前言

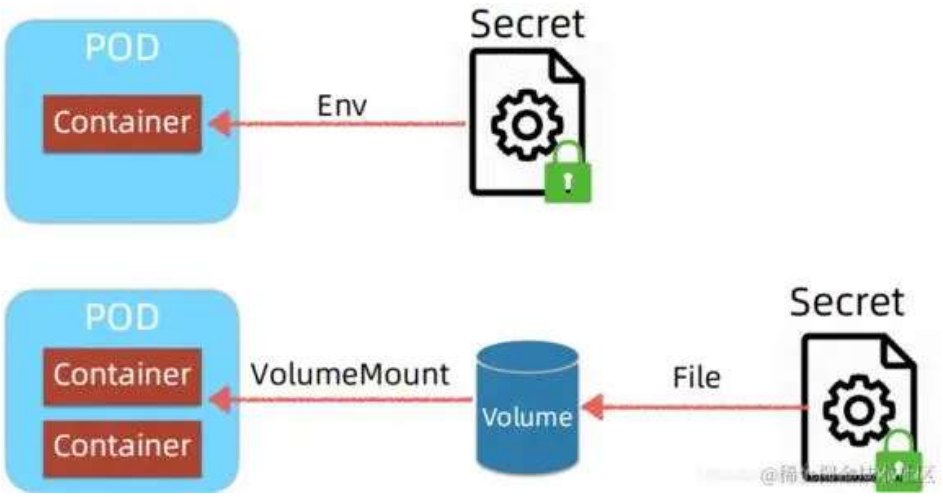
在之前几个章节中，我们实现了对一个前端镜像的简单部署流程。

可是，我们在部署时，难免会遇到一些要存放**机密内容**的需求。例如我们的数据库密码，用户名密码，公钥私钥，`token` 等等机密内容，甚至还有我们 `docker` 私有库的密码。而这些内容，显然是不能写死在代码里面，更不可能明文挂载进去的。

那么我们有没有什么好的解决方案能够使用呢？这一章我们就来学习 `Kubernetes` 中的一个概念 —— `Secret`

什么是 Secret

Secret 是 Kubernetes 内的一种资源类型，可以用它来存放一些机密信息（密码，token，密钥等）。信息被存入后，我们可以使用挂载卷的方式挂载进我们的 Pod 内。当然也可以存放docker私有镜像库的登录名和密码，用于拉取私有镜像。



Secret 的几种类型

在 k8s中，secret 也有多种类型可以配置

Opaque 类型

第一种是 `opaque` 类型，这种类型比较常用，一般拿来存放密码，密钥等信息，存储格式为 `base64` 。但是请注意：base64并不是加密格式，依然可以通过decode来解开它。

例如我们创建一组用户名和密码，用户名为 `janlay` 和 `367734wer` 。则可以通过命令 `kubectl create secret generic` 创建：

```
2  --from-literal=password=367734qwer
```

在这里， `default-auth` 为 自定义的名称， `--from-literal` 的后面则跟随一组 `key=value` 。当然你也可以按照此格式继续向后拼接你要存储的信息。

存储成功后，我们可以通过 `kubectl get secret` 命令来查看你存储过的 `Secret` 。在这里可以看到，刚刚创建的密钥组合 `default-auth` 已经展示了出来。

在这里， `NAME` 代表 `Secret` 的名称； `TYPE` 代表 Secret 的类型； `DATA` 是 `Secret` 内存储内容的数量； `AGE` 是创建到现在的时间

```
[root@master deployment]# kubectl get secret
NAME                  TYPE                                  DATA   AGE
default-auth          Opaque                                2      3s
default-token-k8zvv   kubernetes.io/service-account-token   3      40d
[root@master deployment]#
```


我们可以通过 `kubectl edit secret` 命令来编辑 `default-auth` 的内容，来看看里面到底存了什么内容：

shell 复制代码

```
1  kubectl edit secret default-auth
```

> 这里也可以用 kubectl get secret [secret名称] -o yaml 命令，将内容打印到终端上查看。其中 -o yaml 代表输出为 yaml 格式内容，当然也可以输出 json 等格式内容

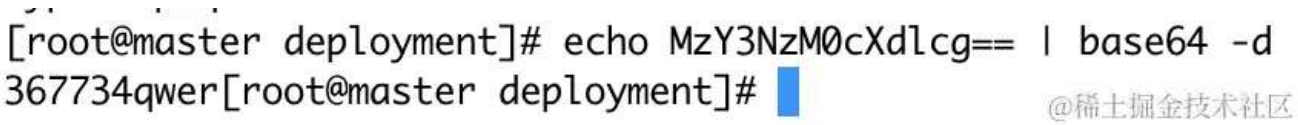
```
apiVersion: v1
data:
  password: MzY3NzM0cXdlcg==
  username: amFubGF5
kind: Secret
metadata:
  creationTimestamp: "2020-11-19T11:24:22Z"
  name: default-auth
  namespace: default
  resourceVersion: "1374025"
  selfLink: /api/v1/namespaces/default/secrets/
  uid: 4525ee16-0743-4246-96a6-6011e28d9a4b
type: Opaque
~
```


可以看到，data 字段存放了我们存储的信息 `base64` 后的结果。但是这种方式是不安全的，我们可以通过解码base64 来获取真实

shell 复制代码

```shell
echo MzY3NzM0cXdlcg== | base64 -d
```

> 这里可以使用 Linux 自带的 base64 命令进行解码。其中 -d 代表 --decode，解码的意思



解码后，我们可以清晰的看到原始内容。

那么除了通过命令创建，可不可以通过配置文件创建呢？答案是可以的。我们新建一个文件，名称叫 `admin-auth.yaml` ，输入以下配置：

yaml 复制代码

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: admin-auth
stringData:
  username: wss
  password: wss@1234
type: Opaque
```

在这里， `name` 代表 `Secret` 的名称，名称为 `admin-auth` ； `type` 代表它的类型，类型为 `Opaque` ； `stringData` 代表存储的内容，格式为 `key:value` 。

我们保存后退出，使用 `kubectl apply -f` 命令生效这份配置。接着使用 `kubectl get secret admin-auth -o yaml` 查看下内容：

shell 复制代码

```shell
kubectl apply -f admin-auth.yaml
kubectl get secret admin-auth -o yaml
```

```
data:
  password: d3NzQDEyMzQ=
  username: d3Nz
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Secret","metadata":{"annotat
lt"},"stringData":{"password":"wss@1234","username":"wss"},"t
  creationTimestamp: "2020-11-19T14:08:06Z"
```


可以看到，创建是正常成功的。

私有镜像库认证

第二种是私有镜像库认证类型，这种类型也比较常用，一般在拉取私有库的镜像时使用。

在这里我们依然可以通过命令行进行创建。只不过类型变成了 `docker-registry` ：

shell 复制代码

```shell
kubectl create secret docker-registry private-registry \
--docker-username=[用户名] \
--docker-password=[密码] \
--docker-email=[邮箱] \
--docker-server=[私有镜像库地址]
```

创建成功后，我们可以使用 `kubectl get secret` 命令查看下我们配置的私有库密钥组：

shell 复制代码

```shell
kubectl get secret private-registry -o yaml
```

```
[root@master secret]# kubectl get secret private-registry -o yaml
apiVersion: v1
data:
  .dockerconfigjson: eyJhdXRocyI6eyJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8iOnsidXN
3b3JkIjoiMzY3NzM0IiwiZW1haWwiOiJqYW5sYXk4ODQxODEzMTdAZ21haWwuY29tIiwiYXV0aCI6Ill
kind: Secret
```


可以看到，k8s 自动帮我们填写了一个key，为 `.dockerconfigjson` ；value则是一串 base64 值。我们依然可以使用 `base64 -d` 命令查看下里面到底是啥：

shell 复制代码

```shell
echo [value] | base64 -d
```

```
{ auths :{ https://index.docker.io/v1/ :{ username : admin , password : 507754 , email : junlay8841815i7@gma
il.com","auth":"YWRtaW46MzY3NzM0"}}}[root@master secret]#
```


通过解码后可以看到， k8s 会自动帮我们创建一串 dockerconfig 的 json 串。在 k8s 拉取镜像时，则可以使用这个 json 串来用于身份认证。

当然，私有镜像库密钥组也可以通过配置文件创建。编辑文件名为 private-registry-file.yaml 文件，并输入以下内容：

yaml 复制代码

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: private-registry-file
data:
  .dockerconfigjson: eyJhdXRocyI6eyJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8iOnsidXNlcm5hbWUiOiJhZG1pbiIsInBhc3N3b3JkIjoi
type: kubernetes.io/dockerconfigjson
```

大家可能发现在这里创建镜像库认证时，声明的配置文件更像是一份 dockerconfig ，而不只是单纯的镜像库身份认证。

在这里， data 内的字段必须为 .dockerconfigjson ，值则是一串 dockerconfigjson 的 base64 值； type 则为 kubernetes.io/dockerconfigjson ，意思是声明一份 dockerconfig 的配置

保存后退出，使用 kubectl apply -f 命令让该配置生效。并使用 kubectl get secret 命令查看下我们配置的详情：

shell 复制代码

```shell
kubectl apply -f ./private-registry-file.yaml
kubectl get secret private-registry-file -o yaml
```

```
[root@master secret]# kubectl get secret private-registry-file -o yaml
apiVersion: v1
data:
  .dockerconfigjson: eyJhdXRocyI6eyJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8iOnsidXNlcm5hbWUiOiJhZG1pbiIsInBhc3N
3b3JkIjoiMzY3NzM0IiwiZW1haWwiOiJqYW5sYXk4ODQxODEzMTdAZ21haWwuY29tIiwiYXV0aCI6IlllXUnRhVzQ2TXpZM056TTAifX19
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{".dockerconfigjson":"eyJhdXRocyI6eyJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8iOn
sidXNlcm5hbWUiOiJhZG1pbiIsInBhc3N3b3JkIjoiMzY3NzM0IiwiZW1haWwiOiJqYW5sYXk4ODQxODEzMTdAZ21haWwuY29tIiwiYXV0aC
I6IlllXUnRhVzQ2TXpZM056TTAifX19"},"kind":"Secret","metadata":{"annotations":{},"name":"private-registry-file"
,"namespace":"default"},"type":"kubernetes.io/dockerconfigjson"}
```


可以看到，配置内容和命令行创建的是一样的。创建成功

使用方法

上面我们写了如何声明一个 Secret。在声明后，我们需要在实际的配置中使用，才有实际意义。在 K8S 中，一共有三种可以使用 Secret 的方式。

第一种是通过存储卷的方式挂载进去。我们可以编辑下 `front-v1` 的 `deployment` 配置文件去配置下。

第一步：在Pod层面设置一个外部存储卷，存储卷类型为 `secret` 。在 `template.spec` 下填写。这里代表声明了一个外置存储卷，存储卷名称为 `admincert` ，类型为 `secret` ； `Secret` 的名称为 `admin-auth` ：

yaml 复制代码

```yaml
volumes:
- name: admincert
  secret:
    secretName: admin-auth
```

```yaml
template:
  metadata:
    labels:
      app: nginx-v1
  spec:
   # hostNetwork: true
   volumes:
   - name: admincert
     secret:
       secretName: admin-auth
   containers:
    - name: nginx
      image: registry.cn-hangzhou.a
```

第二步：在容器配置配置存储卷。在 `containers.name[]` 下填写字段 `volumeMounts` 。这里的 `name` 值和上面的卷名是对应的。 `mountPath` 是要挂载到容器内哪个目录，这里代表挂载到用户目录下； `readonly` 则代表文件是不是只读：

yaml 复制代码

```yaml
volumeMounts:
- name: admincert
  mountPath: /root
  readOnly: true
```

```
    image: registry.cn-hangzhou.aliyuncs.com/janlay/k8s_test:v1
    volumeMounts:
    - name: admincert
      mountPath: /root
      readOnly: true
    ports:
    - containerPort: 80
      hostPort: 0
```


编辑完后，保存并退出。使用 `kubectl apply -f` 命令生效下配置文件。

shell 复制代码

```
kubectl apply -f ./v1.yaml
```

此时， `Pod` 会被杀死重新创建。我们可以通过 `kubectl get pods` 来查看现在运行的 `Pod`

```
[root@master deployment]# kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
front-v1-55857b98f4-7p4nc      1/1     Running   0          16m
front-v2-848dbd46f6-c7ncz      1/1     Running   0          6h11m
front-v2-848dbd46f6-rpfc8      1/1     Running   0          6h11m
```


此时可以看到，我们的 `Pod` 状态为 `Running` 运行状态。

在运行正常的情况下，我们可以使用 `kubectl exec` 命令在 `Pod` 容器内执行我们要执行的命令。在这里，我们查看下 `Pod` 镜像内的 `/root` 文件夹里面都有啥文件：

> kubectl exec 命令格式：kubectl exec [POD] -- [COMMAND]

shell 复制代码

```
kubectl exec -it [POD_NAME] -- ls /root
```

```
[root@master deployment]# kubectl exec -it front-v1-55857b98f4-7p4nc -- ls /root
password  username
```


可以看到，分别有2个文件，都是我们在 `secret` 内配置的 `key` 。接着使用 `kubectl exec` 命令，查看下文件内容：

shell 复制代码

```
kubectl exec -it [POD_NAME] -- cat /root/password
kubectl exec -it [POD_NAME] -- cat /root/username
```

此时，代表挂载成功，可以使用。

环境变量注入

第二种是将 `Secret` 注入进容器的环境变量。同样需要配置下 `deployment` 文件。找到 `containers` ，下面新加一个 `env` 字段：

其中， `env[].name` 为环境变量的 `key` ， `valueFrom` 为值； `secretKeyRef` 则代表是一个 `Secret` 类型的 `value` 。

`secretKeyRef.name` 则是要引用的 `secret` 的名称， `key` 则是 `secret` 中配置的 `key` 值。

yaml 复制代码

```yaml
env:
  - name: USERNAME
        valueFrom:
                secretKeyRef:
                name: admin-auth
        key: username
  - name: PASSWORD
    valueFrom:
        secretKeyRef:
        name: admin-auth
      key: password
```

```yaml
containers:
  - name: nginx
    image: registry.cn-hangzhou.aliyuncs.com/janlay/k8s_test:v1
    env:
      - name: USERNAME
        valueFrom:
          secretKeyRef:
            name: admin-auth
            key: username
      - name: PASSWORD
        valueFrom:
          secretKeyRef:
            name: admin-auth
            key: password
```


编辑完后，保存并退出。使用 `kubectl apply -f` 命令生效下配置文件。

shell 复制代码

```shell
kubectl apply -f ./v1.yaml
```

```shell
kubectl exec -it [POD_NAME] -- env
```

```
[root@master deployment]# kubectl exec -it front-v1-745998f559-bzmhf -- env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=front-v1-745998f559-bzmhf
TERM=xterm
USERNAME=wss
PASSWORD=wss@1234
```


可以看到，我们配置的2个环境变量均已被注入进去。

Docker 私有库认证

第三种是 Docker 私有库类型，这种方法只能用来配置 私有镜像库认证。

首先，我们先尝试不加认证去拉取一个私有库镜像。编辑下 `front-v1` 的 `deployment` ，把镜像换成私有库的镜像。保存后使用 `kubectl apply` 生效配置：

```yaml
image: [镜像库地址]/jenkins-test:latest
```

接着使用 `kubectl get pods` 查看下目前pod的状态：

```
[root@master deployment]# kubectl get pods
NAME                        READY   STATUS             RESTARTS   AGE
front-v1-9fbf5845d-8sglb    0/1     ImagePullBackOff   0          2m19s
front-v2-848dbd46f6-c7ncz   1/1     Running            1          8h
front-v2-848dbd46f6-rpfc8   1/1     Running            2          8h
front-v2-848dbd46f6-wknlr   1/1     Running            1          8h
```


可以看到， `front-v1` 的 `Pod` 并无法拉取下来镜像。我们使用 `kubectl describe` 命令查看下该 Pod 的具体状态：

```shell
kubectl describe pods [POD_NAME]
```

找到 `Events` 那一块，可以其中一条 `message` 写着：**unauthorized: access to the requested resource is not authorized（要请求的资源没有认证）。此时不登录，无法拉取私有镜像。**

那怎么办呢？这里我们需要配置下 `deployment` 文件。

找到 `spec` ，下面新加一个 `imagePullSecrets` 字段。该字段代表了在拉取Pod所需要的镜像时，需要的认证信息。其中， `name` 字段为上面我们配置过的私有镜像库认证名。

```
2   - name: private-registry-file
```

```
spec:
 # hostNetwork: true
 volumes:
 - name: admincert
   secret:
     secretName: admin-auth
 imagePullSecrets:
  - name: private-registry-file
 containers:
  - name: nginx
    image: 172.16.81.7:8082/jenkins-test:latest
    env:
```


编辑后保存，使用 `kubectl apply -f` 命令生效配置文件。接着看下 Pod 的运行状态。

```
[root@master deployment]# kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
front-v1-7f8b97bbd5-dbvdb     1/1     Running   0          74s
front-v2-848dbd46f6-c7ncz     1/1     Running   1          8h
front-v2-848dbd46f6-rpfc8     1/1     Running   2          8h
front-v2-848dbd46f6-wknlr     1/1     Running   1          8h
```


此时我们发现，Pod 可以成功拉取私有镜像了。

‹ 上一章　　下一章 ›

留言

输入评论（Enter换行，Ctrl + Enter发送）

发表评论

全部评论（2）

Asuka14024 LV.1 JY.5 前端工程师 5月前