CONFIGMAP & SPRINGBOOT

8/30/21 - Anh

# PURPOSE

* Externalize configuration using ConfigMap (environment vars & file)
* Springboot app consumes the configuration (env vars, file, path to the volume)

# OVERVIEW

Create a local minikube cluster. Create a simple Springboot REST application. Deploy the application to the local cluster. Create 2 ConfigMaps: one is a new environment variable, the other ConfigMap is from the src/resources/application.properties file. Hit the REST endpoints to see the environment variable and the key-value in application.properties.

Delete and re-create the ConfigMap(s) with new values. Hit the REST endpoints again. They are still showing the old values until we restart the pod.

# SETUP

## MATERIAL NEEDED

* Java 11+
* Maven 3.6.3+
* Minikube 1.20+

## LOCAL CLUSTER

The demo Springboot application uses fabric8 maven plugin to build and deploy to the local cluster. The key to make it work is to configure it to the Docker Registry of the local Kubernetes, not your normal local Docker Registry. Run this command to redirect the images to the Docker registry inside your local Kubernetes: ***eval $(minikube docker-env)***. Fail to do this, the build process will deploy your image to your local Docker and you won’t see it in your cluster.

## CREATE CONFIGMAPS

Although fabric8 can also create ConfigMaps, I don’t know how to do it yet. So, I created them by hand for now. This must be done first before we build and deploy the application which uses the ConfigMaps. You’ll get an error if the deployment couldn’t find the ConfigMaps.

* kubectl create configmap greeting-cm --from-literal=greeter.prefix="Hello from env var"
* kubectl create configmap spring-app-config --from-file=src/main/resources/application.properties

## BUILD AND DEPLOY

* mvn clean install
* ./mvnw clean fabric8:deploy

If this is successful, open the kubernetes dashboard to see a complete picture of the deployment (i.e. “minikube dashboard” command).

In src/main/fabric8/deployment.yaml, we map the app.properties to “/deployments/config”. You could shell into the running pod to verify this path and file existed (i.e. “kubectl exec --stdin --tty <podname> -- /bin/bash” command).

## ACTIVITY

First, we check the “default” values.

| **Command** | **Description** |
| --- | --- |
| curl $(minikube service hellokube --url)/greeting; echo ""; | The result is the value from the environment variable “greeter.prefix” |
| curl $(minikube service hellokube --url)/configvalue; echo ""; | The result is the value of “user.myvar1” from the app.properties file mounted on the pod path /deployments/config, using @Value annotation. |
| curl $(minikube service hellokube --url)/configdto; echo ""; | The result is the same as above but we use @Configuration annotation and a POJO class. |

Next, we go to the dashboard, edit the ConfigMaps, enter some different values for greeter.prefix and user.myvar1. We run the above and they won’t show the newly edited values until we delete the pod (using dashboard menu). Kubernetes will restart another pod and we will see the new values.

## USEFUL COMMANDS

| **Commands** | **Descriptions** |
| --- | --- |
| mvn clean install | Compile our Springboot app |
| kubectl create configmap greeting-cm --from-literal=greeter.prefix="Hello from env var" | Create a ConfigMap which defines a single environment variable. “greeting-cm” is the name of the ConfigMap. |
| kubectl create configmap spring-app-config --from-file=src/main/resources/application.properties | Create a ConfigMap from src/../application.properties file. “spring-app-config” is the name of the ConfigMap. |
| ./mvnw clean fabric8:deploy | Deploy our Springboot app to the local cluster |
| minikube dashboard | Verify everything is up and running. View and edit pod and configMaps here is easier. |
| kubectl exec --stdin --tty hellokube-5b69dcd4c-ktg6l -- /bin/bash | Shell into a pod (to check if the volume is mounted, example: /deployments/config) |

# CONCLUSION

There are many different ways to create a ConfigMap (environment variables, key-value file, mounted as a volume). Each might require a different method to consume it (getEnv, getProperties, bean, Kubernetes API). This demo shows 2 different ways of creating the ConfigMap: as environment variables and as a mounted volume (the mounted file is application.properties). The demo also shows 2 ways of consuming the 2 ConfigMaps: @Value and @Configuration (bean?). Both are standard Spring I/O methods. It does not have the last consuming method: Kubernetes API which might be the desired solution (i.e. update ConfigMaps and the application read them right away without restarting).

# REFERENCES

| https://developers.redhat.com/blog/2017/10/03/configuring-spring-boot-kubernetes-configmap#setup | Most useful |
| --- | --- |
| https://blog.nebrass.fr/playing-with-spring-boot-on-kubernetes/ | More about Kubernetes than Springboot |
| https://spring.io/blog/2020/08/14/config-file-processing-in-spring-boot-2-4 | More about Springboot 2.4 profile than Kubernetes. It mentions the Volume mount ConfigMap. no code examples. |
| https://itnext.io/working-with-kubernetes-configmaps-part-1-volume-mounts-f0ace283f5aa | Article with example code (Github) to consume ConfigMap mounted as a volume. However, it is in the Go language. |
| https://dzone.com/articles/k8s-knowhow-using-a-configmap | Article with Github example to consume a configmap mounted as volume |

END