Kubernetes 配置管理中心 ConfigMap

前言：

课程名称：Kubernetes 配置管理中心 ConfigMap

实验环境：

本章节 Kubernetes 集群环境如下：

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
工作节点	192.168.128.21	k8s-node01	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化
工作节点	192.168.128.22	k8s-node02	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化

应用部署的一个最佳实践是将应用所需的配置信息与程序进行分离，这样可以使应用程序被更好地复用，通过不同的配置也能实现更灵活的功能。将应用打包为容器镜像后，可以通过环境变量或者外挂文件的方式在创建容器时进行配置注入，但在大规模容器集群的环境中，对多个容器进行不同的配置将变得非常复杂。从 Kubernetes 1.2 开始提供了一种统一的应用配置管理方案——ConfigMap。

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

1、ConfigMap 概述

1.1、什么是 ConfigMap?

在 Kubernetes 中，ConfigMap 用来存储应用程序的配置信息、命令行参数、属性文件等数据，可以将这些信息和应用程序本身分离，从而方便管理和维护。

ConfigMap 是一个 Kubernetes API 对象，它由一些键值对组成，并提供了 API 来创建、更新和删除这些键值对。在存储键值对的时候，ConfigMap 支持从多种来源读取数据，如目录、文件、命令行参数等，从而方便用户将数据导入到 Kubernetes 集群中。

在使用 ConfigMap 时，用户可以将其作为一种独立的对象在 Kubernetes 中进行创建和管理。一旦创建，ConfigMap 就可以在 Kubernetes 中的任何其他资源中使用，如 Pod 定义、容器定义、环境变量等。用户可以通过 Kubernetes API 的 Client 库、kubectl 工具和其他客户端工具访问和修改 ConfigMap。

一种常见的 ConfigMap 使用场景是在 Pod 中使用环境变量来进行配置。用户可以在 ConfigMap 中定义所需的属性、配置文件路径等信息，随后在 Pod 定义文件中使用这些环境变量引用它们。当 Pod 启动时，它会自动读取这些环境变量并使用它们来配置应用程序。

总的来说，ConfigMap 为 Kubernetes 中的配置管理提供了一种简单、灵活、可重用的方式。它可以帮助避免硬编码配置信息，提高了应用程序的可维护性和可移植性。

总结：

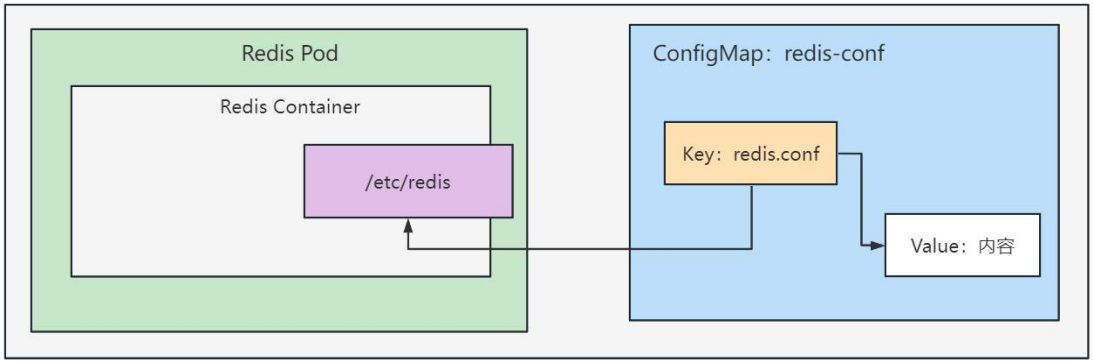
Configmap 是 k8s 中的资源对象，用于保存非机密性的配置的，数据可以用

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

key/value 键值对的形式保存，也可通过文件的形式保存。

1.2、ConfigMap 能解决哪些问题？

我们在部署服务的时候，每个服务都有自己的配置文件，如果一台服务器上部署多个服务：nginx、tomcat、apache 等，那么这些配置都存在这个节点上，假如一台服务器不能满足线上高并发的要求，需要对服务器扩容，扩容之后的服务器还是需要部署多个服务：nginx、tomcat、apache，新增加的服务器上还是要管理这些服务的配置，如果有一个服务出现问题，需要修改配置文件，每台物理节点上的配置都需要修改，这种方式肯定满足不了线上大批量的配置变更要求。所以，k8s 中引入了 Configmap 资源对象，可以当成 volume 挂载到 pod 中，实现统一的配置管理。



1、Configmap 是 k8s 中的资源，相当于配置文件，可以有一个或者多个 Configmap。

2、Configmap 可以做成 Volume，k8s pod 启动之后，通过 volume 形式映射到容器内部指定目录上。

3、容器中应用程序按照原有方式读取容器特定目录上的配置文件。

4、在容器看来，配置文件就像是打包在容器内部特定目录，整个过程对应用没有任何侵入。

1.3、Configmap 应用场景

ConfigMap 是 Kubernetes 集群中的一种资源对象，它用于存储非敏感的配置数据，如环境变量、命令行参数、密钥、配置文件等，可以在多个应用程序之间共享。

ConfigMap 的应用场景包括但不限于以下几种：

1、配置管理：通过 ConfigMap，可以将应用程序所需要的配置信息独立出来，让不同的应用程序可以共用同一个 ConfigMap，避免了重复的配置信息。

2、环境变量管理：可以将容器中的环境变量集中管理，便于维护和修改，同时可以避免暴露敏感信息。

3、容器镜像版本管理：将容器镜像使用的配置文件打包到 ConfigMap 中，每次升级镜像时都可以使用相应版本的 ConfigMap。

4、多节点部署：在多个节点上部署相同的应用程序时，可以使用 ConfigMap 来传递配置信息，避免重复配置。

5、动态配置：应用程序可以通过监控 ConfigMap 中的配置变化，自动更新配置信息，实现动态配置。

总之，ConfigMap 是 Kubernetes 集群中非常重要的一种资源对象，可以为应用程序提供方便的配置管理。

1.4、ConfigMap 局限性

ConfigMap 虽然在 Kubernetes 集群中非常实用，但是也存在一些局限性，主要包括以下几点：

1、不支持敏感数据：ConfigMap 主要用于存储非敏感的配置数据，例如环境变量、命令行参数、配置文件等，如果需要存储敏感数据，应该使用 Secret 对象。

2、容量限制：ConfigMap 在设计上不是用来保存大量数据的。ConfigMap 对象的大小存在一定的限制，单个 ConfigMap 最大可容纳数据为 1MB。当需要存储的数据量超过 1MB 时，需要拆分成多个 ConfigMap 或选择其他存储方式，可以考虑挂载存储卷或者使用独立的数据库或者文件服务。

2、ConfigMap 定义详解

ConfigMap 资源可以通过如下命令查看相关语法：

```
[root@k8s-master01 ~]# kubectl explain ConfigMap
```

- ConfigMap 资源说明

属性名称	取值类型	取值说明
apiVersion	string	Api 版本
kind	string	资源类型
metadata	Object	元数据
metadata.name	String	控制器的名称
metadata.namespace	String	控制器所属的命名空间，默认值为 default
metadata.labels[]	List	自定义标签列表
binaryData	map[string]string	ConfigMap.binaryData 是 ConfigMap 对象中用于存储二进制数据的字段。它被设计用来存储如证书、密钥等二进制数据。 使用 ConfigMap.binaryData 字段，开发人员可以使用 base64 编码将二进制数据存储为字符串，然后将其存储在 ConfigMap 对象中。在使用时，可以通过将 ConfigMap 挂载为卷并将其解码来访问二进制数据。
data	map[string]string	用于存储以键值对形式的配置数据。ConfigMap.data 是一个字典，其中的每个键值对都对应着一个配置数据项。

immutable	boolean	ConfigMap.immutable 是 Kubernetes 中 ConfigMap 对象的一个字段，它用于标识 ConfigMap 是否为不可变的。如果将 immutable 设置为 true，那么就无法对 ConfigMap 进行任何修改。在这种情况下，只有删除 ConfigMap 并重新创建它才能更新其中的数据。在 Kubernetes 中，ConfigMap 对象通常在部署应用程序时用于存储配置信息。使用 immutable 字段可以确保这些配置信息不会在应用程序运行期间被意外更改，从而增加系统的安全性和稳定性。但是，需要注意的是，将 immutable 设置为 true 以后，将无法通过 ConfigMap 更新应用程序的配置信息，这可能会导致一些不便。

3、Configmap 创建方法

创建 ConfigMap 有以下两种方式：

1、通过 YAML 配置文件方式创建

2、通过 kubectl 命令行方式创建

3.1、通过 YAML 配置文件方式创建（推荐）

● 实例 1：将应用所需的变量定义为 ConfigMap 的用法

（1）创建资源清单文件

```
[root@k8s-master01 ~]# vi configmap-appvars.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cm-appvars
data:
  apploglevel: info
  appdatadir: /var/data
```

对上面的 yaml 文件说明：

这是一个 Kubernetes 中的 ConfigMap 对象，它用于存储应用程序的相关配置信息。具体来说，它包括两个键值对：

● apploglevel：表示应用程序的日志级别，这里设置为 info，表示只记录一些重要的信息。

● appdatadir：表示应用程序的数据目录，这里设置为/var/data，表示应用程序将数据存储在这个目录中。

需要说明的是，这些配置信息可以在 Kubernetes 中被其他对象（比如 Deployment）引用，并用于配置应用程序的不同方面。例如，Deployment 可以将这些配置信息通过环境变量的方式注入到应用程序的容器中，以实现灵活的配置管理。

（2）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f configmap-appvars.yaml
```

```
    configmap/cm-appvars created
```

（3）查看创建好的 configmap

```
1、查看 configmap 资源
    [root@k8s-master01 ~]# kubectl get configmap cm-appvars
    NAME         DATA   AGE
    cm-appvars   2      22s

2、查看 cm-appvars configmap 资源的详细信息
    [root@k8s-master01 ~]# kubectl describe configmap cm-appvars
    Name:         cm-appvars
    Namespace:    default
    Labels:       <none>
    Annotations:  <none>

    Data
    ====
    appdatadir:
    ----
    /var/data
    apploglevel:
    ----
    info

    BinaryData
    ====

    Events:  <none>
```

● 实例 2：将应用所需的配置文件定义为 ConfigMap 的用法

configmap-appconfigfiles.yaml 描述了将配置文件 nginx.conf 定义为 ConfigMap 的用法，设置 key 为配置文件的别名，value 则是配置文件的全部文本内容：

（1）创建资源清单文件

```
    [root@k8s-master01 ~]# vi configmap-appconfigfiles.yaml
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: cm-appconfigfiles
    data:
      nginx.conf: |
```

```
        user nginx;
        worker_processes auto;
        error_log /var/log/nginx/error.log;
        pid /run/nginx.pid;

        # Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
        include /usr/share/nginx/modules/*.conf;

        events {
            worker_connections 1024;
        }

        http {
            log_format    main    '$remote_addr  -  $remote_user  [$time_local]
"$request" '
                              '$status $body_bytes_sent "$http_referer" '
                              '"$http_user_agent" "$http_x_forwarded_for"';

            access_log  /var/log/nginx/access.log  main;

            sendfile            on;
            tcp_nopush          on;
            tcp_nodelay         on;
            keepalive_timeout   65;
            types_hash_max_size 4096;

            include             /etc/nginx/mime.types;
            default_type        application/octet-stream;

            include /etc/nginx/conf.d/*.conf;

            server {
                listen       80;
                listen       [::]:80;
                server_name  _;
                root         /usr/share/nginx/html;

                # Load configuration files for the default server block.
                include /etc/nginx/default.d/*.conf;

                error_page 404 /404.html;
                location = /404.html {
```

```
                }

                error_page 500 502 503 504 /50x.html;
                location = /50x.html {
                }
            }
        }
```

对上面的 yaml 文件说明：

这是一个 Kubernetes 中的 ConfigMap 对象，用于存储 nginx 的配置文件。它包括以下信息：

- 名称为 cm-appconfigfiles；
- 存储的文本为 nginx.conf；
- nginx.conf 包含了 nginx 的详细配置

需要说明的是，这个 ConfigMap 对象中包含了一个文本文件，用 | 符号进行了缩进，可以保持文件原有的格式。在 Kubernetes 中，这个 ConfigMap 对象可以被其他对象（比如 Deployment）引用，并用于配置 nginx 的不同方面。

（2）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f configmap-appconfigfiles.yaml
configmap/cm-appconfigfiles created
```

（3）检查创建好的 configmap 资源

```
1、查看 configmap 资源
    [root@k8s-master01 ~]# kubectl get configmap cm-appconfigfiles
    NAME                DATA   AGE
    cm-appconfigfiles   1      12m

2、查看 cm-appconfigfiles configmap 资源的详细信息
    [root@k8s-master01 ~]# kubectl describe configmap cm-appconfigfiles
    Name:         cm-appconfigfiles
    Namespace:    default
    Labels:       <none>
    Annotations:  <none>

    Data
    ====
    nginx.conf:
    ----
    user nginx;
    worker_processes auto;
    error_log /var/log/nginx/error.log;
```

```
    pid /run/nginx.pid;

    # Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
    include /usr/share/nginx/modules/*.conf;

    events {
        worker_connections 1024;
    }

    http {
        log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                          '$status $body_bytes_sent "$http_referer" '
                          '"$http_user_agent" "$http_x_forwarded_for"';

        access_log  /var/log/nginx/access.log  main;

        sendfile            on;
        tcp_nopush          on;
        tcp_nodelay         on;
        keepalive_timeout   65;
        types_hash_max_size 4096;

        include             /etc/nginx/mime.types;
        default_type        application/octet-stream;

        include /etc/nginx/conf.d/*.conf;

        server {
            listen       80;
            listen       [::]:80;
            server_name  _;
            root         /usr/share/nginx/html;

            # Load configuration files for the default server block.
            include /etc/nginx/default.d/*.conf;

            error_page 404 /404.html;
            location = /404.html {
            }

            error_page 500 502 503 504 /50x.html;
            location = /50x.html {
```

```
            }
        }
    }


    BinaryData
    ====

    Events:  <none>
```

3.2、通过 kubectl 命令行方式创建（不推荐）

不使用 YAML 文件，直接通过 kubectl create configmap 命令也可以创建 ConfigMap 资源，可以使用参数--from-file 或--from-literal 指定内容，并且可以在一行命令中指定多个参数。

● kubectl 命令行创建 ConfigMap 语法

```
    1、通过--from-file 参数从文件中进行创建，可以指定 key 的名称
    kubectl create configmap NAME --from-file=[key=]source_file
--from-file=[key=]source_file ......

    2、通过--from-file 参数从目录中进行创建，该目录下的每个配置文件名都被设置为
key，文件的内容被设置为 value，语法为：
    kubectl create configmap NAME --from-file=source_file

    3、使用--from-literal 时会从文本中进行创建，直接将指定的 key#=value#创建为
ConfigMap 的内容，语法为：
    kubectl create configmap NAME --from-literal=key1=value1
--from-literal=key2=value2 ......
```

● 实战演练

实例 1：指定文件创建 configmap，文件名为 key，内容为 value

```
（1）准备配置文件
    [root@k8s-master01 ~]# mkdir conf
    [root@k8s-master01 ~]# echo "info" > conf/apploglevel

（2）创建 configmap
    [root@k8s-master01 ~]# kubectl create configmap apploglevel
--from-file=conf/apploglevel
    configmap/apploglevel created

（3）查看 configmap 资源
```

```
[root@k8s-master01 ~]# kubectl get configmap apploglevel
NAME          DATA   AGE
apploglevel   1      2m28s

(4) 查看 apploglevel configmap 资源的详细信息
[root@k8s-master01 ~]# kubectl describe configmap apploglevel
Name:         apploglevel
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
apploglevel:
----
info


BinaryData
====

Events:  <none>
```

实例 2：指定文件创建 configmap，自定义 key，文件内容为 value

```
(1) 准备配置文件
[root@k8s-master01 ~]# mkdir conf
[root@k8s-master01 ~]# echo "/var/data" > conf/appdatadir

(2) 创建 configmap
[root@k8s-master01 ~]# kubectl create configmap data
--from-file=data=conf/appdatadir
configmap/data created

(3) 查看 configmap 资源
[root@k8s-master01 ~]# kubectl get configmap data
NAME   DATA   AGE
data   1      24s

(4) 查看 data configmap 资源的详细信息
[root@k8s-master01 ~]# kubectl describe configmap data
Name:         data
Namespace:    default
```

```
Labels:       <none>
Annotations:  <none>

Data
====
data:
----
/var/data


BinaryData
====

Events:  <none>
```

实例 3：指定目录创建 configmap

```
（1）查看目录下的文件
[root@k8s-master01 ~]# ll conf
total 8
-rw-r--r-- 1 root root 10 Jun  8 23:44 appdatadir
-rw-r--r-- 1 root root  5 Jun  8 23:26 apploglevel

（2）指定目录创建 configmap
[root@k8s-master01 ~]# kubectl create configmap app-config --from-file=conf
configmap/app-config created

（3）查看 configmap 资源
[root@k8s-master01 ~]# kubectl get configmap app-config
NAME         DATA   AGE
app-config   2      43s

（4）查看 app-config configmap 资源的详细信息
[root@k8s-master01 ~]# kubectl describe configmap app-config
Name:         app-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
appdatadir:
----
```

```
/var/data

apploglevel:
----
info


BinaryData
====

Events:  <none>
```

4、在 Pod 中使用 ConfigMap

容器应用对 ConfigMap 的使用有以下两种方法。

1、通过环境变量获取 ConfigMap 中的内容。

2、通过 Volume 挂载的方式将 ConfigMap 中的内容挂载为容器内部的文件或目录。

4.1、通过变量方式使用 ConfigMap

在 Kubernetes 中，可以通过环境变量的方式使用 ConfigMap，并将其中的配置数据传递给容器。这种使用方式有以下优点：

1、方便配置管理：配置信息被集中保存在 ConfigMap 中，通过环境变量组合成一个“配置模板”，不需要修改容器镜像或创建多个更新的 YAML 文件。这使得配置管理更加方便，可以在不同的环境中使用相同的容器镜像，而只需要在不同的 ConfigMap 中保存不同的配置信息即可。

2、安全性高：使用环境变量传递 ConfigMap 中的配置数据，可以避免在容器启动时直接将配置文件放置在容器内，从而避免了敏感信息的泄露。

3、配置的复用：多个容器可以使用同一个 ConfigMap 中的配置信息，这种方式不仅避免了重复的配置信息，而且可以保证容器中的配置信息一致。

4、与现有系统集成：通过使用环境变量的方式，可以与现有的程序或脚本进行集成，提高了系统之间的兼容性。

综上所述，使用环境变量的方式使用 ConfigMap 可以方便地管理和传递配置信息，提高应用程序的灵活性和可扩展性，并提高了安全性和可用性。

实战：定义 configmap，创建 pod 以环境变量的方式使用 configmap。

（1）创建 ConfigMap

```
1、创建 configmap 资源清单文件
    [root@k8s-master01 ~]# vi cm-appvars.yaml
    apiVersion: v1
```

```
  kind: ConfigMap
  metadata:
    name: cm-appvars
  data:
    apploglevel: info
    appdatadir: /var/data
```

2、更新资源清单文件

```
  [root@k8s-master01 ~]# kubectl apply -f cm-appvars.yaml
  configmap/cm-appvars created
```

（2）创建 Pod 并使用 ConfigMap，创建资源清单文件

```
  [root@k8s-master01 ~]# vi cm-appvars-pod.yaml
  apiVersion: v1
  kind: Pod
  metadata:
    name: cm-appvars-pod
  spec:
    containers:
    - name: cm-appvars-pod
      image: nginx:latest
      env:
      - name: APPLOGLEVEL
        valueFrom:
          configMapKeyRef:
            name: cm-appvars
            key: apploglevel
      - name: APPDATADIR
        valueFrom:
          configMapKeyRef:
            name: cm-appvars
            key: appdatadir
```

对上面的 yaml 文件说明：

这是一个 Kubernetes 中的 Pod 对象，用于创建一个运行 nginx 容器的 Pod。它包括以下信息：

- 名称为 cm-appvars-pod。
- 它只有一个容器，名称也为 cm-appvars-pod。
- 容器镜像为 nginx:latest，使用的是最新的 nginx 镜像。
- 容器中定义了两个环境变量：APPLOGLEVEL 和 APPDATADIR，分别从 configMap 中获取。
- APPLOGLEVEL 的值来自于名为 cm-appvars 的 ConfigMap 对象中的 apploglevel 属性。

- APPDATADIR 的值来自于名为 cm-appvars 的 ConfigMap 对象中的 appdatadir 属性。

这个配置文件的作用是在容器启动时，从 ConfigMap 中获取一些配置参数，并将它们作为环境变量传递给容器。在容器内部，可以通过读取环境变量来获取这些参数并进行相应的配置。

（3）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f cm-appvars-pod.yaml
pod/cm-appvars-pod created
```

（4）测试

```
1、查看 pod
    [root@k8s-master01 ~]# kubectl get pods
    NAME             READY   STATUS    RESTARTS   AGE
    cm-appvars-pod   1/1     Running   0          58s

2、进入 pod，查看环境变量
    [root@k8s-master01 ~]# kubectl exec -it cm-appvars-pod -- bash
    root@cm-appvars-pod:/# env | egrep "APPDATADIR|APPLOGLEVEL"
    APPDATADIR=/var/data
    APPLOGLEVEL=info
```

从上面可以看出容器内部的环境变量使用 ConfigMap cm-appvars 中的值进行了正确设置。

4.2、通过 volumeMount 使用 ConfigMap

在 Kubernetes 中，还可以通过 VolumeMount 的方式使用 ConfigMap，将其中的配置数据挂载到容器的文件系统中。这种使用方式有以下优点：

1、灵活性更高：与环境变量相比，VolumeMount 可以将 ConfigMap 中的配置信息整体映射到容器的文件系统中，容器内的应用可以像使用本地文件一样使用配置信息。这样做的好处是应用程序获取配置更灵活，可以方便地支持各种配置文件格式和读取方式。

2、可读性更强：由于 VolumeMount 将 ConfigMap 中的配置作为本地文件来处理，因此可以利用容器文件系统提供的一些常规操作来快速获取和查看配置信息。

3、配置的复用：多个容器可以使用同一个 ConfigMap 中的配置信息，这种方式不仅避免了重复的配置信息，而且可以保证容器中的配置信息一致。

4、更新更加方便：在使用 VolumeMount 方式下进行的 ConfigMap 更新，会自动同步到使用该 ConfigMap 的所有 Pod 上，这样就不需要手动进行 Pod 重启等操作，提高了应用程序的可用性。

5、安全性高：使用 VolumeMount 的方式，可以避免在容器启动时直接将配置文件放置在容器内，从而避免了敏感信息的泄露。

综上所述，通过 VolumeMount 使用 ConfigMap 可以提高灵活性、可读性、便捷性、可复用性和安全性，是一个更加全面的获取配置信息的方式。

实战：通过 volumeMount 使用 ConfigMap

（1）创建 ConfigMap

1、创建 configmap 资源清单文件

```
[root@k8s-master01 ~]# vi cm-appconfigfiles.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cm-appconfigfiles
data:
  key-serverxml: |
    <?xml version='1.0' encoding='utf-8'?>
    <Server port="8005" shutdown="SHUTDOWN">
      <Listener
className="org.apache.catalina.startup.VersionLoggerListener" />
      <Listener   className="org.apache.catalina.core.AprLifecycleListener"
SSLEngine="on" />
      <Listener
className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
      <Listener
className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
      <Listener
className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
      <GlobalNamingResources>
        <Resource name="UserDatabase" auth="Container"
                  type="org.apache.catalina.UserDatabase"
                  description="User database that can be updated and saved"

factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
                  pathname="conf/tomcat-users.xml" />
      </GlobalNamingResources>

      <Service name="Catalina">
        <Connector port="8080" protocol="HTTP/1.1"
                   connectionTimeout="20000"
                   redirectPort="8443" />
        <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
        <Engine name="Catalina" defaultHost="localhost">
          <Realm className="org.apache.catalina.realm.LockOutRealm">
```

```
                <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                       resourceName="UserDatabase"/>
              </Realm>
              <Host name="localhost"  appBase="webapps"
                    unpackWARs="true" autoDeploy="true">
                <Valve    className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
                       prefix="localhost_access_log" suffix=".txt"
                       pattern="%h %l %u %t &quot;%r&quot; %s %b" />

              </Host>
            </Engine>
          </Service>
        </Server>
      key-loggingproperties: "handlers
        =                                  1catalina.org.apache.juli.FileHandler,
2localhost.org.apache.juli.FileHandler,
        3manager.org.apache.juli.FileHandler,
4host-manager.org.apache.juli.FileHandler,
        java.util.logging.ConsoleHandler\r\n\r\n.handlers                    =
1catalina.org.apache.juli.FileHandler,

java.util.logging.ConsoleHandler\r\n\r\n1catalina.org.apache.juli.FileHandler.l
evel
        =       FINE\r\n1catalina.org.apache.juli.FileHandler.directory       =
${catalina.base}/logs\r\n1catalina.org.apache.juli.FileHandler.prefix
        =    catalina.\r\n\r\n2localhost.org.apache.juli.FileHandler.level    =
FINE\r\n2localhost.org.apache.juli.FileHandler.directory
        =
${catalina.base}/logs\r\n2localhost.org.apache.juli.FileHandler.prefix        =
localhost.\r\n\r\n3manager.org.apache.juli.FileHandler.level
        =       FINE\r\n3manager.org.apache.juli.FileHandler.directory        =
${catalina.base}/logs\r\n3manager.org.apache.juli.FileHandler.prefix
        =   manager.\r\n\r\n4host-manager.org.apache.juli.FileHandler.level   =
FINE\r\n4host-manager.org.apache.juli.FileHandler.directory
        =
${catalina.base}/logs\r\n4host-manager.org.apache.juli.FileHandler.prefix =
        host-manager.\r\n\r\njava.util.logging.ConsoleHandler.level           =
FINE\r\njava.util.logging.ConsoleHandler.formatter
        =
java.util.logging.SimpleFormatter\r\n\r\n\r\norg.apache.catalina.core.Container
Base.[Catalina].[localhost].level
```

```
        =
INFO\r\norg.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers
        =
2localhost.org.apache.juli.FileHandler\r\n\r\norg.apache.catalina.core.Containe
rBase.[Catalina].[localhost].[/manager].level
        =
INFO\r\norg.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager]
.handlers
        =
3manager.org.apache.juli.FileHandler\r\n\r\norg.apache.catalina.core.ContainerB
ase.[Catalina].[localhost].[/host-manager].level
        =
INFO\r\norg.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/host-ma
nager].handlers
        = 4host-manager.org.apache.juli.FileHandler\r\n\r\n"

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f cm-appconfigfiles.yaml
    configmap/cm-appconfigfiles created
```

（2）创建 Pod 时使用 item 指定挂载 configmap 中的 key

当 configmap 资源中定义了多个 key 时，可以在创建 pod 时使用 item 指定 configmap 中的 key，挂载我们需要的配置文件。

```
1、创建 pod 的资源清单文件
    [root@k8s-master01 ~]# vi cm-pod-1.yaml
    apiVersion: v1
    kind: Pod
    metadata:
      name: cm-pod-1
    spec:
      containers:
      - name: cm-test-app
        image: kubeguide/tomcat-app:v1
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: serverxml
          mountPath: /configfiles
      volumes:
      - name: serverxml
        configMap:
          name: cm-appconfigfiles
```

```
        items:
        - key: key-serverxml
          path: server.xml
        - key: key-loggingproperties
          path: logging.properties
```

对上面的 yaml 文件说明如下：

```
  apiVersion: v1
  kind: Pod
  metadata:
    name: cm-pod-1
  spec:
    containers:
    - name: cm-test-app
      image: kubeguide/tomcat-app:v1
      ports:
      - containerPort: 8080
      volumeMounts:
      - name: serverxml    #指定 volume 的名称
        mountPath: /configfiles    #将 volume 定义的内容，挂载到/configfiles 目录下
    volumes:
    - name: serverxml    #定义 volume 的名称为 serverxml
      configMap:
        name: cm-appconfigfiles    #使用 ConfigMap“cm-appconfigfiles”
        items:
        - key: key-serverxml    #指定 ConfigMap“cm-appconfigfiles”的 key 为“key-serverxml”，将此 key 的 value 挂载到 pod 内。
          path: server.xml    #当挂载到容器内时，叫什么文件名
        - key: key-loggingproperties  #指定 ConfigMap“cm-appconfigfiles”的 key 为“key-loggingproperties”，将此 key 的 value 挂载到 pod 内。
          path: logging.properties    #当挂载到容器内时，叫什么文件名
```

2、创建 pod

```
[root@k8s-master01 ~]# kubectl apply -f cm-pod-1.yaml
pod/cm-pod-1 created
```

3、查看 pod

```
[root@k8s-master01 ~]# kubectl get pods
NAME       READY   STATUS    RESTARTS   AGE
cm-pod-1   1/1     Running   0          3m34s
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
4、进入容器内查看挂载的内容是否存在
    [root@k8s-master01 ~]# kubectl exec -it cm-pod-1 -- bash
    root@cm-pod-1:/usr/local/tomcat# ls /configfiles/
    logging.properties  server.xml
```

（3）创建 Pod 时不使用 item 指定挂载 configmap 中的 key

在创建 pod 时如果不使用 item 指定 configmap 中的 key，则当挂载到容器内时，会将 configmap 中的 key 作为文件名、value 作为文件内容，全部挂载到容器内指的目录下。

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi cm-pod-2.yaml
    apiVersion: v1
    kind: Pod
    metadata:
      name: cm-pod-2
    spec:
      containers:
      - name: cm-test-app
        image: kubeguide/tomcat-app:v1
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: serverxml
          mountPath: /configfiles
      volumes:
      - name: serverxml
        configMap:
          name: cm-appconfigfiles

对上面的 yaml 文件说明如下：
    [root@k8s-master01 ~]# vi cm-pod-2.yaml
    apiVersion: v1
    kind: Pod
    metadata:
      name: cm-pod-2
    spec:
      containers:
      - name: cm-test-app
        image: kubeguide/tomcat-app:v1
        ports:
        - containerPort: 8080
        volumeMounts:
```

```
        - name: serverxml    #引用 Volume 的名称
          mountPath: /configfiles    #挂载到容器内的目录
      volumes:
      - name: serverxml    #定义 Volume 的名称
        configMap:
          name: cm-appconfigfiles  #使用 ConfigMap “cm-appconfigfiles”

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f cm-pod-2.yaml
    pod/cm-pod-2 created

3、查看 pod
    [root@k8s-master01 ~]# kubectl get pods cm-pod-2
    NAME       READY   STATUS    RESTARTS   AGE
    cm-pod-2   1/1     Running   0          59s

4、进入容器内查看挂载的内容是否存在
    [root@k8s-master01 ~]# kubectl exec -it cm-pod-2 -- bash
    root@cm-pod-2:/usr/local/tomcat# ls /configfiles/
    key-loggingproperties  key-serverxml
```

4.3、使用 ConfigMap 的限制条件

使用 ConfigMap 的限制条件如下：

1、Pod 要挂载使用 ConfigMap 资源，那么 ConfigMap 必须在创建 Pod 之前创建好。否则，describe 查看 pod 信息会看到没有找到对应的 configmap 资源，会导致容器一直处于创建中。

2、ConfigMap 受 Namespace 限制，只有处于相同 Namespace 中的 Pod 才可以引用它。

3、kubelet 只支持可以被 API Server 管理的 Pod 使用 ConfigMap。由 kubelet 管理并自动创建的静态 Pod 是无法引用 ConfigMap 资源的。

5、Configmap 热更新

5.1、Configmap 唯一会热更新动作（mountPath）

ConfigMap 是 Kubernetes 中的一种对象，用于存储应用程序的配置信息。ConfigMap 可以通过热更新来动态修改应用程序的配置，而无需重启应用程序容器。

（1）创建并更新 configmap 资源清单文件

```
1、创建资源清单文件
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 ~]# vi mysql-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql
  labels:
    app: mysql
data:
    log: "1"
    lower: "1"
    my.cnf: |
      [mysqld]
      server-id=1

2、更新资源清单文件
[root@k8s-master01 ~]# kubectl apply -f mysql-configmap.yaml
configmap/mysql created
```

（2）挂载到 pod 中做使用

```
1、创建资源清单文件
[root@k8s-master01 yaml]# vi mysql-pod-volume.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-pod-volume
spec:
  containers:
  - name: mysql
    image: busybox
    command: [ "/bin/sh","-c","sleep 3600" ]
    volumeMounts:
    - name: mysql-config
      mountPath: /tmp/config
  volumes:
  - name: mysql-config
    configMap:
      name: mysql
  restartPolicy: Never

2、更新资源清单文件
[root@k8s-master01 ~]# kubectl apply -f mysql-pod-volume.yaml
pod/mysql-pod-volume created
```

```
3、进入容器，查看挂载的文件
    [root@k8s-master01 ~]# kubectl exec -it mysql-pod-volume -- /bin/sh
    / # cd /tmp/config/
    /tmp/config # ls
    log     lower   my.cnf
    /tmp/config # cat log
    1
```

（3）下面使用热更新，把 log 文件更新为 2

```
1、修改 configmap 资源清单文件
    [root@k8s-master01 ~]# vi mysql-configmap.yaml
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: mysql
      labels:
        app: mysql
    data:
        log: "2"
        lower: "3"
        my.cnf: |
          [mysqld]
          server-id=4

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f mysql-configmap.yaml
    configmap/mysql configured

3、进入容器，查看文件内容是否更新
    [root@k8s-master01 ~]# kubectl exec -it mysql-pod-volume -- /bin/sh
    /tmp/config # cat log
    2

    /tmp/config # cat lower
    3

    /tmp/config # cat my.cnf
    [mysqld]
    server-id=4
```

从上面可以看出更新 configmap 资源时，等待大约 10s 左右，会将更新的数据同步给所有使用到此 configmap 资源的 pod。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

注意事项：

1、更新 ConfigMap 后，使用 ConfigMap 挂载的 Env（变量）不会同步更新。

2、如果使用 subPath 进行挂载，不会同步更新。如下：

```
        volumeMounts:
        - name: mysql-config
          mountPath: /tmp/config/my.cnf
          subPath: my.cnf
      volumes:
      - name: mysql-config
        configMap:
          name: mysql
          items:
          - key: my.cnf
            path: my.cnf
```

如果我们使用 subPath，configmap 更新之后 pod 也不会更新。

5.2、使用 subPath 挂载 configmap（测试是否会热更新）

使用 Kubernetes 的 subPath 挂载 ConfigMap 有以下好处：

1、灵活控制文件和目录的使用：在 Kubernetes 中，使用 subPath 可以将 ConfigMap 和 Secret 作为文件挂载到容器中，而不是将整个文件的内容直接挂载到 Pod 的目录中。这样可以更灵活地控制 Pod 中使用哪些文件和目录。

2、避免覆盖原有文件：当使用 subPath 时，volume 会支持将 ConfigMap/Secret 挂载到容器的路径，但不会覆盖容器路径下原有的文件。这样可以避免覆盖原有目录下的文件。

下面测试 subPath 挂载 configmap，测试是否会触发热更新。

（1）创建环境

```
1、创建 cm
    [root@k8s-master01 ~]# vi mysql-configmap.yaml
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: mysql
      labels:
        app: mysql
    data:
        log: "1"
        lower: "1"
        my.cnf: |
```

```
        [mysqld]
        server-id=1

[root@k8s-master01 ~]# kubectl apply -f mysql-configmap.yaml
configmap/mysql created
```

2、创建 pod

```
[root@k8s-master01 ~]# vi mysql-pod-volume.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-pod-volume
spec:
  containers:
  - name: mysql
    image: busybox
    command: [ "/bin/sh","-c","sleep 3600" ]
    volumeMounts:
    - name: mysql-config
      mountPath: /tmp/config/my.cnf
      subPath: my-test.cnf
    - name: mysql-config
      mountPath: /tmp/config/log
      subPath: log-test
    - name: mysql-config
      mountPath: /tmp/config/lower
      subPath: lower-test
  volumes:
  - name: mysql-config
    configMap:
      name: mysql
      items:
      - key: my.cnf
        path: my-test.cnf
      - key: log
        path: log-test
      - key: lower
        path: lower-test
  restartPolicy: Never
```

对上面的资源清单文件说明：

```
    volumeMounts:
```

```
        - name: mysql-config  #使用的卷名称
          mountPath: /tmp/config/my.cnf  #卷在容器中的挂载路径
          subPath: my-test.cnf   #path 定义的名称。
      volumes:
      - name: mysql-config  #卷的名称
        configMap:
          name: mysql  #ConfigMap 的名称
          items:
          - key: my.cnf  #这个是从 ConfigMap 中提取的项的键
            path: my-test.cnf  #这个名称要与 subPath 名称关联起来。

    [root@k8s-master01 ~]# kubectl apply -f mysql-pod-volume.yaml
    pod/mysql-pod-volume created

    [root@k8s-master01 ~]# kubectl exec -it mysql-pod-volume -- sh
    / # ls /tmp/config
    log     lower   my.cnf
    / # cat /tmp/config/log
    1
    / # cat /tmp/config/lower
    1
    / # cat /tmp/config/my.cnf
    [mysqld]
    server-id=1
```

（2）测试热更新

```
1、更新 configmap
    [root@k8s-master01 ~]# sed -i 's#log: "1"#log: "2"#g' mysql-configmap.yaml
    [root@k8s-master01 ~]# sed -i 's#lower: "1"#lower: "2"#g' mysql-configmap.yaml
    [root@k8s-master01     ~]#     sed     -i     's#server-id=1#server-id=2#g'
mysql-configmap.yaml

    [root@k8s-master01 ~]# kubectl apply -f mysql-configmap.yaml
    configmap/mysql configured

2、等待 1 分钟过去后，也没有热更新
    [root@k8s-master01 ~]# kubectl exec -it mysql-pod-volume -- sh
    / # cd /tmp/config/
    /tmp/config # cat log
    1
    /tmp/config # cat lower
    1
```

```
/tmp/config # cat my.cnf
[mysqld]
server-id=1
```

5.3、Reloader 实现 ConfigMap/Secret 热更新

- 什么是 Reloader

Reloader 是一个用于实现 Kubernetes ConfigMap/Secret 热更新的工具。它可以帮助开发人员在运行时动态更新 configMap，而无需重新启动应用程序或重新部署资源。

ConfigMap 是 Kubernetes 中的一种资源，用于存储配置数据，以便在容器中注入环境变量或配置文件。在传统的部署方式中，如果需要更改 ConfigMap 的值，开发人员需要重新创建并重新部署相关的资源。这不仅耗时，而且在频繁更改配置的情况下很不实用。

Reloader 的出现解决了这个问题。它通过监听 ConfigMap/Secret 的更改，自动重新加载应用程序，使其能够即时获取最新的配置信息。这样，开发人员可以实时测试和验证配置更改，而无需经历繁琐的部署过程。

Reloader 项目地址：

https://github.com/stakater/Reloader

- Reloader 的工作原理

1、安装和配置 Reloader

首先，需要在 Kubernetes 集群中安装和配置 Reloader。

2、监听 ConfigMap 更改

一旦 Reloader 部署并启动，它会开始监听 ConfigMap/Secret 资源。当资源发生变化时，Reloader 会捕获这些更改并触发重新加载操作。

3、重新加载应用程序

当 ConfigMap/Secret 发生更改时，Reloader 会自动重新加载应用程序，以便它能够使用最新的配置数据。这可以通过重启容器、使用热重载机制或使用其他适当的机制来完成。

4、验证和测试

在应用程序重新加载后，开发人员可以验证配置是否已成功更新，并进行必要的测试以确保一切正常工作。

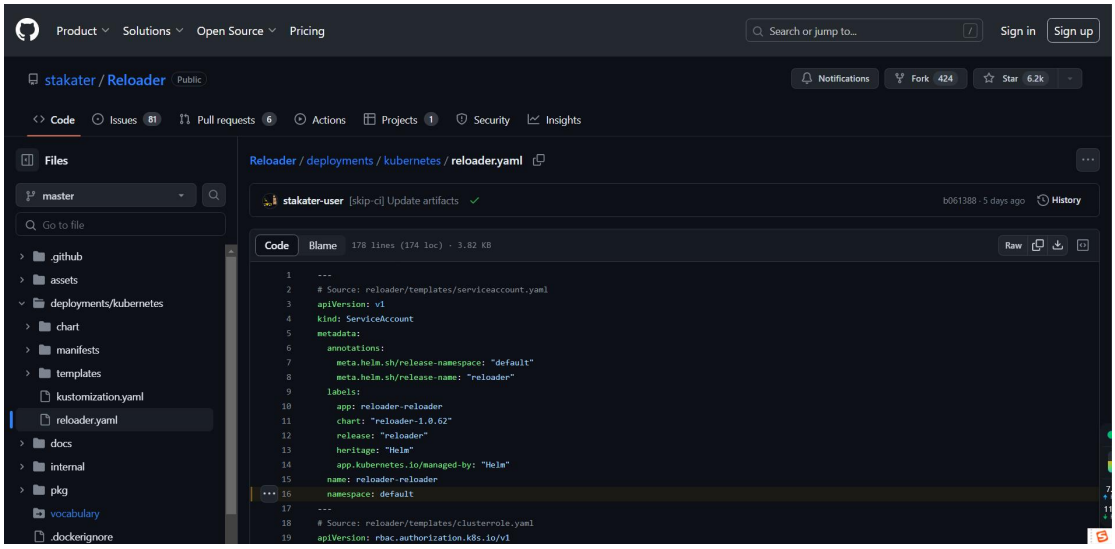
总之，Reloader 是一个方便的工具，可以帮助开发人员在 Kubernetes 中实现 ConfigMap/Secret 的热更新。它简化了配置管理的过程，并允许开发人员快速测试和验证配置更改，从而提高开发效率和灵活性。

Reloader 可以自动监控 ConfigMap/Secrret 资源的变化来触发对应的 deployment/statefulset/daemonset 的滚动更新。

● Reloader 部署

部署参考 yaml 文件：

https://github.com/stakater/Reloader/blob/master/deployments/kubernetes/reloader.yaml



部署 Reloader：

1、下载 yaml 文件

```
[root@k8s-master01 ~]# wget https://github.com/stakater/Reloader/blob/master/deployments/kubernetes/reloader.yaml
```

2、修改 yaml 文件中的 image

```
[root@k8s-master01 ~]# cat reloader.yaml | grep image:
        - image: "stakater/reloader:v1.0.62"
```

3、为所有的资源添加

3、部署 Reloader

```
[root@k8s-master01 ~]# kubectl apply -f reloader.yaml
```

4、查看 pod

```
[root@k8s-master01 ~]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
reloader-reloader-547fff497-xq68v   1/1     Running   0          11s
```

● Reloader 实战（实现 configmap 资源热更新）

（1）创建 confgmap，创建 deployment 资源使用 configmap

```
1、创建 cm
    [root@k8s-master01 ~]# cat mysql-configmap.yaml
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: mysql
      labels:
        app: mysql
    data:
        my.cnf: |
          [mysqld]
          server-id=1

    [root@k8s-master01 ~]# kubectl apply -f mysql-configmap.yaml
    configmap/mysql created

2、创建 deployment
    [root@k8s-master01 test]# cat mysql-deployment.yaml
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: mysql
      annotations:
        reloader.stakater.com/auto: "true"
    spec:
      replicas: 2
      selector:
        matchLabels:
          app: mysql
      template:
        metadata:
          labels:
            app: mysql
        spec:
          containers:
          - name: mysql
            image: nginx:latest
            volumeMounts:
            - name: mysql-config
              mountPath: /tmp/config/my.cnf
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
            subPath: my-test.cnf
        volumes:
        - name: mysql-config
          configMap:
            name: mysql
            items:
            - key: my.cnf
              path: my-test.cnf
```

对上面 annotations 注解说明：

reloader.stakater.com/auto: "true"，只要部署挂载的 ConfigMap 或者 secret 有更新，则滚动更新。

```
[root@k8s-master01 test]# kubectl apply -f mysql-deployment.yaml
deployment.apps/mysql created
```

（2）扩展 annotations 注解

如果一个部署中有多个 ConfigMap 或者 Secret，那么只想指定的 ConfigMap 或者 Secret 更新的话才更新这个部署，可以指定名称，如下：

```
1、更新单个或多个 configmap 资源
    kind: Deployment
    metadata:
      annotations:
        configmap.reloader.stakater.com/reload: "a-configmap,b-configmap"
    spec:
      template:
        metadata:

2、更新单个或多个 secret 资源
    kind: Deployment
    metadata:
      annotations:
        secret.reloader.stakater.com/reload: "a-secret,b-secret"
    spec:
      template:
        metadata:
```

（3）更新 configmap

```
1、提前监听 pod 变动
    [root@k8s-master01 ~]# kubectl get pods -w

2、更新 configmap 资源
```

```
    [root@k8s-master01    test]#    sed    -i    's#server-id=1#server-id=2#g'
mysql-configmap.yaml
    [root@k8s-master01 test]# kubectl apply -f mysql-configmap.yaml
    configmap/mysql configured
```

3、分析 pod 变动过程

```
    [root@k8s-master01 ~]# kubectl get pods -w
```

```
[root@k8s-master01 ~]# kubectl get pods -w
NAME                                READY   STATUS    RESTARTS   AGE
mysql-596b687754-988j4              1/1     Running   0          5m50s
mysql-596b687754-fhv8w              1/1     Running   0          5m50s
reloader-reloader-547fff497-xq68v   1/1     Running   0          11m
mysql-56fcbbbd4-cc8x4               0/1     Pending   0          0s
mysql-56fcbbbd4-cc8x4               0/1     Pending   0          0s
mysql-56fcbbbd4-cc8x4               0/1     ContainerCreating   0          0s
mysql-56fcbbbd4-cc8x4               0/1     ContainerCreating   0          1s
mysql-56fcbbbd4-cc8x4               1/1     Running             0          17s
mysql-596b687754-988j4              1/1     Terminating         0          6m56s
mysql-56fcbbbd4-4vr2v               0/1     Pending             0          0s
mysql-56fcbbbd4-4vr2v               0/1     Pending             0          0s
mysql-56fcbbbd4-4vr2v               0/1     ContainerCreating   0          0s
mysql-596b687754-988j4              1/1     Terminating         0          6m56s
mysql-56fcbbbd4-4vr2v               0/1     ContainerCreating   0          1s
mysql-596b687754-988j4              0/1     Terminating         0          6m57s
mysql-596b687754-988j4              0/1     Terminating         0          6m57s
mysql-596b687754-988j4              0/1     Terminating         0          6m57s
mysql-56fcbbbd4-4vr2v               1/1     Running             0          17s
mysql-596b687754-fhv8w              1/1     Terminating         0          7m13s
mysql-596b687754-fhv8w              1/1     Terminating         0          7m13s
mysql-596b687754-fhv8w              0/1     Terminating         0          7m14s
mysql-596b687754-fhv8w              0/1     Terminating         0          7m14s
mysql-596b687754-fhv8w              0/1     Terminating         0          7m14s
```


（4）检查

```
    [root@k8s-master01 ~]# kubectl exec -it mysql-56fcbbbd4-4vr2v -- bash
    root@mysql-56fcbbbd4-4vr2v:/# cat /tmp/config/my.cnf
    [mysqld]
    server-id=2
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。