**ANG-307**

*POC Document*

|  |  |
| --- | --- |
| POC Title: | Implementation of HashiCorp with .NET API |
| Created by: | Md. Sahidul Islam, Jr. Software Engineer |
| Reviewed by: | Md. Mahedee Hasan, Head of Soft. Dev Eng |
| Approved by: | Md. Mahedee Hasan, Head of Soft. Dev Eng |
| Confidentiality level: | Internal |

**DOCUMENT CHANGE HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Description of change** |
| 21-July-2022 | 1.0 | Md. Sahidul Islam | Initial Draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[1. Starting a dev Server 4](#_Toc109294251)

[2. Using HashiCorp Vault C# Client with .NET CORE 4](#_Toc109294252)

[2.1. Step 1: Retrieve the demo application 5](#_Toc109294253)

[2.2. Add a configuration provider to access Vault 5](#_Toc109294254)

[2.3. Run the application using a static database password 9](#_Toc109294255)

[2.1. Run the application using a dynamic database username and password 11](#_Toc109294256)

# Starting a dev Server

1. Dev Server 🡺Start a dev server by running: vault server –dev
2. Root Client 🡺 Login as root and create a role for the application. Please follow this [Secure Key Vault.docx](https://asainternational.sharepoint.com/:w:/r/sites/KB/Information%20Technology/R%26D/Docs/Secure%20Key%20Vault.docx?d=w01386c13df1343c9add3a969d393defc&csf=1&web=1&e=IAiSCr)

# Using HashiCorp Vault C# Client with .NET CORE

If your .NET application needs some secrets (e.g. database credentials), your organization might offer HashiCorp Vault to store and manage them for you. As a developer, you need a way to retrieve secrets from Vault for your application to use.

This tutorial demonstrates how to use a Vault C# client to retrieve static and dynamic Microsoft SQL Server database credentials from Vault. The ASP.NET Core application uses [Vault Sharp](https://github.com/rajanadar/VaultSharp), a library which provides lightweight client-side support for connecting to Vault. When database credentials change, you will need to restart the example application.

## Step 1: Retrieve the demo application

You will find the demo application here:

* We have used 192.168.97.22:1433 server as database storage.
* We will perform CRUD operations in database named “HashiCorp” by using connection strings from the vault.
* For static credentials, put a key-value pair like this to secret engine of the vault. For help: [Secure Key Vault.docx](https://asainternational.sharepoint.com/:w:/r/sites/KB/Information%20Technology/R%26D/Docs/Secure%20Key%20Vault.docx?d=w01386c13df1343c9add3a969d393defc&csf=1&web=1&e=IAiSCr) .
* You can also use VaultSharp to put the credentials. <http://192.168.100.42/dev/ang-rnd/-/tree/master/src/SKS/HashiCorp>
* For dynamic credentials:
  + Enable database engine: vault secrets enable database
  + Enable mssql plugin: vault write database/config/my-mssql-database plugin\_name=mssql-database-plugin connection\_url='sqlserver://{{username}}:{{password}}@192.168.97.22:1433' allowed\_roles="my-role" username="sa" password="Test@123"
  + Create a role, which can generate db credentialas:

vault write database/roles/my-role db\_name=my-mssql-database creation\_statements="CREATE LOGIN [{{name}}] WITH PASSWORD = '{{password}}'; CREATE USER [{{name}}] FOR LOGIN [{{name}}]; GRANT SELECT ON SCHEMA::dbo TO [{{name}}];" default\_ttl="1h" max\_ttl="24h"

## Add a configuration provider to access Vault

In ASP.NET Core, you can add a [configuration provider](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-5.0) to retrieve configuration information outside of appsettings.json. The ASP.NET Core demo application uses a configuration provider for Vault to connect to Vault and retrieve secrets on application startup.

Open the CustomOptions/VaultConfiguration.cs file in your preferred text editor to view. The VaultConfigurationProvider constructor takes a set of options to connect to Vault using the approle auth method.

public VaultConfigurationProvider(VaultOptions config)

{

\_config = config;

var vaultClientSettings = new VaultClientSettings(

\_config.Address,

new AppRoleAuthMethodInfo(\_config.Role,

\_config.Secret)

);

\_client = new VaultClient(vaultClientSettings);

}

The options in VaultOptions include the Vault address, the role identifer, secret, mount path for the secrets (projects-api/), and secret type (secrets or database).

public class VaultOptions

{

public string Address { get; set; }

public string Role { get; set; }

public string Secret { get; set; }

public string MountPath { get; set; }

public string SecretType { get; set; }

}

If you do this in your own .NET application, you will need to reconfigure the Vault configuration to authenticate with the method communicated by your operations team.

The Vault configuration provider also overrides the Load method with a method to retrieve the database credentials from Vault based on the SecretType and store it into database:userID and database:password configuration.

public override void Load()

{

LoadAsync().Wait();

}

public async Task LoadAsync()

{

await GetDatabaseCredentials();

}

public async Task GetDatabaseCredentials()

{

var userID = "";

var password = "";

// TRUNCATED

Data.Add("database:userID", userID);

Data.Add("database:password", password);

}

If you configure this for your own application, you can update the GetDatabaseCredentials with a more generic method to retrieve the secrets you need from Vault. In the demo application, you can retrieve the static database password from secret/my-secret-password or dynamic database username and password from database.

Next, the VaultExtensions class creates a configuration builder called AddVault that creates the Vault client when you build the application.

public static class VaultExtensions

{

public static IConfigurationBuilder AddVault(this IConfigurationBuilder configuration,

Action<VaultOptions> options)

{

var vaultOptions = new VaultConfigurationSource(options);

configuration.Add(vaultOptions);

return configuration;

}

}

Open Program.cs. The demo application references the AddVault. If the Vault role is defined, the application will retrieve the Vault configuration from appsettings.json.

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Configuration.AddVault(options =>

{

var vaultOptions = builder.Configuration.GetSection("Vault");

options.Address = vaultOptions["Address"];

options.Role = vaultOptions["Role"];

options.MountPath = vaultOptions["MountPath"];

options.SecretType = vaultOptions["SecretType"];

options.Secret = vaultOptions["Secret\_Id"];

});

builder.Services.Configure<VaultOptions>(builder.Configuration.GetSection("Vault"));

var dbBuilder = new SqlConnectionStringBuilder(

builder.Configuration.GetConnectionString("Database")

);

## Run the application using a static database password

The demo application uses a static database password stored in Vault's key-value store to connect to the MSSQL database. You can use the key-value store to store API tokens and other static sensitive information. You manually manage static secrets, assuming responsibility for their rotation.

Open appsettings.json with your terminal. The Vault.SecretType defaults to secrets, which the application uses to access that statically defined database password.

$ appsettings.json

"Vault": {

"Address": "http://127.0.0.1:8200",

"Role": "f83c2996-8101-f208-41d8-90d344c2bb55",

"Secret\_Id": "ba1092a9-bfc4-6b20-ca09-ed95459c8a0c",

"MountPath": "secret/my-secret-password",

"SecretType": "secrets"}

Open ProjectApi/CustomOptions/VaultConfiguration.cs. The configuration provider receives the SecretType of secrets. If it receives that secret type, it will retrieve the database password you discovered in Vault at projects-api/secrets/static.

public async Task GetDatabaseCredentials()

{

var userID = "";

var password = "";

if (\_config.SecretType == "secrets")

{

Secret<SecretData> secret = \_client.V1.Secrets.KeyValue.V2.ReadSecretAsync(

path: "/my-secret-password",

mountPoint: "secret"

).Result;

userID = "sa";

password = secret.Data.Data["password"].ToString();

}

//TRUCATED

Data.Add("database:userID", userID);

Data.Add("database:password", password);

}

}

## Run the application using a dynamic database username and password

Your Vault administrator may provide you with dynamic database usernames and passwords instead, which allows Vault to issue a new set of credentials based on a time-to-live parameter. When a database username and password expires, you must reload the application to retrieve new credentials from Vault.

The demo application will use a dynamic database username and password managed by Vault's database secrets engine to connect to the MSSQL database. Vault can be configured with [secrets engines](https://www.vaultproject.io/docs/secrets) to manage the rotation of secrets.

Open the appsettings.json in your preferred text editor, and change the Vault.SecretType defaults to database instead of secrets.

The appsettings.json file should look as follow.

{

...TRUNCATED...

"Vault": {

"Address": "http://127.0.0.1:8200",

"Role": "f83c2996-8101-f208-41d8-90d344c2bb55",

"Secret\_Id": "ba1092a9-bfc4-6b20-ca09-ed95459c8a0c",

"MountPath": "secret/my-secret-password",

"SecretType": "database"

}}

Open CustomOptions/VaultConfiguration.cs. The configuration provider receives the SecretType of database. If it receives that secret type, it will retrieve the database username and password Vault generates at database/creds/projects-api-role.

public async Task GetDatabaseCredentials()

{

var userID = "";

var password = "";

//TRUNCATED

if (\_config.SecretType == "database")

{

Secret<UsernamePasswordCredentials> dynamicDatabaseCredentials =

await \_client.V1.Secrets.Database.GetCredentialsAsync(

\_config.Role);

userID = dynamicDatabaseCredentials.Data.Username;

password = dynamicDatabaseCredentials.Data.Password;

}

Data.Add("database:userID", userID);

Data.Add("database:password", password);

}

}

The demo application accesses the Vault endpoint at database/creds/projects-api-role to get a new database username and password. Vault has been configured to generate a new username and password that expire after two minutes. The expiration time of the secret can be updated to the time commensurate to your security policy for a database.

REFERENCES:

<https://www.vaultproject.io/docs/secrets/databases/mysql-maria>

<https://learn.hashicorp.com/tutorials/vault/dotnet-httpclient>

<https://learn.hashicorp.com/tutorials/vault/getting-started-dynamic-secrets>