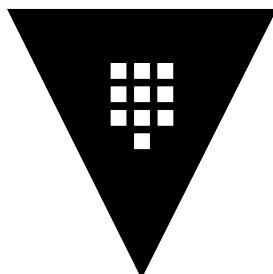


# Mastering Vault HashiCorp

Complete Guide

HashiCorp Vault



HashiCorp  
**Vault**

MARYEM CHERIF

*Network Security Engineer*

# Contents

<b>1</b>	<b>Introduction à HashiCorp Vault</b>	<b>1</b>
1.1	Qu'est-ce que Vault? . . . . .	1
1.1.1	Pourquoi utiliser Vault? . . . . .	1
1.2	Architecture de Vault . . . . .	2
1.2.1	Composants principaux . . . . .	2
1.2.2	Concepts clés . . . . .	3
<b>2</b>	<b>Installation et Configuration</b>	<b>4</b>
2.1	Prérequis . . . . .	4
2.2	Installation de Vault . . . . .	4
2.2.1	Installation sur Linux . . . . .	4
2.2.2	Installation avec Docker . . . . .	5
2.3	Démarrage de Vault en mode Dev . . . . .	5
<b>3</b>	<b>Premiers Pas avec Vault</b>	<b>8</b>
3.1	Utilisation de l'interface CLI . . . . .	8
3.1.1	Écrire et lire des secrets . . . . .	8
3.1.2	Mettre à jour et supprimer des secrets . . . . .	9
3.2	Interface Web UI . . . . .	9
<b>4</b>	<b>Secrets Engines</b>	<b>11</b>
4.1	Introduction aux Secrets Engines . . . . .	11
4.1.1	Types de Secrets Engines . . . . .	11
4.2	KV Secrets Engine . . . . .	11
4.2.1	KV Version 1 vs Version 2 . . . . .	11
4.3	Database Secrets Engine . . . . .	12
4.3.1	Configuration pour PostgreSQL . . . . .	12
4.3.2	Avantages des credentials dynamiques . . . . .	14
<b>5</b>	<b>Méthodes d'Authentification</b>	<b>15</b>
5.1	Introduction aux Auth Methods . . . . .	15
5.1.1	Auth methods courantes . . . . .	15
5.2	Token Authentication . . . . .	15
5.3	UserPass Authentication . . . . .	16
5.4	AppRole Authentication . . . . .	16
<b>6</b>	<b>Policies et Contrôle d'Accès</b>	<b>19</b>
6.1	Comprendre les Policies . . . . .	19
6.1.1	Structure d'une policy . . . . .	19

6.1.2	Capabilities disponibles . . . . .	19
6.2	Créer et appliquer des Policies . . . . .	20
6.2.1	Policy pour développeur . . . . .	20
<b>7</b>	<b>Configuration Production</b>	<b>22</b>
7.1	Définitions Dev vs Production . . . . .	22
7.2	Configuration du serveur . . . . .	22
7.3	Initialisation et déverrouillage (Unsealing) . . . . .	23
7.4	Audit Logging . . . . .	24
<b>8</b>	<b>Projet Pratique: Application Web Sécurisée</b>	<b>26</b>
8.1	Architecture du projet . . . . .	26
8.2	Préparation de l'environnement . . . . .	27
8.3	Configuration de Vault . . . . .	28
8.4	Application Python avec HashiCorp Vault . . . . .	30
8.4.1	Structure du projet . . . . .	30
8.4.2	Client Vault en Python . . . . .	30
8.4.3	Application Flask principale . . . . .	32
8.4.4	Installation et démarrage . . . . .	34
8.4.5	Tests des endpoints . . . . .	34
8.4.6	Explications pour débutants . . . . .	35
8.4.7	Lien GitHub . . . . .	35
<b>9</b>	<b>Haute Disponibilité et Clustering</b>	<b>37</b>
9.1	Introduction . . . . .	37
9.2	Architecture HA . . . . .	37
9.2.1	Explication des composants . . . . .	38
9.3	Configuration d'un cluster Vault avec Consul . . . . .	38
9.3.1	Étape 1 : Configurer le stockage HA . . . . .	38
9.3.2	Étape 2 : Déploiement avec Docker Compose . . . . .	39
9.3.3	Étape 3 : Vérifier le cluster . . . . .	40
9.4	Configuration Auto-Unseal avec AWS KMS . . . . .	40
9.4.1	Étape 1 : Créer la clé KMS . . . . .	40
9.4.2	Étape 2 : Configurer Vault . . . . .	40
9.4.3	Étape 3 : Initialiser Vault . . . . .	41
<b>10</b>	<b>Intégration de Vault avec Kubernetes</b>	<b>42</b>
10.1	Déploiement de Vault sur un cluster Kubernetes mono-nœud . . . . .	42
10.1.1	Installation avec Helm . . . . .	42
10.2	Création des Persistent Volumes (PV) et Persistent Volume Claims (PVC) . . . . .	43
10.2.1	Création des volumes sur le nœud . . . . .	43
10.2.2	Définition du PV et PVC . . . . .	43
10.3	Initialisation et Unseal du Pod Vault . . . . .	44
10.3.1	Initialisation . . . . .	44
10.3.2	Déverrouillage (Unseal) . . . . .	44
10.4	Authentification Kubernetes . . . . .	44
10.4.1	Activation et configuration . . . . .	45
10.4.2	Création d'une policy et d'un rôle . . . . .	45
10.5	Injection des secrets avec Vault Agent Injector . . . . .	45

10.5.1 Exemple de pod utilisant Vault Agent Injector . . . . .	45
<b>11 Monitoring et Troubleshooting de Vault</b>	<b>47</b>
11.1 Collecte des métriques Vault . . . . .	47
11.1.1 Configuration de la telemetry pour Prometheus . . . . .	47
11.1.2 Scraper les métriques depuis Vault . . . . .	48
11.1.3 Visualisation avec Grafana . . . . .	48
11.2 Troubleshooting courant . . . . .	49
11.2.1 Vault sealed après redémarrage . . . . .	49
11.2.2 Performance dégradée . . . . .	49
11.2.3 Problèmes de connexion . . . . .	50
<b>12 Sécurité Avancée</b>	<b>51</b>
12.1 Rotation des secrets . . . . .	51
12.1.1 Rotation automatique des clés . . . . .	51
12.1.2 Révocation de secrets . . . . .	52
12.2 Response Wrapping . . . . .	52
12.2.1 Principe de fonctionnement . . . . .	52
12.3 Sentinel Policies (Vault Enterprise) . . . . .	53
12.3.1 Contrôle d'accès avancé . . . . .	53
<b>13 Bonnes Pratiques</b>	<b>55</b>
13.1 Gestion des secrets . . . . .	55
13.2 Architecture recommandée . . . . .	56
13.3 Checklist de sécurité . . . . .	57
<b>14 Cas d'Usage Avancés</b>	<b>59</b>
14.1 PKI (Public Key Infrastructure) . . . . .	59
14.1.1 Étapes de configuration . . . . .	59
14.2 SSH Secrets Engine . . . . .	60
14.2.1 Configuration des certificats SSH . . . . .	60
14.3 TOTP (Time-based One-Time Password) . . . . .	61
14.3.1 Configuration TOTP . . . . .	61
<b>15 Conformité et Audit</b>	<b>63</b>
15.1 Configuration de l'audit . . . . .	63
15.1.1 Principe des audit devices . . . . .	63
15.1.2 Multiples destinations d'audit . . . . .	63
15.2 Analyse des logs d'audit . . . . .	64
15.2.1 Exemples d'analyses avec jq . . . . .	64
15.3 Rapports de conformité . . . . .	65
15.3.1 Génération automatique de rapports . . . . .	65
<b>16 Commandes Utiles</b>	<b>68</b>
16.1 Commandes CLI essentielles . . . . .	68
<b>17 Guide de Dépannage</b>	<b>70</b>
17.1 Problèmes courants et solutions . . . . .	70
<b>18 Glossaire</b>	<b>71</b>

# Chapter 1

## Introduction à HashiCorp Vault

### 1.1 Qu'est-ce que Vault?

HashiCorp Vault est une solution moderne de gestion des secrets et de protection des données sensibles. Dans les environnements DevOps et de cybersécurité, la gestion des secrets (mots de passe, clés API, certificats, tokens) est un défi critique. Vault apporte une approche centralisée, sécurisée et automatisée pour stocker, distribuer et contrôler l'accès aux informations sensibles.

#### 1.1.1 Pourquoi utiliser Vault?

Sans un outil comme Vault, les organisations sont confrontées à de nombreux problèmes de sécurité et de gestion opérationnelle.

##### Problèmes fréquents sans Vault

- Secrets codés en dur dans le code source, ce qui augmente le risque de fuite
- Partage de mots de passe par email ou chat, non sécurisé et difficilement traçable
- Absence de rotation automatique des secrets, entraînant une exposition prolongée
- Pas de traçabilité des accès aux secrets, compliquant l'audit et la conformité
- Gestion manuelle sujette aux erreurs humaines

Vault résout ces problèmes en offrant des mécanismes robustes pour la **sécurité, la rotation et l'audit des secrets**.

## Solutions apportées par Vault

- **Stockage centralisé et sécurisé** : Tous les secrets sont conservés dans un emplacement unique et chiffré.
- **Chiffrement automatique** : Les données sont chiffrées au repos et en transit, assurant leur confidentialité.
- **Rotation automatique** des credentials : Réduit le risque d'exploitation en cas de fuite.
- **Audit logging complet** : Toutes les requêtes et réponses sont enregistrées pour la traçabilité et la conformité.
- **Contrôle d'accès granulaire (ACL)** : Les utilisateurs et applications n'ont accès qu'aux secrets nécessaires.
- **Génération dynamique de secrets** : Création de secrets à durée de vie limitée pour minimiser l'exposition.

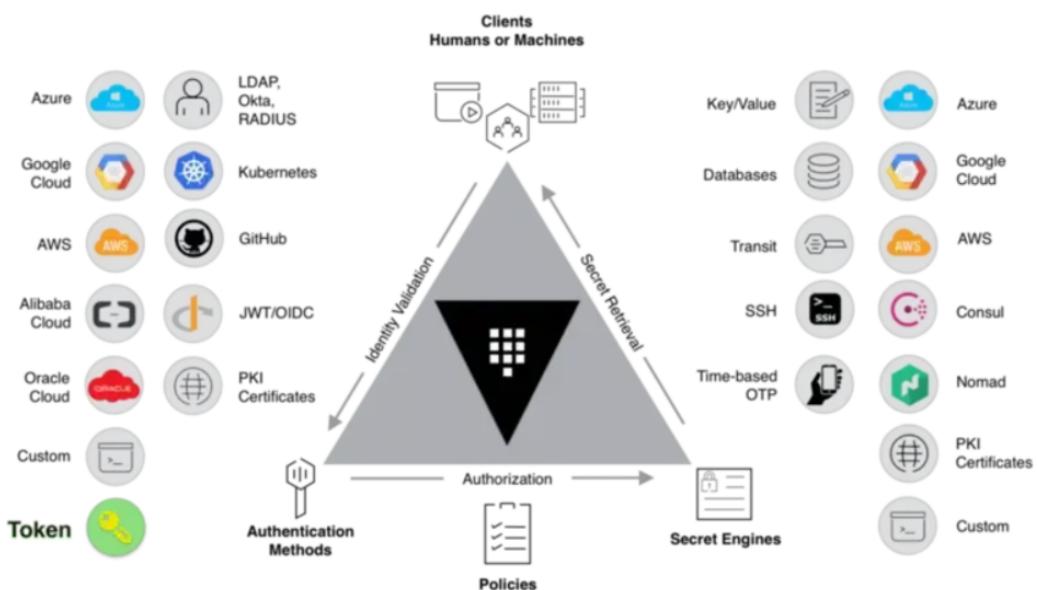


Figure 1.1: Vue d'ensemble de l'architecture de HashiCorp Vault

## 1.2 Architecture de Vault

Vault repose sur une architecture modulaire et sécurisée, permettant à la fois la haute disponibilité, la résilience et la protection des données.

### 1.2.1 Composants principaux

- **Storage Backend** : Fournit le stockage persistant des données chiffrées (ex. Consul, etcd, filesystem). Il assure la durabilité et la résilience des secrets.

- **Barrier** : Couche de chiffrement qui protège toutes les données et empêche l'accès tant que Vault est scellé.
- **Secrets Engines** : Modules spécialisés qui stockent, génèrent ou chiffrent différents types de secrets (KV, Database, AWS, Transit, etc.).
- **Auth Methods** : Méthodes d'authentification pour valider l'identité des utilisateurs ou applications (tokens, LDAP, AWS, Kubernetes, AppRole).
- **Audit Devices** : Enregistre tous les accès et opérations sur Vault pour garantir la traçabilité et faciliter les audits.
- **Policies** : Règles de contrôle d'accès définissant qui peut faire quoi et sur quel secret.

### 1.2.2 Concepts clés

**Seal / Unseal** : Vault démarre toujours dans un état “sealed” (scellé), où toutes les données sont chiffrées et inaccessibles. Pour le rendre opérationnel, il faut le “unseal” avec un nombre requis de clés de déchiffrement, garantissant ainsi la sécurité des données même en cas de compromission du serveur.

**Root Token** : Jeton initial possédant tous les priviléges, utilisé uniquement pour les opérations critiques et la configuration initiale. Il doit être révoqué après l'initialisation pour limiter les risques.

**Lease** : Durée de vie assignée à chaque secret. Après expiration, le secret est automatiquement révoqué, réduisant la fenêtre d'exposition et améliorant la sécurité globale.

# Chapter 2

## Installation et Configuration

### 2.1 Prérequis

- Système Linux/MacOS/Windows
- Docker (optionnel mais recommandé pour les tests)
- 1 GB RAM minimum
- Connaissances de base en ligne de commande

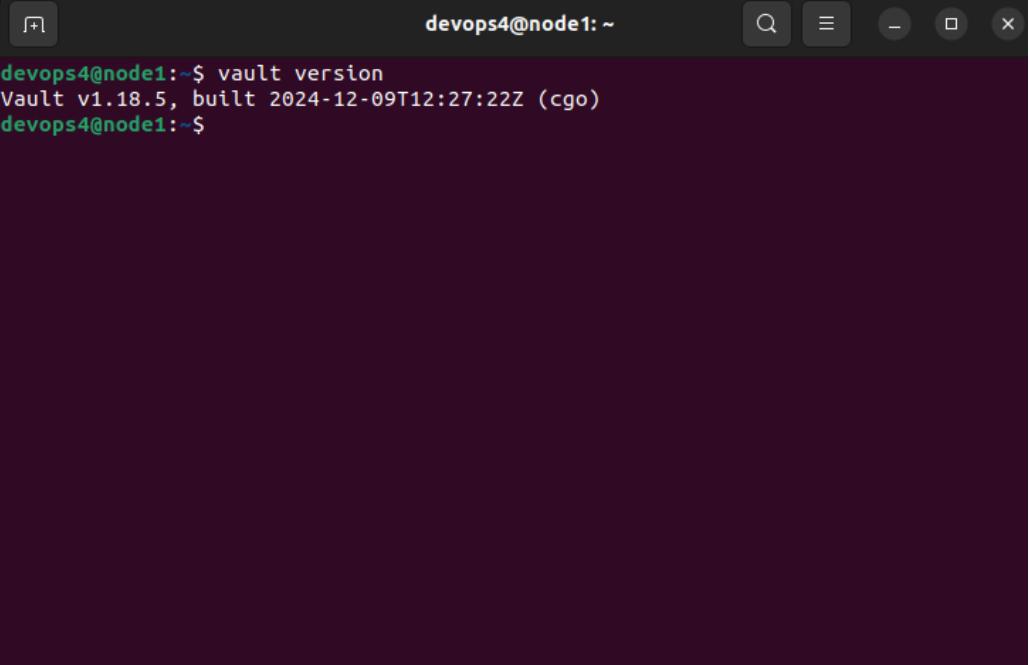
### 2.2 Installation de Vault

- Documentation officielle installation sur : <https://developer.hashicorp.com/vault/install>

#### 2.2.1 Installation sur Linux

Listing 2.1: Installation de Vault sur Linux

```
1 # T l charger Vault
2 wget https://releases.hashicorp.com/vault/1.15.0/vault_1.15.0_linux_amd64.
   zip
3
4 # D compresser
5 unzip vault_1.15.0_linux_amd64.zip
6
7 # D placer dans le PATH
8 sudo mv vault /usr/local/bin/
9
10 # V rifier l'installation
11 vault version
```



```
devops4@node1:~$ vault version
Vault v1.18.5, built 2024-12-09T12:27:22Z (cgo)
devops4@node1:~$
```

Figure 2.1: Sortie de la commande vault version

### 2.2.2 Installation avec Docker

Listing 2.2: Lancement de Vault avec Docker

```
1 # Mode d'veloppement (NON PRODUCTION!)
2 docker run --cap-add=IPC_LOCK -d --name=vault-dev \
3   -p 8200:8200 \
4   -e 'VAULT_DEV_ROOT_TOKEN_ID=myroot' \
5   hashicorp/vault:latest
6
7 # V rifier que le conteneur fonctionne
8 docker ps
```

## 2.3 Démarrage de Vault en mode Dev

### Attention

Le mode développement est UNIQUEMENT pour les tests. Ne jamais utiliser en production!

- Le serveur démarre unsealed
- Stockage en mémoire (perte des données au redémarrage)
- Un seul unsealed key
- Communication HTTP (non chiffrée)

Listing 2.3: Démarrage de Vault en mode développement

```
1 # D marrer le serveur en mode dev
```

```

2 vault server -dev
3
4 # Dans un autre terminal, configurer les variables d'environnement
5 export VAULT_ADDR='http://127.0.0.1:8200'
6 export VAULT_TOKEN='root' # Token affiché au démarrage
7
8 # Tester la connexion
9 vault status

```

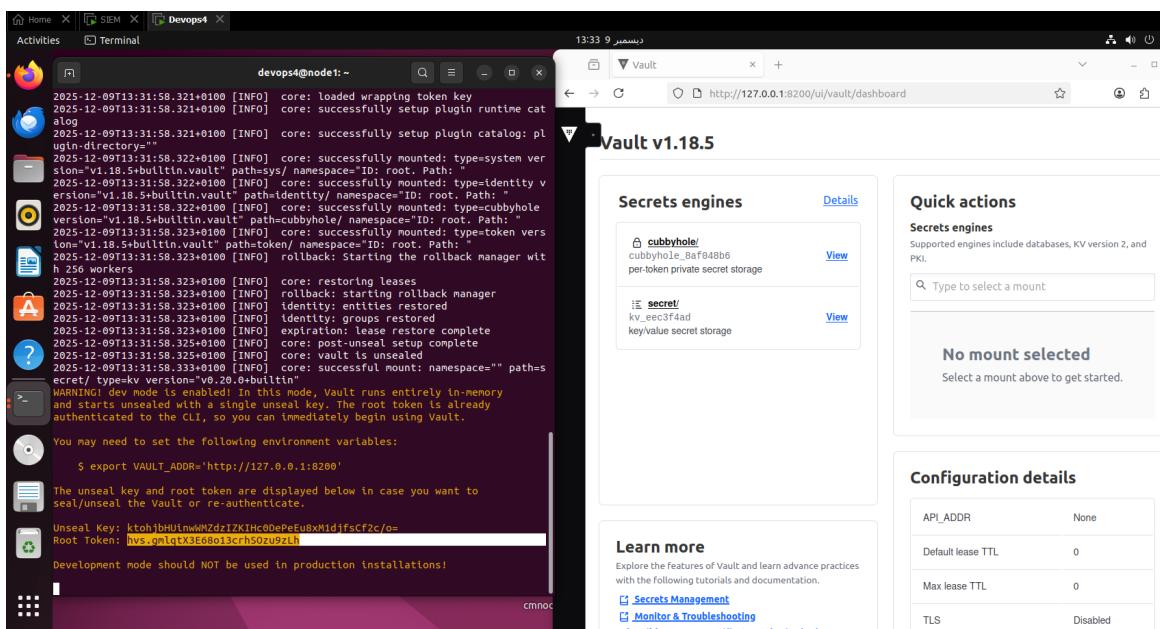
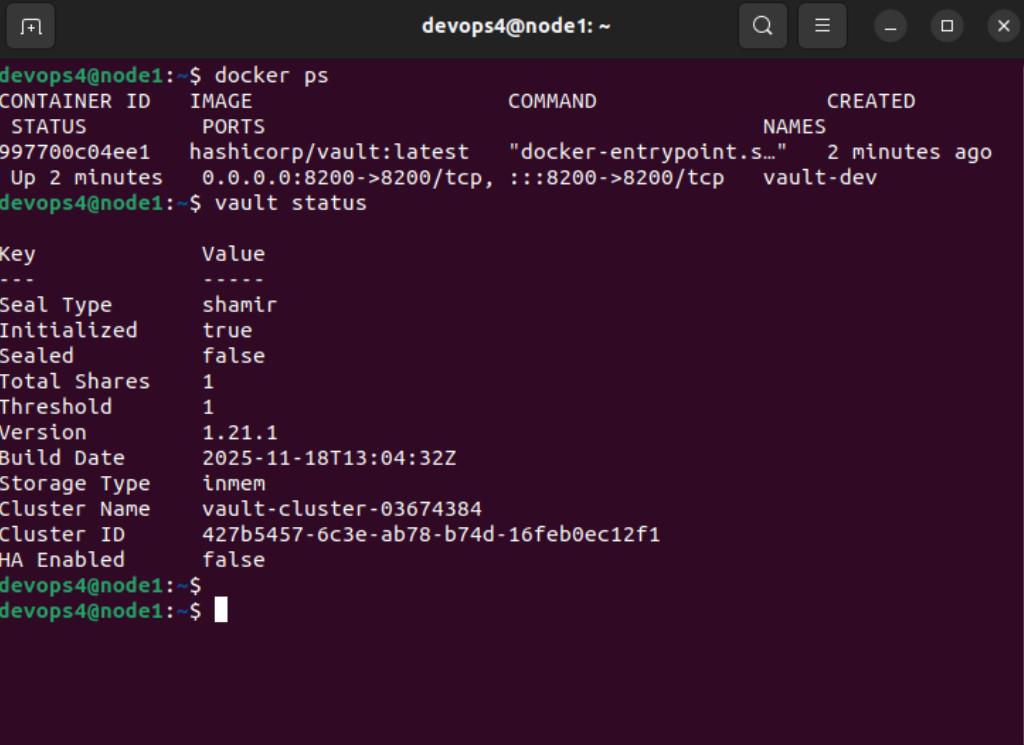


Figure 2.2: Démarrage du serveur Vault en mode développement



The screenshot shows a terminal window with the title "devops4@node1: ~". The window contains the following text:

```
devops4@node1:~$ docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED
STATUS         PORTS          NAMES
997700c04ee1  hashicorp/vault:latest "docker-entrypoint.s..."  2 minutes ago
Up 2 minutes   0.0.0.0:8200->8200/tcp, :::8200->8200/tcp  vault-dev
devops4@node1:~$ vault status
Key          Value
---          ---
Seal Type    shamir
Initialized   true
Sealed        false
Total Shares  1
Threshold    1
Version      1.21.1
Build Date   2025-11-18T13:04:32Z
Storage Type  inmem
Cluster Name  vault-cluster-03674384
Cluster ID   427b5457-6c3e-ab78-b74d-16feb0ec12f1
HA Enabled   false
devops4@node1:~$
```

Figure 2.3: Sortie de la commande vault status

# Chapter 3

## Premiers Pas avec Vault

### 3.1 Utilisation de l'interface CLI

#### 3.1.1 Écrire et lire des secrets

Listing 3.1: Opérations de base avec les secrets

```
1 # créer un secret
2 vault kv put secret/myapp/config \
3     username=admin \
4     password=supersecret123
5
6 # Lire un secret
7 vault kv get secret/myapp/config
8
9 # Lire un champ spécifique
10 vault kv get -field=password secret/myapp/config
11
12 # Lister les secrets
13 vault kv list secret/myapp/
```

```
devops4@node1:~$ vault kv put secret/myapp/config username=admin password=supersecret123
===== Secret Path =====
secret/data/myapp/config

===== Metadata =====
Key          Value
---          -----
created_time 2025-12-09T13:36:59.332344797Z
custom_metadata <nil>
deletion_time n/a
destroyed      false
version        1
```

Figure 3.1: vault kv put output

```
devops4@node1:~$ vault kv get secret/myapp/config
===== Secret Path =====
secret/data/myapp/config

===== Metadata =====
Key          Value
---          -----
created_time 2025-12-09T13:36:59.332344797Z
custom_metadata <nil>
deletion_time n/a
destroyed      false
version        1

===== Data =====
Key          Value
---          -----
password     supersecret123
username     admin
```

Figure 3.2: Lecture d'un secret avec vault kv get

### 3.1.2 Mettre à jour et supprimer des secrets

Listing 3.2: Mise à jour et suppression de secrets

```
1 # Mettre à jour (ajouter/modifier des champs)
2 vault kv patch secret/myapp/config \
3   api_key=abc123xyz
4
5 # Supprimer un secret
6 vault kv delete secret/myapp/config
7
8 # Voir l'historique des versions (KV v2)
9 vault kv metadata get secret/myapp/config
10
11 # Restaurer une version précédente
12 vault kv undelete -versions=1 secret/myapp/config
```

## 3.2 Interface Web UI

Vault dispose d'une interface web accessible par défaut sur <http://127.0.0.1:8200/ui>

The screenshot shows the main dashboard of Vault v1.21.1. On the left, a sidebar menu includes Dashboard, Secrets Engines (selected), Secrets Recovery (Enterprise), Access, Policies, Tools, Monitoring, Client Count, and Seal Vault. The main content area has three main sections: 'Secrets engines' (listing 'cubbyhole/' and 'secret/'), 'Quick actions' (with a search bar and message 'No mount selected'), and 'Configuration details' (table showing API\_ADDR: None, Default lease TTL: 0, Max lease TTL: 0, and TLS: Disabled). A 'Learn more' section at the bottom provides links to 'Secrets Management' and 'Monitor & Troubleshooting'.

Figure 3.3: Dashboard principal de l'interface Web

The screenshot shows the 'Secrets / secret / myapp / config' page. The sidebar is identical to Figure 3.3. The main content shows a table for 'myapp/config' with two entries: 'password' (Value: supersecret123) and 'username' (Value: admin). The table includes columns for Key, Value, Delete, Destroy, Copy, Version (Version 1 created Dec 09, 2025 02:36 PM), and Create new version. Navigation tabs include Overview, Secret (selected), Metadata, Paths, and Version History.

Figure 3.4: Gestion des secrets dans l'interface Web

# Chapter 4

## Secrets Engines

### 4.1 Introduction aux Secrets Engines

Les Secrets Engines sont des composants Vault qui stockent, génèrent ou chiffrent des données. Chaque engine est monté à un chemin spécifique.

#### 4.1.1 Types de Secrets Engines

- **KV (Key-Value)**: Stockage simple de paires clé-valeur
- **Database**: Génération dynamique de credentials pour bases de données
- **AWS**: Génération dynamique de credentials AWS
- **PKI**: Génération et gestion de certificats X.509
- **Transit**: Chiffrement as-a-service
- **SSH**: Génération de certificats SSH

### 4.2 KV Secrets Engine

#### 4.2.1 KV Version 1 vs Version 2

Fonctionnalité	KV v1	KV v2
Versioning (historique des secrets)	Non	Oui
Soft delete (suppression temporaire)	Non	Oui
TTL sur secrets	Non	Oui
Check-and-Set (verrouillage de version)	Non	Oui

Table 4.1: Comparaison KV v1 et KV v2

Listing 4.1: Activer KV v2

```
1 # Activer KV v2
2 vault secrets enable -path=secret-v2 kv-v2
3
```

```

4 # Ajouter un secret avec m t adonn es
5 vault kv put secret-v2/prod/database username=dbadmin password=
   SecurePass123 max_ttl=24h
6
7 # Lire le secret
8 vault kv get secret-v2/prod/database
9
10 # Voir l'historique des versions
11 vault kv metadata get secret-v2/prod/database

```

## 4.3 Database Secrets Engine

Le Database Secrets Engine permet de créer des credentials dynamiques pour des bases de données comme PostgreSQL.

Principe :

- Vault génère automatiquement des comptes avec un mot de passe temporaire.
- Chaque application/utilisateur a ses propres credentials.
- Rotation et révocation automatiques.

### 4.3.1 Configuration pour PostgreSQL

Listing 4.2: Configuration du Database Secrets Engine avec PostgreSQL

```

1 #!/bin/bash
2
3 # --- Variables ---
4 VAULT_ADDR='http://127.0.0.1:8200'
5 VAULT_TOKEN='myroot'
6 DB_USER='vaultadmin'
7 DB_PASS='vaultpassword'
8 DB_NAME='mydb'
9 ROLE_NAME='readonly'
10
11 # --- Export Vault env variables ---
12 export VAULT_ADDR
13 export VAULT_TOKEN
14
15 # --- Installer PostgreSQL si n cessaire ---
16 if ! command -v psql > /dev/null; then
17     echo "PostgreSQL non trouv . Installation..."
18     sudo apt update
19     sudo apt install postgresql postgresql-contrib -y
20 fi
21
22 # --- Cr er utilisateur PostgreSQL et base ---
23 sudo -u postgres psql <<EOF
24 -- Cr er l'utilisateur Vault
25 CREATE ROLE $DB_USER LOGIN PASSWORD '$DB_PASS' CREATEROLE;
26
27 -- Cr er la base
28 CREATE DATABASE $DB_NAME;
29

```

```

30 -- Donner accès à la base et aux tables
31 GRANT CONNECT ON DATABASE $DB_NAME TO $DB_USER;
32 \c $DB_NAME
33 GRANT USAGE ON SCHEMA public TO $DB_USER;
34 GRANT SELECT ON ALL TABLES IN SCHEMA public TO $DB_USER;
35 ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO
   $DB_USER;
36 EOF
37
38 # --- Activer le Database Secrets Engine ---
39 vault secrets enable database
40
41 # --- Configurer la connexion PostgreSQL pour Vault ---
42 vault write database/config/postgresql \
  plugin_name=postgresql-database-plugin \
  allowed_roles="$ROLE_NAME" \
  connection_url="postgresql://{{username}}:{{password}}@127.0.0.1:5432/
    $DB_NAME" \
  username="$DB_USER" password="$DB_PASS"
43
44 # --- Créer un rôle Vault pour gérer des credentials dynamiques ---
45 vault write database/roles/$ROLE_NAME \
  db_name=postgresql \
  creation_statements="CREATE ROLE \"{{name}}\" WITH LOGIN PASSWORD '{{
    password}}' VALID UNTIL '{{expiration}}'; GRANT SELECT ON ALL
    TABLES IN SCHEMA public TO \"{{name}}\";" \
  default_ttl="1h" max_ttl="24h"
46
47 # --- Générer un credential dynamique pour test ---
48 vault read database/creds/$ROLE_NAME

```

The screenshot shows the HashiCorp Vault UI at the URL [192.168.1.18:8200/ui/vault/secrets/database/overview](http://192.168.1.18:8200/ui/vault/secrets/database/overview). The left sidebar has a dark theme with the following navigation items:

- Vault
- Dashboard
- Secrets Engines** (selected)
- Secrets Recovery (Enterprise)
- Access
- Policies
- Tools
- Monitoring
- Client Count
- Seal Vault

The main content area is titled "Secrets / database" and shows the "database" secrets engine. It has three cards:

- Connections**: Configure new +. Shows 1 connection.
- Roles**: Create new +. Shows 1 role.
- Get Credentials**: Role to use (Search bar) and Get credentials button.

At the bottom, there are links for Vault 1.21.1, Upgrade to Vault Enterprise, Documentation, Support, Terms, Privacy, Security, Accessibility, and © 2025 HashiCorp.

Figure 4.1: Génération de credentials dynamiques pour PostgreSQL

#### 4.3.2 Avantages des credentials dynamiques

##### Bénéfices

- Credentials uniques par application/utilisateur
- Rotation automatique à l'expiration
- Révocation immédiate possible
- Traçabilité complète des accès
- Réduction de la surface d'attaque

# Chapter 5

## Méthodes d'Authentification

### 5.1 Introduction aux Auth Methods

Vault supporte de nombreuses méthodes d'authentification pour s'adapter à différents environnements.

#### 5.1.1 Auth methods courantes

- **Token:** Authentification par token (méthode de base)
- **UserPass:** Nom d'utilisateur et mot de passe
- **LDAP/AD:** Intégration avec Active Directory
- **AppRole:** Pour les applications et automatisation
- **Kubernetes:** Pour les pods Kubernetes
- **AWS:** Authentification via IAM ou EC2
- **GitHub:** Via les teams et organisations GitHub

### 5.2 Token Authentication

Listing 5.1: Gestion des tokens

```
1 # Cr er un token avec une politique sp cifique
2 vault token create -policy=myapp-policy
3
4 # Cr er un token avec TTL limit (Limiter sa dur e de vie)
5 vault token create -ttl=1h -policy=readonly
6
7 # R voquer ou renouveler un token.
8 vault token revoke s.abc123xyz
9
10 # Lister les tokens actifs (root requis)
11 vault list auth/token/accessors
12
13 # Renouveler un token
14 vault token renew
```

## 5.3 UserPass Authentication

Listing 5.2: Configuration UserPass

```

1 # Activer le userpass auth
2 vault auth enable userpass
3
4 # Cr er un utilisateur avec des policies
5 vault write auth/userpass/users/john \
    password=supersecret \
    policies=developer
6
7
8
9 # Se connecter avec l'utilisateur
10 vault login -method=userpass \
    username=john \
    password=supersecret
11
12
13
14 # Changer le mot de passe si n cessaire.
15 vault write auth/userpass/users/john/password \
    password=newsecretpassword
16

```

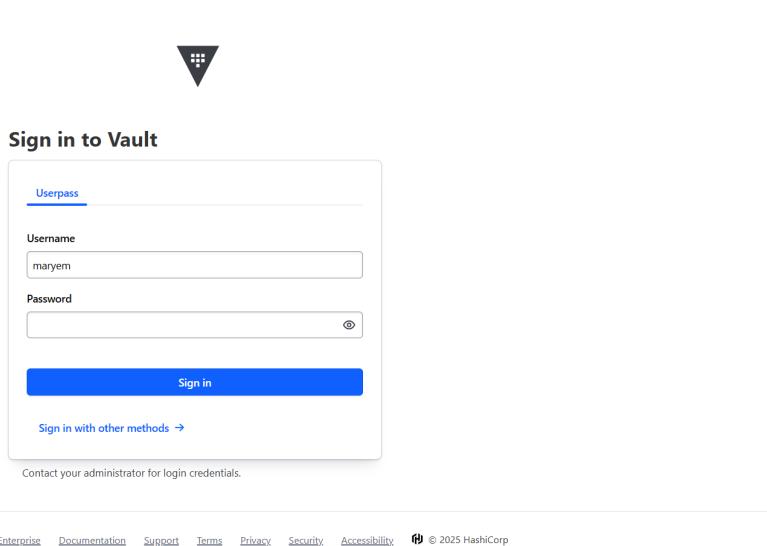


Figure 5.1: Authentification avec UserPass

## 5.4 AppRole Authentication

AppRole est la méthode recommandée pour les applications et l'automatisation.

Listing 5.3: Configuration AppRole Authentication dans Vault

```

1 #!/bin/bash
2 #
3 # Script explicatif pour AppRole
4 #
5
6 # --- Exporter les variables Vault ---
7 export VAULT_ADDR='http://127.0.0.1:8200',
8 export VAULT_TOKEN='myroot' # token root pour config initiale
9

```

```

10 # --- Activer l'authentification AppRole ---
11 vault auth enable approle
12 # Cette commande active le backend AppRole pour que les applications
13 # puissent s'authentifier.
14
15 # --- Cr er un r le AppRole pour l'application ---
16 vault write auth/approle/role/my-application \
17     secret_id_ttl=10m \           # dure de vie du SecretID
18     token_num_uses=10 \          # nombre de fois que le token peut tre
19     # utilis
20     token_ttl=20m \             # TTL du token g n r
21     token_max_ttl=30m \          # TTL max du token
22     secret_id_num_uses=40 \      # nombre de fois que le SecretID peut
23     # tre utilis pour login
24     policies=myapp-policy       # attache la politique de permissions au
25     # token
26
27 # --- R cup rer le RoleID du r le cr ---
28 vault read auth/approle/role/my-application/role-id
29 # Le RoleID est constant et servira avec le SecretID pour l'
30 # authentification.
31
32 # --- G n rer un SecretID temporaire ---
33 vault write -f auth/approle/role/my-application/secret-id
34 # Le SecretID est secret et combin avec le RoleID pour login.
35
36 # --- Se connecter avec RoleID et SecretID ---
37 vault write auth/approle/login \
38     role_id=xxxx-xxxx-xxxx \
39     secret_id=yyyy-yyyy-yyyy
40 # Remplace xxxx et yyyy par les valeurs r cup r es.
41 # La sortie est un token d'acc s que l'application peut utiliser pour
42 # lire des secrets.
43
44 # --- Exemple d'utilisation du token pour lire un secret ---
45 export VAULT_TOKEN=s.abc123xyz
46 vault kv get secret/myapp/config

```

```

root@node1:~# export VAULT_TOKEN="hvs.CAESIEwonb0esI0@wHBoFrhkJSzN-nn194WagnsS0-5ZSVZGh4KHGh2cy5aOTBPbxhsVU84NElvV3FJcFGOW5CeWo"
root@node1:~# vault token lookup
Key          Value
...
accessor      VNycsguMPrviV0LMtwDsEuTyn
creation_time 1765539375
creation_ttl 1h
display_name  approle
entity_id    57678328-ae28-0da2-0cd4-7b2ae776c564
expire_time   2025-12-12T12:36:15.285730794Z
explicit_max_ttl 0s
hmac          hvs.CAESIEwonb0esI0@wHBoFrhkJSzN-nn194WagnsS0-5ZSVZGh4KHGh2cy5aOTBPbxhsVU84NElvV3FJcFGOW5CeWo
issue_time   2025-12-12T11:36:15.287350294Z
meta          map[role_name:myapp-role]
num_uses     0
orphan       true
path         auth/approle/login
policies    [default myapp-policy]
renewable    true
ttl          57m23s
type         service
root@node1:~# vault kv get secret/myapp/config
===== Secret Path =====
secret/data/myapp/config

===== Metadata =====
Key          Value
...
created_time 2025-12-12T11:28:06.824267554Z
custom_metadata <nil>
deletion_time n/a
destroyed     false
version       1

===== Data =====
Key          Value
password    123
username    admin
root@node1:~#

```

Figure 5.2: Workflow d'authentification AppRole

The screenshot shows the HashiCorp Vault UI interface. On the left, there's a sidebar with navigation links like Vault, Dashboard, Secrets Engines (selected), Secrets Recovery, Access, Policies, Tools, Monitoring, Client Count, and Seal Vault. The main content area has a breadcrumb path: Secrets / secret / myapp / config. The title is "myapp/config". Below the title, there are tabs for Overview, Secret (selected), Metadata, Paths, and Version History. A "JSON" button is available. There are buttons for Delete, Destroy, Copy, Version 1, and Create new version. A table lists two secrets: "password" with value "123" and "username" with value "admin". A note indicates "Version 1 created Dec 12, 2025 12:28 PM". At the bottom, there are links for Vault 1.21.1, Upgrade to Vault Enterprise, Documentation, Support, Terms, Privacy, Security, Accessibility, and a copyright notice for © 2025 HashiCorp.

Figure 5.3: Workflow d'authentification AppRole 1

# Chapter 6

## Policies et Contrôle d'Accès

### 6.1 Comprendre les Policies

Les policies Vault utilisent le langage HCL (HashiCorp Configuration Language) pour définir les permissions.

#### 6.1.1 Structure d'une policy

Listing 6.1: Exemple de policy Vault

```
1 # Policy pour une application de lecture seule
2 path "secret/data/myapp/*" {
3     capabilities = ["read", "list"]
4 }
5
6 path "database/creds/readonly" {
7     capabilities = ["read"]
8 }
9
10 # Accès complet à un chemin spécifique
11 path "secret/data/myapp/config" {
12     capabilities = ["create", "read", "update", "delete", "list"]
13 }
14
15 # Dénier explicitement l'accès
16 path "secret/data/admin/*" {
17     capabilities = ["deny"]
18 }
```

#### 6.1.2 Capabilities disponibles

- **create:** Créer de nouvelles données
- **read:** Lire les données existantes
- **update:** Mettre à jour les données
- **delete:** Supprimer les données
- **list:** Lister les clés/chemins

- **deny:** Refuser explicitement l'accès
- **sudo:** Opérations privilégiées

## 6.2 Créer et appliquer des Policies

Listing 6.2: Gestion des policies

```

1 # Cr er une policy      partir d'un fichier
2 vault policy write myapp-policy myapp-policy.hcl
3
4 # Lire une policy
5 vault policy read myapp-policy
6
7 # Lister toutes les policies
8 vault policy list
9
10 # Supprimer une policy
11 vault policy delete myapp-policy
12
13 # Tester une policy (dry-run)
14 vault token create -policy=myapp-policy

```

### 6.2.1 Policy pour développeur

Listing 6.3: developer-policy.hcl

```

1 # Acc s complet aux secrets de d veloppement
2 path "secret/data/dev/*" {
3     capabilities = ["create", "read", "update", "delete", "list"]
4 }
5
6 # Lecture seule pour les secrets de staging
7 path "secret/data/staging/*" {
8     capabilities = ["read", "list"]
9 }
10
11 # Pas d'acc s    la production
12 path "secret/data/prod/*" {
13     capabilities = ["deny"]
14 }
15
16 # G n ration de credentials DB de dev
17 path "database/creds/dev-readonly" {
18     capabilities = ["read"]
19 }

```

Listing 6.4: Appliquer la policy développeur

```

1 # Cr er la policy
2 vault policy write developer developer-policy.hcl
3
4 # Cr er un utilisateur avec cette policy
5 vault write auth/userpass/users/alice \
6     password=dev123 \
7     policies=developer

```

```
8
9 # Tester avec le compte alice
10 vault login -method=userpass username=alice password=dev123
11 vault kv get secret/dev/app-config # OK
12 vault kv get secret/prod/app-config # DENIED
```

```
devops4@node1:~$ vault login -method=userpass username=alice password=dev123
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "Vault login"
again. Future Vault requests will automatically use this token.

Key          Value
--          -----
token        hvs.CAESIJAD_RULDRVc0MT8XJV_u11lutVolYgAq3z0lltkwTv2Ch4KHGh2cy5USnRnbHoxY1Z5TDBmM8pjZkhCcGZGZ08
token_accessor  gseKM3B0a9wXPwpH489x15oe
token_duration    768h
token_renewable   true
token_policies  ["default" "developer"]
identity_policies []
polices        ["default" "developer"]
token_meta.username  alice
===== secret Path =====
secret/data/dev/app-config

===== Metadata =====
Key          Value
--          -----
created_time  2025-12-16T18:38:39.162635342Z
custom_metadata <nil>
deletion_time  n/a
destroyed      false
version        1

===== Data =====
Key          Value
--          -----
password      devpass
username      devuser
devops4@node1:~$ vault kv get secret/prod/app-config
Error reading secret/data/prod/app-config: Error making API request.
URL: GET http://127.0.0.1:8200/v1/secret/data/prod/app-config
Code: 403. Errors:
* 1 error occurred:
  * permission denied
```

Figure 6.1: Test des permissions avec différentes policies

# Chapter 7

## Configuration Production

Cette partie montre comment Vault fonctionne en mode production, contrairement au mode *dev*. L'objectif de cette section est de :

- comprendre comment Vault stocke réellement les secrets,
- voir le processus d'initialisation,
- générer les *unseal keys* et le *root token*,
- apprendre à déverrouiller Vault manuellement.

**En production :**

- Vault ne démarre pas automatiquement en mode non scellé,
- Vault est scellé (*sealed*) au démarrage,
- l'accès dépend du déverrouillage manuel via les *unseal keys*.

### 7.1 Différences Dev vs Production

Aspect	Dev	Production
Stockage	In-memory	File / Consul / Cloud
TLS	Non obligatoire	Obligatoire
Déverrouillage	Auto	Manuel (Shamir / Auto-unseal)
HA (High Availability)	Non	Oui (cluster)
Audit	Non	Obligatoire

Table 7.1: Différences entre mode Dev et Production

### 7.2 Configuration du serveur

Voici la configuration utilisée pour le serveur Vault en production. Nous avons choisi un stockage local (*file*) et désactivé TLS pour nos tests internes.

Listing 7.1: config.hcl - Configuration production

```

1 storage "file" {
2     path = "/home/devops4/vault_data"
3 }
4
5 listener "tcp" {
6     address = "127.0.0.1:8200"
7     tls_disable = 1
8 }
9
10 ui = true
11 log_level = "info"

```

> Remarque : le chemin de stockage a été modifié pour un répertoire dans le home utilisateur afin d'éviter les problèmes de permissions sur '/var/lib/vault'.

## 7.3 Initialisation et déverrouillage (Unsealing)

Listing 7.2: Initialiser et déverrouiller Vault

```

1 # D finir l'adresse de Vault
2 export VAULT_ADDR=http://127.0.0.1:8200
3
4 # D marrer le serveur Vault
5 vault server -config=/home/devops4/config.hcl &
6
7 # Initialiser Vault (générer les 5 unseal keys et le root token)
8 vault operator init -key-shares=5 -key-threshold=3
9
10 # Sauvegarder PROCIEUSEMENT les unseal keys et le root token!
11 # Exemples:
12 # Unseal Key 1: ...
13 # Unseal Key 2: ...
14 # Initial Root Token: ...
15
16 # Déverrouiller Vault avec 3 clés
17 vault operator unseal <key1>
18 vault operator unseal <key2>
19 vault operator unseal <key3>
20
21 # Vérifier que Vault est déverrouillé
22 vault status
23
24 # Se connecter avec le root token
25 vault login <root_token>

```

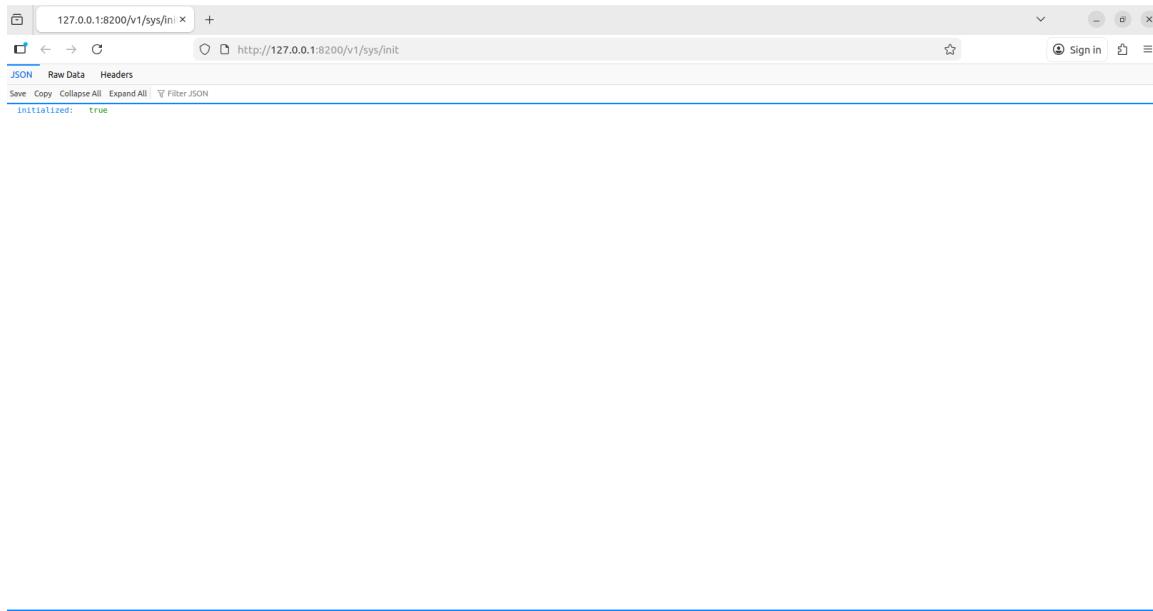


Figure 7.1: Exemple de sortie de `vault operator init` et du processus de déverrouillage

> Après cette étape, Vault est prêt pour créer des secrets, définir des policies et gérer les utilisateurs en production.

### CRITIQUE

Les unseal keys et le root token DOIVENT être:

- Stockés de manière sécurisée (pas dans le code!)
- Distribués à différentes personnes
- Sauvegardés dans plusieurs endroits
- Protégés physiquement et logiquement

La perte de ces clés = perte définitive de l'accès aux données!

## 7.4 Audit Logging

Listing 7.3: Configuration de l'audit

```

1 # Activer l'audit vers un fichier
2 vault audit enable file file_path=/var/log/vault/audit.log
3
4 # Activer l'audit vers syslog
5 vault audit enable syslog
6
7 # Lister les devices d'audit
8 vault audit list
9
10 # Désactiver un device d'audit
11 vault audit disable file/

```

The screenshot shows the 'Unseal Vault' page. At the top, a heading says 'Vault is sealed'. Below it, a text block explains: 'Unseal Vault by entering portions of the unseal key. This can be done via multiple mechanisms on multiple computers. Once all portions are entered, the root key will be decrypted and Vault will unseal.' A form field labeled 'Unseal Key Portion' contains a single line of text. Below the field is a blue 'Unseal' button. At the bottom of the page is a link to 'Seal/unseal documentation'.

Vault   Upgrade to Vault Enterprise   Documentation   Support   Terms   Privacy   Security   Accessibility    © 2025 HashiCorp

Figure 7.2: Accès vault avec clés d'unseal

# Chapter 8

## Projet Pratique: Application Web Sécurisée

### 8.1 Architecture du projet

Cette application Python démontre l'intégration sécurisée de HashiCorp Vault avec une application Flask, afin de :

- Éviter les mots de passe en dur dans le code
- Utiliser AppRole pour l'authentification machine-to-machine
- Utiliser Secrets dynamiques PostgreSQL
- Utiliser Transit Engine pour le chiffrement/déchiffrement
- Centraliser et auditer l'accès aux secrets

**Pourquoi c'est important ?** Dans un environnement DevOps / Cloud, les applications ne doivent jamais stocker des secrets dans le code ou les variables Git.

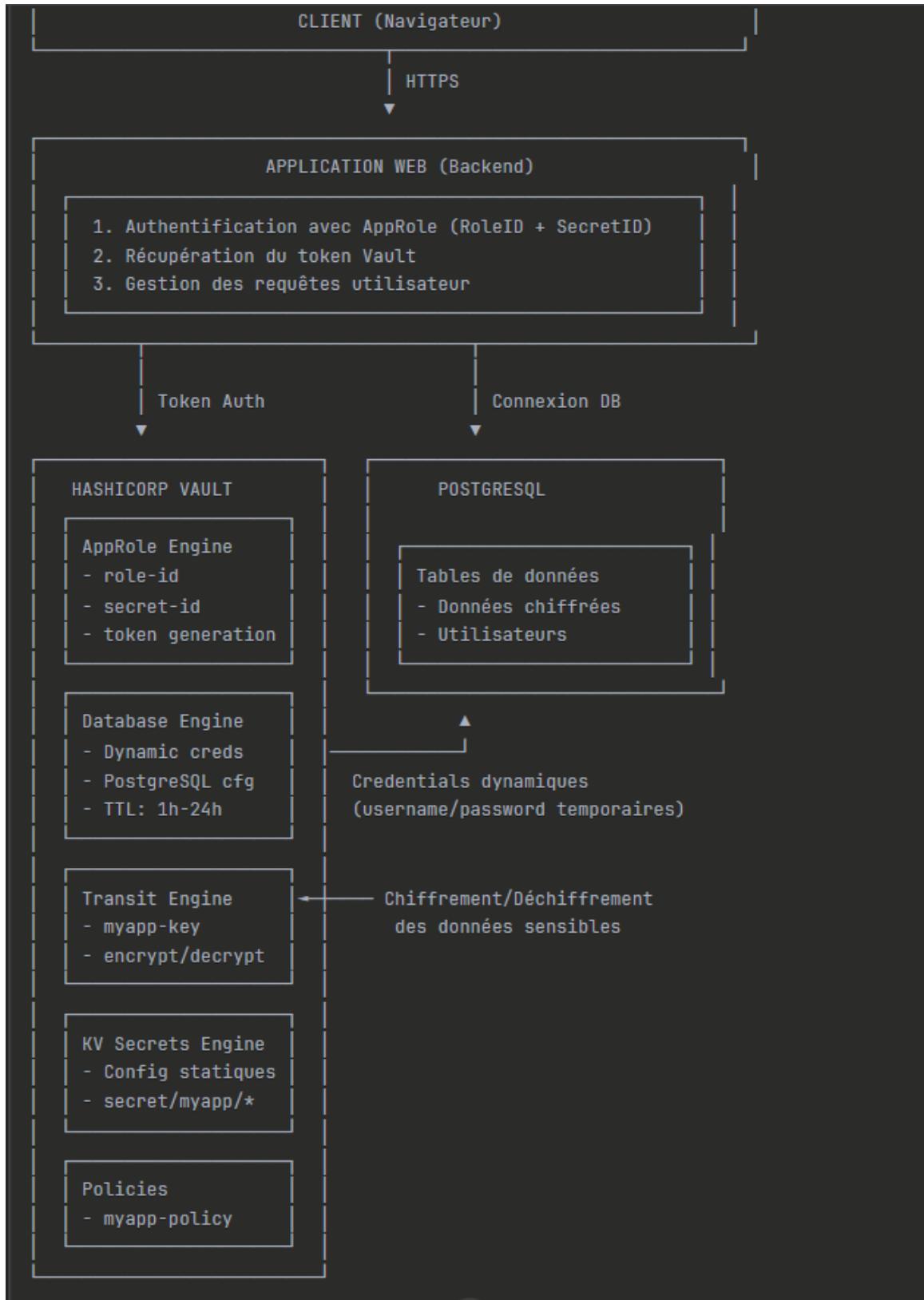


Figure 8.1: Architecture du projet pratique

## 8.2 Préparation de l'environnement

Listing 8.1: Setup Docker Compose

```

1 # docker-compose.yml
2 version: '3.8'
3
4 services:
5   vault:
6     image: hashicorp/vault:latest
7     ports:
8       - "8200:8200"
9     environment:
10      VAULT_DEV_ROOT_TOKEN_ID: myroot
11      VAULT_DEV_LISTEN_ADDRESS: 0.0.0.0:8200
12     cap_add:
13       - IPC_LOCK
14
15   postgres:
16     image: postgres:14
17     environment:
18       POSTGRES_PASSWORD: postgres
19       POSTGRES_DB: myapp
20     ports:
21       - "5432:5432"

```

Listing 8.2: Démarrer l'environnement

```

1 # Lancer les services
2 docker-compose up -d
3
4 # Vérifier que tout fonctionne
5 docker-compose ps
6
7 # Configurer les variables d'environnement
8 export VAULT_ADDR='http://127.0.0.1:8200'
9 export VAULT_TOKEN='myroot'

```

```

devops4@node4: ~ $ docker-compose up -d
docker-compose[1]: Starting 15/15
[+] Building 0.0s (0/0)
[+] Running 3/3
  ✓ Network devops4_default      Created
  ✓ Container devops4-vault-1    Started
  ✓ Container devops4-postgres-1 Created
Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint devops4-postgres-1 (cc0de464709d5165d5d0a18823a70c0c
61d3eb5d75737898fa8576667c846987): failed to bind host port for 0.0.0.0:5432:172.18.0.2:5432/tcp: address already in use
NAME           IMAGE          COMMAND         SERVICE        CREATED        STATUS        PORTS
devops4-vault-1  hashicorp/vault:latest "docker-entrypoint.s..." vault    3 seconds ago  Up 1 second  0.0.0.0:8200->8200/tcp, :::8200->8200/tcp
devops4@node4: ~ $

```

Figure 8.2: Démarrage des services avec Docker Compose

## 8.3 Configuration de Vault

## Listing 8.3: Configuration initiale de Vault

```
1 # 1. Activer le database engine
2 vault secrets enable database
3
4 # 2. Configurer PostgreSQL
5 vault write database
6 /config/postgresql \
7     plugin_name=postgresql-database-plugin \
8     allowed_roles="myapp-role" \
9     connection_url="postgresql://{{username}}:{{password}}@postgres:5432/
10    myapp?sslmode=disable" \
11    username="postgres" \
12    password="postgres"
13
14 # 3. Cr er un r le pour l'application
15 vault write database/roles/myapp-role \
16     db_name=postgresql \
17     creation_statements="CREATE ROLE \"{{name}}\" WITH LOGIN PASSWORD '{{password}}' VALID UNTIL '{{expiration}}'; \
18     GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA public TO \
19     \"{{name}}\";" \
20     default_ttl="1h" \
21     max_ttl="24h"
22
23 # 4. Activer le Transit engine pour le chiffrement
24 vault secrets enable transit
25 vault write -f transit/keys/myapp-key
26
27 # 5. Activer AppRole pour l'authentification
28 vault auth enable approle
29
30 # 6. Cr er une policy pour l'application
31 cat > myapp-policy.hcl <<EOF
32 # Acc s aux credentials de base de donn es
33 path "database/creds/myapp-role" {
34     capabilities = ["read"]
35 }
36
37 # Acc s au chiffrement/d chiffrement
38 path "transit/encrypt/myapp-key" {
39     capabilities = ["update"]
40 }
41
42 path "transit/decrypt/myapp-key" {
43     capabilities = ["update"]
44 }
45
46 # Acc s aux secrets statiques de l'app
47 path "secret/data/myapp/*" {
48     capabilities = ["read", "list"]
49 }
50 EOF
51
52 vault policy write myapp-policy myapp-policy.hcl
53
54 # 7. Cr er le r le AppRole
55 vault write auth/approle/role/myapp \
56     token_policies="myapp-policy" \
```

```

55     token_ttl=1h \
56     token_max_ttl=4h
57
58 # 8. R cup rer RoleID et SecretID
59 vault read auth/approle/role/myapp/role-id
60 vault write -f auth/approle/role/myapp/secret-id
61
62 # 9. Ajouter des secrets statiques
63 vault kv put secret/myapp/config \
64     app_name="My Secure App" \
65     env="development" \
66     log_level="debug"

```

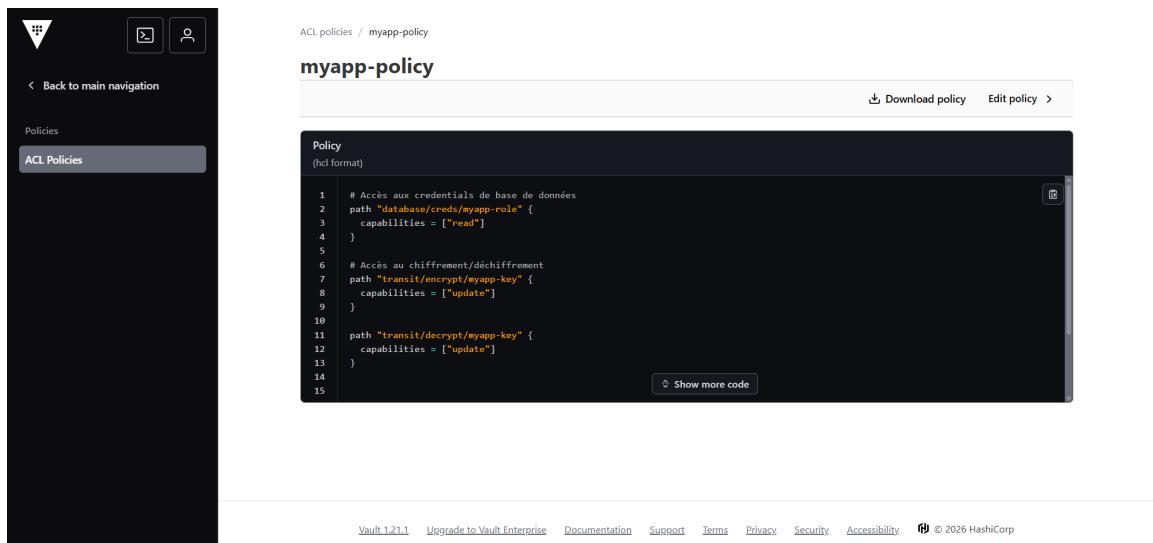


Figure 8.3: Configuration complète de Vault pour le projet

## 8.4 Application Python avec HashiCorp Vault

Cette section décrit une application Python utilisant Vault pour la gestion des secrets, l’authentification AppRole, le chiffrement/déchiffrement de données et la génération de credentials dynamiques pour une base de données PostgreSQL.

### 8.4.1 Structure du projet

Listing 8.4: Structure du projet Python

```

1 myapp/
2     app.py           # Application principale Flask
3     vault_client.py # Client Python pour interagir avec Vault
4     requirements.txt # Dépendances Python
5     .env.example    # Exemple de variables d'environnement

```

### 8.4.2 Client Vault en Python

Le fichier `vault_client.py` contient une classe `VaultClient` qui gère :

- L’authentification avec AppRole

- La lecture des secrets KV
- Le chiffrement et déchiffrement avec le moteur Transit
- La génération de credentials dynamiques pour PostgreSQL
- Le renouvellement du token Vault si nécessaire

Listing 8.5: vault\_client.py : Classe de gestion Vault

```

1 import hvac
2 import base64
3 import os
4 from typing import Dict
5
6 class VaultClient:
7     def __init__(self):
8         # Recupération des variables d'environnement
9         self.vault_addr = os.getenv('VAULT_ADDR', 'http://localhost:8200')
10        self.role_id = os.getenv('VAULT_ROLE_ID')
11        self.secret_id = os.getenv('VAULT_SECRET_ID')
12        self.client = None
13        self._authenticate()
14
15    def _authenticate(self):
16        """Authentification auprès de Vault via AppRole"""
17        self.client = hvac.Client(url=self.vault_addr)
18        response = self.client.auth.approle.login(
19            role_id=self.role_id,
20            secret_id=self.secret_id
21        )
22        print(f"Authentifié avec succès")
23        print(f"Token TTL: {response['auth']['lease_duration']}s")
24
25    # Méthode pour recuperer des credentials PostgreSQL dynamiques
26    def get_db_credentials(self) -> Dict[str, str]:
27        response = self.client.secrets.database.generate_credentials(
28            name='myapp-role'
29        )
30        creds = {
31            'username': response['data']['username'],
32            'password': response['data']['password'],
33            'lease_id': response['lease_id'],
34            'lease_duration': response['lease_duration']
35        }
36        print(f"Credentials DB générés : {creds['username']}")
37        return creds
38
39    # Méthode pour chiffrer des données avec Transit
40    def encrypt_data(self, plaintext: str) -> str:
41        plaintext_b64 = base64.b64encode(plaintext.encode()).decode()
42        response = self.client.secrets.transit.encrypt_data(
43            name='myapp-key',
44            plaintext=plaintext_b64
45        )
46        return response['data']['ciphertext']
47
48    # Méthode pour déchiffrer des données avec Transit

```

```

19     def decrypt_data(self, ciphertext: str) -> str:
20         response = self.client.secrets.transit.decrypt_data(
21             name='myapp-key',
22             ciphertext=ciphertext
23         )
24         return base64.b64decode(response['data']['plaintext']).decode()
25
26     # M thode pour lire des secrets KV
27     def get_config(self, path: str) -> Dict:
28         response = self.client.secrets.kv.v2.read_secret_version(path=path)
29
30         return response['data']['data']
31
32     # Renouvellement du token Vault si besoin
33     def renew_token(self):
34         self.client.auth.token.renew_self()
35         print("    Token renouvel ")

```

```

devops4node1:5 python3 app.py
✓ Authentifié avec succès, Token TTL: 3600s
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.1.20:5000
Press CTRL+C to quit
* Restarting with stat
✓ Authentifié avec succès, Token TTL: 3600s
* Debugger is active!
* Debugger PIN: 179-318-053
192.168.1.19 - - [05/Jan/2026 11:32:57] "GET / HTTP/1.1" 404 -
192.168.1.19 - - [05/Jan/2026 11:32:57] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [05/Jan/2026 11:33:36] "GET / HTTP/1.1" 404 -
192.168.1.19 - - [05/Jan/2026 11:34:55] "GET /health HTTP/1.1" 200 -

```

Figure 8.4: Authentification réussie du client Vault via AppRole

### 8.4.3 Application Flask principale

Le fichier `app.py` met en place :

- Un serveur Flask exposant plusieurs endpoints
- La gestion automatique des credentials PostgreSQL dynamiques
- Le chiffrement/déchiffrement de données via Vault Transit
- La lecture de secrets statiques depuis KV

Listing 8.6: `app.py` : Application principale

```

1 from flask import Flask, jsonify, request
2 import psycopg2
3 from vault_client import VaultClient
4 import time
5
6 app = Flask(__name__)
7 vault = VaultClient()
8
9 db_creds = None
10 db_creds_expiry = 0
11
12 # Fonction pour obtenir une connexion DB avec credentials dynamiques
13 def get_db_connection():

```

```

14     global db_creds, db_creds_expiry
15     if time.time() > db_creds_expiry:
16         db_creds = vault.get_db_credentials()
17         db_creds_expiry = time.time() + db_creds['lease_duration'] - 300
18     conn = psycopg2.connect(
19         host='localhost',
20         database='myapp',
21         user=db_creds['username'],
22         password=db_creds['password']
23     )
24     return conn
25
26 # Endpoint de santé
27 @app.route('/health')
28 def health():
29     config = vault.get_config('myapp/config')
30     return jsonify({
31         'status': 'healthy',
32         'app_name': config['app_name'],
33         'environment': config['env']
34     })
35
36 # Endpoint pour chiffrer des données
37 @app.route('/encrypt', methods=['POST'])
38 def encrypt():
39     data = request.json.get('data')
40     if not data:
41         return jsonify({'error': 'Missing data'}), 400
42     ciphertext = vault.encrypt_data(data)
43     return jsonify({'ciphertext': ciphertext})
44
45 # Endpoint pour déchiffrer des données
46 @app.route('/decrypt', methods=['POST'])
47 def decrypt():
48     ciphertext = request.json.get('ciphertext')
49     if not ciphertext:
50         return jsonify({'error': 'Missing ciphertext'}), 400
51     plaintext = vault.decrypt_data(ciphertext)
52     return jsonify({'plaintext': plaintext})
53
54 # Endpoint pour recuperer les 10 premiers utilisateurs depuis la DB
55 @app.route('/users', methods=['GET'])
56 def get_users():
57     conn = get_db_connection()
58     cursor = conn.cursor()
59     cursor.execute("SELECT id, email FROM users LIMIT 10")
60     users = cursor.fetchall()
61     cursor.close()
62     conn.close()
63     return jsonify({
64         'users': [{id: u[0], email: u[1]} for u in users],
65         'db_user': db_creds['username']
66     })
67
68 if __name__ == '__main__':
69     db_creds = vault.get_db_credentials()
70     db_creds_expiry = time.time() + db_creds['lease_duration'] - 300
71     app.run(debug=True, host='0.0.0.0', port=5000)

```

## 8.4.4 Installation et démarrage

Listing 8.7: Installation des dépendances et lancement de l'application

```

1 # Installer les dépendances Python
2 pip install -r requirements.txt
3
4 # Configurer les variables d'environnement pour AppRole Vault
5 export VAULT_ADDR='http://localhost:8200'
6 export VAULT_ROLE_ID='<votre-role-id>'
7 export VAULT_SECRET_ID='<votre-secret-id>'
8
9 # Démarrer l'application Flask
10 python app.py

```

## 8.4.5 Tests des endpoints

Listing 8.8: Tester les endpoints de l'application

```

1 # Vérifier la santé de l'application
2 curl http://localhost:5000/health
3
4 # Chiffrement de données
5 curl -X POST http://localhost:5000/encrypt \
6   -H "Content-Type: application/json" \
7   -d '{"data": "informations sensibles"}'
8
9 # Déchiffrement de données
10 curl -X POST http://localhost:5000/decrypt \
11   -H "Content-Type: application/json" \
12   -d '{"ciphertext": "vault:v1:..."}'
13
14 # Lecture des utilisateurs depuis PostgreSQL avec credentials dynamiques
15 curl http://localhost:5000/users

```

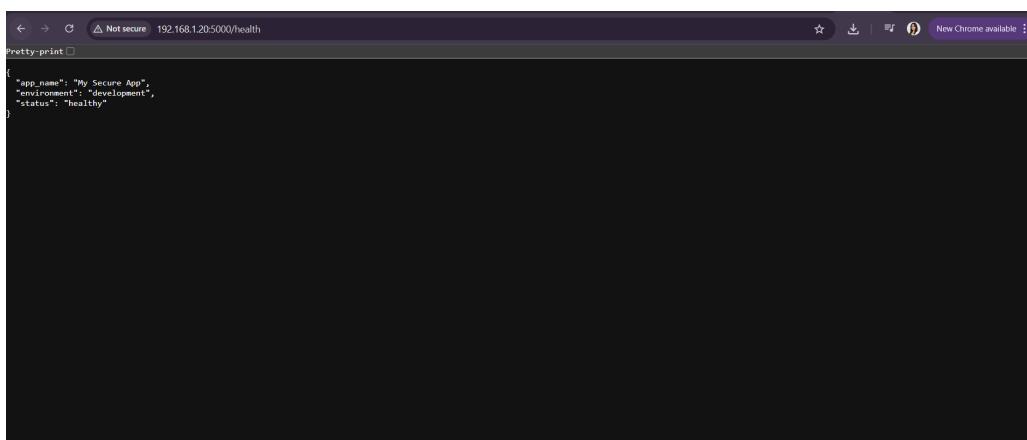
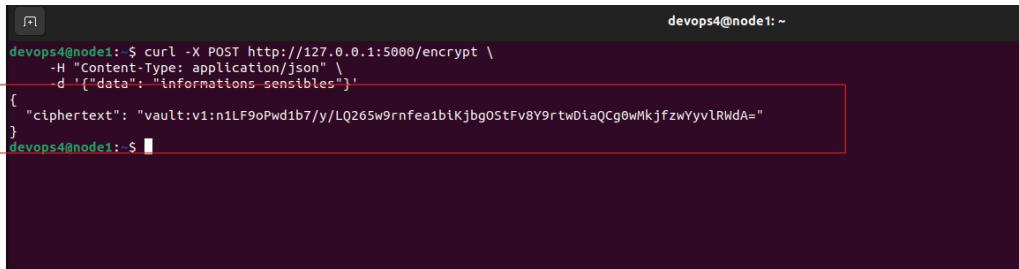


Figure 8.5: Réponse JSON de l'endpoint /health



```
devops4@node1: ~
devops4@node1: $ curl -X POST http://127.0.0.1:5000/encrypt \
-H "Content-Type: application/json" \
-d '{"data": "informations sensibles"}'
{
  "ciphertext": "vault:v1:n1LF9oPwd1b7/y/LQZ65w9rnfea1b1Kjbg0StFv8Y9rtwDiaQCg0wMkjfzwYyvlRwdA="
}
devops4@node1: $
```

Figure 8.6: Chiffrement de données via l'API avec Vault Transit

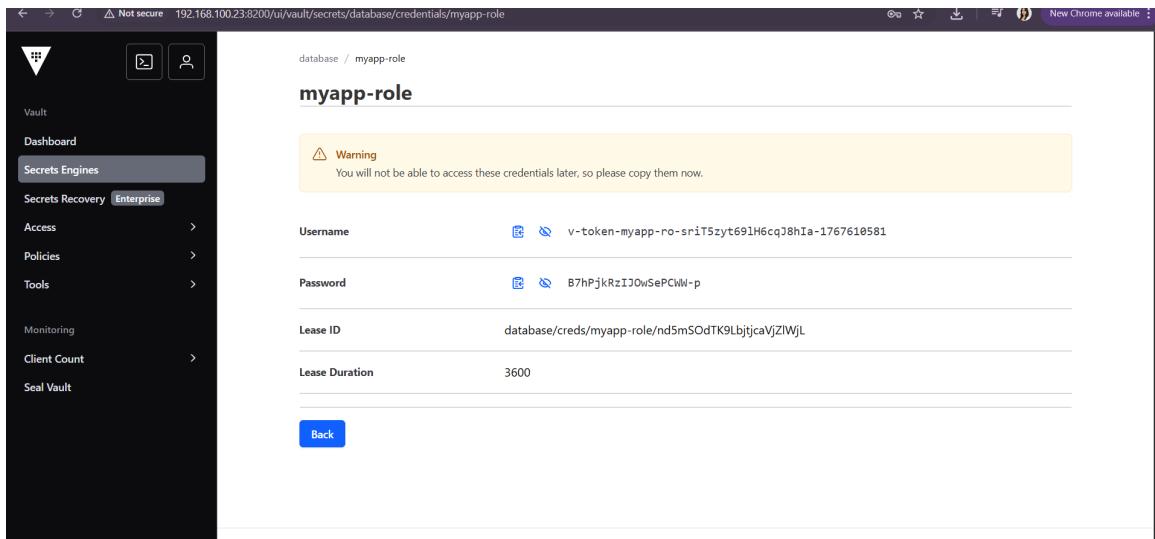


Figure 8.7: Configuration du rôle AppRole ‘myapp’ dans Vault UI

#### 8.4.6 Explications pour débutants

- **VaultClient** : centralise toutes les interactions avec Vault.
- **AppRole** : méthode d'authentification sécurisée entre l'application et Vault.
- **KV v2** : stockage de secrets statiques, par exemple configuration.
- **Transit** : moteur de chiffrement/déchiffrement, utilisé pour protéger les données sensibles.
- **Database dynamic credentials** : Vault crée des utilisateurs temporaires pour PostgreSQL, ce qui améliore la sécurité.
- **Endpoints Flask** : fournissent des interfaces HTTP pour tester la lecture, le chiffrement et l'accès aux secrets.

#### 8.4.7 Lien GitHub

Le code source complet de cette application est disponible sur GitHub. Les étudiants peuvent cloner le projet et l'exécuter localement pour suivre les exercices et tests.

- Dépôt GitHub : <https://github.com/maryeeem/myapp-vault.git>

- Pour cloner le projet :

```
1 git clone https://github.com/maryeeem/myapp-vault.git
2 cd myapp-vault
```

# Chapter 9

## Haute Disponibilité et Clustering

### 9.1 Introduction

HashiCorp Vault peut être déployé en mode **Haute Disponibilité (HA)** pour assurer la continuité de service et la tolérance aux pannes. Dans ce mode, plusieurs nœuds Vault collaborent pour traiter les requêtes tout en garantissant que les secrets restent accessibles même si un nœud tombe en panne.

### 9.2 Architecture HA

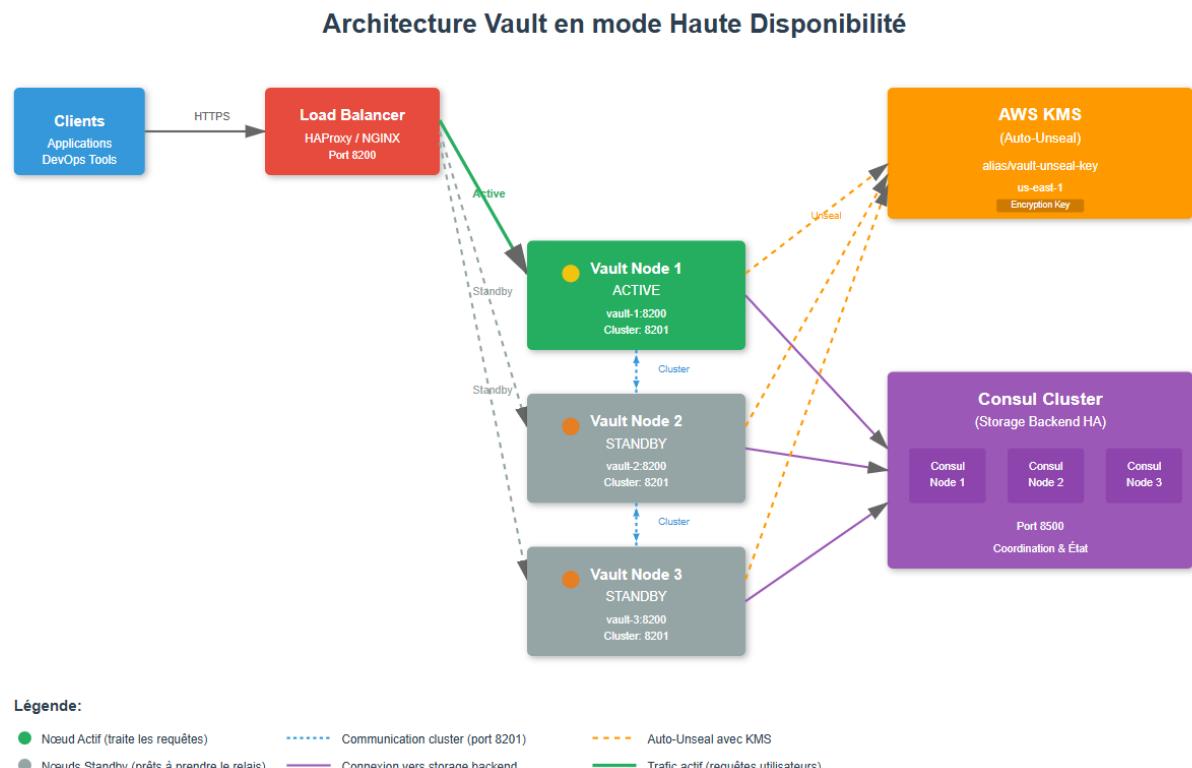


Figure 9.1: Architecture Vault en mode Haute Disponibilité

### 9.2.1 Explication des composants

- **Active node** : Un seul nœud actif qui gère toutes les requêtes d'écriture et de lecture.
- **Standby nodes** : Nœuds en attente, qui peuvent prendre le relais si le nœud actif tombe en panne.
- **Storage backend HA** : Backend de stockage compatible HA, comme Consul ou etcd, pour synchroniser l'état du cluster.
- **Load balancer** : Redirige les requêtes vers le nœud actif et répartit la charge.

**Figure à fournir :** Schéma montrant 1 nœud actif, plusieurs standby, un backend de stockage HA et un load balancer.

---

## 9.3 Configuration d'un cluster Vault avec Consul

### 9.3.1 Étape 1 : Configurer le stockage HA

Le fichier `vault-node1.hcl` définit le stockage backend et les adresses API et Cluster :

Listing 9.1: Exemple de configuration d'un nœud Vault avec Consul

```

1 storage "consul" {
2     address = "consul:8500"
3     path    = "vault/"
4 }
5
6 listener "tcp" {
7     address      = "0.0.0.0:8200"
8     tls_cert_file = "/vault/config/vault.crt"
9     tls_key_file  = "/vault/config/vault.key"
10 }
11
12 api_addr = "https://vault-node1:8200"
13 cluster_addr = "https://vault-node1:8201"
14
15 ui = true
16
17 seal "awskms" {
18     region      = "us-east-1"
19     kms_key_id = "alias/vault-key"
20 }
```

*Explication :*

- `storage "consul"` : définit Consul comme backend HA pour synchroniser l'état du cluster.
- `listener "tcp"` : expose Vault via TLS.
- `seal "awskms"` : configure Auto-Unseal (ici AWS KMS).

### 9.3.2 Étape 2 : Déploiement avec Docker Compose

Le fichier `docker-compose.yml` permet de lancer un cluster de 3 nœuds Vault avec Consul :

Listing 9.2: Docker Compose pour cluster Vault HA

```

1 version: '3.8'
2
3 services:
4   consul:
5     image: consul:latest
6     command: agent -server -bootstrap -expect=1 -ui -client=0.0.0.0
7     ports:
8       - "8500:8500"
9
10  vault-1:
11    image: hashicorp/vault:latest
12    ports:
13      - "8200:8200"
14    volumes:
15      - ./config/vault-node1.hcl:/vault/config/vault.hcl
16    environment:
17      VAULT_API_ADDR: https://vault-1:8200
18    depends_on:
19      - consul
20    cap_add:
21      - IPC_LOCK
22
23  vault-2:
24    image: hashicorp/vault:latest
25    ports:
26      - "8201:8200"
27    volumes:
28      - ./config/vault-node2.hcl:/vault/config/vault.hcl
29    environment:
30      VAULT_API_ADDR: https://vault-2:8200
31    depends_on:
32      - consul
33    cap_add:
34      - IPC_LOCK
35
36  vault-3:
37    image: hashicorp/vault:latest
38    ports:
39      - "8202:8200"
40    volumes:
41      - ./config/vault-node3.hcl:/vault/config/vault.hcl
42    environment:
43      VAULT_API_ADDR: https://vault-3:8200
44    depends_on:
45      - consul
46    cap_add:
47      - IPC_LOCK

```

*Explication :*

- Chaque service Vault est un nœud du cluster.
- `depends_on` assure que Consul démarre avant Vault.

- IPC\_LOCK empêche Vault de swapper la mémoire sensible.

### 9.3.3 Étape 3 : Vérifier le cluster

- Vérifier le statut de chaque noeud avec la commande :

```
1 vault status
```

- Faire une capture d'écran pour montrer les noeuds Active et Standby.

```
devops@node1:~/vault_ha_project$ docker-compose up -d
[+] Building 0.0s (0/0)
default
[+] Running 3/3
  ✓ Network vault_ha_project_default      Created
    0.2s
  ✓ Container vault_ha_project-consul-1   Started
    0.1s
  ✓ Container vault_ha_project-vault-1-1  Started
    0.1s
  ✓ Container vault_ha_project-vault-2-1  Started
    0.1s
  ✓ Container vault_ha_project-vault-3-1  Started
    0.1s
devops@node1:~/vault_ha_project$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
6691azfd097f        hashicorp/vault:latest "docker-entrypoint.s..." 16 seconds ago   Up 15 seconds       0.0.0.0:8200->8200/tcp, [::]:8200->8200/tcp
313f6abbab59        hashicorp/vault:latest "docker-entrypoint.s..." 16 seconds ago   Up 15 seconds       0.0.0.0:8202->8200/tcp, [::]:8202->8200/tcp
af966020f496        hashicorp/vault:latest "docker-entrypoint.s..." 16 seconds ago   Up 15 seconds       0.0.0.0:8201->8200/tcp, [::]:8201->8200/tcp
35c141d151eb        hashicorp/consul:1.22.2  "docker-entrypoint.s..." 16 seconds ago   Up 16 seconds       8300-8302/tcp, 8301-8302/udp, 8600/tcp, 8600/udp, 0.0.0.0:8500->8500/tcp, [::]:8500/udp
devops@node1:~/vault_ha_project$
```

Figure 9.2: Statut du cluster Vault avec 3 noeuds

## 9.4 Configuration Auto-Unseal avec AWS KMS

Vault peut se déverrouiller automatiquement grâce à AWS KMS, ce qui évite de saisir manuellement les clés de déverrouillage.

### 9.4.1 Étape 1 : Créer la clé KMS

```
1 aws kms create-key --description "Vault Unseal Key"
2 aws kms create-alias --alias-name alias/vault-unseal-key --target-key-id <
key-id>
```

### 9.4.2 Étape 2 : Configurer Vault

Ajouter dans le fichier `vault.hcl` :

```
1 seal "awskms" {
2     region      = "us-east-1"
3     kms_key_id = "alias/vault-unseal-key"
4     endpoint    = "https://kms.us-east-1.amazonaws.com"
5 }
```

### 9.4.3 Étape 3 : Initialiser Vault

```
1 vault operator init -recovery-shares=5 -recovery-threshold=3
```

- Capturer la sortie de l'initialisation montrant les clés de récupération et que Vault est Auto-Unseal.

*Explication :*

- Vault peut se déverrouiller seul grâce à KMS.
- Les `recovery-shares` et `threshold` permettent de récupérer Vault si la KMS n'est pas accessible.

# Chapter 10

## Intégration de Vault avec Kubernetes

Ce chapitre présente l'intégration de HashiCorp Vault avec Kubernetes afin de sécuriser la gestion des secrets dans des applications conteneurisées. L'objectif est de permettre aux pods Kubernetes d'accéder aux secrets de manière dynamique, sécurisée et sans stocker d'informations sensibles dans le code ou les fichiers de configuration.

### 10.1 Déploiement de Vault sur un cluster Kubernetes mono-nœud

Dans un environnement de test ou de lab avec un seul nœud Kubernetes, il est recommandé de déployer **un seul pod Vault** pour éviter les problèmes de volumes hostPath et de planification.

#### 10.1.1 Installation avec Helm

Les étapes suivantes permettent d'installer Vault avec un pod unique et l'interface graphique activée.

Listing 10.1: Installation de Vault sur Kubernetes mono-nœud avec Helm

```
1 # Ajouter le dépôt Helm officiel de HashiCorp
2 helm repo add hashicorp https://helm.releases.hashicorp.com
3 helm repo update
4
5 # Créer le namespace Vault
6 kubectl create namespace vault
7
8 # Installer Vault en mono-pod (HA désactivé)
9 helm install vault hashicorp/vault \
10   --namespace vault \
11   --set server.ha.enabled=false \
12   --set ui.enabled=true \
13   --set ui.serviceType=LoadBalancer \
14   --set server.dataStorage.enabled=true \
15   --set server.dataStorage.storageClass=standard \
16   --set server.dataStorage.size=1Gi
17
18 # Vérifier l'état du pod
19 kubectl get pods -n vault
```

```

devops4@node1: ~ helm repo add hashicorp https://helm.releases.hashicorp.com
devops4@node1: ~ helm repo update
devops4@node1: ~ kubectl create namespace vault
devops4@node1: ~ helm install vault hashicorp/vault \
--namespace vault \
--set service.enabled=true \
--set replicas=3 \
--set util.enabled=true
"hashicorp" already exists with the same configuration, skipping
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "hashicorp" chart repository
...Reloading the chart cache from the "secrets-store-csi-driver" chart repository
Update Complete. #Happy Helming!
Error from server (AlreadyExists): namespaces "vault" already exists
Error: INSTALLATION FAILED: cannot re-use a name that is still in use
devops4@node1: ~ kubectl get pods -n vault
NAME          READY   STATUS    RESTARTS   AGE
vault-0        1/1     Running   23 (11m ago)  209d
vault-agent-injector-56459c7545-647w9  1/1     Running   168 (11m ago)  209d
devops4@node1: ~

```

Figure 10.1: Pod Vault déployé sur un cluster Kubernetes mono-nœud

**Remarque :** Pour un cluster mono-nœud, l’option ‘hostPath’ ou une PV locale fonctionne correctement car le pod est toujours planifié sur le même nœud.

## 10.2 Création des Persistent Volumes (PV) et Persistent Volume Claims (PVC)

Pour permettre à Vault de stocker les secrets de manière persistante :

### 10.2.1 Création des volumes sur le nœud

Listing 10.2: Préparer les dossiers sur le nœud

```

1 sudo mkdir -p /mnt/data/vault
2 sudo chown -R 1000:1000 /mnt/data/vault
3 sudo chmod -R 700 /mnt/data/vault

```

### 10.2.2 Définition du PV et PVC

Listing 10.3: Persistent Volume et Persistent Volume Claim pour Vault

```

1 apiVersion: v1
2 kind: PersistentVolume
3 metadata:
4   name: vault-pv
5 spec:
6   capacity:
7     storage: 1Gi
8   accessModes:
9     - ReadWriteOnce
10  storageClassName: standard
11  persistentVolumeReclaimPolicy: Delete
12  hostPath:
13    path: "/mnt/data/vault"
14  ---
15 apiVersion: v1
16 kind: PersistentVolumeClaim
17 metadata:
18   name: vault-data

```

```

19   namespace: vault
20 spec:
21   accessModes:
22     - ReadWriteOnce
23   resources:
24     requests:
25       storage: 1Gi
26   storageClassName: standard
27   volumeName: vault-pv

```

Appliquer ces fichiers :

Listing 10.4: Appliquer PV et PVC

```

1 kubectl apply -f vault-pv.yaml
2 kubectl apply -f vault-pvc.yaml

```

Vérifier que le PVC est bien **Bound** :

```

1 kubectl get pvc -n vault

```

## 10.3 Initialisation et Unseal du Pod Vault

Une fois le pod Vault lancé, il doit être initialisé et déverrouillé pour fonctionner.

### 10.3.1 Initialisation

Listing 10.5: Initialisation de Vault

```

1 kubectl exec -n vault vault-0 -- vault operator init \
2   -key-shares=5 \
3   -key-threshold=3 \
4   -format=json > cluster-keys.json

```

### 10.3.2 Déverrouillage (Unseal)

Listing 10.6: Déverrouillage de Vault

```

1 kubectl exec -n vault vault-0 -- vault operator unseal <UNSEAL_KEY_1>
2 kubectl exec -n vault vault-0 -- vault operator unseal <UNSEAL_KEY_2>
3 kubectl exec -n vault vault-0 -- vault operator unseal <UNSEAL_KEY_3>

```

Vérifier le statut :

```

1 kubectl exec -n vault vault-0 -- vault status

```

## 10.4 Authentification Kubernetes

Vault peut vérifier l'identité des pods via leur ServiceAccount, ce qui permet un accès sécurisé aux secrets.

## 10.4.1 Activation et configuration

Listing 10.7: Activation de l'auth Kubernetes

```

1 kubectl exec -n vault vault-0 -- vault auth enable kubernetes
2
3 kubectl exec -n vault vault-0 -- vault write auth/kubernetes/config \
4   kubernetes_host="https://kubernetes.default.svc:443" \
5   kubernetes_ca_cert=@/var/run/secrets/kubernetes.io/serviceaccount/ca.crt \
6   token_reviewer_jwt=@/var/run/secrets/kubernetes.io/serviceaccount/token

```

## 10.4.2 Création d'une policy et d'un rôle

Listing 10.8: Policy Vault pour l'application

```

1 path "secret/data/k8s-app/*" {
2   capabilities = ["read"]
3 }

```

Listing 10.9: Association du rôle à un ServiceAccount

```

1 kubectl exec -n vault vault-0 -- vault policy write k8s-app k8s-app-policy \
2   .hcl
3
4 kubectl exec -n vault vault-0 -- vault write auth/kubernetes/role/myapp \
5   bound_service_account_names=myapp \
6   bound_service_account_namespaces=default \
7   policies=k8s-app \
8   ttl=24h

```

## 10.5 Injection des secrets avec Vault Agent Injector

### 10.5.1 Exemple de pod utilisant Vault Agent Injector

Listing 10.10: Pod Kubernetes utilisant Vault Agent Injector

```

1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: myapp
5   annotations:
6     vault.hashicorp.com/agent-inject: "true"
7     vault.hashicorp.com/role: "myapp"
8     vault.hashicorp.com/agent-inject-secret-database: "secret/data/k8s-app
9       /database"
10    vault.hashicorp.com/agent-inject-template-database: |
11      {{- with secret "secret/data/k8s-app/database" -}}
12        export DB_USERNAME="{{ .Data.data.username }}"
13        export DB_PASSWORD="{{ .Data.data.password }}"
14      {{- end -}}
15
16 spec:
17   serviceAccountName: myapp
18   containers:
19     - name: app
20       image: myapp:latest

```

```

19   command: ["/bin/sh"]
20   args: ["-c", "source /vault/secrets/database && ./app"]

```

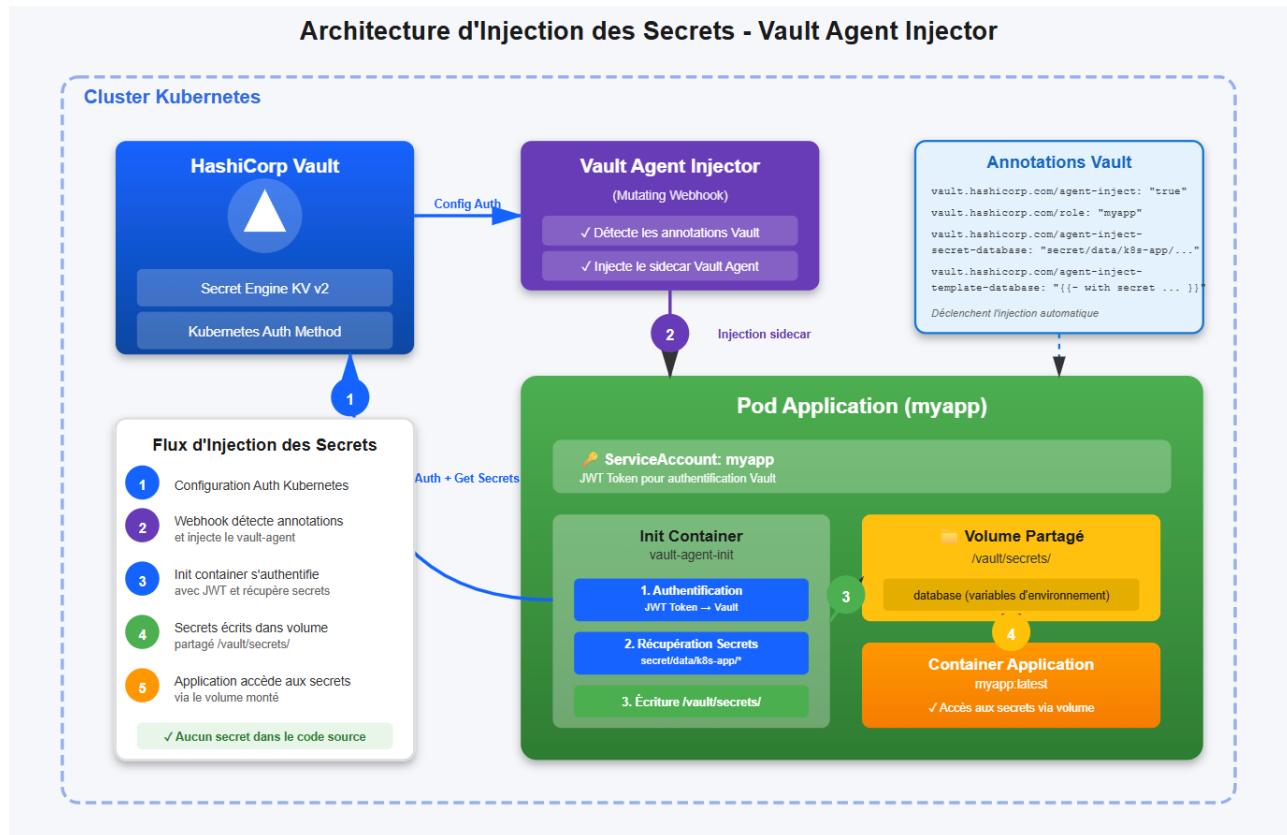


Figure 10.2: Injection automatique des secrets via Vault Agent Injector

# Chapter 11

## Monitoring et Troubleshooting de Vault

Vault est un composant critique pour la gestion des secrets et des tokens. Un bon monitoring permet de détecter rapidement les problèmes et de garantir la disponibilité et la sécurité. Dans ce chapitre, nous détaillons comment collecter les métriques avec Prometheus, les visualiser avec Grafana, et résoudre les problèmes courants.

### 11.1 Collecte des métriques Vault

Vault expose des métriques détaillées sur son fonctionnement interne. Ces métriques sont essentielles pour surveiller :

- L'état de déverrouillage (sealed/unsealed)
- Les nœuds actifs et standby dans un cluster HA
- L'utilisation mémoire et CPU
- L'activité des tokens et des leases
- Les performances générales

#### 11.1.1 Configuration de la telemetry pour Prometheus

Pour que Vault expose ses métriques au format Prometheus, il faut configurer le bloc ‘telemetry’ dans le fichier de configuration de Vault (`vault.hcl`) :

Listing 11.1: Configuration telemetry dans vault.hcl

```
1 telemetry {
2     prometheus_retention_time = "30s"    # Intervalle de rafraîchissement des
3                                         # métriques
4     disable_hostname = true                # Supprime l'ajout automatique du
5                                         # hostname
6 }
```

> Cette configuration permet à Prometheus de scraper régulièrement Vault et d'obtenir des métriques à jour.

### 11.1.2 Scraper les métriques depuis Vault

Après avoir démarré Vault et défini le token root, on peut récupérer les métriques Prometheus via l'API :

Listing 11.2: Récupération des métriques Prometheus

```

1 export VAULT_ADDR=http://192.168.100.23:8200
2 export VAULT_TOKEN=hvs.0wl4mAk7WK20nkeCQGFhiq0J
3
4 curl -H "X-Vault-Token: $VAULT_TOKEN" \
      http://192.168.100.23:8200/v1/sys/metrics?format=prometheus

```

Métriques importantes à surveiller :

- `vault core unsealed` : indique si Vault est sealed ou déverrouillé
- `vault core active` : indique si le nœud est actif ou standby
- `vault runtime alloc bytes` : mémoire utilisée par Vault
- `vault token creation` : nombre de tokens créés
- `vault expire num leases` : nombre de leases actifs

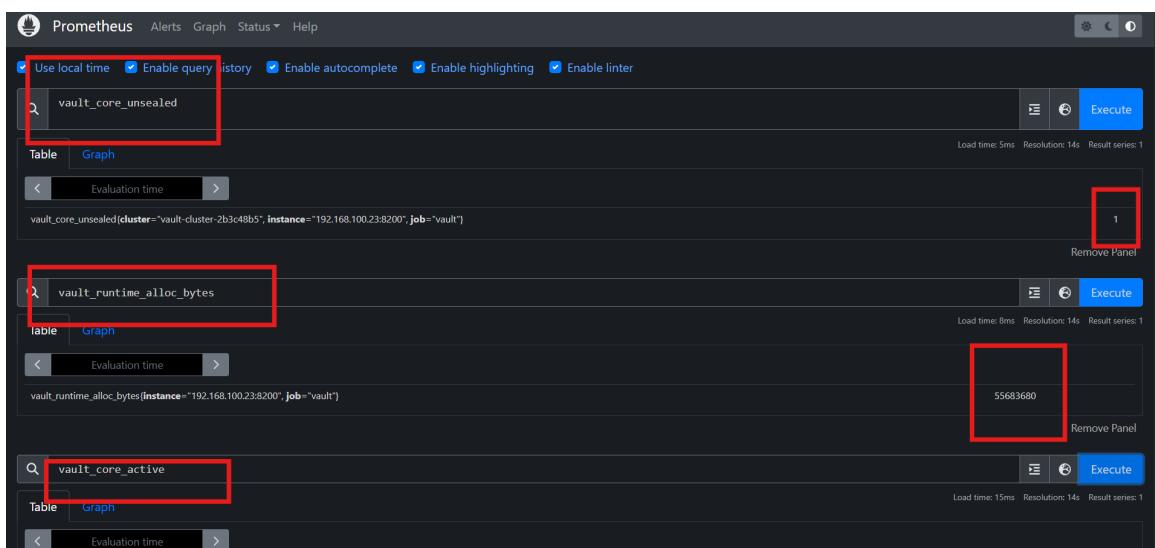


Figure 11.1: Métriques Vault accessibles via Prometheus

### 11.1.3 Visualisation avec Grafana

Une fois les métriques disponibles via Prometheus, Grafana permet de créer des dashboards graphiques :

- Installer Grafana et connecter Prometheus comme datasource
- Créer un dashboard avec des panels pour chaque métrique clé
- Visualiser par exemple : utilisation mémoire, nombre de tokens, état des noeuds, performance des leases

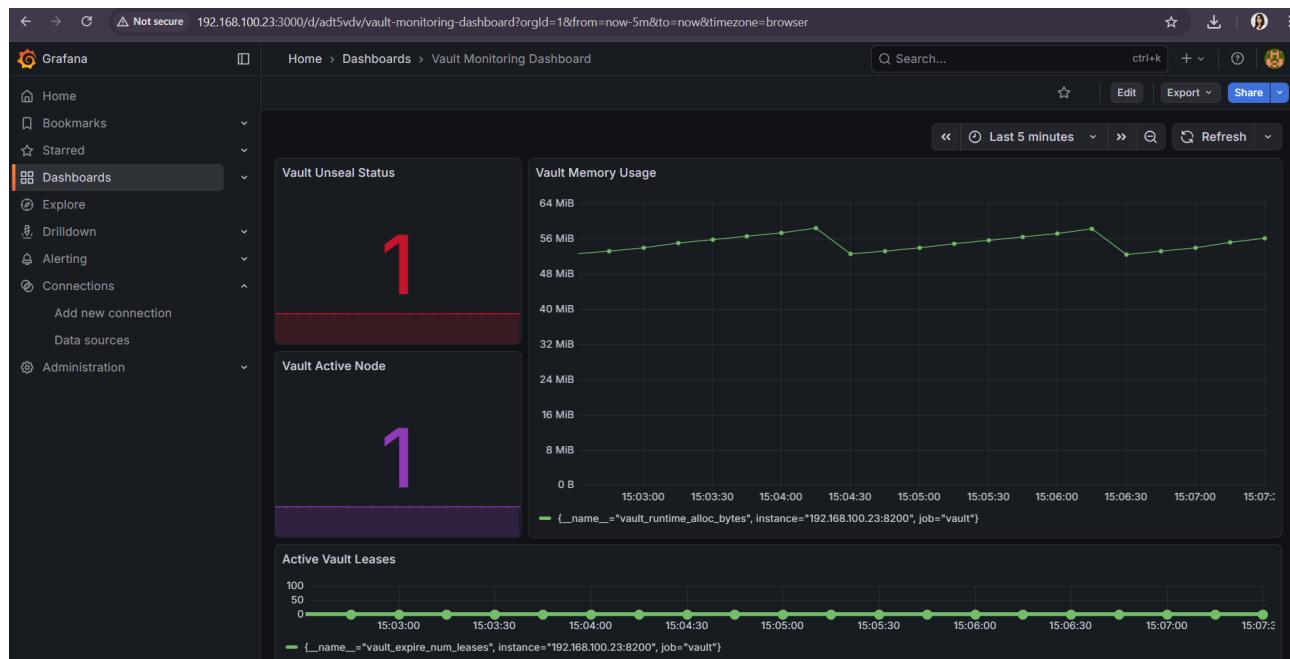


Figure 11.2: Exemple de dashboard Grafana pour Vault

## 11.2 Troubleshooting courant

### 11.2.1 Vault sealed après redémarrage

Si Vault apparaît comme ‘sealed’ après un redémarrage, il faut vérifier l’état et déverrouiller manuellement si nécessaire :

Listing 11.3: Diagnostiquer un problème de seal

```

1 # V rifier l' tat de Vault
2 vault status
3
4 # Consulter les logs r cents
5 journalctl -u vault -n 100
6
7 # D verrouiller Vault
8 vault operator unseal
9
10 # V rifier l'auto-unseal si configur
11 vault read sys/seal-status

```

### 11.2.2 Performance dégradée

Pour détecter et diagnostiquer les problèmes de performance :

Listing 11.4: Analyse des performances Vault

```

1 # Consulter les leases actifs
2 vault list sys/leases/lookup/
3
4 # V rifier les tokens actifs
5 vault token lookup -accessor <accessor-id>

```

```
6
7 # Analyser les logs d'audit
8 tail -f /var/log/vault/audit.log | jq
9
10 # Profiling pour analyser le CPU/m moire
11 vault debug -duration=30s -output=vault-debug.tar.gz
```

### 11.2.3 Problèmes de connexion

Pour diagnostiquer les problèmes réseau ou TLS :

Listing 11.5: Diagnostics de connexion

```
1 # V rifier la sant de Vault
2 curl -k https://vault.example.com:8200/v1/sys/health
3
4 # Tester TLS
5 openssl s_client -connect vault.example.com:8200
6
7 # V rifier les certificats PKI
8 vault read sys-mounts/pki/cert/ca
9
10 # Activer les logs d taill s
11 export VAULT_LOG_LEVEL=trace
12 vault status
```

# Chapter 12

## Sécurité Avancée

Ce chapitre présente les mécanismes de sécurité avancée offerts par HashiCorp Vault. Ces fonctionnalités permettent de renforcer la protection des secrets sensibles, de réduire les risques liés à la compromission des accès et d'assurer une gestion sécurisée du cycle de vie des secrets.

### 12.1 Rotation des secrets

La rotation des secrets est un principe fondamental de la sécurité moderne. Elle consiste à renouveler périodiquement les clés, mots de passe ou identifiants afin de limiter leur durée d'exposition en cas de fuite ou de compromission.

Vault automatise cette rotation pour plusieurs types de secrets, réduisant ainsi les erreurs humaines et améliorant la conformité aux bonnes pratiques de sécurité.

#### 12.1.1 Rotation automatique des clés

Vault permet de configurer une rotation automatique pour les clés cryptographiques et les identifiants dynamiques, notamment via les moteurs *Transit* et *Database*.

Listing 12.1: Configuration de la rotation automatique des secrets

```
1 # Activer la rotation automatique pour une clé Transit
2 vault write transit/keys/myapp-key/config \
3   auto_rotate_period=24h
4
5 # Définir la durée de vie maximale des connexions à la base de données
6 vault write database/config/postgresql \
7   max_connection_lifetime=1h
8
9 # Forcer une rotation manuelle immédiate
10 vault write -f transit/keys/myapp-key/rotate
```

La rotation automatique garantit que :

- les clés cryptographiques sont régulièrement renouvelées ;
- les identifiants de bases de données expirent automatiquement ;
- les applications utilisent toujours des secrets valides et à jour.

### 12.1.2 Révocation de secrets

La révocation permet d'invalider immédiatement un secret ou un accès compromis. Vault offre plusieurs niveaux de révocation : par lease, par chemin ou par utilisateur.

Listing 12.2: Révocation des secrets dans Vault

```

1 # R voquer un lease sp cifique
2 vault lease revoke database/creds/myapp/abc123
3
4 # R voquer tous les leases associ s un chemin
5 vault lease revoke -prefix database/creds/myapp/
6
7 # R voquer tous les tokens associ s un utilisateur
8 vault token revoke -mode=path auth/userpass/login/alice

```

La révocation est particulièrement utile dans les cas suivants :

- compromission d'un identifiant ;
- départ d'un employé ;
- incident de sécurité nécessitant une coupure immédiate des accès.

Figure 12.1: Processus de révocation des leases dans Vault

## 12.2 Response Wrapping

Le *Response Wrapping* est un mécanisme de sécurité avancé permettant de transmettre des secrets de manière temporaire et sécurisée. Vault encapsule le secret dans un token à usage unique, avec une durée de vie limitée.

### 12.2.1 Principe de fonctionnement

- Le secret réel n'est jamais transmis directement.
- Un token de wrapping est généré avec un TTL défini.

- Le token ne peut être utilisé qu'une seule fois.

Listing 12.3: Utilisation du Response Wrapping

```

1 # Générer un secret wrapped avec une durée de vie de 60 secondes
2 vault kv get -wrap-ttl=60s secret/myapp/config
3
4 # Le destinataire récupère le secret (une seule fois)
5 vault unwrap <wrapping-token>
6
7 # Générer un Secret ID AppRole wrapped
8 vault write -wrap-ttl=120s -f \
9     auth/approle/role/myapp/secret-id

```

Ce mécanisme est particulièrement adapté aux scénarios suivants :

- transmission sécurisée de secrets entre équipes ;
- intégration CI/CD ;
- déploiement automatisé d'applications.

```

devops4@node1: $ vault kv get -wrap-ttl=60s secret/myapp/config
...
wrapping_token: hvs.CAESI...                         DJ6T1zhyWlkWXA
wrapping_accessor: EVYTGp9pBoedDZlJJp6Ne6Qz
wrapping_token_ttl: 1m
wrapping_token_creation_time: 2026-01-06 16:09:15.180966811 +0100 CET
wrapping_token_creation_path: secret/myapp/config
devops4@node1: $ vault kv get -wrap-ttl=60s secret/myapp/config
Key                                         Value
wrapping_token: hvs.CAESI...                         TLJN0d0QXlKdTdqNFpkYU05QVBqTk
wrapping_accessor: QD3dz246ADfgPYXnayQakvlIV
wrapping_token_ttl: 1m
wrapping_token_creation_time: 2026-01-06 16:10:10.704552784 +0100 CET
wrapping_token_creation_path: secret/myapp/config
devops4@node1: $

```

Figure 12.2: Mécanisme de Response Wrapping dans Vault

## 12.3 Sentinel Policies (Vault Enterprise)

Sentinel est un langage de politiques utilisé dans la version Enterprise de Vault. Il permet d'appliquer des règles de gouvernance et de conformité avant l'exécution des opérations sensibles.

### 12.3.1 Contrôle d'accès avancé

Les politiques Sentinel permettent par exemple :

- d'imposer l'authentification multifacteur (MFA) ;
- de restreindre les opérations à des plages horaires spécifiques ;
- de bloquer certaines actions en dehors du cadre défini.

Listing 12.4: Exemple de politique Sentinel

```

1 import "time"
2
3 # Autoriser les opérations sensibles uniquement
4 # pendant les heures ouvrables

```

```
5 main = rule {
6     time.now.weekday in ["Monday", "Tuesday", "Wednesday", "Thursday", "
7         Friday"] and
8     time.now.hour >= 9 and
9     time.now.hour <= 18 and
10    request.operation in ["create", "update", "delete"]
11 }
```

Grâce à Sentinel, Vault devient un outil central de gouvernance de la sécurité, garantissant que les accès respectent les politiques internes et les exigences de conformité.

# Chapter 13

## Bonnes Pratiques

Dans ce chapitre, nous présentons les recommandations essentielles pour sécuriser et administrer HashiCorp Vault dans un environnement de production. L'objectif est de garantir la confidentialité, l'intégrité et la disponibilité des secrets, tout en facilitant la maintenance et la conformité.

### 13.1 Gestion des secrets

La gestion des secrets est au cœur de Vault. Une configuration correcte et des pratiques rigoureuses permettent de limiter les risques de fuite ou de compromission des informations sensibles.

#### À FAIRE

- **Utiliser des credentials dynamiques** : Générer des secrets temporaires pour limiter l'exposition.
- **Définir des TTL courts** : Réduire la durée de vie des secrets selon le principe du moindre privilège.
- **Activer l'audit logging dès le début** : Suivre toutes les actions pour détecter les anomalies et garantir la traçabilité.
- **Utiliser des policies granulaires** : Accorder uniquement les permissions nécessaires pour chaque utilisateur ou application.
- **Implémenter la rotation automatique des secrets** : Éviter la réutilisation prolongée des credentials sensibles.
- **Monitorer les métriques critiques** : Surveiller les performances et les alertes pour anticiper les incidents.
- **Tester régulièrement les backups** : S'assurer que les sauvegardes sont fiables et restaurables.
- **Utiliser TLS partout** : Chiffrer toutes les communications pour prévenir les interceptions.

## À ÉVITER

- **Utiliser le mode dev en production :** Ce mode n'est pas sécurisé et ne doit jamais être exposé.
- **Stocker le root token :** Le root token doit être révoqué après initialisation pour éviter les compromissions.
- **Partager les tokens entre applications :** Chaque application doit avoir son propre token pour limiter l'impact d'un incident.
- **Désactiver l'audit logging :** Empêche de tracer les actions et complique la détection des intrusions.
- **Utiliser des TTL trop longs :** Augmente la fenêtre d'exposition en cas de fuite.
- **Négliger la rotation des clés :** Les secrets statiques sont plus vulnérables aux attaques.
- **Ignorer les logs d'audit :** Les logs sont essentiels pour les analyses post-incident.
- **Communication HTTP non chiffrée :** Expose les secrets aux interceptions réseau.

## 13.2 Architecture recommandée

Pour un déploiement sécurisé et résilient, il est conseillé de suivre l'architecture de production illustrée ci-dessous. Cette configuration inclut le high-availability, la réplication des données et l'auto-unseal via KMS pour un fonctionnement fiable.

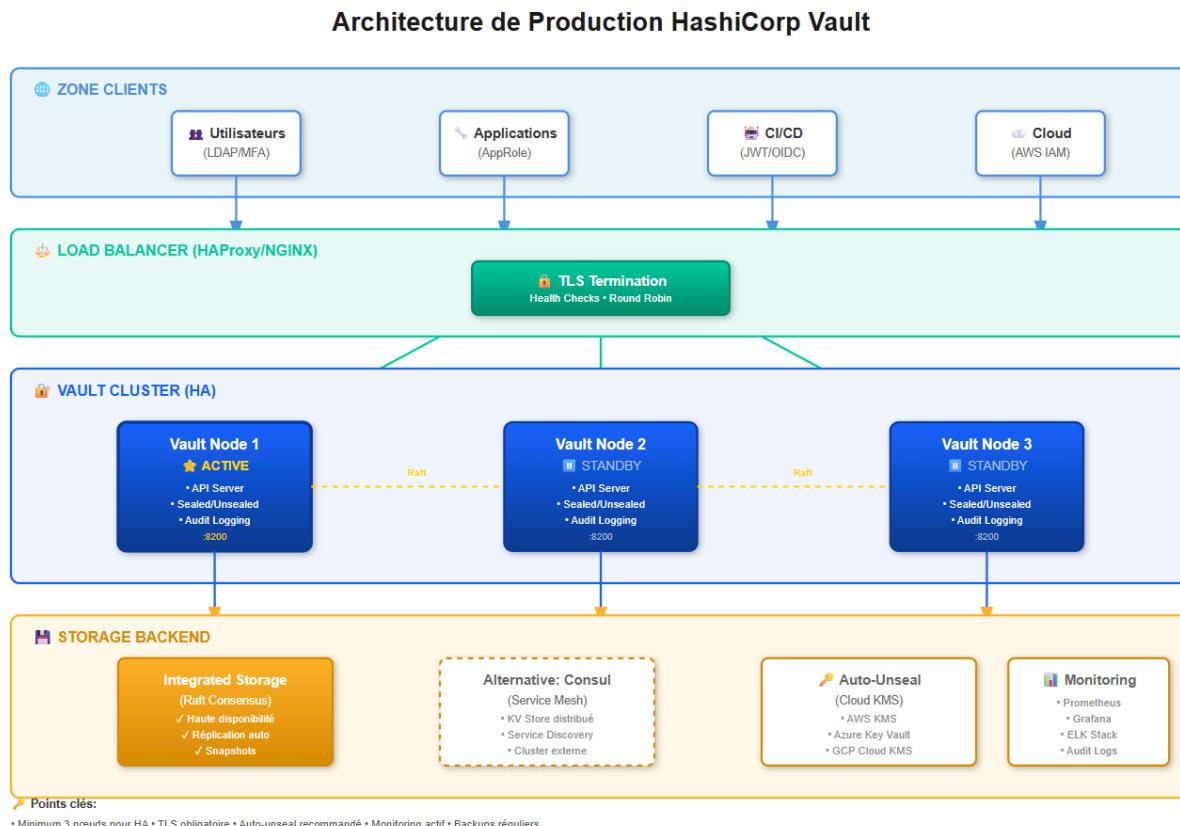


Figure 13.1: Architecture de production recommandée pour Vault

### 13.3 Checklist de sécurité

La checklist suivante regroupe les actions essentielles pour sécuriser un déploiement Vault. Elle peut servir de référence lors de l'audit ou de la mise en production.

#### 1. Initialisation

Générer un nombre suffisant de clés unseal (5 ou plus) pour garantir la résilience.

Distribuer les clés à différentes personnes de confiance pour éviter un point de défaillance unique.

Sauvegarder les clés dans plusieurs coffres-forts sécurisés.

Révoquer le root token initial après configuration.

#### 2. Configuration

Activer TLS avec des certificats valides pour chiffrer les communications.

Configurer l'auto-unseal via un KMS pour simplifier la mise en service après un reboot.

Activer l'audit logging sur plusieurs destinations pour garantir la disponibilité des logs.

Mettre en place le mode HA avec au moins 3 nœuds pour la haute disponibilité.

### 3. Accès

Utiliser AppRole pour les applications automatisées.

Implémenter une authentification multi-facteurs (MFA) pour les utilisateurs humains.

Définir des policies suivant le principe du moindre privilège.

Planifier et automatiser la rotation régulière des secrets.

### 4. Monitoring

Mettre en place des alertes sur les métriques critiques pour détecter les anomalies.

Examiner régulièrement les logs d'audit pour identifier tout comportement suspect.

Effectuer des tests de disaster recovery pour s'assurer de la résilience.

Maintenir une documentation complète et à jour pour toutes les configurations et procédures.

# Chapter 14

## Cas d'Usage Avancés

HashiCorp Vault ne se limite pas au stockage et à la rotation de secrets simples. Il offre des moteurs avancés qui permettent de gérer des infrastructures cryptographiques (PKI), de signer des certificats SSH ou de générer des OTP pour l'authentification à deux facteurs. Ce chapitre détaille ces cas d'usage avancés.

### 14.1 PKI (Public Key Infrastructure)

Le moteur PKI de Vault permet de créer et gérer des certificats X.509 de manière centralisée, en remplaçant ou en complétant une autorité de certification traditionnelle.

#### 14.1.1 Étapes de configuration

1. **Activer le moteur PKI** : Il faut activer le moteur PKI sur un chemin spécifique dans Vault.

```
1 vault secrets enable pki
```

2. **Générer une CA racine interne** : La CA (Certificate Authority) interne émettra les certificats pour vos services.

```
1 vault write pki/root/generate/internal \
2   common_name="My Company Root CA" \
3   ttl=87600h
```

3. **Configurer les URLs** : Ces URLs seront utilisées pour publier les certificats et la CRL (Certificate Revocation List).

```
1 vault write pki/config/urls \
2   issuing_certificates="https://vault.example.com:8200/v1/pki/ca" \
3   crl_distribution_points="https://vault.example.com:8200/v1/pki/crl"
```

4. **Créer un rôle pour les certificats** : Un rôle définit les domaines autorisés, les sous-domaines et la durée maximale de vie des certificats.

```
1 vault write pki/roles/example-dot-com \
2   allowed_domains="example.com" \
3   allow_subdomains=true \
4   max_ttl="720h"
```

5. **Émettre un certificat** : On peut générer un certificat pour un service ou une application.

```
1 vault write pki/issue/example-dot-com \
2   common_name="app.example.com" \
3   ttl="24h"
```

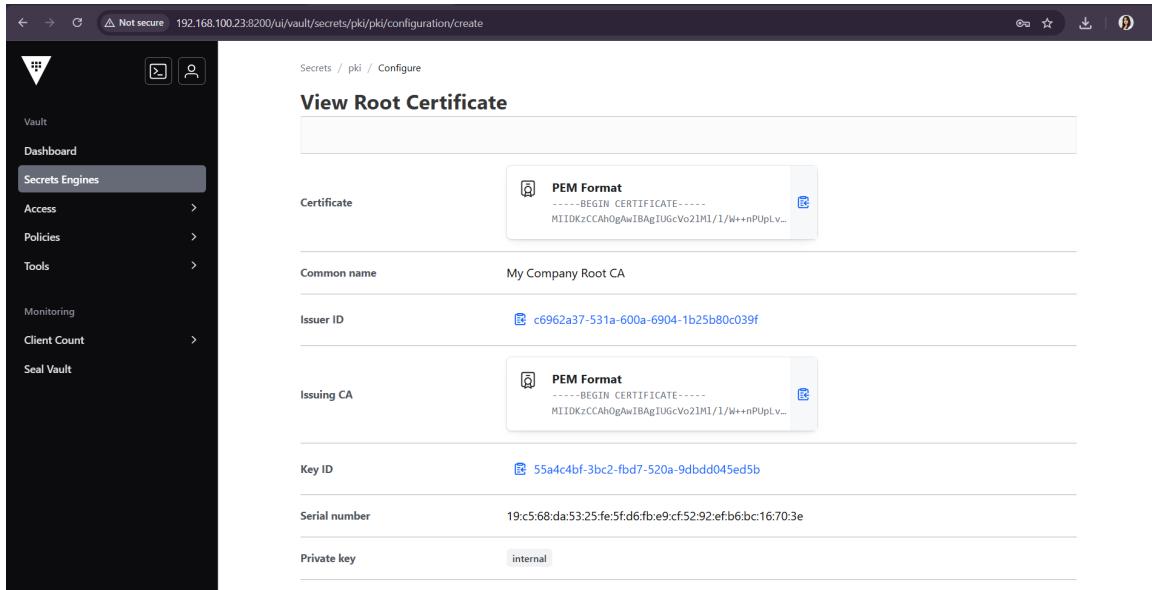


Figure 14.1: Génération d'un certificat avec le moteur PKI de Vault

## 14.2 SSH Secrets Engine

Vault peut gérer des certificats SSH signés, permettant d'authentifier des utilisateurs ou des machines sans gérer de clés statiques.

### 14.2.1 Configuration des certificats SSH

1. Activer le moteur SSH :

```
1 vault secrets enable ssh
```

2. Configurer une CA SSH : Vault génère une clé de signature qui servira à signer les clés publiques des utilisateurs.

```
1 vault write ssh/config/ca generate_signing_key=true
```

3. Créer un rôle SSH : Le rôle définit qui peut obtenir des certificats, leur durée et les extensions autorisées.

```
1 vault write ssh/roles/admin \
2   key_type=ca \
3   ttl=8h \
4   max_ttl=24h \
```

```

5      allow_user_certificates=true \
6      allowed_users="*" \
7      allowed_extensions="permit-pty,permit-port-forwarding" \
8      default_extensions="permit-pty=,permit-port-forwarding="

```

4. **Signer une clé publique :** L'utilisateur soumet sa clé publique et Vault retourne un certificat signé.

```
1 vault write ssh/sign/admin public_key=@$HOME/.ssh/id_rsa.pub
```

5. **Connexion SSH avec certificat :** Le certificat signé peut être utilisé pour se connecter à la machine distante.

```
1 ssh -i ~/.ssh/id_rsa -i ~/.ssh/id_rsa-cert.pub user@server
```

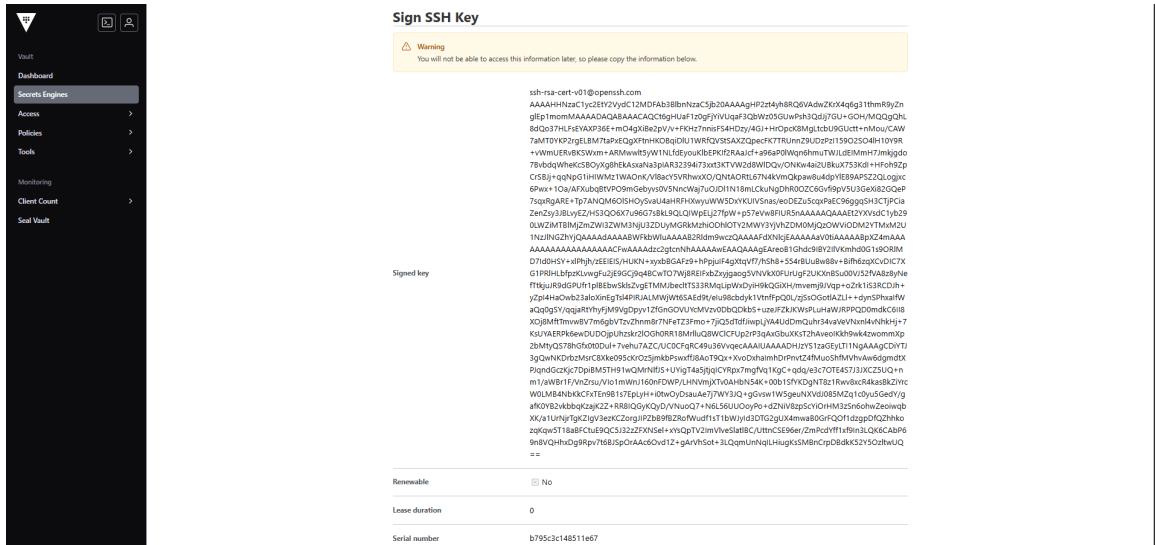


Figure 14.2: Signature d'une clé SSH via Vault

## 14.3 TOTP (Time-based One-Time Password)

Vault peut également générer des OTP (One-Time Password) pour l'authentification multifactor.

### 14.3.1 Configuration TOTP

1. **Activer le moteur TOTP :**

```
1 vault secrets enable totp
```

2. **Créer une clé TOTP pour un utilisateur/service :**

```

1 vault write totp/keys/aws-mfa \
2     issuer=AWS \
3     account_name=maryem@example.com \
4     generate=true

```

3. Obtenir le QR Code : L'utilisateur peut scanner le code dans une application Authenticator (Google Authenticator, Authy...)

```
1 vault read -field=barcode totp/keys/aws-mfa
```

4. Générer un code TOTP :

```
1 vault read totp/code/aws-mfa
```

# Chapter 15

## Conformité et Audit

La conformité et l'audit constituent des piliers essentiels de la sécurité des systèmes d'information. Dans un environnement manipulant des secrets sensibles (clés API, mots de passe, certificats), il est indispensable de disposer d'une traçabilité complète des accès et des opérations effectuées.

HashiCorp Vault intègre un mécanisme d'audit natif permettant d'enregistrer toutes les requêtes traitées par le serveur, qu'elles soient autorisées ou refusées. Ces journaux d'audit sont indispensables pour répondre aux exigences de normes telles que ISO 27001, SOC 2 ou PCI-DSS.

### 15.1 Configuration de l'audit

#### 15.1.1 Principe des audit devices

Un *audit device* dans Vault est une destination vers laquelle toutes les requêtes et réponses du serveur sont enregistrées sous forme de logs structurés au format JSON. Vault permet de configurer plusieurs audit devices simultanément afin de répondre aux besoins de sécurité, de redondance et d'intégration avec des outils tiers (SIEM).

#### 15.1.2 Multiples destinations d'audit

Dans cette configuration, trois types de destinations d'audit sont activés :

- un fichier local pour l'archivage,
- un service syslog pour la centralisation système,
- un socket réseau pour l'intégration avec un SIEM.

Listing 15.1: Configurer plusieurs audit devices

```
1 # Audit vers un fichier local
2 vault audit enable file file_path=/var/log/vault/audit.log
3
4 # Audit vers syslog
5 vault audit enable syslog tag="vault" facility="LOCAL7"
6
7 # Audit via socket (int gration SIEM)
8 vault audit enable socket \
9   address=siem.example.com:514 \
10  socket_type=tcp
```

```

1 # V rifier les audit devices actifs
2 vault audit list -detailed
3

```

Cette approche multi-destination garantit que les événements critiques sont conservés même en cas de défaillance d'un des mécanismes de journalisation.

=

La figure ?? illustre les audit devices actifs, confirmant que Vault enregistre correctement les événements sur plusieurs supports.

## 15.2 Analyse des logs d'audit

Les logs d'audit générés par Vault sont au format JSON, ce qui facilite leur analyse à l'aide d'outils tels que jq. Cette analyse permet d'identifier les comportements anormaux, les accès non autorisés et les tentatives d'attaque.

### 15.2.1 Exemples d'analyses avec jq

Listing 15.2: Analyser les logs d'audit avec jq

```

1 # Afficher toutes les op rations d'un utilisateur donn
2 cat audit.log | jq 'select(.auth.display_name=="john")'
3
4 # Filtrer les op rations sur un chemin sp cifique
5 cat audit.log | jq 'select(.request.path | contains("secret/prod"))'
6
7 # Identifier les op rations ayant chou
8 cat audit.log | jq 'select(.error != "")'
9
10 # Top 10 des utilisateurs les plus actifs
11 cat audit.log | jq -r '.auth.display_name' | sort | uniq -c | sort -rn |
12     head -10
13
14 # D tecter des acc s hors heures de bureau
15 cat audit.log | jq 'select(.time | fromdateiso8601 | strftime("%H") |
16     tonumber < 8 or tonumber > 18)'

```

Ces requêtes permettent de détecter rapidement des comportements suspects tels que des accès inhabituels ou des tentatives répétées échouées.

```
devops4@node1:~$ cat /tmp/vault_audit.log | jq 'select(.request.path | contains("secret"))'
[{"auth": {"accessor": "hmac-sha256:5e3713bc0063ff3c9191f472d581450f21c88ad856f83b5f3bd27a985dcc0f72", "client_token": "hmac-sha256:0e9ea4890363c617c2771dd90a10b234802c3cie05211e9bf7b51a2503b5001", "display_name": "root", "policies": ["root"]}, "policy_results": {"allowed": true}, "token_policies": [{"root"}], "token_issue_time": "2026-01-06T14:01:02+01:00", "token_type": "service"}, "request": {"client_id": "0DHqq2D77kL2JTPS2kTMjbkfVmUu0TzMl0jXcFy8=", "client_token": "hmac-sha256:0e9ea4890363c617c2771dd90a10b234802c3cie05211e9bf7b51a2503b5001", "client_token_accessor": "hmac-sha256:5e3713bc0063ff3c9191f472d581450f21c88ad856f83b5f3bd27a985dcc0f72", "id": "c2da5d72-0c10-dac2-b97d-f5d99f406e09", "mount_class": "secret", "mount_point": "sys", "mount_version": "v1.18.5+builtin.vault", "mount_type": "system", "namespace": {"id": "root"}, "operation": "read", "path": "sys/internal/ui/mounts/secret/audit-demo", "remote_address": "192.168.100.23", "remote_port": 54096}, "time": "2026-01-06T19:32:10.943130643Z", "type": "request"}], [{"auth": {"accessor": "hmac-sha256:5e3713bc0063ff3c9191f472d581450f21c88ad856f83b5f3bd27a985dcc0f72", "client_token": "hmac-sha256:0e9ea4890363c617c2771dd90a10b234802c3cie05211e9bf7b51a2503b5001", "display_name": "root", "policies": ["root"]}],
```

Figure 15.1: Analyse des logs d'audit Vault à l'aide de jq

## 15.3 Rapports de conformité

Afin de faciliter les audits périodiques, il est utile d'automatiser la génération de rapports de conformité à partir des logs d'audit. Le script Python suivant analyse les journaux Vault et produit un rapport synthétique au format HTML.

### 15.3.1 Génération automatique de rapports

Listing 15.3: Script de génération de rapport de conformité

```
#!/usr/bin/env python3
import json
import sys
from datetime import datetime
from collections import Counter

def analyze_audit_log(log_file):
    """Analyse le log d'audit Vault"""

    stats = {
        'total_operations': 0,
        'failed_operations': 0,
        'users': Counter(),
        'paths': Counter(),
        'operations': Counter(),
        'errors': []
    }

    with open(log_file, 'r') as f:
        for line in f:
            try:
                entry = json.loads(line)
                stats['total_operations'] += 1
```

```
24
25         user = entry.get('auth', {}).get('display_name', 'unknown')
26         stats['users'][user] += 1
27
28         path = entry.get('request', {}).get('path', 'unknown')
29         stats['paths'][path] += 1
30
31         op = entry.get('request', {}).get('operation', 'unknown')
32         stats['operations'][op] += 1
33
34         if entry.get('error'):
35             stats['failed_operations'] += 1
36             stats['errors'].append({
37                 'time': entry.get('time'),
38                 'user': user,
39                 'path': path,
40                 'error': entry.get('error')
41             })
42     except json.JSONDecodeError:
43         continue
44
45     return stats
46
47 def generate_report(stats):
48     """Génère un rapport HTML de conformité"""
49
50     html = f"""
51     <html>
52     <head>
53         <title>Rapport de Conformité Vault</title>
54         <style>
55             body {{ font-family: Arial, sans-serif; margin: 20px; }}
56             table {{ border-collapse: collapse; width: 100%; }}
57             th, td {{ border: 1px solid #ddd; padding: 10px; }}
58             th {{ background-color: #3498db; color: white; }}
59         </style>
60     </head>
61     <body>
62         <h1>Rapport de Conformité Vault</h1>
63         <p>Génééré le : {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}</p>
64
65         <h2>Statistiques Générales </h2>
66         <ul>
67             <li>Total d'opérations : {stats['total_operations']}
68             <li>Opérations échouées : {stats['failed_operations']}
69         </ul>
70
71         <h2>Utilisateurs les plus actifs</h2>
72         <table>
73             <tr><th>Utilisateur </th><th>Opérations </th></tr>
74         """
75
76         for user, count in stats['users'].most_common(10):
77             html += f"<tr><td>{user}</td><td>{count}</td></tr>"
78
79     html += "</table></body></html>"
```

```
$0     return html
$1
$2 if __name__ == '__main__':
$3     stats = analyze_audit_log(sys.argv[1])
$4     report = generate_report(stats)
$5     with open('compliance_report.html', 'w') as f:
$6         f.write(report)
```

The screenshot shows a web browser window with the title 'Rapport de Conformité Vault'. The URL in the address bar is 'file:///home/devops4/compliance\_report.html'. The page content is a table titled 'Rapport de Conformité Vault' with three columns: 'Chemin', 'Opération', and 'Nombre'. The data in the table is as follows:

Chemin	Opération	Nombre
sys/audit/test	update	1
sys/audit/file	update	1
sys/audit	read	2
sys/metrics	read	146
sys/internal/ui/mounts/secret/audit-demo	read	6
secret/audit-demo	create	2
secret/audit-demo	read	2
secret/audit-demo	delete	2

Figure 15.2: Rapport de conformité HTML généré automatiquement

La figure 15.2 montre un exemple de rapport de conformité généré, facilitant les audits de sécurité et la démonstration de conformité réglementaire.

# Chapter 16

## Commandes Utiles

Ce chapitre regroupe les principales commandes de HashiCorp Vault utilisées au quotidien par les administrateurs systèmes et les ingénieurs DevSecOps. Ces commandes permettent de superviser l'état du serveur, de gérer les secrets, les politiques de sécurité, l'authentification ainsi que les mécanismes d'audit.

### 16.1 Commandes CLI essentielles

Les commandes suivantes constituent la base de l'administration de Vault via l'interface en ligne de commande (CLI). Elles couvrent les opérations les plus courantes liées à la supervision, à la gestion des accès et à la sécurité.

Listing 16.1: Commandes Vault essentielles

```
1 # V rification de l' tat et de la sant du serveur
2 vault status
3 vault read sys/health
4
5 # Gestion des secrets (KV engine)
6 vault kv put secret/path key=value          # Cr ation ou mise jour
     d un secret
7 vault kv get secret/path                   # Lecture d un secret
8 vault kv delete secret/path                # Suppression d un secret
9 vault kv list secret/                     # Liste des secrets disponibles
10
11 # Authentification et gestion des tokens
12 vault login                                # Authentification Vault
13 vault token lookup                         # Informations sur le token
     courant
14 vault token renew                          # Renouvellement du token
15
16 # Gestion des politiques de s curit (ACL)
17 vault policy list                          # Liste des policies existantes
18 vault policy read policy-name            # Lecture d une policy
19 vault policy write policy-name policy.hcl# Cr ation ou mise jour
     d une policy
20
21 # Configuration et gestion de l audit
22 vault audit list                           # Liste des audit devices actifs
23 vault audit enable file file_path=/var/log/vault/audit.log
24
25 # Gestion des moteurs de secrets
```

```
26 vault secrets list          # Liste des secrets engines  
27     activ s  
28 vault secrets enable -path=custom database  
29 vault secrets disable custom/  
30  
30 # Gestion des méthodes d authentification  
31 vault auth list           # Méthodes d authentification  
32     disponibles  
32 vault auth enable userpass  
33 vault auth disable userpass/
```

# Chapter 17

## Guide de Dépannage

Ce chapitre présente les problèmes les plus fréquemment rencontrés lors de l'utilisation de HashiCorp Vault, ainsi que les solutions recommandées. Il constitue une référence rapide pour le diagnostic et la résolution des incidents opérationnels.

### 17.1 Problèmes courants et solutions

Problème	Solution recommandée
Vault scellé (Sealed)	Exécuter la commande <code>vault operator unseal</code> et fournir le nombre requis de clés de déchiffrement afin de rendre Vault opérationnel.
Permission denied	Vérifier les capacités du token à l'aide de la commande <code>vault token capabilities &lt;token&gt; &lt;path&gt;</code> et ajuster la policy si nécessaire.
Connexion refusée	Vérifier que le serveur Vault est en cours d'exécution et que la variable d'environnement <code>VAULT_ADDR</code> pointe vers la bonne adresse.
Erreur de certificat TLS	Contrôler la validité des certificats TLS. En environnement de test, il est possible de désactiver temporairement la vérification avec <code>VAULT_SKIP_VERIFY=true</code> .
Token expiré	Renouveler le token avec <code>vault token renew</code> ou effectuer une nouvelle authentification.

Table 17.1: Guide de dépannage des problèmes courants rencontrés avec Vault

# Chapter 18

## Glossaire

Cette section définit les principaux termes techniques utilisés dans l'écosystème HashiCorp Vault afin de faciliter la compréhension des concepts abordés dans ce document.

**Seal / Unseal** État de protection de Vault. Lorsqu'il est scellé, les données chiffrées sont inaccessibles. Le déverrouillage (Unseal) permet de charger la clé maître en mémoire.

**Token** Jeton d'authentification utilisé pour accéder à Vault, associé à des politiques de sécurité définissant les permissions.

**Lease** Durée de validité d'un secret dynamique, après laquelle celui-ci expire automatiquement.

**Policy** Ensemble de règles de contrôle d'accès (ACL) définissant les actions autorisées sur des chemins Vault spécifiques.

**Secret Engine** Composant responsable de la génération, du stockage et de la gestion des secrets (KV, Database, AWS, etc.).

**Root Token** Token initial disposant de privilèges complets, généré lors de l'initialisation de Vault.

**AppRole** Méthode d'authentification destinée aux applications automatisées et aux workloads non humains.

**Transit** Moteur de chiffrement fournissant des services de cryptographie sans stocker les données sensibles.

**PKI** Infrastructure de gestion des clés publiques permettant la génération et la gestion de certificats TLS.

**Namespace** (Vault Enterprise) Mécanisme d'isolation logique permettant de séparer plusieurs environnements au sein d'un même cluster.

**Replication** (Vault Enterprise) Fonctionnalité permettant la réPLICATION des données entre plusieurs clusters Vault.

## Fin du voyage, début de l'aventure

*Cette documentation est le fruit d'une exploration passionnée  
dans l'univers fascinant de la cybersécurité.*

*Chaque page reflète un engagement envers la rigueur,  
l'innovation et la soif d'apprendre.*

— Maryem Cherif —

« La sécurité n'est pas une destination, c'est un voyage continu. »