Creating Generic Secrets

In this lesson, we will create and explore generic Secrets.

WE'LL COVER THE FOLLOWING

- Creating Secrets
- Working with Secrets
  - JSON Representation
  - Decoding the Values

Creating Secrets

The commands to create Secrets are almost the same as those we used to create ConfigMaps. We can, for example, generate Secrets based on literal values.

```
kubectl create secret \
    generic my-creds \
    --from-literal=username=jdoe \
    --from-literal=password=incognito
```

> The major difference is that we specified the type of the Secret as `generic`.

It could also be `docker-registry` or `tls`. We won't explore those two, but only say that the former can be used to provide `kubelet` with credentials it needs to pull images from private registries. The latter is used for storing certificates.

In this chapter, we'll focus on the `generic` type of secrets which happen to use the same syntax as ConfigMaps.

Just as with ConfigMaps, generic Secrets can use `--from-env-file`, `--from-file`, and `--from-literal` as sources. They can be mounted as files, or transformed into environment variables. Since creating Secrets is so similar to creating ConfigMaps, we won't go into all the permutations we can do.

For now, we created a Secret called `my-creds` which holds two literal values.

Working with Secrets

Let's take a look at the Secrets we now have in the cluster.

```
kubectl get secrets
```

The **output** is as follows.

```
NAME                  TYPE                                  DATA  AGE
default-token-n6fs4   kubernetes.io/service-account-token   3     33m
my-creds              Opaque                                2     6s
```

We can see that the newly created Secret is available and that it has two pieces of data.

JSON Representation

Let's see the `json` representation of the Secret and try to find out how to retrieve it.

```
kubectl get secret my-creds -o json
```

The **output** is as follows (`metadata` is removed for brevity).

```
{
    "apiVersion": "v1",
    "data": {
        "password": "aW5jb2duaXRv",
        "username": "amRvZQ=="
    },
    "kind": "Secret",
    "metadata": {
        ...
    },
    "type": "Opaque"
}
```

We can see that the `data` field contains the `password` and the `username`. They coincide with the literal values we specified in the command that created the Secret.

Decoding the Values

You'll notice that the values are "strange". They are encoded. If we'd like to see the original values we stored as secrets, we'll need to decode them.

```
kubectl get secret my-creds \
    -o jsonpath="{.data.username}" \
    | base64 --decode
```

We used `jsonpath` to filter the output so that only the `username` data is retrieved. Since the value is encoded, we piped the output to `base64` command that decoded it for us. The result is `jdoe`.

Similarly, the command that will retrieve and decode the second Secret data is as follows.

```
kubectl get secret my-creds \
    -o jsonpath="{.data.password}" \
    | base64 --decode
```

The **output** is `incognito`.

In the next lesson, we will explore how to mount generic Secrets.