### **1. Préparer l’authentification côté HVault**

| **Étape** | **Détails** |
| --- | --- |
| **1.1 Activer la méthode d’authentification cert (ou Kubernetes si tu préfères)** | vault auth enable cert  ou vault auth enable kubernetes |
| **1.2 Générer / charger le certificat client** | - Certificat **X.509** signé par l’autorité interne (PKI CTDF).  - Import dans le keystore du composant (Java, Python…). |
| **1.3 Déclarer le certificat à Vault** | Pour cert : vault write auth/cert/certs/<app\_name> display\_name="<app\_name>" certificate=@client.crt policies="<policy\_list>"  — ou map service-account → rôle Kubernetes. |

### **2. Activer le secrets-engine ElasticSearch**

bash

CopierModifier

vault secrets enable -path=elastic\_ctdf elasticsearch

vault write elastic\_ctdf/config \

url="https://<cluster>:9200" \

username="vault\_manager" \

password="<password>" \

verify=true \

allowed\_roles="rag\_ro,evaluator\_rw"

**Idée :** garder le secrets-engine dans le même namespace que la BDD DFS pour rester cohérent.

### **3. Créer une *policy* par “certificat” (ou rôle)**

| **Rôle / Policy** | **Droits ElasticSearch** | **Path Vault autorisé** |
| --- | --- | --- |
| rag\_ro | lecture seule (indices RAG) | elastic\_ctdf/creds/rag\_ro |
| evaluator\_rw | lecture + écriture | elastic\_ctdf/creds/evaluator\_rw |

Exemple de policy :

hcl

CopierModifier

# policy rag\_ro.hcl

path "elastic\_ctdf/creds/rag\_ro" {

capabilities = ["read"]

}

### **4. Générer un token temporaire à l’exécution**

1. **Le composant présente son certificat** POST /v1/auth/cert/login (ou /auth/kubernetes/login)
2. **Vault rend un token TTL court** (15 min, 1 h…) lié à la *policy*.
3. **Le composant lit les credentials dynamiques** GET /v1/elastic\_ctdf/creds/rag\_ro → reçoit un *username/password* ElasticSearch à durée de vie contrôlée (par ex. 1 h).
4. **ElasticSearch invalide automatiquement l’utilisateur** quand le lease expire – pas de secret “qui traîne”.

### **5. Points d’attention / REX**

* **TTL & renouvellement** – Ajuster le max\_ttl du secrets-engine et la durée des tokens pour éviter les renouvellements trop fréquents.
* **Rotation transparente** – Les credentials ElasticSearch sont créés à la volée ; aucun dépôt de secrets statiques dans Git/CI.
* **Audit** – Les logs Vault montrent qui a demandé quel rôle et quand, pratique pour les revues sécurité.
* **Namespace** – si plusieurs Squads doivent accéder au même cluster, préférer un secrets-engine par Squad pour isoler les droits.

### **TL;DR pour ta réponse Teams**

*“Pour accéder proprement aux credentials ElasticSearch via HVault :  
 1️⃣ On crée un certificat X.509 par composant et on l’enregistre dans Vault via la méthode d’auth cert (ou K8s).  
 2️⃣ On active le secrets-engine ElasticSearch dans le namespace CTDF et on définit les rôles (rag\_ro, evaluator\_rw…).  
 3️⃣ On écrit une -policy par rôle ; chaque certificat hérite de la policy qui lui correspond.  
 4️⃣ À l’exécution, le composant présente son certificat → Vault lui retourne un token éphémère → il lit elastic\_ctdf/creds/<role> pour obtenir des identifiants dynamiques ElasticSearch.  
 Cela nous évite de stocker des mots de passe statiques et on bénéficie de la rotation / revocation automatique de Vault.”*