Les Services Kubernetes

Une abstraction réseau essentielle pour exposer vos applications dans un cluster Kubernetes

Dans cette présentation :

- Concept de Service Kubernetes
- Problématique des Pods éphémères
- Service Discovery avec DNS
- Types de Services (ClusterIP, NodePort, LoadBalancer)
- Exemples pratiques d'utilisation
- Bonnes pratiques d'exposition d'applications

Introduction : Qu'est-ce qu'un Service Kubernetes ?

Définition

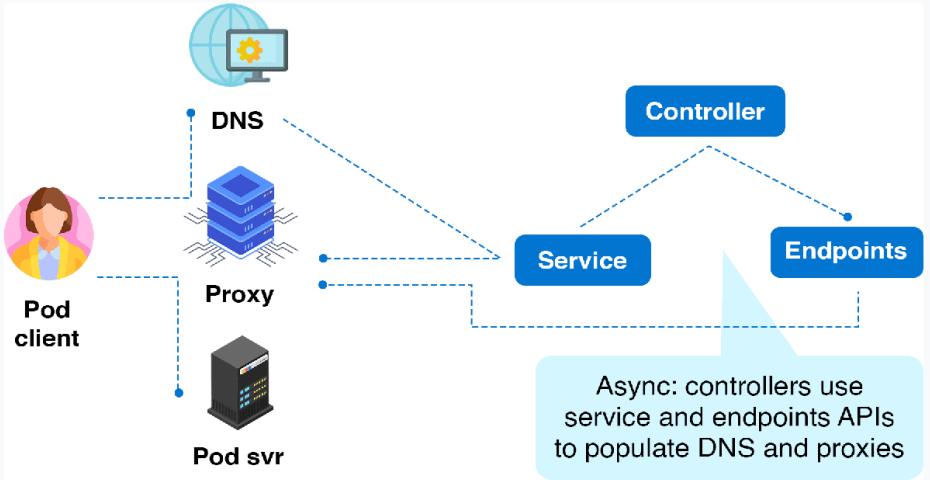
Un Service Kubernetes est une abstraction réseau qui définit un ensemble logique de Pods et une politique pour y accéder depuis l'intérieur ou l'extérieur du cluster.

Il permet d'exposer une application exécutée sur un ensemble de Pods, en fournissant un point d'accès unique et stable, indépendant des changements dans l'infrastructure sous-jacente.

Problématique résolue

Les Pods sont éphémères :

- Ils sont créés et détruits dynamiquement
- Chaque Pod obtient sa propre adresse IP
- Ces adresses changent constamment
- Comment maintenir des connexions fiables ?



Objectifs principaux

- Fournir une adresse IP stable et un nom DNS
- Équilibrer la charge entre plusieurs Pods
- Découvrir et communiquer avec d'autres services
- Exposer des applications à l'extérieur du cluster

Documentation officielle : kubernetes.io/docs/concepts/services-networking/service/

Le problème des Pods éphémères

La nature éphémère des Pods

Dans Kubernetes, les Pods sont conçus pour être des ressources éphémères qui peuvent être créés, détruits et remplacés à tout moment selon les besoins du cluster.

Cette nature transitoire est essentielle pour l'élasticité et la résilience de Kubernetes, mais crée un défi majeur pour les communications réseau.

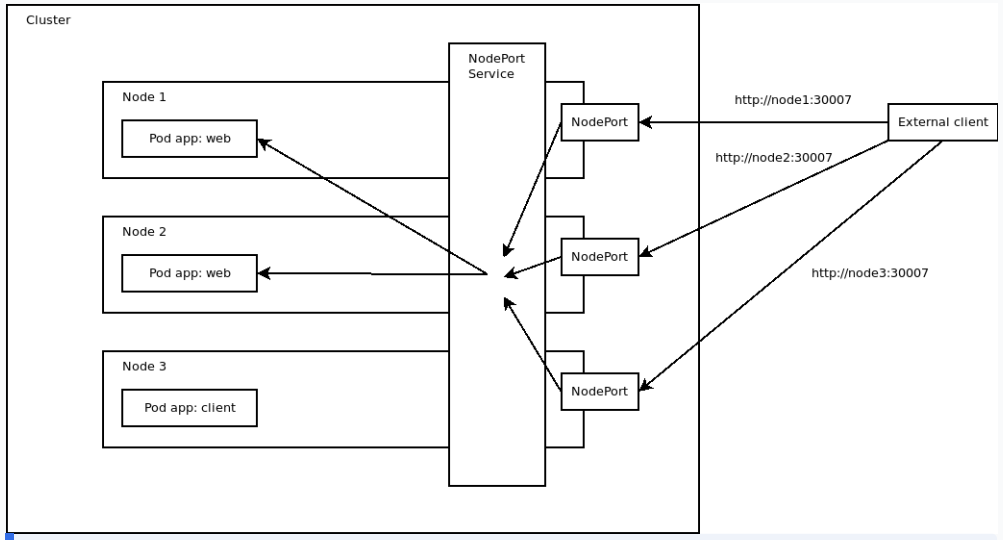
Problèmes de communication

- Les Pods obtiennent de nouvelles IP lorsqu'ils sont recréés
- Les redémarrages, mises à l'échelle et mises à jour changent les adresses
- Les clients ne peuvent pas suivre ces changements constants
- Impossible de maintenir des connexions fiables sans abstraction

Services : la solution

Les Services Kubernetes résolvent ce problème en :

- Fournissant un point de terminaison stable et permanent
- Abstrayant les changements d'IP des Pods sous-jacents
- Assurant une distribution du trafic entre les Pods disponibles



Pourquoi les Services sont essentiels

IP:10.0.0.23 → Pod A Pod détruit

my-service → Service (IP stable) → Pod disponible

mon-app.namespace → DNS interne → Service

- Équilibrage de charge automatique entre les Pods
- Découverte de service transparente pour les applications

Documentation officielle : kubernetes.io/docs/concepts/services-networking/service/

Service Discovery : DNS et Variables d'environnement

Mécanismes de découverte

Kubernetes offre deux mécanismes principaux pour permettre aux clients de découvrir et de communiquer avec les Services :

Variables d'environnement

Pour chaque Service actif, kubelet ajoute automatiquement des variables d'environnement dans les Pods :

```
{SERVICE_NAME}_SERVICE_HOST=10.0.0.11
{SERVICE_NAME}_SERVICE_PORT=80
{SERVICE_NAME}_PORT=tcp://10.0.0.11:80
```

Les noms de services sont convertis en majuscules avec tirets remplacés par underscores.

DNS

Kubernetes configure un serveur DNS (comme CoreDNS) pour le cluster qui surveille l'API et crée des enregistrements DNS pour chaque Service.

Format de nommage :

```
mon-service.mon-namespace.svc.cluster.local
```

Les Pods du même namespace peuvent utiliser juste mon-service

Le DNS est la méthode recommandée car plus flexible et plus facile à utiliser.

Avantages de la découverte de services

- Communication sans connaître les adresses IP des Pods
- Répartition automatique du trafic entre les Pods
- Adaptation dynamique aux changements d'infrastructure
- Indépendance entre le client et l'implémentation du service

Documentation : kubernetes.io/docs/concepts/services-networking/service/#discovering-services

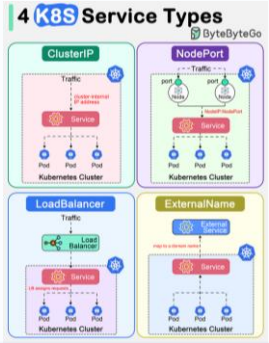
Les types de Services Kubernetes

Kubernetes propose différents types de Services pour répondre à divers besoins d'exposition d'applications :

ClusterIP

Par défaut - Service accessible uniquement à l'intérieur du cluster

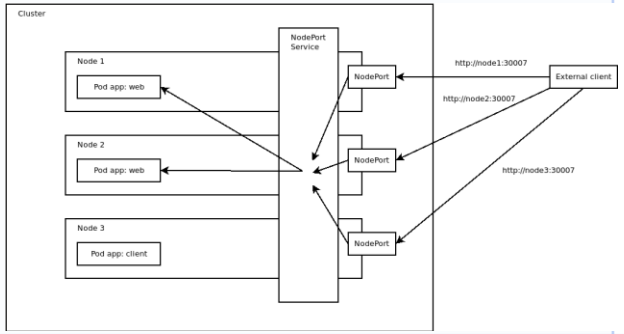
- IP virtuelle stable dans le cluster
- Communication Pod à Pod
- Équilibrage de charge interne
- Non accessible depuis l'extérieur



NodePort

Expose le Service sur un port statique sur chaque nœud du cluster

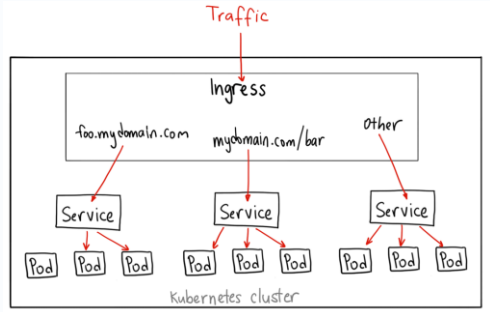
- Port fixe (30000-32767) sur chaque nœud
- Accessible depuis l'extérieur
- Inclut un ClusterIP automatiquement
- Pas de load balancing entre nœuds



LoadBalancer

Expose le Service via un load balancer externe fourni par le cloud

- IP publique externe fournie par le cloud
- Load balancing externe automatique
- Inclut ClusterIP et NodePort
- Idéal pour les applications de production



Comment choisir le type de Service approprié ?

ClusterIP: Applications internes au cluster sans besoin d'accès externe

NodePort: Environnements de développement ou tests nécessitant un accès externe simple

LoadBalancer: Applications de production nécessitant un accès externe hautement disponible

Documentation: kubernetes.io/docs/concepts/services-networking/service/

Exemple pratique : exposer une application Web

Déploiement d'une application

Commençons par déployer une application web simple avec 2 réplicas :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: nginx
        image: nginx:1.19
        ports:
        - containerPort: 80
```

Vérification des Services

```
$ kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)
web-internal ClusterIP 10.96.45.123 none 8080/TCP
web-external NodePort 10.96.78.234 none 80:30080/TCP
```

```
$ kubectl get endpoints
NAME ENDPOINTS
web-internal 10.244.1.5:80,10.244.2.7:80
web-external 10.244.1.5:80,10.244.2.7:80
```


kubectl expose deployment web-app --name=web-internal --port=8080 --target-port=80

Exemple pratique : exposer une application Web

Création des Services

Exposons l'application avec deux types de Services :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: nginx
        image: nginx:1.19
        ports:
        - containerPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: web-service
spec:
  selector:
    app: web
  ports:
  - port: 80
    targetPort: 80
  type: LoadBalancer
```

Test des connexions

Interne - Depuis un pod dans le cluster :

curl web-internal:8080

curl 10.96.45.123:8080

Externe - Depuis l'extérieur du cluster :

curl http://node-ip:30080

Points importants

Les Services créent des endpoints qui pointent vers les pods

Kubernetes DNS permet d'utiliser les noms de Services

ClusterIP accessible uniquement depuis le cluster

NodePort accessible depuis l'extérieur sur chaque nœud

kubectl expose deployment web-app --name=web-internal --port=8080 --target-port=80

Comparaison des types de Services

Un aperçu complet des différences clés entre les types de Services Kubernetes pour vous aider à choisir la meilleure option selon votre cas d'usage :

Caractéristique	ClusterIP	NodePort	LoadBalancer
Accessibilité	Interne au cluster uniquement	Externe via IP du nœud + port	Externe via IP dédiée
Adresse IP	IP virtuelle interne	IP du nœud sur port fixe (30000-32767)	IP externe fournie par le cloud
Exposition	Service à service dans le cluster	Exposé sur chaque nœud	Exposé via un load balancer externe
Cloud requis	✗ Non	✗ Non	✓ Oui
Load balancing	Entre Pods uniquement	Entre Pods uniquement	Entre Pods et entre nœuds
Cas d'usage idéal	Communication interne entre microservices	Tests, démos, environnements sans load balancer	Applications de production exposées publiquement
Recommandé pour	✓ Communication interne ✗ Applications externes	✓ Dev/Test ! Production	✓ Applications publiques ✓ Production

À retenir

ClusterIP est le type par défaut et sera automatiquement créé pour NodePort et LoadBalancer comme base.

Choisir le bon type

Sélectionnez en fonction de l'audience cible et de l'environnement d'exécution.

Conclusion et bonnes pratiques

Points clés à retenir

- Les Services résolvent le problème des Pods éphémères
- ClusterIP pour la communication interne au cluster
- Service Discovery avec DNS et variables d'environnement
- NodePort et LoadBalancer pour l'exposition externe

Bonnes pratiques pour l'utilisation des Services

Choisir le bon type de Service

Sélectionner le type approprié selon le contexte : ClusterIP pour l'interne, NodePort/LoadBalancer pour l'externe, ou utiliser Ingress pour des routes HTTP complexes.

Sécuriser les Services exposés

Pour les Services exposés en externe, toujours implémenter des mesures de sécurité appropriées comme TLS, authentification et limites d'accès réseau.

Bien définir les sélecteurs

Utiliser des labels cohérents et significatifs pour vos Pods afin que les Services puissent correctement identifier les endpoints cibles.

Utiliser la découverte DNS

Privilégier la découverte DNS intégrée plutôt que de coder en dur les adresses IP, pour une meilleure portabilité et résilience.

Nommer les ports

Toujours nommer les ports dans un Service pour faciliter la référence et améliorer la lisibilité, surtout pour les Services multi-ports.

Surveiller les performances

Mettre en place une surveillance des métriques de Service comme la latence et le trafic pour optimiser la performance du réseau.

En résumé : Les Services Kubernetes sont essentiels pour construire des applications résilientes et découplées. Avec les bonnes pratiques, ils offrent une couche d'abstraction réseau puissante et flexible.