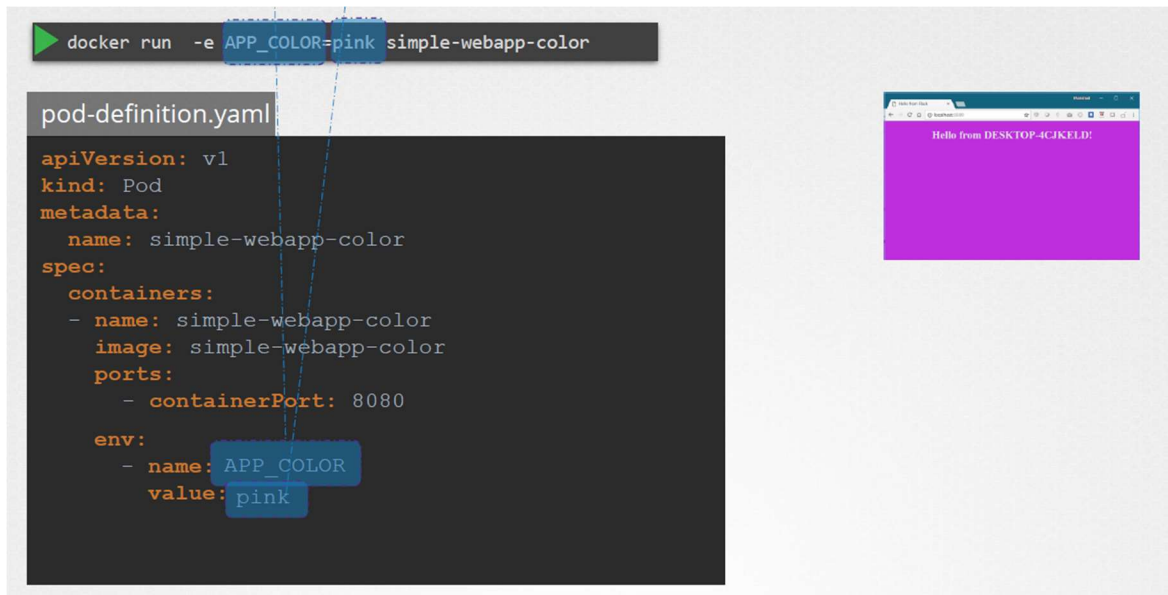
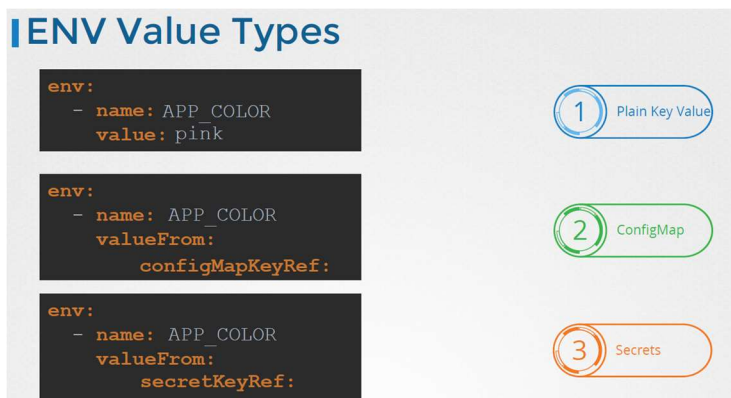


Configure Environment Variables in Applications

In this lecture we will see how to set an environment variable in Kubernetes. Given a pod definition file, which uses the same image as the docker command we ran in the last lecture. To set an environment variable, use the ENV property. ENV is array, so every item under the ENV property starts with a dash indicating an item in the array. Each item has a name and a value property. The name is a name of the environment variable made available with the container, and the value is its value.



What we just saw was a direct way of specifying the environment variables using your plain key-value pair format. However, there are other ways of setting the environment variables, such as using config maps and secrets. The difference in this case is that instead of specifying a value, we say "valueFrom" and then a specification of a config map or secret. We will discuss config maps and secret keys in the upcoming lectures



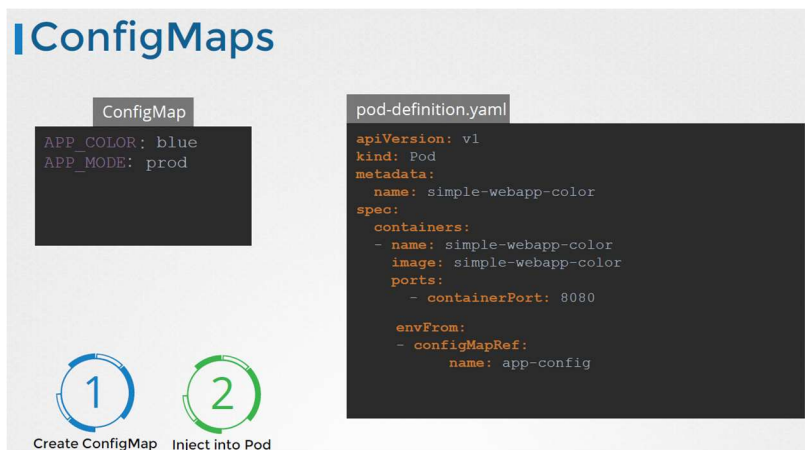
Configuring ConfigMaps in Applications

In this lecture, we will discuss how to work with configuration data in Kubernetes. In the previous lecture, we saw how to define environment variables in a pod definition 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
```

When you have a lot of pod definition files, it will become difficult to manage the environment data stored within the files.

We can take this information out of the pod definition file and manage it centrally using configuration maps. Config maps are used to pass configuration data in the form of key-value pairs in Kubernetes. When a pod is created, inject the config map into the pod so the key-value pairs are available as environment variables for the application hosted inside the container in the pod.



There are two phases involved in configuring config maps. First, **create the config map**, and second **inject them into the pod**.

Just like any other Kubernetes object, there are two ways of creating a config map. The imperative way without using a config map definition file, and the declarative way, by using a config map definition file.

If you do not wish to create a config map definition file, you could simply use the `kubectl create config map` command and specify the required arguments. Let's take a look at that first.

With this method, you can directly specify the key-value pairs on the command line. To create a config map with the given values, run the `kubectl create config map` command. The command is followed by the config name and the option **--from-literal**. The **--from-literal** option is used to specify the key-value pairs in the command itself.

In this example, we are creating a config map by the name app-config with a key-value pair of **app-color=blue**. If you wish to add additional key-value pairs, simply specify the `--from-literal` option multiple times. However, this will get complicated when you have too many configuration items.

Another way to input configuration data is through a 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 and stored under the name of the file.

Create ConfigMaps

ConfigMap

```
APP_COLOR: blue
APP_MODE: prod
```

Imperative

```
kubectl create configmap
<config-name> --from-literal=<key>=<value>

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

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

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

1

Create ConfigMap

Let us now look at the declarative approach. For this, we create a definition file, just like how we did for the pod. The file has an `apiVersion`, `kind`, `metadata`, and instead of `spec`, here we have `data`. The `apiVersion` is `V1`, the `kind` is `config map`. Under `metadata`, we specify a name for the config map. We will call it `app-config`. Under `data`, we define the configuration data in a key-value format.

Run the `kubectl create` command and specify the configuration file name. So, that creates the `app-config` config map with the values we specify.

Create ConfigMaps

ConfigMap

```
APP_COLOR: blue
APP_MODE: prod
```

Declarative

```
kubectl create -f

config-map.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_COLOR: blue
  APP_MODE: prod

kubectl create -f config-map.yaml
```

1

Create ConfigMap

You can create as many config maps as you need in the same way for various different purposes. Here I have one for my application, another for my SQL, and another one for Redis. So, it is important to name the config maps appropriately, as you will be using these names later while associating them with pods.

app-config	mysql-config	redis-config
APP_COLOR: blue APP_MODE: prod	port: 3306 max_allowed_packet: 128M	port: 6379 rdb-compression: yes

To view config maps, run the `kubectl get configmaps` command. This lists the newly created config map named `app-config`. The `kubectl describe configmaps` command lists the configuration data as well under the data section.

```
kubectl describe configmaps
```

```
Name:      app-config
Namespace: default
Labels:    <none>
Annotations: <none>

Data
====
APP_COLOR:
----
blue
APP_MODE:
----
prod
Events:    <none>
```

```
kubectl get configmaps
```

NAME	DATA	AGE
app-config	2	3s

Now that we have the config map created, let us proceed with step two, configuring it with a pod. Here I have a simple pod definition file that runs my simple web application.

To inject an environment variable, add a new property to the container called `envFrom`. The `envFrom` property is a list. So, we can pass as many environment variables as required. Each item in the list corresponds to a config map item. Specify the name of the config map we created earlier. This is how we inject a specific config map from the ones we created before. Creating the pod definition file now creates a web application with a blue background.

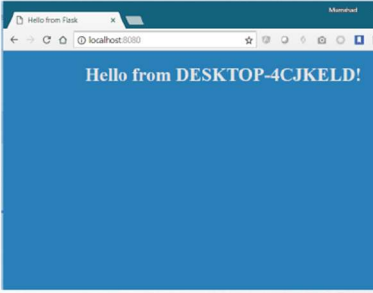
pod-definition.yaml

```
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
```



What we just saw was using config maps to inject environment variables. There are other ways to inject configuration data into pods. You can inject it as a single environment variable or you can inject the whole data as files in a volume.

```
envFrom:
- configMapRef:
  name: app-config
```

ENV

SINGLE ENV

```
env:
- name: APP_COLOR
  valueFrom:
    configMapKeyRef:
      name: app-config
      key: APP_COLOR
```

```
volumes:
- name: app-config-volume
  configMap:
    name: app-config
```

VOLUME