**Proof of Concept**

Serving configuration using Kubernetes ConfigMap

Author Aleksandrs Krivickis

[Alex.a.krivickis@hsbc.com.hk](mailto:Alex.a.krivickis@hsbc.com.hk)

Contents

No table of contents entries found.

# Scope

* Find out if it is possible to reliably serve configuration to Kubernetes nodes using ConfigMap
* Determine a format in which configuration is stored and if it is possible to use application-profile-label structure
* Explore the way configuration updates are pushed, find out if it is possible to change it on the fly

# Investigation

## Multiple ways of delivering configuration

1. Serve configuration directly to system’s environment and retrieve it using native Java procedure - System.getenv(). <https://developers.redhat.com/blog/2017/10/03/configuring-spring-boot-kubernetes-configmap/>
2. Mount application.properties or application.yml files in the Spring Boot application from ConfigMaps. <https://dzone.com/articles/configuring-spring-boot-on-kubernetes-with-configm>



## Updating configuration

Some applications may need to detect changes on external property sources and update their internal status to reflect the new configuration. The reload feature of Spring Cloud Kubernetes is able to trigger an application reload when a related ConfigMap or Secret change.

This feature is disabled by default and can be enabled using the configuration property spring.cloud.kubernetes.reload.enabled=true (for example, in the bootstrap.properties file).

The following levels of reload are supported (property spring.cloud.kubernetes.reload.strategy):

**refresh**

*(default)* only configuration beans annotated with @ConfigurationProperties or @RefreshScope are reloaded. This reload level leverages the refresh feature of Spring Cloud Context.

**NOTE**

The PropertySource reload feature can only be used for *simple* properties (that is, not collections) when the reload strategy is set to refresh. Properties backed by collections must not be changed at runtime.

**restart\_context**

the whole Spring *ApplicationContext* is gracefully restarted. Beans are recreated with the new configuration.

**shutdown**

the Spring *ApplicationContext* is shut down to activate a restart of the container. When using this level, make sure that the lifecycle of all non-daemon threads is bound to the ApplicationContext and that a replication controller or replica set is configured to restart the pod.

## Configuration structure

It is possible to ensure application-profile-label structure:

* By delegating appropriate profile names. <https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-profiles.html>
* Another way to implement structure would be defining a different config Mount for different application

# Summary and comparison with Spring Cloud Config

|  |  |  |
| --- | --- | --- |
| **Feature** | **Spring Cloud Config** | **Kubernetes ConfigMap** |
| New config is available on demand | No, update hook must be triggered | Yes, if reload configuration variable is enabled |
| application-profile-label structure | Yes | Yes |
| Nested configuration variables | Yes | Yes |

# References