Creating ConfigMaps and Secrets

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Module Overview

ConfigMaps Core Concepts

Creating a ConfigMap

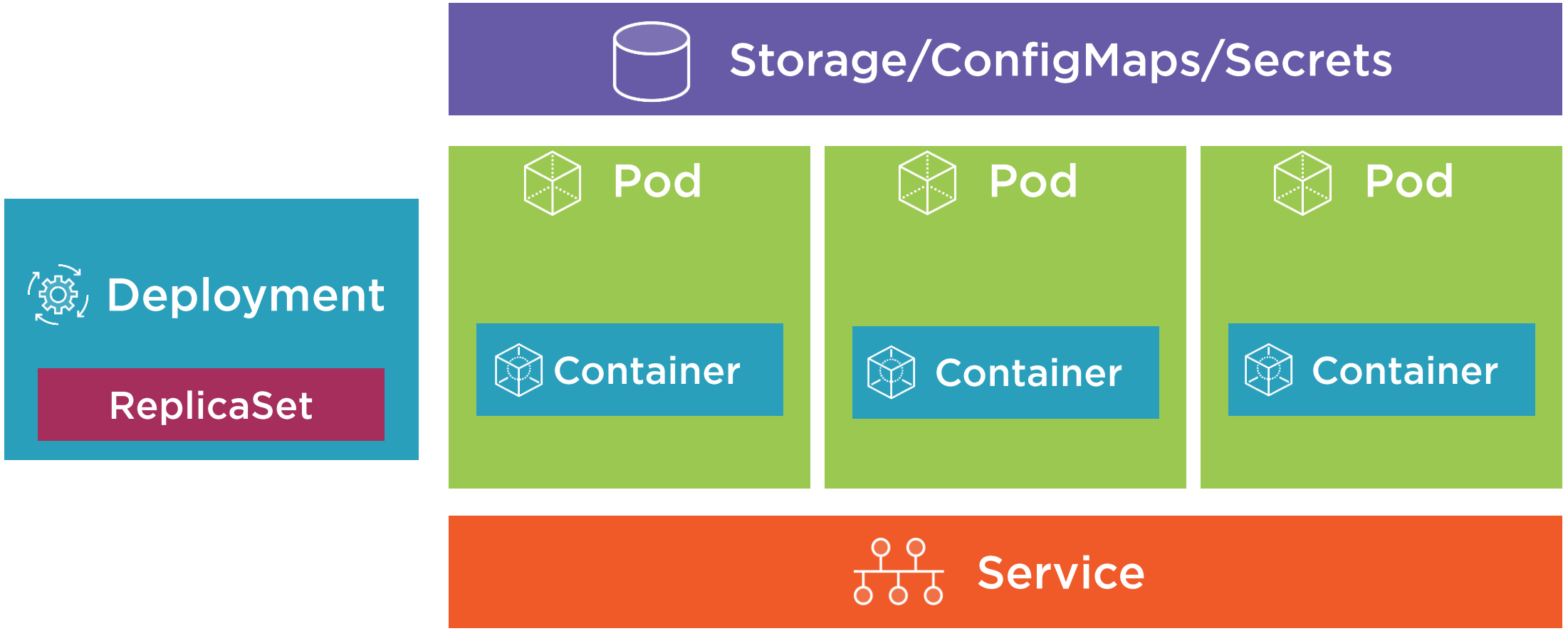
Using a ConfigMap

Secrets Core Concepts

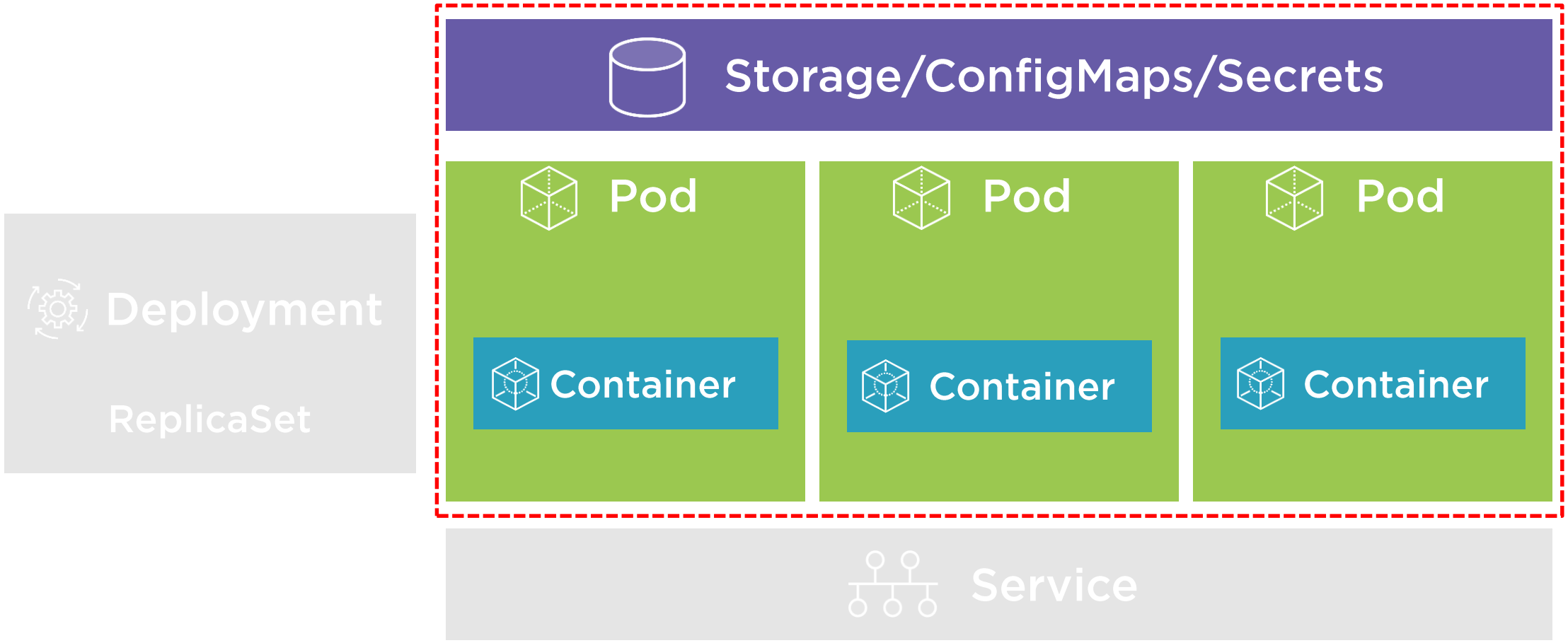
Creating a Secret

Using a Secret

You Are Here


Storage/ConfigMaps/Secrets
Pod
Pod
Pod
Deployment
ReplicaSet
Container
Container
Container
Service

You Are Here


Storage/ConfigMaps/Secrets
Pod
Pod
Pod
Container
Container
Container
Deployment
ReplicaSet
Service

ConfigMaps Core Concepts

ConfigMaps provide a way to store configuration information and provide it to containers.

ConfigMaps

Provides a way to inject configuration data into a container

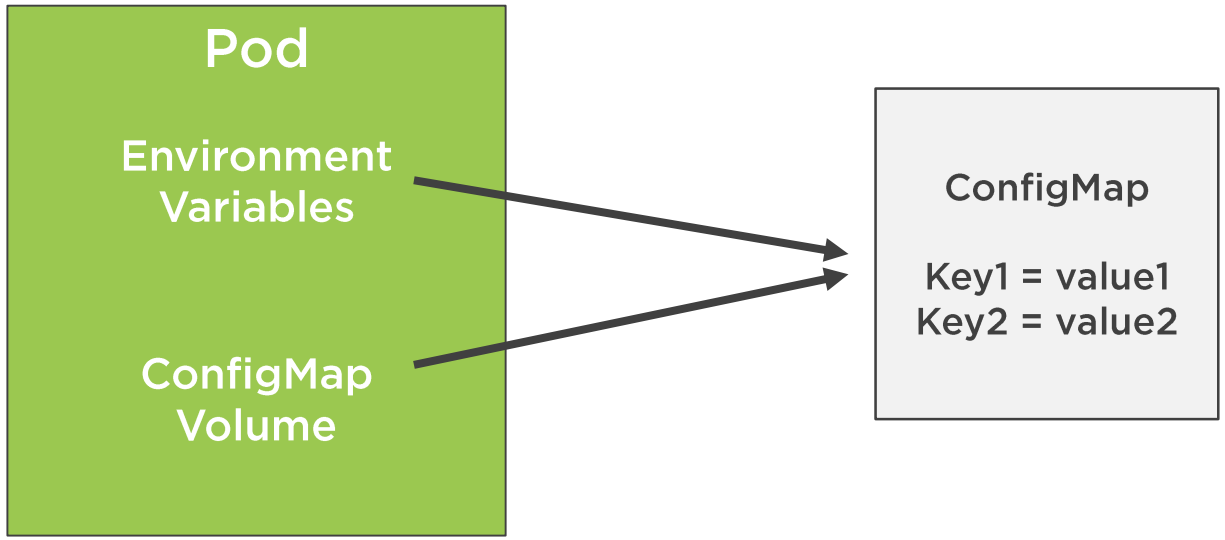
Can store entire files or provide key/value pairs:

- Store in a File. Key is the filename, value is the file contents (can be JSON, XML, keys/values, etc.).
- Provide on the command-line
- ConfigMap manifest

Accessing ConfigMap Data in a Pod

ConfigMaps can be accessed from a Pod using:

- **Environment variables (key/value)**
- **ConfigMap Volume (access as files)**



Creating a ConfigMap

Defining Values in a ConfigMap Manifest

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-settings
  labels:
    app: app-settings
data:
  enemies: aliens
  lives: "3"
  enemies.cheat: "true"
  enemies.cheat.level=noGoodRotten
```

```
# Create from a ConfigMap manifest
kubectl create -f file.configmap.yml
```

- A ConfigMap resource
- Name of ConfigMap
- ConfigMap data

Defining Key/Value Pairs in a File

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
```

```
# Create a ConfigMap using data from a file
kubectl create configmap [cm-name]
  --from-file=[path-to-file]
```

```
apiVersion: v1
kind: ConfigMap
data:
  game.config: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
```

- Key/value pairs defined in a file named game.config
- Nested properties can be defined and assigned a value

- Note that the file name is used as the key for the values
- Your application can now work with the content just as it would a normal configuration file (JSON, XML, keys/values, could be used)

Defining Key/Value Pairs in an Env File

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
```

```
# Create a env ConfigMap using data from a file
kubectl create configmap [cm-name]
  --from-env-file=[path-to-file]
```

```
apiVersion: v1
kind: ConfigMap
data:
  enemies=aliens
  lives=3
  enemies.cheat=true
  enemies.cheat.level=noGoodRotten
```

- Key/value pairs can be defined in an "environment" variables file (game-config.env)
- Nested properties can be defined and assigned a value
- Note that the file name is NOT included as a key

```
# Create a ConfigMap using data from a config file
kubectl create configmap [cm-name] --from-file=[path-to-file]

# Create ConfigMap from an env file
kubectl create configmap [cm-name] --from-env-file=[path-to-file]


# Create a ConfigMap from individual data values
kubectl create configmap [cm-name]
  --from-literal=apiUrl=https://my-api
  --from-literal=otherKey=otherValue

# Create from a ConfigMap manifest
kubectl create -f file.configmap.yml
```

Creating a ConfigMap

A ConfigMap can be created using **kubectl create**

Key command-line switches include:

- --from-file
- --from-env-file
- --from-literal

Using a ConfigMap

```
# Get a ConfigMap
kubectl get cm [cm-name] -o yaml
```

Getting a ConfigMap

kubectl get cm can be used to get a ConfigMap and view its contents

Accessing a ConfigMap: Environment Vars

Pods can access ConfigMap values through environment vars

ENEMIES environment variable created (value=aliens)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-settings
data:
  enemies: aliens
  lives: "3"
  enemies.cheat: "true"
  enemies.cheat.level=noGoodRotten
```

```
apiVersion: apps/v1
...
spec:
  template:
    ...
  spec:
    containers: ...
    env:
    - name: ENEMIES
      valueFrom:
        configMapKeyRef:
          key: app-settings
          name: enemies
```

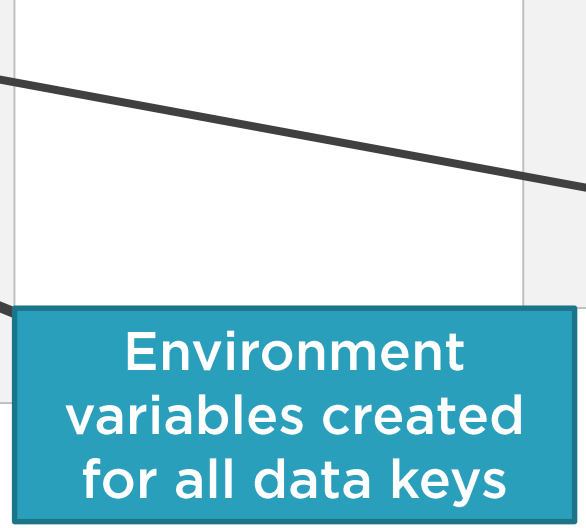
Environment variable name

Accessing a ConfigMap: Environment Vars

envFrom can be used to load all ConfigMap keys/values into environment variables

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-settings
data:
  enemies: aliens
  lives: "3"
  enemies.cheat: "true"
  enemies.cheat.level=noGoodRotten
```

```
apiVersion: apps/v1
...
spec:
  template:
    ...
  spec:
    containers: ...
      envFrom:
      - configMapRef:
        name: app-settings
```



Accessing a ConfigMap: Volume

ConfigMap values can be loaded through a Volume

Each key is converted to a file - value is added into the file

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-settings
data:
  enemies: aliens
  lives: "3"
  enemies.cheat: "true"
  enemies.cheat.level=noGoodRotten
```

```
apiVersion: apps/v1
...
spec:
  template:
    ...
  spec:
    volumes:
      - name: app-config-vol
        configMap:
          name: app-settings
    containers:
      volumeMounts:
        - name: app-config-vol
          mountPath: /etc/config
```

ConfigMaps in Action

Secrets Core Concepts

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key.

Secrets

Kubernetes can store sensitive information (passwords, keys, certificates, etc.)

Avoids storing secrets in container images, in files, or in deployment manifests

Mount secrets into pods as files or as environment variables

Kubernetes only makes secrets available to Nodes that have a Pod requesting the secret

Secrets are stored in tmpfs on a Node (not on disk)

Enable encryption at rest for cluster data (https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data)

Limit access to etcd (where Secrets are stored) to only admin users

Use SSL/TLS for etcd peer-to-peer communication

Manifest (YAML/JSON) files only base64 encode the Secret

Pods can access Secrets so secure which users can create Pods. Role-based access control (RBAC) can be used.

https://kubernetes.io/docs/concepts/configuration/secret/#best-practices

Secrets Best Practices

Creating a Secret

Creating a Secret

Secrets can be created using kubectl create secret

```
# Create a secret and store securely in Kubernetes
kubectl create secret generic my-secret
  --from-literal=pwd=my-password

# Create a secret from a file
kubectl create secret generic my-secret
  --from-file=ssh-privatekey=~/.ssh/id_rsa
  --from-file=ssh-publickey=~/.ssh/id_rsa.pub

# Create a secret from a key pair
kubectl create secret tls tls-secret --cert=path/to/tls.cert
  --key=path/to/tls.key
```

Question:

Can I declaratively define secrets using YAML?

Answer:

Yes – but any secret data is only base64 encoded in the manifest file!

Defining a Secret in YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: db-passwords
type: Opaque
data:
  app-password: cGFzc3dvcmQ=
  admin-password: dmVyeV9zZWNyZXQ=
```

- Define a Secret
- Secret name
- Keys/values for Secret

Using a Secret

```
# Get secrets
kubectl get secrets
```

```
danwahlin — -bash — 73×6
iMac-3:~ danwahlin$ k get secrets
+ kubectl get secrets
NAME                  TYPE                                  DATA   AGE
db-passwords          Opaque                                2      34m
default-token-rxmjb   kubernetes.io/service-account-token   3      66d
```

```
# Get YAML for specific secret
kubectl get secrets db-passwords -o yaml
```

```
danwahlin — -bash — 73×16
iMac-3:~ danwahlin$ k get secrets db-passwords -o yaml
+ kubectl get secrets db-passwords -o yaml
apiVersion: v1
data:
  mongodb-password: cGFzc3dvcmQ=
  mongodb-root-password: cGFzc3dvcmQ=
kind: Secret
metadata:
  creationTimestamp: "2019-03-22T00:40:05Z"
  name: db-passwords
  namespace: default
  resourceVersion: "3481795"
  selfLink: /api/v1/namespaces/default/secrets/db-passwords
  uid: 0982413e-4c3b-11e9-b7f0-025000000001
type: Opaque
```

Listing Secret Keys

A list of secrets can be retrieved using kubectl get secrets

Accessing a Secret: Environment Vars

Pods can access Secret values through environment vars

DATABASE_PASSWORD environment var created

```
apiVersion: v1
kind: Secret
metadata:
  name: db-passwords
type: Opaque
data:
  db-password: cGFzc3dvcmQ=
  admin-password: dmVyeV9zZWNyZXQ=
```

```
apiVersion: apps/v1
...
spec:
  template:
    ...
  spec:
    containers: ...
    env:
    - name: DATABASE_PASSWORD
      valueFrom:
        secretKeyRef:
          name: db-passwords
          key: db-password
```

Accessing a Secret: Volumes

Pods can access secret values through a volume

Each key is converted to a file - value is added into the file

```
apiVersion: v1
kind: Secret
metadata:
  name: db-passwords
type: Opaque
data:
  db-password: cGFzc3dvcmQ=
  admin-password: dmVyeV9zZWNyZXQ=
```

```
apiVersion: apps/v1
...
spec:
  template:
    ...
  spec:
    volumes:
      - name: secrets
        secret:
          secretName: db-passwords
        containers:
        volumeMounts:
          - name: secrets
            mountPath: /etc/db-passwords
            readOnly: true
```

Secrets in Action

Summary

ConfigMaps provide a way to store configuration data

Secrets provide a way to store sensitive data or files

Access key/value pairs using environment variables or volumes

Use caution when working with Secrets and ensure proper security is in place