

Ateliers Techniques - Kubernetes



kubernetes

CHENIOUR Yassine
DENA Nico
LECOMTE Thibaud

Sommaire

01 Pourquoi Kubernetes ?

02 Docker & Kubernetes

03 Architectures

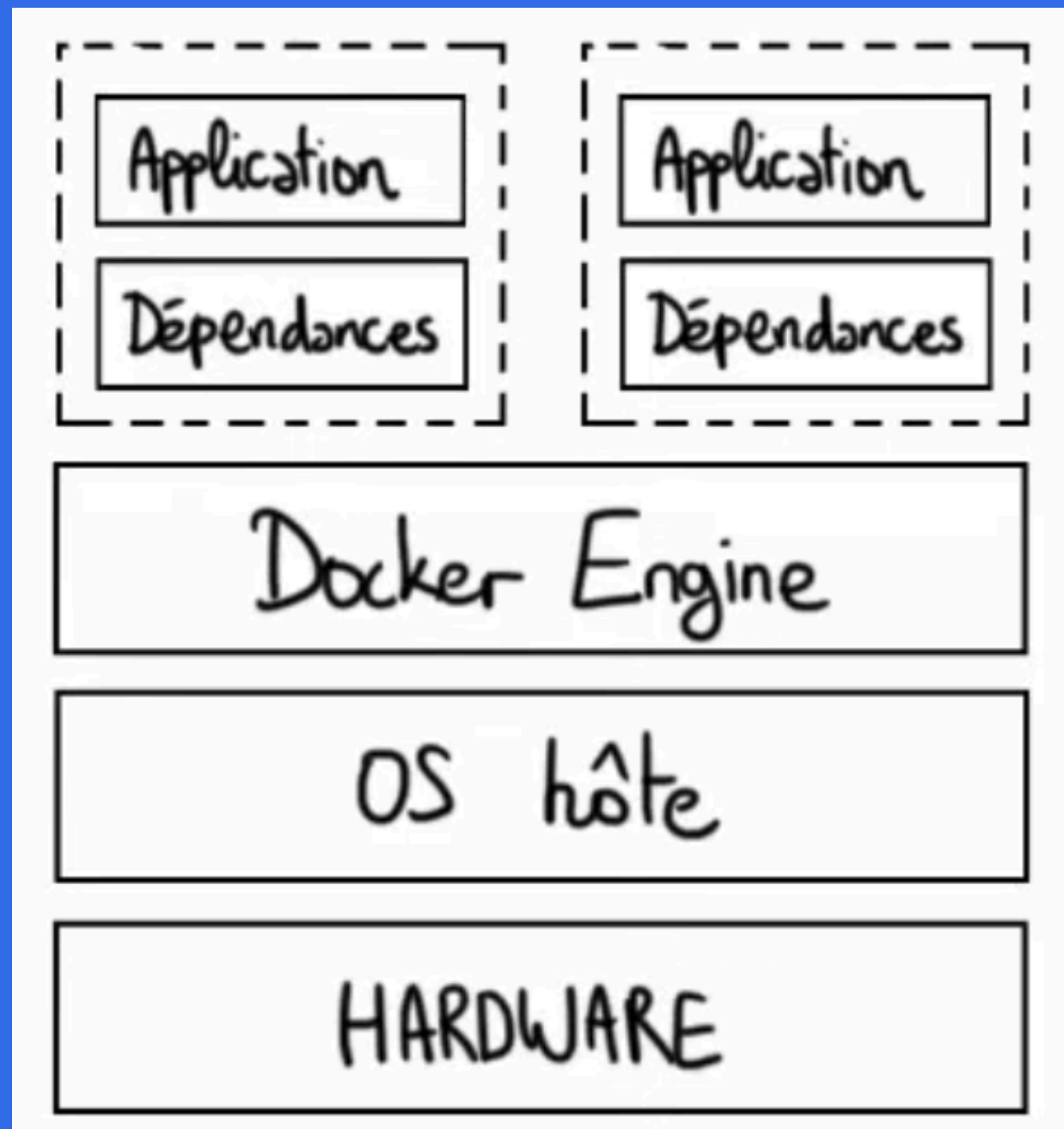
04 Concepts

05 Cas d'utilisation

06 Commandes utiles



RAPPELS CONTENEURS



→ Conteneurisation : **virtualisation** de l'environnement d'exécution

- Ainsi chaque **conteneur** a ses propres :

- Processus
- Systèmes de fichiers (donc application)
- Dépendances

→ Ne virtualise pas l'OS (contrairement au VM)

→ Créé à partir d'un modèle (**Image** Docker)

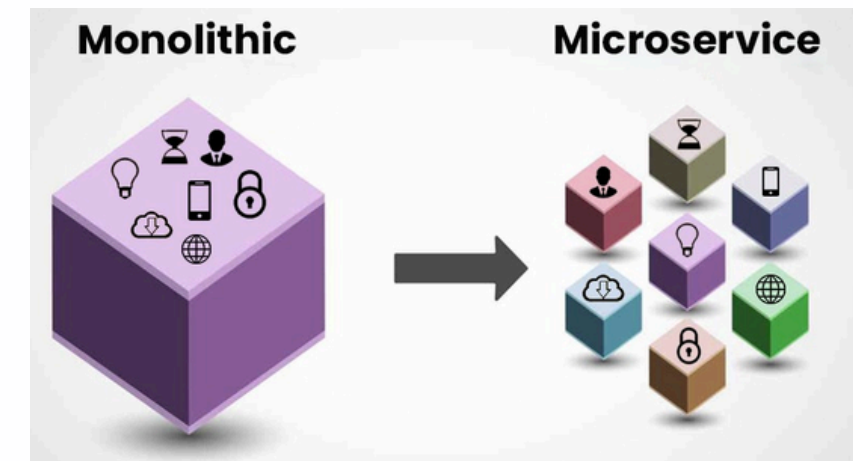
- Un conteneur est donc une **instance** d'une image

→ Points clés : **Cloisonnement, isolation, portabilité**

POURQUOI KUBERNETES ?

Evolution des architectures

- Historiquement, les applications étaient développées comme un seul bloc (**monolithic**)
- Les architectures modernes ont ensuite migré vers du **microservice** (services indépendants)
- Les **conteneurs** sont devenus le format naturel pour cloisonner ces services (compatibilité cloud)



ORCHESTRATION



Problème : Orchestrer les conteneurs

- Les architectures modernes sont parfois composées de **milliers de conteneurs**
- Défis : visibilité, redémarrage, répartition de charges, déploiement, monitoring
- La gestion de ces conteneurs s'appelle "**orchestration**" (solution aux défis)
- Exemples d'orchestrateur : Docker Swarm, apache mesos, nomad, Kubernetes
- **Kubernetes** est l'orchestrateur le plus populaire et domine le marché

Kubernetes (K8s)

- Système open-source **d'orchestration de conteneurs** développé par Google
- **Automatise** le déploiement, la mise à l'échelle et la gestion d'applications conteneurisées
- "**Chef d'orchestre**" qui gère des milliers de conteneurs Docker à notre place
- Très populaire pour son intégration en environnement **cloud** et la **scalabilité** qu'il offre



kubernetes

DOCKER ET KUBERNETES

DIFFERENCES

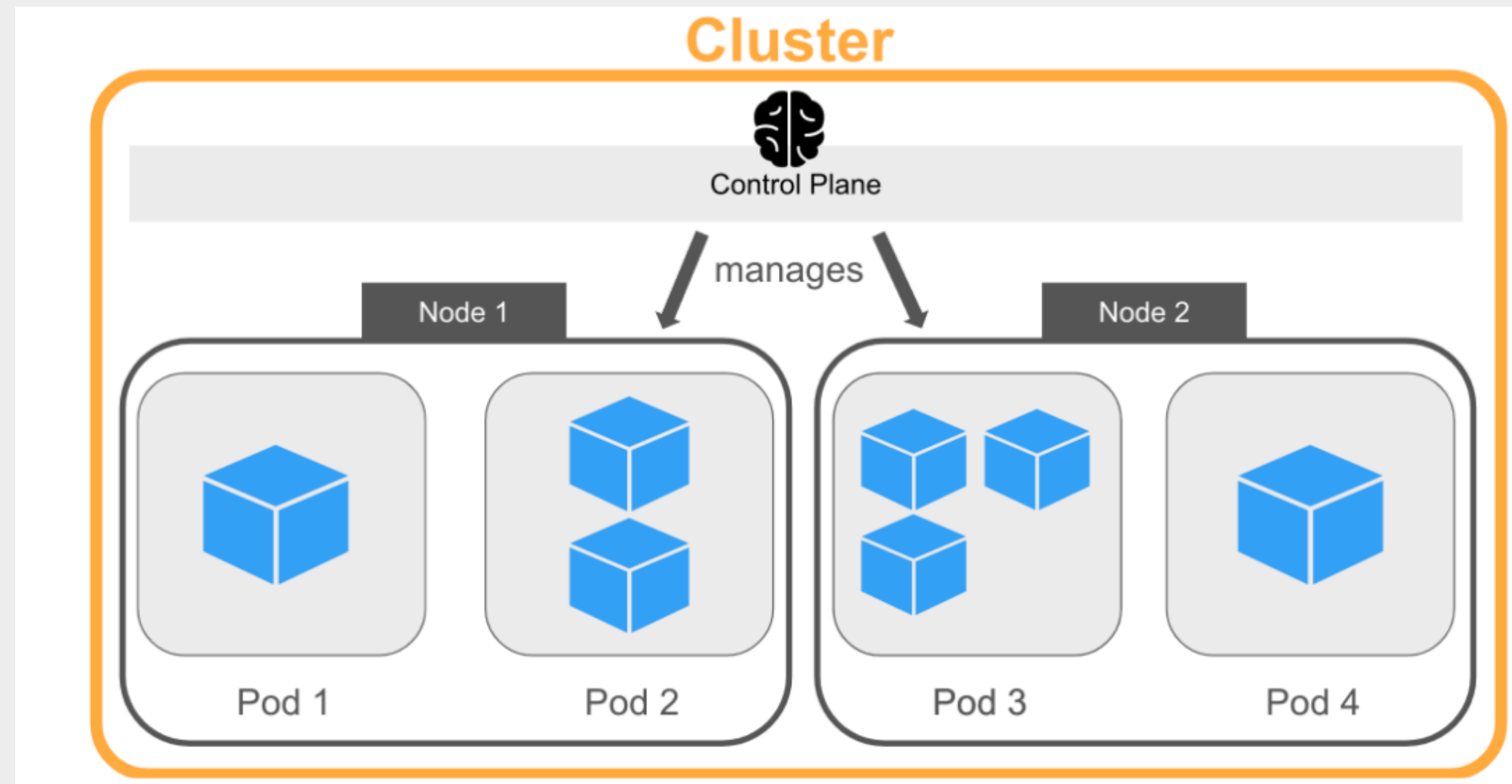
 VS 

	DOCKER	KUBERNETES
Restart auto	Aucun	Auto-healing
Scaling	Manuel	Automatique
Load Balancing	Aucun	Natif
Configuration	Impérative	Déclarative
Monitoring	Limité	Intégré
Echelle	Une machine hôte	Cluster de machines

COMPLEMENTARITE

- En réalité, ils ne s'opposent pas mais sont **complémentaires**
 - **Docker** produit et met à jour les images (et conteneurs)
 - **Kubernetes** les déploie et les gère en production
- Kubernetes ne crée pas de conteneurs lui-même
 - C'est un **Container Engine** (souvent Docker) qui s'en charge

OVERVIEW



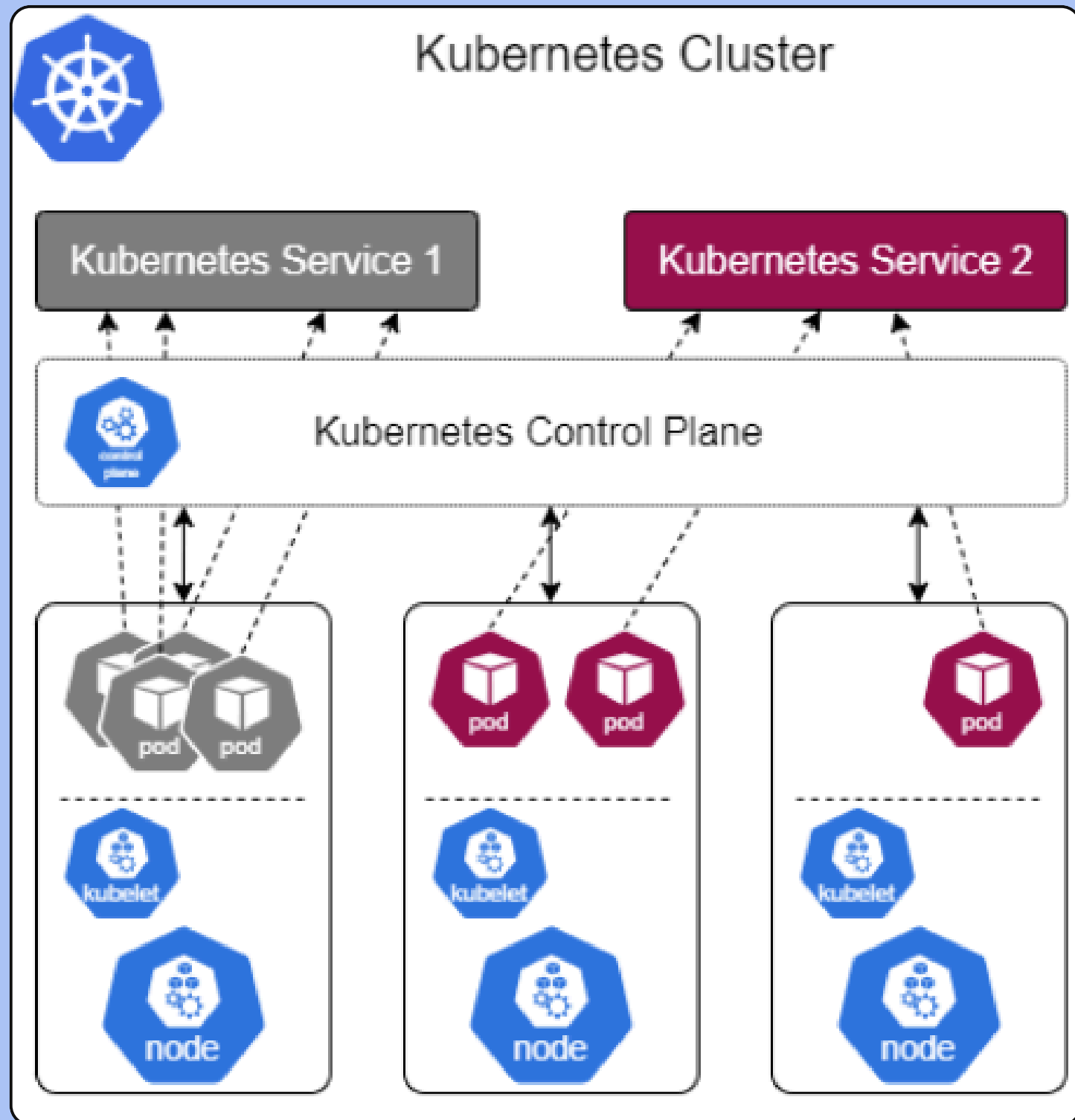
Cluster : ensembles de machines (physiques ou virtuelles) sur lesquelles Kubernetes déploie et gère les applications

Control Plane : C'est le cerveau du cluster, il prend les décisions et gère l'état global. Il est composé de processus clés.

Nodes : Une machine du cluster qui exécute les applications.

Pods : Ensemble d'un ou plusieurs conteneurs. C'est la plus petite unité de calcul que nous pouvons déployer dans Kubernetes.

CONCEPTS FONDAMENTAUX



Cluster

- **Ensemble de machines** connectées entre elles
- **Unité de base**, tous se passe dedans
- **Scalable** de quelques nodes à des centaines
- Possède un **Control Plane** (ex-Master)
 - Gère le pilotage global du cluster

Node

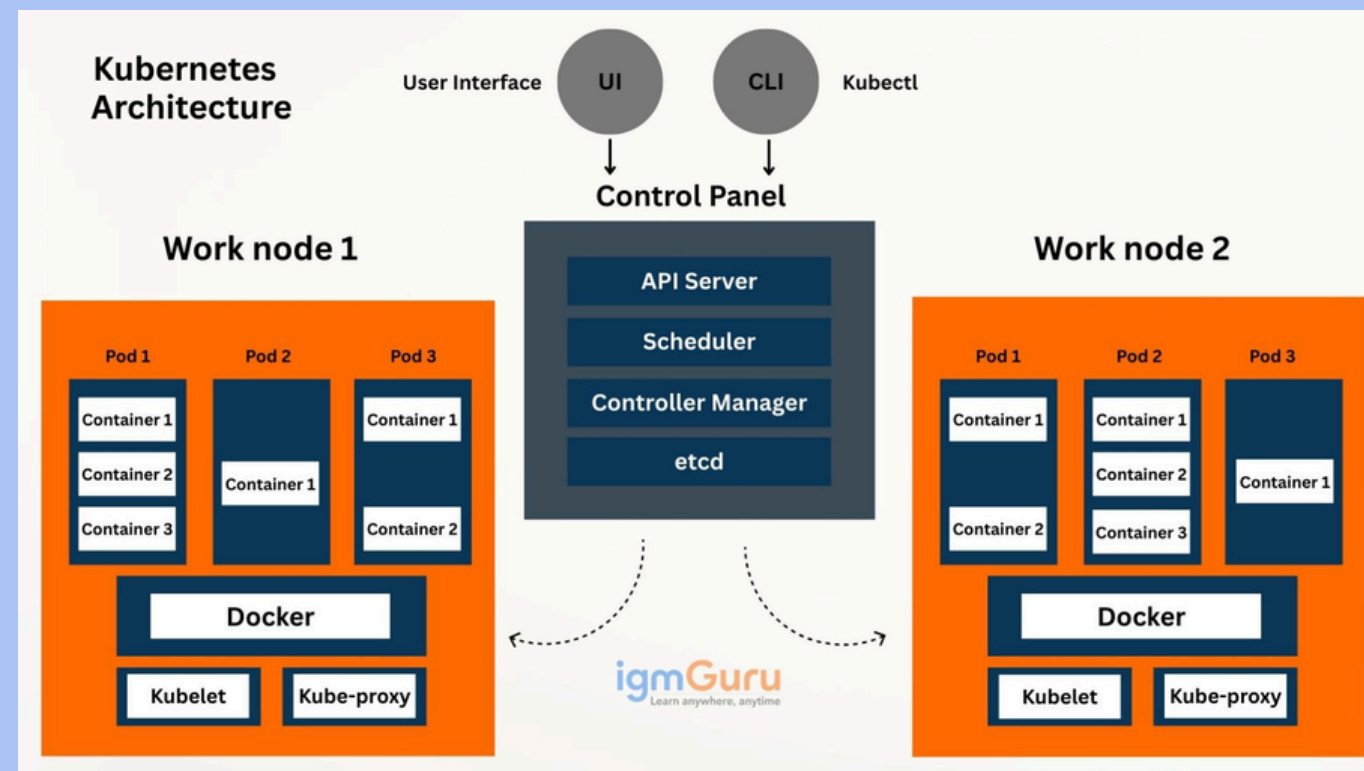
- **Machine individuelle** qui exécute les apps
- Chaque Node a un Kubelet
 - Agent qui exécute et surveille les Pods
 - Communique avec le Control Plane
- Héberge des Pods

Pod

- **Plus petite unité** de déploiement
- Encapsule **un ou plusieurs conteneurs** partageant les même réseau et stockage
- K8s opère sur les Pods, pas directement sur les conteneurs
- Les Pods sont **éphémères** (vie courte)

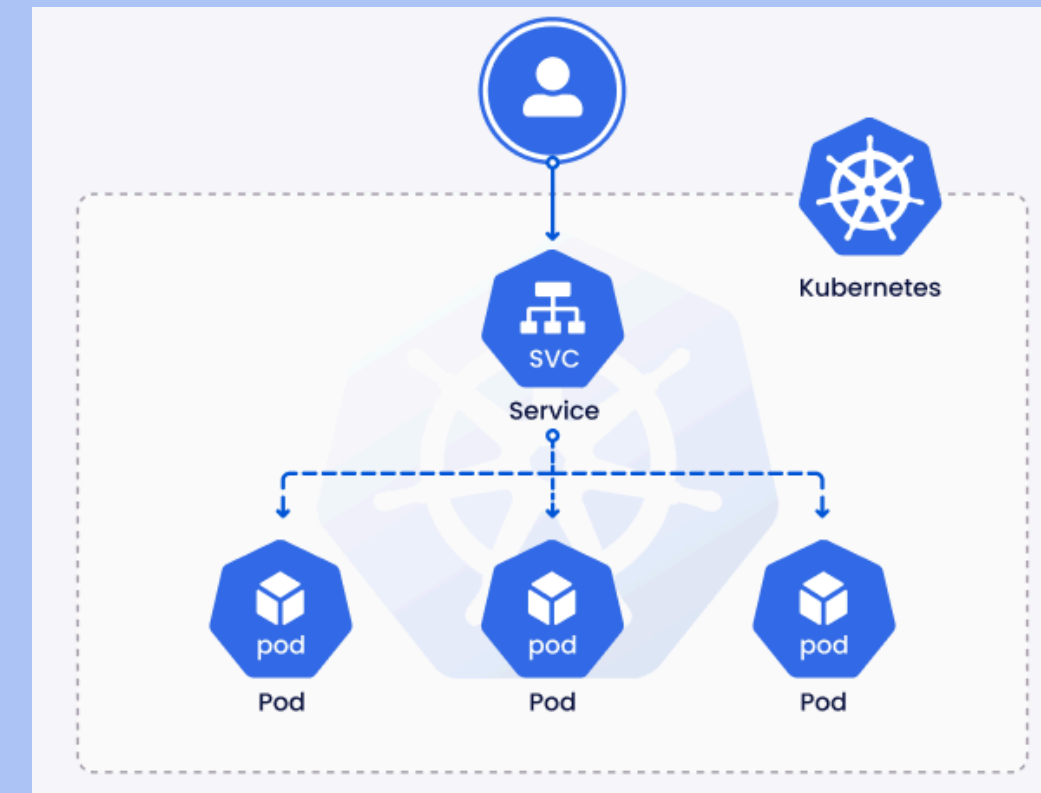
CONTROL PLANE ET SERVICE

Control Plane



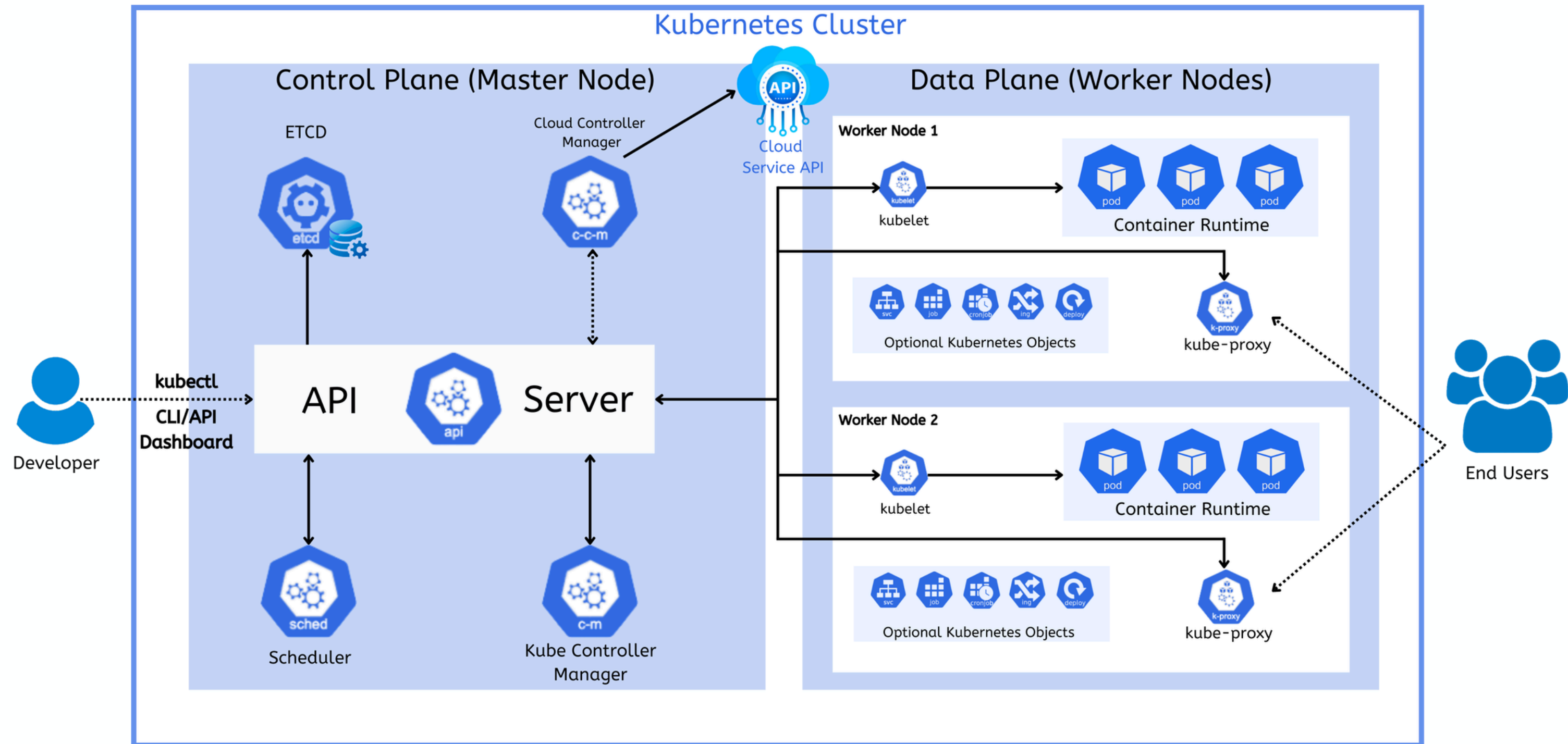
- Gère le **pilotage global** du cluster
- S'exécute sur une ou plusieurs machines
- Se compose de **processus** :
 - **API Server** : point d'entrée du cluster
 - **Scheduler** : décide quel Pod doit être exécuté sur quel Node
 - **etcd** : BDD key-value qui stocke l'état courant du cluster (Pods, Services, Deployments...)
 - **Controller Manager** : surveille l'état du cluster, s'assure qu'il correspond à l'état désiré

Service



- Fournit une **connectivité réseau** stable vers un ou plusieurs Pods
- Nécessaire pour **connecter** un Pod à l'extérieur ou permettre la **communication** entre Pods
- Comme les Pods sont éphémères, le Service fournit une **adresse IP et un DNS stables** indépendamment du cycle de vie des Pods
- Il existe **plusieurs types de Services** selon le besoin d'exposition :
 - **ClusterIP** : accessible uniquement à l'intérieur du cluster (défaut)
 - **NodePort** : expose le service sur un port de chaque Node
 - **LoadBalancer** : expose le service à l'extérieur et distribue le trafic entre les instances de Pods

SCHEMA COMPLET



MANIFEST

Manifest

- **Fichier de configuration** (format YAML)
- Il **décrit l'état désiré** d'une ressource Kubernetes
- kind = **type** de ressource (Deployment ou Statefulset)
- 2 sections importantes :
 - **metadata** : informations essentielles sur la ressource
 - **spec** : déclarer les spécifications ou l'état désiré

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: <deployment name>
  labels:
    app: <a label for the application>
spec:
  replicas: <number of initial replicas>
  selector:
    matchLabels:
      app: <matches the label above>
  template:
    metadata:
      labels:
        app: <label to be given to each pod>
    spec:
      containers:
        - name: <container name>
          image: <the image to be used>
          ports:
            - containerPort: <ports for networking>
```

Deployment vs StatefulSet

→ Ce sont des ressources qui gèrent le déploiement de Pods

Un **Deployment** est conçu pour les applications **stateless** :

- Chaque Pod est **identique** et interchangeable,
- L'ordre de démarrage n'a pas d'importance
- Exemple : API, Web Server

Un **StatefulSet** est pour les applications **stateful** :

- Chaque Pod a une identité stable et persistante (nom, **stockage**)
- **L'ordre** de démarrage est garanti
- Exemple : Base de données

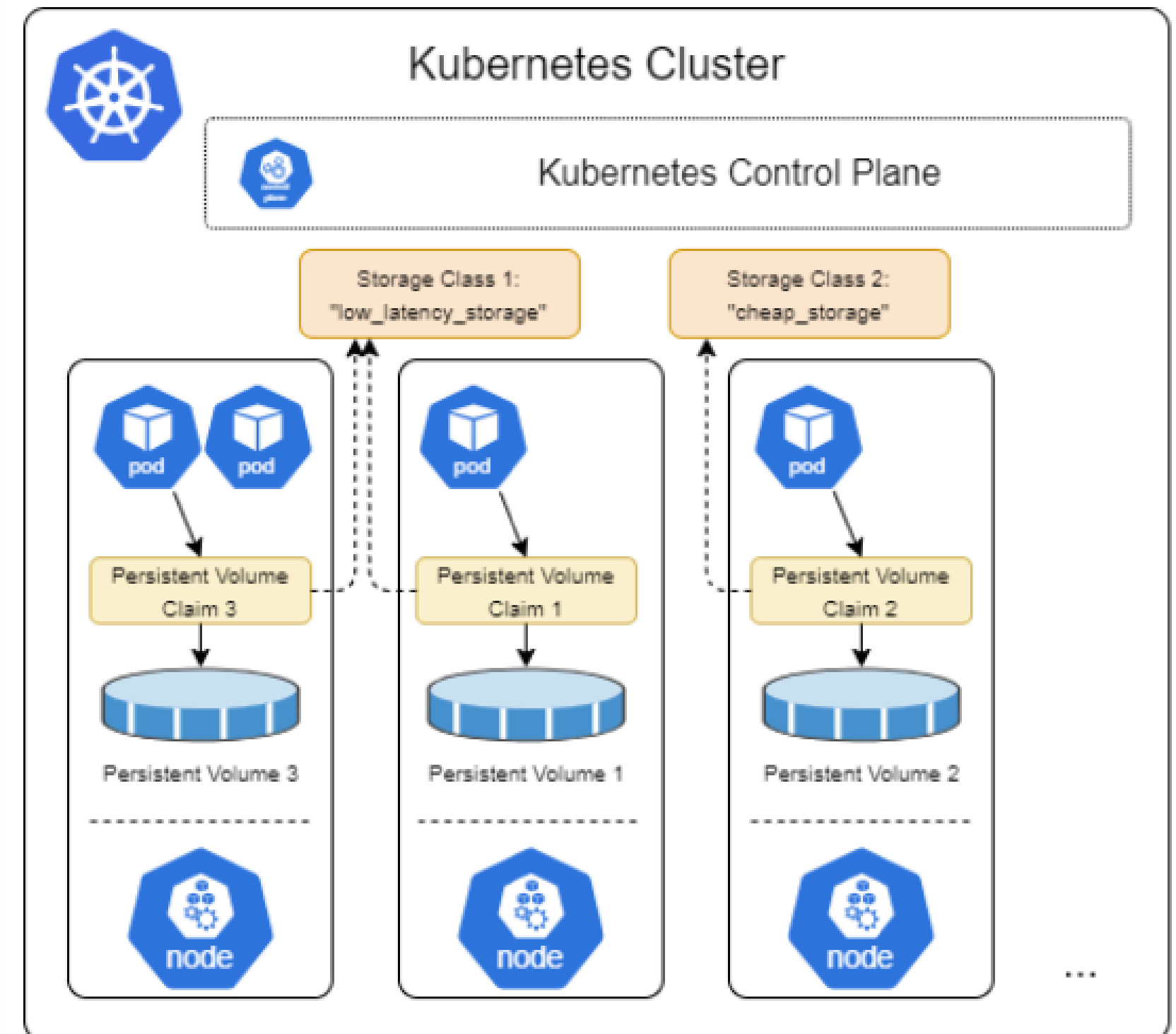
STORAGE

Pourquoi?

- Par défaut, le stockage d'un Pod est éphémère
 - Si le Pod meurt, les données sont perdues.
- Le storage Kubernetes permet de persister les données indépendamment du cycle de vie des Pods.

Les concepts de Storage

- PersistentVolume (PV) : **unité de stockage** provisionnée dans le cluster
- PersistentVolumeClaim (PVC) : **demande de stockage** émise par un Pod — il réclame un PV correspondant à ses besoins
- StorageClass (SC) : définit un **type de stockage** disponible et permet le provisionnement automatique de PV à la demande



EXEMPLES DANS MANIFEST

Pod with PersistentVolume

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  ...
  volumeMounts:
  - name: pv-mydata
    mountPath: /mydata
  volumes:
  - name: pv-mydata
    persistentVolumeClaim:
      claimName: datacamp-pvc
```

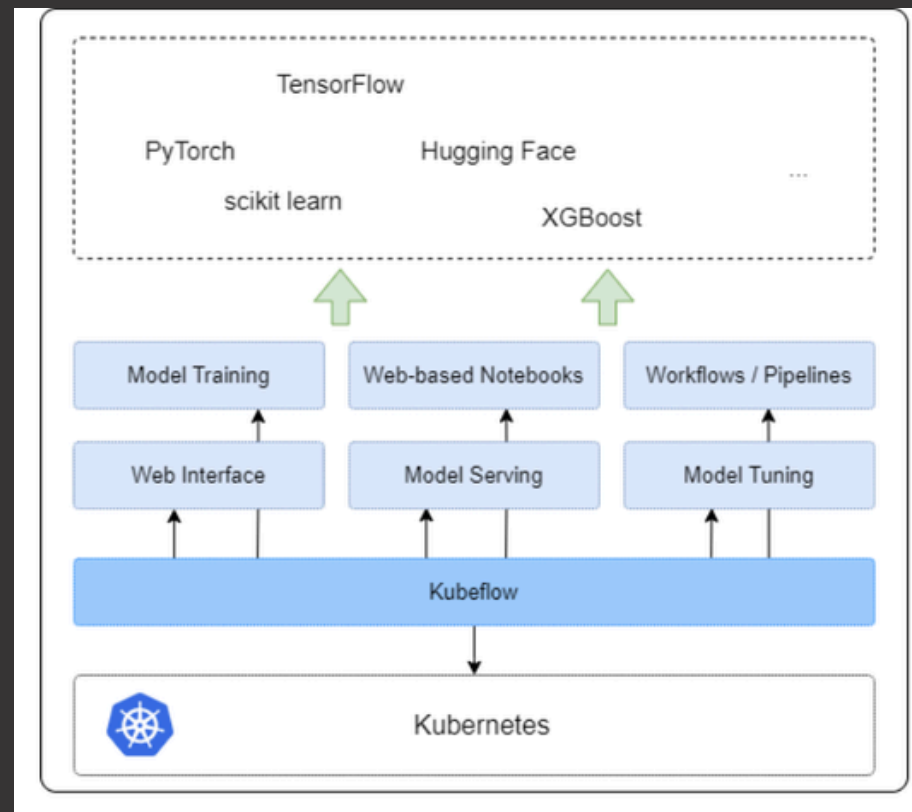
PersistentVolumeClaim with StorageClass

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: datacamp-pvc
spec:
  storageClassName: "standard"
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

CAS D'UTILISATION

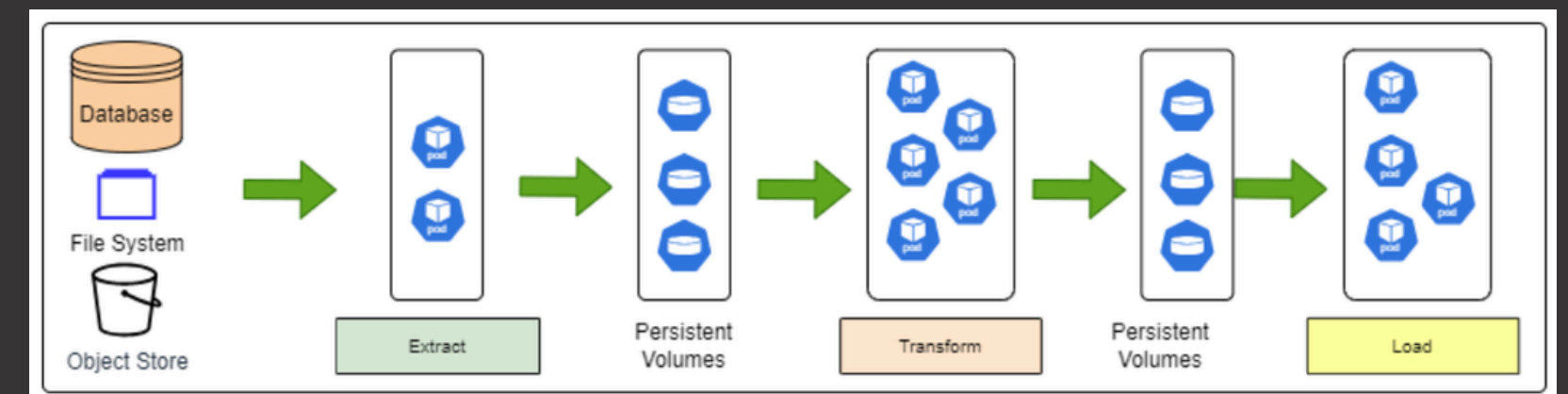
Kubernetes n'est pas qu'un outil DevOps, il peut servir d'infrastructures pour la Data et l'IA

MLOPS



- Permet de **déployer** et de **gérer** des modèles ML en prod
- **Scale** automatiquement les ressources (CPU/GPU) selon les besoins d'entraînement
- Chaque étape (entraînement, test, déploiement) tourne dans son propre **conteneur isolé**
- **Kubeflow** (orchestre tout le cycle de vie d'un modèle)

DATA PIPELINE



- Permet d'exécuter des **pipelines** de traitement de données (ETL) de façon automatisée et fiable
- Chaque étape du pipeline ETL (Extract, Transform, Load) s'exécute dans des **Pods dédiés**
- Les données sont conservées entre chaque étape grâce au **stockage persistant** (PV/PVC)
- S'adapte automatiquement au volume de données à traiter

PRÉSENTATION DE L'ATELIER

1.1 - PRISE EN MAIN

- Démarrer un **cluster** K8s local avec Docker Desktop
- Déployer et inspecter ses premiers **Pods**
- Self-healing & scaling avec les **Deployments**
- Isoler des environnements avec les **Namespaces**
- Exposer une application avec un **Service** NodePort

1

2

3

Au choix

1.2 - DÉPLOIEMENT

- Porter une **architecture** Docker Compose vers Kubernetes
- Déployer **FastAPI** + Streamlit sur K8s
- **Connecter** les services via le DNS interne
- Rolling Update · Blue/Green · Canary

1.3 - AVANCÉES (BONUS)

- ConfigMaps · Secrets · PersistentVolumeClaims
- Planifier des tâches avec les **CronJobs**
- Déployer un **stack** IA (MCP + PydanticAI)
- (À faire si vous avez terminé les 4 TDs)

2 - DATA PIPELINE

- Déployer un objet-store **MinIO** compatible S3
- Ingérer des données météo via un **CronJob**
- Transformer les données avec **dbt** en Job K8s
- Exposer les résultats via **FastAPI**
- (Bonus) Supervision avec Prometheus + Grafana

3 - MLOPS

- Déployer **PostgreSQL** avec stockage persistant
- Tracker les expériences ML avec **MLflow**
- **Entraîner** et enregistrer un modèle diabète
- Servir le modèle via **FastAPI** + interface **Streamlit**
- Auto-scaling avec **HPA**

COMMANDES KUBECTL ESSENTIELLES

KUBECTL EST UN OUTIL DE LIGNE DE COMMANDE PERMETTANT D'INTÉRAGER AVEC KUBERNETES

COMMANDE	DESCRIPTION
<code>kubectl get pods</code>	Lister tous les Pods
<code>kubectl describe pod <name></code>	Afficher les détails d'un Pod (état, événements, erreurs)
<code>kubectl logs <pod></code>	Afficher les logs d'un Pod
<code>kubectl logs -f <pod></code>	Suivre les logs en temps réel
<code>kubectl exec -it <pod> -- bash</code>	Ouvrir un terminal interactif dans un Pod
<code>kubectl apply -f file.yaml</code>	Appliquer un manifest YAML (créer ou mettre à jour)
<code>kubectl delete pod <name></code>	Supprimer un Pod
<code>kubectl scale deployment <name> --replicas=5</code>	Modifier manuellement le nombre de réplicas
<code>kubectl get all</code>	Lister tous les objets du cluster (Pods, Services, Deployments...)
<code>kubectl port-forward <pod> 8080:80</code>	Rediriger un port local vers un port du Pod

PASSONS À LA PRATIQUE !

