

**Lab Manual- Set up Secrets Store with TLS**

**Prepared for**:

**Date:** 18th Dec 2023

**Prepared by:**

Document Name: Lab Manual **Document Number** AZLabn916

**Contributor:**

Contents

[1. Objective 3](#_Toc162157383)

[2. Certificate Rotation for the Kubelet 4](#_Toc162157384)

[1. Understanding the certificate rotation configuration 4](#_Toc162157385)

[3. Secrets 5](#_Toc162157386)

[4. Types of Secret 5](#_Toc162157387)

[5. CERTIFICATE WITH K8 SECRET SAMPLE ARCHITECTURE 6](#_Toc162157388)

[6. Azure Well-Architected Framework review - Azure Application Gateway v2 6](#_Toc162157389)

[1. Reliability 6](#_Toc162157390)

[1.1. Design checklist 6](#_Toc162157391)

[2. Security 7](#_Toc162157392)

[1.2. Design checklist 7](#_Toc162157393)

[3. Cost optimization 7](#_Toc162157394)

[1. Design checklist 7](#_Toc162157395)

[4. Operational excellence 7](#_Toc162157396)

[1. Design checklist 8](#_Toc162157397)

[5. Performance efficiency 8](#_Toc162157398)

[1. Design checklist 8](#_Toc162157399)

[7. AKS Cluster with single node and Azure CNI 8](#_Toc162157400)

[8. Create TLS certificate for the Deployment 9](#_Toc162157401)

[9. Save certificate into a kubernetes secret object 10](#_Toc162157402)

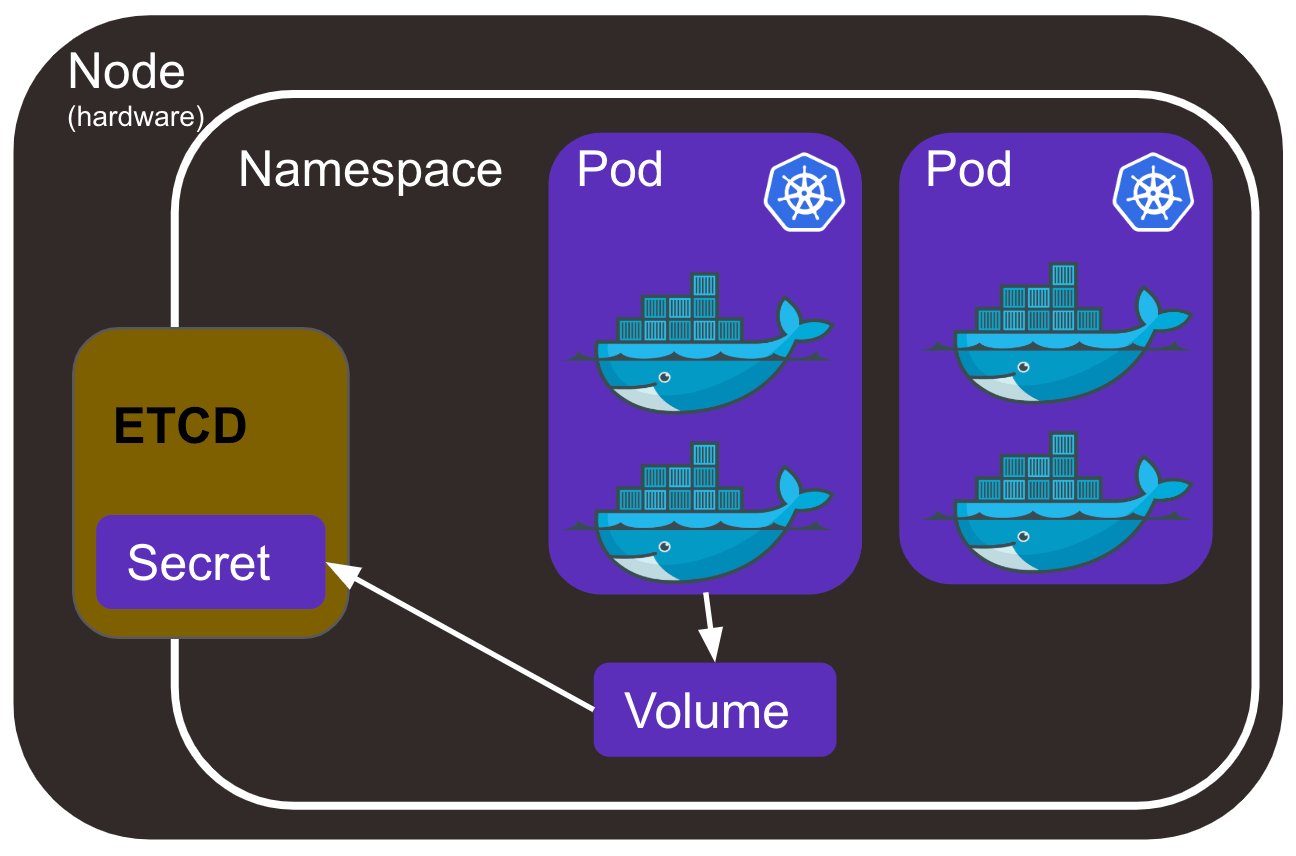
[10. Deploy sample application that uses the TLS secret to configure HTTPS 11](#_Toc162157403)

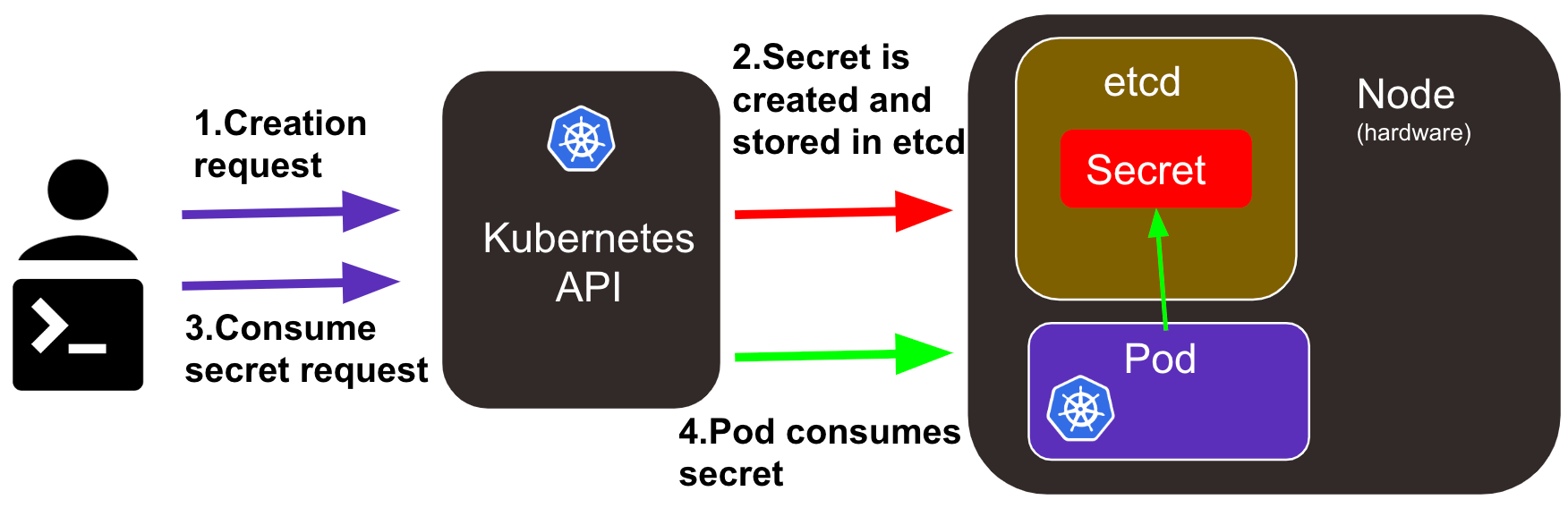
[11. Verify TLS certificate is working 13](#_Toc162157404)

# Objective

A Kubernetes secret is a resource for safely storing small chunks of data such as ssl certificate and keys, database passwords or ssh keys from and access it from a [Kubernetes pod](https://security.padok.fr/technologies/kubernetes). It is similar to a configMap, which does not aim to be protected, only smaller (max 1MB) and encoded (base64).

A secret is stored in the [**ETCD**](https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/). ETCD stands for "/etc directory", **the place where configuration files are stored on Linux systems**. It is a distributed key-value store used for storing data across a large number of nodes. A **secret belongs to a namespace**and can only **be accessed by pods within it.**





This lab will show how to configure a kubernetes deployment to use TLS certificate to enable HTTPS. We'll take as an example a .NET application.

Lab steps:

1. Create an AKS cluster
2. Create TLS certificate for the Deployment
3. Save certificate into a kubernetes secret object
4. Deploy sample application that uses the TLS secret to configure HTTPS
5. Verify TLS certificate is working

# Certificate Rotation for the Kubelet

Kubernetes contains [kubelet certificate rotation](https://kubernetes.io/docs/reference/access-authn-authz/kubelet-tls-bootstrapping/), that will automatically generate a new key and request a new certificate from the Kubernetes API as the current certificate approaches expiration. Once the new certificate is available, it will be used for authenticating connections to the Kubernetes API

## Understanding the certificate rotation configuration

When a kubelet starts up, if it is configured to bootstrap (using the --bootstrap-kubeconfig flag), it will use its initial certificate to connect to the Kubernetes API and issue a certificate signing request. You can view the status of certificate signing requests using:

kubectl get csr

Initially a certificate signing request from the kubelet on a node will have a status of Pending. If the certificate signing requests meets specific criteria, it will be auto approved by the controller manager, then it will have a status of **Approved**. Next, the controller manager will sign a certificate, issued for the duration specified by the --cluster-signing-duration parameter, and the signed certificate will be attached to the certificate signing request.

# Secrets

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a [Pod](https://kubernetes.io/docs/concepts/workloads/pods/) specification or in a [container image](https://kubernetes.io/docs/reference/glossary/?all=true#term-image). Using a Secret means that you don't need to include confidential data in your application code.

Because Secrets can be created independently of the Pods that use them, there is less risk of the Secret (and its data) being exposed during the workflow of creating, viewing, and editing Pods. Kubernetes, and applications that run in your cluster, can also take additional precautions with Secrets, such as avoiding writing sensitive data to nonvolatile storage.

Secrets are similar to [ConfigMaps](https://kubernetes.io/docs/concepts/configuration/configmap/" \o "" \t "_blank) but are specifically intended to hold confidential data.

**Caution:**

Kubernetes Secrets are, by default, stored unencrypted in the API server's underlying data store (etcd). Anyone with API access can retrieve or modify a Secret, and so can anyone with access to etcd. Additionally, anyone who is authorized to create a Pod in a namespace can use that access to read any Secret in that namespace; this includes indirect access such as the ability to create a Deployment.

In order to safely use Secrets, take at least the following steps:

1. [Enable Encryption at Rest](https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/) for Secrets.
2. [Enable or configure RBAC rules](https://kubernetes.io/docs/reference/access-authn-authz/authorization/) with least-privilege access to Secrets.
3. Restrict Secret access to specific containers.
4. [Consider using external Secret store providers](https://secrets-store-csi-driver.sigs.k8s.io/concepts.html#provider-for-the-secrets-store-csi-driver).

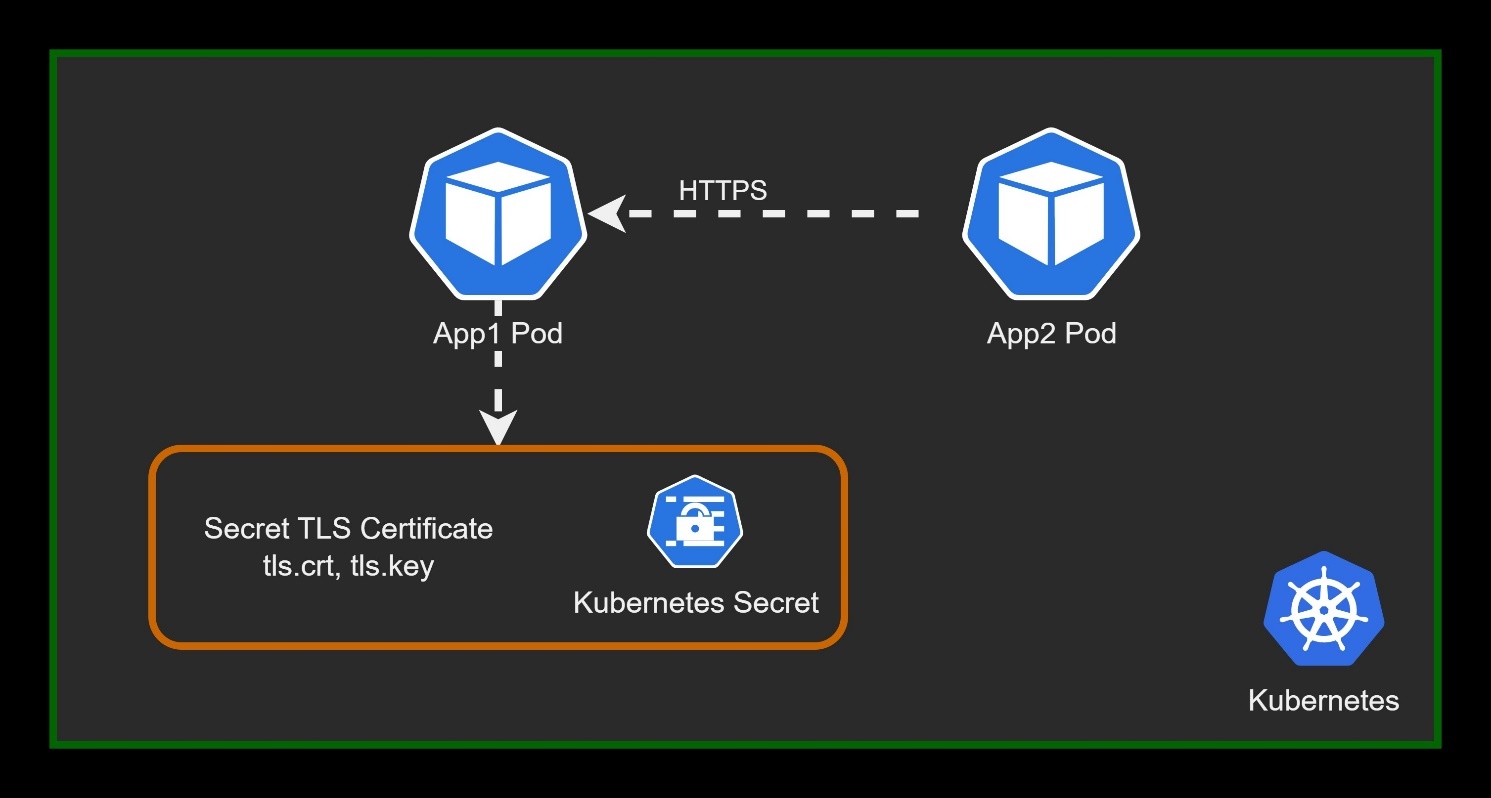
# Types of Secret

When creating a Secret, you can specify its type using the type field of the [Secret](https://kubernetes.io/docs/reference/kubernetes-api/config-and-storage-resources/secret-v1/) resource, or certain equivalent kubectl command line flags (if available). The Secret type is used to facilitate programmatic handling of the Secret data.

Kubernetes provides several built-in types for some common usage scenarios. These types vary in terms of the validations performed and the constraints Kubernetes imposes on them.

| **Built-in Type** | **Usage** |
| --- | --- |
| Opaque | arbitrary user-defined data |
| kubernetes.io/service-account-token | ServiceAccount token |
| kubernetes.io/dockercfg | serialized ~/.dockercfg file |
| kubernetes.io/dockerconfigjson | serialized ~/.docker/config.json file |
| kubernetes.io/basic-auth | credentials for basic authentication |
| kubernetes.io/ssh-auth | credentials for SSH authentication |
| kubernetes.io/tls | data for a TLS client or server |
| bootstrap.kubernetes.io/token | bootstrap token data |

# CERTIFICATE WITH K8 SECRET SAMPLE ARCHITECTURE



# Azure Well-Architected Framework review - Azure Application Gateway v2

## Reliability

In the cloud, we acknowledge that failures happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component. Use the following information to minimize failed instances.

### Design checklist

As you make design choices for Application Gateway, review the [Reliability design principles](https://learn.microsoft.com/en-gb/azure/well-architected/resiliency/principles).

* Deploy the instances in a [zone-aware configuration](https://learn.microsoft.com/en-us/azure/application-gateway/application-gateway-autoscaling-zone-redundant), where available.
* Use Application Gateway with Web Application Firewall (WAF) within a virtual network to protect inbound HTTP/S traffic from the Internet.
* In new deployments, use Azure Application Gateway v2 unless there is a compelling reason to use Azure Application Gateway v1.
* Plan for rule updates
* Use health probes to detect backend unavailability
* Review the impact of the interval and threshold settings on health probes
* Verify downstream dependencies through health endpoints

## Security

Security is one of the most important aspects of any architecture. Application Gateway provides features to employ both the principle of least privilege and defense-in-defense. We recommend you review the [Security design principles](https://learn.microsoft.com/en-gb/azure/well-architected/security/principles).

### Design checklist

* Set up a TLS policy for enhanced security
* Use AppGateway for TLS termination
* Use Azure Key Vault to store TLS certificates
* When re-encrypting backend traffic, ensure the backend server certificate contains both the root and intermediate Certificate Authorities (CAs)
* Use an appropriate DNS server for backend pool resources
* Comply with all NSG restrictions for Application Gateway
* Refrain from using UDRs on the Application Gateway subnet
* Be aware of Application Gateway capacity changes when enabling WAF

## Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. We recommend you review the [Cost optimization design principles](https://learn.microsoft.com/en-gb/azure/well-architected/cost-optimization/principles).

### Design checklist

* Familiarize yourself with Application Gateway pricing
* Review underutilized resources
* Stop Application Gateway instances that are not in use
* Have a scale-in and scale-out policy
* Review consumption metrics across different parameters

## Operational excellence

Monitoring and diagnostics are crucial for ensuring operational excellence of your Application Gateway and the web applications or backends behind the gateway. You can not only measure performance statistics but also use metrics to troubleshoot and remediate issues quickly. We recommend you review the [Operational Excellence design principles](https://learn.microsoft.com/en-gb/azure/well-architected/operational-excellence/principles).

### Design checklist

* Monitor capacity metrics
* Enable diagnostics on Application Gateway and Web Application Firewall (WAF)
* Use Azure Monitor Network Insights
* Match timeout settings with the backend application
* Monitor Key Vault configuration issues using Azure Advisor
* Configure and monitor SNAT port limitations
* Consider SNAT port limitations in your design

## Performance efficiency

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. We recommend you review the [Performance efficiency principles](https://learn.microsoft.com/en-gb/azure/well-architected/scalability/principles).

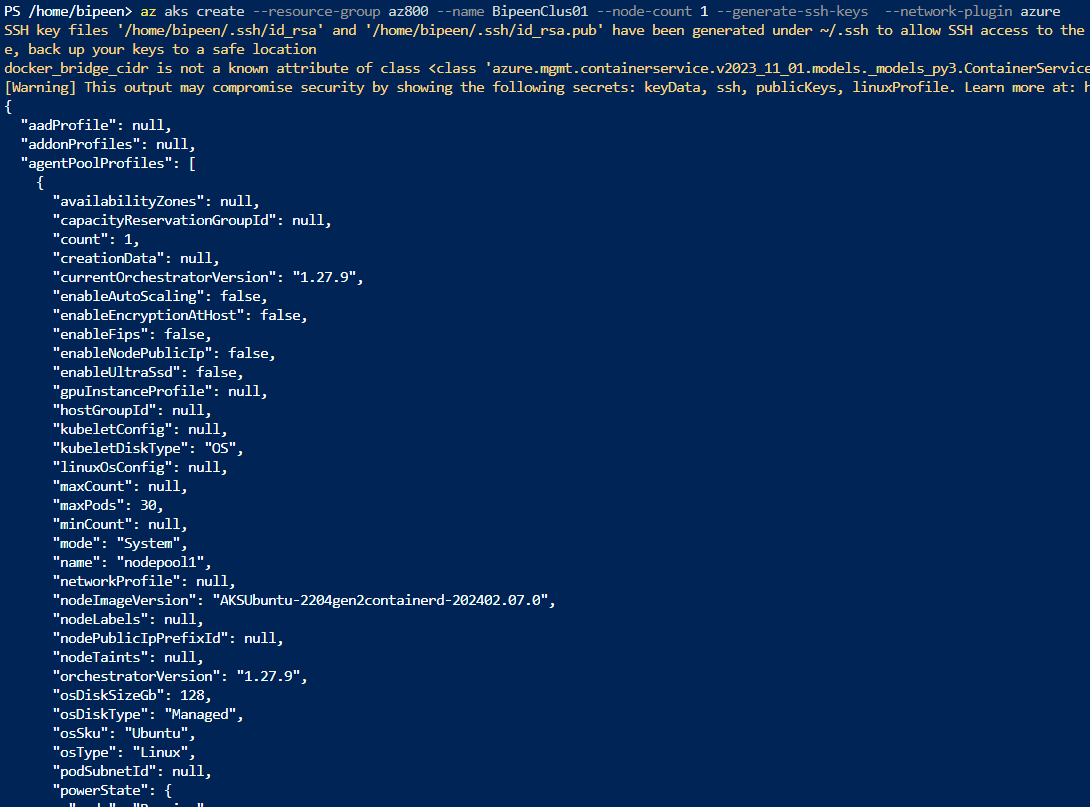
### Design checklist

* Estimate the Application Gateway instance count
* Define the maximum instance count
* Define the minimum instance count
* Define Application Gateway subnet size
* Take advantage of Application Gateway V2 features for autoscaling and performance benefits

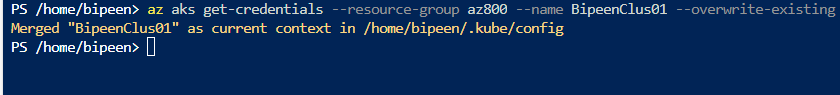
<https://learn.microsoft.com/en-gb/azure/well-architected/service-guides/azure-application-gateway>

# AKS Cluster with single node and Azure CNI

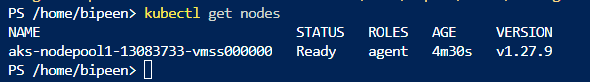
az aks create --resource-group az800 --name BipeenClus01 --node-count 1 --generate-ssh-keys --network-plugin azure



az aks get-credentials --resource-group az800 --name BipeenClus01 --overwrite-existing



kubectl get nodes



# Create TLS certificate for the Deployment

$APP\_CERT\_NAME="app-tls-cert"

$SERVICE\_NAME="app-svc"

openssl req -x509 -nodes -days 365 -newkey rsa:2048 `

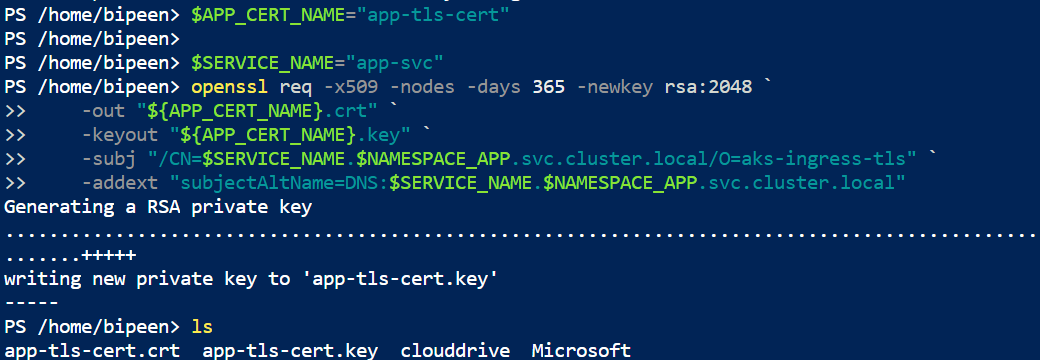
    -out "${APP\_CERT\_NAME}.crt" `

    -keyout "${APP\_CERT\_NAME}.key" `

    -subj "/CN=$SERVICE\_NAME.$NAMESPACE\_APP.svc.cluster.local/O=aks-ingress-tls" `

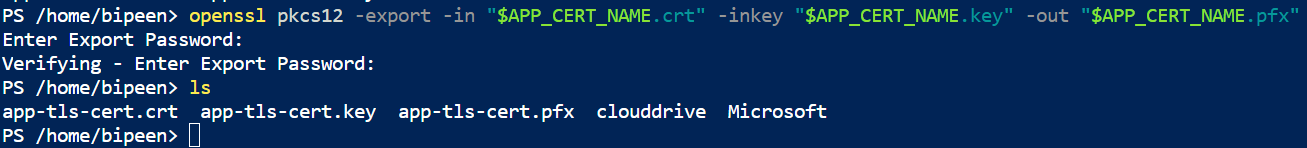
    -addext "subjectAltName=DNS:$SERVICE\_NAME.$NAMESPACE\_APP.svc.cluster.local"

openssl pkcs12 -export -in "$APP\_CERT\_NAME.crt" -inkey "$APP\_CERT\_NAME.key" -out "$APP\_CERT\_NAME.pfx"



openssl pkcs12 -export -in "$APP\_CERT\_NAME.crt" -inkey "$APP\_CERT\_NAME.key" -out "$APP\_CERT\_NAME.pfx"

When Ask for Password just press enter for no password



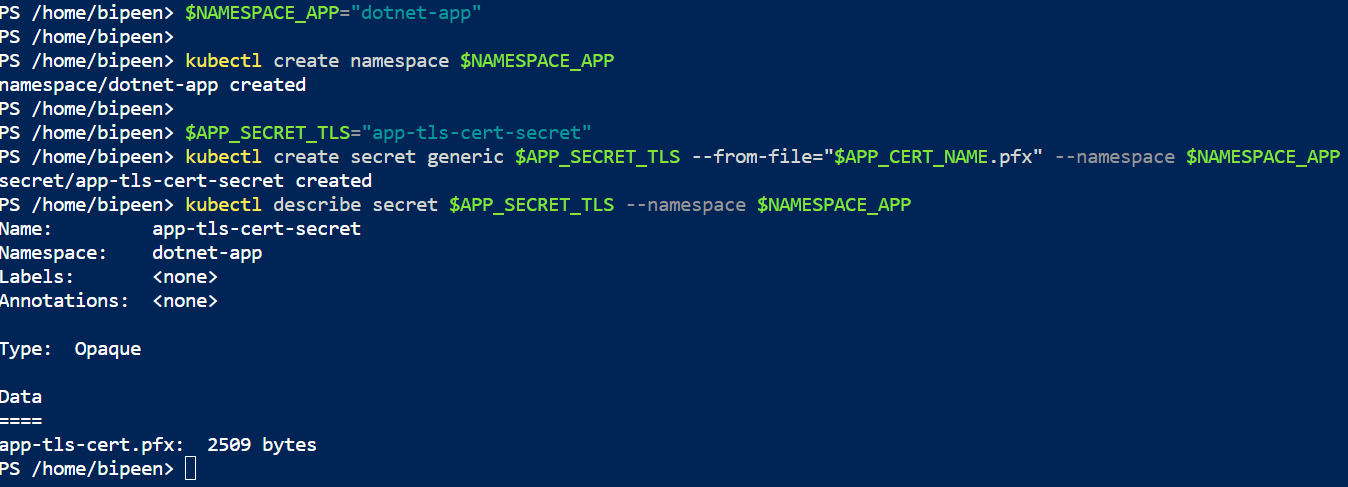
# Save certificate into a kubernetes secret object

$NAMESPACE\_APP="dotnet-app"

kubectl create namespace $NAMESPACE\_APP

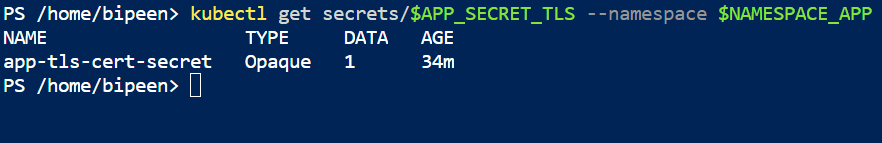
$APP\_SECRET\_TLS="app-tls-cert-secret"

kubectl create secret generic $APP\_SECRET\_TLS --from-file="$APP\_CERT\_NAME.pfx" --namespace $NAMESPACE\_APP

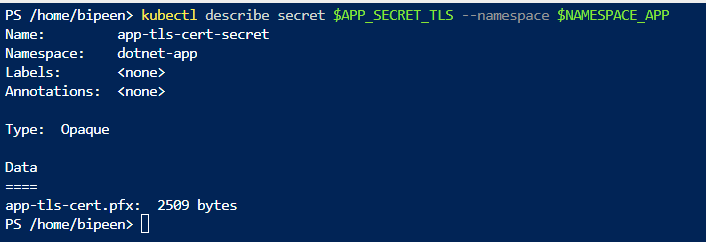


**Check the secret.**

kubectl get secrets/$APP\_SECRET\_TLS --namespace $NAMESPACE\_APP

****

kubectl describe secret $APP\_SECRET\_TLS --namespace $NAMESPACE\_APP



# Deploy sample application that uses the TLS secret to configure HTTPS

**app-deploy-svc.yaml**

IT also mount the Mount SSL certificates in the Pod with Kubernetes secret

apiVersion: v1

kind: Service

metadata:

  labels:

    app: demo-app

  name: app-svc

spec:

  ports:

  - port: 443

    protocol: TCP

    targetPort: 443

  selector:

    app: demo-app

  type: ClusterIP

---

apiVersion: apps/v1

kind: Deployment

metadata:

  labels:

    app: demo-app

  name: demo-app

spec:

  replicas: 3

  selector:

    matchLabels:

      app: demo-app

  template:

    metadata:

      labels:

        app: demo-app

    spec:

      restartPolicy: Always

      volumes:

      - name: demo-app-tls

        secret:

          secretName: app-tls-cert-secret

      containers:

      - name: demo-app

        image: mcr.microsoft.com/dotnet/samples:aspnetapp

        ports:

        - containerPort: 443

        volumeMounts:

        - name: demo-app-tls

          mountPath: /secrets/tls-cert

          readOnly: true

        env:

        - name: ASPNETCORE\_Kestrel\_\_Certificates\_\_Default\_\_Password

          value: ""

        - name: ASPNETCORE\_Kestrel\_\_Certificates\_\_Default\_\_Path

          value: /secrets/tls-cert/app-tls-cert.pfx

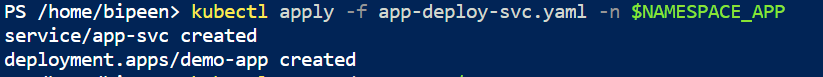
        - name: ASPNETCORE\_URLS

          value: "https://+;http://+"

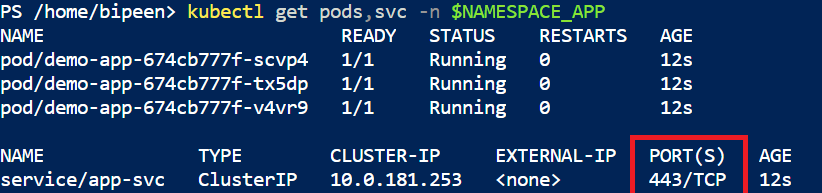
        - name: ASPNETCORE\_HTTPS\_PORT

          value: "443"

kubectl apply -f app-deploy-svc.yaml -n $NAMESPACE\_APP



kubectl get pods,svc -n $NAMESPACE\_APP

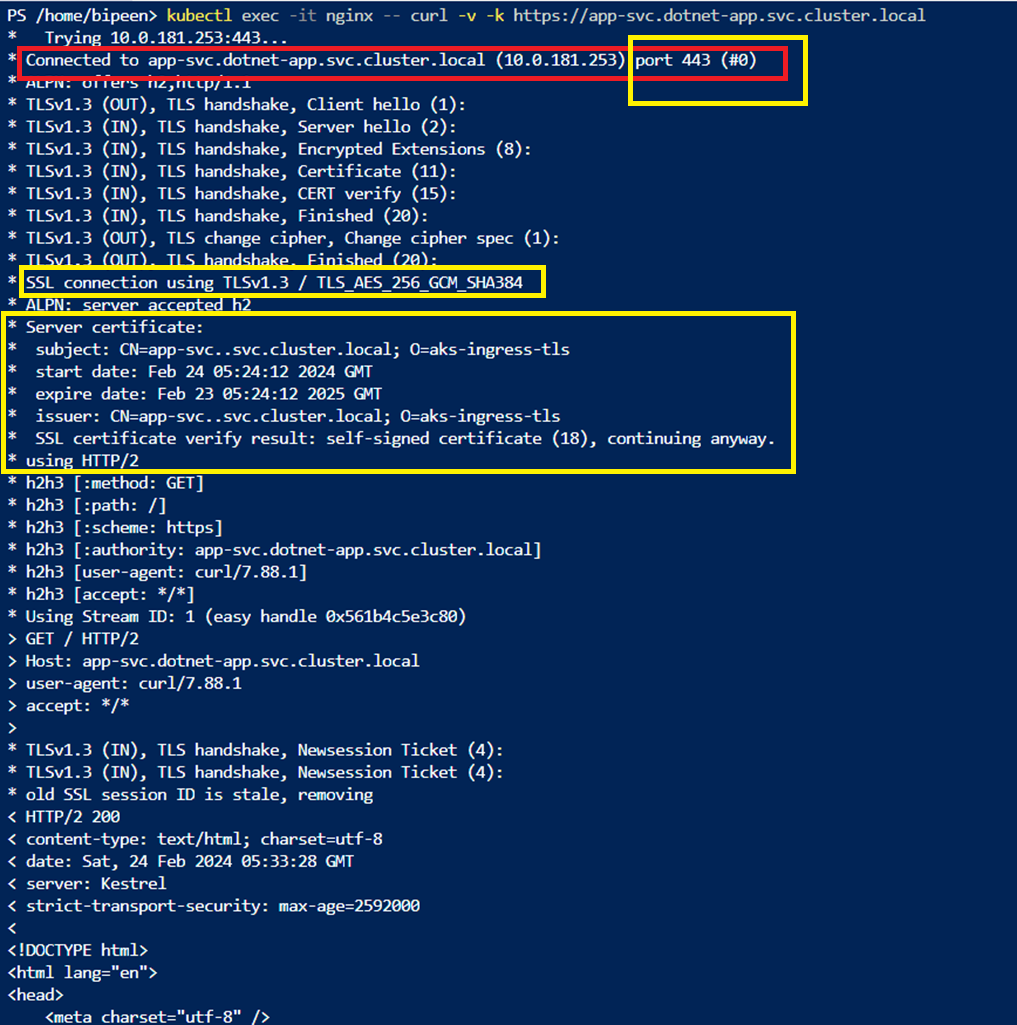


# Verify TLS certificate is working

kubectl run nginx --image=nginx



kubectl exec -it nginx -- curl -v -k https://app-svc.dotnet-app.svc.cluster.local



For Additional to use Certificate with secret and Key vault

<https://learn.microsoft.com/en-us/azure/aks/csi-secrets-store-nginx-tls>