KUBERNETES CONFIGMAP: AN INTRODUCTION

In this post, I'm covering ConfigMaps in Kubernetes. I assume you have a basic understanding of Kubernetes, pods, and K8s use cases.

To follow along with the demo, you will need kubectl and minikube installed.

What is a ConfigMap?

When working with 12 factor apps, one of the factors is **configs**. That means, when you're dealing with microservices, you must figure out how your configurations will be applied.

If you want to deploy to multiple environments, like stage, dev, and prod, it's a bad practice to bake the configs into the application because of environment differences. Ideally, you'll want to separate configurations in order to match the deploy environment. This is where ConfigMap comes into play.

ConfigMaps allow you to decouple configuration artifacts from image content. This allows K8S to make your containerized application portable without you needing to worry about configurations. Both users and system components are able to store config data in ConfigMap.

ConfigMap is somewhat similar to secrets, but it provides a way of working with strings that don't contain sensitive information.

How to create ConfigMap

Creating a ConfigMap is fairly simple and straightforward. You can create it with directories, files, or literal values. We'll demonstrate each: the rest of the blog post will use a simple example to demonstrate how to work with a ConfigMap.

Create ConfigMap from a directory

To create ConfigMap from a directory, you must first create or have an existing directory with your configs there.

```
$ mkdir configmap-demo
```

Now, wget the configs into our ConfigMap demo directory.

```
$ wget
https://k8s.io/docs/tasks/configure-pod-container/configmap/kubectl/game.prop
erties -O configmap-demo
```

```
$ wget
https://k8s.io/docs/tasks/configure-pod-container/configmap/kubectl/ui.proper
ties -O configmap-demo
```

We then downloaded test config files into our directory, which we can create a ConfigMap from—it will have all the files we downloaded.

```
$ kubectl create configmap demo-configmap --from-file=configmap-demo
configmap "demo-configmap" created
```

If we describe our ConfigMap, we will see both files as data entries with their contents.

```
$ kubectl describe configmap demo-configmap
Name:         demo-configmap
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
game.properties:
----
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
ui.properties:
----
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice

Events:  <none>
```

Create ConfigMap from a file

Creating from a file is very similar to creating from a directory. All you'll need to do is pass in the name of the file to **–from-file argument**. When creating ConfigMap this way, you can pass in as many files as you want to the **–from-file argument** and it will add it to the ConfigMap.

Create ConfigMap from a literal value

Creating ConfigMap this way mean you can specify your configuration directly from command line without creating any file or directory. For example **kubectl create ConfigMap special-config –from-literal=special.how=very –from-literal=special.type=charm**. You can have multiple key-value pairs if needed.

Using ConfigMap in a pod

Let's create a test ConfigMap:

```
$ kubectl create configmap special-config --from-literal=special.how=very
configmap "special-config" created
```

Then create a pod to use the configmap as an env variable.

demo-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-pod
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command:
      env:
        # Define the environment variable
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              # The ConfigMap containing the value you want to assign to
SPECIAL_LEVEL_KEY
              name: special-config

              # Specify the key associated with the value
              key: special.how
  restartPolicy: Never
```

```
$ Kubectl create -f demo-pod.yaml
pod "configmap-demo-pod" created
```

We can now see that there is an environment variable set in the pod with our defined configmap

value.

```
$ kubectl logs configmap-demo-pod | grep SPECIAL_LEVEL_KEY
SPECIAL_LEVEL_KEY=very
```

Additional resources

For more on Kubernetes, explore these resources:

- Kubernetes Guide, with 20+ articles and tutorials
- BMC DevOps Blog
- The State of Kubernetes in 2020
- Bring Kubernetes to the Serverless Party
- How eBay is Reinventing Their IT with Kubernetes & Replatforming Program