Defining ConfigMaps as YAML

In this lesson, we will explore defining ConfigMaps as YAML files.

WE'LL COVER THE FOLLOWING

All ConfigMaps we created so far were done through `kubectl create cm` commands. It would be a shame if we could not specify them through YAML definitions, just like other Kubernetes resources and objects. Fortunately, we can. Everything in Kubernetes can be defined as YAML, and that includes ConfigMaps as well.

Looking into the YAML

Even though we have not yet specified ConfigMaps as YAML, we have seen the format quite a few times throughout this chapter. Since we cannot be sure whether you can create a ConfigMap YAML file from memory, let's make things easy on ourselves and use `kubectl` to output our existing `my-config` ConfigMap in YAML format.

```
kubectl get cm my-config -o yaml
```

The **output** is as follows.

```
apiVersion: v1
data:
  something: else
  weather: sunny
kind: ConfigMap
metadata:
```

```
  name: my-config
  ...
```

Just as with any other Kubernetes object, ConfigMap has `apiVersion`, `kind`, and `metadata`. The data is where the maps are defined. Each must have a key and a value. In this example, there's the key `weather` with the value `sunny`.

Deploying Prometheus

Let's try to translate that knowledge into the objects we'd need to deploy Prometheus.

Looking into the Definition

```
cat cm/prometheus.yml
```

The **output**, limited to the relevant parts, is as follows.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
spec:
  ...
  template:
    ...
    spec:
      containers:
        ...
        volumeMounts:
        - mountPath: /etc/prometheus
          name: prom-conf
      volumes:
      - name: prom-conf
        configMap:
          name: prom-conf
...
apiVersion: v1
kind: ConfigMap
metadata:
  name: prom-conf
data:
  prometheus.yml: |
    global:
      scrape_interval:     15s

    scrape_configs:
      - job_name: prometheus
        metrics_path: /prometheus/metrics
        static_configs:
          - targets:
            - localhost:9090
```

Line 12-14: The `Deployment` object defines the `volumeMounts` that references the `prom-conf` Volume, which is a `configMap`. We saw quite a few similar examples before.

Line 25: The `ConfigMap` object's `data` section has only one key (`prometheus.yml`). Once this ConfigMap is mounted as a volume, the name of the file will be the same as the key (`prometheus.yml`).

The value has a bit of "special" syntax. Unlike the previous example where the value was a single word written directly after the colon, the structure of the value is now a bit more complex. To be more precise, it contains multiple lines.

When working with a large value, we can start with the pipe sign (`|`). Kubernetes will interpret the value as "everything that follows, as long as it is indented." You'll notice that all the lines of the value are at least two spaces to the right of the beginning of the key (`prometheus.yml`). If you'd like to insert an additional key, all you'd need to do is to add it on the same level (indentation), as the other `prometheus.yml`.

Creating the Application

Let's create the application and confirm that everything works as expected.

```
cat cm/prometheus.yml | sed -e \
    "s/192.168.99.100/$(minikube ip)/g" \
    | kubectl create -f -

kubectl rollout status deploy prometheus

open "http://$(minikube ip)/prometheus/targets"
```

We created the objects (with the help of `sed` transformations), we waited until the Deployment rolled out, and, finally, we opened the Prometheus targets screen in a browser. The result should be a green target towards Prometheus' internal metrics.

In the next lesson, we will explore where one should not use ConfigMaps.