Le Contrôleur Sealed Secrets

1. Comprendre le concept (Le problème vs La solution)

Le Problème

Normalement, un "Secret" Kubernetes est juste encodé en Base64. C'est comme écrire un mot de passe à l'envers : n'importe qui peut le traduire en 2 secondes. Tu ne peux donc pas mettre tes fichiers de secrets sur GitHub.

La Solution (Sealed Secrets)

On utilise le **chiffrement asymétrique** :

- **La Clé Publique (Le Cadenas) :** Tu l'utilises sur ton ordinateur ou ton VPS pour verrouiller (sceller) un secret. Une fois scellé, **personne** (pas même toi) ne peut voir ce qu'il y a dedans.
- **La Clé Privée (La Clé du Coffre) :** Elle reste **uniquement** à l'intérieur de ton cluster K3s, dans le contrôleur. Lui seul a le pouvoir d'ouvrir le secret pour le lire.

---

2. Installation du Contrôleur (Le "Gardien")

Le contrôleur est un programme qui tourne en permanence dans ton cluster pour surveiller les nouveaux secrets scellés.

Commande d'installation (via Helm) :

Bash

```
# 1. Ajouter le dépôt Bitnami
helm repo add sealed-secrets https://bitnami-labs.github.io/sealed-secrets

# 2. Installer le contrôleur dans le namespace de gestion
helm install sealed-secrets sealed-secrets/sealed-secrets -n kube-system
```

3. Installation de l'outil Client (kubeseal)

kubeseal est l'outil que tu installes sur ton VPS pour transformer tes secrets "en clair" en secrets "scellés".

Commandes d'installation sur le VPS :

Bash

```
# 1. Récupérer la dernière version

KUBESEAL_VERSION=$(curl -s https://api.github.com/repos/bitnami-labs/sealed-secrets/releases/latest | jq -r .tag_name | sed 's/v//')
```

```
# 2. Télécharger et installer le binaire

curl-L "https://github.com/bitnami-labs/sealed-secrets/releases/download/v${KUBESEAL_VERSION}/kubeseal-${KUBESEAL_VERSION}-linux-amd64.tar.gz" -o kubeseal.tar.gz

tar -xvzf kubeseal.tar.gz kubeseal

sudo install -m 755 kubeseal /usr/local/bin/kubeseal
```

4. Le Script: seal-secret.sh

Il automatise le chiffrement et surtout, il **détruit les preuves** (le fichier en clair).

Création du script :

```
Bash

cat <<'EOF' > seal-secret.sh

#!/bin/bash

# Fichiers de configuration

CERT_FILE="sealed-secrets-public.pem"

OUTPUT_FILE="my-sealed-secrets.yaml"


# 1. Récupérer le certificat public du cluster s'il n'existe pas

# C'est ce certificat qui sert de "Cadenas"

if [ ! -f "$CERT_FILE" ]; then

   echo " 📥 Récupération du certificat public du cluster..."

   kubeseal --controller-name=sealed-secrets --controller-namespace=kube-system --fetch-cert > $CERT_FILE

fi


# 2. Vérifier qu'un fichier a été donné en argument

if [ -z "$1" ]; then

   echo " ❌ Erreur : Tu dois donner un fichier .yaml à chiffrer."

   exit 1

fi


# 3. Chiffrer le secret (Scellement)

echo " 🔒 Chiffrement du secret..."

kubeseal --cert $CERT_FILE < "$1" >> $OUTPUT_FILE


# 4. Suppression immédiate du fichier en clair pour la sécurité

rm -f "$1"

echo " ✅ Terminé ! Le secret est scellé dans $OUTPUT_FILE et le fichier original a été supprimé."

EOF

chmod +x seal-secret.sh
```

5. Workflow Quotidien : Étape par Étape

Désormais, quand tu veux ajouter un secret (ex: un mot de passe de base de données), tu fais ceci :

1. **Créer un secret temporaire (YAML en clair) :**

Bash

```
kubectl create secret generic platform-prod-secrets \
 --from-literal=DB_PASSWORD="MonSuperMotDePasse" \
 --dry-run=client -o yaml > temp-secret.yaml
```

2. **Lancer le script de scellement :**

Bash

```
./seal-secret.sh temp-secret.yaml
```

   - *Le fichier temp-secret.yaml disparaît immédiatement.*
   - *Le fichier my-sealed-secrets.yaml est mis à jour avec le code chiffré.*

3. **Appliquer dans le cluster :**

Bash

```
kubectl apply -f my-sealed-secrets.yaml
```

``` [cite: 74]

## 6. Sécurité Critique : La Clé Privée (La "Master Key")

**Attention :** Si tu réinstalles ton VPS et que tu ne sauvegardes pas la clé privée du contrôleur, ton fichier my-sealed-secrets.yaml deviendra **inutilisable**. C'est comme si tu avais perdu la seule clé capable d'ouvrir ton coffre-fort.

**Commande pour sauvegarder ta clé privée (À mettre à l'abri sur une clé USB) :**

Bash

```
kubectl get secret -n kube-system -l sealedsecrets.bitnami.com/sealed-secrets-key -
```
```

🛠️ Procédure de Modification (Étape par Étape)

Étape 1 : Créer le nouveau secret "en clair" localement

Tu dois d'abord générer un fichier temporaire qui contient la nouvelle valeur.

```
kubectl create secret generic platform-prod-secrets \
--from-literal=MASTER_DB_PASSWORD="NOUVEAU_MOT_DE_PASSE_ICI" \
--dry-run=client -o yaml > secret-temp.yaml
```

Étape 2 : Utiliser le script pour sceller la modification

Lance ton script sur ce fichier temporaire.

```
./seal-secret.sh secret-temp.yaml
```

Le script prend ton secret-temp.yaml.

Il demande au contrôleur de le chiffrer. Il ajoute (ou remplace) le résultat dans ton fichier my-sealed-secrets.yaml. Il supprime secret-temp.yaml pour que ton mot de passe ne reste pas sur le disque.

Étape 3 : Envoyer la mise à jour au cluster

Maintenant que ton fichier my-sealed-secrets.yaml contient la version chiffrée de ton nouveau mot de passe, dis à Kubernetes de le prendre en compte :

```
kubectl apply -f my-sealed-secrets.yaml
```

Étape 4 : Vérifier la prise en compte

Le contrôleur Sealed Secrets va détecter le changement, déchiffrer le nouveau mot de passe et mettre à jour le "vrai" secret Kubernetes. Pour vérifier que c'est fait :

```
kubectl describe sealedsecret platform-prod-secrets
```

Cherche la ligne "Synced" à la fin.

```
kubectl rollout restart deployment platform-prod-bastion
```

. Comment faire si j'ai oublié les autres clés du secret ? Si tu veux modifier une seule clé sans casser les autres, tu dois d'abord récupérer les valeurs actuelles (puisque tu n'as plus le fichier en clair) :

Bash

```
# Pour voir la valeur actuelle d'une clé
kubectl get secret platform-prod-secrets -o jsonpath='{.data.NOM_DE_LA_CLE}' | base64 –decode
```

1. Voir ce qui est actuellement dans le cluster (Déchiffré)

Utilise cette commande pour afficher toutes les données du secret actuel en clair :

Bash

```
kubectl get secret platform-prod-secrets -o json | jq '.data | map_values(@base64d)'
```

Note : Si jq n'est pas installé, utilise cette commande pour vérifier une clé spécifique (par exemple AUTHORIZED_ORIGINS ou WEBSITE_DOMAIN) : kubectl get secret platform-prod-secrets -o jsonpath='{.data.WEBSITE_DOMAIN}' | base64 --decode && echo ""

2. Comparer avec ton fichier exemple

Maintenant, affiche ton fichier de référence pour voir si les valeurs correspondent :

Bash

```
cat values-secrets.example.yaml
```

3. Ce qu'il faut vérifier en priorité (Authorized Origins)

Dans ton architecture, les "origines autorisées" sont souvent liées à ces variables:

- **WEBSITE_DOMAIN** : L'URL de ton frontend (ex: http://162.19.251.31:4321).
- **API_DOMAIN** : L'URL de ton API (ex: https://api.auth.annestay.com).