**DevOps in K8s — ConfigMap**

A logo of a ship wheel

Description automatically generated with medium confidence

One crucial aspect in K8s is managing application configurations, securely storing and using sensitive information (such as passwords, tokens, etc.), configuring container runtime resources, enforcing security control, identity authentication, and so on.

Mutable configurations of applications in K8s are achieved through a resource object called **ConfigMap**. Often, applications require reading configuration information from configuration files, command-line parameters, or environment variables.

You should avoid hardcoding such information directly into the application program since modifying the code and creating a new image every time we switch to a different service is impractical. **ConfigMap** empowers us to inject configuration information into the container, allowing us to store not just individual properties but the entire configuration file.

**What is ConfigMap**

ConfigMap is an API object that provides a way to store configuration data in **key-value** pairs, separate from the container image. ConfigMaps can be used to store configuration parameters such as database connection strings, environment variables, and command-line arguments that are required by applications running in a container.

For example,

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: app-config  
data:  
 database-url: jdbc:postgresql://localhost:5432/mydb  
 api-key: ABC1234

The above ConfigMap creates a “app-config” with two key-value pairs: “database-url” and “api-key”. These values can then be used in the container’s environment variables or command-line arguments.

spec:  
 containers:  
 - name: myapp  
 image: myapp:latest  
 env:  
 - name: DATABASE\_URL  
 valueFrom:  
 configMapKeyRef:  
 name: app-config  
 key: database-url  
 - name: API\_KEY  
 valueFrom:  
 configMapKeyRef:  
 name: app-config  
 key: api-key

**ConfigMap Syntax**

**config field**

The “config” field in a ConfigMap is used to store configuration data in files, where each key represents the filename and the value is the contents of the file. This is useful when an application requires configuration data to be stored in a file rather than as individual key-value pairs.

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: app-config  
data:  
 database-url: jdbc:postgresql://localhost:5432/mydb  
 api-key: ABC1234  
 config.yml: |-  
 foo: bar  
 baz: qux

In this example, the “app-config” ConfigMap contains three key-value pairs, including a “config.yml” key with a value that represents the contents of a YAML file. The |- symbol after the config.yml: key indicates that the value is a multi-line string literal.

Then the Deployment can mount the “config.yml” from “app-config”:

spec:  
 containers:  
 - name: myapp  
 image: myapp:latest  
 volumeMounts:  
 - name: config-volume  
 mountPath: /config  
 env:  
 - name: DATABASE\_URL  
 valueFrom:  
 configMapKeyRef:  
 name: app-config  
 key: database-url  
...  
 volumes:  
 - name: config-volume  
 configMap:  
 name: app-config  
 items:  
 - key: config.yml  
 path: config.yml

**“|” operator**

The “|” symbol is a YAML block scalar style that instructs YAML parsers to preserve line breaks and indentation within the string value. This is useful for storing configuration data that requires a specific format, such as scripts or configuration files.

In ConfigMaps, the “|” symbol is used to create multi-line strings in YAML files.

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: app-config  
data:  
 config.yml: |  
 foo: bar  
 baz: qux

* |+ : preserves any trailing line breaks in the string value.
* |- : strips any trailing line breaks from the string value.

For example:

value: |  
 hello  
# {"value": "hello\n"}  
  
value: |-  
 hello  
# {"value": "hello"}  
  
value: |+  
 hello  
# {"value": "hello\n\n"}

**Check ConfigMap Usage**

You can always use kubectl create configmap -h to check how to use ConfigMap:

$ kubectl create configmap -h  
Create a config map based on a file, directory, or specified literal value.  
  
 A single config map may package one or more key/value pairs.  
  
 When creating a config map based on a file, the key will default to the basename of the file, and the value will  
default to the file content. If the basename is an invalid key, you may specify an alternate key.  
  
 When creating a config map based on a directory, each file whose basename is a valid key in the directory will be  
packaged into the config map. Any directory entries except regular files are ignored (e.g. subdirectories, symlinks,  
devices, pipes, etc).  
...

**Create ConfigMap from File**

In K8s, you can create a ConfigMap from a file using the kubectl create configmap command.

$ cat config.properties  
database-url=jdbc:postgresql://localhost:5432/mydb  
api-key=ABC1234

Create ConfigMap:

$ kubectl create configmap app-config --from-file=config.properties  
  
$ ]$ kubectl get configmap app-config2 -oyaml  
apiVersion: v1  
data:  
 config.properties: |  
 database-url=jdbc:postgresql://localhost:5432/mydb  
 api-key=ABC1234  
kind: ConfigMap  
metadata:  
 creationTimestamp: "2023-04-23T21:09:54Z"  
 name: app-config2  
 namespace: default  
 resourceVersion: "71326"  
 uid: 046b3b8d-b028-4be7-85a8-881f8c1c3fb8

**ConfigMap Common Usage**

After successfully creating a ConfigMap, there are multiple ways to use its data within a Pod. The following are some common approaches:

* Setting environment variables using ConfigMap data.

spec:  
 containers:  
 - name: myapp  
 image: myapp:latest  
 env:  
 - name: DATABASE\_URL  
 valueFrom:  
 configMapKeyRef:  
 name: app-config  
 key: database-url  
 - name: API\_KEY  
 valueFrom:  
 configMapKeyRef:  
 name: app-config  
 key: api-key

In this example, the env field in the Pod's container definition is used to set environment variables. The valueFrom field is used to reference the app-config ConfigMap and the specific key (database-url and api-key) whose values should be used as the environment variable value.

* Passing command-line arguments to the container using the values stored in the ConfigMap.

spec:  
 containers:  
 - name: myapp  
 image: myapp:latest  
 command: ["java", "-jar", "myapp.jar", "--db-url=$(DATABASE\_URL)", "--api-key=$(API\_KEY)"]  
 env:  
 - name: DATABASE\_URL  
 valueFrom:  
 configMapKeyRef:  
 name: app-config  
 key: database-url  
 - name: API\_KEY  
 valueFrom:  
 configMapKeyRef:  
 name: app-config  
 key: api-key

In this example, the command field in the Pod's container definition is used to pass command-line arguments to the container. The --db-url and --api-key arguments reference the DATABASE\_URL and API\_KEY environment variables, respectively.

* Mounting a volume inside the Pod and using ConfigMap data to populate configuration files within the volume.

apiVersion: v1  
kind: Pod  
metadata:  
 name: myapp  
spec:  
 containers:  
 - name: myapp  
 image: myapp:latest  
 volumeMounts:  
 - name: config-volume  
 mountPath: /config  
 volumes:  
 - name: config-volume  
 configMap:  
 name: app-config  
 items:  
 - key: config.yml  
 path: config.yml

In this example, the volumes field in the Pod definition is used to define the config-volume volume, which is populated with the contents of the config.yml file in the app-config ConfigMap. The items field is used to specify the key (config.yml) and the path inside the volume (/config/config.yml) where the configuration file should be stored.

**Note**

* It’s worth noting that if a ConfigMap is mounted into a Pod as a volume, any updates to the ConfigMap (or recreating it) **will result in the configuration information mounted in the Pod being hot updated**. To achieve application hot updates, you can include scripts to monitor changes in the configuration files and then reload the relevant services.
* Only Pods created via the K8s API are capable of using ConfigMaps. Other types of Pods (such as static Pods) are unable to use them.
* ConfigMap files are restricted to a maximum size of 1MB to comply with ETCD’s requirements.

**Conclusion**

A diagram of a software company

Description automatically generated with medium confidence