Charles SIEPEN

IUT Grand Ouest Normandie
BUT Informatique - Semestre 5
Module R5.09 - Virtualisation Avancée

16 novembre 2025

Contenu basé sur le cours Linux Foundation - Introduction to Kubernetes

Plan

Contexte

Objectifs

Ce CM consolide vos connaissances TD/TP sur 3 axes essentiels :

1. **Labels et Annotations** : Différence et nomenclature
2. **Ingress, Services et Pods** : Flux réseau complet
3. **Volumes** : Persistance vs non-persistance

Point clé

Ces concepts sont **fondamentaux** et évalués au CC.

Partie 1

Labels et Annotations

Organisation et métadonnées des ressources

Règle d'or

Labels → Sélection et comportement K8s

Annotations → Métadonnées (sans sélection)

Impact

- **Labels** : Utilisés par les `selectors` (Services, Deployments...)
- **Annotations** : Informations techniques pour outils externes

Labels - Caractéristiques

Rôle :

- Sélectionner des ensembles d'objets via `selectors`
- Organiser et regrouper les ressources
- Influencer le comportement de Kubernetes

Contraintes :

- Clé : max 63 caractères
- Valeur : max 63 caractères
- Format : alphanumérique, tirets, underscores, points

Labels - Exemple pratique

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  selector:
    app: webapp          # Selectionne les pods avec ce label
    tier: frontend       # ET ce label
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Résultat : Le Service route le trafic vers tous les pods ayant `app: webapp` **ET** `tier: frontend`.

Labels - Nomenclature recommandée

Attention

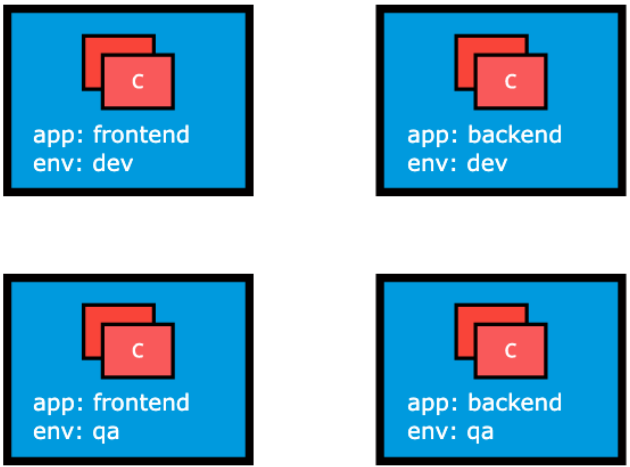
Une nomenclature incohérente peut causer :

- Collisions entre sélecteurs
- Règles ambiguës (NetworkPolicy)
- Difficultés de maintenance

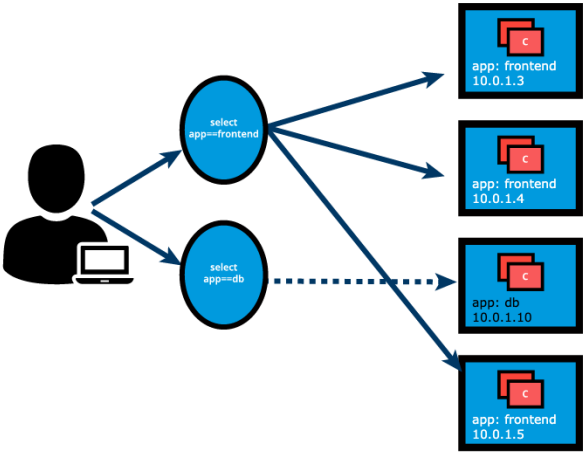
Labels Kubernetes standard :

- `app.kubernetes.io/name` : nom de l'application
- `app.kubernetes.io/instance` : instance spécifique
- `app.kubernetes.io/version` : version
- `app.kubernetes.io/component` : rôle (database, frontend...)

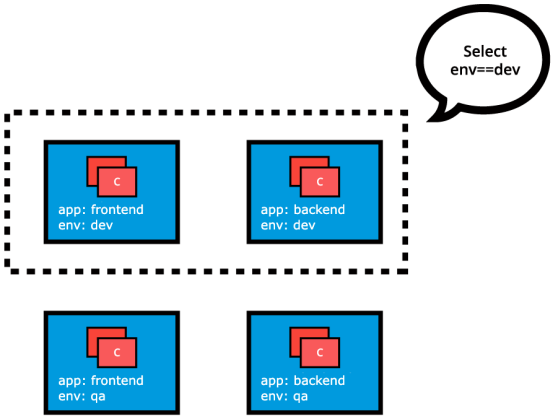
Labels et Selectors - Illustration



Groupement de Pods avec Labels et Selectors



Selectors en action



Annotations - Caractéristiques

Rôle :

- Stocker des informations techniques
- Configurer des outils externes (monitoring, CI/CD)
- Documenter les ressources
- Transmettre des données aux contrôleurs

Différences avec les labels :

- Aucune limite de taille stricte
- Peuvent contenir JSON/YAML
- **Ne participent PAS à la sélection**

Annotations - Exemple Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webapp-ingress
  annotations:
    # Configuration pour Ingress Controller
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/rate-limit: "100"

    # Metadonnees pour monitoring
    prometheus.io/scrape: "true"
    prometheus.io/port: "9090"
```

Labels vs Annotations - Récapitulatif

Critère	**Labels**	**Annotations**
Sélection ?	**OUI**	**NON**
Impact K8s ?	Oui	Non
Taille	Limitée	Flexible
Exemple	`app: frontend`	`description: ...`

Partie 2

Ingress, Services et Pods

Flux réseau et exposition des applications

Flux réseau - Vue d'ensemble

Chaîne du trafic

Client externe → Ingress Controller → Ingress (règles) → Service → Pods

Point clé

Chaque composant a un rôle distinct dans l'acheminement du trafic.

Distinction CRITIQUE

Ingress (objet)

- Ressource Kubernetes
- Décrit les règles de routage
- Fichier YAML déclaratif
- **Ne fait rien seul**

Ingress Controller

- Composant logiciel (pod)
- Reverse proxy (nginx, Traefik...)
- Implémente les règles
- **Gère le trafic réel**

Erreur fréquente

Créer un Ingress sans déployer d'Ingress Controller = **inutile !**

Contrôleurs Ingress populaires

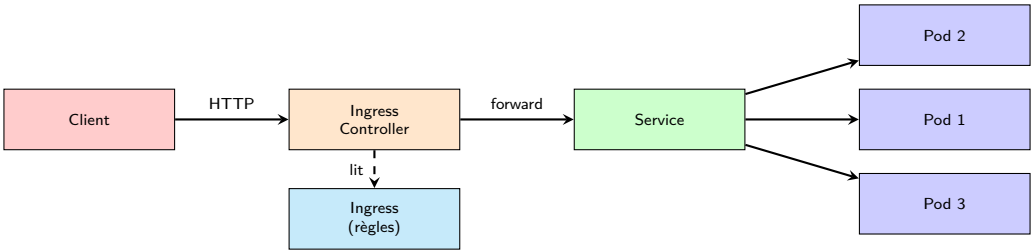
- **ingress-nginx** : Basé sur Nginx (le plus utilisé)
- **Traefik** : Moderne avec dashboard
- **HAProxy Ingress** : Performant pour charge élevée
- **Contour** : Basé sur Envoy proxy

Vérification :

Example (Commande)

`kubectl get pods -n ingress-nginx`
Doit afficher le pod `ingress-nginx-controller`

Flux complet illustré



Scénario : Utilisateur accède à `http://app.iut.local/api`

Services - Rôle et types

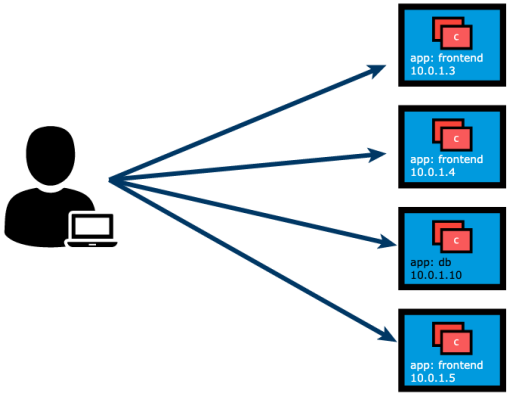
Abstraction pour cibler des pods via labels

- **ClusterIP** (défaut) : IP interne au cluster
- **NodePort** : Expose un port sur chaque nœud
- **LoadBalancer** : Provisionne un LB externe (cloud)

Fonctionnement :

1. Service sélectionne pods via labels
2. Kubernetes crée des Endpoints (liste IPs)
3. Trafic réparti (load balancing round-robin)

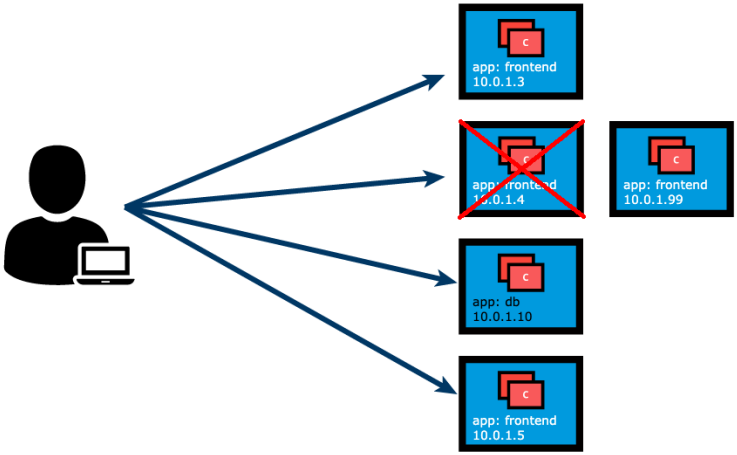
Accès via IP des Pods (problématique)



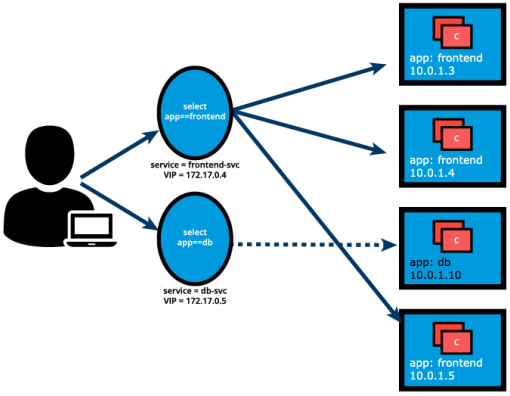
Problème

Les IPs des pods changent à chaque recréation !

Nouveau Pod après crash



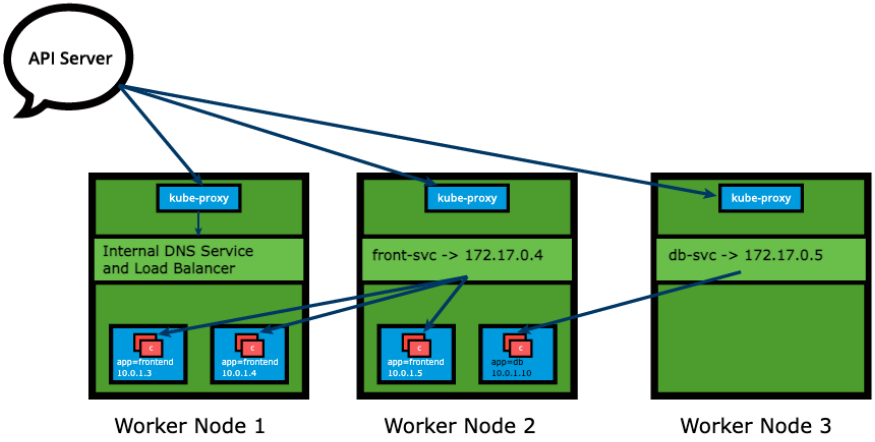
Solution : Service Object



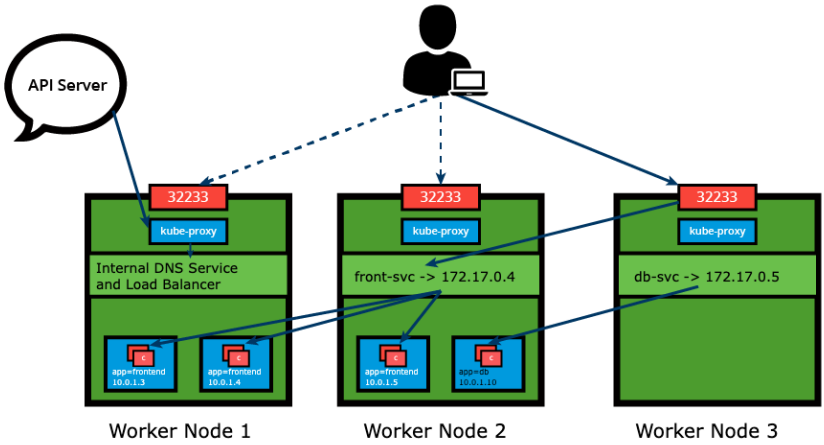
Avantage

Le Service maintient une IP stable et route vers les pods disponibles.

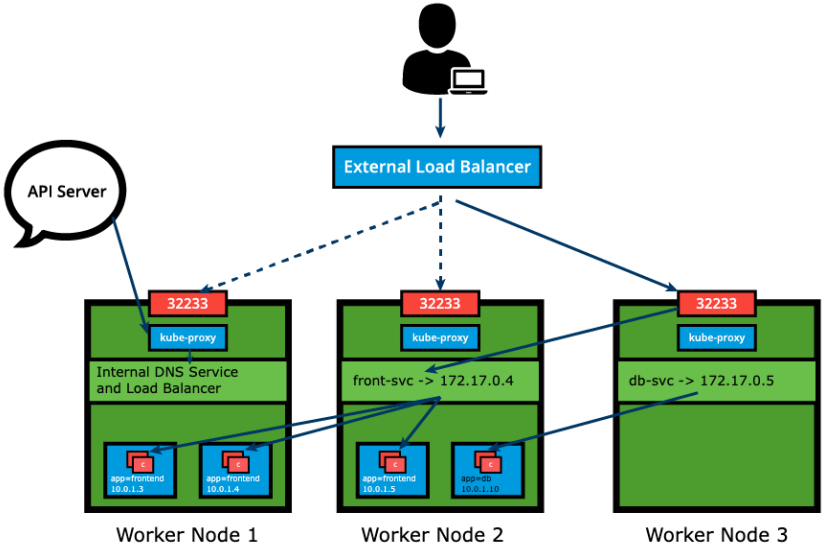
kube-proxy, Services et Endpoints



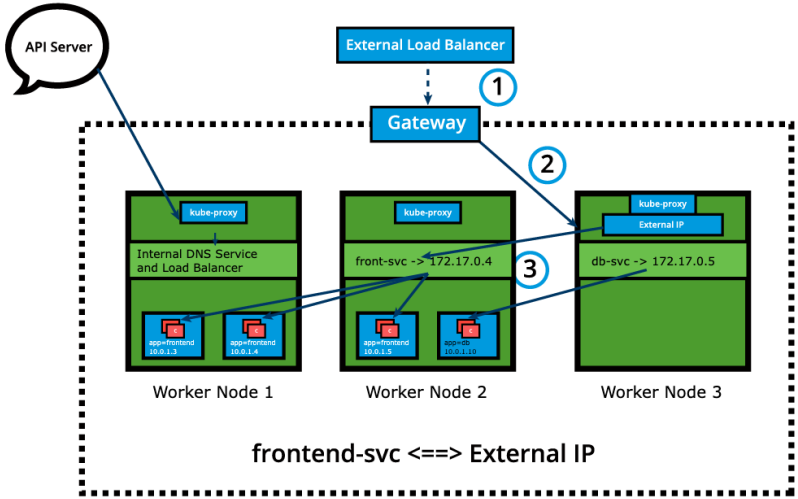
NodePort - Exposition externe



LoadBalancer - Cloud Provider



ExternalIP


API Server
External Load Balancer
1
Gateway
2
kube-proxy
Internal DNS Service
and Load Balancer
app=frontend
10.0.1.3
app=frontend
10.0.1.4
Worker Node 1
kube-proxy
front-svc -> 172.17.0.4
3
app=frontend
10.0.1.5
app=db
10.0.1.10
Worker Node 2
kube-proxy
External IP
db-svc -> 172.17.0.5
Worker Node 3
frontend-svc <==> External IP

NetworkPolicy - Egress

Contrôle du trafic sortant des pods

Cas d'usage

- Créer une whitelist interne
- Bloquer l'accès Internet
- Micro-segmentation réseau

Prérequis

Nécessite un CNI compatible (Calico, Cilium, Weave Net)

Solutions avancées (hors scope BUT)

Service Mesh (Istio, Linkerd)

- Chiffrement mTLS automatique
- Telemetry et tracing distribué
- Traffic management (retry, circuit breaker)
- Canary deployments

Note

Ces solutions dépassent le niveau BUT mais sont utilisées en production.

Partie 3

Volumes et Persistance

Stockage de données dans Kubernetes

Problématique

Concept fondamental

Les conteneurs sont **éphémères** : quand un pod est détruit, toutes ses données sont perdues.

Solution : Les volumes permettent de conserver les données au-delà du cycle de vie du pod.

Déclaration vs Montage

Deux étapes distinctes :

1. **Déclaration** du volume dans `spec.volumes` (niveau pod)
2. **Montage** du volume dans `volumeMounts` (niveau conteneur)

Analogie

- Déclaration = Créer le disque dur
- Montage = Brancher le disque dans l'ordinateur

Exemple - Déclaration et montage

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webapp
spec:
  # 1. DECLARATION (niveau pod)
  volumes:
    - name: config-volume
      configMap:
        name: app-config

  containers:
    - name: webapp
      image: nginx
      # 2. MONTAGE (niveau conteneur)
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
          readOnly: true
```

Volumes non persistants (éphémères)

emptyDir

- Créé avec le pod
- Supprimé avec le pod
- Partagé entre conteneurs

Cas d'usage :

- Cache temporaire
- Fichiers de travail

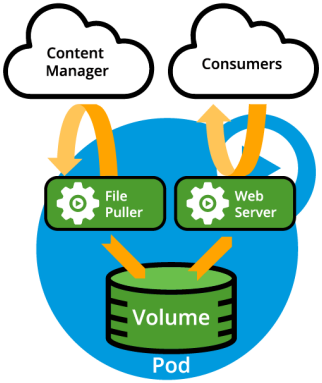
ConfigMap / Secret

- Lecture seule
- Lié au pod
- Configuration/credentials

Cas d'usage :

- Fichiers de config
- Variables d'env
- Certificats

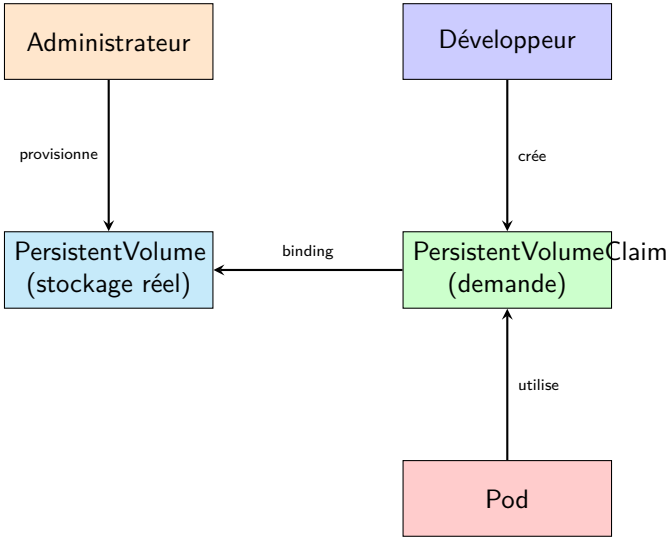
Shared Volume dans un Pod



Concept

Plusieurs conteneurs dans un même pod peuvent partager un volume.

Volumes persistants - Architecture PV/PVC



Séparation des responsabilités

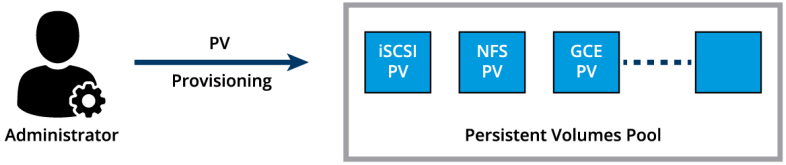
PersistentVolume - Access Modes

- `ReadWriteOnce (RWO)` : Lecture/écriture par un seul nœud
- `ReadOnlyMany (ROX)` : Lecture seule par plusieurs nœuds
- `ReadWriteMany (RWX)` : Lecture/écriture par plusieurs nœuds

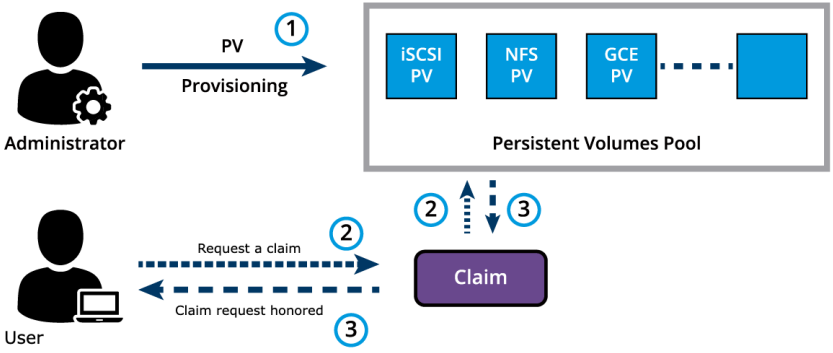
Reclaim Policy :

- `Retain` : Données conservées après suppression PVC
- `Delete` : Volume supprimé automatiquement
- `Recycle` : Données effacées, volume recyclé (déprécié)

PersistentVolume - Illustration



PersistentVolumeClaim



PVC utilisé dans un Pod

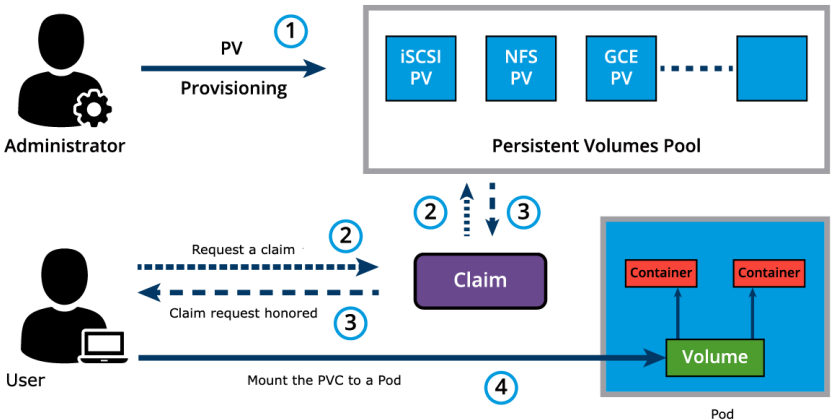


Tableau comparatif

Type	Cycle de vie	Cas d'usage	Persistant ?
`emptyDir`	Lié au pod	Cache temporaire	Non
`configMap`	Lié au ConfigMap	Configuration	Non
`secret`	Lié au Secret	Credentials	Non
`PV/PVC`	Découplé du pod	Bases de données	**Oui**

Règle simple

- **Stateless** (frontend) : Pas de PVC
- **Stateful** (database) : PVC obligatoire

Démonstration

Application front + back

Mise en pratique des concepts

Application front + back (chapitre 3)

Concepts illustrés :

1. **Load balancing** : Plusieurs réplicas backend
2. **High Availability** : Si un pod tombe, Service redirige
3. **Health checks** : Liveness & Readiness probes
4. **Ingress** : Routage basé sur path

Démo live

- Déployer backend (3 réplicas) + frontend (2 réplicas)
- Tester load balancing en tuant des pods
- Observer récupération automatique

Health Checks - Probes

```yaml
containers:
  - name: api
    image: backend:v1
    livenessProbe:
      httpGet:
        path: /health
        port: 3000
      initialDelaySeconds: 10
      periodSeconds: 5
    readinessProbe:
      httpGet:
        path: /ready
        port: 3000
      initialDelaySeconds: 5
```

Différence :

- **Liveness** : Échec → redémarrage pod
- **Readiness** : Échec → retrait des endpoints

Synthèse

Récapitulatif

Les points clés à retenir

Points clés - Labels vs Annotations

- **Labels** : Sélection (selectors) $\rightarrow$ comportement K8s
- **Annotations** : Métadonnées techniques (sans sélection)
- Nomenclature cohérente = éviter collisions

À retenir

Labels influencent le comportement, annotations NON.

Points clés - Ingress / Services / Pods

- **Ingress (objet)** : Règles de routage
- **Ingress Controller** : Implémentation (nginx, traefik...)
- **Service** : Abstraction ciblant pods via labels
- **NetworkPolicy Egress** : Contrôle trafic sortant

Critique

Ingress objet $\neq$ Ingress Controller. L'un ne fonctionne pas sans l'autre !

Points clés - Volumes

- **Déclaration** : `spec.volumes` (niveau pod)
- **Montage** : `volumeMounts` (niveau conteneur)
- **Non persistants** : emptyDir, ConfigMap, Secret
- **Persistants** : PV/PVC (découplés du pod)

Règle

- Stateless : Pas de PVC
- Stateful (BDD) : PVC obligatoire

Questions pour auto-évaluation

1. Quelle est la différence entre un label et une annotation ?
2. Que se passe-t-il si vous créez un Ingress sans Ingress Controller ?
3. Expliquez le flux HTTP de l'utilisateur jusqu'au pod.
4. Différence entre `spec.volumes` et `volumeMounts` ?
5. Pourquoi utiliser PVC au lieu d'emptyDir pour une BDD ?

Questions ?

Merci de votre attention

Prochaines étapes : Pratiquer avec chapitre 3 GitLab

Crédits et Ressources

Linux Foundation

Sources des illustrations et du contenu :

- **Linux Foundation** - Introduction to Kubernetes (LFS158x)
- Cours officiel : `https://training.linuxfoundation.org/training/introduction-to-kubernetes/`
- Documentation officielle Kubernetes : `https://kubernetes.io/docs/`