# Kubernetes Secrets

## Create secret

This example shows how Secrets can be used to allow Pods access to a database.

Create secret files

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

Now let’s create the secrets objects out of the files.

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

Confirm secret was created

```
kubectl get secrets
```

Now  describe the secret

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

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

Type:            Opaque

Data
====
password.txt:    12 bytes
```

```
username.txt:    5 bytes
```

Notice that neither `get` nor `describe` shows the contents of the file. This is to protect the secret from being exposed, or being stored in a terminal log.

## Create Secret Manually

You can also create a secret object in a JSON or YAML file and create the object.

To do this each item must be base64 encoded

```
echo -n "admin" | base64

echo -n "1f2d1e2e67df" | base64
```

Now let's write a secret object using encoded data from above.

```
apiVersion: v1

kind: Secret

metadata:

  name: mysecret

type: Opaque

data:

  username: YWRtaW4=

  password: MWYyZDFlMmU2N2Rm
```

Create a secret from above file

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

### Decoding Secrets

Using `kubectl` and `base64` we can decode our secrets

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

We will get output similar to

```
apiVersion: v1
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
kind: Secret
metadata:
  creationTimestamp: 2016-01-22T18:41:56Z
  name: mysecret
  namespace: default
  resourceVersion: "164619"
  selfLink: /api/v1/namespaces/default/secrets/mysecret
  uid: cfee02d6-c137-11e5-8d73-42010af00002
type: Opaque
```

Now let’s decode it

```
echo "MWYyZDFlMmU2N2Rm" | base64 --decode
1f2d1e2e67df
```

## Using Secrets

Secrets can be mounted as data volumes or be exposed as environment variables to be used by a container in a pod. They can also be used by other parts of the system, without being directly exposed to the pod. For example, they can hold credentials that other parts of the system should use to interact with external systems on your behalf.

To use Secrets as files from a Pod we can use the following YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
```

```
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    secret:
      secretName: mysecret
```

Inside the container that mounts a secret volume, the secret keys appear as files and the secret values are base-64 decoded and stored inside these files. This is the result of commands executed inside the container from the example above:

List secrets

```
kubectl exec -it mypod ls /etc/foo
```

Output username

```
kubectl exec -it mypod cat /etc/foo/username
```

Output password

```
kubectl exec -it mypod cat /etc/foo/password
```

### Secrets as Environment Variables

Use the following YAML to create a Pod that uses environment variable secret

```
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
```

Now let’s confirm secret is available in the container.

Log into container

```
kubectl exec -it secret-env-pod bash
```

Echo the environment variables

```
echo $SECRET_USERNAME
echo $SECRET_PASSWORD
```

## Lab complete