<https://docs.microsoft.com/en-us/azure/key-vault/general/overview>

* **Secrets Management** - Azure Key Vault can be used to Securely store and tightly control access to tokens, passwords, certificates, API keys, and other secrets
* **Key Management** - Azure Key Vault can be used as a Key Management solution. Azure Key Vault makes it easy to create and control the encryption keys used to encrypt your data.
* **Certificate Management** - Azure Key Vault lets you easily provision, manage, and deploy public and private Transport Layer Security/Secure Sockets Layer (TLS/SSL) certificates for use with Azure and your internal connected resources.

***Authentication*** *is done via Azure Active Directory.*

***Authorization*** *may be done via Azure role-based access control (Azure* ***RBAC****)* ***or Key Vault access policy****. Azure RBAC can be used for both management of the vaults and access data stored in a vault, while key vault access policy can only be used when attempting to access data stored in a vault.*

Azure Key Vaults may be either software-protected or, with the Azure Key Vault Premium tier, hardware-protected by hardware security modules (HSMs)

* Replicating the contents of your Key Vault within a region and to a secondary region. Data replication ensures high availability and takes away the need of any action from the administrator to trigger the failover.
* **Azure Storage account keys**: Can manage keys of an Azure Storage account for you. Internally, Key Vault can list (sync) keys with an Azure Storage Account, and regenerate (rotate) the keys periodically. For more information, see [Manage storage account keys with Key Vault](https://docs.microsoft.com/en-us/azure/key-vault/secrets/overview-storage-keys).  
  <https://rakhesh.com/azure/using-azure-key-vault-to-manage-storage-account-keys/>

Objects stored in Key Vault are versioned whenever a new instance of an object is created. Each version is assigned a unique identifier and URL

# RBAC

<https://docs.microsoft.com/en-us/azure/key-vault/general/rbac-guide?tabs=azure-cli>

The Azure RBAC model provides the ability to set permissions on different scope levels: management group, subscription, resource group, or individual resources. Azure RBAC for key vault also provides the ability to have separate permissions on individual keys, secrets, and certificates

Our recommendation is to use a vault per application per environment (Development, Pre-Production, and Production).

Individual keys, secrets, and certificates permissions should be used only for specific scenarios:

There is a set of built on roles

Assignment can be done at subscription / RG/ key-vault level /secret level

az role assignment create --role "Key Vault Reader" --assignee {i.e user@microsoft.com} --scope /subscriptions/{subscriptionid}/resourcegroups/{resource-group-name}

az role assignment create --role "Key Vault Secrets Officer" --assignee {i.e jalichwa@microsoft.com} --scope /subscriptions/{subscriptionid}/resourcegroups/{resource-group-name}/providers/Microsoft.KeyVault/vaults/{key-vault-name}

az role assignment create --role "Key Vault Secrets Officer" --assignee {i.e user@microsoft.com} --scope /subscriptions/{subscriptionid}/resourcegroups/{resource-group-name}/providers/Microsoft.KeyVault/vaults/{key-vault-name}/secrets/RBACSecret

KEYS   
  
Cryptographic keys in Key Vault are represented as JSON Web Key [JWK] objects.

HSM-protected keys (also referred to as HSM-keys) are processed in an HSM (Hardware Security Module) and always remain HSM protection boundary.

HSM support also symmetric keys, software protected keys support asymmetric keys only

# Secrets

Key Vault stores and manages secrets as sequences of octets (8-bit bytes), with a maximum size of 25k bytes each

Key Vault also supports a contentType field for secrets. Clients may specify the content type of a secret to assist in interpreting the secret data when it's retrieved. The maximum length of this field is 255 characters.

All secrets in your Key Vault are stored encrypted. Key Vault encrypts secrets at rest with a hierarchy of encryption keys, with all keys in that hierarchy are protected by modules that are FIPS 140-2 compliant. This encryption is transparent, and requires no action from the user. The Azure Key Vault service encrypts your secrets when you add them, and decrypts them automatically when you read them.

In addition to the secret data, the following attributes may be specified: : exp, nbf,enabled

Read only : Created, updated

Access Control for secrets managed in Key Vault, is provided at the level of the Key Vault that contains those secrets.

The following permissions can be used, on a per-principal basis, in the secrets access control entry on a vault, and closely mirror the operations allowed on a secret object:

* Permissions for secret management operations
  + *get*: Read a secret
  + *list*: List the secrets or versions of a secret stored in a Key Vault
  + *set*: Create a secret
  + *delete*: Delete a secret
  + *recover*: Recover a deleted secret
  + *backup*: Back up a secret in a key vault
  + *restore*: Restore a backed up secret to a key vault
* Permissions for privileged operations
  + *purge*: Purge (permanently delete) a deleted secret

You can specify additional application-specific metadata in the form of tags.

* [Assign a Key Vault access policy using CLI](https://docs.microsoft.com/en-us/azure/key-vault/general/assign-access-policy-cli)
* [Assign a Key Vault access policy using PowerShell](https://docs.microsoft.com/en-us/azure/key-vault/general/assign-access-policy-powershell)
* [Assign a Key Vault access policy using the Azure portal](https://docs.microsoft.com/en-us/azure/key-vault/general/assign-access-policy-portal)

PORTAL

Enter the name of the user, app or service principal in the search field, select the appropriate result, then choose **Select**.

<https://docs.microsoft.com/en-us/azure/key-vault/general/authentication>

Authentication with Key Vault works in conjunction with [Azure Active Directory (Azure AD)](https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis), which is responsible for authenticating the identity of any given **security principal**.

* A **user** security principal identifies an individual who has a profile in Azure Active Directory.
* A **group** security principal identifies a set of users created in Azure Active Directory. Any roles or permissions assigned to the group are granted to all of the users within the group.
* A **service principal** is a type of security principal that identifies an application or service, which is to say, a piece of code rather than a user or group. A service principal's object ID is known as its **client ID** and acts like its username. The service principal's **client secret** acts like its password.

For applications, there are two ways to obtain a service principal:

* Recommended: enable a system-assigned **managed identity** for the application.

With managed identity, Azure internally manages the application's service principal and automatically authenticates the application with other Azure services. Managed identity is available for applications deployed to a variety of services.

For more information, see the [Managed identity overview](https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/overview). Also see [Azure services that support managed identity](https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/services-support-managed-identities), which links to articles that describe how to enable managed identity for specific services (such as App Service, Azure Functions, Virtual Machines, etc.).

* If you cannot use managed identity, you instead **register** the application with your Azure AD tenant, as described on [Quickstart: Register an application with the Azure identity platform](https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app). Registration also creates a second application object that identifies the app across all tenants.you then can use clientid/secret or clientid/certificate for autnetication

However, whatever is the kind of identity, that identity must get an access token and provide it to the AKV apis endpoint

By default, Key Vault allows access to resources through public IP addresses. For greater security, you can also restrict access to specific IP ranges, service endpoints, virtual networks, or private endpoints.

# Authentication to Key Vault in application code

Key Vault SDK is using Azure Identity client library, which allows seamless authentication to Key Vault across environments with same code

**Azure Identity client libraries**

| **AUTHENTICATION TO KEY VAULT IN APPLICATION CODE** | | | |
| --- | --- | --- | --- |
| **.NET** | **Python** | **Java** | **JavaScript** |
| [Azure Identity SDK .NET](https://docs.microsoft.com/en-us/dotnet/api/overview/azure/identity-readme) | [Azure Identity SDK Python](https://docs.microsoft.com/en-us/python/api/overview/azure/identity-readme) | [Azure Identity SDK Java](https://docs.microsoft.com/en-us/java/api/overview/azure/identity-readme) | [Azure Identity SDK JavaScript](https://docs.microsoft.com/en-us/javascript/api/overview/azure/identity-readme) |

DotNet Core (/net 5-6)

## Azure Key Vault configuration provider in ASP.NET Core

<https://docs.microsoft.com/en-us/aspnet/core/security/key-vault-configuration?view=aspnetcore-6.0#:~:text=Azure%20Key%20Vault%20is%20a,access%20to%20sensitive%20configuration%20data>.

It’s the typical way to interact with key vault. So that you don’t’ have to “touch” your existing code.

using System.Security.Cryptography.X509Certificates;

using Azure.Identity;

var builder = WebApplication.CreateBuilder(args);

if (builder.Environment.IsProduction()){

using var x509Store = new X509Store(StoreLocation.CurrentUser);

x509Store.Open(OpenFlags.ReadOnly);

var x509Certificate = x509Store.Certificates

.Find( X509FindType.FindByThumbprint,

builder.Configuration["AzureADCertThumbprint"],

validOnly: false)

.OfType<X509Certificate2>()

.Single();

builder.Configuration.AddAzureKeyVault(

new Uri($"https://{builder.Configuration["KeyVaultName"]}.vault.azure.net/"),

new ClientCertificateCredential(

builder.Configuration["AzureADDirectoryId"],

builder.Configuration["AzureADApplicationId"],

x509Certificate));

}

var app = builder.Build();

-----------------------------------------------------------------------------------------------------------

using Azure.Identity;

var builder = WebApplication.CreateBuilder(args);

if (builder.Environment.IsProduction()){

builder.Configuration.AddAzureKeyVault(

new Uri($"https://{builder.Configuration["KeyVaultName"]}.vault.azure.net/"),

new DefaultAzureCredential());

}

-----------------------------------------------------------------------------------

**For apps that use a user-assigned managed identity, configure the managed identity's Client ID using one of the following approaches:**

Set the AZURE\_CLIENT\_ID environment variable.

Set the DefaultAzureCredentialOptions.ManagedIdentityClientId property when calling AddAzureKeyVault:

C#

Copy

builder.Configuration.AddAzureKeyVault(

new Uri($"https://{builder.Configuration["KeyVaultName"]}.vault.azure.net/"),

new DefaultAzureCredential(new DefaultAzureCredentialOptions

{

ManagedIdentityClientId = builder.Configuration["AzureADManagedIdentityClientId"]

}));

AZURE APP CONFIGURATION

<https://docs.microsoft.com/en-us/azure/azure-app-configuration/overview>

Azure App Configuration provides a service to centrally manage application settings and feature flags

Settings are flat but you can leverage Key grouping and labels to then filter when getting data from you app

App or function access to App Configuration

You can provide access to App Configuration for Web Apps or Azure Functions by using any of the following methods:

Through the Azure portal, enter the connection string to your App Configuration store in the Application settings of App Service.

Store the connection string to your App Configuration store in Key Vault and reference it from App Service.

Use Azure managed identities to access the App Configuration store. For more information, see Integrate with Azure managed identities.

Push configuration from App Configuration to App Service. App Configuration provides an export function (in Azure portal and the Azure CLI) that sends data directly into App Service. With this method, you don't need to change the application code at all.

https://docs.microsoft.com/en-us/azure/azure-app-configuration/howto-best-practices

To check : refresh with sentinel key

<https://docs.microsoft.com/en-us/azure/azure-app-configuration/enable-dynamic-configuration-aspnet-core?tabs=core5x>

https://stackoverflow.com/questions/58253996/dynamically-update-net-core-config-from-azure-app-configuration

Prefix and similar

Labels

-------------

DEVOPS : upload json

mporting configuration data into App Configuration

App Configuration offers the option to bulk import your configuration settings from your current configuration files using either the Azure portal or CLI.

az appconfig kv import -s file --format json --path "~/Import.json" --content-type "application/json" --separator : --depth 2

---------

Feature management

<https://docs.microsoft.com/en-us/azure/azure-app-configuration/use-feature-flags-dotnet-core?tabs=core5x>

<https://docs.microsoft.com/en-us/azure/azure-app-configuration/howto-integrate-azure-managed-service-identity?tabs=core5x>

<https://docs.microsoft.com/en-us/azure/azure-app-configuration/howto-feature-filters-aspnet-core?WT.mc_id=Portal-Microsoft_Azure_AzConfig>

but for 100% control

<https://andrewlock.net/creating-a-custom-feature-filter-adding-feature-flags-to-an-asp-net-core-app-part-4/>

# MVC views

Open \_ViewImports.cshtml in the Views directory, and add the feature manager tag helper:

HTML

Copy

@addTagHelper \*, Microsoft.FeatureManagement.AspNetCore

In MVC views, you can use a <feature> tag to render content based on whether a feature flag is enabled:

HTML

Copy

<feature name="FeatureA">

<p>This can only be seen if 'FeatureA' is enabled.</p>

</feature>

To display alternate content when the requirements are not met the negate attribute can be used.

HTML

Copy

<feature name="FeatureA" negate="true">

<p>This will be shown if 'FeatureA' is disabled.</p>

</feature>

The feature <feature> tag can also be used to show content if any or all features in a list are enabled.

HTML

Copy

<feature name="FeatureA, FeatureB" requirement="All">

<p>This can only be seen if 'FeatureA' and 'FeatureB' are enabled.</p>

</feature>

<feature name="FeatureA, FeatureB" requirement="Any">

<p>This can be seen if 'FeatureA', 'FeatureB', or both are enabled.</p>

</feature>