## **What Are Kubernetes Secrets?**

The software development life cycle typically consists of working with [multiple environments](https://www.ibm.com/docs/en/urbancode-deploy/7.0.2?topic=applications-application-environments). As a result, you may want to pass dynamic values to your application at runtime to control its behavior. This kind of application configuration in Kubernetes can be carried out using either ConfigMaps or Secrets.

ConfigMaps and Secrets are distinct Kubernetes objects that have a similar function but serve different purposes.

Secrets handle small amounts of sensitive data to reduce the risk of accidental exposure of confidential information. Secret data should be stored and handled in a way that can be easily hidden and [possibly encrypted](https://www.containiq.com/post/kubernetes-security-best-practices) at rest if the environment is configured as such. Kubernetes is [insecure by default](https://cloud.redhat.com/blog/most-common-kubernetes-security-issues-and-concerns-to-address). This extends to Secrets because they are not encrypted. As soon as the Secret is injected into the Pod, the Pod itself can see the Secret data in plain text.

Secrets are stored inside the Kubernetes data store (i.e., [an etcd database](https://www.containiq.com/post/etcd)) and are created before they can be used inside a Pods manifest file. Furthermore, Secrets have a size limit of 1 MB. When it comes to implementation, you can either mount Secrets as volumes or expose them as environment variables inside the Pod manifest files.

There are a few different types of Secrets in Kubernetes:

* ****Opaque:**** The default Secret type if one isn’t specified in the manifest configuration file. It allows you to provide arbitrary configuration data in key-value pairs.
* ****Service account token:**** These store a token that identifies a specific service account. Of course, don’t forget to set the kubernetes.io/service-account.name annotation to an existing service account.
* ****Docker config:**** This stores the credentials for a specific Docker registry for container images.
* ****Basic authentication:**** These store basic authentication credentials. Note that their data fields need to include the username and password keys.
* ****SSH authentication:**** These store SSH credentials. You’ll have to specify an ssh-privatekey key-value pair in the data or stringData field.
* ****TLS:**** These Secrets store certificates and the associated keys used for TLS. You need to make sure the tls.key and the tls.crt keys are included in the data field of the Secret’s configuration.
* ****Bootstrap token:**** These are tokens used during the node bootstrap process, used to sign ConfigMaps.

## **Implementing Secrets**

In this section, you will create a Secret and mount it using the two approaches mentioned above.

### **Connect to a Kubernetes Cluster**

The first step is to ensure that you are connected to a [Kubernetes cluster](https://www.containiq.com/post/kubernetes-cluster), either local or remote. The cluster setup and configuration used in this post can be found in [this public repository](https://github.com/LukeMwila/amazon-eks-cluster).

You can verify the context of your cluster connection with the following command:

kubectl config current-context

### **Create a Secret Manifest File**

Once your connection is established, proceed to create a manifest file for a Secret. The Secret will contain key-value pairs for a username and password.

You can base64-encode the values for your secret as follows:

echo -n 'jsmith' | base64

echo -n 'mysecretpassword' | base64

Note that the outputs from the previous commands need to be pasted into the YAML file in the next section.

apiVersion: v1kind: Secretmetadata:

name: my-secrettype: Opaquedata:

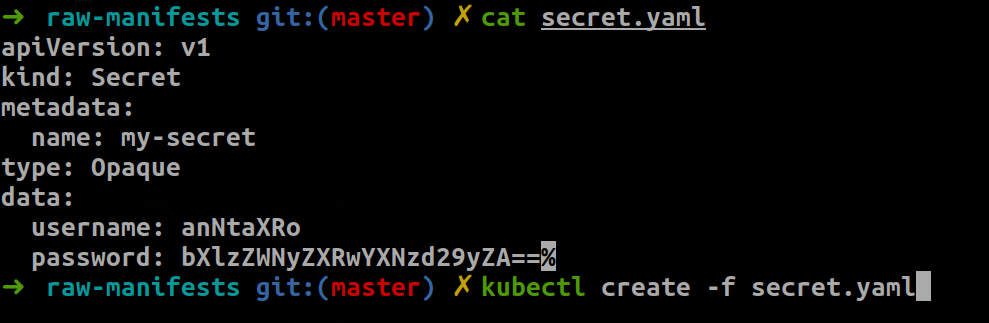
username: anNtaXRo

password: bXlzZWNyZXRwYXNzd29yZA==

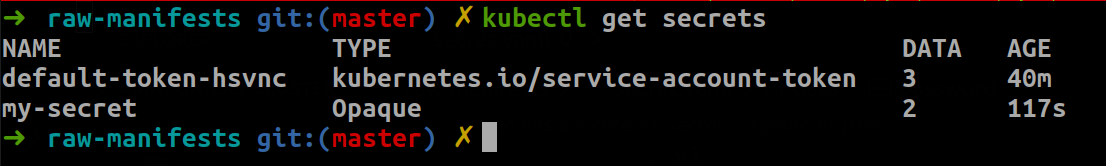
### **Create a Secret in a Kubernetes Cluster**

To create the Secret, use the kubectl command to reference the manifest file you just created. The request will be sent to the API Server in the [Kubernetes Control Plane](https://www.containiq.com/post/kubernetes-control-plane) for the request to be actioned. Afterward, the data will be stored in the etcd data store of your cluster.

kubectl create -f secret.yaml



Create Secret



Get Secret

### **Mount Secret Values with Environment Variables**

In this step, you will create a Pod that will expose the Secret values as environment variables in a container at runtime. Once the Pod has been created, check the logs to verify that the values are accessible to the container based on an executed shell command.

apiVersion: v1kind: Podmetadata:

name: env-podspec:

containers:

- name: express-test

image: lukondefmwila/express-test:latest

command: ['sh', '-c', 'echo "Username: $USER" "Password: $PASSWORD"']

env:

- name: USER

valueFrom:

secretKeyRef:

name: my-secret

key: username

- name: PASSWORD

valueFrom:

secretKeyRef:

name: my-secret

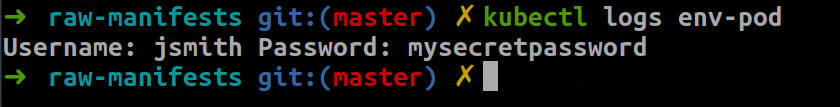
key: password

Create the above Pod manifest file and execute the following command to create it in your cluster:

kubectl create -f env-pod.yaml

Once the Pod has been created, you can check the logs by running kubectl logs <name-of-pod>.

‍



Pod Logs

‍

### **Mount Secret Values with Volumes**

Secrets can also be passed to containers in the form of mounted volumes. This will cause the configuration data to appear in files available to the container file system. Each top-level key in the configuration data will appear as a file containing all keys below that top-level key.

apiVersion: v1kind: Podmetadata:

name: volume-podspec:

containers:

- name: express-test

image: lukondefmwila/express-test:latest

volumeMounts:

- name: secret-volume

mountPath: /etc/config/secret

volumes:

- name: secret-volume

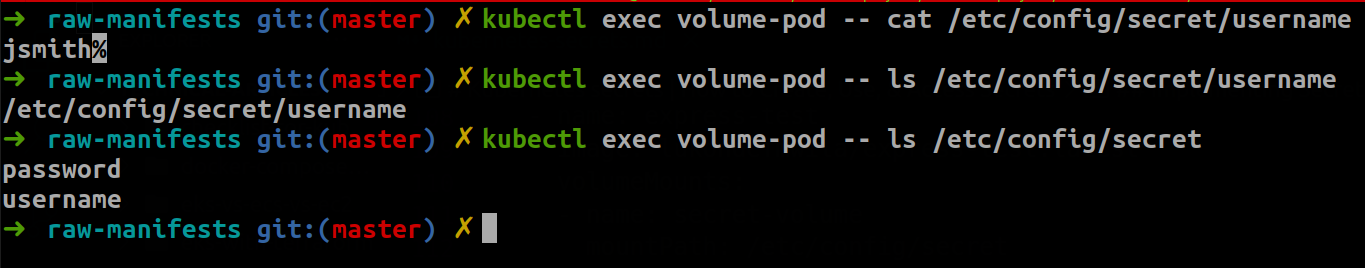
secret:

secretName: my-secret

To test the Pod’s accessibility of the volume mounted secret, you can [connect to the container](https://www.containiq.com/post/using-kubectl-exec-shell-commands-examples) and run read commands in the volume directory.

kubectl exec volume-pod -- ls /etc/config/secret

kubectl exec volume-pod -- cat /etc/config/secret/username



Volume Mounted Secrets

‍

## **Conclusion**

As we’ve discussed in this article, software teams typically configure computing environments to drive certain behavior in the development process. When developing in a Kubernetes environment, there are two ways of accomplishing this: ConfigFile and Secrets. If you have truly sensitive data, you should now be well equipped to create, use, and access Secrets in your container workloads.