k8s-基础篇-配置管理

ConfigMap

什么是configmap

```
一般用 ConfigMap 去管理一些配置文件，或者一些大量的环境变量信息。

ConfigMap 将配置和 Pod分开，有一个nginx, nginx.conf-> configmap， nginx 去读取configmap的信息。更易于配置文件的更新和管理。

Secret: Secret更倾向于存储和共享敏感、加密的配置信息。
```

配置文档：https://kubernetes.io/zh/docs/tasks/configure-pod-container/configure-pod-configmap/

创建 ConfigMap

通过yaml / json文件创建（推荐）

这种是我比较推荐的方式，创建configmap.yaml:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-conf
  namespace: test
data:
  test-conf: |+
    SESSION_LIFETIME: 3600
    URL: "http://test-server:8080"
```

执行命令：

```
kubectl create -f configmap.yaml
```

若报错："namespace 'test' not found"，则需要先创建namespace：

```
kubectl create namespace test
```

通过--from-file

分别指定单个文件和目录，指定目录可以创建一个包含该目录中所有文件的configmap：

```
kubectl create configmap *** --from-file=/path
```

将--from-file指定为单个文件就可以从单个文件中创建：

```
kubectl create configmap *** --from-file=file1
```

其中，--from-file可以使用多次，比如：

```
kubectl create configmap *** --from-file=file1 --from-file=file2
```

通过key-value字符串创建

```
kubectl create configmap *** --from-literal=config1=123 --from-literal=config2=234
```

通过env文件创建

通过环境文件创建：

```
kubectl create configmap *** --from-env-file=env.txt
```

其中，env.txt的文件格式为:

```
config1=***
config2=***
```

当使用多个--from-env-file从多个数据源创建configmap时，仅最后一个env文件有效。

查看

可以使用以下命令查看创建成功的configmap:

命令	说明
kubectl get configmaps	查看所有configmap
kubectl get configmaps -n namespace1	查看命名空间为namespace1的所有configmap
kubectl describe configmaps configmap1	查看configmap1的详细信息
kubectl get configmaps configmap1 -o yaml	以yaml文件形式展现configmap详细信息

使用

configmap创建成功之后，如何在pod中使用呢？有以下几种方法：

注意

使用ConfigMap有以下几个限制条件：

1. ConfigMap必须在pod之前创建
2. configmap受namespace的限制，只能相同namespace的pod才可以引用

env

通过环境变量获取ConfigMap中的内容。

首先创建configmap:

```
kubectl create configmap test-config --from-literal=env_model=prd -n test
```

接下来用作环境变量，创建pod.yaml：

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  namespace: test
spec:
  containers:
    - name: test-container
      image: test:v0.1
      env:
        - name: TEST-CONF
          valueFrom:
            configMapKeyRef:
              name: test-config
              key: env_model
```

执行命令创建Pod：

```
kubectl create -f pod.yaml
```

创建成功之后，执行命令查看pod的详细信息，可以看到已经将configmap中的配置添加到环境变量：

```
kubectl describe pod test-pod -n test
```

同时，也支持多个configmap共同创建环境变量。

volume

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，这是我平时用的较多的方式。

接下来使用最开始创建的test-conf为例说明，将configmap挂载到特定目录，并保存为指定文件：

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  namespace: test
spec:
  containers:
    - name: test-container
      image: test:v0.1
      volumeMounts:
      - name: test-volume
        mountpath: /app/config
  volumes:
    - name: test-volume
    configMap:
      name:test-conf
      items:
      - key: test-conf
        path: config.yaml
```

Secret

什么是secret

文档：https://kubernetes.io/zh/docs/concepts/configuration/secret/

> 用于存储和管理一些敏感数据，比如密码，token，密钥等敏感信息。它把 Pod 想要访问的加密数据存放到 Etcd 中。然后用户就可以通过在 Pod 的容器里挂载 Volume 的方式或者环境变量的方式访问到这些 Secret 里保存的信息了。

使用 kubectl 管理 Secret

创建Secret

一个 `Secret` 可以包含 Pod 访问数据库所需的用户凭证。 例如，由用户名和密码组成的数据库连接字符串。 你可以在本地计算机上，将用户名存储在文件 `./username.txt` 中，将密码存储在文件 `./password.txt` 中。

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

在这些命令中，`-n` 标志确保生成的文件在文本末尾不包含额外的换行符。 这一点很重要，因为当 `kubectl` 读取文件并将内容编码为 base64 字符串时，多余的换行符也会被编码。

`kubectl create secret` 命令将这些文件打包成一个 Secret 并在 API 服务器上创建对象。

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

输出类似于：

```
secret/db-user-pass created
```

默认密钥名称是文件名。 你可以选择使用 `--from-file=[key=]source` 来设置密钥名称。例如：

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

你不需要对文件中包含的密码字符串中的特殊字符进行转义。

你还可以使用 `--from-literal=<key>=<value>` 标签提供 Secret 数据。 可以多次使用此标签，提供多个键值对。 请注意，特殊字符（例如：`$`，`\`，`*`，`=` 和 `!`）由你的 shell 解释执行，而且需要转义。

在大多数 shell 中，转义密码最简便的方法是用单引号括起来。 比如，如果你的密码是 `S!B\*d$zDsb=`， 可以像下面一样执行命令：

```
kubectl create secret generic dev-db-secret \
  --from-literal=username=devuser \
  --from-literal=password='S!B\*d$zDsb='
```

验证Secret

检查 secret 是否已创建：

```
kubectl get secrets
```

输出类似于：

```
NAME                  TYPE                                  DATA      AGE
db-user-pass          Opaque                                2         51s
```

你可以查看 `Secret` 的描述：

```
kubectl describe secrets/db-user-pass
```

输出类似于：

```
Name:            db-user-pass
Namespace:       default
Labels:          <none>
Annotations:     <none>

Type:            Opaque

Data
====
password:    12 bytes
username:    5 bytes
```

`kubectl get` 和 `kubectl describe` 命令默认不显示 `Secret` 的内容。 这是为了防止 `Secret` 被意外暴露或存储在终端日志中。

解码Secret

要查看创建的 Secret 的内容，运行以下命令：

```
kubectl get secret db-user-pass -o jsonpath='{.data}'
```

输出类似于：

```
{"password":"MWYyZDFlMmU2N2Rm","username":"YWRtaW4="}
```

现在你可以解码 `password` 的数据：

```
echo 'MWYyZDFlMmU2N2Rm' | base64 --decode
```

输出类似于：

```
1f2d1e2e67df
```

清理

删除创建的 Secret：

```
kubectl delete secret db-user-pass
```

使用配置文件管理 Secret

创建配置文件

你可以先用 JSON 或 YAML 格式在文件中创建 Secret，然后创建该对象。 Secret 资源包含2个键值对： `data` 和 `stringData`。 `data` 字段用来存储 base64 编码的任意数据。 提供 `stringData` 字段是为了方便，它允许 Secret 使用未编码的字符串。 `data` 和 `stringData` 的键必须由字母、数字、`-`，`_` 或 `.` 组成。

例如，要使用 Secret 的 `data` 字段存储两个字符串，请将字符串转换为 base64 ，如下所示：

```
echo -n 'admin' | base64
```

输出类似于：

```
YWRtaW4=
echo -n '1f2d1e2e67df' | base64
```

输出类似于：

```
MWYyZDFlMmU2N2Rm
```

编写一个 Secret 配置文件，如下所示：

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

注意，Secret 对象的名称必须是有效的 DNS 子域名.

> **说明：**

> Secret 数据的 JSON 和 YAML 序列化结果是以 base64 编码的。 换行符在这些字符串中无效，必须省略。 在 Darwin/macOS 上使用 `base64` 工具时，用户不应该使用 `-b` 选项分割长行。 相反地，Linux 用户 *应该* 在 `base64` 地命令中添加 `-w 0` 选项， 或者在 `-w` 选项不可用的情况下，输入 `base64 | tr -d '\n'`。

对于某些场景，你可能希望使用 `stringData` 字段。 这字段可以将一个非 base64 编码的字符串直接放入 Secret 中， 当创建或更新该 Secret 时，此字段将被编码。

上述用例的实际场景可能是这样：当你部署应用时，使用 Secret 存储配置文件， 你希望在部署过程中，填入部分内容到该配置文件。

例如，如果你的应用程序使用以下配置文件:

```
apiUrl: "https://my.api.com/api/v1"
username: "<user>"
password: "<password>"
```

你可以使用以下定义将其存储在 Secret 中:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
stringData:
  config.yaml: |
    apiUrl: "https://my.api.com/api/v1"
    username: <user>
    password: <password>
```

创建 Secret 对象

现在使用 `kubectl apply` 创建 Secret：

```
kubectl apply -f ./secret.yaml
```

输出类似于：

```
secret/mysecret created
```

检查 Secret

`stringData` 字段是只写的。获取 Secret 时，此字段永远不会输出。 例如，如果你运行以下命令：

```
kubectl get secret mysecret -o yaml
```

输出类似于：

```
apiVersion: v1
data:
  config.yaml:
YXBpVXJsOiAiaHR0cHM6Ly9teS5hcGkuY29tL2FwaS92MSIKdXNlcm5hbWU6IHt7dXNlcm5hbWV9fQpwYXNzd29
yZDoge3twYXNzd29yZH19
kind: Secret
metadata:
  creationTimestamp: 2018-11-15T20:40:59Z
  name: mysecret
  namespace: default
  resourceVersion: "7225"
  uid: c280ad2e-e916-11e8-98f2-025000000001
type: Opaque
```

命令 `kubectl get` 和 `kubectl describe` 默认不显示 `Secret` 的内容。 这是为了防止 `Secret` 意外地暴露给旁观者或者保存在终端日志中。 检查编码数据的实际内容，请参考解码 secret.

如果在 `data` 和 `stringData` 中都指定了一个字段，比如 `username`，字段值来自 `stringData`。 例如，下面的 Secret 定义:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
stringData:
  username: administrator
```

结果有以下 Secret:

```
apiVersion: v1
data:
  username: YWRtaW5pc3RyYXRvcg==
kind: Secret
metadata:
  creationTimestamp: 2018-11-15T20:46:46Z
  name: mysecret
  namespace: default
  resourceVersion: "7579"
  uid: 91460ecb-e917-11e8-98f2-025000000001
type: Opaque
```

其中 `YWRtaW5pc3RyYXRvcg==` 解码成 `administrator`。

清理

删除你创建的 Secret：

```
kubectl delete secret mysecret
```

ConfigMap&Secret使用SubPath

挂载的时候，不覆盖其他文件
为了支持单一个pod多次使用同一个volume而设计，subpath翻译过来是子路径的意思，如果是数据卷挂载在容器，指的是存储卷目录的子路径，如果是配置项configMap/Secret，则指的是挂载在容器的子路径。

1）创建configMap

```
[root@k8s-master consecret]# cat conf-subpath.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: conf-subpath-zltest
  namespace: default
data:
  example.property.1: hello      # key-value键值对
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
```

2）在Pod中使用configMap

```
[root@k8s-master consecret]# cat pod-conf-subpath.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    purpose: test-configmap-volume
  name: pod-conf-testvolume
spec:
  containers:
    - name: test-configmap-volume
      image: nginx
      volumeMounts:
        - name: config-volume
          mountPath: /etc/nginx/example.property.1       # 容器挂载目录
          subPath: example.property.1                    # 将key名称作为文件名，hello作为文
件内容
```

```
  volumes:
    - name: config-volume
      configMap:
         name: conf-subpath-zltest      # 指定使用哪个CM

[root@k8s-master consecret]# kubectl create -f pod-conf-subpath.yaml
[root@k8s-master consecret]# kubectl describe pod  pod-conf-testvolume
Name:         pod-conf-testvolume
Namespace:    default
Priority:     0
Node:         k8s-master/192.168.126.129
Start Time:   Wed, 03 Jun 2020 11:46:36 +0800
Labels:       purpose=test-configmap-volume
Annotations:  cni.projectcalico.org/podIP: 10.122.235.249/32
              cni.projectcalico.org/podIPs: 10.122.235.249/32
Status:       Running
IP:           10.122.235.249
IPs:
  IP:  10.122.235.249
Containers:
  test-configmap-volume:
    Container ID:
docker://e2cf37cb24af32023eb5d22389545c3468104a4344c47363b5330addc40cb914
    Image:          nginx
    Image ID:       docker-
pullable://nginx@sha256:883874c218a6c71640579ae54e6952398757ec65702f4c8ba7675655156fcca
6
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Wed, 03 Jun 2020 11:46:53 +0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /etc/nginx/example.property.1 from config-volume (rw,path="example.property.1")
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-74s86 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  config-volume:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      conf-subpath-zltest
    Optional:  false
  default-token-74s86:
```

```
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-74s86
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:          <none>
```

在容器挂载路径验证下是否将configMap中example.property.1挂载在容器中，且是否会覆盖掉原有的目录

```
root@pod-conf-testvolume:/# cd  /etc/nginx
root@pod-conf-testvolume:/etc/nginx# ls
conf.d            fastcgi_params  koi-win    modules     scgi_params   win-utf
example.property.1  koi-utf        mime.types    nginx.conf  uwsgi_params
```

从上可以看到example.property.1已经挂载到容器中，且未对目录原有的文件进行覆盖

```
root@pod-conf-testvolume:/etc/nginx# cd example.property.1
bash: cd: example.property.1: Not a directory
root@pod-conf-testvolume:/etc/nginx# cat example.property.1
helloroot@pod-conf-testvolume:/etc/nginx#
```

从上可以验证configMap的subpath用法支持将configMap中的每对key-value以key名称作为文件名，value作为文件内容挂载到容器的目录中。

ConfigMap&Secret 热更新

ConfigMap 和 Secret 如果是以 subPath 的形式挂载的，那么 Pod 是不会感知到ConfigMap 和Secret 的更新。

如果 Pod 的变量来自于 ConfigMap 和 Secret 中定义的变量，那么 ConfigMap 和 Secret更新后，也不会更新Pod中的变量

Kubernetes中提供configmap，用来管理应用的配置，configmap具备热更新的能力，但只有通过目录挂载的configmap才具备热更新能力，其余通过环境变量，通过subPath挂载的文件都不能动态更新。

使用volume的方式挂载configmap之后，当configmap更新之后，变量的值会发生变化

但是中间会存在一定的时间间隔，大约是10左右，这主要是因为kubelet 对pod的同步间隔是10秒，另外需要注意的是当使用subpath将configmap中的某个文件单独挂载到目录下，那这个文件是无法热更新的，这是configmap本身的逻辑决定的。

参考：https://blog.csdn.net/qingyafan/article/details/102848860

postStart: 容器启动之前执行的命令

preStrop: 容器停止之前执行的命令

热更新ConfigMap 和 Secert

```
kubectl create cm nginx-conf --from-file=nginx.conf --dry-run -oyaml | kubectl replace
-f
```

ConfigMap&Secret 不可变

比如秒杀系统中，设置了某个值是100，只能100个人秒杀成功，这个值不可变

```
kubectl create cm test-immutable --from-file=/etc/kubernetes/admin.kubeconfig

# 最后加上
kubectl edit cm test-immutable

immutable: true

# 再去修改 kubectl edit cm test-immutable,  发现无法修改
```