Guide de Déploiement d'Applications sur Kubernetes

2 septembre 2025

Résumé

Ce document est un guide de référence complet et une collection de manifestes YAML générés pour le déploiement d'une application conteneurisée sur un cluster Kubernetes. Il couvre l'installation des prérequis, la génération des fichiers de configuration, ainsi qu'un guide pas-à-pas pour un déploiement sécurisé et professionnel.

Table des matières

1 Introduction et Questions à l'Utilisateur

Pour générer les fichiers de configuration, nous avons besoin de quelques informations sur votre application et votre environnement. Nous allons vous poser ces questions au fur et à mesure pour une approche progressive et efficace.

2 Configuration du Cluster et Détection de l'Environnement

La première étape est de vous assurer que vous disposez d'un cluster Kubernetes opérationnel.

Question à poser à l'utilisateur :

— "Avez-vous déjà un cluster Kubernetes fonctionnel et l'outil `kubectl` installé ? (Oui/Non)"

2.1 Comparaison des clusters locaux

Si vous n'avez pas de cluster, nous vous recommandons d'utiliser une solution légère pour en démarrer un localement. Voici une comparaison pour vous aider à choisir la bonne option.

Solution	Cas d'usage idéal	Avantages	Inconvénients
Minikube	Développement mono-nœud simple.	Très mature, facile à démarrer, fonctionne sur toutes les plate-formes (Windows, macOS, Linux).	Peut être gourmand en ressources, plus lent au démarrage.
Kind (Kubernetes in Docker)	Exécution de clusters multi-nœuds pour le développement et les tests.	Rapide, léger, utilise Docker pour les nœuds, idéal pour l'intégration continue.	Moins adapté pour des cas d'utilisation complexes, dépend de Docker.
K3s	Environnements légers, IoT, edge computing.	Extrêmement rapide et léger, faible consommation de ressources, tout-en-un.	Moins d'options de configuration que Kubernetes standard, communauté plus petite.

2.2 Schéma de l'architecture Kubernetes

Pour mieux comprendre l'architecture, voici un schéma simplifié d'un cluster Kubernetes, montrant la relation entre le plan de contrôle (Master Node) et les nœuds de travail (Worker Nodes).

Le **Master Node** est le cerveau du cluster. Il gère l'état et la coordination des Worker Nodes. Les **Worker Nodes** sont les machines sur lesquelles vos conteneurs (pods) s'exécutent.

2.3 Installation des clusters locaux

Si vous n'avez pas de cluster, suivez les instructions de la solution de votre choix ci-dessous.

2.3.1 Installation de Minikube

Minikube nécessite un pilote de machine virtuelle (comme VirtualBox, HyperKit) ou un moteur de conteneur (Docker, Podman).

Script d'installation pour Linux :

```
#!/bin/bash
curl -LO https://storage.googleapis.com/minikube/releases/latest/
  ↪ minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
echo "D marrage du cluster Minikube..."
minikube start
```

Listing 1 – Script d'installation de Minikube

2.3.2 Installation de Kind

Kind utilise Docker pour exécuter des nœuds Kubernetes. Assurez-vous que Docker est déjà installé et en cours d'exécution.

Script d'installation de Kind :

```
#!/bin/bash
[ $(uname -m) = x86_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl
  ↪ /v0.23.0/kind-linux-amd64
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind
echo "Cr ation d'un cluster Kind..."
kind create cluster
```

Listing 2 – Script d'installation de Kind

2.3.3 Installation de K3s

K3s est un choix excellent pour un cluster léger et rapide.

Script d'installation de K3s :

```
#!/bin/bash
curl -sfL https://get.k3s.io | sh -
echo "V rification de la connexion au cluster..."
sudo kubectl get nodes
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
echo "export KUBECONFIG=/etc/rancher/k3s/k3s.yaml" >> ~/.bashrc
source ~/.bashrc
```

Listing 3 – Script d'installation de K3s

2.4 Si la réponse est "Oui" : Utilisation d'un cluster existant

Passez directement à la section suivante pour utiliser les manifestes générés.

3 Manifestes de Base : Déploiement, Service et Ingress

Cette section génère les fichiers essentiels pour déployer, exposer et router le trafic vers votre application.

Questions à poser à l'utilisateur :

— "Quel est le nom de votre application ? (Ex : `mon-app`)"

— "Quel est le nom de l'image Docker ? (Ex : `mon-app:1.0`)"

— "Combien d'instances (répliques) souhaitez-vous démarrer ? (Ex : `3`)"

— "Sur quel port l'application s'exécute-t-elle à l'intérieur du conteneur ? (Ex : `8080`)"

— "Quel est le nom d'hôte (nom de domaine) pour accéder à l'application ? (Ex : `api.monsite.com`)"

Manifestes générés :

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: <nom_app>-deployment
  labels:
    app: <nom_app>
spec:
  replicas: <replicas_count>
  selector:
    matchLabels:
      app: <nom_app>
  template:
    metadata:
      labels:
        app: <nom_app>
    spec:
      containers:
      - name: <nom_app>
        image: <image_name>
        ports:
        - containerPort: <app_port>
        resources:
          limits:
            memory: "512Mi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /actuator/health
            port: <app_port>
          initialDelaySeconds: 60
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /actuator/health
            port: <app_port>
          initialDelaySeconds: 20
          periodSeconds: 5
---
# service.yaml
apiVersion: v1
kind: Service
```

```yaml
metadata:
  name: <nom_app>-service
spec:
  selector:
    app: <nom_app>
  ports:
    - protocol: TCP
      port: 80
      targetPort: <app_port>
  type: ClusterIP
---
# ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <nom_app>-ingress
  annotations:
    traefik.ingress.kubernetes.io/router.entrypoints: web
spec:
  rules:
  - host: <ingress_hostname>
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: <nom_app>-service
            port:
              number: 80
```

Listing 4 – Fichiers deployment.yaml

4 Organisation et Sécurité

Questions à poser à l'utilisateur :

- "Souhaitez-vous créer un espace de noms dédié pour votre application ? (Oui/Non)"
- "Avez-vous des données de configuration ou des secrets à injecter dans votre application ?"

4.1 Espace de Noms (`Namespace`) et Quotas

La création d'un espace de noms est une bonne pratique pour organiser les ressources. Pour une gestion avancée, vous pouvez ajouter des quotas.

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: <nom_de_l_espace_de_noms>
```

Listing 5 – namespace.yaml

4.2 Configuration et Secrets

Pour externaliser la configuration et sécuriser les informations sensibles. Une bonne pratique est d'utiliser les `Secrets` couplés à un outil comme `Sealed Secrets` de Bitnami pour les stocker de manière chiffrée dans Git.

```
# configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: <nom_app>-config
data:
  # Ajoutez vos paires cl  -valeur ici
  APP_PROFILE: prod
  DB_HOST: "database-service"
---
# secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: <nom_app>-secret
type: Opaque
stringData:
  # Ajoutez vos secrets ici
  DB_PASSWORD: "my-secret-password"
```

Listing 6 – configmap.yaml et secret.yaml

4.3 Règles de Sécurité avancées (`PodSecurityContext`)

Pour renforcer la sécurité, le conteneur peut être forcé à s'exécuter avec un utilisateur non-root, en modifiant la section `template` de votre `deployment.yaml` :

```
    spec:
      securityContext:
        runAsUser: 1000
        runAsGroup: 3000
        fsGroup: 2000
      containers:
      - name: <nom_app>
        # ...
```

Listing 7 – Exemple de PodSecurityContext

5 Gestion du Stockage (Persistent Volumes PVC)

Pour les applications qui ont besoin de conserver leurs données (bases de données, caches, etc.), les Persistent Volumes et les Persistent Volume Claims sont essentiels. Un **PV** est une ressource de stockage dans le cluster, tandis qu'un **PVC** est une requête pour utiliser ce stockage. C'est comme un administrateur qui achète un disque dur (PV) et un utilisateur qui demande un espace de stockage (PVC) sur ce disque.

Question à poser à l'utilisateur :

— "Votre application a-t-elle besoin de stockage persistant ? (Oui/Non)"

— Si "Oui" : "Quelle est la taille de stockage requise ? (Ex : `5Gi`)"

— "Quel est le nom de la classe de stockage à utiliser ? (Ex : `standard-storage`)"

Manifestes générés :

```
# pv.yaml (Persistent Volume)
# Remarque : La plupart des installations modernes de Kubernetes
  ↪ g rent cela automatiquement via une StorageClass.
# Ce manifeste est un exemple de configuration manuelle.
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <nom_app>-pv
spec:
  capacity:
    storage: <storage_size>
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: <storage_class_name>
  hostPath:
    path: "/mnt/data/<nom_app>"
---
# pvc.yaml (Persistent Volume Claim)
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: <nom_app>-pvc
spec:
  storageClassName: <storage_class_name>
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <storage_size>
```

Listing 8 – Fichiers pv.yaml et pvc.yaml

Intégrer le PVC au Déploiement : Pour que votre application utilise le stockage, vous devez modifier la section ‘spec‘ de votre ‘deployment.yaml‘ pour lier le PVC au conteneur.

```
    spec:
      containers:
      - name: <nom_app>
        # ... (reste du conteneur)
        volumeMounts:
        - name: <nom_app>-storage
          mountPath: "/data" # Chemin o  les donn es seront mont es
  ↪ dans le conteneur
      volumes:
      - name: <nom_app>-storage
        persistentVolumeClaim:
          claimName: <nom_app>-pvc
```

Listing 9 – Ajouts à deployment.yaml

6 Assistants d'Automatisation (Makefile)

Un fichier Makefile a été généré pour simplifier les commandes de déploiement et de gestion.

Fichier généré :

```
NAMESPACE ?= <nom_de_l_espace_de_noms>
APP_NAME = <nom_app>

.PHONY: apply delete status

apply:
    @echo "D ploiement des ressources dans le namespace ${NAMESPACE}..."
    kubectl apply -f namespace.yaml
    kubectl apply -f pv.yaml
    kubectl apply -f pvc.yaml
    kubectl apply -f deployment.yaml
    kubectl apply -f service.yaml
    kubectl apply -f ingress.yaml
    kubectl apply -f configmap.yaml
    kubectl apply -f secret.yaml

delete:
    @echo "Suppression des ressources dans le namespace ${NAMESPACE}..."
    kubectl delete -f ingress.yaml
    kubectl delete -f service.yaml
    kubectl delete -f deployment.yaml
    kubectl delete -f pvc.yaml
    kubectl delete -f pv.yaml
    kubectl delete -f secret.yaml
    kubectl delete -f configmap.yaml
    kubectl delete -f namespace.yaml

status:
    @echo "Statut des pods et du stockage dans le namespace ${NAMESPACE}..."
    kubectl get pods --namespace=${NAMESPACE}
    kubectl get pvc --namespace=${NAMESPACE}
    kubectl get services --namespace=${NAMESPACE}
    kubectl get deployments --namespace=${NAMESPACE}
```

Listing 10 – Makefile pour le déploiement

7 Guide d'Utilisation et Dépannage

Ce guide vous aidera à appliquer vos manifestes, à vérifier le statut de votre déploiement et à diagnostiquer les problèmes courants.

Étape 1 : Appliquer les manifestes

Cette commande utilise le fichier `Makefile` pour appliquer l'ensemble de vos manifestes YAML au cluster. Les ressources seront créées dans un ordre précis pour garantir le bon déroulement du déploiement.

Ouvrez votre terminal, naviguez vers le dossier contenant les fichiers générés et exécutez la commande :

```
make apply
```

Étape 2 : Vérifier le déploiement

Après avoir appliqué les manifestes, il est essentiel de vérifier que toutes les ressources ont été créées et que les pods sont en cours d'exécution.

* **Vérifier les pods** Cette commande vous permet de voir le statut de vos conteneurs. Un statut **'Running'** (en cours d'exécution) indique que tout fonctionne correctement.

```
make status
```

* **Vérifier le Service** Assurez-vous que votre service a bien été exposé.

```
kubectl get service <nom_app>-service
```

* **Vérifier l'Ingress** Confirmez que votre Ingress est bien configuré et a reçu une adresse IP externe (si un Ingress Controller est installé et fonctionnel).

```
kubectl get ingress <nom_app>-ingress
```

7.1 Dépannage des problèmes courants

Si votre déploiement échoue, l'un des problèmes suivants est probablement la cause. Voici comment les identifier et les résoudre.

7.1.1 Pod en état 'CrashLoopBackOff'

Un pod dans cet état démarre, puis plante, puis redémarre en boucle. C'est l'un des problèmes les plus fréquents.

* **Causes possibles** :

— L'application elle-même a une erreur de configuration (variable d'environnement manquante, mauvais fichier de configuration, etc.).

— L'application n'arrive pas à démarrer (par exemple, elle ne trouve pas la base de données).

* **Comment diagnostiquer** :

```
# Affiche les logs pour identifier la cause de l'erreur
kubectl logs <nom_du_pod>

# Affiche les événements récents liés au pod
kubectl describe pod <nom_du_pod>
```

7.1.2 Pod en état 'ImagePullBackOff'

Cela signifie que Kubernetes n'a pas pu télécharger l'image Docker spécifiée dans votre manifeste.

* **Causes possibles** :

— Le nom de l'image est mal orthographié ou la balise (tag) est incorrecte.

— L'image se trouve dans un registre privé et Kubernetes n'a pas les identifiants nécessaires pour y accéder.

* **Comment diagnostiquer** :

```
# Vérifie si le nom de l'image est correct dans le manifeste
kubectl describe pod <nom_du_pod>
```

7.1.3 PersistentVolumeClaim (PVC) en état 'Pending'

Le PVC est en attente, car il ne peut pas se lier à un PersistentVolume.

* **Causes possibles** :

— Aucune **'StorageClass'** n'est définie ou le provisionneur est mal configuré.

— Il n'y a pas de PV disponible qui corresponde aux exigences du PVC (taille, modes d'accès).

* **Comment diagnostiquer** :

```
# Examine les événements du PVC pour voir la raison de l'échec
kubectl describe pvc <nom_de_votre_pvc>
```

7.1.4 Problèmes de connexion au cluster

Ce type de problème se produit lorsque `kubectl` ne peut pas communiquer avec l'API du cluster.

* **Exemple d'erreur** :

```
kubectl get nodes
The connection to the server 127.0.0.1:6443 was refused - did you specify the right h
```

* **Causes possibles** :

— Le service du cluster (comme `k3s.service`) n'est pas démarré.

— Le fichier de configuration `KUBECONFIG` a des erreurs de permission ou n'est pas correctement défini.

— Un problème de ressources (manque d'espace disque ou de RAM) empêche le cluster de démarrer complètement.

* **Comment diagnostiquer** :

```
# Vérifier le statut du service K3s
sudo systemctl status k3s

# Vérifier l'espace disque
```

```
df -h

# Re-exporter la variable KUBECONFIG
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
```

7.1.5 Problèmes d'accès via l'Ingress

L'Ingress est créé, mais vous ne pouvez pas accéder à votre application.
* **Causes possibles** :

— Le service est mal configuré et ne route pas le trafic vers les pods.

— Le contrôleur Ingress (comme NGINX ou Traefik) n'est pas installé ou ne fonctionne pas.

* **Comment diagnostiquer** :

```
# Vérifiez que l'Ingress est bien lié à votre service
kubectl get ingress <nom_app>-ingress -o wide

# Si votre contrôleur Ingress est Traefik, vérifiez ses logs
kubectl logs -l app.kubernetes.io/name=traefik -n <votre_namespace_ingress>
```