ORCHESTRATION DE CONTENEURS AVEC KUBERNETES

01.
TOUR DE TABLE

+ TOUR DE TABLE
+ VOTRE FORMATEUR
+ PROGRAMME DÉTAILLÉ

TOUR DE TABLE

Votre expérience sur les containers

Vos attentes

VOTRE FORMATEUR

Ludovic Quenec’hdu

02. FONDAMENTAUX

+ RAPPELS DOCKER
+ IMAGES ET CONTENEURS
+ VOLUMES ET RÉSEAUX
+ RAPPELS ANSIBLE

02.
FONDAMENTAUX

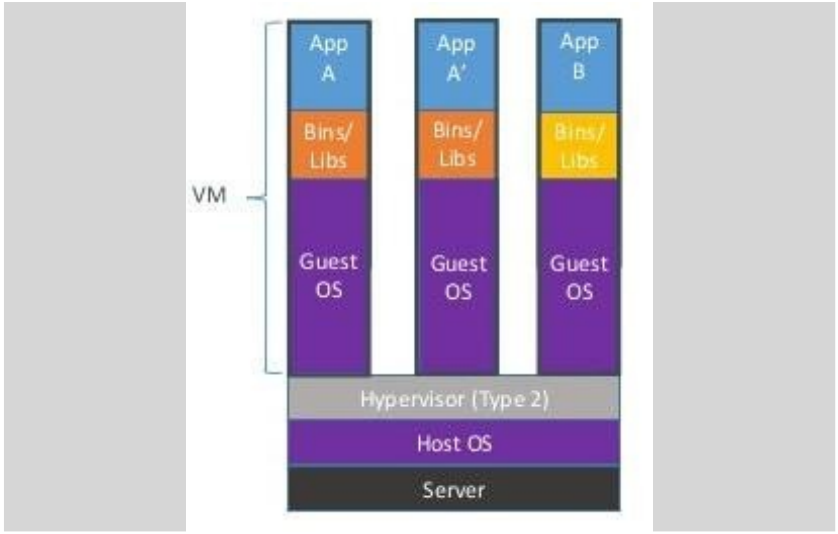
RAPPELS Conteneurisation

RAPPELS DOCKER

VIRTUALISATION VS CONTENEURISATION

Virtualisation

- Un serveur physique héberge un hyperviseur
- Ce dernier abstrait le matériel de son hôte et offre une interface matérielle « virtuelle »
- Des « machines » peuvent ainsi être instanciées, sur lesquelles un OS est installé

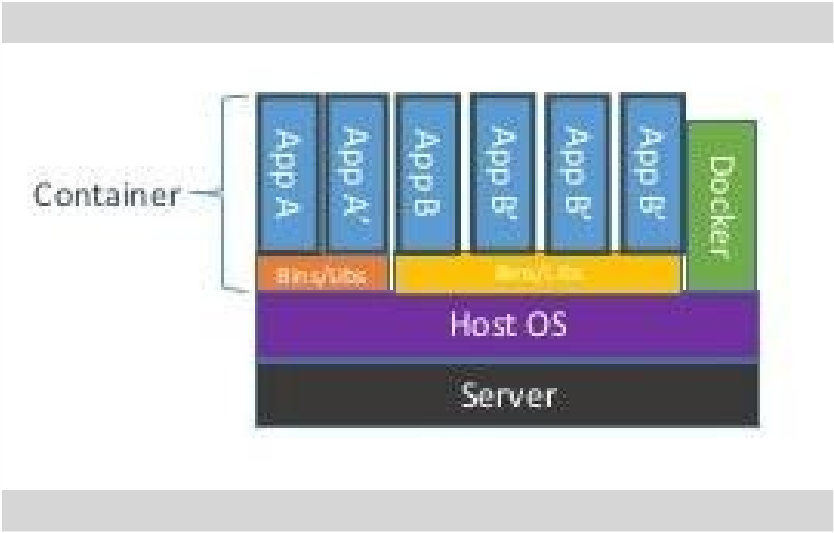


RAPPELS DOCKER

VIRTUALISATION VS CONTENEURISATION

Conteneurisation

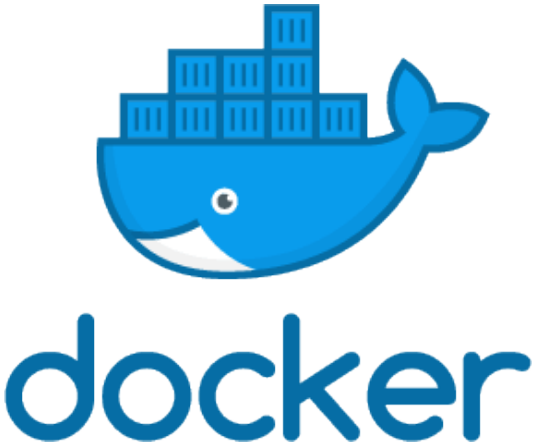
- S’appuie sur les fonctionnalités d’isolation du noyau linux
- Un gestionnaire de conteneurs est installé sur l’hôte
- Des « conteneurs » sont instanciés et sont isolés les uns des autres, bien qu’ils s’exécutent sur l’hôte



RAPPELS DOCKER

HISTORIQUE

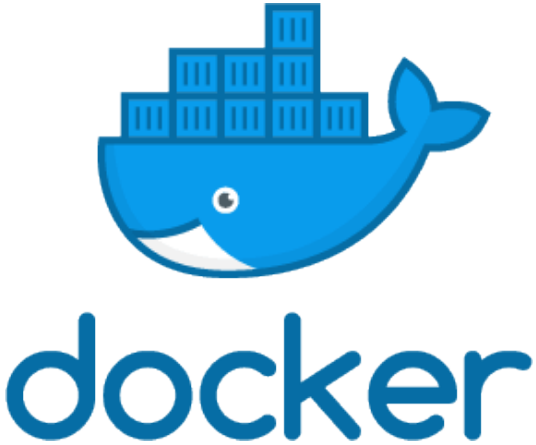
- **Créé en 2013**
- **S'appuie sur la technologie LXC (Linux Containers) et les cgroups du noyau Linux**
- **Solution rapidement adoptée**
- **$235 millions de financement au total**

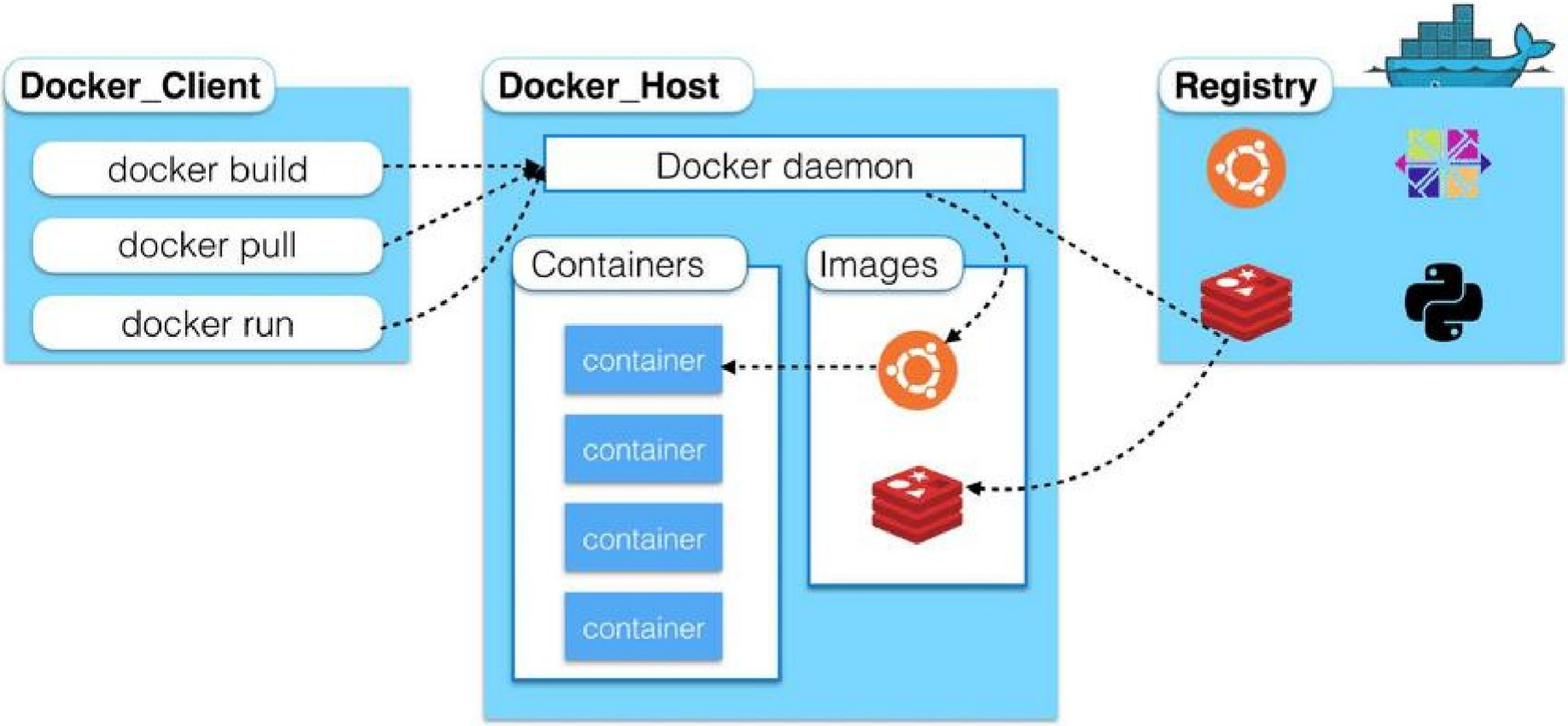


RAPPELS DOCKER

AVANTAGES

- **Solution qui fonctionne sur Linux non modifié (contrairement à OpenVZ par exemple)**
- **Facile à installer sur un poste de développement**
- **Communauté active et dynamique, outillage puissant**




Docker_Client
docker build
docker pull
docker run
Docker_Host
Docker daemon
Containers
container
container
container
container
Images
Registry

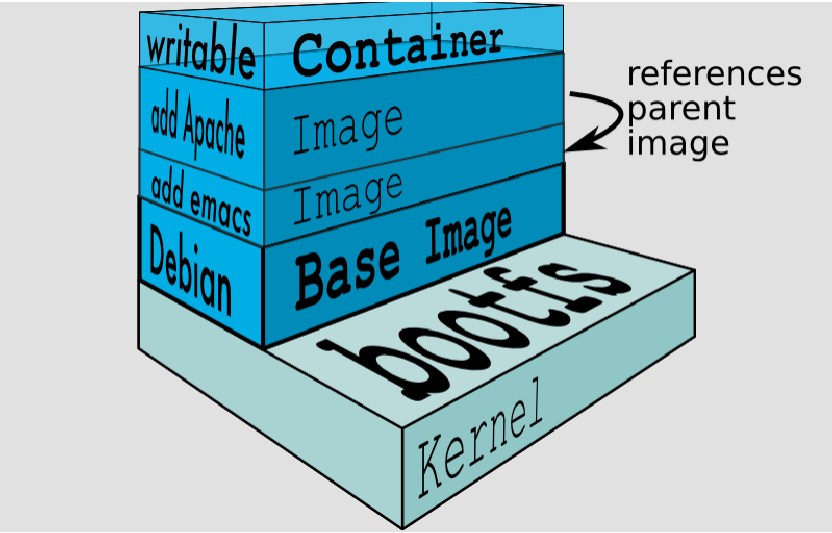
02.
FONDAMENTAUX

IMAGES ET CONTENEURS

IMAGES ET CONTENEURS

LES IMAGES DOCKER

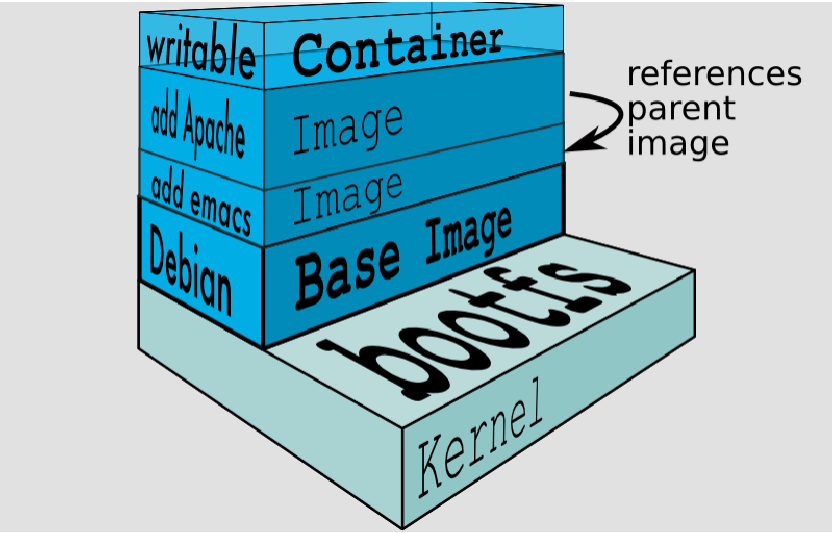
- **Analogie venant de la programmation orientée objet :**
  - Image = Classe
  - Conteneur = Instance
- **Chaque image est construite une seule fois**
- **On peut construire N conteneurs avec 1 image**



IMAGES ET CONTENEURS

LES IMAGES DOCKER

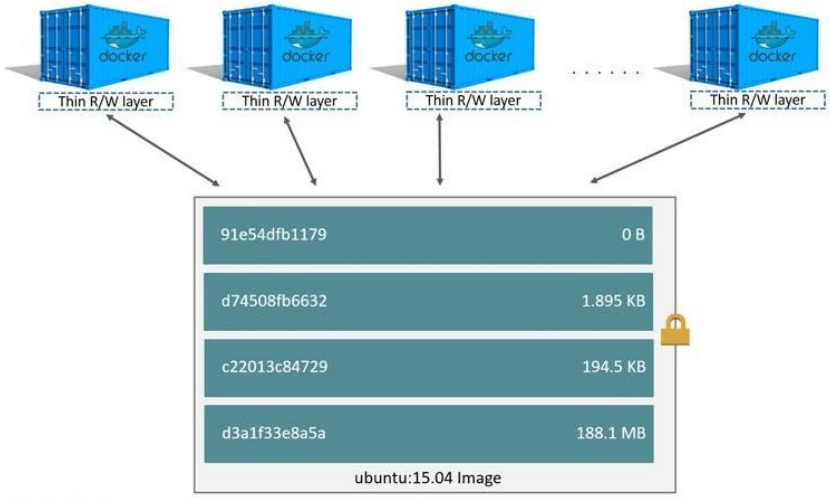
- **Les images sont définies grâce à un langage déclaratif**
- **Une image est un ensemble de « couches » de fichiers (comme des calques)**
- **Chaque image peut s'appuyer sur une autre image comme couche de démarrage**



IMAGES ET CONTENEURS

LES CONTENEURS DOCKER

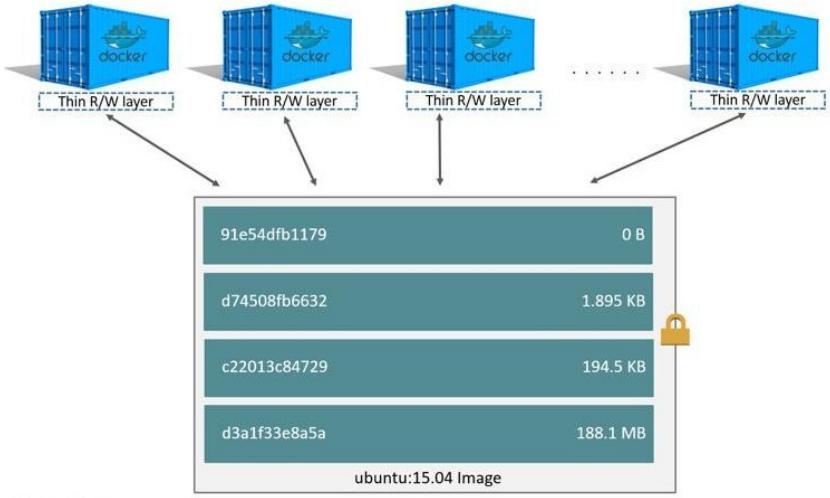
- **Un conteneur instancié depuis une image se voit attribuer :**
  - Toutes les couches de son image en lecture seule
  - Une couche supplémentaire accessible en écriture, propre au conteneur
- **Mécanisme de Copy on Write lors de la modification d'un fichier provenant de l'image**



IMAGES ET CONTENEURS

LES CONTENEURS DOCKER

- **Conséquences du système de couches de Docker :**
  - Économie de place, les couches de l'image ne sont pas copiées
  - Un conteneur qui écrit beaucoup prend plus de place qu'un conteneur qui ne fait que de la lecture
  - Les conteneurs démarrent très rapidement (la seule opération est la création de la couche supérieure)
  - Données supprimées en même temps que les conteneurs



02.
FONDAMENTAUX

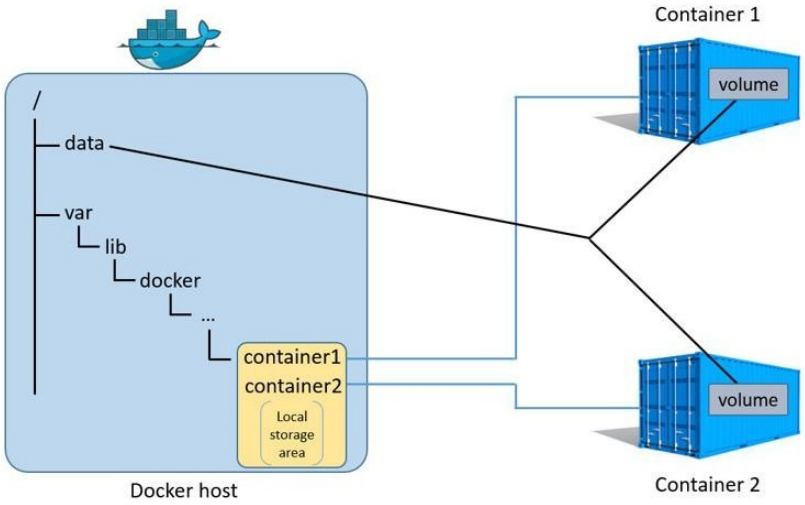
VOLUMES ET RÉSEAUX

VOLUMES ET RÉSEAUX

BONNES PRATIQUES DE GESTION DES DONNÉES CONTENEURISÉES

« Conteneurs immuables »

- **Les contenu des conteneurs ne varie jamais (aucune écriture)**
- **Les écritures se font sur un volume externe**
- **De cette manière, les conteneurs peuvent partager des espaces de stockage communs**



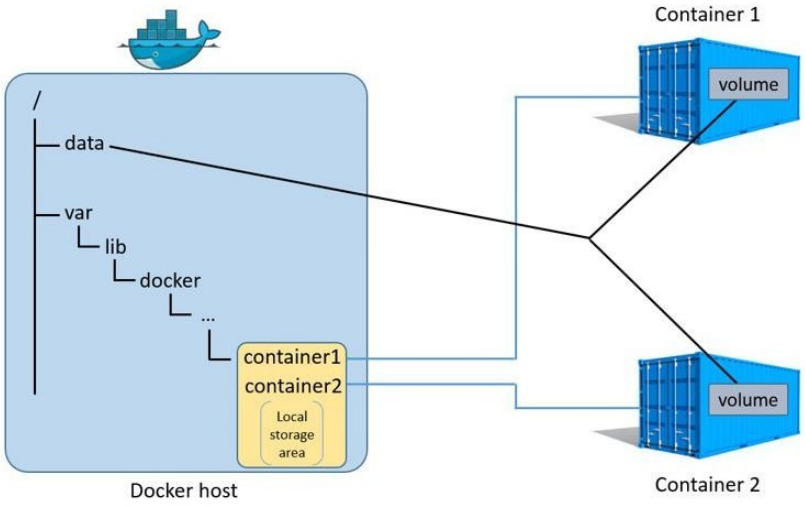
VOLUMES ET RÉSEAUX

VOLUMES DOCKER

L'option --mount permet d'associer un espace de stockage au conteneur créé.

Il existe 3 types d'espace de stockage :

- **Les volumes**
- **Les binds**
- **Les montages tmpfs**

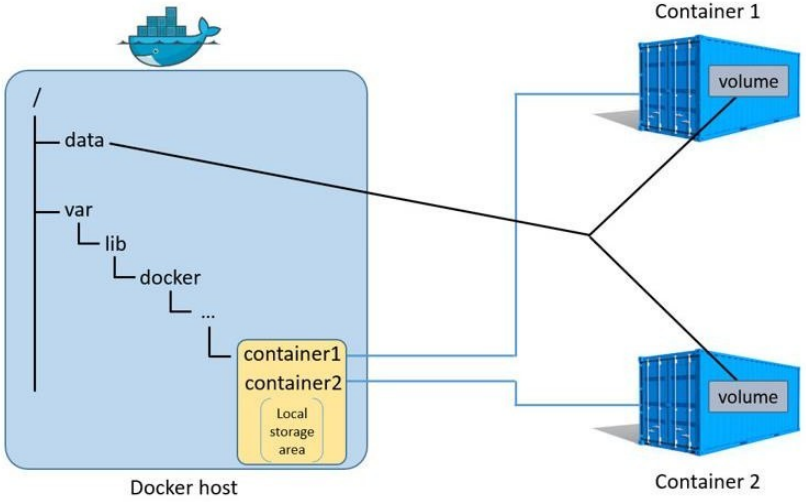


VOLUMES ET RÉSEAUX

VOLUMES DOCKER

Montage de type Volume

- **Espace sur l'hôte géré par docker**
- **Le montage se fait avec l'option --mount ou --volume**

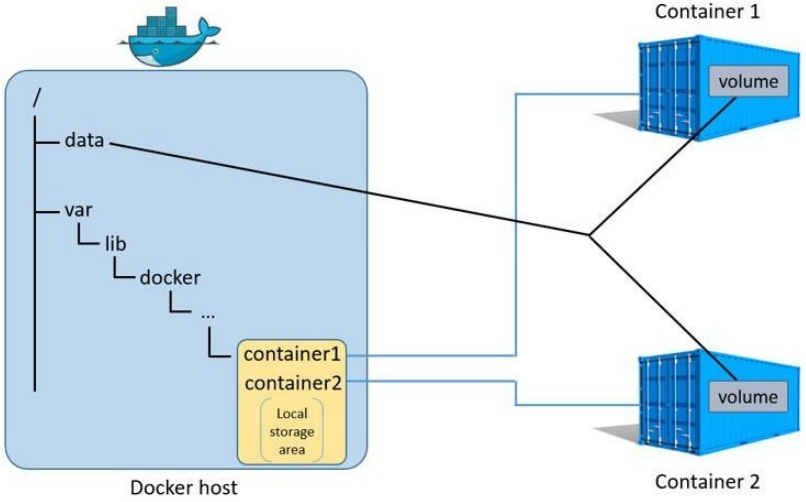


VOLUMES ET RÉSEAUX

VOLUMES DOCKER

Montage de type Bind

- **Espace sur l'hôte indépendant de Docker**
- **Un répertoire de l'hôte est directement monté dans le conteneur**
- **Plus complexe à maintenir que les volumes (nécessite de s'assurer de l'existence des répertoires)**

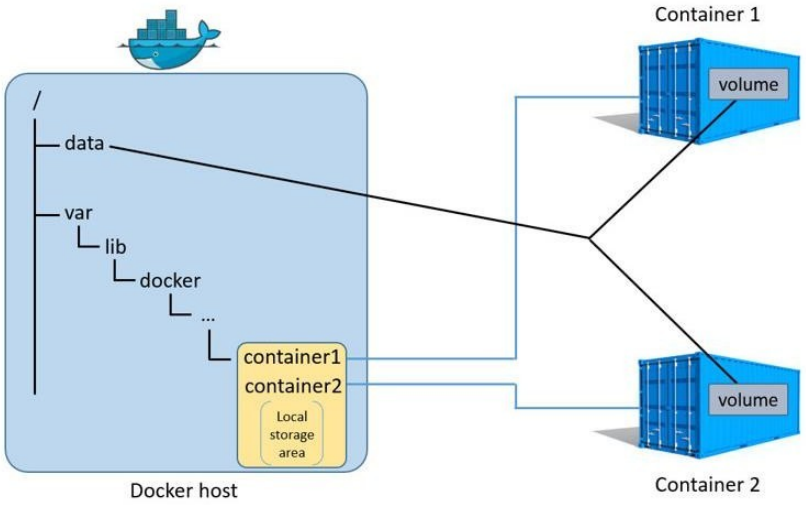


VOLUMES ET RÉSEAUX

VOLUMES DOCKER

Montage de type Tmpfs

- **Espace de stockage créé en mémoire et monté sur l'hôte**
- **Utile pour les conteneurs qui nécessitent une zone de travail avec de bonnes performances**



VOLUMES ET RÉSEAUX

RÉSEAUX DOCKER

Il est possible de gérer des réseaux indépendants et isolés directement avec docker

- **Permet l'isolation d'un ensemble de conteneurs entre eux**
- **Permet également la communication réseau de l'hôte vers les conteneurs**

VOLUMES ET RÉSEAUX

RÉSEAUX DOCKER

Docker s'appuie sur un driver réseau pour fournir la couche réseau aux conteneurs

Quelques drivers communs :

- BRIDGE **: une interface de type bridge est créée sur l'hôte et isole ce dernier des conteneurs**
- HOST **: les conteneurs s'interfacent directement avec le réseau de l'hôte**
- OVERLAY **: réseau permettant de connecter les démons docker sur plusieurs hôtes de manière transparente**

03.
INTRODUCTION À KUBERNETES

+ BREF HISTORIQUE
+ PRINCIPE D'ORCHESTRATION DE CONTENEUR
+ AVANTAGES DE L'ORCHESTRATION DE CONTENEUR
+ CLOUD NATIVE COMPUTING FOUNDATION ET OPEN CONTAINER INITIATIVE

03.

INTRODUCTION À KUBERNETES

BREF HISTORIQUE

INTRODUCTION À KUBERNETES

BREF HISTORIQUE

Créé par deux ingénieurs de Google en 2003 et rendu open source en 2014, Kubernetes s'impose progressivement comme solution de référence d'orchestration de conteneurs

HISTORIQUE

+ 2003 : Création de Borg, outil de gestion de cluster
+ 2013 : Publication du papier présentant Omega [1]
+ 2014 : Introduction d'une version allégée de Omega, sous le nom de Kubernetes
+ 2015 : Kubernetes v1.0
+ 2017 : Kubernetes 1.6, version de stabilisation adoptée par de plus en plus d'entreprises

1. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/41684.pdf

INTRODUCTION À KUBERNETES

BREF HISTORIQUE

Kubernetes (grec ancien, n.m.) [koo-ber-nay'-tace] :

TIMONIER, HOMME DE BARRE, MAÎTRE DE NAVIGATION

HISTORIQUE

+ 2003 : Création de Borg, outil de gestion de cluster
+ 2013 : Publication du papier présentant Omega [1]
+ 2014 : Introduction d'une version allégée de Omega, sous le nom de Kubernetes
+ 2015 : Kubernetes v1.0
+ 2017 : Kubernetes 1.6, version de stabilisation adoptée par de plus en plus d'entreprises

1. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/41684.pdf

03.

INTRODUCTION À KUBERNETES

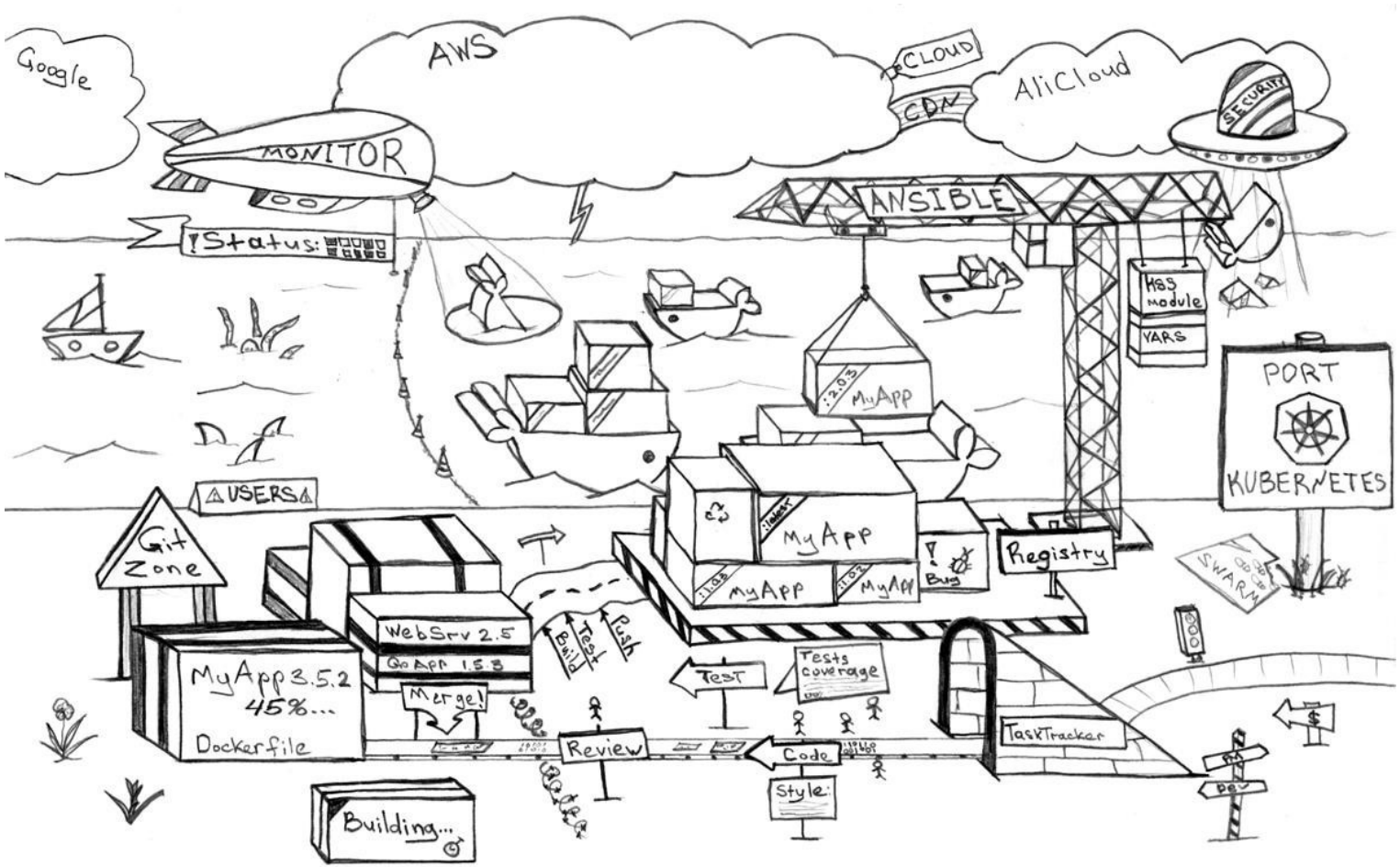
PRINCIPE D'ORCHESTRATION DE CONTENEURS

PRINCIPE D'ORCHESTRATION DE CONTENEURS

DÉFINITION

Aussi appelé CaaS (Containers as a Service)

Un orchestrateur de conteneur gère la définition des services, leur instanciation, leur contrôle et leur distribution sur plusieurs hôtes, ainsi que l'accès au réseau et aux ressources physiques de ces hôtes.


Google
AWS
CLOUD
CDN
AliCloud
SECURITY
MONITOR
Status:
ANSIBLE
Hes module
VARS
PORT
KUBERNETES
MyApp
:2.0.3
USERS
Git Zone
MyApp 3.5.2
45%...
Dockerfile
WebSrv 2.5
QoApp 1.5.3
Merge!
Test
Build
Push
MyApp
:latest
:1.0.3
:2.0.2
Bug
Registry
SWARM
Tests coverage
Test
Review
Code
Style
Building...
TaskTracker
DEV

PRINCIPE D'ORCHESTRATION DE CONTENEURS

PROBLÈMES RÉSOLUS PAR L'ORCHESTRATION DE CONTENEURS

- **Que faire lorsqu'un conteneur se termine de manière inattendue ?**
- **Comment choisir les nœuds du cluster où déployer tel ou tel conteneur ?**
- **Comment s'assurer de la haute disponibilité d'une application ?**
- **Comment limiter la consommation des ressources de chaque conteneur ?**
- **Comment connecter les conteneurs entre eux, entre différents hôtes ?**
- **Comment partager des volumes de données persistantes entre les conteneurs, sur différents hôtes ?**
- **Comment s'assurer de l'équilibrage du cluster ?**

PRINCIPE D'ORCHESTRATION DE CONTENEURS

CATTLE VS PET

Serveur « animal de compagnie **»**

- **Créé, configuré et nommé à la main**
- **«** lovely-server.aelion.fr **»**
- **Chaque problème peut devenir une catastrophe**

Serveur « bétail **»**

- **Créé automatiquement et en grand nombre**
- **«** aow3ferm34r12.aelion.fr **»**
- **Problème => abattoir**

03.

INTRODUCTION À KUBERNETES

AVANTAGES DE L'ORCHESTRATION DE CONTENEURS

AVANTAGES DE L'ORCHESTRATION DE CONTENEURS

Plusieurs points de vue à considérer :

- **Performances**
- **Sécurité**
- **Résilience**
- **Exploitabilité**

AVANTAGES DE L'ORCHESTRATION DE CONTENEURS

Performances :

- **Contrôle des ressources allouées à chaque conteneur**
- **Bonne exploitation des ressources disponibles dans le datacenter, pas de ressources allouées et non exploitées**
- **L'orchestration en elle-même n'ajoute pas d'impact sur les performances des applications déployées**
- **Passage à l'échelle trivial et automatisable**

AVANTAGES DE L'ORCHESTRATION DE CONTENEURS

Sécurité :

- **Contrôle précis des règles réseau**
- **Possibilité d'ajouter des middleware de sécurité en amont de tout conteneur déployé**
- **Dans le cas de Kubernetes, API de gestion des droits**

AVANTAGES DE L'ORCHESTRATION DE CONTENEURS

Résilience :

- **Le cœur de la valeur apportée par l'orchestration de conteneurs**
- **Les applications sont surveillées, recrées, passées à l'échelle si nécessaire**

AVANTAGES DE L'ORCHESTRATION DE CONTENEURS

Exploitabilité :

- **Aucun composant n'est « enfermé », il est toujours possible de descendre au plus bas niveau d'exécution pour des besoins de diagnostic**
- **Surveillances/alertes faciles à automatiser**
- **Dans le cas de Kubernetes, tous les outils sont prévus pour faciliter l'exploitation**

03.

INTRODUCTION À KUBERNETES

CLOUD NATIVE COMPUTING FOUNDATION OPEN CONTAINER INITIATIVE

CLOUD NATIVE COMPUTING FOUNDATION

CNCF

The Foundation’s mission is to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self healing multi-tenant nodes.

CLOUD NATIVE COMPUTING FOUNDATION

PRÉREQUIS

- **Distribuer sous forme de conteneurs**
- **Gestion dynamique de la configuration**
- **Orienté micro services**

CLOUD NATIVE COMPUTING FOUNDATION

LES RÔLES

- **Intendance des projets**
- **Faire grossir et évoluer l'écosystème**
- **Rendre la technologie accessible**
- **Promouvoir la technologie**

OPEN CONTAINER INITIATIVE

OCI

- **Créé sous la Linux Foundation**
- **Objectif : créer un standard Open Source de format d'images, de conteneurs et de méthodes d'exécution de ces derniers**
- **Non lié à des produits**

04. ARCHITECTURE DE KUBERNETES

+ NŒUDS MASTER ET WORKERS
+ KUBELET, API-SERVER, PROXY, DNS ...

04.

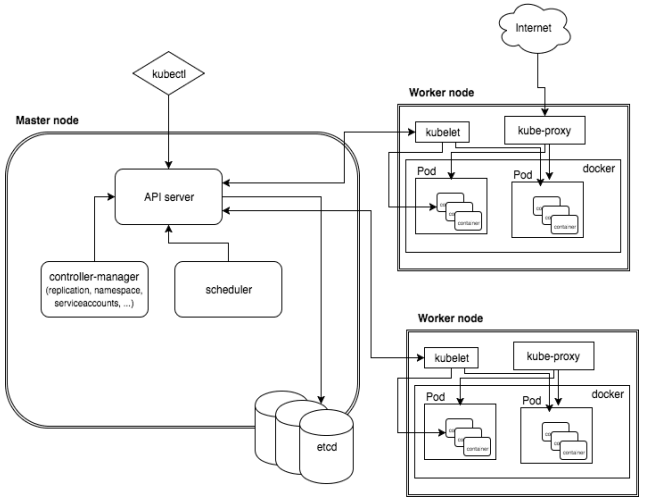
ARCHITECTURE DE KUBERNETES

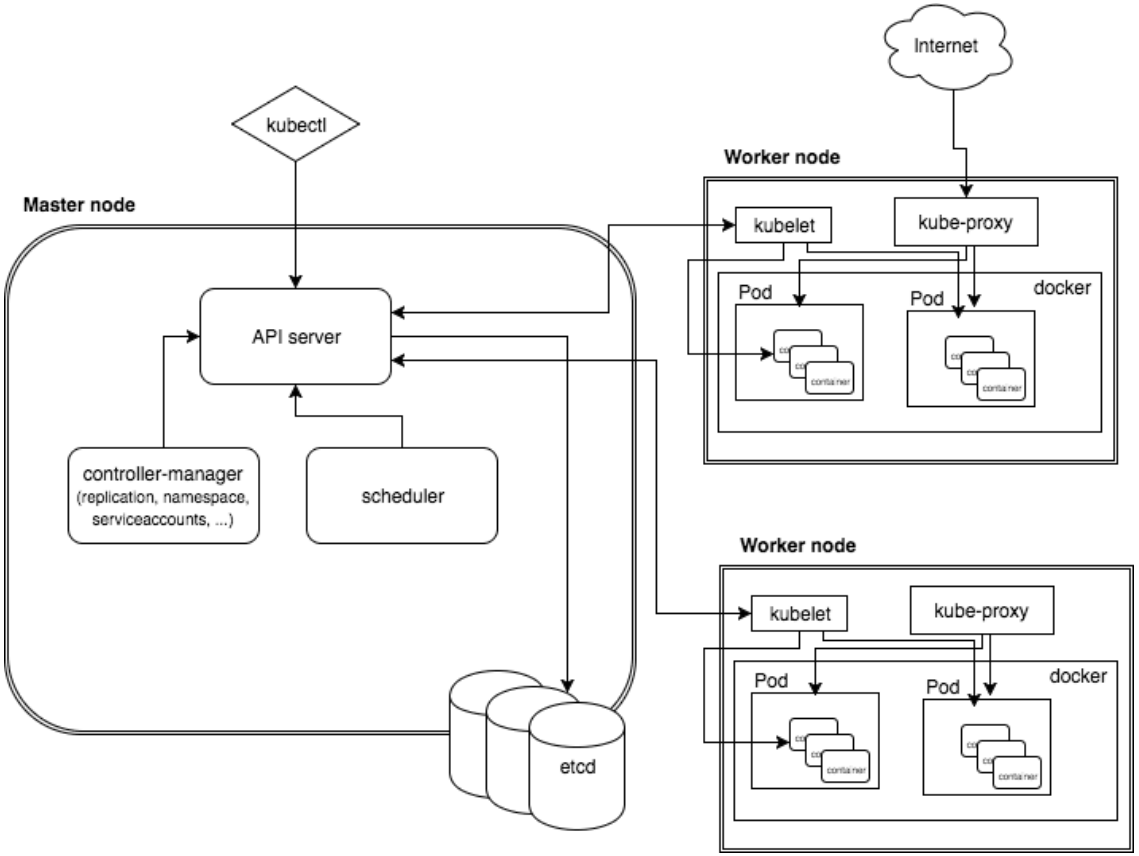
NŒUDS MASTER ET WORKERS

NŒUDS MASTER ET WORKERS

ARCHITECTURE GÉNÉRALE

- **Kubernetes est écrit en Go, compilé statiquement**
- **Un ensemble de binaires sans dépendances**
- **Faciles à conteneuriser et à packager**
- **Peut se déployer uniquement avec des conteneurs sans dépendances d'OS**

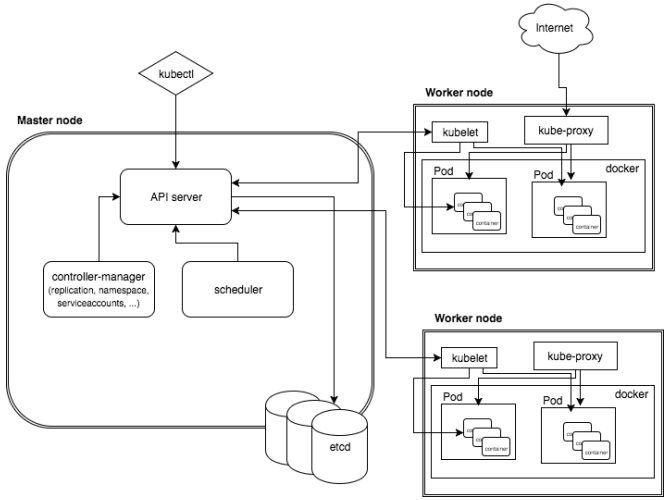



Internet
kubectl
Worker node
Master node
kubelet
kube-proxy
Pod
Pod
docker
API server
container
controller-manager
(replication, namespace,
serviceaccounts, ...)
scheduler
Worker node
kubelet
kube-proxy
Pod
Pod
docker
container
etcd

NŒUDS MASTER ET WORKERS

ARCHITECTURE GÉNÉRALE

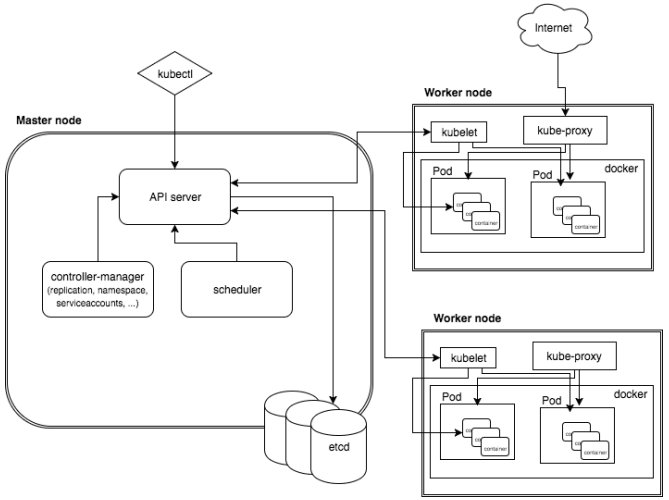
- **Nœud master :**
  - Expose l'API de gestion
  - Stockage de la configuration
  - Héberge les applications de contrôle
  - Parfois appelés « control plane »
- **Nœud worker :**
  - Reçoit les ordres en provenance de l'API
  - Responsable de la manipulation des conteneurs
  - Responsable du routage réseau
  - Anciennement appelés « minions »



NŒUDS MASTER ET WORKERS

ARCHITECTURE GÉNÉRALE

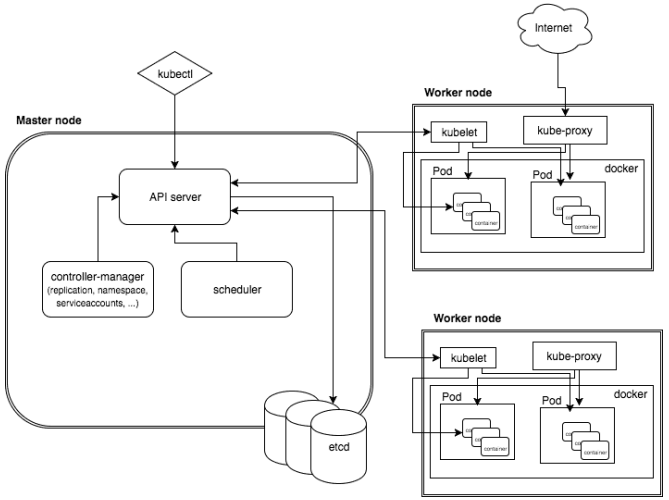
- **Un nœud peut être à la fois master et worker (rarement)**
- **On aura généralement quelques nœuds master pour assurer la haute disponibilité du control-plane**
- **Les nœuds workers viennent quant à eux renforcer la haute disponibilité des applications**



NŒUDS MASTER ET WORKERS

RÉSILIENCE

- **Les composants système de Kubernetes sont conteneurisés et donc volatiles**
- **Le cœur du cluster Kubernetes est le composant de stockage distribué où il stocke son état (etcd)**
- **Pour faire une sauvegarde du cluster, il suffit de faire des sauvegardes régulières du contenu d'etcd**



04.

ARCHITECTURE DE KUBERNETES

KUBELET, API-SERVER, PROXY, DNS ...

COMPOSANTS SYSTÈME

KUBELET, API-SERVER, PROXY, DNS ...

Kubernetes s'appuie sur ses composants pour fonctionner, l'ensemble est modulaire.

COMPOSANTS MASTER

+ API Server
+ Etcd
+ Controller manager
+ Scheduler
+ DNS

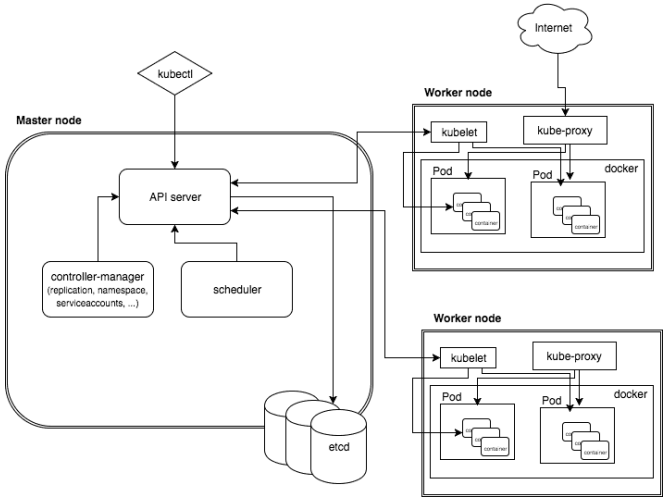
COMPOSANTS WORKER

+ Kubelet
+ Proxy

COMPOSANTS MASTER

API SERVER

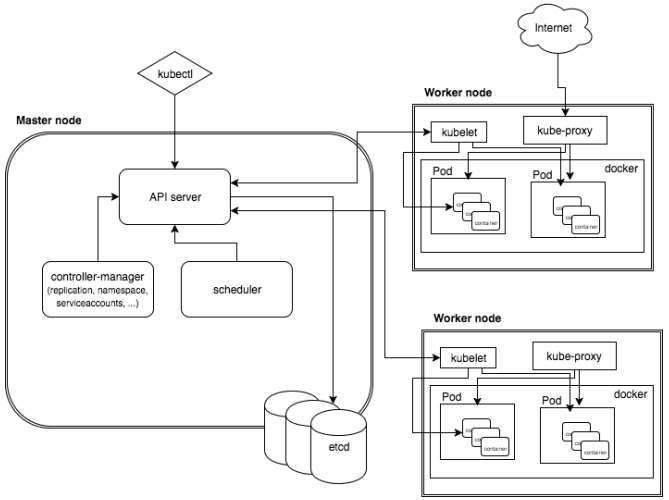
- **Expose l'API Kubernetes**
- **Les configurations d'objets (Pods, Service, RC, etc.) se font via l'API server**
- **Hub de communication entre les autres composants**
- **Sans état, peut passer à l'échelle facilement**



COMPOSANTS MASTER

ETCD

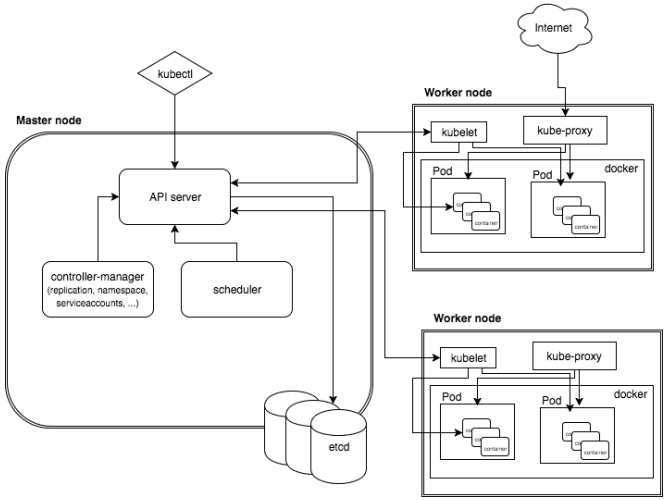
- **Système de stockage distribué très fiable**
- **Utilisé par Kubernetes pour stocker l'état complet du cluster**
- **Attention de toujours avoir un nombre impair de nœuds etcd (3, 5, ou 7)**



COMPOSANTS MASTER

CONTROLLER MANAGER

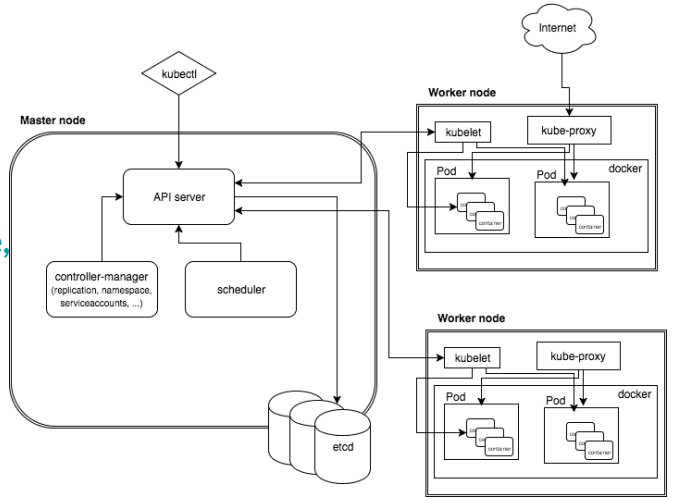
- **Agrégat des différents autres controllers**
- **Contient le controller de pods, de services, de replication, etc.**
- **Responsable de l'application des changements requis par l'API**



COMPOSANTS MASTER

SCHEDULER

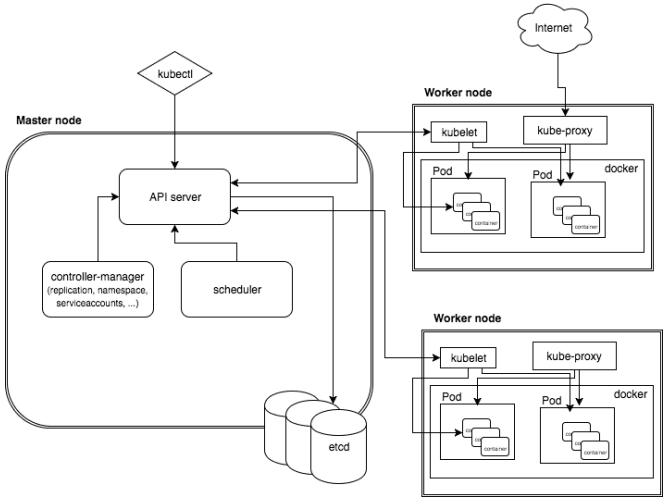
- **Composant responsable de l'ordonnancement**
- **En fonction de règles implicites (CPU, RAM, stockage disponible, etc.)**
- **En fonction de règles explicites (règles d'affinité et anti-affinité, labels, etc.)**



COMPOSANTS MASTER

DNS

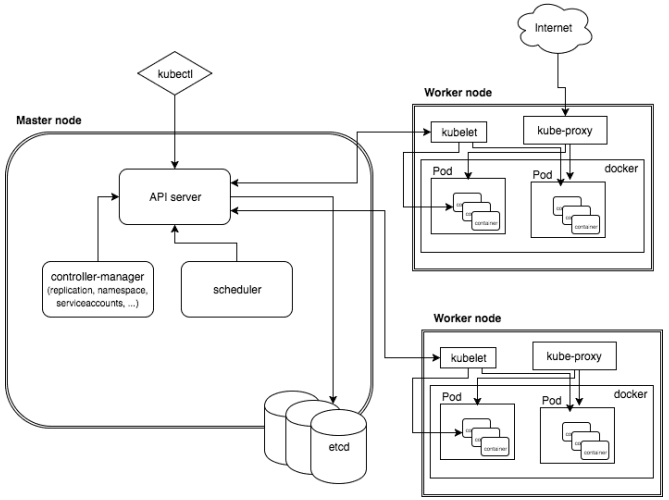
- **Serveur DNS interne**
- **Chaque service Kubernetes se voit attribué un nom sous la forme : service.namespace.svc.cluster.local**
- **Très utile pour la découverte de services et pour connecter les applications entre elles**



COMPOSANTS WORKERS

KUBELET

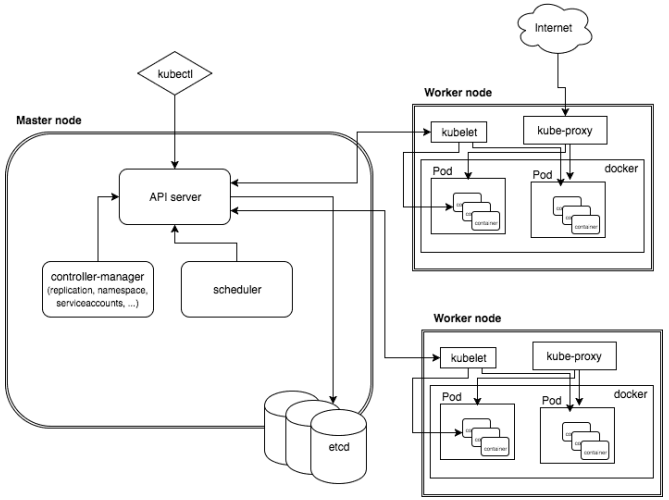
- **Composant principal sur les workers**
- **Lance les conteneurs sur les hôtes**
- **Exécute les sondes de bonne santé dans les conteneurs**
- **Reçoit des manifestes principalement en provenance de l'api-server (possible aussi depuis un répertoire ou une url HTTP)**



COMPOSANTS WORKERS

PROXY

- **Composant responsable d'interpréter les objets Service et d'exposer les conteneurs concernés en conséquence**
- **S'appuie sur netfilter (composant kernel de filtrage réseau, habituellement manipulé avec le programme iptables)**
- **Réalise également un load-balancing simple entre les conteneurs (round-robin)**



05.

CRÉER UN CLUSTER KUBERNETES

+ SOLUTIONS POSSIBLES
+ CRÉER SON CLUSTER AVEC MINIKUBE

05.

CRÉER UN CLUSTER KUBERNETES

SOLUTIONS POSSIBLES

CRÉER UN CLUSTER

DIFFÉRENTES SOLUTIONS

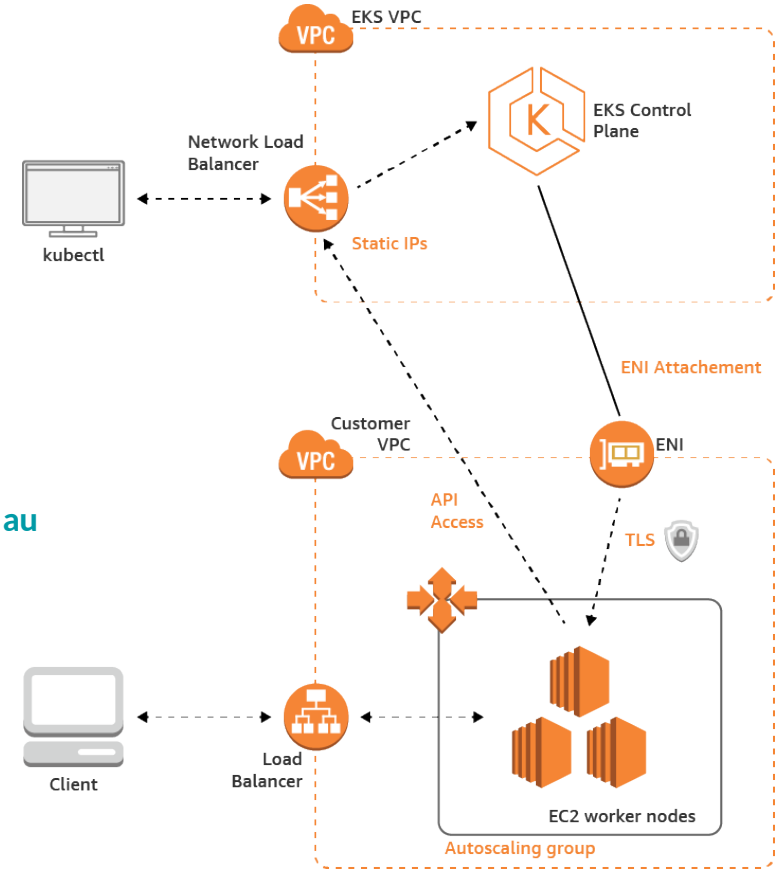
- **Solutions « grand clouds » :**
  - Amazon Elastic Kubernetes Service (EKS)
  - Google Kubernetes Engine (GKS)
  - Microsoft Azure Kubernetes Service (AKS)
- **Solutions hébergées :**
  - Giant Swarm
- **Solutions « on-premise »**
  - Rancher 2
  - Kubeadm

CRÉER UN CLUSTER

FONCTIONNEMENT DES SOLUTIONS « GRAND CLOUD »

- **Le fournisseur cloud permet de déployer le control- plane Kubernetes comme entité indépendante**

- **Il faut ensuite ajouter les éléments d'infrastructure nécessaires au fonctionnement du cluster :**
  - Nœuds de calcul
  - Load balancer
  - Stockage objet ou block



05.

CRÉER UN CLUSTER KUBERNETES

CRÉER SON CLUSTER AVEC MINIKUBE

INSTALLER MINIKUBE

Activer hyper-v :

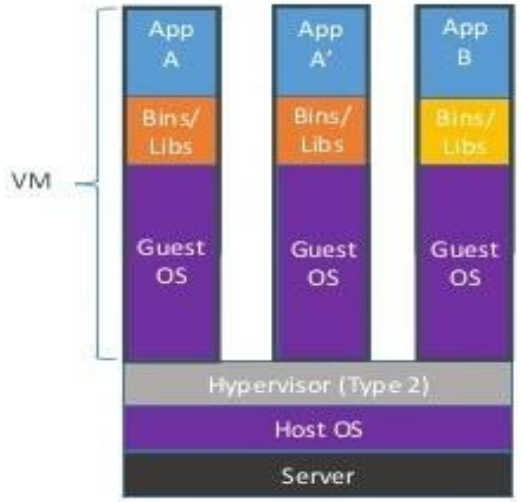
```
Enable-WindowsOptionalFeature -Online
-  FeatureName Microsoft-Hyper-V -All
```

Installer chocolatey et les paquets nécessaires :

```
choco install kubernetes-cli  choco
install minikube
```

Démarrer :

```
minikube start --vm-driver hyperv
```



06.

INTRODUCTION À KUBECTL

+ FICHIER KUBE CONFIG
+ COMMANDES ET OPTIONS DE BASE
+ OBTENIR LES INFORMATIONS DU CLUSTER
+ CONTEXTES ET NAMESPACES

06.

INTRODUCTION À KUBECTL

FICHIER KUBE CONFIG

FICHIER KUBE CONFIG

GÉNÉRALITÉS

- **Fichier yaml situé dans ~/.kube/config**
- **Contient la liste des clusters, contextes et informations d'authentification**
- **Peut être manipulé directement ou avec la commande kubectl config**

```yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority: C:\Users\jdoe\.minikube\ca.crt
    server: https://192.168.1.22:8443
  name: minikube
contexts:
- context:
    cluster: minikube
    user: minikube
  name: minikube
- context:
    cluster: minikube
    namespace: staging
    user: minikube
  name: staging
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: C:\Users\jdoe\.minikube\client.crt
    client-key: C:\Users\jdoe\.minikube\client.key
```

FICHIER KUBE CONFIG

TRAVAILLER AVEC PLUSIEURS CLUSTERS

- Il est possible de travailler sur plusieurs clusters différents
- Il suffit de déclarer le cluster dans le fichier de configuration en indiquant le chemin vers le certificat racine de l'API ainsi que l'adresse de cette dernière
- Il est également possible d'utiliser des fichiers de configuration séparés

```
export
KUBECONFIG=$KUBECONFIG:config2
```

```yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority: C:\Users\jdoe\.minikube\ca.crt
    server: https://192.168.1.22:8443
  name: minikube
contexts:
- context:
    cluster: minikube
    user: minikube
  name: minikube
- context:
    cluster: minikube
    namespace: staging
    user: minikube
  name: staging
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: C:\Users\jdoe\.minikube\client.crt
    client-key: C:\Users\jdoe\.minikube\client.key
```

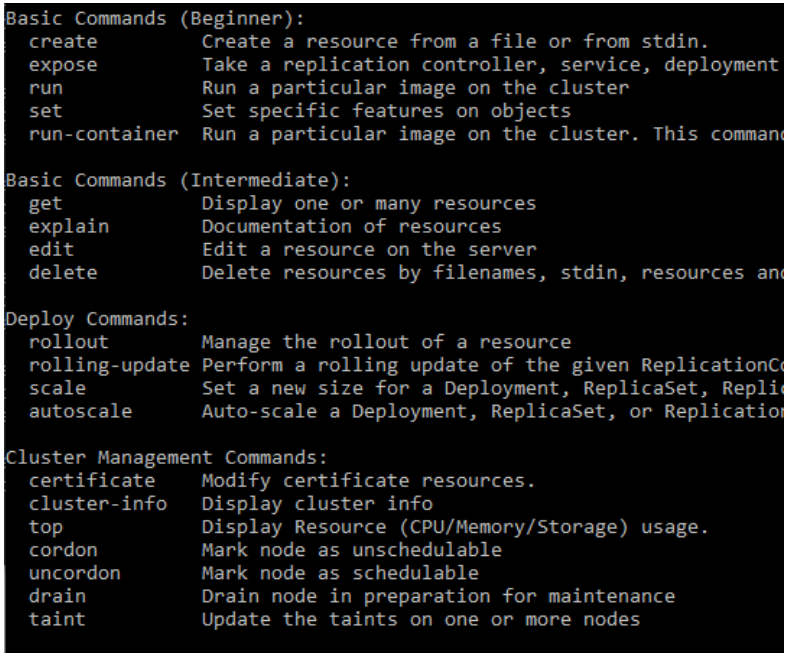
06.

INTRODUCTION À KUBECTL

COMMANDES ET OPTIONS DE BASE

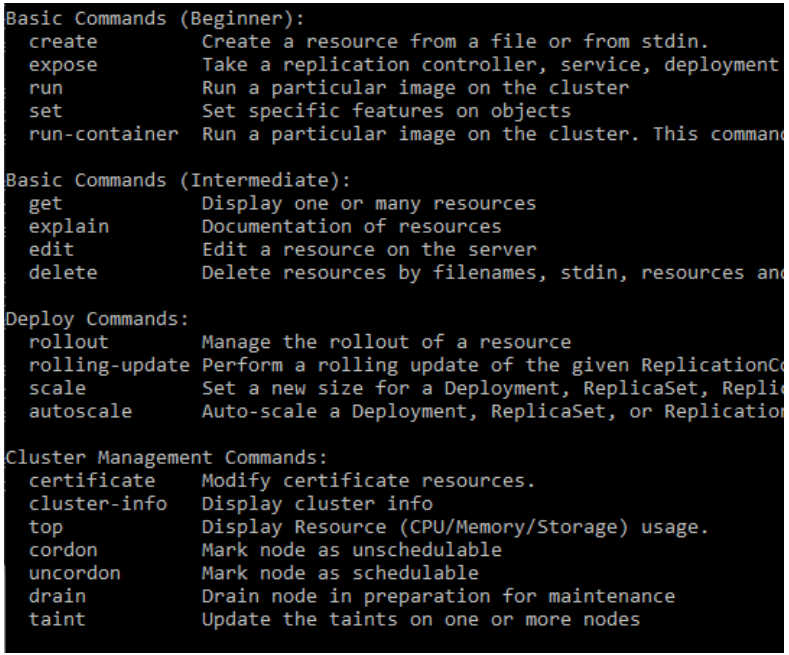
COMMANDES ET OPTIONS DE BASE

- **Commandes les plus courantes :**
  - **get** <type> <nom> :
    afficher les ressources, option -o pour spécifier le format de sorite
  - **apply** -f <fichier> :
    création/mise à jour des ressources selon un fichier de manifeste (commande idempotente)
  - **edit** <type> <nom> :
    modification interactive d'une ressource
  - **describe** <type> <nom> :
    affichage des informations détaillées d'une ressource



COMMANDES ET OPTIONS DE BASE

- **Options utiles de la commande get :**
  - **--namespace** / **-n** : afficher les ressources dans un namespace spécifique
  - **--all-namespaces** : afficher les ressources dans tous les namespaces
  - **--selector** / **-l** : afficher toutes les ressources correspondant à un label spécifique
  - **--watch** / **-w** : surveiller les changements après affichage des informations



06.

INTRODUCTION À KUBECTL

OBTENIR LES INFORMATIONS DU CLUSTER

OBTENIR LES INFORMATIONS DU CLUSTER

- **Obtenir un aperçu rapide du cluster :**
  - **get nodes** :
    afficher la liste des nœuds et leur état
  - **cluster-info** :
    affiche l'adresse du nœud maître

- **Obtenir des informations détaillées**
  - **get nodes -o yaml**
  - **cluster-info dump**

```
Basic Commands (Beginner):
  create         Create a resource from a file or from stdin.
  expose         Take a replication controller, service, deployment
  run            Run a particular image on the cluster
  set            Set specific features on objects
  run-container  Run a particular image on the cluster. This comman

Basic Commands (Intermediate):
  get            Display one or many resources
  explain        Documentation of resources
  edit           Edit a resource on the server
  delete         Delete resources by filenames, stdin, resources an

Deploy Commands:
  rollout        Manage the rollout of a resource
  rolling-update Perform a rolling update of the given ReplicationC
  scale          Set a new size for a Deployment, ReplicaSet, Repli
  autoscale      Auto-scale a Deployment, ReplicaSet, or Replicatio

Cluster Management Commands:
  certificate    Modify certificate resources.
  cluster-info   Display cluster info
  top            Display Resource (CPU/Memory/Storage) usage.
  cordon         Mark node as unschedulable
  uncordon       Mark node as schedulable
  drain          Drain node in preparation for maintenance
  taint          Update the taints on one or more nodes
```

06.

INTRODUCTION À KUBECTL

CONTEXTES ET NAMESPACES

CONTEXTES ET NAMESPACES

FONCTIONNEMENT DES NAMESPACES

- **Les namespaces fournissent une séparation logique des ressources, par exemple :**
  - Par utilisateurs
  - Par projet / application
  - Etc.
- **Les objets existent au sein de leur namespace uniquement**
- **Permet d'éviter les collisions de noms**

```
Basic Commands (Beginner):
  create         Create a resource from a file or from stdin.
  expose         Take a replication controller, service, deployment
  run            Run a particular image on the cluster
  set            Set specific features on objects
  run-container  Run a particular image on the cluster. This comman

Basic Commands (Intermediate):
  get            Display one or many resources
  explain        Documentation of resources
  edit           Edit a resource on the server
  delete         Delete resources by filenames, stdin, resources an

Deploy Commands:
  rollout        Manage the rollout of a resource
  rolling-update Perform a rolling update of the given ReplicationC
  scale          Set a new size for a Deployment, ReplicaSet, Repli
  autoscale      Auto-scale a Deployment, ReplicaSet, or Replicatio

Cluster Management Commands:
  certificate    Modify certificate resources.
  cluster-info   Display cluster info
  top            Display Resource (CPU/Memory/Storage) usage.
  cordon         Mark node as unschedulable
  uncordon       Mark node as schedulable
  drain          Drain node in preparation for maintenance
  taint          Update the taints on one or more nodes
```

CONTEXTES ET NAMESPACES

FONCTIONNEMENT DES NAMESPACES

- **Toutes les commandes acceptent le paramètre -n / --namespace**

- **Il est possible de créer un contexte dans la configuration de kubectl pour ne pas avoir à préciser le namespace à chaque commande**

```
Basic Commands (Beginner):
  create         Create a resource from a file or from stdin.
  expose         Take a replication controller, service, deployment
  run            Run a particular image on the cluster
  set            Set specific features on objects
  run-container  Run a particular image on the cluster. This comman

Basic Commands (Intermediate):
  get            Display one or many resources
  explain        Documentation of resources
  edit           Edit a resource on the server
  delete         Delete resources by filenames, stdin, resources an

Deploy Commands:
  rollout        Manage the rollout of a resource
  rolling-update Perform a rolling update of the given ReplicationC
  scale          Set a new size for a Deployment, ReplicaSet, Repli
  autoscale      Auto-scale a Deployment, ReplicaSet, or Replicatio

Cluster Management Commands:
  certificate    Modify certificate resources.
  cluster-info   Display cluster info
  top            Display Resource (CPU/Memory/Storage) usage.
  cordon         Mark node as unschedulable
  uncordon       Mark node as schedulable
  drain          Drain node in preparation for maintenance
  taint          Update the taints on one or more nodes
```

TP « INTRODUCTION À KUBECTL »

07.

DÉPLOYER SA PREMIÈRE APPLICATION

+ INTRODUCTION AUX PODS
+ QUE METTRE DANS UN POD ?
+ CRÉER UN POD AVEC KUBECTL
+ INTRODUCTION AUX FICHIERS DE MANIFEST
+ METADATA, LABELS, ANNOTATIONS

07.

DÉPLOYER SA PREMIÈRE APPLICATION

INTRODUCTION AUX PODS

INTRODUCTION AUX PODS

- **Plus petite unité orchestrable dans Kubernetes**
- **Un pod est un ensemble indissociable de un ou plusieurs conteneurs**
- **Les conteneurs d'un pod sont orchestrés sur un même hôte**

SPÉCIFICATIONS PARTAGÉES ENTRE LES PODS ET LEURS CONTENEURS

+ Stack ip (network namespace)
+ Communication inter-process (PID namespace)
+ Volumes

07.

DÉPLOYER SA PREMIÈRE APPLICATION

QUE METTRE DANS UN POD ?

QUE METTRE DANS UN POD ?

- **Quelques questions à se poser pour déterminer si deux conteneurs doivent cohabiter dans un pod :**
  - Les conteneurs doivent-ils passer à l'échelle ensemble ?
  - Les conteneurs accèdent-ils à des ressources locales communes ?
  - De manière générale, les deux conteneurs fonctionneront ils si ils sont démarrés sur des machines différentes ?
- **Si la réponse à cette dernière question est « non », il faudra probablement composer les conteneurs dans un même pod.**

07.

DÉPLOYER SA PREMIÈRE APPLICATION

CRÉER UN POD AVEC KUBECTL

CRÉER UN POD AVEC KUBECTL

KUBECTL RUN

- **La commande kubectl run permet d'instancier des conteneurs avec une interface similaire à docker run**

```
kubectl run busybox --image=busybox --restart=Never --tty
-i --generator=run-pod/v1 --env "POD_IP=10.42.3.147"
```

CRÉER UN POD AVEC KUBECTL

UTILISATION DES MANIFESTES

- **Généralement, on créé les pods ainsi que toutes les ressources en utilisant des manifestes**

```
kubectl create -f mespods.yml
```

- **On peut également utiliser la commande get avec un manifeste**
- **La commande apply, très utile, permet une création idempotente**

07.

DÉPLOYER SA PREMIÈRE APPLICATION

INTRODUCTION AUX FICHIERS MANIFESTE

INTRODUCTION AUX FICHIERS MANIFESTE

PRINCIPE

- **Toutes les ressources Kubernetes sont stockées sous la forme de manifestes**
- **Ces manifestes servent à réaliser toutes les opérations CRUD sur les ressources, via l'api-server**
- **Les manifestes peuvent être écrits en json ou en yaml**
- **On utilise généralement le yaml pour sa simplicité d'écriture et de lecture**

INTRODUCTION AUX FICHIERS MANIFESTE

EXEMPLE

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

INTRODUCTION AUX FICHIERS MANIFESTE

SECTIONS IMPORTANTES

- apiVersion : **certaintes ressources ne sont disponibles que dans certaines versions de l'api, par exemple** extensions/v1beta1

- kind : **le type de ressource décrite (pod, service, replicaset etc.)**

- metadata : **les méta-données associées à la ressource, dont notamment les labels**

- spec : **la description de la ressource en elle-même, le contenu de cette section dépend du type de ressource créée**

INTRODUCTION AUX FICHIERS MANIFESTE

SECTIONS IMPORTANTES

- containers **: 1 à N descriptions de conteneurs à créer dans le pod**
- nodeSelector **: liste de labels utilisés par le scheduler pour choisir sur quel nœud déployer le pod**
- restartPolicy **: indique comment gérer le cycle de vie du pod en cas d'arrêt de ce dernier (Always, Never, OnFailure)**
- imagePullSecrets **: liste de secrets à utiliser pour s'authentifier auprès du dépôt docker**

07.

DÉPLOYER SA PREMIÈRE APPLICATION

METADATA, LABELS, ANNOTATIONS

METADATA, LABELS, ANNOTATIONS

PRINCIPE DE FONCTIONNEMENT

- **Le scheduler de Kubernetes utilise les labels et les annotations dans toutes les situations de planification**
- **La bonne utilisation des labels est importante pour la panification des pods sur les nœuds**
- **Elle est cruciale lors de la création de ReplicaSets et de Deployments (cf chapitre 11)**

METADATA, LABELS, ANNOTATIONS

EXEMPLE

- **Ajouter un label sur un nœud**

```
kubectl label nodes minikube realm=web
```

- **Indiquer le label dans la section spec du pod**

```yaml
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    realm: web
```

METADATA, LABELS, ANNOTATIONS

ANNOTATIONS

- **En plus des labels, Kubernetes permet d'ajouter des annotations dans les méta-données de toutes les ressources**

- **Les annotations sont souvent définies et interprétées par certains controllers pour configurer des comportements particuliers**

- **Exemple : spécifier quel ingress controller utiliser (cf chapitre 10)**

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: nginx
5   annotations:
6     kubernetes.io/ingress.class: "external"
```

TP « CRÉER UNE PREMIÈRE APPLICATION »

08.

SUIVI ET DIAGNOSTIC D'UN POD EN FONCTIONNEMENT

+ LISTER LES PODS ET AFFICHER LES DÉTAILS
+ ACCÈS AUX LOGS
+ EXÉCUTER DES COMMANDES À L'INTÉRIEUR DU POD
+ ÉCHANGER DES FICHIERS AVEC LE POD
+ SONDES DE DISPONIBILITÉ ET DE BONNE SANTÉ

08.

SUIVI ET DIAGNOSTIC D'UN POD EN FONCTIONNEMENT

LISTER LES PODS ET AFFICHER LES DÉTAILS

LISTER LES PODS ET AFFICHER LES DÉTAILS

RAPPELS COMMANDE GET

- **Lister toutes les ressources dans le namespace du contexte courant**

```
kubectl get all
```

- **Lister toutes les ressources de tous les namespaces**

```
kubectl get all --all-namespaces
```

- **Lister tous les pods d'un namespace donné**

```
kubectl get pod -n monnamespace
```

LISTER LES PODS ET AFFICHER LES DÉTAILS

FORMATS DE SORTIE DE LA COMMANDE GET

- **Par défaut, seule la liste de ressources est retournée**

```
kubectl get all
```

- **Lister tous les pods du namespace courant au format yaml**

```
kubectl get all -o yaml
```

- **Afficher le manifeste d'un pod**

```
kubectl get pod monpod -o yaml
```

08.

SUIVI ET DIAGNOSTIC D'UN POD EN FONCTIONNEMENT

ACCÈS AUX LOGS

ACCÈS AUX LOGS

COMMANDE LOGS

- **Les logs des pods en fonctionnement sont directement accessibles depuis kubectl**

```
kubectl logs monpod
```

- **La commande logs propose une interface similaire à tail**

```
kubectl logs -f --tail=500 monpod
```

ACCÈS AUX LOGS

COMMANDE LOGS

- **Il est également possible d'obtenir les logs de tous les pods correspondant à un label**

```
kubectl logs -l run=hello-minikube
```

- **--since permet d'obtenir les logs générés depuis N secondes**

```
kubectl logs -l run=hello-minikube --since=10s
```

ACCÈS AUX LOGS

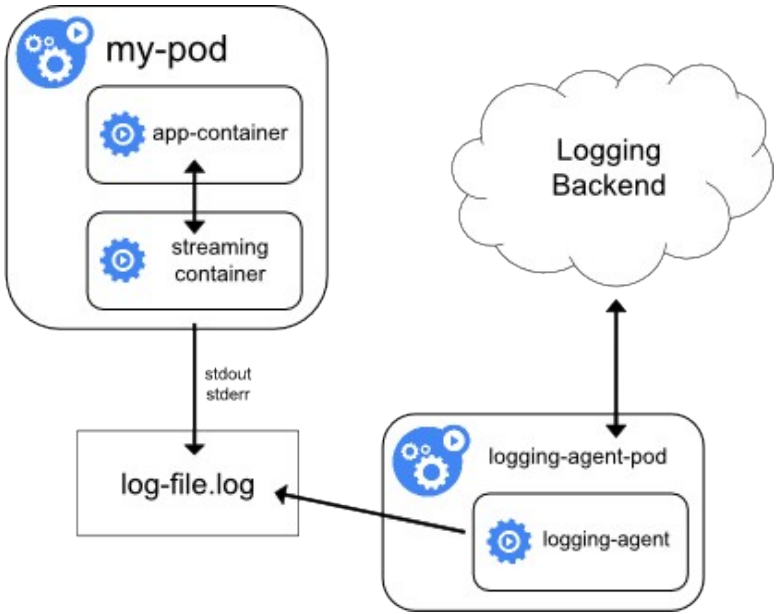
GESTION DES LOGS DANS KUBERNETES

- **Kubernetes collecte tous les logs docker via un log driver**
- **Le driver par défaut collecte toutes les sorties standard des conteneurs (stdout et stderr)**
- **Les logs collectés sont stockés au format json sur le nœud**
- **Kubernetes ne gère pas la rotation des logs ni la centralisation de la collecte**

ACCÈS AUX LOGS

GESTION DES LOGS DANS KUBERNETES

- **Exemple d'une architecture de collecte de logs avec sidecar et agent de collecte**



08.

SUIVI ET DIAGNOSTIC D'UN POD EN FONCTIONNEMENT

EXÉCUTER DES COMMANDES À L'INTÉRIEUR DU POD

EXÉCUTER DES COMMANDES À L'INTÉRIEUR D'UN POD

- **Kubectl permet d'exécuter des commandes à l'intérieur d'un conteneur**

```
➢ kubectl exec monpod echo hello
hello
```

- **On peut de cette manière exécuter un terminal intéractif à l'intérieur d'un conteneur**

```
kubectl exec -it monpod /bin/bash
```

- **Attention cependant, certaines images très minimalistes ne contiennent pas d'émulateur de terminal**

EXÉCUTER DES COMMANDES À L'INTÉRIEUR D’UN POD

- **Pour les pods contenant plusieurs conteneurs, il faut spécifier le conteneur**

```
➢ kubectl exec monpod -c moncontenur echo hello
hello
```

EXÉCUTER DES COMMANDES À L'INTÉRIEUR D'UN POD

- **Il est souvent nécessaire de vérifier si un pod écoute bien sur un port donné**
- **La configuration de l'accès réseau est un sujet en soit, mais il est possible de prendre un raccourci avec kubectl, à des fins de diagnostic**
- **La commande port-forward expose un port sur le client qui redirige vers un port du pod**

```
➢ kubectl port-forward monpod 8080:8080
➢ curl localhost:8080
```

08.

SUIVI ET DIAGNOSTIC D'UN POD EN FONCTIONNEMENT

ÉCHANGER DES FICHIERS AVEC LE POD

ÉCHANGER DES FICHIERS AVEC LE POD

- **Il est possible de copier un fichier de votre poste vers un conteneur du pod**
- **L'interface est similaire à scp**

```
kubectl cp ~/monfichier monpod:/chemin/destination -c monconteneur
```

- **L'opération inverse est également possible**

```
kubectl cp monpod:/chemin/source ~/destination -c monconteneur
```

- **Attention, pour que la commande** cp **fonctionne, il est nécessaire que l'utilitaire** tar **soit installé dans les conteneurs ciblés**

08.

SUIVI ET DIAGNOSTIC D'UN POD EN FONCTIONNEMENT

SONDES DE DISPONIBILITÉ ET DE BONNE SANTÉ

SONDES DE DISPONIBILITÉ ET DE BONNE SANTÉ

- **Les sondes sont les yeux du système d'orchestration, elles ont un rôle très important**
- **Deux types de sondes doivent être définies pour les pods**
  - Les sondes de bonne santé (liveliness)
  - Les sondes de disponibilité (readiness)
- **La qualité de l'orchestration découle en partie de la qualité de ces sondes, il faut donc leur apporter un soin particulier**

SONDES DE DISPONIBILITÉ ET DE BONNE SANTÉ

- **Vérifier l'existence d'un processus ne suffit pas**
- **Un serveur HTTP peut par exemple être lancé, mais mal configuré et inapte à servir des requêtes**
- **La sonde de bonne santé doit vérifier le bon fonctionnement de l'application**

SONDE LIVELINESS

+ Vérifie le bon fonctionnement de l'application
+ Requête HTTP, TCP ou exécution d'une commande
+ Les conteneurs qui échouent le test de **liveliness** sont **redémarrés**

SONDES DE DISPONIBILITÉ ET DE BONNE SANTÉ

- **Certaines applications sont parfois temporairement inaptes à servir des requêtes**
- **Une application peut par exemple mettre un certain temps à démarrer**
- **Dans ce cas, il n'est pas pertinent de la redémarrer, mais il ne s'agit pas de lui transmettre des requêtes non plus**

SONDE READINESS

+ Vérifie qu'une application soit prête à servir des requêtes
+ Requête HTTP, TCP ou exécution d'une commande
+ Les conteneurs qui échouent le test de **readiness** sont **exclus du load-balancing**

SONDES DE DISPONIBILITÉ ET DE BONNE SANTÉ

- Exemple de sonde liveliness HTTP
- Dans le cas suivant, l'application doit répondre 200 à l'adresse /healthz si elle est en bonne santé
- La réponse doit être conditionnée à un vrai test d'état

```
spec:
  containers:
  ...
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

TP « SUIVI ET DIAGNOSTIC »

09.

ACCÈS RÉSEAU AUX APPLICATIONS

+ COMPRENDRE LE RÉSEAU VIRTUEL DE KUBERNETES
+ PRINCIPE DE CNI, EXEMPLE DE FLANNEL ET CALICO
+ RENDRE SON SERVICE ACCESSIBLE SUR LE RÉSEAU VIRTUEL
+ INTRODUCTION AUX SERVICES ET LEURS DIFFÉRENTS TYPES

09.

ACCÈS RÉSEAU AUX APPLICATIONS

COMPRENDRE LE RÉSEAU VIRTUEL DE KUBERNETES

RÉSEAU VIRTUEL DE KUBERNETES

PROBLÉMATIQUE

- **Du point de vue réseau, plusieurs problématiques sont à résoudre :**
  - Communication inter-conteneurs au sein d'un pod
  - Communication inter-pods
  - Communication pod -> service
  - Communication externe -> service
- **Contrairement à Docker Swarm par exemple, Kubernetes nécessite une couche réseau supplémentaire pour fonctionner**

RÉSEAU VIRTUEL DE KUBERNETES

PROBLÉMATIQUE

- **Concernant la communication intra-pod :**
  - Les conteneurs dans un même pod partagent la même adresse ip
  - Ils peuvent donc communiquer les uns avec les autres sur localhost
- **La communication externe -> service <- pod s'appuie sur la communication inter-pods**
- **Reste donc à trouver implémenter une solution pour la communication inter-pods pour résoudre toutes les problématiques d'un coup**

RÉSEAU VIRTUEL DE KUBERNETES

PROBLÉMATIQUE

- **Kubernetes impose des pré-requis réseaux qu'il s'agit de remplir pour qu'un cluster fonctionne :**
  - Tous les conteneurs doivent pouvoir communiquer avec tous les autres sans NAT
  - Tous les nœuds doivent pouvoir communiquer avec tous les conteneurs sans NAT
  - Tous les conteneurs doivent pouvoir communiquer avec tous les nœuds sans NAT
  - L'adresse IP vue par un conteneur doit être la même que celle vue par les autres

RÉSEAU VIRTUEL DE KUBERNETES

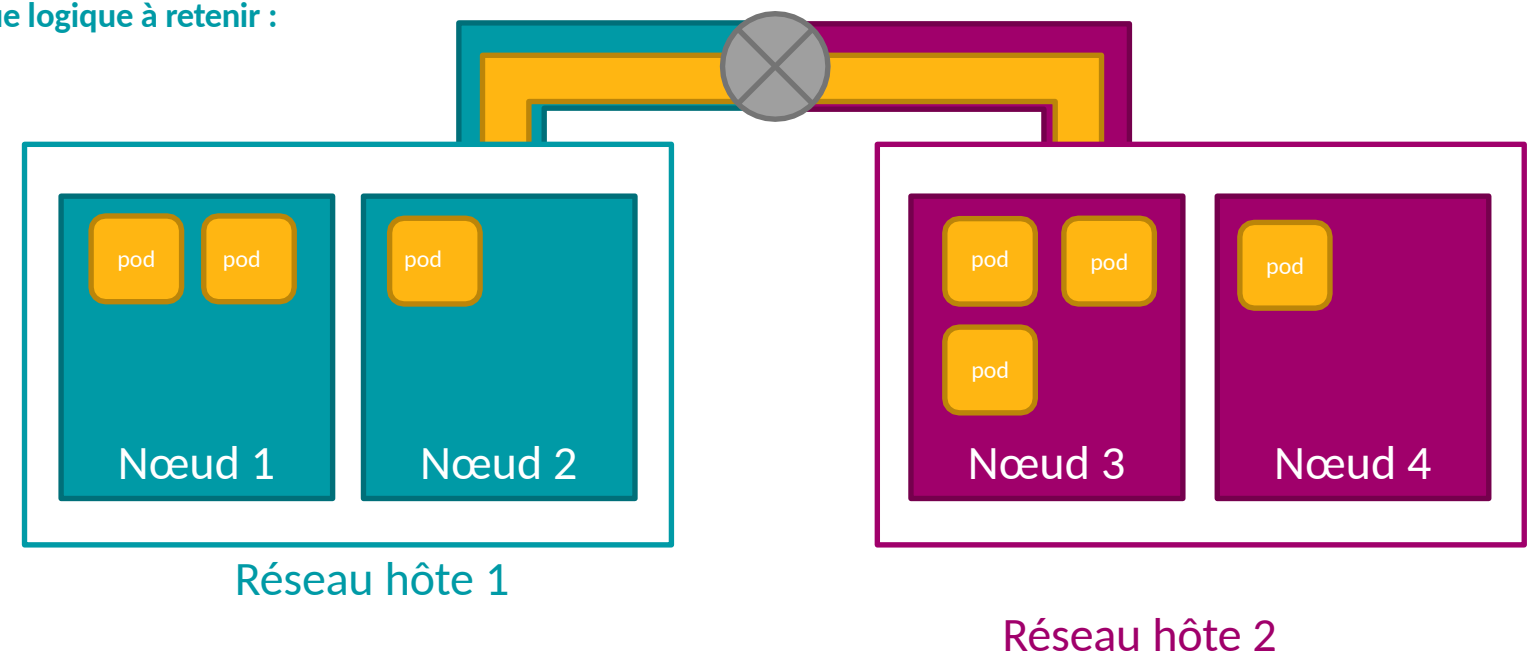
SOLUTIONS POSSIBLES

- **Implémenter cet ensemble de contraintes « manuellement » est une tâche complexe**
- **Un s'appuiera donc sur des solutions packagées qui viennent créer la couche réseau nécessaire au fonctionnement de Kubernetes**
- **Ces solutions sont de natures différentes, elles peuvent s'appuyer par exemple sur l'exploitation de VXLANs**

RÉSEAU VIRTUEL DE KUBERNETES

SOLUTIONS POSSIBLES

Vue logique à retenir :



09.

ACCÈS RÉSEAU AUX APPLICATIONS

PRINCIPE DE CNI, EXEMPLE DE FLANNEL ET CALICO

PRINCIPE DE CNI, FLANNEL ET CALICO

CONTAINER NETWORK INTERFACE

- **Standard de la CNCF**
- **Définit une interface entre l'hôte et l'orchestrateur de conteneurs permettant d'ajouter et de supprimer des réseaux**
- **Kubernetes fait appel à cette interface à la création des pods, pour créer les réseaux correspondant**
- **Kubernetes ne fournit pas l'implémentation de l'interface CNI, c'est l'opérateur du cluster qui installe un plugin CNI responsable de l'exécution des requêtes**

PRINCIPE DE CNI, FLANNEL ET CALICO

CONTAINER NETWORK INTERFACE

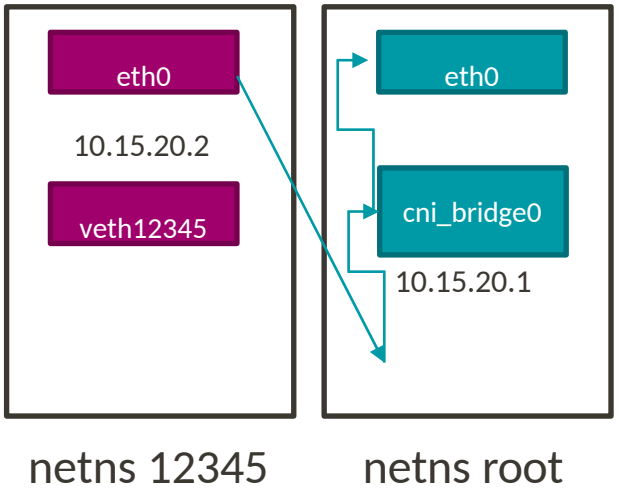
- **La spécification de l'interface, adoptée en 2017, est quasiment triviale**
- **4 commandes possibles :**
  - GET : récupère l'état d'une interface réseau
  - ADD : créé une interface
  - DEL : supprime une interface
  - VERSION : renvoie la liste des versions de l'interface CNI supportées par le plugin

PRINCIPE DE CNI, FLANNEL ET CALICO

CONTAINER NETWORK INTERFACE

- **Déroulement de l'ajout d'un pod et de la configuration du réseau par le plugin CNI (exemple avec le plugin** bridge**) :**

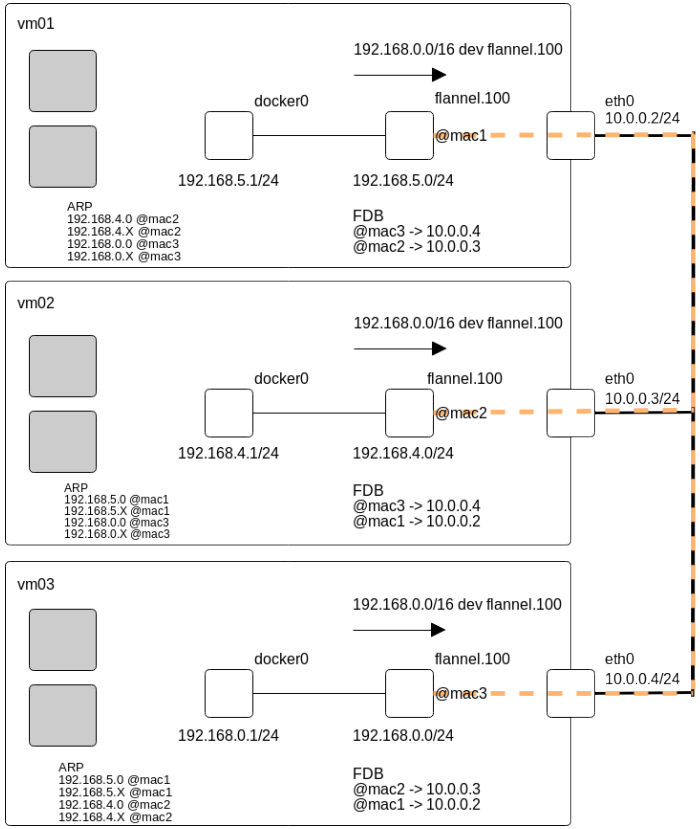
  - Kubernetes créé le conteneur 12345
  - Un namespace réseau du même nom est créé
  - Kubernetes envoie une requête ADD à l'interface CNI
  - Le plugin créé une interface cni_bridge0 si elle n'existe pas déjà
  - Le plugin créé une paire d'interfaces veth dans le namespace du conteneur
  - Le plugin déplace une des extrémités de la paire dans le namespace racine
  - Le plugin assigne des IPs aux interfaces
  - Le plugin met en place le routage veth -> bridge



PRINCIPE DE CNI, FLANNEL ET CALICO

CONTAINER NETWORK INTERFACE

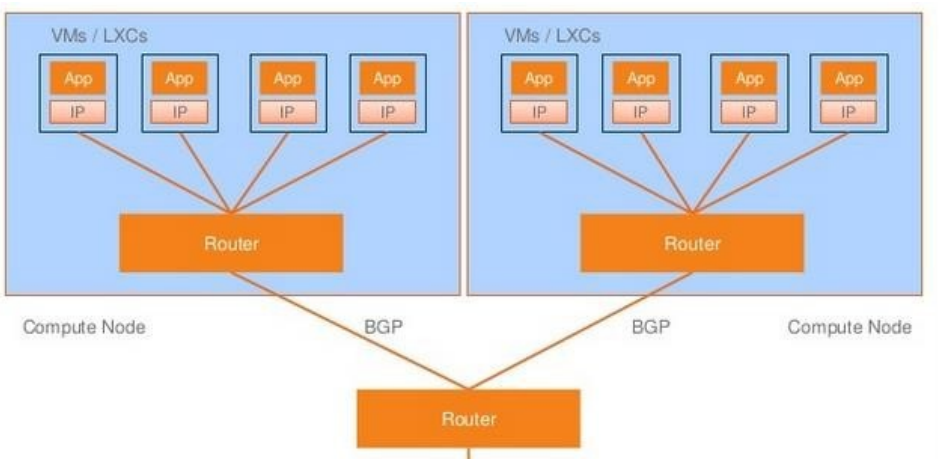
- **Flannel est une des solutions d'overlay réseau les plus utilisées**
- **Flannel s'interface avec le bridge docker0 par défaut de docker et relie les hôtes en utilisant des VXLAN**
- **Solution très simple mais qui peut souffrir de l'impact de performances des VXLAN**



PRINCIPE DE CNI, FLANNEL ET CALICO

CONTAINER NETWORK INTERFACE

- **Calico a une approche très différente et propose l'utilisation de BGP plutôt que des VXLAN**
- **Les nœuds deviennent des AS BGP et annoncent des préfixes qui correspondent aux conteneurs**
- **Solution très performante car n'impliquant aucune encapsulation**



09.

ACCÈS RÉSEAU AUX APPLICATIONS

RENDRE SON SERVICE ACCESSIBLE SUR LE RÉSEAU VIRTUEL

RENDRE SON SERVICE ACCESSIBLE SUR LE RÉSEAU VIRTUEL

- **Le premier niveau d'ouverture réseau se fait au niveau du conteneur**
- **Par défaut, si l'application dans un conteneur écoute sur un port, celui-ci ne sera accessible que depuis l'intérieur du conteneur (ou depuis les autres conteneurs du pod)**
- **Pour exposer un port sur le réseau virtuel il faut déclarer le port au niveau de la spécification du conteneur**

RENDRE SON SERVICE ACCESSIBLE SUR LE RÉSEAU VIRTUEL

- **Les ports sont à déclarer dans la section** ports
- **Chaque port peut être nommé, un port doit avoir un nom unique dans le pod**
- **Il est possible de spécifier le protocol (UDP ou TCP)**
- **Grâce au principe de fonctionnement du réseau virtuel plusieurs pods peuvent écouter sur le même port, chacun ayant sa propre adresse IP**

```yaml
spec:
  containers:
  - name: my-nginx
    image: nginx
    ports:
    - name: http
      containerPort: 80
      protocol: TCP
```

09.

ACCÈS RÉSEAU AUX APPLICATIONS

INTRODUCTION AUX SERVICES ET LEURS DIFFÉRENTS TYPES

LES SERVICES ET LEURS DIFFÉRENTS TYPES

PROBLÉMATIQUE

- **Exposer un port sur l'adresse IP d'un conteneur n'est généralement pas suffisant dans la plupart des cas d'utilisation**

- **On veut souvent pouvoir répondre à l'une des problématiques suivantes :**

  - Atteindre le port exposé depuis un autre pod
  - Atteindre le port exposé depuis un autre pod, même si les pods redémarrent et changent d'adresse IP
  - Exposer sur l'hôte un port correspondant à celui d'un pod
  - Exposer sur tous les hôtes un port correspondant à celui d'un pod
  - Atteindre le port exposé depuis l'extérieur du cluster

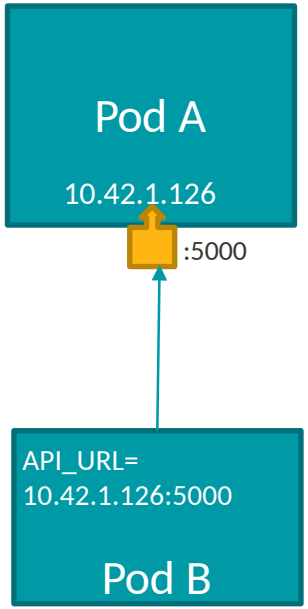
LES SERVICES ET LEURS DIFFÉRENTS TYPES

PROBLÉMATIQUE

- **Étant donné l'application hypothétique suivante constituée de deux pods :**
  - Le pod A expose une API Rest sur le port 5000
  - Le pod B consomme cette API Rest. Pour ce faire, il récupère l'adresse de l'API dans la variable d'environnement API_URL

- **De manière naïve, on lance le pod A, on récupère son adresse IP et on configure le pod B en utilisant cette adresse**

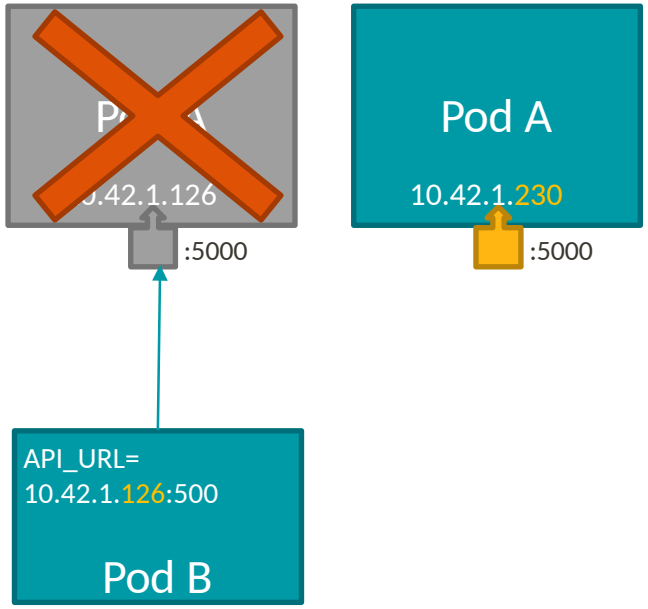
```
export API_URL=10.42.1.126:5000
```



LES SERVICES ET LEURS DIFFÉRENTS TYPES

PROBLÉMATIQUE

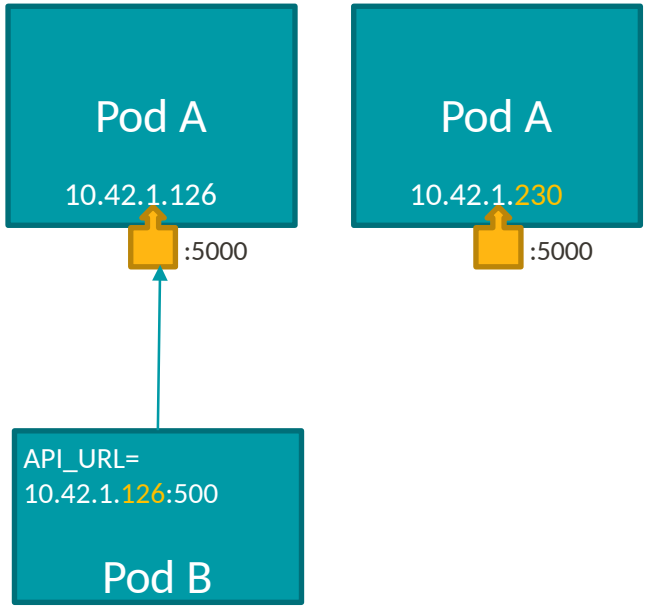
- **L'application fonctionnera un temps, mais des limites vont rapidement apparaître :**
  - Si le pod A meurt, il sera redémarré et changera d'IP. Le pod B ne fonctionnera plus



LES SERVICES ET LEURS DIFFÉRENTS TYPES

PROBLÉMATIQUE

- **L'application fonctionnera un temps, mais des limites vont rapidement apparaître :**
  - Si le pod A meurt, il sera redémarré et changera d'IP. Le pod B ne fonctionnera plus
  - Si le pod A monte en charge, il peut passer à l'échelle en étant répliqué.
  - Malheureusement dans ce cas, le pod B ne pointera que sur la première instance du pod A et la réplication est inutile.



LES SERVICES ET LEURS DIFFÉRENTS TYPES

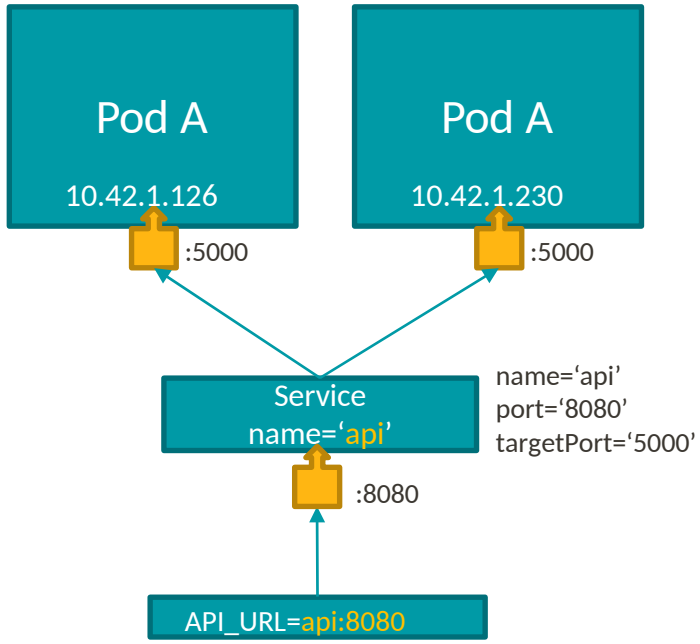
PROBLÉMATIQUE

- **Pour éviter ces travers, Kubernetes permet de créer des ressources de type Service**
- **Un Service est un objet réseau**
- **On associe un Service à un ensemble de pods par sélection de labels**
- **Le Service sert alors de façade aux différents pods et réalise un load-balancing simple entre ces derniers**
- **Grâce au kube-dns, le service obtient un nom dns unique qui permet de s'abstenir de manipuler des adresses IP**

LES SERVICES ET LEURS DIFFÉRENTS TYPES

EXEMPLE

- **En reprenant l'exemple précédent, on ajoute un service nommé 'api' qui sélectionne les pod par leur label**
- **Le service obtient lui même une adresse IP ainsi qu'un nom DNS**
- **Le nom DNS complet est** api.<namespace>.svc.cluster.local
- **Le pod B étant dans le même namespace, il peut utiliser le nom abbrégé** api

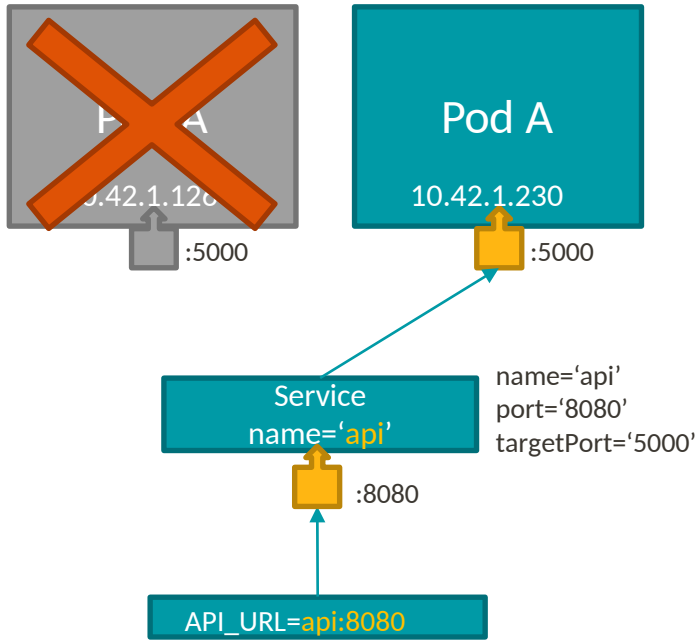


LES SERVICES ET LEURS DIFFÉRENTS TYPES

EXEMPLE

- **Dans cette situation, si l'un des réplicas du pod A meurt, cela n'a aucune conséquence sur le fonctionnement du pod B**

- **Le service détectera la panne via la sonde de readiness et exclura le réplica du load-balancing**



LES SERVICES ET LEURS DIFFÉRENTS TYPES

CRÉATION D'UN SERVICE

- **Pour créer un service, il faut à minima spécifier le** port **à exposer sur le service**
- **Le** targetPort **correspond au port exposé sur les pods**
- **Si** targetPort **est omis, il prend la valeur de** port
- **Le** selector **permet de sélectionner les pods sur lesquels envoyer les requêtes**

```yaml
apiVersion: v1
kind: Service
metadata:
  name: api
spec:
  type: ClusterIP
  ports:
  - port: 8080
    targetPort: 5000
    protocol: TCP
  selector:
    run: podA
```

LES SERVICES ET LEURS DIFFÉRENTS TYPES

TYPE DE SERVICE

- **Le type par défaut d'un Service est** ClusterIP
- **Il existe 4 types de Service :**
  - ClusterIP
  - NodePort
  - LoadBalancer
- **Chacun propose un fonctionnement différent, cette variété permet de créer des architectures très riches**

```yaml
apiVersion: v1
kind: Service
metadata:
  name: api
spec:
  type: ClusterIP
  ports:
  - port: 8080
    targetPort: 5000
    protocol: TCP
  selector:
    run: podA
```

LES SERVICES ET LEURS DIFFÉRENTS TYPES

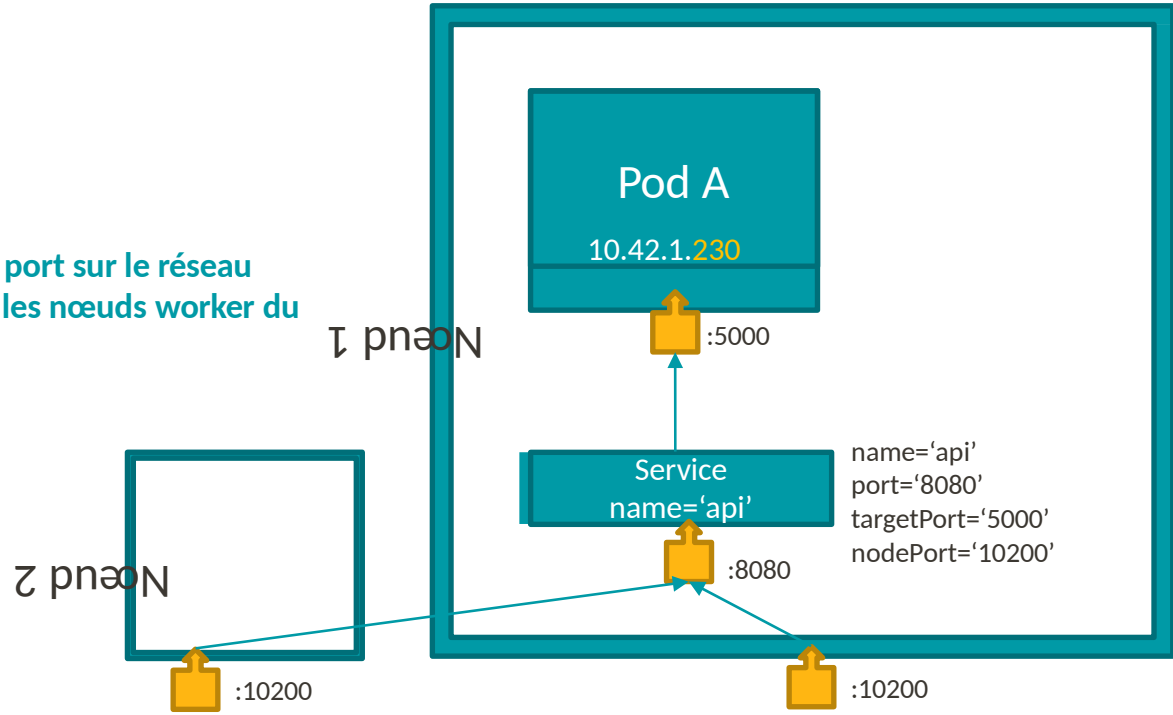
SERVICE CLUSTERIP

- **Service de type** ClusterIP
- **Comme vu précédemment, un service de type** ClusterIP **se voit attribué une adresse IP sur le réseau virtuel**
- **Il est donc atteignable uniquement depuis le réseau virtuel, au même titre qu'un pod**

LES SERVICES ET LEURS DIFFÉRENTS TYPES

SERVICE NODEPORT

- **Service de type** NodePort
- **Un service de ce type expose à la fois un port sur le réseau virtuel, mais également un port sur tous les nœuds worker du cluster**
- **Le diagramme ci-contre est une vue logique inexacte, c'est en réalité le kube-proxy qui est responsable du passage des requêtes**



LES SERVICES ET LEURS DIFFÉRENTS TYPES

SERVICE NODEPORT

- **Attention, si certains nœuds disposent d'une interface réseau munie d'une adresse IP publique, le nodePort exposé sera accessible publiquement**
- **Il est possible de spécifier sur quelle plage d'IP les nodePort ont le droit d'écouter avec l'option --nodeport-addresses**
- **Attention également, dans ce mode de fonctionnement les collisions de port sur les nœuds sont possibles et probables**

10.
ACCÈS RÉSEAU DEPUIS L’EXTÉRIEUR

+ UTILISER LES EXTERNALIPS
+ SERVICES DE TYPE LOADBALANCER
+ PRINCIPE DES INGRESS ET DE L'INGRESS \
  CONTROLLER
+ EXEMPLES D'INGRESS CONTROLLERS :
  NGINX, TRAEFIK, HAPROXY

ACCÈS RÉSEAU DEPUIS L'EXTÉRIEUR

PROBLÉMATIQUE

- **Pour rappel, les différents cas de communication réseau à couvrir sont les suivants :**
  - Communication inter-conteneurs au sein d'un pod
  - Communication inter-pods
  - Communication pod -> service
  - Communication externe -> service
- **Les 3 premiers sont couverts par l'utilisation des services tels que nous venons de l'aborder, reste à résoudre la problématique des communications entrantes**

10.

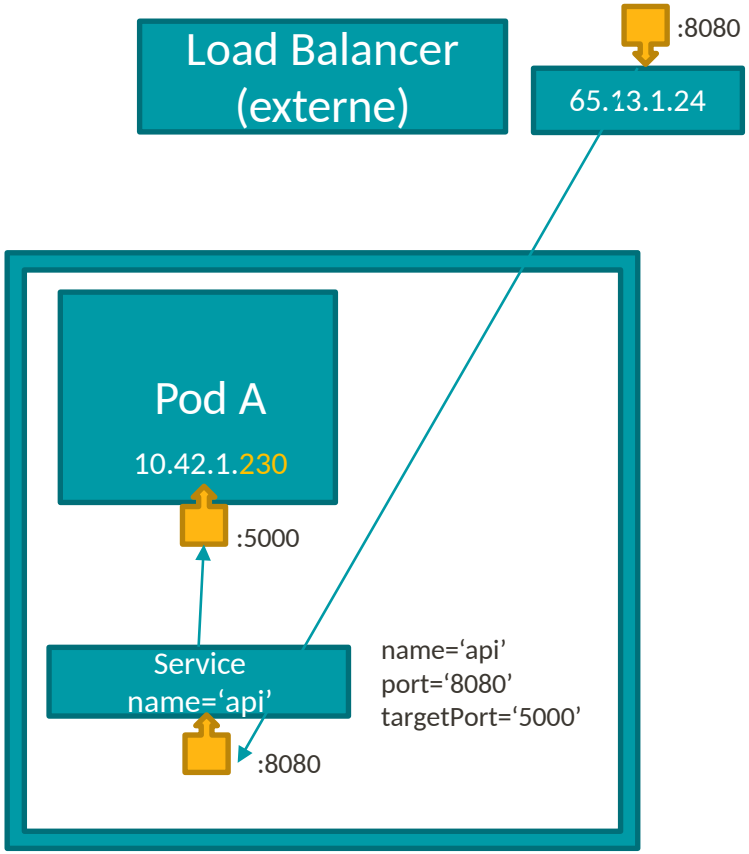
ACCÈS RÉSEAU DEPUIS L'EXTÉRIEUR

SERVICES DE TYPE LOADBALANCER

LES SERVICES ET LEURS DIFFÉRENTS TYPES

SERVICE LOADBALANCER

- **Service de type** LoadBalancer
- **Les services de type** LoadBalancer **font appel à un service externe au cluster pour obtenir une adresse ip (généralement publique) sur laquelle exposer le port**
- **Les grand cloud provider fournissent des instances de leur load balancer habituel avec un certain coût associé**
- **Sur des clusters Bare metal, des solutions existent : Metal LB**



10.

ACCÈS RÉSEAU DEPUIS L’EXTÉRIEUR

PRINCIPE DES INGRESS ET DE L'INGRESS CONTROLLER

ACCÈS RÉSEAU DEPUIS L'EXTÉRIEUR

PRINCIPE DES INGRESS ET DE L'INGRESS CONTROLLER

- **Pour résumer, nous avons pour l'instant 3 moyens d'exposer un pod sur l'extérieur :**
  - Service ClusterIP
  - Service NodePort avec un nœud dans le cluster ayant une adresse publique
  - Service de type LoadBalancer

ACCÈS RÉSEAU DEPUIS L'EXTÉRIEUR

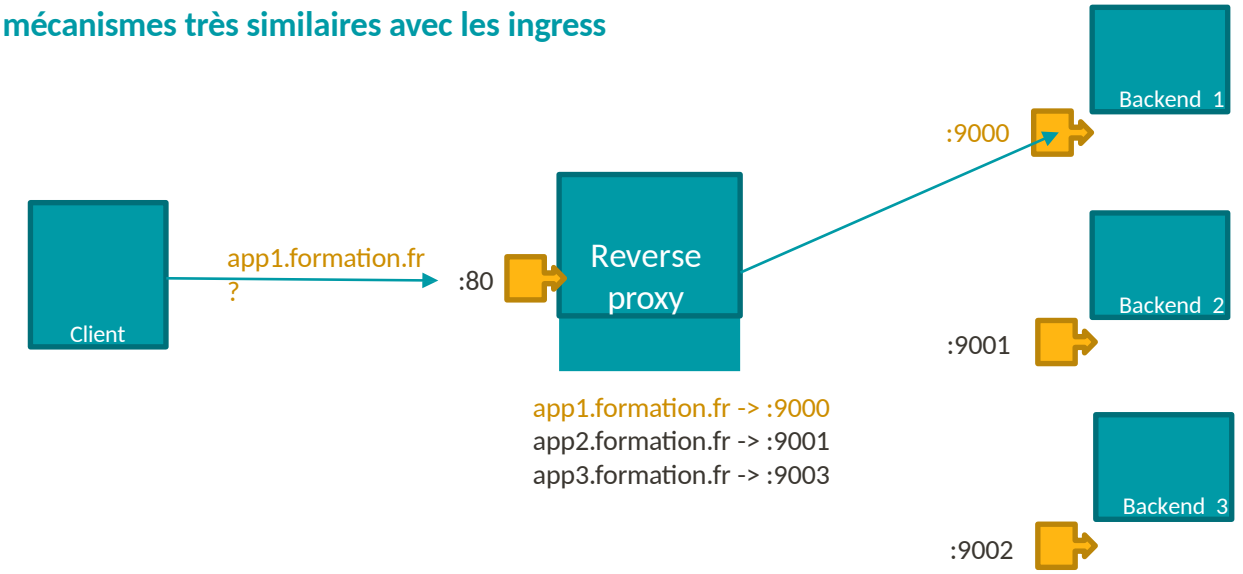
PRINCIPE DES INGRESS ET DE L'INGRESS CONTROLLER

- **Malheureusement, ces solutions ont des limites importantes:**
  - Utiliser des IP publiques sur les noeuds demande de la maintenance manuelle
  - Les IP publiques sont rares et chères, tous les nœuds du cluster ne peuvent pas en porter
  - Il est nécessaire de configurer un load-balancer en amont du cluster, et de maintenir la liste des IP publiques vers lesquelles transmettre les données
  - Les services de type LoadBalancer ont le mérite d'automatiser la gestion des IP publique
  - Reste le problème du coût de ces adresses
  - Une adresse IP publique par application n'est pas une approche soutenable du problème

ACCÈS RÉSEAU DEPUIS L'EXTÉRIEUR

PRINCIPE DES INGRESS ET DE L'INGRESS CONTROLLER

- **Cette problématique n'est pas propre à Kubernetes**
- **La manière habituelle de la régler est d'utiliser un reverse-proxy**
- **Kubernetes offre des mécanismes très similaires avec les ingress**

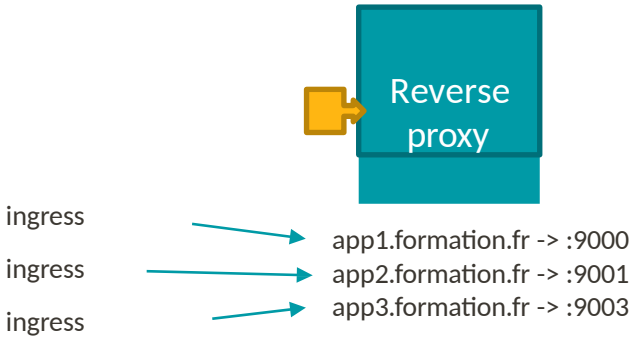


ACCÈS RÉSEAU DEPUIS L'EXTÉRIEUR

DÉFINITIONS

Un ingress correspond à une des règles d'un reverse proxy

Traduction littérale : « entrée »



INGRESS

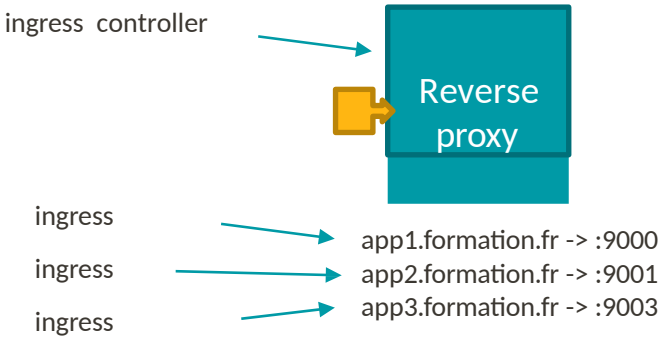
+ Un ingress = une règle de routage
+ Interpreté par un ingress controller
+ Créer un ingress = créer une porte d'entrée
+ Défini par un domaine et un backend correspondant

ACCÈS RÉSEAU DEPUIS L'EXTÉRIEUR

DÉFINITIONS

L'ingress controller correspond à l'application reverse proxy en elle-même

C'est lui qui « implémente » les ingress



INGRESS CONTROLLER

+ Consomme les ingress
+ Responsable de réaliser le routage
+ Reçoit le trafic en provenance de l'extérieur

ACCÈS RÉSEAU DEPUIS L'EXTÉRIEUR

CRÉER UN INGRESS

- **Exposer une application vers l'extérieur avec un ingress nécessite deux étapes :**
  - Créer un service ClusterIP pour l'application
  - Créer l'ingress en déclarant le service comme backend

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: gitlab
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: "letsencrypt-staging"

spec:
  tls:
  - hosts:
    - gitlab.formation.tech
    secretName: gitlab-ludo
  rules:
  - host: gitlab.ludovic.tech
    http:
      paths:
      - path: /
        pathType: Exact
        backend:
          service:
            name: kuard
            port:
              number: 80
```

ACCÈS RÉSEAU DEPUIS L'EXTÉRIEUR

CRÉER UN INGRESS

- **Les ingress offrent beaucoup de possibilités :**
  - Routage par nom de domaine
  - Routage par chemin
  - Mise en place de certificats TLS
  - etc.
- **Certaines options spécifiques à l'ingress controller sont généralement passées par des annotations**

```yaml
spec:
  tls:
  - hosts:
    - gitlab.formation.tech
    secretName: gitlab-ludo
  rules:
  - host: gitlab.ludovic.tech
    http:
      paths:
      - path: /
        pathType: Exact
        backend:
          service:
            name: app1
            port:
              number: 80
  - host: app2.ludovic.tech
    http:
      paths:
      - path: /
        pathType: Exact
        backend:
          service:
            name: app2
            port:
              number: 80
```

10.

ACCÈS RÉSEAU DEPUIS L'EXTÉRIEUR

EXEMPLES D'INGRESS CONTROLLERS : NGINX …

EXEMPLES D'INGRESS CONTROLLERS

NGINX

- **Nginx est une solution très fréquemment utilisée pour mettre en place des reverse-proxy**
- **Il n’est donc pas étonnant que ce soit également l’ingress controller de référence pour Kubernetes**

EXEMPLES D'INGRESS CONTROLLERS

NGINX

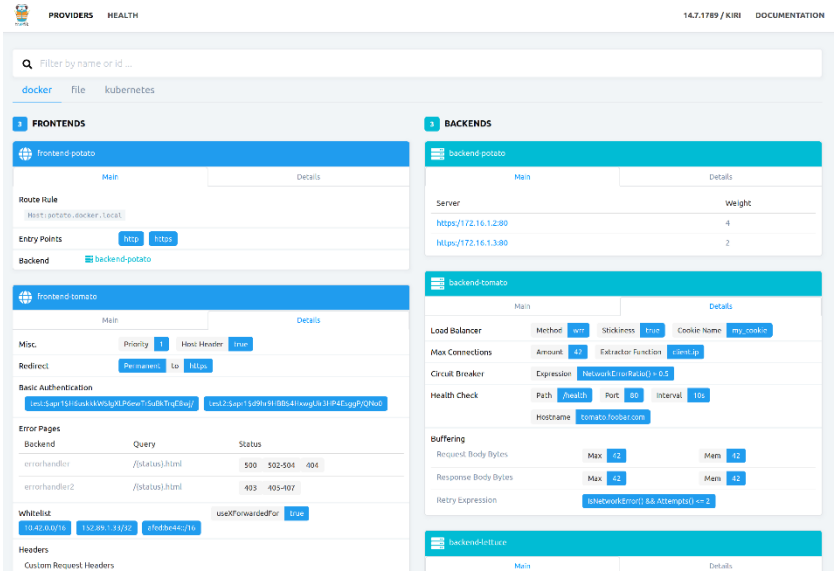
- **Nginx est une solution très fréquemment utilisée pour mettre en place des reverse-proxy**
- **Il n'est donc pas étonnant que ce soit également l'ingress controller de référence pour Kubernetes**
- **Nginx permet l'installation de modules tiers pour ajouter des fonctionnalités puissantes à l'ingress controller (intégration let's encrypt, annotations, etc.)**

EXEMPLES D'INGRESS CONTROLLERS

TRAEFIK

- **Traefik est une solution intégrée de load balancer et de reverse proxy également très utilisée**

- **Traefik propose de nombreuses fonctionnalités sans ajout de module tiers :**
  - Rate limiting
  - Intégration let’s encrypt
  - Authentification basic
  - Interface web de contrôle



TP « RÉSEAU, INGRESS ET INGRESS CONTROLLER »

11.

RÉPLICATION DES PODS

+ OBJECTIFS DE LA RÉPLICATION
+ PRINCIPE DE BOUCLE DE RÉCONCILIATION
+ CRÉER ET MANIPULER DES REPLICASET
+ PRINCIPE DE PASSAGE À L'ÉCHELLE HORIZONTAL (VS VERTICAL)
+ METTRE EN PLACE L'AUTO-SCALING HORIZONTAL
+ INTRODUCTION AUX DAEMONSET

11.

RÉPLICATION DES PODS

OBJECTIFS DE LA RÉPLICATION

RÉPLICATION DES PODS

OBJECTIFS DE LA RÉPLICATION

- **Un des avantages de Kubernetes est la possibilité de définir des objectifs de haute disponibilité qui seront maintenus automatiquement par le système**

- **De tels objectifs sont par exemple :**
  - Maintenir en permanence 3 instances de l'application A
  - Maintenir en permanence une instance par nœud de l'application B
  - Maintenir en permanence une instance par nœud portant le label gpu_enabled=true pour l'application C

RÉPLICATION DES PODS

OBJECTIFS DE LA RÉPLICATION

- **La réplication peut avoir différents objectifs fonctionnels :**
  - Redondance, pour augmenter la tolérance aux pannes
  - Passage à l'échelle, plus d'instances pour supporter une montée en charge
  - Fragmentation des systèmes de stockage ou de calcul distribués
- **Quelque soit l'objectif, Kubernetes permet de gérer la réplication avec des outils simples et puissants**

11.

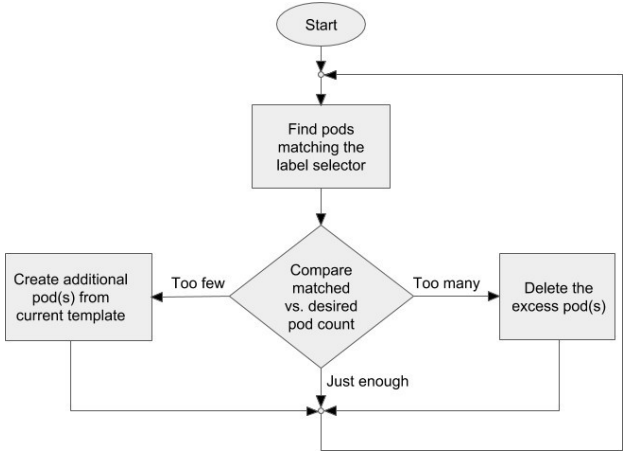
RÉPLICATION DES PODS

PRINCIPE DE BOUCLE DE RÉCONCILIATION

RÉPLICATION DES PODS

PRINCIPE DE BOUCLE DE RÉCONCILIATION

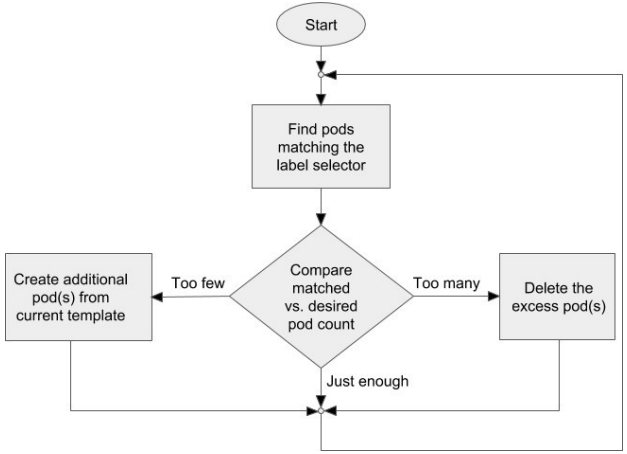
- **Kubernetes se base sur les notions d'**état observé **et d'**état désiré
- **Lors de la création d'une ressource (avec** kubectl apply **par exemple), c'est l'**état désiré **qui est mis à jour**
- **Autrement dit, l'utilisateur ne fait qu'émettre des « souhaits » au près de Kubernetes**



RÉPLICATION DES PODS

PRINCIPE DE BOUCLE DE RÉCONCILIATION

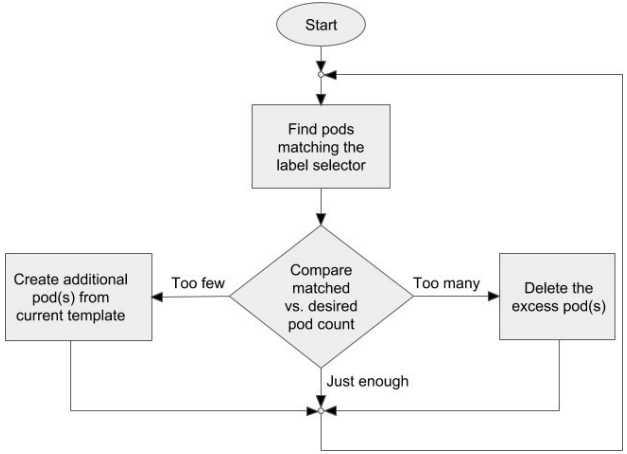
- **La boucle de réconciliation est une boucle infinie qui observe l'**état courant
- **L'**état courant **est comparé à l'**état désiré **à chaque passage de la boucle**
- **Les actions à mener sont déduites de la différence entre l'**état observé **et l'**état désiré**, puis implémentées**



RÉPLICATION DES PODS

PRINCIPE DE BOUCLE DE RÉCONCILIATION

- **Le système évolue en permanence**
- **L'utilisateur n'a pas besoin d'expliciter les actions à mener mais simplement de déclarer son objectif**
- **Le système se répare lui-même en cas de problème**



11.

RÉPLICATION DES PODS

CRÉER ET MANIPULER DES REPLICASET

CRÉER ET MANIPULER DES REPLICASET

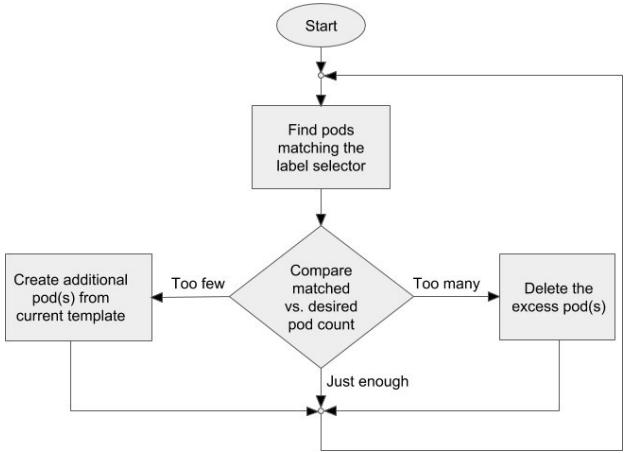
PROBLÉMATIQUE

- **Il n’est bien entendu pas question de créer des pods supplémentaires à la main pour gérer la réplication**
- **Les ReplicaSets sont les ressources de réplication les plus simples**

CRÉER ET MANIPULER DES REPLICASET

FONCTIONNEMENT

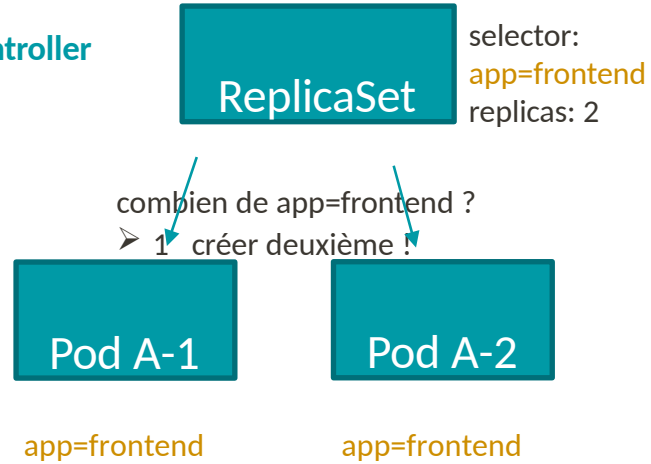
- **Un ReplicaSet est défini par trois composants principaux :**
  - Un template de pod à créer
  - Un nombre de réplicas requis
  - Un ensemble de labels permettant d'identifier les réplicas à gérer



CRÉER ET MANIPULER DES REPLICASET

POURQUOI UTILISER DES LABELS ?

- **Dans l'hypothèse où les ReplicaSets n'utilisaient pas de labels pour controller leurs réplicas, ils seraient obligés de « posséder » les pods**
- **Il deviendrait alors impossible de répliquer un pod existant sans le supprimer pour le recréer à l'intérieur d'un ReplicaSet**
- **Le fonctionnement par label permet de découpler les Pods de la notion de réplication**



CRÉER ET MANIPULER DES REPLICASET

CRÉER UN REPLICASET

- **La création d'un ReplicaSet est triviale une fois que l'on sait créer un pod**
  - replicas : indique le nombre de réplicas voulus
  - selector.matchLabels : indique les labels permettant de retrouver les réplicas
  - template : spécifications des pods à créer
- **Attention les labels du template doivent être identiques à ceux du selector**

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: monapp
  template:
    metadata:
      labels:
        app: monapp
    spec:
      containers:
      - name: application1
        image: application1:latest
        ports:
        - containerPort: 80
```

11.

RÉPLICATION DES PODS

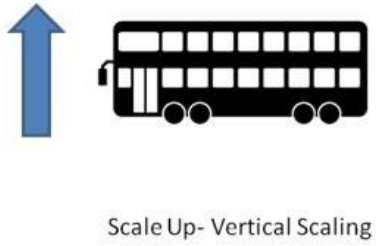
PRINCIPE DE PASSAGE À L'ÉCHELLE HORIZONTAL (VS VERTICAL)

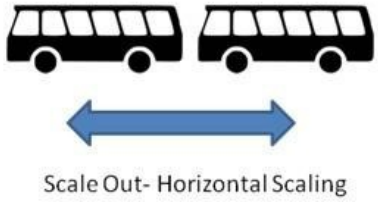
PRINCIPE DE PASSAGE À L'ÉCHELLE HORIZONTAL (VS VERTICAL)

Rappels

- **Vertical scaling** :
  ajout de ressources (RAM, CPU etc.) à des machines existantes dans un cluster

- **Horizontal scaling** :
  ajout de machines dans un cluster





PRINCIPE DE PASSAGE À L'ÉCHELLE HORIZONTAL (VS VERTICAL)

- **Kubernetes gère également les notions de scaling vertical / horizontal**
- **Le scaling vertical se fait par l'allocation de ressources aux pods**
- **Le scaling horizontal se fait par l'utilisation des ReplicaSet**

```
kubectl scale monpod --replicas=10
```

11.

RÉPLICATION DES PODS

METTRE EN PLACE L'AUTO-SCALING HORIZONTAL

AUTO-SCALING HORIZONTAL

- **Le fait de pouvoir faire passer une application à l'échelle en une ligne de commande est déjà confortable**

```
kubectl scale monpod --replicas=10
```

- **Il serait encore plus efficace que les applications passent à l'échelle automatiquement lorsque leur charge devient trop élevée**

- **C'est ce qu'il est possible de mettre en place avec l'autoscaling horizontal des ReplicaSets**

AUTO-SCALING HORIZONTAL

PRÉ-REQUIS

- **Avant de pouvoir mettre en place l'auto-scaling, il faut configurer nos pods en ajoutant une requête de ressources**
- **Les requêtes de ressources servent à indiquer les ressources minimums qui doivent être disponibles sur un nœud pour que le scheduler place le pod dessus**
- **Exemple :**
  - ressources: request:
    cpu: 100m
  - Au moins 10% d'un CPU doivent être non réservé sur un nœud pour que le pod soit placé dessus (100m = 100 milli-cpu)

AUTO-SCALING HORIZONTAL

MISE EN PLACE DE L'AUTO-SCALING

- **Comment souvent, l'interface est très simple**
- **La création se fait via kubectl**

```
kubectl autoscale monreplicaset --min=1 --max=5 --cpu-percent=50
```

- **Pour l'instant, seule la métrique de consommation CPU peut être surveillée**
- **La commande précédente créé une ressource HorizontalPodaudoScaler**

```
kubectl get hpa
kubectl describe monhpa
```

11.

RÉPLICATION DES PODS

INTRODUCTION AUX DAEMONSET

INTRODUCTION AUX DAEMONSET

PROBLÉMATIQUE

- **Les ReplicaSets permettent de définir un nombre de réplicas fixe ou de l'ajuster automatiquement avec un auto-scaler**

- **Dans certains cas, le nombre de réplicas n'a pas d'importance et il est simplement souhaitable d'avoir un réplica par nœud**

- **C'est le cas par exemple lors du déploiement d'un IngressController, du CNI, du Loadbalancer, etc ..**

- **Les DaemonSet permettent de répondre à cette problématique**

INTRODUCTION AUX DAEMONSET

FONCTIONNEMENT

- **Un DaemonSet s'appuie sur une boucle de contrôle et fonctionne de la même manière qu'un ReplicaSet**

- **Le nombre de réplicas d'un DaemonSet est implicite**
- **1 réplica sera créé et maintenu sur chaque nœud du cluster**
- **Il est toutefois possible de cibler les nœuds portant des labels spécifiques**

INTRODUCTION AUX DAEMONSET

CRÉATION D'UN DAEMONSET

- **Un DaemonSet s'appuie sur une boucle de contrôle et fonctionne de la même manière qu'un ReplicaSet**
- **Le nombre de réplicas d'un DaemonSet est implicite**
- **1 réplica sera créé et maintenu sur chaque nœud du cluster**
- **Il est toutefois possible de cibler les nœuds portant des labels spécifiques** (nodeSelector **dans le template de pod)**

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: demon
spec:
  selector:
    matchLabels:
      app: mondemon
  template:
    metadata:
      labels:
        app: mondemon
    spec:
      nodeSelector:
        realm: internal
      containers:
      - name: demon1
        image: demon1:latest
        ports:
        - containerPort: 80
```

TP « RÉPLICATION DES PODS »

12.

GÉRER LE CYCLE DE VIE D'UNE APPLICATION

+ PRINCIPE DE ROLLING-UPDATE
+ CRÉER ET GÉRER DES DÉPLOIEMENTS
+ STRATÉGIES ET MISE EN OEUVRE

CYLE DE VIE DES APPLICATIONS

PROBLÉMATIQUE

- **Nos applications fonctionnent en production, répliquées par des ReplicaSets et répondent aux pics de charge grâce aux auto-scalers**

- **Vient le moment de déployer une nouvelle version, comment faire ?**

- **Une solution naïve serait de supprimer les ReplicaSets et de les recréer avec une nouvelle image, provoquant une indisponibilité de service**

- **Bien entendu, Kubernetes a une solution plus élégante à nous proposer !**

12.

GÉRER LE CYCLE DE VIE D'UNE APPLICATION

PRINCIPE DE ROLLING-UPDATE

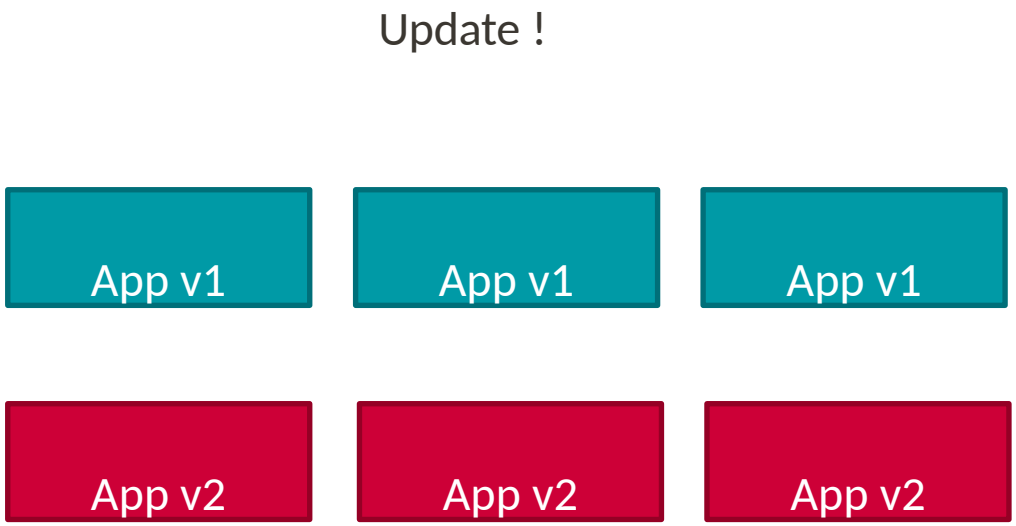
PRINCIPE DE ROLLING-UPDATE

DÉFINITION

- **Une rolling-update consiste à créer une partie des nouveaux pods avant de supprimer une partie des anciens**
- **De cette manière, l'application concernée n'est jamais indisponible pendant une mise à jour**
- **Si un service pointe vers les applications en question, il n'inclura les nouveaux pods que lorsqu'ils seront prêts grâce aux sondes de readiness**
- **Attention toutefois, deux versions de l'application vont vivre ensemble pendant un certain temps, il faut donc s'assurer de leur compatibilité (migrations de données, versions d'api etc.)**

PRINCIPE DE ROLLING-UPDATE

DÉFINITION



12.

GÉRER LE CYCLE DE VIE D'UNE APPLICATION

CRÉER ET GÉRER DES DÉPLOIEMENTS

CRÉER ET GÉRER DES DÉPLOIEMENTS

- **Les ressources de type** Deployment **permettent de réaliser des rolling-updates dans Kubernetes**
- **Un** Deployment **« encapsule » un** ReplicaSet
- **L'interface est quasiment identique à celle des** ReplicaSet
- **On remarque les paramètres supplémentaires** maxSurge **et** maxUnavailable

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

CRÉER ET GÉRER DES DÉPLOIEMENTS

MAXSURGE ET MAXUNAVAILABLE

- maxSurge **: nombre maximum de réplicas pouvant être créés en plus du nombre de réplicas configurés**
  - Avec replicas = 3 et maxSurge = 1
    on aura jusq'à 4 réplicas en même temps

- maxUnavailable **: nombre maximum de réplicas pouvant être retirés**
  - Avec replicas = 3 et maxUnavailable = 1
    on aura au minimum 1 réplica en fonctionnement pendant la mise à jour

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

12.

GÉRER LE CYCLE DE VIE D'UNE APPLICATION

STRATÉGIES ET MISE EN ŒUVRE

STRATÉGIES ET MISE EN ŒUVRE

RECREATE ET ROLLOUT

- **Il existe deux stratégies de déploiement configurable dans une ressource** Deployment

  - Rollout : la stratégie de déploiement en rolling-update que nous venons de décrire
  - Recreate : les pods du ReplicaSet sont tous détruits puis recréés

- **La stratégie Recreate peut sembler peu intéressante mais il est parfois nécessaire de l'utiliser (cas d'un montage de volume persistant en ReadWriteOnce par exemple)**

STRATÉGIES ET MISE EN ŒUVRE

DÉCLENCHER UNE ROLLING-UPDATE

- **La mise en œuvre d'une rolling-update se fait via kubectl**

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
```

- **Une fois la mise à jour déclenchée, il est possible de suivre sa progression**

```
kubectl rollout status deployment/nginx-deployment
```

- **Il est également possible de consulter l'historique des déploiements**

```
kubectl rollout history deployment/nginx-deployment
```

- **Enfin il est possible d'annuler une mise à jour pour revenir à une révision précédente**

```
kubectl rollout undo deployment/nginx-deployment --to-revision=2
```

TP « CYCLE DE VIE DES APPLICATIONS »

13.

LANCER DES TÂCHES À EXÉCUTION UNIQUE

+ PRINCIPE ET INTÉRÊT DES JOB
+ LANCER DES JOB
+ PARALLELISME ET WORK-POOLS

13.

LANCER DES TÂCHES À EXÉCUTION UNIQUE

PRINCIPE ET INTÉRÊT DES JOB

PRINCIPE ET INTÉRÊT DES JOB

LES 12 FACTORS

1. Base de code
2. Dépendances
3. Configuration
4. Services Externes
5. Build, Release, Run
6. Processus
7. Associations de ports
8. Concurrence
9. Jetable
10. Parité dev/prod
11. Logs
12. Processus d’administration

PRINCIPE ET INTÉRÊT DES JOB

DES PROCESSUS DE DIFFÉRENTS TYPES

Processus longue durée

- **Application web**
- **Base de données**

► **Se termine lorsque le processus n'est plus utile ou lors d'une mise à jour.**

Processus courte durée

- **Migration de base de données**
- **Préparation des dossiers d'une application**
- **Globalement tous les processus couverts par le chapitre XII du manifeste 12factor[1]**

► **Se termine à la fin de l'exécution**

PRINCIPE ET INTÉRÊT DES JOB

JOBS

- **Un objet Kubernetes**
- **En charge de la bonne exécution de la tâche**

Cycle de vie d'un objet Job :

- **Création**
- **Tourne jusqu'à la fin fructueuse de la tâche (exit 0)**
- **En cas d'échec (exit != 0)**

PRINCIPE ET INTÉRÊT DES JOB

PATRONS DE JOB

Table 10-1. Job patterns

Type	Use case	Behavior	completions	parallelism
One shot	Database migrations	A single pod running once until successful termination	1	1
Parallel fixed completions	Multiple pods processing a set of work in parallel	One or more Pods running one or more times until reaching a fixed completion count	1+	1+
Work queue: parallel Jobs	Multiple pods processing from a centralized work queue	One or more Pods running once until successful termination	1	2+

13.

LANCER DES TÂCHES À EXÉCUTION UNIQUE

LANCER DES JOB

LANCER DES JOB

AFFICHAGE D'UN JOB

- **Un job est un pod comme un autre**
- Il se configure à l'aide d'une spec
- Une fois lancé on accède à ses information avec kubectl
- On le cible à l'aide d'un sélecteur

```
$ kubectl get pod -l job-name=oneshot -a
NAME READY STATUS RESTARTS AGE oneshot-
0wm49
0/1
Error 0 1m oneshaot-6h9s2 0/1 Error 0 39s
oneshot-hkzw0 1/1 Running 0 6s oneshot-
k5swz  0/1
Error 0 28s oneshot-m1rdw 0/1 Error 0 19s
oneshot-x157b 0/1 Error 0 57s
```

LANCER DES JOB

LANCEMENT D'UN JOB

- **Un job oneshot peut se lancer avec kubectl run**
- On donne tous les éléments de lancement d'un pod
- On spécifie les arguments spécifiques au job

```
$ kubectl run -i oneshot \
  --image=gcr.io/kuar-demo/kuard-amd64:1 \
  --restart=OnFailure \
  -- --keygen-enable \
  --keygen-exit-on-complete \
  --keygen-num-to-gen 10
[...]
(ID 0) Workload starting  (ID 0 1/10) Item
done:
SHA256:nAsUsG54XoKRkJwyN+OShkUPKew3mwq7OCc
[...]
(ID 0 9/10) Item done:
SHA256:UOmYex79qqbI1MhcIfG4hDnGKonlsij2k3s
(ID 0 10/10) Item done:
SHA256:WCR8wIGOFag84Bsa8f/9QHuKqF+0mEnCADY
(ID 0) Workload exiting
```

13.

LANCER DES TÂCHES À EXÉCUTION UNIQUE

PARALLELISME ET WORK-POOLS

14.

DÉPLOIEMENT ET PARTAGE DES ÉLÉMENTS DE CONFIGURATION

+ PRINCIPE DE GÉNÉRICITÉ ET D'INDÉPENDANCE DE L'APPLICATION ET DE LA PLATEFORME
+ CRÉER DES CONFIGMAPS
+ DIFFÉRENTS MOYENS DE CONSOMMATION DES CONFIGMAPS
+ CRÉER ET CONSOMMER DES SECRETS

14.

DÉPLOIEMENT ET PARTAGE DES ÉLÉMENTS DE CONFIGURATION

PRINCIPE DE GÉNÉRICITÉ ET D'INDÉPENDANCE DE L'APPLICATION ET DE LA PLATEFORME

INDÉPENDANCE APPLICATION / PLATEFORME

RAPPELS CLOUD : LES 12 FACTEURS

- **12 règles de base à respecter pour créer des applications compatibles avec une approche Cloud**
- **Des principes généraux, les détails étant propres à chaque projet**

INDÉPENDANCE APPLICATION / PLATEFORME

RAPPELS CLOUD : LES 12 FACTEURS

1. Base de code
2. Dépendances
3. Configuration
4. Services Externes
5. Build, Release, Run
6. Processus

7 Associations de ports 8 Concurrence

9 Jetable

10 Parité dev/prod

11 Logs

12 Processus d'administration

INDÉPENDANCE APPLICATION / PLATEFORME

RAPPELS CLOUD : LES 12 FACTEURS

3 - Configuration

- **Ne pas stocker la configuration dans le code**
- **L'environnement de déploiement est responsable du stockage de la configuration (variables d'environnement)**

INDÉPENDANCE APPLICATION / PLATEFORME

PRINCIPE DANS KUBERNETES

- **Kubernetes permet de prendre le facteur 3 au mot grâce aux** ConfigMaps **et aux** Secrets

- **Ces ressources permettent de stocker les éléments de configuration de manière indépendante à l'intérieur du cluster**

- **Il n'est donc plus nécessaire de conserver plusieurs fichiers de configuration à l'intérieur des dépôts de code**

14.

DÉPLOIEMENT ET PARTAGE DES ÉLÉMENTS DE CONFIGURATION

CRÉER DES CONFIGMAPS

CRÉER DES CONFIGMAPS

MANIFESTE YAML

- **Il existe plusieurs manière de créer des** ConfigMaps
- **L'exemple ci-contre est un manifeste yaml qui créé un** ConfigMap **contenant deux clés (**game.properties **et** ui.properties**)**
- **Ces clés pourraient être consommées de différentes manières par les Pod, notamment sous forme de fichiers**

```
apiVersion: v1
data:
  game.properties: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
```

CRÉER DES CONFIGMAPS

AUTRES MÉTHODES DE CRÉATION

- **Il est également possible de créer un** ConfigMap **directement depuis un ou plusieurs fichiers avec kubectl**

```
kubectl create configmap maconfig --from-file=game.properties --from-file=ui.properties
```

- **On peut aussi simplement passer des couples clé/valeur directement à kubectl**

```
kubectl create configmap maconfig --from-literal=cle=valeur
```

CRÉER DES CONFIGMAPS

AUTRES MÉTHODES DE CRÉATION

- **Enfin il est possible de créer un** ConfigMap **en utilisant un fichier contenant des variables d'environnement**
- **Le fichier ne doit contenir que des couples** CLE=VALEUR**, un par ligne**

```
kubectl create configmap maconfig --from-env-file=config.env
```

- **Le** ConfigMap **aura alors la forme ci-contre :**

```
apiVersion: v1
data:
  allowed: '"true"'
  enemies: aliens
  lives: "3"
kind: ConfigMap
```

14.

DÉPLOIEMENT ET PARTAGE DES ÉLÉMENTS DE CONFIGURATION

DIFFÉRENTS MOYENS DE CONSOMMATION DES CONFIGMAPS

CONSOMMATION DES CONFIGMAPS

PROBLÉMATIQUE

- **Une fois créés, les** ConfigMaps **doivent être reliés aux Pods**
- **Il existe deux moyens de consommer les données des** ConfigMaps
  - Injection de variables d'environnements
  - Montage par volume

CONSOMMATION DES CONFIGMAPS

INJECTION DE VARIABLES D'ENVIRONNEMENTS

- **Étant donné le** ConfigMap **nommé** db-conf **suivant**
- data:
  DB_HOST: db-0 DB_PORT: 3306

- **Le manifeste ci-contre permettra aux pods créés d'accéder aux valeurs du** ConfigMap **depuis les variables d'environnement du même nom**

```
$ env
...
DB_HOST=db-0  DB_PORT=3306
```

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
    - name: configmap-test
      ...
      env:
        - name: DB_HOST
          valueFrom:
            configMapKeyRef:
              name: db-conf
              key: DB_HOST
        - name: DB_PORT
          valueFrom:
            configMapKeyRef:
              name: db-port
              key: DB_PORT
```

CONSOMMATION DES CONFIGMAPS

MONTAGE PAR VOLUME

- **Étant donné le** ConfigMap **nommé** db-conf **suivant**
- data: database.properties: | db_host=db-0 db_port=3306

- **Le manifeste ci-contre permettra aux pods créés d'accéder à un fichier** database.proporties **correspondant à celui du** ConfigMap

```
$ cat /etc/config/database.properties
db_host=db-0 db_port=3306
```

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
    - name: configmap-test
      ...
      volumeMounts:
      - name: db-config-volume
        mountPath: /etc/config
  volumes:
    - name: db-config-volume
      configMap:
        name: db-conf
        items:
        - key: database.properties
          path: database.properties
```

14.

DÉPLOIEMENT ET PARTAGE DES ÉLÉMENTS DE CONFIGURATION

CRÉER ET CONSOMMER DES SECRETS

CRÉER ET CONSOMMER DES SECRETS

PROBLÉMATIQUE

- **Les** Secrets **sont des ressources similaires aux** ConfigMaps
- **Ils sont dédiés aux stockage de données sensibles comme les mots de passe ou les clés d'API**
- **Le valeurs contenues dans un secret sont encodés en base64**
- **Les** Secrets **n'apportent pas de sécurité en soit, mais il est possible de limiter leur accès à certains utilisateurs (cf chapitre 16)**

CRÉER ET CONSOMMER DES SECRETS

FONCTIONNEMENT

- **Les** Secrets **fonctionnent de manière très similaire aux** ConfigMap
- **La seule différence étant qu'il faut spécifier les valeurs sous forme de base64**
- **Le champ type peut prendre 3 valeurs :**
  - Opaque : type par défaut indiquant que le secret contient des valeurs arbitraires
  - ServiceAccount : secret créé pour chaque service account,contenant les informations d'authentification
  - ImagePullSecret : secret dédié à contenir les informations d'un dépôt docker privé

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: monsecret
type: Opaque
data:
  db_user: YWRtaW4=
  password: MWYyZDF1MmU2N2Rm
```

CONSOMMATION DES SECRETS

INJECTION DE VARIABLES D'ENVIRONNEMENTS

- **Les secrets peuvent être injectés dans les pods comme les** ConfigMap
- **Étant donné le** Secret **nommé** db-secret **suivant**
- data:
  db-user: YWRtaW4=
  db_password: MWYyZDFlMmU2N2Rm

- **Le manifeste ci-contre permettra aux pods créés d'accéder aux valeurs du** ConfigMap **depuis les variables d'environnement du même nom**

```
$ env
...
DB_USER=admin  DB_PASSWORD=YWRtaMWYyZDFlMmU2N2Rmse64
```

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
    - name: secret-test
      ...
      env:
        - name: DB_USER
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: db_user
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: db_password
```

CONSOMMATION DES SECRETS

INJECTION DE VARIABLES D'ENVIRONNEMENTS

- **Étant donné le** Secret **nommé** db-secret **suivant**
- data:
  db-user: YWRtaW4=
  db_password: MWYyZDFlMmU2N2Rm

- **Le manifeste ci-contre permettra de monter les valeurs du secret dans des fichiers du même nom**

```
$ cat /etc/config/db-user
admin
$ cat /etc/config/db-password  YWRtaMWYyZDFlMmU2N2Rmse64
```

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
    - name: secret-test
      ...
    volumeMounts:
    - name: db-credentials
      mountPath: "/etc/config"
  volumes:
  - name: db-credentials
    secret:
      secretName: db-secret
      defaultMode: 700
```

15.

GÉRER LES DONNÉES PERSISTANTES ET LES APPLICATIONS STATEFUL

+ PROBLÉMATIQUE DU STOCKAGE PERSISTANT DANS LE CLOUD
+ INTRODUCTION AUX VOLUMES ET LEURS DIFFÉRENTS TYPES
+ CRÉER UNE APPLICATION EN EXPLOITANT UN VOLUME EMPTYDIR
+ SOLUTIONS DE STOCKAGE DISTANT
+ INTRODUCTION AUX STATEFULSETS

15.

GÉRER LES DONNÉES PERSISTANTES ET LES APPLICATIONS STATEFUL

STOCKAGE PERSISTANT DANS LE CLOUD

STOCKAGE PERSISTANT DANS LE CLOUD

PROBLÉMATIQUE

- **Dans un environnement Cloud, le stockage persistant est une problématique complexe**
- **Les conteneurs sont de nature volatile, ils peuvent être créés, détruits, déplacés dans des intervalles de temps court**
- **Stocker des fichiers sur les nœuds n'est pas envisageable, un conteneur pouvant apparaitre, disparaitre et réapparaitre sur un autre nœud où ses fichiers ne sont pas présents**

INDÉPENDANCE APPLICATION / PLATEFORME

RETOUR SUR LES 12 FACTORS

1. Base de code
2. Dépendances
3. Configuration
4. Services Externes
5. Build, Release, Run
6. Processus

7 Associations de ports

8 Concurrence

9 Jetable

10 Parité dev/prod

11 Logs

12 Processus d'administration

INDÉPENDANCE APPLICATION / PLATEFORME

RAPPELS CLOUD : LES 12 FACTEURS

6 - Processus

- **« Exécutez l'application comme un ou plusieurs processus sans état »**
- **« Les processus 12 facteurs sont sans état et ne partagent rien. Toute donnée qui doit être persisté doit être stockée dans un service externe stateful, typiquement une base de données. »**

STOCKAGE PERSISTANT DANS LE CLOUD

PROBLÉMATIQUE

- **En réalité, la plupart des applications ont besoin de stocker des fichiers et ne peuvent pas se contenter d'une base de donnée externe**

- **Kubernetes propose de nombreuses solutions pour répondre à cette problématique**

15.

GÉRER LES DONNÉES PERSISTANTES ET LES APPLICATIONS STATEFUL

INTRODUCTION AUX VOLUMES ET LEURS DIFFÉRENTS TYPES

LES VOLUMES ET LEURS DIFFÉRENTS TYPES

PRINCIPE

- **Kubernetes offre la possibilité de créer des volumes et de les monter à l'intérieur des Pods (par exemple pour monter des Secrets comme vu dans le chapitre 14)**

- **Docker lui-même offre la possibilité de créer et de monter des volumes (cf chapitre 02)**

- **Kubernetes étend cette possibilité et propose des interfaces pour s'interconnecter avec des systèmes de stockage persistant externes**

LES VOLUMES ET LEURS DIFFÉRENTS TYPES

TYPES DE VOLUMES

Kubernetes propose un certain nombre de volumes « locaux », c'est-à-dire propre au cluster ou hébergés localement sur les hôtes

Les volumes de type secret **et** configMap **ont été abordés au chapitre précédent**

VOLUMES LOCAUX

+ secrets
+ configMap
+ emptyDir
+ hostPath

LES VOLUMES ET LEURS DIFFÉRENTS TYPES

VOLUMES EMPTYDIR

- Les volumes emptyDir sont créés et détruits avec les Pod auxquels ils sont attachés
- Dans ce sens, ils n'offrent pas une solution de stockage strictement persistant
- Ils sont cependant pratiques pour certaines opérations, ou pour partager un espace de travail entre les conteneurs d'un même pod
- À noter, un conteneur peut être détruit et recréé au sein d'un Pod sans affecter le volume emptyDir. Ce dernier n'est supprimé qu'à la suppression du **Pod**

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

LES VOLUMES ET LEURS DIFFÉRENTS TYPES

VOLUMES EMPTYDIR

- **Il est également possible de créer un volume emptyDir en mémoire, à la manière d'un volume Docker tmpFs**

- **emptyDir: medium: Memory**

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir:
      medium: Memory
```

LES VOLUMES ET LEURS DIFFÉRENTS TYPES

VOLUMES HOSTPATH

- **Un volume hostPath permet de monter n'importe quel chemin de l'hôte à l'intérieur du conteneur**
- **Le champ type peut prendre différentes valeurs selon la cible du montage côté hôte :**
  - DirectoryOrCreate : créer un répertoire si il n'existe pas
  - Directory : monter un répertoire existant
  - File / FileOrCreate : monter un fichier
  - Socker : monter une socket unix
  - CharDevice / BlockDevice : monter un char/block device
- **Les volumes hostPath présentent l'inconvénient majeur de rendre le comportement de Pod non déterministe**

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      path: /data
      type: Directory
```

15.

GÉRER LES DONNÉES PERSISTANTES ET LES APPLICATIONS STATEFUL

SOLUTIONS DE STOCKAGE DISTANT

SOLUTIONS DE STOCKAGE DISTANT

TYPES DE VOLUMES

Kubernetes propose une grande quantité d'interfaces de connexion vers des solutions de stockage distants

VOLUMES DISTANTS

+ nfs
+ rbd
+ awsElasticBlockStore
+ azureDisk
+ cephfs

SOLUTIONS DE STOCKAGE DISTANT

PERSISTENT VOLUMES ET PERSISTENT VOLUME CLAIMS

- **Les types de volume vus précédemment ne nécessitent aucune interactions avec des services extérieurs**
- **Pour les systèmes de stockage externes, il est nécessaire de provisionner les espaces de stockage avant de les utiliser**
- **Kubernetes fournit les mécanismes pour gérer le cycle de vie de ces espaces de stockage**

SOLUTIONS DE STOCKAGE DISTANT

PERSISTENT VOLUMES ET PERSISTENT VOLUME CLAIMS

- **La gestion du provisionnement des volumes se fait grâce à deux ressources**
- **PersistentVolume** : représente un **espace de stockage** provisionné et prêt à être utilisé
- **PersistentVolumeClaim** : représente une **requête** d'espace de stockage

SOLUTIONS DE STOCKAGE DISTANT

PERSISTENT VOLUMES ET PERSISTENT VOLUME CLAIMS

- **Un PersistentVolume spécifie entre autre la configuration d'accès au système de stockage distant**

- **Dans l'exemple ci-contre, on spécifie :**
  - Un volume de 1 go sur un cluster Ceph distant
  - L'adresse du moniteur Ceph
  - Le nom du pool sur le cluster
  - Le nom du secret contenant les paramètres d'authentification sur le cluster Ceph
  - Un identifiant de classe de stockage : ceph

```yaml
apiVersion: v1
kind: PersistentVolume
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 1Gi
  persistentVolumeReclaimPolicy: Delete
  rbd:
    image: kubernetes-dynamic-pvc-0a37541a
    keyring: /etc/ceph/keyring
    monitors:
    - 192.168.1.201:6789
    pool: staging
    secretRef:
      name: ceph-user-secret
    user: staging
  storageClassName: ceph
```

SOLUTIONS DE STOCKAGE DISTANT

PERSISTENT VOLUMES ET PERSISTENT VOLUME CLAIMS

- **Un** PersistentVolumeClaim **permet à un utilisateur d'émettre une requête pour un espace de stockage, le** PVC **est monté comme un volume**

Un PV **est attaché à un** PVC **dans les conditions suivantes :**

- pvc.capacity.storage >= pv.resources.requests.storage ET
- pv.storageClassName = "" || pvc.storageClassName = pvc.storageClassName

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  finalizers:
  - kubernetes.io/pvc-protection
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ceph
```

SOLUTIONS DE STOCKAGE DISTANT

PERSISTENT VOLUMES ET PERSISTENT VOLUME CLAIMS

- **Pour résumer :**
  - **Un administrateur** peut provisionner des volumes en créant des **PersistentVolume** ainsi que les éléments de stockage correspondant dans le système externe
  - **Un utilisateur** peut requêter des volumes en créant des **PersistentVolumeClaim**

- **Ce système présente encore quelques défauts :**
  - L'administrateur doit gérer le provisionnement des volumes
  - Si le cluster est à court de PersistentVolume, l'utilisateur devra attendre l'intervention de l'administrateur

SOLUTIONS DE STOCKAGE DISTANT

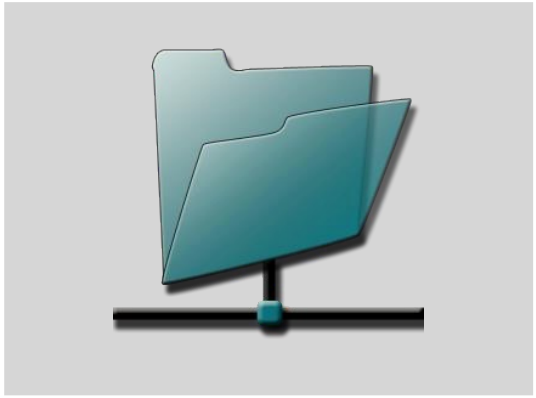
STORAGE CLASS ET DYNAMIC PROVISIONING

- **Kubernetes offre une solution d'automatisation du provisionnement des volumes à travers la ressource** StorageClass
- **La ressource** StorageClass **définit un template de** PersistentVolume
- **Elle accepte également un attribut provisioner**
- **Le rôle du provisioner est de créer dynamiquement un** PersistentVolume **selon le template, ainsi qu'un volume correspondant dans le système externe**

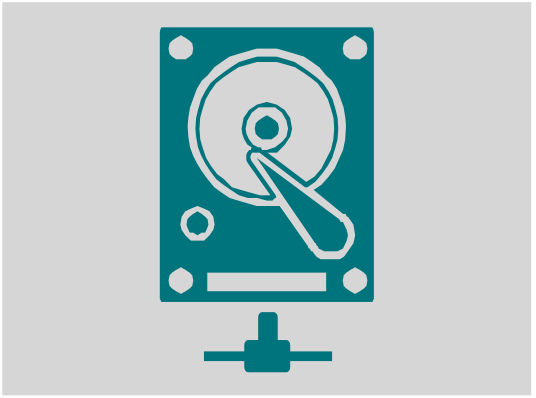
SOLUTIONS DE STOCKAGE DISTANT

RAPPEL SUR LES DIFFÉRENTS TYPES DE SERVICE DE STOCKAGE

STOCKAGE FICHIER



STOCKAGE BLOCK



STOCKAGE OBJET

SOLUTIONS DE STOCKAGE DISTANT

EXEMPLE AVEC AMAZON EBS

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  zones: us-east-1d, us-east-1c
  iopsPerGB: "10"
  fsType: ext4
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: front-claim
  finalizers:
  - kubernetes.io/pvc-protection
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: slow
```

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: dockerfile/nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: myvolume
  volumes:
    - name: myvolume
      persistentVolumeClaim:
        claimName: myclaim
```

SOLUTIONS DE STOCKAGE DISTANT

EXEMPLE AVEC AMAZON EBS

- **type :**
  - gp2 : general purpose
  - io1 : disques SSD faible latence
  - st1 : disques à haut débit
  - sc1 : disques « froids » pour archivage
- **zone :** emplacement géographique du stockage
- **iopsPerGB :** quantité d'entrées/sorties par seconde par GB à provisionner

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  zones: us-east-1d, us-east-1c
  iopsPerGB: "10"
  fsType: ext4
```

SOLUTIONS DE STOCKAGE DISTANT

EXEMPLE AVEC AZURE DISK STORAGE

- **kind :** permet de créer les disques dans un compte de stockage partagé ou dans un compte de stockage dédié

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/azure-disk
parameters:
  storageaccounttype: Standard_LRS
  kind: Shared
```

SOLUTIONS DE STOCKAGE DISTANT

EXEMPLE AVEC NFS

- Certains plugins de stockage ne disposent pas d'un provisioner correspondant préinstallé, c'est le cas de NFS
- Il est cependant possible de trouver ces provisioners dans le dépôt d'incubation de Kubernetes : https://github.com/kubernetes-incubator/external-storage
- L'installation est simple

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: example-nfs
provisioner: example.com/nfs
parameters:
mountOptions: "vers=4.1"
```

```
$ kubectl create -f deploy/kubernetes/deployment.yaml
service "nfs-provisioner" created  deployment "nfs-provisioner" created
```

15.

GÉRER LES DONNÉES PERSISTANTES ET LES APPLICATIONS STATEFUL

INTRODUCTION AUX STATEFULSETS

INTRODUCTION AUX STATEFULSETS

PROBLÉMATIQUE

- **Comme évoqué précédemment, les pods sont par nature volatiles**
- **Pour autant, certaines applications sont par nature persistantes**
- **C’est le cas des systèmes de stockage distribués par exemple**

INTRODUCTION AUX STATEFULSETS

FONCTIONNEMENT

- **Les** StatefulSets **ont une interface similaire aux** ReplicaSets**, mais possèdent quelques nuances qui les rendent particulièrement adaptés au déploiement d'applications persistantes**

- **Remarquer notamment la section** volumeClaimTemplates

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

INTRODUCTION AUX STATEFULSETS

FONCTIONNEMENT

- **Un StatefulSet offre un certain nombre de garanties :**
  - Les pods sont ordonnés, et démarrés dans l'ordre, les uns après les autres
  - Les pods disposent chacun d'un nom réseau fixe
    ex: **web-0.nginx.default.svc.cluster.local**
  - Chaque pod dispose d'un volume persistant (issu du template)

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

INTRODUCTION AUX STATEFULSETS

FONCTIONNEMENT

- **Ces garanties facilitent le déploiement d'applications persistantes**
- **Exemple avec mysql en mode master-slave :**

- Du point de vue serveur, il est possible de configurer un nœud master sur le Pod d'index 0

```
if [[ $ordinal -eq 0 ]]; then
  cp /mnt/config-map/master.cnf /mnt/conf.d/
else
  cp /mnt/config-map/slave.cnf /mnt/conf.d/
fi
```

- Du point de vue client, il est possible d'adresser directement le Pod master avec un nom DNS invariant

```
env:
  - name: DB_HOST
    value: mysql-0.mysql-ha.svc.cluster.local
```

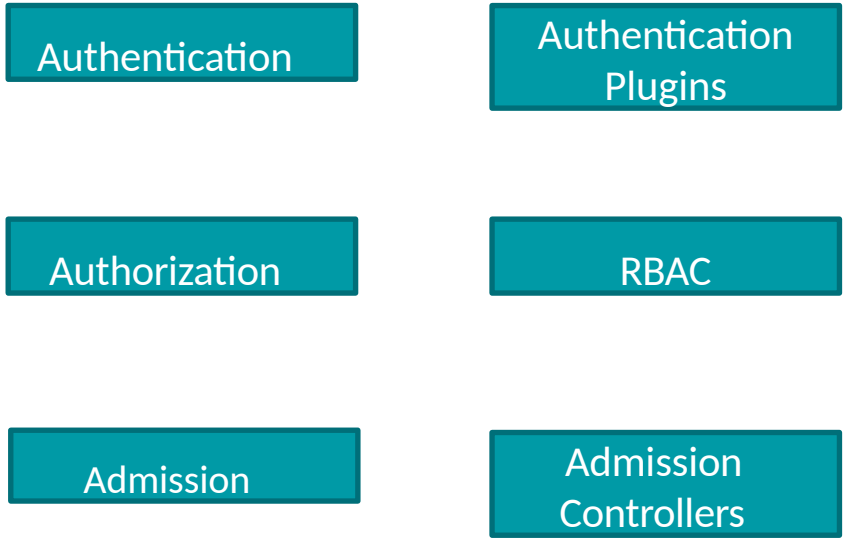
16.

SÉCURITÉ DANS KUBERNETES

+ INTRODUCTION ET MANIPULATION DE L'API RBAC
+ INTRODUCTION AUX NETWORKPOLICIES
+ INTRODUCTION AUX PODSECURITYPOLICIES

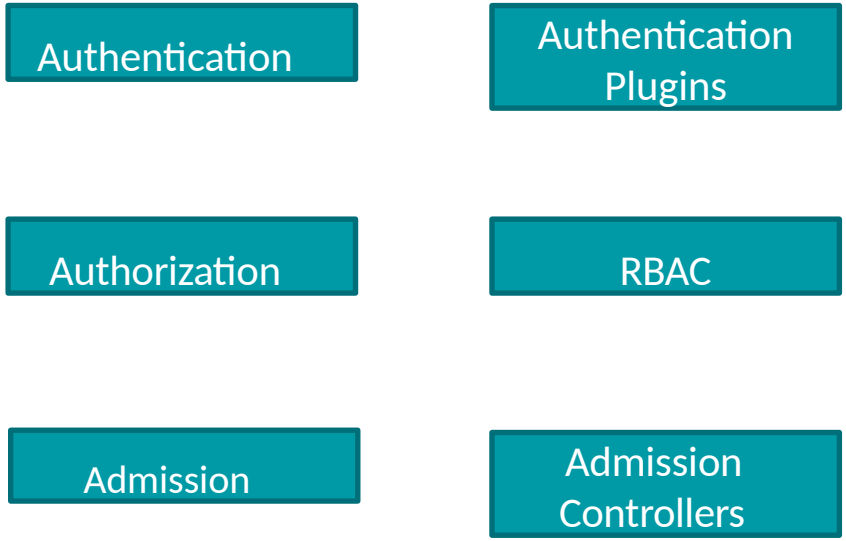
SÉCURITÉ DANS KUBERNETES

- **Kubernetes propose 3 couches de sécurité gérées par des composants différents :**
- « Authentication » : qui suis-je ?
- « Authorization » : à quoi ai-je le droit d'accéder ?
- « Admission » : l'action que je souhaites faire est-elle légale ?



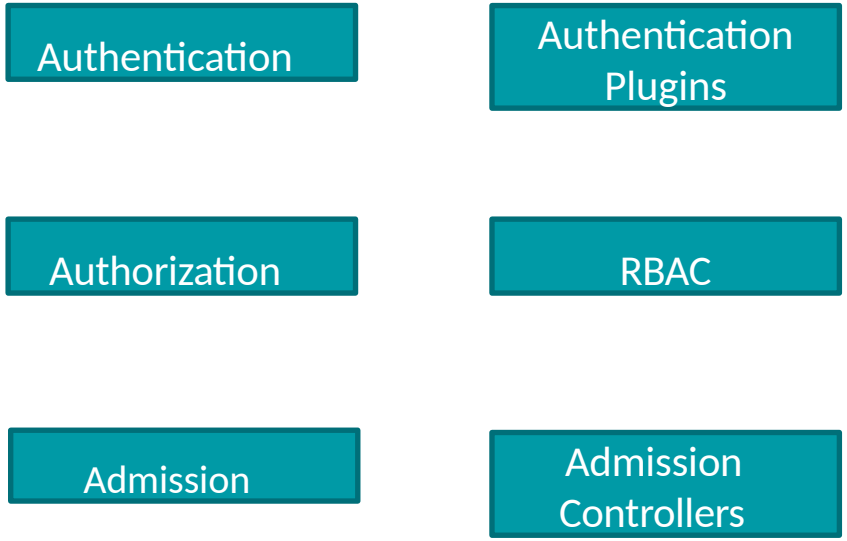
SÉCURITÉ DANS KUBERNETES

- **Authentication**
- Gérée par les authentication plugins
- Par exemple :
  - Certificat client x509
  - Tokens
  - Authentification basic
  - OpenID



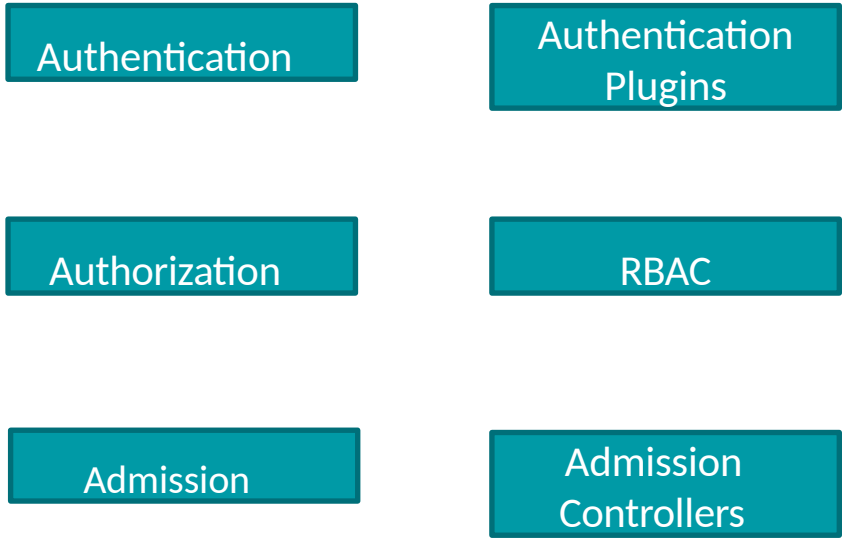
SÉCURITÉ DANS KUBERNETES

- **Authentication**
- Il peut y avoir plusieurs **Authentication Plugins** agissant en même temps
- Chacun analyse les paramètres de la requête
- **Si un seul d'entre eux répond OK, la requête est authentifiée**



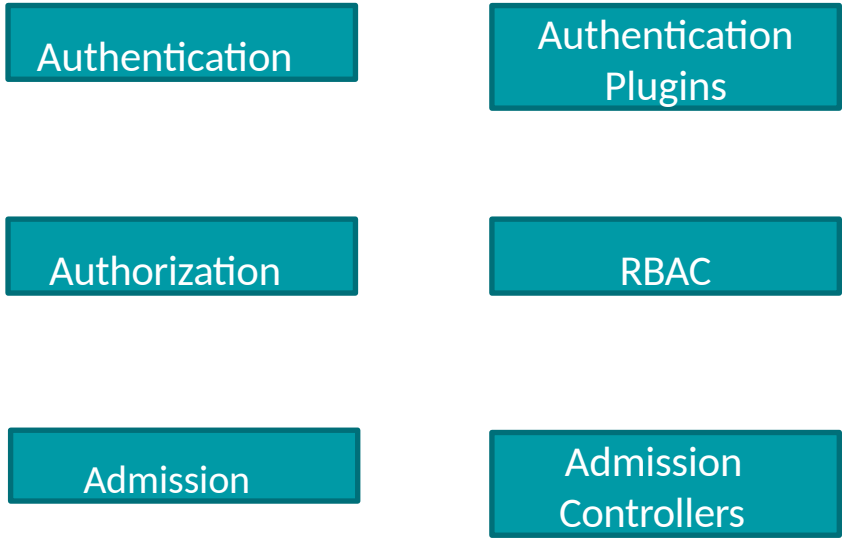
SÉCURITÉ DANS KUBERNETES

- **Authorization**
- Gérée par RBAC que nous allons aborder en détails



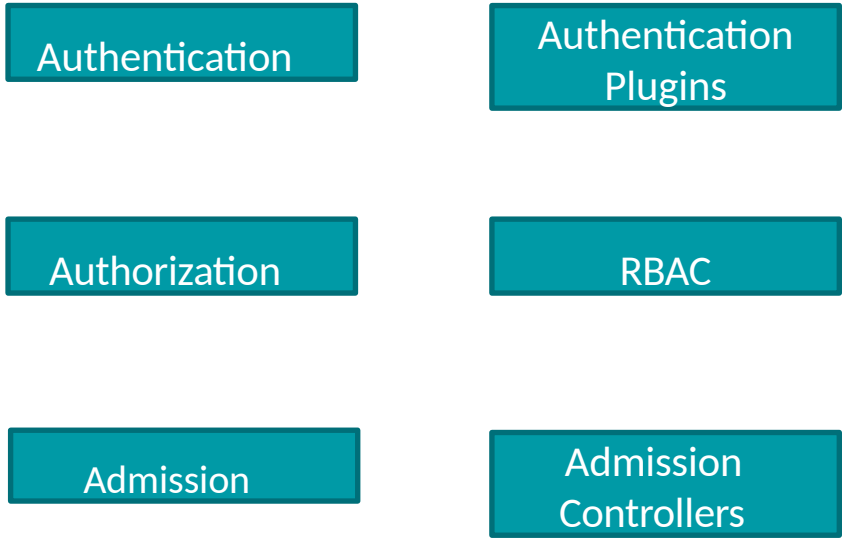
SÉCURITÉ DANS KUBERNETES

- **Admission**
- Gérée par les admission controllers
- Généralement, plusieurs admission controllers sont installés par défaut et agissent en même temps
- **Si un seul répond KO, la requête est rejetée**



SÉCURITÉ DANS KUBERNETES

- **Admission**
- Les Admission Controllers réalisent des tâches variées, certaines n'étant pas forcément en rapport avec le contrôle d'accès
- Exemple :
  - Injecter des valeurs par défaut dans les requêtes (StorageClass par exemple)
  - Empêcher kubectl exec sur un conteneur privilégié
  - Etc.



SÉCURITÉ DANS KUBERNETES

- **Kubernetes gère deux types d'utilisateurs différents**

- Les **Users** sont des utilisateurs gérés par un service externe (basic, token, OpenID etc.). Il n'existe pas d'API permettant de créer des Users dans Kubernetes

- Les **ServiceAccounts** sont des utilisateurs techniques, dédiés à être utilisés par des applications. Ils sont créés et gérer directement dans le cluster.

SÉCURITÉ DANS KUBERNETES

- **Kubernetes gère deux types d'utilisateurs différents**
- Exemple ci contre : ServiceAccount par défaut, dans le namespace par défaut
- Chaque ServiceAccount se voit automatiquement attribué un token dans un Secret

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
  namespace: default
secrets:
- name: default-token-lq689
```

16.

SÉCURITÉ DANS KUBERNETES

INTRODUCTION ET MANIPULATION DE L'API RBAC

INTRODUCTION À L'API RBAC

- **RBAC = Role-Based Access Control**
- **API de gestion des droits dans Kubernetes**
- **Définit un ensemble de ressources permettant de lier ressources, droits d'accès et utilisateurs.**
- **Apparue en beta dans Kubernetes 1.6, stable depuis Kubernetes 1.8**

INTRODUCTION À L'API RBAC

- **L'API RBAC définit 4 ressources :**

- Role
- ClusterRole
- RoleBinding
- ClusterRoleBinding

- **Par défaut les sujets n'ont aucun droits, les Role permettent d'en ajouter (gestion additive du contrôle d'accès)**

INTRODUCTION À L'API RBAC

- **Role**
- Défini en ensemble de droits sur un ensemble de ressources d'API
- Limité à un namespace donné
- Exemple ci-contre :
  Rôle 'pod-reader' = droits de lecture sur les pods

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

INTRODUCTION À L'API RBAC

- **ClusterRole**
- Même principe que les Roles
- Périmètre global (cluster entier)
- Exemple ci-contre :
  Rôle 'secret-reader' = droits de lecture sur les secrets,dans tout le cluster
- Noter l'absence de namespace dans les meta-données

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

INTRODUCTION À L'API RBAC

- **Zoom sur les règles :**

- Chaque API est définie par un nom de ressource et un groupe

- Par exemple
  - « pods » dans le groupe « core »
  - « deployment » dans le groupe « apps »

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

INTRODUCTION À L'API RBAC

- **Zoom sur les règles :**
- Pour spécifier le groupe core on indique une chaine vide
- Certaines ressources existent dans plusieurs groupes, on peut indiquer tous ces groupes comme ci-contre

```
kind: Role
apiVersion: rbac.authorization.k8s.io
metadata:
  namespace: default
  name: deployer
rules:
- apiGroups: ["extensions", "apps"]
  resources: ["deployments"]
  verbs: ["*"]
```

INTRODUCTION À L'API RBAC

- **Zoom sur les règles :**
- On peut utiliser un wildcard à la fois pour les ressources et les verbes
- Exemple ci-contre : créer un administrateur de namespace

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: namespace-admin
rules:
  - apiGroups:
    - ""
    - batch
    - extensions
    - apps
    resources:
    - '*'
    verbs:
    - '*'
```

INTRODUCTION À L'API RBAC

- **RoleBinding**
- Associe un Role à un sujet
- Le rôle indiqué doit être présent dans le namespace de création du RoleBinding
- Le sujet peut être un utilisateur, un ServiceAccount ou un groupe

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

INTRODUCTION À L'API RBAC

- **RoleBinding**
- Un RoleBinding peut également associer un ClusterRole à un sujet
- Dans ce cas les ressources ciblées par le ClusterRole sont accessibles dans le namespace de création du RoleBinding

INTRODUCTION À L'API RBAC

- **ClusterRoleBinding**
- Associe un ClusterRole à un sujet
- Permet de définir des droits globaux
- Exemple ci-contre : tous les utilisateurs du groupe manager ont le droit de lire les Secrets de tous les namespaces

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

INTRODUCTION À L'API RBAC

- **Aggrégation de rôles**
- On peut agréger plusieurs Roles en un seul
- C'est ce qui permet entre autre d'étendre les Roles par défaut
- Exemple ci-contre : Role admin par défaut qui agrège tous les Roles portant le label *rbac.authorization.k8s.io/aggregate-to-admin="true"*

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
aggregationRule:
  clusterRoleSelectors:
  - matchLabels:
      rbac.authorization.k8s.io/aggregate-to-admin: "true"
rules:
  ...
```

16.

SÉCURITÉ DANS KUBERNETES

INTRODUCTION AUX NETWORKPOLICIES

INTRODUCTION AUX NETWORK POLICIES

- **En plus de la gestion des droits d'accès, Kubernetes offre une API de contrôle d'accès réseau**
- **Les objects NetworkPolicy permettent de définir quel trafic réseau entrant ou sortant est autorisé**
- **La CNI sous-jacente est responsable de l'implémentation des règles définies (Calico est l'implémentation de référence)**

INTRODUCTION AUX NETWORK POLICIES

- **Par défaut, tout le trafic est autorisé vers et venant des Pods**

- À partir du moment où une NetworkPolicy existe, seul le traffic qu'elle autorise sera passant
- Une NetworkPolicy qui ne définit aucune règle va donc bloquer tout le trafic (cf exemple ci-contre)

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-deny-all
spec:
  podSelector:
    matchLabels:
      app: web
  ingress: []
  egress: []
```

INTRODUCTION AUX NETWORK POLICIES

- **Structure du manifeste**
- Une NetworkPolicy est restreinte à un namespace
- On sélectionne les pods auxquelle elle s'applique avec un selecteur de labels
- Les catégories **ingress** et **egress** permettent de définir les règles pour le trafic entrant et sortant respectivement

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-deny-all
spec:
  podSelector:
    matchLabels:
      app: web
  ingress: []
  egress: []
```

INTRODUCTION AUX NETWORK POLICIES

- **Structure du manifeste**
- Les attributs possibles dans ingress et egress ont la même signification :
  - ports : liste de ports autorisés
  - from/to : cibles autorisées
- Quelle règle implémente l'exemple ci-contre ?

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow-5000
spec:
  podSelector:
    matchLabels:
      app: apiserver
  ingress:
  - ports:
    - port: 5000
    from:
    - podSelector:
        matchLabels:
          role: monitoring
```

INTRODUCTION AUX NETWORK POLICIES

- **Structure du manifeste**
- La cible peut être déterminée de 3 manières différentes :
  - **ipBlock** : CIDR autorisé à accéder aux pods
  - **namespaceSelector** : sélecteur de label permettant de cibler un ou plusieurs namespaces
  - **podSelector** : sélecteur de label permettant de sélectionner un ou plusieurs pods

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow-5000
spec:
  podSelector:
    matchLabels:
      app: apiserver
  ingress:
  - ports:
    - port: 5000
    from:
    - podSelector:
        matchLabels:
          role: monitoring
```

INTRODUCTION AUX NETWORK POLICIES

- **Structure du manifeste**
- La cible peut être déterminée de 3 manières différentes
- On peut combiner plusieurs cibles en même temps

16.

SÉCURITÉ DANS KUBERNETES

INTRODUCTION AUX PODSECURITYPOLICIES

POD SECURITYPOLICIES

- **Nous avons en main les mécanismes pour limiter :**
  - L'accès à l'API avec RBAC
  - La communication réseau avec les NetworkPolicies
- **Il reste une composante importante à sécuriser : Docker en lui même**

POD SECURITYPOLICIES

- **Docker n’étant pas un système de virtualisation, certaines zones de perméabilité avec l’hôte et les autres conteneurs peuvent apparaitre**

- **Ne pas oublier :**
  - Chaque conteneur partage le kernel de l’hôte
  - Les utilisateurs et les groupes sont aussi partagés
  - Un conteneur peut monter des volumes sur l’hôte
  - Un conteneur peut choisir arbitrairement son uid/gid

POD SECURITYPOLICIES

- **Dans ces conditions, le scénario suivant est possible :**
  - Démarrer un conteneur avec l'uid 0
  - Monter /root/.ssh/ en volume
  - Insérer une clé publique dans /root/.ssh/authorized_keys
- **Ceci pourrait permettre de créer une backdoor sur l'hôte.**

POD SECURITYPOLICIES

- **Il est possible de sécuriser l'accès extérieur au cluster de manière robuste avec plusieurs couches de sécurité (firewall, netfilter, anti-ddos etc.)**

- **Cependant ces solutions ne traitement pas les problèmes les plus souvent sous-estimés en termes de sécurité :**
- La vulnérabilité humaine (social engineering)
- L'attaquant interne (employé malicieux)

POD SECURITYPOLICIES

- **Pour mitiger ces risques, Kubernetes nous permet de définir des objets de type PodSecurityPolicy**
- **Un objet PodSecurityPolicy définit un ensemble de règles relatives au contexte de sécurité que les Pods peuvent utiliser**

POD SECURITYPOLICIES

- **Pour mitiger ces risques, Kubernetes nous permet de définir des objets de type PodSecurityPolicy**

- **Les PodSecurityPolicy sont des objets globaux qui s'appliquent à tous les Pods créés**
- **Ils définissent un ensemble de règles relatives au contexte de sécurité que les Pods peuvent utiliser**

POD SECURITYPOLICIES

- **Chaque conteneur d'un Pod peut « demander » certains droits privilégiés lors de sa création**

- **Exemple :**
  - Utilisateur ou groupe spécifique à utiliser
  - Démarrage en mode privilégié
  - Capabilities unix à ajouter

```yaml
apiVersion: v1
kind: Pod
metadata:
  ...
spec:
  containers:
    ...
    securityContext:
      runAsGroup: 1000
      runAsUser: 1000
      allowPrivilegeEscalation: true
      capabilities: ['CAP_NET_ADMIN', 'CAP_SHOWN']
```

POD SECURITYPOLICIES

- **Les PodSecurityPolicies permettent de définir des limites sur les valeurs que peuvent prendrent ces attributes**

- **Exemple ci-contre :**
  - Interdire les conteneurs privilégiés
  - Interdire l'exécution en tant que root
  - Autoriser tous les groupes
  - Interdire tous les montages de volumes sauf nfs
  - Interdire tous les hostPort sauf le numéro 100

```yaml
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: example
spec:
  privileged: false
  runAsUser:
    rule: MustRunAsNonRoot
  fsGroup:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  volumes:
  - 'nfs'
  hostPorts:
  - min: 100
    max: 100
```

POD SECURITYPOLICIES

- **Si un Pod ne spécifie pas certaines valeurs, la PodSecurityPolicy modifie le manifeste en ajoutant les bonnes valeurs par défaut**

- **Lorsque plusieurs PodSecurityPolicies peuvent s'appliquer à un Pod, l'ordre des priorités est le suivant :**
  - La première valide qui ne nécessite aucun changements sur le Pod
  - Sinon, la première valide dans l'ordre alphabétique

```yaml
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: example
spec:
  privileged: false
  runAsUser:
    rule: MustRunAsNonRoot
  fsGroup:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  volumes:
  - 'nfs'
  hostPorts:
  - min: 100
    max: 100
```

17.

KUBERNETES ET SON ÉCOSYSTÈME

+ DÉPLOYER DES SOLUTIONS AVEC HELM
+ ISTIO, CERT-MANAGET, EXTERNAL-DNS, ROOK, ETC...

17.

KUBERNETES ET SON ÉCOSYSTÈME

DÉPLOYER DES SOLUTIONS AVEC HELM

DÉPLOYER DES SOLUTIONS AVEC HELM

- **Kubernetes offre beaucoup de flexibilité avec tous les types de ressources qu'il propose**
- **La gestion des nombreux manifestes que composent une application peut être laborieuse**
- **La tracabilité de l'architecture peut aussi devenir un problème, ainsi que la gestion des environnements de déploiement**

DÉPLOYER DES SOLUTIONS AVEC HELM

- **Helm est un gestionnaire de paquets pour Kubernetes**

DÉPLOYER DES SOLUTIONS AVEC HELM

- **Un peu de vocabulaire :**

- L'outil en ligne de commande permettant de manipuler Helm s'appelle **helm**
- helm s'appuie sur un agent côté serveur nommé **tiller**
- Un « paquet » helm s'appelle un **chart**
- Un chart est défini par un ensemble de **templates** que l'on déploie en leur passant des **valeurs**
    - Un chart déployé s'appelle une **release**

DÉPLOYER DES SOLUTIONS AVEC HELM

- **Pour commencer à déployer des charts Helm, il faut d'abord installer tiller**

```
$ helm init
```

- **Pour créer un chart vide :**

```
$ helm create mon-application
```

DÉPLOYER DES SOLUTIONS AVEC HELM

- **La commande create créé tous les fichiers du chart, avec quelques templates de base**

- **La structure est la suivante :**

- **Charts.yaml** : fichier de méta-données
- **charts/** : contient les dépendances
- **templates/** : templates de manifestes
- **values.yaml** : valeurs par défaut

```
./Chart.yaml
./charts
./.helmignore
./templates
./templates/deployment.yaml
./templates/NOTES.txt
./templates/ingress.yaml
./templates/service.yaml
./templates/_helpers.tpl
./values.yaml
```

DÉPLOYER DES SOLUTIONS AVEC HELM

- **Les templates sont des manifestes contenant des placeholders**
- **Helm fournit un ensemble de clés correspondant aux méta-données du chart**
- .Release : méta-données de la release
- .Chart : méta-données du chart
- .Template : méta-données du template en cours

```
apiVersion: v1
kind: Service
metadata:
  name: {{ template "mon-application.fullname" . }}
  labels:
    app: {{ template "mon-application.name" . }}
    chart: {{ template "mon-application.chart" . }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app: {{ template "mon-application.name" . }}
    release: {{ .Release.Name }}
```

DÉPLOYER DES SOLUTIONS AVEC HELM

- **La clé .Values contient les valeurs passées lors du déploiement**
- Le fichier values.yaml (cf. ci-contre) contient les valeurs par défaut
- Les valeurs peuvent passées ou surchargées lors du déploiement

```
$ helm install rook --namespace rook-ceph rook-stable/rook-ceph –-values values.yaml
```

DÉPLOYER DES SOLUTIONS AVEC HELM

- **Lors du déploiement on spécifie un nom de release et le répertoire contenant le chart**

```
$ helm install rook --namespace rook-ceph rook-stable/rook-ceph
```

- **On peut ensuite lister les déploiements et les mettre à jour**

```
$ helm list --namespace rook-ceph
$ helm upgrade --namespace rook-ceph rook
```

DÉPLOYER DES SOLUTIONS AVEC HELM

- **Un grand nombre de charts existants permettent d'installer des applications très rapidement**

```
$ helm repo update
$ helm install stable/mysql
```

- **Comme tout bon gestionnaire de paquets, Helm propose un dépôt officiel**
- **Il est possible de créer ses propres dépôts**

19.
RESSOURCES ET RÉFÉRENCES

RESSOURCES ET RÉFÉRENCES

Documentation générale, fonctionnement et attributs principaux des ressources :

https://kubernetes.io/docs/concepts/

Référence d'API :

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.12/

Énoncés des travaux pratiques :

http://formation-kubernetes.public.datailor.fr/

Kubernetes: Up and Running, Kelsey Hightower (ISBN 978-1491935675)

Mastering Kubernetes, Gigi Sayfan (ISBN 978-1786461001)