SECRET MANAGEMENT

Azure Key Vault

BY
Timothy Fabelurin

Kubernetes has "Secret" object which contains sensitive data like key, password, or token. The data save are usually meant for specific pods or container images and this makes the application code to devoid any confidential data. However, this is secret is usually not encrypted by default. Anyone that has access to the API sever data store (etcd) can compromise the data. To deal with this issue, it is recommended to have the secret managed more securely by encrypting the secret and enabling RBAC. Azure key vault was in used in this implementation to store and retrieve digital keys and credentials. Azure Key Vault logging can be enabled to monitor how and who accessed the keys (this was implemented in this project).

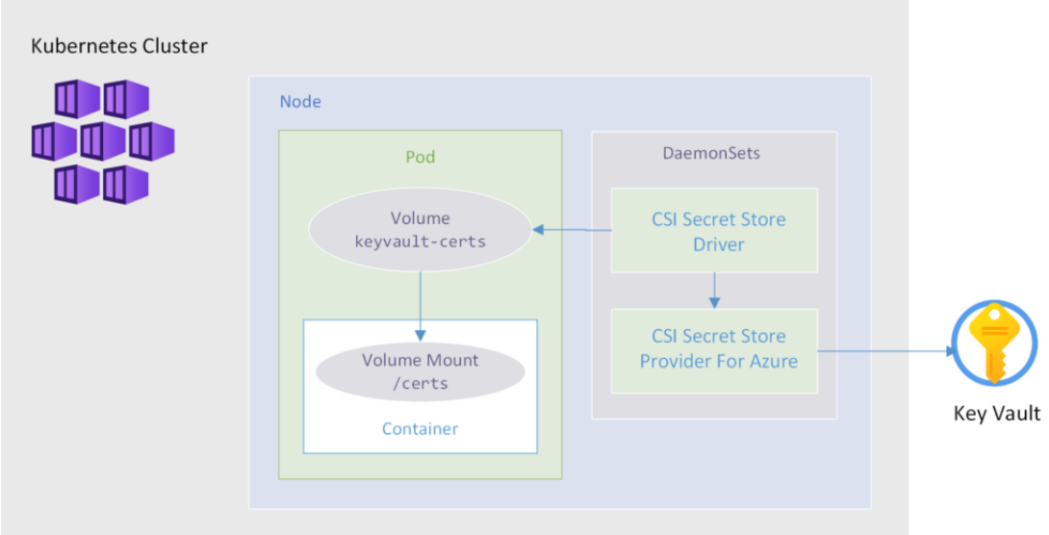


Figure 4.1 Flow from key Vault to a pod and volume mounted on container

Using **pod managed identity** is also an important feature available in Azure cloud. With this feature, a pod can authenticate itself against azure services that support manage identity such as SQL or Storage. Though this was not implemented but it is very vital feature that helps in the resolving issues surrounding stored tokens and keys compromise in cloud. The flow below described how it works.

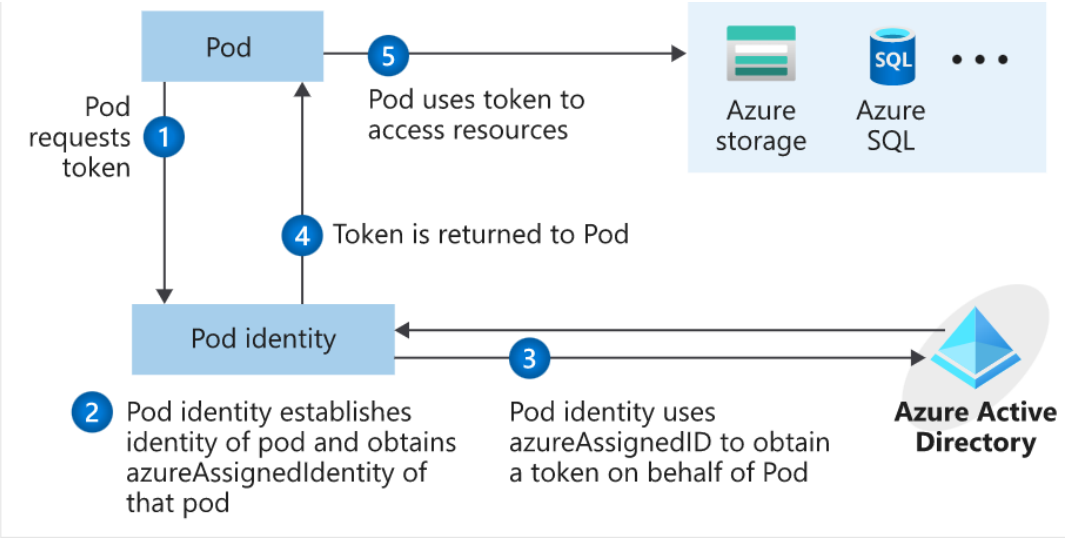


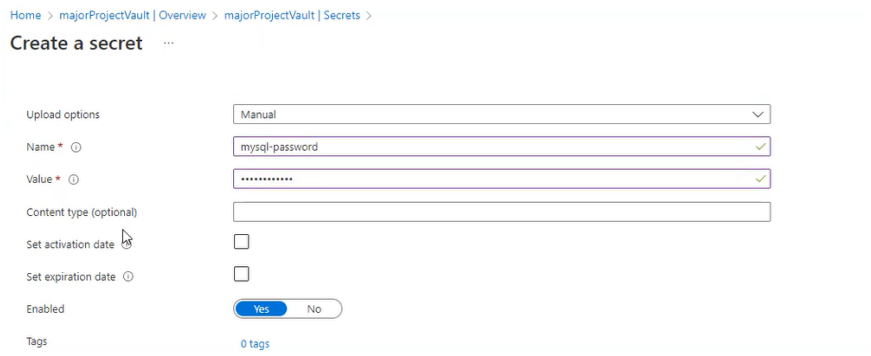
*Figure 4.**Error! No text of specified style in document.**..2 Pod Manage Identity flow (Rhoads, 2021)*

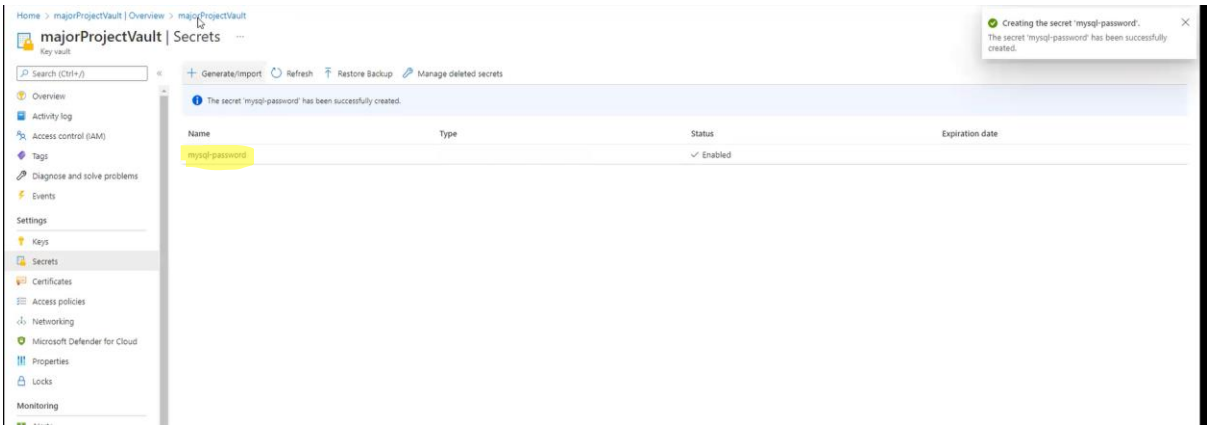
Process

Where the passwords and secrets are stored is very important. The secret encoded in base-64 is used when setting up the cluster, the secret can be easily decoded using numerous online tools (Choudhary, 2022).

This was reviewed and Azure key vault was implemented. With this change in configuration, resources can retrieve the secret in the key Vault when needed which has better encryption and more securely saved.

1. Key Vault named ***majorProjectVault*** was created and then secrete was manually created the name was ***mysql-password*** and the value saved in the vault.

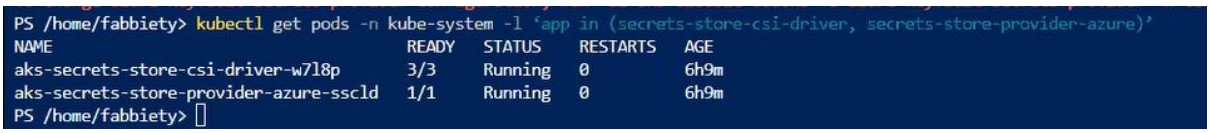




2. Secret Store CSI Driver support was enabled. The function of the CSI driver is for allowing the integration of the Azure keyVault with the AKS cluster. The following commands were run to enable the addons:

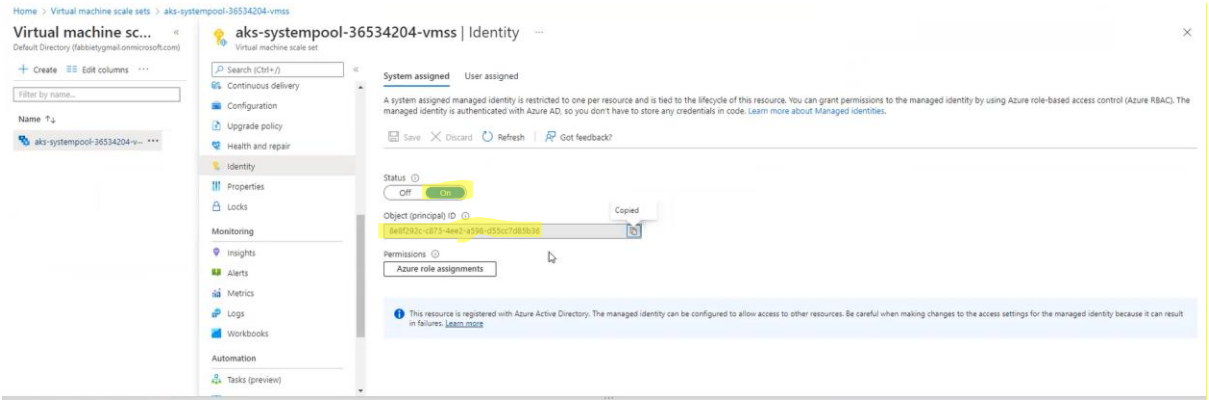
   az aks enable-addons --addons azure-keyvault-secrets-provider --name terraform-aks-secure-cluster --resource-group terraform-aks-secure

   Verification of the CSI installation captured below.

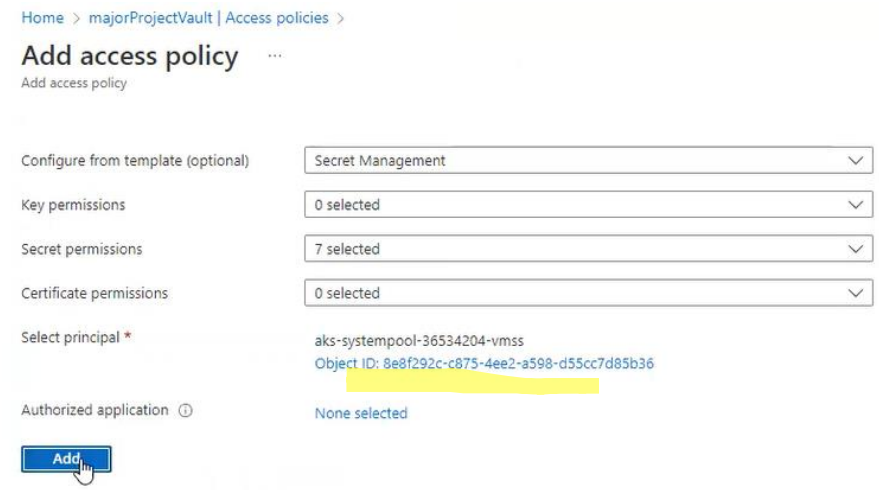


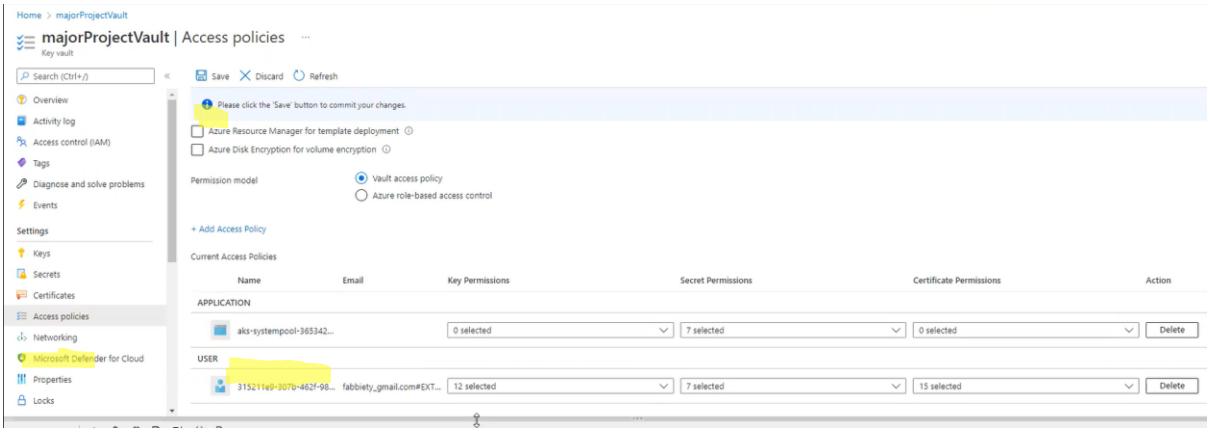
3. The aim at this point was to provide virtual machine scale set pool whict the AKS cluster was using an Identity principal name. This principal name will then be assigned permissions to access the secrets saved in the key Vault.

   The image below captured the identity enabled and the principal object name.



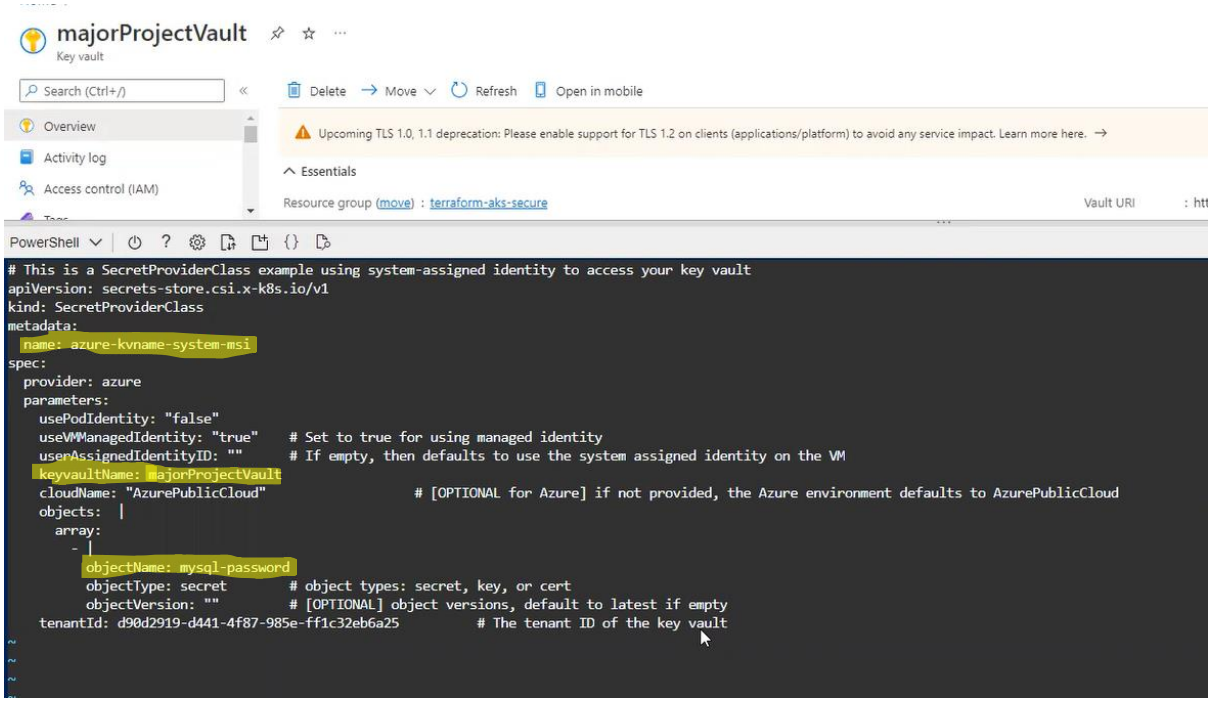
4. To grant the identity permissions that enable it to read and view key vault contents, the below two images shows the process.



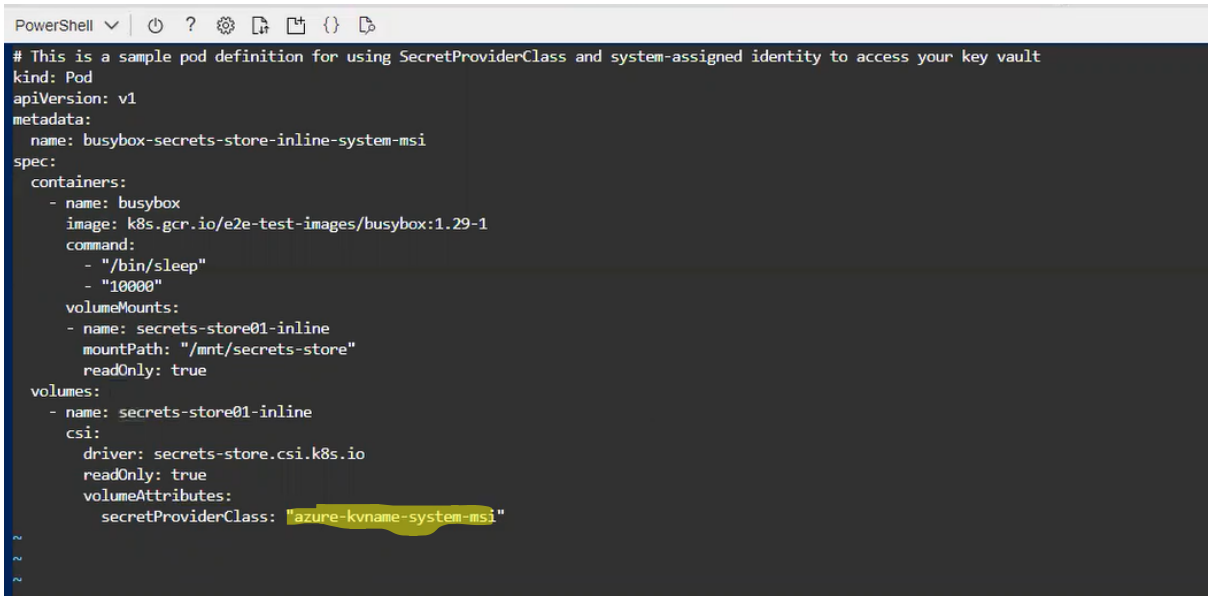


5. For the secrete to actually get access, Secret Provider Class had to be explicity defined. This presents the name of the secrets that need to be imported from the perticular key Vault. The yaml file was created and thereafter applied to the target cluster using this command

kubectl apply -f secretproviderclass.yaml



6. Finally, to create the secret in the kubernets, a pod that mounted a volume using CSI and referenced the created Secrete Provider Class was created.



```
PS /home/fabbiety> kubectl apply -f pod.yaml
pod/busybox-secrets-store-inline-system-msi created
PS /home/fabbiety> kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
busybox-secrets-store-inline-system-msi   1/1     Running   0          4s
PS /home/fabbiety> kubectl exec busybox-secrets-store-inline -- ls /mnt/secrets-store/
Error from server (NotFound): pods "busybox-secrets-store-inline" not found
PS /home/fabbiety>
```

Validating the secrets from the created pod

```
PS /home/fabbiety> kubectl exec busybox-secrets-store-inline-system-msi -- ls /mnt/secrets-store/
mysql-password
PS /home/fabbiety> kubectl exec busybox-secrets-store-inline-system-msi -- cat /mnt/secrets-store/mysql-password
Password@123
PS /home/fabbiety>
```

This is the recommended way of managing secretes in AKS cluster (and cloud in general) with all values stored in key vault. The content of the vault can be key or certificate or as seen in the demonstration above, secrets. Only specific secret in a specific key vault can be accessed after permission is expressly given. The whole management of the public and private keys are done by the platform provider.