

Mounting Generic Secrets

In this lesson, we will mount the generic Secret to secure the deployed Jenkins.

WE'LL COVER THE FOLLOWING ^

- Looking into the Definition
- Applying the Definition
- Verification

Looking into the Definition

Let's see how we could mount the Secret we created.

```
cat secret/jenkins.yml
```



The **output**, limited to the relevant parts, is as follows.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins
spec:
  ...
  template:
    ...
    spec:
      containers:
        - name: jenkins
          image: vfarci/jenkins
          env:
            - name: JENKINS_OPTS
              value: --prefix=/jenkins
          volumeMounts:
            - name: jenkins-home
              mountPath: /var/jenkins_home
            - name: jenkins-creds
              mountPath: /etc/secrets
      volumes:
        - name: jenkins-home
          emptyDir: {}
        - name: jenkins-creds
```



```
- name: jenkins-creds
secret:
  secretName: my-creds
  defaultMode: 0444
  items:
    - key: username
      path: jenkins-user
    - key: password
      path: jenkins-pass
...
```

Line 19-20: We added `jenkins-creds` that mounts the `/etc/secrets` directory.

Line 24-26: The `jenkins-creds` Volume references the Secret named `my-creds`.

Line 27: Since we want the process inside the container to only read the Secret, we set the `defaultMode` to `0444`. That will give read permissions to everyone. Typically, we'd set it to `0400`, thus giving the read permissions only to the `root` user. However, since the Jenkins image uses the `jenkins` user, we gave read permissions to everyone instead of only to the `root` user.

Line 29-32: Finally, since the image expected files named `jenkins-user` and `jenkins-pass`, we made explicit paths. Otherwise, Kubernetes would create files `username` and `password`.

Applying the Definition

Let's apply the new definition.

```
kubectl apply -f secret/jenkins.yml

kubectl rollout status deploy jenkins
```



We applied the definition and waited until the new objects were rolled out.

Verification

Now we can check whether the correct files are indeed stored in the `/etc/secrets` directory.

```
POD_NAME=$(kubectl get pods \
  -l service=jenkins,type=master \
  -o jsonpath="{.items[*].metadata.name}")

kubectl exec -it $POD_NAME \
  -- ls /etc/secrets
```



The **output** of the latter command is as follows.

```
jenkins-pass jenkins-user
```



The files we need are indeed injected. To be on the safe side, we'll also check the content of one of them.

```
kubectl exec -it $POD_NAME \  
-- cat /etc/secrets/jenkins-user
```



The **output** is **jdoe**, the username of our newly deployed Jenkins.

Finally, let's confirm that the application is indeed secured.

```
open "http://$(minikube ip)/jenkins"
```



You'll see that, this time, the link to create new jobs is gone.

Please use **jdoe** and **incognito** if you'd like to login to your newly deployed and (more) secured Jenkins.

The next lesson is a brief comparison of Kubernetes Secrets with ConfigMaps.