NOTE:

Configuring ConfigMaps in Applications

Topic to be discussed is configuration data in K8s

Last topic we discussed about how to define environment variables in a pod definition file

when we have alot of pod-definition-files, it will become difficult to manage the environment data stored within the query's file

pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
spec:
  containers:
  - name : simple-webapp-color
    image: simple-webapp-color
    ports:
      - containerPort : 8080
    env:
      - name : APP_COLOR
        value: blue
      - name : APP_MODE
        value: prod
```

we can take this information out of the pod definition file & manage it centrally using "configuration maps"

ConfigMap

⇒ Config maps are used to pass configuration data in the form of key value pairs in k8s

ConfigMap

APP_COLOR : blue

APP_MODE : prod

when a pod is created, inject the config map into the pod, so the key value pairs are available as environment variable for the application hosted inside the container in the pod

0:55

pod-definition.yaml

```
apiVersion : v1
kind: Pod
metadata:
   name: Sample-webapp-color
spec:
   containers:
   -name: Simple-webapp-color
    image: Simple-webapp-color
    ports:
      - containerPort: 8080

    envFrom:
    - configMapRef:
         name: app-color
```

There're 2 phases involved in configuring configg maps

1. Create ConfigMap
2. Inject into pod

Just like any other k8s object, there are 2 ways of creating config map

Imperative way

without using a config map definition file

Kubectl create configmap

Declarative way:-

By using config map definition file

Kubectl create -f

If you don't wish to create a config map definition file, we can simply use the Imperative way:- "Kubectl create configmap" command & specify the required arguments

=> With this method we can directly specify the key, value pairs in the command line

To create the configmap of the given values,

Syntax

Kubectl create configmap

<config-name> --from-literal = <key> = <value>

the --from-literal option is used to specify the key value pairs in the command itself

In the Eg below,
We are creating a config map by the name "app config" with the key value pair of app color equals blue

⇒ If you wish to have additional key value pairs simply specify the `--from-literal` option multiple times

```
kubectl create configmap \
    app-config --from-literal = APP_COLOR = blue \
               --from-literal = APP_MOD = prod
```

If we mention the "--from-literal" multiple times this will get complicated when you have too many configuration items

Another way to input configuration data is through a file

```
kubectl create configmap
    <Config-name> --from-file = <path-to-file>
```

⇒ Use the from file option to specify a path to the file that contains the required data

⇒ The data from this file is read & stored under the "name of the file"

```
kubectl create configmap
    app-config --from-file = app_config.properties
```

2:39
Declarative

For this, we created a definition file, just like how we did for the pod.

The file has

*apiVersion *kind. & instead of spec we have metadata

config-map.yaml

```
apiVersion : v1
kind : ConfigMap
metadata :
  name : app-config
data :
  APP_COLOR : blue
  APP_MODE : prod
```

ConfigMap
APP_COLOR: blue
APP_MODE: prod

⇒ Under "data" add the "configuration data" in a "key: value" format.

Now run

kubectl create -f config-map.yaml

We can create as many config maps as we need in the same way for various different purpose

```
mysql-config
port : 3306
max_allowed_packet : 128M
```

```
redis-config
port : 6379
rdb-Compression : yes
```

⇒ It is very important to name the config maps appropriately [mysql-config, redis-config] as we will be using these names later while associating it with pods

To view the config maps

kubectl get configmaps command

```
kubectl get configmaps
NAME          DATA    AGE
app-config    2       35
```

This lists the newly created configmap named "app-config"

kubectl describe configmaps

The describe command lists the configuration data as well under the data section

Step 1:-

Create Configmap

Step 2:-

Inject into the pod

pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
  labels:
    name: simple-webapp-color
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
      - containerPort: 8080
    envFrom:
      - configMapRef:
          name: app-config
```

config-map.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_COLOR: blue
  APP_MODE: prod
```

kubectl create -f pod-definition.yaml

Creating the pod definition file now creates a web application with a blue background.

what we just saw was using "Config maps" to inject environment variables.

Other ways to inject Configuration data into pods

* Inject it as a Single environment Variable
* Inject the whole data as files in a Volume.