# K8s ConfigMap 介绍与应用

AUTHOR: 彭玲 TIME: 2021/12/30

---



---

# ConfigMap 简介

ConfigMap 是一种 API 对象，用来将**非机密性的**数据保存到 `键值对` 中。使用时，Pods 可以将其用作 `环境变量`、`命令行参数` 或者 `存储卷中的配置文件`。

ConfigMap 将您的 `环境配置` 信息 和 `容器镜像` 解耦，便于应用配置的修改。

# ConfigMap 创建方式

## 基于文件创建 ConfigMap

可以使用 `kubectl create configmap` 基于单个文件或多个文件创建 ConfigMap。例如：

```
anxin@node38:~/pengling/k8s/configure-pod-container/configmap$ vi
game.properties

enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
```

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
```

从`game.properties`文件创建 ConfigMap：

```
anxin@node38:~/pengling/k8s/configure-pod-container/configmap$ kubectl create configmap game-config-2 --from-file=game.properties
configmap/game-config-2 created
```

将产生以下 ConfigMap：

```
anxin@node38:~$ kubectl describe configmaps game-config-2
Name:       game-config-2
Namespace:   default
Labels:     <none>
Annotations:  <none>

Data
====
game.properties: # 文件名：作为 ConfigMap 的 data 部分的 key
----
enemies=aliens # 文件内容：成为 key 对应的值
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
Events:  <none>
```

### 定义从文件创建 ConfigMap 时要使用的键

在使用`--from-file`参数时，你可以定义在 ConfigMap 的`data`部分出现键名， 而不是按默认行为使用文件名：

`<my-key-name>`是你要在 ConfigMap 中使用的键名，`<path-to-file>`是你想要键表示数据源文件的位置。

```
kubectl create configmap game-config-3 --from-file=<my-key-name>=<path-to-file>
```

### 根据`字面值`创建 ConfigMap

可以将`kubectl create configmap`与`--from-literal`参数一起使用，从命令行定义`文字值`：

```
kubectl create configmap special-config --from-literal=special.how=very --from-literal=special.type=charm
```

你可以传入多个`键值`(key-value) 对。命令行中提供的每对键值在 ConfigMap 的 `data` 部分中均表示为单独的条目。

# 使用 ConfigMap 配置 Redis

## 1. ConfigMap data 键值为`""`

### 创建 ConfigMap

创建一个配置模块为`空`(`""`) 的 ConfigMap:

```
cat <<EOF >./example-redis-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-redis-config
data:
  redis-config: "" # redis-config 键
EOF
```

应用上面创建的 ConfigMap 配置文件:

```
# 创建 ConfigMap
anxin@node38:~/pengling/k8s/configmap-redis$ kubectl apply -f example-redis-config.yaml
configmap/example-redis-config created
```

### 创建 Redis 的 Pod

pods/config/redis-pod.yaml 如下:

```
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
  - name: redis
    image: redis:5.0.4
    command:
      - redis-server
      - "/redis-master/redis.conf"
    env:
    - name: MASTER
      value: "true"
    ports:
    - containerPort: 6379
    resources:
      limits:
        cpu: "0.1"
    volumeMounts:
    - mountPath: /redis-master-data
      name: data
    - mountPath: /redis-master # config 卷被挂载到 /redis-master
```

```
      name: config
  volumes:
    - name: data
      emptyDir: {}
    - name: config
      configMap:
        name: example-redis-config # ConfigMap 名称
        items:
        - key: redis-config # key 和 path 会将 example-redis-config 中的 redis-config 键 公开在
          path: redis.conf # config 卷上一个名为 redis-config 的文件中
```

创建 Redis pod：

```
anxin@node38:~/pengling/k8s/configmap-redis$ kubectl apply -f ./redis-pod.yaml
pod/redis created
```

最终效果：

将 ConfigMap `example-redis-config` 配置中 `data.redis-config` 的数据作为 Pod 中的 `/redis-master/redis.conf` 公开。

## 检查创建的对象

```
anxin@node38:~$ kubectl get pod/redis configmap/example-redis-config
NAME         READY     STATUS    RESTARTS   AGE # Pod 对象
pod/redis    1/1       Running   0          122m

NAME                      DATA   AGE # ConfigMap 对象
configmap/example-redis-config  1      123m
```

## 查看 ConfigMap 详情

回顾一下，我们在 `example-redis-config` ConfigMap 保留了`空("")`的 `redis-config` 键：

```
anxin@node38:~$ kubectl describe configmap/example-redis-config
Name:         example-redis-config
Namespace:    default
Labels:       <none>
Annotations:
Data
====
redis-config: # 一个空的 redis-config 键
----

Events:  <none>
```

## 查看 Redis 配置 (`redis-cli`)

使用 `kubectl exec` 进入 pod，运行 `redis-cli` 工具检查当前配置：

```
anxin@node38:~$ kubectl exec -it redis -- redis-cli
127.0.0.1:6379> CONFIG GET maxmemory # 查看 maxmemory
1) "maxmemory"
2) "0" # 默认值 0
127.0.0.1:6379> CONFIG GET maxmemory-policy # 查看 maxmemory-policy
1) "maxmemory-policy"
2) "noeviction" # 默认值 noeviction
```

## 2. ConfigMap data 键值为非""

### 修改 ConfigMap

pods/config/example-redis-config.yaml 修改后如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-redis-config
data:
  redis-config: |
    maxmemory 2mb
    maxmemory-policy allkeys-lru
```

应用更新的 ConfigMap：

```
anxin@node38:~/pengling/k8s/configmap-redis$ kubectl apply -f example-redis-config.yaml
configmap/example-redis-config configured
```

### 查看 ConfigMap 详情

可以看到我们刚刚添加的配置：

```
anxin@node38:~$ kubectl describe configmap/example-redis-config
Name:         example-redis-config
Namespace:    default
Labels:       <none>
Annotations:
Data
====
redis-config:
----
maxmemory 2mb # 添加的配置
maxmemory-policy allkeys-lru

Events:  <none>
```

### 重启 Pod ( Redis )

通过 `kubectl exec` 使用 `redis-cli` 再次检查 Redis Pod，查看是否已应用配置。

结果：`maxmemory` 和 `maxmemory-policy` 均保持默认值。因为**需要重新启动 Pod** 才能从关联的 ConfigMap 中获取更新的值。

删除并重新创建 Pod：

```
anxin@node38:~$ kubectl delete pod redis
pod "redis" deleted

anxin@node38:~$ kubectl apply -f ./redis-pod.yaml
pod/redis created
```

重新检查 Redis 配置值：

```
anxin@node38:~$ kubectl exec -it redis -- redis-cli
127.0.0.1:6379> CONFIG GET maxmemory
1) "maxmemory"
2) "2097152" # maxmemory 已更新
127.0.0.1:6379> CONFIG GET maxmemory-policy
1) "maxmemory-policy"
2) "allkeys-lru" # maxmemory-policy 已更新
```

## 删除创建的资源

删除创建的资源，清理你的工作：

```
anxin@node38:~$ kubectl delete pod/redis configmap/example-redis-config
pod "redis" deleted
configmap "example-redis-config" deleted
```