In this scenario, you'll learn how manage secrets using Kubernetes. Kubernetes allows you to create secrets that are mounted to a pod via environment variables or as a volume.

This allows secrets, such as SSL certificates or passwords, to only be managed via an infrastructure team in a secure way instead of having the passwords stored within the application's deployment artefacts.

Step 2 - Create Secrets

Kubernetes requires secrets to be encoded as Base64 strings.

Using the command line tool we can create the Base64 strings and store them as variables to use in a file. `username=$(echo -n "admin" | base64) password=$(echo -n "a62fjbd37942dcs" | base64)`

The secret is defined using *yaml*. Below we'd using the variables defined above and providing them with friendly labels which our application can use. This will create a collection of key/value secrets that can be accessed via the name, in this case *test-secret*

```
echo "apiVersion: v1 kind: Secret metadata: name: test-secret type: Opaque
data: username: $username password: $password" >> secret.yaml
```

This *yaml* file can be used to with Kubectl to create our secret. When launching pods that require access to the secret we'll refer to the collection via the friendly-name.

Task: Create the secret

Use kubectl to create our secret.

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

The following command allows you to view all the secret collections defined.

```
kubectl get secrets
```

In the next step we'll use these secrets via a Pod.

```
controlplane $ kubectl create -f secret.yaml
secret/test-secret created
controlplane $ kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-mpdbz   kubernetes.io/service-account-token   3      4m46s
test-secret           Opaque                                2      5s
controlplane $
```

Step 3 - Consume via Environment Variables

In the file *secret-env.yaml* we've defined a Pod which has environment variables populated from the previously created secret.

View the file using `cat secret-env.yaml`

To populate the environment variable we define the name, in this case SECRET_USERNAME, along with the name of the secrets collection and the key which containers the data.

```
controlplane $ cat secret-env.yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
    - name: mycontainer
      image: alpine:latest
      command: ["sleep", "9999"]
      env:
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: test-secret
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: test-secret
              key: password
  restartPolicy: Never
```

Task

Launch the Pod using `kubectl create -f secret-env.yaml`

Once the Pod started, you output the populated environment variables. `kubectl exec -it secret-env-pod env | grep SECRET_`

Kubernetes decodes the base64 value when populating the environment variables. You should see the original username/password combination we defined. These variables can now be used for accessing APIs, Databases etc.

You can check the status of a Pod using `kubectl get pods`.

In the next step we'll mount the secrets as files.

Step 4 - Consume via Volumes

The use of environment variables for storing secrets in memory can result in them accidentally leaking. The recommend approach is to use mount them as a Volume.

The Pod specification can be viewed using `cat secret-pod.yaml`.

To mount the secrets as volumes we first define a volume with a well-known name, in this case, secret-volume, and provide it with our stored secret.

```
controlplane $ cat secret-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-vol-pod
spec:
  volumes:
  - name: secret-volume
    secret:
      secretName: test-secret
  containers:
    - name: test-container
      image: alpine:latest
      command: ["sleep", "9999"]
      volumeMounts:
          - name: secret-volume
            mountPath: /etc/secret-volume
```

Task

Create our new Pod using `kubectl create -f secret-pod.yaml`

Once started you can interact with the mounted secrets. For example, you can list all the secrets available as if they're regular data. For example `kubectl exec -it secret-vol-pod ls /etc/secret-volume`

Reading the files allows us to access the decoded secret value. To access username we'd use `kubectl exec -it secret-vol-pod cat /etc/secret-volume/username`

For the password, we'd read the password file `kubectl exec -it secret-vol-pod cat /etc/secret-volume/password`