Jonathan

Follow 29 Followers About

Azure Kubernetes Service (AKS) with Azure Key Vault (AKV) Part 1 — Secrets Store CSI

Jonathan Apr 7 · 5 min read

The title of this article is awfully long, but the purpose for having both secrets store CSI and AKV provider in AKS environment is really simple, letting AKS to get AKV's resource, including secrets, certificates and keys, as native resource. In order to achieve that, we would have to implement pod identity or Azure active directory service principal (AAD SP) as the object that has sufficient permissions. We would use AAD SP for the following content.

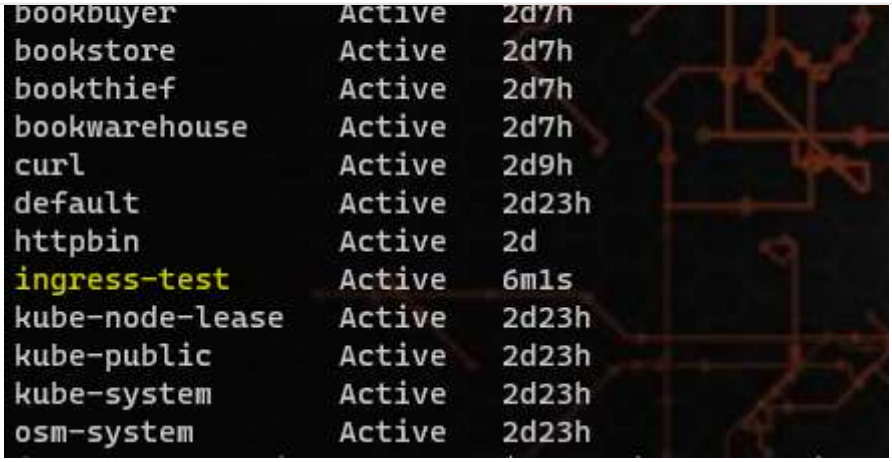
Step-By-Step Guidance

1. Create an AKS resource: Create a resource group then an AKS cluster
2. Create an AKV resource

```
az keyvault create -g <resource group name> -n <key vault name> --
location <Ex: westus2, eastus...>
```

3. Create a namespace to store every resource for this demonstration

```
kubectl create ns <namespace name>
```

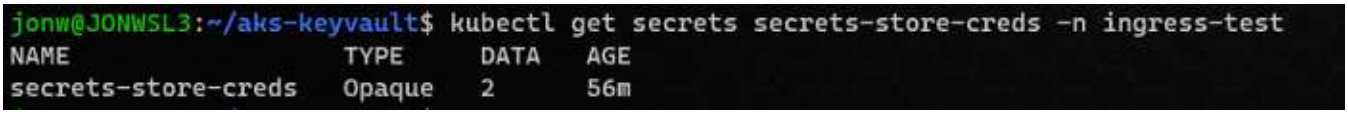


4. Create an AAD SP and set appropriate permissions to it to manage AKV

```
export SERVICE_PRINCIPAL_CLIENT_SECRET="$(az ad sp create-for-rbac --skip-assignment --name http://secrets-store-test --query 'password' -otsv)"

export SERVICE_PRINCIPAL_CLIENT_ID="$(az ad sp show --id http://secrets-store-test --query 'appId' -otsv)"

az keyvault set-policy -n <AKV name> --secret-permissions get --spn ${SERVICE_PRINCIPAL_CLIENT_ID}
```

5. Create a secret within the AKS cluster as the identity managing AKV in the future steps. Label the secret.

```
# Create a secret with AAD SP client ID and secret
kubectl create secret generic secrets-store-creds --from-literal
clientid=${SERVICE_PRINCIPAL_CLIENT_ID} --from-literal
clientsecret=${SERVICE_PRINCIPAL_CLIENT_SECRET} -n <namespace name>

# Label the just-created secret
kubectl label secret secrets-store-creds secrets-
store.csi.k8s.io/used=true

# Check whether the secret has been created in the environment
kubectl get secrets secrets-store-cred -n <namespace name>
```

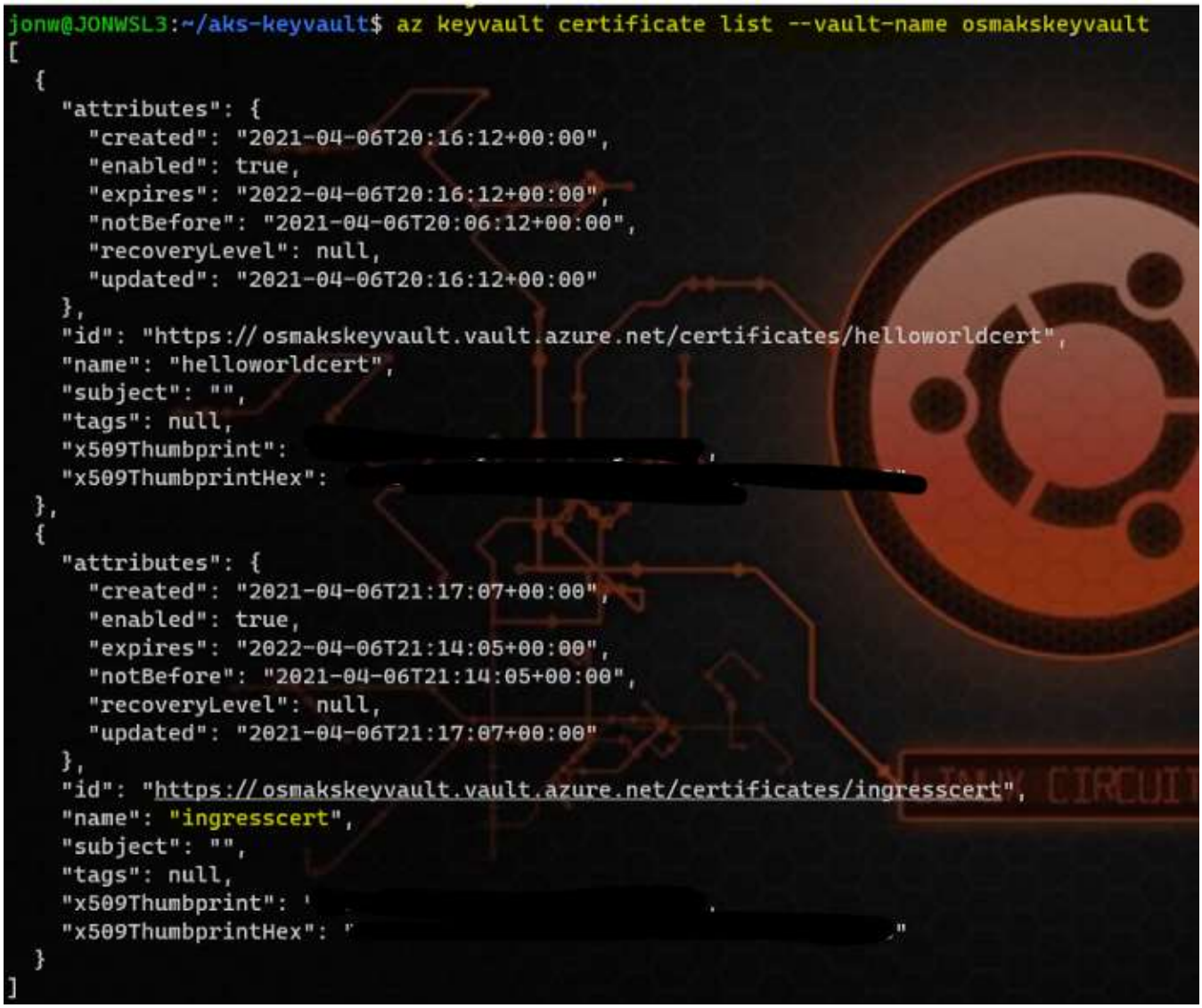


6.. Generate a TLS certificate in the Linux environment

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
    -out ingress-tls.crt \
    -keyout ingress-tls.key \
    -subj "/CN=demo.test.com/O=ingress-tls"
```

7. Import the TLS certificate in AKV

```
# export the a .pfx file with both .crt and .key
# skip Password prompt
openssl pkcs12 -export -in ingress-tls.crt -inkey ingress-tls.key  -
out <certificate name>.pfx

az keyvault certificate import --vault-name <AKV name> -n
<certificate name> -f <certificate name>.pfx
```



8. Deploy SecretProviderClass. Create a new file named “secretproviderclass.yaml” and create it via “kubectl apply -f secretproviderclass.yaml”.

```
# the content of secretproviderclass.yaml
apiVersion: secrets-store.csi.x-k8s.io/v1alpha1
kind: SecretProviderClass
metadata:
  name: azure-tls
spec:
  provider: azure
  secretObjects:
  - secretName: ingress-tls-csi
    type: kubernetes.io/tls
    data:
    - objectName: <certificate name>
      key: tls.key
    - objectName: <certificate name>
```

```
    keyvaultName: <AKV name>              # the name of the keyvault
    objects: |
      array:
        - |
          objectName: <certificate name>
          objectType: secret
    tenantId: <AAD tenant ID>       # the tenant ID of the KeyVault

# create the secret provider class
kubectl apply -f secretproviderclass.yaml -n <namespace name>

# Check whether secretproviderclass has been created successfully
kubectl get secretproviderclasss -n <namespace name>
```

```
jonw@JONWSL3:~/aks-keyvault$ kubectl get secretproviderclass -n ingress-test
NAME        AGE
azure-tls   59m
```

9. Create NGINX Ingress Controller

```
# add and HELM repo
helm repo add ingress-nginx https://kubernetes.github.io/ingress-
nginx

# update HELM repo
helm repo update

# create the NGINX Ingress Controller with HELM
helm install ingress-nginx/ingress-nginx --generate-name \
    --namespace <namespace name> \
    --set controller.replicaCount=2 \
    --set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
    --set
defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux \
    -f - <<EOF
controller:
  extraVolumes:
      - name: secrets-store-inline
        csi:
          driver: secrets-store.csi.k8s.io
          readOnly: true
          volumeAttributes:
            secretProviderClass: "azure-tls"
          nodePublishSecretRef:
            name: secrets-store-creds
  extraVolumeMounts:
      - name: secrets-store-inline
        mountPath: "/mnt/secrets-store"
```

```
# Check whether all related resources are up and running, especially
the secret "ingress-tls-csi"
kubectl get deploy,pod,svc,ing,secret -n <namespace name>
```

10. Create test applications and Ingress

```
# create app 1
kubectl apply -f https://raw.githubusercontent.com/Azure/secrets-
store-csi-driver-provider-
azure/5eff51f5a04b5e91db5c18080c30316a5dee772a/docs/sample/ingress-
controller-tls/deployment-app-one.yaml -n <namespace name>

# create app 2
kubectl apply -f https://raw.githubusercontent.com/Azure/secrets-
store-csi-driver-provider-
azure/5eff51f5a04b5e91db5c18080c30316a5dee772a/docs/sample/ingress-
controller-tls/deployment-app-two.yaml -n <namespace name>

# create Ingress
kubectl apply -f https://raw.githubusercontent.com/Azure/secrets-
store-csi-driver-provider-
azure/5eff51f5a04b5e91db5c18080c30316a5dee772a/docs/sample/ingress-
controller-tls/ingress.yaml -n <namespace name>

# Check whether all related resources are up and running
kubectl get deploy,pod,svc,ing -n <namespace name>
```

11. Get the Ingress public IP address and try to curl the service

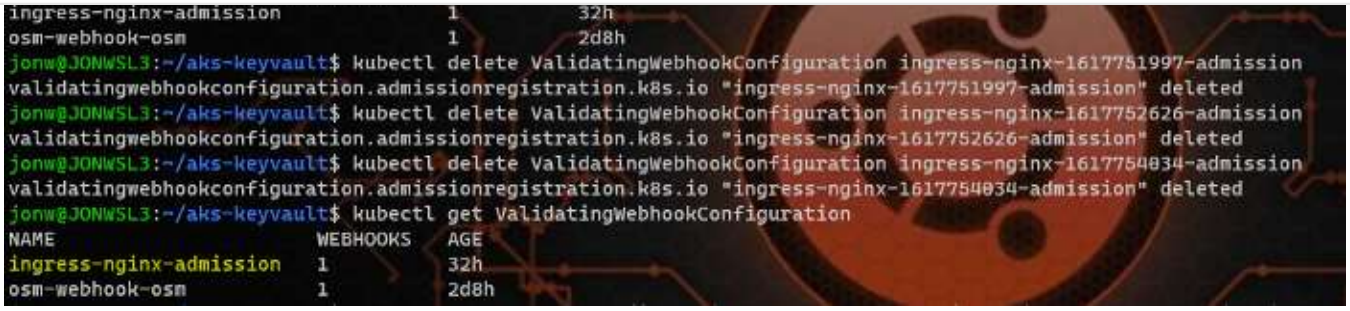
```
# Get Ingress public IP address
kubectl get ing -n <namespace name>

# curl the service
curl -v -k --resolve demo.test.com:443:<public IP address>
https://demo.test.com
```

```
Error from server (InternalError): error when creating
"https://raw.githubusercontent.com/Azure/secrets-store-csi-driver-
provider-
azure/5eff51f5a04b5e91db5c18080c30316a5dee772a/docs/sample/ingress-
controller-tls/ingress.yaml": Internal error occurred: failed
calling webhook "validate.nginx.ingress.kubernetes.io": Post
"https://ingress-nginx-1617752626-controller-admission.ingress-
test.svc:443/networking/v1beta1/ingresses?timeout=10s": service
"ingress-nginx-1617752626-controller-admission" not found
```

The workaround is to remove the unnecessary ValidatingWebhookConfiguration.

```
# Get Ingress service
kubectl get svc -n <namespace name>

# Get all ValidatingWebhookConfigurations
kubectl get ValidatingWebhookConfiguration -n <namespace name>

# Delete every ValidatingWebhookConfigurations that does not have
the same naming convention you are seeing in Ingress service
kubectl delete ValidatingWebhookConfiguration -n <namespace name>
```

I was having too many ValidatingWebhookConfiguration when first queried. After deleting, I have left only the one that has the same naming convention seeing in Ingress service.



Here are all the links I have taken for reference for composing this article.

- Secrets Store CSI and AKV Provider GitHub for NGINX Ingress Controller TLS Connection
- Secrets Store CSI and AKV Provider GitHub for Pod
- Azure CLI for AAD SP
- NGINX Installation on AKS or Azure-Bare-Metal K8s
- How to Replace String in Linux
- How to Resolve NGINX Error Message about Ingress Validation

Hope this would save the time of people trying to learn how to leverage secrets store CSI and AKV provider as native AKS resources. Happy learning!

Azure Key Vault Azure Kubernetes Service Nginx Ingress Secret Store Csi Demo