

Module 5: Manage Storage for Application Configuration and Data



Lesson 1: Externalize the Configuration of Applications

Configure applications by using Kubernetes secrets and configuration maps to initialize environment variables and to provide text and binary configuration files.

Configuring Kubernetes Applications

- Prebuilt image uses default configuration
- Customization needed
 - name of application
 - labels
 - image source
 - storage configuration
 - environment variables
- Format
 - YAML
 - JSON

The manifest file in YAML format

```
apiVersion: apps/v1
kind: Deployment 2
metadata: 3
 name: hello-deployment
spec: 4
  replicas: 1
  selector:
    matchLabels:
      app: hello-deployment
  template:
    metadata:
      labels:
        app: hello-deployment
   spec: 5
      containers:
      - env: 6
        - name: ENV VARIABLE 1
         valueFrom:
            secretKeyRef:
              key: hello
              name: world
        image: quay.io/hello-image:latest
```

- 1 API version of the resource.
- 2 Deployment resource type.
- 3 In this section, you specify the metadata of your application, such as the name.
- 4 You can define the general configuration of the resource that is applied to the deployment, such as the number of replicas (pods), the selector label, and the template data.
- 5 In this section, you specify the configuration for your application, such as the image name, the container name, ports, environment variables, and more.
- 6 You can define the environment variables to configure your application needs.

Configuration Maps

- API object
- Store configuration
- Accept key-value pairs
- Keep configuration files, environment variables or arguments
- Decouple configuration from image
- Keep containerized applications portable
- Alternative: Secret resource type

Configuration Map (cm) manifest in YAML

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-configmap
  namespace: my-app
data: 2
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
binaryData: 0
  bar: L3Jvb3QvMTAw
```

1. ConfigMap resource type.

2. Contains the configuration data.

3. Points to an encoded file in base64 that contains non-UTF-8 data, for example, a binary Java keystore file. Place a key followed by the encoded file.

Creating/Deleting Configuration Maps

Creating CMs

```
[user@host ~]$ kubectl create configmap my-config \
--from-literal key1=config1 --from-literal key2=config2
```

You can also use the cm shortname to create a configuration map.

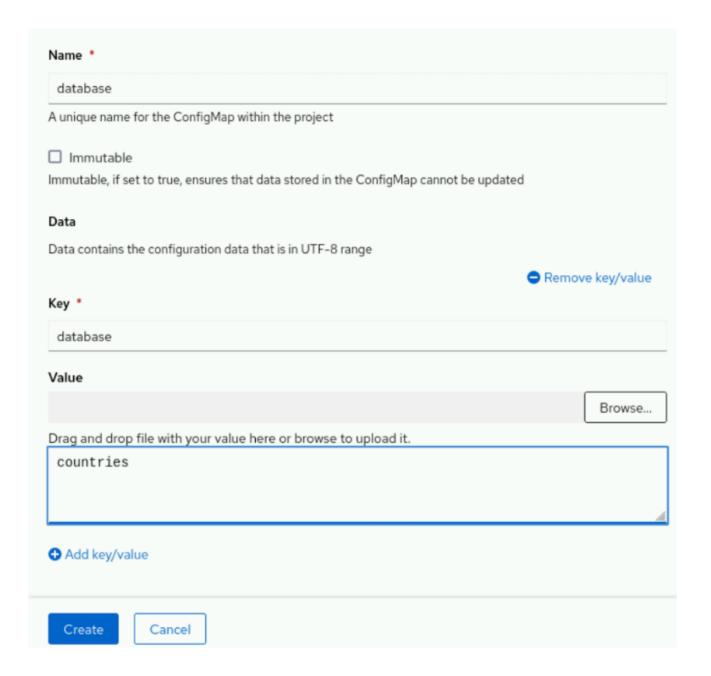
```
[user@host ~]$ oc create cm my-config \
--from-literal key1=config1 --from-literal key2=config2
```

Deleting CMs

```
[user@host ~]$ oc delete configmap/demo-map -n demo
```

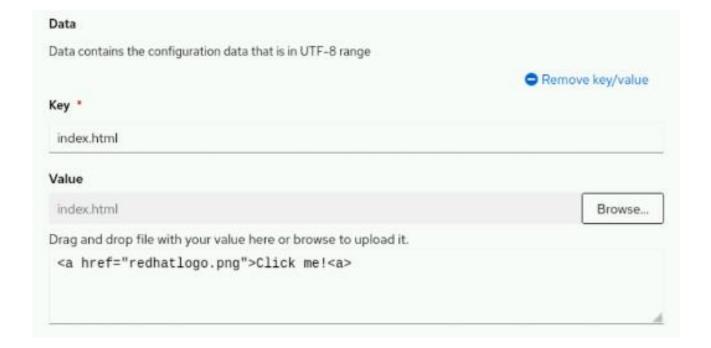
Creating Secrets using web console

- From web console, click Workloads ->
 ConfigMaps menu
- Then click Create ConfigMap



Creating Secrets using web console

- Click Browse -> Value field
- Key field : Specify any meaningful name
- Value field : Browse to the file



Secrets

- API object
- Sensitive information stored in Base64-encoded format
- Case use:
 - Username and Passwords
 - Sensitive configuration files
 - Credentials to external source (SSH key or Oauth token)
 - TLS certificates
 - Docker configuration secrets (credential)
- To encrypt secrets or configmaps
 - encrypt the Etcd database

Secrets manifest in YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: example-secret
  namespace: my-app
type: Opaque 💶
data: 2
  username: bXl1c2VyCg==
  password: bX1QQDU1Cg==
stringData:
  hostname: myapp.mydomain.com
  secret.properties:
    property1=valueA
    property2=valueB
```

1. Specifies the type of secret.

2. Specifies the encoded string and data.

3. Specifies the decoded string and data.

```
Decoding secret

[user@host] echo bX11c2VyCg== | base64 --decode
myuser
[user@host] echo bX1QQDU1Cg== | base64 --decode
myP@55
```

Creating Secrets

Create a generic secret that contains key-value pairs from literal values that are typed on the command line:

```
[user@host ~]$ oc create secret generic secret_name \
--from-literal key1=secret1 \
--from-literal key2=secret2
```

Create a generic secret by using key names that are specified on the command line and values from files:

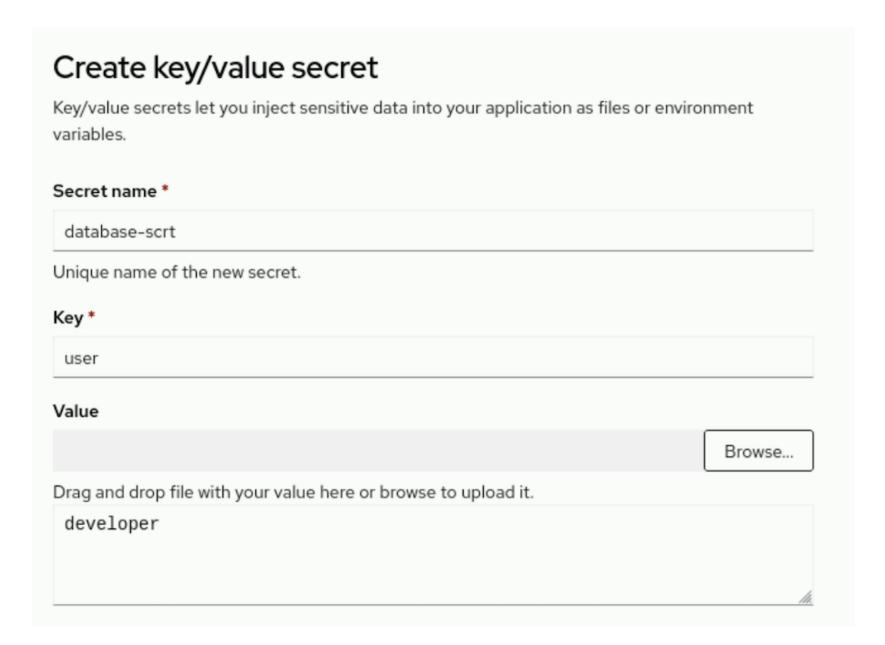
```
[user@host ~]$ kubectl create secret generic ssh-keys \
--from-file id_rsa=/path-to/id_rsa \
--from-file id_rsa.pub=/path-to/id_rsa.pub
```

Create a TLS secret that specifies a certificate and the associated key:

```
[user@host ~]$ oc create secret tls secret-tls \
--cert /path-to-certificate --key /path-to-key
```

Creating Secrets using web console

- From web console, click
 Workloads -> Secrets menu
- Then click Create → select
 Key/value secret



Creating Secrets to external source's credential

- From web console, click Workloads ->
 Secrets menu
- Then click Create → select Image pull secret



Use ConfigMaps to initialize environment variable

- Use environment variable to configure application
- Example:
 - Set database name
 - Customized user name

```
apiVersion: v1
kind: ConfigMap
metadata:
   name: config-map-example
   namespace: example-app

data:
   database.name: sakila 2
   database.user: redhat 3
```

- 1. The project where the configuration map resides. ConfigMap objects can be referenced only by pods in the same project.
- 2. Initializes the database.name variable to the sakila value.
- 3. Initializes the database.user variable to the redhat value.

Inject ConfigMaps to application

```
apiVersion: v1
kind: Pod
metadata:
 name: config-map-example-pod
 namespace: example-app
spec:
  containers:
    - name: example-container
     image: registry.example.com/mysql-80:1-237
     command: [ "/bin/sh", "-c", "env" ]
       - name: MYSQL DATABASE 2
          valueFrom:
            configMapKeyRef:
             name: config-map-example 3
              key: database.name 4
        - name: MYSQL_USER
          valueFrom:
            configMapKeyRef:
              name: config-map-example 5
              key: database.user 6
              optional: true
```

- 1 he attribute to specify environment variables for the pod.
- 2 The name of a pod environment variable where you are populating a key's value.
- 3 5 Name of the ConfigMap object to pull the environment variables from.
- 4 6 The environment variable to pull from the ConfigMap object.
- 7 Sets the environment variable as optional. The pod is started even if the specified ConfigMap object and keys do not exist.

Inject ConfigMaps to application

```
apiVersion: v1
kind: Pod
metadata:
  name: config-map-example-pod2
  namespace: example-app
spec:
  containers:

    name: example-container

      image: registry.example.com/mysql-80:1-237
      command: [ "/bin/sh", "-c", "env" ]
  envFrom:
    configMapRef:
        name: config-map-example 2
  restartPolicy: Never
```

- The attribute to pull all environment variables from a ConfigMap object.
- 2. The name of the ConfigMap object to pull environment variables from.

Inject Secrets to application

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "export" ]
     env: 0
        - name: TEST_SECRET_USERNAME_ENV_VAR
         valueFrom: 2
            secretKeyRef: 3
              name: test-secret
              key: username 5
```

- 1. Specifies the environment variables for the pod.
- 2. Indicates the source of the environment variables.
- 3. The secretKeyRef source object of the environment variables.
- 4. Name of the secret, which must exist.
- 5. The key that is extracted from the secret is the username for authentication.

Inject ConfigMap as volume to deployment

```
[user@host ~]$ oc create configmap demo-map \
--from-file=config-files/httpd.conf
```

You can similarly add a configuration map as a volume by using the following command:

```
[user@host ~]$ oc set volume deployment/demo \
--add --type configmap \
--configmap-name demo-map \
--mount-path /app-secrets
```

To confirm that the volume is attached to the deployment, use the following command:

```
[user@host ~]$ oc set volume deployment/demo
demo
configMap/demo-map as volume-du9in
mounted at /app-secrets
```

Inject Secrets as volume to deployment

```
[user@host ~]$ oc create secret generic demo-secret \
--from-literal user=demo-user \
--from-literal root_password=zT1KTgk
```

You can also create a generic secret by specifying key names on the command line and values from files:

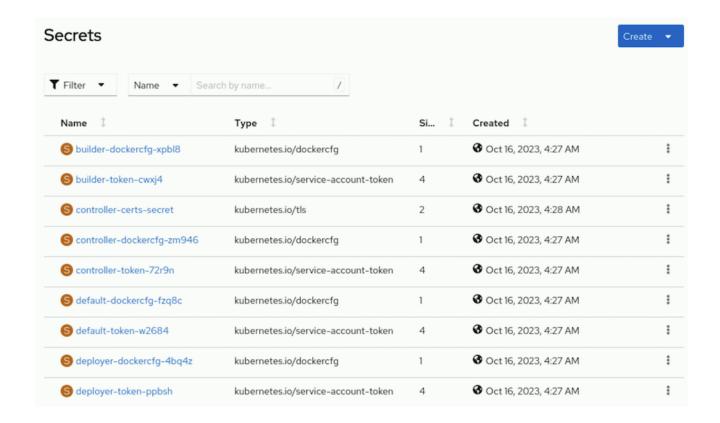
```
[user@host ~]$ oc create secret generic demo-secret \
--from-file user=/tmp/demo/user \
--from-file root_password=/tmp/demo/root_password
```

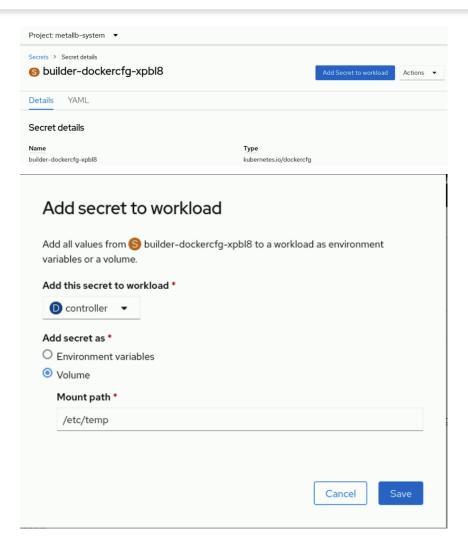
You can mount a secret to a directory within a pod. Kubernetes creates a file for each key in the secret that uses the name of the key. The content of each file is the decoded value of the secret. The following command shows how to mount secrets in a pod:

```
[user@host ~]$ oc set volume deployment/demo \ 1
--add --type secret \ 2
--secret-name demo-secret \ 3
--mount-path /app-secrets 4
```

Inject secret as volume to deployment using web console

- From web console, click Workloads -> Secrets menu
- Select a secret -> Add Secret to Workload





The oc set env Command

- Update secret / configmap
- Use --prefix option to prefix parameter
- Example
- 1. Create secret containing a user name and root password

```
[user@host ~]$ oc create secret generic demo-secret \
--from-literal user=demo-user \
--from-literal root_password=zT1KTgk
```

2. Inject the secret into deployment with the -prefix option

```
[user@host ~]$ oc set env deployment/demo \
--from secret/demo-secret --prefix MYSQL_
```

- 3. Resulting in
 - MYSQL_user=demo-user
 - MYSQL_root_password=zT1KTgk

Updating Secrets and Configuration Maps

- Use oc extract secret to retrieve secrets into local system directory
- Use --confirm option to overwrite if file exists in the destination directory
- Example
- 1. Extracct the secrets into /tmp/demo

```
[user@host ~]$ oc extract secret/demo-secrets -n demo \
   --to /tmp/demo --confirm
[user@host ~]$ ls /tmp/demo/
user root_password
[user@host ~]$ cat /tmp/demo/root_password
zT1KTgk
[user@host ~]$ echo k8qhcw3m0 > /tmp/demo/root_password
```

2. Update the secret from another credential file

```
[user@host ~]$ oc set data secret/demo-secrets -n demo \
--from-file /tmp/demo/root_password
```

3. Restart pods to re-read the updated secrets

NOTE: If using volume, kubelet agent detect and propagate changes using eventually consistent approach

Deleting Secrets and Configuration Maps

• First remove secrets or config maps from deployment or pods

```
[user@host ~]$ kubectl delete secret/demo-secrets -n demo

[user@host ~]$ oc delete configmap/demo-map -n demo
```

Guided Exercise: Externalize the Configuration of Applications

You should be able to:

- Create a web application deployment.
- Expose the web application deployment to external access.
- Create a configuration map from two files.
- Mount the configuration map in the web application deployment.

You should be able to:

Lab:
Deploy Managed and
Networked
Applications on
Kubernetes

- Deploy a MySQL database from a container image.
- Deploy a web application from a container image.
- Configure environment variables for a deployment.
- Expose the web application for external access.