kubeseal Sealed Secrets CLI

CONCEPTS

Only the controller's private key can decrypt a SealedSecret, so sealed files are safe to commit to Git.

Secret plain, local only
↓ kubeseal
SealedSecret encrypted, in Git
↓ kubectl apply
Controller holds private key
↓ decrypts
Secret in the cluster

The default strict scope binds name and namespace into the ciphertext: renaming or moving the file breaks decryption.

Sealing keys rotate (every 30 days by default) but old keys stay, so old sealed files keep working. Back up the keys.

SEAL

kubeseal -f *<secret>* **-w** *<sealed>*
Seal a Secret file

kubeseal < *<secret>* **>** *<sealed>*
Seal via stdin and stdout

kubectl create secret generic *<name>* **--dry-run=client --from-literal=**<*k*>=<*v*> **-o json | kubeseal >** *<sealed>*
Create and seal in one pipe

kubeseal -o yaml < *<secret>*
Output YAML instead of JSON

kubeseal --namespace *<ns>* **<** *<secret>*
Seal for a namespace

kubectl apply -f *<sealed>*
Deploy the SealedSecret

kubectl get sealedsecret
List SealedSecrets

kubectl get secret *<name>*
Check the controller made the Secret

CERTIFICATE

kubeseal --fetch-cert > *<pub.pem>*
Download the controller's public cert

CERTIFICATE (cont.)

kubeseal --cert *<pub.pem>* **<** *<secret>*
Seal offline with a saved cert

CONTROLLER

kubeseal --controller-name *<name>*
Name of the controller service

kubeseal --controller-namespace *<ns>*
Namespace of the controller

kubeseal --version
Show kubeseal version

SCOPES

kubeseal --scope strict
Bind name and namespace (default)

kubeseal --scope namespace-wide
Allow renaming inside the namespace

kubeseal --scope cluster-wide
Allow any name and namespace

Scope annotations

Annotation on the Secret	Effect
`sealedsecrets.bitnami.com/namespace-wide: "true"`	Namespace-wide scope
`sealedsecrets.bitnami.com/cluster-wide: "true"`	Cluster-wide scope, wins over the other

MAINTAIN

kubeseal --validate < *<sealed>*
Check the controller can decrypt it

kubeseal --re-encrypt < *<sealed>* **>** *<new>*
Re-encrypt with the newest key

kubeseal --merge-into *<sealed>* **<** *<secret>*
Add values to an existing SealedSecret

kubectl get secret -n kube-system -l sealedsecrets.bitnami.com/sealed-secrets-key -o yaml > *<main.key>*
Back up the sealing keys

MAINTAIN (cont.)

kubectl apply -f *<main.key>*
Restore keys, then restart the controller pod

RAW MODE

echo -n *<value>* **| kubeseal --raw --name** *<n>* **--namespace** *<ns>*
Encrypt one value (strict)

echo -n *<value>* **| kubeseal --raw --scope cluster-wide**
Encrypt one value, cluster-wide

SECRET ANNOTATIONS

Controller behaviour

Annotation (sealedsecrets.bitnami.com/)	Effect
`managed: "true"`	Take over an existing Secret
`patch: "true"`	Add keys, keep others (0.23+)
`skip-set-owner-references: "true"`	Secret outlives its SealedSe

RECIPES

Seal a new secret

```
kubectl create secret generic db
--dry-run=client \
  --from-literal=password=<pw> -n
app -o json \
  | kubeseal > db-sealed.json
kubectl apply -f db-sealed.json
```

Seal in CI without cluster access

```
# once, with cluster access
kubeseal --fetch-cert > pub.pem
# later, anywhere
kubeseal --cert pub.pem -f
secret.yaml -w sealed.yaml
```

Add a value to a sealed secret

```
echo -n <new> | kubectl create
secret generic db \
  --dry-run=client --from-
file=token=/dev/stdin -o json \
  | kubeseal --merge-into db-
sealed.json
```

Rotate to the latest key

```
kubeseal --re-encrypt < db-
sealed.json > tmp.json \
  && mv tmp.json db-sealed.json
kubectl apply -f db-sealed.json
```