

Kubernetes pour Débutants: Guide d'Installation et de Configuration

Ce guide détaillé vous accompagnera pas à pas dans l'installation, la configuration et l'utilisation de Kubernetes (K3s) pour les développeurs et administrateurs système débutants. Nous couvrirons l'ensemble du processus, depuis la préparation de l'infrastructure jusqu'à la mise en place de contrôles d'accès avancés.



Préparation de l'Infrastructure Kubernetes (K3s)

Configuration requise

Pour commencer notre aventure Kubernetes, nous devons créer trois machines virtuelles Debian sans interface graphique, qui serviront de base à notre infrastructure :

- kubes-01.local
- kubes-02.local
- kubes-03.local

Chaque VM fonctionnera initialement comme un nœud K3s autonome avant de former un cluster.

Mise à jour du système

```
sudo apt update && sudo apt upgrade -y
```

Installation des outils

```
sudo apt install curl vim git -y
```

Installation de K3s

```
curl -sfL https://get.k3s.io | sh -
```

En cas d'erreur de résolution d'hôte, ajoutez l'entrée suivante dans `/etc/hosts` : `127.0.1.1 kubes-03.local`

Vérifiez l'installation avec : **`sudo kubectl get nodes`** - Chaque VM doit apparaître comme "Ready" et "control-plane".

Déploiement d'Applications Simples

1

Architecture autonome

À cette étape, chaque VM Debian exécute K3s de façon indépendante, sans former de cluster. Cela signifie que chaque machine dispose de son propre environnement Kubernetes isolé.

2


Gestion des ports

Les mêmes NodePorts (30080, 30081, 30306) peuvent être utilisés sur toutes les VMs sans conflit, car chaque VM est un système isolé avec ses propres ports réseau.

3

Préparation au clustering

Cette configuration isolée changera à partir du Job 03, lorsque nous formerons un cluster K3s. À ce moment, les ports NodePort deviendront uniques à l'échelle du cluster entier.

 **Attention :** Lorsque le cluster sera formé (1 master + 2 workers), deux services ne pourront plus utiliser le même NodePort sur des pods différents, car tous les nœuds partageront le même espace de ports.

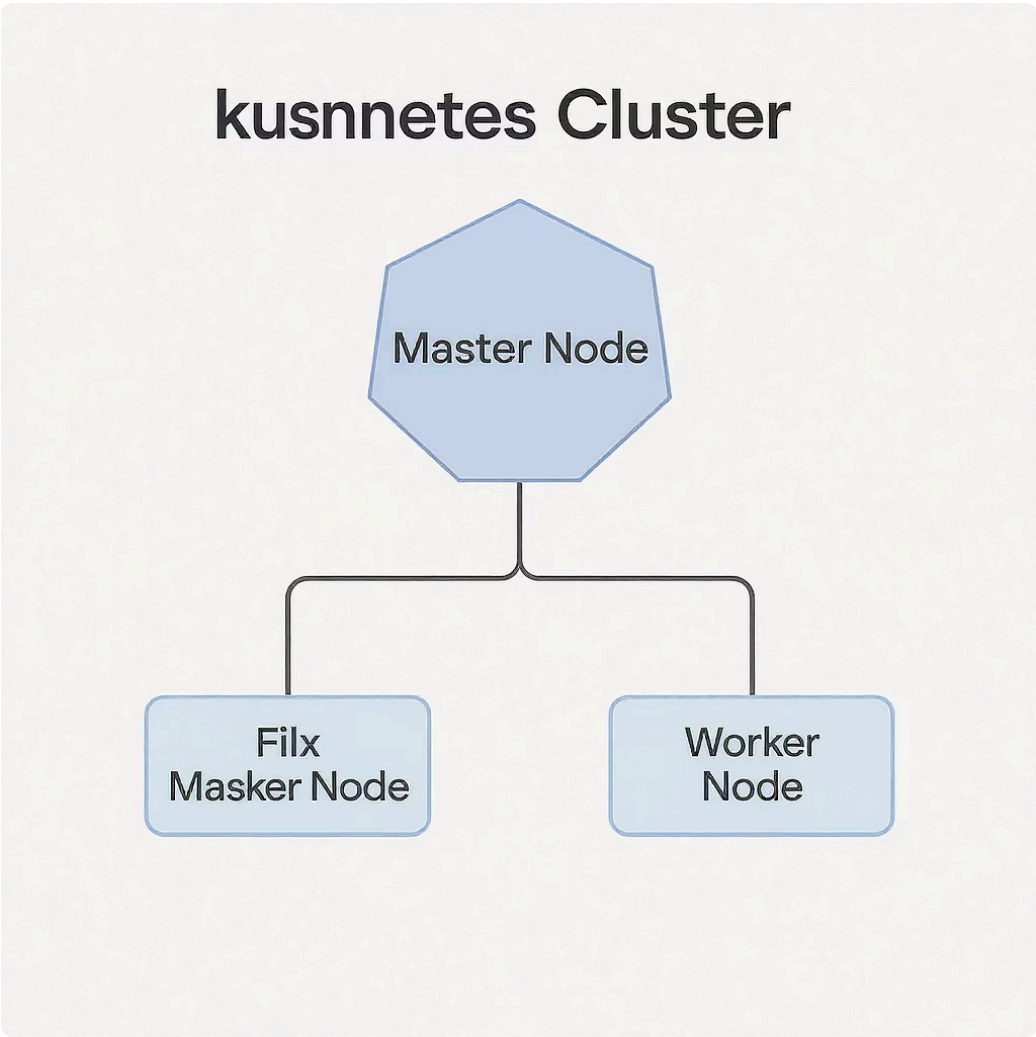
Création du Cluster K3s

Architecture du cluster

Notre objectif est maintenant de transformer nos trois machines isolées en un cluster Kubernetes coordonné avec :

- 1 nœud master (kubes-01.local) - contrôlant le cluster
- 2 nœuds workers (kubes-02.local, kubes-03.local) - exécutant les charges de travail

Cette architecture permet une meilleure répartition des ressources et introduit des fonctionnalités de haute disponibilité.



❏ kubectl ne fonctionnera que sur le nœud master par défaut. Ce n'est pas bloquant, mais peut être configuré sur les workers si nécessaire.

Installation sur le master

```
curl -sfL https://get.k3s.io | sh -
```

Installation sur les workers

```
curl -sfL https://get.k3s.io | K3S_URL=https://IP_MASTER:6443  
K3S_TOKEN="TOKEN" sh -
```

Récupération du token

```
sudo cat /var/lib/rancher/k3s/server/node-token
```

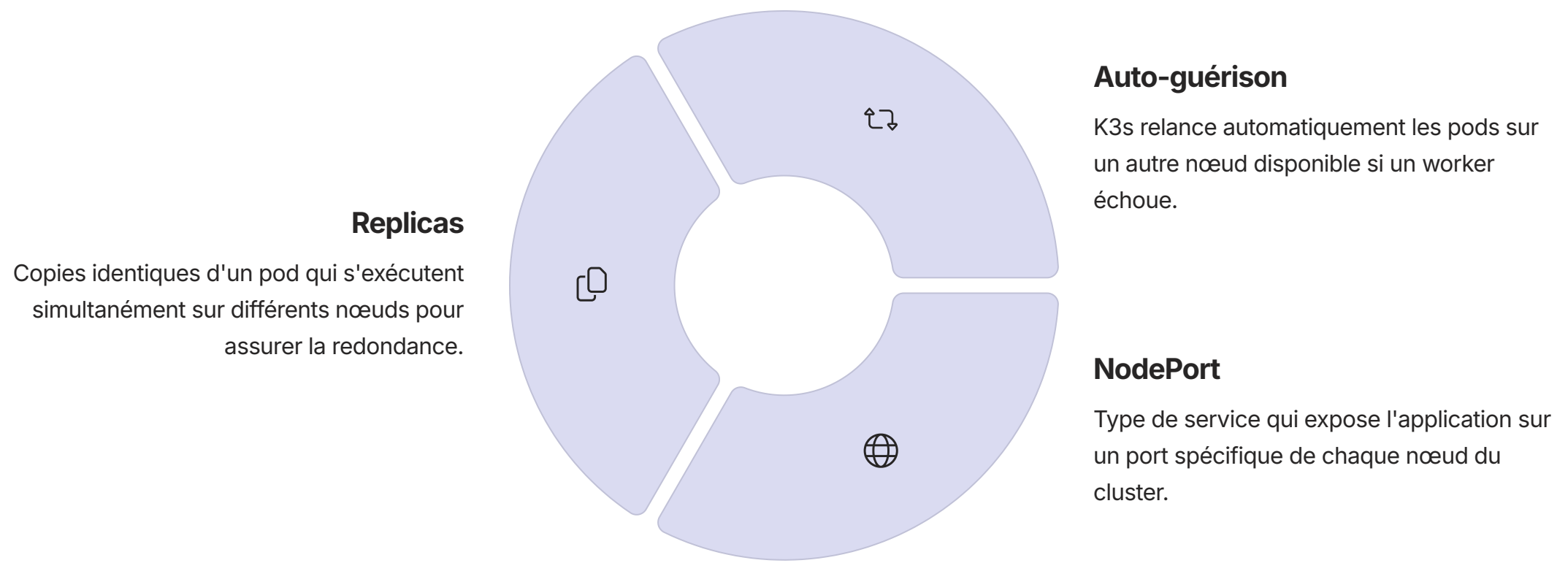
Vérification

```
sudo kubectl get nodes
```

Haute Disponibilité Kubernetes

Principe fondamental de la haute disponibilité

La haute disponibilité (HA) dans Kubernetes garantit que vos applications restent accessibles même si un nœud tombe en panne. C'est l'un des avantages majeurs de l'utilisation d'un cluster.



Application	NodePort	Fonction
NGINX	30080	Serveur web hautes performances
Apache	30081	Serveur web polyvalent
MariaDB	30306	Base de données relationnelle

```

deployment.apps/mariadb unchanged
service/mariadb-service unchanged
dome@kubes-01:~$ sudo kubectl get pods -o wide
NAME                                READY    STATUS    RESTARTS   AGE    IP             NODE                NOMINATED NODE    READINESS GATES
apache-5fd955856f-8gdpr             1/1      Running   0           13m    10.42.0.13     kubes-01.local      <none>             <none>
apache-5fd955856f-dfcxj             1/1      Running   0           13m    10.42.1.2      kubes-02.local      <none>             <none>
apache-5fd955856f-smkhj             1/1      Running   0           13m    10.42.2.3      kubes-03.local      <none>             <none>
mariadb-6d69b65547-4tbrp            1/1      Running   0           13m    10.42.1.3      kubes-02.local      <none>             <none>
mariadb-6d69b65547-ccmfn            1/1      Running   0           13m    10.42.2.4      kubes-03.local      <none>             <none>
mariadb-6d69b65547-jdxlh            1/1      Running   0           13m    10.42.0.12     kubes-01.local      <none>             <none>

```

Démonstration Technique & Commandes

Déploiement et surveillance

Pour mettre en place nos applications avec haute disponibilité :

```

# Déploiement des applications
kubectl apply -f apps-ha.yaml

# Vérification des pods
kubectl get pods -o wide

# Vérification des nœuds
kubectl get nodes

```

Simulation de panne

```

# Éteindre un nœud worker
sudo poweroff

# Vérifier la détection de panne
kubectl get nodes

# Forcer le redéploiement (optionnel)
kubectl drain kubes-02.local \
  --ignore-daemonsets \
  --delete-emptydir-data

# Vérifier la redistribution des pods
kubectl get pods -o wide

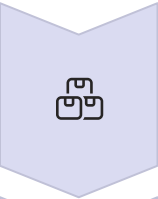
```

Lorsqu'un nœud tombe en panne, Kubernetes détecte automatiquement le problème (statut "NotReady") et commence à redistribuer les pods affectés sur les nœuds restants. Ce processus assure la continuité de service malgré la défaillance matérielle.

Stockage Persistant Kubernetes

Architecture de stockage

L'objectif du stockage persistant est de garantir que les données des applications comme nginx et mariadb ne soient pas perdues lors du redémarrage d'un pod ou d'un nœud.



Pod

Conteneur qui a besoin de stockage



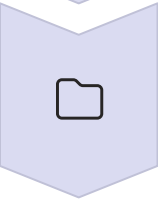
PVC

PersistentVolumeClaim - demande de stockage



PV

PersistentVolume - ressource de stockage



Stockage physique

Dossier local sur le nœud

Configuration requise

Création des dossiers de stockage sur tous les nœuds :

```
sudo mkdir -p /data/nginx
sudo mkdir -p /data/mariadb
sudo chmod -R 777 /data
```

Exemple de PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Montage dans un déploiement

```
volumeMounts:
  - mountPath: "/usr/share/nginx/html"
    name: nginx-storage
volumes:
  - name: nginx-storage
    persistentVolumeClaim:
      claimName: nginx-pvc
```

```
dome@kubes-01:~$ sudo kubectl apply -f nginx-pvc.yaml
persistentvolumeclaim/nginx-pvc created
dome@kubes-01:~$ sudo kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx configured
dome@kubes-01:~$ sudo kubectl apply -f mariadb-pv.yaml
persistentvolume/mariadb-pv created
dome@kubes-01:~$ sudo kubectl apply -f mariadb-pvc.yaml
persistentvolumeclaim/mariadb-pvc created
dome@kubes-01:~$ sudo kubectl apply -f mariadb-deployment.yaml
```

Commandes et Tests de Validation

Application et vérification

```
# Déploiement des ressources de stockage
kubectl apply -f nginx-pvc.yaml
kubectl apply -f nginx-deployment.yaml
kubectl apply -f mariadb-pvc.yaml
kubectl apply -f mariadb-deployment.yaml

# Vérification du stockage
kubectl get pv
kubectl get pvc
kubectl get pods
```

1. Accès au pod

```
kubectl exec -it <nginx-pod> -- /bin/sh
```

2. Création d'un fichier test

```
echo "HELLO PERSISTENCE" > /usr/share/nginx/html/test.html
```

3. Suppression du pod

```
kubectl delete pod <nginx-pod>
```

4. Vérification des données

```
cat /usr/share/nginx/html/test.html
```

✅ **Résultat attendu :** Le fichier "test.html" avec son contenu est préservé même après la recreation du pod, confirmant que le stockage persistant fonctionne correctement.

ConfigMap et Secret

ConfigMap : Configuration externe

Les ConfigMaps permettent de séparer la configuration des conteneurs, facilitant les modifications sans reconstruire l'image.

Création du ConfigMap

```
kubectl create configmap nginx-config \
--from-literal=index.html="
```

Bonjour depuis
ConfigMap !

```
dome@kubes-01:~$ sudo sudo kubectl exec -it $(sudo kubectl get pods -l app=nginx -o js
onpath="{.items[1].metadata.name}") -- cat /usr/share/nginx/html/index.html
<h1>Bonjour depuis ConfigMap !</h1>
```

Montage dans Nginx

```
volumeMounts:
- name: nginx-html
  mountPath: /usr/share/nginx/html/index.html
  subPath: index.html
volumes:
- name: nginx-html
  configMap:
    name: nginx-config
```

Vérification

```
kubectl exec -it -- \
cat /usr/share/nginx/html/index.html
```

Secret : Données sensibles

Les Secrets permettent de stocker et gérer les informations sensibles comme les mots de passe.



```
# Création du Secret
kubectl create secret generic mariadb-secret \
--from-literal=MARIADB_ROOT_PASSWORD=motdepasse
```

```
# Utilisation dans un déploiement
env:
- name: MARIADB_ROOT_PASSWORD
  valueFrom:
    secretKeyRef:
      name: mariadb-secret
      key: MARIADB_ROOT_PASSWORD
```

```
# Vérification
kubectl get pods -l app=mariadb
```

ServiceAccount

```
# serviceaccount-lecteur.yaml
kind: ServiceAccount
metadata:
  name: lecteur-pod

kubectl apply -f serviceaccount-lecteur.yaml
```

RBAC : Contrôle d'Accès par Rôle

Objectif du RBAC

Le contrôle d'accès basé sur les rôles (RBAC) permet de définir précisément quelles actions sont autorisées pour chaque utilisateur ou service du cluster, renforçant ainsi la sécurité.

Création d'un Role

```
# role-pod-reader.yaml
kind: Role
metadata:
  name: pod-reader
rules:
- apiGroups: ["" ]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]

kubectl apply -f role-pod-reader.yaml
```

Génération du token

```
kubectl -n default create token lecteur-pod
```

```

root@kubernetes-01:~# $ curl -k -H "Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6InpGZTl
00F9zNfDlBuk0TElsdFlmU2R0TVvp1M1BHbtZWZLFiaHhyZXlmTzQifQ.eyJhdWQiOiJsiaHR0cHM6Y9rdwJlcm
5ldGvZmlrLzNmF1bHQucc3ZjZlMnsdXN0ZXIubG9jYWwiLCJrM3MiSwiZXhwIjojbHNzUzMdG4MjcxCjYXQioE3
NTMwODQDNzEsIm1lczyt6Imh0dHBzO08va3ViZXJlcy5kZWZhWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwian
RpIjoimTkkY2NjMDMTA0NS00ODk1LTgzYTItZmJmYWE2NDYyYzNhIiwia3ViZXJlcy5kbpyI6eyJuYWll
c3BhY2UiOiJkZWZhWx0Iiwic2VydmllZWFwfjY291bnQiOnsiYmFtZSI6Imxly3RldXIitcG9kIiwidWlkIjoimT
djMTliYWETmjK20S090NzM4LEWzNDgtNzQ2NWlSNtM0MTU4In19LCJuYmYiOiE3NTMwODQ2NzEsInN1YiI6ItnN5
c3RlbTpzXXJ2aWNLYWNj3bvudDpkZWZhWx00mxly3RldXIitcG9kIn0.RyYFvLL4g0Ns5qkAFCCbeTx F9171CqR
UnzVUJ7P5scLABTOYEWCoBLIWU7MWSn4D-HQcjklGpujJe8dGPvhn209ipYmw5BIPicz3wv8bChe46DMTPxvrE
S2xxIoAZ2ak29qt0FI1HhRgLRYXab0wfnvJuIo0gTOK80NrB9SxpbyYaadTw6HK21NoJmdnxzm0mc6_tldLn
tvr_xbTsQYb678ZOHi2JtkBFekAneVTMQpnkFCgwknSdKJzPyVTGT0k10QqzYED00kao3U1V9SFkujsNFpETCE
zIX0tHQK366VZJHoZoKdJKE7KhnpC449p7eatVwN46HHL03gvK61w" https://192.168.40.153:6443/
api/v1/namespaces/default/pods
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "16294"
  },
  "items": [
    {
      "metadata": {
        "name": "apache-5fd955856f-8gdpr",
        "generateName": "apache-5fd955856f-",
        "namespace": "default",

```

Création d'un RoleBinding

```
# rolebinding-lecteur.yaml
kind: RoleBinding
metadata:
  name: read-pods-binding
subjects:
- kind: ServiceAccount
  name: lecteur-pod
roleRef:
  kind: Role
  name: pod-reader
apiGroup: rbac.authorization.k8s.io
```

```
kubectl apply -f rolebinding-lecteur.yaml
```

Tests d'accès

```
# Test autorisé (lecture)
curl -k -H "Authorization: Bearer TOKEN" \
  https://IP:6443/api/v1/namespaces/default/pods
```

```
# Test interdit (suppression)
curl -k -X DELETE -H "Authorization: Bearer TOKEN" \
  https://IP:6443/api/v1/namespaces/default/pods/POD
```

❌ Le test de suppression doit retourner une erreur 403 Forbidden, prouvant que le RBAC limite correctement l'accès aux seules actions autorisées dans le rôle défini.

Présentation de Helm

Objectif de Helm

Helm est le **gestionnaire de paquets** pour Kubernetes, simplifiant le déploiement, la gestion et la personnalisation des applications.

```
dome@kubes-01:~$ curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100 11913  100 11913    0     0  160k      0  --:--:-- --:--:-- --:--:-- 166k
Downloading https://get.helm.sh/helm-v3.18.4-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
[sudo] Mot de passe de dome :
helm installed into /usr/local/bin/helm
```

Simplifie le déploiement : Permet d'installer des applications complexes (WordPress, MariaDB) en une seule commande.

Gère le cycle de vie : Facilite les mises à jour, les restaurations et les suppressions d'applications.

Utilise des Charts : Des "recettes" préconfigurées pour emballer et distribuer les applications Kubernetes.

Installation et Configuration

Pour installer Helm sur votre système :

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

Configurez Helm pour interagir avec votre cluster K3s :

```
mkdir -p ~/.kube
sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
sudo chown $USER:$USER ~/.kube/config
export KUBECONFIG=~/.kube/config
```

Ajoutez un dépôt de Charts (comme Bitnami) pour accéder à un large éventail d'applications prêtes à l'emploi :

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
```

Déploiement et Personnalisation

Déployer une Application

Helm simplifie le déploiement. Utilisez `helm install` pour installer une application prête à l'emploi à partir d'un Chart.

```
helm install monwordpress  
bitnami/wordpress
```

Personnalisation Avancée

Pour des configurations spécifiques, créez un fichier YAML (ex: `wordpress-values.yaml`) et appliquez-le lors de l'installation ou de la mise à jour.

```
wordpressUsername: admin  
wordpressPassword:  
motdepassefort  
service:  
  type: NodePort  
mariadb:  
  auth:  
    database: wordpressdb  
    username: wpuser  
    password: wpsecret
```

Installation avec Personnalisation

Utilisez l'option `-f` pour spécifier votre fichier de valeurs personnalisé.

```
helm install wp-perso  
bitnami/wordpress -f wordpress-  
values.yaml
```

Mettre à Jour une Application

Modifiez les configurations ou les versions directement avec `helm upgrade`. Utilisez `--set` pour les changements rapides ou `-f` pour un fichier complet.

```
helm upgrade wp-perso  
bitnami/wordpress --set  
wordpressPassword=nouveaupass
```

Supprimer une Application

Désinstallez une application déployée par Helm et toutes ses ressources associées avec `helm uninstall`.

```
helm uninstall wp-perso
```

```
dome@kubes-01:~$ helm install monwordpress bitnami/wordpress  
NAME: monwordpress  
LAST DEPLOYED: Mon Jul 21 11:41:41 2025  
NAMESPACE: default  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
NOTES:  
CHART NAME: wordpress  
CHART VERSION: 25.0.3  
APP VERSION: 6.8.2  
  
Did you know there are enterprise versions of the Bitnami catalog? For enhanced secure  
software supply chain features, unlimited pulls from Docker, LTS support, or applicatio  
n customization, see Bitnami Premium or Tanzu Application Catalog. See https://www.arro  
w.com/globalecs/na/vendors/bitnami for more information.  
  
** Please be patient while the chart is being deployed **  
  
Your WordPress site can be accessed through the following DNS name from within your clu  
ster:  
  
    monwordpress.default.svc.cluster.local (port 80)  
  
To access your WordPress site from outside the cluster follow the steps below:  
  
1. Get the WordPress URL by running these commands:  
  
    NOTE: It may take a few minutes for the LoadBalancer IP to be available.  
    Watch the status with: 'kubectl get svc --namespace default -w monwordpress'  
  
    export SERVICE_IP=$(kubectl get svc --namespace default monwordpress --template "{{  
range (index .status.loadBalancer.ingress 0) }}{{ . }}{{ end }}" )
```