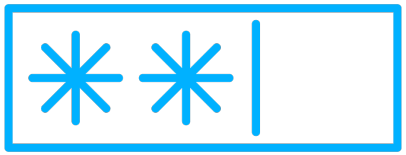
Create and Consume Secrets

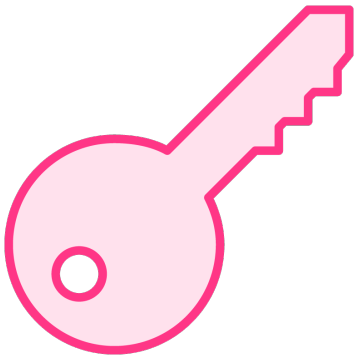
Elle Krout
Principal Course Author, Pluralsight

Kubernetes Secrets

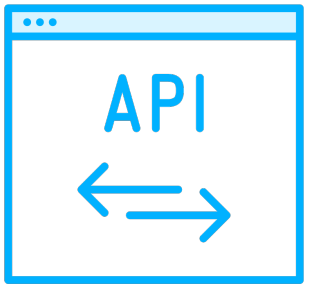
Kubernetes Secret Data



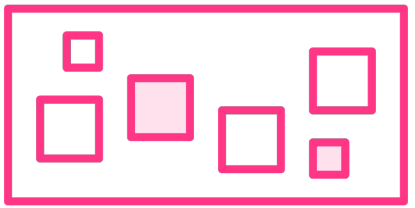
Passwords



Key Pairs



API Keys



Private Registries

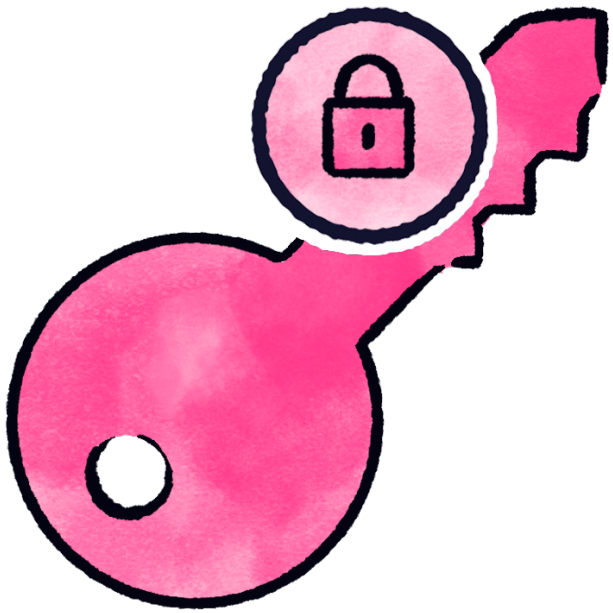
Secrets Can Be Passed into Containers As...

Arguments

Environmental Variables

Files

Secrets



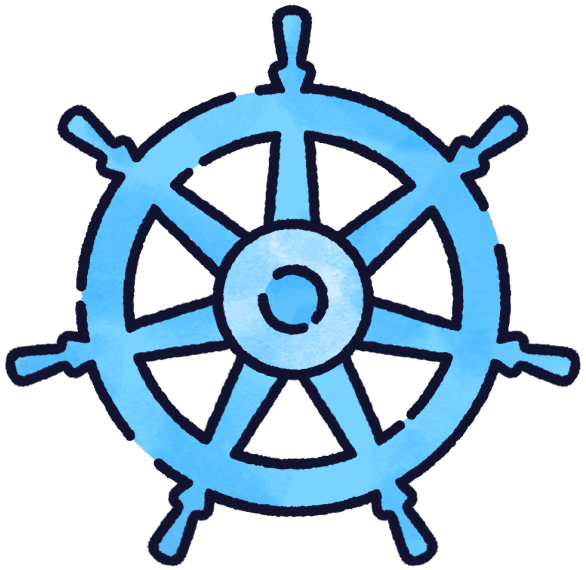
Stored unencrypted in `etcd`

Accessible by anyone with API access

Consider enabling encryption at rest

– Not part of the exam

Use restrictive RBAC policies



For the Exam

Focus on creating, encoding, and consuming secrets in pods.

Creating Kubernetes Secrets

Secret Object Manifest

db-secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
data:
  username: YWRtaW4=
  password: UGE1NXcwcmQ=
```

Secret Object Types

Scope	Matches
`Opaque`	Arbitrary user-defined data
`kubernetes.io/service-account-token`	Service account token data
`kubernetes.io/dockercfg`	Serialized `~/.dockercfg` file data
`kubernetes.io/dockerconfigjson`	serialized `~/.docker/config.json` file data
`kubernetes.io/basic-auth`	Credentials for basic authentication
`kubernetes.io/ssh-auth`	Credentials for SSH authentication
`kubernetes.io/tls`	Data for a TLS client or server
`bootstrap.kubernetes.io/token`	Bootstrap token data

Secret Object Manifest (TLS)

tls-secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: kubernetes.io/tls
data:
  tls.cert: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCg...
  tls.key: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0...
```

Secret Object Manifest (Docker Registry)

registry-secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: docker-registry-secret
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: eyJhdXRocyI6eyJodHRwczovL215c...
```

kubernetes.io/docs/home

Exam Tip!

Be aware of your weak spots ahead of time and leverage the docs during these tasks.

When using the `data` field, content must be pre-encoded

If not, use `stringData`

Secret Object Manifest (Unencoded)

db-secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
data:
  username: user01
  password: encodethispassword
```

Encode Data

```
> echo -n "encodethispassword" | base64
```

```
ZW5jb2RldGhpc3Bhc3N3b3Jk
```

Create Secret via CLI

```
> kubectl create secret tls tls-secret --cert=/cert/path --key=/key/path

secret/tls-secret created
```

Create Secret (Help)

```
> kubectl create secret docker-registry --help
```

```
Create a new secret for use with Docker registries.

        Dockercfg secrets are used to authenticate against Docker registries.

        When using the Docker command line to push images, you can authenticate to a given registry by
running:
        '$ docker login DOCKER_REGISTRY_SERVER --username=DOCKER_USER --password=DOCKER_PASSWORD --
email=DOCKER_EMAIL'.

That produces a ~/.dockercfg file that is used by subsequent 'docker push' and 'docker pull' commands
to authenticate to the registry. The email address is optional.

        When creating applications, you may have a Docker registry that requires authentication.  In
        order for the nodes to pull images on your behalf, they must have the credentials.  You can
        provide this information by creating a dockercfg secret and attaching it to your service
        account.
```

View Secrets

```
> kubectl get secret

NAME          TYPE                  DATA   AGE
db-secret     Opaque                2      28s
webhook-tls   kubernetes.io/tls     2      21d
```

View Secret Information

```
> kubectl describe secret db-secret

Name:         db-secret
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
password:  8 bytes
username:  4 bytes
```

View Secret Data

```
> kubectl get secret db-secret -o jsonpath='{.data.username}'

dXNlcg==
```

Decode Secret Data

```
> kubectl get secret db-secret -o jsonpath='{.data.username}' | base64 --decode

user


> echo "dXNlcg==" | base64 --decode

user
```

Secrets

Let you securely manage sensitive data in Kubernetes, whether through manifests for via the CLI.

Using Secrets in Pods

Secrets as Environment Variables

envfrom-secret-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envfrom-secret-pod
spec:
  containers:
  - name: myapp
    image: busybox
    command: ["sh", "-c", "env"]
    envFrom:
      - secretRef:
          name: db-credentials
```

Secrets as Environment Variables (Partial)

env-secret-pod.yaml

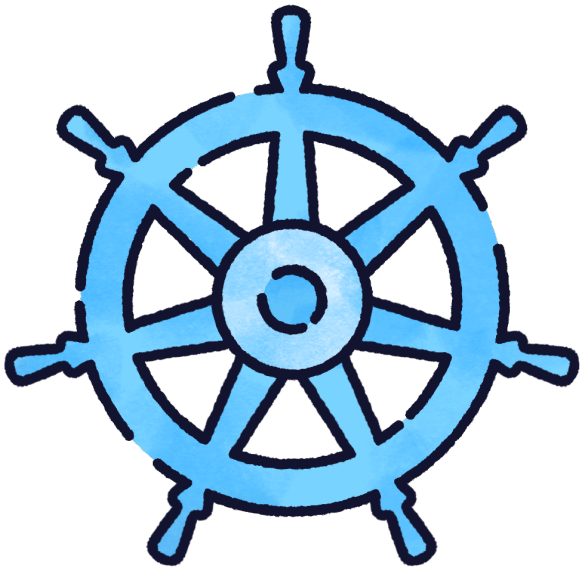
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: env-secret-pod
spec:
  containers:
  - name: myapp
    image: busybox
    command: ["sh", "-c", "env"]
    env:
      - name: DB_USER
        valueFrom:
          secretKeyRef:
            name: db-credentials
            key: username
```

Secrets as Volumes

volume-secret-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: volume-secret-pod
spec:
  containers:
  - name: myapp
    image: busybox
    command: ["sh", "-c", “env"]
    volumeMounts:
      - name: secret-volume
        mountPath: "/etc/secret"
        readOnly: true
    volumes:
      - name: secret-volume
        secret:
          name: db-credentials
```

Secret data is decoded once in the pod and can be accessed from the pod.



When Pulling in Secrets...

Consider your application, as well as the level of access needed, exposing only what's needed to the pod.

Demo: Working with Secrets in Kubernetes

Exam Scenario

A deployment found at `secret-service.yaml` needs secure environmental variables found at `secret.env` provided to its pod. Likewise, it also needs the TLS certificate and key provided at `tls.crt` and `tls.key` mounted. Configure the deployment to use this secure data