

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
- v1.14.1, avril 2019

Fonctionnalités

- Automatise le déploiement, la montée en charge et le management des applications microservices conteneurisées
- Boucle de réconciliation vers l'état souhaité (self-healing)
- Gestion d'applications stateless et stateful
- Orchestration du stockage
- Gestion des Secrets et des Configurations
- Long-running et batch jobs
- RBAC

Les concepts de base

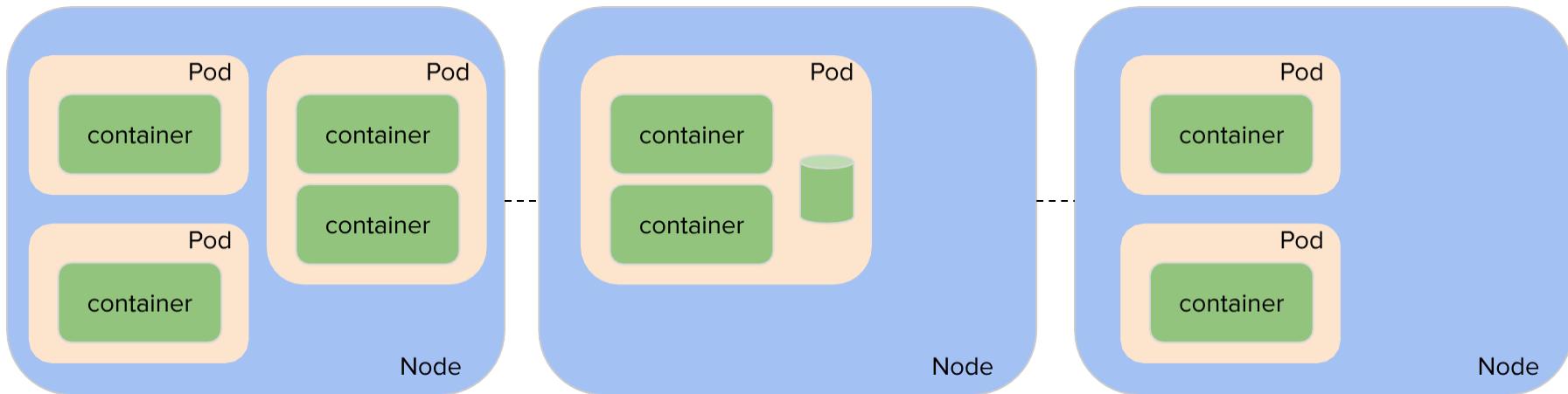
Les concepts de base

Un **cluster** K8S ⇒ un ensemble de **node** Linux ou Windows (VM / bare metal)



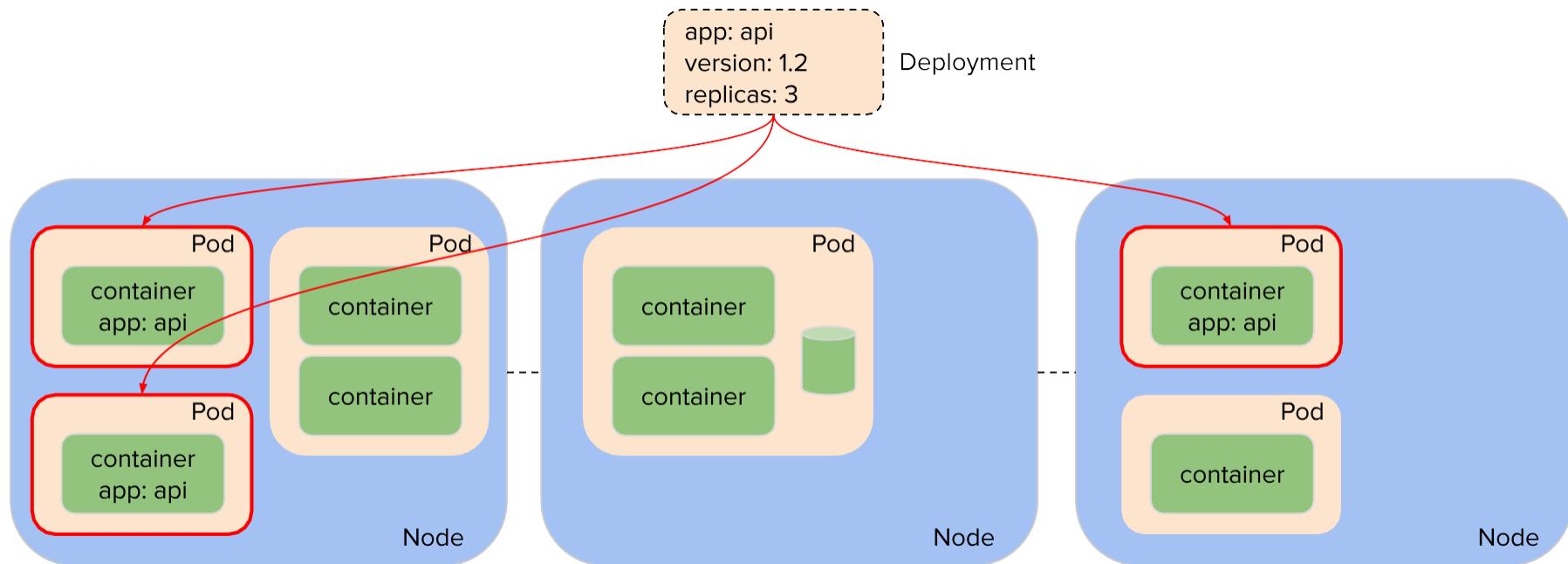
Les concepts de base

Les **nodes** font tourner des **Pods** : groupe de **containers** qui partagent réseau/stockage



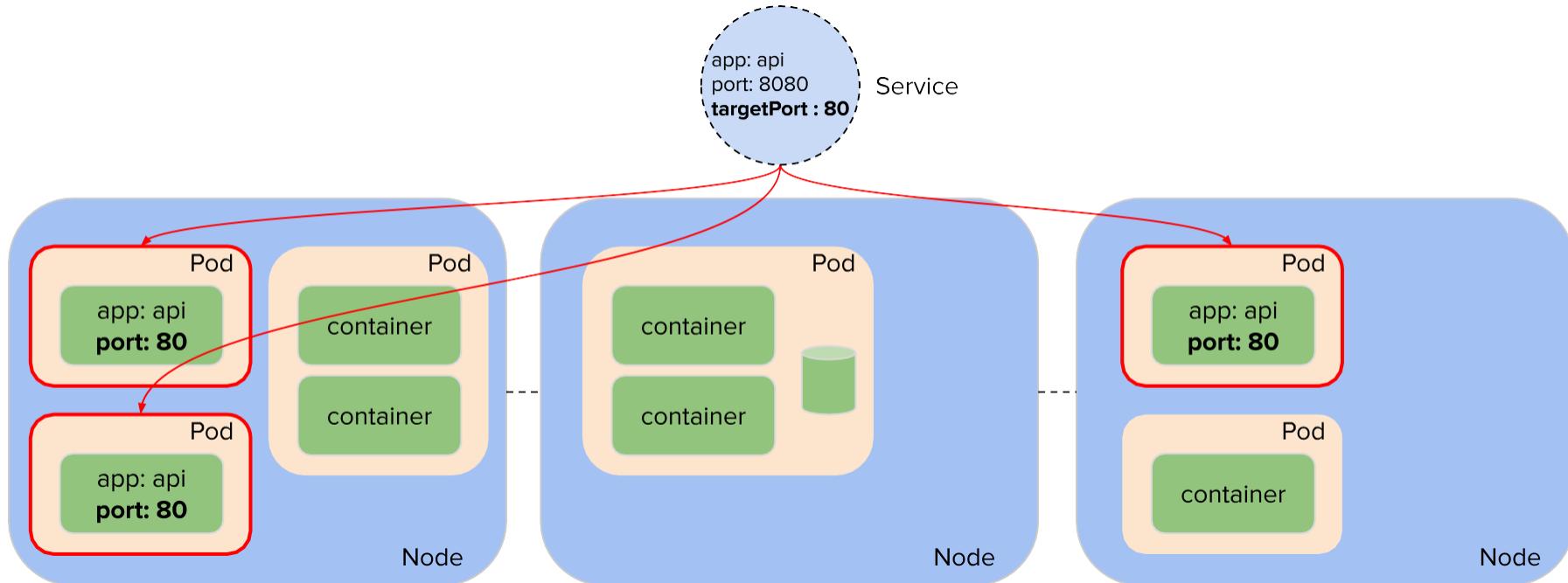
Les concepts de base

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



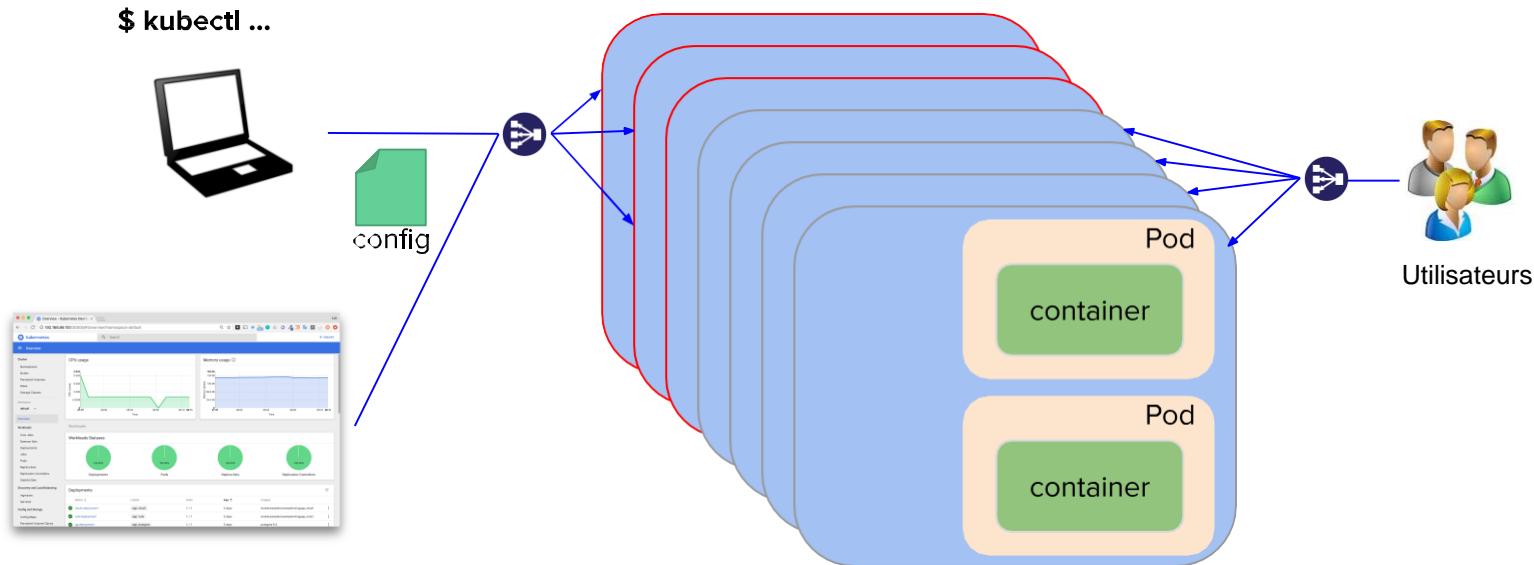
Les concepts de base

Un **Service** permet d'exposer un ensemble de **Pods** à l'intérieur / extérieur du cluster



Les concepts de base

La gestion du cluster est faite via le binaire **kubectl** ou l'interface web



Installation

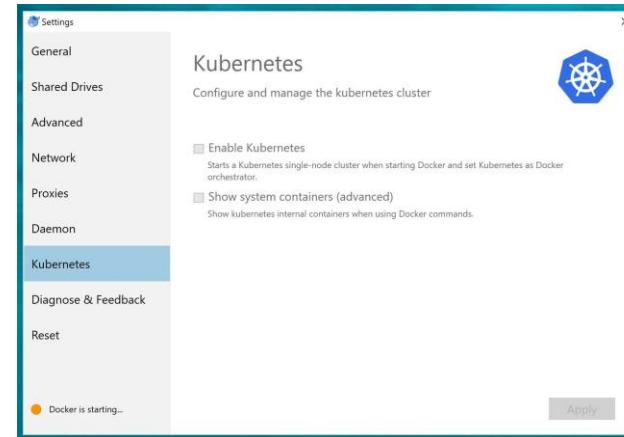
Cluster de développement

Minikube

- Tous les composants de Kubernetes dans une seule VM locale
- Un hyperviseur
 - plusieurs hyperviseurs supportés (VirtualBox, VMWare, HyperKit, ...)
 - <https://www.virtualbox.org/>
- Le binaire Minikube
 - <https://github.com/kubernetes/minikube>
- Le binaire kubectl
 - client pour communiquer avec le cluster via l'API server
 - <https://kubernetes.io/docs/tasks/tools/install-kubectl/>

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>



Cluster de production



Solutions managées

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 - GKE

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

The screenshot shows the 'Créer un cluster Kubernetes' (Create a Kubernetes cluster) page. On the left, there's a sidebar with several cluster creation models:

- Cloner un cluster existant (Clone an existing cluster)
- Cluster standard (Standard cluster)
- Votre premier cluster (Your first cluster)
- Applications nécessitant une utilisation intensive des processeurs (Applications requiring intensive processor usage)
- Applications qui utilisent beaucoup de mémoire (Applications that use a lot of memory)

The main form on the right is filled with the following values:

- Nom: mykube
- Type d'emplacement: Zone (Zone)
- Zone: europe-west3-a
- Version maître: 1.11.8-gke.6 (par défaut)
- Pools de nœuds:
 - default-pool:
 - Nombre de nœuds: 3
 - Type de machine: 2 vCPU, 7.5 Go de mémoire, Personnaliser (Customize)
 - Mettez à jour votre compte pour créer des instances possédant jusqu'à 96 coeurs (Update your account to create instances with up to 96 cores)

At the bottom, there are 'Créer' (Create), 'Réinitialiser' (Reset), and 'Requête REST ou ligne de commande équivalente' (REST API or equivalent command-line equivalent) buttons.

```
$ gcloud init
```

```
$ gcloud container clusters create kube-demo  
--cluster-version=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

```
# Création d'un groupe
$ az group create --name kube-group --location westeurope

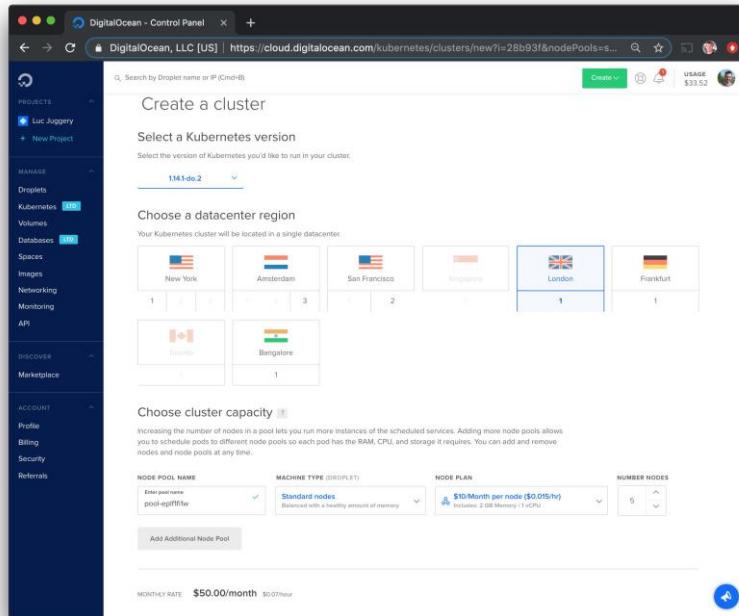
# Création du cluster
$ az aks create --name kube-demo --resource-group kubegroup --node-count 3 --generate-ssh-keys

# Mise à jour de la configuration de kubectl
$ az aks get-credentials --resource-group kubegroup --name kube-demo
```

<https://docs.microsoft.com/fr-fr/cli/azure/install-azure-cli>

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=workers;size=s-2vcpu-4gb;count=5" \
--update-kubeconfig=true
```

<https://github.com/digitalocean/doctl>

Cluster de production



Solutions non managées

De nombreuses solutions

- 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

Installation avec kubeadm

- <https://kubernetes.io/docs/setup/independent/install-kubeadm/>
- Nécessite que les nodes aient été préalablement provisionnés
- Single master ou HA cluster

```
# Initialisation du Master
```

```
$ kubeadm init
```

```
# Mise en place du réseau pour la communication entre les Pods
```

```
$ export kubever=$(kubectl version | base64 | tr -d '\n')
```

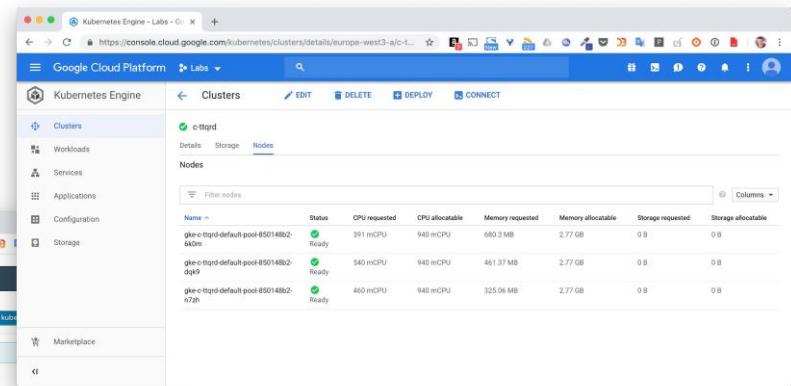
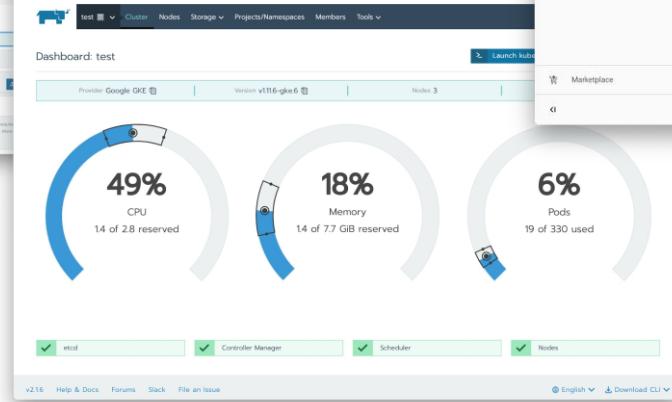
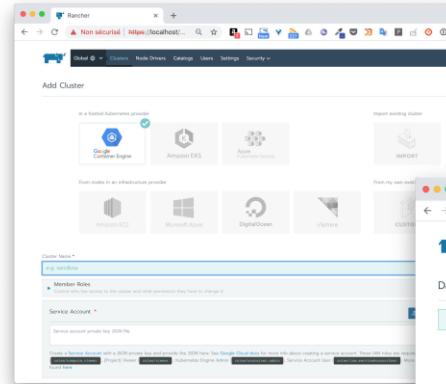
```
$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$kubever"
```

```
# Ajout de node supplémentaires
```

```
$ kubeadm join --token <token> <master-ip>:<master-port>
```

Installation avec Rancher

```
$ docker run -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher:latest
```

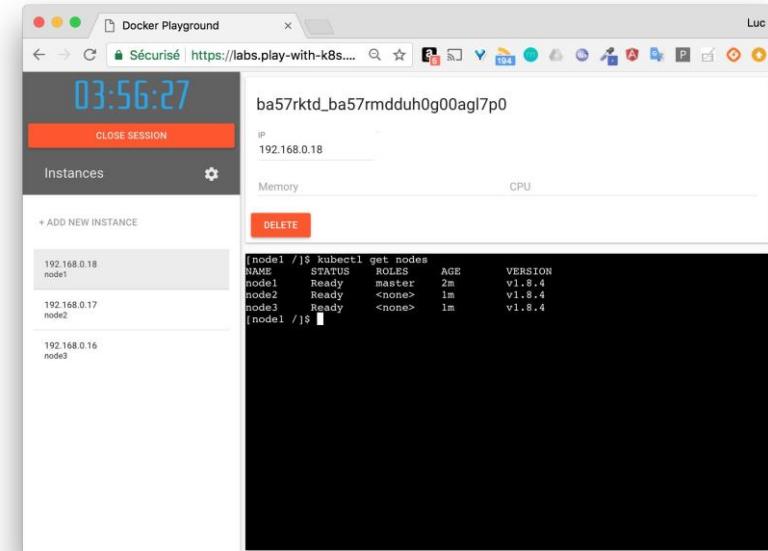


Name	Status	CPU requested	CPU allocatable	Memory requested	Memory allocatable	Storage requested	Storage allocatable
gke-e-ittdr-default-pool-850148b2-640m	Ready	391 mCPU	940 mCPU	680.3 MB	2.77 GB	0 B	0 B
gke-e-ittdr-default-pool-850148b2-dgk9	Ready	540 mCPU	940 mCPU	461.37 MB	2.77 GB	0 B	0 B
gke-e-ittdr-default-pool-850148b2-n7zh	Ready	460 mCPU	940 mCPU	325.06 MB	2.77 GB	0 B	0 B

Playground en ligne

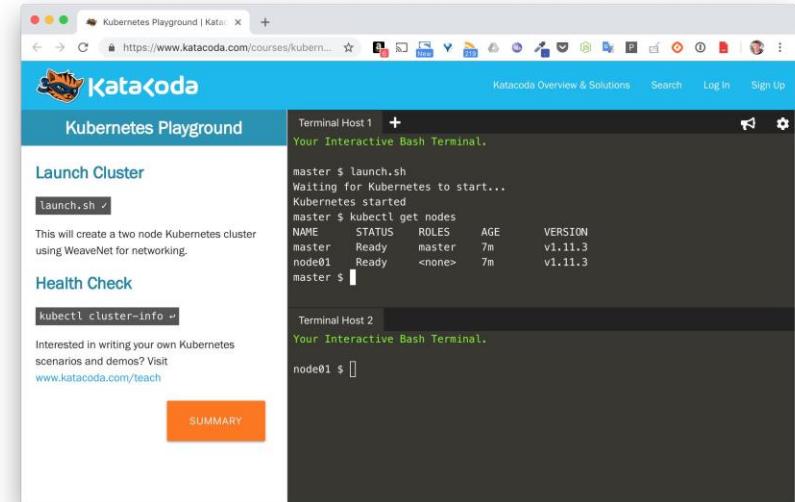
Play with Kubernetes

- a.k.a. PWK, créé par [Marcos Lilljedahl](#) et [Jonathan Leibiusky](#)
- <https://play-with-k8s.com>
- Mise en place d'un cluster avec [kubeadm](#)
- 5 instances utilisables pendant 4h



Katacoda

- Crée par [Ben Hall / https://katacoda.com](https://katacoda.com)
- Plateforme d'apprentissage : nombreuses technologies au catalogue
- <https://www.katacoda.com/courses/kubernetes/playground>
- Cluster avec 2 nodes



Architecture

Les différents types de nodes

```
$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
kube-01	Ready	master	32m	v1.13.4
kube-02	Ready	<none>	24m	v1.13.4
kube-03	Ready	<none>	24m	v1.13.4

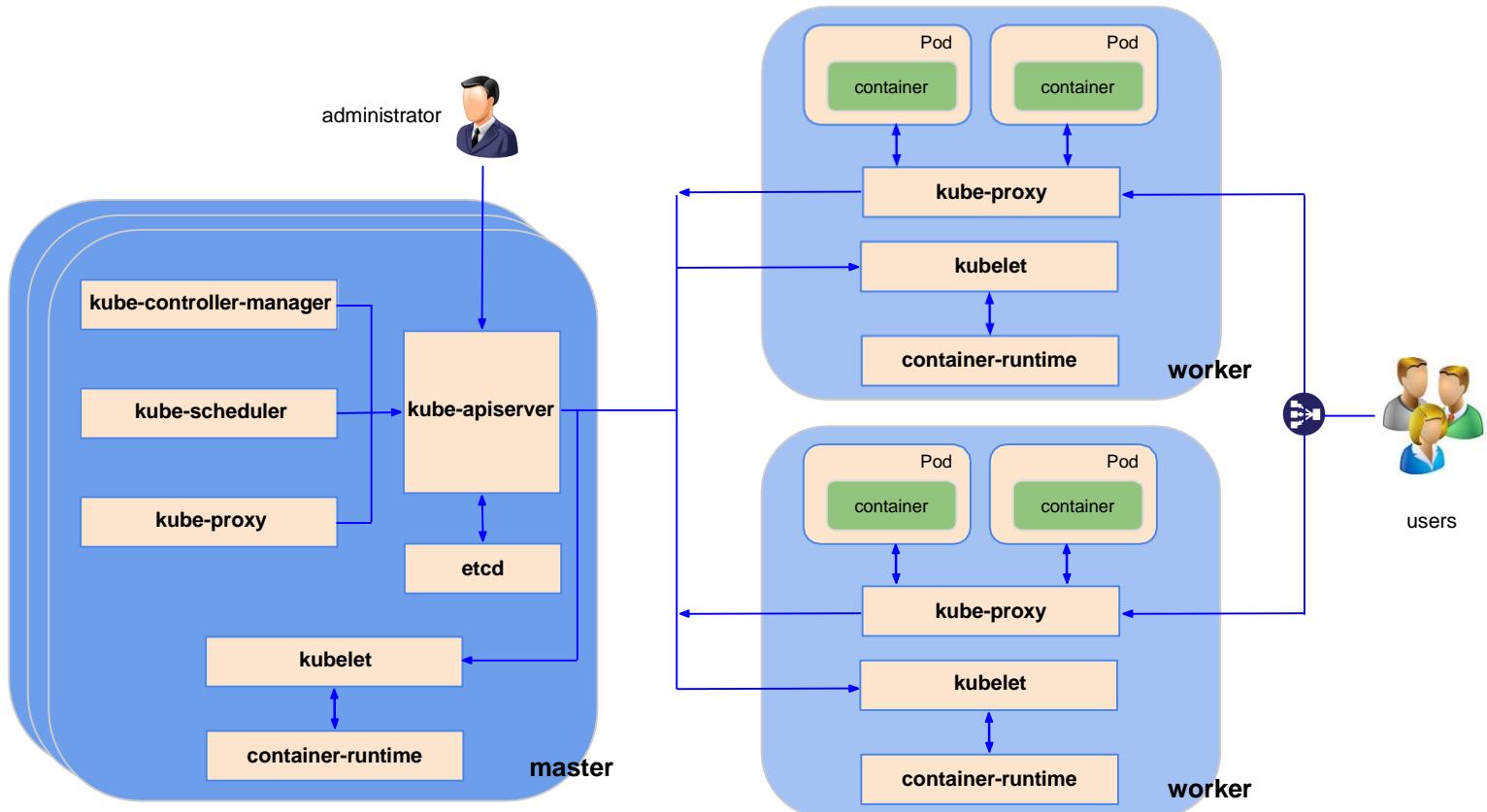
Master

- responsable de la gestion du cluster (“control plane”)
- expose l’API server
- schedule les Pods sur les nodes du cluster

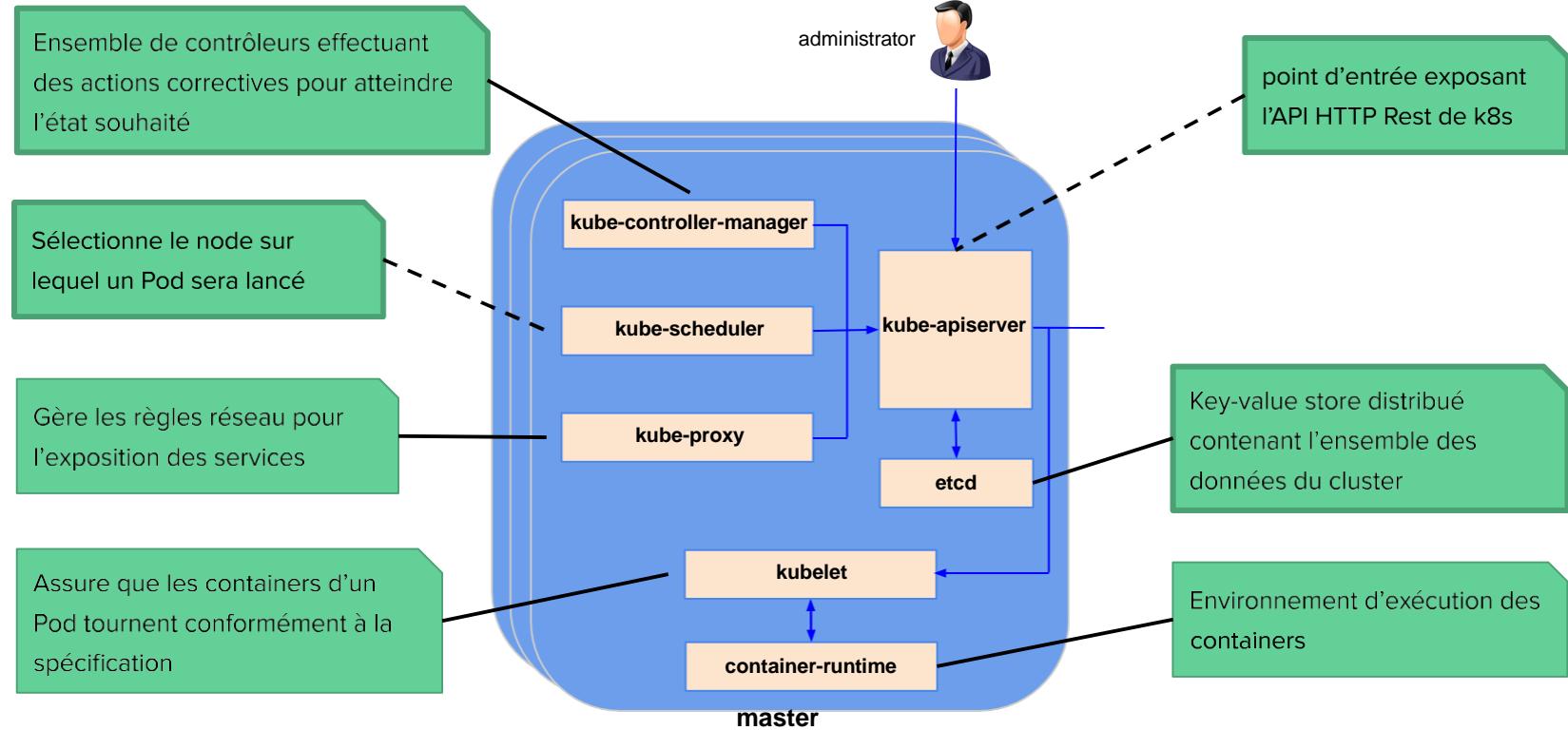
Worker / Node

- node sur lequel sont lancés les Pods applicatifs
- communique avec le Master
- fournit les ressources aux Pods

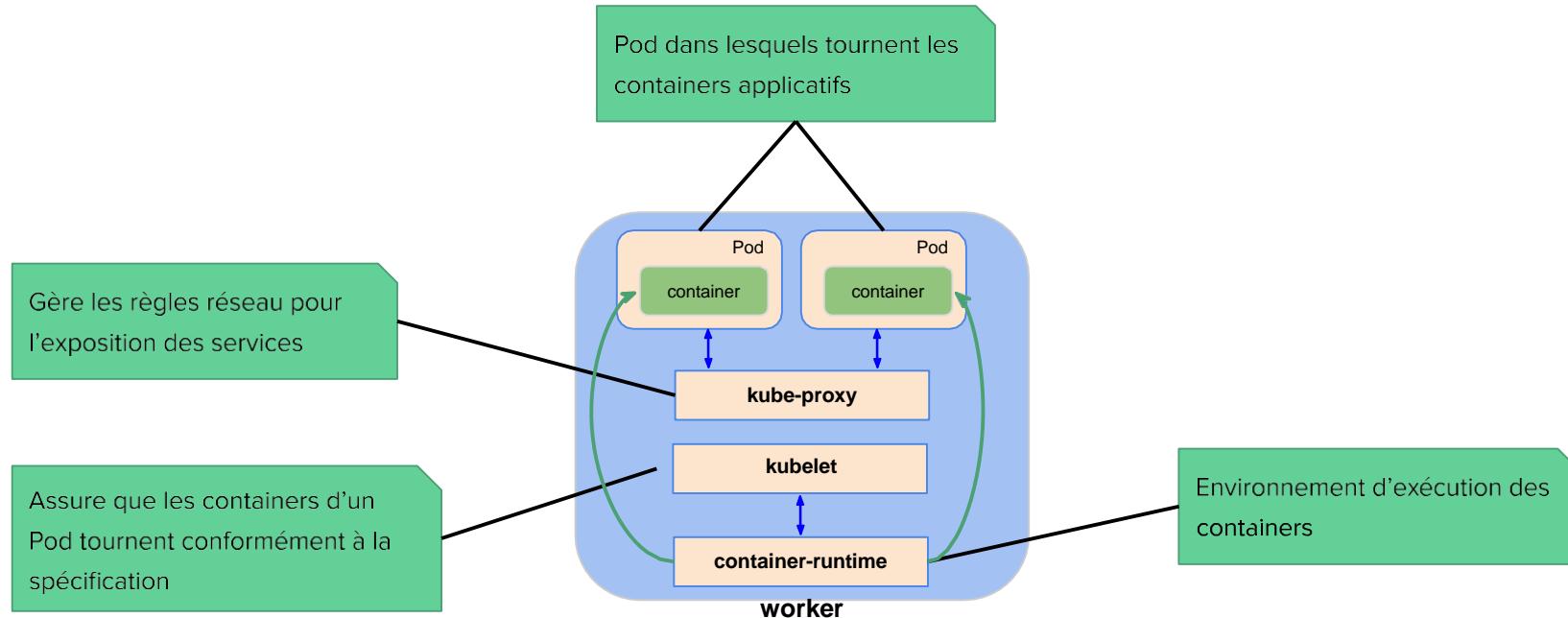
Les processus : vision d'ensemble



Les processus : vision d'ensemble



Les processus : vision d'ensemble



Les processus : vision d'ensemble

```
$ kubectl get pod -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
coredns-86c58d9df4-q84tg	1/1	Running	0	5m
coredns-86c58d9df4-qm6sx	1/1	Running	0	5m
etcd-k8s-node-1	1/1	Running	0	4m
kube-apiserver-k8s-node-1	1/1	Running	0	4m
kube-controller-manager-k8s-node-1	1/1	Running	0	4m
kube-proxy-nc9cb	1/1	Running	0	5m
kube-proxy-wll56	1/1	Running	0	2m
kube-proxy-x77gd	1/1	Running	0	2m
kube-scheduler-k8s-node-1	1/1	Running	0	4m
weave-net-29x5x	2/2	Running	0	30s
weave-net-77ppt	2/2	Running	0	30s
weave-net-q7fb5	2/2	Running	0	30s

Processus tournant sur les masters

Processus tournant sur chacun des nodes (masters et workers)

Processus répliqués sur le cluster (ex: serveur DNS add-on)

Les processus : API Server

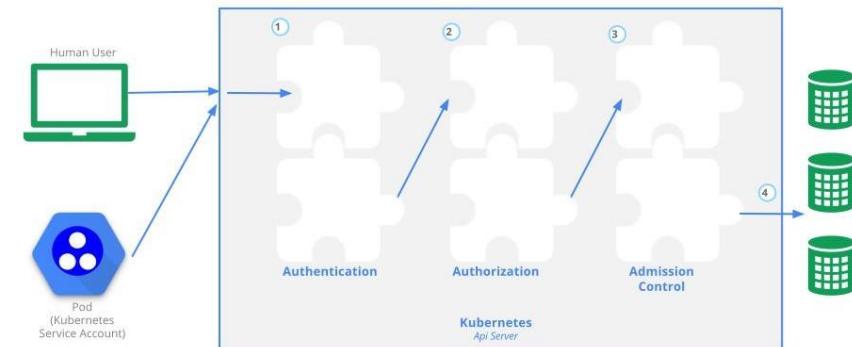
- API Rest / spécification OpenAPI

<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.14>

- Clients: kubectl, interface web, application tournant dans le cluster

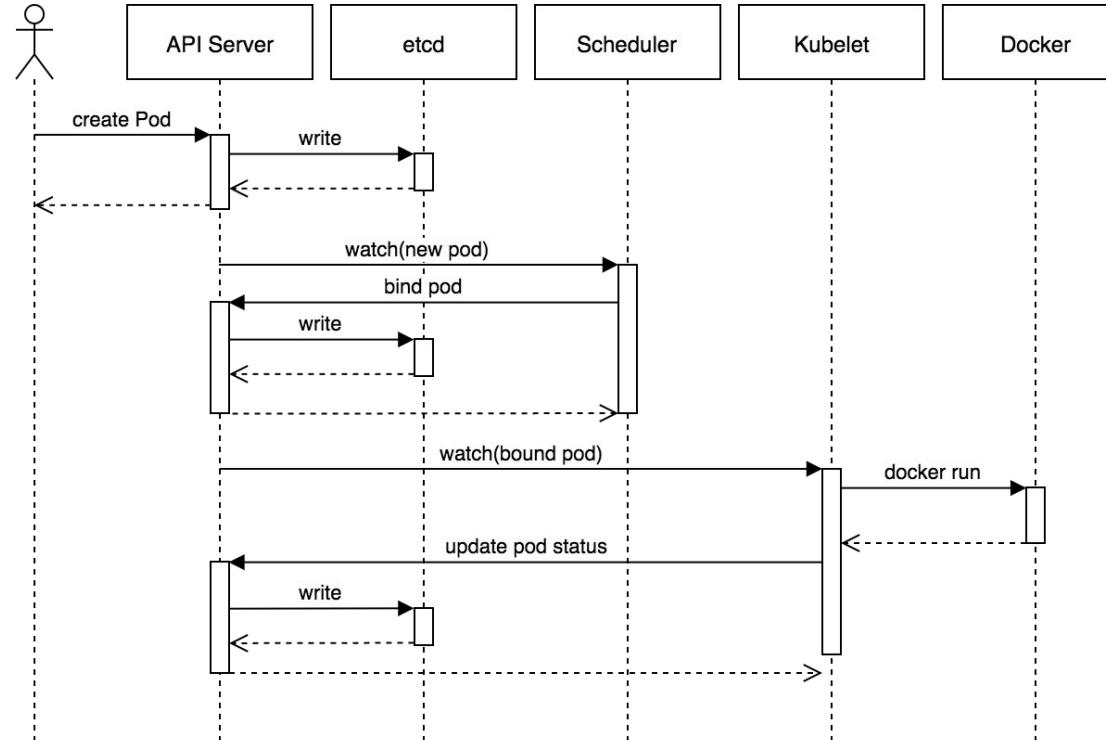
- Chaque requête passe dans un pipeline

- authentification
 - autorisation
 - admission contrôleur
 -



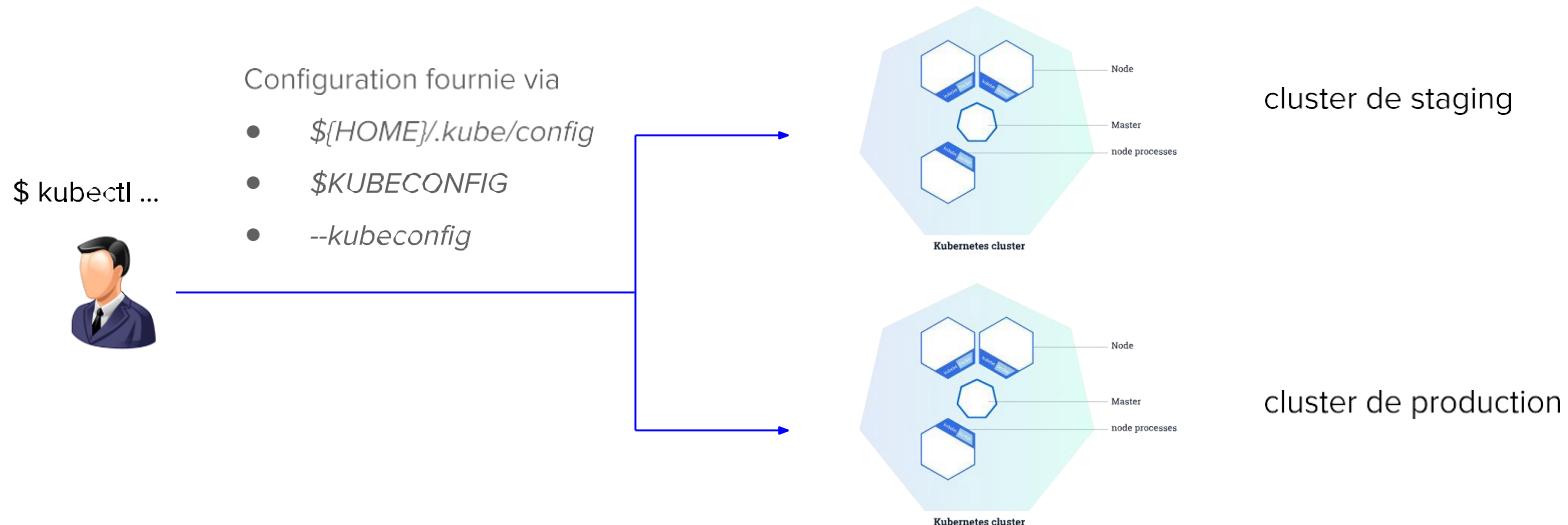
<https://kubernetes.io/docs/admin/accessing-the-api/>

Les processus : workflow de création d'un Pod



Context d'utilisation

- Définie le cluster auquel on s'adresse et avec quel utilisateur
- <https://github.com/ahmetb/kubectx>



Les ressources

Différentes catégories

- Gestion des applications lancées sur le cluster (Deployment, Pod, ...)
- Discovery et load balancing (Service, ...)
- Configuration des applications (ConfigMap, Secret, ...)
- Stockage (PersistentVolume, PersistentVolumeClaim, ...)
- Configuration du cluster et metadata (Namespace, ...)
- Des objets supplémentaires pour ajouter des fonctionnalités (CRD)

Différentes catégories : une même structure yaml

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

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

Définition d'une application

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

Exposition d'une application

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: www-domain
spec:
  rules:
    - host: www.example.com
      http:
        paths:
          - backend:
              serviceName: www
              servicePort: 80
```

Exposition / routing

Metadata : Labels et Annotations

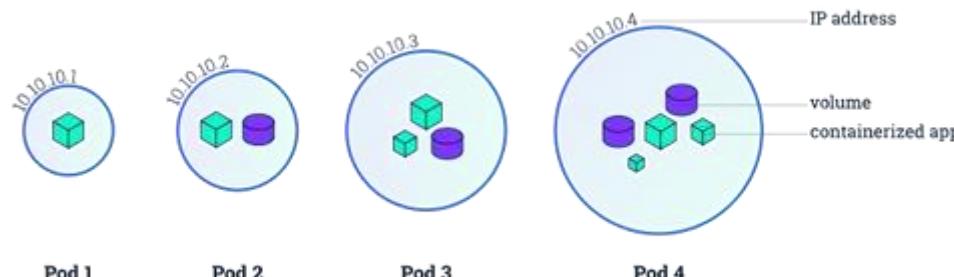
- Utilisés pour attacher des metadata aux objets
- Format : key / value
- Labels
 - utilisés pour la sélection d'objets
 - récupération de collections
- Annotations
 - ne sert pas à sélectionner des objets
 - non interne à Kubernetes
 - utilisées par des outils et librairies clientes

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

Pod

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

Exemple - server http

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

```
$ cat nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: www
spec:
  containers:
  - name: nginx
    image: nginx:1.14-alpine
```

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          L'objet Pod est stable et disponible depuis la v1 de l'API
kind: Pod
metadata:
  name: www
spec:
  containers:
    - name: nginx
      image: nginx:1.14-alpine
```

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: www
spec:
  containers:
    - name: nginx
      image: nginx:1.14-alpine
```

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: www
spec:
  containers:
    - name: nginx
      image: nginx:1.14-alpine
```

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: www
spec:
  containers:          Spécification des containers lancés dans le Pod (un seul ici)
  - name: nginx
    image: nginx:1.14-alpine
```

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

Les commandes de base

```
# 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/1.14.2

# Shell interactif dans un Pod
$ kubectl exec -t -i www -- /bin/sh
/ #
```

Les commandes de base

```
# 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

# Log du Pod
$ kubectl logs www

# Suppression du Pod
$ kubectl delete pod www
pod "www" deleted
```

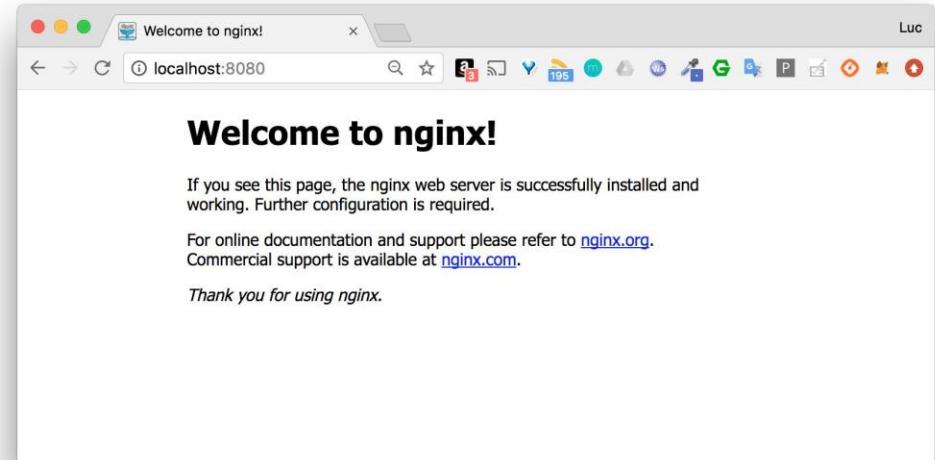
Ensemble des événements survenus lors du lancement du Pod



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

- Un container **pause** dans chaque Pod
- 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

- 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

Pod avec plusieurs containers : exemple

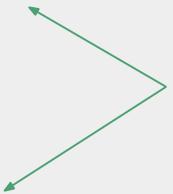
```
apiVersion: v1
kind: Pod
metadata:
  name: wp
spec:
  containers:
    - image: wordpress:4.9-apache
      name: wordpress
      env:
        - name: WORDPRESS_DB_PASSWORD
          value: mysqlpwd
        - name: WORDPRESS_DB_HOST
          value: 127.0.0.1
    - image: mysql:5.7
      name: mysql
      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 : exemple

```
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: {}
```



Ajout de variables d'environnement dans les containers

Pod avec plusieurs containers : exemple

```
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: {}
```

Définition d'un volume: répertoire sur la machine hôte lié au cycle de vie du Pod

Pod avec plusieurs containers : exemple

```
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

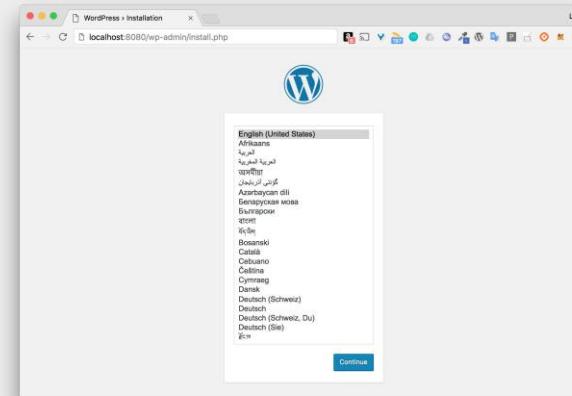
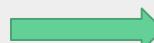
Définition d'un volume: répertoire sur la machine hôte lié au cycle de vie du Pod

Pod avec plusieurs containers : exemple

```
# 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
```

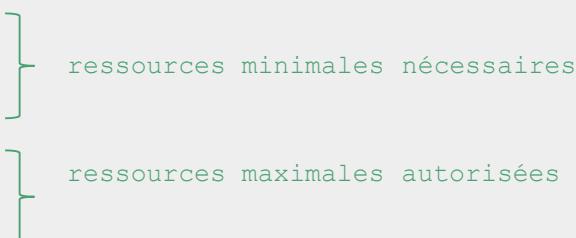
```
# Exposition du port 80 du container wordpress
$ kubectl port-forward wp 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Handling connection for 8080
```



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 diagram consists of two curly braces placed vertically next to each other. The top brace groups the 'requests' section under 'ressources minimales nécessaires'. The bottom brace groups the 'limits' section under 'ressources maximales autorisées'.

ressources minimales nécessaires

ressources maximales autorisées

Readiness et Liveness probes

- Utilisé par *Kubelet*
- Readiness
 - permet d'attendre que le container soit démarré
 - pas de traffic envoyé au Pod tant que le test n'est pas passé
- Liveness
 - permet de savoir si l'application fonctionne correctement
 - redémarre le container si le test échoue
- Différents type
 - HTTP
 - TCP
 - via une commande

Readiness et Liveness probes

```
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
    - name: goproxy
      image: k8s.gcr.io/goproxy:0.1
      ports:
        - containerPort: 8080
      readinessProbe:
        tcpSocket:
          port: 8080
        initialDelaySeconds: 5
        periodSeconds: 10
      livenessProbe:
        tcpSocket:
          port: 8080
        initialDelaySeconds: 15
        periodSeconds: 20
```

Assure que l'application est démarrée et qu'elle est prête à recevoir du trafic

Readiness et Liveness probes

```
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

Assure que l'application fonctionne correctement

Scheduling : nodeSelector

- Permet de schéduler des Pods sur certains nodes seulement
- Se base sur les labels existant sur les nodes

```
$ 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
...
...
```

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

Scheduling : 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”)
 - requiredDuringSchedulingRequiredDuringExecution (à venir)

Scheduling : 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**

Scheduling : 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
 - ...

Scheduling : 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**

Scheduling : tolerations

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

```
$ 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
...
...
```

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

Clé *toleration* dans la spécification du Pod

Network Policy

- Définit comment des groupes de Pods peuvent communiquer entre eux
- La ressource **NetworkPolicy** utilise les labels pour sélectionner les Pods et définir le trafic qui est autorisé vers ceux-ci
- Nécessite un plugin Réseau qui supporte les Network Policies

Network Policy

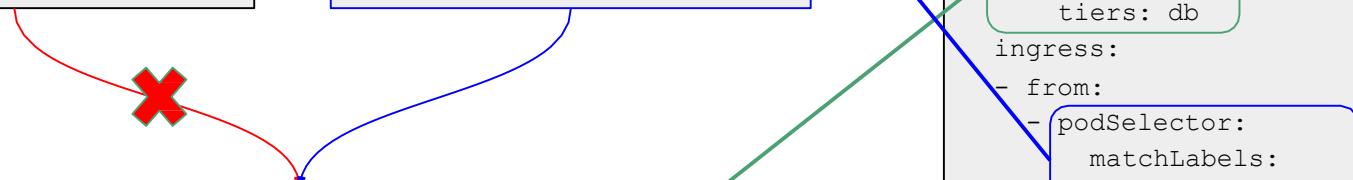
```
apiVersion: v1
kind: Pod
metadata:
  name: www
  labels:
    tiers: front
...
```

```
apiVersion: v1
kind: Pod
metadata:
  name: api
  labels:
    tiers: backend
...
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-policy
spec:
```

```
podSelector:
  matchLabels:
    tiers: db
ingress:
- from:
  - podSelector:
    matchLabels:
      tiers: backend
```

```
apiVersion: v1
kind: Pod
metadata:
  name: db
  labels:
    tiers: db
...
```



Seuls les Pods du backend peuvent accéder à l'API

Network Policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - ipBlock:
            cidr: 172.17.0.0/16
            except:
              - 172.17.1.0/24
        - namespaceSelector:
            r: matchLabels:
                project: myproject
  ...
...
```

```
...
  - podSelector:
      matchLabels:
        role: frontend
  ports:
    - protocol: TCP
      port: 6379
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
  ports:
    - protocol: TCP
      port: 5978
```

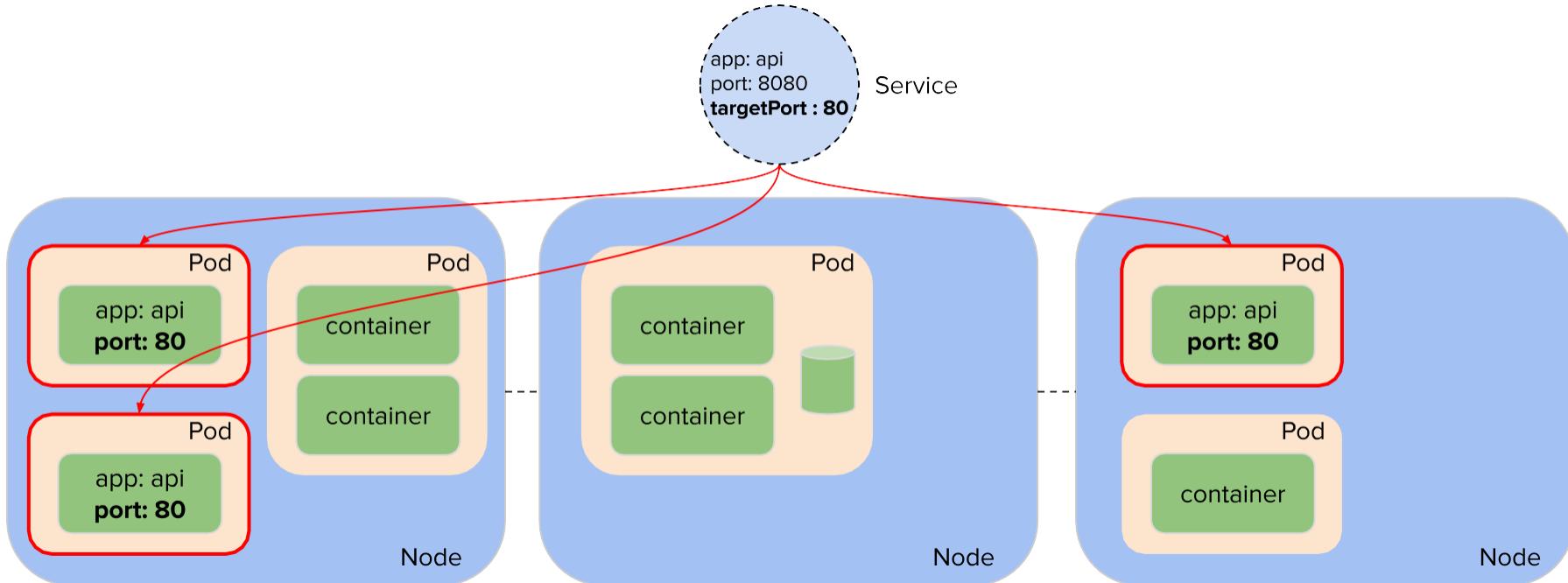
En résumé

- 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 **Deployment**
- Exposition dans le cluster ou vers l'extérieur via un **Service**

Service

Utilisation

Un **Service** permet d'exposer un ensemble de **Pods** à l'intérieur / extérieur du cluster



Utilisation

- Abstraction définissant un ensemble logique de Pods
- Groupement basé sur l'utilisation de labels
- Assure le découplage
 - replicas / instances du microservice
 - consommateurs du microservice
- En charge de la répartition de la charge entre les Pods sous-jacents
- Adresse IP persistante

Différents types

- ClusterIP (défaut) : exposition à l'intérieur du cluster
- NodePort : exposition vers l'extérieur
- LoadBalancer : intégration avec un Cloud Provider
- ExternalName : définit un alias vers un service extérieur au cluster

Spécification : exemple de type ClusterIP

```
$ cat vote-service-clusterIP.yaml
apiVersion: v1           Service est stable depuis la version 1 de l'API
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
```

Spécification : exemple de type ClusterIP

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

Spécification : exemple de type ClusterIP

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

Spécification : exemple de type ClusterIP

```
$ cat vote-service-clusterIP.yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:           Indique les Pods que le service va regrouper
    app: vote        (les Pods ayant le label "app: vote")
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
```

Spécification : exemple de type ClusterIP

```
$ cat vote-service-clusterIP.yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: ClusterIP          Type par défaut, d'autres Pods accèdent au service par son nom
  ports:
  - port: 80
    targetPort: 80
```

Spécification : exemple de type ClusterIP

```
$ cat vote-service-clusterIP.yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: ClusterIP
  ports:
    - port: 80          Le service expose le port 80 dans le cluster
      targetPort: 80    Requêtes envoyées sur le port 80 d'un des Pods du groupe
```

Spécification : exemple de type ClusterIP

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

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

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

Spécification : exemple de type ClusterIP

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

# Lancement du Service de type ClusterIP
$ kubectl create -f vote-service-clusterIP.yaml

# Lancement d'un Pod utilisé pour le debug
$ kubectl create -f pod-debug.yaml →

# Accès au Service vote depuis le Pod debug
$ kubectl exec -ti debug sh
/ # apk update && apk add curl
/ # curl http://vote-service
(code html de l'interface vote)
...
# Résolution DNS via SERVICE_NAME.NAMESPACE
/ # curl http://vote-service.default
...
```

```
apiVersion: v1
kind: Pod
metadata:
  name: debug
spec:
  containers:
    - name: debug
      image: alpine
      command:
        - "sleep"
        - "10000"
```

La commande “sleep 10000” est découpée et chaque élément est une entrée dans la liste

Spécification : exemple de type NodePort

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

Spécification : exemple de type NodePort

```
$ cat vote-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: NodePort
  ports:
    - port: 80          Le service expose le port 80 dans le cluster
      targetPort: 80
      nodePort: 31000
```

Spécification : exemple de type NodePort

```
$ cat vote-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: NodePort
  ports:
  - port: 80
    targetPort: 80          Requêtes envoyées sur le port 80 d'un des Pods du groupe
    nodePort: 31000
```

Spécification : exemple de type NodePort

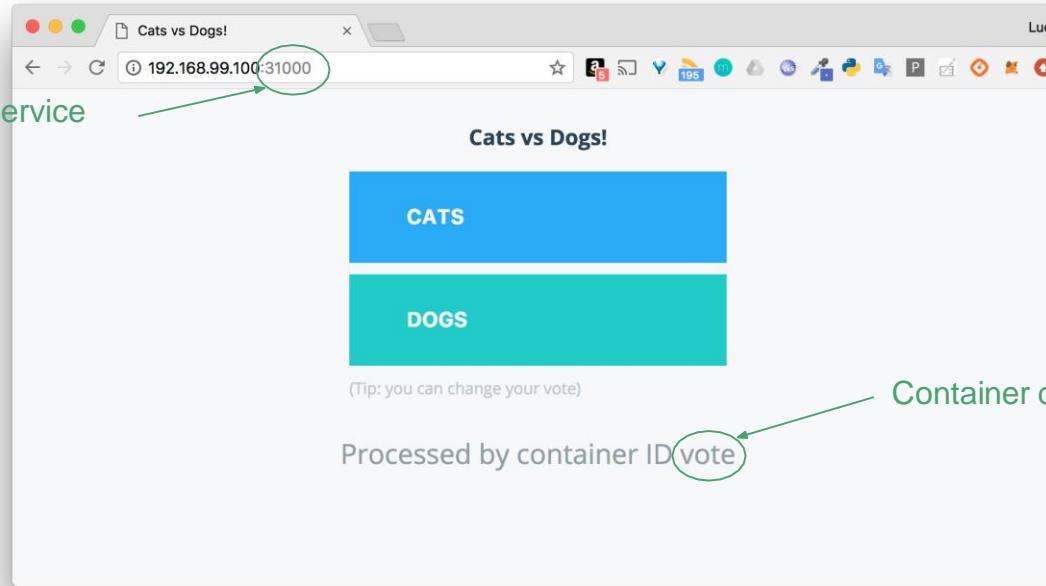
```
$ cat vote-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31000
```

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

Spécification : exemple de type NodePort

```
# Lancement du Service  
$ kubectl create -f vote-service.yaml
```

Port exposé par le service



Container défini dans le Pod

Spécification : exemple de type LoadBalancer

```
$ cat vote-service-lb.yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-service-lb
spec:
  selector:
    app: vote
  type: LoadBalancer          Service de type LoadBalancer, exposé sur chaque node du cluster
  ports:
  - port: 8080
    targetPort: 80
$ kubectl create -f vote-service-lb.yaml
```

Spécification : exemple de type LoadBalancer sur GCP

```
$ kubectl svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT (S)	AGE
svc/vote-service-lb	LoadBalancer	10.11.244.119	35.205.42.144	8080:30837/TCP	59s

The screenshot shows the Google Cloud Platform Network Services interface. On the left, a sidebar menu includes 'Services réseau' (selected), 'Équilibrage de charge' (highlighted in green), 'Cloud DNS', 'CDN Cloud', and 'Cloud NAT'. The main panel displays 'Détails de l'équilibreur de charge' for a load balancer named 'a68345282f00711e88ca842010af0008'. The 'Frontend' section shows a single rule: 'TCP' on port '35.205.42.144:8080'. The 'Backend' section lists three instances: 'gke-sophia-default-pool-f429fc1e-f4jl', 'gke-sophia-default-pool-f429fc1e-tprc', and 'gke-sophia-default-pool-f429fc1e-n419', all marked as healthy (green checkmark). A green oval highlights the 'IP-Port' field in the frontend configuration.

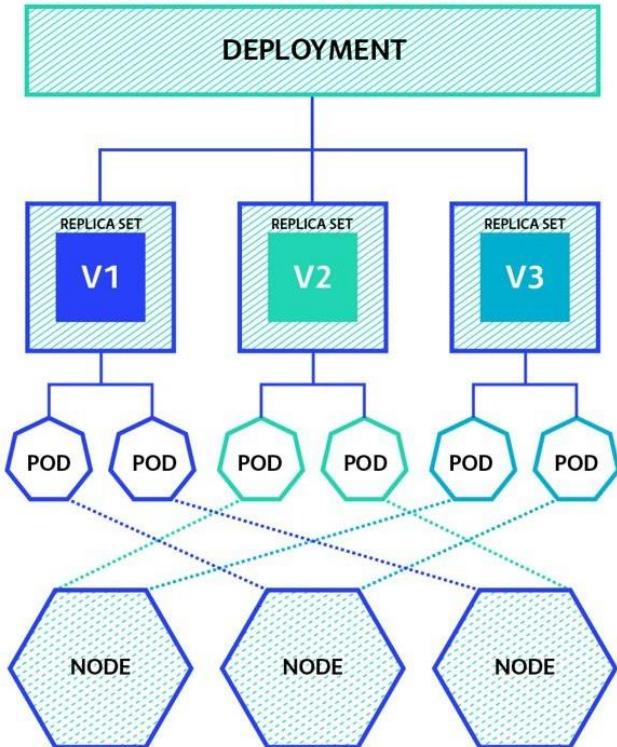
Création d'un nouveau LoadBalancer dans GCP



Deployment

Utilisation

- Différents niveaux d'abstraction
 - Deployment
 - ReplicaSet
 - Pod
- Un Deployment gère des ReplicaSet
- ReplicaSet
 - une version de l'application
 - gère un ensemble de Pods de même spécification
 - assure que les Pods sont actifs
- Pod généralement créés via un Deployment



Utilisation

- Un Deployment définit un “état souhaité”
 - spécification d'un Pod et du nombre de réplicas voulu
- Un contrôleur pour faire converger l'état courant vers l'état souhaité
- Gère les mises à jour d'une application
 - rollout / rollback / scaling
- Différentes stratégies de mise à jour
 - rolling update, blue / green, canary, ...

Spécification : exemple

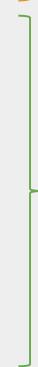
```
$ cat vote-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vote
template:
  metadata:
    labels:
      app: vote
  spec:
    containers:
      - name: vote
        image: instavote/vote
        ports: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



Deployment



ReplicaSet



Pod

Spécification : exemple

```
$ cat vote-deployment.yaml
apiVersion: apps/v1                                     Version de l'API dans laquelle l'objet Deployment est défini
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
```

Spécification : exemple

```
$ cat vote-deployment.yaml
apiVersion: apps/v1
kind: Deployment                                "Deployment" est le type de la resource considérée
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
```

Spécification : exemple

```
$ cat vote-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:                                     Le nom "vote-deploy" est donné au deployment
  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
```

Spécification : exemple

```
$ cat vote-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote-deploy
spec:                                     Définition de la façon dont seront lancés les Pods
  replicas: 3
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      labels:
        app: vote
    spec:
      containers:
        - name: vote
          image: instavote/vote
          ports:
            - containerPort: 80
```

Spécification : exemple

```
$ cat vote-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote-deploy
spec:
  replicas: 3          3 replicas seront créés pour ce Deployment
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      labels:
        app: vote
    spec:
      containers:
        - name: vote
          image: instavote/vote
          ports:
            - containerPort: 80
```

Spécification : exemple

```
$ cat vote-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote-deploy
spec:
  replicas: 3
  selector:                                Détermine les Pods qui seront managés par ce Deployment
    matchLabels:                            Seuls les Pods ayant le label "app: vote" seront considérés
      app: vote
  template:
    metadata:
      labels:
        app: vote
    spec:
      containers:
        - name: vote
          image: instavote/vote
          ports:
            - containerPort: 80
```

Spécification : exemple

```
$ cat vote-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vote
  template:          Spécification des Pods, correspond à la clé spec utilisée lors de la
    metadata:        définition d'un Pod
      labels:
        app: vote
  spec:
    containers:
      - name: vote
        image: instavote/vote
        ports:
          - containerPort: 80
```

Spécification : exemple

```
# Lancement du Deployment
$ kubectl create -f vote-deployment.yaml
deployment "vote-deploy" created

# Liste des Deployments
$ kubectl get deploy
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
vote-deploy  3          3          3           3           31s

# Liste des ReplicaSet
$ kubectl get rs
NAME      DESIRED   CURRENT   READY   AGE
vote-deploy-584c4c76db  3          3          3       34s

# Liste des Pods
$ kubectl get pod
NAME                  READY   STATUS    RESTARTS   AGE
vote-deploy-584c4c76db-65r7r  1/1     Running   0          36s
vote-deploy-584c4c76db-g19ps  1/1     Running   0          36s
vote-deploy-584c4c76db-s7gdn  1/1     Running   0          36s
```

Un ReplicaSet est créé et associé au Deployment



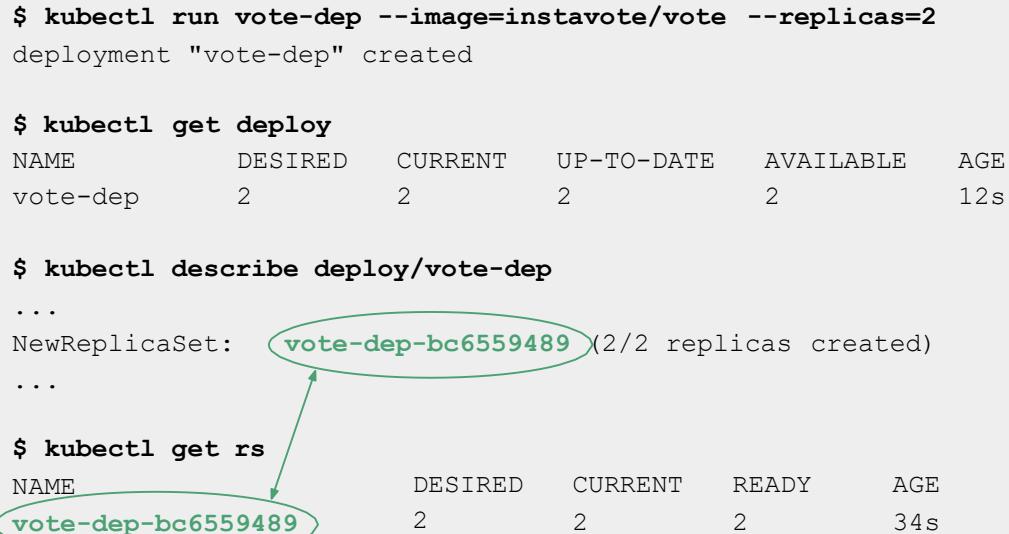
Le ReplicaSet gère les 3 Pods (replicas) définis dans la spécification du Deployment

Création d'un Deployment avec `kubectl run`

```
$ kubectl run vote-dep --image=instavote/vote --replicas=2
deployment "vote-dep" created

$ kubectl get deploy
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
vote-dep   2          2          2             2            12s

$ kubectl describe deploy/vote-dep
...
NewReplicaSet: vote-dep-bc6559489 (2/2 replicas created)
...
$ kubectl get rs
NAME      DESIRED   CURRENT   READY   AGE
vote-dep-bc6559489   2          2          2       34s
```



Stratégies de mise à jour d'un Deployment

- **recreate**
 - supprime l'ancienne version et créée 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
- **canary**
 - nouvelle version pour un sous-ensemble d'utilisateurs
- **a/b testing**
 - nouvelle version pour un sous-ensemble **défini** d'utilisateurs (basé sur header HTTP, cookie, ...)
 - ex: test d'une nouvelle fonctionnalité

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
  labels:
    app: www
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    replicas: 2
  template:
    metadata:
      labels:
        app: www
    spec:
      ...
```

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 "vote-deploy" created

Enregistrement des changements effectués dans chaque révision

1ère mise à jour de l'image du container vote défini dans la spécification

```
$ kubectl set image deployment/vote-deploy vote=instavote/vote:indent
```

deployment "vote-deploy" image updated

2nde mise à jour de l'image

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

deployment "vote-deploy" image updated

Liste des ReplicaSets pour ce Deployment

```
$ kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
vote-deploy-584c4c76db	0	0	0	3m
vote-deploy-585cd89dd4	0	0	0	45s
vote-deploy-64f888dd55	3	3	3	14s

ReplicaSet actif

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

Mise à jour d'un Deployment : rolling update

```
$ kubectl describe deploy/vote-deploy
```

```
...
```

```
NewReplicaSet: vote-deploy-64f888dd55 (3/3 replicas created)
```

```
Events:
```

Type	Reason	Age	From	Message
----	-----	----	----	-----
Normal	ScalingReplicaSet	19m	deployment-controller	Scaled up replica set vote-deploy-584c4c76db to 3
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled up replica set vote-deploy-585cd89dd4 to 1
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled down replica set vote-deploy-584c4c76db to 2
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled up replica set vote-deploy-585cd89dd4 to 2
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled down replica set vote-deploy-584c4c76db to 1
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled up replica set vote-deploy-585cd89dd4 to 3
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled down replica set vote-deploy-584c4c76db to 0
Normal	ScalingReplicaSet	16m	deployment-controller	Scaled up replica set vote-deploy-64f888dd55 to 1
Normal	ScalingReplicaSet	16m	deployment-controller	Scaled down replica set vote-deploy-585cd89dd4 to 2
Normal	ScalingReplicaSet	16m (x4 over 16m)	deployment-controller	(combined from similar events): Scaled down replica set vote-deploy-585cd89dd4 to 0

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
deployments "vote-deploy"
REVISION  CHANGE-CAUSE
1          kubectl create --filename=vote-deployment.yaml --record=true
2          kubectl set image deployment/vote-deploy vote=instavote/vote:indent
3          kubectl set image deployment/vote-deploy vote=instavote/vote:movies

$ kubectl describe deploy/vote-deploy
...
Annotations:  deployment.kubernetes.io/revision=3
              kubernetes.io/change-cause=kubectl set image deployment/vote-deploy
vote=instavote/vote:movies
```

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.Y`

```
$ kubectl rollout undo deployment/vote-deploy  
deployment "vote-deploy"
```

```
$ kubectl rollout undo deployment/vote-deploy --to-revision=1  
deployment "vote-deploy"
```