# Exploring Built-In Secrets

In this lesson, we will go through the built-in Kubernetes Secrets.

WE'LL COVER THE FOLLOWING



## Creating Jenkins Objects #

We'll create the same Jenkins objects we defined earlier.

```
kubectl create \
    -f secret/jenkins-unprotected.yml \
    --record --save-config

kubectl rollout status deploy jenkins
```

We created an Ingress, a Deployment, and a Service object. We also executed the `kubectl rollout status` command that will tell us when the deployment is finished and this may take some time.

The `secret/jenkins-unprotected.yml` definition does not use any new feature so we won't waste time going through the YAML file. Instead, we'll open Jenkins UI in a browser.

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

Upon closer inspection, you'll notice that there is no login button. Jenkins is currently unprotected. The image does allow the option to define an initial administrative username and password.

If the files `/etc/secrets/jenkins-user` and `/etc/secrets/jenkins-pass` are present, the init script will read them, and use the content inside those files to

define the username and the password.

Since we're already familiar with ConfigMaps, we could use them to generate those files. However, since the user and the password should be better protected than other configuration entries, we'll switch to Secrets.

If you're interested in details, please explore the jenkins/Dockerfile from the vfarcic/docker-flow-stack repository. The important part is that it expects `/etc/secrets/jenkins-user` and `/etc/secrets/jenkins-pass` files. If we can provide them, in a relatively secure manner, our Jenkins will be (more) secured by default.

## The Built-In Secrets #

We'll start by checking whether we already have some Secrets in the cluster.

```
kubectl get secrets
```

The **output** is as follows.

```
NAME                  TYPE                                  DATA AGE
default-token-l9fhk kubernetes.io/service-account-token 3    32m
```

We did not create any Secret, and yet one is available in the system.

The `default-token-l9fhk` Secret was created automatically by Kubernetes. It contains credentials that can be used to access the API.

Moreover, Kubernetes automatically modifies the Pods to use this Secret. Unless we tweak Service Accounts, every Pod we create will have this Secret. Let's confirm that is indeed true.

```
kubectl describe pods
```

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

```
...
    Mounts:
      /var/jenkins_home from jenkins-home (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-l9fhk (ro)
```

```
...
Volumes:
  jenkins-home:
    Type:    EmptyDir (a temporary directory that shares a pods lifetime)
    Medium:
  default-token-l9fhk:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-l9fhk
    Optional:    false
...
```

We can see that two volumes are mounted. The first one (`/var/jenkins_home`) was defined by us. It's the same mount volume we used in the previous chapter, and it is meant to preserve Jenkins' state by mounting its home directory.

The second mount is the more interesting one. We can see that it references the auto-generated Secret `default-token-l9fhk` and that it mounts it as `/var/run/secrets/kubernetes.io/serviceaccount`. Let's take a look at that directory.

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

kubectl exec -it $POD_NAME -- ls \
    /var/run/secrets/kubernetes.io/serviceaccount
```

The **output** is as follows.

```
ca.crt  namespace  token
```

By auto-mounting that Secret, we got three files. They are required if we'd like to access the API server from within the containers. `ca.crt` is the certificate, the `namespace` contains the namespace the Pod is running in, and the last one is the token we'd need to establish communication with the API.

We won't go into examples that prove those files can be used to access the API server securely. Just remember that if you ever need to do that, Kubernetes has you covered through that auto-generated Secret.

Let's get back to the task at hand. We want to make Jenkins more secure by providing it with an initial username and password.

In the next lesson, we'll explore how to create generic Secrets.