



FORMATION KUBERNETES

2 jours

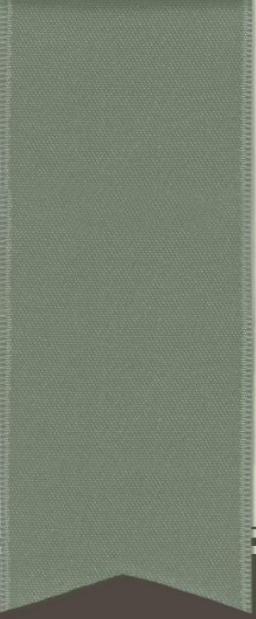


kubernetes



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 5: Les objets - Secret**
- **Module 6: Les objets - ConfigMap**
- **Module 7: Les objets - Namespace**
- **Module 8: Les objets - Ingress**
- **Module 9: Les objets - Volume**
- **Module 10: Les objets - Helm**



MODULE 1

Introduction à Kubernetes

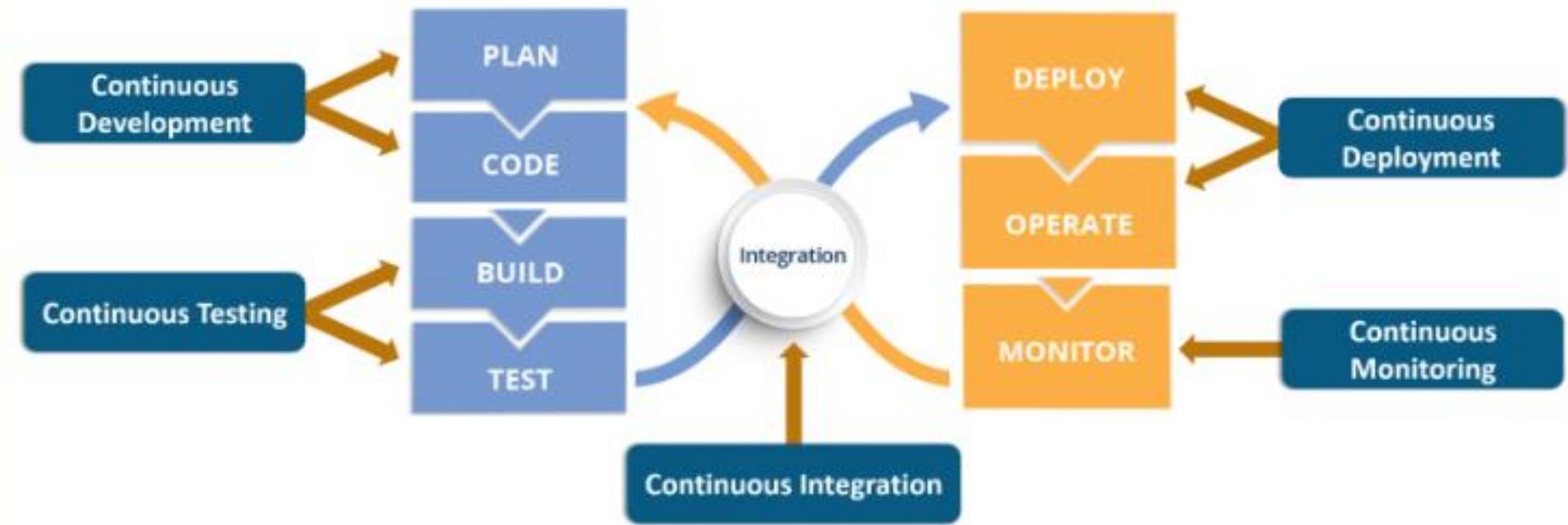


kubernetes

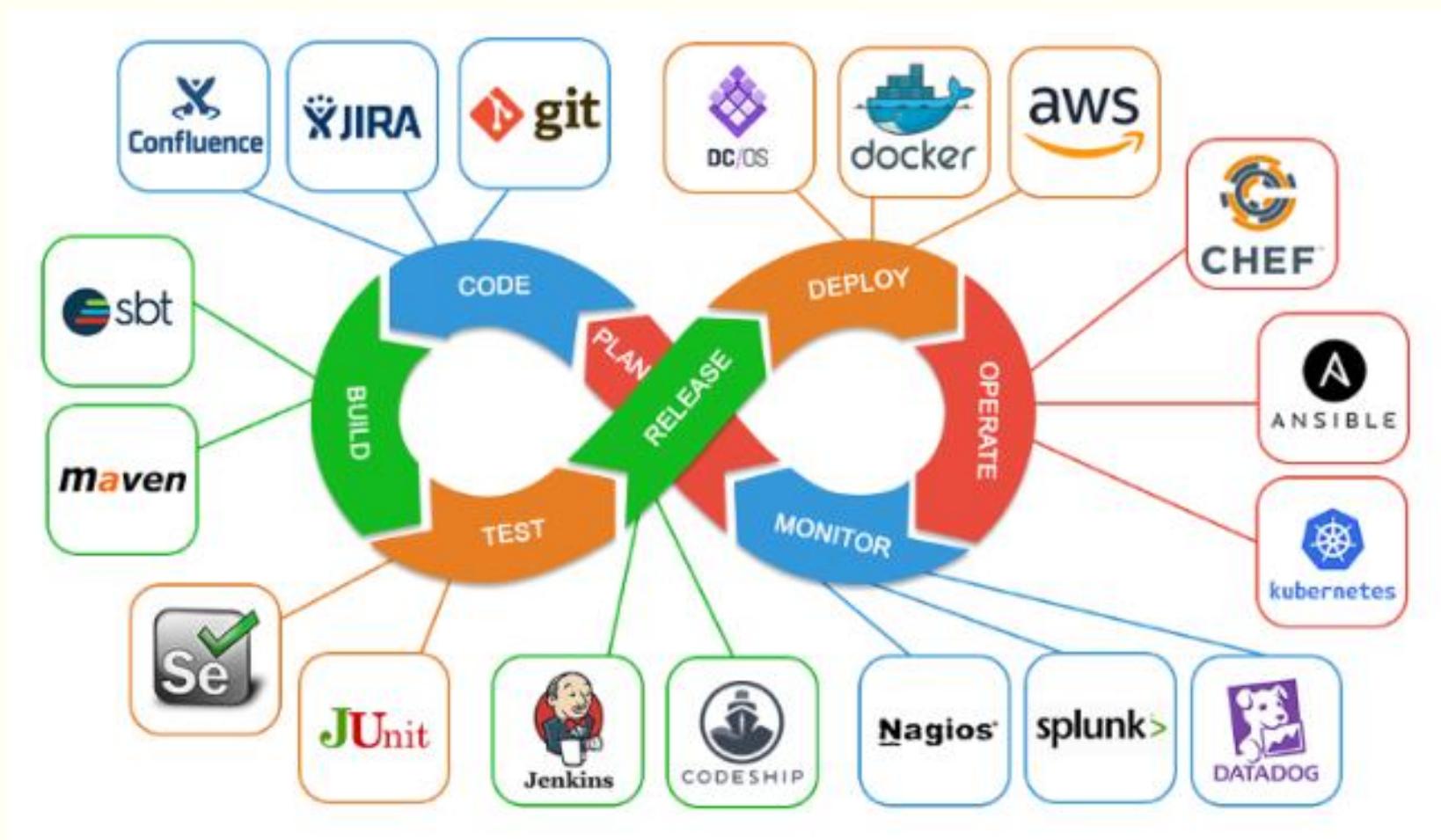
Introduction à Kubernetes

- 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

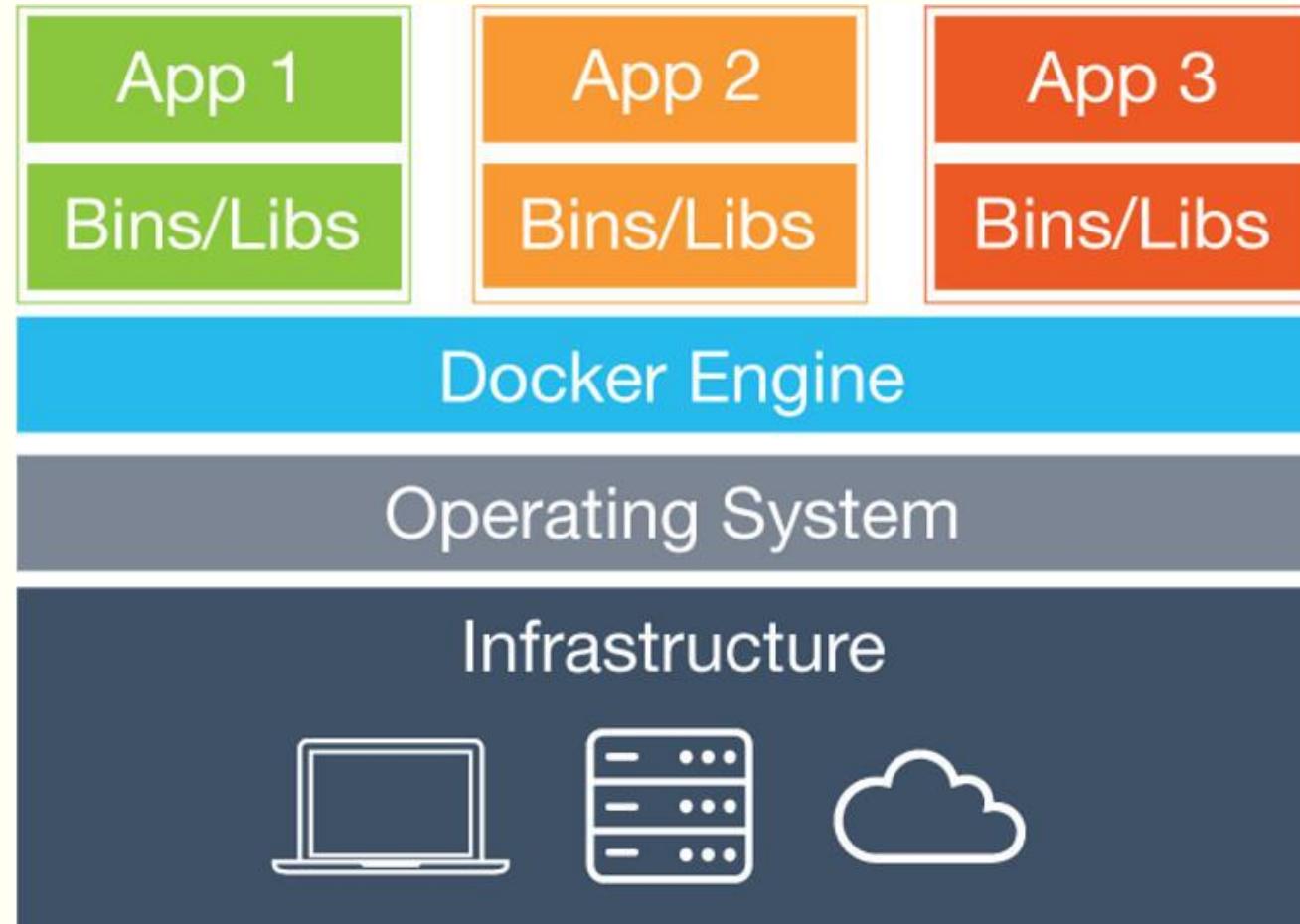
- Annoncé par Google en 2014. Sa première version sort en 2015
- Il arrête et redémarre pas moins des millions de containers chaque semaine
- Des services comme Gmail, Search, Apps ou Map tournent dans des conteneurs gérés par Kubernetes

Définition d'un container

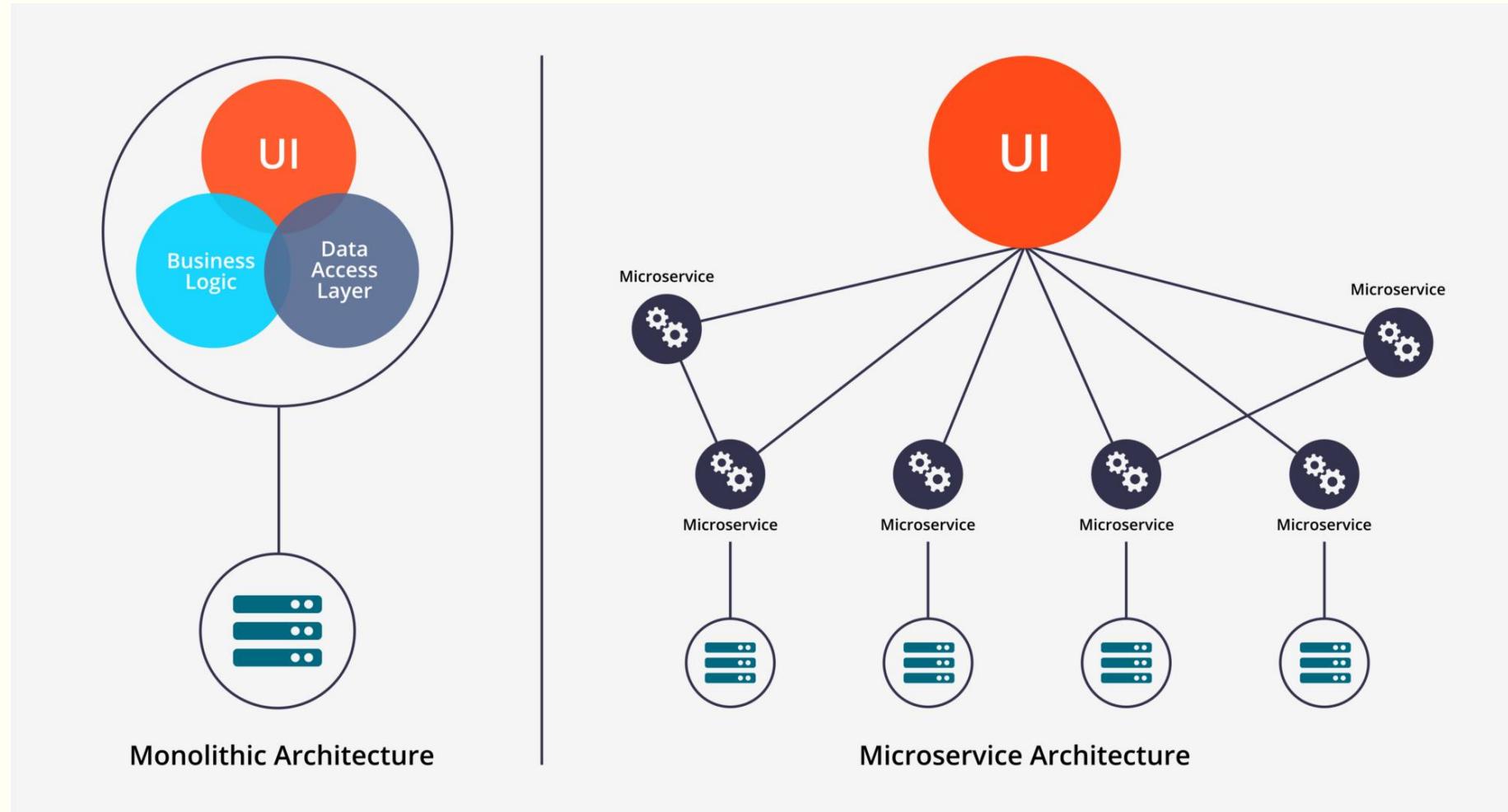
Un conteneur, c'est un environnement d'exécution complet comprenant :

- 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



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 conteneurs 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**
- **ReplicatSet:** assure que les réplicas spécifiés sont actifs
- **Deployment:** défini l'état désiré d'une application

Communication avec le cluster

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

Communication avec le cluster : kubectl

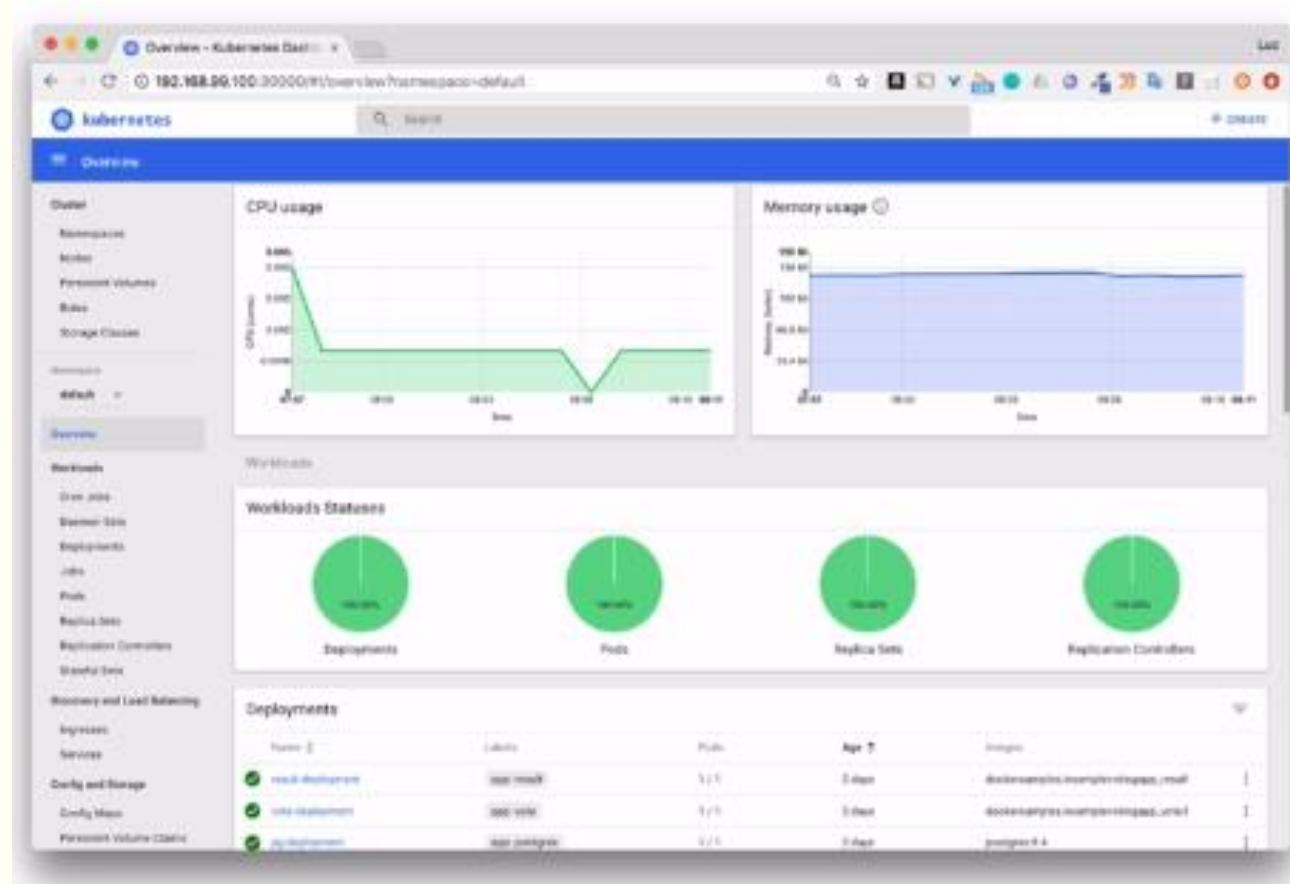
- Utilitaire en ligne de commande
- *kubectl [command] [TYPE] [NAME] [flags]*

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

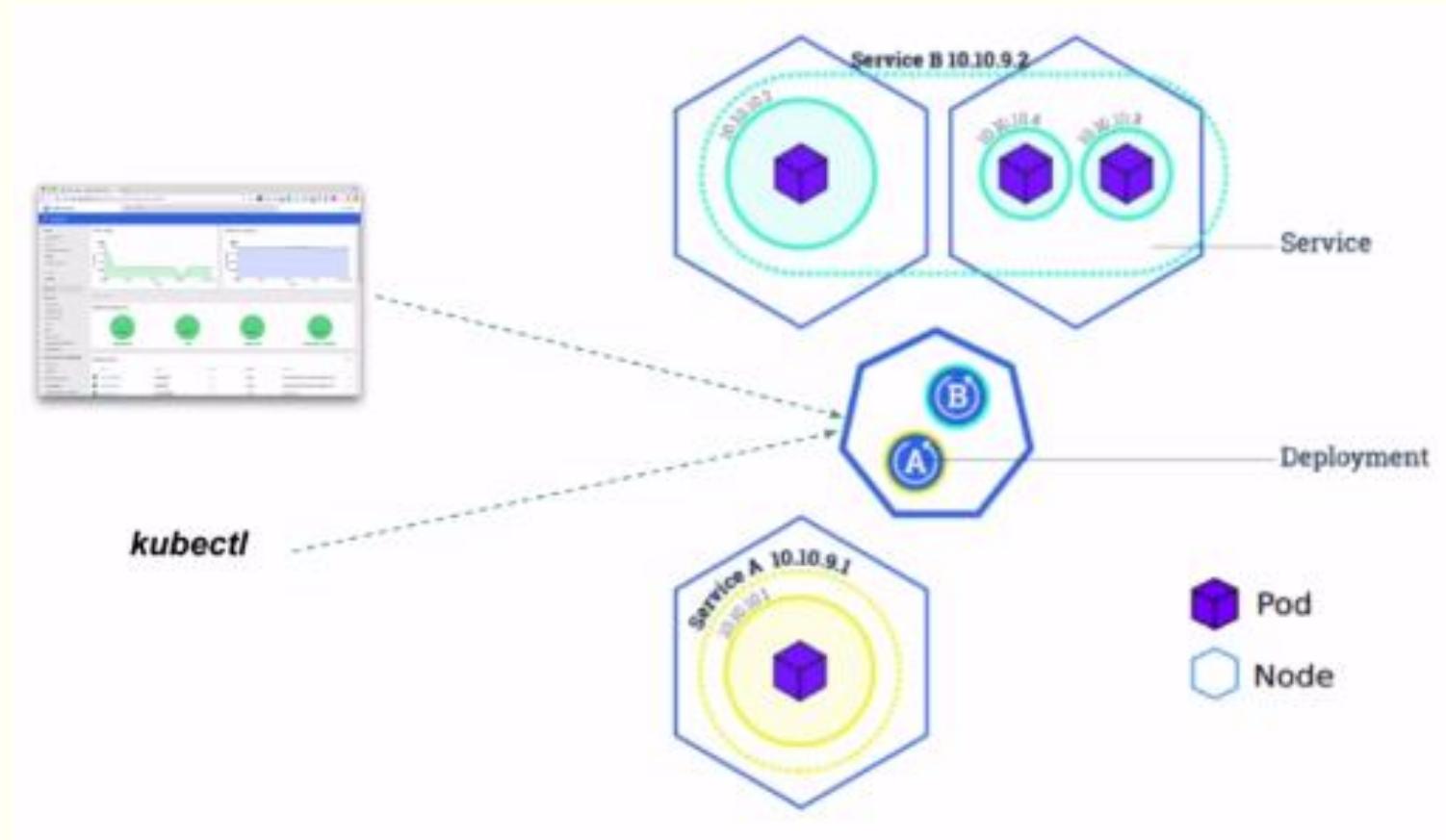
- Installation sur Mac, Linux, windows
 - <https://kubernetes.io/docs/tasks/tools/install-kubectl/>

Communication avec le cluster : interface web

- Dashboard donnant une vision globale du cluster



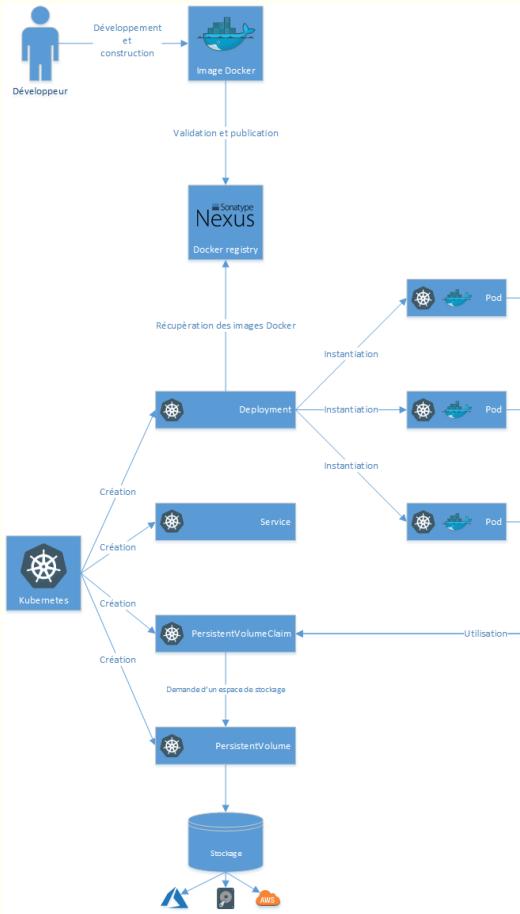
Vue d'ensemble



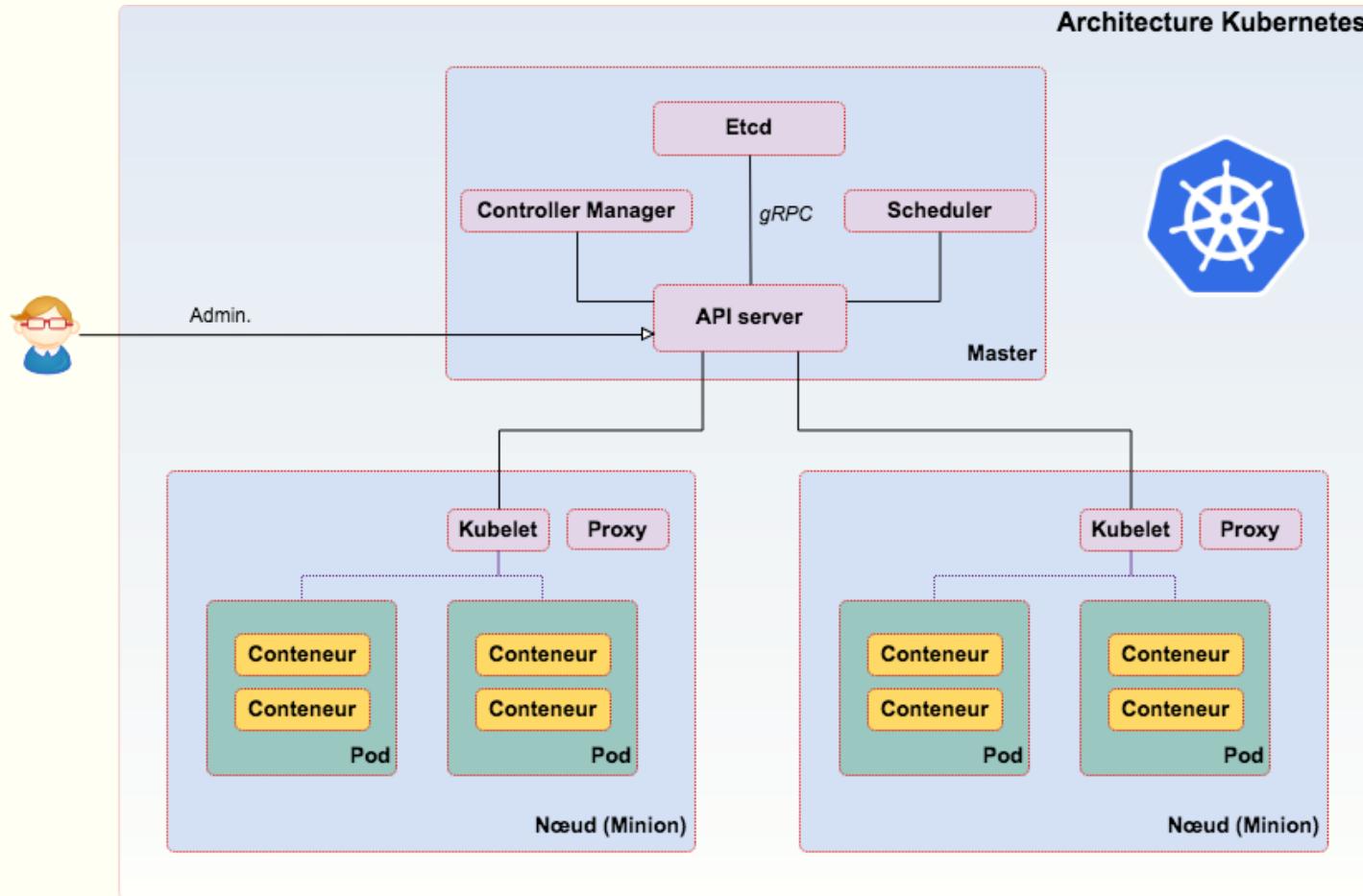
Comment Kubernetes répond aux questions posées en introduction ?

- Les **Deployments** répondent aux problématiques :
 - ✓ de dysfonctionnement en assurant que le nombre de répliques actifs demandé soit toujours respecté.
 - ✓ de déploiement sur des emplacements spécifiques (node) avec la notion de nodeSelector
 - ✓ de scaling en ajustant le nombre de répliques 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 nouveau** provenant du cluster et pouvant aboutir, par exemple, au démarrage d'un nouveau « pod » lorsque le « champ **réplicat** » 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-schedule
```

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**

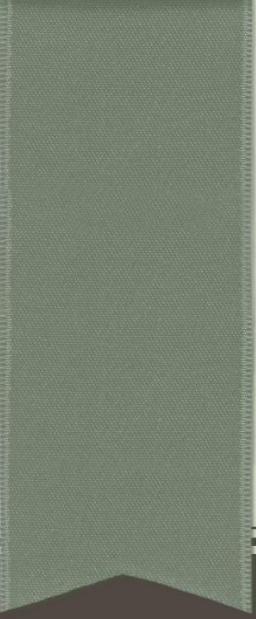
Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Installation de minikube



MODULE 2

Les objets: Pod



kubernetes

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 événements 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 persistence des données de la base
 - type **emptyDir** : associé au cycle de vie du Pod

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

Pod avec plusieurs containers

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

Pod avec plusieurs containers

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

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: {}
```

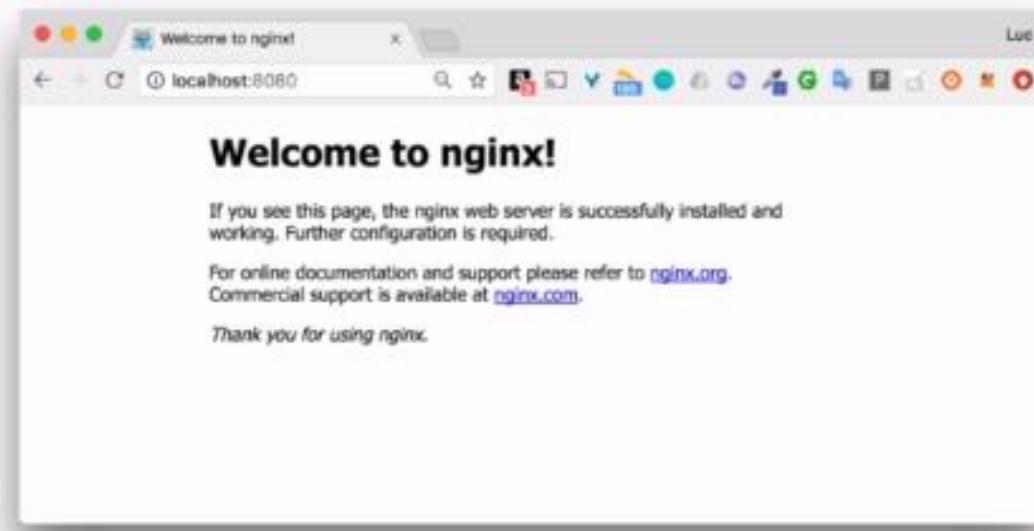
Montage du volume dans le container mysql

Définition d'un volume: répertoire sur la machine hôte

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
```

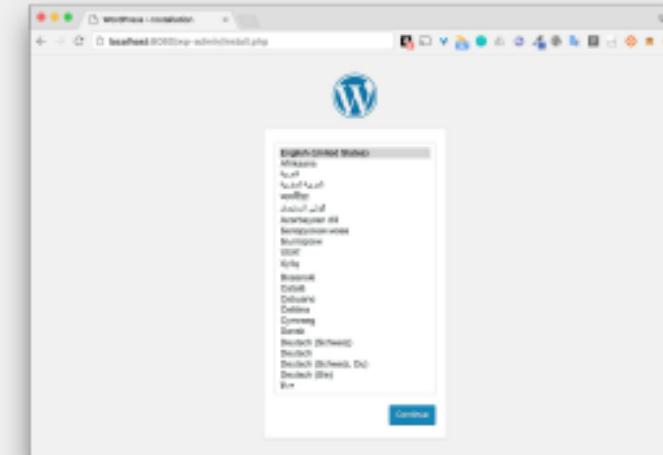


Pod avec plusieurs containers

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

# Liste des Pod présent
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   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
```



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

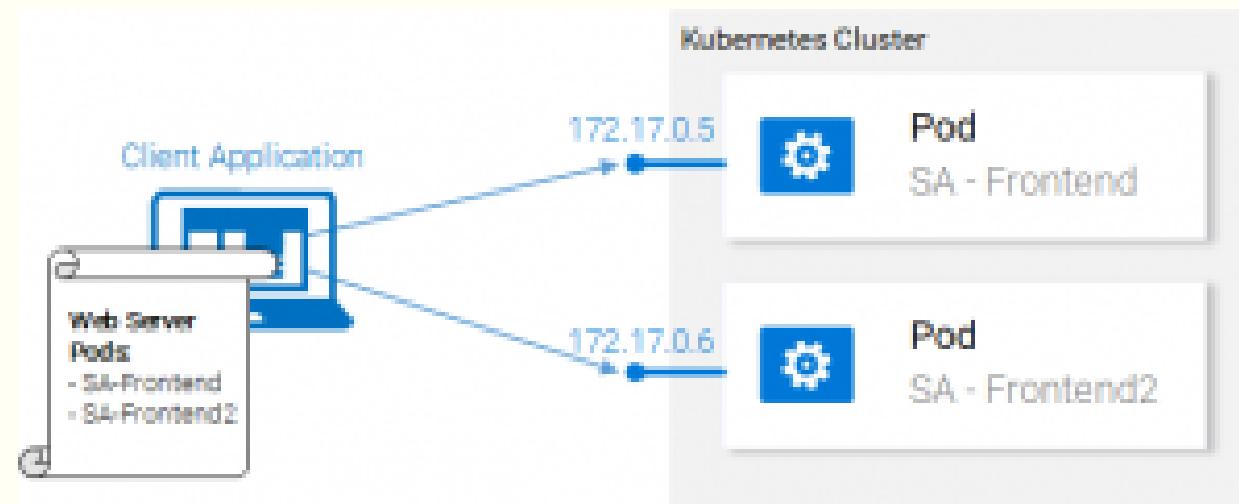
Les objets - Service



kubernetes

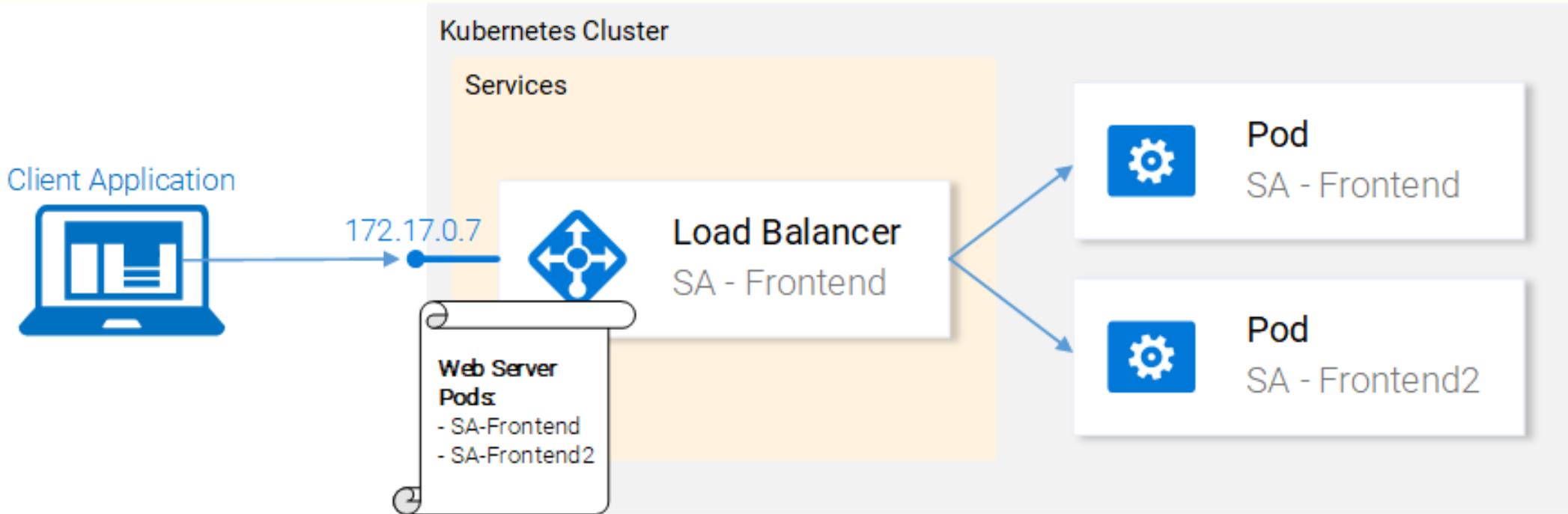
Les objets - Service

- 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 (via Pod sur un autre namespace)
- NodePort : exposition vers l'extérieur
- LoadBalancer : intégration avec un Cloud Provider
 - AWS, GCE, Azure, ...
- ExternalName : définit 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
  type: ClusterIP      (les Pods ayant le label "app: vote")
  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
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          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

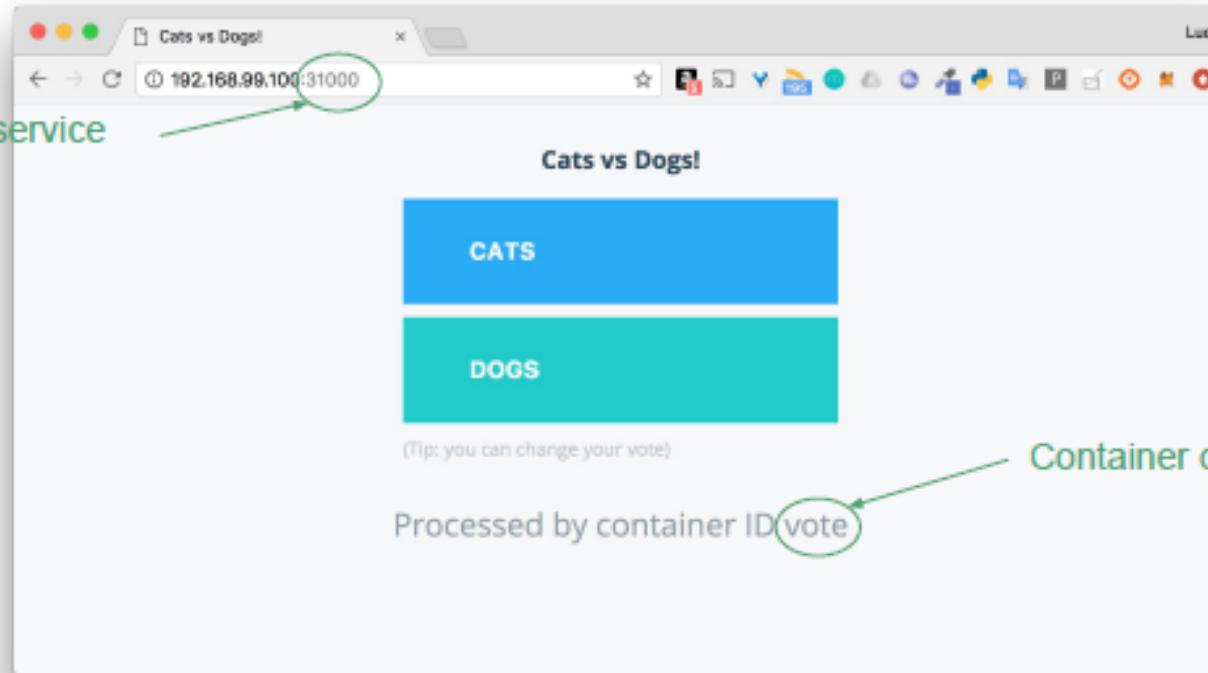
```
$ cat vote-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31000
```

Service accessible depuis le port 31000 de chaque node

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



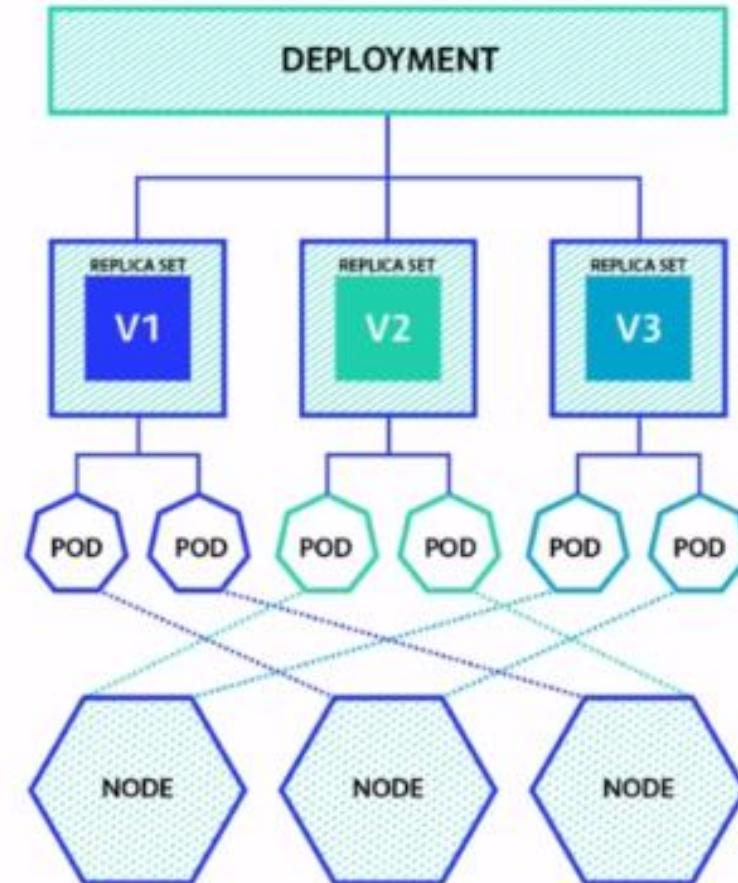
kubernetes

Les objets - Deployment

- Permet d'assurer le fonctionnement d'un ensemble de PODs
- Un **deployment** définit la configuration des pods, le nombre de répliques 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 défini un “état souhaité”
 - spécification d'un Pod et du nombre de répliques 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
```



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
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” est le type de la resource considérée

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
Le nom "vote-deploy" est donné au deployment
```

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

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

ReplicaSet actif

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 : 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



MODULE 5

Les objets - Secret



kubernetes

Utilisation

- Objet permettant la protection des données sensibles
 - clé privée, mot de passe, chaîne de connexion à un service tiers, ...
- Évite de définir ces informations dans les images ou spécifications
- Donne un meilleur contrôle sur leur utilisation
- Crée par un utilisateur ou par le système (clés d'accès à l'API)
- Utilisé dans un Pod
 - via un volume monté dans un ou plusieurs containers
 - via kubelet lors de la récupération de l'image
- Stocké dans etcd (encryption expérimentale)

Différents types

- generic
 - Secret créé à partir d'un fichier, d'un répertoire ou d'une valeur littérale
- docker-registry
 - Secret utilisé pour l'authentification à un Docker registry
- TLS
 - Secret utilisé pour la gestion des clés (PKI)

Secret de type generic : création

```
# Création de fichiers contenant les credentials de connexion à un service tiers
$ echo -n "admin" > ./username.txt
$ echo -n "45fe3efa" > ./password.txt

# Création de l'objet Secret avec kubectl
$ kubectl create secret generic service-creds --from-file=./username.txt --from-file=./password.txt
secret "service-creds" created

# Création depuis des valeurs littérales
$ kubectl create secret generic service-creds2 \
  --from-literal=username=admin --from-literal=password=45fe3efa
secret "service-creds2" created

# Liste des Secrets présents
$ kubectl get secrets
  NAME          TYPE           DATA   AGE
  default-token-ps46p  kubernetes.io/service-account-token  3      82d
  service-creds   Opaque          2      3s
  service-creds2   Opaque          2      3m
```

Secret généré par kubernetes pour permettre aux services internes d'accéder à l'API server

Secret de type generic : création

```
# Inspection du contenu de l'objet
$ kubectl describe secrets/service-creds
Name:          service-creds
Namespace:     default
Labels:        <none>
Annotations:   <none>

Type:  Opaque

Data
=====
password.txt: 8 bytes
username.txt: 5 bytes
```

```
# Détails de la spécification de l'objet
$ kubectl get secrets service-creds -o yaml
apiVersion: v1
data:
  password.txt: NDVmZTNlZmE=
  username.txt: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: 2018-02-28T22:37:47Z
  name: service-creds
  namespace: default
  resourceVersion: "589455"
  selfLink: /api/v1/namespaces/default/secrets/service-creds
  uid: ffd1b396-1cd7-11e8-ba7f-080027f0e385
type: Opaque
```

Encodé en base64

```
$ echo "NDVmZTNlZmE=" | base64 -D
45fe3efa
$ echo "YWRtaW4=" | base64 -D
admin
```

Secret de type generic : création

```
# Conversion de la donnée sensible en base64
$ echo -n "mongodb://admin:45fe3efa@mgserv1.org/mgmt" | base64
bW9uZ29kYjovL2FkbWlu0jQ1ZmUzZWZhQG1nc2VydjEub3JnL21nbXQ=

# Spécification de l'objet Secret
$ cat mongo-creds.yaml
apiVersion: v1
kind: Secret
metadata:
  name: mongo-creds
data:
  mongoURL: bW9uZ29kYjovL2FkbWlu0jQ1ZmUzZWZhQG1nc2VydjEub3JnL21nbXQ=

# Création de l'objet Secret
$ kubectl create -f ./mongo-creds.yaml
secret "mongo-creds" created
```

Secret de type generic : utilisation (volume 1)

```
$ cat pod-secret-volume-1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
    - name: alpine
      image: alpine
      command:
        - "sleep"
        - "10000"
      volumeMounts:
        - name: creds
          mountPath: "/etc/creds"
          readOnly: true
    volumes:
      - name: creds
        secret:
          secretName: mongo-creds
```



Montage du volume dans le container sur le point de montage spécifié

Définition d'un volume basé sur le Secret mongo-creds

Secret de type generic : utilisation (volume 1)

```
# Création du Pod alpine
$ kubectl create -f pod-secret-volume-1.yaml
pod "alpine" created

# Lancement d'un shell interactif dans le container alpine
$ kubectl exec -ti alpine -- sh
/ # cat /etc/creds/mongoURL
mongodb://admin:45fe3efa@mgserv1.org/mgmt
```

Secret de type generic : utilisation (volume 2)

```
$ cat pod-secret-volume-2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
    - name: alpine
      image: alpine
      command:
        - "sleep"
        - "10000"
      volumeMounts:
        - name: creds
          mountPath: "/etc/creds"
          readOnly: true
    volumes:
      - name: creds
        secret:
          secretName: service-creds
          items:
            - key: username.txt
              path: service/user
            - key: password.txt
              path: service/pass
```



Montage du volume dans le container sur le point de montage spécifié

Définition d'un volume basé sur le Secret service-creds
On précise les path relatifs ou les clés seront montées

Secret de type generic : utilisation (volume 2)

```
# Création du Pod alpine
$ kubectl create -f pod-secret-volume-2.yaml
pod "alpine" created

# Lancement d'un shell interactif dans le container api
$ kubectl exec -ti alpine -- sh
/ # cat /etc/creds/service/user
admin
/ # cat /etc/creds/service/pass
45fe3efa
```

Secret de type generic : utilisation (env)

```
$ cat pod-secret-env.yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
  - name: alpine
    image: alpine
    command:
    - "sleep"
    - "10000"
    env:
    - name: MONGO_URL
      valueFrom:
        secretKeyRef:
          name: mongo-creds
          key: mongoURL
```

Secret de type generic : utilisation (env)

```
# Création du Pod api
$ kubectl create -f pod-secret-env.yaml
pod "api" created

# Lancement d'un shell interactif dans le container api
$ kubectl exec -ti alpine -- sh
/ # env | grep MONGO
MONGO_URL=mongodb://admin:45fe3efa@mgserv1.org/mgmt
```

Secret de type docker-registry : création

```
$ kubectl create secret docker-registry registry-creds \
--docker-server=REGISTRY_FQDN --docker-username=USERNAME --docker-password=PASSWORD --docker-email=EMAIL

$ kubectl get secret registry-creds -o yaml
apiVersion: v1
data:
  .dockercfg: eyJhdXRocI6eyJodHR...QVWtRPSJ9fX0=
kind: Secret
metadata:
  creationTimestamp: 2018-03-02T22:26:14Z
  name: registry-creds
  namespace: default
  resourceVersion: "626790"
  selfLink: /api/v1/namespaces/default/secrets/registry-creds
  uid: b7b70613-1e68-11e8-ba7f-080027f0e385
type: kubernetes.io/dockercfg
```

Secret de type docker-registry : utilisation

```
# Pod utilisant une image privée
$ cat pod-private-image.yaml
apiVersion: v1
kind: Pod
metadata:
  name: private-image
spec:
  containers:
  - name: api
    image: my_private_image
  imagePullSecrets:
  - name: registry-creds

$ kubectl create -f pod-private-reg.yaml
pod "private-reg" created
```

Secret de type TLS : création

- Gestion des PKI
- Crée à partir d'un couple clé publique / clé privée

```
# Création du couple de clés
$ openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 365 -out cert.pem

# Création du Secret à partir des clés
$ kubectl create secret tls domain-pki --cert cert.pem --key key.pem

$ kubectl get secret domain-pki -o yaml
apiVersion: v1
data:
  tls.crt: LS0tLS1CRUdJT...GSUNBVEUtLS0tLQo=
  tls.key: LS0tLS1CRUdJT...TESBLRVktLS0tLQo=
kind: Secret
Metadata:
  name: domain-pki
  ...
type: kubernetes.io/tls
```

Secret de type TLS : utilisation

```
$ cat pod-secret-tls.yaml
apiVersion: v1
kind: Pod
metadata:
  name: proxy
spec:
  containers:
    - name: proxy
      image: nginx:1.12.2
      volumeMounts:
        - name: tls
          mountPath: "/etc/ssl/certs/"
  volumes:
    - name: tls
      secret:
        secretName: domain-pki
```



Montage du volume dans le container sur le point de montage spécifié

Définition d'un volume basé sur le Secret domain-pki

Secret de type TLS : utilisation

```
# Lancement du Pod
$ kubectl create -f pod-secret-tls.yaml
pod "proxy" created

# Lancement d'un shell interactif dans le container
$ kubectl exec -ti proxy -- sh
/ # ls /etc/ssl/certs
tls.crt  tls.key
```

Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Exercice



MODULE 6

Les objets - ConfigMap



kubernetes

ConfigMap : Utilisation

- Découplage d'une application 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
- Crée à partir d'un fichier, d'un répertoire, ou de valeurs littérales
- Contient une ou plusieurs paires de clé / valeur

Création à partir d'un fichier

```
$ cat nginx.conf
user www-data;
worker_processes 4;
pid /run/nginx.pid;
events {
    worker_connections 768;
}
http {
    server {
        listen *:8000;
        location / {
            proxy_pass http://localhost;
        }
    }
}

$ kubectl create configmap nginx-config --from-file=./nginx.conf
configmap "nginx-config" created
```

Création à partir d'un fichier

```
$ kubectl get cm nginx-config -o yaml
apiVersion: v1
data:
  nginx.conf: |
    user www-data;
    worker_processes 4;
    ...
kind: ConfigMap
metadata:
  creationTimestamp: 2018-03-03T15:35:46Z
  name: nginx-config
  namespace: default
  resourceVersion: "635910"
  selfLink: /api/v1/namespaces/default/configmaps/nginx-config
  uid: 8ac176fc-1ef8-11e8-ba7f-080027f0e385
```

← Le contenu du fichier de configuration nginx.conf est disponible sous la clé data de la ConfigMap

Création à partir d'un fichier d'environnement

```
# Fichier d'environnement constitué de couples key=value
$ cat app.env
log_level=WARN
env=production
cache=redis

# Création d'une ConfigMap à partir du fichier .env
$ kubectl create configmap app-config-env --from-env-file=./app.env
configmap "app-config-env" created

$ kubectl get cm app-config-env -o yaml
apiVersion: v1
data:
  cache: redis
  env: production
  log_level: WARN
kind: ConfigMap
metadata:
  creationTimestamp: 2018-03-04T14:45:09Z
  name: app-config-env
  namespace: default
  ...
  
```

Chaque couple clé:valeur du fichier d'environnement est défini sous la forme clé:valeur dans l'objet ConfigMap

Création à partir de valeurs littérales

```
# Utilisation de l'option --from-literal pour chaque couple clé=valeur
$ kubectl create configmap app-config-lit \
--from-literal=log_level=WARM \
--from-literal=env=production \
--from-literal=cache=redis
configmap "app-config-lit" created

$ kubectl get cm app-config-lit -o yaml
apiVersion: v1
data:
  cache: redis
  env: production
  log_level: WARM
kind: ConfigMap
metadata:
  creationTimestamp: 2018-03-04T14:49:51Z
  name: app-config-lit
  namespace: default
  ...
```

Utilisation dans un Pod : volume

```
$ cat pod-config-volume.yaml
apiVersion: v1
kind: Pod
metadata:
  name: www
spec:
  containers:
    - name: proxy
      image: nginx:1.12.2
      ports:
        - containerPort: 8000
      volumeMounts:
        - name: config
          mountPath: "/etc/nginx/"
    - name: api
      image: lucj/city:1.0
      ports:
        - containerPort: 80
  volumes:
    - name: config
      configMap:
        name: nginx-config
```



Montage du volume dans le container sur le point de montage spécifié

Définition d'un volume basé sur la ConfigMap nginx-config

Utilisation dans un Pod : volume

```
# Création du Pod www
$ kubectl create -f pod-config-volume.yaml
pod "www" created

# Lancement d'un shell interactif dans le container proxy du Pod www
$ kubectl exec -ti www --container proxy -- bash
root@www:/# cat /etc/nginx/nginx.conf
user www-data;
worker_processes 4;
pid /run/nginx.pid;
events {
    worker_connections 768;
}
http {
    server {
        listen *:8000;
        location / {
            proxy_pass http://localhost;
        }
    }
}
```

Utilisation dans un Pod : volume

```
# Lancement d'un shell interactif dans le container api du Pod www
$ kubectl exec -ti www --container api -- sh
/app # apk update && apk add curl
/app # curl localhost
{"message":"www suggests to visit Robjazpaw"}

# Lancement d'un shell interactif dans le container proxy du Pod www
$ kubectl exec -ti www --container proxy -- bash
root@www:/# apt-get update && apt-get install -y curl
root@www:/# curl localhost:8000
{"message":"www suggests to visit Tubogbaj"}
```

Utilisation dans un Pod : volume

```
# Lancement d'un shell interactif dans le container api du Pod www
$ kubectl exec -ti www --container api -- sh
/app # apk update && apk add curl
/app # curl localhost
{"message":"www suggests to visit Robjazpaw"}
```

Le container *api* écoute sur le port 80 à l'intérieur du Pod

```
# Lancement d'un shell interactif dans le container proxy du Pod www
$ kubectl exec -ti www --container proxy -- bash
root@www:/# apt-get update && apt-get install -y curl
root@www:/# curl localhost:8000
{"message":"www suggests to visit Tubogbaj"}
```

Le container *www* écoute sur le port 8000 à l'intérieur du Pod et formarde les requêtes au container *api*

Utilisation dans un Pod : variable d'environnement

```
$ cat pod-config-env.yaml
apiVersion: v1
kind: Pod
metadata:
  name: w3
spec:
  containers:
  - name: www
    image: nginx:1.12.2
    env:
      - name: LOG_LEVEL
        valueFrom:
          configMapKeyRef:
            name: app_config-lit
            key: log_level
      - name: CACHE
        valueFrom:
          configMapKeyRef:
            name: app-config-env
            key: cache
```

Utilisation dans un Pod : variable d'environnement

```
$ cat pod-config-env.yaml
apiVersion: v1
kind: Pod
metadata:
  name: w3
spec:
  containers:
    - name: www
      image: nginx:1.12.2
      env:
        - name: LOG_LEVEL
          valueFrom:
            configMapKeyRef:
              name: app_config-lit
              key: log_level
        - name: CACHE
          valueFrom:
            configMapKeyRef:
              name: app-config-env
              key: cache
```



La valeur d'une variable d'environnement est lue depuis la clé correspondante dans la ConfigMap

Utilisation dans un Pod : variable d'environnement

```
# Création du Pod www
$ kubectl create -f pod-config-env.yaml
pod "www" created

# Lancement d'un shell interactif dans le container proxy du Pod www
$ kubectl exec -ti w3 --container www -- bash
# / env
HOSTNAME=w3
NJS_VERSION=1.12.2.0.1.14-1~stretch
NGINX_VERSION=1.12.2-1~stretch
CACHE=production
LOG_LEVEL=WARM
...
```

Utilisation dans un Pod : variable d'environnement

```
# Création du Pod www
$ kubectl create -f pod-config-env.yaml
pod "www" created

# Lancement d'un shell interactif dans le container proxy du Pod www
$ kubectl exec -ti w3 --container www -- bash
# / env
HOSTNAME=w3
NJS_VERSION=1.12.2.0.1.14-1~stretch
NGINX_VERSION=1.12.2-1~stretch
CACHE=production
LOG_LEVEL=WARM
```

Les variables d'environnement ont les valeurs spécifiées dans la ConfigMap

Les Volumes Kubernetes

- Un volume Kubernetes n'est pas la même chose qu'un volume Docker.
- Dans Docker, un volume est simplement un dossier sur disque ou dans un autre conteneur.
- Jusque récemment, les cycles de vie n'étaient pas gérés et il s'agissait uniquement de volumes persistés sur disque local.
- A l'inverse, un volume Kubernetes a un cycle de vie défini.

Les Volumes Kubernetes

- Un volume Kubernetes est :
 - ✓ Un dossier, pouvant contenir des données
 - ✓ Accessible par tous les conteneurs d'un pod
- Les plugins de volume définissent :
 - ✓ Comment un dossier est mis en place
 - ✓ Le média qui le stocke
 - ✓ Le contenu du dossier
- La durée de vie d'un volume est celle d'un pod *ou plus long.*
- Le plus important est que Kubernetes supporte énormément de types de volumes.

Les plugins de volume Kubernetes

- Fonctionnent de la même façon que les volumes Docker pour les volumes hôte :
 - ✓ **EmptyDir** ~= volumes docker
 - ✓ **HostPath** ~= volumes hôte
- Support de multiples backend de stockage :
 - ✓ GCE : PD
 - ✓ AWS : EBS
 - ✓ glusterFS / NFS
 - ✓ Ceph
 - ✓ iSCSI

Persistent volume claim (PVC)

- Le principe est de séparer la déclaration d'un volume persistant de son pod.
- On déclare d'abord les volumes persistants avec un processus spécifique, puis on rattache le pod à un volume disponible avec le ***persistent volume claim***.
- Exemple PV
- Créons un volume persistant pv1 avec 10GB et un pv2 avec 100GB, voici la définition de pv2 :

```
1 # pv2.yaml
2 apiVersion: v1
3 kind: PersistentVolume
4 metadata:
5   name : mypv2
6 spec:
7   accessModes:
8     - ReadWriteOnce
9   capacity:
10    storage: 100Gi
11   persistentVolumeReclaimPolicy: Retain
12   gcePersistentDisk:
13     fsType: ext4
14     pdName: panda-disk2
15
```

Persistent volume claim (PVC)

- Et voici comment on le crée :

```
1 $ kubectl create -f pv1.yaml
2 persistentvolume "pv1" created
3 $ kubectl create -f pv2.yaml
4 persistentvolume "pv2" created
5 $ kubectl get pv
6   NAME      CAPACITY  ACCESSMODES  STATUS    CLAIM
7   pv1       10Gi      RWO         Available
8   pv2       100Gi     RWO         Available
9
```

Exemple PVC

- Maintenant que nous avons un volume persistant, nous pouvons le rattacher à un container avec PVC :

```
1 # pvc.yaml
2 apiVersion: v1
3 kind: PersistentVolumeClaim
4 metadata:
5   name: mypvc
6   namespace: testns
7 spec:
8   accessModes:
9     - ReadWriteOnce
10  resources:
11    requests:
12      storage: 100Gi
13
```

Exemple PVC

- Lorsqu'un PVC est créé, Kubernetes va rattacher le pod à un volume persistant disponible :

```
1 $ kubectl create -f pvc.yaml
2 persistentvolumeclaim "mypvc" created
3 $ kubectl get pv
4 NAME      CAPACITY   ACCESSMODES   STATUS      CLAIM
5 pv1       10Gi       RWO          Available
6 pv2       100Gi      RWO          Bound       testns/mypvc
```

Exemple PVC

- Vous pouvez configurer un PVC directement dans la déclaration d'un pod :

```
1 # pod.yaml
2 apiVersion: v1
3 kind: Pod
4 metadata:
5   name: sleepypod
6 spec:
7   volumes:
8     - name: data
9       persistentVolumeClaim:
10         claimName: mypvc
11 containers:
12   - name: sleepycontainer
13     image: gcr.io/google_containers/busybox
14     command:
15       - sleep
16       - "6000"
17     volumeMounts:
18       - name: data
19         mountPath: /data
20         readOnly: false
21
```

Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Démo



MODULE 7

Les objets - Namespace



kubernetes

KUBERNETES:NAMESPACES

- Fournissent une séparation logique des ressources par exemple :
 - ✓ Par utilisateurs
 - ✓ Par projet / applications
 - ✓ Autres...
- Les objets existent uniquement au sein d'un namespace donné
- Évitent la collision de nom d'objets

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

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

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

Création

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

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

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

Utilisation

- Pod avec namespace spécifié dans les metadata

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

Ce Pod sera déployé dans le namespace nommé development

Utilisation

```
# Lancement d'un Pod dans le namespace development
$ kubectl create -f nginx-pod-dev.yaml
pod "nginx" created

# Liste des Pods dans le namespace default
$ kubectl get po
No resources found.

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

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

Utilisation

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

# Liste des Deployments dans le namespace development
$ kubectl get deploy --namespace development
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
www        2          2          2           2           20s

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

Définition dans un context

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

← Le namespace **default** est utilisé

Définition dans un 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 ← development est le namespace utilisé
    user: minikube
    name: minikube
current-context: minikube
preferences: {}
```

*development est le namespace utilisé
dans le context minikube*

Définition dans un context

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

```
# Liste des Deployments
```

```
$ kubectl get deploy
```

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

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

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

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

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

```
No resources found.
```

Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Démo



MODULE 8

Les objets - Ingress



kubernetes

KUBERNETES: INGRESSRESOURCE

- Définition de règles de routage applicatives (HTTP/HTTPS)
- Traffic load balancing, SSL termination, name based virtual hosting
- Définies dans l'API et ensuite implémentées par un Ingress Controller



Ingress

- Ensemble de règles pour la connection aux services du cluster depuis internet
- Nécessite qu'un **Ingress Controller** soit déployé
 - container Docker avec un processus de contrôle
 - load balancer managé par Kubernetes
 - exemples : GCE, nginx, Traefik, HAProxy
- Différents cas d'usage
 - routage HTTP, ex: hôtes virtuels basés sur le nom
 - terminaison TLS
 - Load balancing
- Add-ons pour Minikube: “*\$ minikube addons enable ingress*”

Ingress

D'autres options pour exposer les services à l'extérieur

- Service de type NodePort
 - statique
 - load balancer externe pour le dispatch entre plusieurs nodes
- Service de type Load Balancer
 - nécessite l'utilisation d'un cloud provider (AWS, GCE, Azure, ...)

Ingress : routage par nom de domaine

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

Ingress : routage par nom de domaine

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

Objet présent dans l'API beta

Ingress : routage par nom de domaine

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

Ressource de type Ingress

Ingress : routage par nom de domaine

```
$ cat www-ingress-domain.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:                                     Définition du nom de la ressource
  name: www-domain
spec:
  rules:
    - host: www.example.com
      http:
        paths:
          - backend:
              serviceName: www
              servicePort: 80
```

Ingress : routage par nom de domaine

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

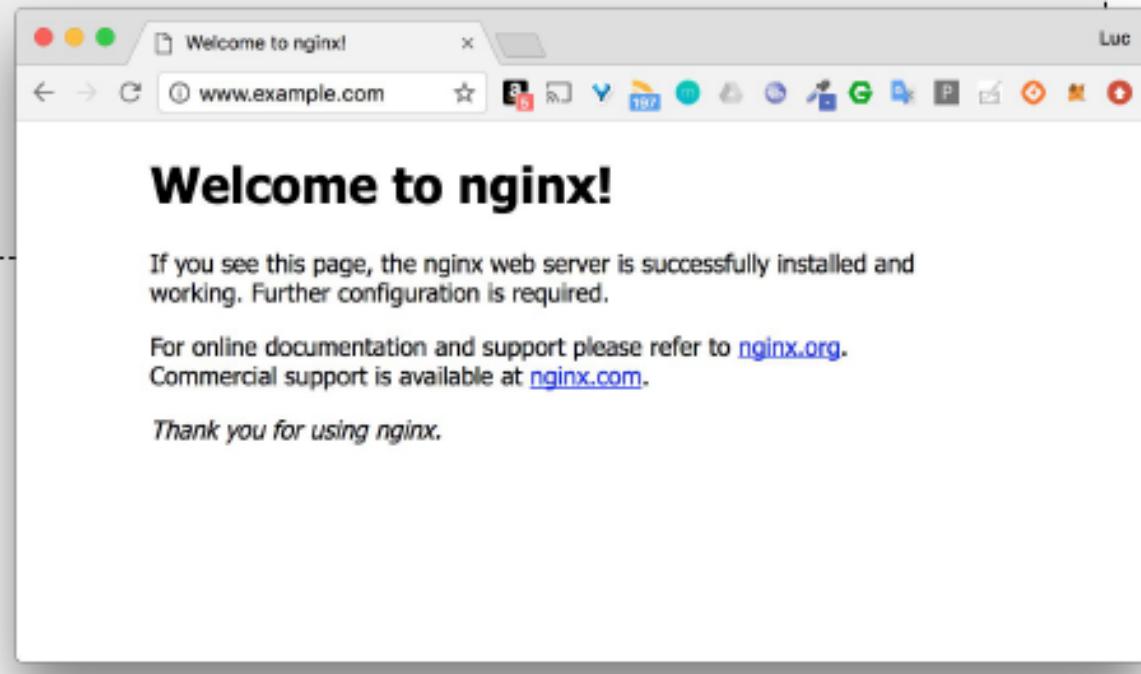
Les requêtes envoyées sur www.example.com seront forwardées sur le service www

Ingress : routage par nom de domaine

```
# Création d'un Deployment basé sur nginx
$ kubectl run www --image=nginx:1.12.2
deployment "www" created

# Exposition du Deployment via un Service
$ kubectl expose deployment www --port=80 --target-port=80
service "www" exposed

# Création de l'objet Ingress
$ kubectl create -f www-ingress-domain.yaml
ingress "www-domain" created
```



Ingress : routage via le path de la requête

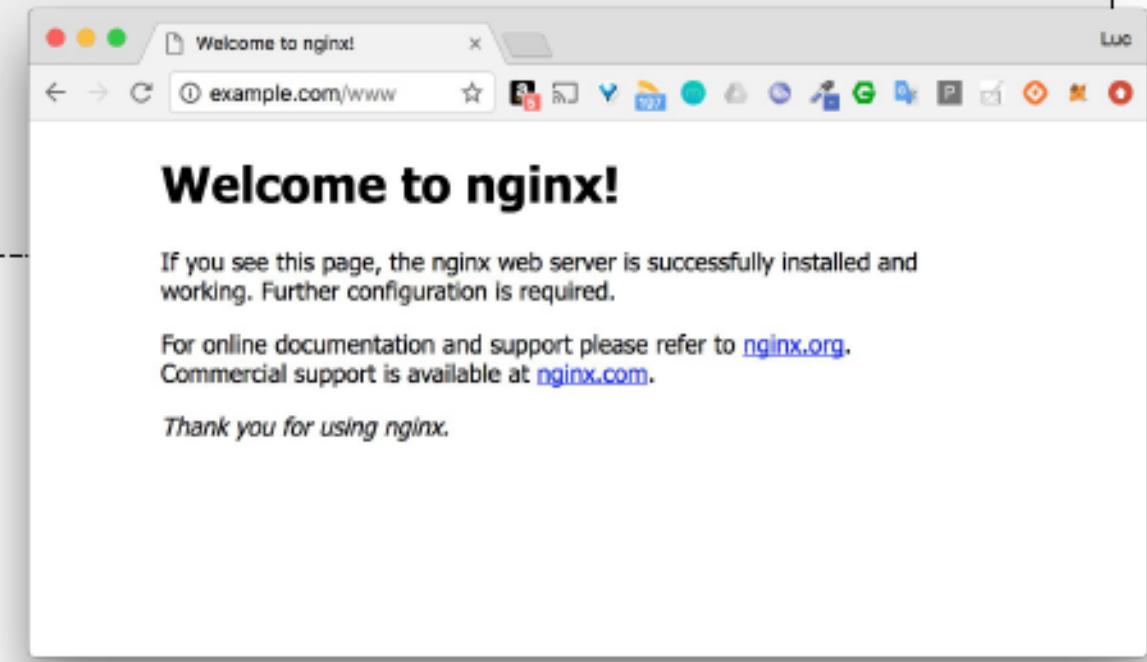
```
$ cat www-ingress-path.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  name: www-path
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /www
        backend:
          serviceName: w3
          servicePort: 80
```

Ingress : routage via le path de la requête

```
# Création d'un Deployment basé sur nginx
$ kubectl run w3 --image=nginx:1.12.2
deployment "w3" created

# Exposition du Deployment via un Service
$ kubectl expose deployment w3 --port=80 --target-port=80
service "w3" exposed

# Création de l'objet Ingress
$ kubectl create -f www-ingress-domain.yaml
ingress "www-domain" created
```



Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Démo



MODULE 9

Les objets - Volume



kubernetes

Volume

- Découple les données du cycle de vie d'un container
- Permet aux containers d'un même Pod de partager des données
- Type défini dans la spécification du Pod
 - emptyDir, configMap, Secret, nfs, awsElasticBlockStore, azureDisk, ...
 - <https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes>
 - plugins

Volume: EmptyDir

- Suit le cycle de vie d'un Pod
- N'est pas supprimé si l'un des containers du Pod crash
- Vide à la création
- Peut-être utilisé sur un tmpfs (disk RAM)
- Points de montage potentiellement différents dans chaque container d'un Pod

Volume: EmptyDir

```
$ cat mongo-emptydir.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mongo
spec:
  containers:
    - image: mongo:3.6
      name: mongo
      volumeMounts:
        - mountPath: /data/db
          name: data
  volumes:
    - name: data
      emptyDir: {}
```

Définition d'un volume de type emptyDir

Volume: EmptyDir

```
$ cat mongo-emptydir.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mongo
spec:
  containers:
    - image: mongo:3.6
      name: mongo
      volumeMounts:
        - mountPath: /data/db
          name: data
  volumes:
    - name: data
      emptyDir: {}
```



Le volume est monté dans le container mongo sur /data/db

Définition d'un volume de type emptyDir

Volume: EmptyDir

```
# Lancement du Pod basé sur mongo
$ kubectl create -f mongo-emptydir.yaml

# Lancement d'un shell interactif dans le container mongo
$ kubectl exec -ti mongo -- bash

# Insertion d'une donnée de test
root@mongo:/# mongo
> use test
switched to db test
> db.k8s.insert({ok: 1})    // Insertion d'une donnée de test

# Terminaison du process mongod
root@mongo:/# kill 1  // kubectl détecte l'arrêt du container mongo et en lance un nouveau

# Lancement d'un shell interactif dans le nouveau container mongo
$ kubectl exec -ti mongo -- bash
root@mongo:/# mongo
> use test
switched to db test
> db.k8s.find()           // Le nouveau container accède à la donnée de test
{ "_id" : ObjectId("5a9f2078df4433ccad102e96"), "ok" : 1 }
```

Volume: hostPath

- Montage d'une ressource de la machine hôte dans un Pod
- Différents types
 - Directory
 - File
 - Socket
 - CharDevice
 - BlockDevice
- Exemple de cas d'utilisation
 - monitoring de la machine hôte
 - socket de communication avec le daemon Docker

Volume: hostPath

```
$ cat mongo-hostpath.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mongo
spec:
  containers:
    - image: mongo:3.6
      name: mongo
      volumeMounts:
        - mountPath: /data/db
          name: data
  volumes:
    - name: data
      hostPath:
        path: /data-db
```

Définition d'un volume de type hostPath.

Volume: hostPath

```
$ cat mongo-hostpath.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mongo
spec:
  containers:
    - image: mongo:3.6
      name: mongo
      volumeMounts:
        - mountPath: /data/db
          name: data
  volumes:
    - name: data
      hostPath:
        path: /data-db
```



Le répertoire /data-db de l'hôte est monté dans le répertoire /data/db du container mongo

Définition d'un volume de type hostPath.

Volume: hostPath

```
# Création du Pod
$ kubectl create -f mongo-hostpath.yaml
pod "mongo" created

# Connection ssh dans la VM de Minikube
$ minikube ssh
#
# Inspection du répertoire /data-db
# cd /data-db/
# ls
WiredTiger    _mdb_catalog.wt          index-3-2054357870167412243.wt
WiredTiger.lock    collection-0-2054357870167412243.wt journal
WiredTiger.turtle    collection-2-2054357870167412243.wt mongod.lock
WiredTiger.wt      diagnostic.data      sizeStorer.wt
WiredTigerLAS.wt    index-1-2054357870167412243.wtstorage.bson
```

Volume: PersistentVolume(PV)

- Abstraction de la gestion des volumes
- Stockage provisionné statiquement ou dynamiquement (via une StorageClass)
- Nombreux types différents
 - NFS / iSCSI / ...
 - GCEPersistentDisk
 - AWSElasticBlockStore
 - AzureFile / AzureDisk
 - CephFS / Ceph Block Device
 - <https://kubernetes.io/docs/concepts/storage/persistent-volumes/#types-of-persistent-volumes>

Volume: PersistentVolumeClaim(PVC)

- Une demande de stockage
- Consomme un PersistentVolume
- Spécifie des contraintes supplémentaires
 - taille
 - type
 - mode d'accès
- Création d'un binding entre PVC et PV

Volume : PV / PVC

```
# Spécification d'un PersistentVolume
$ cat pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: "pv"
spec:
  storageClassName: manual
  capacity:
    storage: "1Gi"
  accessModes:
    - "ReadWriteOnce"
  hostPath:
    path: /data/pv
```

Persistent Volume de type **hostPath**
Utilisation pour des tests, non supporté dans un
cluster de production car lié à un hôte

Volume : PV / PVC

```
# Création du PersistentVolume
$ kubectl create -f pv.yaml
persistentvolume "pv" created

# Liste des PersistentVolume (pv est un raccourci pour désigner un persistentvolume)
$ kubectl get pv
NAME      CAPACITY   ACCESS MODES  RECLAIM POLICY  STATUS      CLAIM      STORAGECLASS  REASON      AGE
pv        1Gi        RWO          Retain        Available           2s
```

```
# Liste des PersistentVolumeClaim
```

```
$ kubectl get pvc
No resources found.
```

Aucun Pod n'utilise de PersistentVolume
via un PersistentVolumeClaim



Volume : PV / PVC

```
# Spécification d'un PersistentVolumeClaim
$ cat pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi

# Création du PVC
$ kubectl create -f pvc.yaml
persistentvolumeclaim "claim" created

# Pas encore associé à un PersistentVolume
$ kubectl get pvc
NAME      STATUS      VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
claim     Bound       pvc-4c9a54dd-31ba-11e8-9614-080027f0e385   1Gi        RWO          standard      2s
```

Volume : PV / PVC

```
# Spécification d'un Pod utilisant un volume via un PersistentVolumeClaim
$ cat mongo-pvc.yaml
kind: Pod
apiVersion: v1
metadata:
  name: mongo
spec:
  containers:
    - name: mongo
      image: mongo:3.6
      volumeMounts:
        - mountPath: /data/db
          name: data-db
  volumes:
    - name: data-db
      persistentVolumeClaim:
        claimName: claim
```

Volume : PV / PVC

```
# Création du Pod
$ kubectl create -f mongo-pvc.yaml
pod "mongo" created

# Connection ssh dans la VM de Minikube
$ minikube ssh
#
# Inspection du répertoire /data/pv
# ls /data/pv
WiredTiger    _mdb_catalog.wt          index-3-5026428328222840801.wt
WiredTiger.lock    collection-0-5026428328222840801.wt journal
WiredTiger.turtle    collection-2-5026428328222840801.wt mongod.lock
WiredTiger.wt      diagnostic.data      sizeStorer.wt
WiredTigerLAS.wt    index-1-5026428328222840801.wtstorage.bson
```

Répertoire de stockage défini pour le PersistentVolume "pv"

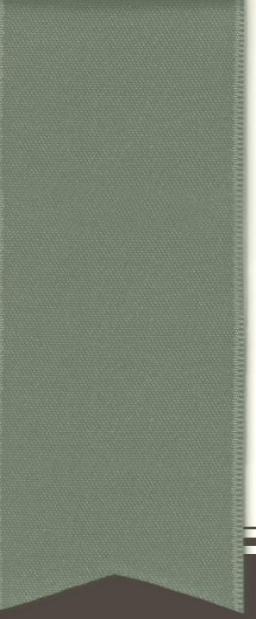
Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Démo



MODULE 10

Helm



kubernetes

Gestionnaire de packages pour kubernetes

- **Charts** : application packagée
 - description du package Chart.yaml
 - un ou plusieurs templates (spécifications Kubernetes)
- Architecture
 - helm : client local / CICD
 - tiller : daemon tournant dans le cluster, gestion des installations de Charts
- Charts stables : <https://github.com/kubernetes/charts/tree/master/stable>
- Charts incubés : <https://github.com/kubernetes/charts/tree/master/incubator>

Mise en place

- Installation
 - <https://github.com/kubernetes/helm/blob/master/docs/install.md>
- Initialisation

```
$ helm init  
  
$ kubectl get pods --namespace kube-system  
NAME                      READY   STATUS    RESTARTS   AGE  
tiller-deploy-865dd6c794-t2sj8   1/1     Running   0          33s  
...
```

daemon *tiller* lancé dans un Deployment

Les charts disponibles

```
$ helm search
```

WARNING: Deprecated index file format. Try 'helm repo update'

NAME	CHART VERSION	APP VERSION	DESCRIPTION
stable/acs-engine-autoscaler	2.1.3	2.1.1	Scales worker nodes within agent pools
stable/aerospike	0.1.7	v3.14.1.2	A Helm chart for Aerospike in Kubernetes
stable/anchore-engine	0.1.4	0.1.6	Anchore container analysis and policyevaluatio...
stable/artifactory	7.0.5	5.9.1	Universal Repository Manager supporting all maj...
stable/artifactory-ha	0.1.2	5.9.1	Universal Repository Manager supporting all maj...
stable/aws-cluster-autoscaler	0.3.2		Scales worker nodes within autoscaling groups.
stable/centrifugo	2.0.0	1.7.3	Centrifugo is a real-time messaging server.
stable/cert-manager	0.2.3	0.2.3	A Helm chart for cert-manager
stable/chaoskube	0.7.0	0.8.0	Chaoskube periodically kills random pods in you...
stable/chronograf	0.4.2		Open-source web application written in Go and R...
...			
testing/elasticsearch	0.2.0		Elasticsearch Restful Search Engine
testing/example-nginx	0.0.2		An example nginx + git-sync application
testing/example-todo	0.0.6		Example Todo application backed by Redis
testing/jenkins	0.2.0		The leading open-source continuous integration ...
testing/memcached	0.1.0		A simple Memcached cluster
testing/mysql	0.2.0		Chart running MySQL.
...			

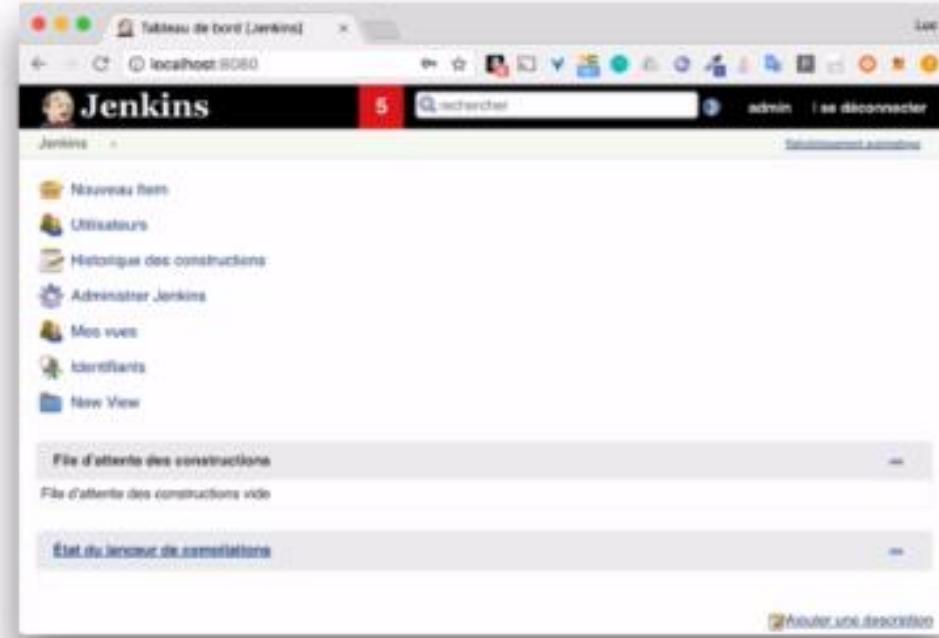
charts stable

charts en testing

Utilisation: exemple chart Jenkins

```
$ helm install stable/jenkins
NAME: tufted-seastar
...
NOTES:
1. Get your 'admin' user password by running:
   printf $(kubectl get secret --namespace default tufted-seastar-jenkins -o
   jsonpath=".data.jenkins-admin-password" | base64 --decode);echo → ko8oLDscLB
2. Get the Jenkins URL to visit by running these commands in the same shell:
   NOTE: It may take a few minutes for the LoadBalancer IP to be available.
   You can watch the status of by running 'kubectl get svc --namespace default -w
   tufted-seastar-jenkins'
   export SERVICE_IP=$(kubectl get svc --namespace default tufted-seastar-jenkins --template "{{ range (index
   .status.loadBalancer.ingress 0) }}{{ . }}{{ end }}")
   echo http://$SERVICE_IP:8080/login → http://localhost:8080/login
3. Login with the password from step 1 and the username: admin
...
```

Utilisation: exemple chart Jenkins



Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Démo