

# TLS Proxy: Custom domain support for Key Vault using Application Gateway

## Contents

Overview .....	1
Pre-requisites .....	1
Solution – Detailed Process .....	2
Configuration of Key Vault .....	2
Configuration of Application Gateway .....	3
Configure Key Vault for authentication .....	4
Verifying using an SDK Client .....	6
Callouts/Caveats .....	8
ETA .....	9

## Overview

KeyVault supports KeyVault URLs like <https://tlsproxy-poc-keyvault.vault.azure.net>. It does not support custom domain URLs like <https://tlsproxy-poc-keyvault.cloudapp.net> (or <https://https://tlsproxy-poc-keyvault.contoso.com>). This feature can be supported by adding Application Gateway as a frontend for KeyVault which maps custom domain URL to Application Gateway URL.

In this document, all the components are configured using Azure portal. A secret is retrieved using a SDK client from a KeyVault. Client sends HTTP request using custom domain URL and KeyVault supports only KeyVault URL. AppGw sits between them and routes the traffic coming from client to the KeyVault endpoint.

## Pre-requisites

1. Azure Key Vault with a Secret
2. Application Gateway using L7 capabilities (HTTP)
3. Client SDK ([Documentation](#))

## Solution – Detailed Process

### Configuration of Key Vault

#### 1. Create a KeyVault with its private endpoint added to Application Gateway Virtual Network.

While selecting the network, use private endpoint option and create a new private endpoint. Choose the same resource group and vnet as application gateway and subnet should part of the same Vnet as the Application Gateway.

The screenshot shows the 'Create private endpoint' dialog in the Microsoft Azure portal. The left pane shows the 'Create key vault' page with the 'Networking' tab selected. The right pane shows the configuration for the private endpoint.

**Create private endpoint**

Subscription: Applw IUL Internal Subscription 1  
Resource group: tls-rg  
Location: East US  
Name: tls-private-endpoint  
Target sub-resource: Vault

**Networking**

To deploy the private endpoint, select a virtual network subnet. [Learn more about private endpoint networking](#)

Virtual network: us-vnet  
Subnet: us-vnet/subnet-1 (10.1.0.0/24)

**Private DNS integration**

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more about private DNS integration](#)

Integrate with private DNS zone: Yes (selected) No  
Private DNS Zone: privatelink.vaultcore.azure.net

Buttons: Review + create, < Previous, Next: Tags >, OK, Discard

#### 2. Add a Secret in Key Vault



The screenshot shows the 'Create a secret' page in the Microsoft Azure portal. The left pane shows the 'Create a secret' page with the 'Manual' upload option selected. The right pane shows the configuration for the secret.

**Create a secret**



Upload options: Manual  
Name: tlsproxy-poc-keyvault-secret  
Value: [Redacted]  
Content type (optional):  
Set activation date: ☐  
Set expiration date: ☐  
Enabled: Yes (selected) No

After secret addition, it will look like below.

Home > tlsproxy-poc-keyvault > tlsproxy-poc-keyvault > tlsproxy-poc-keyvault-secret >

 **31d5cbb0e3034e54bda8bf5badf62cba**  ...

Secret Version

 Save  Discard


---

Properties

Created 7/29/2021, 5:08:35 PM

Updated 7/29/2021, 5:08:35 PM

Secret Identifier

`https://tlsproxy-poc-keyvault.vault.azure.net/secrets/tlsproxy-poc-keyvault-secret/31d5cbb0e3034e54bda8bf5badf62cba` 

Settings

Set activation date ⓘ

☐

Set expiration date ⓘ

☐

Enabled

☒ Yes ☐ No

---

Tags

0 tags >


---

Secret

Content type (optional)

[Hide Secret Value](#)

Secret value

This is the secret value. 

## Configuration of Application Gateway

1. In backend pool, add target as default KeyVault URI.

Home > tlsproxy-poc-rg > Create a resource > Application Gateway >

### Create application gateway

✓ Basics ✓ Frontends **3 Backends** 4 Configuration 5 Tags 6 Review + create

A backend pool is a collection of resources to which your application gateway can send traffic. A backend pool can contain virtual machines, virtual machine scale sets, app services, IP addresses, or fully qualified domain names (FQDN).

[Add a backend pool](#)

Backend pool	Targets
No results	

#### Add a backend pool.


A backend pool is a collection of resources to which your application gateway can send traffic. A backend pool can contain virtual machines, virtual machines scale sets, IP addresses, domain names, or an App Service.

Name \*  ✓

Add backend pool without targets ☐ Yes ☒ No

Backend targets

1 item

Target type	Target
<input type="text" value="IP address or FQDN"/>	<input type="text" value="tlsproxy-poc-keyvault.vault.azure.net"/> ✓  ...
<input type="text" value="IP address or FQDN"/>	<input type="text"/>

2. For HTTP Setting, choose “well-known CA certificate” and override the hostname by picking it from the backend target. Custom probe can be added later.

Home > tlsproxy-poc-rg > Create a resource > Application Gateway >

## Create application gateway

✓ Basics ✓ Frontends ✓ Backends **Configuration** Tags Review + create

Create routing rules that link your frontends(s) and backends(s). You can also add more backend pools, add a second frontend IP configuration if you haven't already, or edit previous configurations.

**Frontends**  
 + Add a frontend IP  
 Public (new) tlsproxy-poc-appgw-ip

**Routing rules**  
 + Add a routing rule

### Add a HTTP setting

← Discard changes and go back to routing rules

HTTP settings name \*

Backend protocol ☐ HTTP ☒ HTTPS

Backend port \*

**Trusted root certificate**  
For end-to-end SSL encryption, the backends must be in the allowlist of the application gateway. Upload the public certificate of the backend servers to this HTTP setting.

Use well known CA certificate ☒ Yes ☐ No

**Additional settings**

Cookie-based affinity ☐ Enable ☒ Disable

Connection draining ☐ Enable ☒ Disable

Request time-out (seconds) \*

Override backend path

**Host name**  
By default, Application Gateway does not change the incoming HTTP host header from the client and sends the header unaltered to the backend. Multi-tenant services like App service or API management rely on a specific host header or SNI extension to resolve to the correct endpoint. Change these settings to overwrite the incoming HTTP host header.

Override with new host name ☒ Yes ☐ No

Host name override ☐ Pick host name from backend target ☒ Override with specific domain name

Create custom probes ☐ Yes ☒ No

### 3. Add a Custom Health Probe with HTTPS protocol and Path as “/healthstatus”.

Microsoft Azure (Preview) Search resources, services, and docs (G+)

Home > tlsproxy-poc-keyvault > tlsproxy-poc-keyvault > appgw | Health probes

appgw Application gateway

Search (Ctrl+/)

Overview  
Activity log  
Access control (IAM)  
Tags  
Diagnose and solve problems

**Settings**

Configuration  
Web application firewall  
Backend pools  
HTTP settings  
Frontend IP configurations  
SSL settings (Preview)  
Listeners  
Rules  
Rewrites  
**Health probes**  
Properties  
Locks  
Monitoring

### tls-health-probe

Name

Protocol ☐ HTTP ☒ HTTPS

Host \*

Pick host name from backend HTTP settings ☐ Yes ☒ No

Pick port from backend HTTP settings ☒ Yes ☐ No

Path \*

Interval (seconds) \*

Timeout (seconds) \*

Unhealthy threshold \*

Use probe matching conditions ☐ Yes ☒ No

HTTP settings

☒ I want to test the backend health before adding the health probe

**Test** **Discard**

## Configure Key Vault for authentication

### 1. Using PowerShell, create a service principal using the following command.

```
az ad sp create-for-rbac -n <your-application-name> --skip-assignment
```

```
PS C:\WINDOWS\system32> az ad sp create-for-rbac -n tlsproxy-poc-app --skip-assignment
Changing "tlsproxy-poc-app" to a valid URI of "http://tlsproxy-poc-app", which is the required format used for service principal names
{
  "appId": "3f4c03b0-6163-4415-ba69-7a4ebe76d509",
  "displayName": "tlsproxy-poc-app",
  "name": "http://tlsproxy-poc-app",
  "password": "n7mtb8Q3oL1GBVWuKKVxeVZ-JTIJ0TAs-m",
  "tenant": "72f988bf-86f1-41af-91ab-2d7cd011db47"
}
PS C:\WINDOWS\system32>
```

2. In the Key Vault, add an access policy with Get Secret permissions and Service Principal created above.

Home > tlsproxy-poc-keyvault > Add access policy

Add access policy

Configure from template (optional)

Key permissions

Secret permissions

Certificate permissions

Select principal \*

Authorized application

**Principal**  
Select a principal

tlsproxy-poc-app  
3f4c03b0-6163-4415-ba69-7a4ebe76d509  
Selected

**Selected items**

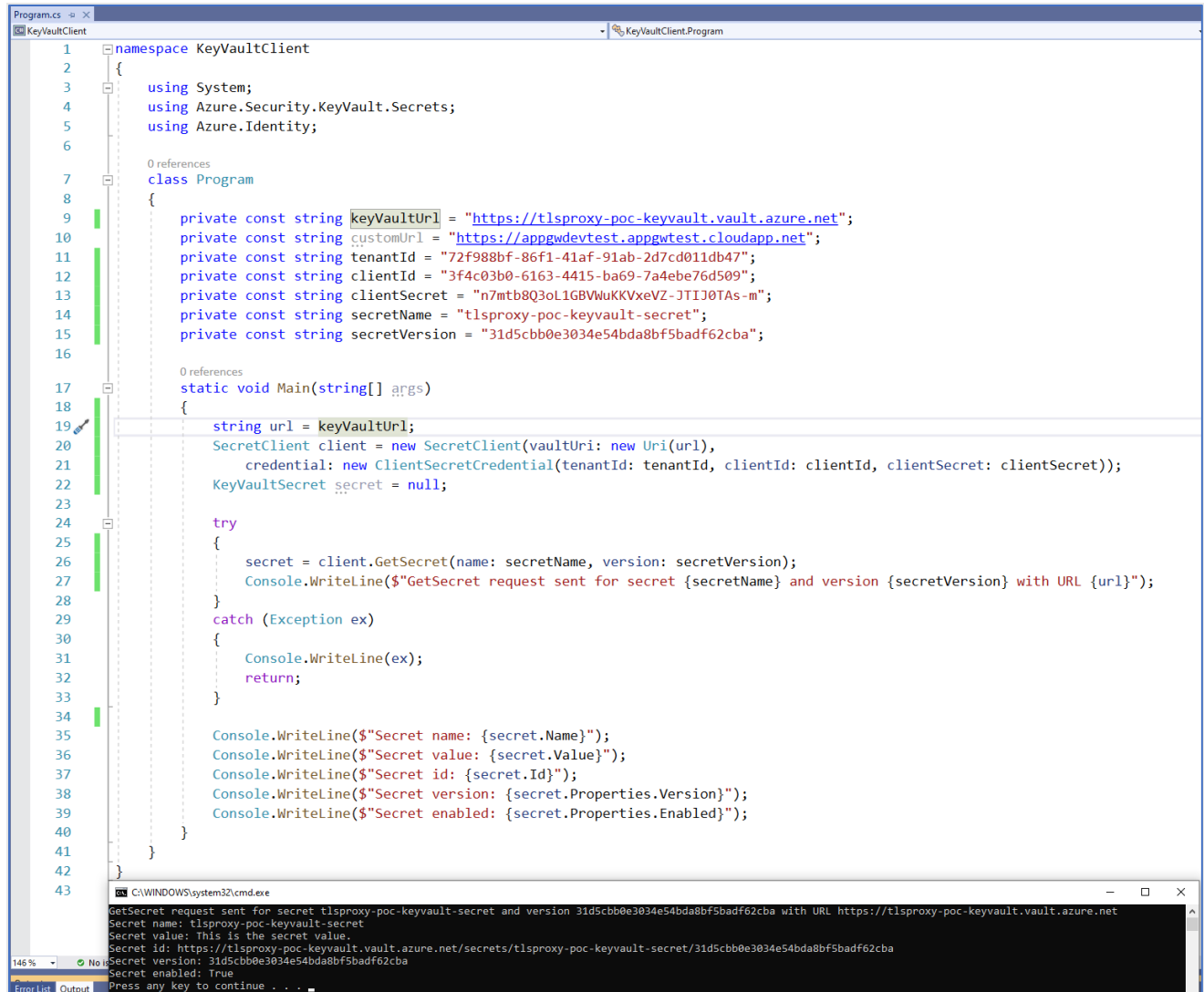
tlsproxy-poc-app  
3f4c03b0-6163-4415-ba69-7a4ebe76d509

**Before we proceed, the custom domain's DNS should be setup to resolve to Application Gateway IP.**

Verifying using an SDK Client

Create a client (C# Program) using Key Vault SDK. We will send a request through both default and custom URLs to check the outputs.

**Default URL** - GetSecret request is sent using the default Key Vault URL <https://tlsproxy-poc-keyvault.vault.azure.net>



```
1 namespace KeyVaultClient
2 {
3     using System;
4     using Azure.Security.KeyVault.Secrets;
5     using Azure.Identity;
6
7     0 references
8     class Program
9     {
10         private const string keyVaultUrl = "https://tlsproxy-poc-keyvault.vault.azure.net";
11         private const string customUrl = "https://appgwdevtest.appgwtest.cloudapp.net";
12         private const string tenantId = "72f988bf-86f1-41af-91ab-2d7cd011db47";
13         private const string clientId = "3f4c03b0-6163-4415-ba69-7a4ebe76d509";
14         private const string clientSecret = "n7mtb8Q3oL1GBVWuKKVxeVZ-3TlJ0TAs-m";
15         private const string secretName = "tlsproxy-poc-keyvault-secret";
16         private const string secretVersion = "31d5cbb0e3034e54bda8bf5badf62cba";
17
18         0 references
19         static void Main(string[] args)
20         {
21             string url = keyVaultUrl;
22             SecretClient client = new SecretClient(vaultUri: new Uri(url),
23                 credential: new ClientSecretCredential(tenantId: tenantId, clientId: clientId, clientSecret: clientSecret));
24             KeyVaultSecret secret = null;
25
26             try
27             {
28                 secret = client.GetSecret(name: secretName, version: secretVersion);
29                 Console.WriteLine($"GetSecret request sent for secret {secretName} and version {secretVersion} with URL {url}");
30             }
31             catch (Exception ex)
32             {
33                 Console.WriteLine(ex);
34                 return;
35             }
36
37             Console.WriteLine($"Secret name: {secret.Name}");
38             Console.WriteLine($"Secret value: {secret.Value}");
39             Console.WriteLine($"Secret id: {secret.Id}");
40             Console.WriteLine($"Secret version: {secret.Properties.Version}");
41             Console.WriteLine($"Secret enabled: {secret.Properties.Enabled}");
42         }
43     }
44 }
```

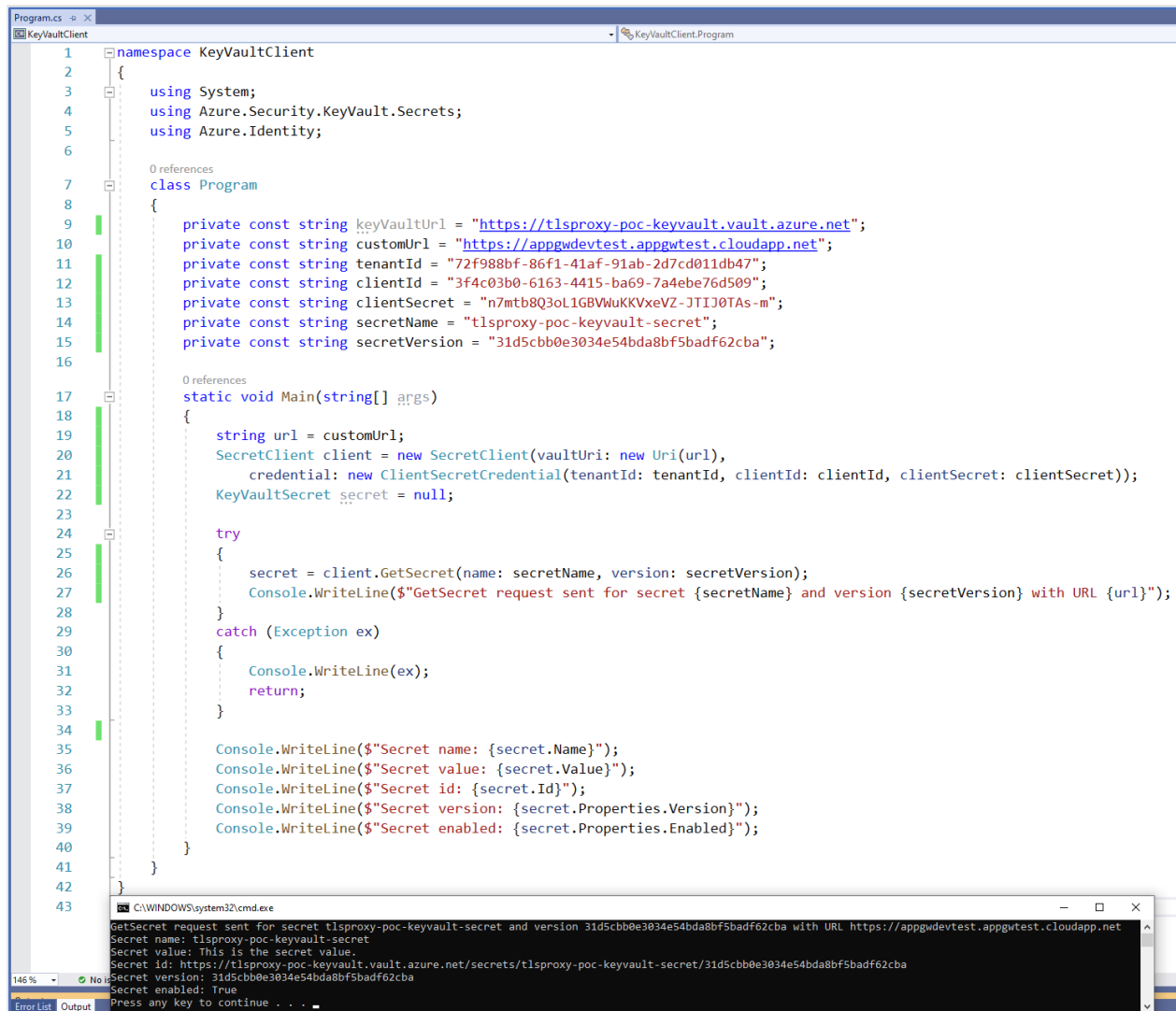
C:\WINDOWS\system32\cmd.exe

```
GetSecret request sent for secret tlsproxy-poc-keyvault-secret and version 31d5cbb0e3034e54bda8bf5badf62cba with URL https://tlsproxy-poc-keyvault.vault.azure.net
Secret name: tlsproxy-poc-keyvault-secret
Secret value: This is the secret value.
Secret id: https://tlsproxy-poc-keyvault.vault.azure.net/secrets/tlsproxy-poc-keyvault-secret/31d5cbb0e3034e54bda8bf5badf62cba
Secret version: 31d5cbb0e3034e54bda8bf5badf62cba
Secret enabled: True
Press any key to continue . . .
```

Output

```
GetSecret request sent for secret tlsproxy-poc-keyvault-secret and version 31d5cbb0e3034e54bda8bf5badf62cba with URL https://tlsproxy-poc-keyvault.vault.azure.net
Secret name: tlsproxy-poc-keyvault-secret
Secret value: This is the secret value.
Secret id: https://tlsproxy-poc-keyvault.vault.azure.net/secrets/tlsproxy-poc-keyvault-secret/31d5cbb0e3034e54bda8bf5badf62cba
Secret version: 31d5cbb0e3034e54bda8bf5badf62cba
Secret enabled: True
```

Custom URL - GetSecret request is sent using the Custom URL <https://appgwdevtest.appgwtest.cloudapp.net>



```
1 namespace KeyVaultClient
2 {
3     using System;
4     using Azure.Security.KeyVault.Secrets;
5     using Azure.Identity;
6
7     0 references
8     class Program
9     {
10         private const string keyVaultUrl = "https://tlsproxy-poc-keyvault.vault.azure.net";
11         private const string customUrl = "https://appgwdevtest.appgwtest.cloudapp.net";
12         private const string tenantId = "72f988bf-86f1-41af-91ab-2d7cd011db47";
13         private const string clientId = "3f4c03b0-6163-4415-ba69-7a4ebe76d509";
14         private const string clientSecret = "n7mtb8Q3oL1GBVWuKKVxeVZ-JTIJ0TAs-m";
15         private const string secretName = "tlsproxy-poc-keyvault-secret";
16         private const string secretVersion = "31d5cbb0e3034e54bda8bf5badf62cba";
17
18         0 references
19         static void Main(string[] args)
20         {
21             string url = customUrl;
22             SecretClient client = new SecretClient(vaultUri: new Uri(url),
23                 credential: new ClientSecretCredential(tenantId: tenantId, clientId: clientId, clientSecret: clientSecret));
24             KeyVaultSecret secret = null;
25
26             try
27             {
28                 secret = client.GetSecret(name: secretName, version: secretVersion);
29                 Console.WriteLine($"GetSecret request sent for secret {secretName} and version {secretVersion} with URL {url}");
30             }
31             catch (Exception ex)
32             {
33                 Console.WriteLine(ex);
34                 return;
35             }
36
37             Console.WriteLine($"Secret name: {secret.Name}");
38             Console.WriteLine($"Secret value: {secret.Value}");
39             Console.WriteLine($"Secret id: {secret.Id}");
40             Console.WriteLine($"Secret version: {secret.Properties.Version}");
41             Console.WriteLine($"Secret enabled: {secret.Properties.Enabled}");
42         }
43     }
```

C:\WINDOWS\system32\cmd.exe

```
GetSecret request sent for secret tlsproxy-poc-keyvault-secret and version 31d5cbb0e3034e54bda8bf5badf62cba with URL https://appgwdevtest.appgwtest.cloudapp.net
Secret name: tlsproxy-poc-keyvault-secret
Secret value: This is the secret value.
Secret id: https://tlsproxy-poc-keyvault.vault.azure.net/secrets/tlsproxy-poc-keyvault-secret/31d5cbb0e3034e54bda8bf5badf62cba
Secret version: 31d5cbb0e3034e54bda8bf5badf62cba
Secret enabled: True
Press any key to continue . . .
```

## Output

```
GetSecret request sent for secret tlsproxy-poc-keyvault-secret and version 31d5cbb0e3034e54bda8bf5badf62cba with URL
https://appgwdevtest.appgwtest.cloudapp.net
Secret name: tlsproxy-poc-keyvault-secret
Secret value: This is the secret value.
Secret id: https://tlsproxy-poc-keyvault.vault.azure.net/secrets/tlsproxy-poc-keyvault-secret/31d5cbb0e3034e54bda8bf5badf62cba
Secret version: 31d5cbb0e3034e54bda8bf5badf62cba
Secret enabled: True
```

## Client Code

```
namespace KeyVaultClient
{
    using System;
    using Azure.Security.KeyVault.Secrets;
    using Azure.Identity;

    class Program
    {
        private const string keyVaultUrl = "https://tlsproxy-poc-keyvault.vault.azure.net";
        private const string customUrl = "https://appgwdevtest.appgwtest.cloudapp.net";
        private const string tenantId = "72f988bf-86f1-41af-91ab-2d7cd011db47";
        private const string clientId = "3f4c03b0-6163-4415-ba69-7a4ebe76d509";
        private const string clientSecret = "n7mtb8Q3oL1GBVWuKKVxeVZ-JTIJ0TAs-m";
        private const string secretName = "tlsproxy-poc-keyvault-secret";
        private const string secretVersion = "31d5cbb0e3034e54bda8bf5badf62cba";

        static void Main(string[] args)
        {
            string url = customUrl;
            SecretClient client = new SecretClient(vaultUri: new Uri(keyVaultUrl), credential: new
            ClientSecretCredential(tenantId: tenantId, clientId: clientId, clientSecret: clientSecret));
            KeyVaultSecret secret = null;

            try
            {
                secret = client.GetSecret(name: secretName, version: secretVersion);
                Console.WriteLine($"GetSecret request sent for secret {secretName} and version
{secretVersion} with URL {url}");
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
                return;
            }

            Console.WriteLine($"Secret name: {secret.Name}");
            Console.WriteLine($"Secret value: {secret.Value}");
            Console.WriteLine($"Secret id: {secret.Id}");
            Console.WriteLine($"Secret version: {secret.Properties.Version}");
            Console.WriteLine($"Secret enabled: {secret.Properties.Enabled}");
        }
    }
}
```

## Callouts/Caveats

1. Some API's return the ID/URI of resource having the actual/default hostname of keyvault even while calling the api's from custom domain. The response in these api's would need to be modified to replace the actual hostname with custom domain. Example: In API Get Keys there is nextLink field which points to the next url for getting the paginated response for keys. The hostname in next url would be replaced to custom domain, and everything should work fine as is.



## Request

```
GET https://myvault.vault.azure.net//keys?maxresults=1&api-version=7.2
```

## Response

```
{
  "value": [
    {
      "kid": "https://myvault.vault.azure.net/keys/sdktestkey",
      "attributes": {
        "enabled": true,
        "created": 1493937656,
        "updated": 1493937656,
        "recoveryLevel": "Recoverable+Purgeable"
      }
    }
  ],
  "nextLink": "https://myvault.vault.azure.net:443/keys?api-version=7.2&$skiptoken=eyJJOZXh0TWYya2VyljoiMiE5NiFNREF3TURJMOIXdGxIUzlwVUVVSQIZFVmxSVmxCVkZSU1NVSIZWRVZUVkVWVZFZDRXdNREF3TWpnaE1qQXhOeTB3TIMwd05GUXdNVG94TVRveE5pNDNNekE0TnpReVdpRS0iLCJUYXJnZXRMb2NhdGlvbil6MH0&maxresults=1"
}
```

2. The health probe must contain the path **/healthstatus** else the backend health check would respond with 404.
3. The Pick Host Name from Backend should be set to true in HTTP Setting to work correctly.
4. While configuring Private Link from clients other than portal, the subnet in which the endpoint would reside must have "privateLinkServiceNetworkPolicies" property disabled. More on this [here](#).
5. The DNS of the custom domain should point to the Application Gateway public IP.
6. Due to current limitation with Application Gateway, any PaaS service's default FQDN should be added to the backend pool **after** configuring the private link, otherwise the default FQDN continues to point to its public IP. [ETA for this fix: Early 2022. As a workaround, you can even perform any PUT operation on gateway which will refresh its DNS.]
7. This document assumes that the Listener is created with HTTPS protocol and an appropriate certificate.

ETA

Private preview – 15<sup>th</sup> February 2022

-----End of File-----