**Getting Started with ConfigMaps**

Explore a bit about the ConfigMap Volume type.

**We'll cover the following**

* [The Need of the Hour](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/7AvVGAY0q18#The-Need-of-the-Hour)
* [The ConfigMap](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/7AvVGAY0q18#The-ConfigMap)

**The Need of the Hour**

ConfigMaps allow us to keep configurations separate from application images. Such separation is useful when other alternatives are not a good fit.

Almost every application can be fine-tuned through configuration. Traditional software deployment methods fostered the use of configuration files. However, we are not discussing traditional, but advanced, distributed, and immutable deployments through Kubernetes schedulers.

Usage of fundamentally new technology often requires new processes and different architecture, if we are to leverage its potential to its maximum. On the other hand, we cannot just throw away everything we have and start new.

We’ll have to try to balance new principles and legacy needs.

If we were to start developing a new application today, it would be, among other things, distributed, scalable, stateless, and fault tolerant. Those are some of today’s needs. While we might question how many of us know how to design an application with those quality attributes in mind, hardly anyone would argue against having any of them. What is often forgotten is the configuration. Which mechanism should your new application use to configure itself? *How about environment variables?*

Environment variables fit well into distributed systems. They are easy to define, and they are portable. They are the ideal choice for configuration mechanism of new applications.

However, in some cases, the configuration might be too complex for environment variables. In such situations, we might need to fall back to files (hopefully YAML). When those cases are combined with legacy applications which are almost exclusively using file-based configuration, it is evident that we cannot rely only on environment variables.

When a configuration is based on files, the best approach we can take is to bake the configuration into a Docker image. That way, we are going down the fully-immutable road. Still, that might not be possible when our application needs different configuration options for various clusters (e.g., testing and production).

We don’t want to convert this into a discussion that ends with “you do NOT need a different configuration for different environments”. Rather just assume that you might have an excellent reason for something like that. In such a case, baking config files into images will not do the trick. That’s where ConfigMaps comes into play.

**The ConfigMap**

ConfigMap allows us to “inject” configuration into containers. The source of the configs can be files, directories, or literal values. The destination can be files or environment variables.

**ConfigMap** takes a configuration from a source and mounts it into running containers as a *volume*.

That’s all the theory you’ll get up-front. Instead of a lengthy explanation, we’ll run some examples, and comment on the features we experience. We’ll be learning by doing, instead of learning by memorizing theory.

For setting ConfigMap you will use the following command after creating the cluster.

kubectl config current-context

However for practising the code on the platform context is already configured.

# Injecting Configuration from a Single File

Learn to inject configuration from a single file using the ConfigMap Volume.

**We'll cover the following**

* [Creating a ConfigMap](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/JYRnmn7xom2#Creating-a-ConfigMap)
  + [Looking into the description](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/JYRnmn7xom2#Looking-into-the-description)
* [Mounting the ConfigMap](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/JYRnmn7xom2#Mounting-the-ConfigMap)
  + [Pod with mounted ConfigMap](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/JYRnmn7xom2#Pod-with-mounted-ConfigMap)
  + [Creating the pod](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/JYRnmn7xom2#Creating-the-pod)
  + [Verification](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/JYRnmn7xom2#Verification)
* [Deleting the objects](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/JYRnmn7xom2#Deleting-the-objects)
* [Try it yourself](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/JYRnmn7xom2#Try-it-yourself)

## Creating a ConfigMap

In its purest, and probably the most common form, a ConfigMap takes a single file. For example, we can create one from the prometheus-conf.yml file.

kubectl create cm my-config --from-file=prometheus-conf.yml

We created a ConfigMap (cm) called my-config. The data of the map is the content of the prometheus-conf.yml file.

### Looking into the description

Let’s describe it, and see what we’ll get.

kubectl describe cm my-config

The **output** is as follows.

Name: my-config

Namespace: default

Labels: <none>

Annotations: <none>

Data

====

prometheus-conf.yml:

----

global:

scrape\_interval: 15s

scrape\_configs:

- job\_name: prometheus

metrics\_path: /prometheus/metrics

static\_configs:

- targets:

- localhost:9090

Events: <none>

The important part is located below “Data”. We can see the key which, in this case, is the name of the file (“prometheus-conf.yml”). Further down you can see the content of the file. If you look at the definition of “prometheus-conf.yml” in the playground at the end of this lesson, you’ll see that it is the same as what we saw from the ConfigMap’s description.

## Mounting the ConfigMap

ConfigMap is useless by itself. It is yet another Volume that, like all the others, needs a mount.

### Pod with mounted ConfigMap

Let’s take a look at a Pod specification defined in alpine.yml.

apiVersion: v1

kind: Pod

metadata:

name: alpine

spec:

containers:

- name: alpine

image: alpine

command: ["sleep"]

args: ["100000"]

volumeMounts:

- name: config-vol

mountPath: /etc/config

volumes:

- name: config-vol

configMap:

name: my-config

The essential sections are volumeMounts and volumes. Since volumeMounts are the same no matter the type of the Volume, there’s nothing special about it. We defined that it should be based on the volume called config-vol and that it should mount the path /etc/config. The volumes section uses configMap as the type and, in this case, has a single item name, that coincides with the name of the ConfigMap we created earlier.

### Creating the pod

Let’s create the Pod and see what happens.

kubectl create -f alpine.yml

kubectl get pods

Please confirm that the Pod is indeed running before moving on.

### Verification

Let’s see the content of the /etc/config directory inside the Pod’s only container.

kubectl exec -it alpine -- ls /etc/config

The **output** is as follows.

prometheus-conf.yml

The /etc/config now has a single file that coincides with the file we stored in the ConfigMap.

Let’s add -l to the ls command we executed a moment ago.

kubectl exec -it alpine -- ls -l /etc/config

The **output** is as follows.

total 0

lrwxrwxrwx 1 root root 26 Jun 20 06:04 prometheus-conf.yml -> ..data/prometheus-conf.yml

You’ll see that “prometheus-conf.yml” is a link to “…data/prometheus-conf.yml”.

If you dig deeper, you’ll see that “…data” is also a link to the directory named from a timestamp. And so on, and so forth. For now, the exact logic behind all the links and the actual files is not of great importance. From the functional point of view, there is prometheus-conf.yml, and our application can do whatever it needs to do with it.

Let’s confirm that the content of the file inside the container is indeed the same as the source file we used to create the ConfigMap.

kubectl exec -it alpine -- cat /etc/config/prometheus-conf.yml

The **output** should be the same as the contents of the prometheus-conf.yml file.

We saw one combination of ConfigMap. Let’s see what else we can do with it.

## Deleting the objects

We’ll remove the objects we created thus far and start over.

kubectl delete -f alpine.yml

The command to delete the ConfigMap is as follows.

kubectl delete cm my-config

## Try it yourself

A list of all the commands used in the lesson is given below.

kubectl create cm my-config --from-file=prometheus-conf.yml

kubectl describe cm my-config

kubectl create -f alpine.yml

kubectl get pods

kubectl exec -it alpine -- ls /etc/config

kubectl exec -it alpine -- ls -l /etc/config

kubectl exec -it alpine -- cat /etc/config/prometheus-conf.yml

kubectl delete -f alpine.yml

kubectl delete cm my-config

# Injecting Configurations from Multiple Files

Learn how to inject configuration from multiple files and from a directory.

**We'll cover the following**

* [Creating a ConfigMap from multiple files](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/g2OvZOBENwZ#Creating-a-ConfigMap-from-multiple-files)
  + [Deleting the objects](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/g2OvZOBENwZ#Deleting-the-objects)
* [Creating a ConfigMap from a directory](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/g2OvZOBENwZ#Creating-a-ConfigMap-from-a-directory)
  + [Deleting the objects](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/g2OvZOBENwZ#Deleting-the-objects)
* [Try it yourself](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/g2OvZOBENwZ#Try-it-yourself)

## Creating a ConfigMap from multiple files

Let's see what happens when we execute the commands that follow.

kubectl create cm my-config --from-file=cm/prometheus-conf.yml --from-file=cm/prometheus.yml

kubectl create -f cm/alpine.yml

#Run the following command separately kubectl exec -it alpine -- ls /etc/config

We created a ConfigMap with two files, and we created the same Pod based on the alpine.yml definition. Finally, we output the list of files from the /etc/config directory inside the Pod’s only container. The **output** of the latter command is as follows.

prometheus-conf.yml prometheus.yml

We can see that both files are present in the container. That leads us to the conclusion that a ConfigMap can **contain multiple files**, and all will be created inside containers that mount it.

### Deleting the objects

Let’s delete the objects (again), and explore one more option behind the --from-file argument.

kubectl delete -f alpine.yml

#Run the following command separately to delete the configmap

kubectl delete cm my-config

### Deleting the objects

Let’s delete the objects (again), and explore one more option behind the --from-file argument.

kubectl delete -f alpine.yml

#Run the following command separately to delete the configmap

kubectl delete cm my-config

## Creating a ConfigMap from a directory

The --from-file argument might lead you to the conclusion that you can specify only a file path as its value. It works with directories as well. We can, for example, add all files from the cm directory to a ConfigMap.

kubectl create cm my-config --from-file=cm

We created my-config ConfigMap with the directory cm. Let’s describe it, and see what’s inside.

kubectl describe cm my-config

The **output** is as follows (content of the files is removed for brevity).

Name: my-config

Namespace: default

Labels: <none>

Annotations: <none>

Data

====

alpine-env-all.yml:

----

...

alpine-env.yml:

----

...

alpine.yml:

----

...

my-env-file.yml:

----

...

prometheus-conf.yml:

----

...

prometheus.yml:

----

...

Events: <none>

We can see that all six files from the cm directory are now inside the “my-config” ConfigMap.

We’re sure you already know what will happen if we create a Pod that mounts that ConfigMap. We’ll check it out anyways.

kubectl create -f cm/alpine.yml

#Run the below command separately after the "alpine" container is created

kubectl exec -it alpine -- ls /etc/config

The **output** of the latter command is as follows.

alpine-env-all.yml alpine.yml prometheus-conf.yml

alpine-env.yml my-env-file.yml prometheus.yml

All the files are there, and the time has come to move away from files and directories.

### Deleting the objects[**#**](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/g2OvZOBENwZ#Deleting-the-objects)

So, let’s remove the objects first, and discuss the other sources.

kubectl delete -f alpine.yml

#Run the following command separately to delete the configmap

kubectl delete cm my-config

## Try it yourself[#](https://www.educative.io/module/lesson/a-practical-guide-to-kubernetes/g2OvZOBENwZ#Try-it-yourself)

A list of all the commands used in the lesson is given below.

kubectl create cm my-config --from-file=cm/prometheus-conf.yml --from-file=cm/prometheus.yml

kubectl create -f cm/alpine.yml

#Run the following command separately

kubectl exec -it alpine -- ls /etc/config

kubectl delete -f cm/alpine.yml

#Run the following command separately to delete the configmap

kubectl delete cm my-config

kubectl create cm my-config --from-file=cm

kubectl describe cm my-config

kubectl create -f cm/alpine.yml

#Run the below command separately after the "alpine" container is created

kubectl exec -it alpine -- ls /etc/config

kubectl delete -f cm/alpine.yml

#Run the following command separately to delete the configmap

kubectl delete cm my-config