

本文由 [简悦 SimpRead](#) 转码，原文地址 www.imooc.com

在我们日常的开发中，肯定会涉及到多种环境，比如日常开发环境、预发环境、生产环境，我们将这些环境区分开一般都是通过配置，不同环境对应不同的配置。这是一种将代码和配置分离的思想，在 Kubernetes 中，就相当于将镜像和配置分离。

在 Kubernetes 中对于配置提供了两种对象：

- ConfigMap：普通配置存储。
- Secret：密文存储，比如数据库密码等。

下面我们就来看一下这两种配置的基本使用方法。

1. ConfigMap 创建

我们可以通过命令 `kubectl create ConfigMap <cm-name> <data-source>` 命令创建 ConfigMap，通过 `kubectl get ConfigMap <cm-name>` 查看指定的 ConfigMap 的内容。

通过目录创建

我们先在本地创建一个目录，然后将官方示例的两个配置文件

`https://kubernetes.io/examples/ConfigMap/game.properties` 和

`https://kubernetes.io/examples/ConfigMap/ui.properties` 下载到指定目录中。

```
$ mkdir ConfigMap
$ wget https://kubernetes.io/examples/ConfigMap/game.properties
$ wget https://kubernetes.io/examples/ConfigMap/ui.properties
```

我们可以简单看一下文件内容。

```
$ cat 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
```

下面开始创建 ConfigMap。

```
kubectl create ConfigMap game-config --from-file=/path-to-ConfigMap/ConfigMap/
```

查看 ConfigMap 内容如下。

```
$ kubectl get ConfigMap game-config -o yaml
apiVersion: v1
```

```
data:
  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
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: "2020-04-07T02:02:43Z"
  name: game-config
  namespace: default
  resourceVersion: "35384497"
  selfLink: /api/v1/namespaces/default/ConfigMaps/game-config
  uid: de85b232-7873-11ea-a328-00163e16aee6
```

通过文件创建

类似的我们也可以指定通过某个文件创建，只要将上面的最后一个参数的目录改成文件既可，可以连接多个 `--from-file` 参数。

```
$ kubectl create ConfigMap game-config-file --from-file=/path-to-
ConfigMap/ConfigMap/game.properties
```

然后查看该 ConfigMap。

```
$ kubectl get ConfigMap game-config-file -o yaml
apiVersion: v1
data:
  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
kind: ConfigMap
metadata:
  creationTimestamp: "2020-04-07T02:09:18Z"
  name: game-config-file
  namespace: default
  resourceVersion: "35385556"
  selfLink: /api/v1/namespaces/default/ConfigMaps/game-config-file
  uid: c9a01ad0-7874-11ea-a312-00163e16aa1b
```

通过环境变量文件

通过环境变量文件创建 ConfigMap 和通过文件创建的方式类似，将参数 `--from-file` 改成 `--from-env-file` 即可。还有一个重要的区别的是通过环境变量文件创建 ConfigMap 只能使用一个文件。下面是示例。

```
$ kubectl create ConfigMap game-config-env --from-env-file=/path-to-ConfigMap/ConfigMap/game.properties
```

我们再通过 `kubectl get` 获取 ConfigMap。

```
$ kubectl get ConfigMap game-config-env -o yaml
apiVersion: v1
data:
  enemies: aliens
  enemies.cheat: "true"
  enemies.cheat.level: noGoodRotten
  lives: "3"
  secret.code.allowed: "true"
  secret.code.lives: "30"
  secret.code.passphrase: UUDLRLRBABAS
kind: ConfigMap
metadata:
  creationTimestamp: "2020-04-07T15:32:45Z"
  name: game-config-env
  namespace: default
  resourceVersion: "35514135"
  selfLink: /api/v1/namespaces/default/ConfigMaps/game-config-env
  uid: 075dabd9-78e5-11ea-a312-00163e16aa1b
```

对比上面的通过文件创建 ConfigMap 的对象，可以看出来区别在于 data 字段。前面两种方式通过文件创建出来的 ConfigMap 有一个 key 对应到文件名，好处是我们可以通过多个配置文件创建 ConfigMap，然后在不同的配置文件中使使用相同名字的 key。

直接编写 ConfigMap

除了通过文件来创建 ConfigMap 外，我们也可以直接编写一个 ConfigMap 对象的 yaml 文件，然后通过 `kubectl apply` 去创建 ConfigMap。

2. ConfigMap 使用

我们使用 ConfigMap 主要可以通过两种方式来使用：

环境变量

通过环境变量的方式使用 ConfigMap，只能使用形如下面这样的 ConfigMap，也就是通过环境变量文件创建的 ConfigMap。

```
$ kubectl get ConfigMap game-config-env -o yaml
apiVersion: v1
data:
  enemies: aliens
  enemies.cheat: "true"
  enemies.cheat.level: noGoodRotten
  lives: "3"
  secret.code.allowed: "true"
  secret.code.lives: "30"
  secret.code.passphrase: UDDLRLRBABAS
```

我们使用的时候可以使用 ConfigMap 中指定的 key 或者 ConfigMap 中所有的 key，下面是示例。

使用指定的 key

下面的示例是引用 ConfigMap `game-config-env` 中的 key 为 `lives` 的变量，值为 3。

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod2
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
    env:
      - name: ENV_LIVES
        valueFrom:
          configMapKeyRef:
            name: game-config-env
            key: lives
```

我们还是通过 `kubectl apply`，然后通过 `kubectl exec` 登录到 Pod 里面查看环境变量。

```
$ kubectl exec -ti myapp-pod2 -n imooc -- sh
/
ENV_LIVES=3
```

使用所有的 key

除了像上面这种使用 ConfigMap 中某个特定的 key，还可以将 ConfigMap 中所有的 key 直接映射到容器内，下面是一个使用示例。

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod3
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
    envFrom:
    - configMapRef:
        name: game-config-env
```

同样的，我们也是通过 `kubectl apply` 创建 Pod，然后通过 `kubectl exec` 登录到 Pod 内部查看环境变量情况。

```
$ kubectl exec -ti myapp-pod3 -n imooc -- sh
/
MYSERVICE_SERVICE_HOST=10.0.211.41
KUBERNETES_SERVICE_PORT=443
KUBERNETES_PORT=tcp://10.0.0.1:443
MYDB_SERVICE_PORT=80
MYDB_PORT=tcp://10.0.51.122:80
HOSTNAME=myapp-pod3
SHLVL=1
HOME=/root
MYSERVICE_PORT=tcp://10.0.211.41:80
MYSERVICE_SERVICE_PORT=80
MYDB_PORT_80_TCP_ADDR=10.0.51.122
MYDB_PORT_80_TCP_PORT=80
MYDB_PORT_80_TCP_PROTO=tcp
lives=3
MYSERVICE_PORT_80_TCP_ADDR=10.0.211.41
secret.code.passphrase=UDDLRBABAS
MYSERVICE_PORT_80_TCP_PORT=80
MYSERVICE_PORT_80_TCP_PROTO=tcp
TERM=xterm
enemies=aliens
KUBERNETES_PORT_443_TCP_ADDR=10.0.0.1
MYDB_PORT_80_TCP=tcp://10.0.51.122:80
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_PROTO=tcp
MYSERVICE_PORT_80_TCP=tcp://10.0.211.41:80
secret.code.lives=30
secret.code.allowed=true
KUBERNETES_PORT_443_TCP=tcp://10.0.0.1:443
KUBERNETES_SERVICE_PORT_HTTPS=443
```

```
enemies.cheat.level=noGoodRotten
KUBERNETES_SERVICE_HOST=10.0.0.1
PWD=/
MYDB_SERVICE_HOST=10.0.51.122
enemies.cheat=true
```

通过上面的输出可以看到确实是所有环境变量都映射进来了。

通过 volume 挂载

在 Pod 中可以定义多种 volume，比如 `emptyDir`、`hostPath` 等，ConfigMap 也可以作为一种类型的 volume 使用。下面是一个使用 ConfigMap 作为 volume 的使用示例：

- 首先在 Pod 的 spec 字段域定义一个 volumes 字段，下面包含了两个 ConfigMap 类型的 volume：config-volume 和 env-config-volume，分别映射到名字叫 game-config 和 game-config-env 的两个 ConfigMap。
- 在 spec.containers 下的某个 container 的定义中通过 volumeMounts 将上面的 volume 挂载到目录：/etc/config 和 /etc/env-config

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod1
  labels:
    app: myapp
spec:
  containers:
    - name: myapp-container
      image: busybox
      command: ['sh', '-c', 'echo Hello kubernetes! && sleep 3600']
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
        - name: env-config-volume
          mountPath: /etc/env-config
  volumes:
    - name: config-volume
      configMap:
        name: game-config
    - name: env-config-volume
      configMap:
        name: game-config-env
```

我们通过 `kubectl apply` 启动 Pod，然后通过 `kubectl exec` 登录进去看一下挂载点的情况。

```
$ kubectl exec -ti myapp-pod1 -n imooc -- sh
/
enemies                enemies.cheat.level    secret.code.allowed
secret.code.passphrase
enemies.cheat          lives                  secret.code.lives
/
game.properties        ui.properties
```

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

通过上面输出我们可以看到对于通过文件创建的 ConfigMap 在挂载目录下是每个文件对应一个配置文件，而通过环境变量文件创建的 ConfigMap 在挂载目录下是每个 key 对应一个配置文件。

3. ConfigMap 使用限制

对于 ConfigMap 的两种使用方式：环境变量方式和挂载 volume 的方式，在生产环境中一般建议使用第二种方式。对于使用 ConfigMap 的几个限制这里简单提一下：

1. ConfigMap 是通过 etcd 存储的（实际上 kubernetes 中所有 API 对象都是存储在 etcd 中的），etcd 的 value 默认有一个限制是 1M 大小，这个在使用的时候需要注意。
2. 更新问题。有些情况我需要更新 ConfigMap，这个时候就涉及到能不能即时在 Pod 内生效的问题。
 - 环境变量方式如果 Pod 不重启 ConfigMap 是不会自动更新的。
 - 通过 volume 挂载的方式可以在 10s 左右自动更新。

3. Secret 创建

Secret 对象类型一般用来保存敏感信息，比如密码、令牌和 ssh key 等。将这些信息放在 secret 中比放在 Pod 的定义 spec 或者容器镜像中来说更加的安全和灵活，也可以更好的控制。

下面我们来看一下如何创建 secret 对象，有两种方式，一种是通过 kubectl 命令行创建，一种是手动创建。

kubectl 命令行创建

假设我们现在要用 secret 来保存用户名密码，我们先将用户名和密码保存到两个本地文件中

username.txt 和 password.txt 。

```
$ echo -n 'admin' > ./username.txt
$ echo -n '1f2d1e2e67df' > ./password.txt
```

下面使用 `kubectl create secret` 命令行的方式通过引用文件来创建。

```
kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt
```

然后我们可以通过 `kubectl get secret` 的方式来查看 secret 的内容。

```
$ kubectl get secret db-user-pass -n imooc -o yaml
apiVersion: v1
data:
  password.txt: MWYyZDF1MmU2N2Rm
  username.txt: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: "2020-04-08T02:57:28Z"
  name: db-user-pass
  namespace: imooc
  resourceVersion: "35623868"
  selfLink: /api/v1/namespaces/imooc/secrets/db-user-pass
  uid: aee2388b-7944-11ea-a328-00163e16aee6
type: Opaque
```

我们可以看到 data 字段的数据被加密了，其实这里说加密是不准确的，这个数据其实是被做了 base64 编码。我们可以通过 base64 解码看一下。

```
$ echo 'YWRtaW4=' | base64 --decode
admin
```

手动创建

手动创建就是手动创建一个类似上面的 secret 对象的 yaml 文件，然后通过 `kubectl apply` 创建，这里就不赘述了。

5. Secret 使用

Secret 对象的使用主要也是通过两种方式：

环境变量

和 ConfigMap 比较类似，下面是 secret 作为通过环境变量使用的例子。

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
    - name: mycontainer
      image: redis
      env:
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
```



```
name: mysecret
key: password
restartPolicy: Never
```

通过 volume 挂载

下面是将 secret 通过 volume 挂载到 Pod 内部的一个例子。我们可以看到最后的 volume 定义里面有一个 mode 字段，没错，这个就是文件模式。

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
  volumes:
  - name: foo
    secret:
      secretName: mysecret
      items:
      - key: username
        path: my-group/my-username
        mode: 511
```

6. 总结

本文介绍了 Kubernetes 中配置管理的两种方式：ConfigMap 和 Secret，总体来说还比较简单。

}