kubernetes

k8s

Concepts de Kubernetes

- Kubernetes est un système de gestion de conteneurs
- Il exécute et gère les applications conteneurisées sur un cluster
- Qu'est-ce que cela signifie vraiment?

Choses de base que nous pouvons demander à Kubernetes de faire

- Démarrer 5 conteneurs en utilisant l'image eshop/api:v1.3
- Placer un équilibreur de charge interne devant ces conteneurs
- Démarrer 10 conteneurs en utilisant l'image eshop/webfront:v1.3
- Placer un équilibreur de charge public devant ces conteneurs
- C'est période de soldes, les pics de trafic, développer notre cluster et ajouter des conteneurs
- Nouvelle version! Remplacer mes conteneurs par la nouvelle image eshop/webfront:v1.4
- Continuer à traiter les demandes pendant la mise à niveau; mettre à jour mes conteneurs un par un

Autres choses que Kubernetes peut faire

- Mise à l'échelle automatique de base (autoscaling)
- Déploiement bleu/vert, déploiement Canary
- Services de longue durée, mais aussi travaux par lots (ponctuels)
- Surengager notre cluster et éjecter les tâches à faible priorité
- Exécuter des services avec des données avec état (bases de données, etc.)
- Contrôle d'accès précis définissant ce qui peut être fait par qui sur quelles ressources
- Intégration de services tiers (catalogue de services)
- Automatiser des tâches complexes (opérateurs)

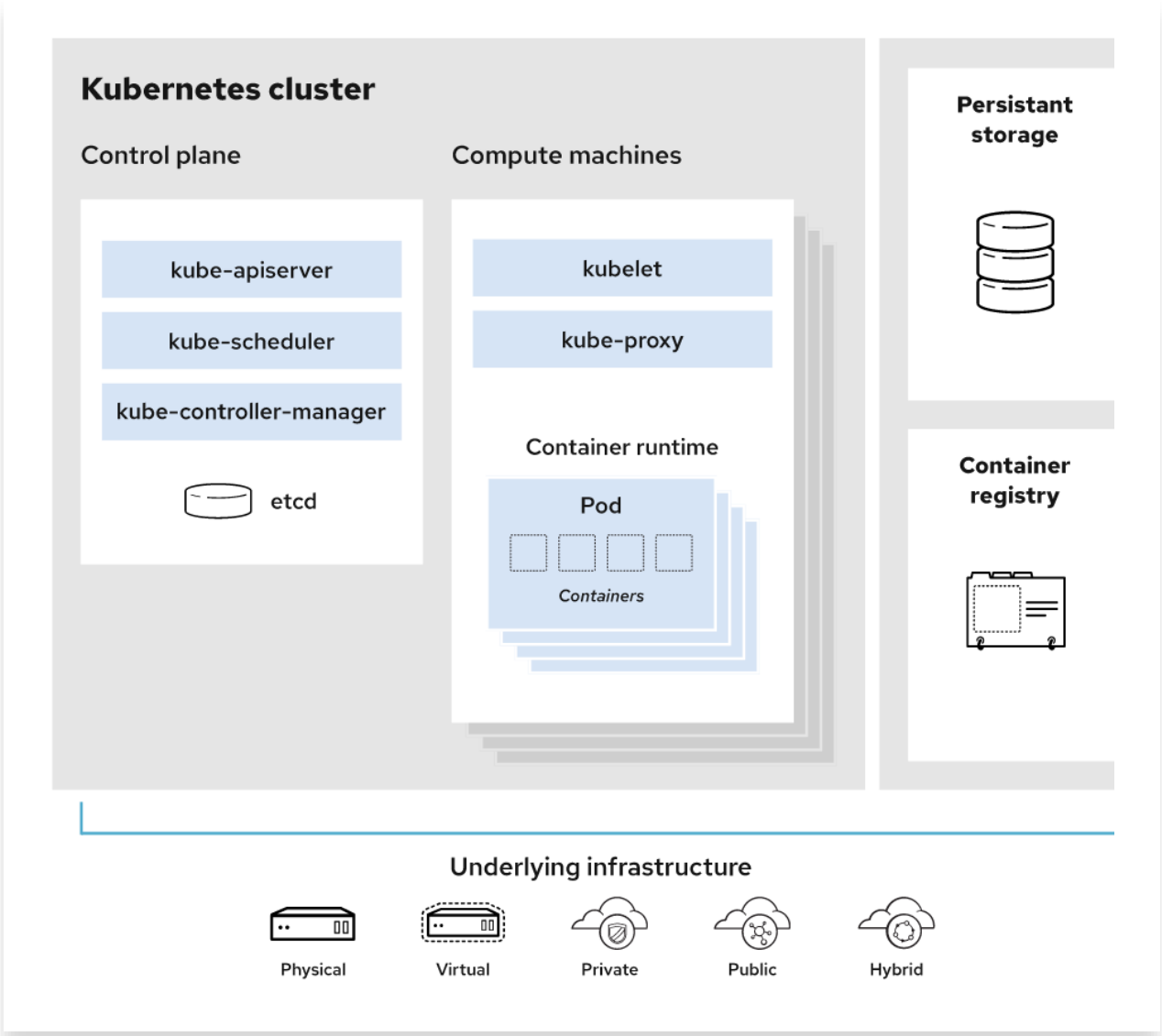
Fonctionnalités

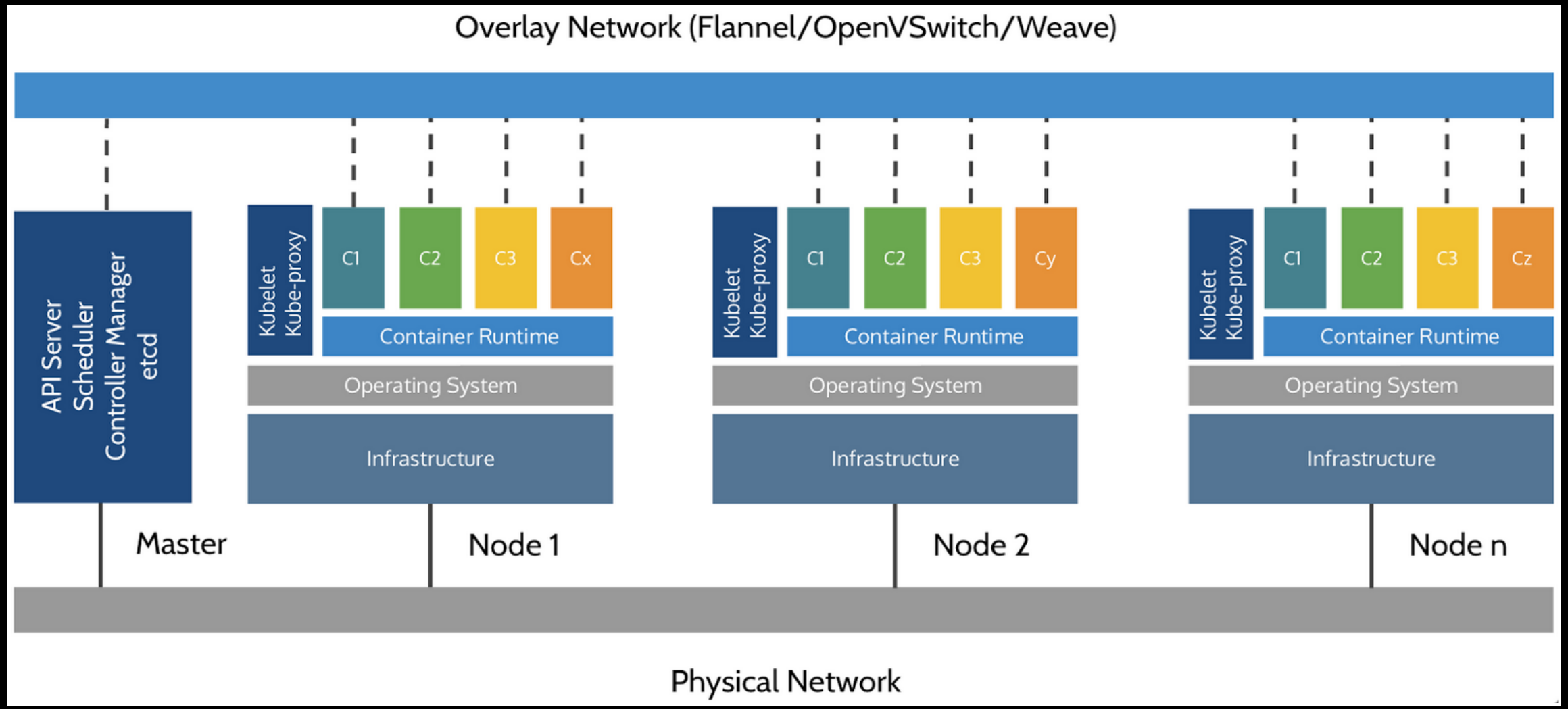
- **Découverte de service et équilibrage de charge** Kubernetes peut exposer un conteneur en utilisant le nom DNS ou en utilisant sa propre adresse IP. Si le trafic vers un conteneur est élevé, Kubernetes est capable d'équilibrer la charge et de distribuer le trafic réseau afin que le déploiement soit stable.
- **Orchestration du stockage** Kubernetes vous permet de monter automatiquement un système de stockage de votre choix, tel que des stockages locaux, des fournisseurs de cloud public, etc.
- **Déploiements et annulations automatisés** Vous pouvez décrire l'état souhaité pour vos conteneurs déployés à l'aide de Kubernetes, et il peut changer l'état réel à l'état souhaité à une vitesse contrôlée. Par exemple, vous pouvez automatiser Kubernetes pour créer de nouveaux conteneurs pour votre déploiement, supprimer des conteneurs existants et adopter toutes leurs ressources dans le nouveau conteneur.

Fonctionnalités

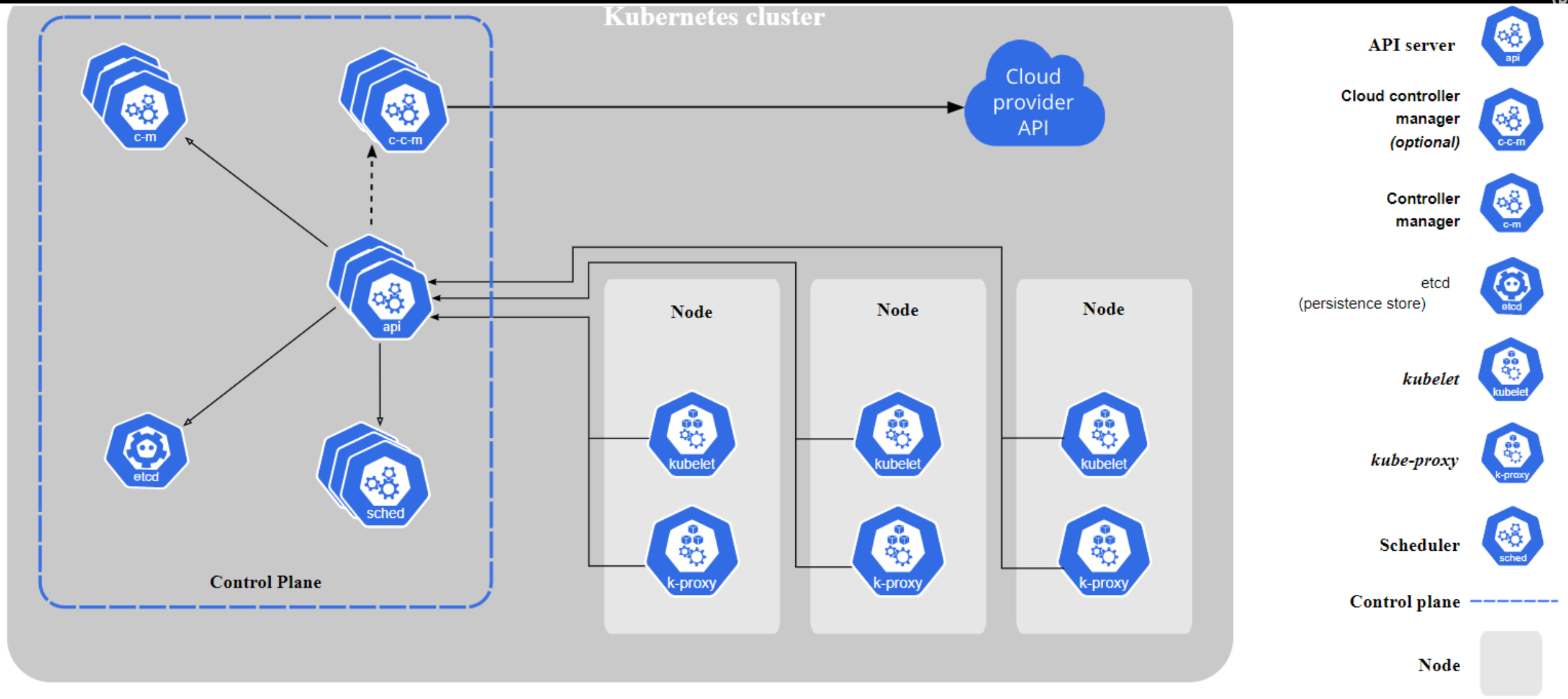
- **Empaquetage** automatique de binaires Vous fournissez à Kubernetes un cluster de nœuds qu'il peut utiliser pour exécuter des tâches en conteneur. Vous indiquez à Kubernetes la quantité de CPU et de mémoire (RAM) dont chaque conteneur a besoin. Kubernetes peut installer des conteneurs sur vos nœuds pour tirer le meilleur parti de vos ressources.
- **L'auto- réparation** Kubernetes redémarre les conteneurs qui échouent, remplace les conteneurs, tue les conteneurs qui ne répondent pas à votre vérification de l'état définie par l'utilisateur et ne les annonce pas aux clients tant qu'ils ne sont pas prêts à être diffusés.
- **Gestion des secrets et de la configuration** Kubernetes vous permet de stocker et de gérer des informations sensibles, telles que les mots de passe, les jetons OAuth et les clés SSH. Vous pouvez déployer et mettre à jour les secrets et la configuration de l'application sans reconstruire vos images de conteneur et sans exposer les secrets dans la configuration de votre pile.

Kubernetes Cluster





Architecture de Kubernetes



Composants de Kubernetes

Composants du plan de contrôle

- Les composants du plan de contrôle prennent des décisions globales concernant le cluster (par exemple, la planification), ainsi que la détection et la réponse aux événements du cluster (par exemple, le démarrage d'un nouveau pod lorsque le champ *replicas* d'un déploiement n'est pas satisfait).
- Les composants du plan de contrôle peuvent être exécutés sur n'importe quelle machine du cluster.
- Pour plus de simplicité, les scripts de configuration démarrent généralement tous les composants du plan de contrôle sur la même machine et n'exécutent pas de conteneurs utilisateur sur cette machine.
- **Le plan de contrôle est également appelé le "master"**

kube-apiserver

- Le serveur API est un composant du plan de contrôle de Kubernetes qui expose l'API Kubernetes.
- Le serveur API est le frontal du plan de contrôle Kubernetes.
- La principale implémentation d'un serveur d'API Kubernetes est kube-apiserver .
- kube-apiserver est conçu pour évoluer horizontalement, c'est-à-dire qu'il évolue en déployant plus d'instances.
- Vous pouvez exécuter plusieurs instances de kube-apiserver et équilibrer le trafic entre ces instances.

etcd

Magasin de valeurs de clé cohérent et hautement disponible utilisé comme magasin de sauvegarde de Kubernetes pour toutes les données du cluster.

Si votre cluster Kubernetes utilise etcd comme magasin de sauvegarde, assurez-vous d'avoir un plan de sauvegarde pour ces données.

kube-scheduler

- Composant du plan de contrôle qui surveille les nouveaux Pods sans nœud assigné et sélectionne un nœud sur lequel les exécuter.
- Les facteurs pris en compte pour les décisions de planification comprennent: les besoins en ressources individuelles et collectives, les contraintes matérielles / logicielles / politiques, les spécifications d'affinité et d'anti-affinité, la localisation des données, l'interférence entre les charges de travail et les délais.

kube-controller-manager

- Composants du Plan de Controle qui exécutent les processus de controle.
- Logiquement, chacun controleur est un processus distinct, mais pour réduire la complexité, ils sont tous compilés en un seul binaire et exécutés dans un seul processus.
- Ces contrôleurs comprennent:
  - Contrôleur de nœud: responsable de remarquer et de répondre lorsque les nœuds tombent en panne.
  - Contrôleur de réplication: responsable de la gestion du nombre correct de pods pour chaque objet contrôleur de réplication dans le système.
  - Contrôleur Endpoints: remplit l'objet Endpoints (c'est-à-dire joint les services et les pods).
  - Contrôleurs de compte de service et de jetons: créez des comptes par défaut et des jetons d'accès API pour les nouveaux espaces de noms.

cloud-controller-manager

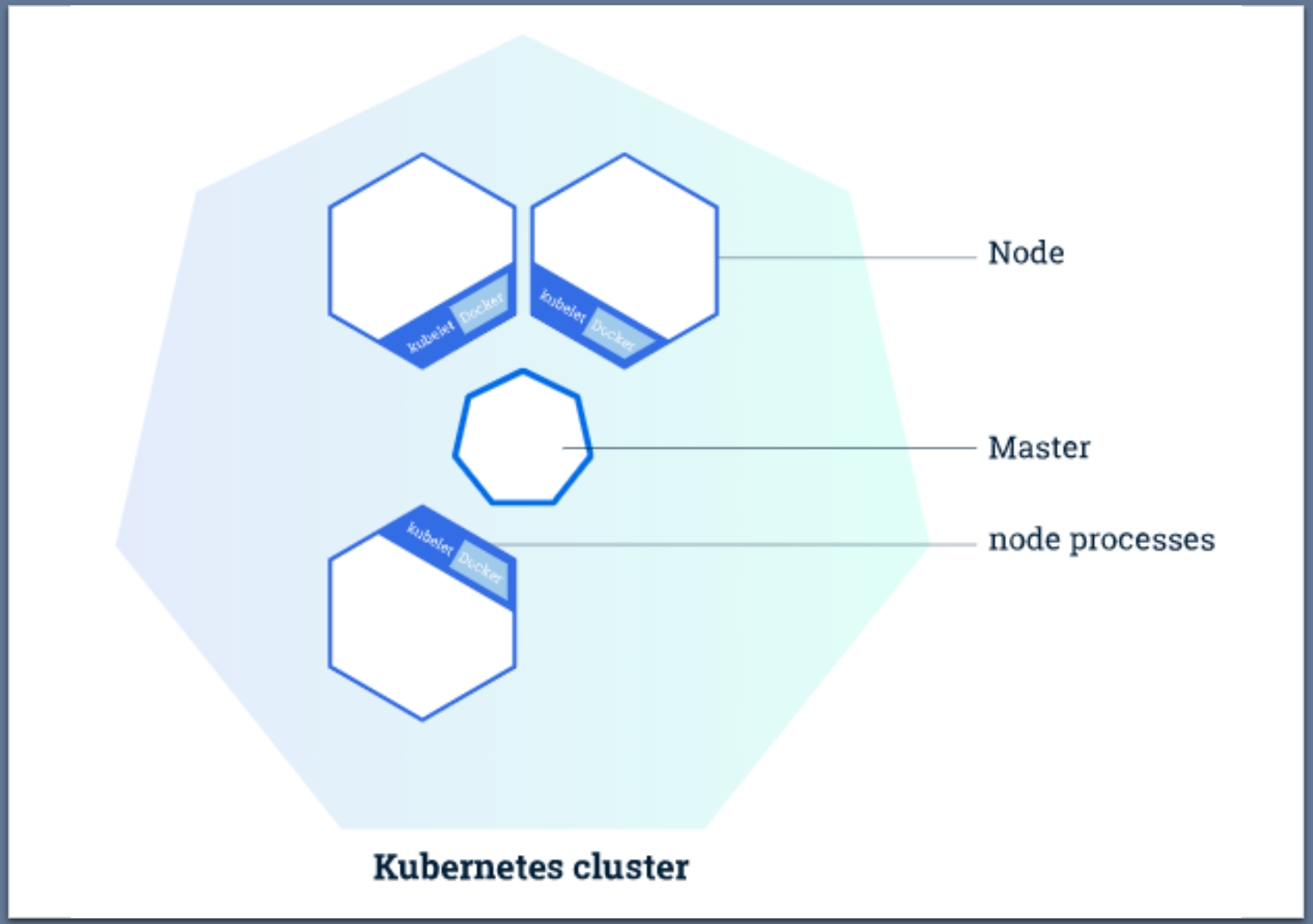
- Un plan de contrôle Kubernetes qui intègre une logique de contrôle spécifique au cloud.
- Permet de lier votre cluster à l'API de votre fournisseur cloud et sépare les composants qui interagissent avec cette plate-forme cloud des composants qui interagissent simplement avec votre cluster.

Les nœuds

- Les nœuds exécutant nos conteneurs exécutent une collection de services:
  - un moteur de conteneur (généralement Docker)
  - kubelet (l’agent du nœud)
  - kube-proxy (un composant réseau)
- Les nœuds étaient anciennement appelés "minions"

Composants du nœud

- **kubelet**
  - Un agent qui s'exécute sur chacun nœud dans le cluster. Il s'assure que les conteneurs s’exécutent dans un Pod.
  - Prend un ensemble de PodSpecs et garantit que les conteneurs décrits dans ces PodSpecs sont en cours d'exécution et sains.
- **kube-proxy**
  - Proxy réseau qui s'exécute sur chaque nœud dans votre cluster.
  - Maintient les règles de réseau sur les nœuds, permettent la communication réseau avec vos pods à partir de sessions réseau à l'intérieur ou à l'extérieur de votre cluster.
- **Container runtime**



Cluster Kubernetes

L’API et les Objets Kubernetes

Interagir avec Kubernetes

- L'API Kubernetes est RESTful
- Elle permet de CRUD des ressources ou objets
- Types courants de ressources :
  - **node** (Bare-metal ou VM - dans le cluster)
  - **pod** (groupe de conteneurs fonctionnant ensemble sur un nœud)
  - **service** (point de terminaison réseau stable pour se connecter à un ou plusieurs conteneurs)
- Déclarer les objets pour décrire l'état souhaité d'un cluster :
  - quelles applications ou autres processus exécuter,
  - quelles images utiliser,
  - nombre de replicas,
  - les ressources réseau et disque à disposer, etc

Interaction avec K8s

- On définit des objets via l'interface en ligne de commande **kubectl** de deux façons :
  - en lançant une commande *kubectl run <conteneur> ..., kubectl expose ...*
  - en décrivant un objet dans un fichier *YAML* ou *JSON* et en le passant au client

    kubectl apply -f monpod.yml
- Vous pouvez utiliser directement l'API Kubernetes pour interagir avec le cluster et définir ou modifier l'état souhaité. Kubernetes est complètement **automatisable** !

Syntaxe de base d’un manifest Kubernetes

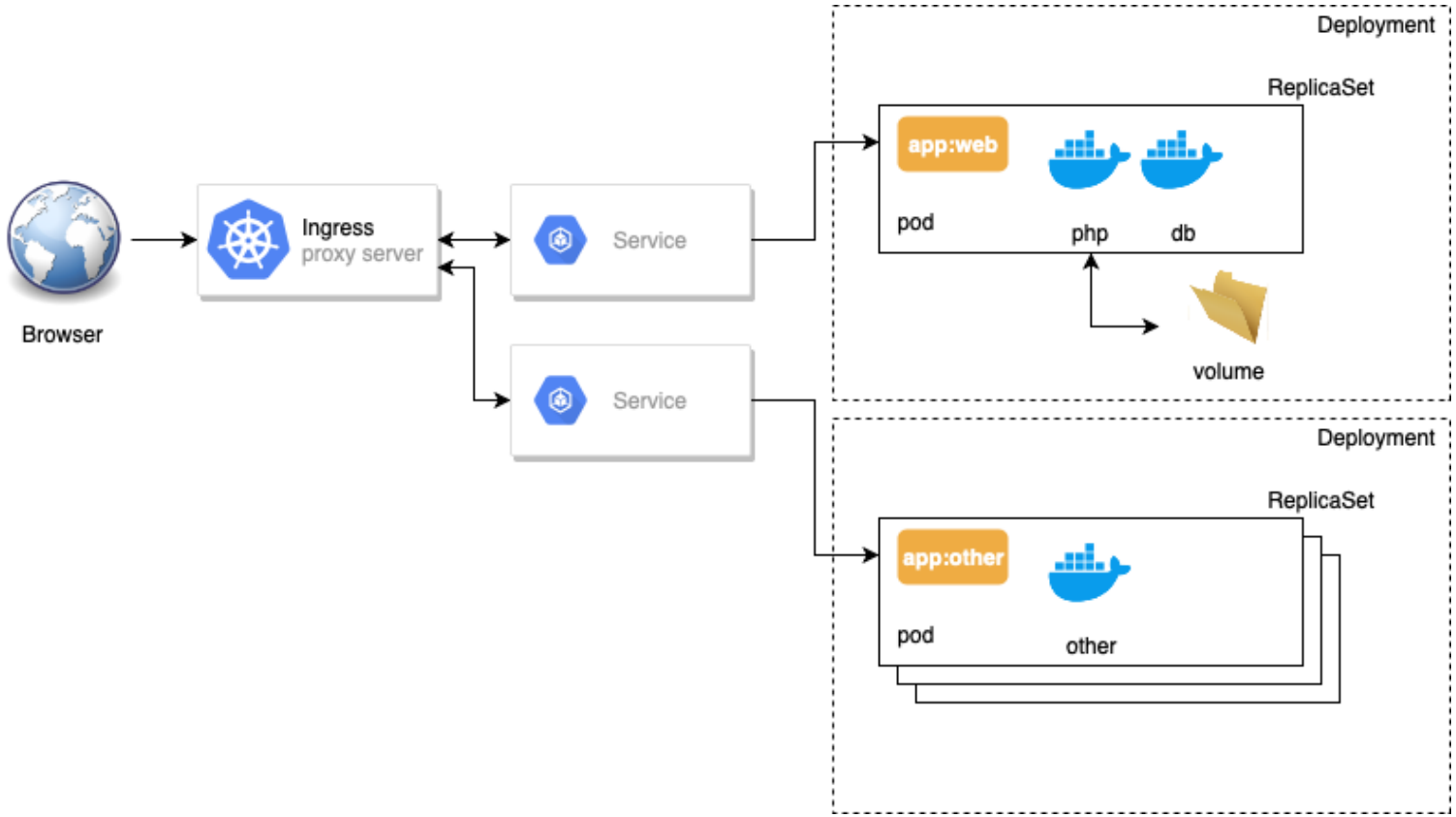
- **apiVersion :** Cette clé indique la version de l'API Kubernetes utilisée pour créer l'objet. Chaque ressource Kubernetes a une apiVersion spécifique, comme v1 pour un Pod ou apps/v1 pour un Déploiement.
- **kind :** Cette clé définit le type de l'objet que vous souhaitez créer. Exemples : Pod, Service, Deployment.
- **metadata :** Cette section contient des données qui aident à identifier sans équivoque l'objet, telles que name, labels et namespace.
- **spec :** Le cœur du manifest. Cette section varie considérablement selon le type d'objet et contient les spécifications détaillées de l'objet, comme les conteneurs à exécuter dans un Pod ou les réplicas dans un Déploiement.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mon-premier-pod
spec:
  containers:
  - name: conteneur-simple
    image: nginx
```

OBJETS DE BASE

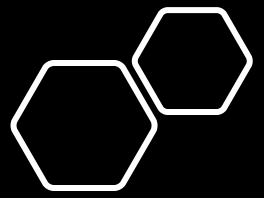
L’architecture découplée des services Kubernetes

- L’ensemble de la gestion d’un service applicatif se décompose dans Kubernetes en 3 à 4 objets articulés entre eux:
- replicatset
- deployment
- service
- (ingress)



Les namespaces

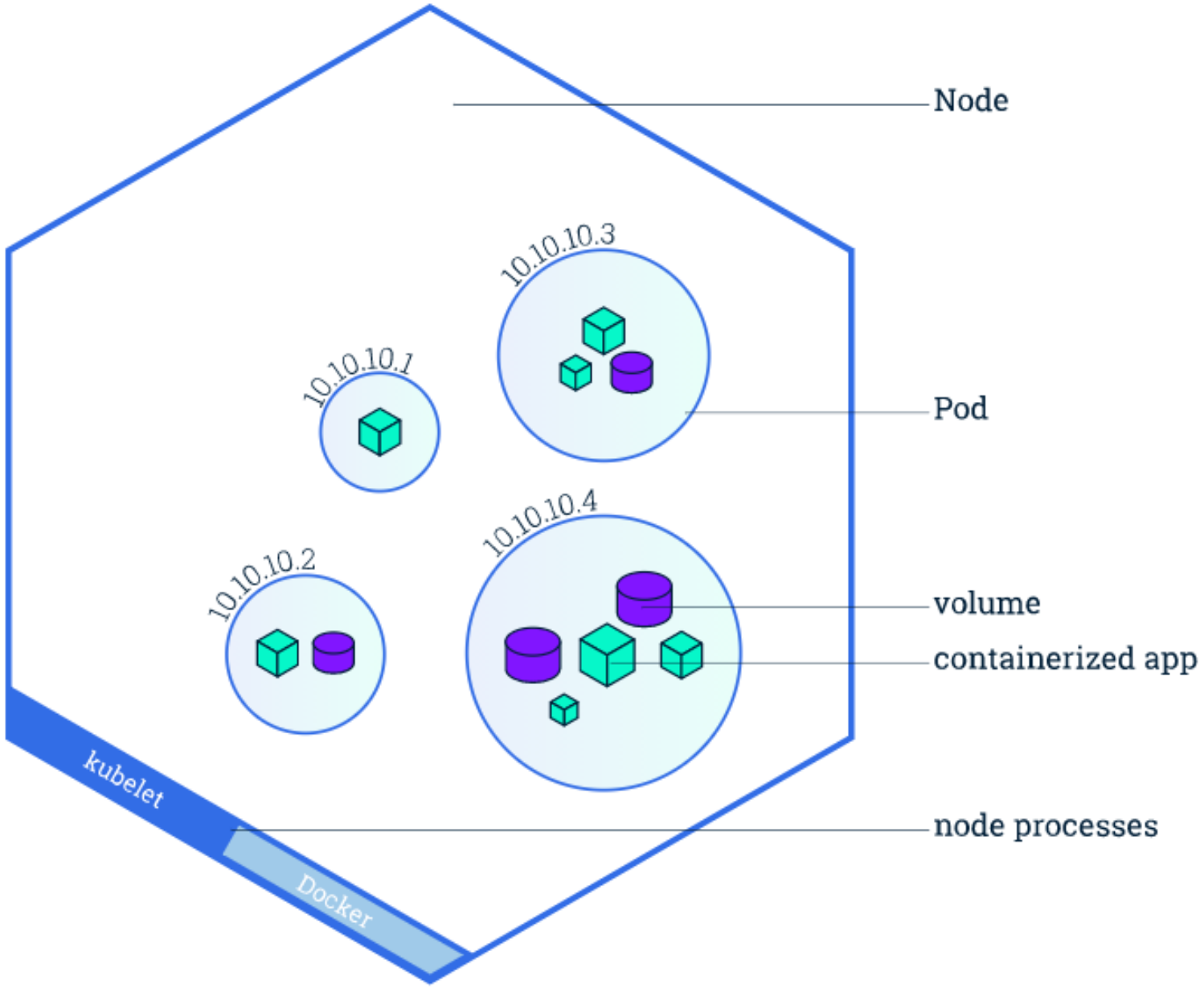
- Tous les objets Kubernetes sont rangés dans différents espaces de travail isolés appelés namespaces.
- Cette isolation permet 3 choses :
  - ne voir que ce qui concerne une tâche particulière lorsqu'on opère sur un cluster
  - créer des limites de ressources (CPU, RAM, etc.) pour le namespace
  - définir des rôles et permissions sur le namespace qui s'appliquent à toutes les ressources à l'intérieur.



Les Pods

- Un Pod représente une unité de déploiement : un petit nombre de conteneurs qui sont étroitement liés et qui partagent :
  - les mêmes ressources de calcul
  - des volumes communs
  - la même IP donc le même nom de domaine
  - peuvent se parler sur localhost
  - peuvent se parler en IPC
  - ont un nom différent et des logs différents

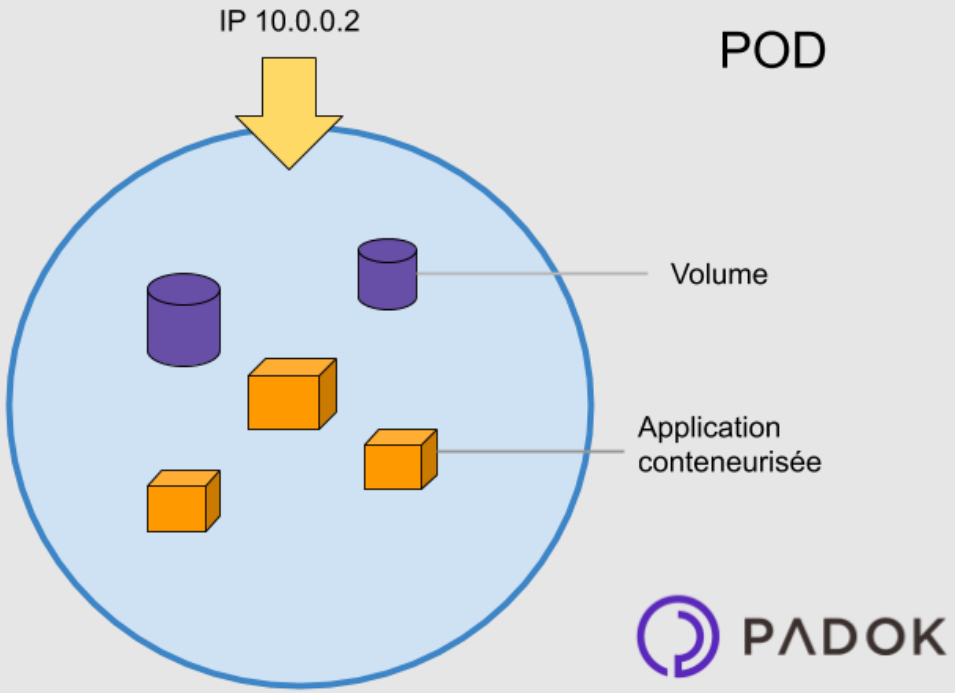
Pods

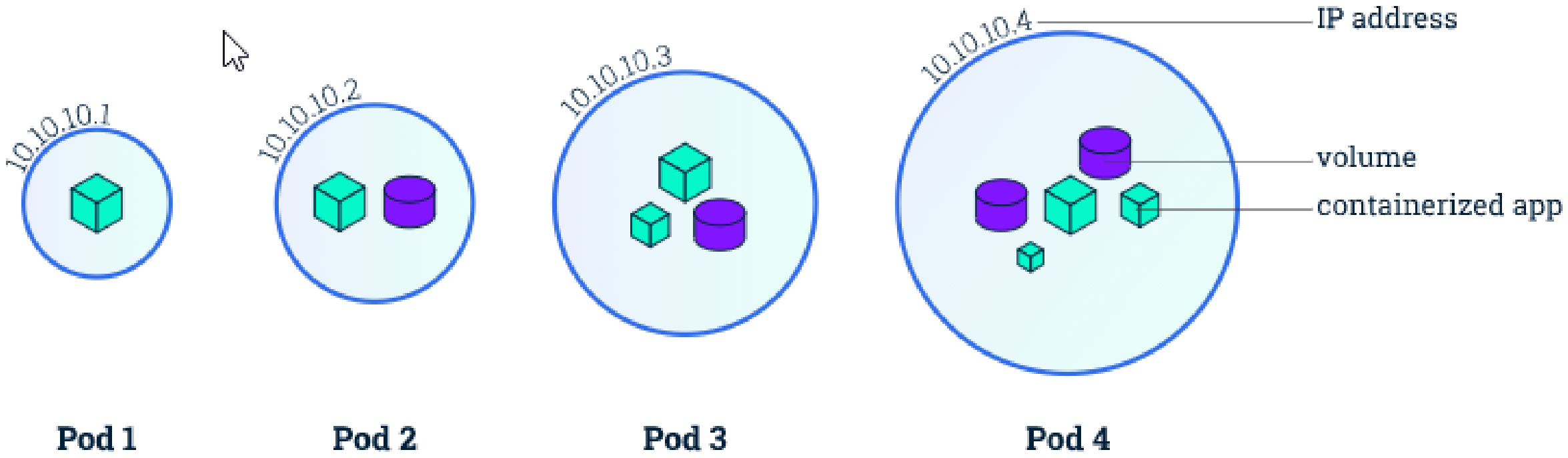


Elies Jebri : elies.jebri@gmail.com

Pod

- Un pod est l'objet de base de Kubernetes. Il englobe les conteneurs, les ressources de stockage et les IP réseau.
- Un pod représente un élément d'une application dans Kubernetes. Ces pods sont lancés dans un cluster Kubernetes qui est composé de noeuds.
- Les pods sont destinés à être éphémères. Kubernetes peut adapter le nombre de ces pods pour s'adapter au trafic entrant, créant ou supprimant ainsi des pods à la demande.
- Kubernetes gère le trafic vers ce nombre fluctuant de pods en utilisant des **services**.

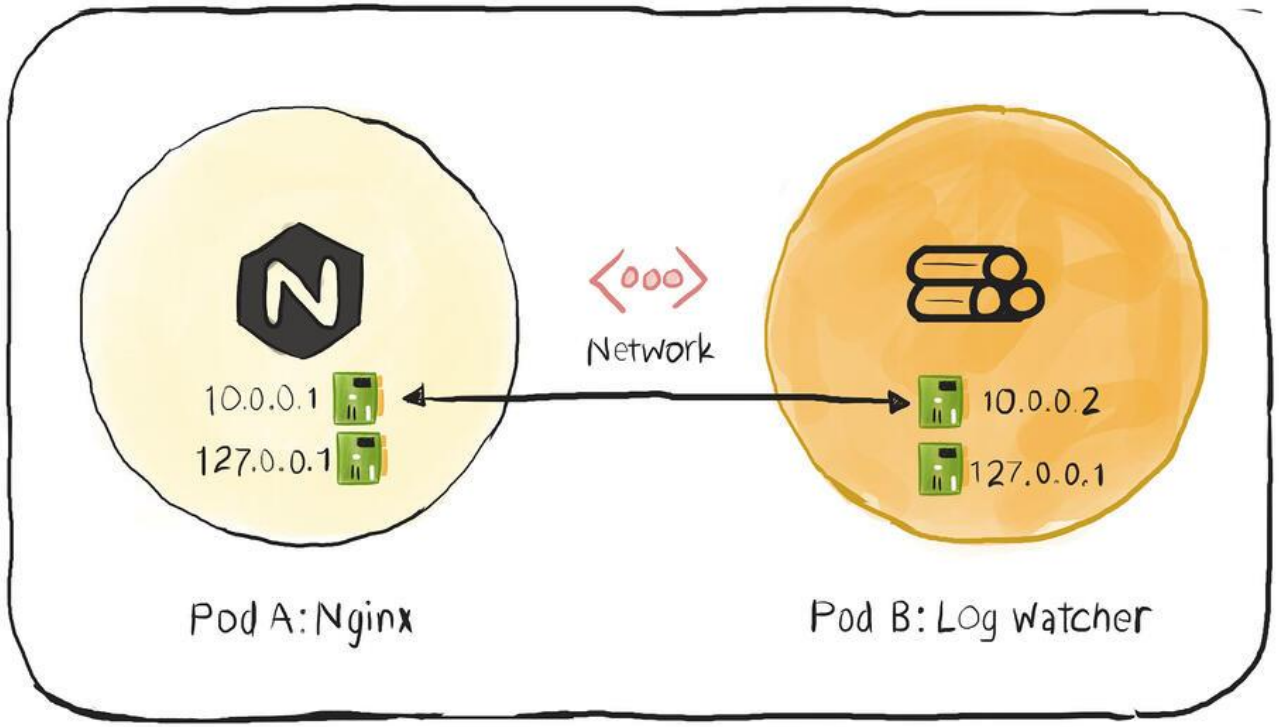



10.10.10.1
10.10.10.2
10.10.10.3
10.10.10.4
IP address
volume
containerized app
Pod 1
Pod 2
Pod 3
Pod 4

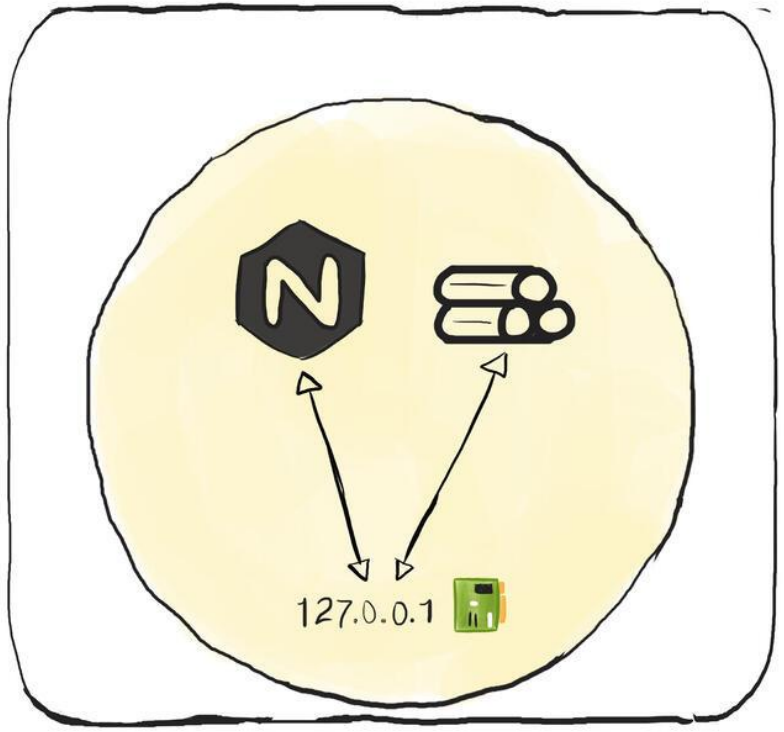
PODs

Containers vs Pods

- Containers



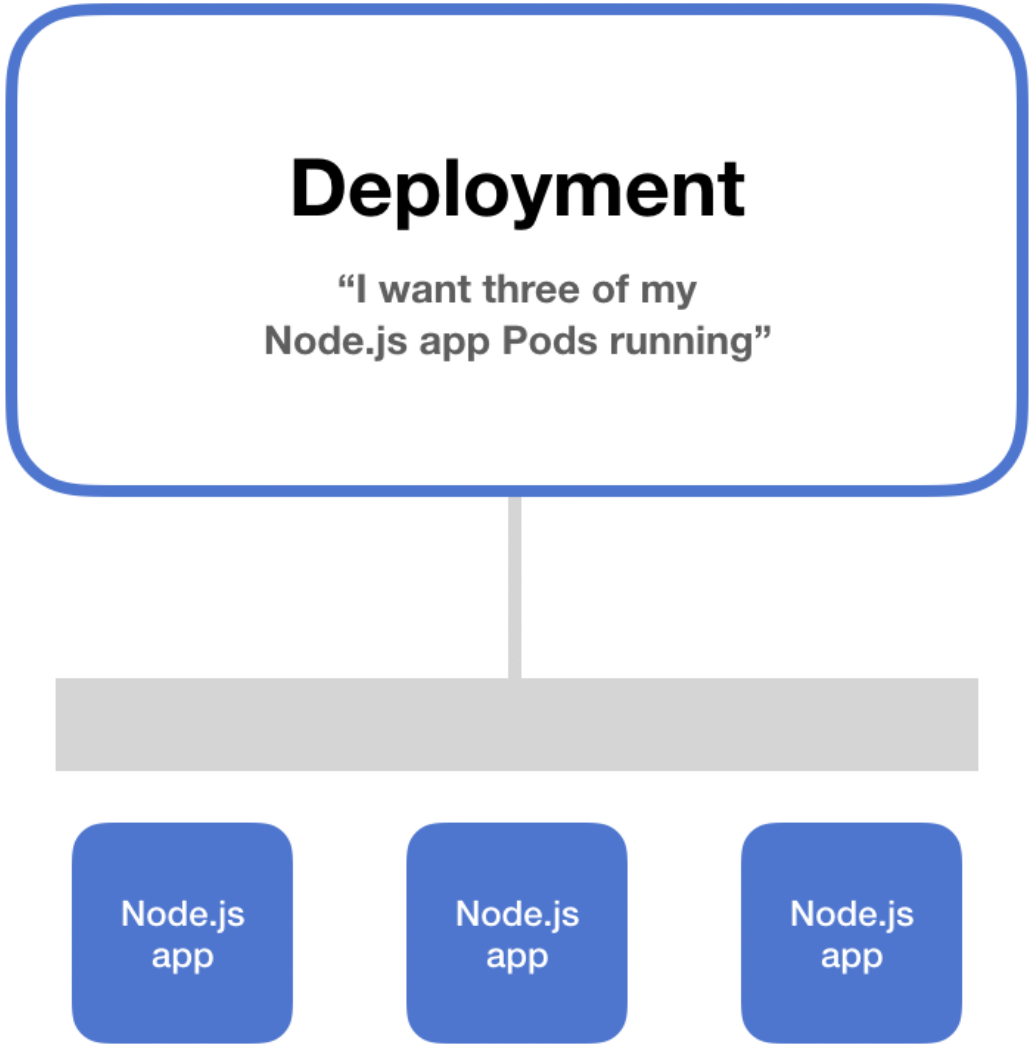
- Pod



Un manifeste de Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: rancher-demo-pod
spec:
  containers:
    - image: monachus/rancher-demo:latest
      name: rancher-demo-container
      ports:
        - containerPort: 8080
          name: http
          protocol: TCP
    - image: redis
      name: redis-container
      ports:
        - containerPort: 6379
          name: http
          protocol: TCP
```

Déploiements

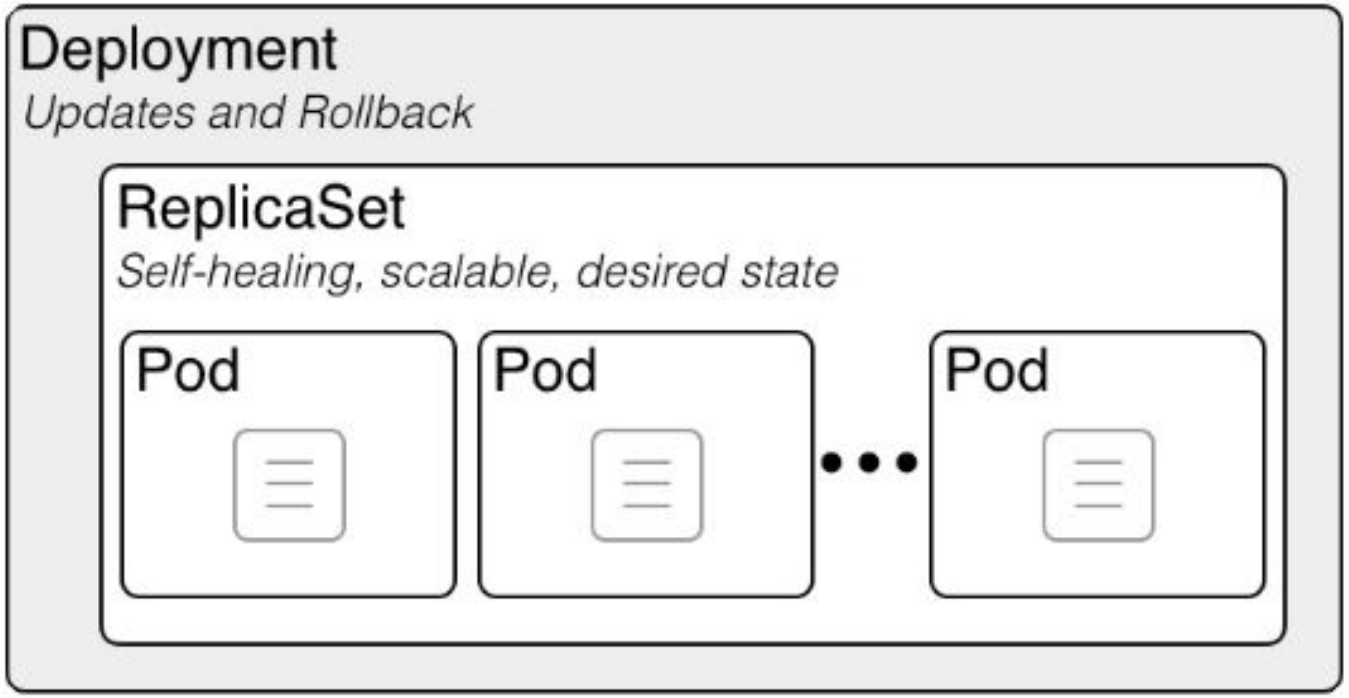


Déploiements

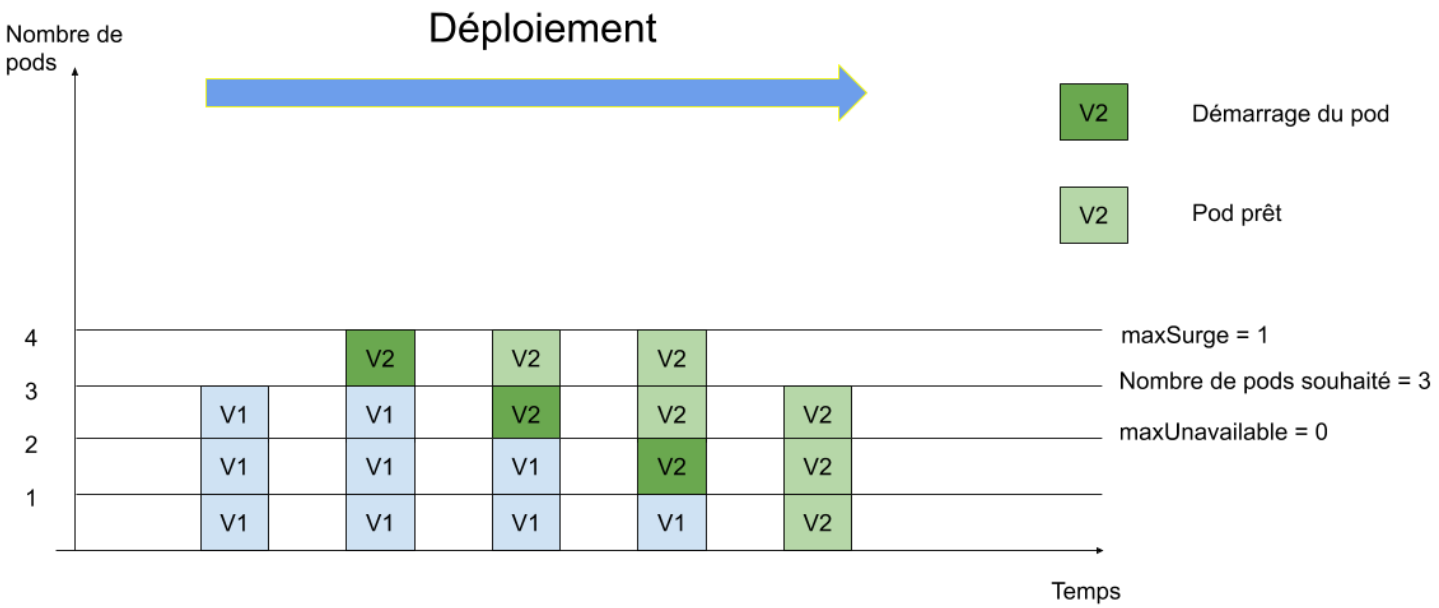
- Objets Kubernetes qui gèrent les pods.
- Permettent de déployer une version spécifique de votre application et de spécifier le nombre de pods dont vous avez besoin pour qu'elle soit opérationnelle.
- Lorsqu'une nouvelle version est prête à être mise en production, le déploiement peut facilement gérer cette mise à jour sans downtime en appliquant deux règles de base :
  - **maxSurge** spécifie le nombre maximum de pods qui peuvent être créés au-delà du nombre de pods désiré.
  - **Max Unavailable** spécifie le nombre maximum de pods qui peuvent être indisponibles pendant le déploiement.

Les Deployments (deploy)

- Ce sont des objets de plus haut niveau que les pods et replicaset.
- Un deployment implique la création d'un ensemble de Pods désignés par une étiquette **label** et regroupés dans un **Replicaset**.



Déploiement

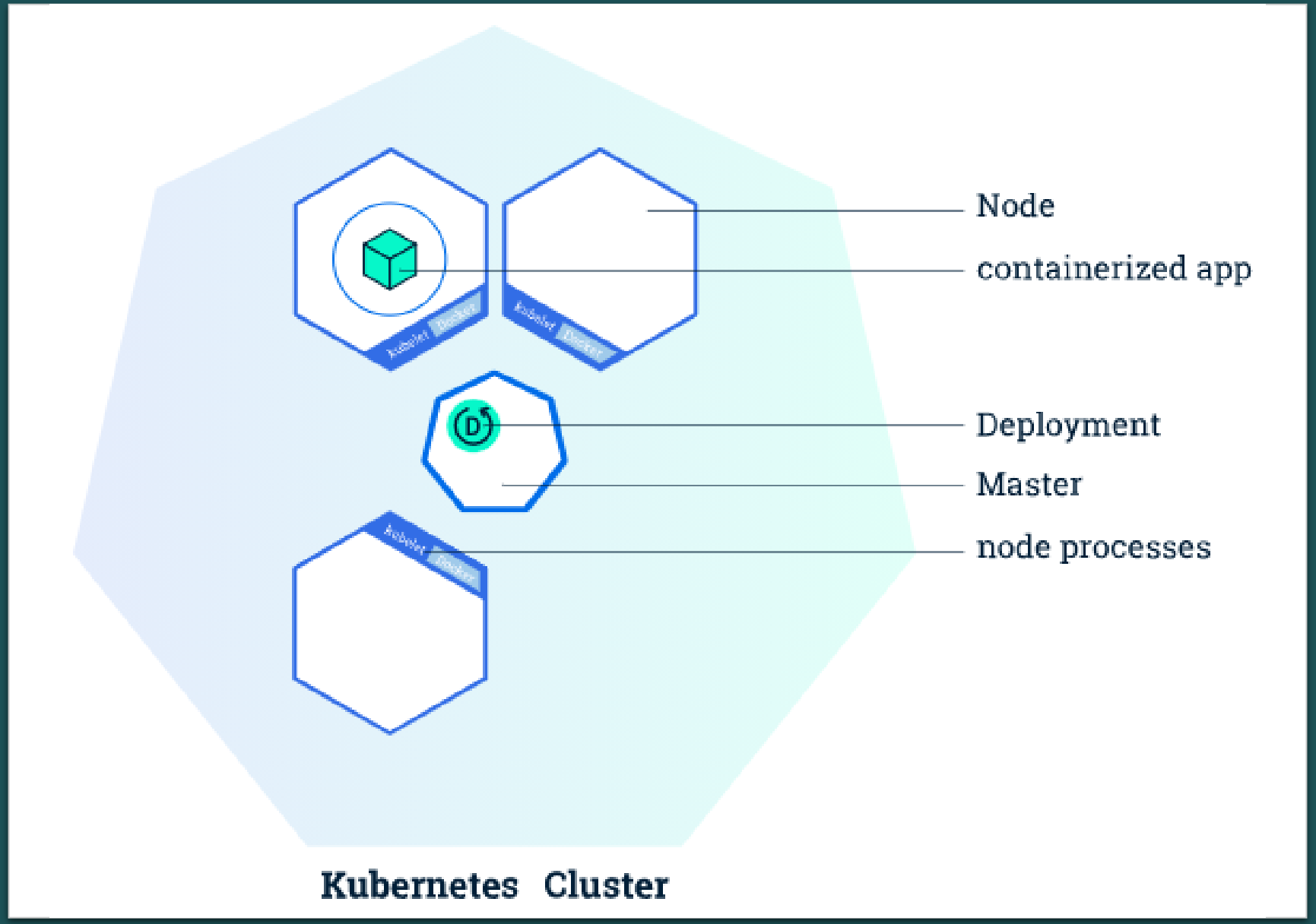


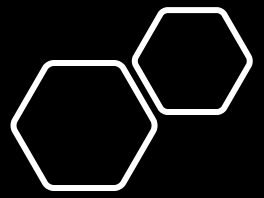
- Ces pods sont orchestrés sur différents nœuds du cluster et peuvent être déplacés de l'un à l'autre.
- Avec une telle facilité de déploiement et de mise à niveau des applications, vous ne devez tout même jamais oublier de tester votre application dans un environnement de staging.
- Vous voulez également que tous vos environnements soient indépendants, Kubernetes utilise des configmaps et des secrets pour cela.

Exemple :

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

Déploiement d'une application sur Kubernetes





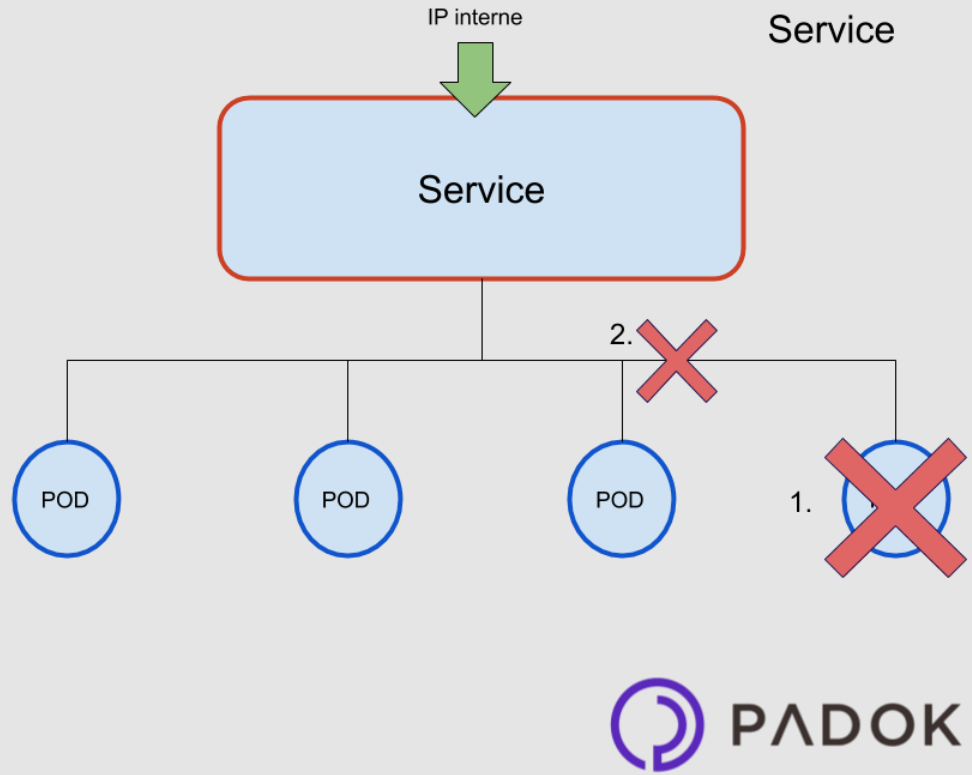
Les ReplicaSets (rs)

- Les ReplicaSet servent à gérer et sont responsables pour:
  - la réplication (avoir le bon nombre d'instances et le scaling)
  - la santé et le redémarrage automatique des pods de l'application (Self-Healing)
  - kubectl get rs pour afficher la liste des replicas.

Les Services

- Désignent un ensemble de pods (grâce à des tags/labels) généralement géré par un déploiement.
- Fournissent un endpoint réseau pour les requêtes à destination de ces pods.
- Configurent une politique permettant d'y accéder depuis l'intérieur ou l'extérieur du cluster.

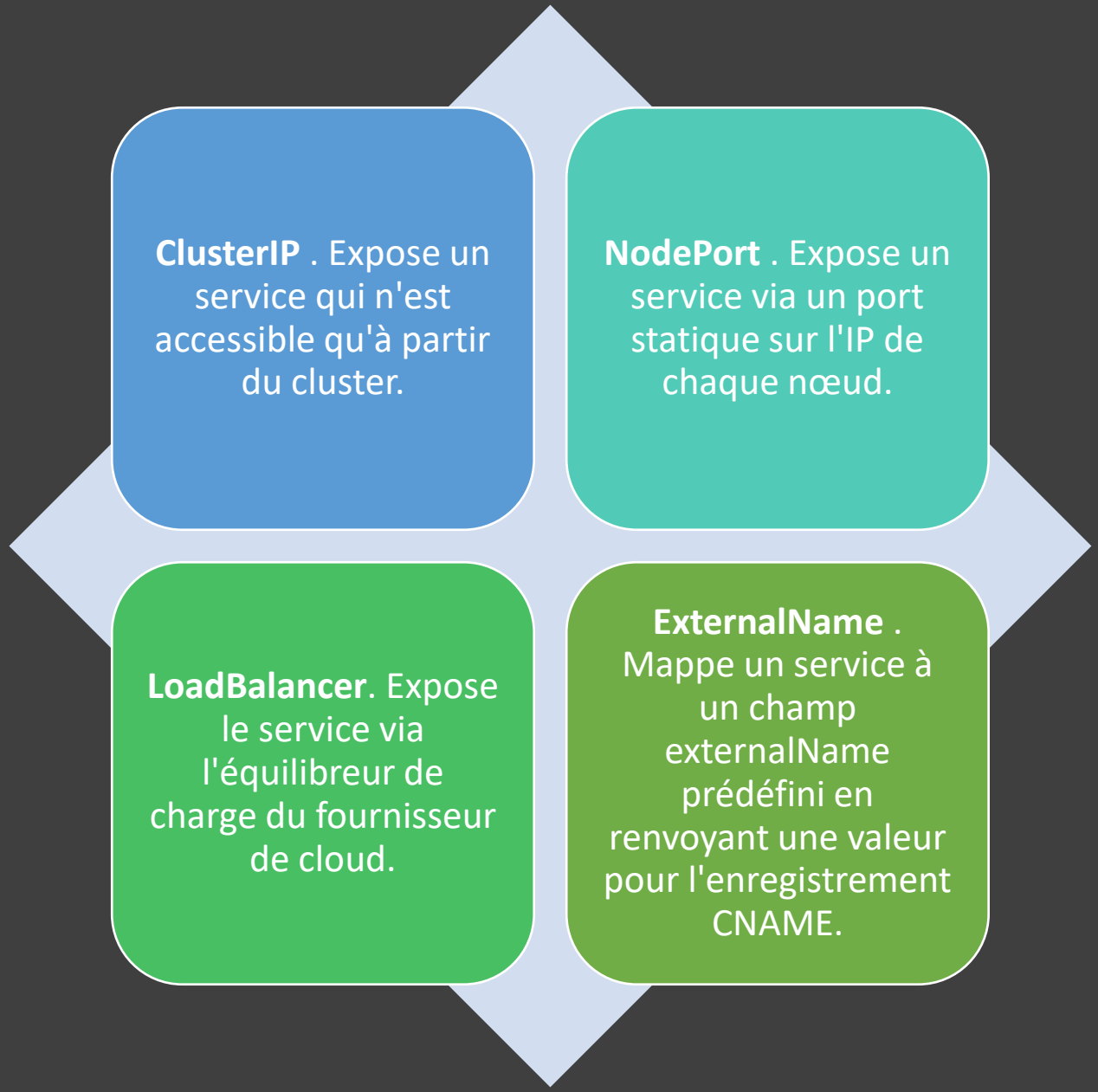
Service



PADOK

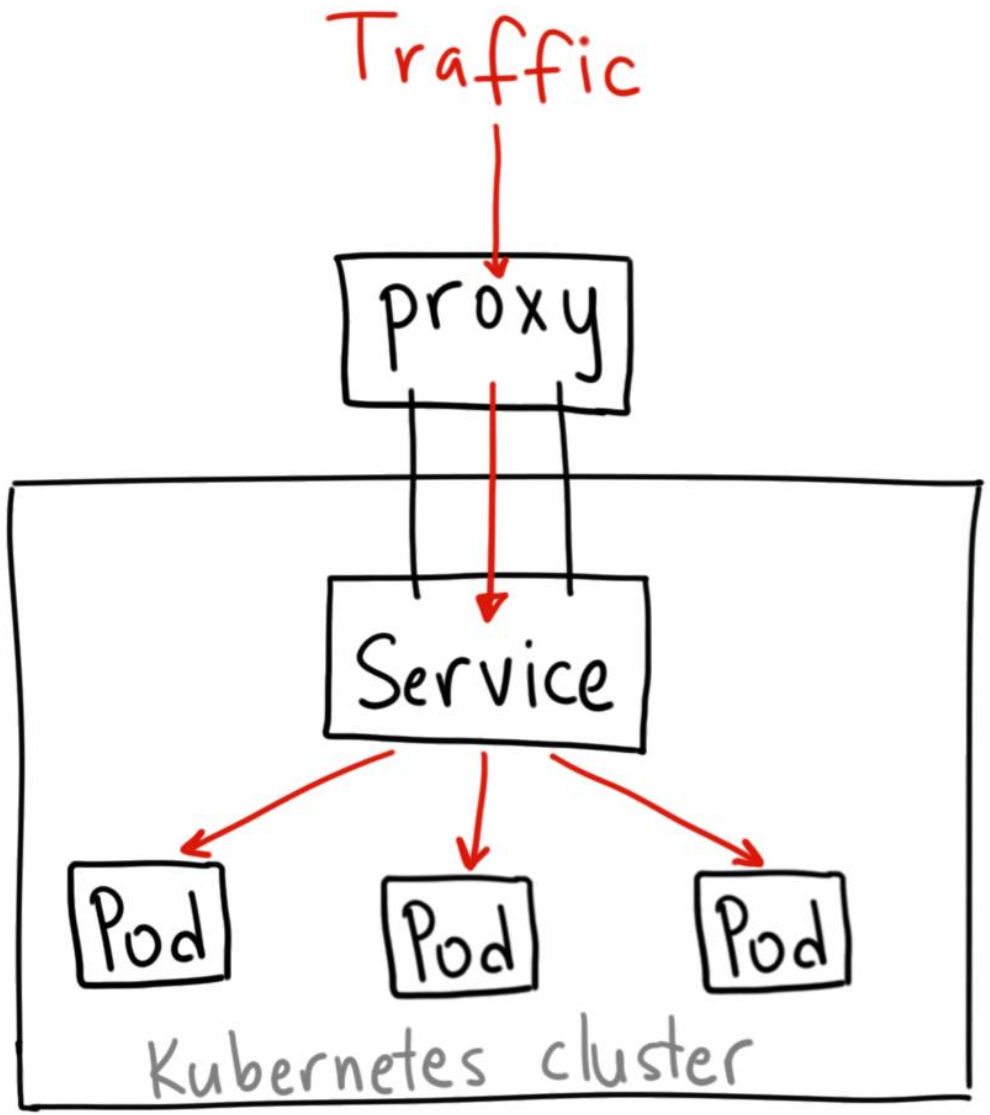
- La nature éphémère des pods les rend peu fiables pour assurer un trafic fluide. La solution de Kubernetes pour pallier cela est le **Service**.
- Ce service maintient une liste logique de pods qui acceptent le trafic entrant et exposent un port interne pour accéder aux pods sous-jacents.
- Le service Kubernetes peut donc gérer les changements de trafic en modifiant le nombre de pods.

Types de services Kubernetes



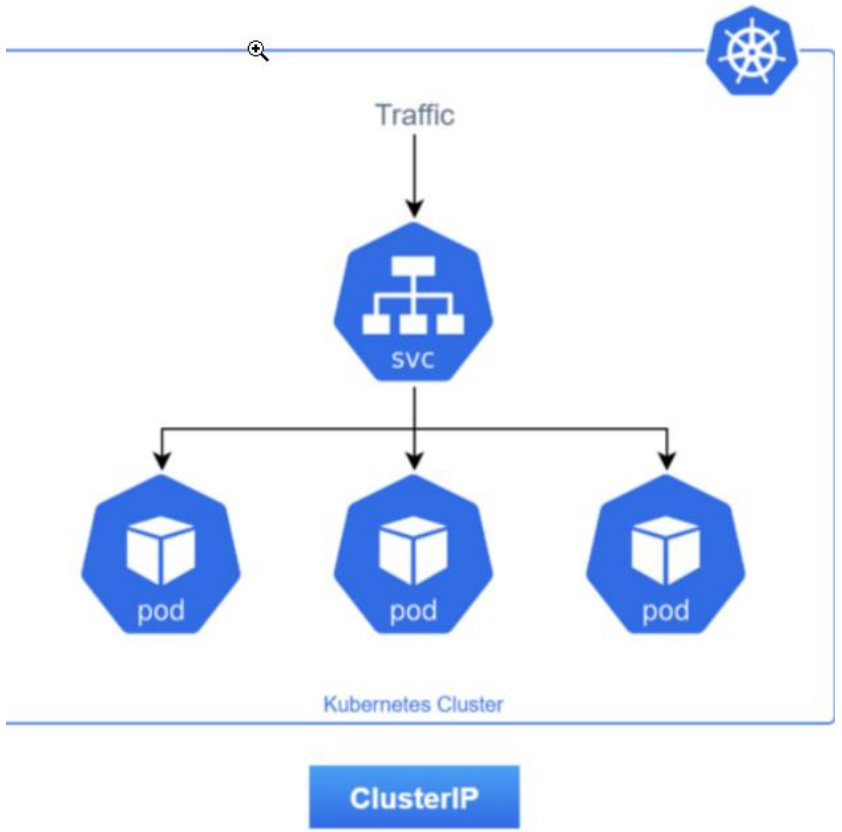
ClusterIP

- Expose le service sur une IP interne au cluster.
- Les autres pods peuvent alors accéder au service de l'intérieur du cluster, mais il n'est pas exposé à l'extérieur.
- Il faut un proxy pour le connecter à l'extérieur



ClusterIP

- ClusterIP est le type de service par défaut et le plus courant.
- Kubernetes attribuera une adresse IP interne au cluster au service ClusterIP. Cela rend le service accessible uniquement au sein du cluster.
- Vous ne pouvez pas faire de demandes de service (pods) depuis l'extérieur du cluster.
- **Cas d'utilisation**
- Communication interservices au sein du cluster. Par exemple, la communication entre les composants front-end et back-end de votre application.

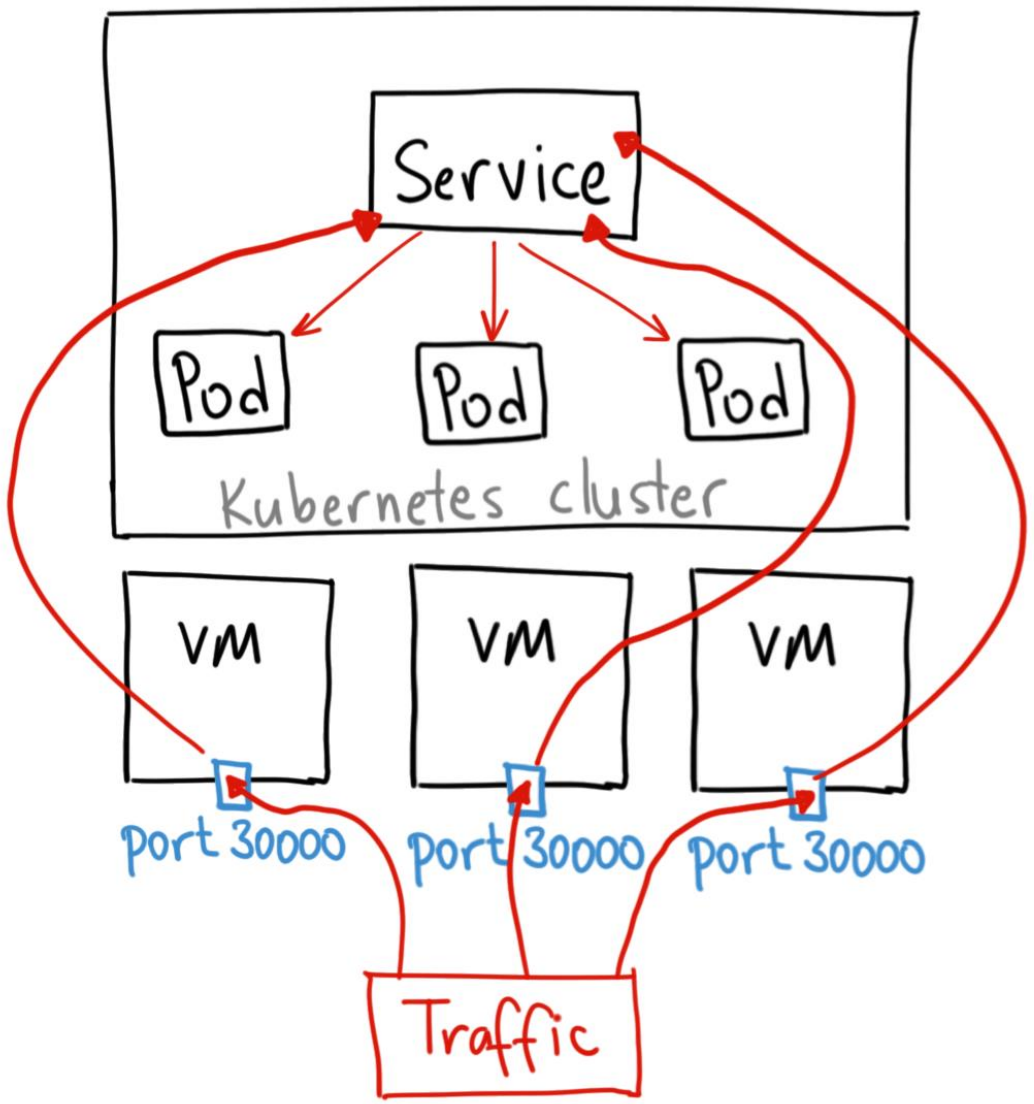


NodePort

expose le service depuis l'IP de chacun des nœuds du cluster en ouvrant un port directement sur le nœud, entre 30000 et 32767.

Cela permet d'accéder aux pods internes répliqués.

Comme l'IP est stable on peut faire pointer un DNS ou Loadbalancer classique dessus.

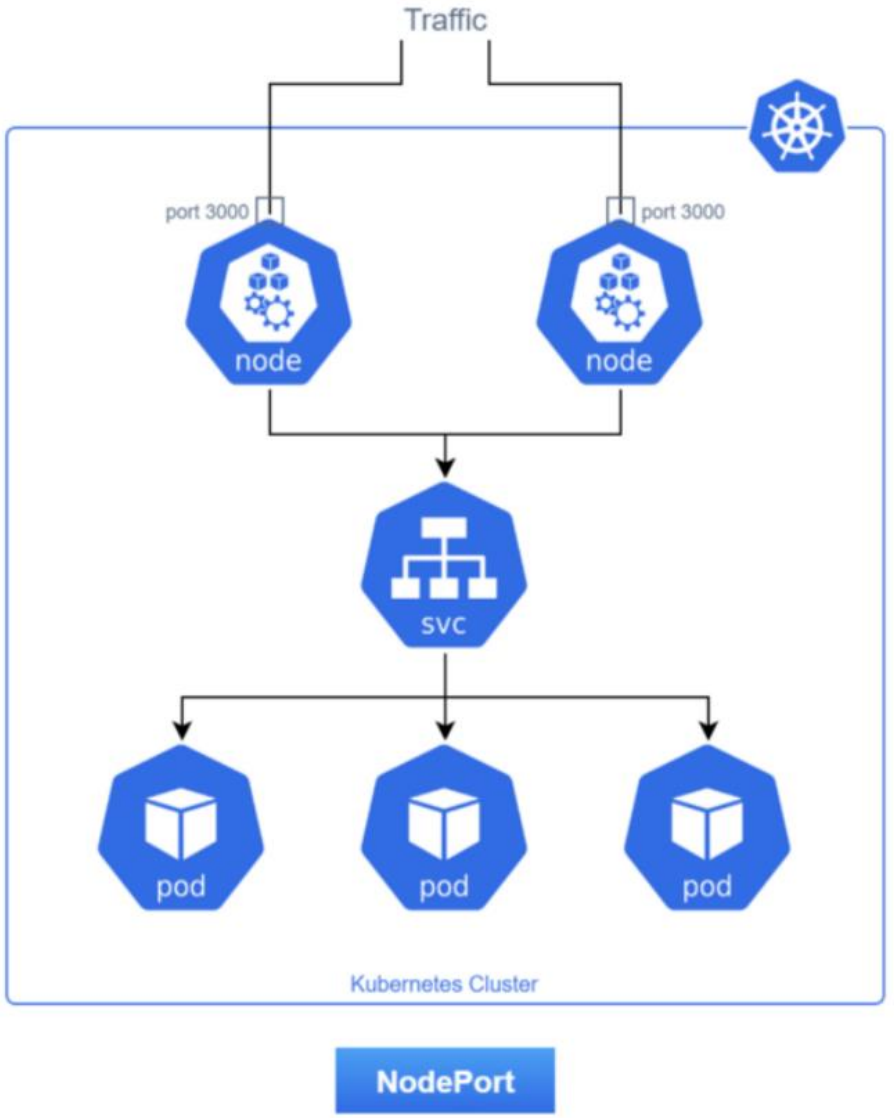


NodePort

- Extension du service ClusterIP vers lequel le service NodePort achemine et qui sera automatiquement créé.
- Expose le service à l'extérieur du cluster et ajoute un port à l'échelle du cluster au-dessus de ClusterIP.
- Expose le service sur l'adresse IP de chaque nœud sur un port statique (le NodePort). Chaque nœud va mandater (proxy) ce port vers votre service. Ainsi, le trafic externe a accès au port fixe sur chaque nœud. Cela signifie que toute demande adressée à votre cluster sur ce port est transmise au service.
- Vous pouvez contacter le service NodePort, depuis l'extérieur du cluster, en demandant **<NodeIP>:<NodePort>.**
- Le port de nœud doit être compris entre 30000 et 32767. L'attribution manuelle d'un port au service est facultative. S'il n'est pas défini, Kubernetes en attribuera automatiquement un.
- Si vous choisissez explicitement le port de nœud, assurez-vous que le port n'a pas déjà été utilisé par un autre service.

Cas d'utilisation

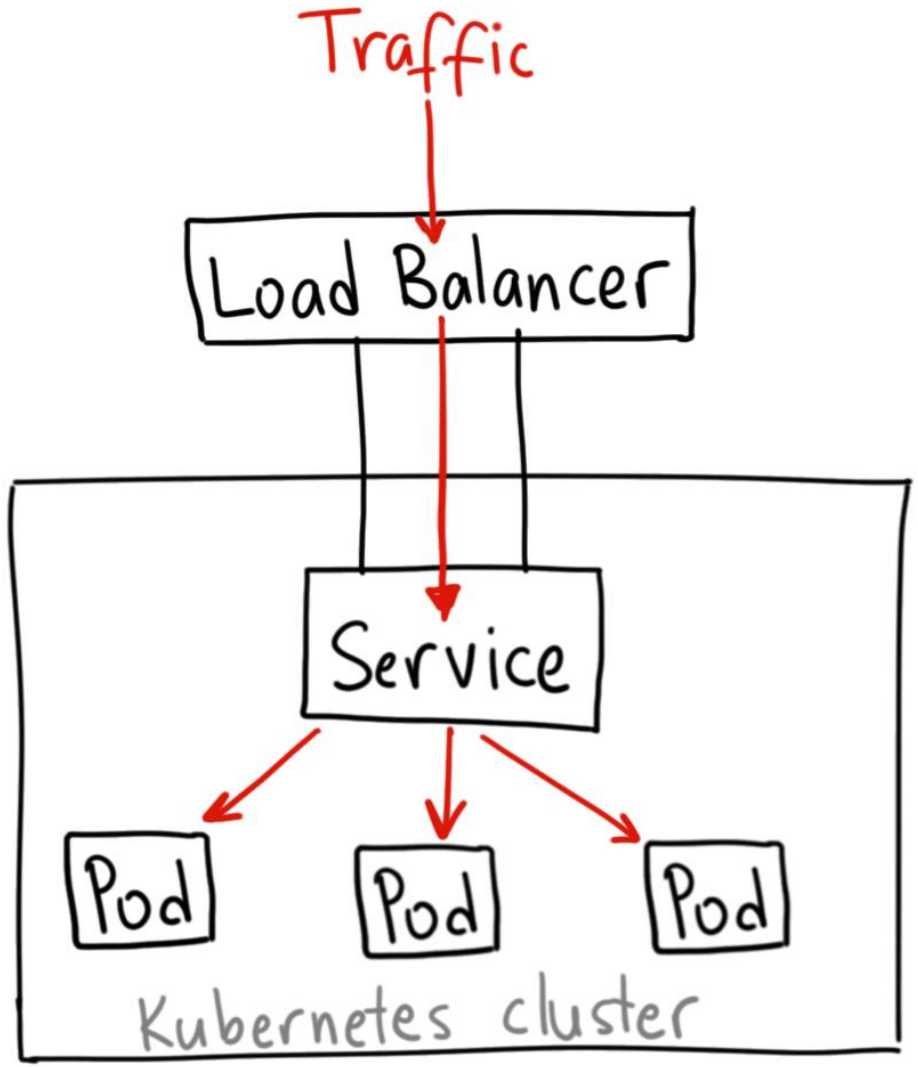
- Lorsque vous souhaitez activer la connectivité externe à votre service.
- L'utilisation d'un NodePort vous donne la liberté de configurer votre propre solution d'équilibrage de charge, de configurer des environnements qui ne sont pas entièrement pris en charge par Kubernetes, ou même d'exposer directement les IP d'un ou plusieurs nœuds.
- Préférez placer un équilibreur de charge au-dessus de vos nœuds pour éviter la défaillance des nœuds.



Load Balancer

Expose le service en externe à l'aide d'un Loadbalancer de fournisseur de cloud.

Les services NodePort et ClusterIP, vers lesquels le Loadbalancer est dirigé sont automatiquement créés.

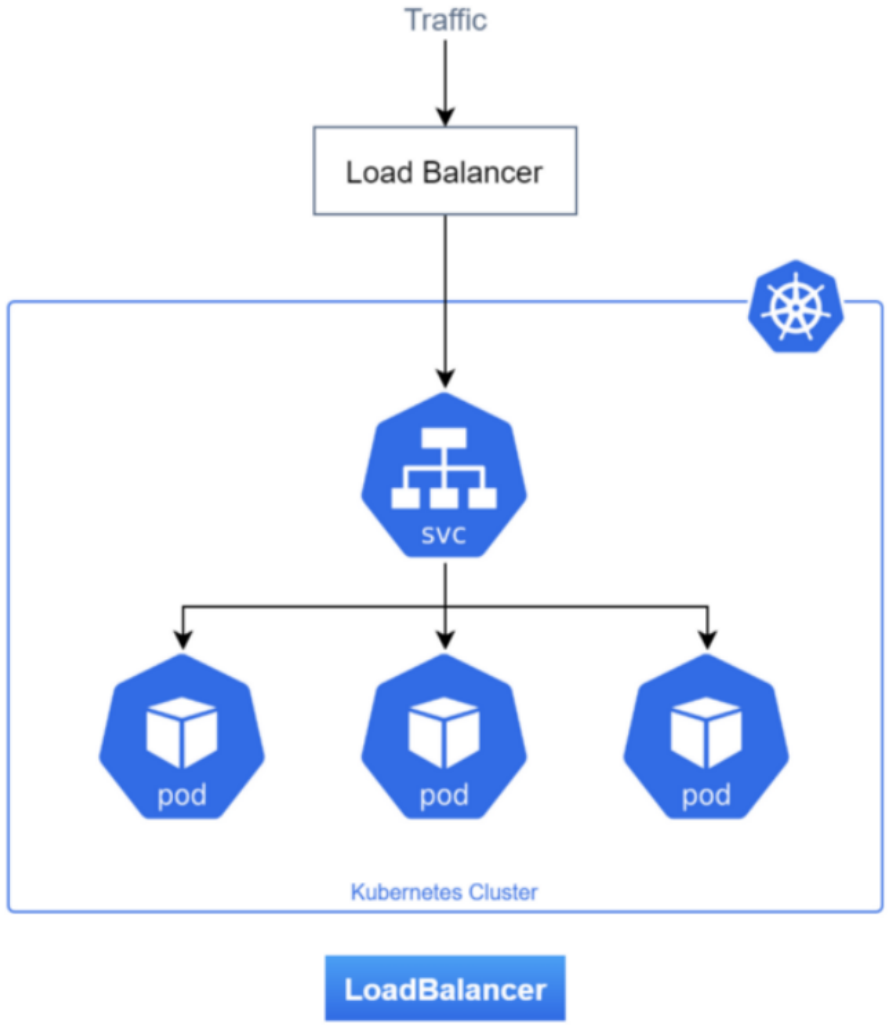


LoadBalancer

- Ce service est une extension du service NodePort. Les services NodePort et ClusterIP, vers lesquels l'équilibreur de charge externe achemine, sont automatiquement créés.
- Il intègre NodePort avec des équilibreurs de charge basés sur le cloud.
- Il expose le Service en externe à l'aide de l'équilibreur de charge d'un fournisseur de cloud.
- Chaque fournisseur de cloud (AWS, Azure, GCP, etc.) possède sa propre implémentation d'équilibreur de charge natif. Le fournisseur de cloud créera un équilibreur de charge, qui acheminera ensuite automatiquement les demandes vers votre service Kubernetes.
- Le trafic provenant de l'équilibreur de charge externe est dirigé vers les pods backend. Le fournisseur de cloud décide comment il est équilibré en charge.
- La création réelle de l'équilibreur de charge se produit de manière asynchrone.
- Chaque fois que vous souhaitez exposer un service au monde extérieur, vous devez créer un nouveau LoadBalancer et obtenir une adresse IP.

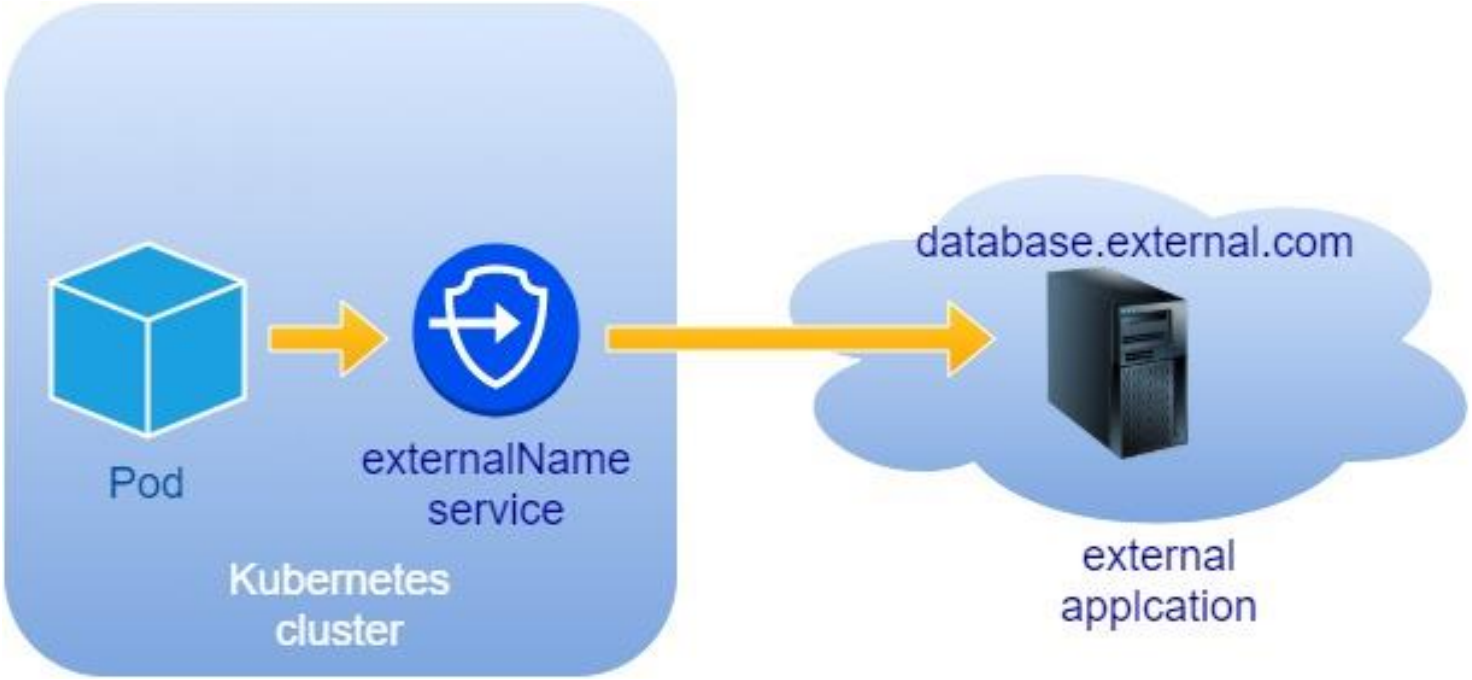
Cas d'utilisation

- Lorsque vous utilisez un fournisseur de cloud pour héberger votre cluster Kubernetes.



ExternalName

Pour pouvoir accéder à une application qui vit en dehors d'un cluster kubernetes , nous utilisons un service appelé externalName .

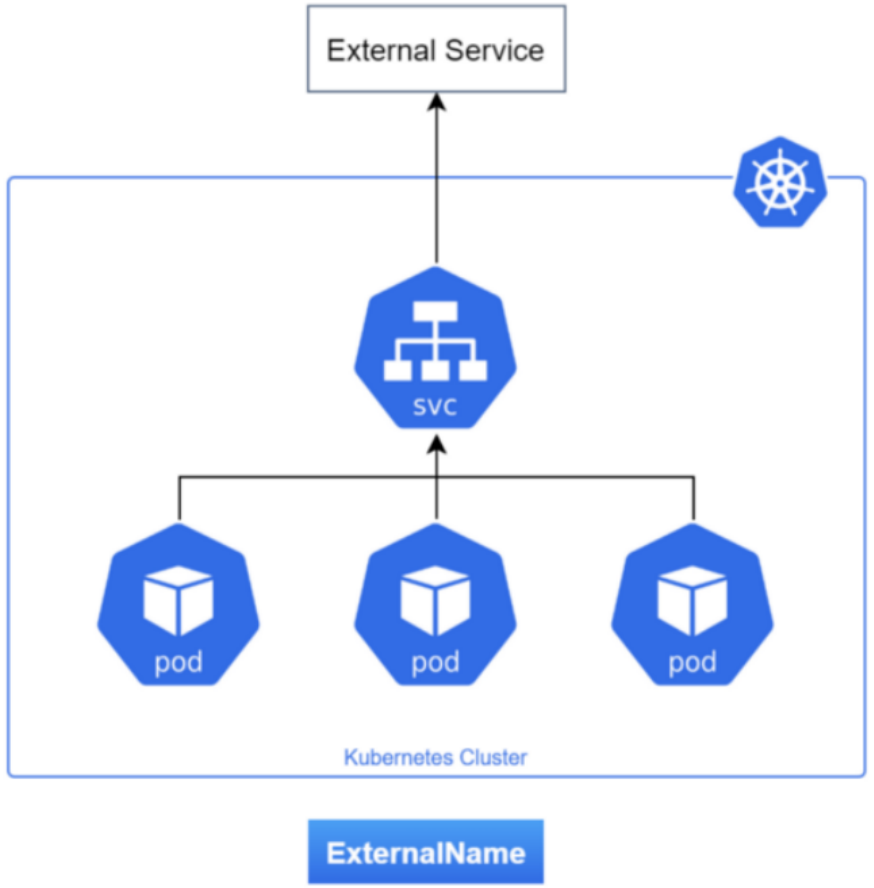


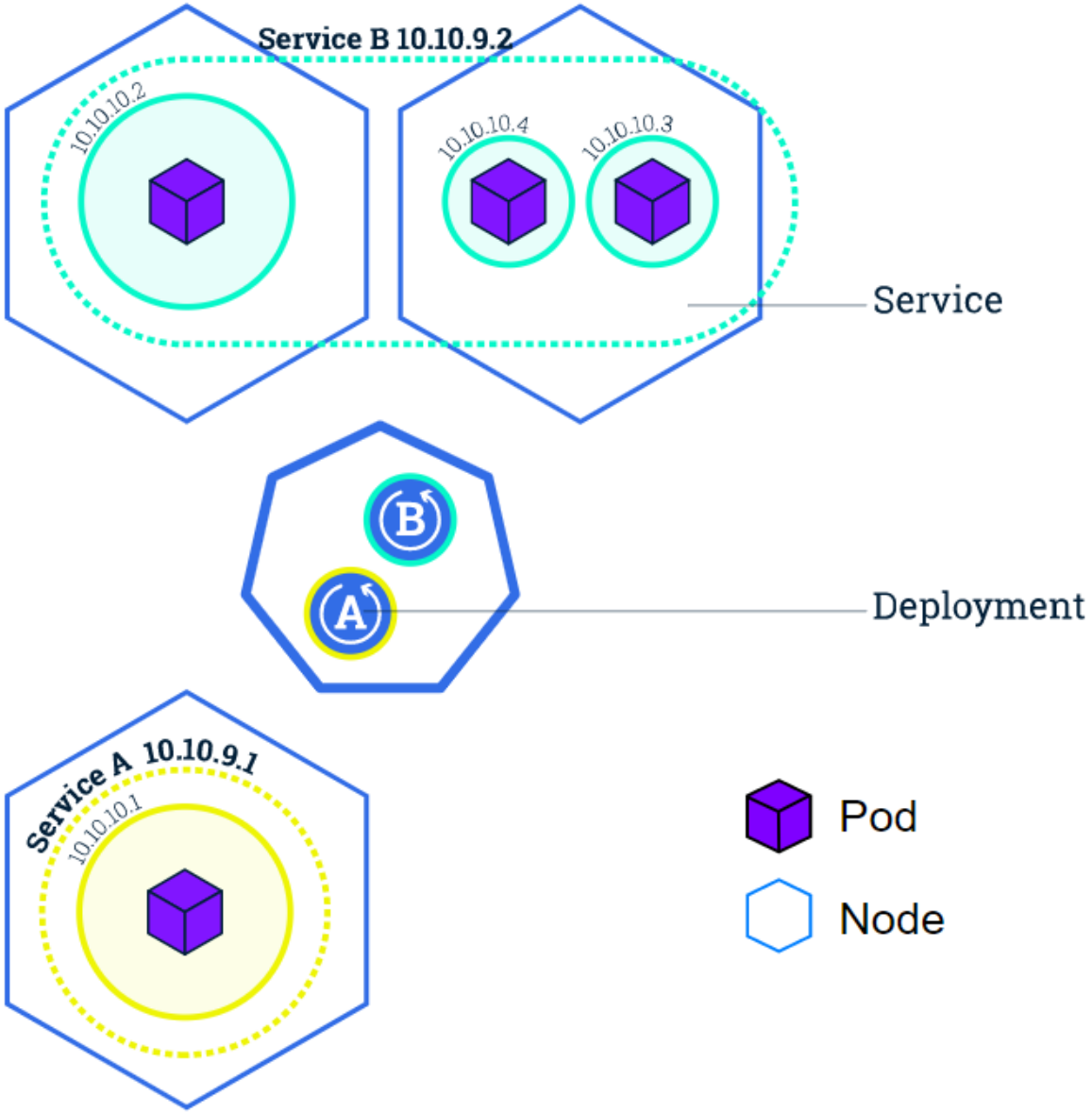
ExternalName

- Les services ExternalName mappent un service à un nom DNS.
- Vous spécifiez ces services avec le paramètre `spec.externalName`.
- Il mappe le service sur le contenu du champ externalName (par exemple foo.bar.example.com), en renvoyant un enregistrement CNAME avec sa valeur.
- Aucune procuration de quelque nature que ce soit n'est établie.

Cas d'utilisation

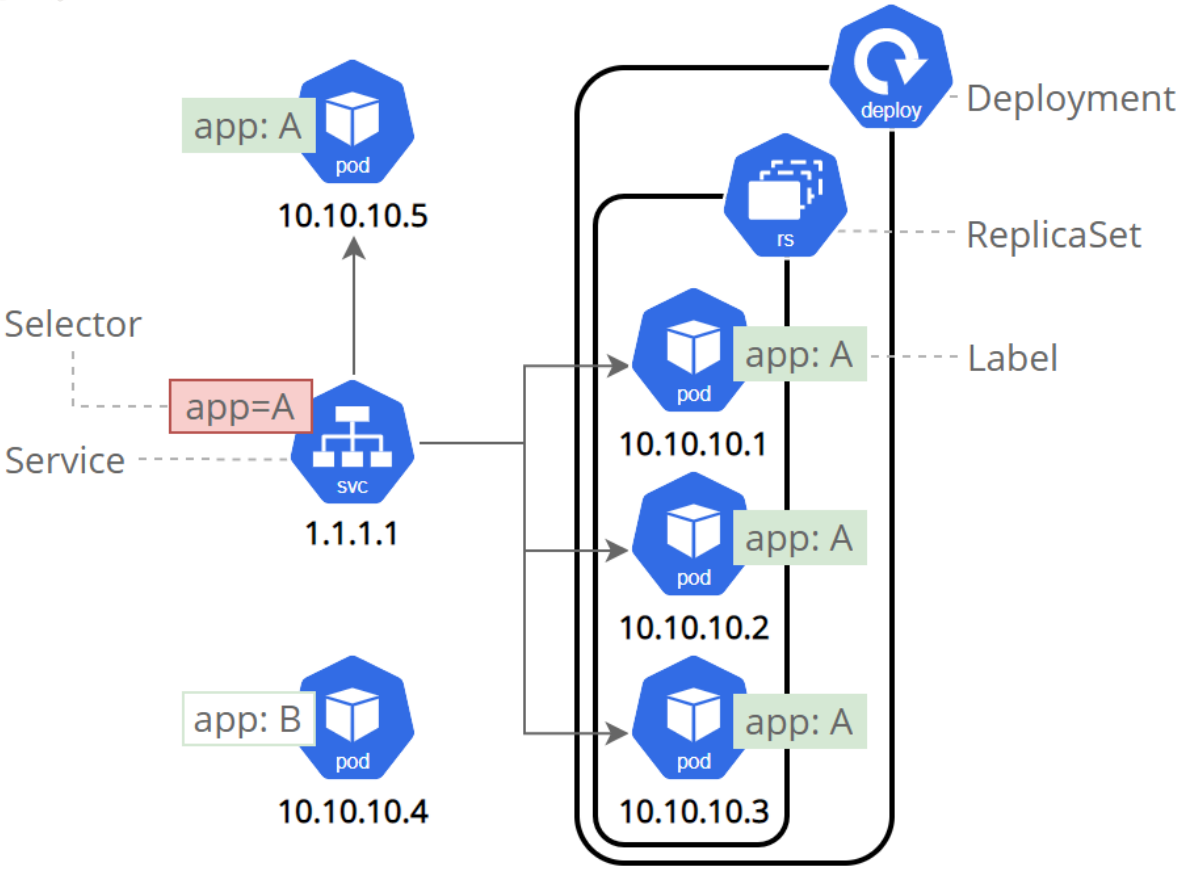
- Créer un service au sein de Kubernetes pour représenter une banque de données externe comme une base de données qui s'exécute en externe à Kubernetes.
- Utiliser ExternalName (en tant que service local) lorsque les pods d’un namespace communiquent avec un service dans un autre namespace.





Services & Labels
Ingress

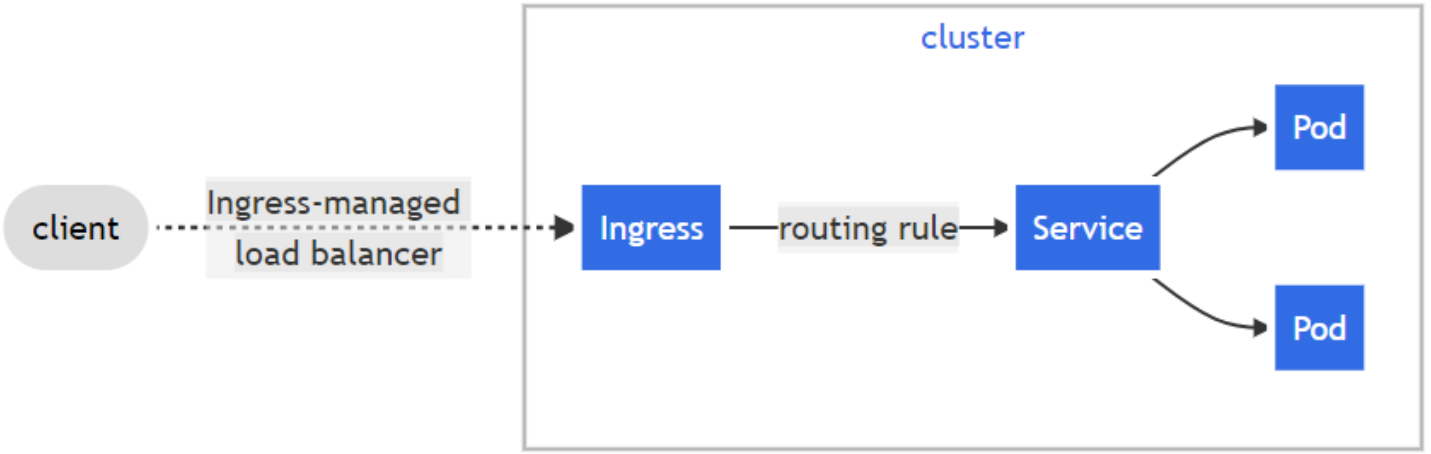
Services et Labels



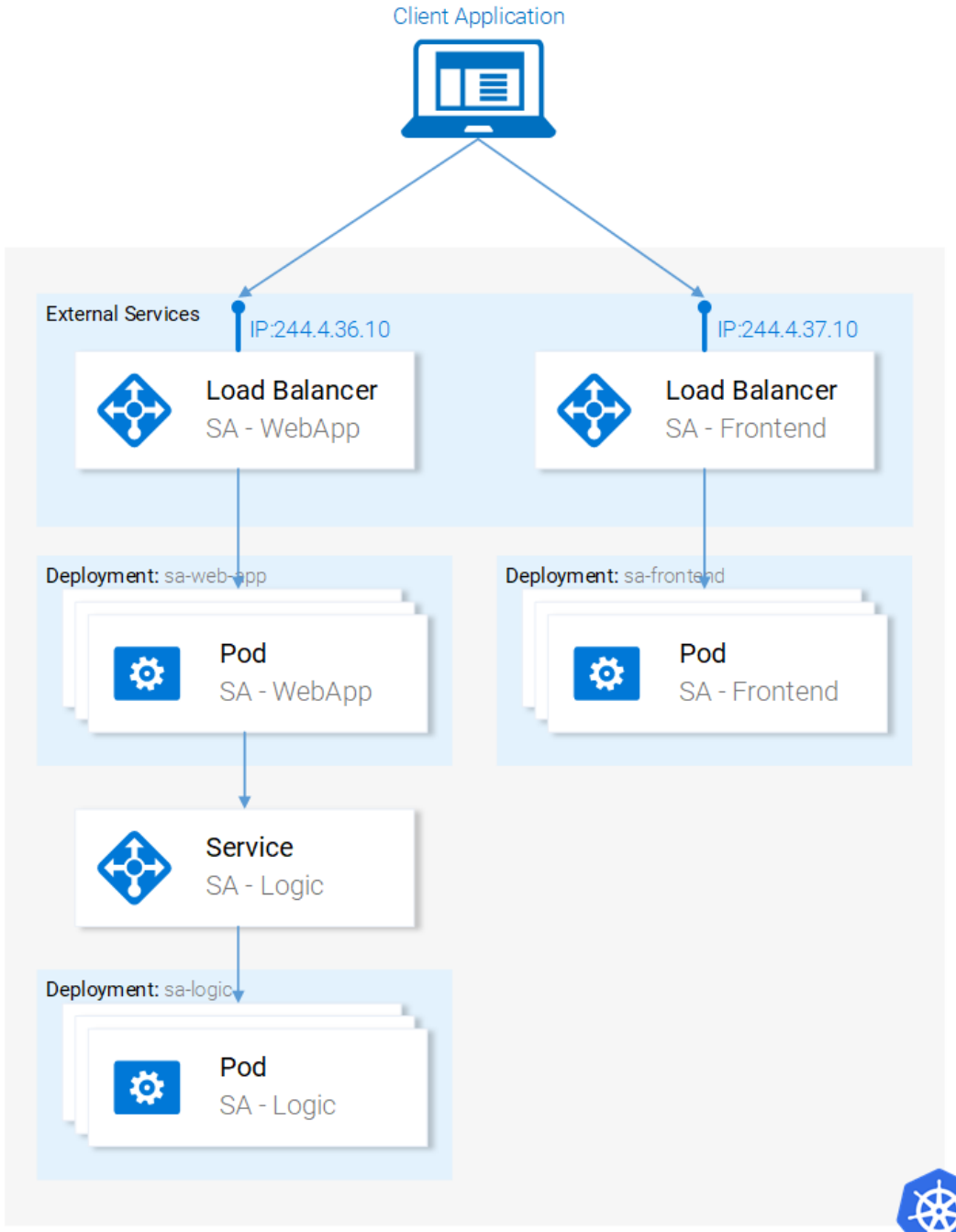
Ingress

- Si l'application doit être accessible de l'extérieur de votre cluster, Kubernetes propose un composant appelé **Ingress**. Il se trouve au dessus du service et gère les requêtes externes avec le cryptage SSL.

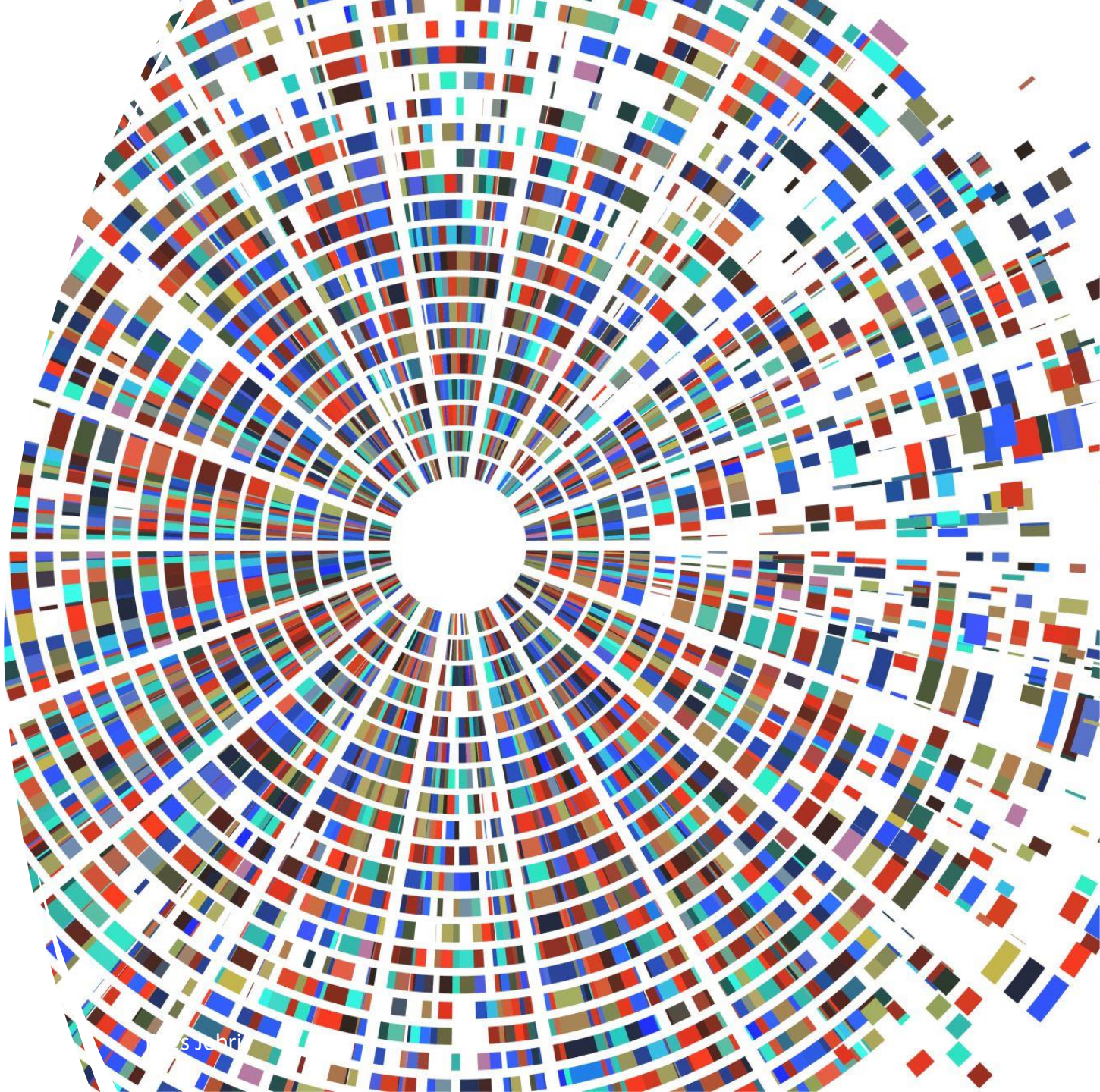
- Kubernetes gère le scaling des pods pour gérer leur mise à jour par exemple avec des **déploiements**.



En résumé



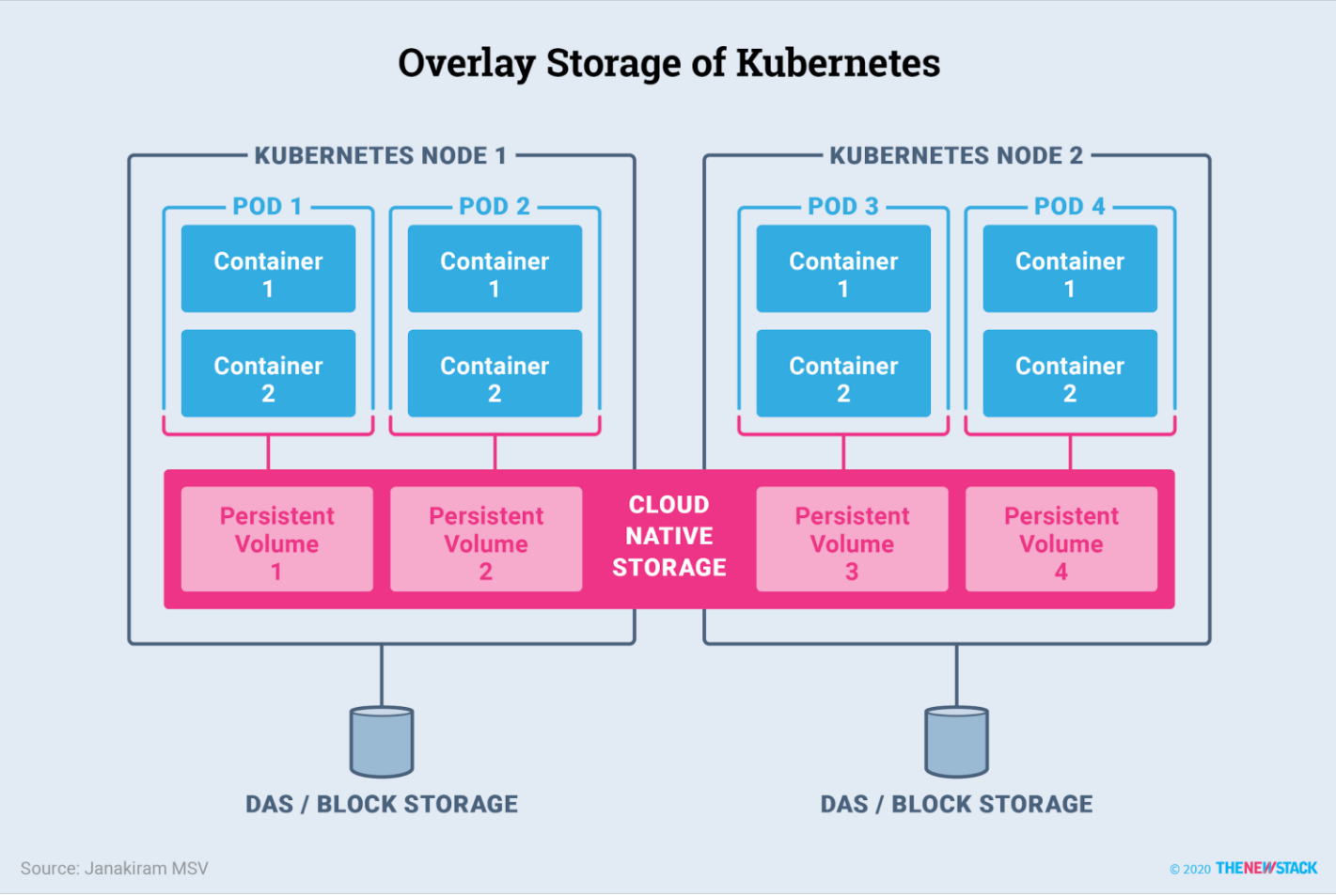
Les volumes Kubernetes



Introduction aux volumes Kubernetes

- Les volumes Kubernetes sont une fonctionnalité clé de la plate-forme de conteneurisation Kubernetes.
- Ils permettent aux conteneurs de stocker des données de manière persistante, même si le conteneur est détruit ou redéployé.
- Les volumes vous permettent également de partager des données entre des conteneurs dans le même pod.
- Cependant, les données de ce volume seront détruites lors du redémarrage du pod.
- Pour résoudre ce problème , Kubernetes dispose de volumes persistants . Les volumes persistants sont un stockage à long terme dans votre cluster Kubernetes.

Overlay Storage of Kubernetes: Expose le Storage vers Pods et Containers

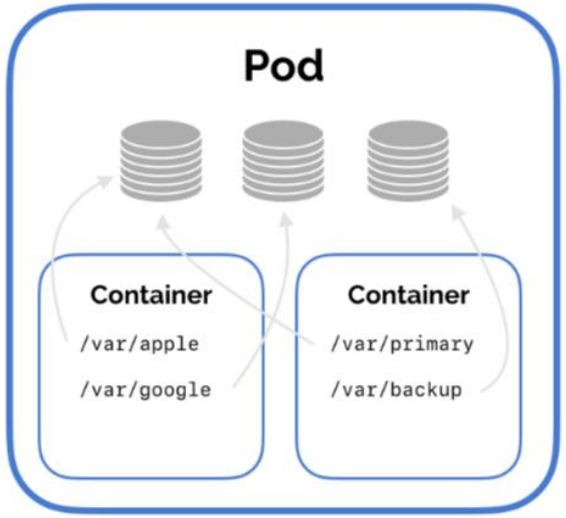


Pods / Container Volumes

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: simple-volume-pod
spec:
  # Volumes are declared by the pod. They share its lifecycle
  # and are communal across containers.
  volumes:
    # Volumes have a name and configuration based on the type of volume.
    # In this example, we use the emptyDir volume type
    - name: simple-vol
      emptyDir: {} # No extra configuration

  # Now, one of our containers can mount this volume and use it like
  # any other directory.
  containers:
    - name: my-container
      volumeMounts:
        - name: simple-vol # This is the name of the volume we set at the
          mountPath: /var/simple # Where to mount this directory in our co

      # Now that we have a directory mounted at /var/simple, let's
      # write to a file inside it!
      image: alpine
      command: ["/bin/sh"]
      args: ["-c", "while true; do date >> /var/simple/file.txt; sleep 5;
```



Volumes in **pod**
Defined in `pod.spec.volumes` field

Volume mounts in **container**
Defined in
`pod.spec.containers.volumeMounts`
Uses one of the volumes

Types de volumes Kubernetes

- Les volumes Kubernetes prennent en charge différents types de stockage, notamment les disques, les fichiers locaux, les systèmes de fichiers réseau et les services de stockage dans le cloud.
- Les types de volumes les plus couramment utilisés sont : **EmptyDir**, **HostPath**, **NFS**, **PersistentVolumeClaim** et **ConfigMap**.

EmptyDir

- EmptyDir est un volume Kubernetes qui est créé lorsqu'un conteneur est démarré et qui est détruit lorsque le conteneur est arrêté.
- Il peut être utilisé pour stocker des fichiers temporaires ou pour partager des fichiers entre plusieurs conteneurs.

```
volumes:
  - name: example-emptydir
    emptyDir: {}
```

HostPath

- HostPath est un volume Kubernetes qui monte un répertoire sur le nœud hôte dans le conteneur.
- Il peut être utilisé pour stocker des données persistantes sur le nœud hôte.

```
volumes:
  - name: example-hostpath
    hostPath:
      path: /path/on/host
```

NFS

- NFS est un système de fichiers réseau qui peut être monté dans un conteneur Kubernetes à l'aide d'un volume Kubernetes.
- Il permet de stocker des données de manière persistante et de les partager entre plusieurs conteneurs.

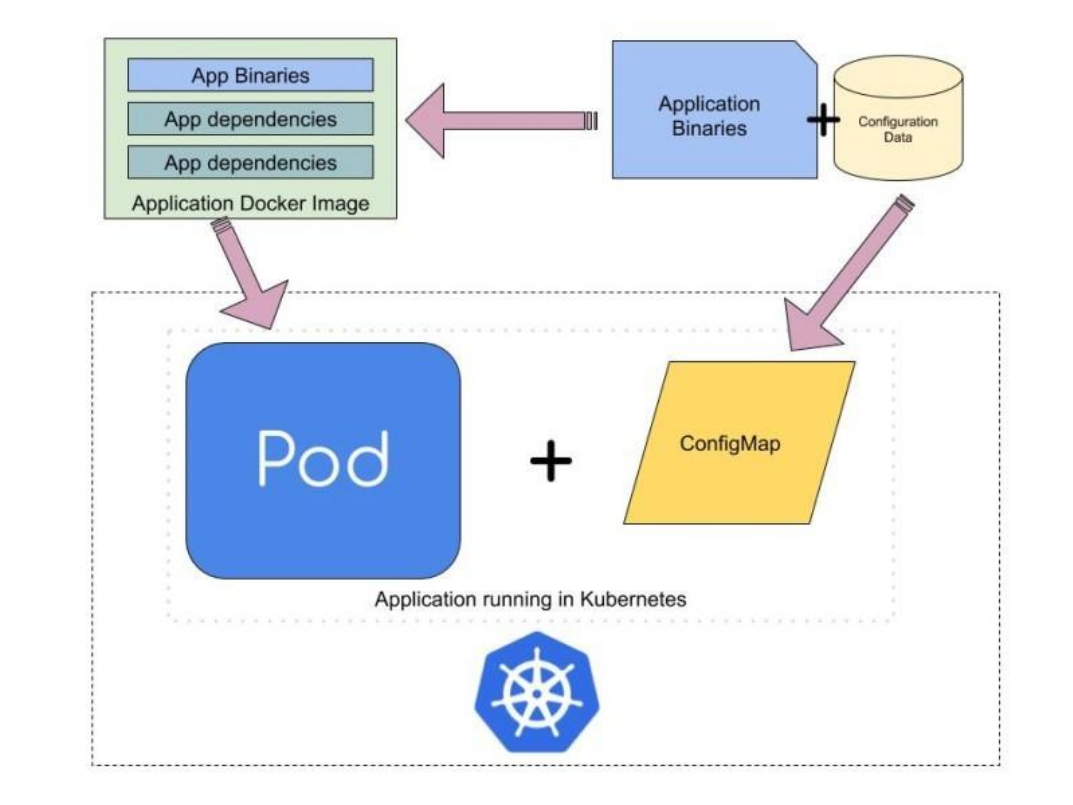
```
volumes:
  - name: example-nfs
    nfs:
      server: nfs-server.example.com
      path: /exported/path
```

PersistentVolume/Claim

- Les PersistentVolumes (PV) représentent des ressources de stockage physiques dans un cluster Kubernetes.
- Les PersistentVolumeClaims (PVC) sont des demandes d'accès à des ressources de stockage persistantes.
- Les PVC consomment des PV, fournissant une abstraction entre le stockage sous-jacent et les applications.

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

ConfigMap & secret



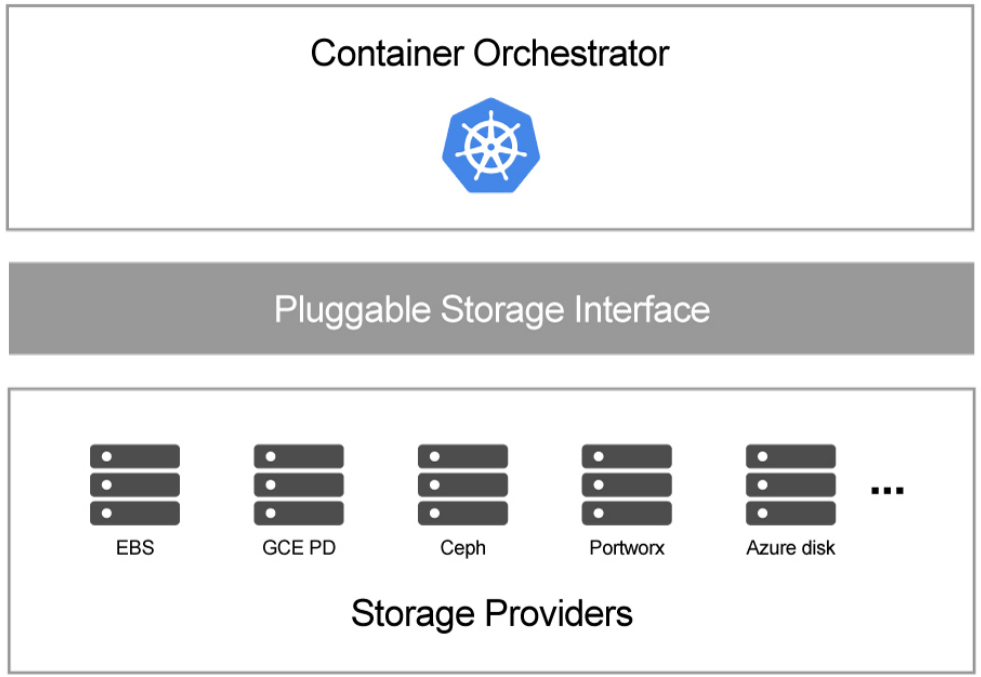
- ConfigMap est un objet Kubernetes qui stocke des données de configuration dans le cluster Kubernetes.
- Il peut être monté dans un conteneur comme un volume Kubernetes et permet de stocker des variables d'environnement, des fichiers de configuration et d'autres données de configuration.

```
volumes:
  - name: example-configmap
    configMap:
      name: example-configmap
```

```
volumes:
  - name: example-secret
    secret:
      secretName: example-secret
```

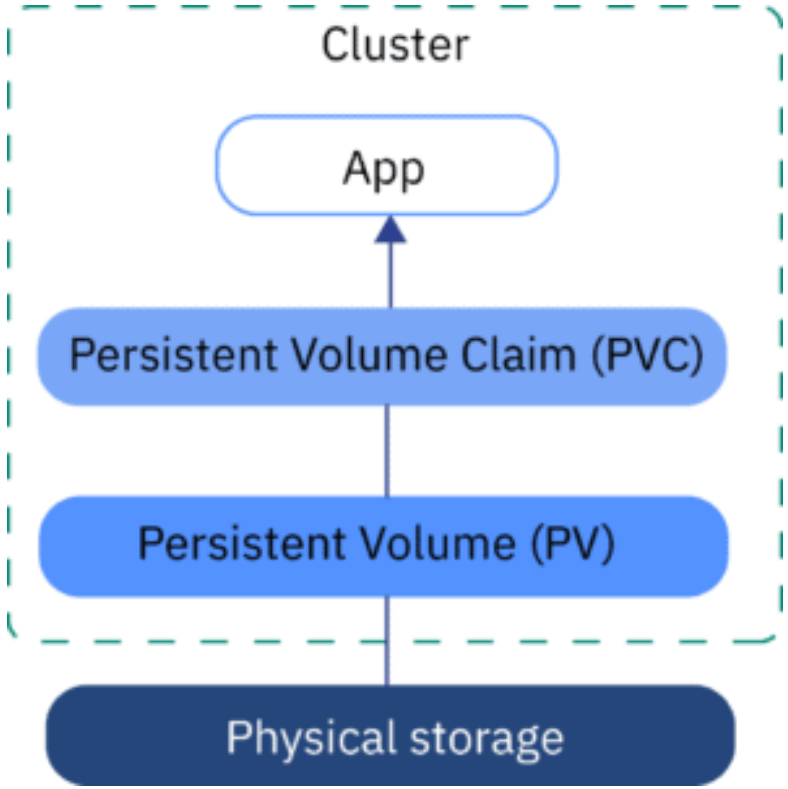
CSI (Container Storage Interface)

- Le CSI est une interface standard pour l'intégration de solutions de stockage externes avec Kubernetes.
- Il permet aux fournisseurs de stockage tiers de développer des pilotes de stockage qui peuvent être utilisés de manière interchangeable.



Storage Classes

- Les Storage Classes sont une autre abstraction dans Kubernetes qui définit les propriétés du stockage, telles que le type de stockage (SSD, HDD), les performances, etc.
- Les utilisateurs peuvent déclarer des PVC sans spécifier de manière explicite le type de stockage, laissant Kubernetes choisir automatiquement une classe de stockage appropriée en fonction des règles définies dans la Storage Class.



Partage de volumes Kubernetes

- Les volumes Kubernetes peuvent être partagés entre plusieurs conteneurs en utilisant des points de montage partagés.
- Cela permet aux conteneurs de partager des données et de communiquer entre eux.

Volumes Kubernetes et déploiements

- Les volumes Kubernetes sont souvent utilisés dans les déploiements Kubernetes pour fournir des données persistantes aux conteneurs.
- Ils permettent de stocker des données de base de données, des fichiers de configuration et d'autres données nécessaires au fonctionnement de l'application.