

# Guide Complet – Bunkerisation Auth Server K3s (2025)

Aziz Benyamina  
18/12/2025

# Table des matières

Guide Complet – Bunkerisation Auth Server K3s (2025) .....	2
1. Introduction .....	2
2. Objectifs de sécurité .....	2
3. Fondations – Installation et durcissement du cluster K3s.....	2
3.1 Installation de K3s.....	2
3.2 Chiffrement des secrets au repos (Encryption at Rest) .....	3
3.3 Activation des audit logs Kubernetes.....	3
4. Sécurité applicative – Bastion V4 et Helm .....	4
4.1 Image Bastion durcie.....	4
4.2 Déploiement Helm avec restrictions de sécurité.....	4
5. Gestion sécurisée des secrets – Sealed Secrets .....	4
5.1 Justification du choix.....	4
5.2 Installation du contrôleur Sealed Secrets .....	5
5.3 Récupération du certificat public .....	5
5.4 Conversion des secrets applicatifs .....	5
5.5 Déploiement et intégration Helm .....	6
6. Maintenance et exploitation .....	6
7. Conclusion.....	6

# Guide Complet – Bunkerisation Auth Server K3s (2025)

## 1. Introduction

Ce document décrit la conception, le déploiement et le durcissement complet d'un serveur d'authentification basé sur **K3s** et **Kubernetes**, destiné à un environnement de production. L'objectif est d'atteindre un niveau de sécurité élevé en appliquant des principes de défense en profondeur, tout en conservant une charge opérationnelle maîtrisée.

L'infrastructure cible repose sur un VPS unique ou faiblement répliqué, avec des exigences fortes en matière de confidentialité des secrets, de traçabilité des actions et d'isolation applicative.

---

## 2. Objectifs de sécurité

- Déployer un cluster Kubernetes léger mais sécurisé (K3s)
  - Garantir le chiffrement des secrets au repos
  - Assurer une traçabilité complète via les audit logs Kubernetes
  - Exécuter les applications dans des conteneurs non privilégiés
  - Supprimer toute présence de secrets en clair sur le disque ou dans Git
  - Adopter une approche compatible GitOps
- 

## 3. Fondations – Installation et durcissement du cluster K3s

### 3.1 Installation de K3s

Le cluster est initialisé sur un VPS vierge à l'aide de K3s, une distribution Kubernetes légère adaptée aux environnements contraints.

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

Vérification de l'état du nœud :

```
kubectl get nodes
```

Le nœud doit apparaître en état Ready avant de poursuivre.

---

### 3.2 Chiffrement des secrets au repos (Encryption at Rest)

Le chiffrement au repos empêche la lecture des secrets Kubernetes directement depuis les fichiers stockés sur le VPS.

*Génération de la clé de chiffrement*

```
openssl rand -base64 32
```

*Configuration du fournisseur de chiffrement*

Création du fichier /etc/rancher/k3s/encryption-config.yaml avec une clé AES.

Activation dans /etc/rancher/k3s/config.yaml :

**kube-apiserver-arg:**

- "encryption-provider-config=/etc/rancher/k3s/encryption-config.yaml"

Application :

```
sudo systemctl restart k3s
```

Les nouveaux secrets stockés par Kubernetes sont désormais chiffrés sur disque.

---

### 3.3 Activation des audit logs Kubernetes

Les audit logs permettent de tracer toutes les actions effectuées sur le cluster.

- Création de la politique d'audit dans /etc/rancher/k3s/audit-policy.yaml
- Configuration de K3s dans /etc/rancher/k3s/config.yaml

**kube-apiserver-arg:**

- "audit-policy-file=/etc/rancher/k3s/audit-policy.yaml"
- "audit-log-path=/var/lib/rancher/k3s/server/logs/audit.log"
- "audit-log-maxage=30"

Redémarrage du service :

```
sudo systemctl restart k3s
```

Consultation des logs :

```
sudo tail -f /var/lib/rancher/k3s/server/logs/audit.log
```

---

## 4. Sécurité applicative – Bastion V4 et Helm

### 4.1 Image Bastion durcie

L’application Bastion est encapsulée dans une image Docker durcie.

- Image de base minimale
- Exécution sous utilisateur non root
- Aucun accès privilégié au système

Build de l’image :

```
docker build -f Dockerfile.provisioner -t bastion:V4 .
```

Import manuel dans K3s (containerd) :

```
docker save bastion:V4 | sudo k3s ctr images import -
```

---

### 4.2 Déploiement Helm avec restrictions de sécurité

Dans `values-prod.yaml`, les contraintes suivantes sont appliquées :

- `runAsNonRoot: true`
- `readOnlyRootFilesystem: true`
- `capabilities.drop: ["ALL"]`
- Limites CPU et mémoire strictes
- Montage d’un volume `emptyDir` sur `/tmp`

Ces paramètres garantissent l’isolation maximale du conteneur.

---

## 5. Gestion sécurisée des secrets – Sealed Secrets

### 5.1 Justification du choix

Le cluster étant déjà durci (audit logs, chiffrement au repos, images hardened), la sécurisation des secrets applicatifs constitue l’étape logique suivante.

Sealed Secrets a été retenu car :

- il s’intègre nativement à Kubernetes,
- il repose sur un chiffrement asymétrique,
- il est compatible GitOps,

- il présente une faible complexité opérationnelle.

Une solution comme HashiCorp Vault n'a pas été retenue, car elle introduit une charge d'exploitation équivalente à celle du cluster Kubernetes lui-même, ce qui est disproportionné pour un nombre limité de services.

---

## 5.2 Installation du contrôleur Sealed Secrets

```
kubectl apply -f https://github.com/bitnami-labs/sealed-secrets/releases/latest/download/controller.yaml
```

Vérification :

```
kubectl get pods -n kube-system | grep sealed-secrets
```

---

## 5.3 Récupération du certificat public

```
kubectl get secret -n kube-system \
  -l sealedsecrets.bitnami.com/sealed-secrets-key \
  -o yaml | \
  grep tls.crt | \
  awk '{print $2}' | \
  base64 -d > sealed-secrets-public.pem
```

Ce certificat permet uniquement le chiffrement des secrets.

---

## 5.4 Conversion des secrets applicatifs

Création temporaire d'un Secret Kubernetes en clair :

```
apiVersion: v1
kind: Secret
metadata:
  name: platform-prod-secrets
  namespace: default
type: Opaque
stringData:
  MASTER_DB_PASSWORD: motdepassefort
  JWT_SECRET: supersecretjwt
```

Chiffrement avec kubeseal :

```
kubeseal \
  --cert sealed-secrets-public.pem \
  --format yaml \
  < secret-temp.yaml > sealed-secret.yaml
```

Suppression immédiate des fichiers en clair :

```
rm -f secret-temp.yaml  
rm -f values-secrets.yaml  
rm -f .env
```

---

## 5.5 Déploiement et intégration Helm

```
kubectl apply -f sealed-secret.yaml
```

Les charts Helm référencent uniquement le secret existant :

```
envFrom:  
- secretRef:  
  name: platform-prod-secrets
```

---

## 6. Maintenance et exploitation

- Redémarrage applicatif : kubectl rollout restart deployment platform-prod-bastion
  - Surveillance : kubectl get pods -w
  - Consultation des logs d'audit Kubernetes
- 

## 7. Conclusion

Cette architecture combine durcissement système, sécurité Kubernetes et bonnes pratiques applicatives. L'absence totale de secrets en clair, l'isolation forte des conteneurs et la traçabilité complète des actions permettent une exploitation en conditions de production avec un niveau de sécurité élevé, adapté aux environnements sensibles.