K8S (8) ConfigMaps and Secrets

When it needs to pass runtime parameters to different container image, we can reply on ConfigMap API.
If the parameters are sensitive info, we can user Secret API.

ConfigMaps:

Pass configuration data as key-value pair. The pod can use them as environment variable, command, or volumes

Couple of different ways to create ConfigMaps

(1) Create a ConfigMap from Literal Values

```
% kubectl create configmap my-config --from-literal=key1=value1 --from-
literal=key2=value2
configmap/my-config created

$ kubectl get configmaps my-config -o yaml
apiVersion: v1
data:
key1: value1
key2: value2
kind: ConfigMap
metadata:
creationTimestamp: "2023-12-25T21:37:28Z"
name: my-config
namespace: default
resourceVersion: "47179"
uid: 876b1e34-b876-47ba-a3ff-0d0edd0d9cda
```

(2) Create a ConfigMap from Definition Manifest

```
# customer1-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: customer1
data:
  TEXT1: Customer1_Company
  TEXT2: Welcomes You
  COMPANY: Customer1 Company Technology Pct. Ltd.
```

```
$ kubectl apply -f customer1-configmap.yaml
configmap/customer1 created

$ kubectl get configmaps customer1 -o yaml
apiVersion: v1
data:
COMPANY: Customer1 Company Technology Pct. Ltd.
TEXT1: Customer1_Company
TEXT2: Welcomes You
kind: ConfigMap
metadata:
annotations:
kubectl.kubernetes.io/last-applied-configuration: |
{"apiVersion":"v1","data":{"COMPANY":"Customer1 Company Technology Pct.
Ltd.","TEXT1":"Customer1_Company","TEXT2":"Welcomes
You"},"kind":"ConfigMap","metadata":{"annotations":
{},"name":"customer1","namespace":"default"}}
creationTimestamp: "2023-12-25T21:54:16Z"
name: customer1
namespace: default
resourceVersion: "47982"
uid: ea734e43-c036-4bed-9e50-b55f48c0ef98
```

(3) Create a ConfigMap from a File

```
# permission-reset.properties
permission=read-only
allowed="true"
resetCount=3
```

```
$ kubectl create configmap permission-config --from-file=permission-reset.properties
configmap/permission-config created

$ kubectl get configmaps permission-config -o yaml
apiVersion: v1
data:
permission-reset.properties: |
permission=read-only
allowed="true"
resetCount=3
kind: ConfigMap
metadata:
creationTimestamp: "2023-12-25T21:59:10Z"
name: permission-config
namespace: default
resourceVersion: "48217"
uid: 88fb43ce-865a-4c2d-ac14-f0b4e3cf797e
```

Couple of different ways to use ConfigMaps inside a pod:

(1) Environment variable

In the following example, every key-value pair we put in **full-config-map** will be environment variable in container **myapp-full-container**.

```
...
containers:
- name: myapp-full-container
  image: myapp
  envFrom:
  - configMapRef:
  name: full-config-map
```

In the following example, the key-value in **config-map-1** and **config-map-2** will become environment variable in **myapp-specific-container**.

The **SPECIFIC_ENV_VAR1** is set to the value of **SPECIFIC_DATA**. (In **config-map-1**, something like **SPECIFIC_DATA**: 123)
The **SPECIFIC_ENV_VAR2** is set to the value of **SPECIFIC_INFO**. (In **config-map-2**, something like **SPECIFIC_INFO**: 456)
For **myapp-specific-container**, it will become **SPECIFIC_ENV_VAR1**: 123 and **SPECIFIC_ENV_VAR2**: 456.

```
...
containers:
- name: myapp-specific-container
  image: myapp
  env:
  - name: SPECIFIC_ENV_VAR1
    valueFrom:
      configMapKeyRef:
        name: config-map-1
        key: SPECIFIC_DATA
  - name: SPECIFIC_ENV_VAR2
    valueFrom:
      configMapKeyRef:
        name: config-map-2
        key: SPECIFIC_INFO
```

(2) Use configMap as Volume
We can mount a ConfigMap object in Pod and pod can read it.
In the following example, we mount the **vol-config-map** ConfigMap as a Volume inside a Pod. For each key in the ConfigMap, a file gets created in the mount path (/etc/config). Each key in the ConfigMap becomes a file (the file is the key's name) and the content of the file is the key's value.

```
...
containers:
- name: myapp-vol-container
  image: myapp
  volumeMounts:
  - name: config-volume
    mountPath: /etc/config
volumes:
- name: config-volume
  configMap:
    name: vol-config-map
```

Secrets:

(1) Create Secrets by Literal Values

```
$ kubectl create secret generic my-secret --from-literal=password=hello1234
secret/my-secret created

$ kubectl get secrets
NAME TYPE DATA AGE
my-secret Opaque 1 5s

$ kubectl get secret my-secret
NAME TYPE DATA AGE
my-secret Opaque 1 25s

$ kubectl describe secret my-secret
Name: my-secret
Namespace: default
Labels: <none>
Annotations: <none>

Type: Opaque

Data
====
password: 9 bytes

$ kubectl get secret my-secret -o yaml // your can retrieve the data in secret in
base64 encoded
```

```
apiVersion: v1
data:
password: aGVsbG8xMjM0
kind: Secret
metadata:
creationTimestamp: "2023-12-26T18:18:11Z"
name: my-secret
namespace: default
resourceVersion: "73420"
uid: 1e23e426-6ff0-4ce1-ac91-c2c223c68bb6
type: Opaque

$ echo 'aGVsbG8xMjM0' | base64 -d // encode the data, notice the last '%' is the
newline symbol generated by echo
hello1234% chenyang@ChenYangs-MBP
```

(2) Create Secrets by Definition Manifest

There are two types of mapping: data and stringData. You need to encode the data in base64 for data, on the other hand, just use plain text in stringData

```
# mypass-data.yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-password-data
type: Opaque
data:
  password: bXlzcWxwYXNzd29yZAo= // $echo mysqlpassword | base64
```

```
$ echo mysqlpassword | base64
bXlzcWxwYXNzd29yZAo=

$ kubectl create -f mypass-data.yaml
secret/my-password-data created

$ kubectl get secrets
NAME TYPE DATA AGE
my-password-data Opaque 1 8s
my-secret Opaque 1 9m48s

$ kubectl get secrets my-password-data -o yaml
```

```
apiVersion: v1
data:
password: bXlzcWxwYXNzd29yZAo=
kind: Secret
metadata:
creationTimestamp: "2023-12-26T18:27:51Z"
name: my-password-data
namespace: default
resourceVersion: "73711"
uid: f64a86b2-59f7-4bdd-88b4-dcc58d919f4c
type: Opaque

$ echo "bXlzcWxwYXNzd29yZAo=" | base64 -d
mysqlpassword
```

```
# mypass-stringData.yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-password-stringdata
type: Opaque
stringData:
  password: mysqlpassword
```

```
$ kubectl apply -f mypass-stringData.yaml
secret/my-password-stringdata created

$ kubectl get secrets
NAME TYPE DATA AGE
my-password-data Opaque 1 2m15s
my-password-stringdata Opaque 1 6s
my-secret Opaque 1 11m

$ kubectl get secret my-password-stringdata -o yaml
apiVersion: v1
data:
password: bXlzcWxwYXNzd29yZA==
kind: Secret
metadata:
annotations:
```

```
kubectl.kubernetes.io/last-applied-configuration: |
{"apiVersion":"v1","kind":"Secret","metadata":{"annotations":{},"name":"my-password-
stringdata","namespace":"default"},"stringData":
{"password":"mysqlpassword"},"type":"Opaque"}
creationTimestamp: "2023-12-26T18:30:00Z"
name: my-password-stringdata
namespace: default
resourceVersion: "73811"
uid: 14174fcf-33ee-4acb-8723-5aa61f85e1a1
type: Opaque

$ echo "bXlzcWxwYXNzd29yZA==" | base64 -d
mysqlpassword%
```

(3) Create Secrets from file

```
$ echo mysqlpassword | base64
bXlzcWxwYXNzd29yZAo=

$ echo -n 'bXlzcWxwYXNzd29yZAo=' > password.txt

$ kubectl create secret generic my-file-password --from-file=password.txt
secret/my-file-password created

$ kubectl get secrets
NAME TYPE DATA AGE
my-file-password Opaque 1 7s
my-password-data Opaque 1 4m47s
my-password-stringdata Opaque 1 2m38s
my-secret Opaque 1 14m

$ kubectl get secret my-file-password -o yaml
apiVersion: v1
data:
password.txt: YlhsemNXeHdZWE56ZDI5eVpBbz0=
kind: Secret
metadata:
creationTimestamp: "2023-12-26T18:32:31Z"
name: my-file-password
namespace: default
resourceVersion: "73874"
uid: 75cc3090-1355-4f79-8f10-ad176463c7ba
type: Opaque

$ echo "YlhsemNXeHdZWE56ZDI5eVpBbz0=" | base64 -d
bXlzcWxwYXNzd29yZAo=% chenyang@ChenYangs-MBP

$ echo "bXlzcWxwYXNzd29yZAo=" | base64 -d
```

```
mysqlpassword
```

Couple of different ways to use Secret inside a pod:

(1) Environment variable

Similar with using ConfigMap, using **secretKeyRef** instead of **configMapKeyRef**. Following example will set environment variable **WORDPRESS_DB_PASSWORD** to the value of **password** key in **my-password** secret.

```
....
spec:
 containers:
 - image: wordpress:4.7.3-apache
   name: wordpress
   env:
   - name: WORDPRESS_DB_PASSWORD
     valueFrom:
       secretKeyRef:
         name: my-password
         key: password
....
```

(2) As volume

Similar with using ConfigMap, mount **my-password** Secret as a volume inside a pod. For each key in the **my-password** Secret, a file created in the mount path (/etc/secret-data).

Each key in the Secret becomes a file (the file is the key's name) and the content of the file is the key's value.

```
....
spec:
  containers:
  - image: wordpress:4.7.3-apache
    name: wordpress
    volumeMounts:
    - name: secret-volume
      mountPath: "/etc/secret-data"
      readOnly: true
  volumes:
  - name: secret-volume
    secret: // This is the difference#1 with ConfigMap, CM uses configMap here.
      secretName: my-password // This is the difference#2, CM uses name here.
....
```

Demo of Using ConfigMaps as Volumes:

This is the simple welcome page, we want this to become the index.html for our nginx server

```
# green/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to GREEN App!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
    background-color: GREEN;
  }
</style>
</head>
<body>
<h1 style=\"text-align: center;\">Welcome to GREEN App!</h1>
</body>
</html>
```

Create a ConfigMap green-web-cm, the data is the content of the html file

```
$ cat green/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to GREEN App!</title>
<style>
body {
width: 35em;
margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif;
background-color: GREEN;
}
</style>
</head>
<body>
<h1 style=\"text-align: center;\">Welcome to GREEN App!</h1>
</body>
</html>
```

```
$ kubectl create configmap green-web-cm --from-file=green/index.html
configmap/green-web-cm created

$ kubectl get cm
NAME DATA AGE
customer1 3 10h
green-web-cm 1 21s
kube-root-ca.crt 1 47h
my-config 2 10h
permission-config 1 10h

$ kubectl describe cm green-web-cm
Name: green-web-cm
Namespace: default
Labels: <none>
Annotations: <none>

Data
====
index.html:
----
<!DOCTYPE html>
<html>
<head>
<title>Welcome to GREEN App!</title>
<style>
body {
width: 35em;
margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif;
background-color: GREEN;
}
</style>
</head>
<body>
<h1 style=\"text-align: center;\">Welcome to GREEN App!</h1>
</body>
</html>


BinaryData
====

Events: <none>
```

Check the deployment for green-web. Pretty standard deployment, providing deployment name green-web, associated with pod label app: green-web.

In the pod template, we define a volume web-config, the volume is using green-web-cm ConfigMap (which we just created in previous step).
In the container definition, we indicate this volume web-config will be mounted in /usr/share/nginx/html (which is the home directory of nginx index file).
So basically we are replace the default nginx home page with what we have in ConfigMap.

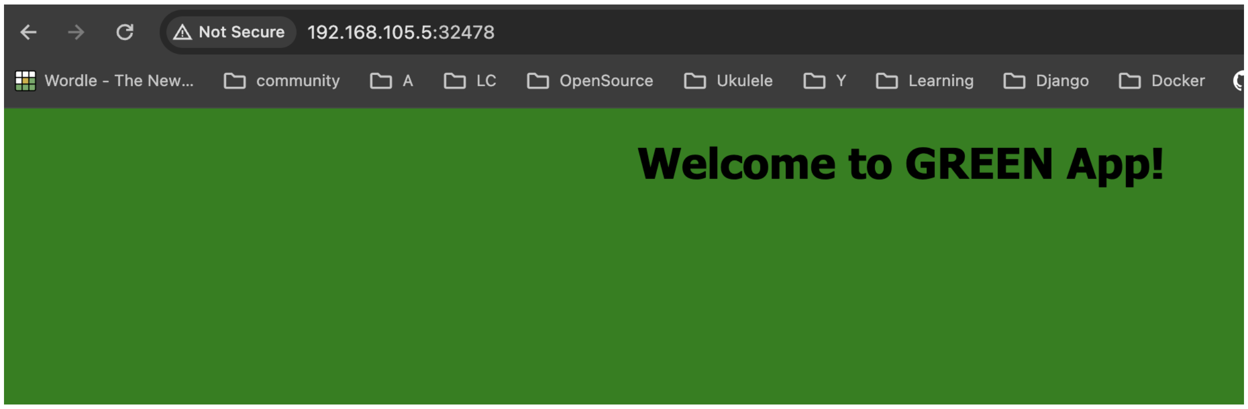
```yaml
# web-green-with-cm.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: green-web
  name: green-web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: green-web
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: green-web
    spec:
      volumes:
      - name: web-config
        configMap:
          name: green-web-cm
      containers:
      - image: nginx
        name: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: web-config
status: {}
```

Create this deployment and check the service url.

```
$ kubectl apply -f web-green-with-cm.yaml
deployment.apps/green-web created

$ kubectl expose deployment green-web --type=NodePort
service/green-web exposed

$ minikube service list
|----------------------|---------------------------|--------------|-----------------
-----------|
| NAMESPACE | NAME | TARGET PORT | URL |
|----------------------|---------------------------|--------------|-----------------
-----------|
| default | blue-app | 80 | http://192.168.105.5:31956 |
| default | green-web | 80 | http://192.168.105.5:32478 |
|----------------------|---------------------------|--------------|-----------------
-----------|
```



Welcome to GREEN App!