le cnam

GLG105: Génie logiciel

Pile de virtualisation du cloud - 2

November 19, 2025

Jacopo Bufalino (CNAM)

Récapitulatif du cours précédent

- Les machines virtuelles sont lentes
- Difficiles à maintenir
- Ne passent pas bien à l'échelle
- Les conteneurs sont rapides, faciles à partager et immuables
  - Résolvent le problème du "Mais ça marche sur ma machine"

Orchestrateurs de Conteneurs

Limitations des runtimes de conteneurs

Rappel du dernier cours : *les runtimes de conteneurs automatisent le cycle de vie de conteneurs individuels*.

Ce que les runtimes gèrent

- Démarrer et arrêter les conteneurs
- Gérer les processus des conteneurs
- Faire respecter les limites de ressources (CPU, mémoire)
- Isoler les conteneurs avec namespaces et cgroups

Ce qu'ils ne gèrent pas

Tout ce qui est nécessaire pour exécuter des groupes de conteneurs à grande échelle et sur plusieurs hôtes

En pratique

Gérer 5 conteneurs :

- ✓ Faisable
- ✓ Redémarrer les conteneurs en panne
- ✓ Modifier la configuration à la main
- ✓ Relancer les commandes échouées

Gérer 500 conteneurs:

- ✗ Des heures de travail manuel
- ✗ Sujet aux erreurs
- ✗ Impossible à maintenir

Exemple

Une application cloud typique a 20-50 conteneurs, chacun avec 3-10 réplicas sur plusieurs hôtes.

Limitation 1 : Gestion du stockage

Les runtimes de conteneurs supportent les volumes, mais n'ont pas d'orchestration de stockage distribué.

Opérations manuelles nécessaires

- Gérer le cycle de vie des volumes (création, suppression, sauvegarde)
- Garantir la disponibilité des données lorsque les conteneurs changent d'hôte
- Gérer les snapshots et la reprise après sinistre
- Coordonner le stockage partagé entre plusieurs conteneurs

Limitation 2 : Complexité réseau

Les runtimes de conteneurs ne fournissent qu'un réseau basique.

Opérations manuelles nécessaires

- Assigner des adresses IP uniques à chaque conteneur
- Configurer la résolution DNS entre services
- Mettre en place des load balancers pour distribuer le trafic
- Gérer le routage réseau à travers plusieurs hôtes
- Mettre à jour les services dépendants quand les IP changent

Exemple réseau

Un frontend et trois backends (A, B, C)

Statut initial :

- Backend-A : 172.17.0.2
- Backend-B : 172.17.0.3
- Backend-C : 172.17.0.4

Frontend configuré avec ces trois IP (par example round-robin)

Après crash de Backend-B :

- Backend-A : 172.17.0.2
- Backend-B : 172.17.0.9
- Backend-C : 172.17.0.4

Le frontend essaye encore 172.17.0.3 $\rightarrow$ échec

Limitation 3 : Passage à l'échelle

Les environnements d'exécution de conteneurs peuvent démarrer des conteneurs, mais ils ne peuvent pas les mettre à l'échelle de manière intelligente.

Des opérations manuelles sont nécessaires :

- Surveiller les métriques de l'application (CPU, mémoire, taux de requêtes)
- Réduire le nombre de conteneurs lorsque la charge diminue
- Choisir quel hôte dispose de la capacité disponible
- Enregistrer les nouvelles instances auprès des répartiteurs de charge
- Lancer automatiquement des conteneurs supplémentaires en cas de pic de trafic
- Vérifier la santé des conteneurs et les relancer automatiquement

Limitation 4 : Environnements multi-hôtes

- Les runtimes gèrent les conteneurs sur **un seul hôte**.
- Un système distribué nécessite plusieurs machines :
  - pour la haute disponibilité
  - pour la tolérance aux pannes
  - pour la répartition de charge
- Sans orchestrateur, l'opération manuelle est nécessaire pour chaque machine.

Les cycles d'exploitation deviennent ingérables

Pour chaque conteneur :

- Déploiement
- Mise à jour
- Surveillance
- Correction en cas de panne
- Réplication
- Mise à l'échelle
- Gestion du réseau
- Gestion du stockage

Problème

Sans automatisation, **le coût opérationnel dépasse rapidement le coût de développement**.

Orchestrateurs de conteneurs

Les orchestrateurs sont des systèmes distribués qui coordonnent plusieurs conteneurs sur un même hôte et entre différents hôtes.

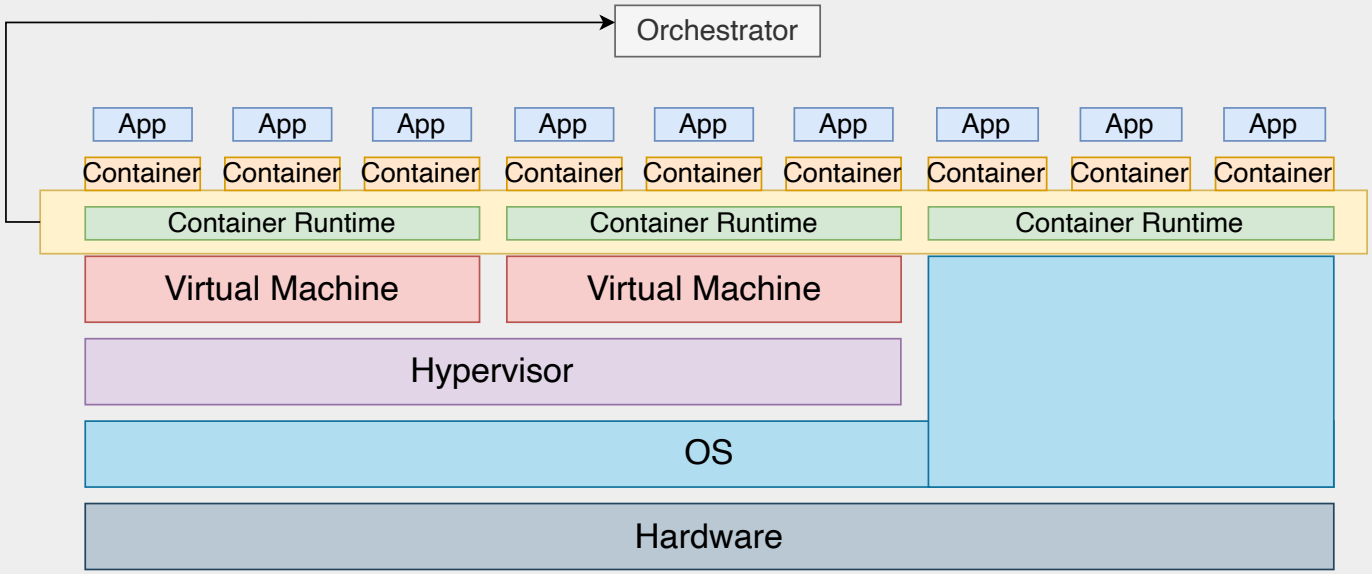
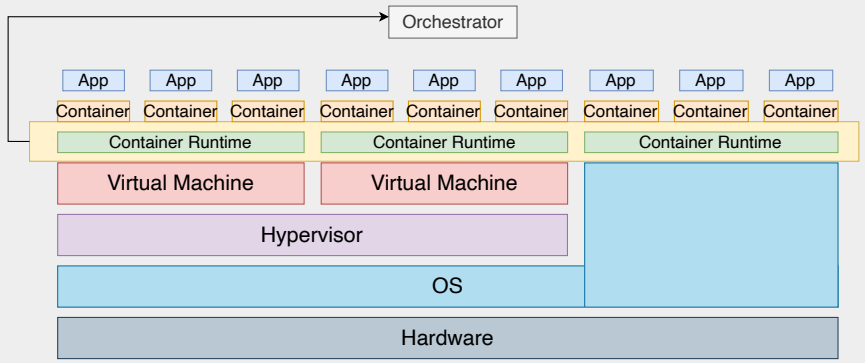


Figure: Vue d'ensemble d'un orchestrateur de conteneurs

Orchestrateurs de conteneurs : fonctionnalités clés

- Planification
- Scalabilité à travers des clusters de nœuds
- Découverte de services (DNS) et équilibrage de charge
- Configuration déclarative

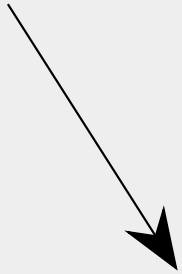
Analysons-les



Configurations déclaratives

Listing 1: Langage impératif

```
docker run -d nginx:1.14.2
docker run -d nginx:1.14.2
docker run -d nginx:1.14.2
docker run -d -n nginx-lb -p 80:80
```



Listing 2: Langage declaratif

```
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:1.14.2
```

Configurations déclaratives

- Simplifie la gestion en définissant un état final
- Améliore le contrôle de version et l'audit des configurations
  - Sécurité ?
- Cohérence et reproductibilité
- Déploiements idempotents
  - Appliquer la même configuration plusieurs fois donne le même état final.

Modèles de déploiement des orchestrateurs de conteneurs

Les orchestrateurs de conteneurs peuvent être déployés selon différents modèles :

- **Sur site :** Installés dans les centres de données de l'entreprise pour un contrôle total.
- **Basés sur le cloud :** Offerts sous forme de services managés par les fournisseurs (par ex. GKE, EKS, AKS).
- **Hybride :** Combine l'infrastructure sur site et les services cloud.

Déploiement sur site

Clusters Kubernetes auto-gérés fonctionnant dans les centres de données d'entreprise ou sur une infrastructure privée.

Avantages :

- Contrôle complet de l'infrastructure
- Souveraineté des données garantie
- Pas de dépendance à un fournisseur cloud
- Coûts prévisibles
- Contrôle de la latence réseau

Inconvénients :

- Investissement élevé
- Nécessite une expertise spécialisée
- Mises à jour et correctifs manuels
- Scalabilité limitée
- Charge de maintenance plus élevée

Basé sur le cloud (managé)

Les fournisseurs cloud proposent l'orchestration de conteneurs

Principaux services managés

- **Google Kubernetes Engine (GKE)**
- **Amazon Elastic Kubernetes Service (EKS)**
- **Amazon Elastic Container Service (ECS)**
- **OpenShift**

Basé sur le cloud : avantages

Avantages opérationnels :

- Pas de gestion du plan de contrôle
- Mises à jour automatiques disponibles
- Haute disponibilité et redondance intégrées
- Scalabilité élastique
- Mise en production plus rapide

Avantages de coût :

- Paiement à l'usage (PaaS)
- Moins de personnel requis
- L'auto-scaling réduit le gaspillage
- Moins de débogage
- Plan de contrôle gratuit (certains fournisseurs)

Basé sur le cloud : inconvénients potentiels

- **Dépendance au fournisseur :** intégrations et fonctionnalités spécifiques au cloud
- **Coûts imprévisibles :** peuvent augmenter avec le trafic / le stockage
- **Résidence des données :** les données peuvent traverser des frontières géographiques
- **Contrôle limité :** impossible de personnaliser la configuration du plan de contrôle
- **Coûts réseau :** les frais de transfert de données peuvent être substantiels
- **Contraintes de conformité :** certains règlements interdisent l'hébergement dans le cloud

Basé sur le cloud (managé)

Les fournisseurs cloud offrent de l'orchestration de conteneurs

Principaux services managés

- **Google Kubernetes Engine (GKE)**
- **Amazon Elastic Kubernetes Service (EKS)**
- **Amazon Elastic Container Service (ECS)**
- **OpenShift**

Modèle de déploiement hybride

Les architectures hybrides combinent l'infrastructure sur site avec les ressources cloud.

Schémas hybrides courants

- **Cloud bursting** : Infrastructure sur site comme primaire, cloud pour la capacité d'appoint
- **Localité des données** : Calcul dans le cloud, données sensibles sur site
- **DR/Sauvegarde** : Production sur site, reprise après sinistre dans le cloud
- **Chemin de migration** : Transition progressive du sur site vers le cloud
- **Edge + Core** : Traitement en edge sur site, traitement central dans le cloud

Déploiement multi-cloud

Les stratégies multi-cloud utilisent plusieurs fournisseurs cloud (sans composante sur site).

Motivations pour le multi-cloud

- **Éviter l'enfermement fournisseur** : Ne pas dépendre d'un seul fournisseur
- **Couverture géographique** : Utiliser le meilleur fournisseur par région
- **Optimisation des coûts** : Tirer parti des prix compétitifs
- **Résilience** : Réduire l'impact des pannes de fournisseur
- **Conformité** : Respecter les exigences de résidence des données par pays

Architecture microservices

L'architecture microservices est une approche consistant à construire des applications comme une collection de petits services indépendants.
L'une des raisons de cette approche est la montée en popularité des conteneurs.

Principes fondamentaux

Chaque service est :

- **Déployable indépendamment** : Peut être mis à jour sans affecter les autres
- **Faiblement couplé** : Dépendances minimales entre les services
- **Organisé autour des capacités métier** : Représente une fonction spécifique

Contraste avec l'architecture monolithique

Au lieu d'une grande application unique, elle est découpée en de nombreux petits services qui communiquent via des protocoles réseau.

Schémas courants d'orchestrateurs

Les orchestrateurs permettent de nouveaux schémas architecturaux en simplifiant la connectivité et la gestion des conteneurs.

Observations clés

- Les orchestrateurs permettent un déploiement facile des applications conteneurisées.
- Ajouter des fonctionnalités via de nouveaux conteneurs est souvent plus simple que des modifications de code
- De nouveaux schémas émergent pour les préoccupations transversales

Patron Sidecar

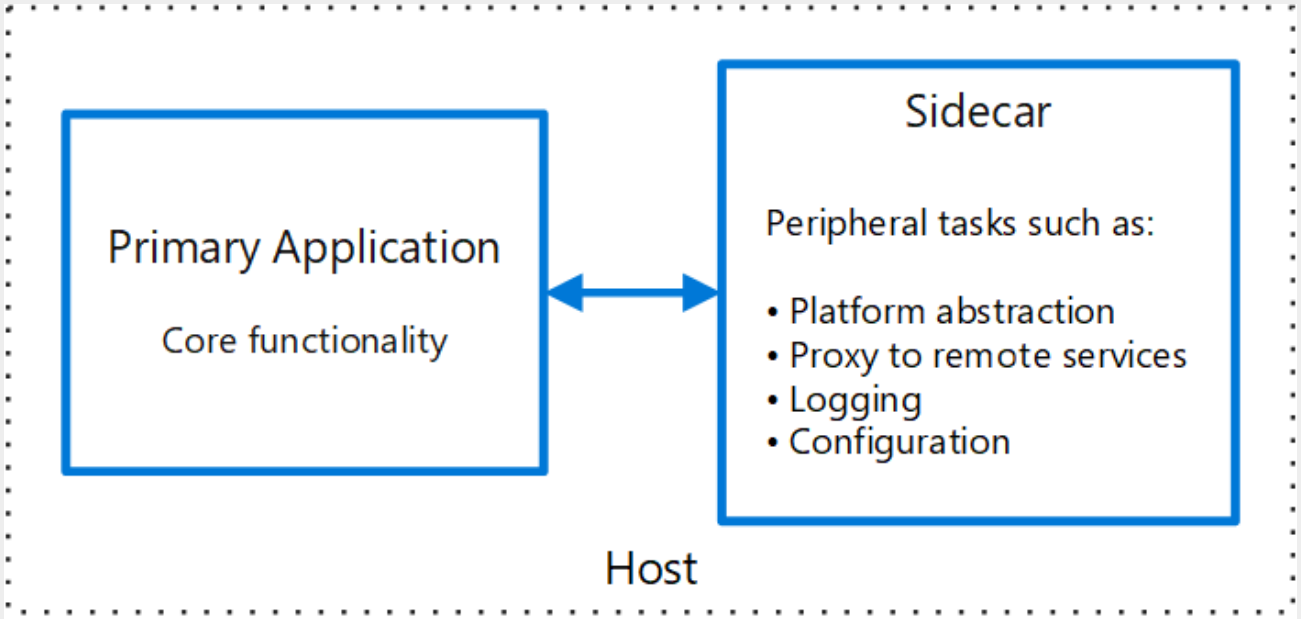
Les applications ont besoin d'exigences non fonctionnelles :

- Journalisation
- Monitoring
- Authentification
- Autorisation

Complexe et chronophage à ajouter à chaque application, spécialement lorsqu'on utilise différents langages de programmation.

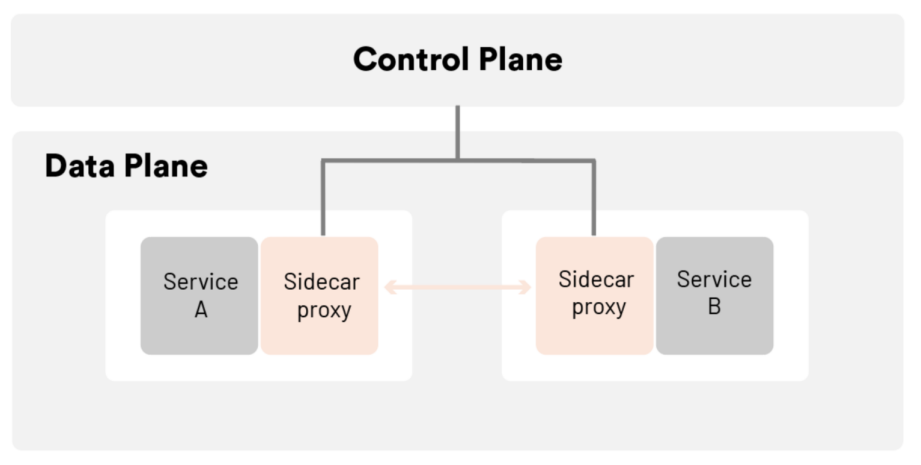
Patron Sidecar – partie 2

Comment ajouter des fonctionnalités sans écrire de nouveau code ?



[0] https://learn.microsoft.com/en-us/azure/architecture/patterns/sidecar

Service Mesh



- Un sidecar par application
- Le sidecar agit comme proxy vers/de l'application
- Plan de contrôle et plan de données séparés
  - Logique d'autorisation
  - Gestion du trafic
- Utile pour gérer les applications de manière centralisée

[0] https://tetrate.io/

Zero Trust

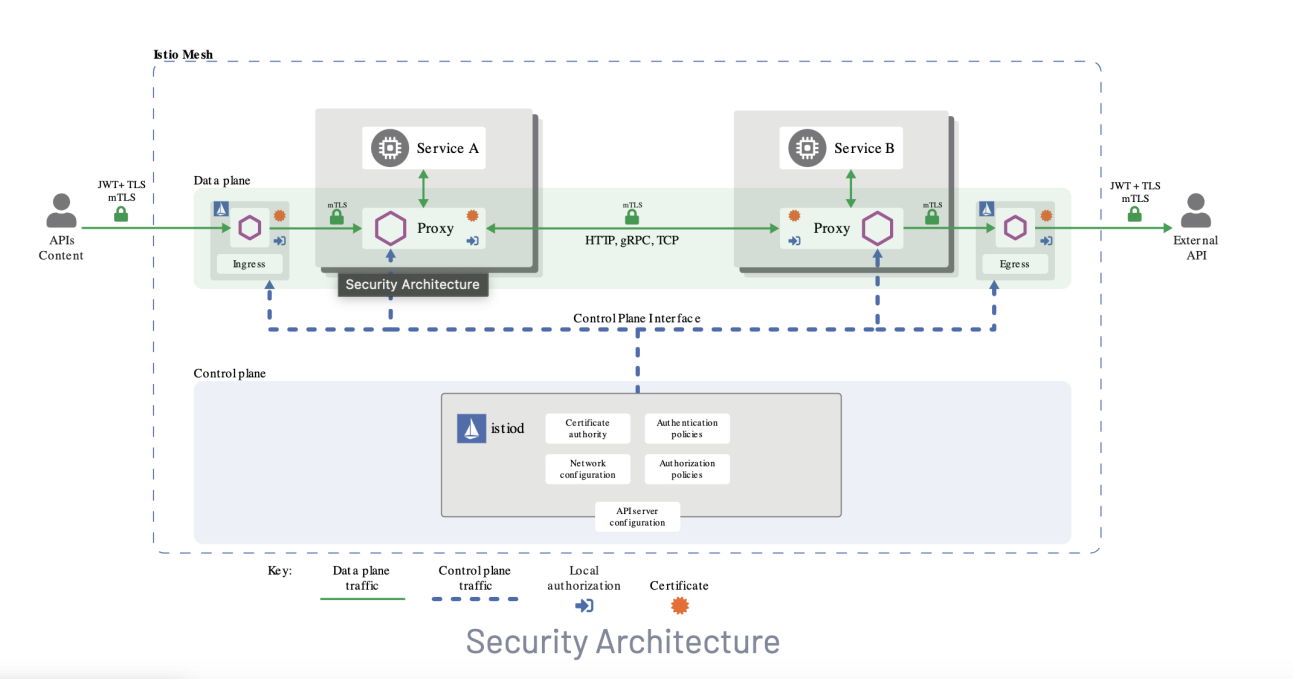


Figure: Source : istio.io

- **Ne jamais faire confiance, toujours vérifier :** Authentifier et autoriser continuellement chaque utilisateur, appareil et flux réseau.
- **Least Privilege Principle :** Accorder les droits minimaux nécessaires aux utilisateurs et systèmes pour accomplir leurs tâches

Kubernetes

Kubernetes

Kubernetes est l'orchestrateur le plus populaire pour les applications cloud conteneurisées.

Sa popularité est principalement due à

- Personnalisabilité : modules et interfaces enfichables
- Code open source
- Ensemble riche de fonctionnalités (p. ex., gestion des certificats TLS, secrets, répartition de charge)

Cas d’usage

Enhancing Patient Care Through Kubernetes-Powered Healthcare Data Management

U-2 Federal Lab achieves flight with Kubernetes

Spotlight on Tech

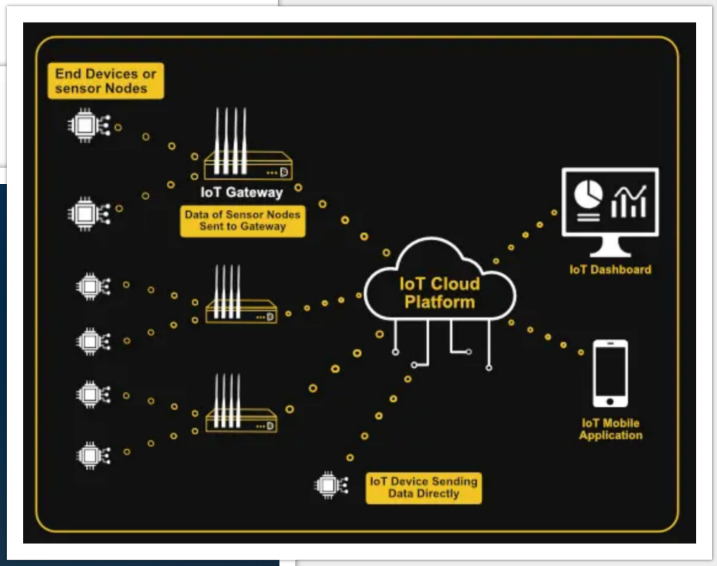
Better Telco with Kubernetes

By Brooke Frischemeier
Head of Product Management, Unified Cloud
Rakuten Symphony

Share this content

in Share | Tweet | Share | Share

August 3, 2023 | 12 minute read



Architecture

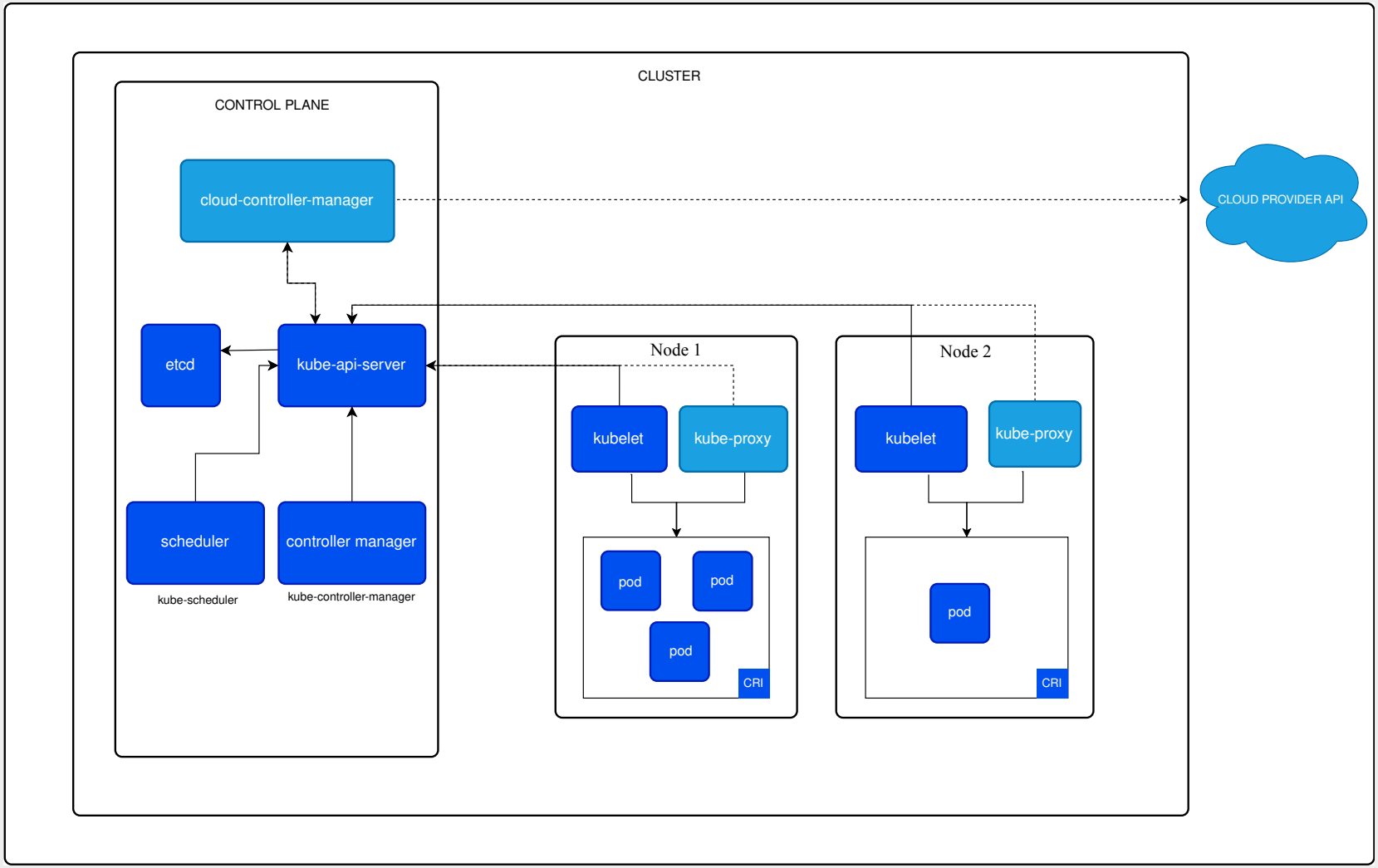


Figure: Architecture d'un cluster Kubernetes

Interaction avec le cluster Kubernetes

Les utilisateurs et applications interagissent avec Kubernetes via le Serveur API, qui est le composant central du plan de contrôle.

Méthodes d'interaction

- **kubectl** : Outil en ligne de commande (le plus courant)
- **Tableaux de bord** : Interface Web/Bureau
- **API REST** : Requêtes HTTPs directes vers le Serveur API
- **Bibliothèques clientes** : Go, Python, Java, JavaScript, etc.

Serveur API

Toute méthode d'interaction communique finalement avec le Serveur API Kubernetes via HTTPs.

Authentification et autorisation

L'accès à l'API Kubernetes est contrôlé par l'authentification et l'autorisation.

Méthodes d'authentification

- **Certificats X.509** : Les plus courants pour les utilisateurs
- **Bearer Tokens** : Tokens ServiceAccount, tokens OIDC
- **Basic Auth** : Nom d'utilisateur/mot de passe (obsolète)
- **OIDC** : Intégration avec les fournisseurs d'identité (Google, Azure AD)

Autorisation (ce que vous pouvez faire)

- **RBAC** : Contrôle d'accès basé sur les rôles (le plus courant)
- **ABAC** : Contrôle d'accès basé sur les attributs
- **Webhook** : Service d'autorisation externe

Un outil, différentes saveurs

Kubernetes est une plateforme d'orchestration unique, mais ouverte à différentes implémentations. Elle repose fortement sur des interfaces :

- Interface CRI : Container Runtime Interface
- Interface CNI : Container Network Interface
- Interface CSI : Container Storage Interface

CRI : Container Runtime Interface

CRI standardise la communication entre le kubelet et les runtimes de conteneurs.

Deux tâches principales

1. Gérer le cycle de vie des conteneurs
   - Créer/démarrer/arrêter des conteneurs
   - Exécuter des commandes (exec, attach)
   - Redirection de ports
2. Gérer les images de conteneurs
   - Tirer et lister les images
   - Cache et stockage des images
   - Supprimer les images inutilisées

Cela est totalement transparent pour l'utilisateur final

CNI : Container Network Interface

CNI standardise la façon dont les plugins réseau configurent le réseau des conteneurs.

Responsabilités du CNI

- Attribuer des adresses IP et configurer les interfaces réseau aux workloads
- Mettre en place les règles de routage et les politiques réseau
- Activer la communication pod-à-pod

Plugins CNI populaires

- Calico : Politiques réseau + routage
- Cilium : Basé sur eBPF, haute performance
- Flannel : Réseau overlay simple (trop simple)

Il n'y a qu'une seule interface commune : la NetworkPolicy (nous en discuterons plus tard)

CSI : Container Storage Interface

CSI permet aux fournisseurs de stockage de développer des plugins sans modifier le cœur de Kubernetes.

Capacités du CSI

- Provision/suppression de volumes persistants
- Attacher/détacher les volumes aux nœuds
- Monter/démonter des volumes dans les pods
- Snapshot et clonage de volumes
- Extension de volume

Drivers CSI

- Fournisseurs cloud : AWS EBS, Azure Disk, GCP PD
- Stockage réseau : NFS, Ceph, GlusterFS, SMB
- Stockage local : OpenEBS, Longhorn

CSI : Container Storage Interface

Il y a deux interfaces principaux: PersistentVolume et PersistentVolumeClaim.

La puissance des interfaces

Extensibilité de Kubernetes via les interfaces

- **CRI** : Choisir un runtime selon les besoins de sécurité/performance
- **CNI** : Sélectionner la solution réseau adaptée à votre architecture
- **CSI** : Utiliser n'importe quel backend de stockage sans enfermement fournisseur

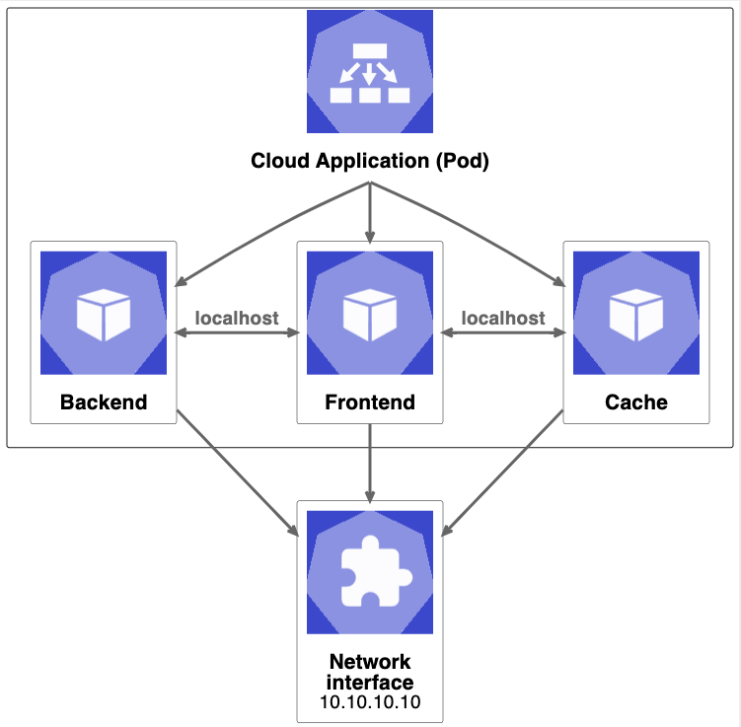
Workloads

Un workload est une application s'exécutant sur Kubernetes. L'unité minimale déployable est le Pod.

Pods

Les pods sont des groupes de conteneurs

- Étroitement couplés
- Cycle de vie commun
- Même Namespaces Linux



Les pods sont des entités éphémères

Ils peuvent être créés et détruits à tout moment.

Workloads

D'autres types de workloads ajoutent des fonctionnalités aux Pods.

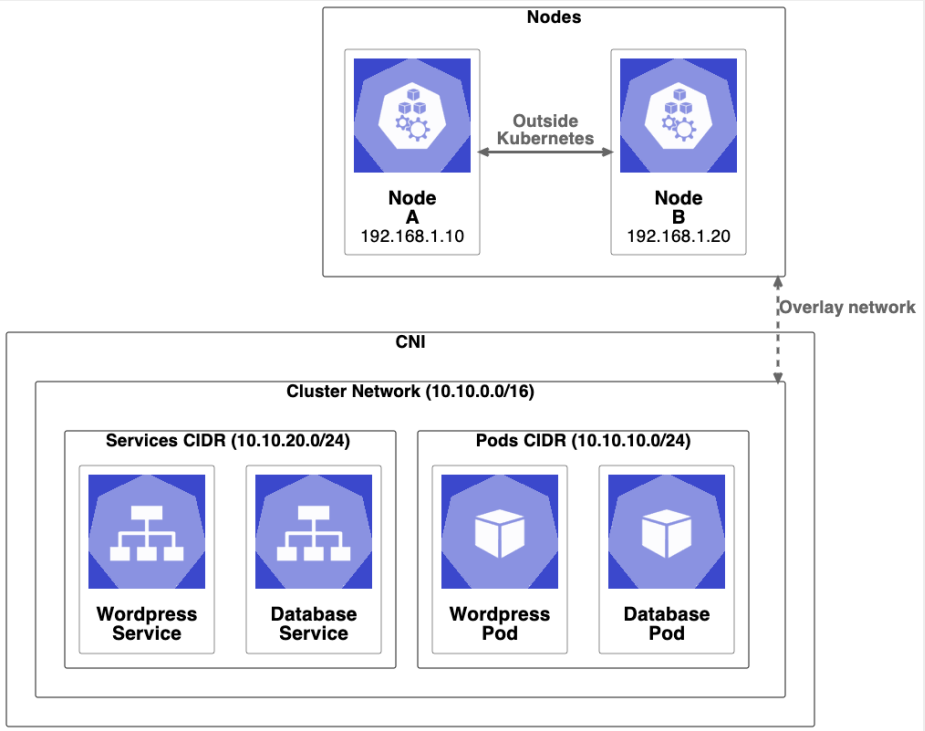
Exemples

- **Deployment** : répliquer des Pods et gérer différentes versions d'applications. Garantit aussi que les Pods sont recréés en cas d'échec.
- **CronJobs** : exécuter un Pod à intervalles définis.

Réseautage

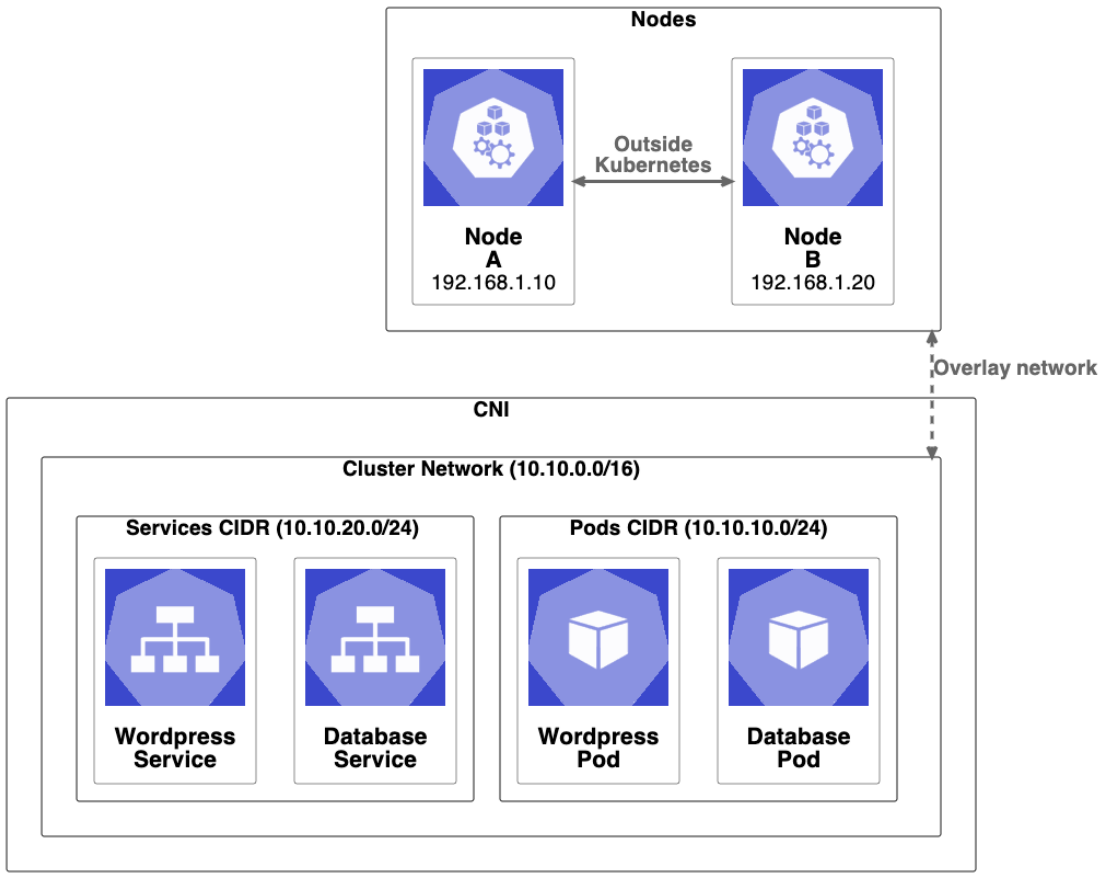
Il existe différents réseaux :

- Réseau des nœuds
- Réseau des conteneurs
- Réseau du cluster
  - Réseau des Services
  - Réseau des Pods



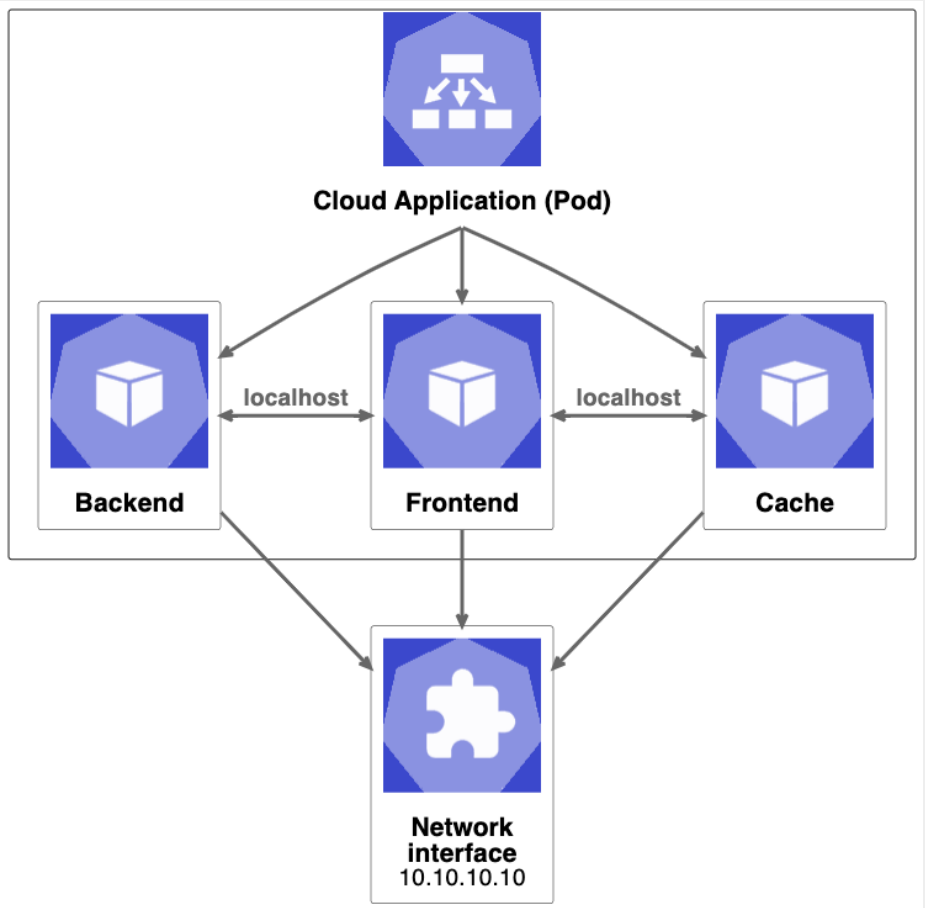
Réseau des nœuds

Les nœuds dans Kubernetes doivent être joignables. Kubernetes est responsable de sécuriser leur communication, par ex. en utilisant mTLS.



Réseau des conteneurs

- Un Pod contient plusieurs conteneurs
- Les mêmes Namespaces Linux
  - Un seul namespace réseau (et interface)
  - La même adresse IP



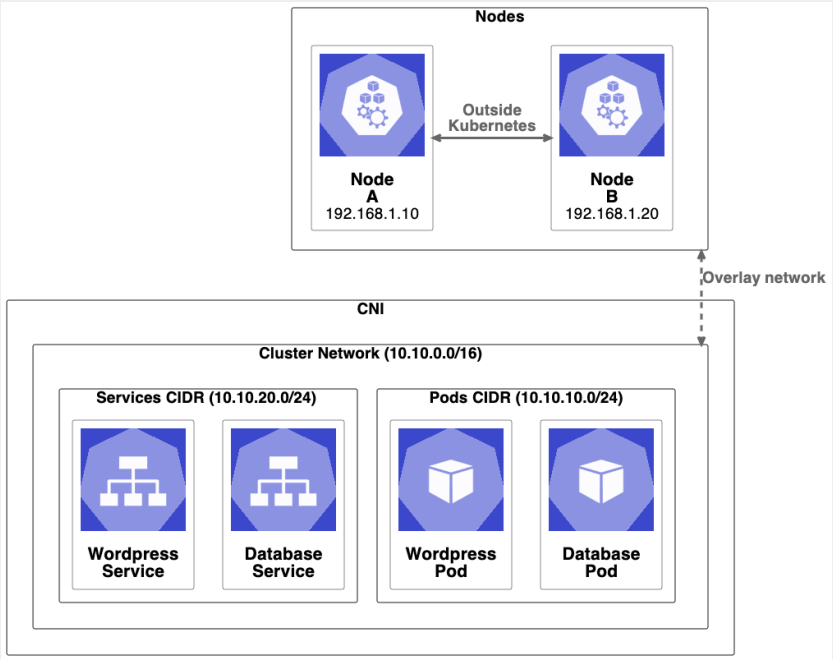
Les Pods sont des entités éphémères

Ils peuvent être créés et détruits à tout moment. Les adresses IP ne sont pas conservées entre les redémarrages.

Réseau du cluster

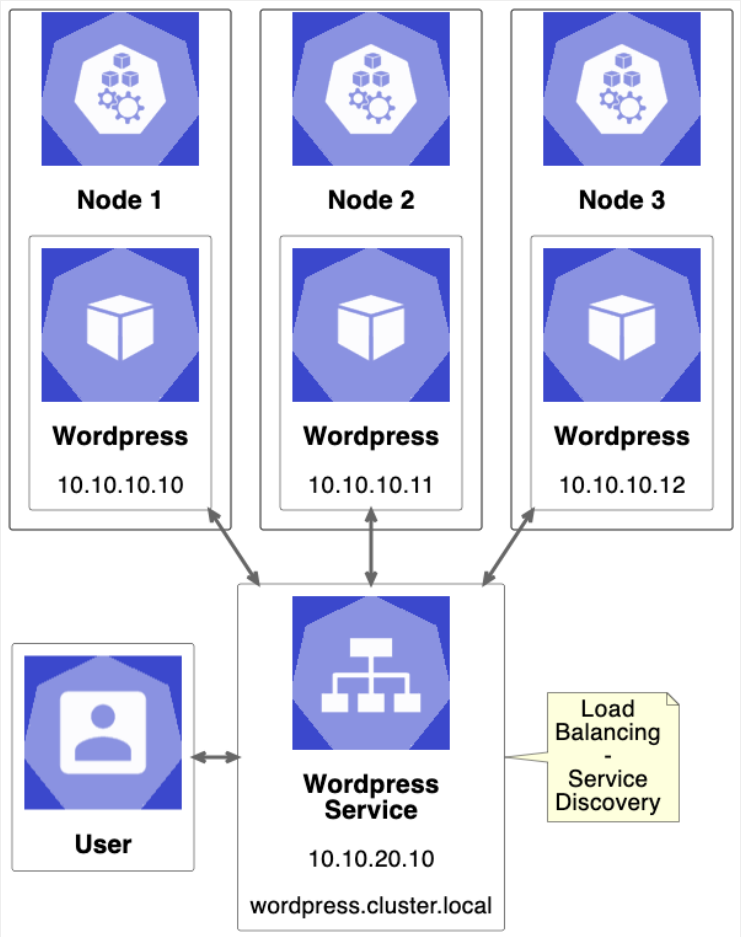
Le réseau du cluster est géré par le plugin CNI (Container Network Interface). Étant une interface, il permet différentes implémentations. Le CNI est responsable de :

- Attribuer des adresses IP aux Pods et Services
- Gérer le réseau overlay entre les nœuds
- Appliquer les politiques de sécurité réseau



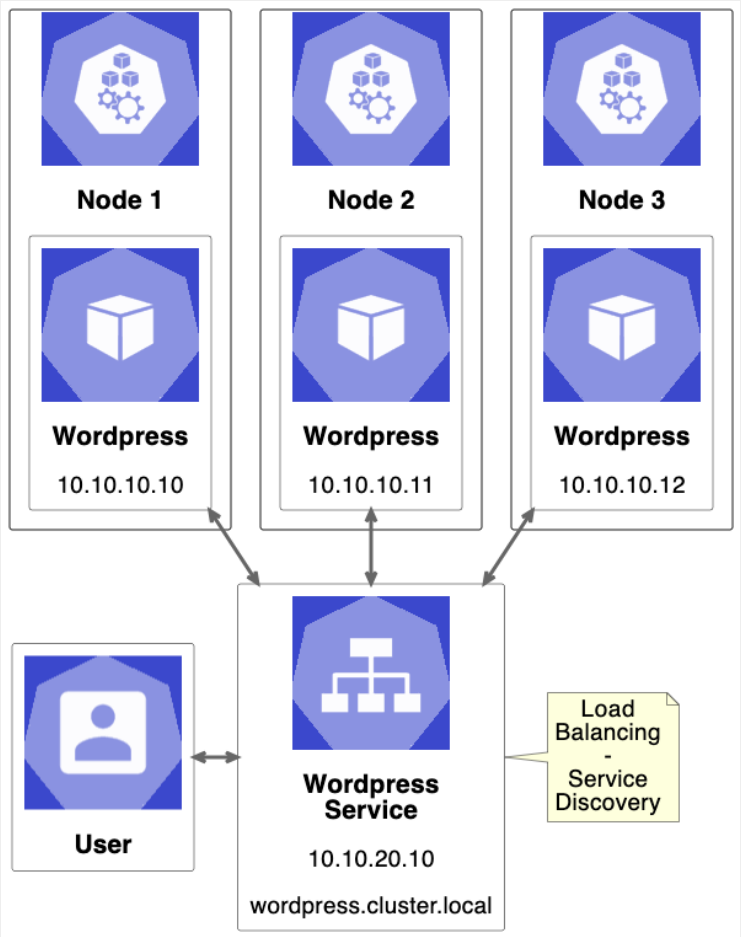
Services

Les Services sont essentiellement des reverse proxies qui assurent la répartition de charge entre les Workloads.



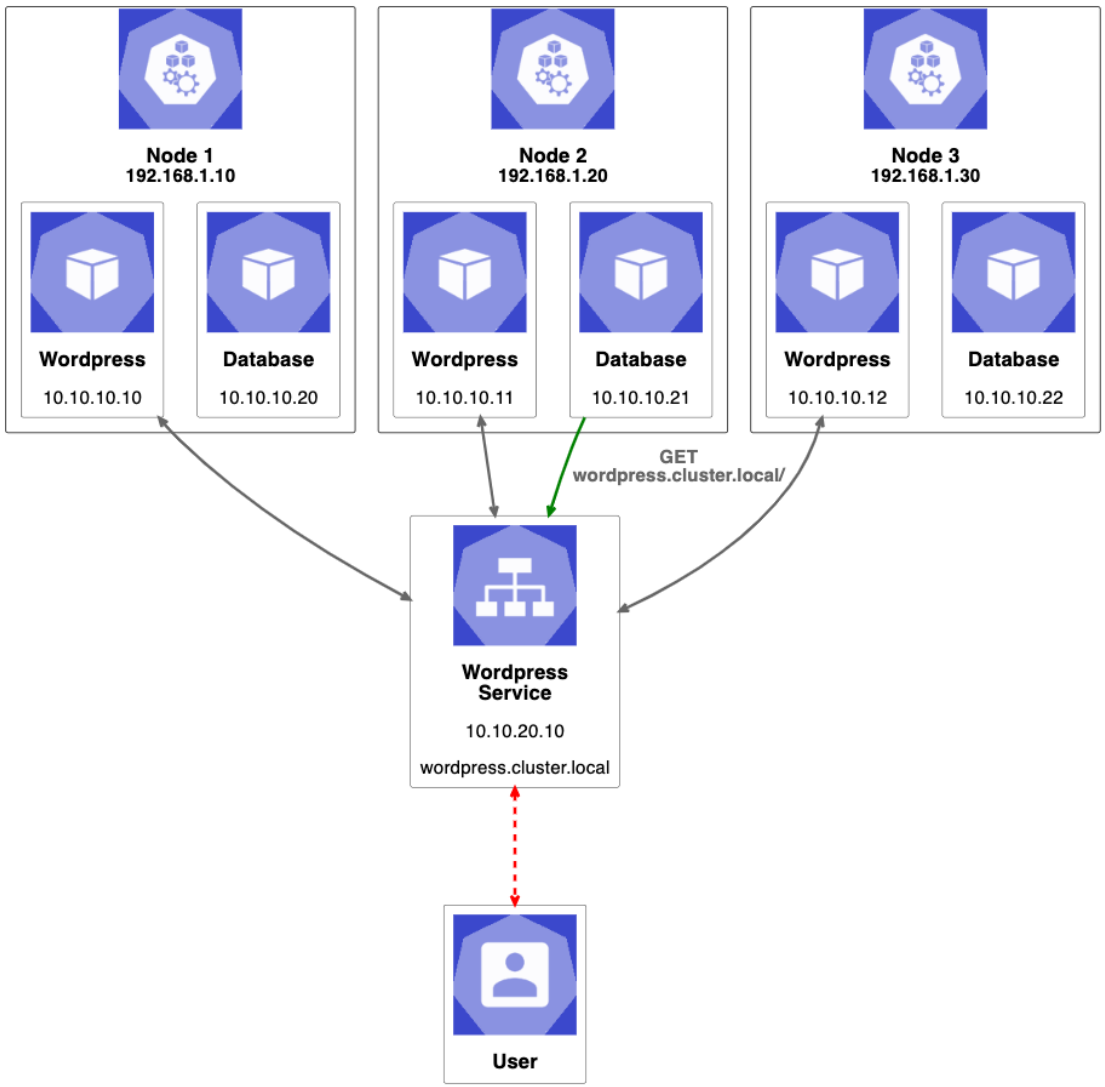
Services

Ils fournissent aussi une interface stable (IP) et/ou un nom de domaine. Kubernetes fournit également un équilibrage automatique de charge entre les Pods. Il existe cinq types de Services.



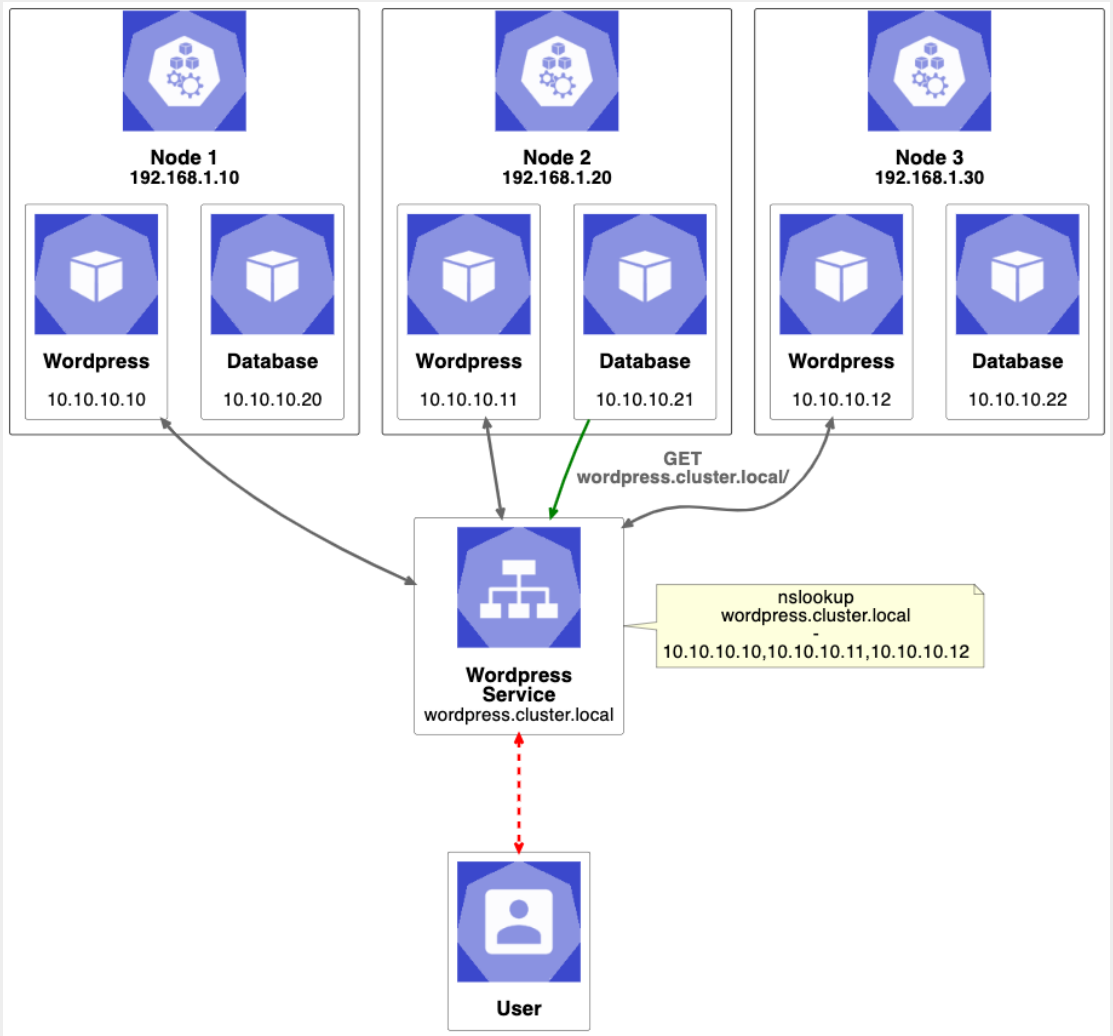
Service – Cluster IP

Cluster IP est accessible uniquement depuis l'intérieur du cluster.



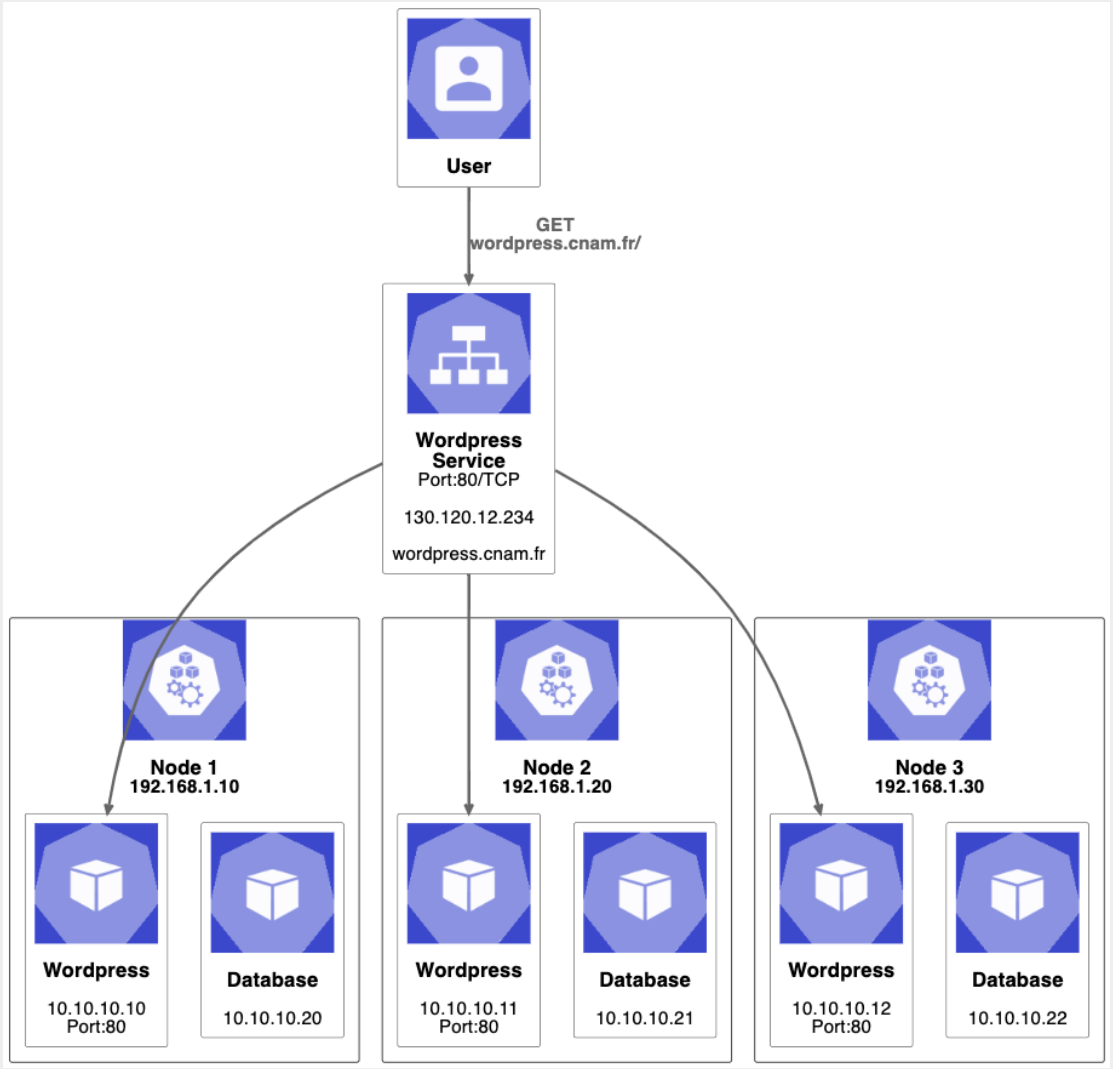
Service – Headless

Les Services headless n'ont pas d'adresse Cluster IP et sont utilisés pour les applications stateful.



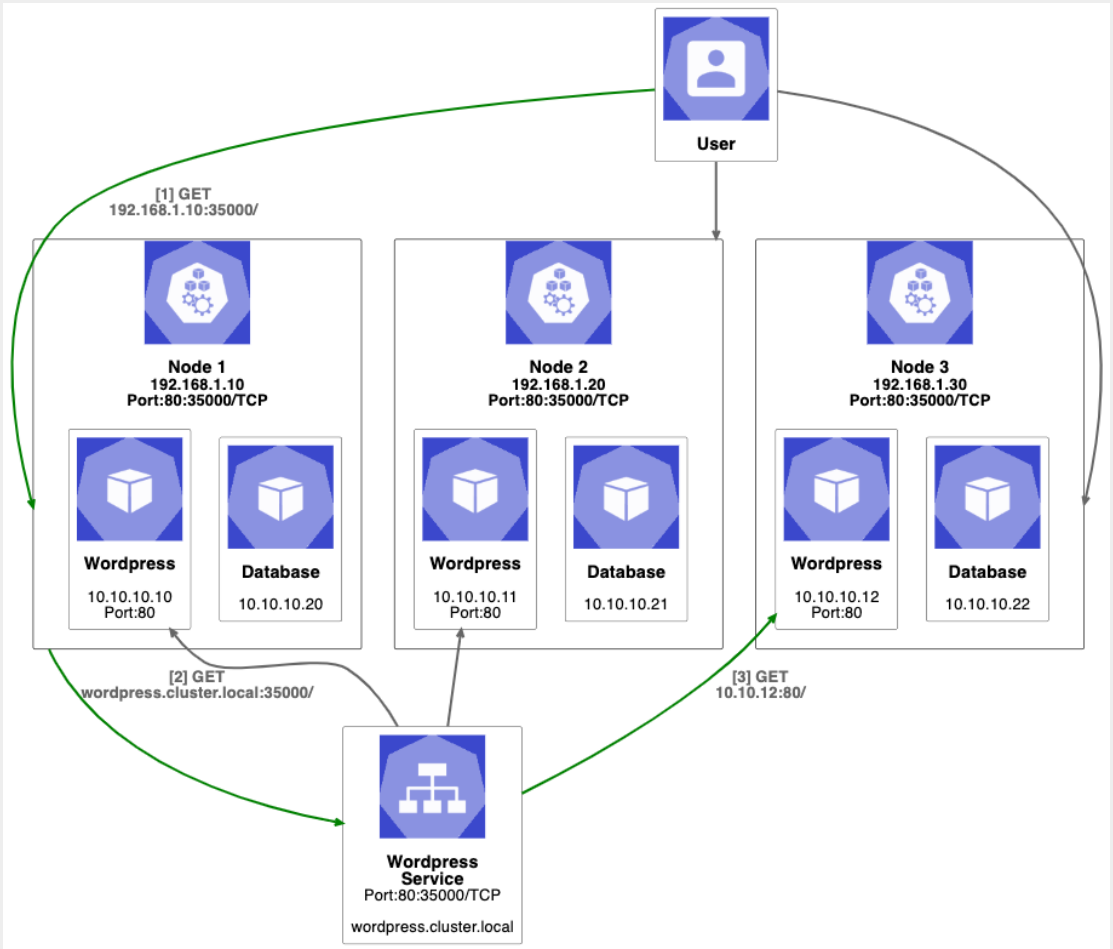
Service – LoadBalancer

Les Services LoadBalancer sont intégrés aux fournisseurs cloud pour gérer le trafic externe.



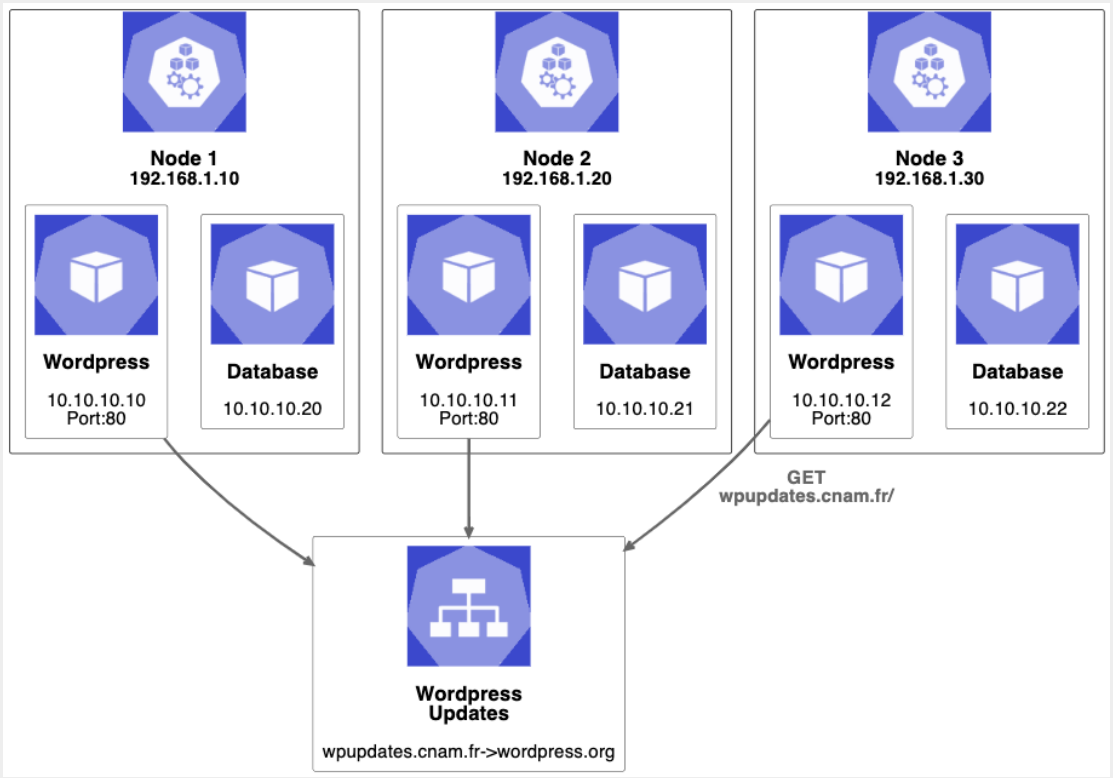
Service – NodePort

NodePort expose le service sur l'adresse IP de chaque nœud sur un port statique.



Service – ExternalName

ExternalName mappe le service vers un nom DNS externe.



Labels et selectors

Si les Pods sont éphémères, comment les grouper ?

Les ressources ont des labels pour les identifier. Les selectors sont utilisés pour grouper des ressources selon leurs labels.

- **Labels** : Paires clé-valeur attachées aux objets
- **Selectors** : Filtrent les objets selon leurs labels

Exemple

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
  labels:
    app: frontend
...
---
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
...
```

Ressource Network Policy

Network Policies

Restreignent les connexions depuis/vers les Pods, Services et Internet. Par défaut, tout le trafic est autorisé mais si une Network Policy est définie, seul le trafic autorisé est permis.

Sélection de la cible

- Labels
- Namespaces
- Blocs IP

Contenu des règles

- Target : Pods auxquels elle s'applique
- Type : Ingress ou Egress
- From / To : Sélecteur de pairs
- Protocol et port à autoriser

Namespaces

Certaines ressources dans Kubernetes sont divisées en Namespaces

Distinction importante

Namespaces Kubernetes $\neq$ Namespaces Linux

Namespaces Linux :

- Fonctionnalité du noyau
- Isolation au niveau processus
- Isolement PID, réseau, montages

Namespaces Kubernetes :

- Construction au niveau API
- Organisation des ressources
- Distinction logique des ressources

Namespaces Kubernetes : Exemple

Exemple

Une entreprise peut avoir des namespaces séparés pour :

- Le déploiement GitLab
- Le service VPN
- Les applications de monitoring

Les ressources namespaced incluent : Services, Deployments, ConfigMaps, NetworkPolicies

Comportement par défaut

Si vous ne spécifiez pas le namespace, les ressources sont créées dans `default`

Gestion de configuration

Les applications ont besoin de configuration et de secrets. Comment les injecter ?

Kubernetes fournit deux ressources

- **ConfigMaps** : Configuration non sensible (URLs, feature flags, fichiers de configuration)
- **Secrets** : Données sensibles (mots de passe, clés API, certificats, tokens)

Séparation des responsabilités

La configuration est gérée séparément des images de conteneurs, permettant à la même image de fonctionner dans différents environnements (dev, staging, prod).

ConfigMaps

Créer un ConfigMap à partir de valeurs littérales :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  database_url: "postgres://db-server:5432/mydb"
  log_level: "info"
  feature_flag: "true"
  app.properties: |
    server.port=8080
    max.connections=100
```

Utilisation dans un Pod en tant que variables d'environnement :

```
env:
  - name: DATABASE_URL
    valueFrom:
      configMapKeyRef:
        name: app-config
        key: database_url
```

Secrets

Créer un Secret :

```
apiVersion: v1
kind: Secret
metadata:
  name: db-credentials
type: Opaque
data:
  username: YWRtaW4=        # base64
  password: cGFzc3dvcmQ=    # base64
```

Utilisation dans un Pod :

```
env:
  - name: DB_USERNAME
    valueFrom:
      secretKeyRef:
        name: db-credentials
        key: username
  - name: DB_PASSWORD
    valueFrom:
      secretKeyRef:
        name: db-credentials
        key: password
```

ConfigMaps vs Secrets : Bonnes pratiques

ConfigMaps :

- Configuration en texte clair
- Peut être vue par quiconque ayant l'accès
- Peut être modifiée sans redémarrage (avec configuration appropriée)
- Versionnable sans risque

Secrets :

- Encodés en base64 (pas chiffrés !)
- Accès RBAC restreint
- Activer le chiffrement au repos
- Ne jamais les committer dans le contrôle de version
- Considérer les gestionnaires de secrets externes

Avertissement de sécurité

Les Secrets Kubernetes offrent une protection basique mais ne sont pas sécurisés par défaut.