ASTIOLAB

INTRODUCTION À L'ORCHESTRATION ET L'ÉCOSYSTÈME KUBERNETES

PLAN DU COURS

- Cours :
  - Rappel : Intérêt de l’orchestration
  - Comprendre le vocabulaire de K8S
  - Scalabilité et résilience
  - Comment déployer une application
- Pratique :
  - Monter son premier cluster Kubernetes et déployer des applications

INTÉRÊT DE L'ORCHESTRATION

POURQUOI L'ORCHESTRATION EST UN STANDARD ?

RAPPEL : UN ORCHESTRATEUR, C'EST QUOI ?

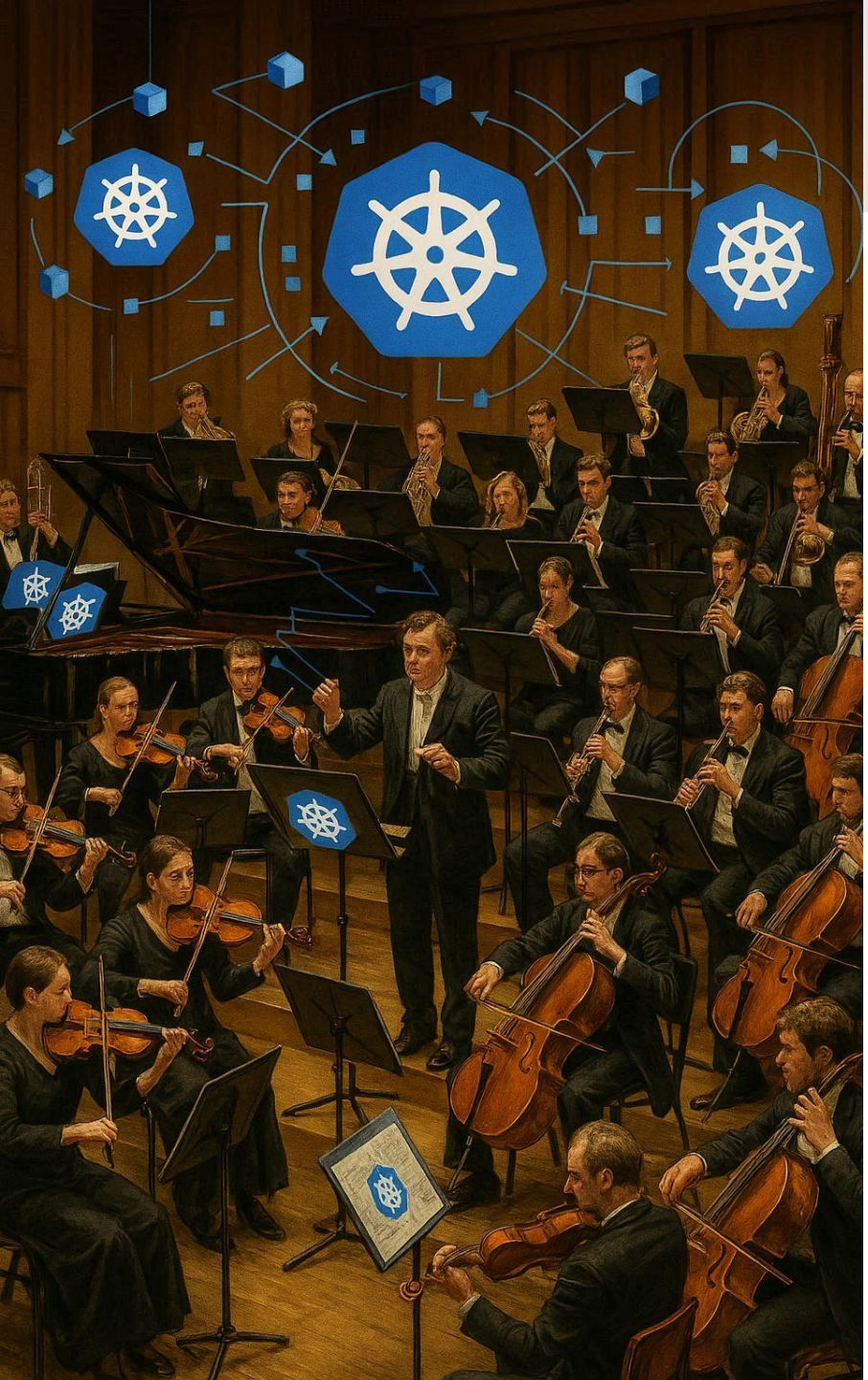
- Outil essentiel pour **automatiser, gérer et optimiser** le déploiement, la mise à l'échelle et l'exploitation d'applications conteneurisées, surtout dans des environnements complexes ou à grande échelle.

RAPPEL : UN ORCHESTRATEUR, C'EST QUOI ?

- Exemples de features possibles :
  - Gestion Automatique des Conteneurs (Déploiement / Redémarrage automatique / Mise à jour sans interruption)
  - Mise à l'Échelle (Scaling)
  - Équilibrage de Charge et Disponibilité
  - Gestion des Réseaux et du Stockage
  - Gestion des Configurations et des Secrets
  - Surveillance et Logging
  - Portabilité et Environnements Multi-Cloud

RAPPEL : UN ORCHESTRATEUR, C'EST QUOI ?

- Sans orchestration, la gestion des conteneurs à grande échelle devient :
  - **Fastidieuse** (tout est manuel).
  - **Peu scalable** (impossible de gérer des centaines de conteneurs).
  - **Peu résiliente** (pas de reprise automatique après une panne).
  - **Risquée** (erreurs humaines, sécurité fragile).



RAPPEL : EXEMPLES D'ORCHESTRATEURS

kubernetes

portainer.io

HashiCorp Nomad

OPENSHIFT

RANCHER BY SUSE

docker SWARM

KUBERNETES EN QUELQUES MOTS

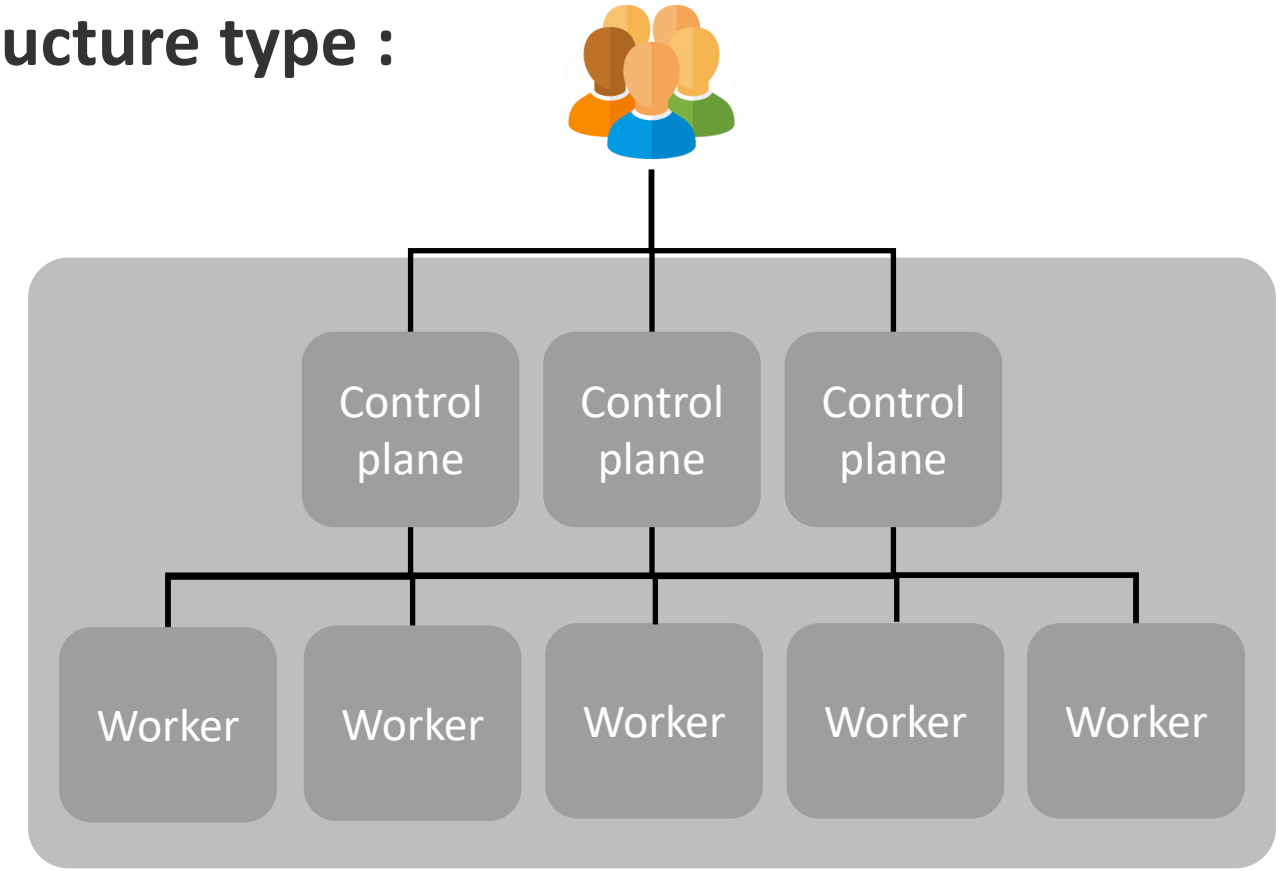
- **Historique :**
  - conçu à l'origine par Google (2014-2015) et désormais maintenu par la Cloud Native Computing Foundation (CNCF)

- **Composé de nœuds :**
  - Deux types de nœuds (nodes) :
    - Control Planes (Masters)
    - Workers (Slaves)

- **Un cluster :**
  - composé de nœuds workers (physiques ou virtuels) gérés par un ou plusieurs nœuds control planes

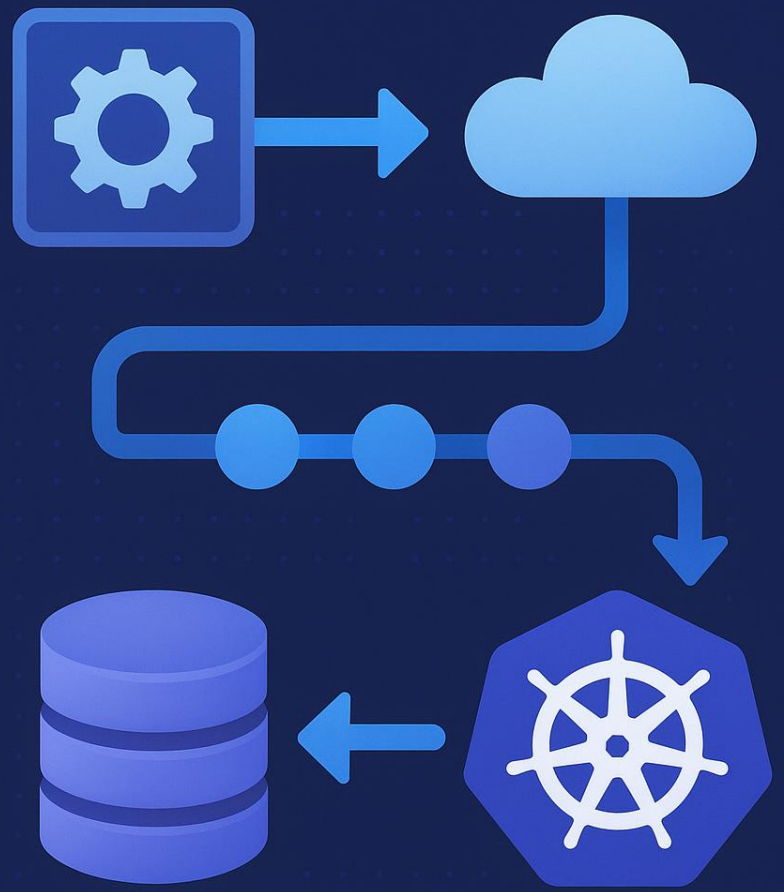
KUBERNETES EN QUELQUES MOTS

- **Schéma d’infrastructure type :**



KUBERNETES EN QUELQUES MOTS

- **Recommandations :**
  - < 110 pods par nœuds
  - < 5000 nœuds
  - < 150,000 pods au total
  - < 300,000 conteneurs au total



MÉTHODES D'INSTALLATION

- Tests / développement (cluster 1 nœud max) :
  - Minikube
  - Kind
- Distributions légères :
  - K3S
  - MicroK8S
- Pour de la production :
  - Kubeadm
  - Rancher
  - Openshift

COMPRENDRE LE VOCABULAIRE DE K8S

L'ENCYCLOPÉDIE DE K8S

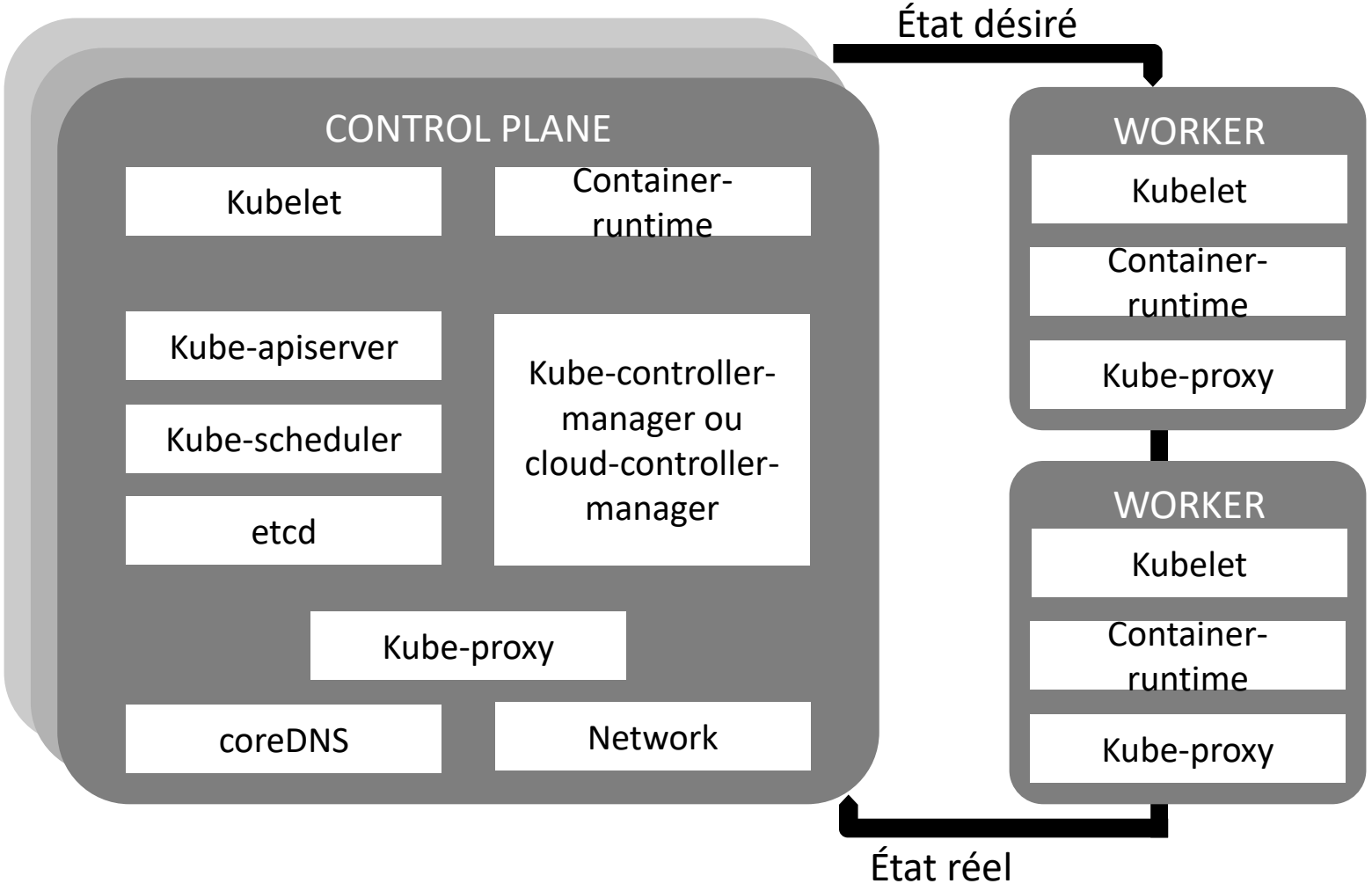
LES PRINCIPAUX OBJETS DE KUBERNETES

Nom	Description
Pod	Unité d'exécution de base d'une application Kubernetes. Représente des process en cours d'exécution dans votre cluster.
Service	Un moyen abstrait d'exposer une application s'exécutant sur un ensemble de pods en tant que service réseau. K8s attribue aux pods leurs propres adresses IP et un nom DNS unique pour un ensemble de pods, et peut équilibrer la charge entre eux.
Volume	Résout la volatilité et le partage de fichiers entre instances de conteneurs dans le même pod.
Namespace	Kubernetes prend en charge plusieurs clusters virtuels sauvegardés par le même cluster physique. Ces clusters virtuels sont appelés Namespace.

LES PRINCIPAUX OBJETS DE KUBERNETES

Nom	Description
Deployments	Fournit des mises à jour déclaratives pour Pods et ReplicaSets.
Replicaset	Permet de maintenir un ensemble stable de Pods à un moment donné. Cet objet est souvent utilisé pour garantir la disponibilité d'un certain nombre identique de Pods.
StatefulSet	StatefulSet est l'objet de l'API de charge de travail utilisé pour gérer des applications avec état (*stateful*). Gère le déploiement et la mise à l'échelle d'un ensemble de Pods, *et fournit des garanties sur l'ordre et l'unicité* de ces Pods.
DaemonSet	Un DaemonSet garantit que tout ou partie des noeuds exécutent une copie du pod. Lorsque des nœuds sont ajoutés au cluster, des pods leur sont ajoutés. Lorsque les nœuds sont supprimés du cluster, ces pods sont détruits. La suppression d'un DaemonSet nettoiera les pods qu'il a créés.
Job	Un job crée un ou plusieurs pods et garantit qu'un nombre spécifié d'entre eux se terminent avec succès. Lorsque le nombre spécifié de réussites est atteint, la tâche (c-à-d le travail) est terminée. La suppression d'un job nettoie les pods créés.

COMPOSANTS DE K8S



RÔLE DES COMPOSANTS

- **Kube-apiserver**
  - Composant qui expose l'API Kubernetes. Il s'agit du front-end pour le plan de contrôle Kubernetes.
- **Etcd-master**
  - Base de données clé-valeur consistante et hautement disponible utilisée comme mémoire de sauvegarde pour toutes les données du cluster.
- **Kube-scheduler**
  - Composant qui surveille les pods nouvellement créés et non assignés à un noeud. Il sélectionne le noeud sur lequel ils vont s'exécuter.
- **CoreDNS**
  - Service de résolution et de nommage interne
- **Kube-proxy**
  - Maintient les règles réseau sur les noeuds
  - Utilise la couche de filtrage de paquets du système d'exploitation s'il y en a une et qu'elle est disponible
  - Transmet le trafic lui-même

RÔLE DES COMPOSANTS

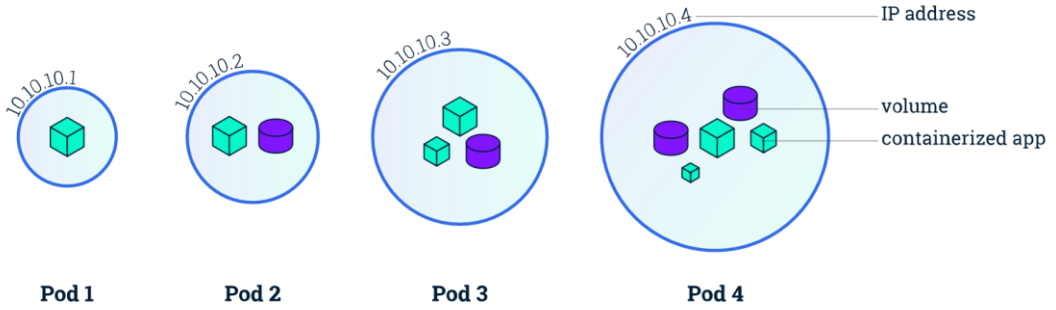
- **Kubelet**
  - service responsable du fonctionnement de l'ensemble des sous-composants k8s sur le noeud
- **Cloud-controller-manager**
  - Exécute les contrôleurs qui interagissent avec les fournisseurs cloud.
- **Kube-controller-manager**
  - Composant du master qui exécute les contrôleurs :
    - Node Controller
    - Replication Controller
    - Endpoints Controller
    - Service Account & Token Controllers
  - → Un seul binaire pour tout ça

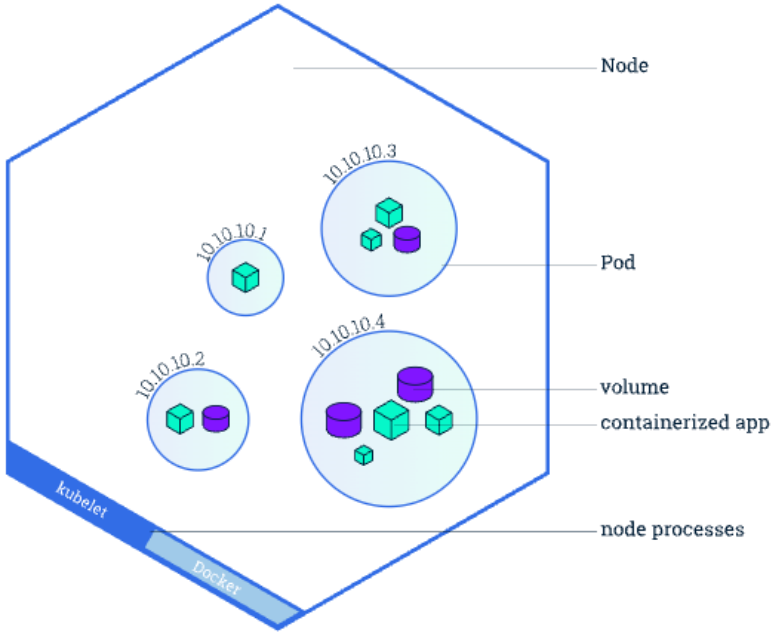
AUTRES COMPOSANTS

- Il est possible d'ajouter des composants pour étendre les fonctionnalités du cluster.
- Exemple :
  - Gestion du réseau
    - Cilium
    - Calico
  - Stockage
    - Minio
    - Rook
  - Virtualisation via Kubernetes
    - Kubevirt
  - Ingress
    - Traefik
    - Cilium

LES PODS ET LEURS RÔLES

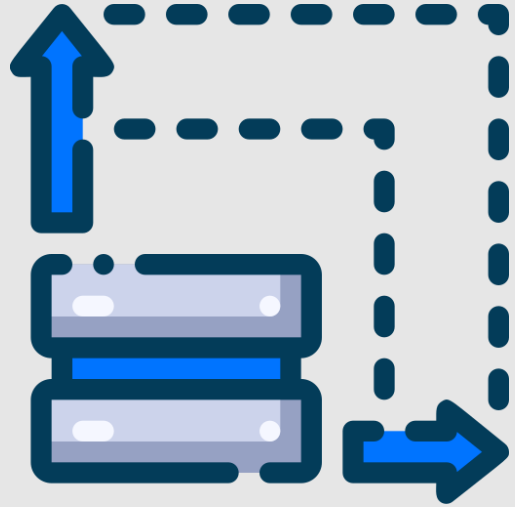
- Éléments d'exécution créés par kubelet
- Exécutent le CRI (container runtime interface) pour matérialiser les conteneurs applicatifs
- Détiennent les ressources hardware allouées par le manifest yaml
- Sont les plus petites unités disponibles dans k8s
- Fournissent toutes les ressources partagées pour leurs conteneurs constitutifs
- Les conteneurs d'un pod sont forcément sur le même nœud





SCALABILITÉ ET RÉSILIENCE

PAS DE PANIQUE, C’EST AUTOMATIQUE



C'EST QUOI ?

- Kubernetes (K8s) est conçu pour **scaler** (ajuster la taille des applications en fonction de la charge) et assurer la **résilience** (maintenir la disponibilité malgré les pannes).

SCALABILITÉ

- La scalabilité permet d'adapter dynamiquement les ressources en fonction de la demande.
- Plusieurs méthodes:
  - Horizontal Pod Autoscaler (HPA): Ajouter/supprimer des **instances** (pods) d'une application.
  - Vertical Pod Autoscaler (VPA): Augmenter/diminuer les **ressources** (CPU, mémoire) d'un pod.
  - Cluster Node Autoscaler: Ajouter/supprimer des **nœuds** (nodes) au cluster.

SCALABILITÉ - HPA

- Présent par défaut sur K8S
- Basé sur des **métriques** collectées par Metrics Server (ou Prometheus)
- A définir pour chaque application
- Ici on check le CPU, si on atteint 50%, on ajoute un pod
- Il est possible aussi de passer par des métriques applicatives via prometheus (ex : requêtes /seconde)

```yaml
### autoscaling HPA
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: mon-app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mon-app
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50

---
```

SCALABILITÉ - VPA

- **Attention** : Nécessite une installation séparée (non inclus par défaut dans K8s).
- permet d'optimiser l'usage des ressources à l'intérieur d'un pod unique.
- le VPA peut être utile pour la **phase d'observation et de recommandation** pour appliquer ensuite les valeurs manuellement.

- Installation de VPA:

```
git clone https://github.com/kubernetes/autoscaler.git
cd autoscaler/vertical-pod-autoscaler
./hack/vpa-up.sh
```

```yaml
### autoscaling VPA
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: mon-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment
    name:       mon-app
  updatePolicy:
    updateMode: "Recreate"

---
```

SCALABILITÉ - VPA

- Plusieurs modes de fonctionnement *(updateMode)*:
  - **Auto:** Applique automatiquement les recommandations de ressources en redémarrant les pods si nécessaire. (non recommandé et surement supprimé dans les prochaines versions)
  - **Recreate:** Ne déclenche pas lui-même la recréation des pods, mais applique les nouvelles valeurs lors du redémarrage.
  - **Initial:** Applique les recommandations seulement lors de la création initiale d'un pod
  - **Off:** Mode d'observation. Le VPA ne change rien mais continue de collecter des métriques et de produire des recommandations.

HPA OU VPA ?

- **Quand choisir le HPA ?**
  - Ton application peut être dupliquée sans risque (stateless).
  - Tu veux répartir la charge sur plusieurs instances.
  - Tu as besoin de réactivité face aux pics de trafic.
  - Tu utilises des métriques personnalisées (ex: Prometheus).

- **Limites du HPA**
  - Inadapté pour les applications stateful (ex: bases de données).
  - Peut entraîner des coûts supplémentaires si mal configuré (trop de pods).

- **Quand choisir le VPA ?**
  - Ton application est stateful et ne peut pas être dupliquée.
  - Tu veux optimiser l'utilisation des ressources sans ajouter de pods.
  - Tes pods ont des besoins en CPU/mémoire difficiles à prédire.
  - Tu veux éviter de redimensionner manuellement les ressources.

- **Limites du VPA**
  - Nécessite une installation séparée (non inclus par défaut dans Kubernetes).
  - Peut entraîner des redémarrages de pods (si updateMode: Auto).
  - Moins réactif que le HPA pour absorber des pics de charge soudains.
  - Ne résout pas les problèmes de haute disponibilité (un seul pod = SPOF).

SCALABILITÉ – CLUSTER NODE AUTOSCALER

- Surtout Intégré aux solutions cloud (AWS, GCP, Azure,…)
- Ajuste automatiquement le nombre de nœuds dans le cluster en fonction des besoins.
- Compliqué de faire ça sur du « on-premise »…

RÉSILIENCE

- Garantit que l'application reste disponible malgré les pannes (nœuds, pods, réseau).
- Plusieurs outils à dispositions :
  - Liveness, Readiness et Startup Probes
  - Pod Disruption Budget (PDB)
  - Anti-Affinity pour la haute disponibilité
  - PersistentVolumes (PV) et PersistentVolumeClaims (PVC)
  - StatefulSets pour les applications stateful

LIVENESS, READINESS ET STARTUP PROBES

- Kubernetes utilise des probes (sondes) pour **surveiller l'état de santé** de nos conteneurs et prendre des actions automatiques en fonction de leur statut.
- Leur rôle:
  - **Liveness probe:** Vérifie si le conteneur fonctionne correctement (pas de crash ou de blocage) et redémarre le conteneur en cas d ’échec
  - **Readiness probe:** Vérifie si le conteneur est prêt à recevoir du trafic sinon retire le pod du Service (plus de trafic envoyé).
  - **Startup probe:** Vérifie si le conteneur a **démarré avec succès** (pour les applications lentes à démarrer).

POD DISRUPTION BUDGET (PDB)

- Limite le nombre de pods qui peuvent être **indisponibles** pendant une maintenance (ex: mise à jour du nœud).

- En utilisant cette fonctionnalité et en l'ayant correctement configurée, il est possible de:
  - garantir la haute disponibilité,
  - minimiser le risque de temps d'arrêt des applications.

ANTI-AFFINITY

- Évite que deux pods d'une même application soient sur le **même nœud** (pour tolérer les pannes de nœud).
- Permet d'influencer la décision du scheduler et placer vos pods de manière plus intelligente.

PERSISTENTVOLUMES ET PERSISTENTVOLUMECLAIMS

- Les PV et PVC permettent de **conserver les données même après la suppression d'un pod**, ce qui est essentiel pour les applications stateful (bases de données, caches, etc.).

- Un **PersistentVolume (PV)** est une **ressource de stockage** dans le cluster, provisionnée par un administrateur ou dynamiquement via un **StorageClass**. Il représente un morceau de stockage physique (ex: disque AWS EBS, NFS, Ceph).

- Un **PersistentVolumeClaim (PVC)** est une **demande de stockage** par un utilisateur. Il est lié à un PV qui répond à ses exigences (taille, mode d'accès, classe de stockage).

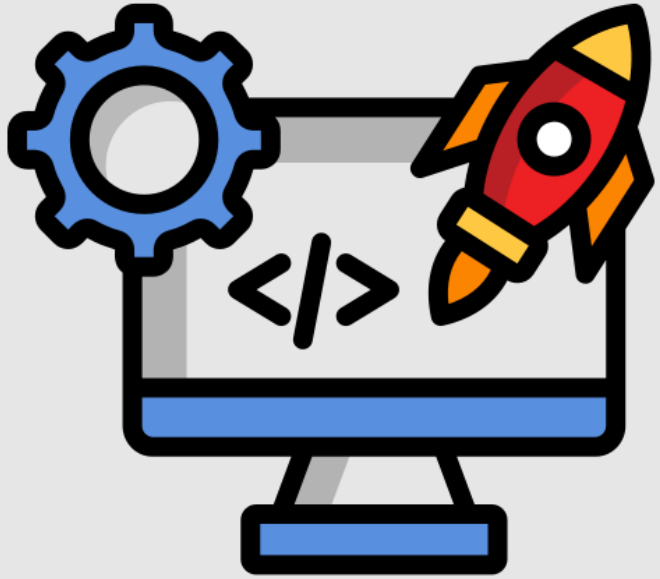
STATEFULSETS

- Un StatefulSet permet de :
  - Déployer et scaler un ensemble de pods, comme un Deployment.
  - Garantir l'ordre et l'unicité des pods :
    - Chaque pod a un identifiant persistant (ex: mon-pod-0, mon-pod-1), même après un redémarrage.
    - Les pods sont créés et supprimés de manière séquentielle (pas en parallèle).
  - Différence clé avec un Deployment :
    - Les pods ne sont pas interchangeables : leur identité est conservée (ex: pour le stockage ou le réseau).
    - Idéal pour les applications stateful (bases de données, Kafka, etc.) :
      - Permet d’associer facilement des volumes persistants (PV/PVC) à des pods spécifiques, même après un échec ou un redémarrage.

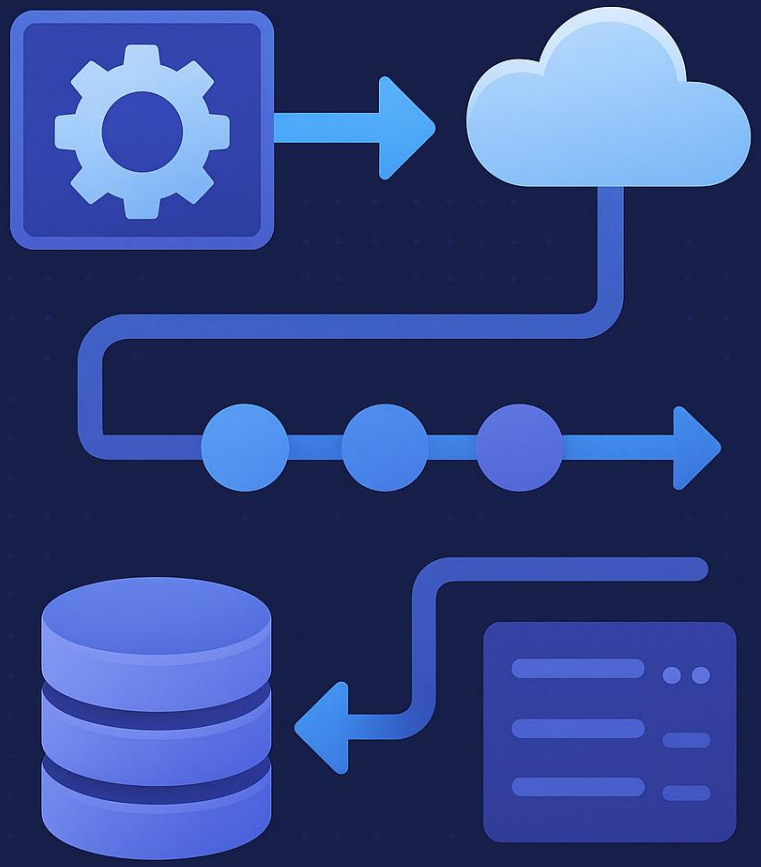
Cas d'usage : Utilisé quand une application a besoin de stabilité d'identité (ex: noms de pods fixes) et de stockage persistant (ex: bases de données).

COMMENT DÉPLOYER UNE APPLICATION

DÉPLOYEZ-LES TOUS !



APPLICATION DEPLOYMENT METHODS



MÉTHODES DE DÉPLOIEMENTS

- Déploiement avec **kubectl** et fichiers YAML/JSON (manifests)
- **Helm** (Pour créer des "paquets" de manifests (charts) avec des valeurs paramétrables.)
- **Kustomize** (Pour personnaliser des manifests sans les modifier)

LES MANIFESTS

- Fichiers décrivant l'état souhaité des ressources que l'on veut déployer ou configurer.
- Un manifest YAML est composé de **champs clés** organisés hiérarchiquement.
- Pour exécuter ce manifest :

```
kubectl apply -f mon-fichier.yaml
```

```
 GNU nano 8.4
apiVersion: apps/v1          # Version de l'API Kubernetes utilisée
kind: Deployment             # Type de ressource (Pod, Service, Deployment, etc.)
metadata:                    # Métadonnées (nom, labels, etc.)
  name: mon-application
  labels:
    app: mon-app
spec:                        # Spécification de la ressource
  replicas: 3                # Nombre de réplicas pour un Deployment
  selector:                  # Sélecteur pour associer des Pods
    matchLabels:
      app: mon-app
  template:                  # Template pour créer les Pods
    metadata:
      labels:
        app: mon-app
    spec:                    # Spécification du Pod
      containers:
      - name: mon-conteneur
        image: nginx:latest
        ports:
        - containerPort: 80
```

LES MANIFESTS : CHAMPS PRINCIPAUX

- **apiVersion**
  - Indique la version de l'API Kubernetes utilisée pour créer la ressource.
  - Exemples :
    - **v1** pour les Pods, Services, ConfigMaps.
    - **apps/v1** pour les Deployments, StatefulSets.
    - **networking.k8s.io/v1** pour les Ingress.
- **kind**
  - Définit le type de ressource.
  - Exemple : Pod, Service, Deployment, ConfigMap, Ingress, etc.
- **metadata**
  - Contient des infos comme le **nom** de la ressource, ses **labels** et son **namespace**

LES MANIFESTS : CHAMPS PRINCIPAUX

- **spec**
  - Spécification de la ressource. Varie selon le **kind**

```yaml
#deployment
spec:                          # Spécification de la ressource
  replicas: 3                  # Nombre de réplicas pour un Deployment
  selector:                    # Sélecteur pour associer des Pods
    matchLabels:
      app: mon-app
  template:                    # Template pour créer les Pods
    metadata:
      labels:
        app: mon-app
    spec:                      # Spécification du Pod
      containers:
      - name: mon-conteneur
        image: nginx:latest
        ports:
        - containerPort: 80

## pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80


# service
spec:
  selector:
    app: mon-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: NodePort        # ClusterIP, NodePort ou LoadBalancer
```

EXEMPLES DE MANIFESTS

```yaml
### Pod
apiVersion: v1
kind: Pod
metadata:
  name: mon-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80

---
```

```yaml
### Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mon-app
  template:
    metadata:
      labels:
        app: mon-app
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80

---
```

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


---
```

EXEMPLES DE MANIFESTS

```yaml
### ConfigMap
apiVersion: v1
kind: ConfigMap
metadata:
  name: ma-config
data:
  config.ini: |
    [section]
    param1 = valeur1
    param2 = valeur2


---
```

```yaml
### Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mon-ingress
spec:
  rules:
  - host: mon-domaine.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: mon-service
            port:
              number: 80

---
```

KUBECTL : DÉPLOYER UN MANIFEST

- **Déploiement / suppression**
  - *kubectl apply -f MON-FICHIER.yaml*
  - *Kubectl delete -f MON-FICHIER.yaml*
- **Tester ses fichiers YAML**
  - *kubectl apply --dry-run=client -f MON-FICHIER.yaml*
- **Vérifier notre déploiement**
  - *kubectl get pods (-n MON-NAMESPACE)*
  - *kubectl get deployments*
  - *kubectl get services*
  - *Kubectl get all --all-namespaces*

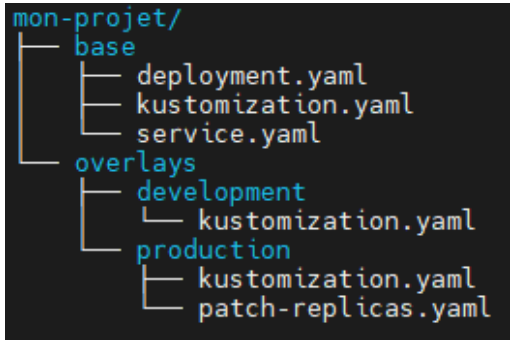
KUSTOMIZE

- **Kustomize est un outil natif intégré à Kubectl** qui permet de personnaliser des fichiers YAML **sans utiliser de templates**. Contrairement à Helm, qui repose sur un système de chartes avec des valeurs dynamiques, Kustomize utilise des **patches et des overlays** pour modifier des configurations existantes.

- L'idée principale est simple : au lieu de dupliquer des fichiers YAML, on applique des modifications ciblées à une **base commune**. Cela permet de gérer plusieurs environnements sans complexifier la maintenance des manifests.

KUSTOMIZE : CONCEPTS CLEFS

- **Base et Overlays**
  - **Base** : Un dossier contenant les manifests Kubernetes de base (ex: deployment.yaml, service.yaml).
  - **Overlay** : Un dossier qui **surcharge** ou **étend** la base pour un environnement spécifique (ex: production/, development/).



- **Fichier kustomization.yaml**
  - C’est le cœur de Kustomize. Il définit :
  - Les **ressources** à inclure (fichiers YAML).
  - Les **transformations** à appliquer (ex: ajouter des labels, modifier des images).
  - Les **générateurs** (ConfigMaps, Secrets).
  - Les **sources distantes** (pour inclure des manifests depuis un dépôt Git).

```
user@debian:~$ cat mon-projet/base/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
  - deployment.yaml
  - service.yaml

user@debian:~$ cat mon-projet/overlays/production/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- ../../base  # Référence à la base

# Surcharger le nombre de réplicas
patches:
- path: patch-replicas.yaml
  target:
    kind: Deployment
    name: mon-app

# Ajouter des labels communs
labels:
- includeSelectors: true
  pairs:
    env: production
```

HELM

- **gestionnaire de paquets** pour K8S (équivalent de apt pour Ubuntu ou npm pour Node.js)
- Il permet de :
  - Déployer des applications complexes
  - Gérer les dépendances entre les ressources Kubernetes.
  - Personnaliser les déploiements avec des valeurs paramétrables.
  - Versionner et partager des configurations via des charts.

HELM : LES CHARTS

- Paquet Helm qui contient :
  - Des **manifests Kubernetes** (fichiers YAML pour Deployments, Services, etc.).
  - Un fichier **Chart.yaml** (métadonnées du chart : nom, version, etc.).
  - Un dossier **templates/** (manifests dynamiques générés avec des variables).
  - Un fichier **values.yaml** (valeurs par défaut pour personnaliser le déploiement).
  - ***En option :*** des dépendances (charts/), des hooks, etc.

DÉPLOIEMENT D'UNE CHART SIMPLE

- Ajout du dépôt de grafana (pour déployer grafana…) et update
  - helm repo add grafana https://grafana.github.io/helm-charts
  - helm repo update
- Installation de grafana
  - helm install my-grafana grafana/grafana
- Afficher les notes de l'application (en option)
  - helm get notes my-grafana

C’EST L’HEURE DE LA PRATIQUE