Azure Key Vault Configuration Provider in ASP.NET Core

02/07/2020 • 29 minutes to read • +7

In this article

By Andrew Stanton-Nurse

This document explains how to use the Microsoft Azure Key Vault Configuration Provider to load app configuration values from Azure Key Vault secrets. Azure Key Vault is a cloud-based service that assists in safeguarding cryptographic keys and secrets used by apps and services. Common scenarios for using Azure Key Vault with ASP.NET Core apps include:

- Controlling access to sensitive configuration data.
- Meeting the requirement for FIPS 140-2 Level 2 validated Hardware Security Modules (HSM's) when storing configuration data.

View or download sample code (how to download)

Packages

Add a package reference to the Microsoft.Extensions.Configuration.AzureKeyVault package.

Sample app

The sample app runs in either of two modes determined by the `#define` statement at the top of the *Program.cs* file:

- `Certificate` – Demonstrates the use of an Azure Key Vault Client ID and X.509 certificate to access secrets stored in Azure Key Vault. This version of the sample can be run from any location, deployed to Azure App Service or any host capable of serving an ASP.NET Core app.
- `Managed` – Demonstrates how to use Managed identities for Azure resources to authenticate the app to Azure Key Vault with Azure AD authentication without credentials stored in the app's code or configuration. When using managed identities to authenticate, an Azure AD Application ID and Password (Client Secret) aren't required. The `Managed` version of the sample must be deployed to Azure. Follow the guidance in the Use the Managed identities for Azure resources section.

For more information on how to configure a sample app using preprocessor directives (`#define`), see Introduction to ASP.NET Core.

Secret storage in the Development environment

Set secrets locally using the Secret Manager tool. When the sample app runs on the local machine in the Development environment, secrets are loaded from the local Secret Manager store.

The Secret Manager tool requires a `<UserSecretsId>` property in the app's project file. Set the property value (`{GUID}`) to any unique GUID:

XML Copy

```xml
<PropertyGroup>
  <UserSecretsId>{GUID}</UserSecretsId>
</PropertyGroup>
```

Secrets are created as name-value pairs. Hierarchical values (configuration sections) use a `:` (colon) as a separator in ASP.NET Core configuration key names.

The Secret Manager is used from a command shell opened to the project's content root, where `{SECRET NAME}` is the name and `{SECRET VALUE}` is the value:

.NET Core CLI Copy

```
dotnet user-secrets set "{SECRET NAME}" "{SECRET VALUE}"
```

Execute the following commands in a command shell from the project's content root to set the secrets for the sample app:

.NET Core CLI Copy

```
dotnet user-secrets set "SecretName" "secret_value_1_dev"
dotnet user-secrets set "Section:SecretName" "secret_value_2_dev"
```

When these secrets are stored in Azure Key Vault in the Secret storage in the Production environment with Azure Key Vault section, the `_dev` suffix is changed to `_prod`. The suffix provides a visual cue in the app's output indicating the source of the configuration values.

Secret storage in the Production environment with Azure Key Vault

The instructions provided by the Quickstart: Set and retrieve a secret from Azure Key Vault using Azure CLI topic are summarized here for creating an Azure Key Vault and storing secrets used by the sample app. Refer to the topic for further details.

1. Open Azure Cloud shell using any one of the following methods in the Azure portal:

   - Select **Try It** in the upper-right corner of a code block. Use the search string "Azure CLI" in the text box.
   - Open Cloud Shell in your browser with the **Launch Cloud Shell** button.
   - Select the **Cloud Shell** button on the menu in the upper-right corner of the Azure portal.

   For more information, see Azure CLI and Overview of Azure Cloud Shell.

2. If you aren't already authenticated, sign in with the `az login` command.

3. Create a resource group with the following command, where `{RESOURCE GROUP NAME}` is the resource group name for the new resource group and `{LOCATION}` is the Azure region (datacenter):

   azure-cli Copy

   ```
   az group create --name "{RESOURCE GROUP NAME}" --location {LOCATION}
   ```

4. Create a key vault in the resource group with the following command, where `{KEY VAULT NAME}` is the name for the new key vault and `{LOCATION}` is the Azure region (datacenter):

   azure-cli Copy

   ```
   az keyvault create --name {KEY VAULT NAME} --resource-group "{RESOURCE
   GROUP NAME}" --location {LOCATION}
   ```

5. Create secrets in the key vault as name-value pairs.

   Azure Key Vault secret names are limited to alphanumeric characters and dashes. Hierarchical values (configuration sections) use `--` (two dashes) as a separator. Colons, which are normally used to delimit a section from a subkey in ASP.NET Core configuration, aren't allowed in key vault secret names. Therefore, two dashes are used and swapped for a colon when the secrets are loaded into the app's configuration.

   The following secrets are for use with the sample app. The values include a `_prod` suffix to distinguish them from the `_dev` suffix values loaded in the Development environment from User Secrets. Replace `{KEY VAULT NAME}` with the name of the key vault that you created in the prior step:

   azure-cli Copy

   ```
   az keyvault secret set --vault-name {KEY VAULT NAME} --name
   "SecretName" --value "secret_value_1_prod"
   az keyvault secret set --vault-name {KEY VAULT NAME} --name "Section--
   SecretName" --value "secret_value_2_prod"
   ```

Use Application ID and X.509 certificate for non-Azure-hosted apps

Configure Azure AD, Azure Key Vault, and the app to use an Azure Active Directory Application ID and X.509 certificate to authenticate to a key vault **when the app is hosted outside of Azure**. For more information, see About keys, secrets, and certificates.

ⓘ Note

Although using an Application ID and X.509 certificate is supported for apps hosted in Azure, we recommend using Managed identities for Azure resources when hosting an app in Azure. Managed identities don't require storing a certificate in the app or in the development environment.

The sample app uses an Application ID and X.509 certificate when the `#define` statement at the top of the *Program.cs* file is set to `Certificate`.

1. Create a PKCS#12 archive (*.pfx*) certificate. Options for creating certificates include MakeCert on Windows and OpenSSL.
2. Install the certificate into the current user's personal certificate store. Marking the key as exportable is optional. Note the certificate's thumbprint, which is used later in this process.
3. Export the PKCS#12 archive (*.pfx*) certificate as a DER-encoded certificate (*.cer*).
4. Register the app with Azure AD (**App registrations**).
5. Upload the DER-encoded certificate (*.cer*) to Azure AD:
   a. Select the app in Azure AD.
   b. Navigate to **Certificates & secrets**.
   c. Select **Upload certificate** to upload the certificate, which contains the public key. A *.cer*, *.pem*, or *.crt* certificate is acceptable.
6. Store the key vault name, Application ID, and certificate thumbprint in the app's *appsettings.json* file.
7. Navigate to **Key vaults** in the Azure portal.
8. Select the key vault that you created in the Secret storage in the Production environment with Azure Key Vault section.
9. Select **Access policies**.
10. Select **Add Access Policy**.
11. Open **Secret permissions** and provide the app with **Get** and **List** permissions.
12. Select **Select principal** and select the registered app by name. Select the **Select** button.
13. Select **OK**.
14. Select **Save**.
15. Deploy the app.

The `Certificate` sample app obtains its configuration values from `IConfigurationRoot` with the same name as the secret name:

- Non-hierarchical values: The value for `SecretName` is obtained with `config["SecretName"]`.
- Hierarchical values (sections): Use `:` (colon) notation or the `GetSection` extension method. Use either of these approaches to obtain the configuration value:
  - `config["Section:SecretName"]`
  - `config.GetSection("Section")["SecretName"]`

The X.509 certificate is managed by the OS. The app calls AddAzureKeyVault with values supplied by the *appsettings.json* file:

C# Copy

```
// using System.Linq;
// using System.Security.Cryptography.X509Certificates;
// using Microsoft.Extensions.Configuration;

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((context, config) =>
        {
            if (context.HostingEnvironment.IsProduction())
            {
                var builtConfig = config.Build();

                using (var store = new X509Store(StoreLocation.CurrentUser))
                {
                    store.Open(OpenFlags.ReadOnly);
                    var certs = store.Certificates
                        .Find(X509FindType.FindByThumbprint,
                            builtConfig["AzureADCertThumbprint"], false);

                    config.AddAzureKeyVault(

$"https://{builtConfig["KeyVaultName"]}.vault.azure.net/",
                        builtConfig["AzureADApplicationId"],
                        certs.OfType<X509Certificate2>().Single());

                    store.Close();
                }
            }
        })
        .UseStartup<Startup>();
```

Example values:

- Key vault name: `contosovault`
- Application ID: `627e911e-43cc-61d4-992e-12db9c81b413`
- Certificate thumbprint: `fe14593dd66b2406c5269d742d04b6e1ab03adb1`

appsettings.json:

JSON Copy

```
{
  "KeyVaultName": "Key Vault Name",
  "AzureADApplicationId": "Azure AD Application ID",
  "AzureADCertThumbprint": "Azure AD Certificate Thumbprint"
}
```

When you run the app, a webpage shows the loaded secret values. In the Development environment, secret values load with the `_dev` suffix. In the Production environment, the values load with the `_prod` suffix.

Use Managed identities for Azure resources

An app deployed to Azure can take advantage of [Managed identities for Azure resources](), which allows the app to authenticate with Azure Key Vault using Azure AD authentication without credentials (Application ID and Password/Client Secret) stored in the app.

The sample app uses Managed identities for Azure resources when the `#define` statement at the top of the *Program.cs* file is set to `Managed`.

Enter the vault name into the app's *appsettings.json* file. The sample app doesn't require an Application ID and Password (Client Secret) when set to the `Managed` version, so you can ignore those configuration entries. The app is deployed to Azure, and Azure authenticates the app to access Azure Key Vault only using the vault name stored in the *appsettings.json* file.

Deploy the sample app to Azure App Service.

An app deployed to Azure App Service is automatically registered with Azure AD when the service is created. Obtain the Object ID from the deployment for use in the following command. The Object ID is shown in the Azure portal on the **Identity** panel of the App Service.

Using Azure CLI and the app's Object ID, provide the app with `list` and `get` permissions to access the key vault:

azure-cli Copy

```
az keyvault set-policy --name {KEY VAULT NAME} --object-id {OBJECT ID} --
secret-permissions get list
```

Restart the app using Azure CLI, PowerShell, or the Azure portal.

The sample app:

- Creates an instance of the `AzureServiceTokenProvider` class without a connection string. When a connection string isn't provided, the provider attempts to obtain an access token from Managed identities for Azure resources.

- A new KeyVaultClient is created with the `AzureServiceTokenProvider` instance token callback.
- The KeyVaultClient instance is used with a default implementation of IKeyVaultSecretManager that loads all secret values and replaces double-dashes (`--`) with colons (`:`) in key names.

C# Copy

```csharp
// using Microsoft.Azure.KeyVault;
// using Microsoft.Azure.Services.AppAuthentication;
// using Microsoft.Extensions.Configuration.AzureKeyVault;

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((context, config) =>
        {
            if (context.HostingEnvironment.IsProduction())
            {
                var builtConfig = config.Build();

                var azureServiceTokenProvider = new
AzureServiceTokenProvider();
                var keyVaultClient = new KeyVaultClient(
                    new KeyVaultClient.AuthenticationCallback(
                        azureServiceTokenProvider.KeyVaultTokenCallback));

                config.AddAzureKeyVault(

$"https://{builtConfig["KeyVaultName"]}.vault.azure.net/",
                    keyVaultClient,
                    new DefaultKeyVaultSecretManager());
            }
        })
        .UseStartup<Startup>();
```

Key vault name example value: `contosovault`

appsettings.json:

JSON Copy

```json
{
  "KeyVaultName": "Key Vault Name"
}
```

When you run the app, a webpage shows the loaded secret values. In the Development environment, secret values have the `_dev` suffix because they're provided by User

Secrets. In the Production environment, the values load with the `_prod` suffix because they're provided by Azure Key Vault.

If you receive an `Access denied` error, confirm that the app is registered with Azure AD and provided access to the key vault. Confirm that you've restarted the service in Azure.

For information on using the provider with a managed identity and an Azure DevOps pipeline, see Create an Azure Resource Manager service connection to a VM with a managed service identity.

Use a key name prefix

AddAzureKeyVault provides an overload that accepts an implementation of IKeyVaultSecretManager, which allows you to control how key vault secrets are converted into configuration keys. For example, you can implement the interface to load secret values based on a prefix value you provide at app startup. This allows you, for example, to load secrets based on the version of the app.

⚠ **Warning**

Don't use prefixes on key vault secrets to place secrets for multiple apps into the same key vault or to place environmental secrets (for example, *development* versus *production* secrets) into the same vault. We recommend that different apps and development/production environments use separate key vaults to isolate app environments for the highest level of security.

In the following example, a secret is established in the key vault (and using the Secret Manager tool for the Development environment) for `5000-AppSecret` (periods aren't allowed in key vault secret names). This secret represents an app secret for version 5.0.0.0 of the app. For another version of the app, 5.1.0.0, a secret is added to the key vault (and using the Secret Manager tool) for `5100-AppSecret`. Each app version loads its versioned secret value into its configuration as `AppSecret`, stripping off the version as it loads the secret.

AddAzureKeyVault is called with a custom IKeyVaultSecretManager:

C# Copy

```
config.AddAzureKeyVault(
    $"https://{builtConfig["KeyVaultName"]}.vault.azure.net/",
    builtConfig["AzureADApplicationId"],
    certs.OfType<X509Certificate2>().Single(),
    new PrefixKeyVaultSecretManager(versionPrefix));
```

The IKeyVaultSecretManager implementation reacts to the version prefixes of secrets to load the proper secret into configuration:

- `Load` loads a secret when its name starts with the prefix. Other secrets aren't loaded.
- `GetKey`:
  - Removes the prefix from the secret name.
  - Replaces two dashes in any name with the `KeyDelimiter`, which is the delimiter used in configuration (usually a colon). Azure Key Vault doesn't allow a colon in secret names.

C# Copy

```csharp
public class PrefixKeyVaultSecretManager : IKeyVaultSecretManager
{
    private readonly string _prefix;

    public PrefixKeyVaultSecretManager(string prefix)
    {
        _prefix = $"{prefix}-";
    }

    public bool Load(SecretItem secret)
    {
        return secret.Identifier.Name.StartsWith(_prefix);
    }

    public string GetKey(SecretBundle secret)
    {
        return secret.SecretIdentifier.Name
            .Substring(_prefix.Length)
            .Replace("--", ConfigurationPath.KeyDelimiter);
    }
}
```

The `Load` method is called by a provider algorithm that iterates through the vault secrets to find the ones that have the version prefix. When a version prefix is found with `Load`, the algorithm uses the `GetKey` method to return the configuration name of the secret name. It strips off the version prefix from the secret's name and returns the rest of the secret name for loading into the app's configuration name-value pairs.

When this approach is implemented:

1. The app's version specified in the app's project file. In the following example, the app's version is set to `5.0.0.0`:

   XML Copy

```xml
<PropertyGroup>
  <Version>5.0.0.0</Version>
</PropertyGroup>
```

2. Confirm that a `<UserSecretsId>` property is present in the app's project file, where `{GUID}` is a user-supplied GUID:

   XML Copy

   ```xml
   <PropertyGroup>
     <UserSecretsId>{GUID}</UserSecretsId>
   </PropertyGroup>
   ```

   Save the following secrets locally with the Secret Manager tool:

   .NET Core CLI Copy

   ```
   dotnet user-secrets set "5000-AppSecret" "5.0.0.0_secret_value_dev"
   dotnet user-secrets set "5100-AppSecret" "5.1.0.0_secret_value_dev"
   ```

3. Secrets are saved in Azure Key Vault using the following Azure CLI commands:

   azure-cli Copy

   ```
   az keyvault secret set --vault-name {KEY VAULT NAME} --name "5000-
   AppSecret" --value "5.0.0.0_secret_value_prod"
   az keyvault secret set --vault-name {KEY VAULT NAME} --name "5100-
   AppSecret" --value "5.1.0.0_secret_value_prod"
   ```

4. When the app is run, the key vault secrets are loaded. The string secret for `5000-AppSecret` is matched to the app's version specified in the app's project file (`5.0.0.0`).

5. The version, `5000` (with the dash), is stripped from the key name. Throughout the app, reading configuration with the key `AppSecret` loads the secret value.

6. If the app's version is changed in the project file to `5.1.0.0` and the app is run again, the secret value returned is `5.1.0.0_secret_value_dev` in the Development environment and `5.1.0.0_secret_value_prod` in Production.

Note

You can also provide your own KeyVaultClient implementation to AddAzureKeyVault. A custom client permits sharing a single instance of the client across the app.

Bind an array to a class

The provider is capable of reading configuration values into an array for binding to a POCO array.

When reading from a configuration source that allows keys to contain colon (`:`) separators, a numeric key segment is used to distinguish the keys that make up an array (`:0:`, `:1:`, … `:{n}:`). For more information, see Configuration: Bind an array to a class.

Azure Key Vault keys can't use a colon as a separator. The approach described in this topic uses double dashes (`--`) as a separator for hierarchical values (sections). Array keys are stored in Azure Key Vault with double dashes and numeric key segments (`--0--`, `--1--`, … `--{n}--`).

Examine the following Serilog logging provider configuration provided by a JSON file. There are two object literals defined in the `WriteTo` array that reflect two Serilog *sinks*, which describe destinations for logging output:

JSON Copy

```json
"Serilog": {
  "WriteTo": [
    {
      "Name": "AzureTableStorage",
      "Args": {
        "storageTableName": "logs",
        "connectionString": "DefaultEnd...ountKey=Eby8...GMGw=="
      }
    },
    {
      "Name": "AzureDocumentDB",
      "Args": {
        "endpointUrl": "https://contoso.documents.azure.com:443",
        "authorizationKey": "Eby8...GMGw=="
      }
    }
  ]
}
```

The configuration shown in the preceding JSON file is stored in Azure Key Vault using double dash (`--`) notation and numeric segments:

Key	Value
`Serilog--WriteTo--0--Name`	`AzureTableStorage`
`Serilog--WriteTo--0--Args--storageTableName`	`logs`
`Serilog--WriteTo--0--Args--connectionString`	`DefaultEnd...ountKey=Eby8...GMGw==`
`Serilog--WriteTo--1--Name`	`AzureDocumentDB`
`Serilog--WriteTo--1--Args--endpointUrl`	`https://contoso.documents.azure.com:443`
`Serilog--WriteTo--1--Args--authorizationKey`	`Eby8...GMGw==`

Reload secrets

Secrets are cached until `IConfigurationRoot.Reload()` is called. Expired, disabled, and updated secrets in the key vault are not respected by the app until `Reload` is executed.

C# Copy

```
Configuration.Reload();
```

Disabled and expired secrets

Disabled and expired secrets throw a KeyVaultErrorException. To prevent the app from throwing, provide the configuration using a different configuration provider or update the disabled or expired secret.

Troubleshoot

When the app fails to load configuration using the provider, an error message is written to the ASP.NET Core Logging infrastructure. The following conditions will prevent configuration from loading:

- The app or certificate isn't configured correctly in Azure Active Directory.
- The key vault doesn't exist in Azure Key Vault.
- The app isn't authorized to access the key vault.
- The access policy doesn't include `Get` and `List` permissions.
- In the key vault, the configuration data (name-value pair) is incorrectly named, missing, disabled, or expired.

- The app has the wrong key vault name (`KeyVaultName`), Azure AD Application Id (`AzureADApplicationId`), or Azure AD certificate thumbprint (`AzureADCertThumbprint`).
- The configuration key (name) is incorrect in the app for the value you're trying to load.
- When adding the access policy for the app to the key vault, the policy was created, but the **Save** button wasn't selected in the **Access policies** UI.

Additional resources

- Configuration in ASP.NET Core
- Microsoft Azure: Key Vault
- Microsoft Azure: Key Vault Documentation
- How to generate and transfer HSM-protected keys for Azure Key Vault
- KeyVaultClient Class
- Quickstart: Set and retrieve a secret from Azure Key Vault by using a .NET web app
- Tutorial: How to use Azure Key Vault with Azure Windows Virtual Machine in .NET

Is this page helpful?

Yes No