Secrets NET8 EF8 Blazor VS DEV V1

Version 1: 3/12/2024

- NET CORE 8.02
- EF 8
- Blazor/Razor
- VS 17.9.2
- Development Environment

Most development secrets are stored Using Azure Key Vault, similar to those used in production. The local workstation Secrets Manager tool stores sensitive data during application development.

There are multiple ways to retrieve your secrets. In this example, we are going to use the "DefaultAzureCredential" method to get an instance of the credentials object.

Safe storage of app secrets in development in ASP.NET Core | Microsoft Learn

Table of Contents

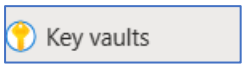
Place a Secret in an Azure Key Vault

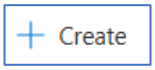
Create Azure Key Vault

If you haven't already, create an Azure Key Vault:

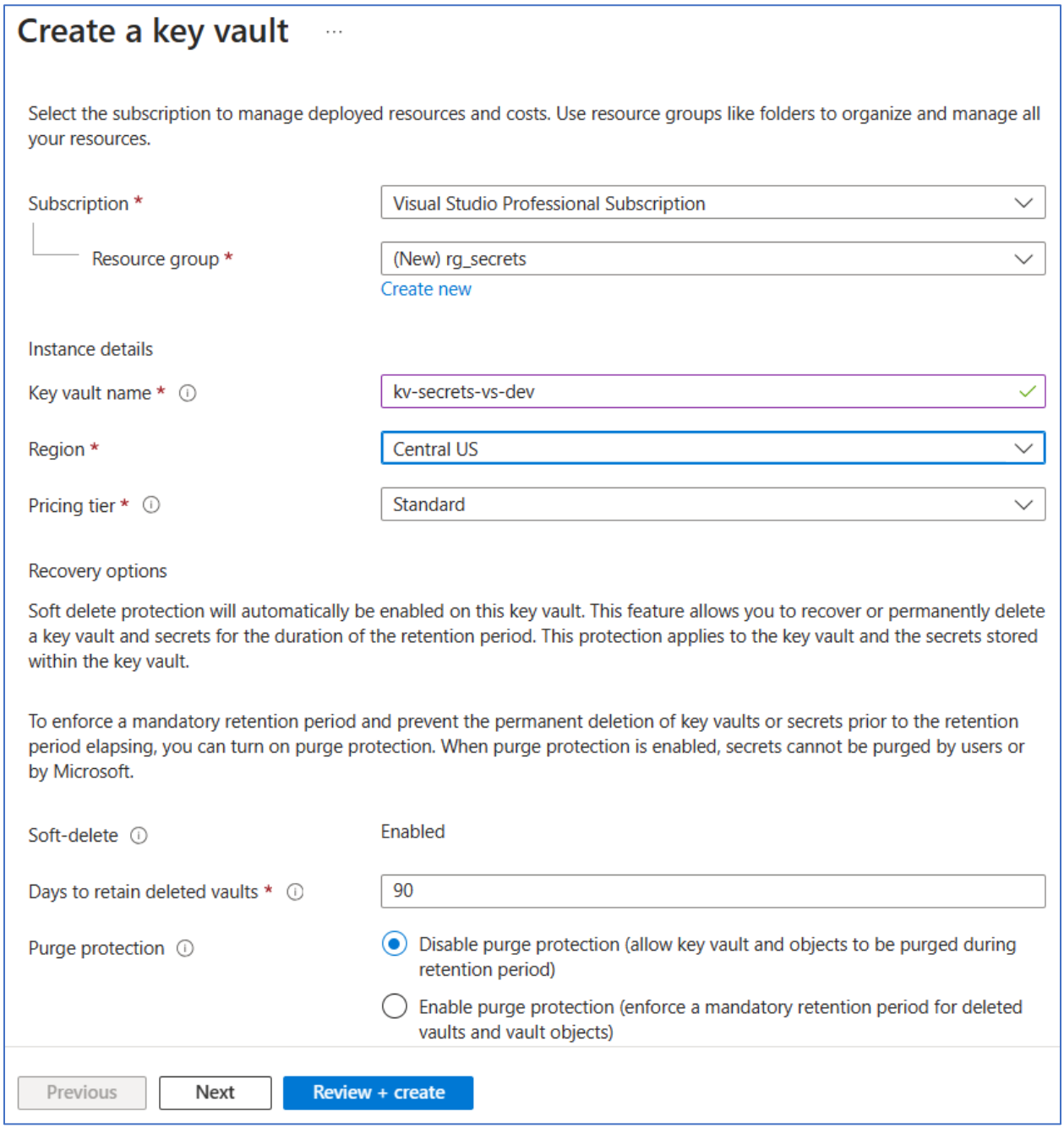
Launch Microsoft Azure:

Type "key vault" into search:

Select Key vaults: 

Select <+Create>: 

to display the Create a key vault form:



Select your Subscription: (In this example I selected my "Visual Studio Professional Subscription"

Select you Resurce Group (or create a new resource group)

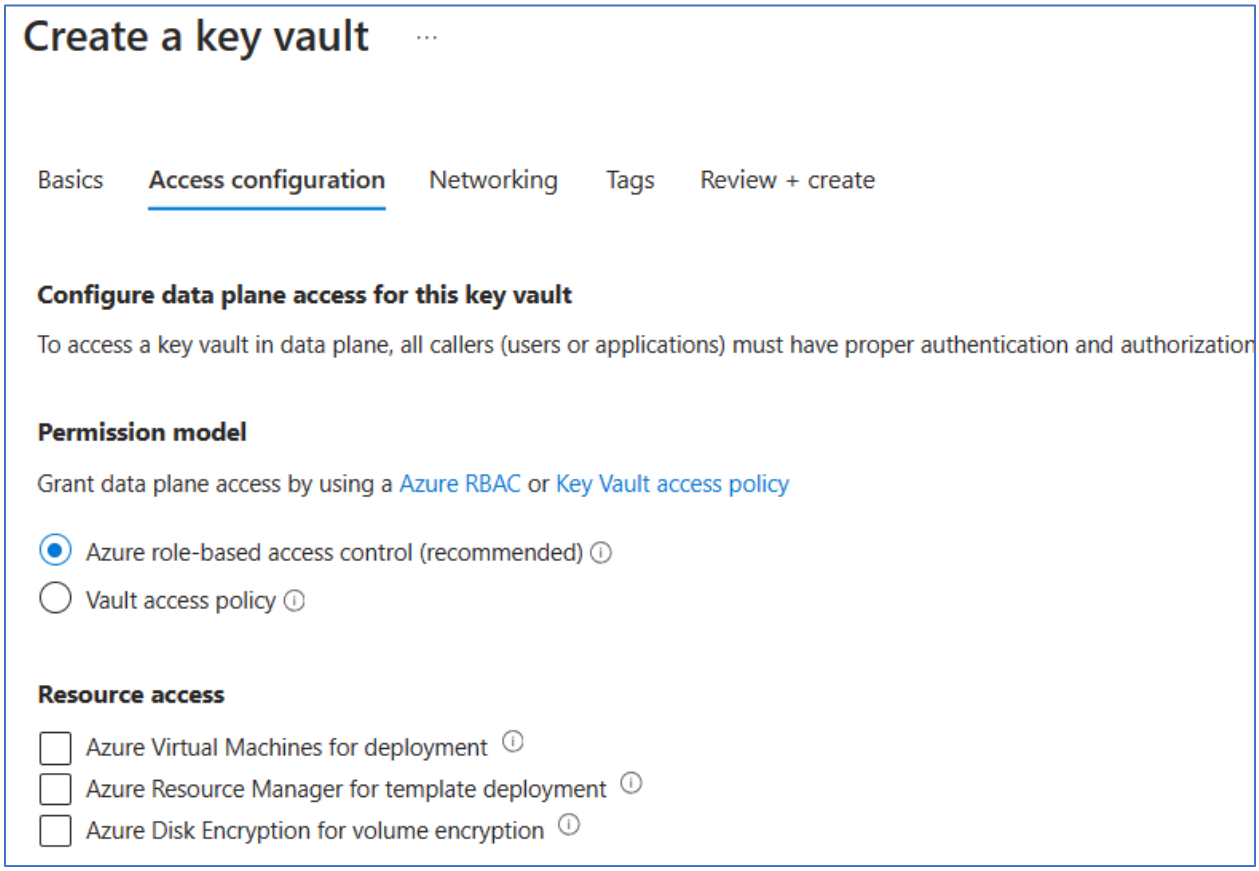
Enter your key vault name (I use a prefix followed by a dash for all of my names wherever I can). In this instance I am using kv-secrets-vs-dev. (This must be globally unique for all users in the Region)

Select the Azure Region (I use Centra US or East US 2 since both of them provide a full slate of services)

Select the Standard pricing tier since it is free.

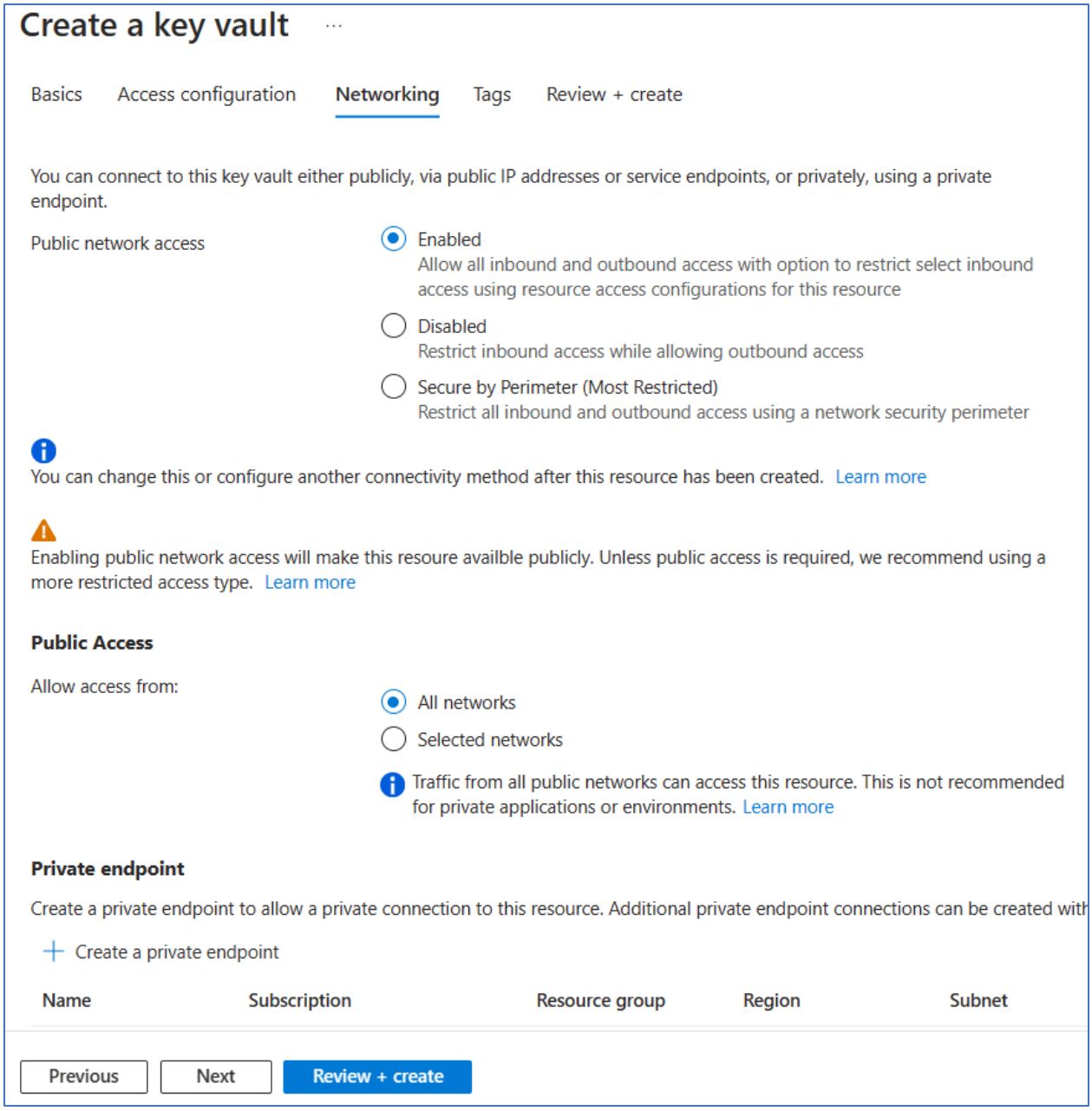
I use the defaults for the Recovery options

Select <Next>:



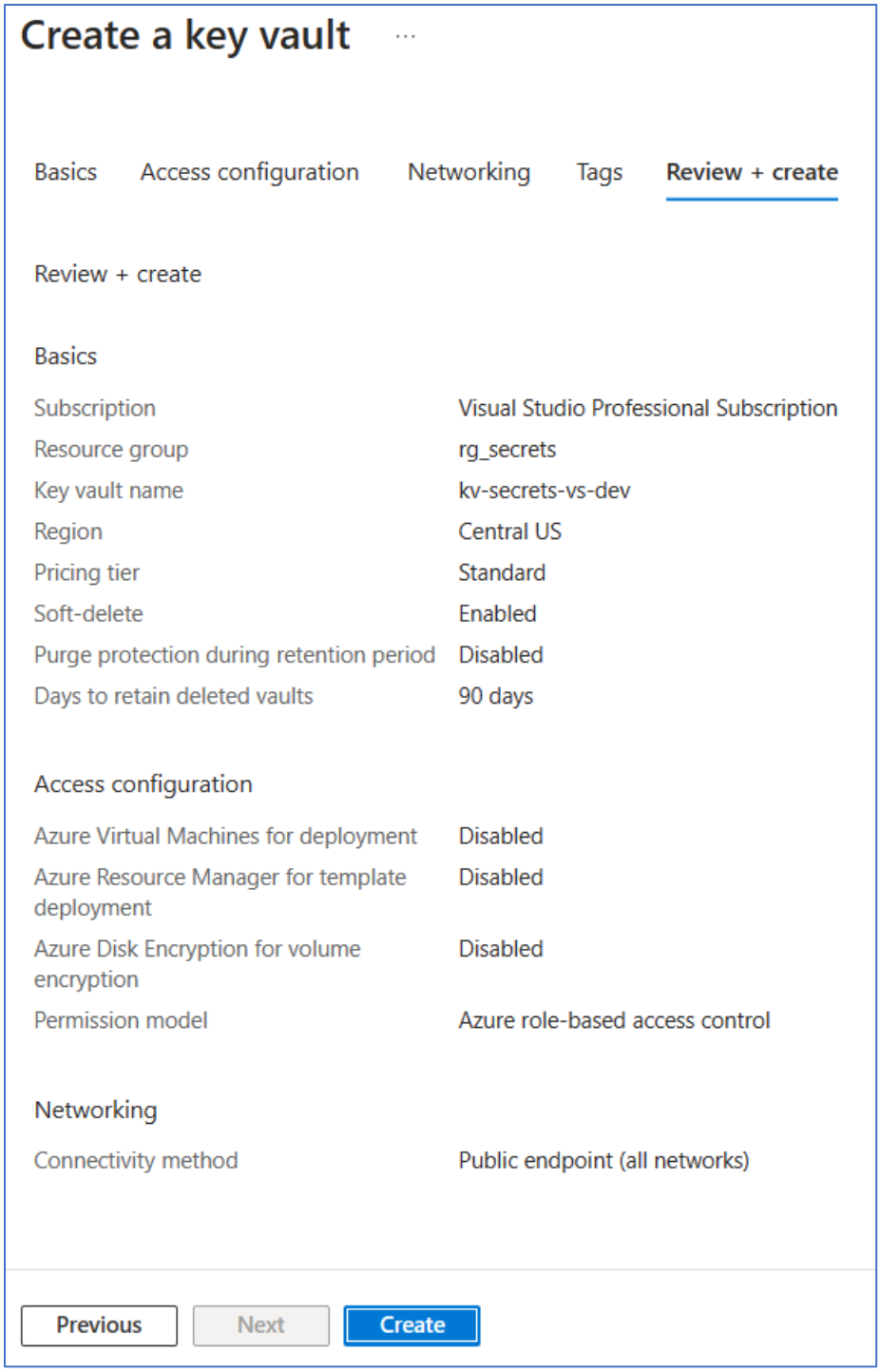
I don't make any changes to the Access Configuration page.

Select <Next>:

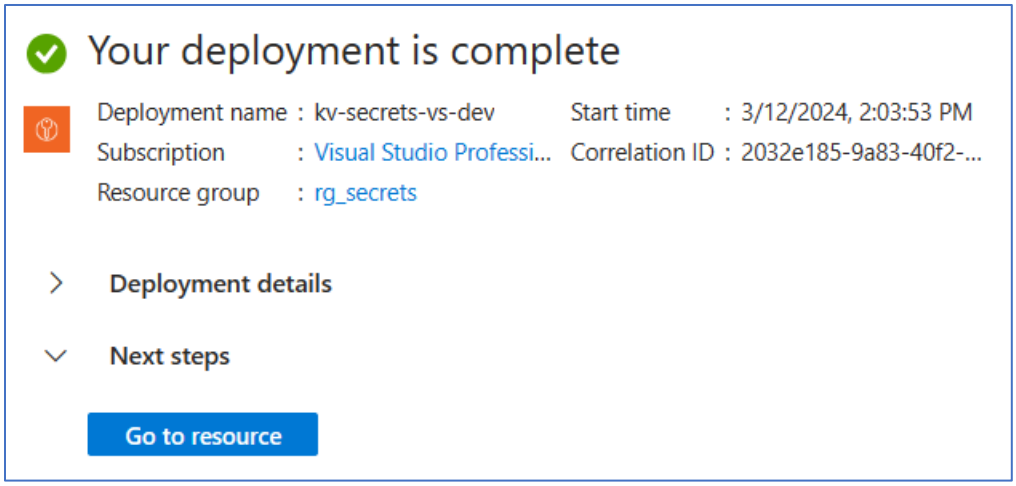


I use the defaults for the Networking page since I want to be able to access the key vault from my personal workstation.

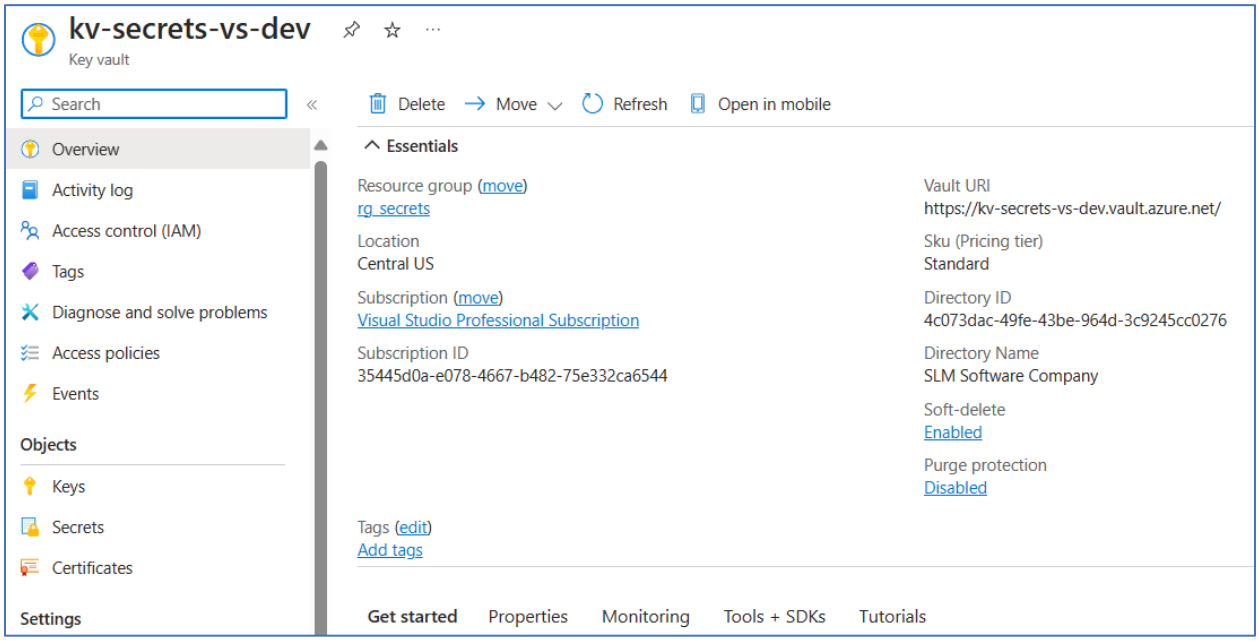
Select <Review + create>



Review your entries and select <Create> to deploy the Key Vault.



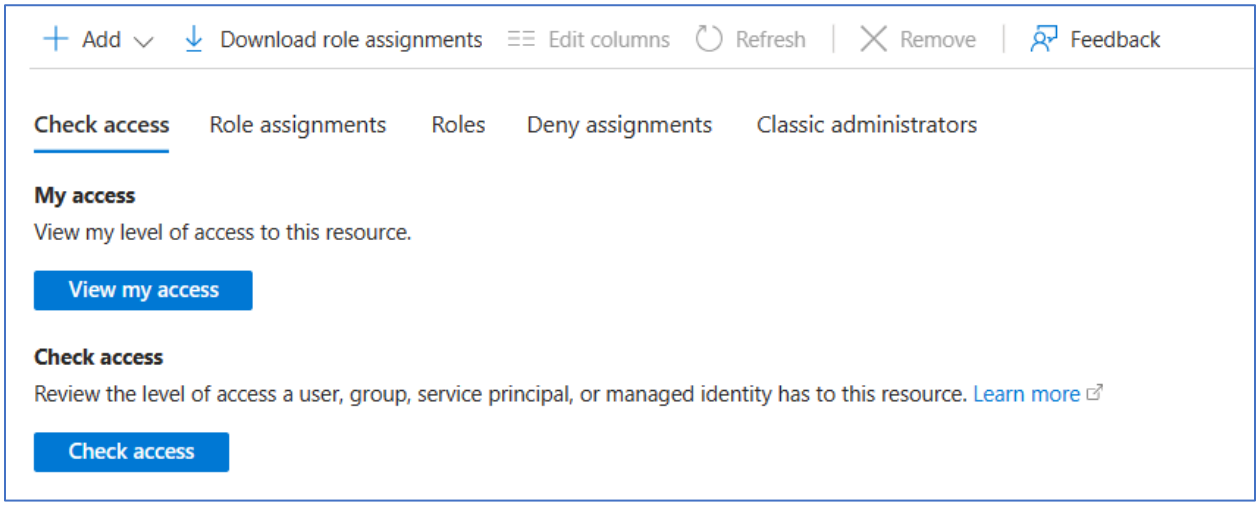
Select <Go to resource>:



You will probably not have enough access rights to add a secret.

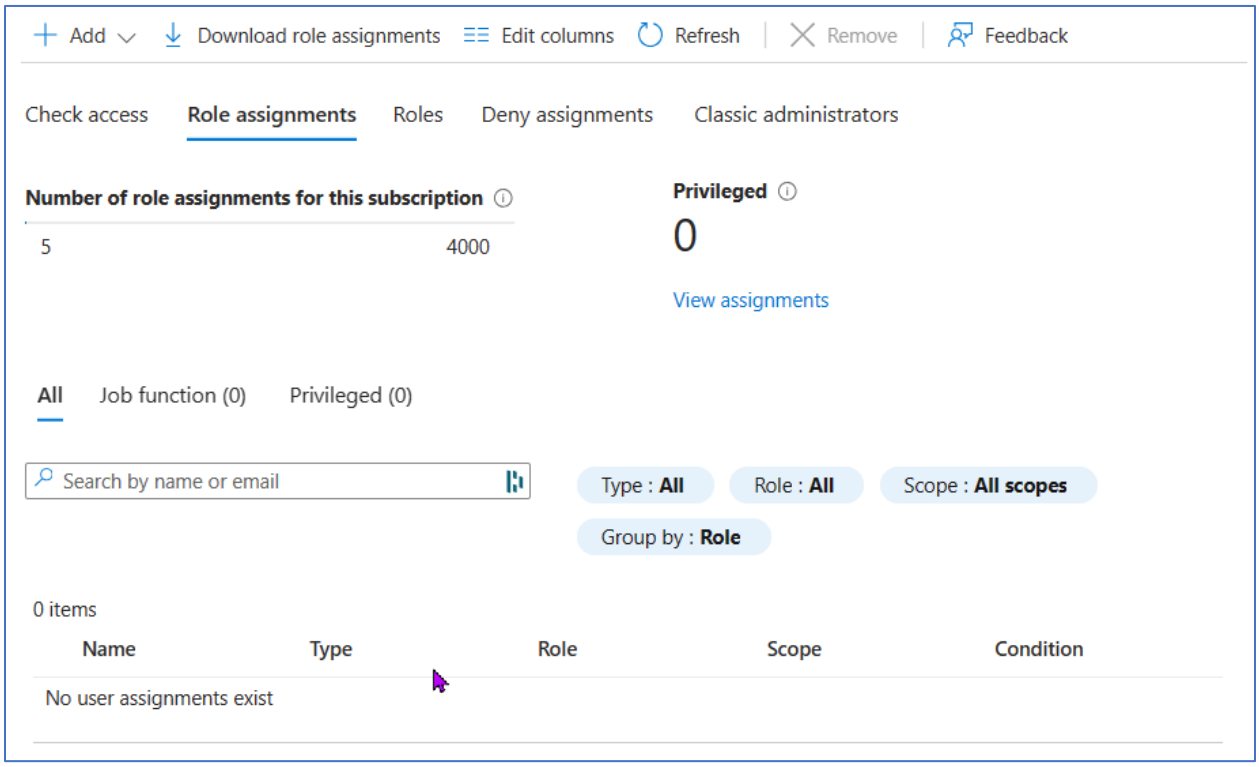
You will want to remember the “Vault URI” from this form. You will need it later.

Select <Access control (IAM)> on the left side nav bar:

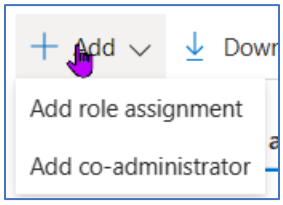


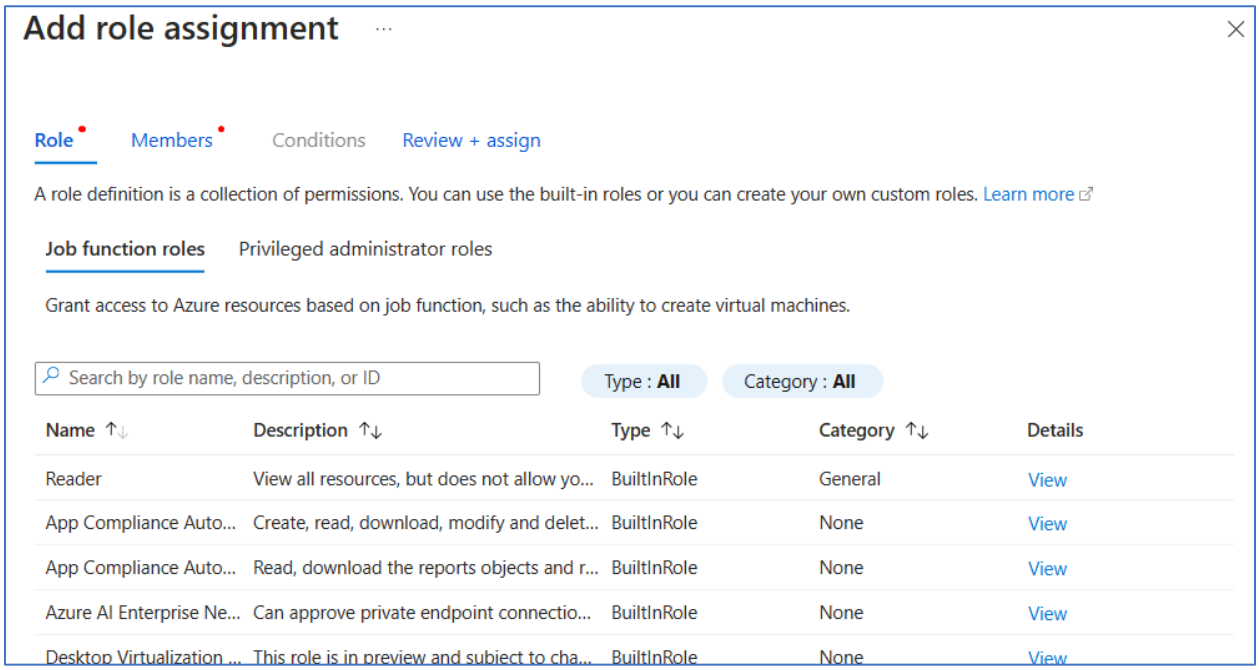
If you are the owner of the Azure Subscription you can add yourself.

Select <Role assignments> from the top menu:



Select <Add> and <Add role assignment> from the top menu:





Enter “Key Vault” into the search by role name box:

Role Members Conditions Review + assign

A role definition is a collection of permissions. You can use the built-in roles or you can create your own custom roles. Learn more

Job function roles Privileged administrator roles

Grant access to Azure resources based on job function, such as the ability to create virtual machines.

key vault

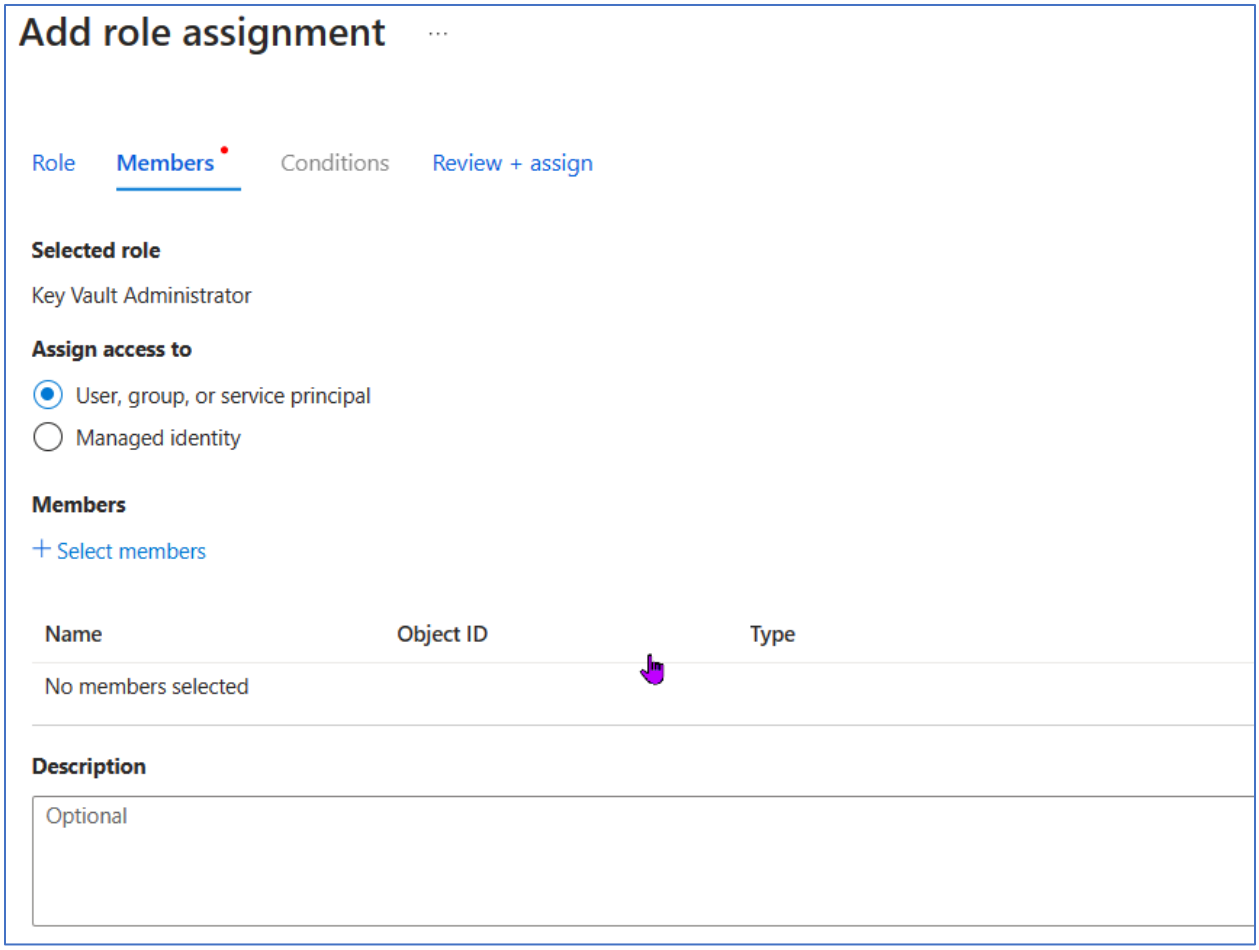
Type : All Category : All

Name	Description	Type	Category	Details
Key Vault Administrator	Perform all data plane operations on a ke...	BuiltInRole	Security	View
Key Vault Certificate U...	Read certificate contents. Only works for ...	BuiltInRole	None	View
Key Vault Certificates ...	Perform any action on the certificates of ...	BuiltInRole	Security	View
Key Vault Contributor	Lets you manage key vaults, but not acce...	BuiltInRole	Security	View
Key Vault Crypto Officer	Perform any action on the keys of a key v...	BuiltInRole	Security	View
Key Vault Crypto Servi...	Read metadata of keys and perform wrap...	BuiltInRole	Security	View
Key Vault Crypto Servi...	Release keys. Only works for key vaults th...	BuiltInRole	None	View
Key Vault Crypto User	Perform cryptographic operations using ...	BuiltInRole	Security	View
Key Vault Data Access ...	Manage access to Azure Key Vault by ad...	BuiltInRole	None	View
Key Vault Reader	Read metadata of key vaults and its certif...	BuiltInRole	Security	View
Key Vault Secrets Officer	Perform any action on the secrets of a ke...	BuiltInRole	Security	View
Key Vault Secrets User	Read secret contents. Only works for key ...	BuiltInRole	Security	View

Review + assign Previous Next

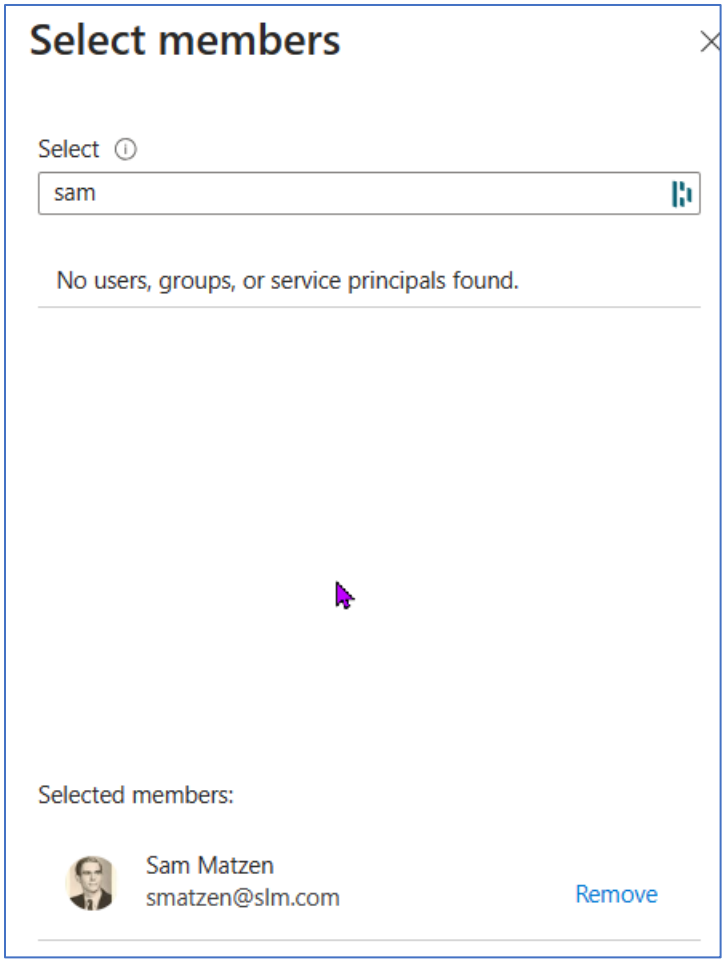
Select “Key Vault Administrator”:

Select <Members> from the top menu:

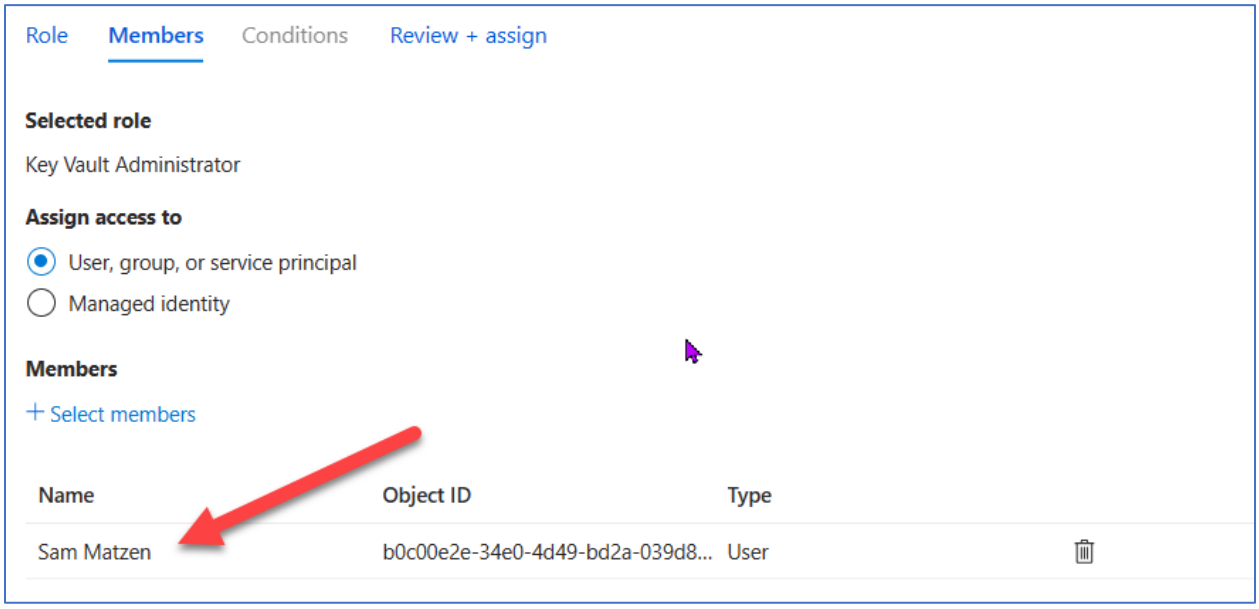


Select <+ Select members>

The Select members dialog displays and you can select yourself and others:

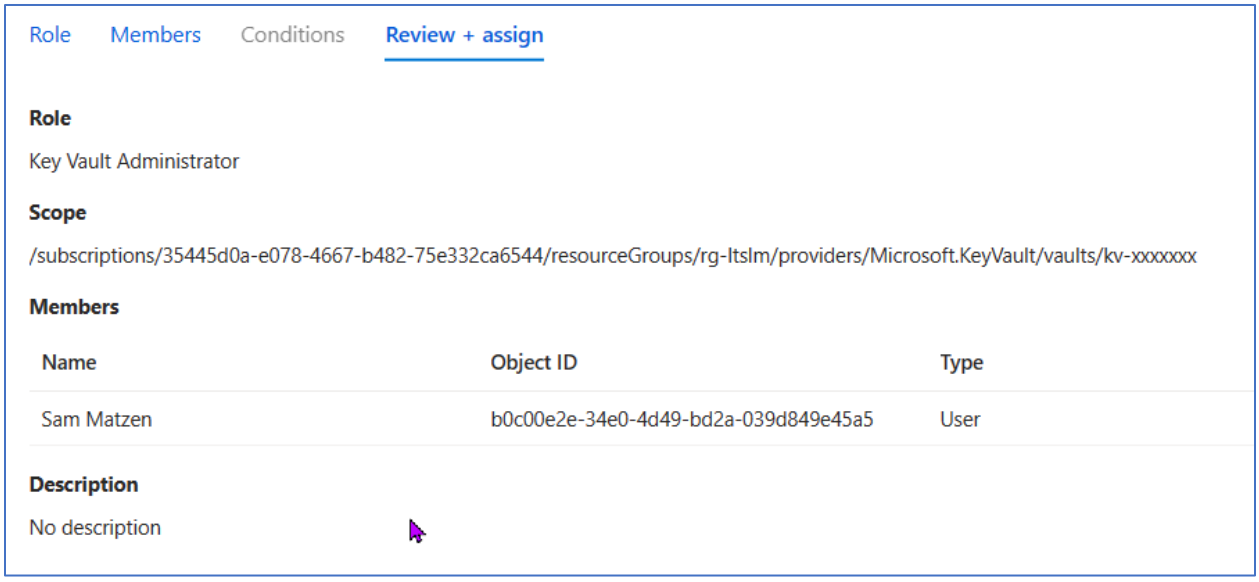


Select <Select> to add the members to the Key Vault Administrators group.



You aren't done yet:

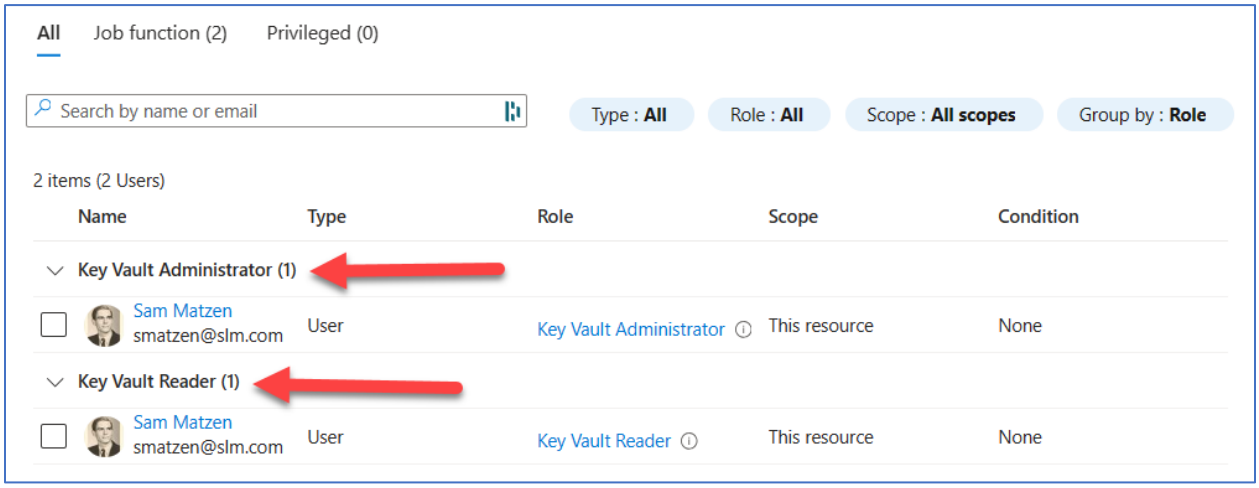
Select <Next> or <Review + assign> to get the Review + assign form:



Select <Review and assign> to assign the selected users to the role.

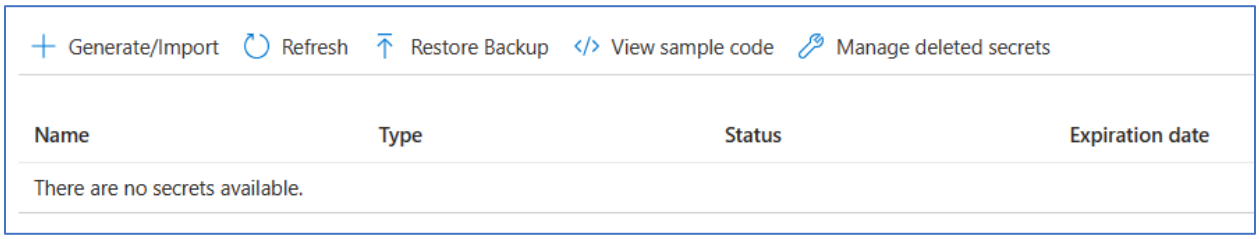
Now, go back and do the same thing for Key Vault Reader so you can have access to the secrets from your workstation.

When completed, your Role Assignments form should look something like this:



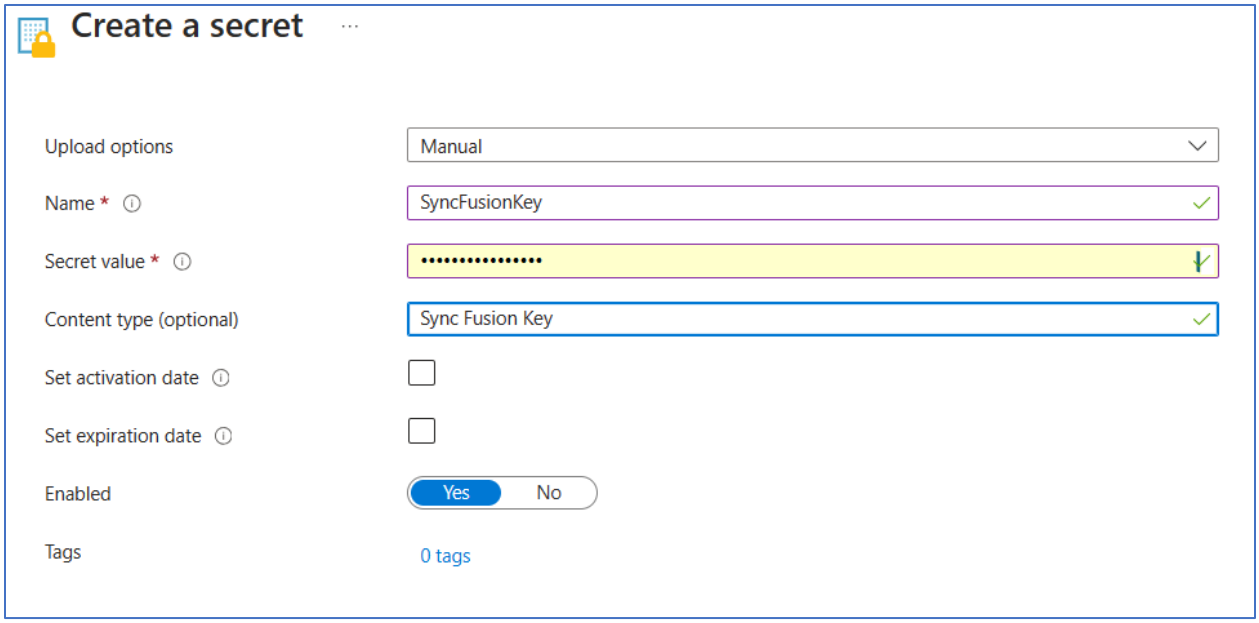
Now you should be able to add the SyncFusion secret:

Select <Secrets> from the menu on the left:



Select <+ Generate/Import> 

:



Select Upload Options of “Manual”

Enter a Name for the key. For this example, I have used “SyncFusionKey”

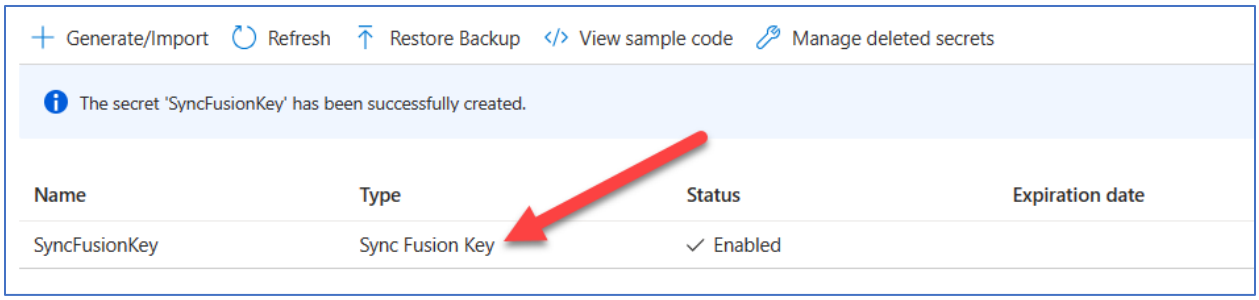
Paste your Sync Fusion Activation Key into the Secret Value

Enter a description of the key in Content Type. This shows up on other screens.

Clear both the Set Activation Date and Set Expiration Date checkboxes.

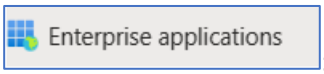
Enabled is the default.

Select <Create> to create the secret and you should see something like:



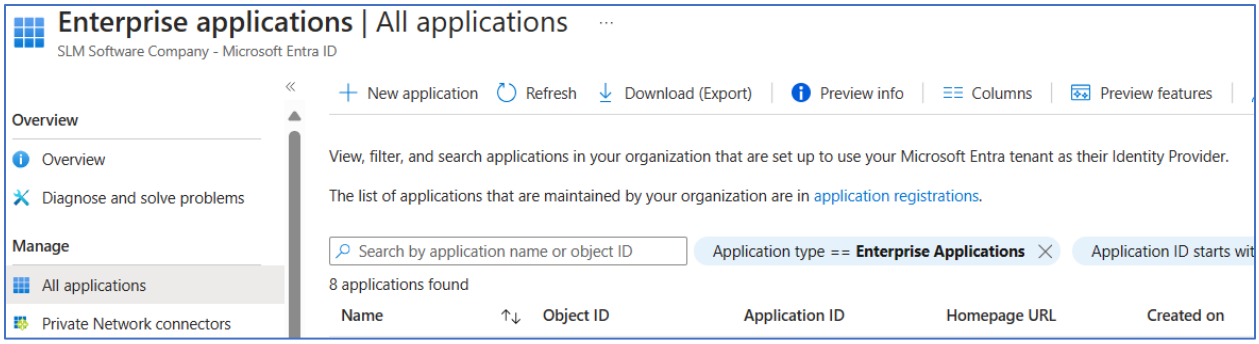
Register your application with Azure

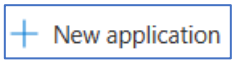
You need to register your application with Azure as an “Enterprise application”:

In Azure, enter “Enterprise Applications” in search and select 

:

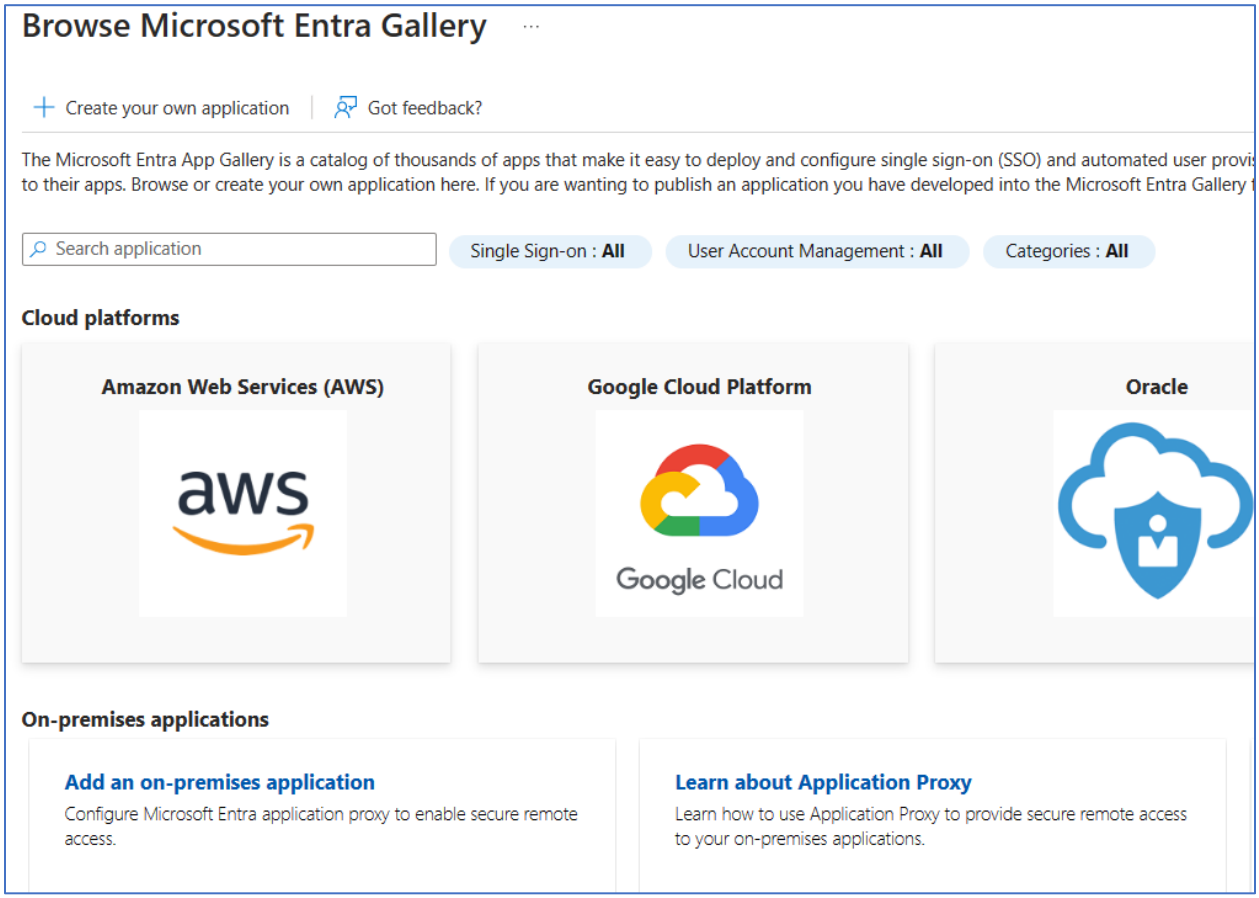
You should see a list of your currently registered applications:



Select <+ New application> 

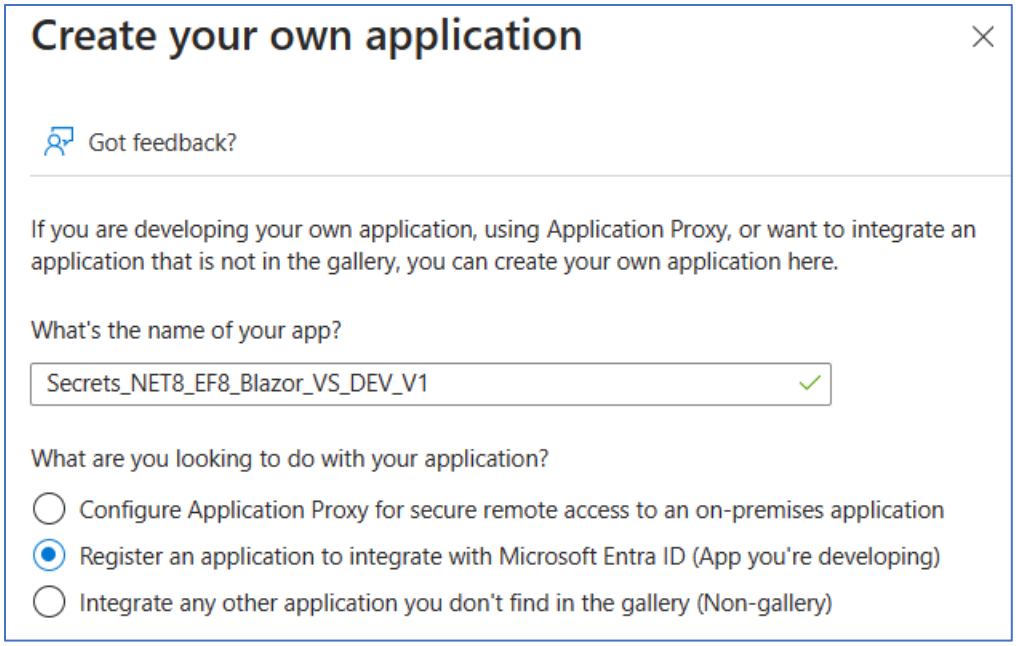
:

You should see a confusing page something like the following that has little to do with registering our new application:



Select <+ Create your own application> 

and the Create your own application dialog displays:

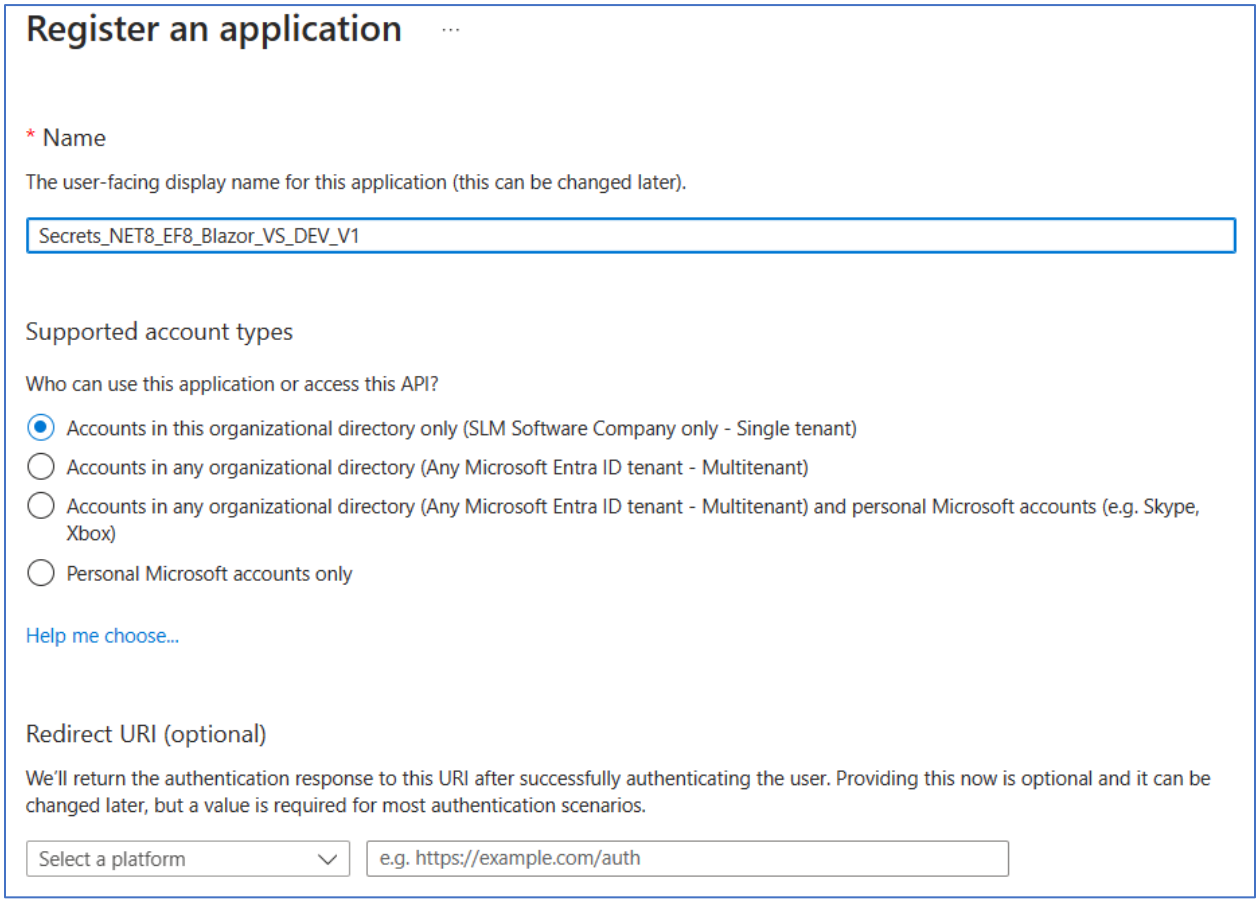


Enter the name of your application in the name text.

Select the “Register an application” radio button.

Select <Create>

The Register an application form displays:

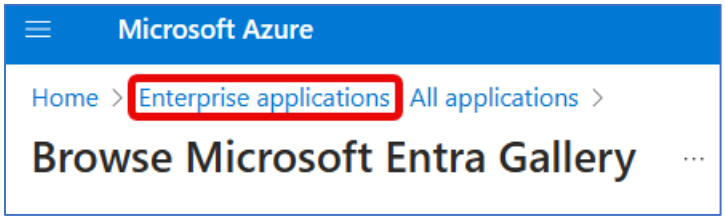


For this exercise, accept the defaults unless you know you want to make additional selections.

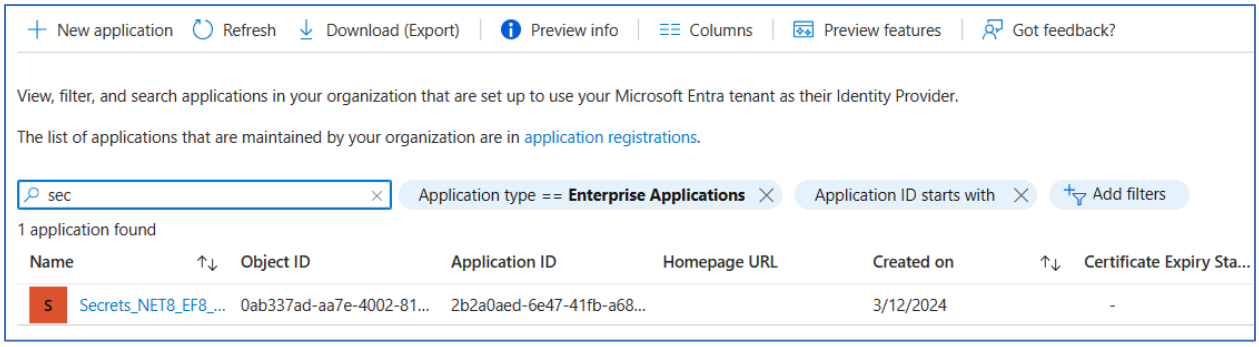
Select <Register>

The Browse Microsoft Entra Gallery form displays where you can register another application, or:

Select “Enterprise applications” from the breadcrumb bar:



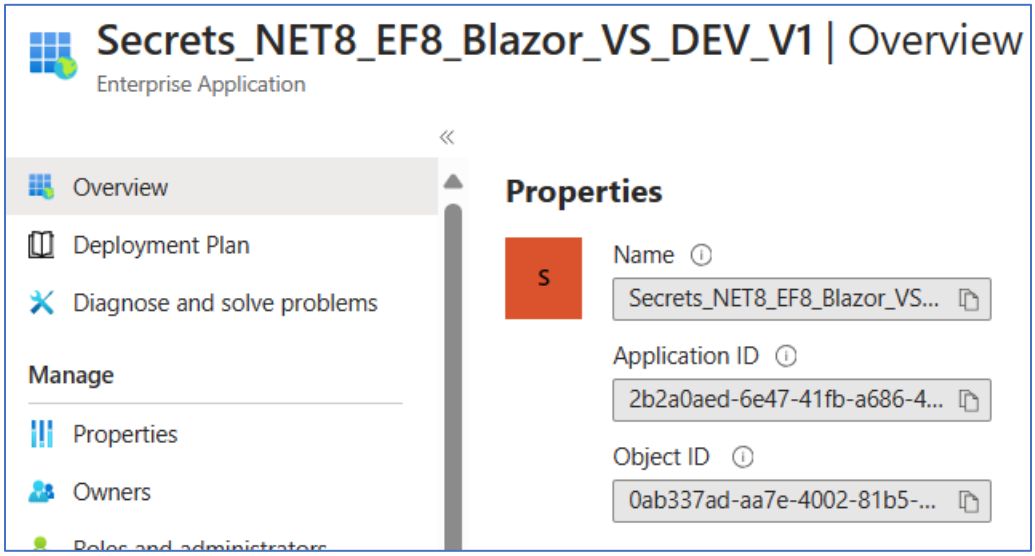
The enterprise applications form displays:



It takes a couple of minutes before your new application shows up in the Enterprise Applications form, so you may need to refresh a few times until is appears.

At a minimum you will need the Application ID to access the secrets from your application.

Select your application from the list to display the application values:



There are many options and additional data associated with an application registration, but we are only interested in the Name, Application ID, and Object ID:

Name

The application name is “Secrets_NET8_EF8_Blazor_VS_DEV_V1”

Application ID

The Application ID is “2b2a0aed-6e47-41fb-a686-4f16fea50504”

Object ID

The Object ID is “0ab337ad-aa7e-4002-81b5-7e4b5f7f0703”

Get Values from App registrations

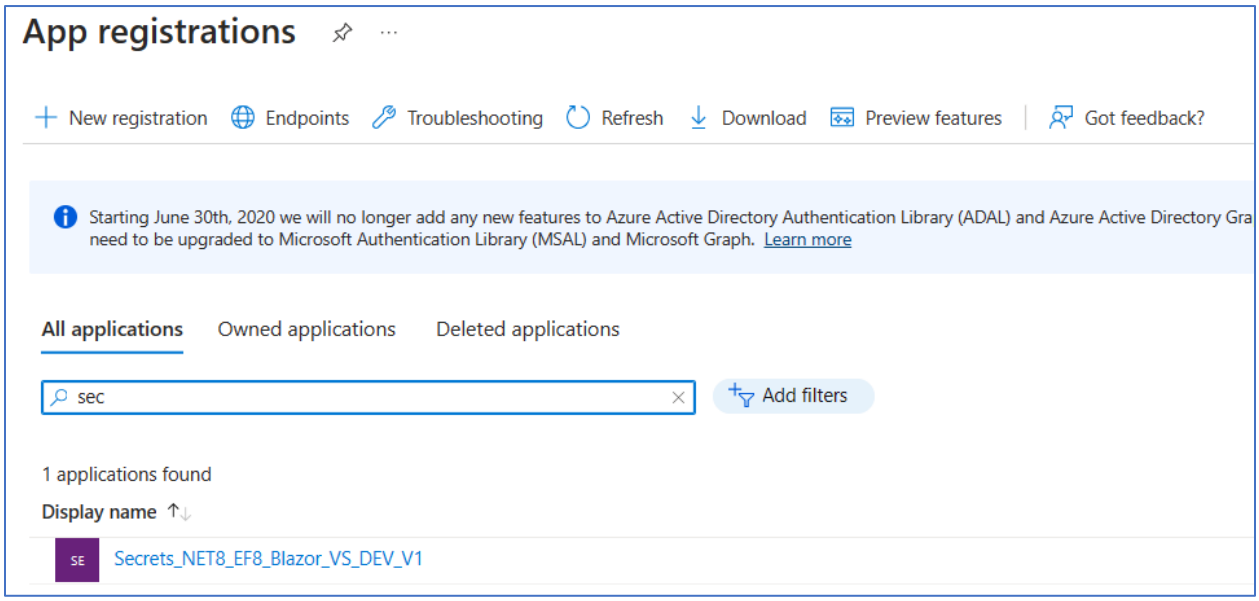
Load the application on the App Registrations screen to get the client ID and tenant ID

Select <Home> from the bread-crumb bar.

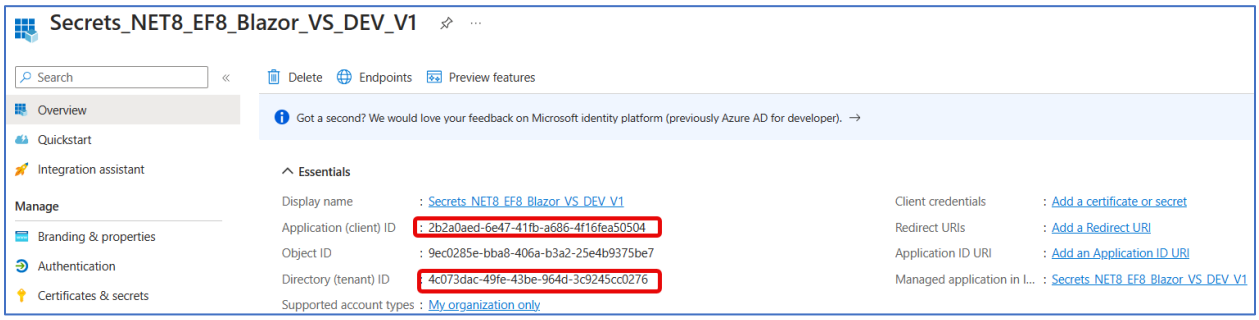
Type “App Registrations” in the search box.

Select <App registrations>

Select <All applications>



Select your new application to view the “client ID” and “tenant ID”



Prepare your project

Using PowerShell run the following on the directory containing your project:

```
dotnet add package Azure.Identity
dotnet add package Azure.Security.KeyVault.Secrets
```

These statements loaded the necessary NuGet packages.

Add Azure Secrets Class to VS Project

In the root of the project, add a new class named Secrets.cs with the following code:

```
using Azure.Identity;
using Azure.Security.KeyVault.Secrets;

namespace KeyVault01.Secrets
{
    public class Secrets()
    {
        public string GetSecret(string SecretName)
        {

            string userAssignedClientId = "2b2a0aed-6e47-41fb-a686-4f16fea50504";
            var credential = new DefaultAzureCredential(new DefaultAzureCredentialOptions
            {
                // set the user assigned identity in Azure
                 ManagedIdentityClientId = userAssignedClientId
                // exclude Managed Identity
                ,ExcludeManagedIdentityCredential = true
            });

            var client = new SecretClient(
                new Uri("https://kv-secrets-vs-dev.vault.azure.net/"), credential);

            var secret = client.GetSecret(SecretName);

            return secret.Value.Value.ToString();
        }
    }
}
```

Replace the value for userAssignedClientId with your application ID.

Replace the Uri with the Uri for your Key Vault.

Add Azure Secrets to Programs.cs

Add the following code to the Programs.cs file to retrieve a secret:

```
// retrieve SyncFusion Key from Key Vault
Secrets secrets = new Secrets();
string SyncFusionKey = secrets.GetSecret("SyncFusionKey");
Console.WriteLine(SyncFusionKey);
```

Replace the secret name as needed.

Other Secrets in the Development Code

Now we have the core secrets like Activation Keys and Connection Strings in the Azure Key Vault, we need to remove the Application ID and Secret Names from the code. We do this by using the Secret Manager.

Initialize user-secrets in .NET CLI

```
cd {your path to repos}\repos\Secrets_NET8_EF8_Blazor_VS_DEV_V1
dotnet user-secrets init -project Secrets_NET8_EF8_Blazor_VS_DEV_V1
```

This command adds a UserSecretsID to the project file:

```
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <UserSecretsId>532b3f6b-92a1-48b7-8c80-9ddaac51ec15</UserSecretsId>
  </PropertyGroup>
```

In this example we have two user-secrets we wan to put in the local secrets store so they are not checked-in to GitHub:

“Application:Id” “2b2a0aed-6e47-41fb-a686-4f16fea50504”

“Application:KeyVaultURI” “https://kv-secrets-vs-dev.vault.azure.net/”

“SecretName:SyncFusionKey” “SyncFusionKey”

Save the initial values with the these .NET CLI commands:

```
cd {your path to repos}\repos\Secrets_NET8_EF8_Blazor_VS_DEV_V1

dotnet user-secrets set "Application:Id" “2b2a0aed-6e47-41fb-a686-4f16fea50504” --project
Secrets_NET8_EF8_Blazor_VS_DEV_V1

dotnet user-secrets set “Application:KeyVaultURI” "https://kv-secrets-vs-dev.vault.azure.net/" --project
Secrets_NET8_EF8_Blazor_VS_DEV_V1

dotnet user-secrets set “SecretName:SyncFusionKey” “SyncFusionKey” --project Secrets_NET8_EF8_Blazor_VS_DEV_V1

dotnet user-secrets list --project Secrets_NET8_EF8_Blazor_VS_DEV_V1
```

The output from the “list” command is:

```
SecretName:SyncFusionKey = SyncFusionKey
Application:KeyVaultURI = https://kv-secrets-vs-dev.vault.azure.net/
Application:Id = 2b2a0aed-6e47-41fb-a686-4f16fea50504
```

Enhance local secrets from Visual Studio

With the secrets loaded into the local secrets store, you can edit the secrets store from Visual Studio by right-clicking the project name and selecting <Manage User Secrets>. (I have seen how the initial configuration can be performed from within Visual Studio, but have never tried it.)

```
{
  "Application:KeyVaultURI": "https://kv-secrets-vs-dev.vault.azure.net/",
  "Application:Id": "2b2a0aed-6e47-41fb-a686-4f16fea50504",
  "SecretName:SyncFusionKey": "SyncFusionKey"
}
```

Edit the secrets as needed and select <Save> to persist the changes.

Enhancements to retrieve local user-secrets

Now we need a way to retrieve the secrets from the local secret store.

In Program.cs:

```
var movieApiKey = builder.Configuration["Movies:ServiceApiKey"];
```

In a .razor page:

```
    private readonly IConfiguration _config;

    public IndexModel(IConfiguration config)
    {
        _config = config;
    }

    public void OnGet()
    {
        var moviesApiKey = _config["Movies:ServiceApiKey"];
```

Changes to Program.cs

```
// retrieve SyncFusion Key from Key Vault
```

```
Secrets secrets = new Secrets();
string SyncFusionKey = secrets.GetSecret(builder.Configuration,
    builder.Configuration["SecretName:SyncFusionKey"] ?? "".ToString());
Console.WriteLine(SyncFusionKey);
```

We need a reference to the Configuration object passed to the GetSecrets method, and retrieve a secret directly from the Configuration object.

Changes to Secrets.cs

```
using Azure.Identity;
using Azure.Security.KeyVault.Secrets;

namespace Secrets_NET8_EF8_Blazor_VS_DEV_V1
{
    public class Secrets()
    {

        public string GetSecret(IConfiguration config, string SecretName)
        {

            string userAssignedClientId = config["Application:Id"] ?? "".ToString();
            var credential = new DefaultAzureCredential(new DefaultAzureCredentialOptions
            {
                // set the user assigned identity in Azure
                ManagedIdentityClientId = userAssignedClientId,
                // exclude Managed Identity
                ExcludeManagedIdentityCredential = true
            });

            var client = new SecretClient(
                new Uri(config["Application:KeyVaultURI"] ?? "".ToString()), credential);

            var secret = client.GetSecret(SecretName);

            return secret.Value.Value.ToString();
        }
    }
}
```

First: We add a parameter for the Configuration (config) object reference.

Second: We retrieve the “Application:Id” from the configuration object.

Third: We retrieve the “Application:KeyVaultURI” from the configuration object.

Our code has no direct references to anything in the Azure Key Vault. The secret names are stored in the local user-secret store, so they are not checked into NuGit.