Kubernetes

Sommaire

Sommaire

Sommaire

Introduction à Kubernetes et au déploiement de conteneurs applicatifs

Rappels sur la Containerisation

Qu'est-ce qu'un conteneur ?

- Un conteneur est une unité de déploiement légère qui encapsule une application et toutes ses dépendances
- Contrairement aux machines virtuelles, les conteneurs partagent le noyau du système d'exploitation hôte
- **Avantages** : Portabilité, isolation, efficacité des ressources, démarrage rapide
- **Différence VM vs Conteneur** :
  - VM : Virtualisation complète du matériel + OS complet
  - Conteneur : Virtualisation au niveau OS + partage du noyau

Évolution vers la containerisation

- **Avant** : Applications monolithiques sur serveurs physiques/VMs
- **Maintenant** : Applications microservices dans des conteneurs
- **Pourquoi** : Scalabilité, maintienabilité, déploiement plus rapide

Docker et ses Concurrents

Docker - Le leader du marché

- **Docker Engine** : Runtime de conteneurs le plus populaire
- **Docker Hub** : Registry public pour partager des images
- **Dockerfile** : Script pour construire des images personnalisées
- **Docker Compose** : Orchestration simple pour quelques conteneurs

Les alternatives à Docker

- **Podman** : Alternative sans démon, compatible OCI, plus sécurisé
- **containerd** : Runtime de bas niveau, utilisé par Kubernetes
- **CRI-O** : Runtime spécialement conçu pour Kubernetes
- **LXC/LXD** : Conteneurs système, alternative plus proche des VMs
- **rkt (CoreOS)** : Maintenant abandonné, mais était une alternative populaire

Standards et compatibilité

- **OCI (Open Container Initiative)** : Standards pour les runtimes et images
- **CRI (Container Runtime Interface)** : Interface standard pour Kubernetes

Le Rôle d'un Orchestrateur

Pourquoi avons-nous besoin d'orchestration ?

- **Complexité à l'échelle** : Gérer 1 conteneur vs 1000 conteneurs
- **Défis sans orchestrateur** :
  - Déploiement manuel et chronophage
  - Gestion des pannes difficile
  - Mise à l'échelle manuelle
  - Communication inter-conteneurs complexe
  - Gestion des ressources inefficace

Les principaux orchestrateurs

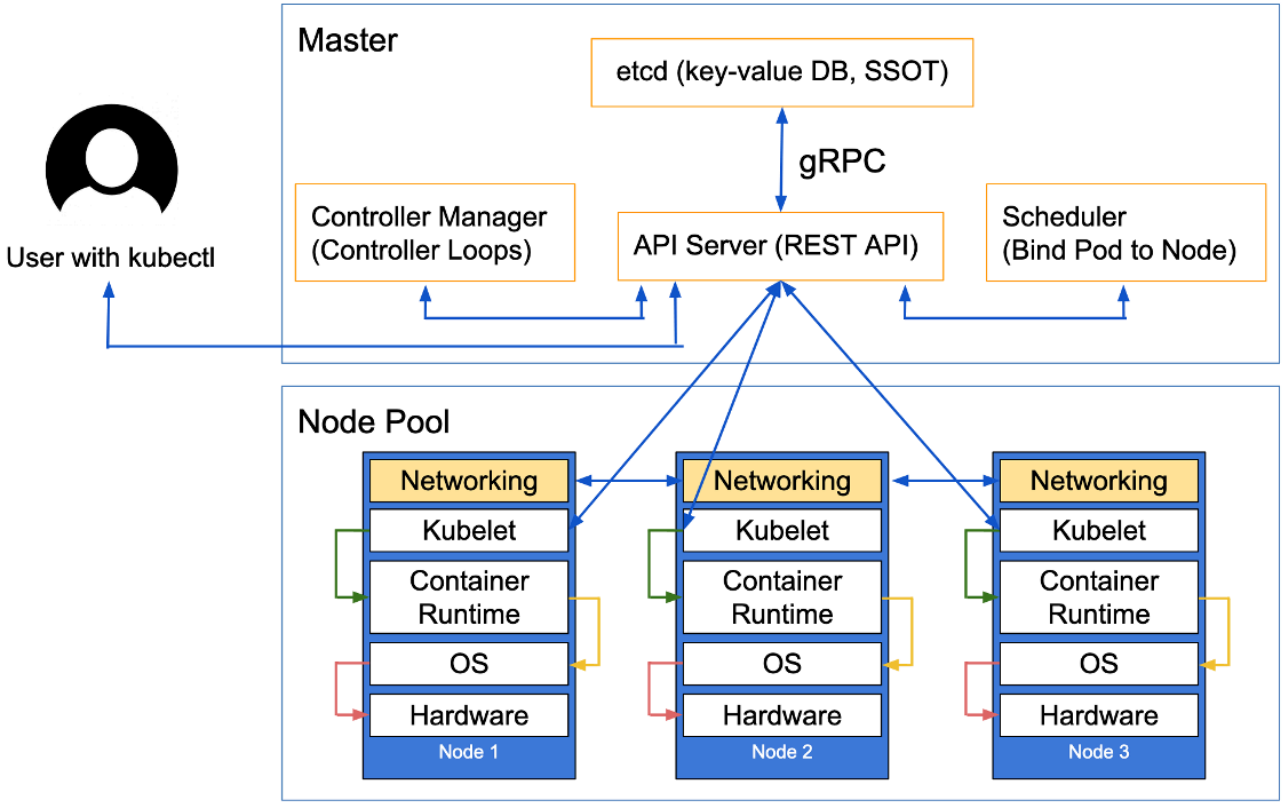
- **Kubernetes** : Le standard de facto, très complet
- **Docker Swarm** : Simple mais moins puissant
- **Apache Mesos** : Pour très grandes infrastructures
- **Amazon ECS** : Service AWS managé

Fonctionnalités attendues d'un orchestrateur

- **Déploiement automatisé** : Lancement et arrêt des conteneurs
- **Mise à l'échelle** : Scale-up/Scale-down automatique
- **Auto-réparation** : Redémarrage des conteneurs défaillants
- **Équilibrage de charge** : Distribution du trafic
- **Découverte de services** : Communication entre composants
- **Gestion des secrets** : Sécurisation des données sensibles
- **Mises à jour** : Déploiement en continu sans interruption
- **Monitoring et logging** : Observabilité des applications

Architecture et Composants Kubernetes

- **Cluster Kubernetes** : Ensemble de machines (nœuds) orchestrant des conteneurs
- **Plan de contrôle (Control Plane)** : Cerveau du cluster
- **Nœuds workers** : Machines qui exécutent les applications



Architecture et Composants Kubernetes

Composants du Plan de Contrôle

A. etcd - La base de données du cluster

- **Rôle** : Stockage clé-valeur distribué et cohérent
- **Contient** : Configuration du cluster, état désiré, secrets
- **Caractéristiques** : Haute disponibilité, consensus Raft
- **Pourquoi critique** : Si etcd tombe, tout le cluster s'arrête

B. API Server (kube-apiserver)

- **Rôle** : Point d'entrée unique pour toutes les opérations
- **Fonctions** :
  - Validation et persistance des objets
  - Authentification et autorisation
  - Interface REST pour kubectl et autres clients
- **Sécurité** : Chiffrement TLS, RBAC

C. Controller Manager (kube-controller-manager)

- **Rôle** : Surveillance et réconciliation de l'état
- **Contrôleurs inclus** :
  - Node Controller : Surveillance des nœuds
  - Deployment Controller : Gestion des déploiements
  - Service Controller : Gestion des services
- **Principe** : Boucle de contrôle continue (état désiré vs état actuel)

D. Scheduler (kube-scheduler)

- **Rôle** : Placement intelligent des pods sur les nœuds
- **Critères de décision** :
  - Ressources disponibles (CPU, RAM)
  - Contraintes de l'application (nodeSelector, affinity)
  - Politiques du cluster
- **Algorithme** : Filtrage puis scoring des nœuds

Architecture et Composants Kubernetes

Composants des Nœuds Workers

A. kubelet - L'agent du nœud

- **Rôle** : Agent qui s'exécute sur chaque nœud worker
- **Responsabilités** :
  - Communication avec l'API Server
  - Gestion du cycle de vie des pods
  - Surveillance de la santé des conteneurs
  - Rapports de statut au plan de contrôle
- **Interface** : Avec le runtime de conteneurs via CRI

B. kube-proxy - Le réseau

- **Rôle** : Gestion du réseau et load balancing
- **Fonctionnalités** :
  - Implémentation des services
  - Distribution du trafic entre les pods
  - Règles iptables/ipvs

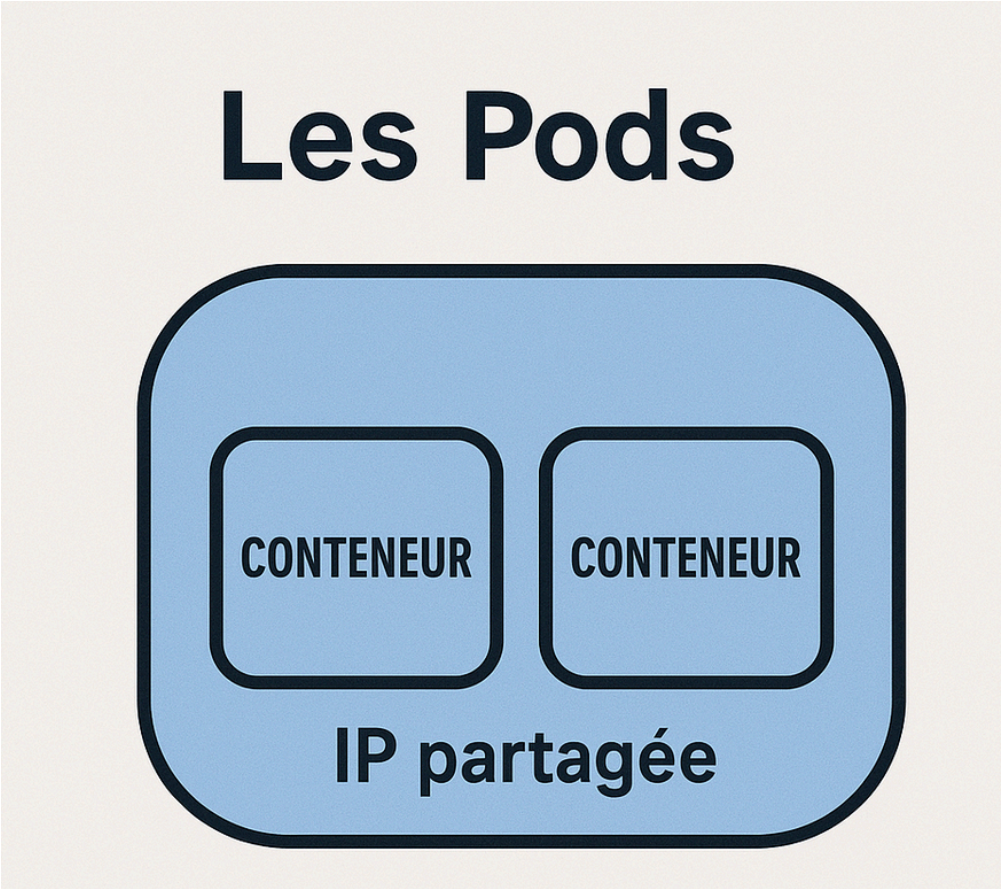
C. Container Runtime

- **Exemples** : containerd, CRI-O, Docker Engine
- **Rôle** : Exécution effective des conteneurs

Introduction aux Concepts Fondamentaux

Les Pods - L'unité de base

- **Définition** : Plus petite unité déployable dans Kubernetes
- **Composition** : Un ou plusieurs conteneurs partageant :
- Même adresse IP
- Même stockage (volumes)
- Même cycle de vie
- **Patterns courants** :
- **Un conteneur par pod** : Le plus fréquent
- **Sidecar** : Conteneur principal + conteneur utilitaire
- **Ambassador** : Proxy pour communication externe
- **Caractéristiques** :
- Éphémère (mortel)
- IP unique dans le cluster
- Pas de redémarrage automatique (rôle des contrôleurs)



Introduction aux Concepts Fondamentaux

Les Répliques et Contrôleurs

A. ReplicaSet

- **Rôle** : Maintenir un nombre désiré de répliques de pods
- **Fonctionnement** :
  - Surveille les pods avec des labels spécifiques
  - Crée/supprime des pods selon le besoin
- **Sélecteur** : Utilise les labels pour identifier ses pods

B. Deployment - Le contrôleur recommandé

- **Avantages** :
  - Gestion des mises à jour progressives (rolling updates)
  - Historique des versions (rollback possible)
  - Stratégies de déploiement configurables
- **Relations** : Deployment → ReplicaSet → Pods

C. Autres types de contrôleurs

- **DaemonSet** : Un pod par nœud (ex: agents de monitoring)
- **StatefulSet** : Pour applications avec état (bases de données)
- **Job** : Exécution ponctuelle
- **CronJob** : Exécution programmée

Introduction aux Concepts Fondamentaux

Les Services - Communication et découverte

A. Problème résolu

- **Pods éphémères** : IPs changent à chaque redémarrage
- **Load balancing** : Distribuer le trafic entre plusieurs pods
- **Découverte** : Comment trouver et contacter une application ?

B. Types de Services

- **ClusterIP** : Accessible uniquement dans le cluster (par défaut)
- **NodePort** : Accessible depuis l'extérieur via port sur chaque nœud
- **LoadBalancer** : Utilise le load balancer du cloud provider
- **ExternalName** : Redirection DNS vers service externe

C. Fonctionnement

- **Sélection** : Utilise les labels pour cibler les pods
- **Endpoints** : Liste automatiquement mise à jour des pods cibles
- **DNS** : Résolution automatique nom-service → IP

Démarrage avec Kubernetes

Le Dashboard Kubernetes

Qu'est-ce que le Dashboard Kubernetes ?

- **Interface graphique web** pour gérer et surveiller un cluster Kubernetes
- **Alternative visuelle** à la ligne de commande kubectl
- **Fonctionnalités** :
  - Visualisation des ressources du cluster
  - Déploiement d'applications via interface graphique
  - Monitoring en temps réel des pods, services, deployments
  - Gestion des secrets, configmaps et volumes
  - Consultation des logs et métriques

Installation du Dashboard

- **Sur cluster local (minikube)** :

```
minikube dashboard
# Ouvre automatiquement le navigateur
```

- **Sur cluster standard** :

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
```

- **Accès sécurisé** : Nécessite un token d'authentification ou kubeconfig

3. **Navigation dans le Dashboard**
   - **Vue Cluster** : Informations générales sur les nœuds et ressources
   - **Vue Workloads** : Deployments, pods, replica sets, jobs
   - **Vue Discovery and Load Balancing** : Services et ingress
   - **Vue Config and Storage** : ConfigMaps, secrets, volumes
   - **Monitoring** : Graphiques de ressources CPU/mémoire

kubectl - L'Outil de Ligne de Commande

Introduction à kubectl

- **Outil officiel** pour interagir avec l'API Kubernetes
- **Interface unifiée** pour tous les clusters Kubernetes
- **Syntaxe générale** :

```
kubectl [commande] [type_ressource] [nom_ressource] [options]
```

- **Configuration** : Utilise le fichier `~/.kube/config` par défaut

Configuration et contextes

```
# Voir la configuration actuelle
kubectl config view

# Lister les contextes disponibles
kubectl config get-contexts

# Changer de contexte
kubectl config use-context nom-du-contexte

# Définir le namespace par défaut
kubectl config set-context --current --namespace=mon-namespace
```

kubectl - L'Outil de Ligne de Commande

Commandes de base - Informations sur le cluster

```
# Informations générales du cluster
kubectl cluster-info

# Version de kubectl et du cluster
kubectl version

# État des nœuds
kubectl get nodes
kubectl describe node nom-du-noeud

# Utilisation des ressources
kubectl top nodes
kubectl top pods
```

kubectl - L'Outil de Ligne de Commande

Commandes CRUD - Gestion des ressources

A. CREATE - Créer des ressources

```
# Créer à partir d'un fichier YAML
kubectl create -f deployment.yaml

# Créer un deployment directement
kubectl create deployment nginx --image=nginx:latest

# Créer un service
kubectl create service clusterip mon-service --tcp=80:80

# Créer un namespace
kubectl create namespace mon-namespace
```

kubectl - L'Outil de Ligne de Commande

Commandes CRUD - Gestion des ressources

B. GET - Consulter les ressources

```
# Lister les pods
kubectl get pods
kubectl get pods -o wide  # Plus de détails
kubectl get pods --all-namespaces

# Lister les deployments
kubectl get deployments
kubectl get deploy  # Forme courte

# Lister les services
kubectl get services
kubectl get svc  # Forme courte

# Format de sortie personnalisé
kubectl get pods -o json
kubectl get pods -o yaml
kubectl get pods -o custom-columns=NAME:.metadata.name,STATUS:.status.phase
```

kubectl - L'Outil de Ligne de Commande

Commandes CRUD - Gestion des ressources

C. DESCRIBE - Détails d'une ressource

```
# Détails complets d'un pod
kubectl describe pod nom-du-pod

# Détails d'un deployment
kubectl describe deployment nom-deployment

# Détails d'un service
kubectl describe service nom-service

# Détails d'un nœud
kubectl describe node nom-du-noeud
```

kubectl - L'Outil de Ligne de Commande

Commandes CRUD - Gestion des ressources

D. DELETE - Supprimer des ressources

```
# Supprimer un pod
kubectl delete pod nom-du-pod

# Supprimer un deployment
kubectl delete deployment nom-deployment

# Supprimer à partir d'un fichier
kubectl delete -f deployment.yaml

# Supprimer tous les pods d'un namespace
kubectl delete pods --all -n mon-namespace
```

kubectl - L'Outil de Ligne de Commande

Commandes de gestion avancées

A. APPLY - Gestion déclarative

```
# Appliquer une configuration (crée ou met à jour)
kubectl apply -f deployment.yaml

# Appliquer tout un dossier
kubectl apply -f ./configs/

# Mode dry-run pour tester
kubectl apply -f deployment.yaml --dry-run=client
```

B. EDIT - Modifier une ressource en cours

```
# Éditer un deployment
kubectl edit deployment nom-deployment

# Éditer avec un éditeur spécifique
EDITOR=nano kubectl edit pod nom-du-pod
```

kubectl - L'Outil de Ligne de Commande

Commandes de gestion avancées

C. SCALE - Mise à l'échelle

```
# Scaler un deployment
kubectl scale deployment nom-deployment --replicas=5

# Auto-scaling
kubectl autoscale deployment nom-deployment --min=2 --max=10 --cpu-percent=80
```

kubectl - L'Outil de Ligne de Commande

Commandes de debugging et monitoring

A. LOGS - Consultation des logs

```
# Logs d'un pod
kubectl logs nom-du-pod

# Logs en temps réel
kubectl logs -f nom-du-pod

# Logs d'un conteneur spécifique
kubectl logs nom-du-pod -c nom-conteneur

# Logs précédents (pod redémarré)
kubectl logs nom-du-pod --previous
```

B. EXEC - Exécution de commandes

```
# Se connecter à un pod
kubectl exec -it nom-du-pod -- /bin/bash

# Exécuter une commande spécifique
kubectl exec nom-du-pod -- ls -la

# Dans un conteneur spécifique
kubectl exec -it nom-du-pod -c nom-conteneur -- /bin/sh
```

kubectl - L'Outil de Ligne de Commande

Commandes de debugging et monitoring

C. PORT-FORWARD - Redirection de ports

```
# Rediriger port local vers pod
kubectl port-forward pod/nom-du-pod 8080:80

# Rediriger vers un service
kubectl port-forward service/nom-service 8080:80
```

Déployer, Démarrer et Exposer un Container

Workflow complet de déploiement

- **Étapes principales** :

1. Déployer l'application (Deployment)
2. Vérifier le déploiement (Pods)
3. Exposer l'application (Service)
4. Tester la connectivité
5. Scaler si nécessaire

Les pods

Modèle et Concept du Pod

Qu'est-ce qu'un Pod ?

- **Définition** : Plus petite unité déployable et exécutable dans Kubernetes
- **Composition** : Un ou plusieurs conteneurs étroitement couplés
- **Ressources partagées** :
  - **Réseau** : Même adresse IP et espace de ports
  - **Stockage** : Volumes partagés entre conteneurs
  - **Cycle de vie** : Démarrent et s'arrêtent ensemble
- **Analogie** : Comme une "machine virtuelle logique" pour conteneurs

Modèle et Concept du Pod

Modèles d'usage des Pods

A. Un conteneur par Pod (le plus courant)

```yaml
# Pod simple avec nginx
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:1.21
    ports:
    - containerPort: 80
```

B. Multi-conteneurs (patterns avancés)

- **Sidecar** : Conteneur utilitaire (logging, monitoring)
- **Ambassador** : Proxy pour communications externes
- **Adapter** : Normalisation des données de sortie

Modèle et Concept du Pod

API et interaction avec les Pods

- **API Version** : v1 (API stable)
- **Kind** : Pod
- **Endpoints principaux** :

```
# Lister tous les pods
GET /api/v1/pods

# Pod spécifique
GET /api/v1/namespaces/{namespace}/pods/{name}

# Logs d'un pod
GET /api/v1/namespaces/{namespace}/pods/{name}/log
```

- **Communication** : Via kubectl, Dashboard ou directement API REST

Modèle et Concept du Pod

Pourquoi des Pods et non des conteneurs directement ?

- **Cohésion applicative** : Grouper des conteneurs interdépendants
- **Abstraction** : Masquer la complexité du runtime de conteneurs
- **Portabilité** : Indépendant du runtime (Docker, containerd, CRI-O)
- **Orchestration** : Unité atomique pour le scheduling et les ressources

Durabilité et Persistance des Pods

Nature éphémère des Pods

- **Principe fondamental** : Les Pods sont **mortels et remplaçables**
- **Pas de garantie de survie** :
  - Redémarrage de nœud
  - Maintenance
  - Problèmes de ressources
  - Pannes matérielles
- **IP non stable** : Nouvelle IP à chaque recréation

Durabilité et Persistance des Pods

Implications de l'éphémérité

```
# Un pod qui meurt n'est PAS automatiquement redémarré
kubectl delete pod mon-pod
# → Le pod disparaît définitivement

# Pour la persistance, utiliser des contrôleurs
kubectl create deployment mon-app --image=nginx
# → Le pod sera recréé automatiquement si il meurt
```

Gestion de la persistance

- **Contrôleurs** : Deployment, ReplicaSet, StatefulSet
- **Volumes persistants** : Pour les données
- **Services** : Pour la découverte réseau stable

Durabilité et Persistance des Pods

Stratégies de redémarrage

- **Always** : Redémarre toujours (défaut)
- **OnFailure** : Redémarre seulement en cas d'échec
- **Never** : Ne redémarre jamais

```
spec:
  restartPolicy: Always  # Always | OnFailure | Never
```

Descripteurs YAML et JSON

Structure d'un descripteur Pod

```yaml
apiVersion: v1          # Version de l'API
kind: Pod               # Type de ressource
metadata:               # Métadonnées
  name: mon-pod
  namespace: default
  labels:
    app: webapp
    version: v1.0
  annotations:
    description: "Pod de démonstration"
spec:                   # Spécification désirée
  containers:
  - name: app-container
    image: nginx:1.21
    ports:
    - containerPort: 80
status:                 # État actuel (géré par Kubernetes)
  phase: Running
  podIP: 10.244.1.5
```

Descripteurs YAML et JSON

Sections principales du descripteur

A. metadata - Informations d'identification

```yaml
metadata:
  name: webapp-pod              # Nom unique dans le namespace
  namespace: production         # Namespace (default si omis)
  labels:                       # Étiquettes clé-valeur
    app: webapp
    tier: frontend
    environment: prod
  annotations:                  # Métadonnées supplémentaires
    kubernetes.io/created-by: "admin"
    company.com/team: "backend-team"
```

Descripteurs YAML et JSON

B. spec - Spécification du Pod

```yaml
spec:
  containers:                  # Liste des conteneurs
  - name: main-app
    image: nginx:1.21
    imagePullPolicy: Always    # Always | IfNotPresent | Never
    ports:
    - containerPort: 80
      protocol: TCP
    env:                       # Variables d'environnement
    - name: DB_HOST
      value: "mysql-service"
    resources:                 # Limites de ressources
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 500m
        memory: 512Mi
    volumeMounts:              # Montage de volumes
    - name: data-volume
      mountPath: /var/data
  volumes:                     # Définition des volumes
  - name: data-volume
    emptyDir: {}
  restartPolicy: Always
  nodeSelector:                # Sélection de nœud
    disktype: ssd
```

Descripteurs YAML et JSON

Formats YAML vs JSON

YAML (recommandé)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
  - name: nginx
    image: nginx:1.21
```

Descripteurs YAML et JSON

JSON (équivalent)

```
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "example-pod"
  },
  "spec": {
    "containers": [
      {
        "name": "nginx",
        "image": "nginx:1.21"
      }
    ]
  }
}
```

Descripteurs YAML et JSON

Validation et bonnes pratiques

```
# Valider la syntaxe sans créer
kubectl apply -f pod.yaml --dry-run=client

# Valider côté serveur
kubectl apply -f pod.yaml --dry-run=server

# Expliquer la structure
kubectl explain pod.spec.containers
```

Le Rôle du Scheduler

Qu'est-ce que le Scheduler ?

- **Composant** : kube-scheduler du plan de contrôle
- **Mission** : Assigner les Pods aux nœuds appropriés
- **Déclenchement** : Quand un Pod est créé sans `nodeName`
- **Algorithme** : Filtrage puis scoring des nœuds

A. Phase de filtrage

- **Critères éliminatoires** :
  - Ressources insuffisantes (CPU, mémoire)
  - Contraintes de nœud (nodeSelector, affinity)
  - Taints/Tolerations incompatibles
  - Volumes non disponibles

B. Phase de scoring

- **Critères de classement** :
  - Équilibrage de charge (répartition)
  - Affinité entre Pods
  - Localité des données
  - Préférences utilisateur
- **Résultat** : Nœud avec le meilleur score

Le Rôle du Scheduler

Contraintes de placement

A. nodeSelector - Sélection simple

```yaml
spec:
  nodeSelector:
    disktype: ssd           # Nœud doit avoir ce label
    zone: europe-west1-a
  containers:
  - name: app
    image: nginx
```

Le Rôle du Scheduler

B. Node Affinity - Sélection avancée

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: disktype
            operator: In
            values: ["ssd", "nvme"]
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: zone
            operator: In
            values: ["europe-west1-a"]
```

Le Rôle du Scheduler

C. Pod Affinity/Anti-Affinity

```
# Placer près d'autres pods avec label app=database
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values: ["database"]
        topologyKey: kubernetes.io/hostname
```

Le Rôle du Scheduler

Taints et Tolerations

```
# Ajouter un taint à un nœud
kubectl taint nodes node1 key=value:NoSchedule

# Pod qui tolère ce taint
spec:
  tolerations:
  - key: "key"
    operator: "Equal"
    value: "value"
    effect: "NoSchedule"
```

Scheduling manuel

```
# Forcer un nœud spécifique
spec:
  nodeName: worker-node-1  # Bypass du scheduler
  containers:
  - name: app
    image: nginx
```

Cycle de Vie des Pods

Phases du cycle de vie

- **Pending** : En attente de scheduling ou de téléchargement d'image
- **Running** : Au moins un conteneur s'exécute
- **Succeeded** : Tous les conteneurs ont terminé avec succès
- **Failed** : Au moins un conteneur a échoué
- **Unknown** : État impossible à déterminer

Détail des transitions d'état

```
# Créer un pod et observer
kubectl create -f pod.yaml
kubectl get pod mon-pod --watch

# Séquence typique :
# Pending → ContainerCreating → Running → Terminating → (supprimé)
```

Cycle de Vie des Pods

Hooks du cycle de vie

```yaml
spec:
  containers:
  - name: app
    image: nginx
    lifecycle:
      postStart:            # Après démarrage du conteneur
        exec:
          command: ["/bin/sh", "-c", "echo Hello > /tmp/hello"]
      preStop:              # Avant arrêt du conteneur
        exec:
          command: ["/bin/sh", "-c", "nginx -s quit"]
```

Cycle de Vie des Pods

Probes de santé

A. Liveness Probe - Redémarrage

```yaml
spec:
  containers:
  - name: app
    livenessProbe:
      httpGet:
        path: /health
        port: 8080
      initialDelaySeconds: 30
      periodSeconds: 10
      failureThreshold: 3
```

Cycle de Vie des Pods

Probes de santé

B. Readiness Probe - Trafic

```yaml
spec:
  containers:
  - name: app
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 5
```

C. Startup Probe - Applications lentes

```yaml
spec:
  containers:
  - name: app
    startupProbe:
      httpGet:
        path: /startup
        port: 8080
      periodSeconds: 10
      failureThreshold: 30  # 5 minutes max
```

Les Init Containers

Concept des Init Containers

- **Définition** : Conteneurs spéciaux qui s'exécutent avant les conteneurs principaux
- **Caractéristiques** :
  - Exécution séquentielle (un après l'autre)
  - Doivent se terminer avec succès
  - Redémarrent en cas d'échec
- **Cas d'usage** :
  - Préparation de données
  - Attente de services dépendants
  - Configuration initiale

Les Init Containers

Patterns courants avec Init Containers

A. Attente de dépendances

```yaml
initContainers:
- name: wait-for-redis
  image: redis:alpine
  command: ['redis-cli', '--scan']
  env:
  - name: REDIS_HOST
    value: "redis-service"
```

B. Migration de base de données

```yaml
initContainers:
- name: db-migration
  image: myapp/migrator:latest
  env:
  - name: DB_URL
    value: "postgres://db-service:5432/mydb"
  command: ["./migrate.sh"]
```

Les Init Containers

Patterns courants avec Init Containers

C. Téléchargement de configuration

```
initContainers:
- name: config-downloader
  image: appropriate/curl
  command: ['sh', '-c']
  args:
  - curl -o /config/app.conf http://config-server/app.conf
  volumeMounts:
  - name: config-volume
    mountPath: /config
```

Debugging des Init Containers

```
# Voir l'état des init containers
kubectl describe pod webapp-with-init

# Logs d'un init container spécifique
kubectl logs webapp-with-init -c wait-for-db

# Logs en temps réel
kubectl logs -f webapp-with-init -c setup-data
```

Pods Stateless vs Stateful

Pods Stateless (sans état)

A. Caractéristiques

- **Aucune donnée persistante** dans le conteneur
- **Interchangeables** : Tous les pods sont identiques
- **Scaling horizontal facile** : Ajout/suppression sans impact
- **Exemples** : Serveurs web, APIs REST, microservices

B. Exemple de Pod Stateless

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: web-stateless
spec:
  containers:
  - name: nginx
    image: nginx:1.21
    ports:
    - containerPort: 80
    # Pas de volumes persistants
    # Configuration via ConfigMap/Variables d'env
```

Pods Stateless vs Stateful

Pods Stateful (avec état)

A. Caractéristiques

- **Données persistantes** nécessaires
- **Identité stable** : Nom et stockage constants
- **Ordre de démarrage** important
- **Exemples** : Bases de données, systèmes de fichiers distribués

B. Exemple de Pod Stateful

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: database-stateful
spec:
  containers:
  - name: mysql
    image: mysql:8.0
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "secret"
    volumeMounts:
    - name: mysql-data
      mountPath: /var/lib/mysql
    ports:
    - containerPort: 3306
  volumes:
  - name: mysql-data
    persistentVolumeClaim:
      claimName: mysql-pvc
```

Pods Stateless vs Stateful

Gestion avec StatefulSet

```yaml
# Pour les applications stateful, utiliser StatefulSet
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql-stateful
spec:
  serviceName: mysql-service
  replicas: 3
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:8.0
        volumeMounts:
        - name: mysql-storage
          mountPath: /var/lib/mysql
  volumeClaimTemplates:
  - metadata:
      name: mysql-storage
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 10Gi
```

Pods Stateless vs Stateful

Aspect	Stateless	Stateful
Données	Temporaires	Persistantes
Identité	Interchangeable	Unique et stable
Scaling	Horizontal facile	Complexe
Backup	Pas nécessaire	Critique
Contrôleur	Deployment	StatefulSet

Organisation avec Labels, Sélecteurs et Namespaces

Labels - Système d'étiquetage

A. Principe des labels

- **Paires clé-valeur** attachées aux objets
- **Sélection et regroupement** des ressources
- **Métadonnées** pour l'organisation

```yaml
metadata:
  labels:
    app: webapp
    version: v2.1
    tier: frontend
    environment: production
    team: backend
```

B. Conventions de nommage

```yaml
# Bonnes pratiques
labels:
  app.kubernetes.io/name: mysql
  app.kubernetes.io/instance: mysql-primary
  app.kubernetes.io/version: "8.0"
  app.kubernetes.io/component: database
  app.kubernetes.io/part-of: wordpress
  app.kubernetes.io/managed-by: helm
```

Organisation avec Labels, Sélecteurs et Namespaces

Sélecteurs - Requêtes sur les labels

A. Sélecteur d'égalité

```
# CLI
kubectl get pods -l app=nginx
kubectl get pods -l environment=production,tier=frontend

# YAML
selector:
  matchLabels:
    app: nginx
    environment: production
```

B. Sélecteur basé sur des ensembles

```
selector:
  matchExpressions:
  - key: tier
    operator: In
    values: ["frontend", "backend"]
  - key: environment
    operator: NotIn
    values: ["development"]
  - key: version
    operator: Exists
```

Organisation avec Labels, Sélecteurs et Namespaces

Namespaces - Isolation logique

A. Concept et utilité

- **Séparation virtuelle** des ressources
- **Multi-tenancy** : Équipes, environnements, projets
- **Isolation des noms** : Même nom dans différents namespaces
- **Politiques** : RBAC, Network Policies, Resource Quotas

B. Gestion des namespaces

```
# Créer un namespace
kubectl create namespace production
kubectl create namespace development

# Lister les namespaces
kubectl get namespaces

# Travailler dans un namespace
kubectl get pods -n production
kubectl config set-context --current --namespace=production
```

Organisation avec Labels, Sélecteurs et Namespaces

C. Pod avec namespace

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webapp
  namespace: production
  labels:
    app: webapp
    env: prod
spec:
  containers:
  - name: nginx
    image: nginx:1.21
```

Organisation avec Labels, Sélecteurs et Namespaces

Gabarits (Templates) et réutilisabilité

A. Utilisation de variables

```
# Template avec substitution
apiVersion: v1
kind: Pod
metadata:
  name: ${APP_NAME}-pod
  labels:
    app: ${APP_NAME}
    version: ${VERSION}
spec:
  containers:
  - name: ${APP_NAME}
    image: ${IMAGE_NAME}:${VERSION}
```

Organisation avec Labels, Sélecteurs et Namespaces

Gabarits (Templates) et réutilisabilité

Organisation hiérarchique exemple

```
# Structure recommandée
Namespaces:
├── production/
│   ├── frontend/ (labels: tier=frontend, env=prod)
│   ├── backend/  (labels: tier=backend, env=prod)
│   └── database/ (labels: tier=data, env=prod)
├── staging/
│   ├── frontend/ (labels: tier=frontend, env=staging)
│   └── backend/  (labels: tier=backend, env=staging)
└── development/
    └── all-in-one/ (labels: env=dev)
```

Les Services

Concept de Service Kubernetes

Qu'est-ce qu'un Service ?

- **Abstraction réseau** : Point d'accès stable pour un ensemble de Pods
- **Problème résolu** : IPs des Pods changent constamment (éphémères)
- **Solution** : IP virtuelle stable + nom DNS + load balancing
- **Rôle** : Découverte de services et communication inter-applications

Pourquoi les Services sont nécessaires ?

- **Pods éphémères** : Créés/détruits dynamiquement
- **IPs changeantes** : Nouvelle IP à chaque recreation de Pod
- **Load balancing** : Distribuer le trafic entre plusieurs Pods
- **Découverte** : Comment une app trouve-t-elle une autre app ?

Types de Services

ClusterIP (par défaut)

- **Usage** : Communication interne au cluster uniquement
- **IP** : Virtuelle, accessible seulement dans le cluster
- **Cas d'usage** : Base de données, APIs internes, microservices

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  type: ClusterIP
  selector:
    app: backend
  ports:
  - port: 80
    targetPort: 8080
```

Types de Services

NodePort

- **Usage** : Accès externe via port sur chaque nœud
- **Port range** : 30000-32767 (par défaut)
- **Cas d'usage** : Applications de développement, accès direct

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp-nodeport
spec:
  type: NodePort
  selector:
    app: webapp
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30080
```

Types de Services (suite)

LoadBalancer

- **Usage** : Utilise le load balancer du cloud provider
- **IP externe** : Fournie par le provider (AWS ELB, GCP LB, etc.)
- **Cas d'usage** : Applications production exposées publiquement

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

Types de Services

ExternalName

- **Usage** : Redirection DNS vers service externe
- **Pas de proxy** : Simple alias DNS
- **Cas d'usage** : Services externes, migration progressive

```yaml
apiVersion: v1
kind: Service
metadata:
  name: external-db
spec:
  type: ExternalName
  externalName: database.example.com
```

Gestion Réseau des Services

Architecture réseau des Services

- **kube-proxy** : Composant réseau sur chaque nœud
- **Modes de fonctionnement** :
  - **iptables** : Règles de redirection (défaut)
  - **ipvs** : Load balancer avancé (haute performance)
  - **userspace** : Proxy en espace utilisateur (obsolète)

Fonctionnement avec iptables

```
# Voir les règles iptables créées par kube-proxy
sudo iptables -t nat -L | grep webapp-service

# Chaîne typique pour un service :
# KUBE-SVC-XXX -> KUBE-SEP-YYY (endpoints)
```

Sélection des Pods (Endpoints)

- **Selector** : Labels pour identifier les Pods cibles
- **Endpoints** : Liste automatique des IPs des Pods correspondants

IP Virtuels et Proxys

Cluster IP virtuelle

- **Allocation** : Depuis le CIDR de service du cluster
- **Caractéristiques** :
- Stable pendant toute la vie du Service
- Routée uniquement dans le cluster
- Pas attachée à une interface réseau physique

```
# Voir les IPs de services
kubectl get services -o wide

# Configuration du cluster
kubectl cluster-info dump | grep service-cluster-ip-range
```

Load balancing algorithmique

- **Round-robin** : Distribution séquentielle (défaut)
- **Session affinity** : Redirection vers le même Pod

```
spec:
  sessionAffinity: ClientIP  # Ou "None"
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 10800
```

IP Virtuels et Proxys

Proxy et redirection

```
# Test de connectivité interne
kubectl run test-pod --image=busybox --rm -it -- sh
# Dans le pod :
wget -qO- http://webapp-service:80
nslookup webapp-service
```

Services Headless

Concept Headless

- **Pas de Cluster IP** : clusterIP: None
- **DNS direct** : Retourne directement les IPs des Pods
- **Usage** : Quand on veut accéder directement aux Pods individuels
- **Cas d'usage** : Bases de données clustérisées, StatefulSets

Service Headless classique

```yaml
apiVersion: v1
kind: Service
metadata:
  name: database-headless
spec:
  clusterIP: None
  selector:
    app: database
  ports:
  - port: 5432
    targetPort: 5432
```

Services Headless

15. **Résolution DNS Headless**

```
# DNS normale : service -> ClusterIP
nslookup webapp-service
# Retourne : 10.96.1.100

# DNS headless : service -> IPs des Pods
nslookup database-headless
# Retourne : 10.244.1.5, 10.244.2.3, 10.244.3.7
```

Service Discovery et DNS

DNS intégré Kubernetes

- **CoreDNS** : Serveur DNS du cluster (remplace kube-dns)
- **Résolution automatique** : Services et Pods ont des noms DNS
- **Domaines** :
- Services : `service-name.namespace.svc.cluster.local`
- Pods : `pod-ip.namespace.pod.cluster.local`

Résolution DNS des Services

```
# Nom court (même namespace)
curl http://webapp-service

# Nom complet (autre namespace)
curl http://webapp-service.production.svc.cluster.local

# Verification DNS
nslookup webapp-service.production.svc.cluster.local
```

Service Discovery et DNS

Résolution DNS des Pods

- **Format Pod DNS** : `pod-ip-avec-tirets.namespace.pod.cluster.local`
- **Exemple** : Pod IP 10.244.1.5 → `10-244-1-5.default.pod.cluster.local`

```
# Activer la résolution DNS pour un Pod
spec:
  subdomain: webapp-service  # Nom du service
  hostname: webapp-01        # Nom du pod
# Résultat : webapp-01.webapp-service.default.svc.cluster.local
```

Configuration DNS avancée

Politique DNS des Pods

```yaml
spec:
  dnsPolicy: ClusterFirst  # Défaut
  # Autres options :
  # - Default : DNS de l'hôte
  # - ClusterFirstWithHostNet : Cluster puis hôte
  # - None : Configuration manuelle
```

Configuration DNS personnalisée

```yaml
spec:
  dnsPolicy: None
  dnsConfig:
    nameservers:
    - 8.8.8.8
    - 1.1.1.1
    searches:
    - my-company.com
    options:
    - name: ndots
      value: "2"
```

Applications et Services

Pattern Application + Service

- **Deployment** : Gère les Pods de l'application
- **Service** : Expose l'application via réseau
- **Lien** : Labels/selectors pour connexion

```yaml
# Application
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
```

Applications et Services

Service correspondant

```yaml
# Service pour l'application
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  selector:
    app: webapp  # Même label que le Deployment
  ports:
  - port: 80
    targetPort: 80
  type: ClusterIP
```

Le rôle d'Ingress

Qu'est-ce qu'Ingress ?

- **Routage HTTP/HTTPS** : Point d'entrée unique pour le cluster
- **Avantages** : Une IP publique pour plusieurs services
- **Fonctionnalités** :
  - Routage basé sur l'hôte ou le chemin
  - Terminaison SSL/TLS
  - Load balancing L7

Le rôle d'Ingress

Ingress vs Services

Aspect	Service LoadBalancer	Ingress
IP publique	Une par service	Une pour tous
Coût	Élevé (multiple LB)	Économique
Protocole	L4 (TCP/UDP)	L7 (HTTP/HTTPS)
SSL	Terminaison externe	Terminaison native

Le rôle d'Ingress

Contrôleur Ingress requis

- **nginx-ingress** : Le plus populaire
- **traefik** : Simple et moderne
- **istio** : Service mesh avancé
- **GKE Ingress** : Intégré Google Cloud

```
# Installer nginx-ingress
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/cloud/deploy.yaml
```

Configuration Ingress

Ingress basique

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webapp-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: webapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: webapp-service
            port:
              number: 80
```

Contrôleurs

Concept des Contrôleurs

Qu'est-ce qu'un Contrôleur ?

- **Boucle de contrôle** : Surveille l'état actuel vs état désiré
- **Réconciliation automatique** : Corrige les écarts détectés
- **Abstraction** : Gère les Pods sans intervention manuelle
- **Principe** : "Déclarer ce qu'on veut, pas comment le faire"

Concept des Contrôleurs

Pourquoi ne pas utiliser directement les Pods ?

- **Pods nus** : Aucune garantie de redémarrage
- **Gestion manuelle** : Scaling, mises à jour, réparations
- **Pas de résilience** : Pod supprimé = perdu définitivement
- **Solution** : Contrôleurs pour automatiser la gestion

Concept des Contrôleurs

Types de contrôleurs principaux

- **ReplicaSet** : Maintient un nombre de répliques
- **Deployment** : Gestion des mises à jour progressives
- **StatefulSet** : Applications avec état persistant
- **DaemonSet** : Un Pod par nœud
- **Job** : Tâches ponctuelles
- **CronJob** : Tâches programmées

ReplicaSet - Concepts de base

Qu'est-ce qu'un ReplicaSet ?

- **Rôle** : Maintenir un nombre spécifié de répliques de Pods
- **Surveillance** : Compte en permanence les Pods vivants
- **Action** : Crée ou supprime des Pods selon le besoin
- **Sélection** : Utilise les labels pour identifier ses Pods

ReplicaSet - Concepts de base

Fonctionnement du ReplicaSet

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
```

ReplicaSet - Concepts de base

Cycle de vie ReplicaSet

- **Création** : Déploie immédiatement 3 Pods
- **Surveillance** : Vérifie constamment le nombre de Pods
- **Auto-réparation** : Si un Pod meurt → en crée un nouveau
- **Scaling** : Modification de `replicas` → ajuste automatiquement

Quand utiliser ReplicaSet

Cas d'usage appropriés

- **Applications stateless** : Serveurs web, APIs REST
- **Haute disponibilité** : Redondance automatique
- **Load balancing** : Plusieurs instances identiques
- **Scaling horizontal** : Ajout/suppression de répliques

Limitations du ReplicaSet

- **Pas de gestion des mises à jour** : Image fixe
- **Pas d'historique** : Aucun rollback possible
- **Mises à jour manuelles** : Suppression/recréation nécessaire
- **Solution** : Utiliser Deployment à la place

Deployment - Concept et avantages

Qu'est-ce qu'un Deployment ?

- **Niveau supérieur** : Gère les ReplicaSets automatiquement
- **Mises à jour** : Rolling updates sans interruption
- **Historique** : Versioning et rollback des déploiements
- **Stratégies** : Différentes approches de mise à jour

Architecture Deployment

```
Deployment
├── ReplicaSet v1 (ancienne version)
│   ├── Pod 1 (nginx:1.20)
│   └── Pod 2 (nginx:1.20)
└── ReplicaSet v2 (nouvelle version)
    ├── Pod 3 (nginx:1.21)
    ├── Pod 4 (nginx:1.21)
    └── Pod 5 (nginx:1.21)
```

Deployment - Concept et avantages

Exemple de Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
```

Stratégies de déploiement

RollingUpdate (par défaut)

- **Principe** : Remplacement progressif des Pods
- **Avantage** : Zéro downtime
- **Paramètres** :
  - `maxUnavailable` : Pods maximum indisponibles
  - `maxSurge` : Pods supplémentaires maximum

```yaml
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 25%  # ou nombre absolu
    maxSurge: 25%        # ou nombre absolu
```

Stratégies de déploiement

Recreate

- **Principe** : Supprime tous les Pods puis recrée
- **Usage** : Applications ne supportant qu'une instance
- **Inconvénient** : Downtime pendant le déploiement

```
strategy:
  type: Recreate
```

Stratégies de déploiement

Gestion des mises à jour

```
# Mettre à jour l'image
kubectl set image deployment/nginx-deployment nginx=nginx:1.22

# Voir le statut du rollout
kubectl rollout status deployment/nginx-deployment

# Voir l'historique
kubectl rollout history deployment/nginx-deployment

# Rollback vers version précédente
kubectl rollout undo deployment/nginx-deployment

# Rollback vers version spécifique
kubectl rollout undo deployment/nginx-deployment --to-revision=2
```

Deployment Controller en action

Processus de mise à jour

```
# État initial : 3 Pods nginx:1.20
kubectl get pods

# Démarrer mise à jour
kubectl set image deployment/nginx-deployment nginx=nginx:1.21

# Observer la transition
kubectl get pods --watch
# Vous verrez :
# 1. Nouveau ReplicaSet créé
# 2. Pods anciens terminés progressivement
# 3. Nouveaux Pods créés progressivement
```

Contrôle avancé des déploiements

```
spec:
  progressDeadlineSeconds: 600  # Timeout du déploiement
  revisionHistoryLimit: 10      # Nombre d'anciennes versions à garder
  paused: false                 # Mettre en pause le déploiement
  strategy:
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
```

StatefulSet - Applications avec état

Qu'est-ce qu'un StatefulSet ?

- **Pour applications stateful** : Bases de données, stockage
- **Identité stable** : Nom et stockage persistants
- **Ordre de démarrage** : Séquentiel et prévisible
- **DNS stable** : Noms de pods prévisibles

Caractéristiques StatefulSet

- **Noms ordonnés** : app-0, app-1, app-2...
- **Stockage dédié** : Volume personnel par Pod
- **Démarrage séquentiel** : app-0 puis app-1 puis app-2
- **Service Headless** : Accès direct à chaque Pod

StatefulSet - Applications avec état

Exemple StatefulSet

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql-stateful
spec:
  serviceName: mysql-headless
  replicas: 3
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:8.0
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "password"
        volumeMounts:
        - name: mysql-data
          mountPath: /var/lib/mysql
  volumeClaimTemplates:
  - metadata:
      name: mysql-data
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 10Gi
```

StatefulSet vs Deployment

Comparaison des caractéristiques

Aspect	Deployment	StatefulSet
Noms Pods	Aléatoires	Ordonnés (app-0, app-1)
Stockage	Partagé/Temporaire	Dédié et persistant
Démarrage	Parallèle	Séquentiel
Mise à jour	Rolling update	Rolling update ordonné
Cas d'usage	Apps stateless	Apps stateful

DaemonSet - Un Pod par nœud

Concept DaemonSet

- **Un Pod par nœud** : Automatiquement sur tous les nœuds
- **Ajout/suppression automatique** : Nouveaux nœuds = nouveau Pod
- **Cas d'usage** : Agents système, monitoring, networking
- **Exemples** : fluentd, node-exporter, kube-proxy

DaemonSet - Un Pod par nœud

Exemple DaemonSet

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: monitoring-agent
spec:
  selector:
    matchLabels:
      app: monitoring-agent
  template:
    metadata:
      labels:
        app: monitoring-agent
    spec:
      containers:
      - name: agent
        image: prom/node-exporter
        ports:
        - containerPort: 9100
        volumeMounts:
        - name: proc
          mountPath: /host/proc
          readOnly: true
        - name: sys
          mountPath: /host/sys
          readOnly: true
      volumes:
      - name: proc
        hostPath:
          path: /proc
      - name: sys
        hostPath:
          path: /sys
      hostNetwork: true
      hostPID: true
```

Sélection de nœuds

```yaml
spec:
  template:
    spec:
      nodeSelector:
        disktype: ssd  # Seulement sur nœuds SSD
      tolerations:     # Tolérer les taints
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
```

Gestion DaemonSet

Commandes DaemonSet

```
# Créer DaemonSet
kubectl apply -f daemonset.yaml

# Voir les DaemonSets
kubectl get daemonset
kubectl get ds  # forme courte
```

Stratégies de mise à jour DaemonSet

```
spec:
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1  # Nombre de nœuds simultanément mis à jour
```

Cas d'usage concrets

- **Monitoring** : node-exporter, datadog-agent
- **Logging** : fluentd, filebeat
- **Networking** : calico-node, weave-net
- **Sécurité** : falco, twistlock-defender
- **Stockage** : ceph-osd, glusterfs

Jobs - Tâches ponctuelles

Concept Job

- **Exécution ponctuelle** : Tâche qui doit se terminer
- **Pas de redémarrage continu** : Contrairement aux Deployments
- **Succès défini** : Complète quand la tâche réussit
- **Cas d'usage** : Migrations, sauvegardes, calculs batch

Job simple

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: data-migration
spec:
  template:
    spec:
      containers:
      - name: migrator
        image: postgres:13
        command: ["psql"]
        args: ["-h", "database", "-c", "UPDATE users SET status='active';"]
      restartPolicy: Never  # Ou OnFailure
  backoffLimit: 4         # Nombre max de tentatives
```

Jobs - Tâches ponctuelles

Job parallèle

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: parallel-processing
spec:
  parallelism: 3      # 3 Pods en parallèle
  completions: 9      # 9 exécutions au total
  template:
    spec:
      containers:
      - name: worker
        image: busybox
        command: ["sh", "-c", "echo Processing item $RANDOM && sleep 30"]
      restartPolicy: Never
```

Gestion des Jobs

Suivi et monitoring Jobs

```
# Voir les Jobs
kubectl get jobs

# Détails d'un Job
kubectl describe job data-migration

# Voir les Pods créés par le Job
kubectl get pods -l job-name=data-migration

# Logs du Job
kubectl logs job/data-migration

# Supprimer un Job (et ses Pods)
kubectl delete job data-migration
```

Gestion des Jobs

Politiques de redémarrage Job

- **Never** : Ne redémarre jamais (crée un nouveau Pod)
- **OnFailure** : Redémarre le Pod en cas d'échec

```yaml
spec:
  template:
    spec:
      restartPolicy: OnFailure
      containers:
      - name: task
        image: busybox
        command: ["sh", "-c", "exit 1"]  # Simule un échec
```

Nettoyage automatique

```yaml
spec:
  ttlSecondsAfterFinished: 3600  # Supprime le Job après 1h
  activeDeadlineSeconds: 1800    # Timeout du Job après 30min
```

CronJob - Tâches programmées

Concept CronJob

- **Tâches périodiques** : Exécution selon un planning
- **Format cron** : Même syntaxe que cron Unix
- **Crée des Jobs** : Un Job par exécution programmée
- **Cas d'usage** : Sauvegardes, rapports, nettoyage

CronJob - Tâches programmées

Exemple CronJob

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: database-backup
spec:
  schedule: "0 2 * * *"  # Tous les jours à 2h du matin
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            image: postgres:13
            command: ["pg_dump"]
            args: ["-h", "database", "-U", "user", "mydb"]
            volumeMounts:
            - name: backup-storage
              mountPath: /backup
          volumes:
          - name: backup-storage
            persistentVolumeClaim:
              claimName: backup-pvc
          restartPolicy: OnFailure
```

Configuration avancée CronJob

Gestion des politiques

```yaml
spec:
  schedule: "0 2 * * *"
  concurrencyPolicy: Forbid  # Allow | Forbid | Replace
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  startingDeadlineSeconds: 300
  suspend: false  # Suspendre temporairement
```

Politiques de concurrence

- **Allow** : Plusieurs Jobs peuvent s'exécuter simultanément
- **Forbid** : Pas de nouvelle exécution si la précédente n'est pas finie
- **Replace** : Annule l'ancienne et démarre la nouvelle

Volumes

Utilité des Volumes

Problématique du stockage dans Kubernetes

- **Containers éphémères** : Données perdues à l'arrêt du conteneur
- **Pods temporaires** : Redémarrage = perte de toutes les données
- **Partage impossible** : Pas de communication par fichiers entre conteneurs
- **Solution** : Volumes pour persistance et partage

Utilité des Volumes

Qu'est-ce qu'un Volume ?

- **Abstraction de stockage** : Indépendant du cycle de vie des conteneurs
- **Montage dans les conteneurs** : Accessible via système de fichiers
- **Partageable** : Plusieurs conteneurs peuvent utiliser le même volume
- **Persistance** : Survit aux redémarrages de conteneurs

Utilité des Volumes

Avantages des Volumes

- **Persistance des données** : Bases de données, logs, configuration
- **Partage entre conteneurs** : Communication via fichiers
- **Découplage** : Séparation des données et de l'application
- **Flexibilité** : Différents types selon les besoins

Types de Volumes

Volumes temporaires

- **emptyDir** : Répertoire vide, vie du Pod
- **Cas d'usage** : Cache, fichiers temporaires, communication inter-conteneurs

```
volumes:
- name: temp-storage
  emptyDir: {}
```

Volumes de configuration

- **configMap** : Fichiers de configuration
- **secret** : Données sensibles (mots de passe, clés)

```
volumes:
- name: config-volume
  configMap:
    name: app-config
- name: secret-volume
  secret:
    secretName: app-secrets
```

Types de Volumes

Volumes système hôte

- **hostPath** : Accès direct au système de fichiers du nœud
- **Attention** : Lié au nœud spécifique

```
volumes:
- name: host-logs
  hostPath:
    path: /var/log
    type: Directory
```

Volumes cloud persistants

- **awsElasticBlockStore** : EBS Amazon
- **gcePersistentDisk** : Disques Google Cloud
- **azureDisk** : Disques Azure

```
volumes:
- name: aws-storage
  awsElasticBlockStore:
    volumeID: vol-12345678
    fsType: ext4
```

Types de Volumes

Volumes réseau

- **nfs** : Network File System
- **cephfs** : Système de fichiers Ceph
- **glusterfs** : GlusterFS distribué

```
volumes:
- name: nfs-storage
  nfs:
    server: nfs-server.example.com
    path: /shared/data
```

Volumes persistants (recommandé)

- **persistentVolumeClaim** : Abstraction pour le stockage persistant
- **Avantage** : Indépendant du provider cloud

```
volumes:
- name: data-storage
  persistentVolumeClaim:
    claimName: my-pvc
```

Partage entre Conteneurs

Concept de partage intra-Pod

- **Même Pod** : Conteneurs partagent les volumes montés
- **Synchronisation** : Lecture/écriture simultanée possible
- **Communication** : Via fichiers plutôt que réseau
- **Cas d'usage** : Sidecar patterns, traitement de logs

Partage entre Conteneurs

Exemple de partage avec emptyDir

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: shared-volume-pod
spec:
  containers:
  - name: writer
    image: busybox
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo $(date) >> /shared/data.log; sleep 10; done"]
    volumeMounts:
    - name: shared-data
      mountPath: /shared

  - name: reader
    image: busybox
    command: ["/bin/sh"]
    args: ["-c", "while true; do tail -f /shared/data.log; sleep 5; done"]
    volumeMounts:
    - name: shared-data
      mountPath: /shared

  volumes:
  - name: shared-data
    emptyDir: {}
```

Partage entre Conteneurs

Pattern Sidecar avec volumes

```
# Application principale + conteneur de logs
containers:
- name: web-app
  image: nginx
  volumeMounts:
  - name: log-volume
    mountPath: /var/log/nginx

- name: log-processor
  image: fluentd
  volumeMounts:
  - name: log-volume
    mountPath: /var/log/nginx
    readOnly: true

volumes:
- name: log-volume
  emptyDir: {}
```

Partage avec ConfigMap et Secret

Partage de configuration

```
# Créer une ConfigMap
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  database.properties: |
    host=mysql-service
    port=3306
    database=myapp
  nginx.conf: |
    server {
      listen 80;
      location / {
        proxy_pass http://backend;
      }
    }
```

Partage avec ConfigMap et Secret

Pod utilisant ConfigMap

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: config-pod
spec:
  containers:
  - name: app
    image: nginx
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
    - name: nginx-config
      mountPath: /etc/nginx/conf.d

  volumes:
  - name: config-volume
    configMap:
      name: app-config
  - name: nginx-config
    configMap:
      name: app-config
      items:
      - key: nginx.conf
        path: default.conf
```

Partage avec ConfigMap et Secret

Gestion des Secrets

```yaml
# Secret pour données sensibles
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
data:
  username: YWRtaW4=  # admin en base64
  password: MWYyZDFlMmU2N2Rm  # password en base64

---
# Pod utilisant le Secret
spec:
  containers:
  - name: app
    volumeMounts:
    - name: secret-volume
      mountPath: /etc/secrets
      readOnly: true
  volumes:
  - name: secret-volume
    secret:
      secretName: db-secret
```

Accès au Filesystem du Nœud

Volume hostPath - Concept

- **Accès direct** : Monte un répertoire/fichier du nœud hôte
- **Persistance limitée** : Lié au nœud spécifique
- **Cas d'usage** : Logs système, monitoring, accès périphériques
- **Attention** : Problème de portabilité entre nœuds

Types de hostPath

```
volumes:
- name: host-volume
  hostPath:
    path: /var/log          # Chemin sur l'hôte
    type: Directory         # Type de vérification
```

Accès au Filesystem du Nœud

Types disponibles :

- **Directory** : Doit être un répertoire existant
- **DirectoryOrCreate** : Crée le répertoire si inexistant
- **File** : Doit être un fichier existant
- **FileOrCreate** : Crée le fichier si inexistant
- **Socket** : Socket Unix existant
- **CharDevice** : Périphérique caractère
- **BlockDevice** : Périphérique bloc

Accès au Filesystem du Nœud

Exemple d'accès aux logs système

```yaml
apiVersion: v1
kind: Pod
metadata:
name: log-monitor
spec:
containers:
- name: log-reader
  image: busybox
  command: ["/bin/sh"]
  args: ["-c", "tail -f /host-logs/messages"]
  volumeMounts:
  - name: system-logs
    mountPath: /host-logs
    readOnly: true

volumes:
- name: system-logs
  hostPath:
    path: /var/log
    type: Directory
```

Exemples pratiques hostPath

Monitoring avec accès système

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: node-monitor
spec:
  containers:
  - name: node-exporter
    image: prom/node-exporter
    args:
    - --path.procfs=/host/proc
    - --path.sysfs=/host/sys
    volumeMounts:
    - name: proc
      mountPath: /host/proc
      readOnly: true
    - name: sys
      mountPath: /host/sys
      readOnly: true

  volumes:
  - name: proc
    hostPath:
      path: /proc
      type: Directory
  - name: sys
    hostPath:
      path: /sys
      type: Directory

  hostNetwork: true
  hostPID: true
```

Persistent Volumes (PV)

Concept Persistent Volume

- **Abstraction stockage** : Indépendant des Pods
- **Provisioning** : Manuel ou automatique
- **Cycle de vie** : Indépendant des Pods qui l'utilisent
- **Réutilisable** : Peut être réclamé par différents Pods

Exemple Persistent Volume

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
  hostPath:
    path: /mnt/data/mysql
```

Persistent Volumes (PV)

Modes d'accès PV

- **ReadWriteOnce (RWO)** : Lecture/écriture par un seul nœud
- **ReadOnlyMany (ROX)** : Lecture seule par plusieurs nœuds
- **ReadWriteMany (RWX)** : Lecture/écriture par plusieurs nœuds

```
accessModes:
- ReadWriteOnce  # Le plus courant
```

Persistent Volume Claims (PVC)

Concept PVC

- **Demande de stockage** : Spécifie les besoins en stockage
- **Binding automatique** : Kubernetes lie PVC et PV compatible
- **Abstraction utilisateur** : L'utilisateur ne voit que le PVC
- **Portable** : Indépendant du type de stockage sous-jacent

Persistent Volume Claims (PVC)

Exemple Persistent Volume Claim

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: manual
```

Persistent Volume Claims (PVC)

Utilisation PVC dans un Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-pod
spec:
  containers:
  - name: mysql
    image: mysql:8.0
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    volumeMounts:
    - name: mysql-data
      mountPath: /var/lib/mysql

  volumes:
  - name: mysql-data
    persistentVolumeClaim:
      claimName: mysql-pvc
```

Storage Classes et provisioning dynamique

Concept Storage Class

- **Provisioning automatique** : Création automatique de PV
- **Types de stockage** : SSD, HDD, réplication, etc.
- **Provider spécifique** : AWS EBS, GCE Disk, Azure Disk
- **Simplification** : Plus besoin de créer manuellement les PV

Exemple Storage Class

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast-ssd
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  zones: europe-west1-a,europe-west1-b
reclaimPolicy: Delete
allowVolumeExpansion: true
```

Storage Classes et provisioning dynamique

PVC avec Storage Class

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fast-storage-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  storageClassName: fast-ssd  # Utilise la Storage Class
```

Politiques de récupération

Reclaim Policies

- **Retain** : PV conservé après suppression du PVC
- **Delete** : PV supprimé avec le PVC (défaut)
- **Recycle** : Données effacées, PV disponible (obsolète)

```
spec:
  persistentVolumeReclaimPolicy: Retain  # ou Delete
```

Cycle de vie PV/PVC

```
1. Provisioning → PV créé (manuel ou automatique)
2. Binding      → PVC lié à un PV compatible
3. Using        → Pod utilise le PVC
4. Releasing    → PVC supprimé
5. Reclaiming   → PV traité selon sa politique
```

Politiques de récupération

États des volumes

```
# Voir les PV et leur état
kubectl get pv
# Available  : Disponible pour binding
# Bound      : Lié à un PVC
# Released   : PVC supprimé, en attente de reclaim
# Failed     : Échec de reclaim automatique

# Voir les PVC
kubectl get pvc
# Pending : En attente de binding
# Bound   : Lié à un PV
```

Configuration et secrets

Problématique de la Configuration

Défis de la configuration dans Kubernetes

- **Images figées** : Configuration hardcodée dans les conteneurs
- **Environnements multiples** : Dev, test, prod avec configs différentes
- **Secrets sensibles** : Mots de passe, clés API, certificats
- **Réutilisabilité** : Même image avec configurations variables

Problématique de la Configuration

Solutions Kubernetes pour la configuration

- **Arguments de ligne de commande** : Paramètres au démarrage
- **Variables d'environnement** : Configuration via env vars
- **ConfigMaps** : Données de configuration non sensibles
- **Secrets** : Données sensibles chiffrées
- **Volumes** : Fichiers de configuration montés

Problématique de la Configuration

Bonnes pratiques

- **Séparation** : Code vs configuration
- **Sécurité** : Jamais de secrets dans les images
- **Versioning** : Traçabilité des changements
- **Principe 12-factor** : Configuration par environnement

Paramètres Ligne de Commande

Command et Args dans les conteneurs

- **command** : Remplace l'ENTRYPOINT du Dockerfile
- **args** : Remplace le CMD du Dockerfile
- **Combinaison** : command + args = commande complète
- **Cas d'usage** : Outils CLI, scripts personnalisés

Syntaxe de base

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
spec:
  containers:
  - name: demo
    image: busybox
    command: ["/bin/sh"]
    args: ["-c", "echo Hello World && sleep 3600"]
```

Exemples Pratiques Command/Args

Base de données avec paramètres

```yaml
containers:
- name: mysql
  image: mysql:8.0
  command: ["mysqld"]
  args:
  - "--innodb-buffer-pool-size=1G"
  - "--max-connections=200"
  - "--log-bin=mysql-bin"
```

Application web avec options

```yaml
containers:
- name: webapp
  image: nginx
  command: ["nginx"]
  args: ["-g", "daemon off;", "-c", "/etc/nginx/custom.conf"]
```

Arguments Dynamiques

Arguments avec variables d'environnement

```
containers:
- name: app
  image: myapp:latest
  env:
  - name: DB_HOST
    value: "mysql-service"
  - name: DB_PORT
    value: "3306"
  command: ["./myapp"]
  args:
  - "--database-url=mysql://$(DB_HOST):$(DB_PORT)/mydb"
  - "--workers=$(WORKER_COUNT)"
  - "--log-level=$(LOG_LEVEL)"
```

Variables d'Environnement

Configuration par variables d'environnement

- **Standard 12-factor** : Configuration via env vars
- **Flexibilité** : Même image, configurations différentes
- **Simplicité** : Lecture facile dans toute application
- **Limitation** : Visibles dans les processus

Définition basique

```yaml
containers:
- name: webapp
  image: nginx
  env:
  - name: DATABASE_URL
    value: "postgresql://user:pass@db:5432/myapp"
  - name: REDIS_URL
    value: "redis://redis-service:6379"
  - name: LOG_LEVEL
    value: "INFO"
  - name: WORKERS
    value: "4"
```

Variables depuis Métadonnées

Variables depuis les métadonnées du Pod

```yaml
env:
- name: POD_NAME
  valueFrom:
    fieldRef:
      fieldPath: metadata.name
- name: POD_NAMESPACE
  valueFrom:
    fieldRef:
      fieldPath: metadata.namespace
- name: NODE_NAME
  valueFrom:
    fieldRef:
      fieldPath: spec.nodeName
- name: POD_IP
  valueFrom:
    fieldRef:
      fieldPath: status.podIP
```

Variables depuis Métadonnées

Variables depuis les ressources du conteneur

```
containers:
- name: app
  resources:
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
  env:
  - name: MEMORY_REQUEST
    valueFrom:
      resourceFieldRef:
        resource: requests.memory
  - name: CPU_LIMIT
    valueFrom:
      resourceFieldRef:
        resource: limits.cpu
```

Variables depuis ConfigMap et Secret

Variables depuis ConfigMap

```yaml
# Référence individuelle
env:
- name: DATABASE_HOST
  valueFrom:
    configMapKeyRef:
      name: app-config
      key: db.host

# Toutes les clés d'une ConfigMap
envFrom:
- configMapRef:
    name: app-config
```

Variables depuis Métadonnées

Variables depuis Secret

```yaml
# Référence individuelle
env:
- name: DB_PASSWORD
  valueFrom:
    secretKeyRef:
      name: db-secret
      key: password

# Toutes les clés d'un Secret
envFrom:
- secretRef:
    name: db-secret
```

ConfigMaps - Concept

Qu'est-ce qu'une ConfigMap ?

- **Stockage configuration** : Données non sensibles
- **Paires clé-valeur** : Configuration structurée
- **Découplage** : Configuration séparée du code
- **Réutilisabilité** : Partageable entre plusieurs Pods

Création d'une ConfigMap

```
# Depuis la ligne de commande
kubectl create configmap app-config \
  --from-literal=database.host=mysql-service \
  --from-literal=database.port=3306 \
  --from-literal=log.level=INFO

# Depuis un fichier
kubectl create configmap nginx-config \
  --from-file=nginx.conf

# Depuis un répertoire
kubectl create configmap app-configs \
  --from-file=./config-files/
```

ConfigMap en YAML

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  # Configuration simple
  database.host: "mysql-service"
  database.port: "3306"
  log.level: "INFO"

  # Fichier de configuration complet
  app.properties: |
    server.port=8080
    server.host=0.0.0.0
    database.url=jdbc:mysql://mysql-service:3306/myapp
    logging.level=INFO

  # Configuration Nginx
  nginx.conf: |
    server {
      listen 80;
      location / {
        proxy_pass http://backend-service;
        proxy_set_header Host $host;
      }
    }
```

Utilisation ConfigMaps - Variables

ConfigMap comme variables d'environnement

```yaml
containers:
- name: app
  image: myapp:latest
  env:
  - name: DB_HOST
    valueFrom:
      configMapKeyRef:
        name: app-config
        key: database.host
  - name: LOG_LEVEL
    valueFrom:
      configMapKeyRef:
        name: app-config
        key: log.level

  # Importer toute la ConfigMap
  envFrom:
  - configMapRef:
      name: app-config
```

Utilisation ConfigMaps - Volumes

ConfigMap comme volume

```
containers:
- name: nginx
  image: nginx
  volumeMounts:
  - name: config-volume
    mountPath: /etc/nginx/conf.d
  - name: app-config-volume
    mountPath: /etc/app

volumes:
- name: config-volume
  configMap:
    name: nginx-config
- name: app-config-volume
  configMap:
    name: app-config
    items:
    - key: app.properties
      path: application.properties
```

Secrets - Concept et Types

Qu'est-ce qu'un Secret ?

- **Données sensibles** : Mots de passe, clés, certificats
- **Encodage Base64** : Pas de chiffrement par défaut
- **Protection** : Stockage sécurisé dans etcd
- **Accès contrôlé** : RBAC pour limiter l'accès

Types de Secrets

- **Opaque** : Données arbitraires (défaut)
- **kubernetes.io/tls** : Certificats TLS
- **kubernetes.io/dockerconfigjson** : Credentials Docker
- **kubernetes.io/basic-auth** : Authentification basique
- **kubernetes.io/ssh-auth** : Clés SSH

Création de Secrets

Création via kubectl

```
# Secret générique
kubectl create secret generic db-secret \
  --from-literal=username=admin \
  --from-literal=password=secret123

# Secret TLS
kubectl create secret tls webapp-tls \
  --cert=webapp.crt \
  --key=webapp.key

# Secret Docker registry
kubectl create secret docker-registry regcred \
  --docker-server=registry.example.com \
  --docker-username=user \
  --docker-password=pass
```

Création de Secrets

Secret en YAML

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
data:
  username: YWRtaW4=     # admin en base64
  password: c2VjcmV0MTIz  # secret123 en base64

# Ou avec stringData (pas d'encodage base64)
stringData:
  username: admin
  password: secret123
```

Utilisation des Secrets

Secret comme variables d'environnement

```
containers:
- name: app
  image: myapp:latest
  env:
  - name: DB_USER
    valueFrom:
      secretKeyRef:
        name: db-secret
        key: username
  - name: DB_PASS
    valueFrom:
      secretKeyRef:
        name: db-secret
        key: password

  # Monter comme volume
  volumeMounts:
  - name: secret-volume
    mountPath: /etc/secrets
    readOnly: true

volumes:
- name: secret-volume
  secret:
    secretName: db-secret
```

Secret pour Registry Privé

Secret pour authentification registry

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: regcred
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: eyJhdXRocyI6eyJyZWdpc3RyeS5leGFtcGxlLmNvbSI6eyJ1c2VybmFtZSI6InVzZXIiLCJwYXNzd29yZCI6InBhc3MifX19

---
# Pod utilisant le secret
spec:
  imagePullSecrets:
  - name: regcred
  containers:
  - name: app
    image: registry.example.com/myapp:latest
```

Bonnes Pratiques de Sécurité

Sécurité des Secrets

- **Chiffrement au repos** : etcd chiffré recommandé
- **RBAC strict** : Limiter l'accès aux Secrets
- **Pas de logs** : Variables secrètes non loggées
- **Rotation** : Renouvellement régulier des secrets

Limitation des risques

```
# Secret monté en lecture seule
volumeMounts:
- name: secret-volume
  mountPath: /etc/secrets
  readOnly: true

# Permissions spécifiques
volumes:
- name: secret-volume
  secret:
    secretName: db-secret
    defaultMode: 0400  # Lecture seule pour le propriétaire
```

Bonnes Pratiques de Sécurité

Alternatives pour secrets sensibles

- **External Secrets Operator** : Intégration avec des gestionnaires externes
- **Sealed Secrets** : Secrets chiffrés dans Git
- **HashiCorp Vault** : Gestionnaire de secrets dédié
- **Cloud providers** : AWS Secrets Manager, Azure Key Vault

Mise à Jour de Configuration

Mise à jour des ConfigMaps

```
# Modifier une ConfigMap
kubectl edit configmap app-config

# Remplacer depuis un fichier
kubectl create configmap app-config \
  --from-file=new-config.yaml \
  --dry-run=client -o yaml | kubectl apply -f -

# Recharger les Pods (redémarrage nécessaire)
kubectl rollout restart deployment/myapp
```

Stratégies de déploiement Kubernetes

Concepts de Déploiement

Qu'est-ce qu'une stratégie de déploiement ?

- **Méthode** : Comment mettre à jour une application en production
- **Objectifs** : Minimiser les interruptions et les risques
- **Considérations** : Disponibilité, performance, rollback
- **Types** : Rolling update, blue-green, canary, recreate

Défis du déploiement en production

- **Zero downtime** : Maintenir le service pendant la mise à jour
- **Cohérence** : Éviter les états incohérents
- **Rollback rapide** : Retour arrière en cas de problème
- **Monitoring** : Surveillance de la santé des déploiements

Avantages de Kubernetes

- **Automatisation** : Déploiements gérés automatiquement
- **Stratégies intégrées** : Rolling update par défaut
- **Historique** : Versioning et rollback faciles
- **Observabilité** : Suivi en temps réel des déploiements

Création Déploiement YAML

Structure d'un fichier Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp-deployment
  labels:
    app: webapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: nginx:1.20
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 500m
            memory: 512Mi
```

Création Déploiement YAML

Sections importantes

- **metadata** : Nom et labels du deployment
- **spec.replicas** : Nombre de pods désirés
- **spec.selector** : Sélecteur pour identifier les pods
- **spec.template** : Template des pods à créer

Stratégies de Mise à Jour

RollingUpdate (par défaut)

- **Principe** : Remplacement progressif des pods
- **Avantage** : Zero downtime
- **Configuration** : maxUnavailable et maxSurge

```
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 25%    # Pods max indisponibles
    maxSurge: 25%          # Pods supplémentaires max
```

Recreate

- **Principe** : Supprime tous les pods puis recrée
- **Usage** : Applications ne supportant qu'une instance
- **Inconvénient** : Downtime pendant mise à jour

```
strategy:
  type: Recreate
```

Rolling Update en Détail

Configuration optimale Rolling Update

```yaml
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 1      # Toujours au moins 2 pods actifs
    maxSurge: 2           # Maximum 5 pods pendant update
```

Processus Rolling Update

```
État initial: 3 pods v1.0
1. Création de 2 nouveaux pods v1.1 (maxSurge=2)
2. Total temporaire: 5 pods (3 v1.0 + 2 v1.1)
3. Suppression de 1 pod v1.0 (maxUnavailable=1)
4. Création de 1 nouveau pod v1.1
5. Suppression progressive des pods v1.0
État final: 3 pods v1.1
```

Avantages Rolling Update

- **Disponibilité continue** : Service jamais interrompu
- **Test progressif** : Détection précoce des problèmes
- **Rollback rapide** : Arrêt possible à tout moment
- **Contrôle fin** : Paramètres ajustables selon les besoins

Mise à Jour avec kubectl

Mise à jour de l'image

```
# Changer l'image du conteneur
kubectl set image deployment/webapp webapp=nginx:1.21

# Mise à jour avec vérification
kubectl set image deployment/webapp webapp=nginx:1.21 --record

# Voir le statut en temps réel
kubectl rollout status deployment/webapp

# Observer les pods pendant la mise à jour
kubectl get pods --watch
```

Mise à jour par édition

```
# Éditer le déploiement directement
kubectl edit deployment webapp

# Ou appliquer un nouveau fichier YAML
kubectl apply -f webapp-v2.yaml
```

Mise à Jour avec kubectl

Suivi du déploiement

```
# Voir l'état du rollout
kubectl rollout status deployment/webapp

# Voir les événements en temps réel
kubectl get events --watch

# Vérifier les nouveaux pods
kubectl describe pods -l app=webapp
```

Mise à Jour Avancée

Mise à jour avec variables d'environnement

```
# Changer une variable d'environnement
kubectl set env deployment/webapp VERSION=2.0

# Ajouter plusieurs variables
kubectl set env deployment/webapp \
  VERSION=2.0 \
  FEATURE_FLAG=enabled \
  LOG_LEVEL=debug
```

Mise à jour des ressources

```
# Changer les limites de ressources
kubectl patch deployment webapp -p='{"spec":{"template":{"spec":{"containers":[{"name":"webapp","resources":{"limits":{"cpu":"1000m","memory":"1Gi"}}}]}}}}'
```

Mise à Jour Avancée

Pause et reprise du déploiement

```
# Mettre en pause un déploiement
kubectl rollout pause deployment/webapp

# Faire plusieurs changements pendant la pause
kubectl set image deployment/webapp webapp=nginx:1.22
kubectl set env deployment/webapp DEBUG=true

# Reprendre le déploiement
kubectl rollout resume deployment/webapp
```

Retour Arrière (Rollback)

Historique des déploiements

```
# Voir l'historique complet
kubectl rollout history deployment/webapp

# Voir les détails d'une révision
kubectl rollout history deployment/webapp --revision=2
```

Rollback simple

```
# Retour à la version précédente
kubectl rollout undo deployment/webapp

# Vérifier le rollback
kubectl rollout status deployment/webapp
kubectl get pods
```

Rollback vers version spécifique

```
# Retour vers une révision particulière
kubectl rollout undo deployment/webapp --to-revision=3

# Voir quelle révision est active
kubectl rollout history deployment/webapp
```

Rollback Avancé

Validation avant rollback

```
# Comparer les révisions
kubectl rollout history deployment/webapp --revision=1
kubectl rollout history deployment/webapp --revision=2

# Rollback avec confirmation
kubectl rollout undo deployment/webapp --to-revision=1 --dry-run=client
```

Gestion de l'historique

```
# Limiter le nombre de révisions conservées
spec:
  revisionHistoryLimit: 5    # Garde seulement 5 versions
```

Rollback en cas d'urgence

```
# Rollback immédiat en cas de problème critique
kubectl rollout undo deployment/webapp

# Scaler à zéro en urgence (arrêt complet)
kubectl scale deployment webapp --replicas=0

# Remettre en service après correction
kubectl scale deployment webapp --replicas=3
```

AutoScaling Horizontal (HPA)

Horizontal Pod Autoscaler

- **Principe** : Ajuste automatiquement le nombre de pods
- **Métriques** : CPU, mémoire, métriques personnalisées
- **Avantages** : Adaptation automatique à la charge
- **Prérequis** : Metrics Server installé

Création HPA basique

```
# HPA basé sur le CPU
kubectl autoscale deployment webapp \
  --cpu-percent=70 \
  --min=2 \
  --max=10

# Voir l'état de l'HPA
kubectl get hpa
kubectl describe hpa webapp
```

AutoScaling Horizontal (HPA)

HPA en YAML

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: webapp-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: webapp
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

AutoScaling Avancé

HPA avec métriques multiples

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: webapp-hpa-advanced
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: webapp
  minReplicas: 3
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 60
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 60
      policies:
      - type: Percent
        value: 100
        periodSeconds: 15
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
```

AutoScaling Vertical (VPA)

Vertical Pod Autoscaler

- **Principe** : Ajuste les ressources CPU/mémoire des pods
- **Modes** : Off, Initial, Auto
- **Installation** : Addon séparé à installer

VPA Configuration

```yaml
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: webapp-vpa
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: webapp
  updatePolicy:
    updateMode: "Auto"    # Off | Initial | Auto
  resourcePolicy:
    containerPolicies:
    - containerName: webapp
      maxAllowed:
        cpu: 2000m
        memory: 2Gi
      minAllowed:
        cpu: 100m
        memory: 128Mi
```

Sécurité dans kubernetes

Concepts de Sécurité Kubernetes

Défis de sécurité dans Kubernetes

- **Multi-tenancy** : Isolation entre applications et équipes
- **Réseau ouvert** : Communication par défaut entre tous les pods
- **Privilèges** : Contrôle des permissions système
- **Images** : Sécurité du code et des dépendances
- **Accès API** : Authentification et autorisation

Concepts de Sécurité Kubernetes

Principe de sécurité par défaut

- **Least privilege** : Permissions minimales nécessaires
- **Defense in depth** : Plusieurs couches de sécurité
- **Zero trust** : Vérification de tous les accès
- **Audit** : Traçabilité de toutes les actions

Concepts de Sécurité Kubernetes

Composants de sécurité Kubernetes

- **Service Accounts** : Identité pour les pods
- **RBAC** : Contrôle d'accès basé sur les rôles
- **Network Policies** : Contrôle du trafic réseau
- **Security Contexts** : Contrôle des privilèges système
- **Pod Security Standards** : Politiques de sécurité des pods

Service Accounts - Concept

Qu'est-ce qu'un Service Account ?

- **Identité pour les pods** : Authentification auprès de l'API Kubernetes
- **Token automatique** : JWT token monté dans le pod
- **Permissions** : Définies via RBAC
- **Namespace isolé** : Un SA par namespace

Service Account par défaut

```
# Voir les service accounts
kubectl get serviceaccounts
kubectl get sa

# Service account par défaut
kubectl describe sa default

# Token associé
kubectl describe secret default-token-xxxxx
```

Service Accounts - Concept

Création d'un Service Account

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: webapp-sa
  namespace: default
automountServiceAccountToken: true
```

```
# Créer via kubectl
kubectl create serviceaccount webapp-sa

# Voir les détails
kubectl describe sa webapp-sa
```

Utilisation Service Accounts

Pod avec Service Account spécifique

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webapp-with-sa
spec:
  serviceAccountName: webapp-sa
  containers:
  - name: webapp
    image: nginx
    # Le token sera automatiquement monté dans
    # /var/run/secrets/kubernetes.io/serviceaccount/
```

Accès à l'API depuis un Pod

```bash
# Depuis l'intérieur d'un pod
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
NAMESPACE=$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace)
CACERT=/var/run/secrets/kubernetes.io/serviceaccount/ca.crt

# Appel à l'API
curl --cacert $CACERT \
  -H "Authorization: Bearer $TOKEN" \
  https://kubernetes.default.svc/api/v1/namespaces/$NAMESPACE/pods
```

Utilisation Service Accounts

Désactiver le montage automatique

```
spec:
  serviceAccountName: webapp-sa
  automountServiceAccountToken: false  # Désactive le montage
  containers:
  - name: webapp
    image: nginx
```

RBAC - Role-Based Access Control

Composants RBAC

- **Role** : Permissions dans un namespace
- **ClusterRole** : Permissions au niveau cluster
- **RoleBinding** : Lie un Role à des utilisateurs/SA
- **ClusterRoleBinding** : Lie un ClusterRole à des utilisateurs/SA

Exemple de Role

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
- apiGroups: [""]
  resources: ["pods/log"]
  verbs: ["get"]
```

RBAC - Role-Based Access Control

RoleBinding correspondant

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: ServiceAccount
  name: webapp-sa
  namespace: default
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

RBAC ClusterRole

ClusterRole pour permissions globales

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: node-reader
rules:
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["metrics.k8s.io"]
  resources: ["nodes", "pods"]
  verbs: ["get", "list"]
```

RBAC - Role-Based Access Control

ClusterRoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: node-reader-binding
subjects:
- kind: ServiceAccount
  name: monitoring-sa
  namespace: monitoring
roleRef:
  kind: ClusterRole
  name: node-reader
  apiGroup: rbac.authorization.k8s.io
```

RBAC - Role-Based Access Control

Permissions granulaires

```yaml
# Role avec permissions spécifiques
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "update", "patch"]
- apiGroups: [""]
  resources: ["services"]
  verbs: ["get", "list", "create"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "delete"]
  resourceNames: ["webapp-pod-1", "webapp-pod-2"]  # Pods spécifiques
```

RBAC Commandes et Debug

Commandes RBAC

```
# Voir les roles et bindings
kubectl get roles
kubectl get rolebindings
kubectl get clusterroles
kubectl get clusterrolebindings

# Détails d'un role
kubectl describe role pod-reader
kubectl describe rolebinding read-pods

# Vérifier les permissions d'un utilisateur/SA
kubectl auth can-i get pods --as=system:serviceaccount:default:webapp-sa
kubectl auth can-i create deployments --as=system:serviceaccount:default:webapp-sa

# Voir toutes les permissions d'un SA
kubectl auth can-i --list --as=system:serviceaccount:default:webapp-sa
```

Network Policies - Contrôle Trafic

Concept Network Policies

- **Pare-feu Kubernetes** : Contrôle du trafic entre pods
- **Sélecteurs** : Basé sur les labels des pods
- **Règles** : Ingress (entrant) et Egress (sortant)
- **CNI requis** : Plugin réseau compatible (Calico, Cilium, etc.)

Comportement par défaut

- **Sans Network Policy** : Tout trafic autorisé
- **Avec Network Policy** : Deny by default pour les pods sélectionnés
- **Isolation progressive** : Ajout de règles pour autoriser

Network Policies - Contrôle Trafic

Network Policy de base

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
  namespace: default
spec:
  podSelector: {}  # Sélectionne tous les pods
  policyTypes:
  - Ingress
  - Egress
  # Pas de règles = deny all
```

Network Policies Exemples

Autoriser trafic spécifique

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-backend
spec:
  podSelector:
    matchLabels:
      tier: backend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          tier: frontend
    ports:
    - protocol: TCP
      port: 8080
```

Network Policies Exemples

Policy avec namespaces

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-prod-namespace
spec:
  podSelector:
    matchLabels:
      app: database
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: production
    - podSelector:
        matchLabels:
          role: app
    ports:
    - protocol: TCP
      port: 3306
```

Network Policies Avancées

Policy Egress (trafic sortant)

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: webapp-egress
spec:
  podSelector:
    matchLabels:
      app: webapp
  policyTypes:
  - Egress
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: database
    ports:
    - protocol: TCP
      port: 3306
  - to: []  # Accès DNS
    ports:
    - protocol: UDP
      port: 53
  - to:
    - namespaceSelector:
        matchLabels:
          name: kube-system
```

Network Policies Avancées

Policy avec IP externes

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-external-api
spec:
  podSelector:
    matchLabels:
      app: webapp
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/8
        except:
        - 10.0.1.0/24
  - to:
    - ipBlock:
        cidr: 192.168.1.100/32  # API externe spécifique
    ports:
    - protocol: TCP
      port: 443
```

Sécurité des Images

Risques des images de conteneurs

- **Vulnérabilités** : CVE dans l'OS ou les applications
- **Malware** : Code malveillant dans les images
- **Images non officielles** : Provenance douteuse
- **Secrets hardcodés** : Mots de passe dans les layers

Bonnes pratiques images

- **Images officielles** : Privilégier les images certifiées
- **Tags spécifiques** : Éviter 'latest', utiliser version précise
- **Images minimales** : Alpine, distroless pour réduire la surface d'attaque
- **Scan de vulnérabilités** : Outils comme Trivy, Clair

Security Contexts - Contrôle Capacités

Qu'est-ce qu'un Security Context ?

- **Contrôle des privilèges** : Utilisateur, groupe, capacités
- **Niveaux** : Pod et conteneur
- **Sécurité renforcée** : Principe du moindre privilège
- **Compliance** : Respect des standards de sécurité

Security Context de base

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsUser: 1000        # UID non-root
    runAsGroup: 3000       # GID
    runAsNonRoot: true     # Interdit root
    fsGroup: 2000          # Groupe pour volumes
  containers:
  - name: app
    image: nginx
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop:
        - ALL
        add:
        - NET_BIND_SERVICE
```

Capabilities et Privilèges

Gestion des capabilities Linux

```yaml
securityContext:
  capabilities:
    drop:
    - ALL              # Supprimer toutes les capabilities
    add:
    - NET_BIND_SERVICE # Autoriser bind sur ports < 1024
    - CHOWN            # Autoriser chown
    - DAC_OVERRIDE     # Bypass des permissions fichiers
```

Capabilities courantes

- **NET_BIND_SERVICE** : Bind ports < 1024
- **NET_RAW** : Sockets raw (ping)
- **SYS_TIME** : Modifier l'horloge système
- **SYS_ADMIN** : Opérations administratives
- **CHOWN** : Changer propriétaire fichiers

Capabilities et Privilèges

Security Context avancé

```yaml
securityContext:
  runAsUser: 65534       # nobody
  runAsNonRoot: true
  readOnlyRootFilesystem: true
  allowPrivilegeEscalation: false
  capabilities:
    drop: ["ALL"]
  seccompProfile:
    type: RuntimeDefault
  seLinuxOptions:
    level: "s0:c123,c456"
```

Pod Security Standards

Niveaux de sécurité

- **Privileged** : Pas de restrictions (dangereux)
- **Baseline** : Restrictions minimales
- **Restricted** : Sécurité renforcée (recommandé)

Namespace avec Pod Security Standards

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: secure-namespace
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

Pod Security Standards

Pod conforme 'restricted'

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: restricted-pod
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: app
    image: nginx:1.21
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop: ["ALL"]
      runAsNonRoot: true
    volumeMounts:
    - name: tmp
      mountPath: /tmp
  volumes:
  - name: tmp
    emptyDir: {}
```

Présentation de l'outil helm

Introduction à Helm

Qu'est-ce que Helm ?

- **Gestionnaire de packages** : APT/YUM pour Kubernetes
- **Templates Kubernetes** : Fichiers YAML paramétrables
- **Charts** : Packages d'applications Kubernetes
- **Releases** : Instances déployées des Charts

Introduction à Helm

Problèmes résolus par Helm

- **Complexité YAML** : Gestion de nombreux fichiers manifest
- **Paramétrage** : Adaptation aux différents environnements
- **Dépendances** : Gestion des composants liés
- **Versioning** : Suivi des versions d'applications
- **Rollback** : Retour arrière simplifié

Introduction à Helm

Écosystème Helm

- **Charts officiels** : Applications populaires pré-packagées
- **Helm Hub** : Registre public de Charts
- **Artifacthub.io** : Nouveau hub communautaire
- **Repositories privés** : Charts d'entreprise

Architecture et Concepts Helm

Composants Helm 3

- **Helm CLI** : Interface en ligne de commande
- **Chart** : Package contenant templates Kubernetes
- **Repository** : Registre de Charts
- **Release** : Instance déployée d'un Chart
- **Values** : Configuration personnalisée

Structure d'un Chart

```
mychart/
├── Chart.yaml          # Métadonnées du chart
├── values.yaml         # Valeurs par défaut
├── charts/             # Dépendances
├── templates/          # Templates Kubernetes
│   ├── deployment.yaml
│   ├── service.yaml
│   ├── configmap.yaml
│   └── NOTES.txt
└── .helmignore         # Fichiers à ignorer
```

Positionnement de Helm

Helm dans l'écosystème Kubernetes

- **Niveau application** : Au-dessus de kubectl
- **Abstraction** : Simplifie le déploiement complexe
- **Standardisation** : Formats et pratiques communes
- **Communauté** : Écosystème riche et actif

Cas d'usage appropriés

- **Applications complexes** : Multiples composants interdépendants
- **Multi-environnements** : Dev, test, prod avec configs différentes
- **Applications tierces** : Bases de données, monitoring, etc.
- **Entreprise** : Standardisation des déploiements

Limitations Helm

- **Courbe d'apprentissage** : Syntaxe de templating complexe
- **Over-engineering** : Complexité excessive pour apps simples
- **Debugging difficile** : Templates imbriqués hard à déboguer
- **Dépendances** : Gestion parfois complexe

Helm vs Kustomize

Philosophies différentes

- **Helm** : Templating avec paramètres dynamiques
- **Kustomize** : Patches et overlays sur YAML de base

Approche Helm (Template-based)

```
# templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "app.fullname" . }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
        resources:
          {{- toYaml .Values.resources | nindent 12 }}
```

Helm vs Kustomize

Approche Kustomize (Patch-based)

```yaml
# base/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: webapp
        image: nginx:1.20

# overlays/production/kustomization.yaml
resources:
- ../../base
patchesStrategicMerge:
- deployment-patch.yaml
images:
- name: nginx
  newTag: 1.21
```

Comparaison Helm vs Kustomize

Tableau comparatif

Aspect	Helm	Kustomize
Apprentissage	Complexe (Go templates)	Simple (YAML patches)
Flexibilité	Très flexible	Modérément flexible
Communauté	Charts pré-faits	Moins de packages
Debugging	Difficile	Plus facile
Dépendances	Natif	Manuel
Versioning	Intégré	Externe (Git)
Integration	Outil séparé	Intégré kubectl

Comparaison Helm vs Kustomize

Quand utiliser Helm

- Applications complexes avec nombreux paramètres
- Réutilisation entre équipes/projets
- Ecosystème de Charts existant
- Gestion de releases avancée

Quand utiliser Kustomize

- Applications simples à moyennes
- Équipes préférant YAML pur
- Workflows GitOps
- Contrôle fin des transformations

Dépôts Helm et Communauté

Écosystème des repositories

- **Artifact Hub** : hub.helm.sh (officiel depuis 2020)
- **Bitnami** : Charts maintenus par Bitnami
- **Stable/Incubator** : Anciens repos officiels (deprecated)
- **Repositories privés** : Harbor, ChartMuseum, Nexus

Artifact Hub - Le hub officiel

- **Centralisation** : Tous les Charts publics
- **Recherche avancée** : Filtres par catégorie, version
- **Sécurité** : Scan de vulnérabilités
- **Documentation** : READMEs et exemples
- **Statistiques** : Popularité et téléchargements

Principales catégories

- **Databases** : PostgreSQL, MySQL, MongoDB, Redis
- **Monitoring** : Prometheus, Grafana, Jaeger
- **CI/CD** : Jenkins, GitLab, ArgoCD
- **Security** : Falco, OPA Gatekeeper

Fonctionnement Communautaire

Gouvernance des Charts

- **Maintainers** : Responsables de Charts spécifiques
- **Reviews** : Process de validation communautaire
- **Standards** : Bonnes pratiques et conventions
- **Tests** : CI/CD automatisé pour validation

Contributions communautaires

- **Issues GitHub** : Bugs et demandes de fonctionnalités
- **Pull Requests** : Corrections et améliorations
- **Documentation** : READMEs et guides d'utilisation
- **Security** : Rapports de vulnérabilités

Qualité et fiabilité

- **Badges** : Indicateurs de qualité sur Artifact Hub
- **Downloads** : Métriques de popularité
- **Versions** : Historique et stabilité
- **Maintien** : Fréquence des mises à jour

Fonctionnement Communautaire

Vérification avant utilisation

```
# Vérifier les détails d'un Chart
helm show chart bitnami/postgresql
helm show values bitnami/postgresql
helm show readme bitnami/postgresql

# Voir l'historique des versions
helm search repo bitnami/postgresql --versions
```

Installation et Configuration Helm

Installation de Helm

```
# Installation via script (Linux/macOS)
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

# Installation via package manager
# macOS
brew install helm

# Ubuntu/Debian
curl https://baltocdn.com/helm/signing.asc | sudo apt-key add -
echo "deb https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm

# Vérification installation
helm version
```

Installation et Configuration Helm

Configuration initiale

```
# Initialisation (Helm 3 - automatique)
helm version

# Complétion bash/zsh
helm completion bash > /etc/bash_completion.d/helm
helm completion zsh > "${fpath[1]}/_helm"

# Configuration de base
helm env
```

Ajout de Dépôts

Gestion des repositories

```
# Lister les repos configurés
helm repo list

# Ajouter des repositories populaires
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo add jetstack https://charts.jetstack.io

# Mettre à jour les repos
helm repo update

# Rechercher dans tous les repos
helm search repo nginx
helm search repo postgres
```

Ajout de Dépôts

Repositories spécialisés

```
# Monitoring
helm repo add grafana https://grafana.github.io/helm-charts
helm repo add elastic https://helm.elastic.co

# Security
helm repo add falcosecurity https://falcosecurity.github.io/charts
helm repo add gatekeeper https://open-policy-agent.github.io/gatekeeper/charts

# Storage
helm repo add rook-release https://charts.rook.io/release

# Service Mesh
helm repo add istio https://istio-release.storage.googleapis.com/charts
helm repo add linkerd https://helm.linkerd.io/stable
```

Repository Privés

Repository avec authentification

```
# Repo avec auth basique
helm repo add mycompany https://charts.company.com \
  --username myuser \
  --password mypass

# Repo avec token
helm repo add private-repo https://charts.example.com \
  --username token \
  --password your-token-here

# Variables d'environnement
export HELM_REPO_USERNAME=myuser
export HELM_REPO_PASSWORD=mypass
helm repo add secure-repo https://secure-charts.com
```

Repository Privés

Repository local ou OCI

```
# Repository local
helm repo add local http://localhost:8080

# OCI Registry (Helm 3.8+)
helm registry login registry.example.com
helm pull oci://registry.example.com/charts/myapp

# Supprimer un repository
helm repo remove bitnami
```

Recherche de Charts

Commandes de recherche

```
# Recherche simple
helm search repo postgres

# Recherche avec versions
helm search repo postgres --versions

# Recherche sur Artifact Hub
helm search hub wordpress

# Recherche avec filtre
helm search repo bitnami/postgres
helm search repo --regexp ".*nginx.*"
```

Recherche de Charts

Analyse d'un Chart

```
# Informations générales
helm show chart bitnami/postgresql

# Valeurs par défaut
helm show values bitnami/postgresql

# Documentation
helm show readme bitnami/postgresql

# Tout en une fois
helm show all bitnami/postgresql

# Templates rendus
helm template my-postgres bitnami/postgresql
```

Installation d'un Package (Release)

Installation basique

```
# Installation simple
helm install my-nginx bitnami/nginx

# Installation avec namespace
helm install my-postgres bitnami/postgresql --namespace database --create-namespace

# Installation avec attente
helm install my-app bitnami/nginx --wait --timeout 300s

# Dry-run pour test
helm install my-nginx bitnami/nginx --dry-run --debug
```

Installation d'un Package (Release)

Installation avec valeurs personnalisées

```
# Valeurs en ligne de commande
helm install my-postgres bitnami/postgresql \
  --set auth.postgresPassword=secretpassword \
  --set primary.persistence.size=20Gi

# Fichier de valeurs
cat > postgres-values.yaml <<EOF
auth:
  postgresPassword: "mysecretpassword"
  database: "myapp"
primary:
  persistence:
    enabled: true
    size: 50Gi
  resources:
    requests:
      memory: 256Mi
      cpu: 250m
EOF

helm install my-postgres bitnami/postgresql -f postgres-values.yaml
```

Gestion des Releases

Commandes de gestion

```
# Lister les releases
helm list
helm list --all-namespaces
helm list --namespace database

# Statut d'une release
helm status my-postgres
helm get all my-postgres

# Historique des releases
helm history my-postgres

# Voir les valeurs utilisées
helm get values my-postgres
helm get values my-postgres --all  # Inclus valeurs par défaut
```

Gestion des Releases

Mise à jour et rollback

```
# Mise à jour avec nouvelles valeurs
helm upgrade my-postgres bitnami/postgresql \
  --set primary.persistence.size=100Gi

# Mise à jour vers nouvelle version Chart
helm upgrade my-postgres bitnami/postgresql --version 12.1.0

# Rollback vers version précédente
helm rollback my-postgres

# Rollback vers version spécifique
helm rollback my-postgres 2
```

Les actions de gestion

Mise à Jour d'une Release

Concept de mise à jour

- **Upgrade in-place** : Modification d'une release existante
- **Préservation état** : Volumes persistants maintenus
- **Rollout progressif** : Stratégies de déploiement respectées
- **Historique** : Versioning automatique des releases

Commandes de base

```
# Mise à jour simple
helm upgrade my-app chart-name

# Mise à jour avec nouvelles valeurs
helm upgrade my-app chart-name --set image.tag=v2.0

# Mise à jour avec fichier values
helm upgrade my-app chart-name -f new-values.yaml

# Mise à jour vers nouvelle version de chart
helm upgrade my-app chart-name --version 2.1.0
```

Options avancées

```
# Upgrade avec attente de completion
helm upgrade my-app chart-name --wait --timeout 600s
```

Stratégies de Mise à Jour

Mise à jour par valeurs

```
# Modifier une image
helm upgrade webapp bitnami/nginx \
  --set image.tag="1.22" \
  --set replicaCount=5

# Ajouter des ressources
helm upgrade webapp bitnami/nginx \
  --set resources.requests.cpu="200m" \
  --set resources.requests.memory="256Mi"

# Modifier la configuration
helm upgrade webapp bitnami/nginx \
  --set-string extraEnvVars[0].name="DEBUG" \
  --set-string extraEnvVars[0].value="true"
```

Mise à Jour d'une Release

Mise à jour avec fichiers multiples

```
# Combiner plusieurs fichiers values
helm upgrade my-app chart-name \
  -f base-values.yaml \
  -f production-values.yaml \
  -f security-patch.yaml

# Valeurs en ligne prioritaires sur fichiers
helm upgrade my-app chart-name \
  -f values.yaml \
  --set image.tag=hotfix-1.2.3
```

Réutilisation des valeurs existantes

```
# Réutiliser valeurs + nouvelles valeurs
helm upgrade my-app chart-name --reuse-values --set newParam=value

# Réinitialiser aux valeurs par défaut puis appliquer
helm upgrade my-app chart-name --reset-values -f new-config.yaml
```

Mécanisme de Rollout

Processus de rollout Helm

```
1. Validation des templates et valeurs
2. Génération des manifests Kubernetes
3. Application des changements (kubectl apply)
4. Attente des conditions de santé (si --wait)
5. Mise à jour du statut de la release
```

Rollout avec hook

```yaml
# Hook pre-upgrade dans un Job
apiVersion: batch/v1
kind: Job
metadata:
  annotations:
    "helm.sh/hook": pre-upgrade
    "helm.sh/hook-weight": "1"
    "helm.sh/hook-delete-policy": hook-succeeded
spec:
  template:
    spec:
      containers:
      - name: db-migration
        image: migrate:latest
        command: ["./migrate", "up"]
      restartPolicy: Never
```

Mécanisme de Rollout

Surveillance du rollout

```
# Suivre le rollout en temps réel
helm upgrade my-app chart-name --wait --debug

# Vérifier le statut après upgrade
helm status my-app

# Voir les événements Kubernetes
kubectl get events --sort-by='.lastTimestamp'

# Surveiller les pods
kubectl get pods -l app.kubernetes.io/instance=my-app --watch
```

Rollback et Historique

Gestion de l'historique

```
# Voir l'historique des releases
helm history my-app

# Historique détaillé
helm history my-app --max 10

# Comparer deux revisions
helm get values my-app --revision 1
helm get values my-app --revision 2
```

Rollback vers version précédente

```
# Rollback automatique vers version précédente
helm rollback my-app

# Rollback vers revision spécifique
helm rollback my-app 3

# Rollback avec attente
helm rollback my-app 2 --wait --timeout 300s

# Rollback dry-run
helm rollback my-app 1 --dry-run
```

Rollback et Historique

Rollback d'urgence

```
# Rollback immédiat en cas de problème critique
helm rollback my-app --force

# Voir les détails d'une revision
helm get all my-app --revision 2

# Nettoyer l'historique (garder seulement N versions)
helm upgrade my-app chart-name --history-max 5
```

Identification des Ressources Actives

Ressources gérées par une release

```
# Voir toutes les ressources d'une release
helm get manifest my-app

# Ressources avec labels Helm
kubectl get all -l app.kubernetes.io/managed-by=Helm
kubectl get all -l app.kubernetes.io/instance=my-app

# Ressources par type
kubectl get deployments,services,configmaps -l app.kubernetes.io/instance=my-app
```

Statut détaillé

```
# Statut complet de la release
helm status my-app

# Notes post-installation
helm get notes my-app

# Hooks exécutés
helm get hooks my-app

# Valeurs utilisées
helm get values my-app --all
```

Rollback et Historique

Debug des ressources

```
# Templates générés
helm get manifest my-app > current-manifests.yaml

# Comparaison avant/après upgrade
helm template my-app chart-name -f new-values.yaml > new-manifests.yaml
diff current-manifests.yaml new-manifests.yaml

# Test de templates
helm template my-app chart-name --debug --dry-run
```

Structure d'un Chart Helm

Anatomie d'un Chart

```
mychart/
├── Chart.yaml              # Métadonnées du chart
├── Chart.lock              # Dépendances lockées
├── values.yaml             # Valeurs par défaut
├── values.schema.json      # Schéma de validation
├── README.md               # Documentation
├── LICENSE                 # Licence
├── .helmignore            # Fichiers à ignorer
├── charts/                # Charts dépendants
│   └── dependency-chart/
├── crds/                  # Custom Resource Definitions
│   └── mycrd.yaml
└── templates/             # Templates Kubernetes
    ├── NOTES.txt          # Notes post-installation
    ├── _helpers.tpl       # Fonctions helper
    ├── deployment.yaml
    ├── service.yaml
    ├── configmap.yaml
    ├── secret.yaml
    ├── ingress.yaml
    └── tests/
        └── test-pod.yaml
```

Structure d'un Chart Helm

Chart.yaml - Métadonnées

```yaml
apiVersion: v2
name: mychart
description: A Helm chart for my application
type: application
version: 0.1.0
appVersion: "1.0"
home: https://example.com
sources:
  - https://github.com/example/mychart
maintainers:
  - name: John Doe
    email: john@example.com
keywords:
  - web
  - nginx
dependencies:
  - name: postgresql
    version: "12.1.0"
    repository: https://charts.bitnami.com/bitnami
    condition: postgresql.enabled
```

Scaffolding d'un Nouveau Projet

Création d'un chart

```
# Créer un nouveau chart
helm create webapp

# Structure générée automatiquement
tree webapp/
webapp/
├── Chart.yaml
├── values.yaml
├── charts/
└── templates/
    ├── NOTES.txt
    ├── _helpers.tpl
    ├── deployment.yaml
    ├── hpa.yaml
    ├── ingress.yaml
    ├── service.yaml
    ├── serviceaccount.yaml
    └── tests/
        └── test-connection.yaml
```

Scaffolding d'un Nouveau Projet

Personnalisation du chart

```
# Valider la structure
helm lint webapp/

# Tester le rendu des templates
helm template my-webapp webapp/

# Installer en mode debug
helm install my-webapp webapp/ --debug --dry-run

# Installation réelle
helm install my-webapp webapp/ --create-namespace --namespace webapp
```

Scaffolding d'un Nouveau Projet

Nettoyage du scaffold

```
# Supprimer les templates non nécessaires
rm webapp/templates/hpa.yaml
rm webapp/templates/ingress.yaml

# Personnaliser values.yaml
cat > webapp/values.yaml <<EOF
replicaCount: 3
image:
  repository: nginx
  tag: "1.21"
  pullPolicy: IfNotPresent
service:
  type: ClusterIP
  port: 80
resources:
  requests:
    cpu: 100m
    memory: 128Mi
EOF
```

Templates et Fonctions

Syntaxe de base Go Templates

```
# templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "webapp.fullname" . }}
  labels:
    {{- include "webapp.labels" . | nindent 4 }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      {{- include "webapp.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      labels:
        {{- include "webapp.selectorLabels" . | nindent 8 }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
        ports:
        - containerPort: {{ .Values.service.targetPort | default 80 }}
        resources:
          {{- toYaml .Values.resources | nindent 12 }}
```

Templates et Fonctions

Fonctions helpers (_helpers.tpl)

```
{{/*
Expand the name of the chart.
*/}}
{{- define "webapp.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" }}
{{- end }}

{{/*
Create a default fully qualified app name.
*/}}
{{- define "webapp.fullname" -}}
{{- if .Values.fullnameOverride }}
{{- .Values.fullnameOverride | trunc 63 | trimSuffix "-" }}
{{- else }}
{{- $name := default .Chart.Name .Values.nameOverride }}
{{- printf "%s-%s" .Release.Name $name | trunc 63 | trimSuffix "-" }}
{{- end }}
{{- end }}

{{/*
Common labels
*/}}
{{- define "webapp.labels" -}}
helm.sh/chart: {{ include "webapp.chart" . }}
{{ include "webapp.selectorLabels" . }}
{{- if .Chart.AppVersion }}
app.kubernetes.io/version: {{ .Chart.AppVersion | quote }}
{{- end }}
app.kubernetes.io/managed-by: {{ .Release.Service }}
{{- end }}
```

Pipelines et Opérateurs

Pipelines de transformation

```
# Transformation de données avec pipelines
spec:
  containers:
  - name: {{ .Chart.Name }}
    image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
    env:
    - name: CONFIG_DATA
      value: {{ .Values.config | toJson | quote }}
    - name: UPPERCASE_NAME
      value: {{ .Values.appName | upper | quote }}
    - name: TRIMMED_VALUE
      value: {{ .Values.description | trim | quote }}
    resources:
      {{- .Values.resources | toYaml | nindent 8 }}
```

Pipelines et Opérateurs

Opérateurs de contrôle

```
# Conditions if/else
{{- if .Values.ingress.enabled }}
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ include "webapp.fullname" . }}
  {{- if .Values.ingress.annotations }}
  annotations:
    {{- toYaml .Values.ingress.annotations | nindent 4 }}
  {{- end }}
spec:
  {{- if .Values.ingress.tls }}
  tls:
  {{- range .Values.ingress.tls }}
    - hosts:
      {{- range .hosts }}
        - {{ . | quote }}
      {{- end }}
      secretName: {{ .secretName }}
  {{- end }}
  {{- end }}
  rules:
  {{- range .Values.ingress.hosts }}
    - host: {{ .host | quote }}
      http:
        paths:
        {{- range .paths }}
        - path: {{ .path }}
          pathType: {{ .pathType }}
          backend:
            service:
              name: {{ include "webapp.fullname" $ }}
              port:
                number: {{ $.Values.service.port }}
        {{- end }}
  {{- end }}
{{- end }}
```

Gestion de Variables Avancée

Valeurs par défaut et override

```yaml
# values.yaml
image:
  repository: nginx
  tag: ""  # Utilise Chart.AppVersion par défaut
  pullPolicy: IfNotPresent

config:
  database:
    host: localhost
    port: 5432
    name: myapp

# Template avec valeurs par défaut
- name: DB_HOST
  value: {{ .Values.config.database.host | default "postgres" }}
- name: DB_PORT
  value: {{ .Values.config.database.port | toString | quote }}
- name: IMAGE_TAG
  value: {{ .Values.image.tag | default .Chart.AppVersion | quote }}
```

Gestion de Variables Avancée

Variables globales et includes

```
# _helpers.tpl - Variables calculées
{{- define "webapp.databaseUrl" -}}
{{- printf "postgresql://%s:%s@%s:%d/%s"
    .Values.config.database.username
    .Values.config.database.password
    .Values.config.database.host
    (.Values.config.database.port | int)
    .Values.config.database.name }}
{{- end }}

# Utilisation dans templates
env:
- name: DATABASE_URL
  value: {{ include "webapp.databaseUrl" . | quote }}
```

Gestion de Variables Avancée

Validation et schémas

```
# values.schema.json
{
  "$schema": "https://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "replicaCount": {
      "type": "integer",
      "minimum": 1,
      "maximum": 10
    },
    "image": {
      "type": "object",
      "properties": {
        "repository": {
          "type": "string",
          "pattern": "^[a-z0-9-./]+$"
        },
        "tag": {
          "type": "string"
        }
      },
      "required": ["repository"]
    }
  },
  "required": ["image"]
}
```