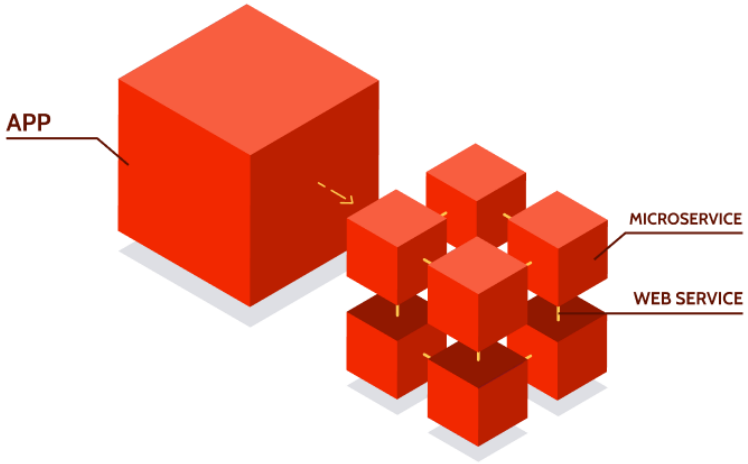


Les Micro services

Techniques de déploiement des micro services avec Kubernetes

Béchir BEJAOUI
Formateur et consultant indépendant

Architecture orientée service:

Le plan

- Problématique
- Introduction de Kubernetes
- Architecture de Kubernetes
- Méthodes d’installation
- Installation de minikube sous Windows
- Les objets K8s

Problématique

- Imaginons la situation de gérer tout un parc où se présente une dizaine voir centaine de conteneurs qui pour une bonne partie de ces conteneurs, il y a des dépendances, une autre partie nécessitent des paramètres classés confidentiels comme des tokens, des mots de passe

- L'idée est comment gérer tout ce là et surtout comment cacher la complexité à l'utilisateur final

- Comment gérer la complexité des dépendances entre les différents conteneurs

- Est-ce que docker compose tout seul fera l'affaire

Introduction de Kubernetes

- Kubernetes, également connu sous le nom de K8s, est un système open source permettant de gérer des applications conteneurisées sur plusieurs hôtes. Il fournit des mécanismes de base pour le déploiement, la maintenance et la mise à l'échelle des applications

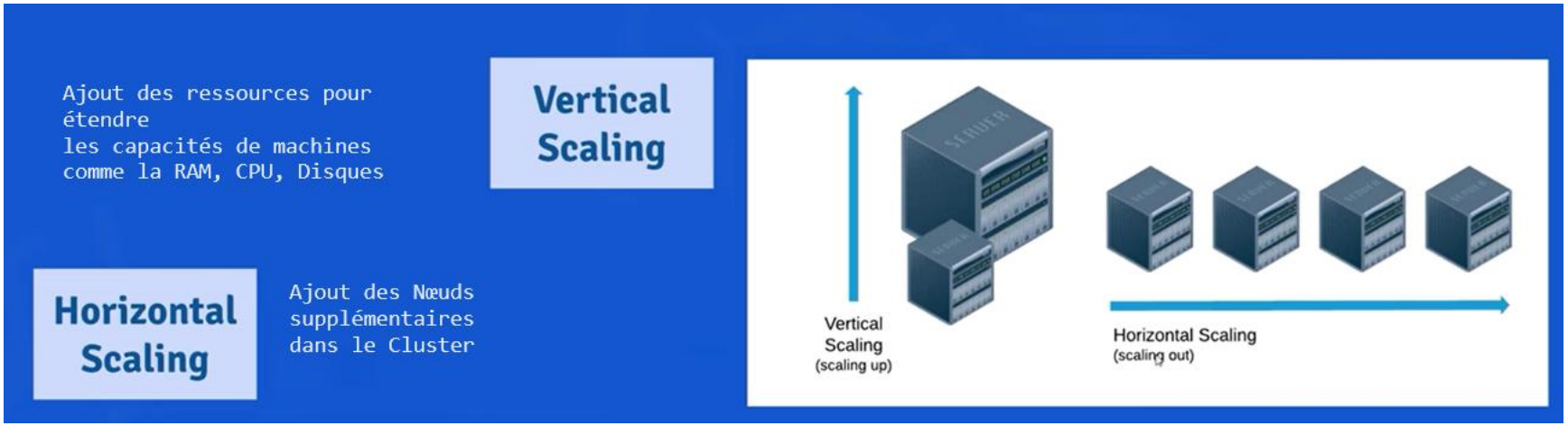
- Kubernetes s'appuie sur une décennie et demie d'expérience chez Google dans l'exécution de charges de travail de production à grande échelle à l'aide d'un système appelé Borg

- En suite, cette solution est devenue open source
  https://github.com/kubernetes/kubernetes

- Il y a toute une documentation officielle qui permet de préparer à la certification
  https://kubernetes.io/docs/home/

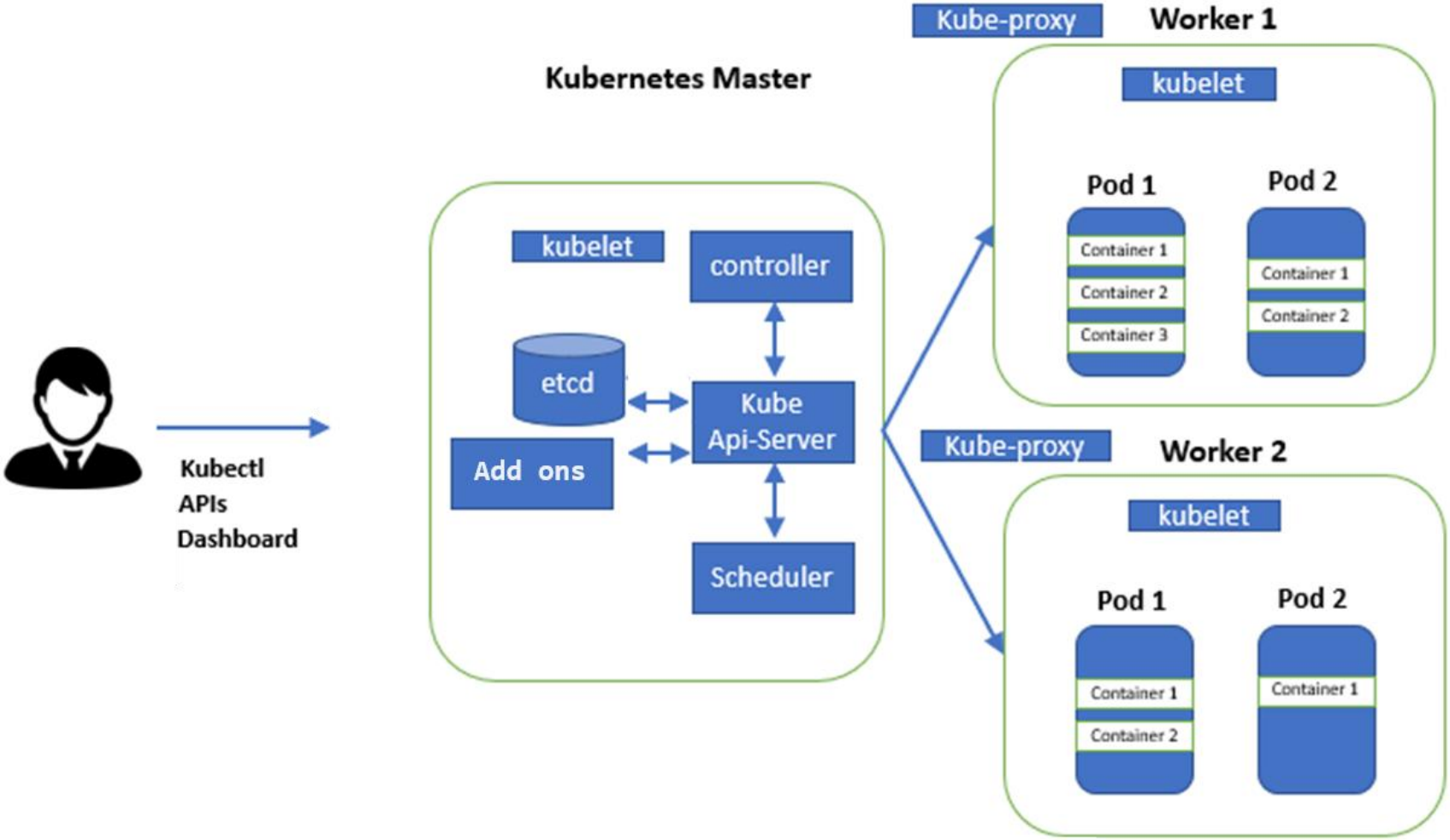
Avantages de Kubernetes

- **Evolutivité:**



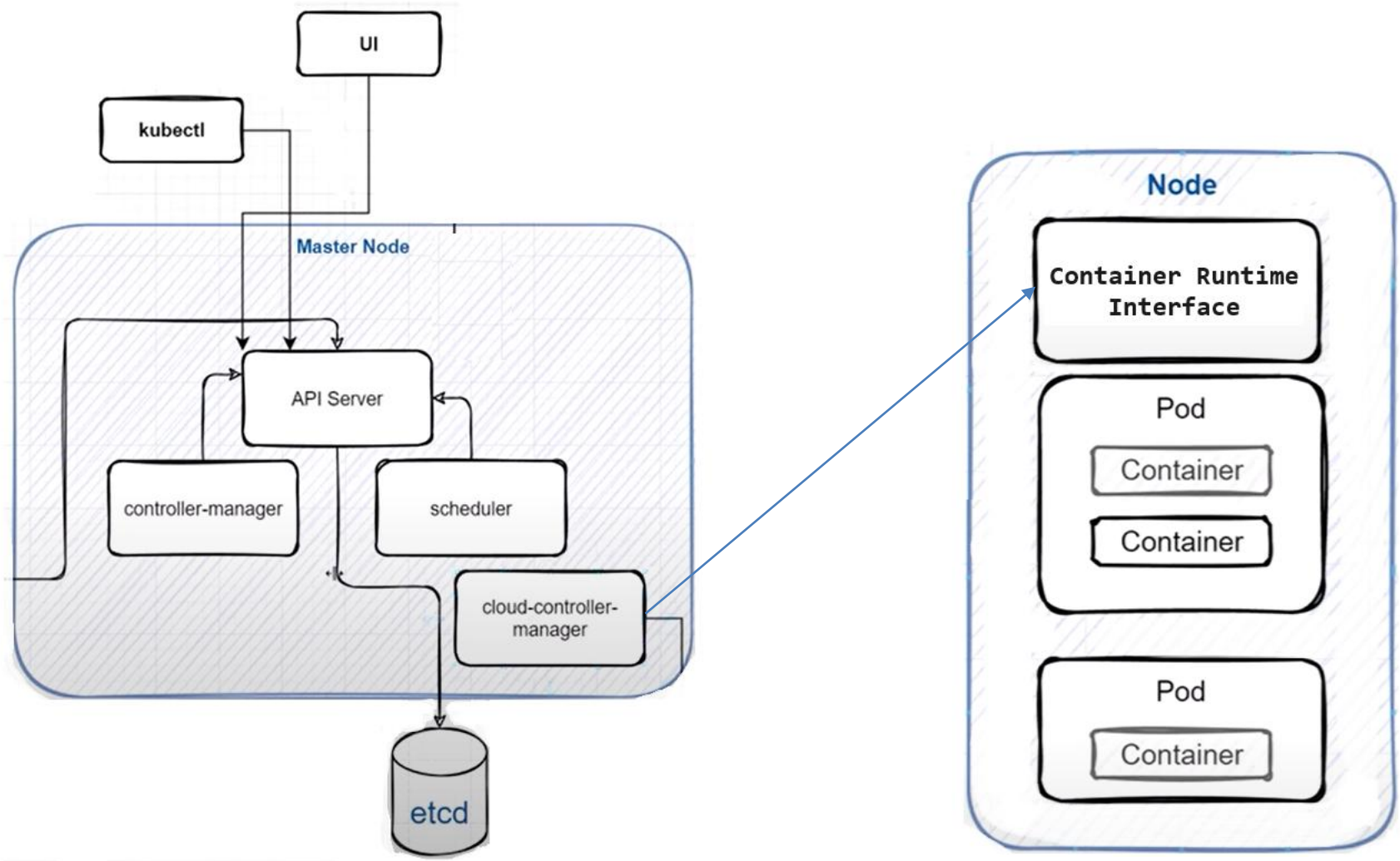
- **Autonomie**
- **Gestion des secrets**
- **La portabilité**

Architecture de Kubernetes



Architecture orientée service:

Les avantages de Kubernetes



Architecture orientée service:

Les méthodes d'installation

- Avant de commencer, pour des raisons de test et d'apprentissage, il va falloir consacrer un minimum de 2 G de RAM pour une installation minimale de Kubernetes

- Les possibilités d'installation
  - **Installation de Minicube:** C'est une installation minimale pour des raisons d'apprentissage
  - **Installation Micro k3s:** C'est une installation minimale
  - **Installation Micro k8s:** C'est une installation minimale avec possibilité d'évolution
  - **Installation de KubeAdm:** C'est une installation à l'echelle de production avec la possibilité de déploiement d'un cluster K8s avec un nœud master et des nœuds workers
  - **Solutions cloud:** Des fournisseurs cloud comme Azure, AWS, Digital Ocean, Google, Linode ou Open Stack offrent chacun sa version de Kubernetes

Installation de Mini cube

Installation Minicube (Linux)

Note: Il est nécessaire d'installer Docker avant

Exécuter ces deux commandes:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube
sudo usermod -aG docker $USER && newgrp docker
sudo minikube start  --driver=docker
```

Pour vérifier l'installation de Minicube:

```
minikube version
```

Il faut installer Kubectl en suite:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s
https://storage.googleapis.com/kubernetes-
release/release/stable.txt`/bin/linux/amd64/kubectl
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
```

Installation Minicube (Windows)

Il faut lancer cette commande

chocolatey install minikube

Ensuite lancer minikube avec le driver virtual box

minikube start --driver=virtualbox --no-vtx-check

Initiation Minicube

Lancer le cluster

minikube start --driver=docker

minikube status

Premières interactions avec Minicube

kubectl cluster-info

kubectl get all

Test du cluster (Un premier contact)

kubectl create deployment nginx-web --image=nginx

kubectl expose deployment nginx-web --type NodePort --port=8080

kubectl get all

docker ps

Curl @IP

kubectl scale deployment nginx-web --replicas=3

Minikube dashboard

kubectl delete deployment --all

Les Objets K8s

Les objets K8s

Pod: C'est le plus petit élément, il est composé d'au moins un conteneur avec un micro réseau interne qui lie les conteneurs

- Chacun des conteneurs à son adresse IP privée
- Un Pod ne communique pas avec le monde externe par défaut
- Un Pod est éphémère

Pour créer un Pod avec une commande (Imperative):

kubectl run nginx-web --image=nginx --restart=Always

Pour créer un Pod avec le fichier manifeste(Declarative):

kubectl **create|apply** -f https://k8s.io/examples/pods/simple-pod.yaml

Pour chercher des informations sur le Pod:

- kubectl get pods
- kubectl get pods –o wide
- kubectl describe pod <nom du Pod> | -o json

Pour exécuter le Pod:

kubectl exec -ti <nom du pod> -- /bin/bash

Pour afficher le journal du Pod:

kubectl logs <nom du pod>

Pour supprimer le Pod:

- kubectl delete pod <nom du Pod>

Les objets K8s

Deployment: C'est un élément qui représente un réplica de Pods qui représentent des applications sans état comme les applications front ends et les serveur web

Pour créer un Deployment avec une commande (Imperative):

kubectl create deployment nginx-web --image=nginx

Pour créer un Deployment avec le fichier manifeste(Declarative):

kubectl **create|apply** -f
https://k8s.io/examples/controllers/nginx-deployment.yaml

Note: La création du Deployment va générer un ReplicaSet et un Pod

Pour chercher des informations sur les Deployments:

kubectl get deployements | deploy
kubectl describe deployment | deploy <nom du deploiment>

Pour changer l'échelle des Deployments:

kubectl scale deploy|deployment nginx-dep –
replicas=3

Pour executer et tester le contenu d'un Pod:

kubectl get pods –o wide
Minikube ssh
curl <@IP du Pod>

Les objets K8s

Replicaset: Un ReplicaSet garantit qu'un nombre spécifié de répliques de Pods s'exécutent à un moment donné

Pour créer un Deployment avec le fichier manifeste:

kubectl apply –f https://kubernetes.io/examples/controllers/frontend.yaml

Deployments vs ReplicatSets

Déploiements (DeploymentSet)	Jeu de réplicas (ReplicaSet)
Abstractions haut niveau qui gèrent les jeux de réplicas. Il fournit des fonctionnalités supplémentaires telles que des mises à jour progressives, la restauration et la gestion des versions de l'application.	**Une abstraction bas niveau qui gère le nombre souhaité de répliques d'un pod**. Il fournit des mécanismes de mise à l'échelle et d'auto réparation de base.
Le Déploiement **gère un modèle de pods** et utilise des jeux de réplicas pour garantir que le nombre spécifié de réplicas du pod est en cours d'exécution.	*ReplicaSet* uniquement **gère le nombre souhaité de répliques d'un pod**.
***Le Déploiement* fournit un mécanisme de déploiement des mises à jour et des restaurations de l'application**, permettant des mises à jour transparentes et réduisant les temps d'arrêt.	**Les applications doivent être mises à jour au départ**
Il fournit un versioning de l'application, permettant de gérer plusieurs versions de la même application. Cela permet également de revenir facilement à une version précédente si nécessaire.	*ReplicaSet* ne fournit pas cette fonctionnalité.

Les objets K8s

StatefulSet: C'est un élément qui représente un réplica de Pods qui représentent des applications avec état comme les bases de données

StatefulSets nécessite actuellement un service Headless responsable des identités des Pod

Le Pod StatefulSet a une identité unique qui consiste en un ordinal, un identité réseau stable et stockage stable non modifiable.

Les objets K8s

Service: C’est l’élément permettant la communication inter Pods et la communication avec le client final via trois types essentiels de services **Cluster IP**, **NodePort** et **LoadBalancer**

ClusterIP : C'est le type par défaut. Il expose le Service sur une adresse IP interne du cluster. De ce fait, le service n'est accessible que depuis l'intérieur du cluster.

NodePort : Il expose le service vers l'extérieur du cluster à l'aide du NAT (la plage de ports autorisés est entre 30000 et 32767).

LoadBalancer : Il utilise l’équilibreur de charge des fournisseurs de cloud. Ainsi, les services NodePort et ClusterIP sont créés automatiquement et sont acheminés par l'équilibreur de charge externe.

Les objets K8s

Exemple d'exposition avec un service Cluster IP:

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

Deployment

```yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: ClusterIP # ligne optionnelle
  selector:
    app: nginx
  ports:
    - port: 5000
      targetPort: 80
```

Service Cluster IP

Les objets K8s

Exemple d'exposition avec un service avec NodePort:

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

Deployment

```yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: ClusterIP # ligne optionnelle
  selector:
    app: nginx
  ports:
    - port: 5000
      targetPort: 80
      nodePort: 30080
```

Service Cluster IP

Les objets K8s

Au lieu de rédiger un template YAML du service, il est possible de créer un service à l'aide de la commande **kubectl expose** à fin d'exposer les pods du Deployment.

Après ceci, on expose les pods de notre Deployment :
kubectl expose deployment nginx-deployment --name myapp-service --type ClusterIP --protocol TCP --port 5000 --target-port 80 --selector='app=myapp'

Voici la commande pour exposer notre Deployment avec un service de type NodePort :

kubectl expose deployment my-deployment --name myapp-service --type NodePort --protocol TCP --port 5000 --target-port 80 --nodePort 30080 --selector='app=myap

Note: Un petit rappel la plage d'IP dans un service de type NodePort se situe entre 30000 et 32767.

Les objets K8s

namespace: Les espaces de noms sont destinés à être utilisés dans des environnements comportant de nombreux utilisateurs répartis sur plusieurs équipes ou projets

Les espaces de noms fournissent une portée pour les objets k8s

Kubernetes inclut cet espace de noms par défaut

Pour lister les espaces nom:

kubectl get ns| namespaces

Pour créer un espace nom:

kubectl create ns <nom d'espace nom>

Pour lister les objets k8s sous un espace nom:

kubectl get all –namespace | -n <nom de l'espace>

Pour supprimer un espace nom:

kubectl delete ns | namesace mynamespace

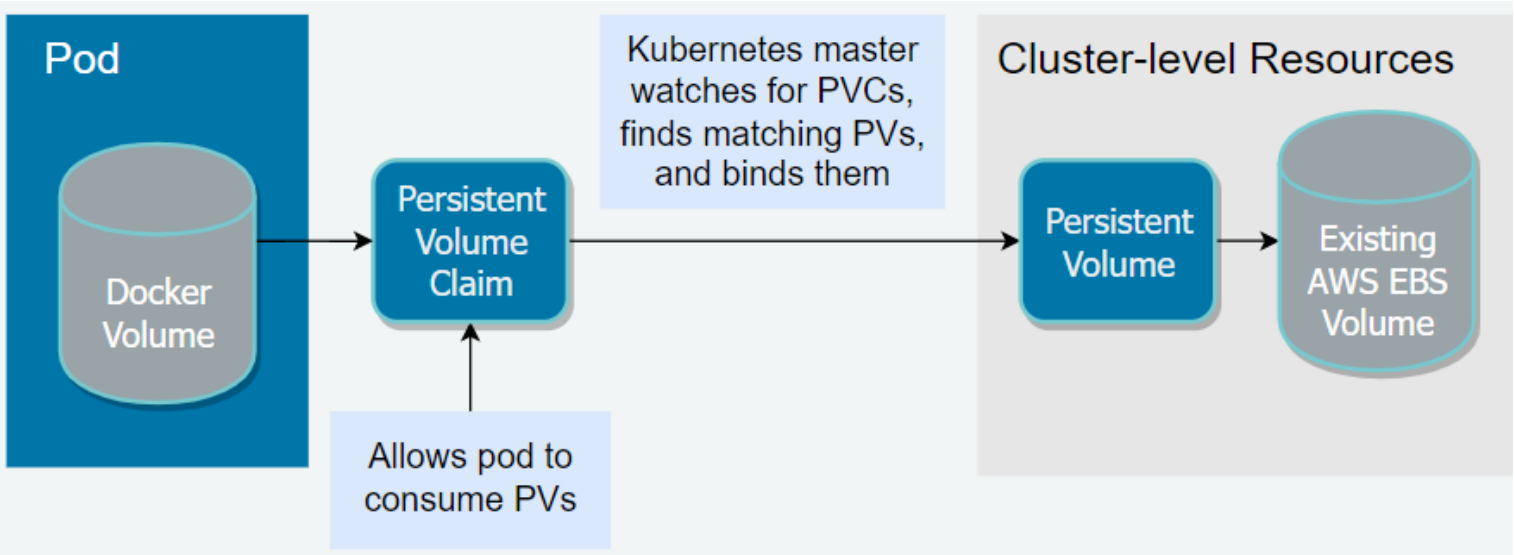
Pour ajouter une ressource à un espace nom:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: mynamespace
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
  namespace: mynamespace
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
    - port: 5000
      targetPort: 80
      nodePort: 30080
```

Les objets K8s

PV,PVC & StorageClass: Ces objets sont inter dépendants, ils permettent de persister les données exploitées par les Pods



Les objets K8s

PV,PVC & StorageClass: Ces objets sont inter dépendants, ils permettent de persister les données exploitées par les Pods

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
spec:
  capacity:
    storage: 2Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /tmp/data
```

1

Exemple de Persistent Volume

G: Giga octets

1G=10^3 est une puissance de dix. le résultat est 1000

Gi: Gibi octets

1Gi=2^10 est la puissance de deux. le résultat est 1024

Les objets K8s

PV,PVC & StorageClass: Ces objets sont inter dépendants, ils permettent de persister les données exploitées par les Pods

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
spec:
  capacity:
    storage: 2Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /tmp/data
```

Exemple de Persistent Volume

Filesystem: Le mode par défaut, les données sont persistées dans le système de fichier de la machine hôte

Block: Ce mode est mieux adapté aux bases de données, les charges élevées cas de simulations, les anciennes applications, l'applications demandant un niveau de sécurité élevé car sa fournit un degré d'isolation élevé

Les objets K8s

PV,PVC & StorageClass: Ces objets sont inter dépendants, ils permettent de persister les données exploitées par les Pods

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
spec:
  capacity:
    storage: 2Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /tmp/data
```

3

Exemple de Persistent Volume

ReadWriteOnce (RWO): Le volume peut être monté en lecture-écriture par un seul nœud

ReadWriteMany(RWM): Le volume peut être monté en lecture-écriture par de nombreux nœuds

ReadOnlyMany(ROM): Le volume peut être monté en lecture seule par de nombreux nœuds

ReadWriteOncePod(RWOP): Le volume peut être monté en lecture-écriture par un seul Pod

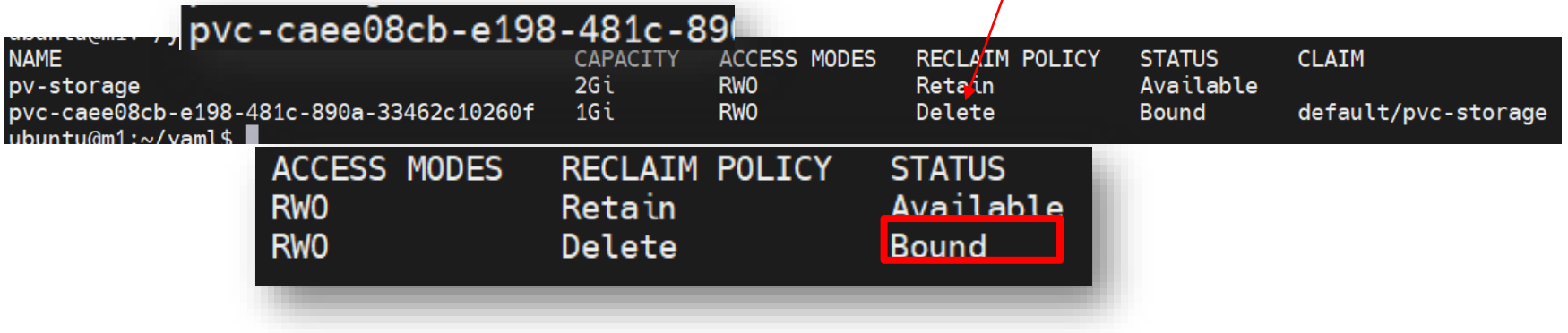
Les objets K8s

Il faut noter que la création d'un PVC va engendrer la création automatique d'un objet PV, lié déjà à ce premier

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-storage
spec:
  resources:
    requests:
      storage: 1Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
```

PV sera effacé si PVC supprimé

Exemple de Persistent Volume Claim



Les objets K8s

Voici un exemple:

https://github.com/kubernetes/examples/blob/master/mysql-wordpress-pd/mysql-deployment.yaml

Les objets K8s

La concept de StorageClass:

Il existe deux modes d'allocation de volumes en Kubernetes

Provisionnement statique:

Il s'agit de la méthode par laquelle l'administrateur du cluster provisionne manuellement les PV.

Approvisionnement dynamique:

A chaque fois qu'un POD nécessite de l'espace de stockage, le cluster Kubernetes fournit dynamiquement le PV requis grâce à l'objet StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: my-storage
provisioner: kubernetes.io/gce-pd
```

Exemple de Persistent Volume

Les objets K8s

ConfigMap: C'est un élément qui permet de stocker les paramètres utilisés par certains objets comme le nom de base de données, la valeur d'un port sous forme clé/valeur au lieu de la définir en dure ou au niveau d'un fichier de configuration

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: redis-configmap
data:
  redis-config: ""
```

Config map

Pour afficher les config map:

kubectl get cm | configmap

Pour utiliser les config map avec des POD:

https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/

Les objets K8s

Secret: Similaire à un objet **ConfigMap** sauf que l'objet en question est stocké sous forme cryptée

Note: La taille maximale des valeurs stockées est 1 M octet

Pour afficher les secrets:

kubectl get secret

Les types de secrets:

- Opaque
- Service Account Token
- Basic authentication
- SSH authentication
- TLS
- Docker config
- Bootstrap token

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
  namespace: default
type: Opaque
# valeurs encodées déjà
data:
  username: cm9vdCA=
  password: dGVzdDEyMysr
# valeurs à encoder base64
stringdata:
  username: root
  password: test123++
```

Secret

Pour utiliser les secrets:

https://kubernetes.io/docs/tasks/configmap-secret/managing-secret-using-kubectl/

Les objets K8s

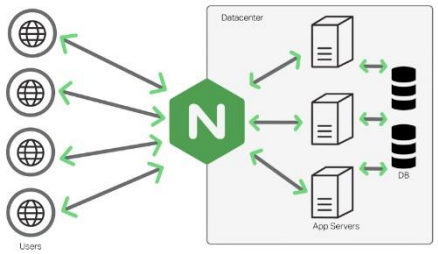
Ingress: C'est un composant k8s qui couvre deux fonctions

Routage

Sécurisation du trafic http en https

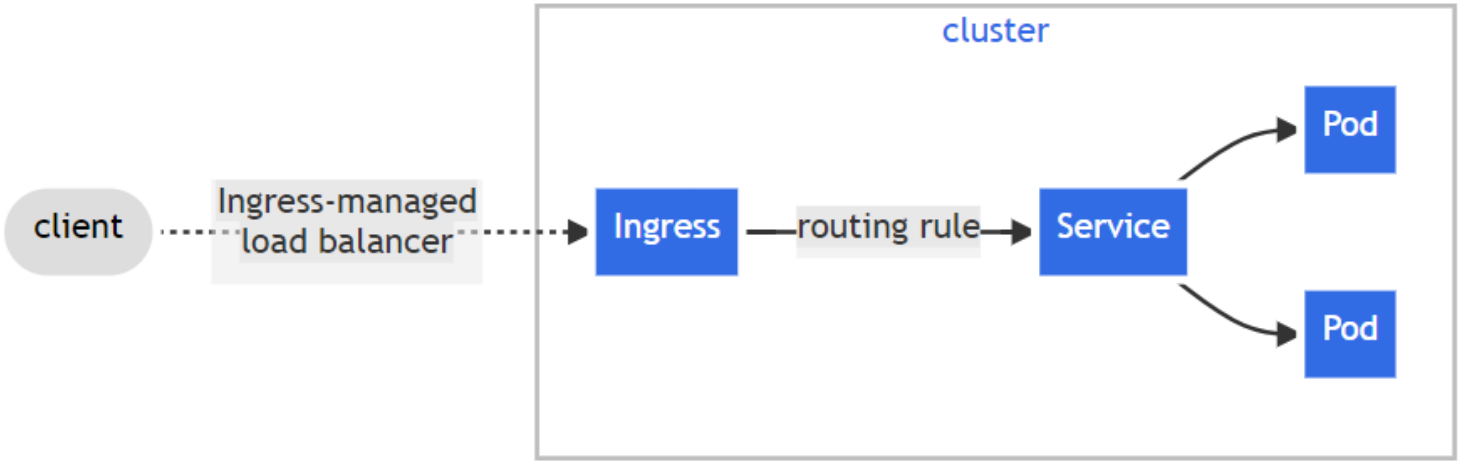
Ingress = Ingress ressource + Ingress controller

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```



Les objets K8s

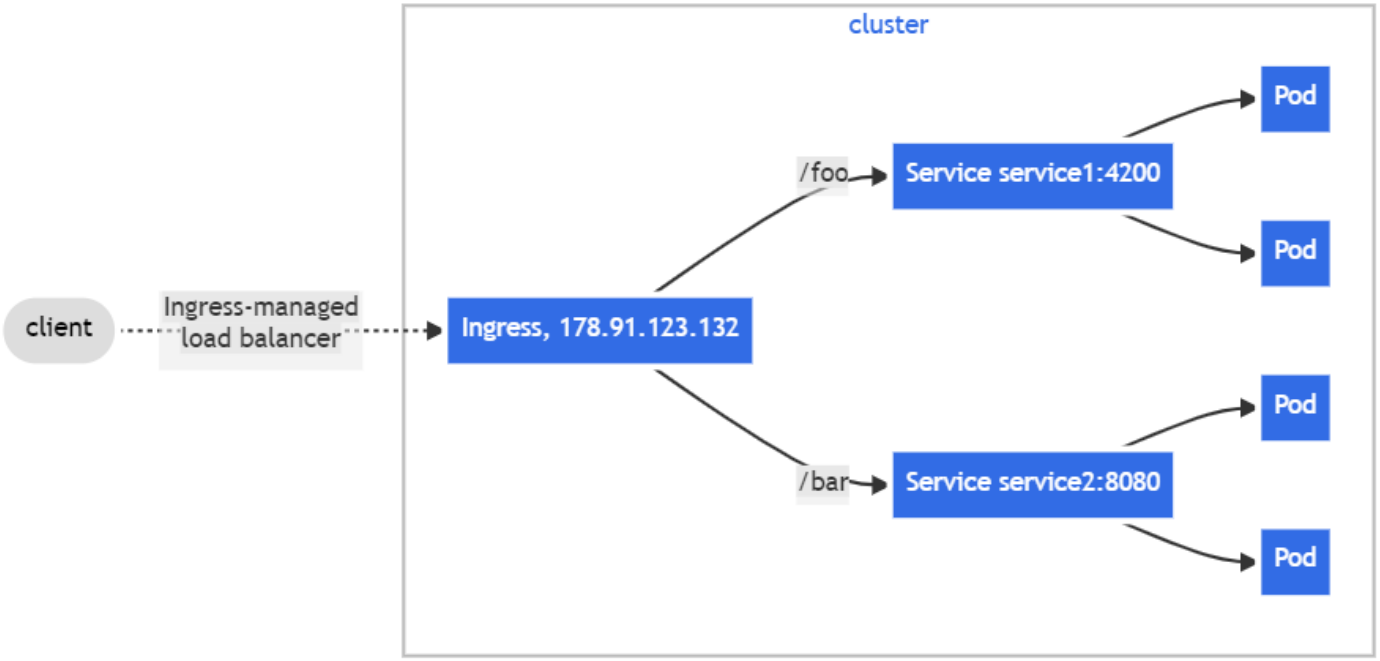
Les types de Ingress:



Le type Ingress simple

Les objets K8s

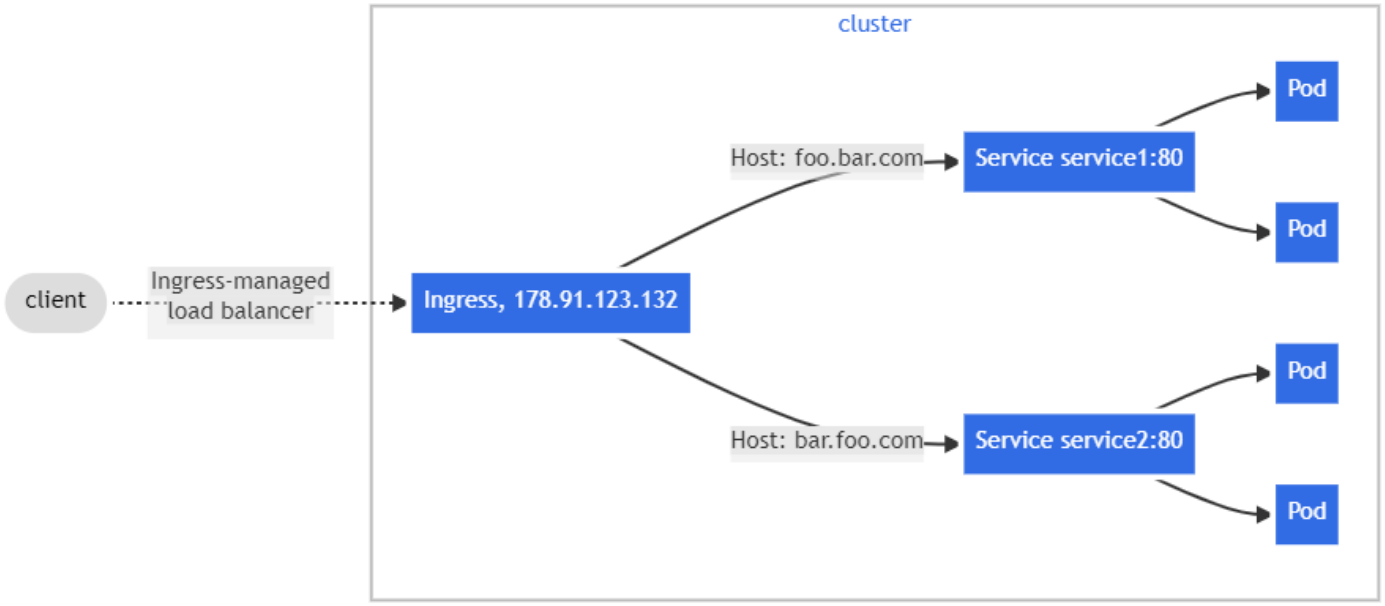
Les types de Ingress:



Le type Ingress Fanaout

Les objets K8s

Les types de Ingress:



Le type Ingress basé sur le nom du domaine

Les objets K8s

Les règles de Ingress:

Hôte	En-tête de l'hôte	Correspondre?
*.foo.com	bar.foo.com	Correspondances basées sur le suffixe partagé
*.foo.com	**baz.bar**.foo.com	Aucune correspondance, le caractère générique ne couvre qu'une seule étiquette DNS
*.foo.com	foo.com	Aucune correspondance, le caractère générique ne couvre exactement une seule étiquette DNS

Les objets K8s

Les règles de Ingress:

Il existe trois règles prises en charge pour le type Fanaout :

Exact : Correspond exactement au chemin de l'URL et en respectant la casse.

Prefix : Correspondances basées sur un préfixe de chemin d'URL divisé par /. La correspondance est sensible à la casse

ImplementationSpecific : Le comportement de la façon dont le chemin est mis en correspondance et géré est laissé à l'implémentation du contrôleur Ingress

Il existe une règle pour le type basé sur le domaine :

Les hôtes peuvent être des correspondances précises par exemple « foo.bar.com» ou un caractère générique *.foo.com