Managing secrets and using secrets in the Azure Kubernetes environment is a very important security aspect. While working with Kubernetes and secrets in particular you will know that they are not secure by default. In fact, they are just base64 encoded strings which can be easily decoded. You can use public and private keys to keep your secrets safe and then allow Kubernetes to access them but that will be an extra overhead to manage them.

Azure supports the possibility to get secrets into an Azure key Vault, from AKS, by using the Secret Store CSI (Container Storage Interface) Driver. AKS clusters may need to store and retrieve secrets, keys, and certificates. The Secrets Store CSI Driver provides cluster support to integrate with Key Vault. When enabled and configured secrets, keys, and certificates can be securely accessed from a pod. In this article, we will discuss a secure way of using secrets that are stored in Azure Key Vault into your Azure Kubernetes Cluster(AKS).

**Create Azure Key-Vault and Secret**

To create an Azure Key Vault and add a secret you need to run the following commands.

az keyvault create — name “aksdemocluster-kv” — resource-group “aksdemo-rg” — location australiaeastaz keyvault secret set — vault-name “aksdemocluster-kv” — name “mysql-password” — value “Test123”

**Enable Secrets Store CSI Driver support**

To upgrade an existing AKS cluster with Azure Key Vault Provider for Secrets Store CSI Driver capability run the following command:

az aks enable-addons --addons azure-keyvault-secrets-provider --name aksdemocluster --resource-group aksdemocluster-rg

**Verify the Secrets Store CSI Driver installation**

Verify that the installation is finished by listing all pods that have the secrets-store-csi-driver and secrets-store-provider-azure labels in the kube-system namespace, and ensure that your output looks similar to the output shown here:

kubectl get pods -n kube-system -l ‘app in (secrets-store-csi-driver, secrets-store-provider-azure)’  
NAME READY STATUS RESTARTS  
aks-secrets-store-csi-driver-4vpkj 3/3 Running 2   
aks-secrets-store-csi-driver-ctjq6 3/3 Running 2   
aks-secrets-store-csi-driver-tlvlq 3/3 Running 2   
aks-secrets-store-provider-azure-5p4nb 1/1 Running 0   
aks-secrets-store-provider-azure-6pqmv 1/1 Running 0   
aks-secrets-store-provider-azure-f5qlm 1/1 Running 0

**Verify system-assigned identity on VMs**

Verify that your virtual machine scale set or availability set nodes have their own system-assigned identity:

az vmss identity show -g MC\_aksdemocluster-rg\_aksdemocluster\_australiaeast -n aks-agentpool-32528728-vmss -o yaml  
az vm identity show -g <resource group> -n <vm name> -o yaml

**Assigns Permissions**

To grant your identity permissions that enable it to read your key vault and view its contents, run the following commands:

# set policy to access secrets in your key vault az keyvault set-policy -n <keyvault-name> — secret-permissions get — spn <identity-principal-id>

**Create Secret Provider Class**

Now it’s time to actually get secrets from the Key Vault. We have to be explicit define and create a Secret Provider Class, where we define all the secrets we need to import from the Key Vault.

# This is a SecretProviderClass example using system-assigned identity to access your key vault  
apiVersion: secrets-store.csi.x-k8s.io/v1  
kind: SecretProviderClass  
metadata:  
 name: azure-kvname-system-msi  
spec:  
 provider: azure  
 parameters:  
 usePodIdentity: "false"  
 useVMManagedIdentity: "true" # Set to true for using managed identity  
 userAssignedIdentityID: "" # If empty, then defaults to use the system assigned identity on the VM  
 keyvaultName: aksdemocluster-kv  
 cloudName: "AzurePublicCloud" # [OPTIONAL for Azure] if not provided, the Azure environment defaults to AzurePublicCloud  
 objects: |  
 array:  
 - |  
 objectName: mysql-password  
 objectType: secret # object types: secret, key, or cert  
 objectVersion: "" # [OPTIONAL] object versions, default to latest if empty  
 tenantId: XXXXXXXXXXXX # The tenant ID of the key vault

Apply the SecretProviderClass to your cluster:

kubectl apply -f secretproviderclass.yaml

**Create POD with Secrets**

By deploying the Secret Provider Class, the secrets will **not** be created in Kubernetes yet. This will only happen with the first pod, which mounts a volume utilizing CSI and referencing our Secret Provider Class. Also, the pod must make use of the selector that we specified in the Azure Identity Binding.

# This is a sample pod definition for using SecretProviderClass and system-assigned identity to access your key vault  
kind: Pod  
apiVersion: v1  
metadata:  
 name: busybox-secrets-store-inline-system-msi  
spec:  
 containers:  
 - name: busybox  
 image: k8s.gcr.io/e2e-test-images/busybox:1.29-1  
 command:  
 - "/bin/sleep"  
 - "10000"  
 volumeMounts:  
 - name: secrets-store01-inline  
 mountPath: "/mnt/secrets-store"  
 readOnly: true  
 volumes:  
 - name: secrets-store01-inline  
 csi:  
 driver: secrets-store.csi.k8s.io  
 readOnly: true  
 volumeAttributes:  
 secretProviderClass: "XXXXXXXXX"

**Validate the secrets**

After the pod starts, the mounted content at the volume path that you specified in your deployment YAML is available.

## show secrets held in secrets-store   
kubectl exec busybox-secrets-store-inline -- ls /mnt/secrets-store/   
   
## print a test secret 'ExampleSecret' held in secrets-store   
kubectl exec busybox-secrets-store-inline -- cat /mnt/secrets-store/mysql-password

As you can see, it is a recommended way to manage secrets in AKS Cluster and all the values will be stored in a key vault (key, secret or certificate). The benefit here is that you can provide access to a specific secret in a specific key vault. And all of this, without managing the public and private keys.