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.

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

`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

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