INTRODUCTION TO KUBERNETES SECRETS



In this blog post, we are going to discuss K8s secrets, including:

- What secrets are
- How to create a secret
- How to work with secrets

I assume you have a basic understanding of Kubernetes and concepts like pod, deployment, service, etc. To follow along, you will need to have kubectl and minikube installed.

(This article is part of our Kubernetes Guide. Use the right-hand menu to navigate.)

What is a K8s secret?

A secret as the name implies is any information that need to be kept confidential such as password, token, etc.

You can technically put the credentials directly into a pod specification in plain text but doing that is not very safe as you can imagine. To solve this, Kubernetes has the concept of secrets where you can store your sensitive info securely and also control how a pod consumes it.

How to create secrets?

In general both user and kubernetes itself can create a secret. If all you need is to access the API securely, then K8s can automatically create a secret attached to a service account which contains credentials to access the API. This is the recommended way to access the API.

In the situation where we need to create our own secret for other uses, we can do that as well. Take for example we have a pod running our application that need to access another system with the username "*example-user*" and password "*example-password*", how do we create the secret and get our pod to recognize and use the credentials? We can create it manually or with a yaml file or "kubectl create".

Manually create a secret

To create secret manually, we must first convert the string to base64.

```
$ echo -n 'example-user' | base64
4oCYZXhhbXBsZS11c2Vy4oCZ

$ echo -n 'example-password' | base64
4oCYZXhhbXBsZS1wYXNzd29yZOKAmQ==
```

Then we create a yaml file (example.yaml) with our secret specs.

```
apiVersion: v1
kind: Secret
metadata:
  name: demo
type: Opaque
data:
  username: 4oCYZXhhbXBsZS11c2Vy4oCZ
  password: 4oCYZXhhbXBsZS1wYXNzd29yZOKAmQ==
```

We can check if secret was created

```
$ kubectl get secret
NAME                  TYPE                                  DATA      AGE
default-token-v8gqd   kubernetes.io/service-account-token   3         6m
demo                  Opaque                                2         11s
```

As you can see we have a system generated secret and the one we just created. We can also describe our secret just to make sure our secret is not in plain text.

```
Type:  Opaque

Data
====
username:  18 bytes
password:  22 bytes
```

Create secret with kubectl

We can also create secret from a file that contains the credentials. So for example we could create the same secret by saving the username in a file called username.txt and password in password.txt. We can then run "*kubectl create secret generic demo-creds --from-file=./username.txt --from-file=./password.txt*".

How to use the secret?

There are two ways to consume the secret we created above:

- Data volume
- Env variable

We will demonstrate both with a simple example.

Consuming secret in a volume

To consume secret in a volume, we must first make sure that the volume is defined in the pod manifest and the name of the secret is specified so that container can use it. We can also do things like change file path and permissions.

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-pod
spec:
  containers:
  - name: mycontainer
    image: redis
    volumeMounts:             #We mount our volume
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:                    #We use the volume and defined our secret
  - name: foo
    secret:
      secretName: demo
```

We can see that our secret is in use

```
Volumes:
  foo:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  demo
    Optional:    false
  default-token-v8gqd:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-v8gqd
    Optional:    false
```

Now to verify that the secret is what we defined above, we can use “kubectl exec” to get into the container and check.

```
$ kubectl exec -ti mypod bash
root@mypod:/data# cat /etc/foo/username
'Example-user'
```

```
root@mypod:/data# cat /etc/foo/password
‘example-password’
```

We can see that the container in the pod is in fact using the credentials we created above. Next let us see what it looks like using the same secret in env variable.

Consuming secret in env variable

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: mycontainer
    image: redis
    env:
      - name: SECRET_USERNAME
        valueFrom:
          secretKeyRef:
            name: demo                  #Name of secret
            key: username               #secret key
      - name: SECRET_PASSWORD
        valueFrom:
          secretKeyRef:
            name: demo
            key: password
  restartPolicy: Never
```

As you can see, instead of the volumes, we are defining env[].valueFrom.secretKeyRef referencing the key in the secret. Let us see what this pod looks like after creating.

```
root@secret-env-pod:/data# env | grep SECRET
SECRET_PASSWORD='example-password'
SECRET_USERNAME='example-user'
```

Final tips for K8s secrets

A few more things to keep in mind:

- When you update the secret that is already being consumed, Kubelet makes sure the key is automatically updated.
- A pod is not the only object that uses a secret, other parts of the system can also use a secret.
- A secret can be consumed by more than one pod.

Additional resources

For more on Kubernetes, explore these resources:

- Kubernetes Guide, with 20+ articles and tutorials
- BMC DevOps Blog
- Bring Kubernetes to the Serverless Party
- How eBay is Reinventing Their IT with Kubernetes & Replatforming Program