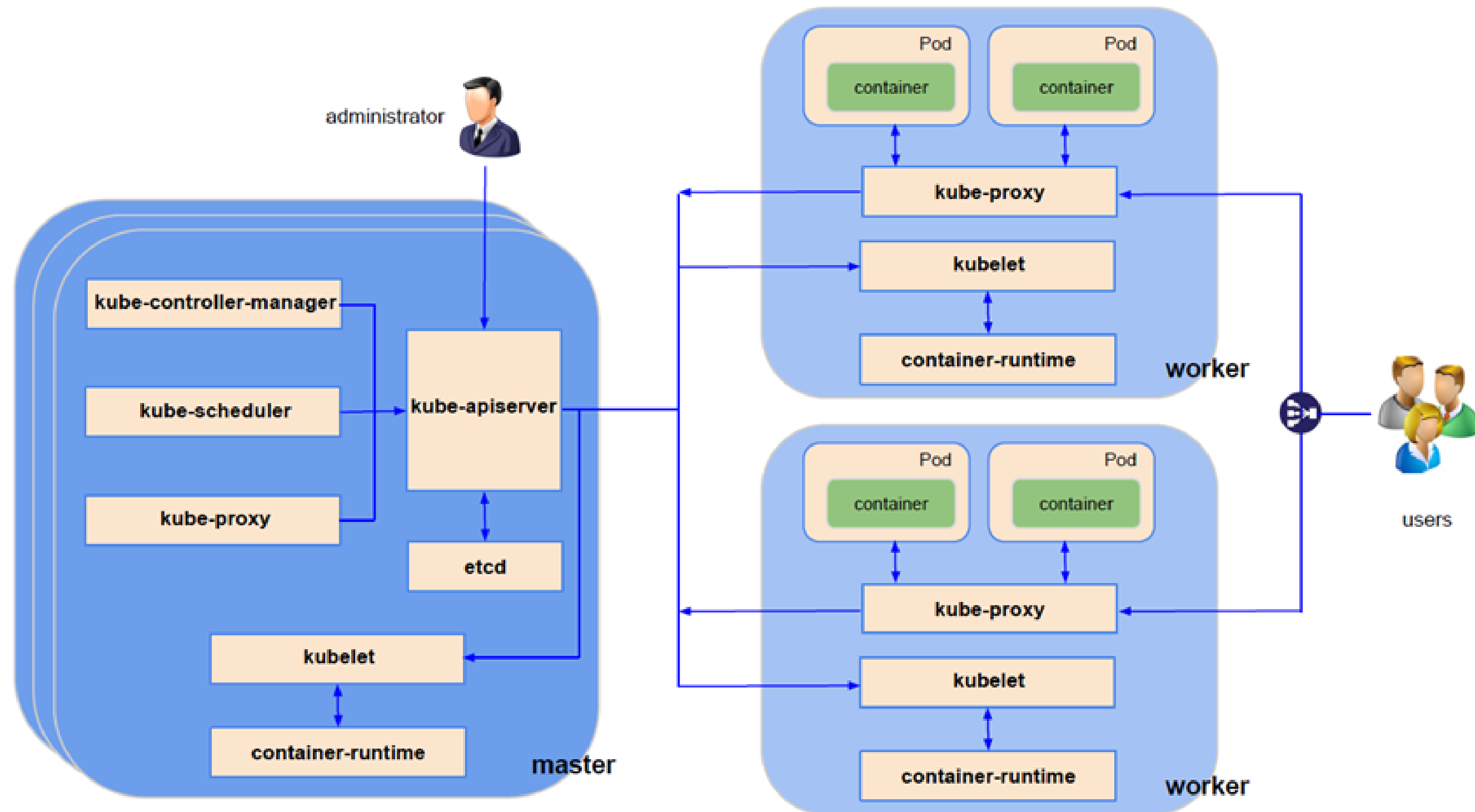
Master

- Responsable de la gestion du cluster ("control plane")
- expose l'API server
- schedule les Pods sur les nodes du cluster

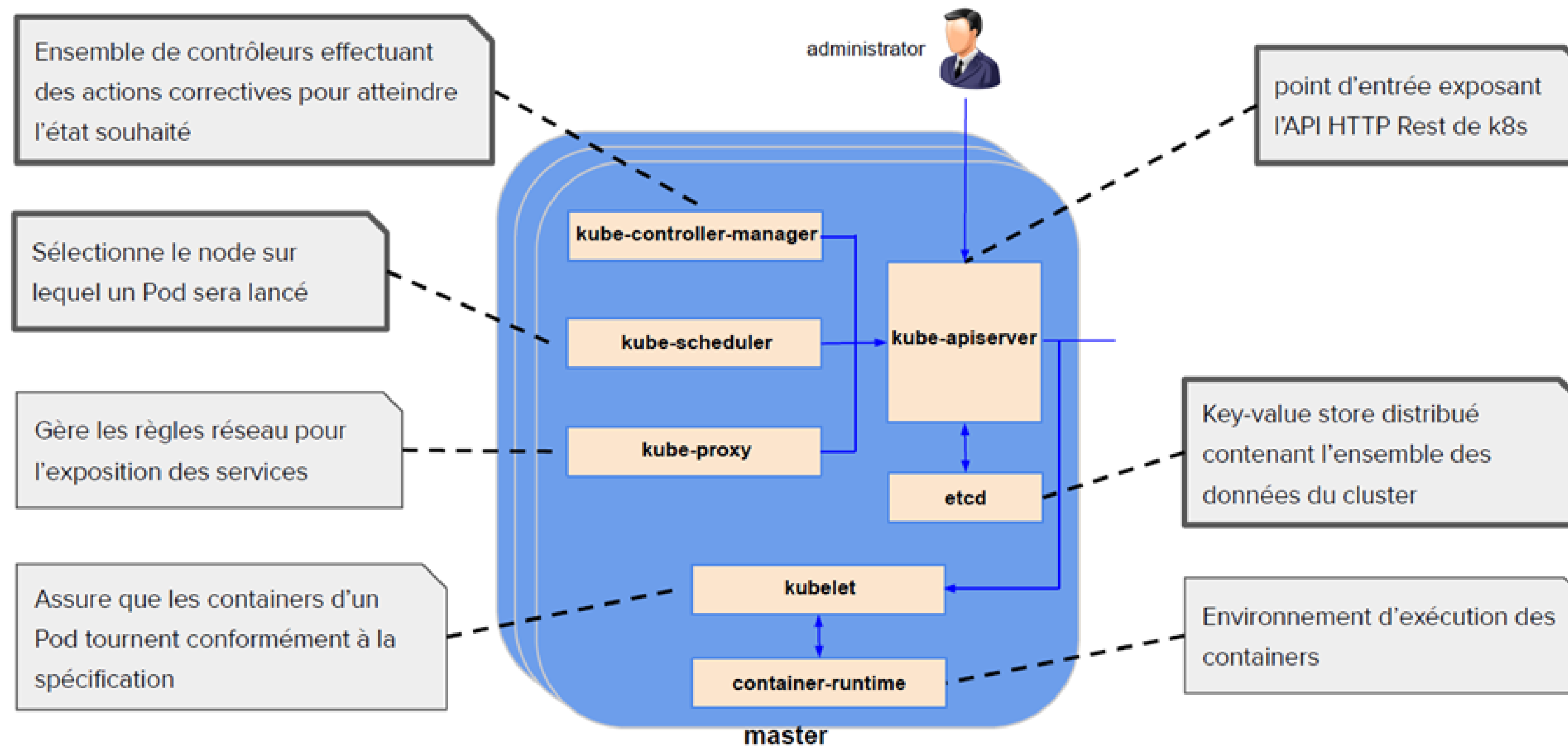
Worker / Node

- node sur lequel sont lancé les Pods applicatifs
- Communique avec le Master
- fournit les ressources aux Pods

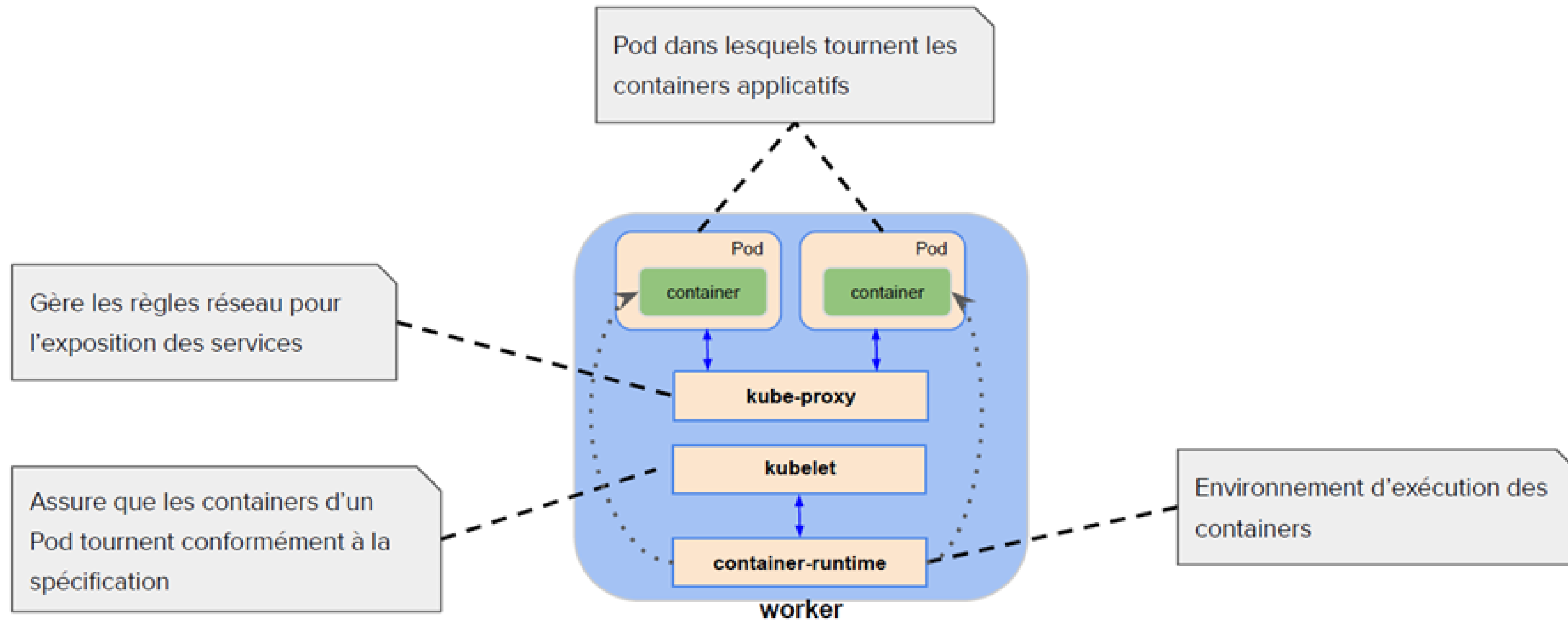
Les processus : vision d'ensemble



Les processus : côté Master



Les processus : côté Worker



Section 3:

Cluster de développement

**Lancer un cluster
Kubernetes en local:
Différentes solutions**

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 le container Docker
- Cluster HA via un fichier de configuration

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

K3S

- Kubernetes léger: la distribution Kubernetes certifiée conçue pour l'IoT et l'Edge computing"

```
$ curl -sfL https://get.k3s.io | sh -
```

Multipass

Multipass: un outil pour le provisionning de VMs

- <https://multipass.run>
- Provisionning de machines virtuelles Ubuntu sur différents hyperviseurs
 - Hyper-V
 - HyperKit
 - Virtualbox
 - KVM
- Très utile pour la mise en place de cluster local

Section 4:

Cluster de production

Cluster Kubernetes managé

- GKE - Google Kubernetes Engine
- AKS - Azure Container Service
- EKS - Amazon Elastic Container Service
- DigitalOcean
- OVH
- <https://kubernetes.io/docs/setup/pick-right-solution/#hosted-solutions>

Cluster Kubernetes managé

- kubeadm - <https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm/>
- kops - <https://github.com/kubernetes/kops>
- kubespray - <https://github.com/kubernetes-sigs/kubespray>
- Rancher - <https://rancher.com/>
- Pharos - <https://www.kontena.io/pharos>
- Docker EE (deploy Swarm et kubernetes)
- Terraform + Ansible

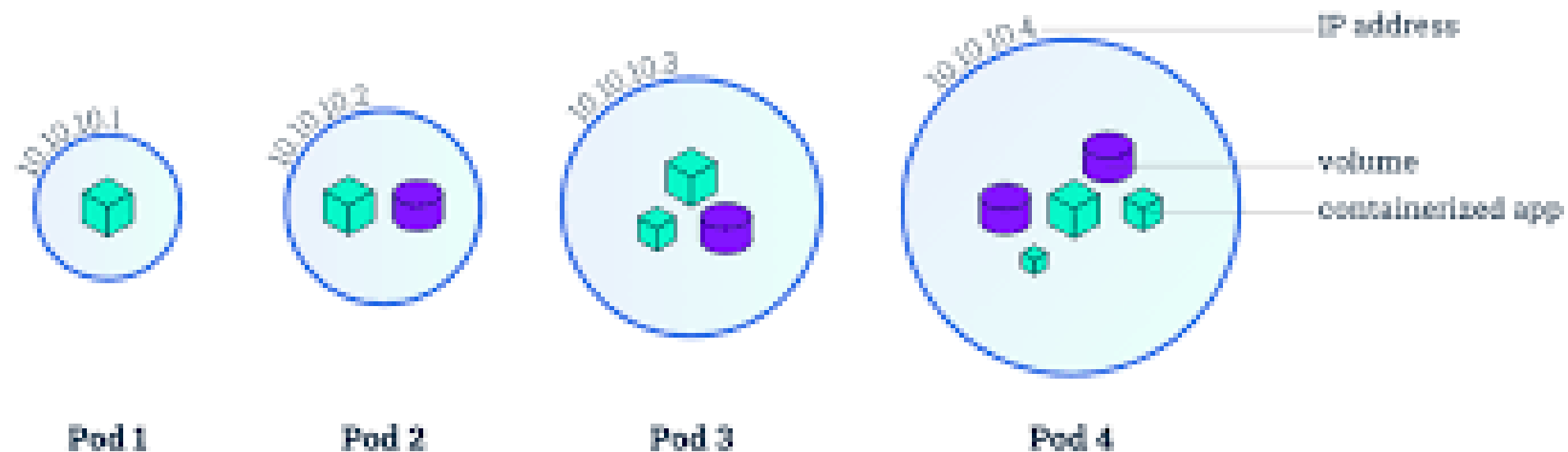
Section 5:

Les objets : Pod

Présentation

Présentation

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



Présentation

- Un pod contient un ensemble de containers
- Une application est découpée en 1 ou plusieurs Pods.
- 1 Pod correspond à un service technique ou métier d'une application
- Adresse IP dédiée, pas de NAT pour la communication entre les Pods
- Scaling horizontal via le nombre de replica d'un Pod
 - création de nouveaux Pod bas sur la même spécification

Exemple de spécification - server http

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

Exemple: Fichier nginx-pod.yaml

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

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

Version de l'API pour la gestion des POD. L'objet Pod est stable et disponible depuis la v1 de l'API

Spécification du type d'objet, ici nous définissons un Pod

Ajout du nom du Pod

Ajout de la spécification du Pod: spécification des containers lancés dans le Pod (un seul ici)

De nombreux paramètres possibles

Cycle de vie

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`
 - namespace "default"
- **Supression 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

Lancement d'une commande dans un Pod

\$ kubectl exec www -- nginx -v

Shell interactif dans un Pod

\$ kubectl exec -t -i www -- /bin/bash

Cycle de vie

Détails du Pod

\$ kubectl describe po/www

#Suppression du Pod

\$ kubectl delete pod www

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:cCONTAINER_PORT`
 - **`$ kubectl port-forward www 8080:80`**

**Pod avec plusieurs
containers**

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 persistance 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

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 persistance 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: {}

Container pour le moteur wordpress

Container pour la base de données mysql

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
```

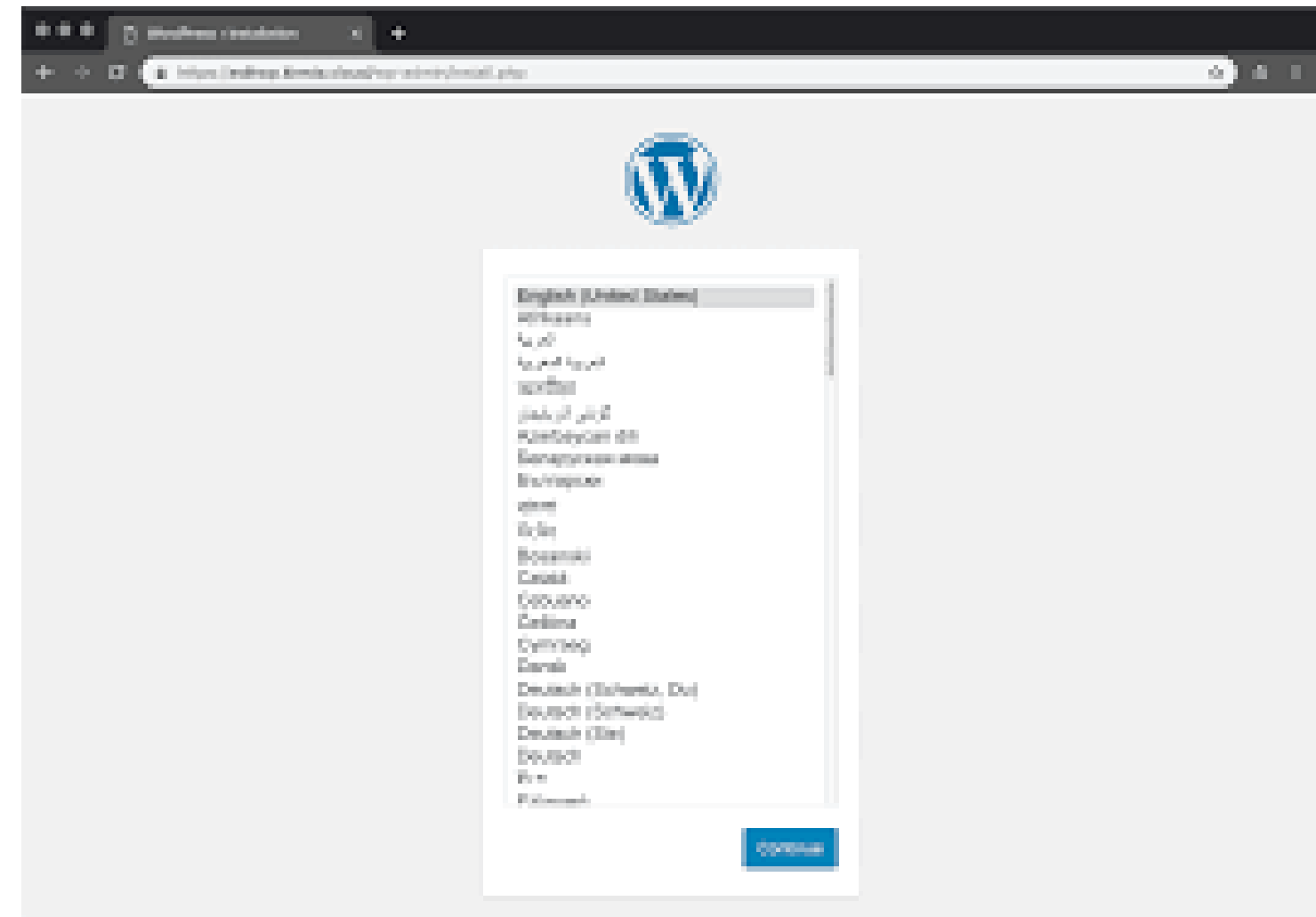
NAME	READY	STATUS	RESTARTS	AGE
wp	2/2	Running	0	18s

```
# 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



Éléments de l'étape de 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 scheduler 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 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
 - 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
```



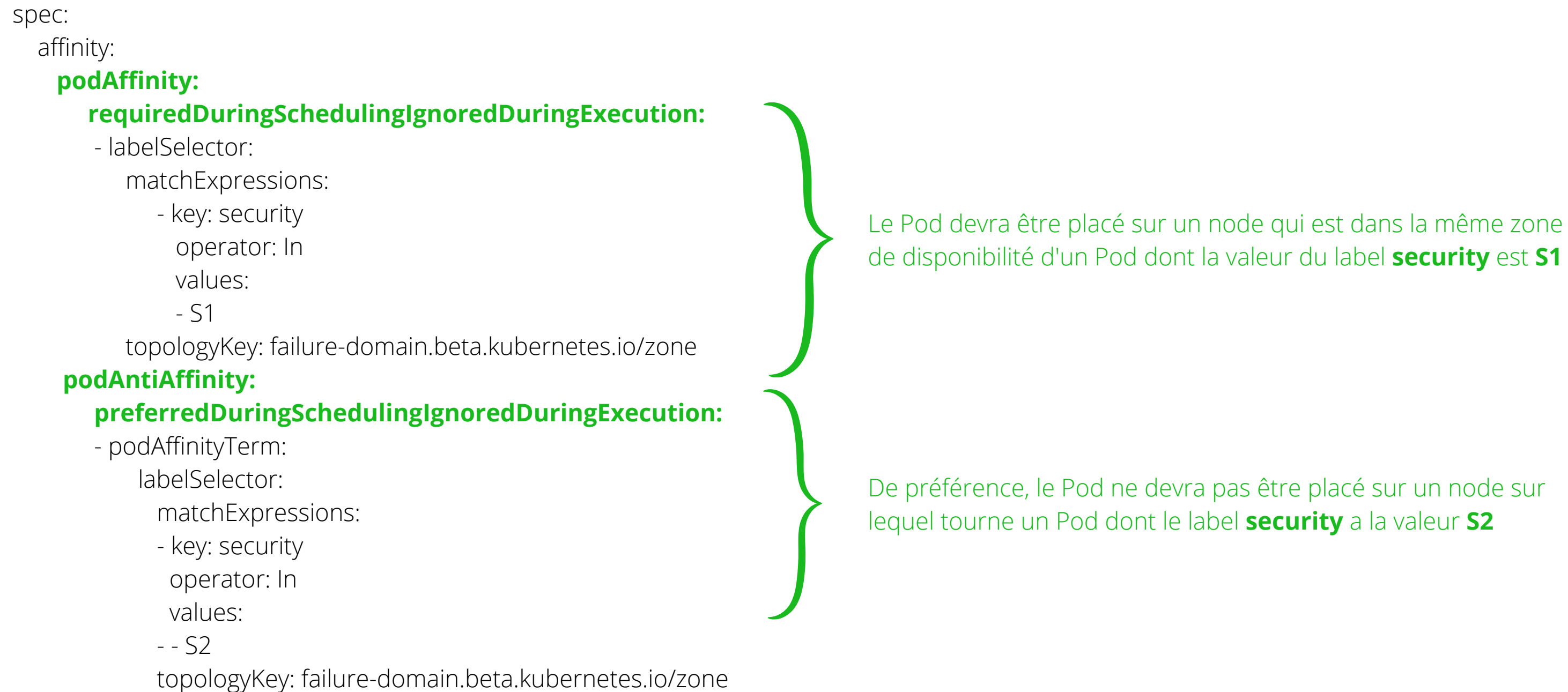
Le Pod devra être placé sur un node dont la valeur du label **kubernetes.io/e2e-az-name** est **e2e-az1** ou **e2e-az2**

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

podAffinity / podAntiAffinity (1/2)

- Permet de scheduler des Pods en fonction de labels présents sur d'autres Pods
- Différentes règles
 - requiredDuringSchedulingIgnoredDuringExecution (contrainte "hard")
 - preferredDuringSchedulingIgnoredDuringExecution (contrainte "soft ")
- Utilise le champs **topologyKey** pour la spécification de domaines topologiques
 - hostname
 - region
 - az
 - ...

nodeAffinity (2/2)



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:
  - image: db
    name: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

} **ressources minimales nécessaires**

} **ressources maximales autorisées**

Création avec l'approche impérative

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

nodeSelector

Tips & Tricks

```
$ 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

En résumé

En résumé

- Application composée de plusieurs Pods qui communiquent entre eux
- Un Pod contient souvent un seul container applicatif
 - en plus du container pause
- Instrumentation d'une application en ajoutant des containers de services
 - monitoring
 - logs
 - service mesh
- Pods généralement créés via un **ReplicaSet** dans un **Deployment**
- Exposition dans le cluster ou vers l'extérieur via un **Service**

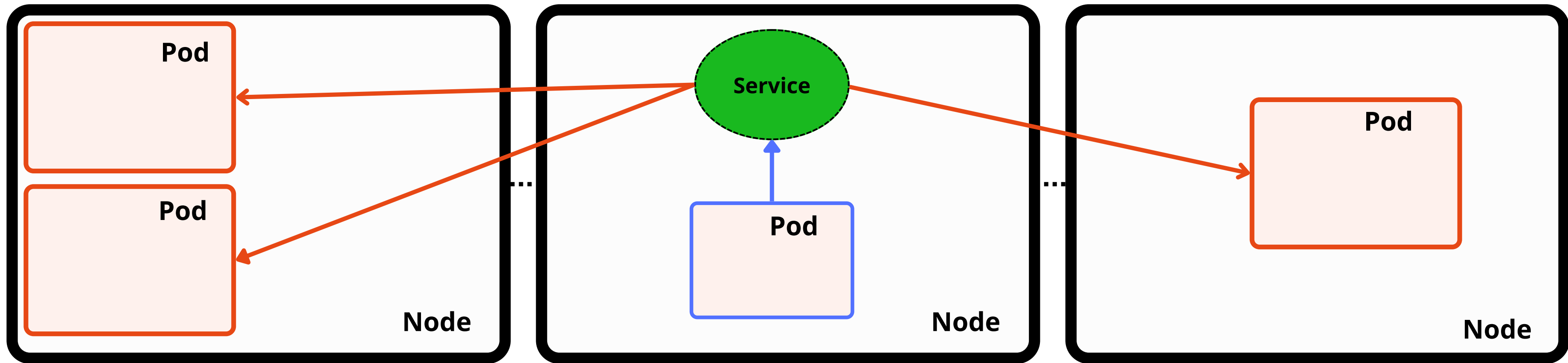
Les objets - Service

Présentation

- 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 (default): 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



Service de type ClusterIP: exemple

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

Spécification du type de l'objet

Spécification du type de l'objet

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
```

Type par défaut, expose des Pods à l'intérieur du cluster

Service.yaml

Service de type ClusterIP: exemple

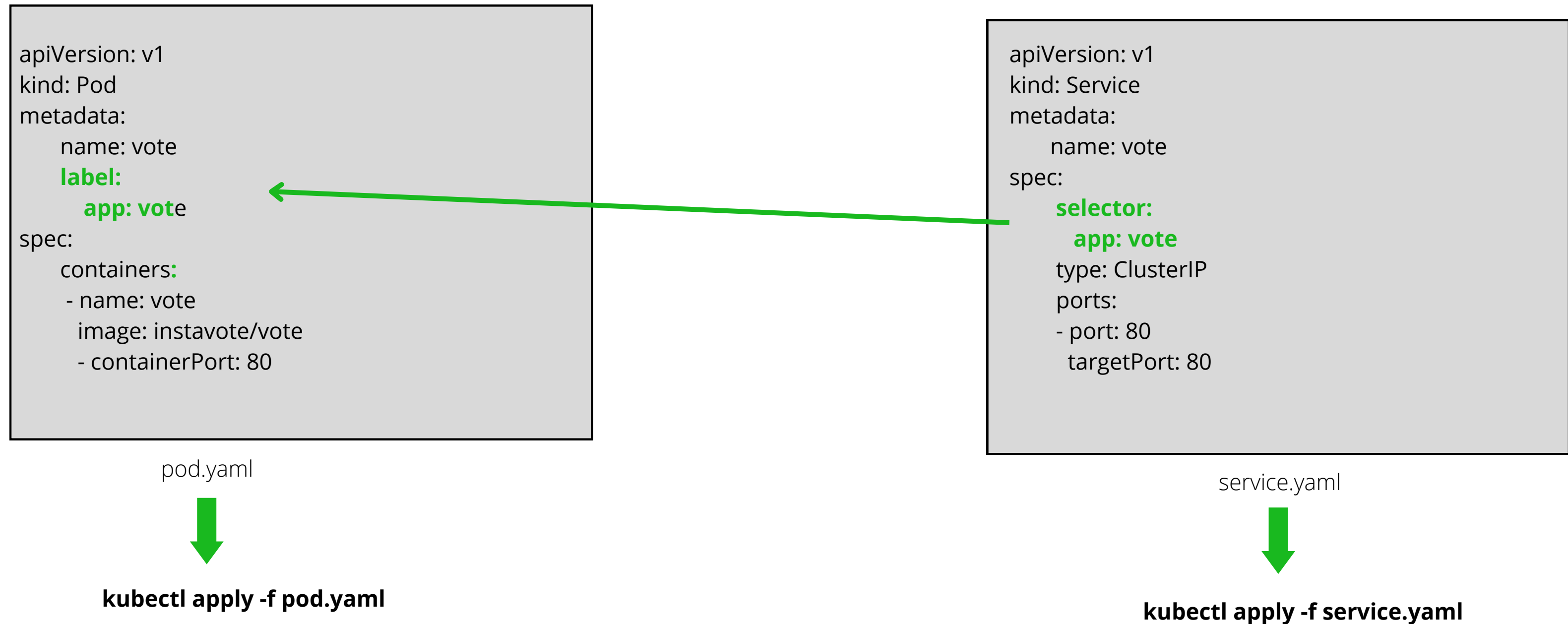
```
apiVersion: v1
kind: Service
metadata:
  name: vote
spec:
  selector:
    app: vote
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
```

Type 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



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>  
  ...
```

Service de type ClusterIP - Port Forward

Service de type ClusterIP: accès via le port-forward

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



Application

Service de type ClusterIP : accès via le proxy

Service de type ClusterIP: accès via le proxy

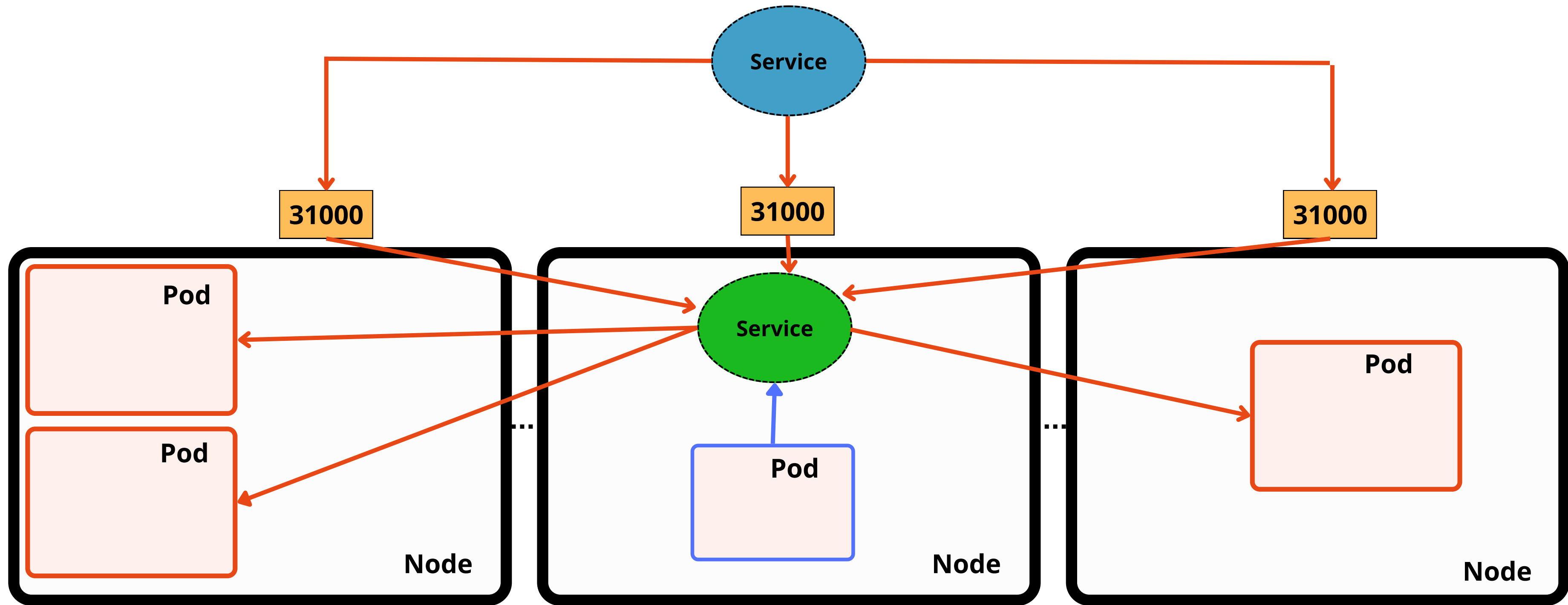
```
# Lancement du proxy (port 8001)  
$ kubectl proxy
```



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



Service de type ClusterIP: exemple

```
apiVersion: v1
kind: Service
metadata:
  name: vote
spec:
  selector:
    app: vote
  type: NodePort    Service de type NodePort, exposé sur chaque node du cluster
  ports:
    - port: 80
      targetPort: 80
      nodePort: 31000
```

Service-np.yaml

Service de type ClusterIP: exemple

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

Service expose le port 80 dand le cluster

Service-np.yaml

Service de type ClusterIP: exemple

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

Requête envoyées sur le port 80 d'un Pods du groupe

Service-np.yaml

Service de type ClusterIP: exemple

```
apiVersion: v1
kind: Service
metadata:
  name: vote
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 ClusterIP: exemple

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

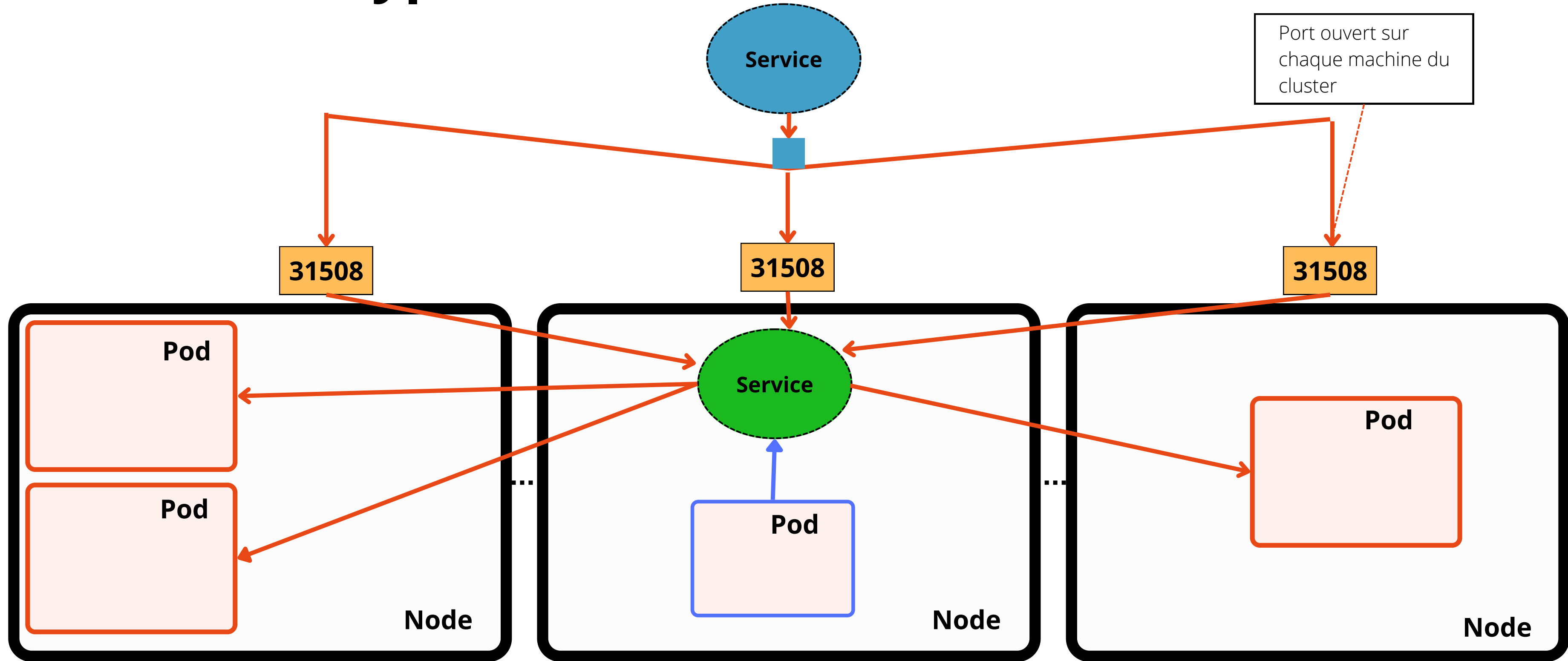
Service-np.yaml



```
$ kubectl apply -f service-np.yaml
```

Service de type LoadBalancer

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 ClusterIP: accès via le proxy

\$ kubectl get svc					
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORTS(S)	AGE
vote-lb	LoadBalancer	10.245.186.209	139.59.203.88	80:31243/TCP	3m4s

Création de Services avec les commandes impératives

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 \  

```



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

Le label du Pod est utilisé dans le selector du Service

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

#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

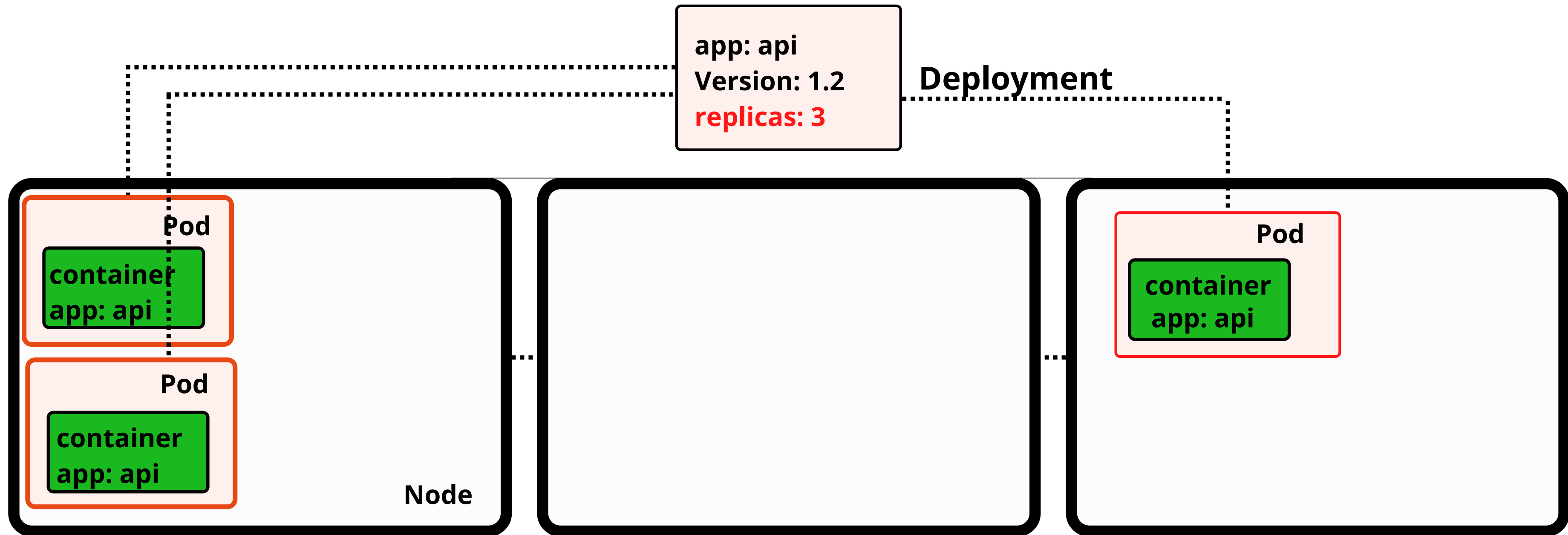
\$ kubectl delete svc/SERVICE_NAME

Les objets : deployment

Rôle

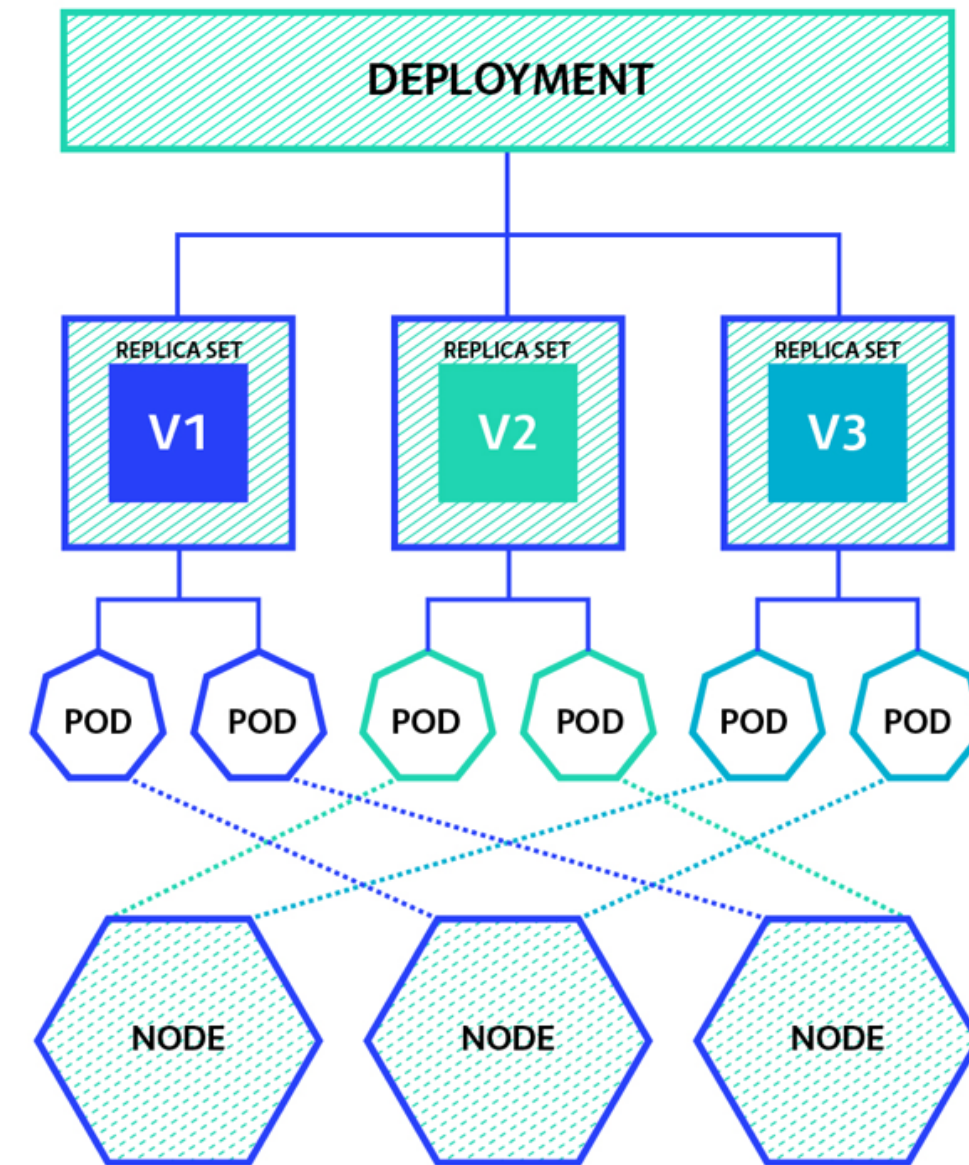
Rôle

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



Spécification d'un Deployment

Spécification d'un Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote
spec:
```

Deployment

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

ReplicaSet

```
  template:
    metadata:
      labels:
        app: vote
    spec:
      containers:
        - name: vote
          image: instavote/vote
          ports:
            - containerPort: 80
```

Pod

deploy.yaml

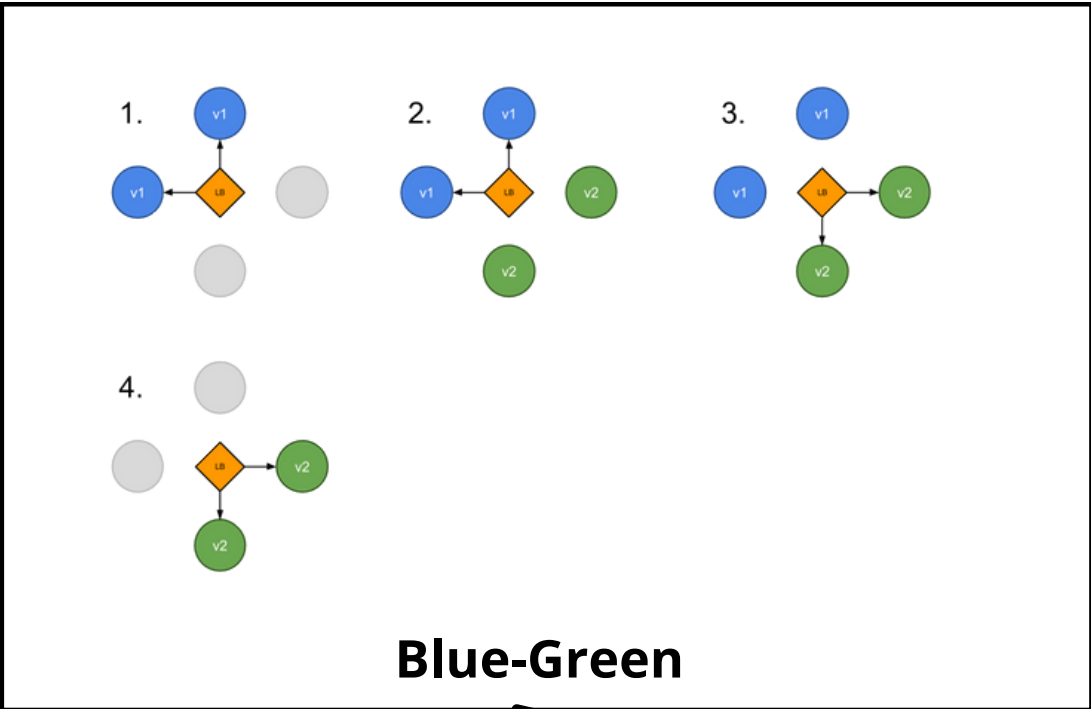
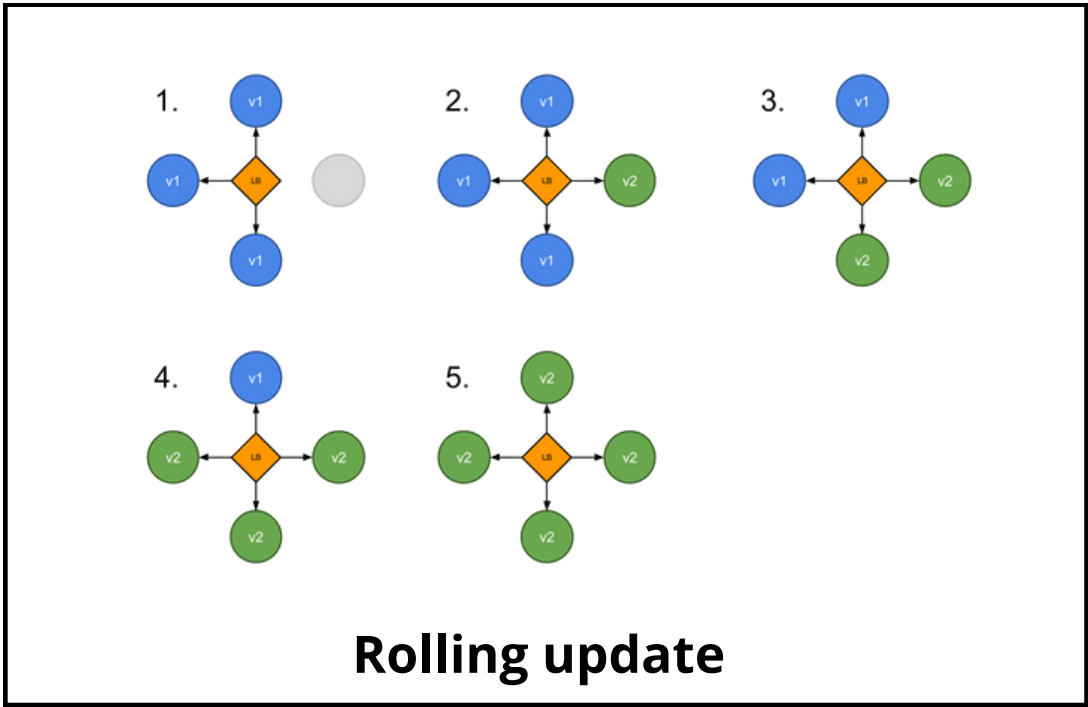
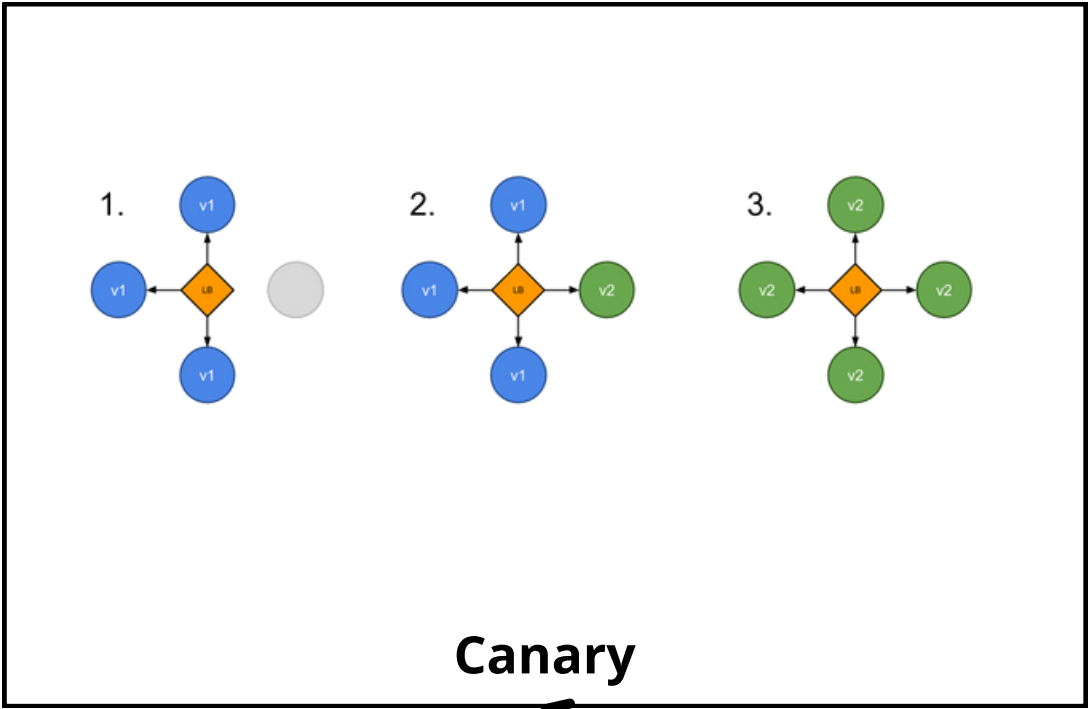
Création avec l'approche impérative

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

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

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

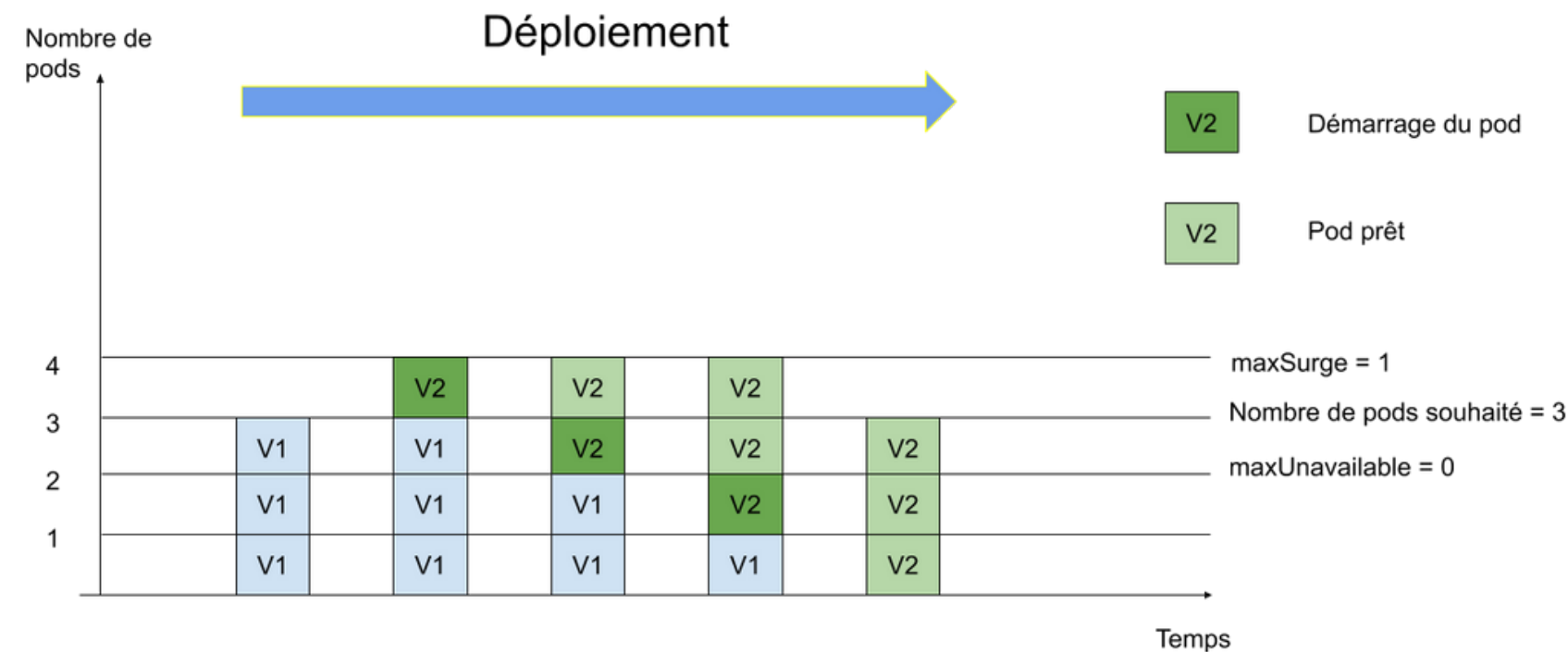
Mise à jour d'un deployment (général)



Options disponibles
dans kubernetes

Mise à jour d'un Deployment: rolling update

- Mise à jour graduelle de l'ensemble des Pods
- Paramètre 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: instavote/vote:indent
          ports:
            - containerPort: 80
```

deploy.yaml



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

Mise 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 l

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 resart deploy/www
```

Mise à l'échelle (scaling)

Scaling horizontal

Modification du nombre de Pods gérés par un Déploiement/replicaSet/StatefulSet

```
# Création d'un Deployment
```

```
$ kubectl create deploy www --image=nginx:1.16
```

```
# Augmentation du nombre de réplicas
```

```
$ kubectl scale deploy/www --replicas=5
```

Présentation de la ressource HorizontalPodAutoscaler

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 apply -f hpa.yaml

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

Création avec une commande impérative

HorizontalPodAutoscaler (1/2)

\$ kubectl get hpa -w

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
www	Deployment/www	0%/50%	2	10	2	43s
www	Deployment/www	14%/50%	2	10	2	76s
www	Deployment/www	80%/50%	2	10	2	2m15
www	Deployment/www	90%/50%	2	10	4	3m17
www	Deployment/www	94%/50%	2	10	6	4m18
...						

Les objets - Namespace

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

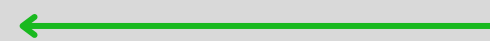
- Ressources créées dans le namespace default si non spécifié

Création

#Création du namespaces development (option 1)

\$ kubectl create namespace development

namespace "development" created



En ligne de commande

Suppression du namespace

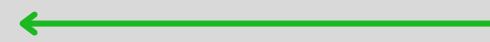
\$ kubectl delete namespace/development

namespace "development" deleted

Création du namespace development (option 2)

\$ cat development.yaml

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



Dans un fichier de spécification

\$ kubectl delete namespace/development

namespace "development" created

Utilisation

```
$ 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	RESARTS	AGE
nginx	1/1	Running	0	17s

Liste des Pods dans l'ensemble des namespace development

\$ kubectl get po --all-namespaces

NAME	READY	STATUS	RESARTS	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	17s

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: apps/v1
```

```
kind: Config
```

```
clusters:
```

```
- cluster:
```

```
  certificate-authority: /Users/radia/.minikube/ca.crt:
```

```
  server: https://192.168.99.100:8443
```

```
  name: minikube
```

```
users:
```

```
- name: minikube
```

```
  user:
```

```
    client-certificate: /Users/radia/.minikube/client.crt:
```

```
    client-key: /Users/radia/.minikube/client.key
```

```
contexts:
```

```
- context:
```

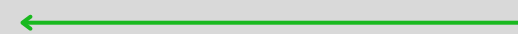
```
  cluster: minikube
```

```
  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
```

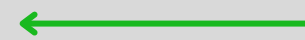
```
  namespace: development
```

```
  user: minikube
```

```
  name: minikube
```

```
current-context: minikube
```

```
preferences: {}
```



development est le namespace utilisé dans le context minikube

Utilisation

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	28s
www	2	2	2	2	1d

Deployment créé dans le namespace development et non default

\$ kubectl get deploy --namespace default

No resources found.

...

Les objets - ConfigMap

Présentation

- Découplage d'une application et de sa configuration et de sa configuration
 - pas de configuration dans le code applicatif !
- Assure la portabilité
- Simplifie la gestion par rapport à l'utilisation de variables d'environnement
- Créée à partir d'un fichier, d'un répertoire, ou de valeurs littérales
- Contient une ou plusieurs paires de clé / valeur

Pod avec plusieurs containers

```
$ cat nginx.conf
user www-data;
worker_processes 4;
```

```
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
```

