



FORMATION KUBERNETES

1 jour



Disposition de titre et de contenu avec liste

- **Module 1: Introduction à Kubernetes**
- **Module 2: Les objets: Pod**
- **Module 3: Les objets - Service**
- **Module 4: Les objets - Deployment**



MODULE 1

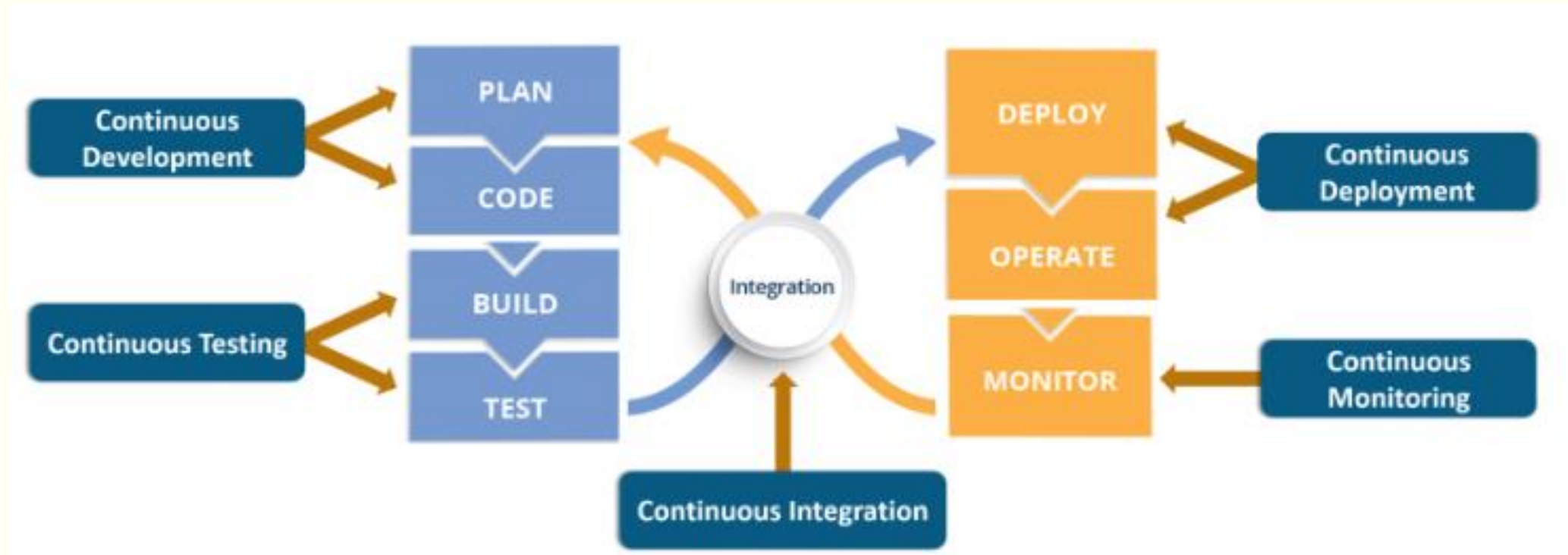
Introduction à Kubernetes



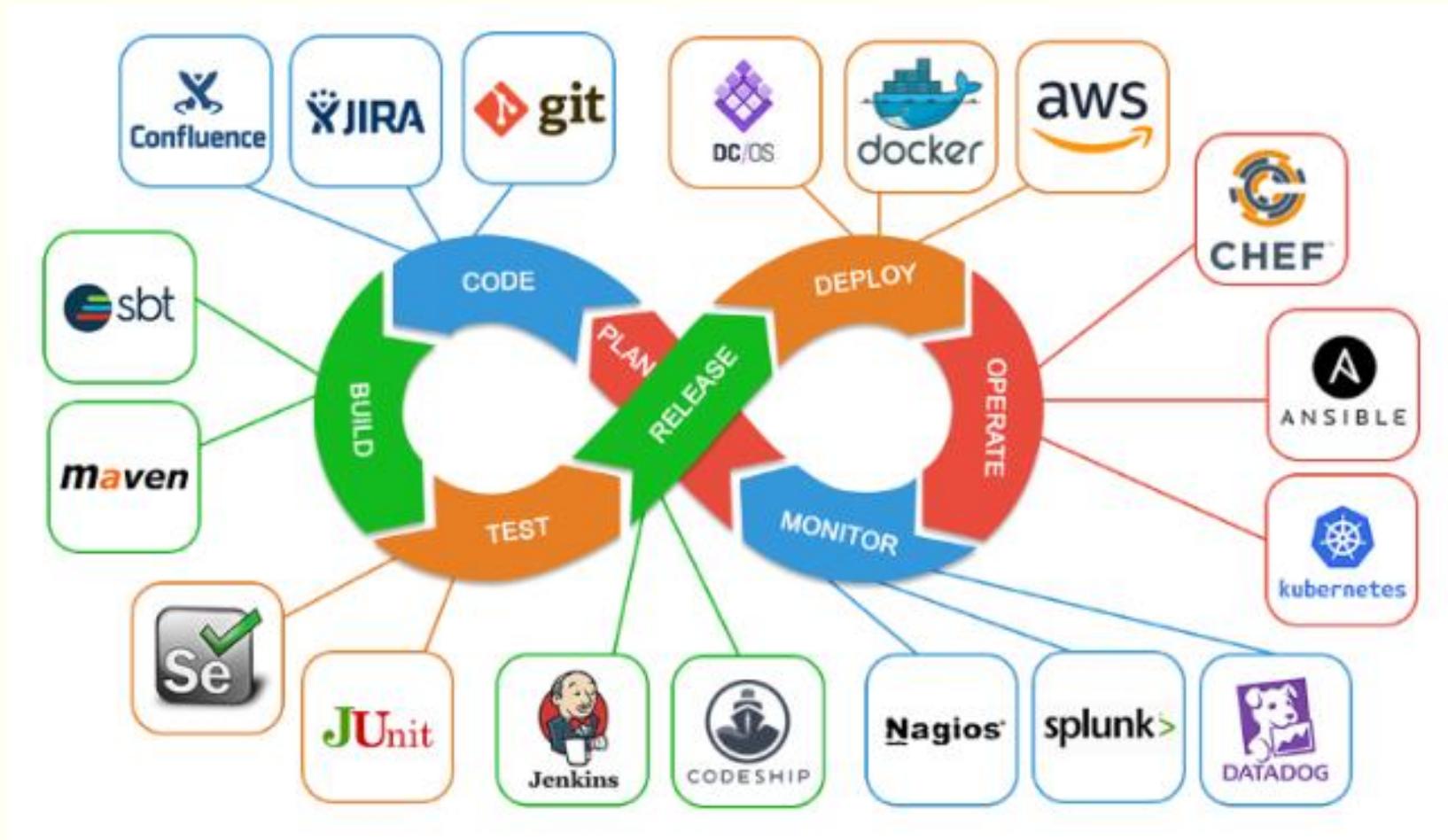
Introduction à Kubernetes

- Cycle de vie DevOps
- Outils DevOps
- Un peu d'histoire !
- Définition d'un container
- Définition d'un orchestrateur de containers
- GitOps pipeline
- Kubernetes : Les composants
- Minikube

Cycle de vie DevOps



Outils DevOps



Histoire de Kubernetes

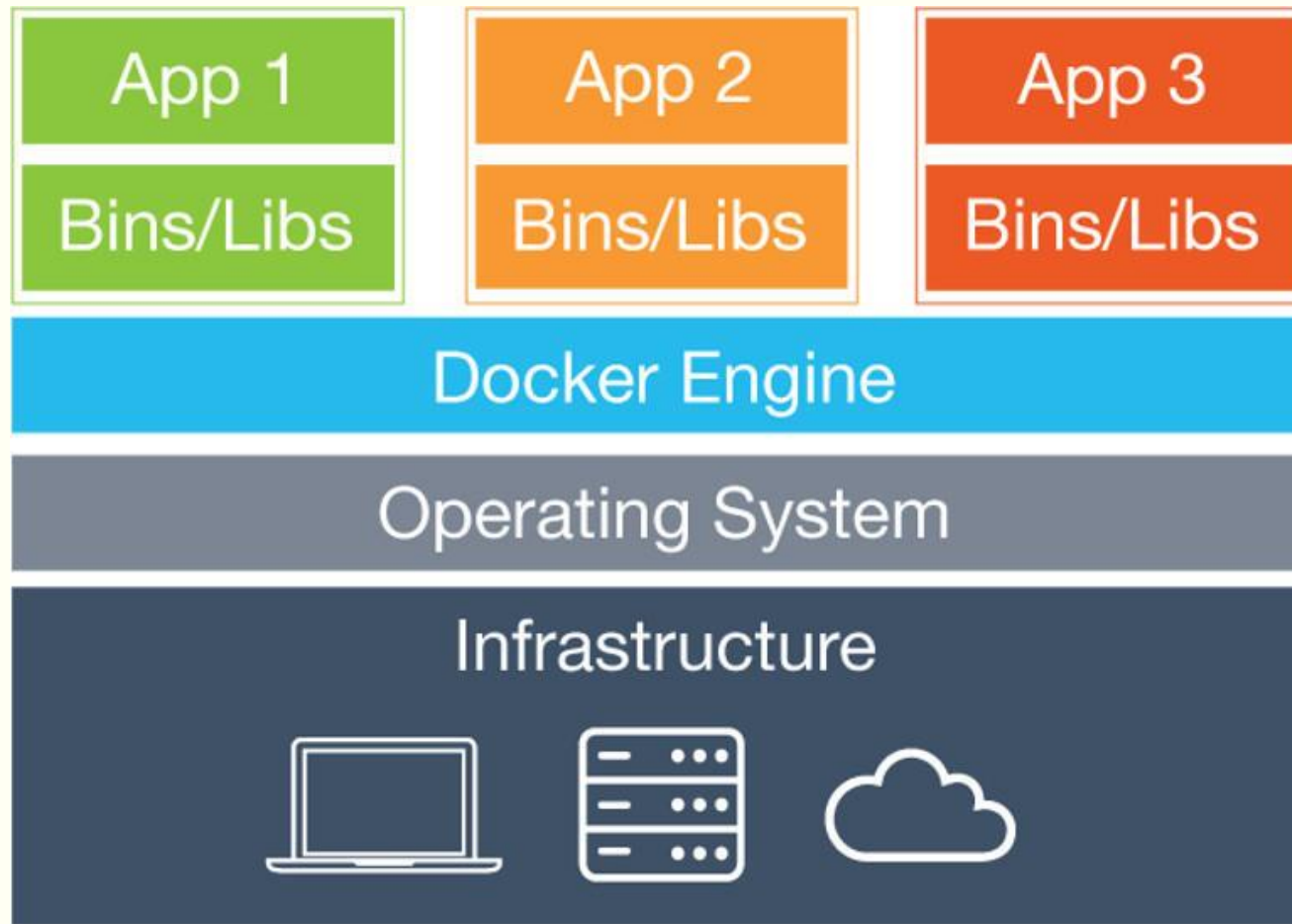
- Annoncé par Google en 2014. Sa première version sort en 2015
- Il arrête et redémarre pas moins des millions de containers chaque semaine
- Des services comme Gmail, Search, Apps ou Map tournent dans des conteneurs gérés par Kubernetes

Définition d'un container

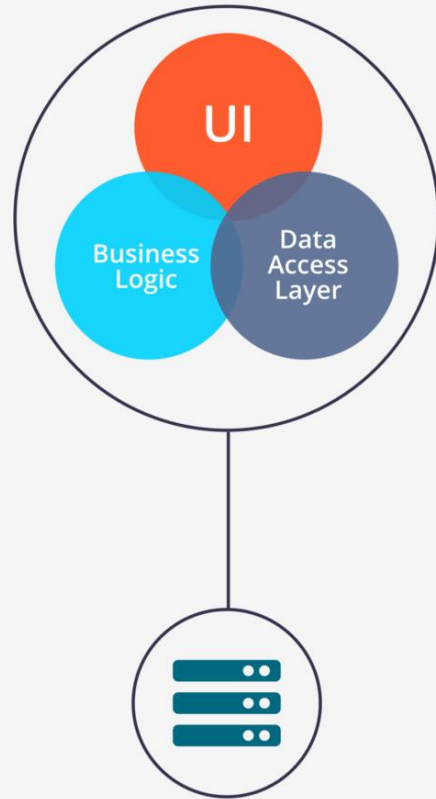
Un conteneur, c'est un environnement d'exécution complet comprenant :

- Un microservice (application)
- Ses dépendances
- Ses bibliothèques et autres fichiers binaires
- Fichiers de configuration
- Le tout est packagé dans ce que l'on nomme chez docker par exemple : **une image**

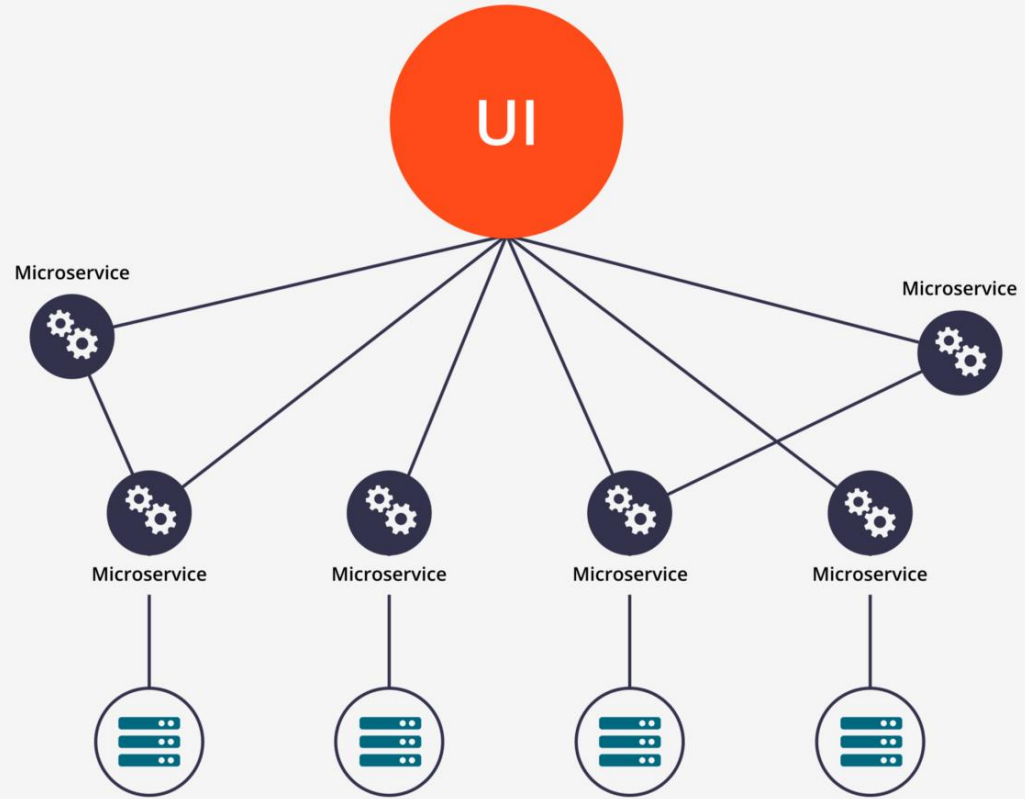
Définition d'un container



Architecture monolithique vs micro-service



Monolithic Architecture



Microservice Architecture

Architecture micro-service

- 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

Définition d'un container

Docker Engine

- Docker runtime
- La machine qui fabrique et lance les images

Docker Hub

- Un catalogue en ligne pour trouver ou stocker ses images dans un repository public ou privé
- Permet le stockage d'image en ligne

Définition d'un container

Avantages :

- Une taille relativement petite (parfois quelques dizaines de mégaoctets).
- Un conteneur peut démarré presque instantanément
- Ils peuvent être instanciés de manière quasi-immédiat lorsqu'ils sont nécessaires et disparaissent lorsqu'ils ne sont plus nécessaires, libérant ainsi des ressources
- Isolation
- Déploiement multiplateforme
- Maintenabilité et évolutivité du socle applicatif

Introduction à l'orchestrateur de conteneurs

- Manipuler quelques conteneurs sur des environnements de développement est une tâche facile.
- Lorsqu'il s'agit de faire passer ces conteneurs en production de nombreuses questions se posent :
 - Comment gérer les dysfonctionnements ?
 - Comment gérer les déploiements et leurs emplacements ?
 - Comment gérer le scaling ?
 - Comment gérer les mises à jours ?
 - Comment gérer la communication entre les conteneurs ?
 - Comment gérer le stockage nécessaire à la persistance des données ?
 - Comment gérer les secrets et la configuration ?
 - etc.

Tout gérer de manière manuelle et sans surcouche au système de conteneurs n'est pas viable, maintenable et pérenne.

Introduction Kubernetes

- Kubernetes est un système de gestion de conteneurs et plus particulièrement un orchestrateur. Il est, à l'origine, un projet Google disponible depuis juin 2014.
- Il est le résultat de la réécriture en Go du système Borg (et d'Omega son successeur) que Google utilise en interne pour gérer son infrastructure.
- Kubernetes a été versé à la Cloud Native Computing Foundation (CNCF) qui a pour vocation de s'assurer de la bonne exécution des applications dans des environnements cloud.
- On retrouve dans cette fondation des produits tels que Prometheus, Fluentdb, Envoy, ...

Introduction Kubernetes

Kubernetes est open source, il permet :

- la planification de containers dans un cluster de machines
- de lancer plusieurs containers dans une machine physique ou VM
- Redémarrer un container éteint pour plusieurs raisons pour garantir la disponibilité
- Lancer un container dans un node spécifique
- Déplacer un container d'un node à un node voisin par exemple lors d'une maintenance

Introduction Kubernetes

- Kubernetes n'est pas uniquement destiné au Cloud mais à tout type d'infrastructure. Un de ses points fort étant de permettre de faire de **l'Infra As Code**.
- Depuis la version 1.10 (Mars 2018), Kubernetes prend plus de distance avec Docker et peut gérer un ensemble de technologies de conteneurs qui vont respecter la norme Container Runtime Interface (Docker, RKT, LXD, ...).
- En 2018, Kubernetes représentait 51% du marché des orchestrateurs contre 11% pour Docker Swarm et 4% pour Mesos (Cf. [Docker Usage Report 2018](#)).

Définition d'un orchestrateur de containers

- Kubernetes (k8s en abrégé) est un orchestrateur de conteneurs initialement développé par Google en 2015 puis offert à la CNF (Cloud Native Computing Foundation).
- La solution est développée avec le langage Go (aussi à l'origine de Google).
- Il existe d'autres orchestrateurs de containers telles que :
 - **Swarm** : l'orchestrateur natif de Docker
 - **Mesos** : Peut faire tourner des container Docker et autres
- Grande modularité : les changements sont intégrables et modifiables

Les concepts de base

- **Cluster**: ensemble de **nodes**
- **Nodes**: machine physique ou VM sur laquelle tourne des **Pods**
- **Pods**: ensemble de **containers** qui partage le réseau et le stockage
- **Replicas**: nombre d'instances d'un **Pod**
- **Service**: regroupement d'instances d'un **Pod**
- **ReplicaSet**: assure que les réplicas spécifiés sont actifs
- **Deployment**: définit l'état désiré d'une application

Communication avec le cluster

- En utilisant le binaire kubectl
- En utilisant l'interface web de management (optionelle)

Communication avec le cluster : kubectl

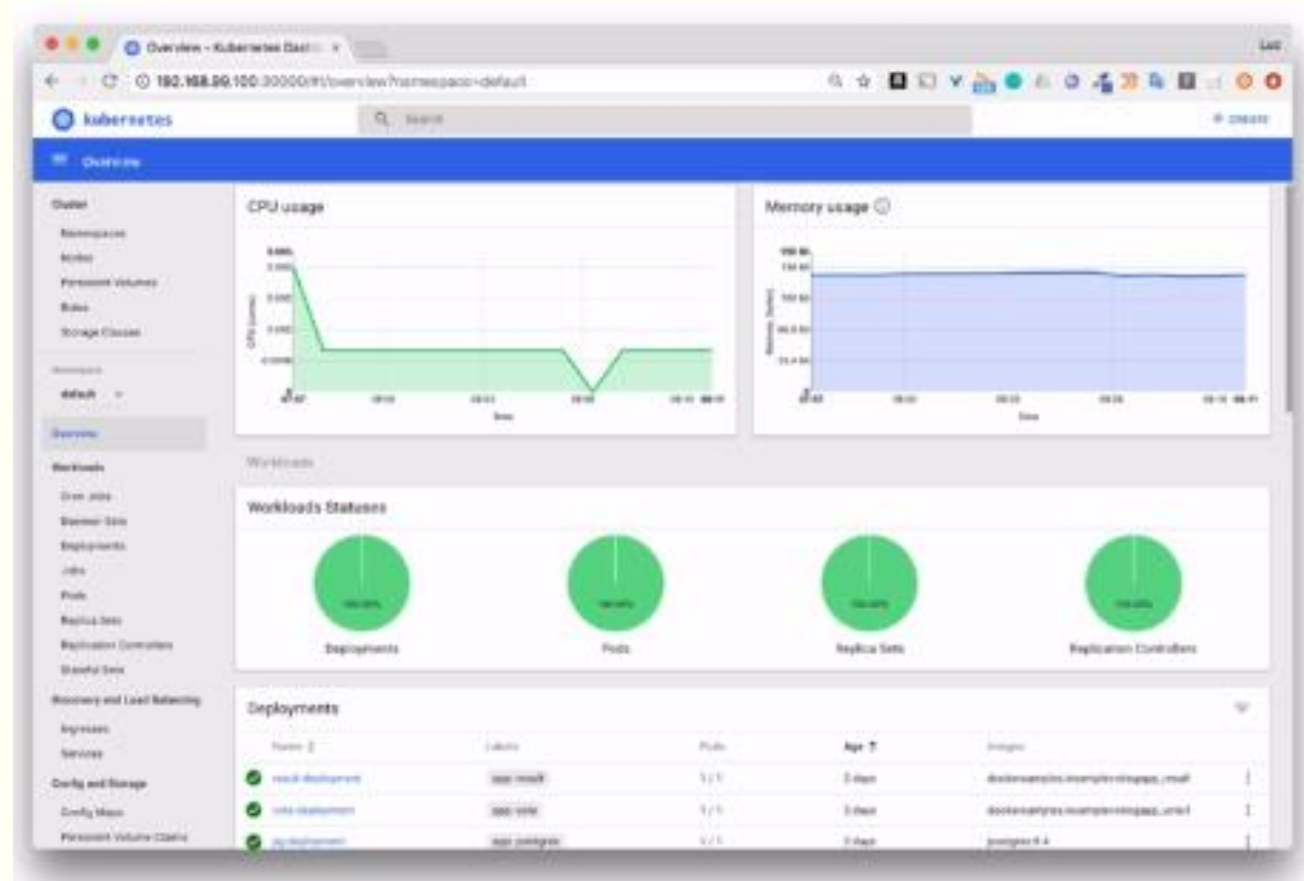
- Utilitaire en ligne de commande
- *kubectl [command] [TYPE] [NAME] [flags]*

```
$ kubectl create -f specification.yaml  
  
$ kubectl run --image=nginx:1.12.2 www  
  
$ kubectl get pods,deploy,svc
```

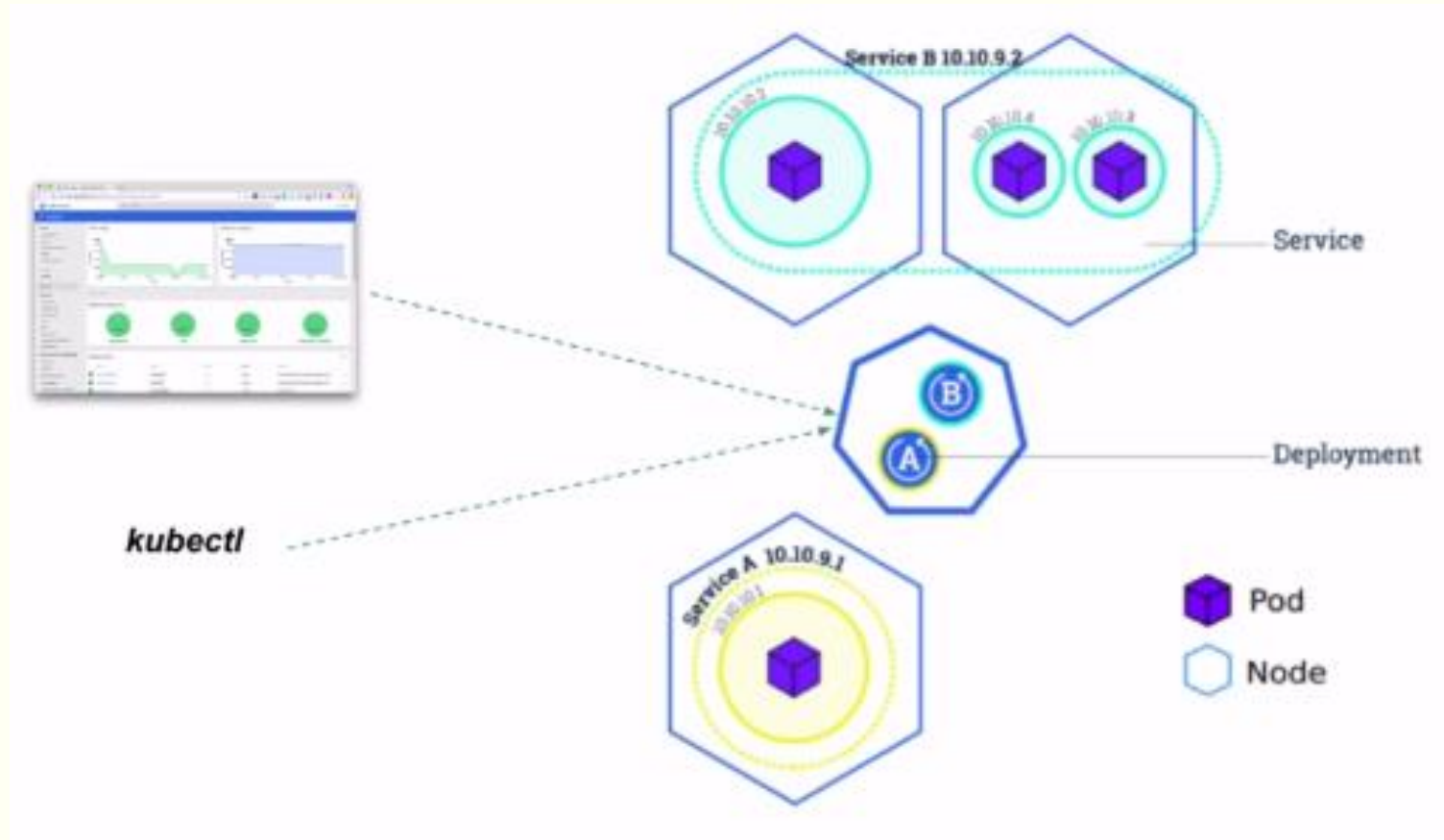
- Installation sur Mac, Linux, windows
 - <https://kubernetes.io/docs/tasks/tools/install-kubectl/>

Communication avec le cluster : interface web

- Dashboard donnant une vision globale du cluster



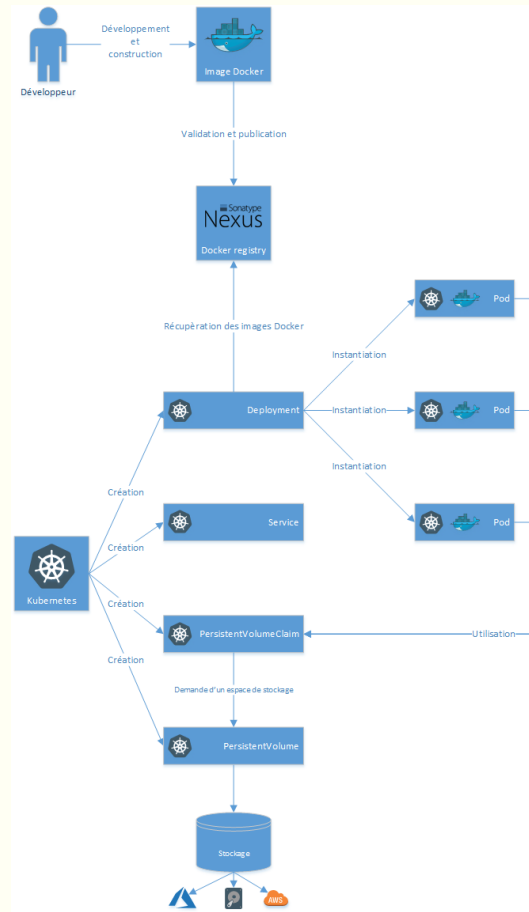
Vue d'ensemble



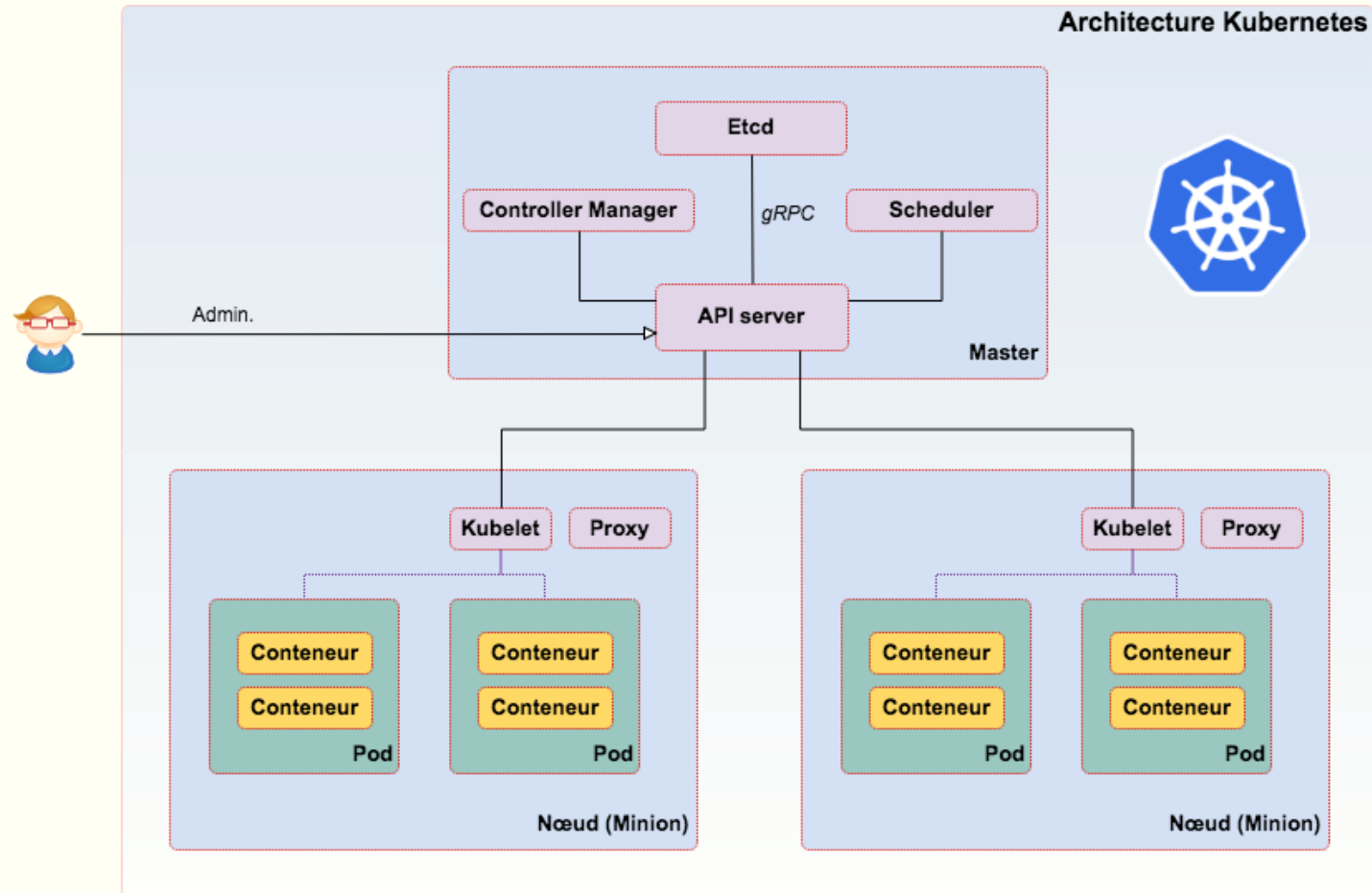
Comment Kubernetes répond aux questions posées en introduction ?

- Les **Deployments** répondent aux problématiques :
 - ✓ de dysfonctionnement en assurant que le nombre de répliquas actifs demandé soit toujours respecté.
 - ✓ de déploiement sur des emplacements spécifiques (node) avec la notion de nodeSelector
 - ✓ de scaling en ajustant le nombre de répliquas en fonction des besoins
 - ✓ de mise à jour en spécifiant la stratégie adéquate et en utilisant les mécanismes de rollback
- Les **Services** répondent aux problématiques de communication avec les conteneurs.
- Les **PersistentVolumes** et **PersistentVolumeClaims** répondent aux problématiques de persistance des données.
- Les ConfigMaps répondent aux problématiques de configuration en sortant ces éléments liés aux environnements (préproduction, production, etc.) des conteneurs.
- Enfin, la notion de **Secrets** permet de garder le contrôle sur les données sensibles de type mot de passe, clé d'API, etc.

Exemple concret



Architecture



Principaux composants

- Dans un cluster Kubernetes, on distingue le nœud **master** qui va piloter les nœuds **workers** (appelé « **minions** » ou « nodes »).

Master

- Prend les décisions importantes liées à la **gestion du cluster** dans le **monitoring**, la **planification** et la **détection d'événement nouveaux** provenant du cluster et pouvant aboutir, par exemple, au démarrage d'un nouveau « pod » lorsque le « champ réplikat » est insatisfait

Composant Node

- Chaque nœud contient les services nécessaires pour faire tourner des « pods » qui chacune regroupe un ou plusieurs containers.

Principaux composants

Master, serveur maître qui exécute les services suivants :

- **API Server** : interface centrale qui permet d'exécuter les commandes sur le cluster.
- **Etcd** : stocke les fichiers de configurations du cluster (stockage clé/valeur distribué).
- **Controller-manager** : regroupe un ensemble de contrôleurs en charge de vérifier :
 - l'état des nœuds,
 - le bon nombre de réplicas de Pods dans le cluster,
 - lier les services aux Pods,
 - gérer les droits d'accès aux différents espaces de noms utilisés dans le cluster.
- **Scheduler** : s'occupe d'assigner un nœud disponible à un Pod.
- **Service DNS** : intégré à Kubernetes, les conteneurs utilisent ce service dès leur création pour la résolution de nom à l'intérieur du cluster.

Principaux composants

Sur chaque nœud (minion) :

- **Kubelet** : en charge du bon fonctionnement des conteneurs sur le nœud (communique avec le serveur maître)
- **Kube-proxy** : gère le trafic réseau et les règles définies. Permet d'accéder aux autres conteneurs pouvant se trouver sur d'autres nœuds.
- **Pods** : unités éphémères d'exécution d'applications Stateless. Il est composé d'un ensemble de conteneurs et de volumes interconnectés.
- **Fluentd** : se charge de transférer les logs vers les serveurs maîtres.

KUBERNETES:KUBELET

- Service principal de Kubernetes
- Permet à Kubernetes de s'auto configurer :
 - Surveille un dossier contenant les manifests (fichiers YAML des différents composants de K8s).
 - Applique les modifications si besoin (upgrade, rollback).
- Surveille l'état des services du cluster via l'API server (kubeapiserver).
- Dossier de manifest sur un nœud master :

```
ls /etc/kubernetes/manifests/  
kube-apiserver.yaml kube-controller-manager.yaml kube-proxy.yaml kube-scheduler.yaml
```

KUBERNETES:KUBELET

- Exemple du manifest kube-proxy :

```
apiVersion: v1
kind: Pod
metadata:
  name: kube-proxy
  namespace: kube-system
  annotations:
    rkt.alpha.kubernetes.io/stage1-name-override: coreos.com/rkt/stage1-fly
spec:
  hostNetwork: true
  containers:
  - name: kube-proxy
    image: quay.io/coreos/hyperkube:v1.3.6_coreos.0
    command:
    - /hyperkube
    - proxy
    - --master=http://127.0.0.1:8080
    - --proxy-mode=iptables
  securityContext:
```

KUBERNETES:KUBE-APISERVER

- Les configurations d'objets (Pods, Service, RC, etc.) se font via l'API server
- Un point d'accès à l'état du cluster aux autres composants via une API REST
- Tous les composants sont reliés à l'API server
- Il gère la montée en charge en déployant des instances supplémentaires de manière horizontale en cas de besoin.

KUBERNETES:KUBE-SCHEDULER

- Planifie les ressources sur le cluster
- En fonction de règles implicites (CPU, RAM, stockage disponible, etc.)
- En fonction de règles explicites (règles d'affinité et antiaffinité, labels, etc.)
- Ce composant affecte la disponibilité, la performance et capacité du système.
 - ✓ Il va par exemple, observer la création de « pod » pour ensuite l'assigner à un nœud donné.

KUBERNETES:KUBE-PROXY

- Responsable de la publication de services
- Utilise iptables
- Route les paquets à destination des PODs et réalise le load balancing TCP/UDP

KUBERNETES:KUBE-CONTROLLER-MANAGER

- Boucle infinie qui contrôle l'état d'un cluster
- Effectue des opérations pour atteindre un état donné
- De base dans Kubernetes : replication controller, endpoints controller, namespace controller et serviceaccounts controller

Kubernetes : Minikube

- **Minikube** exécute un cluster Kubernetes à un nœud à l'intérieur d'une machine virtuelle sur votre ordinateur.
- Il convient aux personnes qui souhaitent essayer Kubernetes ou développer par-dessus.
- Son installation nécessite au préalable un hyperviseur, Kubectl et le Docker Engine. Il démarre avec la commande : **minikube start**

Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Installation de minikube



MODULE 2

Les objets: Pod

Les objets Kubernetes

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, ...)

Les objets Kubernetes

Différentes catégories : une même structure de base

- **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
```

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

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

Les objets: Pod

- Un pod est un élément composé généralement d'un conteneur, une configuration réseaux et une configuration sur le stockage.
- Les **pods** doivent être considérés comme jetable, ils sont instanciés et détruits au cours de la phase de run d'une application.
- Ensemble logique composé de un ou plusieurs conteneurs
- Les conteneurs d'un pod fonctionnent ensemble (instanciation et destruction) et sont orchestrés sur un même hôte

Les objets: Pod

- Les conteneurs partagent certaines spécifications du POD :
 - La stack IP (network namespace)
 - Inter-process communication (PID namespace)
 - Volumes
- C'est la plus petite unité orchestrable dans Kubernetes

Les objets: Pod

- Les PODs sont définis en YAML comme les fichiers docker-compose :

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

Les objets: Pod

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

Les objets: Pod

- **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

Les objets: Pod

- 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 du type de l'élément, ici nous définissons un Pod

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

Les objets: Pod

- 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

Les objets: Pod

- 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

```
# 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 évènements survenus lors du lancement du pod

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: {}
```

Container pour le serveur wordpress

Container pour la base de données mysql

Montage du volume dans le container mysql

Définition d'un volume: répertoire sur la machine hôte

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
```

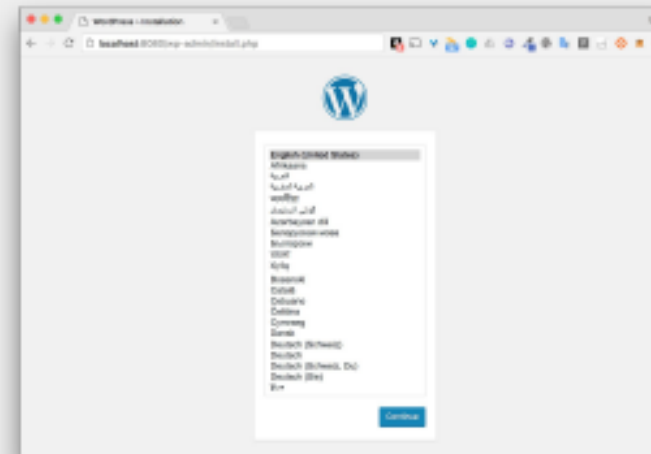
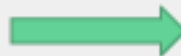


Pod avec plusieurs containers

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

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

# 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
```



Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Exercice : Cycle de vie

TP

Exercice : Pod avec plusieurs containers

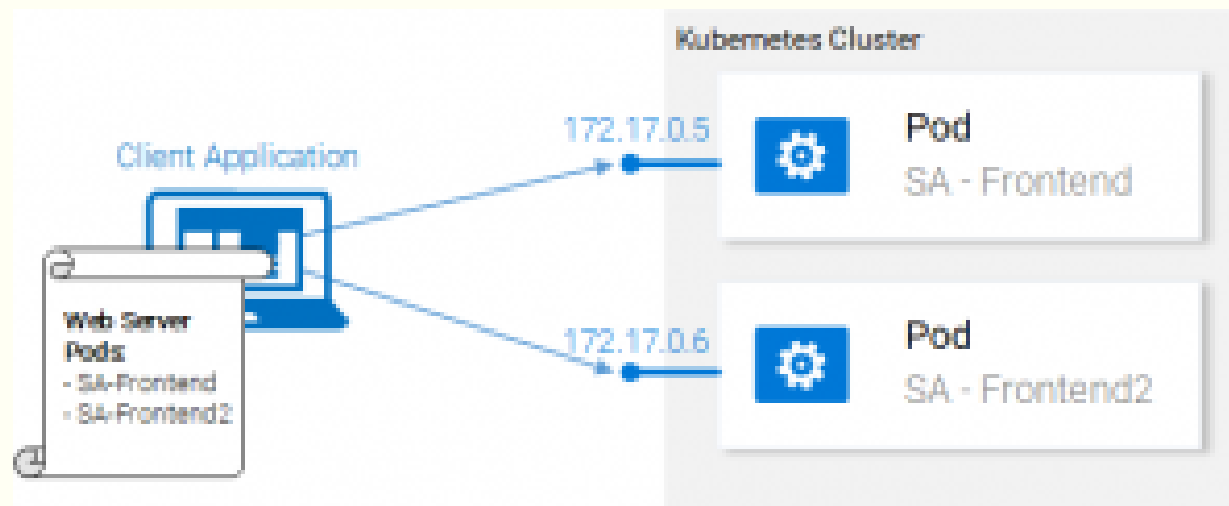


MODULE 3

Les objets - Service

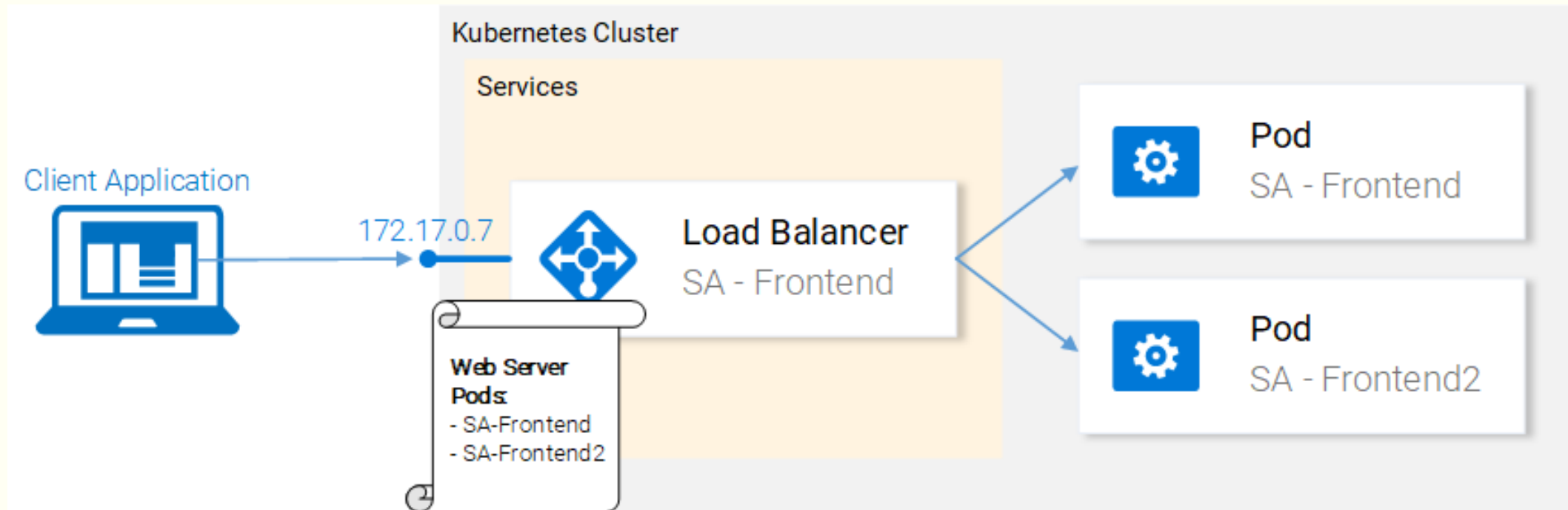
Les objets - Service

- Pour gérer le load balancing et la communication avec les pods, k8s introduit la notion de service comme point unique de communication avec plusieurs pods.
- Les pods étant « mortels », il n'est pas possible de se baser directement sur leurs IPs.



Client maintaining a list of IP addresses

Les objets - Service



- Kubernetes Service maintaining IP addresses

Présentation

- 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
 - via SERVICE_NAME
 - via SERVICE_NAME.NAMESPACE (vi Pod sur un autre namespace)
- NodePort : exposition vers l'extérieur
- LoadBalancer : intégration avec un Cloud Provider
 - AWS, GCE, Azure, ...
- ExternalName : défini un alias vers un service extérieur au cluster

Cycle de vie

- Lancement d'un Service
 - *\$ kubectl create -f SERVICE_SPECIFICATION.yaml*
- Description d'un Service
 - *\$ kubectl describe svc SERVICE_NAME*
- Suppression d'un Service
 - *\$ kubectl delete svc SERVICE_NAME*

Les objets - Service

KUBERNETES: LABELS

- Système de **clé/valeur**
- Organiser les différents objets de Kubernetes (PODs, Services, etc.) d'une manière cohérente qui reflète la structure de l'application
- Corréler des éléments de Kubernetes : par exemple un service vers des PODs

Les objets - Service

Exemple de label :

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

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
```

```
metadata:
```

```
  name: vote-service
```

```
spec:
```

```
  selector:
```

```
    app: vote
```

Indique les Pods que le service va regrouper
(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
```

```
    targetPort: 80
```

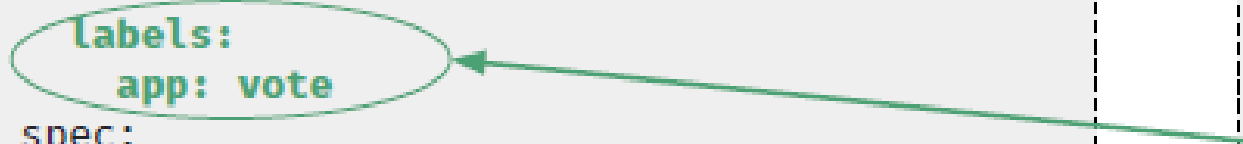
Le service expose le port 80 dans le cluster

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
  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
```

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
```

```
    targetPort: 80
```

```
    nodePort: 31000
```

Le service expose le port 80 dans le cluster

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
```

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

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
```

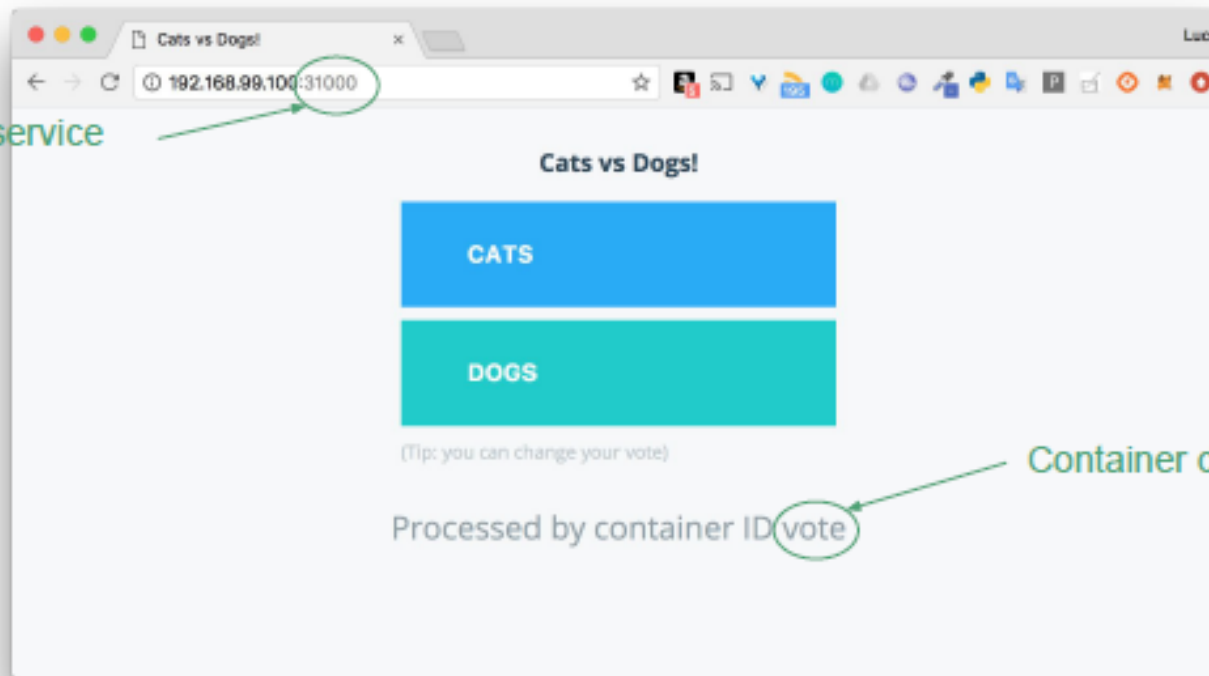
```
    nodePort: 31000
```

Service accessible depuis le port 31000 de chaque node

Spécification : exemple de type NodePort

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

Port exposé par le service



Container défini dans le Pod

Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Exercice : Service de type ClusterIP

TP

Exercice : Pod avec plusieurs containers



MODULE 4

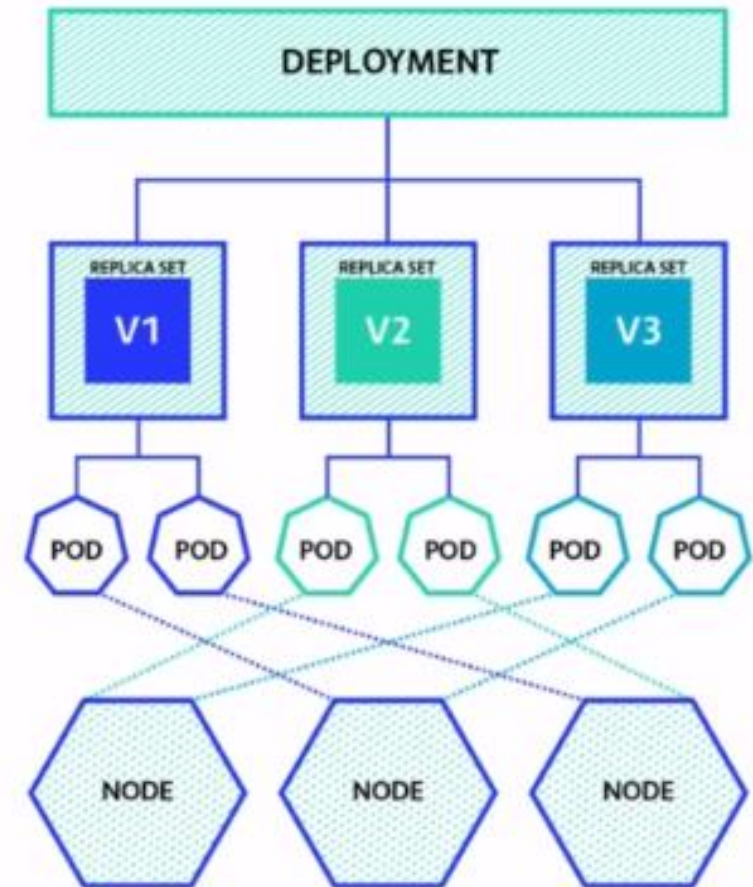
Les objets - Deployment

Les objets - Deployment

- Permet d'assurer le fonctionnement d'un ensemble de PODs
- Un **deployment** définit la configuration des pods, le nombre de répliquas souhaités, la stratégie de mise à jour des pods, etc.
- C'est grâce au **deployment** que le nombre de pods actifs demandé est maintenu sur un environnement.
- En cas de défaillance d'un pod, il est supprimé et un nouveau pod est instancié.
- Souvent combiné avec un objet de type service

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”
- Gère les mises à jour d'une application
 - rollout / rollback / scaling
 - Création d'un nouveau ReplicaSet lors d'une mise à jour de l'application
- 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:
```

```
  name: vote-deploy
```

Le nom "vote-deploy" est donné au deployment

```
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
```

```
  selector:
```

```
    matchLabels:
```

```
      app: vote
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: vote
```

```
    spec:
```

```
      containers:
```

```
        - name: vote
```

```
          image: instavote/vote
```

```
          ports:
```

```
            - containerPort: 80
```

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

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
```

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

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-gl9ps	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

Kubectl run

- Création d'un Deployment pour manager les containers sous-jacents
- `$ kubectl run --help`

```
$ 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 crée la nouvelle
- rolling update
 - release graduelle de la nouvelle version
- blue/green
 - ancienne et nouvelle version déployées ensemble
 - mise à jour du trafic via le Service exposant l'application
- 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

- Déclenché si changement sous la clé *.spec.template*
- Mise à jour graduelle de l'ensemble des Pods
- Paramètres pour contrôler le processus de mise à jour
 - `maxUnavailable`
 - `maxSurge`

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 <code>vote-deploy-584c4c76db</code> to 3
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled up replica set <code>vote-deploy-585cd89dd4</code> to 1
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled down replica set <code>vote-deploy-584c4c76db</code> to 2
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled up replica set <code>vote-deploy-585cd89dd4</code> to 2
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled down replica set <code>vote-deploy-584c4c76db</code> to 1
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled up replica set <code>vote-deploy-585cd89dd4</code> to 3
Normal	ScalingReplicaSet	17m	deployment-controller	Scaled down replica set <code>vote-deploy-584c4c76db</code> to 0
Normal	ScalingReplicaSet	16m	deployment-controller	Scaled up replica set <code>vote-deploy-64f888dd55</code> to 1
Normal	ScalingReplicaSet	16m	deployment-controller	Scaled down replica set <code>vote-deploy-585cd89dd4</code> to 2
Normal	ScalingReplicaSet	16m (x4 over 16m)	deployment-controller	(combined from similar events): Scaled down replica set <code>vote-deploy-585cd89dd4</code> to 0

Mise à jour d'un Deployment : rolling update

- `--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"
```

Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Exercice : Exposition à l'extérieur d'un cluster

TP

Exercice : Rolling update