

Contents

[Key Vault Documentation](#)

[Overview](#)

[About Key Vault](#)

[Quickstarts](#)

[CLI](#)

[PowerShell](#)

[Portal](#)

[Python](#)

[Java](#)

[Node.js](#)

[.NET \(SDK v4\)](#)

[.NET \(SDK v3\)](#)

[ARM template](#)

[Tutorials](#)

[Azure Web Apps](#)

[Azure Key Vault with Azure Web App in .NET](#)

[Azure Virtual Machine](#)

[Linux Virtual Machine](#)

[.NET](#)

[Python](#)

[Windows Virtual Machine](#)

[.NET](#)

[Python](#)

[Storage Account Keys](#)

[Manage storage account keys with Azure CLI](#)

[Manage storage account keys with PowerShell](#)

[Fetch shared access signature tokens in code](#)

[Azure Containers](#)

[Azure Resource Manager Template](#)

[Set up key rotation and auditing](#)

[Samples](#)

[Code samples](#)

[Concepts](#)

[Basic concepts](#)

[Security](#)

[Security overview](#)

[Security baseline](#)

[Security worlds](#)

[Secure your key vault](#)

[Soft-delete](#)

[Monitoring Key Vault events](#)

[Throttling](#)

[Virtual Network Service Endpoints](#)

[Authentication, requests and responses](#)

[Common parameters and headers](#)

[Certificates](#)

[About keys, secrets and certificates](#)

[Get started with certificates](#)

[Certificate creation methods](#)

[Monitor and manage certificate creation](#)

[How-to guides](#)

[Security](#)

[Recommendations](#)

[Authenticate](#)

[Use an App Service managed identity](#)

[Use an access control policy](#)

[Import HSM-protected keys](#)

[Overview](#)

[Import HSM-protected keys to Key Vault \(preview\)](#)

[Import HSM-protected keys to Key Vault \(legacy\)](#)

[Route key vault notifications with Event Grid](#)

[Receive notifications via Event Grid and Logic Apps](#)

[Rotate secrets for single-user resources](#)

[Integrate with Azure Policy](#)

[Integrate with Azure Private Link Service](#)

[Azure Key Vault logging](#)

[Access behind firewalls](#)

[Availability and redundancy](#)

[Change tenant ID](#)

[Create and manage using Azure CLI](#)

[Use Key Vault soft-delete with CLI](#)

[Use Key Vault soft-delete with Azure PowerShell](#)

[Configure firewalls and virtual networks](#)

[Java development](#)

[Use the Spring Boot Starter](#)

Reference

[Key Vault REST error codes](#)

[Azure PowerShell](#)

[Azure CLI](#)

[.NET](#)

[Java](#)

[Node.js](#)

[Python](#)

[REST](#)

[Best Practices](#)

[Resource Manager template](#)

[Authentication, requests and responses](#)

[Common parameters and headers](#)

Develop

[Developer's guide](#)

[Migrate to .NET 2.0](#)

[Save Web App secrets](#)

[Service authentication using .NET](#)

Use the Key Vault Connected Service

Resources

[Build your skills with Microsoft Learn](#)

[Authentication, requests and responses](#)

[Common parameters and headers](#)

[Customer Data](#)

[Pricing](#)

[Region availability](#)

[Service Level Agreement \(SLA\)](#)

[Service limits](#)

[Service status](#)

[Service updates](#)

[Blog](#)

[Product Feedback](#)

What is Azure Key Vault?

4 minutes to read • [Edit Online](#)

Azure Key Vault helps solve the following problems:

- **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 also 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 is also a service that 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.
- **Store secrets backed by Hardware Security Modules** - The secrets and keys can be protected either by software or FIPS 140-2 Level 2 validated HSMs

Why use Azure Key Vault?

Centralize application secrets

Centralizing storage of application secrets in Azure Key Vault allows you to control their distribution. Key Vault greatly reduces the chances that secrets may be accidentally leaked. When using Key Vault, application developers no longer need to store security information in their application. Not having to store security information in applications eliminates the need to make this information part of the code. For example, an application may need to connect to a database. Instead of storing the connection string in the app's code, you can store it securely in Key Vault.

Your applications can securely access the information they need by using URIs. These URIs allow the applications to retrieve specific versions of a secret. There is no need to write custom code to protect any of the secret information stored in Key Vault.

Securely store secrets and keys

Secrets and keys are safeguarded by Azure, using industry-standard algorithms, key lengths, and hardware security modules (HSMs). The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated.

Access to a key vault requires proper authentication and authorization before a caller (user or application) can get access. Authentication establishes the identity of the caller, while authorization determines the operations that they are allowed to perform.

Authentication is done via Azure Active Directory. Authorization may be done via role-based access control (RBAC) or Key Vault access policy. RBAC is used when dealing with the management of the vaults and key vault access policy is used when attempting to access data stored in a vault.

Azure Key Vaults may be either software- or hardware-HSM protected. For situations where you require added assurance you can import or generate keys in hardware security modules (HSMs) that never leave the HSM boundary. Microsoft uses nCipher hardware security modules. You can use nCipher tools to move a key from your HSM to Azure Key Vault.

Finally, Azure Key Vault is designed so that Microsoft does not see or extract your data.

Monitor access and use

Once you have created a couple of Key Vaults, you will want to monitor how and when your keys and secrets

are being accessed. You can monitor activity by enabling logging for your vaults. You can configure Azure Key Vault to:

- Archive to a storage account.
- Stream to an event hub.
- Send the logs to Azure Monitor logs.

You have control over your logs and you may secure them by restricting access and you may also delete logs that you no longer need.

Simplified administration of application secrets

When storing valuable data, you must take several steps. Security information must be secured, it must follow a life cycle, and it must be highly available. Azure Key Vault simplifies the process of meeting these requirements by:

- Removing the need for in-house knowledge of Hardware Security Modules.
- Scaling up on short notice to meet your organization's usage spikes.
- 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.
- Providing standard Azure administration options via the portal, Azure CLI and PowerShell.
- Automating certain tasks on certificates that you purchase from Public CAs, such as enrollment and renewal.

In addition, Azure Key Vaults allow you to segregate application secrets. Applications may access only the vault that they are allowed to access, and they can be limited to only perform specific operations. You can create an Azure Key Vault per application and restrict the secrets stored in a Key Vault to a specific application and team of developers.

Integrate with other Azure services

As a secure store in Azure, Key Vault has been used to simplify scenarios like:

- [Azure Disk Encryption](#)
- The [always encrypted](#) functionality in SQL server and Azure SQL Database
- [Azure App Service](#).

Key Vault itself can integrate with storage accounts, event hubs, and log analytics.

Next steps

- [Quickstart: Create an Azure Key Vault using the CLI](#)
- [Configure an Azure web application to read a secret from Key vault](#)

Quickstart: Set and retrieve a secret from Azure Key Vault using Azure CLI

3 minutes to read • [Edit Online](#)

In this quickstart, you create a key vault in Azure Key Vault with Azure CLI. Azure Key Vault is a cloud service that works as a secure secrets store. You can securely store keys, passwords, certificates, and other secrets. For more information on Key Vault you may review the [Overview](#). Azure CLI is used to create and manage Azure resources using commands or scripts. Once that you have completed that, you will store a secret.

If you don't have an Azure subscription, create a [free account](#) before you begin.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the CLI locally, this quickstart requires the Azure CLI version 2.0.4 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install the Azure CLI](#).

To sign in to Azure using the CLI you can type:

```
az login
```

For more information on login options via the CLI take a look at [sign in with Azure CLI](#)

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. The following example creates a resource group named *ContosoResourceGroup* in the *eastus* location.

```
az group create --name "ContosoResourceGroup" --location eastus
```

Create a Key Vault

Next you will create a Key Vault in the resource group created in the previous step. You will need to provide some information:

- For this quickstart we use **Contoso-vault2**. You must provide a unique name in your testing.
- Resource group name **ContosoResourceGroup**.
- The location **East US**.

```
az keyvault create --name "Contoso-Vault2" --resource-group "ContosoResourceGroup" --location eastus
```

The output of this cmdlet shows properties of the newly created Key Vault. Take note of the two properties listed below:

- **Vault Name:** In the example, this is **Contoso-Vault2**. You will use this name for other Key Vault commands.
- **Vault URI:** In the example, this is <https://contoso-vault2.vault.azure.net/>. Applications that use your vault through its REST API must use this URI.

At this point, your Azure account is the only one authorized to perform any operations on this new vault.

Add a secret to Key Vault

To add a secret to the vault, you just need to take a couple of additional steps. This password could be used by an application. The password will be called **ExamplePassword** and will store the value of **hVFkk965BuUv** in it.

Type the commands below to create a secret in Key Vault called **ExamplePassword** that will store the value **hVFkk965BuUv**:

```
az keyvault secret set --vault-name "Contoso-Vault2" --name "ExamplePassword" --value "hVFkk965BuUv"
```

You can now reference this password that you added to Azure Key Vault by using its URI. Use <https://Contoso-Vault2.vault.azure.net/secrets/ExamplePassword> to get the current version.

To view the value contained in the secret as plain text:

```
az keyvault secret show --name "ExamplePassword" --vault-name "Contoso-Vault2"
```

Now, you have created a Key Vault, stored a secret, and retrieved it.

Clean up resources

Other quickstarts and tutorials in this collection build upon this quickstart. If you plan to continue on to work with subsequent quickstarts and tutorials, you may wish to leave these resources in place. When no longer needed, you can use the [az group delete](#) command to remove the resource group, and all related resources. You can delete the resources as follows:

```
az group delete --name ContosoResourceGroup
```

Next steps

In this quickstart you created a Key Vault and stored a secret in it. To learn more about Key Vault and how to integrate it with your applications, continue on to the articles below.

- Read an [Overview of Azure Key Vault](#)
- See the reference for the [Azure CLI az keyvault commands](#)
- Learn about [keys, secrets, and certificates](#)
- Review [Azure Key Vault best practices](#)

Quickstart: Set and retrieve a secret from Azure Key Vault using PowerShell

3 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

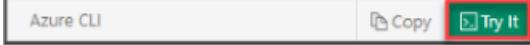
Azure Key Vault is a cloud service that works as a secure secrets store. You can securely store keys, passwords, certificates, and other secrets. For more information on Key Vault, you may review the [Overview](#). In this quickstart, you use PowerShell to create a key vault. You then store a secret in the newly created vault.

If you don't have an Azure subscription, create a [free account](#) before you begin.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use PowerShell locally, this tutorial requires Azure PowerShell module version 1.0.0 or later. Type `$PSVersionTable.PSVersion` to find the version. If you need to upgrade, see [Install Azure PowerShell](#)

module. If you are running PowerShell locally, you also need to run `Login-AzAccount` to create a connection with Azure.

```
Login-AzAccount
```

Create a resource group

Create an Azure resource group with `New-AzResourceGroup`. A resource group is a logical container into which Azure resources are deployed and managed.

```
New-AzResourceGroup -Name ContosoResourceGroup -Location EastUS
```

Create a Key Vault

Next you create a Key Vault. When doing this step, you need some information:

Although we use "Contoso KeyVault2" as the name for our Key Vault throughout this quickstart, you must use a unique name.

- **Vault name** Contoso-Vault2.
- **Resource group name** ContosoResourceGroup.
- **Location** East US.

```
New-AzKeyVault -Name 'Contoso-Vault2' -ResourceGroupName 'ContosoResourceGroup' -Location 'East US'
```

The output of this cmdlet shows properties of the newly created key vault. Take note of the two properties listed below:

- **Vault Name:** In the example that is **Contoso-Vault2**. You will use this name for other Key Vault cmdlets.
- **Vault URI:** In this example that is <https://Contoso-Vault2.vault.azure.net/>. Applications that use your vault through its REST API must use this URI.

After vault creation your Azure account is the only account allowed to do anything on this new vault.

```
Vault Name          : Contoso-Vault2
Resource Group Name: ContosoResourceGroup
Location           : East US
Resource ID         : /subscriptions/
                      /resourceGroups/ContosoResourceGr
Vault URI          : https://Contoso-Vault2.vault.azure.net
Tenant ID          :
SKU                : Standard
Enabled For Deployment? : False
Enabled For Template Deployment? : False
Enabled For Disk Encryption? : False
Access Policies    :
                      Tenant ID      :
                      Object ID      :
                      Application ID:
                      Display Name   :
                      Permissions to Keys: get, create, delete, list, update, import, backup,
                      restore
                      Permissions to Secrets: all
                      Permissions to Certificates: all
Tags               :
```

Adding a secret to Key Vault

To add a secret to the vault, you just need to take a couple of steps. In this case, you add a password that could be used by an application. The password is called **ExamplePassword** and stores the value of **hVFkk965BuUv** in it.

First convert the value of **hVFkk965BuUv** to a secure string by typing:

```
$secretvalue = ConvertTo-SecureString 'hVFkk965BuUv' -AsPlainText -Force
```

Then, type the PowerShell commands below to create a secret in Key Vault called **ExamplePassword** with the value **hVFkk965BuUv**:

```
$secret = Set-AzKeyVaultSecret -VaultName 'Contoso-Vault2' -Name 'ExamplePassword' -SecretValue $secretvalue
```

To view the value contained in the secret as plain text:

```
(Get-AzKeyVaultSecret -vaultName "Contoso-Vault2" -name "ExamplePassword").SecretValueText
```

Now, you have created a Key Vault, stored a secret, and retrieved it.

Clean up resources

Other quickstarts and tutorials in this collection build upon this quickstart. If you plan to continue on to work with other quickstarts and tutorials, you may want to leave these resources in place.

When no longer needed, you can use the [Remove-AzResourceGroup](#) command to remove the resource group, Key Vault, and all related resources.

```
Remove-AzResourceGroup -Name ContosoResourceGroup
```

Next steps

In this quickstart you created a Key Vault and stored a secret in it. To learn more about Key Vault and how to integrate it with your applications, continue on to the articles below.

- Read an [Overview of Azure Key Vault](#)
- See the reference for the [Azure PowerShell Key Vault cmdlets](#)
- Learn about [keys, secrets, and certificates](#)
- Review [Azure Key Vault best practices](#)

Quickstart: Set and retrieve a secret from Azure Key Vault using the Azure portal

2 minutes to read • [Edit Online](#)

Azure Key Vault is a cloud service that provides a secure store for secrets. You can securely store keys, passwords, certificates, and other secrets. Azure key vaults may be created and managed through the Azure portal. In this quickstart, you create a key vault, then use it to store a secret. For more information on Key Vault, review the [Overview](#).

If you don't have an Azure subscription, create a [free account](#) before you begin.

Sign in to Azure

Sign in to the Azure portal at <https://portal.azure.com>.

Create a vault

1. From the Azure portal menu, or from the **Home** page, select **Create a resource**.
2. In the Search box, enter **Key Vault**.
3. From the results list, choose **Key Vault**.
4. On the Key Vault section, choose **Create**.
5. On the **Create key vault** section provide the following information:
 - **Name:** A unique name is required. For this quickstart, we use **Contoso-vault2**.
 - **Subscription:** Choose a subscription.
 - Under **Resource Group**, choose **Create new** and enter a resource group name.
 - In the **Location** pull-down menu, choose a location.
 - Leave the other options to their defaults.
6. After providing the information above, select **Create**.

Take note of the two properties listed below:

- **Vault Name:** In the example, this is **Contoso-Vault2**. You will use this name for other steps.
- **Vault URI:** In the example, this is <https://contoso-vault2.vault.azure.net/>. Applications that use your vault through its REST API must use this URI.

At this point, your Azure account is the only one authorized to perform operations on this new vault.

Add a secret to Key Vault

To add a secret to the vault, you just need to take a couple of additional steps. In this case, we add a password that could be used by an application. The password is called **ExamplePassword** and we store the value of **hVFkk965BuUv** in it.

1. On the Key Vault properties pages, select **Secrets**.
2. Click on **Generate/Import**.
3. On the **Create a secret** screen choose the following values:
 - **Upload options:** Manual.
 - **Name:** ExamplePassword.
 - **Value:** hVFkk965BuUv
 - Leave the other values to their defaults. Click **Create**.

Once that you receive the message that the secret has been successfully created, you may click on it on the list. You can then see some of the properties. If you click on the current version, you can see the value you specified in the previous step.

The screenshot shows the Azure Key Vault portal interface. On the left, there's a list of secrets under 'ExamplePassword'. One secret is selected, showing its details in the right pane. The secret name is '72ce3977e6a3491rt1w649qa6l3e7342'. The properties pane shows it was created and updated on 3/7/2019, 10:27:21 PM. It has a 'Secret Identifier' pointing to the URL <https://contoso-vault2.vault.azure.net/secrets/ExamplePassword/72ce3977e6a3491rt1w649qa6l3e7342>. The 'Settings' section includes options to set activation and expiration dates, and an 'Enabled?' switch which is set to 'Yes'. Below that is a 'Tags' section with '0 tags'. The 'Secret' section contains a 'Content type (optional)' field, a 'Show Secret Value' button, and a redacted value '*****' followed by a copy icon.

By clicking "Show Secret Value" button in the right pane, you can see the hidden value.

Secret

Content type (optional)

[Hide Secret Value](#)

Clean up resources

Other Key Vault quickstarts and tutorials build upon this quickstart. If you plan to continue on to work with subsequent quickstarts and tutorials, you may wish to leave these resources in place. When no longer needed, delete the resource group, which deletes the Key Vault and related resources. To delete the resource group through the portal:

1. Enter the name of your resource group in the Search box at the top of the portal. When you see the resource group used in this quickstart in the search results, select it.
2. Select **Delete resource group**.
3. In the **TYPE THE RESOURCE GROUP NAME:** box type in the name of the resource group and select **Delete**.

Next steps

In this quickstart, you created a Key Vault and stored a secret in it. To learn more about Key Vault and how to integrate it with your applications, continue on to the articles below.

- Read an [Overview of Azure Key Vault](#)
- See the [Azure Key Vault developer's guide](#)
- Learn about [keys, secrets, and certificates](#)
- Review [Azure Key Vault best practices](#)

Quickstart: Azure Key Vault client library for Python

5 minutes to read • [Edit Online](#)

Get started with the Azure Key Vault client library for Python. Follow the steps below to install the package and try out example code for basic tasks.

Azure Key Vault helps safeguard cryptographic keys and secrets used by cloud applications and services. Use the Key Vault client library for Python to:

- Increase security and control over keys and passwords.
- Create and import encryption keys in minutes.
- Reduce latency with cloud scale and global redundancy.
- Simplify and automate tasks for TLS/SSL certificates.
- Use FIPS 140-2 Level 2 validated HSMs.

[API reference documentation](#) | [Library source code](#) | [Package \(Python Package Index\)](#)

Prerequisites

- An Azure subscription - [create one for free](#).
- Python 2.7, 3.5.3, or later
- [Azure CLI](#) or [Azure PowerShell](#)

This quickstart assumes you are running [Azure CLI](#) in a Linux terminal window.

Setting up

Install the package

From the console window, install the Azure Key Vault secrets library for Python.

```
pip install azure-keyvault-secrets
```

For this quickstart, you will need to install the `azure.identity` package as well:

```
pip install azure.identity
```

Create a resource group and key vault

This quickstart uses a pre-created Azure key vault. You can create a key vault by following the steps in the [Azure CLI quickstart](#), [Azure PowerShell quickstart](#), or [Azure portal quickstart](#). Alternatively, you can run the Azure CLI commands below.

IMPORTANT

Each key vault must have a unique name. Replace with the name of your key vault in the following examples.

```
az group create --name "myResourceGroup" -l "EastUS"  
az keyvault create --name <your-unique-keyvault-name> -g "myResourceGroup"
```

Create a service principal

The simplest way to authenticate a cloud-based .NET application is with a managed identity; see [Use an App Service managed identity to access Azure Key Vault](#) for details. For the sake of simplicity however, this quickstart creates a .NET console application. Authenticating a desktop application with Azure requires the use of a service principal and an access control policy.

Create a service principle using the Azure CLI [az ad sp create-for-rbac](#) command:

```
az ad sp create-for-rbac -n "http://mySP" --sdk-auth
```

This operation will return a series of key / value pairs.

```
{  
  "clientId": "7da18cae-779c-41fc-992e-0527854c6583",  
  "clientSecret": "b421b443-1669-4cd7-b5b1-394d5c945002",  
  "subscriptionId": "443e30da-feca-47c4-b68f-1636b75e16b3",  
  "tenantId": "35ad10f1-7799-4766-9acf-f2d946161b77",  
  "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",  
  "resourceManagerEndpointUrl": "https://management.azure.com/",  
  "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",  
  "galleryEndpointUrl": "https://gallery.azure.com/",  
  "managementEndpointUrl": "https://management.core.windows.net/"  
}
```

Take note of the clientId and clientSecret, as we will use them in the [Set environmental variable](#) step below.

Give the service principal access to your key vault

Create an access policy for your key vault that grants permission to your service principal by passing the clientId to the [az keyvault set-policy](#) command. Give the service principal get, list, and set permissions for both keys and secrets.

```
az keyvault set-policy -n <your-unique-keyvault-name> --spn <clientId-of-your-service-principal> --secret-permissions delete get list set --key-permissions create decrypt delete encrypt get list unwrapKey wrapKey
```

Set environmental variables

The DefaultAzureCredential method in our application relies on three environmental variables: `AZURE_CLIENT_ID`, `AZURE_CLIENT_SECRET`, and `AZURE_TENANT_ID`. Set these variables to the clientId, clientSecret, and tenantId values you noted in the [Create a service principal](#) step using the `export VARNAME=VALUE` format. (This method only sets the variables for your current shell and processes created from the shell; to permanently add these variables to your environment, edit your `/etc/environment` file.)

You will also need to save your key vault name as an environment variable called `KEY_VAULT_NAME`.

```
export AZURE_CLIENT_ID=<your-clientID>  
  
export AZURE_CLIENT_SECRET=<your-clientSecret>  
  
export AZURE_TENANT_ID=<your-tenantId>  
  
export KEY_VAULT_NAME=<your-key-vault-name>
```

Object model

The Azure Key Vault client library for Python allows you to manage keys and related assets such as certificates and secrets. The code samples below will show you how to create a client, set a secret, retrieve a secret, and delete a secret.

The entire console app is available at <https://github.com/Azure-Samples/key-vault-dotnet-core-quickstart/tree/master/key-vault-console-app>.

Code examples

Add directives

Add the following directives to the top of your code:

```
import os
from azure.keyvault.secrets import SecretClient
from azure.identity import DefaultAzureCredential
```

Authenticate and create a client

Authenticating to your key vault and creating a key vault client depends on the environmental variables in the [Set environmental variables](#) step above. The name of your key vault is expanded to the key vault URI, in the format "https://.vault.azure.net".

```
credential = DefaultAzureCredential()

client = SecretClient(vault_url=KVUri, credential=credential)
```

Save a secret

Now that your application is authenticated, you can put a secret into your keyvault using the `client.SetSecret` method](./dotnet/api/microsoft.azure.keyvault.keyvaultclientextensions.setsecretasync) This requires a name for the secret -- we're using "mySecret" in this sample.

```
client.set_secret(secretName, secretValue)
```

You can verify that the secret has been set with the `az keyvault secret show` command:

```
az keyvault secret show --vault-name <your-unique-keyvault-name> --name mySecret
```

Retrieve a secret

You can now retrieve the previously set value with the `client.GetSecret` method.

```
retrieved_secret = client.get_secret(secretName)
```

Your secret is now saved as `retrieved_secret.value`.

Delete a secret

Finally, let's delete the secret from your key vault with the `client.DeleteSecret` method.

```
client.delete_secret(secretName)
```

You can verify that the secret is gone with the `az keyvault secret show` command:

```
az keyvault secret show --vault-name <your-unique-keyvault-name> --name mySecret
```

Clean up resources

When no longer needed, you can use the Azure CLI or Azure PowerShell to remove your key vault and the corresponding resource group.

```
az group delete -g "myResourceGroup"
```

```
Remove-AzResourceGroup -Name "myResourceGroup"
```

Sample code

```
import os
import cmd
from azure.keyvault.secrets import SecretClient
from azure.identity import DefaultAzureCredential

keyVaultName = os.environ["KEY_VAULT_NAME"]
KVUri = "https://" + keyVaultName + ".vault.azure.net"

credential = DefaultAzureCredential()
client = SecretClient(vault_url=KVUri, credential=credential)

secretName = "mySecret"

print("Input the value of your secret > ")
secretValue = raw_input()

print("Creating a secret in " + keyVaultName + " called '" + secretName + "' with the value '" + secretValue + 
"`` ...")

client.set_secret(secretName, secretValue)

print(" done.")

print("Forgetting your secret.")
secretValue = ""
print("Your secret is '" + secretValue + "'.")

print("Retrieving your secret from " + keyVaultName + ".") 

retrieved_secret = client.get_secret(secretName)

print("Your secret is '" + retrieved_secret.value + "'.")
print("Deleting your secret from " + keyVaultName + " ...")

client.delete_secret(secretName)

print(" done.")
```

Next steps

In this quickstart you created a key vault, stored a secret, and retrieved that secret. To learn more about Key Vault and how to integrate it with your applications, continue on to the articles below.

- Read an [Overview of Azure Key Vault](#)
- See the [Azure Key Vault developer's guide](#)
- Learn about [keys, secrets, and certificates](#)
- Review [Azure Key Vault best practices](#)

Quickstart: Azure Key Vault client library for Java

6 minutes to read • [Edit Online](#)

Get started with the Azure Key Vault client library for Java. Follow the steps below to install the package and try out example code for basic tasks.

Azure Key Vault helps safeguard cryptographic keys and secrets used by cloud applications and services. Use the Key Vault client library for Java to:

- Increase security and control over keys and passwords.
- Create and import encryption keys in minutes.
- Reduce latency with cloud scale and global redundancy.
- Simplify and automate tasks for TLS/SSL certificates.
- Use FIPS 140-2 Level 2 validated HSMs.

[Source code](#) | [API reference documentation](#) | [Product documentation](#) | [Samples](#)

Prerequisites

- An Azure subscription - [create one for free](#).
- [Java Development Kit \(JDK\)](#) version 8 or above
- [Apache Maven](#)
- [Azure CLI](#) or [Azure PowerShell](#)

This quickstart assumes you are running [Azure CLI](#) and [Apache Maven](#) in a Linux terminal window.

Setting up

Create new Java console app

In a console window, use the `mvn` command to create a new Java console app with the name `akv-java`.

```
mvn archetype:generate -DgroupId=com.keyvault.quickstart  
                      -DartifactId=akv-java  
                      -DarchetypeArtifactId=maven-archetype-quickstart  
                      -DarchetypeVersion=1.4  
                      -DinteractiveMode=false
```

The output from generating the project will look something like this:

```
[INFO] -----
[INFO] Using following parameters for creating project from Archetype: maven-archetype-quickstart:1.4
[INFO] -----
[INFO] Parameter: groupId, Value: com.keyvault.quickstart
[INFO] Parameter: artifactId, Value: akv-java
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.keyvault.quickstart
[INFO] Parameter: packageInPathFormat, Value: com/keyvault/quickstart
[INFO] Parameter: package, Value: com.keyvault.quickstart
[INFO] Parameter: groupId, Value: com.keyvault.quickstart
[INFO] Parameter: artifactId, Value: akv-java
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Project created from Archetype in dir: /home/user/quickstarts/akv-java
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 38.124 s
[INFO] Finished at: 2019-11-15T13:19:06-08:00
[INFO] -----
```

Change your directory to the newly created akv-java/ folder.

```
cd akv-java
```

Install the package

Open the *pom.xml* file in your text editor. Add the following dependency elements to the group of dependencies.

```
<dependency>
  <groupId>com.azure</groupId>
  <artifactId>azure-security-keyvault-secrets</artifactId>
  <version>4.0.0</version>
</dependency>

<dependency>
  <groupId>com.azure</groupId>
  <artifactId>azure-identity</artifactId>
  <version>1.0.0</version>
</dependency>
```

Create a resource group and key vault

This quickstart uses a pre-created Azure key vault. You can create a key vault by following the steps in the [Azure CLI quickstart](#), [Azure PowerShell quickstart](#), or [Azure portal quickstart](#). Alternatively, you can run the Azure CLI commands below.

IMPORTANT

Each key vault must have a unique name. Replace with the name of your key vault in the following examples.

```
az group create --name "myResourceGroup" -l "EastUS"

az keyvault create --name <your-unique-keyvault-name> -g "myResourceGroup"
```

Create a service principal

The simplest way to authenticate a cloud-based application is with a managed identity; see [Use an App Service managed identity to access Azure Key Vault](#) for details. For the sake of simplicity however, this quickstart creates a desktop application, which requires the use of a service principal and an access control policy.

Create a service principle using the Azure CLI [az ad sp create-for-rbac](#) command:

```
az ad sp create-for-rbac -n "http://mySP" --sdk-auth
```

This operation will return a series of key / value pairs.

```
{
  "clientId": "7da18cae-779c-41fc-992e-0527854c6583",
  "clientSecret": "b421b443-1669-4cd7-b5b1-394d5c945002",
  "subscriptionId": "443e30da-feca-47c4-b68f-1636b75e16b3",
  "tenantId": "35ad10f1-7799-4766-9acf-f2d946161b77",
  "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",
  "resourceManagerEndpointUrl": "https://management.azure.com/",
  "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",
  "galleryEndpointUrl": "https://gallery.azure.com/",
  "managementEndpointUrl": "https://management.core.windows.net/"
}
```

Take note of the clientId, clientSecret, and tenantId, as we will use them in the next two steps.

Give the service principal access to your key vault

Create an access policy for your key vault that grants permission to your service principal by passing the clientId to the [az keyvault set-policy](#) command. Give the service principal get, list, and set permissions for both keys and secrets.

```
az keyvault set-policy -n <your-unique-keyvault-name> --spn <clientId-of-your-service-principal> --secret-permissions delete get list set --key-permissions create decrypt delete encrypt get list unwrapKey wrapKey
```

Set environmental variables

The DefaultAzureCredential method in our application relies on three environmental variables: `AZURE_CLIENT_ID`, `AZURE_CLIENT_SECRET`, and `AZURE_TENANT_ID`. Use set these variables to the clientId, clientSecret, and tenantId values you noted in the [Create a service principal](#) step, above. Use the `export VARNAME=VALUE` format to set your environmental variables. (This method only sets the variables for your current shell and processes created from the shell; to permanently add these variables to your environment, edit your `/etc/environment` file.)

You will also need to save your key vault name as an environment variable called `KEY_VAULT_NAME`.

```
export AZURE_CLIENT_ID=<your-clientID>
export AZURE_CLIENT_SECRET=<your-clientSecret>
export AZURE_TENANT_ID=<your-tenantId>
export KEY_VAULT_NAME=<your-key-vault-name>
```

Object model

The Azure Key Vault client library for Java allows you to manage keys and related assets such as certificates and secrets. The code samples below will show you how to create a client, set a secret, retrieve a secret, and delete a secret.

The entire console app is [below](#).

Code examples

Add directives

Add the following directives to the top of your code:

```
import com.azure.identity.DefaultAzureCredentialBuilder;  
  
import com.azure.security.keyvault.secrets.SecretClient;  
import com.azure.security.keyvault.secrets.SecretClientBuilder;  
import com.azure.security.keyvault.secrets.models.KeyVaultSecret;
```

Authenticate and create a client

Authenticating to your key vault and creating a key vault client depends on the environmental variables in the [Set environmental variables](#) step above. The name of your key vault is expanded to the key vault URI, in the format `https://<your-key-vault-name>.vault.azure.net`.

```
String keyVaultName = System.getenv("KEY_VAULT_NAME");  
String kvUri = "https://" + keyVaultName + ".vault.azure.net";  
  
SecretClient secretClient = new SecretClientBuilder()  
    .vaultUrl(kvUri)  
    .credential(new DefaultAzureCredentialBuilder().build())  
    .buildClient();
```

Save a secret

Now that your application is authenticated, you can put a secret into your keyvault using the `secretClient.setSecret` method. This requires a name for the secret -- we've assigned the value "mySecret" to the `secretName` variable in this sample.

```
secretClient.setSecret(new KeyVaultSecret(secretName, secretValue));
```

You can verify that the secret has been set with the [az keyvault secret show](#) command:

```
az keyvault secret show --vault-name <your-unique-keyvault-name> --name mySecret
```

Retrieve a secret

You can now retrieve the previously set value with the `secretClient.getSecret` method.

```
KeyVaultSecret retrievedSecret = secretClient.getSecret(secretName);
```

You can now access the value of the retrieved secret with `retrievedSecret.getValue()`.

Delete a secret

Finally, let's delete the secret from your key vault with the `secretClient.beginDeleteSecret` method.

```
secretClient.beginDeleteSecret(secretName);
```

You can verify that the secret is gone with the [az keyvault secret show](#) command:

```
az keyvault secret show --vault-name <your-unique-keyvault-name> --name mySecret
```

Clean up resources

When no longer needed, you can use the Azure CLI or Azure PowerShell to remove your key vault and the corresponding resource group.

```
az group delete -g "myResourceGroup"
```

```
Remove-AzResourceGroup -Name "myResourceGroup"
```

Sample code

```

package com.keyvault.quickstart;

import java.io.Console;

import com.azure.identity.DefaultAzureCredentialBuilder;

import com.azure.security.keyvault.secrets.SecretClient;
import com.azure.security.keyvault.secrets.SecretClientBuilder;
import com.azure.security.keyvault.secrets.models.KeyVaultSecret;

public class App {

    public static void main(String[] args) throws InterruptedException, IllegalArgumentException {

        String keyVaultName = System.getenv("KEY_VAULT_NAME");
        String kvUri = "https://" + keyVaultName + ".vault.azure.net";

        System.out.printf("key vault name = %s and kv uri = %s \n", keyVaultName, kvUri);

        SecretClient secretClient = new SecretClientBuilder()
            .vaultUrl(kvUri)
            .credential(new DefaultAzureCredentialBuilder().build())
            .buildClient();

        Console con = System.console();

        String secretName = "mySecret";

        System.out.println("Input the value of your secret > ");
        String secretValue = con.readLine();

        System.out.print("Creating a secret in " + keyVaultName + " called '" + secretName + "' with the value "
            + secretValue + "` ... `");

        secretClient.setSecret(new KeyVaultSecret(secretName, secretValue));

        System.out.println("done.");

        System.out.println("Forgetting your secret.");
        secretValue = "";
        System.out.println("Your secret is " + secretValue + "...");

        System.out.println("Retrieving your secret from " + keyVaultName + ".");
        KeyVaultSecret retrievedSecret = secretClient.getSecret(secretName);

        System.out.println("Your secret is " + retrievedSecret.getValue() + "...");
        System.out.print("Deleting your secret from " + keyVaultName + " ... ");

        secretClient.beginDeleteSecret(secretName);

        System.out.println("done.");

    }
}

```

Next steps

In this quickstart you created a key vault, stored a secret, and retrieved that secret. To learn more about Key Vault and how to integrate it with your applications, continue on to the articles below.

- Read an [Overview of Azure Key Vault](#)

- See the [Azure Key Vault developer's guide](#)
- Learn about [keys, secrets, and certificates](#)
- Review [Azure Key Vault best practices](#)

Quickstart: Azure Key Vault client library for Node.js (v4)

5 minutes to read • [Edit Online](#)

Get started with the Azure Key Vault client library for Node.js. Follow the steps below to install the package and try out example code for basic tasks.

Azure Key Vault helps safeguard cryptographic keys and secrets used by cloud applications and services. Use the Key Vault client library for Node.js to:

- Increase security and control over keys and passwords.
- Create and import encryption keys in minutes.
- Reduce latency with cloud scale and global redundancy.
- Simplify and automate tasks for TLS/SSL certificates.
- Use FIPS 140-2 Level 2 validated HSMs.

[API reference documentation](#) | [Library source code](#) | [Package \(npm\)](#)

Prerequisites

- An Azure subscription - [create one for free](#).
- Current [Node.js](#) for your operating system.
- [Azure CLI](#) or [Azure PowerShell](#)

This quickstart assumes you are running [Azure CLI](#) in a Linux terminal window.

Setting up

Install the package

From the console window, install the Azure Key Vault secrets library for Node.js.

```
npm install @azure/keyvault-secrets
```

For this quickstart, you will need to install the `azure.identity` package as well:

```
npm install @azure/identity
```

Create a resource group and key vault

This quickstart uses a pre-created Azure key vault. You can create a key vault by following the steps in the [Azure CLI quickstart](#), [Azure PowerShell quickstart](#), or [Azure portal quickstart](#). Alternatively, you can simply run the Azure CLI commands below.

IMPORTANT

Each key vault must have a unique name. Replace with the name of your key vault in the following examples.

```
az group create --name "myResourceGroup" -l "EastUS"  
az keyvault create --name <your-unique-keyvault-name> -g "myResourceGroup"
```

Create a service principal

The simplest way to authenticate an cloud-based application is with a managed identity; see [Use an App Service managed identity to access Azure Key Vault](#) for details. For the sake of simplicity however, this quickstarts creates a console application. Authenticating a desktop application with Azure requires the use of a service principal and an access control policy.

Create a service principle using the Azure CLI [az ad sp create-for-rbac](#) command:

```
az ad sp create-for-rbac -n "http://mySP" --sdk-auth
```

This operation will return a series of key / value pairs.

```
{  
  "clientId": "7da18cae-779c-41fc-992e-0527854c6583",  
  "clientSecret": "b421b443-1669-4cd7-b5b1-394d5c945002",  
  "subscriptionId": "443e30da-feca-47c4-b68f-1636b75e16b3",  
  "tenantId": "35ad10f1-7799-4766-9acf-f2d946161b77",  
  "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",  
  "resourceManagerEndpointUrl": "https://management.azure.com/",  
  "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",  
  "galleryEndpointUrl": "https://gallery.azure.com/",  
  "managementEndpointUrl": "https://management.core.windows.net/"  
}
```

Take note of the clientId and clientSecret, as we will use them in the [Set environmental variable](#) step below.

Give the service principal access to your key vault

Create an access policy for your key vault that grants permission to your service principal by passing the clientId to the [az keyvault set-policy](#) command. Give the service principal get, list, and set permissions for both keys and secrets.

```
az keyvault set-policy -n <your-unique-keyvault-name> --spn <clientId-of-your-service-principal> --secret-permissions delete get list set --key-permissions create decrypt delete encrypt get list unwrapKey wrapKey
```

Set environmental variables

The DefaultAzureCredential method in our application relies on three environmental variables: `AZURE_CLIENT_ID`, `AZURE_CLIENT_SECRET`, and `AZURE_TENANT_ID`. Set these variables to the clientId, clientSecret, and tenantId values you noted in the [Create a service principal](#) step using the `export VARNAME=VALUE` format. (This only sets the variables for your current shell and processes created from the shell; to permanently add these variables to your environment, edit your `/etc/environment` file.)

You will also need to save your key vault name as an environment variable called `KEY_VAULT_NAME`.

```
export AZURE_CLIENT_ID=<your-clientID>  
  
export AZURE_CLIENT_SECRET=<your-clientSecret>  
  
export AZURE_TENANT_ID=<your-tenantId>  
  
export KEY_VAULT_NAME=<your-key-vault-name>
```

Object model

The Azure Key Vault client library for Node.js allows you to manage keys and related assets such as certificates and secrets. The code samples below will show you how to create a client, set a secret, retrieve a secret, and delete a secret.

The entire console app is available at <https://github.com/Azure-Samples/key-vault-dotnet-core-quickstart/tree/master/key-vault-console-app>.

Code examples

Add directives

Add the following directives to the top of your code:

```
const { DefaultAzureCredential } = require("@azure/identity");
const { SecretClient } = require("@azure/keyvault-secrets");
```

Authenticate and create a client

Authenticating to your key vault and creating a key vault client depends on the environmental variables from the [Set environmental variables](#) step above, and the [SecretClient constructor](#).

The name of your key vault is expanded to the key vault URI, in the format

```
https://<your-key-vault-name>.vault.azure.net .
```

```
const keyVaultName = process.env["KEY_VAULT_NAME"];
const KVUri = "https://" + keyVaultName + ".vault.azure.net";

const credential = new DefaultAzureCredential();
const client = new SecretClient(KVUri, credential);
```

Save a secret

Now that your application is authenticated, you can put a secret into your keyvault using the [client.setSecret method](#). This requires a name for the secret -- we're using "mySecret" in this sample.

```
await client.setSecret(secretName, secretValue);
```

You can verify that the secret has been set with the [az keyvault secret show](#) command:

```
az keyvault secret show --vault-name <your-unique-keyvault-name> --name mySecret
```

Retrieve a secret

You can now retrieve the previously set value with the [client.getSecret method](#).

```
const retrievedSecret = await client.getSecret(secretName);
```

Your secret is now saved as `retrievedSecret.value`.

Delete a secret

Finally, let's delete the secret from your key vault with the [client.beginDeleteSecret method](#).

```
await client.beginDeleteSecret(secretName)
```

You can verify that the secret is gone with the [az keyvault secret show](#) command:

```
az keyvault secret show --vault-name <your-unique-keyvault-name> --name mySecret
```

Clean up resources

When no longer needed, you can use the Azure CLI or Azure PowerShell to remove your key vault and the corresponding resource group.

```
az group delete -g "myResourceGroup"
```

```
Remove-AzResourceGroup -Name "myResourceGroup"
```

Sample code

```

const { DefaultAzureCredential } = require("@azure/identity");
const { SecretClient } = require("@azure/keyvault-secrets");

const readline = require('readline');

function askQuestion(query) {
    const rl = readline.createInterface({
        input: process.stdin,
        output: process.stdout,
    });

    return new Promise(resolve => rl.question(query, ans => {
        rl.close();
        resolve(ans);
    }))
}

async function main() {

    const keyVaultName = process.env["KEY_VAULT_NAME"];
    const KVUri = "https://" + keyVaultName + ".vault.azure.net";

    const credential = new DefaultAzureCredential();
    const client = new SecretClient(KVUri, credential);

    const secretName = "mySecret";
    var secretValue = await askQuestion("Input the value of your secret > ");

    console.log("Creating a secret in " + keyVaultName + " called '" + secretName + "' with the value '" + secretValue + "` ...");
    await client.setSecret(secretName, secretValue);

    console.log("Done.");

    console.log("Forgetting your secret.");
    secretValue = "";
    console.log("Your secret is '" + secretValue + "'.");

    console.log("Retrieving your secret from " + keyVaultName + ".");

    const retrievedSecret = await client.getSecret(secretName);

    console.log("Your secret is '" + retrievedSecret.value + "'.");
    console.log("Deleting your secret from " + keyVaultName + " ...");

    await client.beginDeleteSecret(secretName);

    console.log("Done.");
}

main()

```

Next steps

In this quickstart you created a key vault, stored a secret, and retrieved that secret. To learn more about Key Vault and how to integrate it with your applications, continue on to the articles below.

- Read an [Overview of Azure Key Vault](#)
- See the [Azure Key Vault developer's guide](#)
- Learn about [keys, secrets, and certificates](#)
- Review [Azure Key Vault best practices](#)

Quickstart: Azure Key Vault client library for .NET (SDK v4)

6 minutes to read • [Edit Online](#)

Get started with the Azure Key Vault client library for .NET. Follow the steps below to install the package and try out example code for basic tasks.

Azure Key Vault helps safeguard cryptographic keys and secrets used by cloud applications and services. Use the Key Vault client library for .NET to:

- Increase security and control over keys and passwords.
- Create and import encryption keys in minutes.
- Reduce latency with cloud scale and global redundancy.
- Simplify and automate tasks for TLS/SSL certificates.
- Use FIPS 140-2 Level 2 validated HSMs.

[API reference documentation](#) | [Library source code](#) | [Package \(NuGet\)](#)

Prerequisites

- An Azure subscription - [create one for free](#).
- The [.NET Core 2.1 SDK or later](#).
- [Azure CLI](#) or [Azure PowerShell](#)

This quickstart assumes you are running `dotnet`, [Azure CLI](#), and Windows commands in a Windows terminal (such as [PowerShell Core](#), [Windows PowerShell](#), or the [Azure Cloud Shell](#)).

Setting up

Create new .NET console app

In a console window, use the `dotnet new` command to create a new .NET console app with the name `key-vault-console-app`.

```
dotnet new console -n key-vault-console-app
```

Change your directory to the newly created app folder. You can build the application with:

```
dotnet build
```

The build output should contain no warnings or errors.

```
Build succeeded.  
0 Warning(s)  
0 Error(s)
```

Install the package

From the console window, install the Azure Key Vault client library for .NET:

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

For this quickstart, you will need to install the following packages as well:

```
dotnet add package Azure.Identity
```

Create a resource group and key vault

This quickstart uses a pre-created Azure key vault. You can create a key vault by following the steps in the [Azure CLI quickstart](#), [Azure PowerShell quickstart](#), or [Azure portal quickstart](#). Alternatively, you can simply run the Azure CLI commands below.

IMPORTANT

Each key vault must have a unique name. Replace with the name of your key vault in the following examples.

```
az group create --name "myResourceGroup" -l "EastUS"  
az keyvault create --name <your-unique-keyvault-name> -g "myResourceGroup"
```

```
New-AzResourceGroup -Name myResourceGroup -Location EastUS  
New-AzKeyVault -Name <your-unique-keyvault-name> -ResourceGroupName myResourceGroup -Location EastUS
```

Create a service principal

The simplest way to authenticate a cloud-based .NET application is with a managed identity; see [Use an App Service managed identity to access Azure Key Vault](#) for details. For the sake of simplicity however, this quickstart creates a .NET console application. Authenticating a desktop application with Azure requires the use of a service principal and an access control policy.

Create a service principle using the Azure CLI [az ad sp create-for-rbac](#) command:

```
az ad sp create-for-rbac -n "http://mySP" --sdk-auth
```

This operation will return a series of key / value pairs.

```
{  
  "clientId": "7da18cae-779c-41fc-992e-0527854c6583",  
  "clientSecret": "b421b443-1669-4cd7-b5b1-394d5c945002",  
  "subscriptionId": "443e30da-feca-47c4-b68f-1636b75e16b3",  
  "tenantId": "35ad10f1-7799-4766-9acf-f2d946161b77",  
  "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",  
  "resourceManagerEndpointUrl": "https://management.azure.com/",  
  "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",  
  "galleryEndpointUrl": "https://gallery.azure.com/",  
  "managementEndpointUrl": "https://management.core.windows.net/"  
}
```

Create a service principal using Azure PowerShell [New-AzADServicePrincipal](#) command:

```

# Create a new service principal
$spn = New-AzADServicePrincipal -DisplayName "http://mySP"

# Get the tenant ID and subscription ID of the service principal
$tenantId = (Get-AzContext).Tenant.Id
$subscriptionId = (Get-AzContext).Subscription.Id

# Get the client ID
$clientId = $spn.ApplicationId

# Get the client Secret
$bstr = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($spn.Secret)
$clientSecret = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr)

```

For more details about the service principal with Azure PowerShell, refer to [Create an Azure service principal with Azure PowerShell](#).

Take note of the clientId, clientSecret, and tenantId, as we will use them in the following steps.

Give the service principal access to your key vault

Create an access policy for your key vault that grants permission to your service principal by passing the clientId to the [az keyvault set-policy](#) command. Give the service principal get, list, and set permissions for both keys and secrets.

```
az keyvault set-policy -n <your-unique-keyvault-name> --spn <clientId-of-your-service-principal> --secret-permissions list get set delete purge
```

```
Set-AzKeyVaultAccessPolicy -VaultName <your-unique-keyvault-name> -ServicePrincipalName <clientId-of-your-service-principal> -PermissionsToSecrets list,get,set,delete,purge
```

Set environmental variables

The DefaultAzureCredential method in our application relies on three environmental variables: `AZURE_CLIENT_ID`, `AZURE_CLIENT_SECRET`, and `AZURE_TENANT_ID`. Use set these variables to the clientId, clientSecret, and tenantId values you noted in the [Create a service principal](#) step, above.

You will also need to save your key vault name as an environment variable called `KEY_VAULT_NAME`:

```

setx AZURE_CLIENT_ID <your-clientID>

setx AZURE_CLIENT_SECRET <your-clientSecret>

setx AZURE_TENANT_ID <your-tenantId>

setx KEY_VAULT_NAME <your-key-vault-name>

```

Each time you call `setx`, you should get a response of "SUCCESS: Specified value was saved."

```

AZURE_CLIENT_ID=<your-clientID>

AZURE_CLIENT_SECRET=<your-clientSecret>

AZURE_TENANT_ID=<your-tenantId>

KEY_VAULT_NAME=<your-key-vault-name>

```

Object model

The Azure Key Vault client library for .NET allows you to manage keys and related assets such as certificates and secrets. The code samples below will show you how to create a client, set a secret, retrieve a secret, and delete a secret.

The entire console app is available at <https://github.com/Azure-Samples/key-vault-dotnet-core-quickstart/tree/master/key-vault-console-app>.

Code examples

Add directives

Add the following directives to the top of your code:

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

Authenticate and create a client

Authenticating to your key vault and creating a key vault client depends on the environmental variables in the [Set environmental variables](#) step above. The name of your key vault is expanded to the key vault URI, in the format "https://<your-key-vault-name>.vault.azure.net".

```
string keyVaultName = Environment.GetEnvironmentVariable("KEY_VAULT_NAME");
var kvUri = "https://" + keyVaultName + ".vault.azure.net";

var client = new SecretClient(new Uri(kvUri), new DefaultAzureCredential());
```

Save a secret

Now that your application is authenticated, you can put a secret into your keyvault using the [client.SetSecret method](#). This requires a name for the secret -- we're using "mySecret" in this sample.

```
client.SetSecret(secretName, secretValue);
```

You can verify that the secret has been set with the [az keyvault secret show](#) command:

```
az keyvault secret show --vault-name <your-unique-keyvault-name> --name mySecret
```

```
(Get-AzKeyVaultSecret -VaultName <your-unique-keyvault-name> -Name mySecret).SecretValueText
```

Retrieve a secret

You can now retrieve the previously set value with the [client.GetSecret method](#).

```
KeyVaultSecret secret = client.GetSecret(secretName);
```

Your secret is now saved as `secret.Value`.

Delete a secret

Finally, let's delete the secret from your key vault with the [client.DeleteSecret method](#).

```
client.StartDeleteSecret(secretName);
```

You can verify that the secret is gone with the [az keyvault secret show](#) command:

```
az keyvault secret show --vault-name <your-unique-keyvault-name> --name mySecret
```

```
(Get-AzKeyVaultSecret -VaultName <your-unique-keyvault-name> -Name mySecret).SecretValueText
```

Clean up resources

When no longer needed, you can use the Azure CLI or Azure PowerShell to remove your key vault and the corresponding resource group.

```
az group delete -g "myResourceGroup"
```

```
Remove-AzResourceGroup -Name "myResourceGroup"
```

Sample code

```

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

namespace key_vault_console_app
{
    class Program
    {
        static void Main(string[] args)
        {
            string secretName = "mySecret";

            string keyVaultName = Environment.GetEnvironmentVariable("KEY_VAULT_NAME");
            var kvUri = "https://" + keyVaultName + ".vault.azure.net";

            var client = new SecretClient(new Uri(kvUri), new DefaultAzureCredential());

            Console.Write("Input the value of your secret > ");
            string secretValue = Console.ReadLine();

            Console.WriteLine("Creating a secret in " + keyVaultName + " called '" + secretName + "' with the
value '" + secretValue + "` ...");

            client.SetSecret(secretName, secretValue);

            Console.WriteLine(" done.");

            Console.WriteLine("Forgetting your secret.");
            secretValue = "";
            Console.WriteLine("Your secret is '" + secretValue + "'.");

            Console.WriteLine("Retrieving your secret from " + keyVaultName + ".");
            KeyVaultSecret secret = client.GetSecret(secretName);

            Console.WriteLine("Your secret is '" + secret.Value + "'.");

            Console.WriteLine("Deleting your secret from " + keyVaultName + " ...");

            client.StartDeleteSecret(secretName);

            System.Threading.Thread.Sleep(5000);
            Console.WriteLine(" done.");
        }
    }
}

```

Next steps

In this quickstart you created a key vault, stored a secret, and retrieved that secret. See the [entire console app in GitHub](#).

To learn more about Key Vault and how to integrate it with your applications, continue on to the articles below.

- Implement [Service-to-service authentication to Azure Key Vault using .NET](#)
- Read an [Overview of Azure Key Vault](#)
- See the [Azure Key Vault developer's guide](#)
- Learn about [keys, secrets, and certificates](#)
- Review [Azure Key Vault best practices](#)

Quickstart: Azure Key Vault client library for .NET (SDK v3)

5 minutes to read • [Edit Online](#)

Get started with the Azure Key Vault client library for .NET. Follow the steps below to install the package and try out example code for basic tasks.

NOTE

This quickstart uses the v3.0.4 version of the Microsoft.Azure.KeyVault client library. To use the most up-to-date version of the Key Vault client library, see [Azure Key Vault client library for .NET \(SDK v4\)](#).

Azure Key Vault helps safeguard cryptographic keys and secrets used by cloud applications and services. Use the Key Vault client library for .NET to:

- Increase security and control over keys and passwords.
- Create and import encryption keys in minutes.
- Reduce latency with cloud scale and global redundancy.
- Simplify and automate tasks for TLS/SSL certificates.
- Use FIPS 140-2 Level 2 validated HSMs.

[API reference documentation](#) | [Library source code](#) | [Package \(NuGet\)](#)

NOTE

Each key vault must have a unique name. Replace with the name of your key vault in the following examples.

Prerequisites

- An Azure subscription - [create one for free](#).
- The [.NET Core 2.1 SDK or later](#).
- [Azure CLI](#) or [Azure PowerShell](#)

This quickstart assumes you are running `dotnet`, [Azure CLI](#), and Windows commands in a Windows terminal (such as [PowerShell Core](#), [Windows PowerShell](#), or the [Azure Cloud Shell](#)).

Setting up

Create new .NET console app

In a console window, use the `dotnet new` command to create a new .NET console app with the name `akv-dotnet`.

```
dotnet new console -n akvdotnet
```

Change your directory to the newly created app folder. You can build the application with:

```
dotnet build
```

The build output should contain no warnings or errors.

```
Build succeeded.  
0 Warning(s)  
0 Error(s)
```

Install the package

From the console window, install the Azure Key Vault client library for .NET:

```
dotnet add package Microsoft.Azure.KeyVault
```

For this quickstart, you will need to install the following packages as well:

```
dotnet add package System.Threading.Tasks  
dotnet add package Microsoft.IdentityModel.Clients.ActiveDirectory  
dotnet add package Microsoft.Azure.Management.ResourceManager.Fluent
```

Create a resource group and key vault

This quickstart uses a pre-created Azure key vault. You can create a key vault by following the steps in the [Azure CLI quickstart](#), [Azure PowerShell quickstart](#), or [Azure portal quickstart](#). Alternatively, you can simply run the Azure CLI commands below.

IMPORTANT

Each key vault must have a unique name. Replace with the name of your key vault in the following examples.

```
az group create --name "myResourceGroup" -l "EastUS"  
  
az keyvault create --name <your-unique-keyvault-name> -g "myResourceGroup"
```

Create a service principal

The simplest way to authenticate a cloud-based .NET application is with a managed identity; see [Use an App Service managed identity to access Azure Key Vault](#) for details. For the sake of simplicity however, this quickstart creates a .NET console application. Authenticating a desktop application with Azure requires the use of a service principal and an access control policy.

Create a service principle using the Azure CLI `az ad sp create-for-rbac` command:

```
az ad sp create-for-rbac -n "http://mySP" --sdk-auth
```

This operation will return a series of key / value pairs.

```
{  
    "clientId": "7da18cae-779c-41fc-992e-0527854c6583",  
    "clientSecret": "b421b443-1669-4cd7-b5b1-394d5c945002",  
    "subscriptionId": "443e30da-feca-47c4-b68f-1636b75e16b3",  
    "tenantId": "35ad10f1-7799-4766-9acf-f2d946161b77",  
    "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",  
    "resourceManagerEndpointUrl": "https://management.azure.com/",  
    "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",  
    "galleryEndpointUrl": "https://gallery.azure.com/",  
    "managementEndpointUrl": "https://management.core.windows.net/"  
}
```

Take note of the clientId and clientSecret, as we will use them in the [Authenticate to your key vault](#) step below.

Give the service principal access to your key vault

Create an access policy for your key vault that grants permission to your service principal by passing the clientId to the [az keyvault set-policy](#) command. Give the service principal get, list, and set permissions for both keys and secrets.

```
az keyvault set-policy -n <your-unique-keyvault-name> --spn <clientId-of-your-service-principal> --secret-permissions delete get list set --key-permissions create decrypt delete encrypt get list unwrapKey wrapKey
```

Object model

The Azure Key Vault client library for .NET allows you to manage keys and related assets such as certificates and secrets. The code samples below will show you how to set a secret and retrieve a secret.

The entire console app is available at <https://github.com/Azure-Samples/key-vault-dotnet-core-quickstart/tree/master/akvdotnet>.

Code examples

Add directives

Add the following directives to the top of your code:

```
using System;  
using System.Threading.Tasks;  
using Microsoft.Azure.KeyVault;  
using Microsoft.IdentityModel.Clients.ActiveDirectory;  
using Microsoft.Azure.Management.ResourceManager.Fluent;  
using Microsoft.Azure.Management.ResourceManager.Authentication;
```

Authenticate to your key vault

This .NET quickstart relies on environment variables to store credentials that should not be put in code.

Before you build and run your app, use the `setx` command to set the `akvClientId`, `akvClientSecret`, `akvTenantId`, and `akvSubscriptionId` environment variables to the values you noted above.

Windows

```
setx akvClientId "<your-clientID>"  
setx akvClientSecret "<your-clientSecret>"
```

Linux

```
export akvClientId = "<your-clientID>"  
export akvClientSecret = "<your-clientSecret>"
```

MacOS

```
export akvClientId = "<your-clientID>"  
export akvClientSecret = "<your-clientSecret>"
```

Assign these environment variables to strings in your code, and then authenticate your application by passing them to the [KeyVaultClient class](#):

```
string clientId = Environment.GetEnvironmentVariable("akvClientId");  
string clientSecret = Environment.GetEnvironmentVariable("akvClientSecret");  
  
KeyVaultClient kvClient = new KeyVaultClient(async (authority, resource, scope) =>  
{  
    var adCredential = new ClientCredential(clientId, clientSecret);  
    var authenticationContext = new AuthenticationContext(authority, null);  
    return (await authenticationContext.AcquireTokenAsync(resource, adCredential)).AccessToken;  
});
```

Save a secret

Now that your application is authenticated, you can put a secret into your keyvault using the [SetSecretAsync method](#) This requires the URL of your key vault, which is in the form

`https://<your-unique-keyvault-name>.vault.azure.net/secrets/`. It also requires a name for the secret -- we're using "mySecret".

```
await kvClient.SetSecretAsync($"{kvURL}", secretName, secretValue);
```

You can verify that the secret has been set with the [az keyvault secret show](#) command:

```
az keyvault secret show --vault-name <your-unique-keyvault-name> --name mySecret
```

Retrieve a secret

You can now retrieve the previously set value with the [GetSecretAsync method](#)

```
var keyvaultSecret = await kvClient.GetSecretAsync($"{kvURL}", secretName).ConfigureAwait(false);
```

Your secret is now saved as `keyvaultSecret.Value`.

Clean up resources

When no longer needed, you can use the Azure CLI or Azure PowerShell to remove your key vault and the corresponding resource group.

```
az group delete -g "myResourceGroup"
```

```
Remove-AzResourceGroup -Name "myResourceGroup"
```

Next steps

In this quickstart you created a key vault, stored a secret, and retrieved that secret. See the [entire console app in GitHub](#).

To learn more about Key Vault and how to integrate it with your applications, continue on to the articles below.

- Implement [Service-to-service authentication to Azure Key Vault using .NET](#)
- Read an [Overview of Azure Key Vault](#)
- See the [Azure Key Vault developer's guide](#)
- Learn about [keys, secrets, and certificates](#)
- Review [Azure Key Vault best practices](#)

Quickstart: Set and retrieve a secret from Azure Key Vault using Resource Manager template

6 minutes to read • [Edit Online](#)

Azure Key Vault is a cloud service that provides a secure store for secrets, such as keys, passwords, certificates, and other secrets. This quickstart focuses on the process of deploying a Resource Manager template to create a key vault and a secret.

Resource Manager template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax, which lets you state what you intend to deploy without having to write the sequence of programming commands to create it. If you want to learn more about developing Resource Manager templates, see [Resource Manager documentation](#) and the [template reference](#).

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

To complete this article, you need:

- Your Azure AD user object ID is needed by the template to configure permissions. The following procedure gets the object ID (GUID).

1. Run the following Azure PowerShell or Azure CLI command by select **Try it**, and then paste the script into the shell pane. To paste the script, right-click the shell, and then select **Paste**.

- [CLI](#)
- [PowerShell](#)

```
echo "Enter your email address that is used to sign in to Azure:" &&
read upn &&
az ad user show --id $upn --query "objectId" &&
echo "Press [ENTER] to continue ..."
```

2. Write down the object ID. You need it in the next section of this quickstart.

Create a vault and a secret

Review the template

The template used in this quickstart is from [Azure Quickstart templates](#).

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "keyVaultName": {
      "type": "string",
      "metadata": {
        "description": "Specifies the name of the key vault."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Specifies the location for the key vault."
```

```
"metadata": {
    "description": "Specifies the Azure location where the key vault should be created."
},
"enabledForDeployment": {
    "type": "bool",
    "defaultValue": false,
    "allowedValues": [
        true,
        false
    ],
    "metadata": {
        "description": "Specifies whether Azure Virtual Machines are permitted to retrieve certificates stored as secrets from the key vault."
    }
},
"enabledForDiskEncryption": {
    "type": "bool",
    "defaultValue": false,
    "allowedValues": [
        true,
        false
    ],
    "metadata": {
        "description": "Specifies whether Azure Disk Encryption is permitted to retrieve secrets from the vault and unwrap keys."
    }
},
"enabledForTemplateDeployment": {
    "type": "bool",
    "defaultValue": false,
    "allowedValues": [
        true,
        false
    ],
    "metadata": {
        "description": "Specifies whether Azure Resource Manager is permitted to retrieve secrets from the key vault."
    }
},
"tenantId": {
    "type": "string",
    "defaultValue": "[subscription().tenantId]",
    "metadata": {
        "description": "Specifies the Azure Active Directory tenant ID that should be used for authenticating requests to the key vault. Get it by using Get-AzSubscription cmdlet."
    }
},
"objectId": {
    "type": "string",
    "metadata": {
        "description": "Specifies the object ID of a user, service principal or security group in the Azure Active Directory tenant for the vault. The object ID must be unique for the list of access policies. Get it by using Get-AzADUser or Get-AzADServicePrincipal cmdlets."
    }
},
"keysPermissions": {
    "type": "array",
    "defaultValue": [
        "list"
    ],
    "metadata": {
        "description": "Specifies the permissions to keys in the vault. Valid values are: all, encrypt, decrypt, wrapKey, unwrapKey, sign, verify, get, list, create, update, import, delete, backup, restore, recover, and purge."
    }
},
"secretsPermissions": {
    "type": "array",
```

```
"defaultValue": [
    "list"
],
"metadata": {
    "description": "Specifies the permissions to secrets in the vault. Valid values are: all, get, list, set, delete, backup, restore, recover, and purge."
}
},
"skuName": {
    "type": "string",
    "defaultValue": "Standard",
    "allowedValues": [
        "Standard",
        "Premium"
    ],
    "metadata": {
        "description": "Specifies whether the key vault is a standard vault or a premium vault."
    }
},
"secretName": {
    "type": "string",
    "metadata": {
        "description": "Specifies the name of the secret that you want to create."
    }
},
"secretValue": {
    "type": "securestring",
    "metadata": {
        "description": "Specifies the value of the secret that you want to create."
    }
}
},
"resources": [
{
    "type": "Microsoft.KeyVault/vaults",
    "name": "[parameters('keyVaultName')]",
    "apiVersion": "2018-02-14",
    "location": "[parameters('location')]",
    "properties": {
        "enabledForDeployment": "[parameters('enabledForDeployment')]",
        "enabledForDiskEncryption": "[parameters('enabledForDiskEncryption')]",
        "enabledForTemplateDeployment": "[parameters('enabledForTemplateDeployment')]",
        "tenantId": "[parameters('tenantId')]",
        "accessPolicies": [
            {
                "objectId": "[parameters('objectId')]",
                "tenantId": "[parameters('tenantId')]",
                "permissions": {
                    "keys": "[parameters('keysPermissions')]",
                    "secrets": "[parameters('secretsPermissions')]"
                }
            }
        ],
        "sku": {
            "name": "[parameters('skuName')]",
            "family": "A"
        },
        "networkAcls": {
            "defaultAction": "Allow",
            "bypass": "AzureServices"
        }
    }
},
{
    "type": "Microsoft.KeyVault/vaults/secrets",
    "name": "[concat(parameters('keyVaultName'), '/', parameters('secretName'))]",
    "apiVersion": "2018-02-14",
    "location": "[parameters('location')]",
    "dependsOn": [

```

```
[{"resourceId('Microsoft.KeyVault/vaults', parameters('keyVaultName'))]"  
],  
"properties": {  
    "value": "[parameters('secretValue')]"  
}  
}  
}  
]  
}
```

Two Azure resources are defined in the template:

- **Microsoft.KeyVault/vaults**: create an Azure key vault.
- **Microsoft.KeyVault/vaults/secrets**: create an key vault secret.

More Azure Key Vault template samples can be found [here](#).

Deploy the template

1. Select the following image to sign in to Azure and open a template. The template creates a key vault and a secret.

[Deploy to Azure >](#)

2. Select or enter the following values.

TEMPLATE

- 101-key-vault-create (2 resources)

BASICS

- * Subscription: <Azure subscription name>
- * Resource group: (New) mykeyvault0219rg (Create new)
- * Location: Central US

SETTINGS

- * Key Vault Name: mykeyvault0219
- Location: [resourceGroup().location]
- Enabled For Deployment: false
- Enabled For Disk Encryption: false
- Enabled For Template Deployment: false
- Tenant Id: [subscription().tenantid]
- * Object Id: <Azure AD user object ID>
- Keys Permissions: ["list"]
- Secrets Permissions: ["list"]
- Sku Name: Standard
- * Secret Name: adminpassword
- * Secret Value: *****

TERMS AND CONDITIONS

Template information | Azure Marketplace Terms | Azure Marketplace

By clicking "Purchase," I (a) agree to the applicable legal terms associated with the offering; (b) authorize Microsoft to charge or bill my current payment method for the fees associated the offering(s), including applicable taxes, with the same billing frequency as my Azure subscription, until I discontinue use of the offering(s); and (c) agree that, if the deployment involves 3rd party offerings, Microsoft may share my contact information and other details of such deployment with the publisher of that offering.

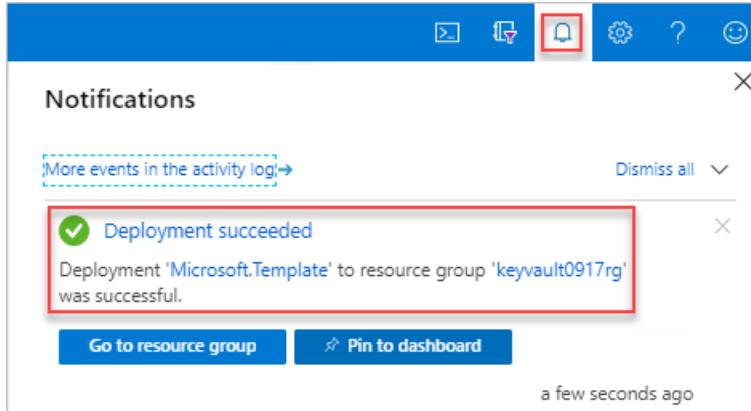
I agree to the terms and conditions stated above

Purchase

Unless it is specified, use the default value to create the key vault and a secret.

- **Subscription:** select an Azure subscription.
- **Resource group:** select **Create new**, enter a unique name for the resource group, and then click **OK**.
- **Location:** select a location. For example, **Central US**.
- **Key Vault Name:** enter a name for the key vault, which must be globally unique within the .vault.azure.net namespace. You need the name in the next section when you validate the deployment.
- **Tenant Id:** the template function automatically retrieves your tenant ID. Don't change the default value.
- **Ad User Id:** enter your Azure AD user object ID that you retrieved from [Prerequisites](#).

- **Secret Name:** enter a name for the secret that you store in the key vault. For example, **adminpassword**.
 - **Secret Value:** enter the secret value. If you store a password, it is recommended to use the generated password you created in Prerequisites.
 - **I agree to the terms and conditions state above:** Select.
3. Select **Purchase**. After the key vault has been deployed successfully, you get a notification:



The Azure portal is used to deploy the template. In addition to the Azure portal, you can also use the Azure PowerShell, Azure CLI, and REST API. To learn other deployment methods, see [Deploy templates](#).

Review deployed resources

You can either use the Azure portal to check the key vault and the secret, or use the following Azure CLI or Azure PowerShell script to list the secret created.

- [CLI](#)
- [PowerShell](#)

```
echo "Enter your key vault name:" &&
read keyVaultName &&
az keyvault secret list --vault-name $keyVaultName &&
echo "Press [ENTER] to continue ..."
```

The output looks similar to:

- [CLI](#)
- [PowerShell](#)

```
[{"id": "https://keyvault0917.vault.azure.net/secrets/adminpassword", "name": "adminpassword", "attributes": {"enabled": true, "notBefore": null, "notAfter": null, "recoveryLevel": "Purgeable", "created": "2019-09-17T19:10:09+00:00", "updated": "2019-09-17T19:10:09+00:00"}, "contentType": null, "managed": null, "tags": null}]
```

Clean up resources

Other Key Vault quickstarts and tutorials build upon this quickstart. If you plan to continue on to work with subsequent quickstarts and tutorials, you may wish to leave these resources in place. When no longer needed,

delete the resource group, which deletes the Key Vault and related resources. To delete the resource group by using Azure CLI or Azure PowerShell:

- [CLI](#)
- [PowerShell](#)

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az group delete --name $resourceGroupName &&
echo "Press [ENTER] to continue ..."
```

Next steps

In this quickstart, you created a key vault and a secret using an Azure Resource Manager template, and validated the deployment. To learn more about Key Vault and Azure Resource Manager, continue on to the articles below.

- Read an [Overview of Azure Key Vault](#)
- Learn more about [Azure Resource Manager](#)
- Get more info on [keys, secrets, and certificates](#)
- Review [Azure Key Vault best practices](#)

Tutorial: Use Azure Key Vault with an Azure web app in .NET

6 minutes to read • [Edit Online](#)

Azure Key Vault helps you protect secrets such as API keys and database connection strings. It provides you with access to your applications, services, and IT resources.

In this tutorial, you learn how to create an Azure web application that can read information from an Azure key vault. The process uses managed identities for Azure resources. For more information about Azure web applications, see [Azure App Service](#).

The tutorial shows you how to:

- Create a key vault.
- Add a secret to the key vault.
- Retrieve a secret from the key vault.
- Create an Azure web app.
- Enable a managed identity for the web app.
- Assign permission for the web app.
- Run the web app on Azure.

Before you begin, read [Key Vault basic concepts](#).

If you don't have an Azure subscription, create a [free account](#).

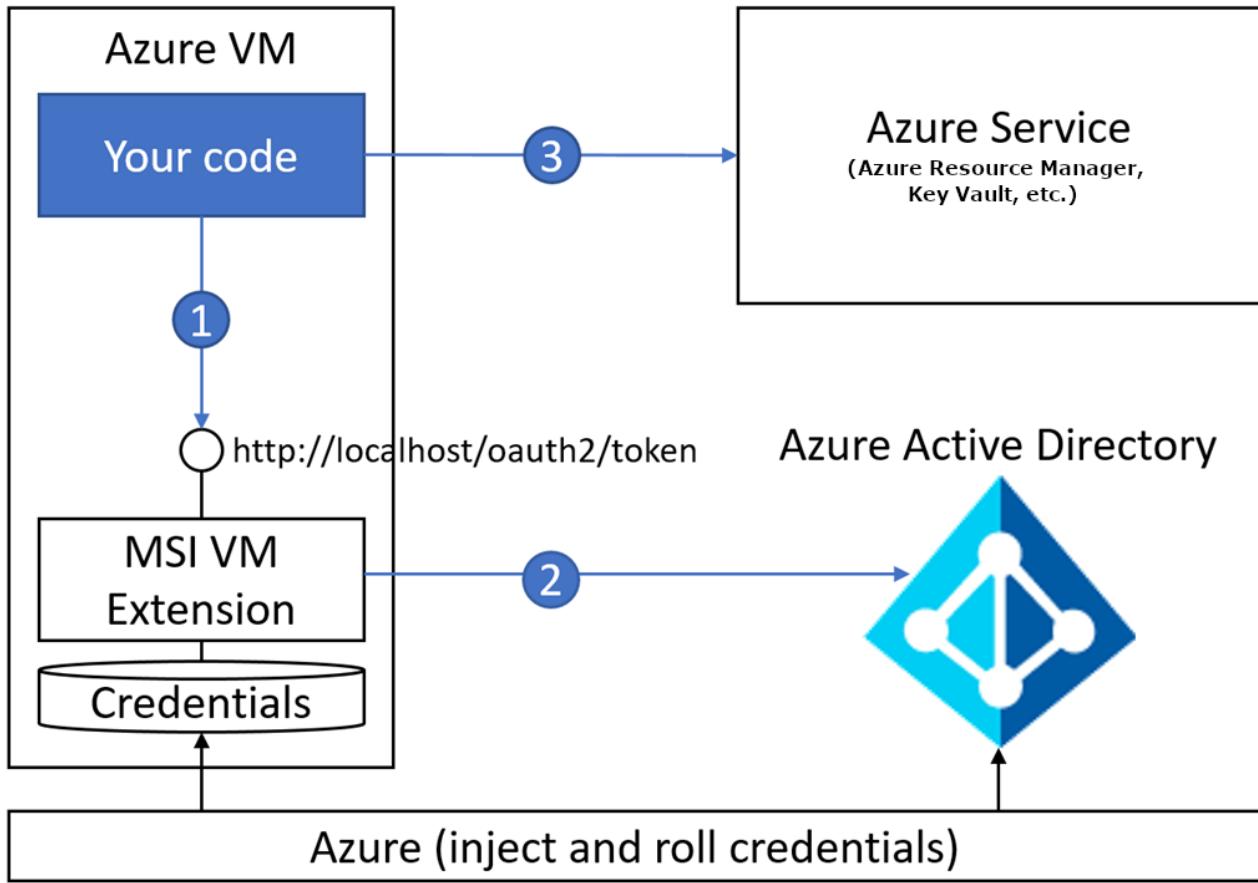
Prerequisites

- For Windows: [.NET Core 2.1 SDK or later](#)
- For Mac: [Visual Studio for Mac](#)
- For Windows, Mac, and Linux:
 - [Git](#)
 - This tutorial requires that you run the Azure CLI locally. You must have the Azure CLI version 2.0.4 or later installed. Run `az --version` to find the version. If you need to install or upgrade the CLI, see [Install Azure CLI 2.0](#).
 - [.NET Core](#)

About Managed Service Identity

Azure Key Vault stores credentials securely, so they're not displayed in your code. However, you need to authenticate to Azure Key Vault to retrieve your keys. To authenticate to Key Vault, you need a credential. It's a classic bootstrap dilemma. Managed Service Identity (MSI) solves this issue by providing a *bootstrap identity* that simplifies the process.

When you enable MSI for an Azure service, such as Azure Virtual Machines, Azure App Service, or Azure Functions, Azure creates a [service principal](#). MSI does this for the instance of the service in Azure Active Directory (Azure AD) and injects the service principal credentials into that instance.



Next, to get an access token, your code calls a local metadata service that's available on the Azure resource. Your code uses the access token that it gets from the local MSI endpoint to authenticate to an Azure Key Vault service.

Log in to Azure

To log in to Azure by using the Azure CLI, enter:

```
az login
```

Create a resource group

An Azure resource group is a logical container into which Azure resources are deployed and managed.

Create a resource group by using the [az group create](#) command.

Then, select a resource group name and fill in the placeholder. The following example creates a resource group in the West US location:

```
# To list locations: az account list-locations --output table
az group create --name "<YourResourceGroupName>" --location "West US"
```

You use this resource group throughout this tutorial.

Create a key vault

To create a key vault in your resource group, provide the following information:

- Key vault name: a string of 3 to 24 characters that can contain only numbers (0-9), letters (a-z, A-Z), and hyphens (-)

- Resource group name
- Location: **West US**

In the Azure CLI, enter the following command:

```
az keyvault create --name "<YourKeyVaultName>" --resource-group "<YourResourceGroupName>" --location "West US"
```

At this point, your Azure account is the only one that's authorized to perform operations on this new vault.

Add a secret to the key vault

Now you can add a secret. It might be a SQL connection string or any other information that you need to keep both secure and available to your application.

To create a secret in the key vault called **AppSecret**, enter the following command:

```
az keyvault secret set --vault-name "<YourKeyVaultName>" --name "AppSecret" --value "MySecret"
```

This secret stores the value **MySecret**.

To view the value that's contained in the secret as plain text, enter the following command:

```
az keyvault secret show --name "AppSecret" --vault-name "<YourKeyVaultName>"
```

This command displays the secret information, including the URI.

After you complete these steps, you should have a URI to a secret in a key vault. Make note of this information for later use in this tutorial.

Create a .NET Core web app

To create a .NET Core web app and publish it to Azure, follow the instructions in [Create an ASP.NET Core web app in Azure](#).

You can also watch this video:

Open and edit the solution

1. Go to the **Pages > About.cshtml.cs** file.
2. Install these NuGet packages:
 - [AppAuthentication](#)
 - [KeyVault](#)
3. Import the following code to the *About.cshtml.cs* file:

```
using Microsoft.Azure.KeyVault;
using Microsoft.Azure.KeyVault.Models;
using Microsoft.Azure.Services.AppAuthentication;
```

Your code in the *AboutModel* class should look like this:

```

public class AboutModel : PageModel
{
    public string Message { get; set; }

    public async Task OnGetAsync()
    {
        Message = "Your application description page.";

        int retries = 0;
        bool retry = false;
        try
        {
            /* The next four lines of code show you how to use AppAuthentication library to fetch
            secrets from your key vault */
            AzureServiceTokenProvider azureServiceTokenProvider = new AzureServiceTokenProvider();
            KeyVaultClient keyVaultClient = new KeyVaultClient(new
            KeyVaultClient.AuthenticationCallback(azureServiceTokenProvider.KeyVaultTokenCallback));
            var secret = await
            keyVaultClient.GetSecretAsync("https://<YourKeyVaultName>.vault.azure.net/secrets/AppSecret")
                .ConfigureAwait(false);
            Message = secret.Value;
        }
        /* If you have throttling errors see this tutorial https://docs.microsoft.com/azure/key-
        vault/tutorial-net-create-vault-azure-web-app */
        ///<exception cref="KeyVaultErrorException">
        ///Thrown when the operation returned an invalid status code
        ///</exception>
        catch (KeyVaultErrorException keyVaultException)
        {
            Message = keyVaultException.Message;
        }
    }

    // This method implements exponential backoff if there are 429 errors from Azure Key Vault
    private static long getWaitTime(int retryCount)
    {
        long waitTime = ((long)Math.Pow(2, retryCount) * 100L);
        return waitTime;
    }

    // This method fetches a token from Azure Active Directory, which can then be provided to Azure Key
    Vault to authenticate
    public async Task<string> GetAccessTokenAsync()
    {
        var azureServiceTokenProvider = new AzureServiceTokenProvider();
        string accessToken = await
        azureServiceTokenProvider.GetAccessTokenAsync("https://vault.azure.net");
        return accessToken;
    }
}

```

Run the web app

1. On the main menu of Visual Studio 2019, select **Debug > Start**, with or without debugging.
2. In the browser, go to the **About** page.
The value for **AppSecret** is displayed.

Enable a managed identity

Azure Key Vault provides a way to securely store credentials and other secrets, but your code needs to authenticate to Key Vault to retrieve them. [Managed identities for Azure resources overview](#) helps to solve this problem by giving Azure services an automatically managed identity in Azure AD. You can use this identity to authenticate to any service that supports Azure AD authentication, including Key Vault, without having to display credentials in

your code.

In the Azure CLI, to create the identity for this application, run the assign-identity command:

```
az webapp identity assign --name "<YourAppName>" --resource-group "<YourResourceGroupName>"
```

Replace <YourAppName> with the name of the published app on Azure.

For example, if your published app name was **MyAwesomeapp.azurewebsites.net**, replace <YourAppName> with **MyAwesomeapp**.

Make a note of the `PrincipalId` when you publish the application to Azure. The output of the command in step 1 should be in the following format:

```
{
  "principalId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "tenantId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "type": "SystemAssigned"
}
```

NOTE

The command in this procedure is the equivalent of going to the [Azure portal](#) and switching the **Identity / System assigned** setting to **On** in the web application properties.

Assign permissions to your app

Replace <YourKeyVaultName> with the name of your key vault, and replace <PrincipalId> with the value of the **PrincipalId** in the following command:

```
az keyvault set-policy --name '<YourKeyVaultName>' --object-id <PrincipalId> --secret-permissions get list
```

This command gives the identity (MSI) of the app service permission to do **get** and **list** operations on your key vault.

Publish the web app to Azure

Publish your web app to Azure once again to verify that your live web app can fetch the secret value.

1. In Visual Studio, select the **key-vault-dotnet-core-quickstart** project.
2. Select **Publish > Start**.
3. Select **Create**.

When you run the application, you should see that it can retrieve your secret value.

Now, you've successfully created a web app in .NET that stores and fetches its secrets from your key vault.

Clean up resources

When they are no longer needed, you can delete the virtual machine and your key vault.

Next steps

[Azure Key Vault Developer's Guide](#)

Tutorial: Use a Linux VM and a .NET app to store secrets in Azure Key Vault

7 minutes to read • [Edit Online](#)

Azure Key Vault helps you to protect secrets such as API Keys and database connection strings that are needed to access your applications, services, and IT resources.

In this tutorial, you set up a .NET console application to read information from Azure Key Vault by using managed identities for Azure resources. You learn how to:

- Create a key vault
- Store a secret in Key Vault
- Create an Azure Linux virtual machine
- Enable a [managed identity](#) for the virtual machine
- Grant the required permissions for the console application to read data from Key Vault
- Retrieve a secret from Key Vault

Before we go any further, read about [key vault basic concepts](#).

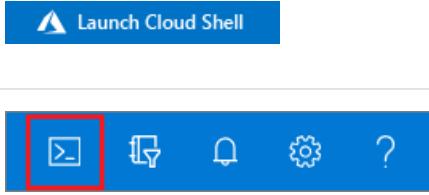
Prerequisites

- [Git](#).
- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- [Azure CLI 2.0 or later](#) or Azure Cloud Shell.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.

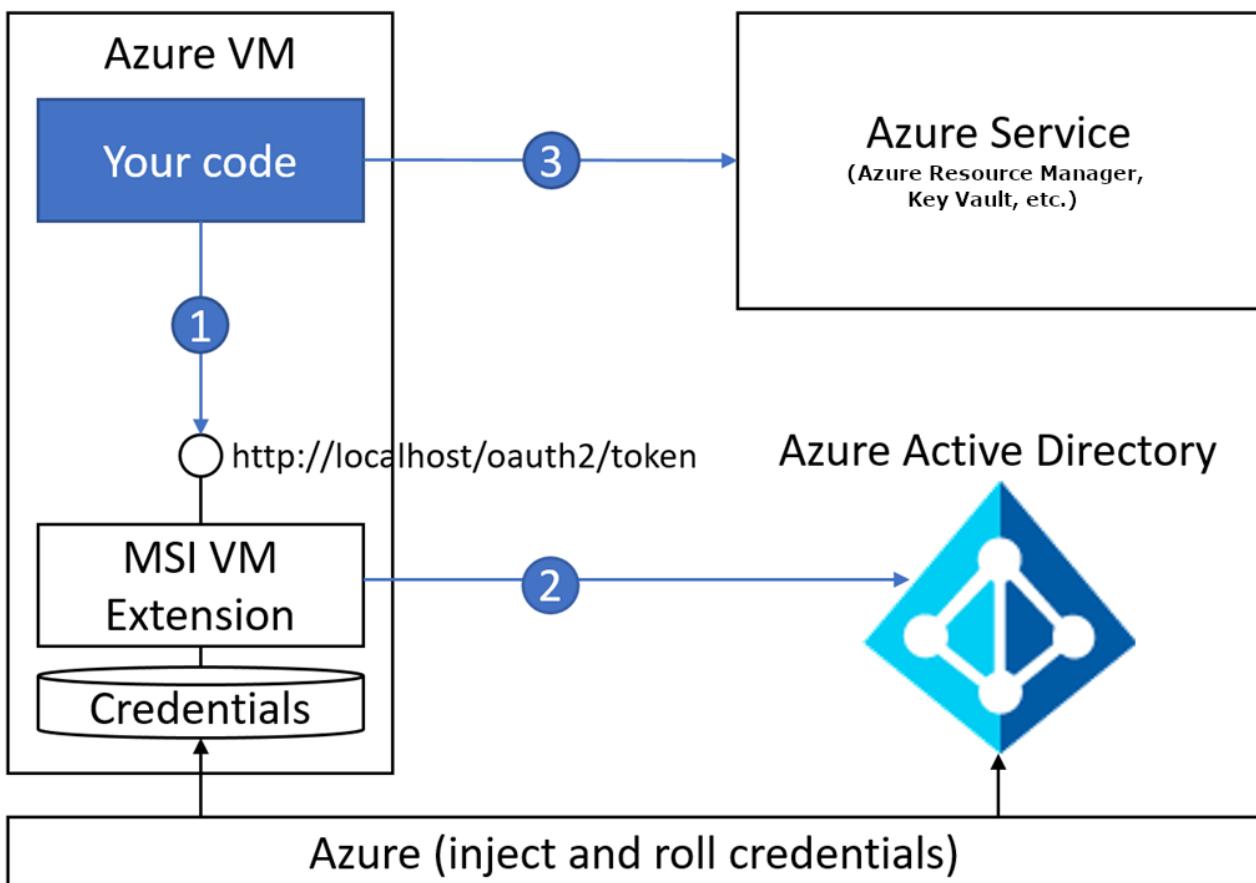
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.

4. Select **Enter** to run the code.

Understand Managed Service Identity

Azure Key Vault can store credentials securely so they aren't in your code, but to retrieve them you need to authenticate to Azure Key Vault. However, to authenticate to Key Vault, you need a credential. It's a classic bootstrap problem. With Azure and Azure Active Directory (Azure AD), Managed Service Identity (MSI) can provide a bootstrap identity that makes it much simpler to get things started.

When you enable MSI for an Azure service like Virtual Machines, App Service, or Functions, Azure creates a service principal for the instance of the service in Azure Active Directory. It injects the credentials for the service principal into the instance of the service.



Next, your code calls a local metadata service available on the Azure resource to get an access token. Your code uses the access token it gets from the local MSI_ENDPOINT to authenticate to an Azure Key Vault service.

Sign in to Azure

To sign in to Azure by using the Azure CLI, enter:

```
az login
```

Create a resource group

Create a resource group by using the `az group create` command. An Azure resource group is a logical container into which Azure resources are deployed and managed.

Create a resource group in the West US location. Pick a name for your resource group and replace `<YourResourceGroupName>` in the following example:

```
# To list locations: az account list-locations --output table  
az group create --name "<YourResourceGroupName>" --location "West US"
```

You use this resource group throughout the tutorial.

Create a key vault

Next, create a key vault in your resource group. Provide the following information:

- Key vault name: a string of 3 to 24 characters that can contain only numbers, letters, and hyphens (0-9, a-z, A-Z, and -).
- Resource group name
- Location: **West US**

```
az keyvault create --name "<YourKeyVaultName>" --resource-group "<YourResourceGroupName>" --location "West US"
```

At this point, only your Azure account is authorized to perform any operations on this new vault.

Add a secret to the key vault

Now, you add a secret. In a real-world scenario, you might be storing a SQL connection string or any other information that you need to keep securely, but make available to your application.

For this tutorial, type the following commands to create a secret in the key vault. The secret is called **AppSecret** and its value is **MySecret**.

```
az keyvault secret set --vault-name "<YourKeyVaultName>" --name "AppSecret" --value "MySecret"
```

Create a Linux virtual machine

Create a VM with the `az vm create` command.

The following example creates a VM named **myVM** and adds a user account named **azureuser**. The `--generate-ssh-keys` parameter is used to automatically generate an SSH key and put it in the default key location (`~/.ssh`). To use a specific set of keys instead, use the `--ssh-key-value` option.

```
az vm create \  
  --resource-group myResourceGroup \  
  --name myVM \  
  --image UbuntuLTS \  
  --admin-username azureuser \  
  --generate-ssh-keys
```

It takes a few minutes to create the VM and supporting resources. The following example output shows that the VM create operation was successful.

```
{  
    "fqdns": "",  
    "id":  
        "/subscriptions/<guid>/resourceGroups/myResourceGroup/providers/Microsoft.Compute/virtualMachines/myVM",  
    "location": "westus",  
    "macAddress": "00-00-00-00-00-00",  
    "powerState": "VM running",  
    "privateIpAddress": "XX.XX.XX.XX",  
    "publicIpAddress": "XX.XX.XXX.XXX",  
    "resourceGroup": "myResourceGroup"  
}
```

Make a note of your `publicIpAddress` in the output from your VM. You'll use this address to access the VM in later steps.

Assign an identity to the VM

Create a system-assigned identity to the virtual machine by running the following command:

```
az vm identity assign --name <NameOfYourVirtualMachine> --resource-group <YourResourceGroupName>
```

The output of the command should be:

```
{  
    "systemAssignedIdentity": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
    "userAssignedIdentities": {}  
}
```

Make a note of the `systemAssignedIdentity`. You use it in the next step.

Give the VM identity permission to Key Vault

Now you can give Key Vault permission to the identity you created. Run the following command:

```
az keyvault set-policy --name '<YourKeyVaultName>' --object-id <VMSystemAssignedIdentity> --secret-permissions  
get list
```

Log in to the VM

Log in to the virtual machine by using a terminal.

```
ssh azureuser@<PublicIpAddress>
```

Install .NET Core on Linux

On your Linux VM:

Register the Microsoft product key as trusted by running the following commands:

```
curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.gpg  
sudo mv microsoft.gpg /etc/apt/trusted.gpg.d/microsoft.gpg
```

Set up desired version host package feed based on operating system:

```

# Ubuntu 17.10
sudo sh -c 'echo "deb [arch=amd64] https://packages.microsoft.com/repos/microsoft-ubuntu-artful-prod artful
main" > /etc/apt/sources.list.d/dotnetdev.list'
sudo apt-get update

# Ubuntu 17.04
sudo sh -c 'echo "deb [arch=amd64] https://packages.microsoft.com/repos/microsoft-ubuntu-zesty-prod zesty
main" > /etc/apt/sources.list.d/dotnetdev.list'
sudo apt-get update

# Ubuntu 16.04 / Linux Mint 18
sudo sh -c 'echo "deb [arch=amd64] https://packages.microsoft.com/repos/microsoft-ubuntu-xenial-prod xenial
main" > /etc/apt/sources.list.d/dotnetdev.list'
sudo apt-get update

# Ubuntu 14.04 / Linux Mint 17
sudo sh -c 'echo "deb [arch=amd64] https://packages.microsoft.com/repos/microsoft-ubuntu-trusty-prod trusty
main" > /etc/apt/sources.list.d/dotnetdev.list'
sudo apt-get update

```

Install .NET and check the version:

```

sudo apt-get install dotnet-sdk-2.1.4
dotnet --version

```

Create and run a sample .NET app

Run the following commands. You should see "Hello World" printed to the console.

```

dotnet new console -o helloworldapp
cd helloworldapp
dotnet run

```

Edit the console app to fetch your secret

Open Program.cs file and add these packages:

```

using System;
using System.IO;
using System.Net;
using System.Text;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

```

It's a two-step process to change the class file to enable the app to access the secret in the key vault.

1. Fetch a token from the local MSI endpoint on the VM that in turn fetches a token from Azure Active Directory.
2. Pass the token to Key Vault and fetch your secret.

Edit the class file to contain the following code:

```

class Program
{
    static void Main(string[] args)
    {
        // Step 1: Get a token from local (URI) Managed Service Identity endpoint which in turn
        fetches it from Azure Active Directory
        var token = GetToken();

        // Step 2: Fetch the secret value from Key Vault
        System.Console.WriteLine(FetchSecretValueFromKeyVault(token));
    }

    static string GetToken()
    {
        WebRequest request =
        WebRequest.Create("http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-
        01&resource=https%3A%2F%2Fvault.azure.net");
        request.Headers.Add("Metadata", "true");
        WebResponse response = request.GetResponse();
        return ParseWebResponse(response, "access_token");
    }

    static string FetchSecretValueFromKeyVault(string token)
    {
        WebRequest kvRequest =
        WebRequest.Create("https://prashanthwinvmvault.vault.azure.net/secrets/RandomSecret?api-version=2016-10-
        01");
        kvRequest.Headers.Add("Authorization", "Bearer " + token);
        WebResponse kvResponse = kvRequest.GetResponse();
        return ParseWebResponse(kvResponse, "value");
    }

    private static string ParseWebResponse(WebResponse response, string tokenName)
    {
        string token = String.Empty;
        using (Stream stream = response.GetResponseStream())
        {
            StreamReader reader = new StreamReader(stream, Encoding.UTF8);
            String responseString = reader.ReadToEnd();

            JObject joResponse = JObject.Parse(responseString);
            JValue ojObject = (JValue)joResponse[tokenName];
            token = ojObject.Value.ToString();
        }
        return token;
    }
}

```

Now you've learned how to perform operations with Azure Key Vault in a .NET application running on an Azure Linux virtual machine.

Clean up resources

Delete the resource group, virtual machine, and all related resources when you no longer need them. To do so, select the resource group for the VM and select **Delete**.

Delete the key vault by using the `az keyvault delete` command:

```

az keyvault delete --name
    [--resource group]
    [--subscription]

```

Next steps

[Azure Key Vault REST API](#)

Tutorial: Use a Linux VM and a Python app to store secrets in Azure Key Vault

6 minutes to read • [Edit Online](#)

Azure Key Vault helps you protect secrets such as the API keys and database connection strings needed to access your applications, services, and IT resources.

In this tutorial, you set up an Azure web application to read information from Azure Key Vault by using managed identities for Azure resources. You learn how to:

- Create a key vault
- Store a secret in your key vault
- Create a Linux virtual machine
- Enable a [managed identity](#) for the virtual machine
- Grant the required permissions for the console application to read data from the key vault
- Retrieve a secret from your key vault

Before you go any further, make sure you understand the [basic concepts about Key Vault](#).

Prerequisites

- [Git](#).
- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- [Azure CLI version 2.0.4 or later](#) or Azure Cloud Shell.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.

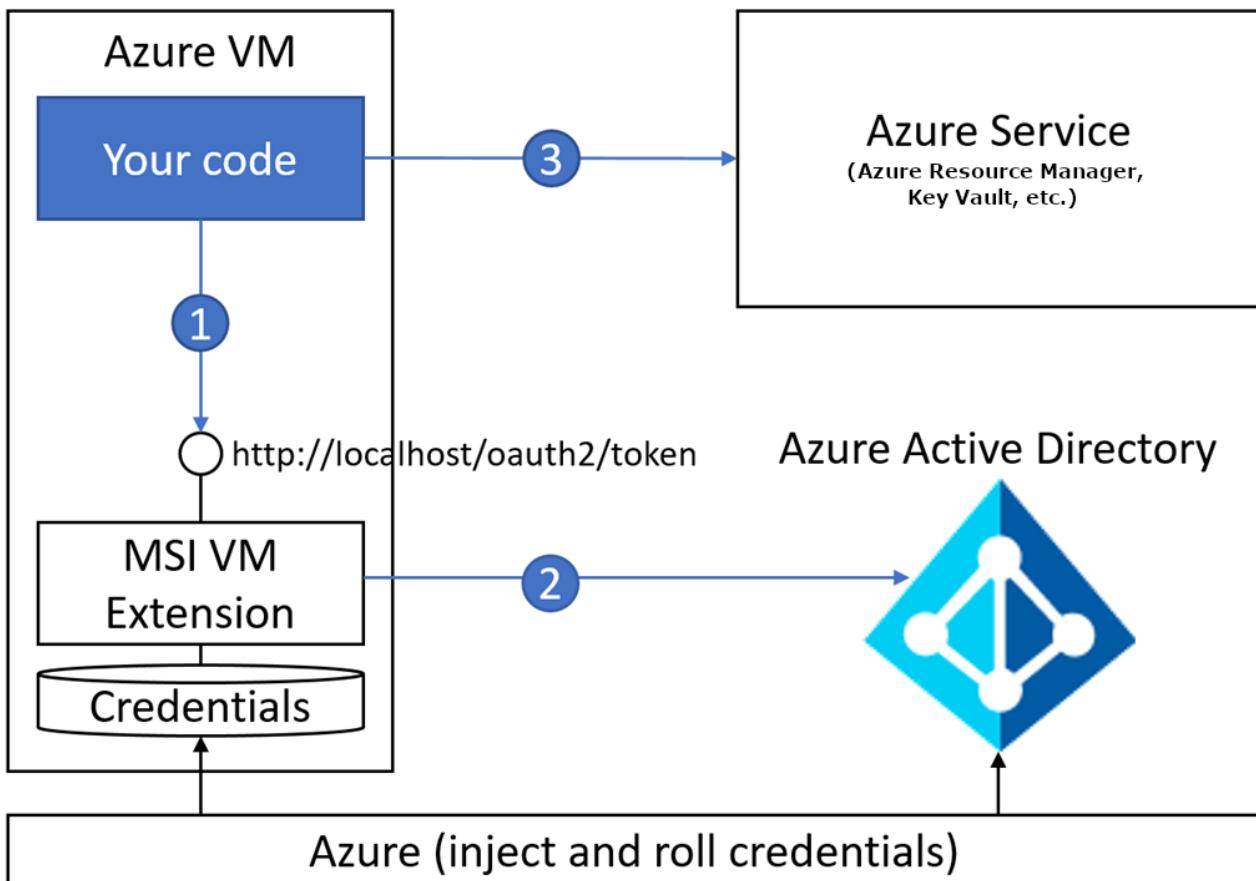
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.

4. Select **Enter** to run the code.

Understand Managed Service Identity

Azure Key Vault can store credentials securely so they aren't in your code. To retrieve them, you need to authenticate to Azure Key Vault. However, to authenticate to Key Vault, you need a credential. It's a classic bootstrap problem. Through Azure and Azure Active Directory (Azure AD), Managed Service Identity (MSI) provides a bootstrap identity that makes it simpler to get things started.

When you enable MSI for an Azure service like Virtual Machines, App Service, or Functions, Azure creates a service principal for the instance of the service in Azure AD. It injects the credentials for the service principal into the instance of the service.



Next, your code calls a local metadata service available on the Azure resource to get an access token. Your code uses the access token that it gets from the local MSI endpoint to authenticate to an Azure Key Vault service.

Sign in to Azure

To sign in to Azure by using the Azure CLI, enter:

```
az login
```

Create a resource group

An Azure resource group is a logical container into which Azure resources are deployed and managed.

Create a resource group by using the `az group create` command in the West US location with the following code.

Replace `YourResourceGroupName` with a name of your choice.

```
# To list locations: az account list-locations --output table  
az group create --name "<YourResourceGroupName>" --location "West US"
```

You use this resource group throughout the tutorial.

Create a key vault

Next, you create a key vault in the resource group that you created in the previous step. Provide the following information:

- Key vault name: The name must be a string of 3-24 characters and must contain only 0-9, a-z, A-Z, and hyphens (-).
- Resource group name.
- Location: **West US**.

```
az keyvault create --name "<YourKeyVaultName>" --resource-group "<YourResourceGroupName>" --location "West US"
```

At this point, your Azure account is the only one that's authorized to perform any operations on this new vault.

Add a secret to the key vault

We're adding a secret to help illustrate how this works. You might want to store a SQL connection string or any other information that needs to be both kept secure and available to your application.

Type the following commands to create a secret in the key vault called *AppSecret*. This secret will store the value **MySecret**.

```
az keyvault secret set --vault-name "<YourKeyVaultName>" --name "AppSecret" --value "MySecret"
```

Create a Linux virtual machine

Create a VM by using the `az vm create` command.

The following example creates a VM named **myVM** and adds a user account named **azureuser**. The `--generate-ssh-keys` parameter automatically generates an SSH key and puts it in the default key location (`~/.ssh`). To create a specific set of keys instead, use the `--ssh-key-value` option.

```
az vm create \  
  --resource-group myResourceGroup \  
  --name myVM \  
  --image UbuntuLTS \  
  --admin-username azureuser \  
  --generate-ssh-keys
```

It takes a few minutes to create the VM and supporting resources. The following example output shows that the VM creation was successful:

```
{  
    "fqdns": "",  
    "id":  
        "/subscriptions/<guid>/resourceGroups/myResourceGroup/providers/Microsoft.Compute/virtualMachines/myVM",  
    "location": "westus",  
    "macAddress": "00-00-00-00-00-00",  
    "powerState": "VM running",  
    "privateIpAddress": "XX.XX.XX.XX",  
    "publicIpAddress": "XX.XX.XXX.XXX",  
    "resourceGroup": "myResourceGroup"  
}
```

Make a note of your own `publicIpAddress` in the output from your VM. You'll use this address to access the VM in later steps.

Assign an identity to the VM

Create a system-assigned identity to the virtual machine by running the following command:

```
az vm identity assign --name <NameOfYourVirtualMachine> --resource-group <YourResourceGroupName>
```

The output of the command is as follows.

```
{  
    "systemAssignedIdentity": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
    "userAssignedIdentities": {}  
}
```

Make a note of the `systemAssignedIdentity`. You use it the next step.

Give the VM identity permission to Key Vault

Now you can give Key Vault permission to the identity you created. Run the following command:

```
az keyvault set-policy --name '<YourKeyVaultName>' --object-id <VMSystemAssignedIdentity> --secret-permissions  
get list
```

Log in to the VM

Log in to the virtual machine by using a terminal.

```
ssh azureuser@<PublicIpAddress>
```

Install Python library on the VM

Download and install the [requests](#) Python library to make HTTP GET calls.

Create, edit, and run the sample Python app

Create a Python file called **Sample.py**.

Open Sample.py and edit it to contain the following code:

```
# importing the requests library
import requests

# Step 1: Fetch an access token from an MSI-enabled Azure resource
# Note that the resource here is https://vault.azure.net for the public cloud, and api-version is 2018-02-01
MSI_ENDPOINT = "http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net"
r = requests.get(MSI_ENDPOINT, headers = {"Metadata" : "true"})

# Extracting data in JSON format
# This request gets an access token from Azure Active Directory by using the local MSI endpoint
data = r.json()

# Step 2: Pass the access token received from the previous HTTP GET call to the key vault
KeyVaultURL = "https://prashanthwinvmmvault.vault.azure.net/secrets/RandomSecret?api-version=2016-10-01"
kvSecret = requests.get(url = KeyVaultURL, headers = {"Authorization": "Bearer " + data["access_token"]})

print(kvSecret.json()["value"])
```

The preceding code performs a two-step process:

1. Fetches a token from the local MSI endpoint on the VM. The endpoint then fetches a token from Azure Active Directory.
2. Passes the token to the key vault and fetches your secret.

Run the following command. You should see the secret value.

```
python Sample.py
```

In this tutorial, you learned how to use Azure Key Vault with a Python app running on a Linux virtual machine.

Clean up resources

Delete the resource group, virtual machine, and all related resources when you no longer need them. To do so, select the resource group for the VM and select **Delete**.

Delete the key vault by using the `az keyvault delete` command:

```
az keyvault delete --name
                   [--resource-group]
                   [--subscription]
```

Next steps

[Azure Key Vault REST API](#)

Tutorial: Use Azure Key Vault with a Windows virtual machine in .NET

5 minutes to read • [Edit Online](#)

Azure Key Vault helps you to protect secrets such as API keys, the database connection strings you need to access your applications, services, and IT resources.

In this tutorial, you learn how to get a console application to read information from Azure Key Vault. To do so, you use managed identities for Azure resources.

The tutorial shows you how to:

- Create a resource group.
- Create a key vault.
- Add a secret to the key vault.
- Retrieve a secret from the key vault.
- Create an Azure virtual machine.
- Enable a [managed identity](#) for the Virtual Machine.
- Assign permissions to the VM identity.

Before you begin, read [Key Vault basic concepts](#).

If you don't have an Azure subscription, create a [free account](#).

Prerequisites

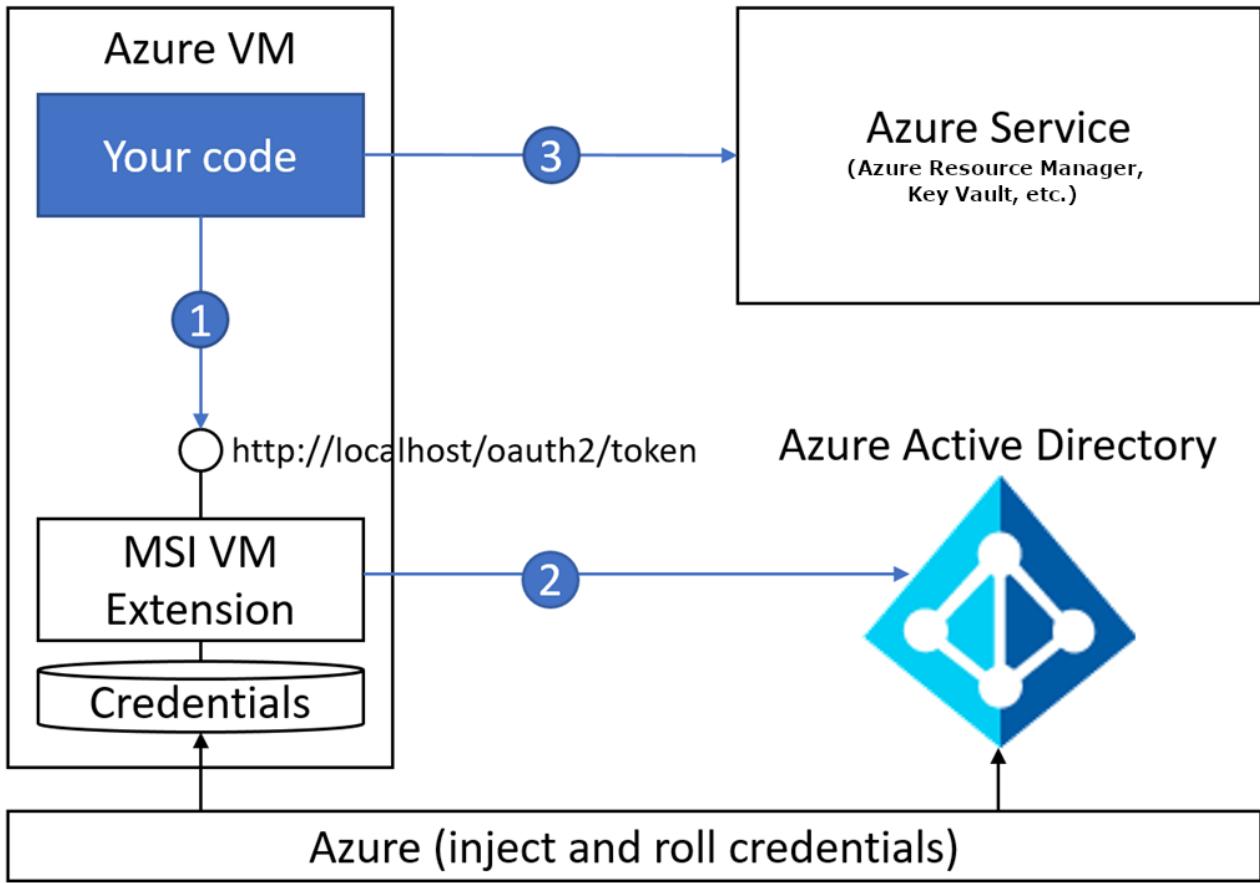
For Windows, Mac, and Linux:

- [Git](#)
- This tutorial requires that you run the Azure CLI locally. You must have the Azure CLI version 2.0.4 or later installed. Run `az --version` to find the version. If you need to install or upgrade the CLI, see [Install Azure CLI 2.0](#).

About Managed Service Identity

Azure Key Vault stores credentials securely, so they're not displayed in your code. However, you need to authenticate to Azure Key Vault to retrieve your keys. To authenticate to Key Vault, you need a credential. It's a classic bootstrap dilemma. Managed Service Identity (MSI) solves this issue by providing a *bootstrap identity* that simplifies the process.

When you enable MSI for an Azure service, such as Azure Virtual Machines, Azure App Service, or Azure Functions, Azure creates a [service principal](#). MSI does this for the instance of the service in Azure Active Directory (Azure AD) and injects the service principal credentials into that instance.



Next, to get an access token, your code calls a local metadata service that's available on the Azure resource. To authenticate to an Azure Key Vault service, your code uses the access token that it gets from the local MSI endpoint.

Create resources and assign permissions

Before you start coding you need to create some resources, put a secret into your key vault, and assign permissions.

Sign in to Azure

To sign in to Azure by using the Azure CLI, enter:

```
az login
```

Create a resource group

An Azure resource group is a logical container into which Azure resources are deployed and managed. Create a resource group by using the [az group create](#) command.

This example creates a resource group in the West US location:

```
# To list locations: az account list-locations --output table
az group create --name "<YourResourceGroupName>" --location "West US"
```

Your newly created resource group will be used throughout this tutorial.

Create a key vault and populate it with a secret

Create a key vault in your resource group by providing the [az keyvault create](#) command with the following information:

- Key vault name: a string of 3 to 24 characters that can contain only numbers (0-9), letters (a-z, A-Z), and hyphens (-)
- Resource group name
- Location: **West US**

```
az keyvault create --name "<YourKeyVaultName>" --resource-group "<YourResourceGroupName>" --location "West US"
```

At this point, your Azure account is the only one that's authorized to perform operations on this new key vault.

Now add a secret to your key vault using the [az keyvault secret set](#) command

To create a secret in the key vault called **AppSecret**, enter the following command:

```
az keyvault secret set --vault-name "<YourKeyVaultName>" --name "AppSecret" --value "MySecret"
```

This secret stores the value **MySecret**.

Create a virtual machine

Create a virtual machine by using one of the following methods:

- [The Azure CLI](#)
- [PowerShell](#)
- [The Azure portal](#)

Assign an identity to the VM

Create a system-assigned identity for the virtual machine with the [az vm identity assign](#) command:

```
az vm identity assign --name <NameOfYourVirtualMachine> --resource-group <YourResourceGroupName>
```

Note the system-assigned identity that's displayed in the following code. The output of the preceding command would be:

```
{
  "systemAssignedIdentity": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "userAssignedIdentities": {}
}
```

Assign permissions to the VM identity

Assign the previously created identity permissions to your key vault with the [az keyvault set-policy](#) command:

```
az keyvault set-policy --name '<YourKeyVaultName>' --object-id <VMSystemAssignedIdentity> --secret-permissions get list
```

Sign in to the virtual machine

To sign in to the virtual machine, follow the instructions in [Connect and sign in to an Azure virtual machine running Windows](#).

Set up the console app

Create a console app and install the required packages using the `dotnet` command.

Install .NET Core

To install .NET Core, go to the [.NET downloads](#) page.

Create and run a sample .NET app

Open a command prompt.

You can print "Hello World" to the console by running the following commands:

```
dotnet new console -o helloworldapp  
cd helloworldapp  
dotnet run
```

Install the packages

From the console window, install the .NET packages required for this quickstart:

```
dotnet add package System.IO;  
dotnet add package System.Net;  
dotnet add package System.Text;  
dotnet add package Newtonsoft.Json;  
dotnet add package Newtonsoft.Json.Linq;
```

Edit the console app

Open the *Program.cs* file and add these packages:

```
using System;  
using System.IO;  
using System.Net;  
using System.Text;  
using Newtonsoft.Json;  
using Newtonsoft.Json.Linq;
```

Edit the class file to contain the code in the following three-step process:

1. Fetch a token from the local MSI endpoint on the VM. Doing so also fetches a token from Azure AD.
2. Pass the token to your key vault, and then fetch your secret.
3. Add the vault name and secret name to the request.

```

class Program
{
    static void Main(string[] args)
    {
        // Step 1: Get a token from the local (URI) Managed Service Identity endpoint, which in turn
        fetches it from Azure AD
        var token = GetToken();

        // Step 2: Fetch the secret value from your key vault
        System.Console.WriteLine(FetchSecretValueFromKeyVault(token));
    }

    static string GetToken()
    {
        WebRequest request = WebRequest.Create("http://169.254.169.254/metadata/identity/oauth2/token?api-
version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net");
        request.Headers.Add("Metadata", "true");
        WebResponse response = request.GetResponse();
        return ParseWebResponse(response, "access_token");
    }

    static string FetchSecretValueFromKeyVault(string token)
    {
        //Step 3: Add the vault name and secret name to the request.
        WebRequest kvRequest =
        WebRequest.Create("https://<YourVaultName>.vault.azure.net/secrets/<YourSecretName>?api-version=2016-10-01");
        kvRequest.Headers.Add("Authorization", "Bearer " + token);
        WebResponse kvResponse = kvRequest.GetResponse();
        return ParseWebResponse(kvResponse, "value");
    }

    private static string ParseWebResponse(WebResponse response, string tokenName)
    {
        string token = String.Empty;
        using (Stream stream = response.GetResponseStream())
        {
            StreamReader reader = new StreamReader(stream, Encoding.UTF8);
            String responseString = reader.ReadToEnd();

            JObject joResponse = JObject.Parse(responseString);
            JValue ojObject = (JValue)joResponse[tokenName];
            token = ojObject.Value.ToString();
        }
        return token;
    }
}

```

The preceding code shows you how to do operations with Azure Key Vault in a Windows virtual machine.

Clean up resources

When they are no longer needed, delete the virtual machine and your key vault.

Next steps

[Azure Key Vault REST API](#)

Tutorial: Use Azure Key Vault with a Windows virtual machine in Python

4 minutes to read • [Edit Online](#)

Azure Key Vault helps you to protect secrets such as API keys, the database connection strings you need to access your applications, services, and IT resources.

In this tutorial, you learn how to get a console application to read information from Azure Key Vault. To do so, you use managed identities for Azure resources.

The tutorial shows you how to:

- Create a key vault.
- Add a secret to the key vault.
- Retrieve a secret from the key vault.
- Create an Azure virtual machine.
- Enable a managed identity.
- Assign permissions to the VM identity.

Before you begin, read [Key Vault basic concepts](#).

If you don't have an Azure subscription, create a [free account](#).

Prerequisites

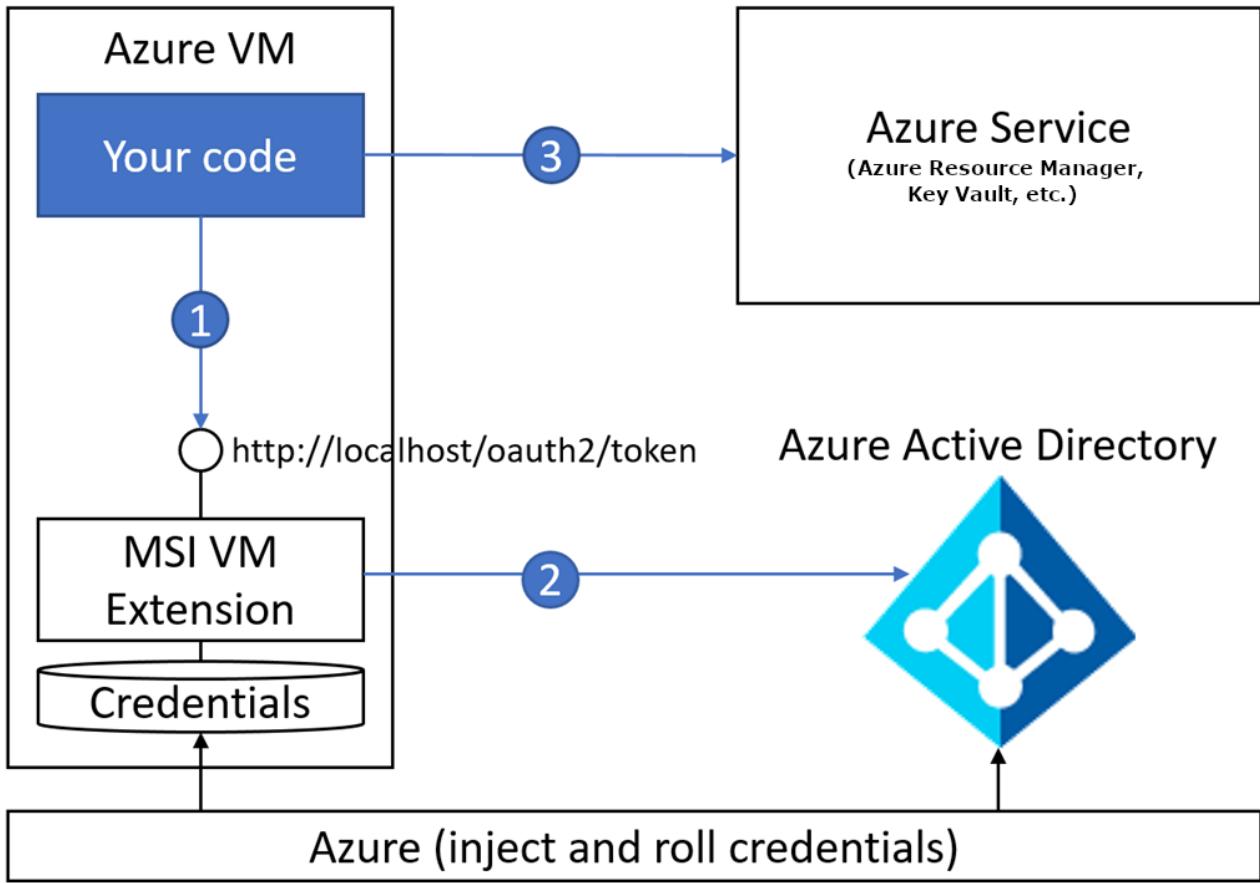
For Windows, Mac, and Linux:

- [Git](#)
- This tutorial requires that you run the Azure CLI locally. You must have the Azure CLI version 2.0.4 or later installed. Run `az --version` to find the version. If you need to install or upgrade the CLI, see [Install Azure CLI 2.0](#).

About Managed Service Identity

Azure Key Vault stores credentials securely, so they're not displayed in your code. However, you need to authenticate to Azure Key Vault to retrieve your keys. To authenticate to Key Vault, you need a credential. It's a classic bootstrap dilemma. Managed Service Identity (MSI) solves this issue by providing a *bootstrap identity* that simplifies the process.

When you enable MSI for an Azure service, such as Azure Virtual Machines, Azure App Service, or Azure Functions, Azure creates a [service principal](#). MSI does this for the instance of the service in Azure Active Directory (Azure AD) and injects the service principal credentials into that instance.



Next, to get an access token, your code calls a local metadata service that's available on the Azure resource. To authenticate to an Azure Key Vault service, your code uses the access token that it gets from the local MSI endpoint.

Log in to Azure

To log in to Azure by using the Azure CLI, enter:

```
az login
```

Create a resource group

An Azure resource group is a logical container into which Azure resources are deployed and managed.

Create a resource group by using the `az group create` command.

Select a resource group name and fill in the placeholder. The following example creates a resource group in the West US location:

```
# To list locations: az account list-locations --output table
az group create --name "<YourResourceGroupName>" --location "West US"
```

You use your newly created resource group throughout this tutorial.

Create a key vault

To create a key vault in the resource group that you created in the preceding step, provide the following information:

- Key vault name: a string of 3 to 24 characters that can contain only numbers (0-9), letters (a-z, A-Z), and hyphens (-)
- Resource group name
- Location: **West US**

```
az keyvault create --name "<YourKeyVaultName>" --resource-group "<YourResourceGroupName>" --location "West US"
```

At this point, your Azure account is the only one that's authorized to perform operations on this new key vault.

Add a secret to the key vault

We're adding a secret to help illustrate how this works. The secret might be a SQL connection string or any other information that you need to keep both secure and available to your application.

To create a secret in the key vault called **AppSecret**, enter the following command:

```
az keyvault secret set --vault-name "<YourKeyVaultName>" --name "AppSecret" --value "MySecret"
```

This secret stores the value **MySecret**.

Create a virtual machine

You can create a virtual machine by using one of the following methods:

- [The Azure CLI](#)
- [PowerShell](#)
- [The Azure portal](#)

Assign an identity to the VM

In this step, you create a system-assigned identity for the virtual machine by running the following command in the Azure CLI:

```
az vm identity assign --name <NameOfYourVirtualMachine> --resource-group <YourResourceGroupName>
```

Note the system-assigned identity that's displayed in the following code. The output of the preceding command would be:

```
{
  "systemAssignedIdentity": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "userAssignedIdentities": {}
}
```

Assign permissions to the VM identity

Now you can assign the previously created identity permissions to your key vault by running the following command:

```
az keyvault set-policy --name '<YourKeyVaultName>' --object-id <VMSystemAssignedIdentity> --secret-permissions get list
```

Log on to the virtual machine

To log on to the virtual machine, follow the instructions in [Connect and log on to an Azure virtual machine running Windows](#).

Create and run a sample Python app

In the next section is an example file named *Sample.py*. It uses a [requests](#) library to make HTTP GET calls.

Edit Sample.py

After you create *Sample.py*, open the file, and then copy the code in this section.

The code presents a two-step process:

1. Fetch a token from the local MSI endpoint on the VM.
Doing so also fetches a token from Azure AD.
2. Pass the token to your key vault, and then fetch your secret.

```
# importing the requests library
import requests

# Step 1: Fetch an access token from a Managed Identity enabled azure resource.
# Note that the resource here is https://vault.azure.net for public cloud and api-version is 2018-02-01
MSI_ENDPOINT = "http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net"
r = requests.get(MSI_ENDPOINT, headers = {"Metadata" : "true"})

# extracting data in json format
# This request gets an access_token from Azure AD by using the local MSI endpoint.
data = r.json()

# Step 2: Pass the access_token received from previous HTTP GET call to your key vault.
KeyVaultURL = "https://prashanthwinvmvault.vault.azure.net/secrets/RandomSecret?api-version=2016-10-01"
kvSecret = requests.get(url = KeyVaultURL, headers = {"Authorization": "Bearer " + data["access_token"]})

print(kvSecret.json()["value"])
```

You can display the secret value by running the following code:

```
python Sample.py
```

The preceding code shows you how to do operations with Azure Key Vault in a Windows virtual machine.

Clean up resources

When they are no longer needed, delete the virtual machine and your key vault.

Next steps

[Azure Key Vault REST API](#)

Manage storage account keys with Key Vault and the Azure CLI

6 minutes to read • [Edit Online](#)

An Azure storage account uses credentials comprising an account name and a key. The key is auto-generated and serves as a password, rather than an as a cryptographic key. Key Vault manages storage account keys by storing them as [Key Vault secrets](#).

You can use the Key Vault managed storage account key feature to list (sync) keys with an Azure storage account, and regenerate (rotate) the keys periodically. You can manage keys for both storage accounts and Classic storage accounts.

When you use the managed storage account key feature, consider the following points:

- Key values are never returned in response to a caller.
- Only Key Vault should manage your storage account keys. Don't manage the keys yourself and avoid interfering with Key Vault processes.
- Only a single Key Vault object should manage storage account keys. Don't allow key management from multiple objects.
- You can request Key Vault to manage your storage account with a user principal, but not with a service principal.
- Regenerate keys by using Key Vault only. Don't manually regenerate your storage account keys.

We recommend using Azure Storage integration with Azure Active Directory (Azure AD), Microsoft's cloud-based identity and access management service. Azure AD integration is available for [Azure blobs and queues](#), and provides OAuth2 token-based access to Azure Storage (just like Azure Key Vault).

Azure AD allows you to authenticate your client application by using an application or user identity, instead of storage account credentials. You can use an [Azure AD managed identity](#) when you run on Azure. Managed identities remove the need for client authentication and storing credentials in or with your application.

Azure AD uses role-based access control (RBAC) to manage authorization, which is also supported by Key Vault.

Service principal application ID

An Azure AD tenant provides each registered application with a [service principal](#). The service principal serves as the Application ID, which is used during authorization setup for access to other Azure resources via RBAC.

Key Vault is a Microsoft application that's pre-registered in all Azure AD tenants. Key Vault is registered under the same Application ID in each Azure cloud.

TENANTS	CLOUD	APPLICATION ID
Azure AD	Azure Government	7e7c393b-45d0-48b1-a35e-2905ddf8183c
Azure AD	Azure public	cfa8b339-82a2-471a-a3c9-0fc0be7a4093
Other	Any	cfa8b339-82a2-471a-a3c9-0fc0be7a4093

Prerequisites

To complete this guide, you must first do the following:

- [Install the Azure CLI](#).
- [Create a key vault](#)
- [Create an Azure storage account](#). The storage account name must use only lowercase letters and numbers. The length of the name must be between 3 and 24 characters.

Manage storage account keys

Connect to your Azure account

Authenticate your Azure CLI session using the `az login` commands.

```
az login
```

Give Key Vault access to your storage account

Use the Azure CLI `az role assignment create` command to give Key Vault access your storage account. Provide the command the following parameter values:

- `--role` : Pass the "Storage Account Key Operator Service Role" RBAC role. This role limits the access scope to your storage account. For a

classic storage account, pass "Classic Storage Account Key Operator Service Role" instead.

- `--assignee-object-id` : Pass the value "93c27d83-f79b-4cb2-8dd4-4aa716542e74", which is the Object ID for Key Vault in the Azure public cloud. (To get the Object ID for Key Vault in the Azure Government cloud, see [Service principal application ID](#).)
- `--scope` : Pass your storage account resource ID, which is in the form
`/subscriptions/<subscriptionID>/resourceGroups/<StorageAccountResourceGroupName>/providers/Microsoft.Storage/storageAccounts/<YourStorageAccountName>`. To find your subscription ID, use the Azure CLI [az account list](#) command; to find your storage account name and storage account resource group, use the Azure CLI [az storage account list](#) command.

```
az role assignment create --role "Storage Account Key Operator Service Role" --assignee-object-id 93c27d83-f79b-4cb2-8dd4-4aa716542e74 --scope  
"/subscriptions/<subscriptionID>/resourceGroups/<StorageAccountResourceGroupName>/providers/Microsoft.Storage/storageAccounts/<YourStorageAccountName>"
```

Create a Key Vault Managed storage account

Create a Key Vault managed storage account using the Azure CLI [az keyvault storage](#) command. Set a regeneration period of 90 days. After 90 days, Key Vault regenerates `key1` and swaps the active key from `key2` to `key1`. `key1` is then marked as the active key. Provide the command the following parameter values:

- `--vault-name` : Pass the name of your key vault. To find the name of your key vault, use the Azure CLI [az keyvault list](#) command.
- `-n` : Pass the name of your storage account. To find the name of your storage account, use the Azure CLI [az storage account list](#) command.
- `--resource-id` : Pass your storage account resource ID, which is in the form
`/subscriptions/<subscriptionID>/resourceGroups/<StorageAccountResourceGroupName>/providers/Microsoft.Storage/storageAccounts/<YourStorageAccountName>`. To find your subscription ID, use the Azure CLI [az account list](#) command; to find your storage account name and storage account resource group, use the Azure CLI [az storage account list](#) command.

```
az keyvault storage add --vault-name <YourKeyVaultName> -n <YourStorageAccountName> --active-key-name key1 --auto-regenerate-key --regeneration-period P90D --resource-id  
"/subscriptions/<subscriptionID>/resourceGroups/<StorageAccountResourceGroupName>/providers/Microsoft.Storage/storageAccounts/<YourStorageAccountName>"
```

Shared access signature tokens

You can also ask Key Vault to generate shared access signature tokens. A shared access signature provides delegated access to resources in your storage account. You can grant clients access to resources in your storage account without sharing your account keys. A shared access signature provides you with a secure way to share your storage resources without compromising your account keys.

The commands in this section complete the following actions:

- Set an account shared access signature definition `<YourSASDefinitionName>`. The definition is set on a Key Vault managed storage account `<YourStorageAccountName>` in your key vault `<YourKeyVaultName>`.
- Create an account shared access signature token for Blob, File, Table, and Queue services. The token is created for resource types Service, Container, and Object. The token is created with all permissions, over https, and with the specified start and end dates.
- Set a Key Vault managed storage shared access signature definition in the vault. The definition has the template URI of the shared access signature token that was created. The definition has the shared access signature type `account` and is valid for N days.
- Verify that the shared access signature was saved in your key vault as a secret.

Create a shared access signature token

Create a shared access signature definition using the Azure CLI [az storage account generate-sas](#) command. This operation requires the `storage` and `setsas` permissions.

```
az storage account generate-sas --expiry 2020-01-01 --permissions rw --resource-types sco --services bfqt --https-only --account-name  
<YourStorageAccountName> --account-key 00000000
```

After the operation runs successfully, copy the output.

```
"se=2020-01-01&sp=***"
```

This output will be the passed to the `--template-id` parameter in the next step.

Generate a shared access signature definition

Use the the Azure CLI [az keyvault storage sas-definition create](#) command, passing the output from the previous step to the `--template-id` parameter, to create a shared access signature definition. You can provide the name of your choice to the `-n` parameter.

```
az keyvault storage sas-definition create --vault-name <YourKeyVaultName> --account-name <YourStorageAccountName> -n <YourSASDefinitionName> --validity-period P2D --sas-type account --template-uri <OutputOfSasTokenCreationStep>
```

Verify the shared access signature definition

You can verify that the shared access signature definition has been stored in your key vault using the Azure CLI [az keyvault secret list](#) and [az keyvault secret show](#) commands.

First, find the shared access signature definition in your key vault using the [az keyvault secret list](#) command.

```
az keyvault secret list --vault-name <YourKeyVaultName>
```

The secret corresponding to your SAS definition will have these properties:

```
"contentType": "application/vnd.ms-sastoken-storage",
"id": "https://<YourKeyVaultName>.vault.azure.net/secrets/<YourStorageAccountName>-<YourSASDefinitionName>>,
```

You can now use the [az keyvault secret show](#) command and the `id` property to view the content of that secret.

```
az keyvault secret show --vault-name <YourKeyVaultName> --id <SasDefinitionID>
```

The output of this command will show your SAS definition string as `value`.

Next steps

- Learn more about [keys, secrets, and certificates](#).
- Review articles on the [Azure Key Vault team blog](#).
- See the [az keyvault storage](#) reference documentation.

Manage storage account keys with Key Vault and Azure PowerShell

7 minutes to read • [Edit Online](#)

An Azure storage account uses credentials comprising an account name and a key. The key is autogenerated and serves as a password, rather than an as a cryptographic key. Key Vault manages storage account keys by storing them as [Key Vault secrets](#).

You can use the Key Vault managed storage account key feature to list (sync) keys with an Azure storage account, and regenerate (rotate) the keys periodically. You can manage keys for both storage accounts and Classic storage accounts.

When you use the managed storage account key feature, consider the following points:

- Key values are never returned in response to a caller.
- Only Key Vault should manage your storage account keys. Don't manage the keys yourself and avoid interfering with Key Vault processes.
- Only a single Key Vault object should manage storage account keys. Don't allow key management from multiple objects.
- You can request Key Vault to manage your storage account with a user principal, but not with a service principal.
- Regenerate keys by using Key Vault only. Don't manually regenerate your storage account keys.

We recommend using Azure Storage integration with Azure Active Directory (Azure AD), Microsoft's cloud-based identity and access management service. Azure AD integration is available for [Azure blobs and queues](#), and provides OAuth2 token-based access to Azure Storage (just like Azure Key Vault).

Azure AD allows you to authenticate your client application by using an application or user identity, instead of storage account credentials. You can use an [Azure AD managed identity](#) when you run on Azure. Managed identities remove the need for client authentication and storing credentials in or with your application.

Azure AD uses role-based access control (RBAC) to manage authorization, which is also supported by Key Vault.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Service principal application ID

An Azure AD tenant provides each registered application with a [service principal](#). The service principal serves as the application ID, which is used during authorization setup for access to other Azure resources via RBAC.

Key Vault is a Microsoft application that's pre-registered in all Azure AD tenants. Key Vault is registered under the same Application ID in each Azure cloud.

TENANTS	CLOUD	APPLICATION ID
---------	-------	----------------

TENANTS	CLOUD	APPLICATION ID
Azure AD	Azure Government	7e7c393b-45d0-48b1-a35e-2905ddf8183c
Azure AD	Azure public	cfa8b339-82a2-471a-a3c9-0fc0be7a4093
Other	Any	cfa8b339-82a2-471a-a3c9-0fc0be7a4093

Prerequisites

To complete this guide, you must first do the following:

- [Install the Azure PowerShell module.](#)
- [Create a key vault](#)
- [Create an Azure storage account](#). The storage account name must use only lowercase letters and numbers. The length of the name must be between 3 and 24 characters.

Manage storage account keys

Connect to your Azure account

Authenticate your PowerShell session using the [Connect-AzAccount](#) cmdlet.

```
Connect-AzAccount
```

If you have multiple Azure subscriptions, you can list them using the [Get-AzSubscription](#) cmdlet, and specify the subscription you wish to use with the [Set-AzContext](#) cmdlet.

```
Set-AzContext -SubscriptionId <subscriptionId>
```

Set variables

First, set the variables to be used by the PowerShell cmdlets in the following steps. Be sure to update the , , and placeholders, and set \$keyVaultSpAppId to `cfa8b339-82a2-471a-a3c9-0fc0be7a4093` (as specified in [Service principal application ID](#), above).

We will also use the Azure PowerShell [Get-AzContext](#) and [Get-AzStorageAccount](#) cmdlets to get your user ID and the context of your Azure storage account.

```
$resourceGroupName = <YourResourceGroupName>
$storageAccountName = <YourStorageAccountName>
$keyVaultName = <YourKeyVaultName>
$keyVaultSpAppId = "cfa8b339-82a2-471a-a3c9-0fc0be7a4093"
$storageAccountKey = "key1"

# Get your User Id
$userId = (Get-AzContext).Account.Id

# Get a reference to your Azure storage account
$storageAccount = Get-AzStorageAccount -ResourceGroupName $resourceGroupName -StorageAccountName
$storageAccountName
```

Give Key Vault access to your storage account

Before Key Vault can access and manage your storage account keys, you must authorize its access to your storage account. The Key Vault application requires permissions to *list* and *regenerate* keys for your storage account. These permissions are enabled through the built-in RBAC role [Storage Account Key Operator Service Role](#).

Assign this role to the Key Vault service principal, limiting scope to your storage account, using the Azure PowerShell [New-AzRoleAssignment](#) cmdlet.

```
# Assign RBAC role "Storage Account Key Operator Service Role" to Key Vault, limiting the access scope to your storage account. For a classic storage account, use "Classic Storage Account Key Operator Service Role."
New-AzRoleAssignment -ApplicationId $keyVaultSpAppId -RoleDefinitionName 'Storage Account Key Operator Service Role' -Scope $storageAccount.Id
```

Upon successful role assignment, you should see output similar to the following example:

```
RoleAssignmentId      : /subscriptions/03f0b111-ce69-483a-a092-d06ea46dfb8z/resourceGroups/rgContoso/providers/Microsoft.Storage/storageAccounts/sacontoso/providers/Microsoft.Authorization/roleAssignments/189cb111-12fb-406e-8699-4eef8b2b9ecz
Scope                 : /subscriptions/03f0b111-ce69-483a-a092-d06ea46dfb8z/resourceGroups/rgContoso/providers/Microsoft.Storage/storageAccounts/sacontoso
DisplayName          : Azure Key Vault
SignInName           :
RoleDefinitionName   : storage account Key Operator Service Role
RoleDefinitionId     : 81a9662b-bebf-436f-a333-f67b29880f12
ObjectId              : 93c27d83-f79b-4cb2-8dd4-4aa716542e74
ObjectType            : ServicePrincipal
CanDelegate           : False
```

If Key Vault has already been added to the role on your storage account, you'll receive a "*The role assignment already exists.*" error. You can also verify the role assignment, using the storage account "Access control (IAM)" page in the Azure portal.

Give your user account permission to managed storage accounts

Use the Azure PowerShell [Set-AzKeyVaultAccessPolicy](#) cmdlet to update the Key Vault access policy and grant storage account permissions to your user account.

```
# Give your user principal access to all storage account permissions, on your Key Vault instance
Set-AzKeyVaultAccessPolicy -VaultName $keyVaultName -UserPrincipalName $userId -PermissionsToStorage get, list, delete, set, update, regeneratekey, getsas, listsas, deletesas, setsas, recover, backup, restore, purge
```

Note that permissions for storage accounts aren't available on the storage account "Access policies" page in the Azure portal.

Add a managed storage account to your Key Vault instance

Use the Azure PowerShell [Add-AzKeyVaultManagedStorageAccount](#) cmdlet to create a managed storage account in your Key Vault instance. The `-DisableAutoRegenerateKey` switch specifies NOT to regenerate the storage account keys.

```
# Add your storage account to your Key Vault's managed storage accounts
Add-AzKeyVaultManagedStorageAccount -VaultName $keyVaultName -AccountName $storageAccountName -AccountResourceId $storageAccount.Id -ActiveKeyName $storageAccountKey -DisableAutoRegenerateKey
```

Upon successful addition of the storage account with no key regeneration, you should see output similar to the following example:

```

Id : https://kvcontoso.vault.azure.net:443/storage/sacontoso
Vault Name : kvcontoso
AccountName : sacontoso
Account Resource Id : /subscriptions/03f0b111-ce69-483a-a092-
d06ea46dfb8z/resourceGroups/rgContoso/providers/Microsoft.Storage/storageAccounts/sacontoso
Active Key Name : key1
Auto Regenerate Key : False
Regeneration Period : 90.00:00:00
Enabled : True
Created : 11/19/2018 11:54:47 PM
Updated : 11/19/2018 11:54:47 PM
Tags :

```

Enable key regeneration

If you want Key Vault to regenerate your storage account keys periodically, you can use the Azure PowerShell [Add-AzKeyVaultManagedStorageAccount](#) cmdlet to set a regeneration period. In this example, we set a regeneration period of three days. After three days, Key Vault will regenerate 'key2' and swap the active key from 'key2' to 'key1'.

```

$regenPeriod = [System.Timespan]::FromDays(3)

Add-AzKeyVaultManagedStorageAccount -VaultName $keyVaultName -AccountName $storageAccountName -
AccountResourceId $storageAccount.Id -ActiveKeyName $storageAccountKey -RegenerationPeriod $regenPeriod

```

Upon successful addition of the storage account with key regeneration, you should see output similar to the following example:

```

Id : https://kvcontoso.vault.azure.net:443/storage/sacontoso
Vault Name : kvcontoso
AccountName : sacontoso
Account Resource Id : /subscriptions/03f0b111-ce69-483a-a092-
d06ea46dfb8z/resourceGroups/rgContoso/providers/Microsoft.Storage/storageAccounts/sacontoso
Active Key Name : key1
Auto Regenerate Key : True
Regeneration Period : 3.00:00:00
Enabled : True
Created : 11/19/2018 11:54:47 PM
Updated : 11/19/2018 11:54:47 PM
Tags :

```

Shared access signature tokens

You can also ask Key Vault to generate shared access signature tokens. A shared access signature provides delegated access to resources in your storage account. You can grant clients access to resources in your storage account without sharing your account keys. A shared access signature provides you with a secure way to share your storage resources without compromising your account keys.

The commands in this section complete the following actions:

- Set an account shared access signature definition.
- Create an account shared access signature token for Blob, File, Table, and Queue services. The token is created for resource types Service, Container, and Object. The token is created with all permissions, over https, and with the specified start and end dates.
- Set a Key Vault managed storage shared access signature definition in the vault. The definition has the template URI of the shared access signature token that was created. The definition has the shared access signature type `account` and is valid for N days.

- Verify that the shared access signature was saved in your key vault as a secret.
-

Set variables

First, set the variables to be used by the PowerShell cmdlets in the following steps. Be sure to update the and placeholders.

We will also use the Azure PowerShell [New-AzStorageContext](#) cmdlets to get the context of your Azure storage account.

```
$storageAccountName = <YourStorageAccountName>
$keyVaultName = <YourKeyVaultName>

$storageContext = New-AzStorageContext -StorageAccountName $storageAccountName -Protocol Https -
StorageAccountKey Key1
```

Create a shared access signature token

Create a shared access signature definition using the Azure PowerShell [New-AzStorageAccountSASToken](#) cmdlets.

```
$start = [System.DateTime]::Now.AddDays(-1)
$end = [System.DateTime]::Now.AddMonths(1)

$sasToken = New-AzStorageAccountSasToken -Service blob,file,Table,Queue -ResourceType Service,Container,Object
-Permission "racwdlup" -ProtocolHttpsOnly -StartTime $start -ExpiryTime $end -Context $storageContext
```

The value of \$sasToken will look similar to this.

```
?sv=2018-11-09&sig=5GWqHFkE0tM7W9al0goXSCOJ0%2B55qJr4J7tHQjCI9S%3D&spr=https&st=2019-09-
18T18%3A25%3A00Z&se=2019-10-19T18%3A25%3A00Z&srt=sco&ss=bfqt&sp=racupwd1
```

Generate a shared access signature definition

Use the the Azure PowerShell [Set-AzKeyVaultManagedStorageSasDefinition](#) cmdlet to create a shared access signature definition. You can provide the name of your choice to the `-Name` parameter.

```
Set-AzKeyVaultManagedStorageSasDefinition -AccountName $storageAccountName -VaultName $keyVaultName -Name
<YourSASDefinitionName> -TemplateUri $sasToken -SasType 'account' -ValidityPeriod
([System.Timespan]::FromDays(30))
```

Verify the shared access signature definition

You can verify that the shared access signature definition has been stored in your key vault using the Azure PowerShell [Get-AzKeyVaultSecret](#) cmdlet.

First, find the shared access signature definition in your key vault.

```
Get-AzKeyVaultSecret -VaultName <YourKeyVaultName>
```

The secret corresponding to your SAS definition will have these properties:

```
Vault Name : <YourKeyVaultName>
Name        : <SecretName>
...
Content Type : application/vnd.ms-sastoken-storage
Tags         :
```

You can now use the [Get-AzKeyVaultSecret](#) cmdlet and the secret `Name` property to view the content of that secret.

```
$secret = Get-AzKeyVaultSecret -VaultName <YourKeyVaultName> -Name <SecretName>
Write-Host $secret.SecretValueText
```

The output of this command will show your SAS definition string.

Next steps

- [Managed storage account key samples](#)
- [About keys, secrets, and certificates](#)
- [Key Vault PowerShell reference](#)

Fetch shared access signature tokens in code

2 minutes to read • [Edit Online](#)

You can manage your storage account with the [shared access signature tokens](#) in your key vault. This article provides examples of C# code that fetches a SAS token and performs operations with it. For information on how to create and store SAS tokens, see [Manage storage account keys with Key Vault and the Azure CLI](#) or [Manage storage account keys with Key Vault and Azure PowerShell](#).

Code samples

In this example, the code fetches a SAS token from your key vault, uses it to create a new storage account, and creates a new Blob service client.

```
// After you get a security token, create KeyVaultClient with vault credentials.  
var kv = new KeyVaultClient(new KeyVaultClient.AuthenticationCallback(securityToken));  
  
// Get a shared access signature token for your storage from Key Vault.  
// The format for SecretUri is https://<YourKeyVaultName>.vault.azure.net/secrets/<ExamplePassword>  
var sasToken = await kv.GetSecretAsync("SecretUri");  
  
// Create new storage credentials by using the shared access signature token.  
var accountSasCredential = new StorageCredentials(sasToken.Value);  
  
// Use the storage credentials and the Blob storage endpoint to create a new Blob service client.  
var accountWithSas = new CloudStorageAccount(accountSasCredential, new Uri  
("https://myaccount.blob.core.windows.net/"), null, null, null);  
  
var blobClientWithSas = accountWithSas.CreateCloudBlobClient();
```

If your shared access signature token is about to expire, you can fetch the shared access signature token from your key vault and update the code.

```
// If your shared access signature token is about to expire,  
// get the shared access signature token again from Key Vault and update it.  
sasToken = await kv.GetSecretAsync("SecretUri");  
accountSasCredential.UpdateSASToken(sasToken);
```

Next steps

- Learn how to [Manage storage account keys with Key Vault and the Azure CLI](#) or [Azure PowerShell](#).
- See [Managed storage account key samples](#)
- [About keys, secrets, and certificates](#)
- [Key Vault PowerShell reference](#)

How to use managed identities with Azure Container Instances

11 minutes to read • [Edit Online](#)

Use [managed identities for Azure resources](#) to run code in Azure Container Instances that interacts with other Azure services - without maintaining any secrets or credentials in code. The feature provides an Azure Container Instances deployment with an automatically managed identity in Azure Active Directory.

In this article, you learn more about managed identities in Azure Container Instances and:

- Enable a user-assigned or system-assigned identity in a container group
- Grant the identity access to an Azure key vault
- Use the managed identity to access a key vault from a running container

Adapt the examples to enable and use identities in Azure Container Instances to access other Azure services. These examples are interactive. However, in practice your container images would run code to access Azure services.

NOTE

Currently you cannot use a managed identity in a container group deployed to a virtual network.

Why use a managed identity?

Use a managed identity in a running container to authenticate to any [service that supports Azure AD authentication](#) without managing credentials in your container code. For services that don't support AD authentication, you can store secrets in an Azure key vault and use the managed identity to access the key vault to retrieve credentials. For more information about using a managed identity, see [What is managed identities for Azure resources?](#)

IMPORTANT

This feature is currently in preview. Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA). Currently, managed identities on Azure Container Instances, are only supported with Linux containers and not yet with Windows containers.

Enable a managed identity

In Azure Container Instances, managed identities for Azure resources are supported as of REST API version 2018-10-01 and corresponding SDKs and tools. When you create a container group, enable one or more managed identities by setting a [ContainerGroupIdentity](#) property. You can also enable or update managed identities after a container group is running - either action causes the container group to restart. To set the identities on a new or existing container group, use the Azure CLI, a Resource Manager template, or a YAML file.

Azure Container Instances supports both types of managed Azure identities: user-assigned and system-assigned. On a container group, you can enable a system-assigned identity, one or more user-assigned identities, or both types of identities.

- A **user-assigned** managed identity is created as a standalone Azure resource in the Azure AD tenant that's trusted by the subscription in use. After the identity is created, the identity can be assigned to one or more Azure resources (in Azure Container Instances or other Azure services). The lifecycle of a user-assigned

identity is managed separately from the lifecycle of the container groups or other service resources to which it's assigned. This behavior is especially useful in Azure Container Instances. Because the identity extends beyond the lifetime of a container group, you can reuse it along with other standard settings to make your container group deployments highly repeatable.

- A **system-assigned** managed identity is enabled directly on a container group in Azure Container Instances. When it's enabled, Azure creates an identity for the group in the Azure AD tenant that's trusted by the subscription of the instance. After the identity is created, the credentials are provisioned in each container in the container group. The lifecycle of a system-assigned identity is directly tied to the container group that it's enabled on. When the group is deleted, Azure automatically cleans up the credentials and the identity in Azure AD.

Use a managed identity

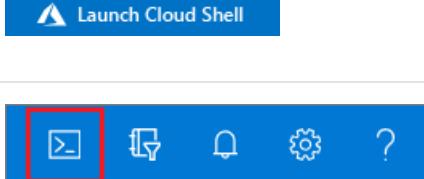
To use a managed identity, the identity must initially be granted access to one or more Azure service resources (such as a web app, a key vault, or a storage account) in the subscription. To access the Azure resources from a running container, your code must acquire an *access token* from an Azure AD endpoint. Then, your code sends the access token on a call to a service that supports Azure AD authentication.

Using a managed identity in a running container is essentially the same as using an identity in an Azure VM. See the VM guidance for using a [token](#), [Azure PowerShell](#) or [Azure CLI](#), or the [Azure SDKs](#).

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the CLI locally, this article requires that you are running the Azure CLI version 2.0.49 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create an Azure key vault

The examples in this article use a managed identity in Azure Container Instances to access an Azure key vault secret.

First, create a resource group named *myResourceGroup* in the *eastus* location with the following [az group create](#) command:

```
az group create --name myResourceGroup --location eastus
```

Use the [az keyvault create](#) command to create a key vault. Be sure to specify a unique key vault name.

```
az keyvault create \
--name mykeyvault \
--resource-group myResourceGroup \
--location eastus
```

Store a sample secret in the key vault using the [az keyvault secret set](#) command:

```
az keyvault secret set \
--name SampleSecret \
--value "Hello Container Instances" \
--description ACIsecret --vault-name mykeyvault
```

Continue with the following examples to access the key vault using either a user-assigned or system-assigned managed identity in Azure Container Instances.

Example 1: Use a user-assigned identity to access Azure key vault

Create an identity

First create an identity in your subscription using the [az identity create](#) command. You can use the same resource group used to create the key vault, or use a different one.

```
az identity create \
--resource-group myResourceGroup \
--name myACIID
```

To use the identity in the following steps, use the [az identity show](#) command to store the identity's service principal ID and resource ID in variables.

```
# Get service principal ID of the user-assigned identity
spID=$(az identity show --resource-group myResourceGroup --name myACIID --query principalId --output tsv)

# Get resource ID of the user-assigned identity
resourceID=$(az identity show --resource-group myResourceGroup --name myACIID --query id --output tsv)
```

Enable a user-assigned identity on a container group

Run the following [az container create](#) command to create a container instance based on Microsoft's [azure-cli](#) image. This example provides a single-container group that you can use interactively to run the Azure CLI to access other Azure services. In this section, only the base Ubuntu operating system is used.

The `--assign-identity` parameter passes your user-assigned managed identity to the group. The long-running command keeps the container running. This example uses the same resource group used to create the key vault, but you could specify a different one.

```
az container create \
--resource-group myResourceGroup \
--name mycontainer \
--image mcr.microsoft.com/azure-cli \
--assign-identity $resourceID \
--command-line "tail -f /dev/null"
```

Within a few seconds, you should get a response from the Azure CLI indicating that the deployment has completed. Check its status with the [az container show](#) command.

```
az container show \
--resource-group myResourceGroup \
--name mycontainer
```

The `identity` section in the output looks similar to the following, showing the identity is set in the container group. The `principalID` under `userAssignedIdentities` is the service principal of the identity you created in Azure Active Directory:

```
[...]
"identity": {
    "principalId": "null",
    "tenantId": "xxxxxxxx-f292-4e60-9122-xxxxxxxxxxxx",
    "type": "UserAssigned",
    "userAssignedIdentities": {
        "/subscriptions/xxxxxxxx-0903-4b79-a55a-
xxxxxxxxxxxx/resourcegroups/danlepi1018/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myACIID": {
            "clientId": "xxxxxxxx-5523-45fc-9f49-xxxxxxxxxxxx",
            "principalId": "xxxxxxxx-f25b-4895-b828-xxxxxxxxxxxx"
        }
    }
},
[...]
```

Grant user-assigned identity access to the key vault

Run the following [az keyvault set-policy](#) command to set an access policy on the key vault. The following example allows the user-assigned identity to get secrets from the key vault:

```
az keyvault set-policy \
--name mykeyvault \
--resource-group myResourceGroup \
--object-id $spID \
--secret-permissions get
```

Use user-assigned identity to get secret from key vault

Now you can use the managed identity within the running container instance to access the key vault. First launch a bash shell in the container:

```
az container exec \
--resource-group myResourceGroup \
--name mycontainer \
--exec-command "/bin/bash"
```

Run the following commands in the bash shell in the container. To get an access token to use Azure Active Directory to authenticate to key vault, run the following command:

```
curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net' -H Metadata:true -s
```

Output:

```
{"access_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N5SEpsWSIsImtpZCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N5SEpsWSJ9.....xxxxxxxxxxxxxx", "refresh_token": "", "expires_in": "28799", "expires_on": "1539927532", "not_before": "1539898432", "resource": "https://vault.azure.net/", "token_type": "Bearer"}
```

To store the access token in a variable to use in subsequent commands to authenticate, run the following command:

```
token=$(curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net' -H Metadata:true | jq -r '.access_token')
```

Now use the access token to authenticate to key vault and read a secret. Be sure to substitute the name of your key vault in the URL ([https://mykeyvault.vault.azure.net/...](https://mykeyvault.vault.azure.net/)):

```
curl https://mykeyvault.vault.azure.net/secrets/SampleSecret/?api-version=2016-10-01 -H "Authorization: Bearer $token"
```

The response looks similar to the following, showing the secret. In your code, you would parse this output to obtain the secret. Then, use the secret in a subsequent operation to access another Azure resource.

```
{"value": "Hello Container Instances", "contentType": "ACIsecret", "id": "https://mykeyvault.vault.azure.net/secrets/SampleSecret/xxxxxxxxxxxx", "attributes": {"enabled": true, "created": 1539965967, "updated": 1539965967, "recoveryLevel": "Purgeable"}, "tags": {"file-encoding": "utf-8"}}
```

Example 2: Use a system-assigned identity to access Azure key vault

Enable a system-assigned identity on a container group

Run the following `az container create` command to create a container instance based on Microsoft's `azure-cli` image. This example provides a single-container group that you can use interactively to run the Azure CLI to access other Azure services.

The `--assign-identity` parameter with no additional value enables a system-assigned managed identity on the group. The identity is scoped to the resource group of the container group. The long-running command keeps the container running. This example uses the same resource group used to create the key vault, but you could specify a different one.

```
# Get the resource ID of the resource group
rgID=$(az group show --name myResourceGroup --query id --output tsv)

# Create container group with system-managed identity
az container create \
    --resource-group myResourceGroup \
    --name mycontainer \
    --image mcr.microsoft.com/azure-cli \
    --assign-identity --scope $rgID \
    --command-line "tail -f /dev/null"
```

Within a few seconds, you should get a response from the Azure CLI indicating that the deployment has completed. Check its status with the [az container show](#) command.

```
az container show \
    --resource-group myResourceGroup \
    --name mycontainer
```

The `identity` section in the output looks similar to the following, showing that a system-assigned identity is created in Azure Active Directory:

```
[...]
"identity": {
    "principalId": "xxxxxxxx-528d-7083-b74c-xxxxxxxxxxxx",
    "tenantId": "xxxxxxxx-f292-4e60-9122-xxxxxxxxxxxx",
    "type": "SystemAssigned",
    "userAssignedIdentities": null
},
[...]
```

Set a variable to the value of `principalId` (the service principal ID) of the identity, to use in later steps.

```
spID=$(az container show --resource-group myResourceGroup --name mycontainer --query identity.principalId --out tsv)
```

Grant container group access to the key vault

Run the following [az keyvault set-policy](#) command to set an access policy on the key vault. The following example allows the system-managed identity to get secrets from the key vault:

```
az keyvault set-policy \
    --name mykeyvault \
    --resource-group myResourceGroup \
    --object-id $spID \
    --secret-permissions get
```

Use container group identity to get secret from key vault

Now you can use the managed identity to access the key vault within the running container instance. First launch a bash shell in the container:

```
az container exec \
    --resource-group myResourceGroup \
    --name mycontainer \
    --exec-command "/bin/bash"
```

Run the following commands in the bash shell in the container. First log in to the Azure CLI using the managed

identity:

```
az login --identity
```

From the running container, retrieve the secret from the key vault:

```
az keyvault secret show \  
  --name SampleSecret \  
  --vault-name mykeyvault --query value
```

The value of the secret is retrieved:

```
"Hello Container Instances"
```

Enable managed identity using Resource Manager template

To enable a managed identity in a container group using a [Resource Manager template](#), set the `identity` property of the `Microsoft.ContainerInstance/containerGroups` object with a `ContainerGroupIdentity` object. The following snippets show the `identity` property configured for different scenarios. See the [Resource Manager template reference](#). Specify a minimum `apiVersion` of `2018-10-01`.

User-assigned identity

A user-assigned identity is a resource ID of the form:

```
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{identityName}"
```

You can enable one or more user-assigned identities.

```
"identity": {  
    "type": "UserAssigned",  
    "userAssignedIdentities": {  
        "myResourceId1": {}  
    }  
}
```

System-assigned identity

```
"identity": {  
    "type": "SystemAssigned"  
}
```

System- and user-assigned identities

On a container group, you can enable both a system-assigned identity and one or more user-assigned identities.

```
"identity": {  
    "type": "System Assigned, UserAssigned",  
    "userAssignedIdentities": {  
        "myResourceID1": {}  
    }  
}  
...  
}
```

Enable managed identity using YAML file

To enable a managed identity in a container group deployed using a [YAML file](#), include the following YAML. Specify a minimum `apiVersion` of `2018-10-01`.

User-assigned identity

A user-assigned identity is a resource ID of the form

```
'/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{identityName}'
```

You can enable one or more user-assigned identities.

```
identity:  
  type: UserAssigned  
  userAssignedIdentities:  
    {'myResourceID1':{}}
```

System-assigned identity

```
identity:  
  type: SystemAssigned
```

System- and user-assigned identities

On a container group, you can enable both a system-assigned identity and one or more user-assigned identities.

```
identity:  
  type: SystemAssigned, UserAssigned  
  userAssignedIdentities:  
    {'myResourceID1':{}}
```

Next steps

In this article, you learned about managed identities in Azure Container Instances and how to:

- Enable a user-assigned or system-assigned identity in a container group
- Grant the identity access to an Azure key vault
- Use the managed identity to access a key vault from a running container
- Learn more about [managed identities for Azure resources](#).
- See an [Azure Go SDK example](#) of using a managed identity to access a key vault from Azure Container Instances.

Use Azure Key Vault to pass secure parameter value during deployment

6 minutes to read • [Edit Online](#)

Instead of putting a secure value (like a password) directly in your template or parameter file, you can retrieve the value from an [Azure Key Vault](#) during a deployment. You retrieve the value by referencing the key vault and secret in your parameter file. The value is never exposed because you only reference its key vault ID. The key vault can exist in a different subscription than the resource group you're deploying to.

This article focuses on the scenario of passing a sensitive value in as a template parameter. It doesn't cover the scenario of setting a virtual machine property to the URL of a certificate in a Key Vault. For a quickstart template of that scenario, see [Install a certificate from Azure Key Vault on a Virtual Machine](#).

Deploy key vaults and secrets

To access a key vault during template deployment, set `enabledForTemplateDeployment` on the key vault to `true`.

If you already have a Key Vault, make sure it allows template deployments.

- [Azure CLI](#)
- [PowerShell](#)

```
az keyvault update --name ExampleVault --enabled-for-template-deployment true
```

To create a new Key Vault and add a secret, use:

- [Azure CLI](#)
- [PowerShell](#)

```
az group create --name ExampleGroup --location centralus
az keyvault create \
  --name ExampleVault \
  --resource-group ExampleGroup \
  --location centralus \
  --enabled-for-template-deployment true
az keyvault secret set --vault-name ExampleVault --name "ExamplePassword" --value "hVFkk965BuUv"
```

As the owner of the key vault, you automatically have access to creating secrets. If the user working with secrets isn't the owner of the key vault, grant access with:

- [Azure CLI](#)
- [PowerShell](#)

```
az keyvault set-policy \
  --upn <user-principal-name> \
  --name ExampleVault \
  --secret-permissions set delete get list
```

For more information about creating key vaults and adding secrets, see:

- [Set and retrieve a secret by using CLI](#)

- Set and retrieve a secret by using Powershell
- Set and retrieve a secret by using the portal
- Set and retrieve a secret by using .NET
- Set and retrieve a secret by using Nodejs

Grant access to the secrets

The user who deploys the template must have the `Microsoft.KeyVault/vaults/deploy/action` permission for the scope of the resource group and key vault. The [Owner](#) and [Contributor](#) roles both grant this access. If you created the key vault, you're the owner so you have the permission.

The following procedure shows how to create a role with the minimum permission, and how to assign the user

1. Create a custom role definition JSON file:

```
{
  "Name": "Key Vault resource manager template deployment operator",
  "IsCustom": true,
  "Description": "Lets you deploy a resource manager template with the access to the secrets in the Key Vault.",
  "Actions": [
    "Microsoft.KeyVault/vaults/deploy/action"
  ],
  "NotActions": [],
  "DataActions": [],
  "NotDataActions": [],
  "AssignableScopes": [
    "/subscriptions/00000000-0000-0000-0000-000000000000"
  ]
}
```

Replace "00000000-0000-0000-0000-000000000000" with the subscription ID.

2. Create the new role using the JSON file:

- [Azure CLI](#)
- [PowerShell](#)

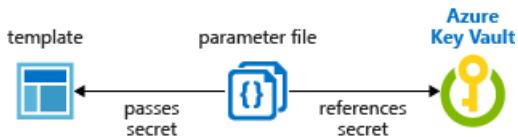
```
az role definition create --role-definition "<path-to-role-file>"
az role assignment create \
--role "Key Vault resource manager template deployment operator" \
--assignee <user-principal-name> \
--resource-group ExampleGroup
```

The samples assign the custom role to the user on the resource group level.

When using a Key Vault with the template for a [Managed Application](#), you must grant access to the **Appliance Resource Provider** service principal. For more information, see [Access Key Vault secret when deploying Azure Managed Applications](#).

Reference secrets with static ID

With this approach, you reference the key vault in the parameter file, not the template. The following image shows how the parameter file references the secret and passes that value to the template.



[Tutorial: Integrate Azure Key Vault in Resource Manager Template deployment](#) uses this method.

The following template deploys a SQL server that includes an administrator password. The password parameter is set to a secure string. But, the template doesn't specify where that value comes from.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "adminLogin": {
            "type": "string"
        },
        "adminPassword": {
            "type": "securestring"
        },
        "sqlServerName": {
            "type": "string"
        }
    },
    "resources": [
        {
            "type": "Microsoft.Sql/servers",
            "apiVersion": "2015-05-01-preview",
            "name": "[parameters('sqlServerName')]",
            "location": "[resourceGroup().location]",
            "tags": {},
            "properties": {
                "administratorLogin": "[parameters('adminLogin')]",
                "administratorLoginPassword": "[parameters('adminPassword')]",
                "version": "12.0"
            }
        }
    ],
    "outputs": {}
}
```

Now, create a parameter file for the preceding template. In the parameter file, specify a parameter that matches the name of the parameter in the template. For the parameter value, reference the secret from the key vault. You reference the secret by passing the resource identifier of the key vault and the name of the secret:

In the following parameter file, the key vault secret must already exist, and you provide a static value for its resource ID.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "adminLogin": {
      "value": "exampleleadadmin"
    },
    "adminPassword": {
      "reference": {
        "keyVault": {
          "id": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.KeyVault/vaults/<vault-name>"
        },
        "secretName": "ExamplePassword"
      }
    },
    "sqlServerName": {
      "value": "<your-server-name>"
    }
  }
}
```

If you need to use a version of the secret other than the current version, use the `secretVersion` property.

```
"secretName": "ExamplePassword",
"secretVersion": "cd91b2b7e10e492ebb870a6ee0591b68"
```

Deploy the template and pass in the parameter file:

- [Azure CLI](#)
- [PowerShell](#)

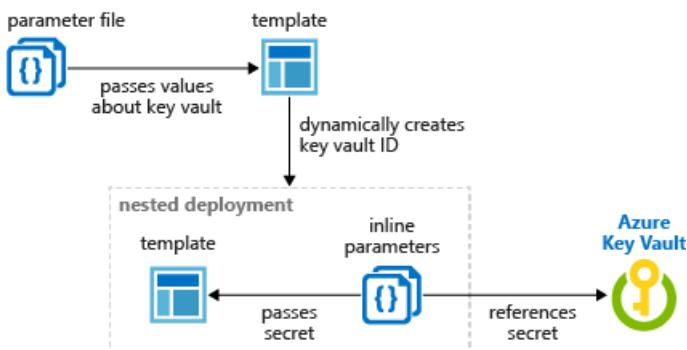
```
az group create --name SqlGroup --location westus2
az deployment group create \
  --resource-group SqlGroup \
  --template-uri <template-file-URI> \
  --parameters <parameter-file>
```

Reference secrets with dynamic ID

The previous section showed how to pass a static resource ID for the key vault secret from the parameter. However, in some scenarios, you need to reference a key vault secret that varies based on the current deployment. Or, you may want to pass parameter values to the template rather than create a reference parameter in the parameter file. In either case, you can dynamically generate the resource ID for a key vault secret by using a linked template.

You can't dynamically generate the resource ID in the parameters file because template expressions aren't allowed in the parameters file.

In your parent template, you add the nested template and pass in a parameter that contains the dynamically generated resource ID. The following image shows how a parameter in the linked template references the secret.



The following template dynamically creates the key vault ID and passes it as a parameter.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "The location where the resources will be deployed."
      }
    },
    "vaultName": {
      "type": "string",
      "metadata": {
        "description": "The name of the keyvault that contains the secret."
      }
    },
    "secretName": {
      "type": "string",
      "metadata": {
        "description": "The name of the secret."
      }
    },
    "vaultResourceGroupName": {
      "type": "string",
      "metadata": {
        "description": "The name of the resource group that contains the keyvault."
      }
    },
    "vaultSubscription": {
      "type": "string",
      "defaultValue": "[subscription().subscriptionId]",
      "metadata": {
        "description": "The name of the subscription that contains the keyvault."
      }
    }
  },
  "resources": [
    {
      "type": "Microsoft.Resources/deployments",
      "apiVersion": "2018-05-01",
      "name": "dynamicSecret",
      "properties": {
        "mode": "Incremental",
        "expressionEvaluationOptions": {
          "scope": "inner"
        },
        "template": {
          "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
          "contentVersion": "1.0.0.0",
          "parameters": {
            "adminLogin": {
              "type": "string"
            }
          }
        }
      }
    }
  ]
}
```

```

},
"adminPassword": {
    "type": "securestring"
},
"location": {
    "type": "string"
},
"variables": {
    "sqlServerName": "[concat('sql-', uniqueString(resourceGroup().id, 'sql'))]"
},
"resources": [
{
    "type": "Microsoft.Sql/servers",
    "apiVersion": "2018-06-01-preview",
    "name": "[variables('sqlServerName')]",
    "location": "[parameters('location')]",
    "properties": {
        "administratorLogin": "[parameters('adminLogin')]",
        "administratorLoginPassword": "[parameters('adminPassword')]"
    }
}
],
"outputs": {
    "sqlFQDN": {
        "type": "string",
        "value": "[reference(variables('sqlServerName')).fullyQualifiedDomainName]"
    }
}
},
"parameters": {
    "location": {
        "value": "[parameters('location')]"
    },
    "adminLogin": {
        "value": "ghuser"
    },
    "adminPassword": {
        "reference": {
            "keyVault": {
                "id": "[resourceId(parameters('vaultSubscription'), parameters('vaultResourceGroupName'), 'Microsoft.KeyVault/vaults', parameters('vaultName'))]"
            },
            "secretName": "[parameters('secretName')]"
        }
    }
}
],
"outputs": {
}
}

```

Next steps

- For general information about key vaults, see [What is Azure Key Vault?](#).
- For complete examples of referencing key secrets, see [Key Vault examples](#).

Set up Azure Key Vault with key rotation and auditing

14 minutes to read • [Edit Online](#)

Introduction

After you have a key vault, you can start using it to store keys and secrets. Your applications no longer need to persist your keys or secrets, but can request them from the vault as needed. A key vault allows you to update keys and secrets without affecting the behavior of your application, which opens up a breadth of possibilities for your key and secret management.

This article walks through how to implement a scheduled rotation of storage account keys, monitor the key vault audit logs, and raise alerts when unexpected requests are made.

You must first create a key vault using the method of your choice:

- [Set and retrieve a secret from Azure Key Vault using Azure CLI](#)
- [Set and retrieve a secret from Azure Key Vault using Azure PowerShell](#)
- [Set and retrieve a secret from Azure Key Vault using Azure portal](#)

Store a secret

To enable an application to retrieve a secret from Key Vault, you must first create the secret and upload it to your vault.

Start an Azure PowerShell session and sign in to your Azure account with the following command:

```
Connect-AzAccount
```

In the pop-up browser window, enter the username and password for your Azure account. PowerShell will get all the subscriptions that are associated with this account. PowerShell uses the first one by default.

If you have multiple subscriptions, you might have to specify the one that was used to create your key vault. Enter the following to see the subscriptions for your account:

```
Get-AzSubscription
```

To specify the subscription that's associated with the key vault you'll be logging, enter:

```
Set-AzContext -SubscriptionId <subscriptionID>
```

Because this article demonstrates storing a storage account key as a secret, you must get that storage account key.

```
Get-AzStorageAccountKey -ResourceGroupName <resourceGroupName> -Name <storageAccountName>
```

After retrieving your secret (in this case, your storage account key), you must convert that key to a secure string, and then create a secret with that value in your key vault.

```
$secretvalue = ConvertTo-SecureString <storageAccountKey> -AsPlainText -Force  
Set-AzKeyVaultSecret -VaultName <vaultName> -Name <secretName> -SecretValue $secretvalue
```

Next, get the URI for the secret you created. You'll need this URI in a later step to call the key vault and retrieve your secret. Run the following PowerShell command and make note of the ID value, which is the secret's URL:

```
Get-AzKeyVaultSecret -VaultName <vaultName>
```

Set up the application

Now that you have a secret stored, you can use code to retrieve and use it after performing a few more steps.

First, you must register your application with Azure Active Directory. Then tell Key Vault your application information so that it can allow requests from your application.

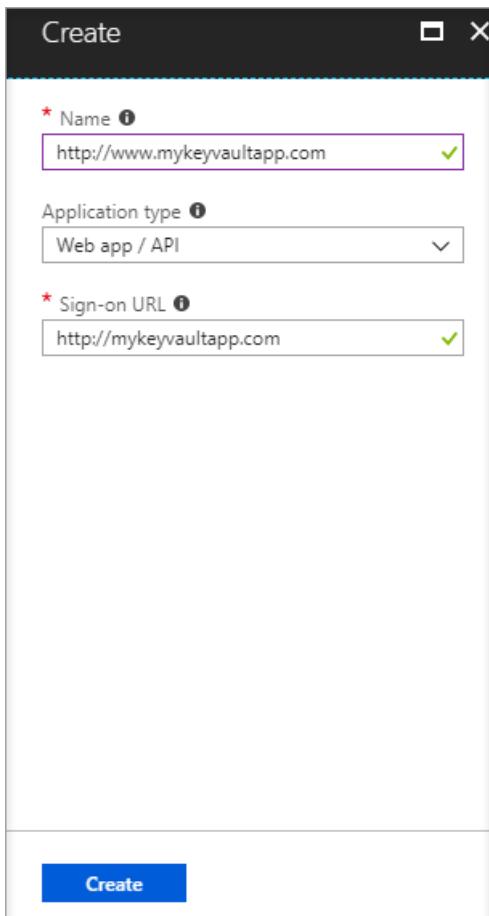
NOTE

Your application must be created on the same Azure Active Directory tenant as your key vault.

1. Open **Azure Active Directory**.
2. Select **App registrations**.
3. Select **New application registration** to add an application to Azure Active Directory.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a navigation sidebar with various service icons. One icon, 'Azure Active Directory', is highlighted with a red box. Within the main content area, under 'Azure Active Directory', another section titled 'App registrations' is highlighted with a red box. At the top of this section, there's a button labeled '+ New application registration' which is also highlighted with a red box. The rest of the interface includes search bars, a list of applications (empty in this view), and standard Azure navigation controls.

4. Under **Create**, leave the application type as **Web app / API** and give your application a name. Give your application a **Sign-on URL**. This URL can be anything you want for this demo.



- After the application is added to Azure Active Directory, the application page opens. Select **Settings**, and then select **Properties**. Copy the **Application ID** value. You'll need it in later steps.

Next, generate a key for your application so it can interact with Azure Active Directory. To create a key, select **Keys** under **Settings**. Make note of the newly generated key for your Azure Active Directory application. You'll need it in a later step. The key won't be available after you leave this section.

DESCRIPTION	EXPIRES	VALUE
No results.		

Before you establish any calls from your application into the key vault, you must tell the key vault about your application and its permissions. The following command uses the vault name and the application ID from your Azure Active Directory app to grant the application **Get** access to your key vault.

```
Set-AzKeyVaultAccessPolicy -VaultName <vaultName> -ServicePrincipalName <clientIDfromAzureAD> -  
PermissionsToSecrets Get
```

You're now ready to start building your application calls. In your application, you must install the NuGet packages that are required to interact with Azure Key Vault and Azure Active Directory. From the Visual Studio Package Manager console, enter the following commands. At the writing of this article, the current version of the Azure Active Directory package is 3.10.305231913, so confirm the latest version and update as needed.

```
Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory -Version 3.10.305231913
```

```
Install-Package Microsoft.Azure.KeyVault
```

In your application code, create a class to hold the method for your Azure Active Directory authentication. In this example, that class is called **Utils**. Add the following `using` statement:

```
using Microsoft.IdentityModel.Clients.ActiveDirectory;
```

Next, add the following method to retrieve the JWT token from Azure Active Directory. For maintainability, you might want to move the hard-coded string values into your web or application configuration.

```
public async static Task<string> GetToken(string authority, string resource, string scope)
{
    var authContext = new AuthenticationContext(authority);

    ClientCredential clientCred = new ClientCredential("<AzureADApplicationClientID>",
<AzureADApplicationClientKey>");

    AuthenticationResult result = await authContext.AcquireTokenAsync(resource, clientCred);

    if (result == null)

        throw new InvalidOperationException("Failed to obtain the JWT token");

    return result.AccessToken;
}
```

Add the necessary code to call Key Vault and retrieve your secret value. First, you must add the following `using` statement:

```
using Microsoft.Azure.KeyVault;
```

Add the method calls to invoke Key Vault and retrieve your secret. In this method, you provide the secret URL that you saved in a previous step. Note the use of the **GetToken** method from the **Utils** class you created previously.

```
var kv = new KeyVaultClient(new KeyVaultClient.AuthenticationCallback(Utils.GetToken));

var sec = kv.GetSecretAsync(<SecretID>).Result.Value;
```

When you run your application, you should now be authenticating to Azure Active Directory and then retrieving your secret value from Azure Key Vault.

Key rotation using Azure Automation

IMPORTANT

Azure Automation runbooks still require the use of the `AzureRM` module.

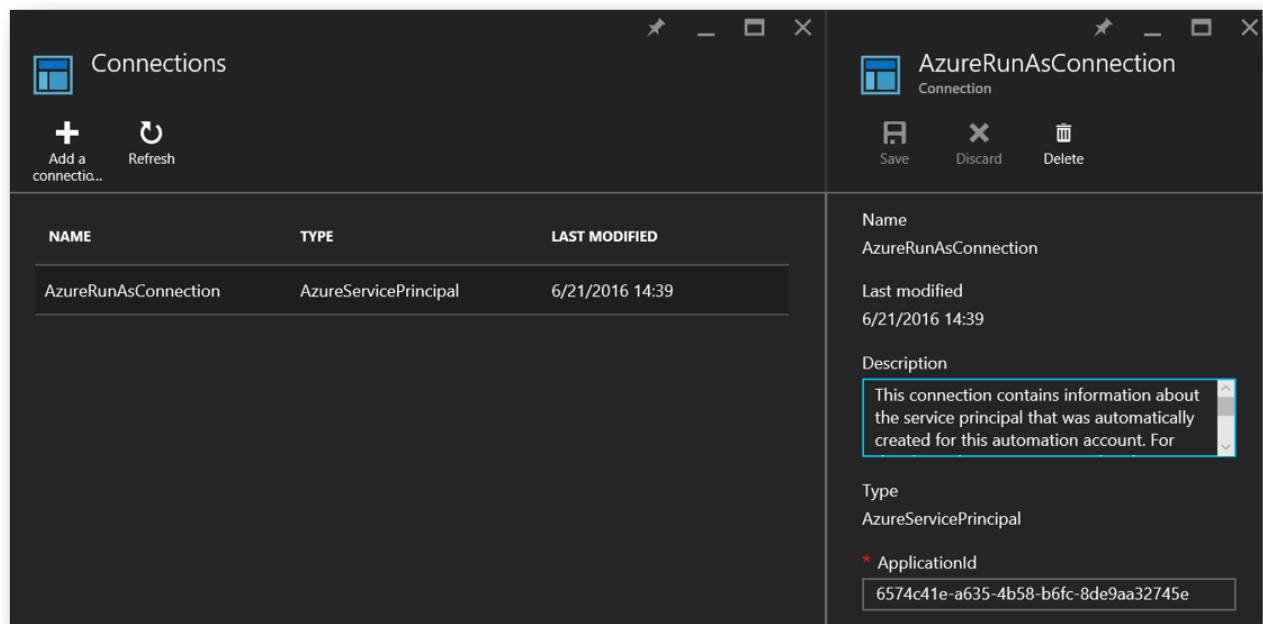
You are now ready to set up a rotation strategy for the values you store as Key Vault secrets. Secrets can be rotated in several ways:

- As part of a manual process
- Programmatically by using API calls

- Through an Azure Automation script

For the purposes of this article, you'll use PowerShell combined with Azure Automation to change an Azure storage account's access key. You'll then update a key vault secret with that new key.

To allow Azure Automation to set secret values in your key vault, you must get the client ID for the connection named **AzureRunAsConnection**. This connection was created when you established your Azure Automation instance. To find this ID, select **Assets** from your Azure Automation instance. From there, select **Connections**, and then select the **AzureRunAsConnection** service principal. Make note of the **ApplicationId** value.



In **Assets**, select **Modules**. Select **Gallery**, and then search for and import updated versions of each of the following modules:

```
Azure
Azure.Storage
AzureRM.Profile
AzureRM.KeyVault
AzureRM.Automation
AzureRM.Storage
```

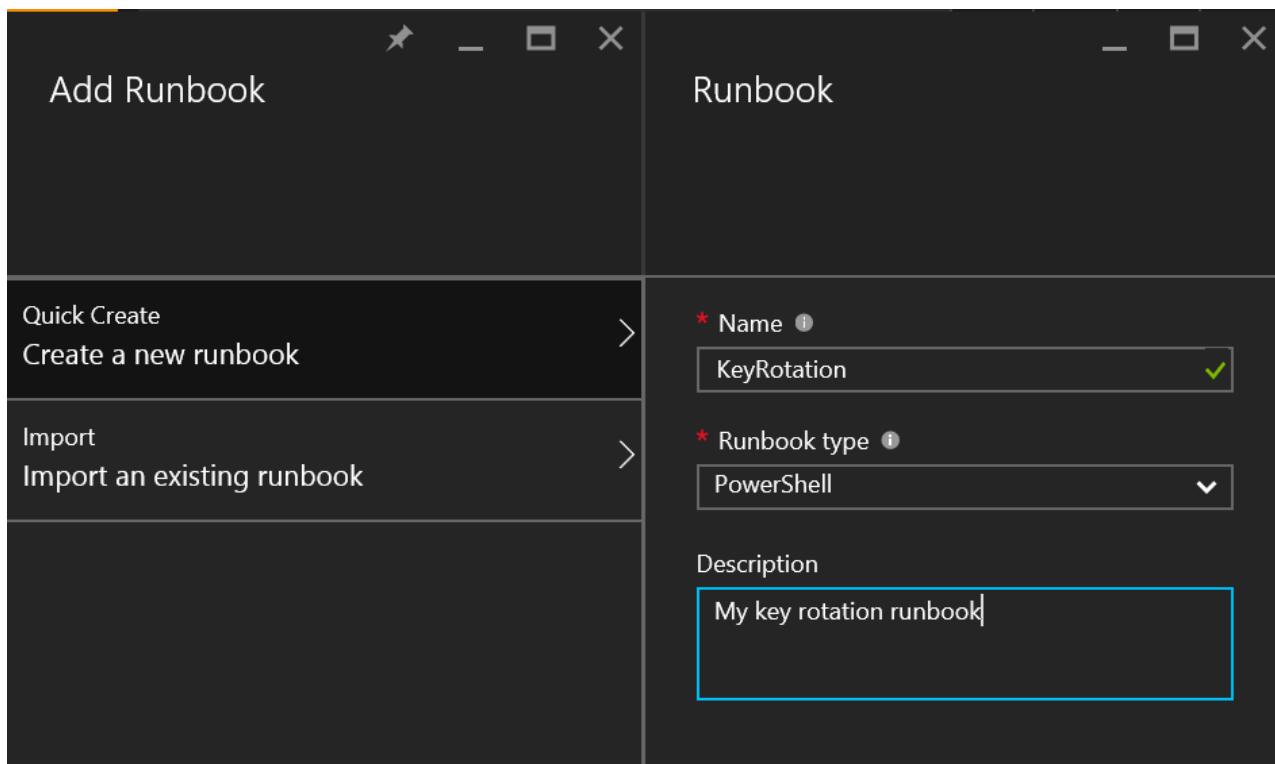
NOTE

At the writing of this article, only the previously noted modules needed to be updated for the following script. If your automation job fails, confirm that you've imported all necessary modules and their dependencies.

After you've retrieved the application ID for your Azure Automation connection, you must tell your key vault that this application has permission to update secrets in your vault. Use the following PowerShell command:

```
Set-AzKeyVaultAccessPolicy -VaultName <vaultName> -ServicePrincipalName <applicationIDfromAzureAutomation> -PermissionsToSecrets Set
```

Next, select **Runbooks** under your Azure Automation instance, and then select **Add Runbook**. Select **Quick Create**. Name your runbook, and select **PowerShell** as the runbook type. You can add a description. Finally, select **Create**.



Paste the following PowerShell script in the editor pane for your new runbook:

```
$connectionName = "AzureRunAsConnection"
try
{
    # Get the connection "AzureRunAsConnection"
    $servicePrincipalConnection=Get-AutomationConnection -Name $connectionName

    "Logging in to Azure..."
    Connect-AzureRmAccount ` 
        -ServicePrincipal ` 
        -TenantId $servicePrincipalConnection.TenantId ` 
        -ApplicationId $servicePrincipalConnection.ApplicationId ` 
        -CertificateThumbprint $servicePrincipalConnection.CertificateThumbprint
    "Login complete."
}
catch {
    if (!$servicePrincipalConnection)
    {
        $ErrorMessage = "Connection $connectionName not found."
        throw $ErrorMessage
    } else{
        Write-Error -Message $_.Exception
        throw $_.Exception
    }
}

# Optionally you can set the following as parameters
$StorageAccountName = <storageAccountName>
$RGName = <storageAccountResourceGroupName>
$VaultName = <keyVaultName>
$SecretName = <keyVaultSecretName>

#Key name. For example key1 or key2 for the storage account
New-AzureRmStorageAccountKey -ResourceGroupName $RGName -Name $StorageAccountName -KeyName "key2" -Verbose
$SAKeys = Get-AzureRmStorageAccountKey -ResourceGroupName $RGName -Name $StorageAccountName

$secretvalue = ConvertTo-SecureString $SAKeys[1].Value -AsPlainText -Force

$secret = Set-AzureKeyVaultSecret -VaultName $VaultName -Name $SecretName -SecretValue $secretvalue
```

In the editor pane, select **Test pane** to test your script. After the script runs without error, you can select **Publish**, and then you can apply a schedule for the runbook in the runbook configuration pane.

Key Vault auditing pipeline

When you set up a key vault, you can turn on auditing to collect logs on access requests made to the key vault. These logs are stored in a designated Azure storage account and can be pulled out, monitored, and analyzed. The following scenario uses Azure functions, Azure logic apps, and key-vault audit logs to create a pipeline that sends an email when an app that doesn't match the app ID of the web app retrieves secrets from the vault.

First, you must enable logging on your key vault. Use the following PowerShell commands. (You can see the full details in [this article about key-vault-logging](#).)

```
$sa = New-AzStorageAccount -ResourceGroupName <resourceGroupName> -Name <storageAccountName> -Type Standard\_\_LRS -Location 'East US'  
$kv = Get-AzKeyVault -VaultName '<vaultName>'  
Set-AzDiagnosticSetting -ResourceId $kv.ResourceId -StorageAccountId $sa.Id -Enabled $true -Category AuditEvent
```

After logging is enabled, audit logs start being stored in the designated storage account. These logs contain events about how and when your key vaults are accessed, and by whom.

NOTE

You can access your logging information 10 minutes after the key vault operation. It will often be available sooner than that.

The next step is to [create an Azure Service Bus queue](#). This queue is where key-vault audit logs are pushed. When the audit-log messages are in the queue, the logic app picks them up and acts on them. Create a Service Bus instance with the following steps:

1. Create a Service Bus namespace (if you already have one that you want to use, skip to step 2).
2. Browse to the Service Bus instance in the Azure portal and select the namespace you want to create the queue in.
3. Select **Create a resource** > **Enterprise Integration** > **Service Bus**, and then enter the required details.
4. Find the Service Bus connection information by selecting the namespace and then selecting **Connection Information**. You'll need this information for the next section.

Next, [create an Azure function](#) to poll the key vault logs within the storage account and pick up new events. This function will be triggered on a schedule.

To create an Azure function app, select **Create a resource**, search the marketplace for **Function App**, and then select **Create**. During creation, you can use an existing hosting plan or create a new one. You can also opt for dynamic hosting. For more information about the hosting options for Azure Functions, see [How to scale Azure Functions](#).

After the Azure function app is created, go to it, and select the **Timer** scenario and **C#** for the language. Then select **Create this function**.

The faster way to functions

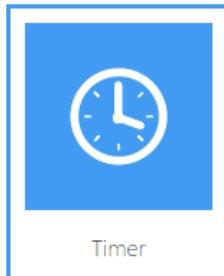
Write any function in minutes – whether to run a simple job that cleans up a database or to build a more complex architecture.



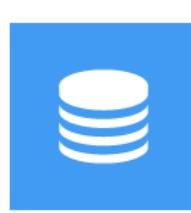
Creating functions is easier than ever before, whatever your chosen OS, platform, or development method. No install required.

Get started quickly with a premade function

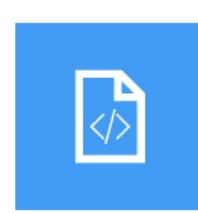
1) Choose a scenario:



Timer



Data processing



Webhook + API

2) Choose a language:

If you'd prefer another supported language, choose "Create a function from scratch".

C#

JavaScript

Create this function

On the **Develop** tab, replace the run.csx code with the following:

```
#r "Newtonsoft.Json"

using System;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Auth;
using Microsoft.WindowsAzure.Storage.Blob;
using Microsoft.ServiceBus.Messaging;
using System.Text;

public static void Run(TimerInfo myTimer, TextReader inputBlob, TextWriter outputBlob, TraceWriter log)
{
    log.Info("Starting");

    CloudStorageAccount sourceStorageAccount = new CloudStorageAccount(new StorageCredentials("<STORAGE_ACCOUNT_NAME>", "<STORAGE_ACCOUNT_KEY>"), true);

    CloudBlobClient sourceCloudBlobClient = sourceStorageAccount.CreateCloudBlobClient();

    var connectionString = "<SERVICE_BUS_CONNECTION_STRING>";
    var queueName = "<SERVICE_BUS_QUEUE_NAME>";

    var sbClient = QueueClient.CreateFromConnectionString(connectionString, queueName);

    DateTime dtPrev = DateTime.UtcNow;
    if(inputBlob != null)
    {
        var txt = inputBlob.ReadToEnd();
        var blob = sourceCloudBlobClient.GetBlobReference("temp");
        blob.UploadText(txt);
        log.Info("File uploaded to blob storage");
    }
}
```

```

        if(!string.IsNullOrEmpty(txt))
        {
            dtPrev = DateTime.Parse(txt);
            log.Verbose($"SyncPoint: {dtPrev.ToString("O")}");
        }
        else
        {
            dtPrev = DateTime.UtcNow;
            log.Verbose($"Sync point file didn't have a date. Setting to now.");
        }
    }

    var now = DateTime.UtcNow;

    string blobPrefix = "insights-logs-
auditevent/resourceId=/SUBSCRIPTIONS/<SUBSCRIPTION_ID>/RESOURCEGROUPS/<RESOURCE_GROUP_NAME>/PROVIDERS/MICROSOFT.KEYVAULT/VAULTS/<KEY_VAULT_NAME>/y=" + now.Year + "/m=" + now.Month.ToString("D2") + "/d=" +
(now.Day).ToString("D2") + "/h=" + (now.Hour).ToString("D2") + "/m=00/";

log.Info($"Scanning: {blobPrefix}");

IEnumerable< IListBlobItem > blobs = sourceCloudBlobClient.ListBlobs(blobPrefix, true);

log.Info($"found {blobs.Count()} blobs");

foreach(var item in blobs)
{
    if (item is CloudBlockBlob)
    {
        CloudBlockBlob blockBlob = (CloudBlockBlob)item;

        log.Info($"Syncing: {item.Uri}");

        string sharedAccessUri = GetContainerSasUri(blockBlob);

        CloudBlockBlob sourceBlob = new CloudBlockBlob(new Uri(sharedAccessUri));

        string text;
        using (var memoryStream = new MemoryStream())
        {
            sourceBlob.DownloadToStream(memoryStream);
            text = System.Text.Encoding.UTF8.GetString(memoryStream.ToArray());
        }

        dynamic dynJson = JsonConvert.DeserializeObject(text);

        //Required to order by time as they might not be in the file
        var results = ((IEnumerable<dynamic>) dynJson.records).OrderBy(p => p.time);

        foreach (var jsonItem in results)
        {
            DateTime dt = Convert.ToDateTime(jsonItem.time);

            if(dt>dtPrev){
                log.Info($"{jsonItem.ToString()}`);

                var payloadStream = new MemoryStream(Encoding.UTF8.GetBytes(jsonItem.ToString()));
                //When sending to ServiceBus, use the payloadStream and set keeporiginal to true
                var message = new BrokeredMessage(payloadStream, true);
                sbClient.Send(message);
                dtPrev = dt;
            }
        }
    }
}

outputBlob.Write(dtPrev.ToString("O"));
}

```

```

static string GetContainerSasUri(CloudBlockBlob blob)
{
    SharedAccessBlobPolicy sasConstraints = new SharedAccessBlobPolicy();

    sasConstraints.SharedAccessStartTime = DateTime.UtcNow.AddMinutes(-5);
    sasConstraints.SharedAccessExpiryTime = DateTime.UtcNow.AddHours(24);
    sasConstraints.Permissions = SharedAccessBlobPermissions.Read;

    //Generate the shared access signature on the container, setting the constraints directly on the
    signature.
    string sasBlobToken = blob.GetSharedAccessSignature(sasConstraints);

    //Return the URI string for the container, including the SAS token.
    return blob.Uri + sasBlobToken;
}

```

NOTE

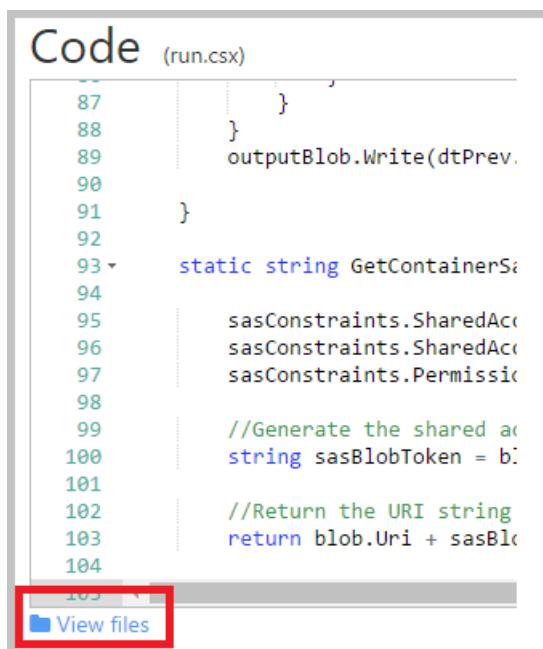
Change the variables in the preceding code to point to your storage account where the key vault logs are written, to the Service Bus instance you created earlier, and to the specific path to the key-vault storage logs.

The function picks up the latest log file from the storage account where the key vault logs are written, grabs the latest events from that file, and pushes them to a Service Bus queue.

Because a single file can have multiple events, you should create a sync.txt file that the function also looks at to determine the time stamp of the last event that was picked up. Using this file ensures that you don't push the same event multiple times.

The sync.txt file contains a time stamp for the last-encountered event. When the logs are loaded, they must be sorted based on their time stamps to ensure that they're ordered correctly.

For this function, we reference a couple additional libraries that aren't available out of the box in Azure Functions. To include these libraries, we need Azure Functions to pull them by using NuGet. Under the **Code** box, select **View Files**.



Add a file called project.json with the following content:

```
{
  "frameworks": {
    "net46": {
      "dependencies": {
        "WindowsAzure.Storage": "7.0.0",
        "WindowsAzure.ServiceBus": "3.2.2"
      }
    }
  }
}
```

After you select **Save**, Azure Functions will download the required binaries.

Switch to the **Integrate** tab and give the timer parameter a meaningful name to use within the function. In the preceding code, the function expects the timer to be called *myTimer*. Specify a [CRON expression](#) for the timer as follows: `0 * * * * *`. This expression will cause the function to run once a minute.

On the same **Integrate** tab, add an input of the type **Azure Blob storage**. This input will point to the sync.txt file that contains the time stamp of the last event looked at by the function. This input will be accessed within the function by using the parameter name. In the preceding code, the Azure Blob storage input expects the parameter name to be *inputBlob*. Select the storage account where the sync.txt file will be located (it could be the same or a different storage account). In the path field, provide the path to the file in the format

`{container-name}/path/to/sync.txt`

Add an output of the type **Azure Blob storage**. This output will point to the sync.txt file you defined in the input. This output is used by the function to write the time stamp of the last event looked at. The preceding code expects this parameter to be called *outputBlob*.

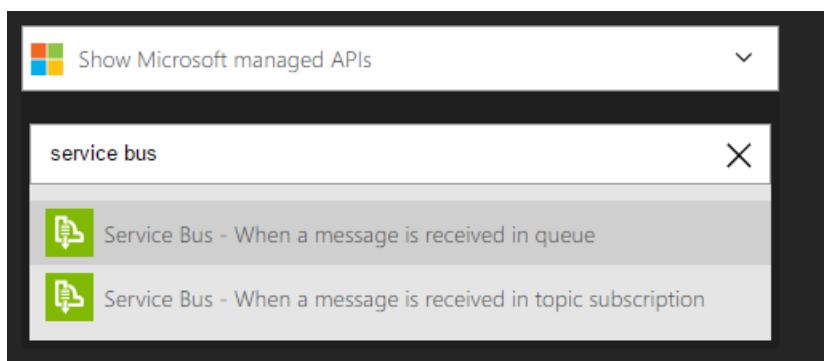
The function is now ready. Make sure to switch back to the **Develop** tab and save the code. Check the output window for any compilation errors and correct them as needed. If the code compiles, then the code should now be checking the key vault logs every minute and pushing any new events into the defined Service Bus queue. You should see logging information write out to the log window every time the function is triggered.

Azure logic app

Next, you must create an Azure logic app that picks up the events that the function is pushing to the Service Bus queue, parses the content, and sends an email based on a condition being matched.

[Create a logic app](#) by selecting **Create a resource > Integration > Logic App**.

After the logic app is created, go to it and select **Edit**. In the logic app editor, select **Service Bus Queue** and enter your Service Bus credentials to connect it to the queue.

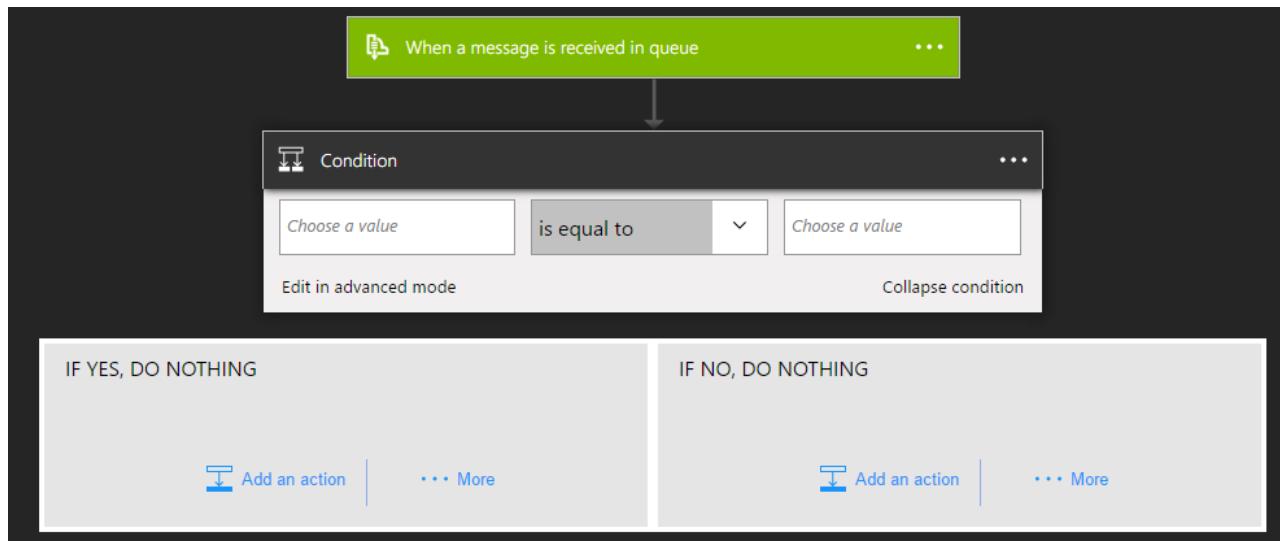


Select **Add a condition**. In the condition, switch to the advanced editor and enter the following code. Replace *APP_ID* with the actual app ID of your web app:

```
@equals('<APP_ID>', json(decodeBase64(triggerBody()['ContentData']))['identity']['claim']['appid'])
```

This expression essentially returns **false** if the *appid* from the incoming event (which is the body of the Service Bus message) isn't the *appid* of the app.

Now, create an action under **IF NO, DO NOTHING**.



For the action, select **Office 365 - send email**. Fill out the fields to create an email to send when the defined condition returns **false**. If you don't have Office 365, look for alternatives to achieve the same results.

You now have an end-to-end pipeline that looks for new key-vault audit logs once a minute. It pushes new logs it finds to a Service Bus queue. The logic app is triggered when a new message lands in the queue. If the *appid* within the event doesn't match the app ID of the calling application, it sends an email.

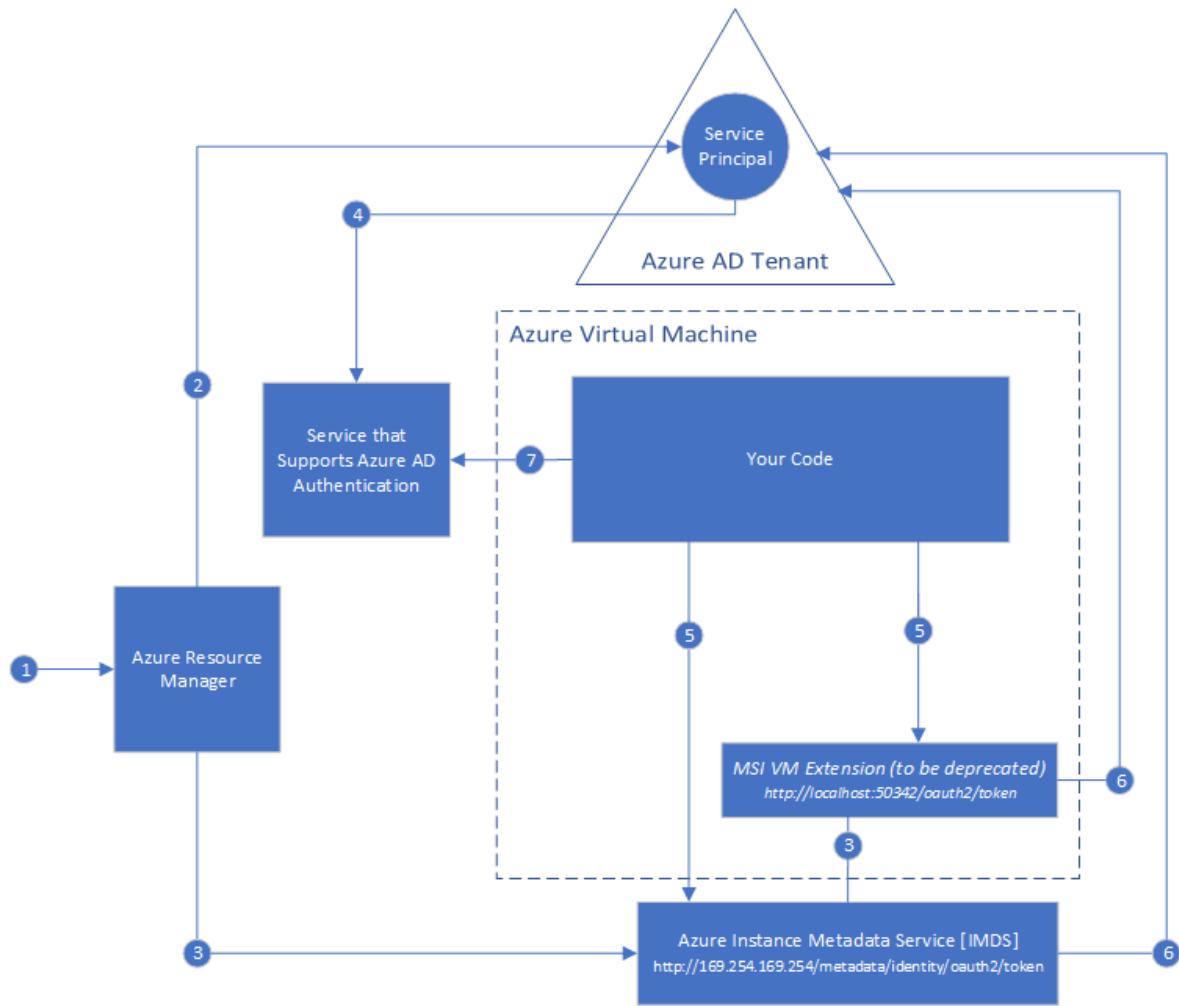
Azure Key Vault basic concepts

6 minutes to read • [Edit Online](#)

Azure Key Vault is a tool for securely storing and accessing secrets. A secret is anything that you want to tightly control access to, such as API keys, passwords, or certificates. A vault is a logical group of secrets.

Here are other important terms:

- **Tenant:** A tenant is the organization that owns and manages a specific instance of Microsoft cloud services. It's most often used to refer to the set of Azure and Office 365 services for an organization.
- **Vault owner:** A vault owner can create a key vault and gain full access and control over it. The vault owner can also set up auditing to log who accesses secrets and keys. Administrators can control the key lifecycle. They can roll to a new version of the key, back it up, and do related tasks.
- **Vault consumer:** A vault consumer can perform actions on the assets inside the key vault when the vault owner grants the consumer access. The available actions depend on the permissions granted.
- **Resource:** A resource is a manageable item that's available through Azure. Common examples are virtual machine, storage account, web app, database, and virtual network. There are many more.
- **Resource group:** A resource group is a container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups, based on what makes the most sense for your organization.
- **Service principal:** An Azure service principal is a security identity that user-created apps, services, and automation tools use to access specific Azure resources. Think of it as a "user identity" (username and password or certificate) with a specific role, and tightly controlled permissions. A service principal should only need to do specific things, unlike a general user identity. It improves security if you grant it only the minimum permission level that it needs to perform its management tasks.
- **Azure Active Directory (Azure AD):** Azure AD is the Active Directory service for a tenant. Each directory has one or more domains. A directory can have many subscriptions associated with it, but only one tenant.
- **Azure tenant ID:** A tenant ID is a unique way to identify an Azure AD instance within an Azure subscription.
- **Managed identities:** Azure Key Vault provides a way to securely store credentials and other keys and secrets, but your code needs to authenticate to Key Vault to retrieve them. Using a managed identity makes solving this problem simpler by giving Azure services an automatically managed identity in Azure AD. You can use this identity to authenticate to Key Vault or any service that supports Azure AD authentication, without having any credentials in your code. For more information, see the following image and the [overview of managed identities for Azure resources](#).



Authentication

To do any operations with Key Vault, you first need to authenticate to it. There are three ways to authenticate to Key Vault:

- **Managed identities for Azure resources:** When you deploy an app on a virtual machine in Azure, you can assign an identity to your virtual machine that has access to Key Vault. You can also assign identities to [other Azure resources](#). The benefit of this approach is that the app or service isn't managing the rotation of the first secret. Azure automatically rotates the identity. We recommend this approach as a best practice.
- **Service principal and certificate:** You can use a service principal and an associated certificate that has access to Key Vault. We don't recommend this approach because the application owner or developer must rotate the certificate.
- **Service principal and secret:** Although you can use a service principal and a secret to authenticate to Key Vault, we don't recommend it. It's hard to automatically rotate the bootstrap secret that's used to authenticate to Key Vault.

Key Vault roles

Use the following table to better understand how Key Vault can help to meet the needs of developers and security administrators.

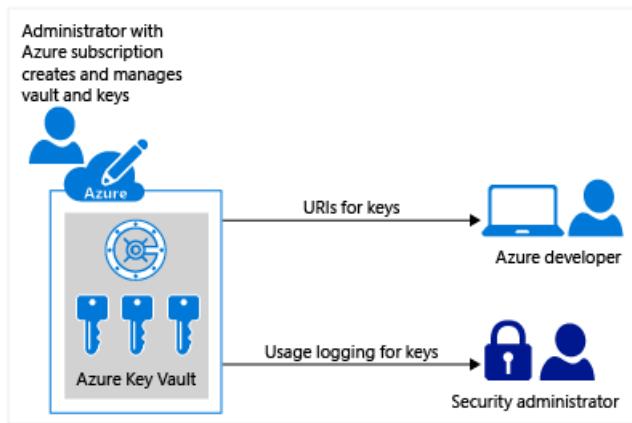
ROLE	PROBLEM STATEMENT	SOLVED BY AZURE KEY VAULT
------	-------------------	---------------------------

ROLE	PROBLEM STATEMENT	SOLVED BY AZURE KEY VAULT
Developer for an Azure application	<p>"I want to write an application for Azure that uses keys for signing and encryption. But I want these keys to be external from my application so that the solution is suitable for an application that's geographically distributed.</p> <p>I want these keys and secrets to be protected, without having to write the code myself. I also want these keys and secrets to be easy for me to use from my applications, with optimal performance."</p>	<ul style="list-style-type: none"> ✓ Keys are stored in a vault and invoked by URI when needed. ✓ Keys are safeguarded by Azure, using industry-standard algorithms, key lengths, and hardware security modules. ✓ Keys are processed in HSMs that reside in the same Azure datacenters as the applications. This method provides better reliability and reduced latency than keys that reside in a separate location, such as on-premises.
Developer for software as a service (SaaS)	<p>"I don't want the responsibility or potential liability for my customers' tenant keys and secrets.</p> <p>I want customers to own and manage their keys so that I can concentrate on doing what I do best, which is providing the core software features."</p>	<ul style="list-style-type: none"> ✓ Customers can import their own keys into Azure, and manage them. When a SaaS application needs to perform cryptographic operations by using customers' keys, Key Vault does these operations on behalf of the application. The application does not see the customers' keys.
Chief security officer (CSO)	<p>"I want to know that our applications comply with FIPS 140-2 Level 2 HSMs for secure key management.</p> <p>I want to make sure that my organization is in control of the key lifecycle and can monitor key usage.</p> <p>And although we use multiple Azure services and resources, I want to manage the keys from a single location in Azure."</p>	<ul style="list-style-type: none"> ✓ HSMs are FIPS 140-2 Level 2 validated. ✓ Key Vault is designed so that Microsoft does not see or extract your keys. ✓ Key usage is logged in near real time. ✓ The vault provides a single interface, regardless of how many vaults you have in Azure, which regions they support, and which applications use them.

Anybody with an Azure subscription can create and use key vaults. Although Key Vault benefits developers and security administrators, it can be implemented and managed by an organization's administrator who manages other Azure services. For example, this administrator can sign in with an Azure subscription, create a vault for the organization in which to store keys, and then be responsible for operational tasks like these:

- Create or import a key or secret
- Revoke or delete a key or secret
- Authorize users or applications to access the key vault, so they can then manage or use its keys and secrets
- Configure key usage (for example, sign or encrypt)
- Monitor key usage

This administrator then gives developers URIs to call from their applications. This administrator also gives key usage logging information to the security administrator.



Developers can also manage the keys directly, by using APIs. For more information, see [the Key Vault developer's guide](#).

Next steps

Learn how to [secure your vault](#).

Azure Key Vault is available in most regions. For more information, see the [Key Vault pricing page](#).

Azure Key Vault security

5 minutes to read • [Edit Online](#)

You need to protect encryption keys and secrets like certificates, connection strings, and passwords in the cloud so you are using Azure Key Vault. Since you are storing sensitive and business critical data, you need to take steps to maximize the security of your vaults and the data stored in them. This article will cover some of the concepts that you should consider when designing your Azure Key Vault security.

Identity and access management

When you create a key vault in an Azure subscription, it's automatically associated with the Azure AD tenant of the subscription. Anyone trying to manage or retrieve content from a vault must be authenticated by Azure AD.

- Authentication establishes the identity of the caller.
- Authorization determines which operations the caller can perform. Authorization in Key Vault uses a combination of [Role based access control](#) (RBAC) and Azure Key Vault access policies.

Access model overview

Access to vaults takes place through two interfaces or planes. These planes are the management plane and the data plane.

- The *management plane* is where you manage Key Vault itself and it is the interface used to create and delete vaults. You can also read key vault properties and manage access policies.
- The *data plane* allows you to work with the data stored in a key vault. You can add, delete, and modify keys, secrets, and certificates.

To access a key vault in either plane, all callers (users or applications) must be authenticated and authorized. Both planes use Azure Active Directory (Azure AD) for authentication. For authorization, the management plane uses role-based access control (RBAC) and the data plane uses a Key Vault access policy.

The model of a single mechanism for authentication to both planes has several benefits:

- Organizations can control access centrally to all key vaults in their organization.
- If a user leaves, they instantly lose access to all key vaults in the organization.
- Organizations can customize authentication by using the options in Azure AD, such as to enable multi-factor authentication for added security

Managing administrative access to Key Vault

When you create a key vault in a resource group, you manage access by using Azure AD. You grant users or groups the ability to manage the key vaults in a resource group. You can grant access at a specific scope level by assigning the appropriate RBAC roles. To grant access to a user to manage key vaults, you assign a predefined `key vault Contributor` role to the user at a specific scope. The following scopes levels can be assigned to an RBAC role:

- **Subscription:** An RBAC role assigned at the subscription level applies to all resource groups and resources within that subscription.
- **Resource group:** An RBAC role assigned at the resource group level applies to all resources in that resource group.
- **Specific resource:** An RBAC role assigned for a specific resource applies to that resource. In this case, the resource is a specific key vault.

There are several predefined roles. If a predefined role doesn't fit your needs, you can define your own role. For more information, see [RBAC: Built-in roles](#).

IMPORTANT

If a user has `Contributor` permissions to a key vault management plane, the user can grant themselves access to the data plane by setting a Key Vault access policy. You should tightly control who has `Contributor` role access to your key vaults. Ensure that only authorized persons can access and manage your key vaults, keys, secrets, and certificates.

Controlling access to Key Vault data

Key Vault access policies grant permissions separately to keys, secrets, or certificate. You can grant a user access only to keys and not to secrets. Access permissions for keys, secrets, and certificates are managed at the vault level.

IMPORTANT

Key Vault access policies don't support granular, object-level permissions like a specific key, secret, or certificate. When a user is granted permission to create and delete keys, they can perform those operations on all keys in that key vault.

To set access policies for a key vault, use the [Azure portal](#), the [Azure CLI](#), [Azure PowerShell](#), or the [Key Vault Management REST APIs](#).

You can restrict data plane access by using [virtual network service endpoints for Azure Key Vault](#). You can configure [firewalls and virtual network rules](#) for an additional layer of security.

Network access

You can reduce the exposure of your vaults by specifying which IP addresses have access to them. The virtual network service endpoints for Azure Key Vault allow you to restrict access to a specified virtual network. The endpoints also allow you to restrict access to a list of IPv4 (internet protocol version 4) address ranges. Any user connecting to your key vault from outside those sources is denied access.

After firewall rules are in effect, users can only read data from Key Vault when their requests originate from allowed virtual networks or IPv4 address ranges. This also applies to accessing Key Vault from the Azure portal. Although users can browse to a key vault from the Azure portal, they might not be able to list keys, secrets, or certificates if their client machine is not in the allowed list. This also affects the Key Vault Picker by other Azure services. Users might be able to see list of key vaults, but not list keys, if firewall rules prevent their client machine.

For more information on Azure Key Vault network address review [Virtual network service endpoints for Azure Key Vault](#)

Monitoring

Key Vault logging saves information about the activities performed on your vault. Key Vault logs:

- All authenticated REST API requests, including failed requests
 - Operations on the key vault itself. These operations include creation, deletion, setting access policies, and updating key vault attributes such as tags.
 - Operations on keys and secrets in the key vault, including:
 - Creating, modifying, or deleting these keys or secrets.
 - Signing, verifying, encrypting, decrypting, wrapping and unwrapping keys, getting secrets, and listing keys and secrets (and their versions).
- Unauthenticated requests that result in a 401 response. Examples are requests that don't have a bearer token, that are malformed or expired, or that have an invalid token.

Logging information can be accessed within 10 minutes after the key vault operation. It's up to you to manage your logs in your storage account.

- Use standard Azure access control methods to secure your logs by restricting who can access them.
- Delete logs that you no longer want to keep in your storage account.

For recommendation on securely managing storage accounts review the [Azure Storage security guide](#)

Next Steps

- [Virtual network service endpoints for Azure Key Vault](#)
- [RBAC: Built-in roles](#)
- [virtual network service endpoints for Azure Key Vault](#)

Azure Security Baseline for Key Vault

28 minutes to read • [Edit Online](#)

The Azure Security Baseline for Key Vault contains recommendations that will help you improve the security posture of your deployment.

The baseline for this services is drawn from the [Azure Security Benchmark version 1.0](#), which provides recommendations on how you can secure your cloud solutions on Azure with our best practices guidance.

For more information, see [Azure Security Baselines overview](#).

Network Security

For more information, see [Security Control: Network Security](#).

1.1: Protect resources using Network Security Groups or Azure Firewall on your Virtual Network

Guidance: Integrate Azure Key Vault with Azure Private Link. Azure Private Link Service enables you to access Azure Services (for example, Azure Key Vault) and Azure hosted customer/partner services over an Private Endpoint in your virtual network. An Azure Private Endpoint is a network interface that connects you privately and securely to a service powered by Azure Private Link. The private endpoint uses a private IP address from your VNet, effectively bringing the service into your VNet. All traffic to the service can be routed through the private endpoint, so no gateways, NAT devices, ExpressRoute or VPN connections, or public IP addresses are needed. Traffic between your virtual network and the service traverses over the Microsoft backbone network, eliminating exposure from the public Internet. You can connect to an instance of an Azure resource, giving you the highest level of granularity in access control.

How to integrate Key Vault with Azure Private Link: <https://docs.microsoft.com/azure/key-vault/private-link-service>

Azure Security Center monitoring: Yes

Responsibility: Customer

1.2: Monitor and log the configuration and traffic of Vnets, Subnets, and NICs

Guidance: Use Azure Security Center and follow network protection recommendations to help secure your Key Vault-configured resources in Azure.

For more information about the Network Security provided by Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-network-recommendations>

Azure Security Center monitoring: Yes

Responsibility: Customer

1.3: Protect critical web applications

Guidance: Not applicable; this recommendation is intended for web applications running on Azure App Service or compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

1.4: Deny communications with known malicious IP addresses

Guidance: Enable Azure DDoS Protection Standard on the Azure Virtual Networks associated with your Key Vault

instances for protection against distributed denial-of-service attacks. Use Azure Security Center Integrated Threat Intelligence to deny communications with known malicious or unused Internet IP addresses.

Manage Azure DDoS Protection Standard using the Azure portal: <https://docs.microsoft.com/azure/virtual-network/manage-ddos-protection> Threat detection for the Azure service layer in Azure Security Center: <https://docs.microsoft.com/azure/security-center/security-center-alerts-service-layer>

Azure Security Center monitoring: Yes

Responsibility: Customer

1.5: Record network packets and flow logs

Guidance: Azure Key Vault does not use network security groups (NSG) and flow logs for Azure Key Vault are not captured. Instead, use Azure Private Link to secure your Azure Key Vault instances and enable diagnostic settings to record metrics and audit events. Integrate Key Vault with Azure Private Link: <https://docs.microsoft.com/azure/key-vault/private-link-service>

Azure Key Vault logging: <https://docs.microsoft.com/azure/key-vault/key-vault-logging>

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

1.6: Deploy network based intrusion detection/intrusion prevention systems (IDS/IPS)

Guidance: This requirement can be met by configuring advanced threat protection (ATP) for Azure Key Vault. ATP provides an additional layer of security intelligence. This tool detects potentially harmful attempts to access or exploit Azure Key Vault accounts.

When Azure Security Center detects anomalous activity, it displays alerts. It also emails the subscription administrator with details of the suspicious activity and recommendations for how to investigate and remediate the identified threats.

Set up advanced threat protection for Azure Key Vault: <https://docs.microsoft.com/azure/security-center/advanced-threat-protection-key-vault>

Azure Security Center monitoring: Yes

Responsibility: Customer

1.7: Manage traffic to web applications

Guidance: Not applicable; this recommendation is intended for web applications running on Azure App Service or compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

1.8: Minimize complexity and administrative overhead of network security rules

Guidance: For resources that need access to your Azure Key Vault instances, use Azure service tags for the Azure Key Vault to define network access controls on network security groups or Azure Firewall. You can use service tags in place of specific IP addresses when creating security rules. By specifying the service tag name (e.g., ApiManagement) in the appropriate source or destination field of a rule, you can allow or deny the traffic for the corresponding service. Microsoft manages the address prefixes encompassed by the service tag and automatically updates the service tag as addresses change.

Azure service tags overview: <https://docs.microsoft.com/azure/virtual-network/service-tags-overview>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

1.9: Maintain standard security configurations for network devices

Guidance: Define and implement standard security configurations for network resources associated with your Azure Key Vault instances with Azure Policy. Use Azure Policy aliases in the "Microsoft.KeyVault" and "Microsoft.Network" namespaces to create custom policies to audit or enforce the network configuration of your Azure Key Vault instances. You may also make use of built-in policy definitions related to Azure Key Vault, such as: [Key Vault should use a virtual network service endpoint](#)

Tutorial: Create and manage policies to enforce compliance:

<https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

Azure Policy Samples:

<https://docs.microsoft.com/azure/governance/policy/samples>

Quickstart: Define and assign a blueprint in the portal:

<https://docs.microsoft.com/azure/governance/blueprints/create-blueprint-portal>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

1.10: Document traffic configuration rules

Guidance: Use tags for resources related to network security and traffic flow for your Azure Key Vault instances to provide metadata and logical organization.

Use any of the built-in Azure policy definitions related to tagging, such as "Require tag and its value" to ensure that all resources are created with tags and to notify you of existing untagged resources.

You may use Azure PowerShell or Azure CLI to look up or perform actions on resources based on their tags.

Use tags to organize your Azure resources:

<https://docs.microsoft.com/azure/azure-resource-manager/resource-group-using-tags>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

1.11: Use automated tools to monitor network resource configurations and detect changes

Guidance: Use Azure Activity Log to monitor network resource configurations and detect changes for network resources related to your Azure Key Vault instances. Create alerts within Azure Monitor that will trigger when changes to critical network resources take place.

View and retrieve Azure Activity Log events:

<https://docs.microsoft.com/azure/azure-monitor/platform/activity-log-view>

Create, view, and manage activity log alerts by using Azure Monitor:

<https://docs.microsoft.com/azure/azure-monitor/platform/alerts-activity-log>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

Logging and Monitoring

For more information, see [Security Control: Logging and Monitoring](#).

2.1: Use approved time synchronization sources

Guidance: Not applicable; Microsoft maintains the time source used for Azure resources, such as Azure Key Vault, for timestamps in the logs.

Azure Security Center monitoring: Not applicable

Responsibility: Microsoft

2.2: Configure central security log management

Guidance: Ingest logs via Azure Monitor to aggregate security data generated by Azure Key Vault. Within Azure Monitor, use Azure Log Analytics workspace to query and perform analytics, and use Azure Storage Accounts for long-term/archival storage. Alternatively, you may enable and on-board data to Azure Sentinel or a third-party SIEM.

Azure Key Vault logging:

<https://docs.microsoft.com/azure/key-vault/key-vault-logging>

Quickstart: How to on-board Azure Sentinel:

<https://docs.microsoft.com/azure/sentinel/quickstart-onboard>

Azure Security Center monitoring: Yes

Responsibility: Customer

2.3: Enable audit logging for Azure resources

Guidance: Enable diagnostic settings on your Azure Key Vault instances for access to audit, security, and diagnostic logs. Activity logs, which are automatically available, include event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

Azure Key Vault logging:

<https://docs.microsoft.com/azure/key-vault/key-vault-logging>

Azure Security Center monitoring: Yes

Responsibility: Customer

2.4: Collect security logs from operating systems

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

2.5: Configure security log storage retention

Guidance: Within Azure Monitor, for the Log Analytics workspace being used to hold your Azure Key Vault logs, set the retention period according to your organization's compliance regulations. Use Azure Storage Accounts for long-term/archival storage.

Change the data retention period: <https://docs.microsoft.com/azure/azure-monitor/platform/manage-cost-storage#change-the-data-retention-period>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

2.6: Monitor and review Logs

Guidance: Analyze and monitor logs for anomalous behavior and regularly review results for your Azure Key

Vault-protected resources. Use Azure Monitor's Log Analytics workspace to review logs and perform queries on log data. Alternatively, you may enable and on-board data to Azure Sentinel or a third party SIEM.

Quickstart: On-board Azure Sentinel: <https://docs.microsoft.com/azure/sentinel/quickstart-onboard>

Get started with Log Analytics in Azure Monitor: <https://docs.microsoft.com/azure/azure-monitor/log-query/get-started-portal>

Get started with log queries in Azure Monitor: <https://docs.microsoft.com/azure/azure-monitor/log-query/get-started-queries>

Azure Security Center monitoring: Yes

Responsibility: Customer

2.7: Enable alerts for anomalous activity

Guidance: In Azure Security Center, enable advanced threat protection (ATP) for Key Vault. Enable diagnostic settings in Azure Key Vault and send logs to a Log Analytics workspace. Onboard your Log Analytics workspace to Azure Sentinel as it provides a security orchestration automated response (SOAR) solution. This allows for playbooks (automated solutions) to be created and used to remediate security issues.

Quickstart: On-board Azure Sentinel:

<https://docs.microsoft.com/azure/sentinel/quickstart-onboard>

Manage and respond to security alerts in Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-managing-and-responding-alerts>

Respond to events with Azure Monitor Alerts:

<https://docs.microsoft.com/azure/azure-monitor/learn/tutorial-response>

Azure Security Center monitoring: Yes

Responsibility: Customer

2.8: Centralize anti-malware logging

Guidance: Not applicable; Azure Key Vault does not process or produce anti-malware related logs.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

2.9: Enable DNS query logging

Guidance: Not applicable; Azure Key Vault does not process or produce DNS related logs.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

2.10: Enable command-line audit logging

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

Identity and Access Control

For more information, see [Security Control: Identity and Access Control](#).

3.1: Maintain an inventory of administrative accounts

Guidance: Maintain an inventory of your Azure Active Directory-registered applications, as well as any user accounts that have access to your Azure Key Vault keys, secrets, and certificates. You may use either the Azure portal or PowerShell to query and reconcile Key Vault access. To view access in PowerShell, use the following command:

```
(Get-AzResource -ResourceId [KeyVaultResourceId]).Properties.AccessPolicies
```

Registering an application with Azure Active Directory: <https://docs.microsoft.com/azure/key-vault/key-vault-manage-with-cli2#registering-an-application-with-azure-active-directory>

Secure access to a key vault: <https://docs.microsoft.com/azure/key-vault/key-vault-secure-your-key-vault>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.2: Change default passwords where applicable

Guidance: Not applicable; Azure Key Vault does not have the concept of default passwords as authentication is provided by Active Directory and secured with Role-based access control.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

3.3: Use dedicated administrative accounts

Guidance: Create standard operating procedures around the use of dedicated administrative accounts that have access to your Azure Key Vault instances. Use Azure Security Center Identity and Access Management (currently in preview) to monitor the number of active administrative accounts.

Monitor identity and access (preview):

<https://docs.microsoft.com/azure/security-center/security-center-identity-access>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.4: Use single sign-on (SSO) with Azure Active Directory

Guidance: Use an Azure service principal in conjunction with the AppId, TenantID, and ClientSecret, to seamlessly authenticate your application and retrieve the token that will be used to access your Azure Key Vault secrets.

Service-to-service authentication to Azure Key Vault using .NET:

<https://docs.microsoft.com/azure/key-vault/service-to-service-authentication>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

3.5: Use multi-factor authentication for all Azure Active Directory based access

Guidance: Enable Azure Active Directory Multi-Factor Authentication and follow Azure Security Center Identity and Access Management (currently in preview) recommendations to help protect your Event Hub-enabled resources.

Planning a cloud-based Azure Multi-Factor Authentication deployment:

<https://docs.microsoft.com/azure/active-directory/authentication/howto-mfa-getstarted>

Monitor identity and access (preview):

<https://docs.microsoft.com/azure/security-center/security-center-identity-access>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.6: Use dedicated machines (Privileged Access Workstations) for all administrative tasks

Guidance: Use a Privileged Access Workstation (PAW) with Azure Multi-Factor Authentication (MFA) configured to log into and configure Key Vault enabled resources.

Privileged Access Workstations: <https://docs.microsoft.com/windows-server/identity/securing-privileged-access/privileged-access-workstations>

Planning a cloud-based Azure Multi-Factor Authentication deployment: <https://docs.microsoft.com/azure/active-directory/authentication/howto-mfa-getstarted>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

3.7: Log and alert on suspicious activity from administrative accounts

Guidance: Use Azure Active Directory (AAD) Privileged Identity Management (PIM) for generation of logs and alerts when suspicious or unsafe activity occurs in the environment. Use AAD risk detections to view alerts and reports on risky user behavior. For additional logging, send Azure Security Center risk detection alerts into Azure Monitor and configure custom alerting/notifications using Action Groups.

Enable advanced threat protection (ATP) for Azure Key Vault to generate alerts for suspicious activity.

Deploy Azure AD Privileged Identity Management (PIM): <https://docs.microsoft.com/azure/active-directory/privileged-identity-management/pim-deployment-plan>

Set up advanced threat protection for Azure Key Vault (preview): <https://docs.microsoft.com/azure/security-center/advanced-threat-protection-key-vault>

Alerts for Azure Key Vault (Preview): <https://docs.microsoft.com/azure/security-center/alerts-reference#alerts-azurekv>

Azure Active Directory risk detections: <https://docs.microsoft.com/azure/active-directory/reports-monitoring/concept-risk-events>

Create and manage action groups in the Azure portal: <https://docs.microsoft.com/azure/azure-monitor/platform/action-groups>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.8: Manage Azure resources from only approved locations

Guidance: Configure the location condition of a Conditional Access policy and manage your named locations. With named locations, you can create logical groupings of IP address ranges or countries and regions. You can restrict access to sensitive resources, such as your Key Vault secrets, to your configured named locations.

What is the location condition in Azure Active Directory Conditional Access?:

<https://docs.microsoft.com/azure/active-directory/reports-monitoring/quickstart-configure-named-locations>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

3.9: Use Azure Active Directory

Guidance: Use Azure Active Directory (AAD) as the central authentication and authorization system for Azure resources such as Key Vault. This allows for Role-based access control (RBAC) to administrate sensitive resources.

Quickstart: Create a new tenant in Azure Active Directory: <https://docs.microsoft.com/azure/active-directory/fundamentals/active-directory-access-create-new-tenant>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

3.10: Regularly review and reconcile user access

Guidance: Review Azure Active Directory (AAD) logs to help discover stale accounts with Azure Key Vault administrative roles. In addition, use AAD access reviews to efficiently manage group memberships, access to enterprise applications that may be used to access Azure Key Vault, and role assignments. User access should be reviewed on a regular basis such as every 90 days to make sure only the right users have continued access.

Azure Active Directory reports and monitoring documentation:

<https://docs.microsoft.com/azure/active-directory/reports-monitoring/>

What are Azure AD access reviews?:

<https://docs.microsoft.com/azure/active-directory/governance/access-reviews-overview>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.11: Monitor attempts to access deactivated accounts

Guidance: Enable diagnostic settings for Azure Key Vault and Azure Active Directory, sending all logs to a Log Analytics workspace. Configure desired alerts (such as attempts to access disabled secrets) within Log Analytics.

Integrate Azure AD logs with Azure Monitor logs: <https://docs.microsoft.com/azure/active-directory/reports-monitoring/howto-integrate-activity-logs-with-log-analytics>

Migrating from the old Key Vault solution: <https://docs.microsoft.com/azure/azure-monitor/insights/azure-key-vault#migrating-from-the-old-key-vault-solution>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.12: Alert on account login behavior deviation

Guidance: Use Azure Active Directory's Identity Protection and risk detection features to configure automated responses to detected suspicious actions related to your Azure Key Vault protected resources. You should enable automated responses through Azure Sentinel to implement your organization's security responses.

Risky sign-ins report in the Azure Active Directory portal: <https://docs.microsoft.com/azure/active-directory/reports-monitoring/concept-risky-sign-ins>

How To: Configure and enable risk policies: <https://docs.microsoft.com/azure/active-directory/identity-protection/howto-identity-protection-configure-risk-policies>

How to onboard Azure Sentinel: <https://docs.microsoft.com/azure/sentinel/quickstart-onboard>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.13: Provide Microsoft with access to relevant customer data during support scenarios

Guidance: Not applicable; Customer Lockbox not supported for Azure Key Vault.

Supported services and scenarios in general availability:

<https://docs.microsoft.com/azure/security/fundamentals/customer-lockbox-overview#supported-services-and-scenarios-in-general-availability>

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

Data Protection

For more information, see [Security Control: Data Protection](#).

4.1: Maintain an inventory of sensitive Information

Guidance: Use tags to assist in tracking Azure resources that store or process sensitive information on Azure Key Vault enabled resources.

Use tags to organize your Azure resources: <https://docs.microsoft.com/azure/azure-resource-manager/resource-group-using-tags>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

4.2: Isolate systems storing or processing sensitive information

Guidance: You can secure access to Azure Key Vault by making use of virtual network service endpoints configured to restrict access to specific subnets.

After firewall rules are in effect, you can only perform Azure Key Vault data plane operations when your request originates from allowed subnets or IP address ranges. This also applies to Azure Key Vault access in the Azure portal. Although you can browse to a key vault from the Azure portal, you may not be able to list keys, secrets, or certificates if your client machine is not on the allowed list. This also affects the Azure Key Vault Picker and other Azure services. You may be able to see lists of Key Vaults, but not list keys, if firewall rules prevent your client machine from doing so.

Configure Azure Key Vault firewalls and virtual networks: <https://docs.microsoft.com/azure/key-vault/key-vault-network-security>

Virtual network service endpoints for Azure Key Vault: <https://docs.microsoft.com/azure/key-vault/key-vault-overview-vnet-service-endpoints>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

4.3: Monitor and block unauthorized transfer of sensitive information

Guidance: All data stored within Azure Key Vault is considered sensitive. Use Azure Key Vault data plane access controls to control access to Azure Key Vault secrets. You may also use Key Vault's built-in firewall to control access at the network layer. To monitor access to Azure Key Vault, enable Key Vault Diagnostic Settings and send logs to an Azure Storage Account or Log Analytics workspace.

Secure access to a key vault: <https://docs.microsoft.com/azure/key-vault/key-vault-secure-your-key-vault>

Configure Azure Key Vault firewalls and virtual networks: <https://docs.microsoft.com/azure/key-vault/key-vault-network-security>

Azure Key Vault logging: <https://docs.microsoft.com/azure/key-vault/key-vault-logging>

Azure Security Center monitoring: Yes

Responsibility: Customer

4.4: Encrypt all sensitive information in transit

Guidance: All traffic to Azure Key Vault for authentication, management, and data plane access, is encrypted and goes over HTTPS: port 443. (However, there will occasionally be HTTP [port 80] traffic for CRL.)

Access Azure Key Vault behind a firewall: <https://docs.microsoft.com/azure/key-vault/key-vault-access-behind-firewall>

Azure Security Center monitoring: Not applicable

Responsibility: Microsoft

4.5: Use an active discovery tool to identify sensitive data

Guidance: Not applicable; all data within Azure Key Vault (secrets, keys, and certificates) is considered sensitive.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

4.6: Use an active discovery tool to identify sensitive data

Guidance: Secure access to the management and data plane of your Azure Key Vault instances.

Secure access to a key vault:

<https://docs.microsoft.com/azure/key-vault/key-vault-secure-your-key-vault>

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

4.7: Use host-based data loss prevention to enforce access control

Guidance: Microsoft manages the underlying infrastructure for Azure Key Vault and has implemented strict controls to prevent the loss or exposure of customer data. Use Azure Security Center to perform baseline scans for your Azure Key Vault-protected resources.

What is Azure Key Vault?: <https://docs.microsoft.com/azure/key-vault/key-vault-overview>

Azure customer data protection: <https://docs.microsoft.com/azure/security/fundamentals/protection-customer-data>

Azure Security Center monitoring: Not applicable

Responsibility: Microsoft

4.8: Encrypt sensitive information at rest

Guidance: All managed objects (key, certificates, and secrets) are encrypted at rest in Azure Key Vault.

Security controls for Azure Key Vault: <https://docs.microsoft.com/azure/key-vault/key-vault-security-controls>

Azure Security Center monitoring: Not applicable

Responsibility: Microsoft

4.9: Log and alert on changes to critical Azure resources

Guidance: Use the Azure Key Vault Analytics solution in Azure Monitor to review Azure Key Vault audit event logs.

Azure Key Vault Analytics solution in Azure Monitor: <https://docs.microsoft.com/azure/azure-monitor/insights/azure-key-vault>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

Vulnerability Management

For more information, see [Security Control: Vulnerability Management](#).

5.1: Run automated vulnerability scanning tools

Guidance: Microsoft performs vulnerability management on the underlying systems that support Azure Key Vault.

Azure Security Center monitoring: Not applicable

Responsibility: Microsoft

5.2: Deploy automated operating system patch management solution

Guidance: N/A; Microsoft performs patch management on the underlying systems that support Key Vault.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

5.3: Deploy automated third-party software patch management solution

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

5.4: Compare back-to-back vulnerability scans

Guidance: Microsoft performs vulnerability management on the underlying systems that support Key Vault.

Azure Security Center monitoring: Not applicable

Responsibility: Microsoft

5.5: Use a risk-rating process to prioritize the remediation of discovered vulnerabilities

Guidance: Use the default risk ratings (Secure Score) provided by Azure Security Center.

Improve your Secure Score in Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-secure-score>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

Inventory and Asset Management

For more information, see [Security Control: Inventory and Asset Management](#).

6.1: Use Azure Asset Discovery

Guidance: Use Azure Resource Graph to query and discover all resources (including Azure Key Vault instances) within your subscription. Ensure you have appropriate (read) permissions in your tenant and are able to enumerate all Azure subscriptions as well as resources within your subscriptions.

Quickstart: Run your first Resource Graph query using Azure Resource Graph Explorer:

<https://docs.microsoft.com/azure/governance/resource-graph/first-query-portal>

Get subscriptions that the current account can access.:

<https://docs.microsoft.com/powershell/module/az.accounts/get-azsubscription?view=azps-3.0.0>

What is role-based access control (RBAC) for Azure resources? <https://docs.microsoft.com/azure/role-based-access-control/overview>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

6.2: Maintain asset metadata

Guidance: Apply tags to Azure Key Vault resources giving metadata to logically organize them into a taxonomy.

How to create and use Tags:

<https://docs.microsoft.com/azure/azure-resource-manager/resource-group-using-tags>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

6.3: Delete unauthorized Azure resources

Guidance: Use tagging, management groups, and separate subscriptions, where appropriate, to organize and track Azure Key Vault instances and related resources. Reconcile inventory on a regular basis and ensure unauthorized resources are deleted from the subscription in a timely manner.

Create an additional Azure subscription:

<https://docs.microsoft.com/azure/billing/billing-create-subscription>

Create management groups for resource organization and management:

<https://docs.microsoft.com/azure/governance/management-groups/create>

Use tags to organize your Azure resources: <https://docs.microsoft.com/azure/azure-resource-manager/resource-group-using-tags>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

6.4: Maintain an inventory of approved Azure resources and software titles

Guidance: Define list of approved Azure resources and approved software for your compute resources

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.5: Monitor for unapproved Azure resources

Guidance: Use Azure policies to put restrictions on the type of resources that can be created in customer subscription(s) using the following built-in policy definitions:

- Not allowed resource types
- Allowed resource types

In addition, use the Azure Resource Graph to query/discover resources within the subscription(s).

Tutorial: Create and manage policies to enforce compliance:

<https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

Quickstart: Run your first Resource Graph query using Azure Resource Graph Explorer:

<https://docs.microsoft.com/azure/governance/resource-graph/first-query-portal>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

6.6: Monitor for unapproved software applications within compute resources

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.7: Remove unapproved Azure resources and software applications

Guidance: Not applicable; this recommendation is intended for Azure as a whole as well as compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.8: Use only approved applications

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.9: Use only approved Azure services

Guidance: Use Azure policies to put restrictions on the type of resources that can be created in customer subscription(s) using the following built-in policy definitions:

- Not allowed resource types
- Allowed resource types

Tutorial: Create and manage policies to enforce compliance:

<https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

Azure Policy Samples: <https://docs.microsoft.com/azure/governance/policy/samples/not-allowed-resource-types>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

6.10: Implement approved application list

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.11: Limit users' ability to interact with AzureResources Manager via scripts

Guidance: Use the Azure Conditional Access to limit users' ability to interact with Azure Resource Manager (ARM) by configuring "Block access" for the "Microsoft Azure Management" App. This can prevent the creation and changes to resources within a high security environment, such as those with Key Vault configuration.

Manage access to Azure management with Conditional Access: <https://docs.microsoft.com/azure/role-based-access-control/conditional-access-azure-management>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

6.12: Limit users' ability to execute scripts within compute resources

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.13: Physically or logically segregate high risk applications

Guidance: Not applicable; this recommendation is intended for web applications running on Azure App Service or compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

Secure Configuration

For more information, see [Security Control: Secure Configuration](#).

7.1: Establish secure configurations for all Azure resources

Guidance: Use Azure Policy aliases in the "Microsoft.KeyVault" namespace to create custom policies to audit or enforce the configuration of your Azure Key Vault instances. You may also use built-in Azure Policy definitions for Azure Key Vault such as:

Key Vault objects should be recoverable

Deploy Diagnostic Settings for Key Vault to Log Analytics workspace

Diagnostic logs in Key Vault should be enabled

Key Vault should use a virtual network service endpoint

Deploy Diagnostic Settings for Key Vault to Event Hub

Use recommendations from Azure Security Center as a secure configuration baseline for your Azure Key Vault instances.

How to view available Azure Policy Aliases:

<https://docs.microsoft.com/powershell/module/az.resources/get-azpolicyalias?view=azps-3.3.0>

Tutorial: Create and manage policies to enforce compliance:

<https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

Azure Security Center monitoring: Yes

Responsibility: Customer

7.2: Establish secure operating system configurations

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

7.3: Maintain secure Azure resource configurations

Guidance: Use Azure policy [deny] and [deploy if not exist] to enforce secure settings across your Azure Key Vault-enabled resources.

Tutorial: Create and manage policies to enforce compliance:

<https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

Understand Azure Policy effects:

<https://docs.microsoft.com/azure/governance/policy/concepts/effects>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

7.4: Maintain secure operating system configurations

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

7.5: Securely store configuration of Azure resources

Guidance: If using custom Azure Policy definitions for your Azure Key Vault enabled resources, use Azure Repos to securely store and manage your code.

How to store code in Azure DevOps:

<https://docs.microsoft.com/azure/devops/repos/git/gitworkflow?view=azure-devops>

Azure Repos Documentation:

<https://docs.microsoft.com/azure/devops/repos/index?view=azure-devops>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

7.6: Securely store custom operating system images

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

7.7: Deploy system configuration management tools

Guidance: Use Azure Policy aliases in the "Microsoft.KeyVault" namespace to create custom policies to alert, audit, and enforce system configurations. Additionally, develop a process and pipeline for managing policy exceptions.

How to configure and manage Azure Policy:

<https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

7.8: Deploy system configuration management tools for operating systems

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

7.9: Implement automated configuration monitoring for Azure services

Guidance: Use Azure Security Center to perform baseline scans for your Azure Key Vault-protected resources.

How to remediate recommendations in Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-remediate-recommendations>

Azure Security Center monitoring: Yes

Responsibility: Customer

7.10: Implement automated configuration monitoring for operating systems

Guidance: Not applicable; this benchmark is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

7.11: Manage Azure secrets securely

Guidance: Use Managed Service Identity in conjunction with Azure Key Vault to simplify and secure secret management for your cloud applications. Ensure that Azure Key Vault soft delete is enabled.

How to integrate with Azure Managed Identities:

<https://docs.microsoft.com/azure/azure-app-configuration/howto-integrate-azure-managed-service-identity>

How to create a Key Vault:

<https://docs.microsoft.com/azure/key-vault/quick-create-portal>

How to provide Key Vault authentication with a managed identity:

<https://docs.microsoft.com/azure/key-vault/managed-identity>

Azure Security Center monitoring: Yes

Responsibility: Customer

7.12: Manage identities securely and automatically

Guidance: Use Managed Service Identity in conjunction with Azure Key Vault to simplify and secure secret management for your cloud applications.

How to integrate with Azure Managed Identities:

<https://docs.microsoft.com/azure/azure-app-configuration/howto-integrate-azure-managed-service-identity>

How to create a Key Vault:

<https://docs.microsoft.com/azure/key-vault/quick-create-portal>

How to provide Key Vault authentication with a managed identity:

<https://docs.microsoft.com/azure/key-vault/managed-identity>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

7.13: Eliminate unintended credential exposure

Guidance: Implement Credential Scanner to identify credentials within code. Credential Scanner will also encourage moving discovered credentials to more secure locations such as Azure Key Vault.

How to setup Credential Scanner: <https://secdevtools.azurewebsites.net/helpcredscan.html>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

Malware Defense

For more information, see [Security Control: Malware Defense](#).

8.1: Use centrally managed anti-malware software

Guidance: Not applicable; this recommendation is intended for compute resources. Microsoft handles anti-malware for underlying platform.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

8.2: Pre-scan files to be uploaded to non-compute Azure resources

Guidance: Microsoft anti-malware is enabled on the underlying host that supports Azure services (for example, Azure Key Vault), however, it does not run on customer content.

Pre-scan any content being uploaded or sent to non-compute Azure resources such as Azure Key Vault. Microsoft cannot access your data in these instances.

Understand Microsoft Antimalware for Azure Cloud Services and Virtual Machines:

<https://docs.microsoft.com/azure/security/fundamentals/antimalware>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

8.3: Ensure anti-malware software and signatures are updated

Guidance: Not applicable; this recommendation is intended for compute resources. Microsoft handles anti-malware for underlying platform.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

Data Recovery

For more information, see [Security Control: Data Recovery](#).

9.1: Ensure regular automated back ups

Guidance: Ensure regular automated backups of your Key Vault Certificates, Keys, Managed Storage Accounts, and Secrets, with the following PowerShell commands:

- `Backup-AzKeyVaultCertificate`
- `Backup-AzKeyVaultKey`
- `Backup-AzKeyVaultManagedStorageAccount`
- `Backup-AzKeyVaultSecret`

Optionally, you may store your Key Vault backups within Azure Backup.

How to backup Key Vault Certificates: <https://docs.microsoft.com/powershell/module/azurerm.keyvault/backup-azurekeyvaultcertificate>

How to backup Key Vault Keys: <https://docs.microsoft.com/powershell/module/azurerm.keyvault/backup-azurekeyvaultkey>

How to backup Key Vault Managed Storage Accounts:

<https://docs.microsoft.com/powershell/module/az.keyvault/add-azkeyvaultmanagedstorageaccount>

How to backup Key Vault Secrets: <https://docs.microsoft.com/powershell/module/azurerm.keyvault/backup-azurekeyvaultsecret>

How to enable Azure Backup: <https://docs.microsoft.com/azure/backup>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

9.2: Perform complete system backups and backup any customer managed keys

Guidance: Perform backups of your Key Vault Certificates, Keys, Managed Storage Accounts, and Secrets, with the following PowerShell commands:

- `Backup-AzKeyVaultCertificate`
- `Backup-AzKeyVaultKey`
- `Backup-AzKeyVaultManagedStorageAccount`
- `Backup-AzKeyVaultSecret`

Optionally, you may store your Key Vault backups within Azure Backup.

How to backup Key Vault Certificates: <https://docs.microsoft.com/powershell/module/azurerm.keyvault/backup-azurekeyvaultcertificate>

How to backup Key Vault Keys: <https://docs.microsoft.com/powershell/module/azurerm.keyvault/backup-azurekeyvaultkey>

How to backup Key Vault Managed Storage Accounts:

<https://docs.microsoft.com/powershell/module/az.keyvault/add-azkeyvaultmanagedstorageaccount>

How to backup Key Vault Secrets: <https://docs.microsoft.com/powershell/module/azurerm.keyvault/backup-azurekeyvaultsecret>

How to enable Azure Backup: <https://docs.microsoft.com/azure/backup>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

9.3: Validate all backups including customer managed keys

Guidance: Periodically perform data restoration of your Key Vault Certificates, Keys, Managed Storage Accounts, and Secrets, with the following PowerShell commands:

- `Restore-AzKeyVaultCertificate`
- `Restore-AzKeyVaultKey`
- `Restore-AzKeyVaultManagedStorageAccount`
- `Restore-AzKeyVaultSecret`

How to restore Key Vault Certificates: <https://docs.microsoft.com/powershell/module/azurerm.keyvault/restore-azurekeyvaultcertificate?view=azurermps-6.13.0>

How to restore Key Vault Keys: <https://docs.microsoft.com/powershell/module/azurerm.keyvault/restore-azurekeyvaultkey?view=azurermps-6.13.0>

How to restore Key Vault Managed Storage Accounts:

<https://docs.microsoft.com/powershell/module/az.keyvault/backup-azkeyvaultmanagedstorageaccount>

How to restore Key Vault Secrets: <https://docs.microsoft.com/powershell/module/azurerm.keyvault/restore-azurekeyvaultsecret?view=azurermps-6.13.0>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

9.4: Ensure protection of backups and customer managed keys

Guidance: Ensure that soft delete is enabled for Azure Key Vault. Soft delete allows recovery of deleted key vaults and vault objects such as keys, secrets, and certificates.

How to use Azure Key Vault's Soft Delete:

<https://docs.microsoft.com/azure/key-vault/key-vault-soft-delete-powershell>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

Incident Response

For more information, see [Security Control: Incident Response](#).

10.1: Create an incident response guide

Guidance: Build out an incident response guide for your organization. Ensure that there are written incident response plans that define all roles of personnel as well as phases of incident handling/management from detection to post-incident review. These processes should have a focus on protecting sensitive systems, such as those using Key Vault secrets.

How to configure Workflow Automations within Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-planning-and-operations-guide>

Guidance on building your own security incident response process:

<https://msrc-blog.microsoft.com/2019/07/01/inside-the-msrc-building-your-own-security-incident-response-process/>

Microsoft Security Response Center's Anatomy of an Incident:

<https://msrc-blog.microsoft.com/2019/07/01/inside-the-msrc-building-your-own-security-incident-response-process>

Customer may also leverage NIST's Computer Security Incident Handling Guide to aid in the creation of their own incident response plan:

<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

10.2: Create an incident scoring and prioritization procedure

Guidance: Security Center assigns a severity to each alert to help you prioritize which alerts should be investigated first. The severity is based on how confident Security Center is in the finding or the analytic used to issue the alert as well as the confidence level that there was malicious intent behind the activity that led to the alert. Additionally, clearly mark subscriptions (for ex. production, non-prod) and create a naming system to clearly identify and categorize Azure resources, especially those processing sensitive data such as Azure Key Vault secrets.

Azure Security Center monitoring: Yes

Responsibility: Customer

10.3: Test security response procedures

Guidance: Conduct exercises to test your systems' incident response capabilities on a regular cadence to help protect your Azure Key Vault instances and related resources. Identify weak points and gaps and revise plan as needed.

Refer to NIST's publication: Guide to Test, Training, and Exercise Programs for IT Plans and Capabilities:

<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-84.pdf>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

10.4: Provide security incident contact details and configure alert notifications for security incidents

Guidance: Security incident contact information will be used by Microsoft to contact you if the Microsoft Security Response Center (MSRC) discovers that your data has been accessed by an unlawful or unauthorized party. Review incidents after the fact to ensure that issues are resolved.

How to set the Azure Security Center Security Contact:

<https://docs.microsoft.com/azure/security-center/security-center-provide-security-contact-details>

Azure Security Center monitoring: Yes

Responsibility: Customer

10.5: Incorporate security alerts into your incident response system

Guidance: Export your Azure Security Center alerts and recommendations using the Continuous Export feature to help identify risks to Azure Key Vault-enabled resources. Continuous Export allows you to export alerts and recommendations either manually or in an ongoing, continuous fashion. You may use the Azure Security Center data connector to stream the alerts to Azure Sentinel.

How to configure continuous export:

<https://docs.microsoft.com/azure/security-center/continuous-export>

How to stream alerts into Azure Sentinel:

<https://docs.microsoft.com/azure/sentinel/connect-azure-security-center>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

10.6: Automate the response to security alerts

Guidance: Use the Workflow Automation feature in Azure Security Center to automatically trigger responses via "Logic Apps" on security alerts and recommendations to protect your Azure Key Vault-protected resources.

How to configure Workflow Automation and Logic Apps:

<https://docs.microsoft.com/azure/security-center/workflow-automation>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

Penetration Tests and Red Team Exercises

For more information, see [Security Control: Penetration Tests and Red Team Exercises](#).

11.1: Conduct regular penetration testing of your Azure resources and ensure remediation of all critical security findings within 60 days

Guidance: You are not to perform pen testing on the Azure Key Vault service directly, however it is encouraged to test your Azure resources which are using Key Vault to ensure the security of the secrets.

You will need to follow the Microsoft Rules of Engagement to ensure your Penetration Tests are not in violation of

Microsoft policies:

<https://www.microsoft.com/msrc/pentest-rules-of-engagement?rtc=1>

You can find more information on Microsoft's strategy and execution of Red Teaming and live site penetration testing against Microsoft-managed cloud infrastructure, services, and applications, here:

<https://gallery.technet.microsoft.com/Cloud-Red-Teaming-b837392e>

Azure Security Center monitoring: Not applicable

Responsibility: Shared

Next steps

- See the [Azure Security Benchmark](#)
- Learn more about [Azure Security Baselines](#)

Azure Key Vault security worlds and geographic boundaries

2 minutes to read • [Edit Online](#)

Azure Key Vault is a multi-tenant service and uses a pool of Hardware Security Modules (HSMs) in each Azure location.

All HSMs at Azure locations in the same geographic region share the same cryptographic boundary (Thales Security World). For example, East US and West US share the same security world because they belong to the US geo location. Similarly, all Azure locations in Japan share the same security world and all Azure locations in Australia, India, and so on.

Backup and restore behavior

A backup taken of a key from a key vault in one Azure location can be restored to a key vault in another Azure location, as long as both of these conditions are true:

- Both of the Azure locations belong to the same geographical location
- Both of the key vaults belong to the same Azure subscription

For example, a backup taken by a given subscription of a key in a key vault in West India, can only be restored to another key vault in the same subscription and geo location; West India, Central India or South India.

Regions and products

- [Azure regions](#)
- [Microsoft products by region](#)

Regions are mapped to security worlds, shown as major headings in the tables:

In the products by region article, for example, the **Americas** tab contains EAST US, CENTRAL US, WEST US all mapping to the Americas region.

NOTE

An exception is that US DOD EAST and US DOD CENTRAL have their own security worlds.

Similarly, on the **Europe** tab, NORTH EUROPE and WEST EUROPE both map to the Europe region. The same is also true on the **Asia Pacific** tab.

Secure access to a key vault

13 minutes to read • [Edit Online](#)

Azure Key Vault is a cloud service that safeguards encryption keys and secrets like certificates, connection strings, and passwords. Because this data is sensitive and business critical, you need to secure access to your key vaults by allowing only authorized applications and users. This article provides an overview of the Key Vault access model. It explains authentication and authorization, and describes how to secure access to your key vaults.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Access model overview

Access to a key vault is controlled through two interfaces: the **management plane** and the **data plane**. The management plane is where you manage Key Vault itself. Operations in this plane include creating and deleting key vaults, retrieving Key Vault properties, and updating access policies. The data plane is where you work with the data stored in a key vault. You can add, delete, and modify keys, secrets, and certificates.

To access a key vault in either plane, all callers (users or applications) must have proper authentication and authorization. Authentication establishes the identity of the caller. Authorization determines which operations the caller can execute.

Both planes use Azure Active Directory (Azure AD) for authentication. For authorization, the management plane uses role-based access control (RBAC) and the data plane uses a Key Vault access policy.

Active Directory authentication

When you create a key vault in an Azure subscription, it's automatically associated with the Azure AD tenant of the subscription. All callers in both planes must register in this tenant and authenticate to access the key vault. In both cases, applications can access Key Vault in two ways:

- **User plus application access:** The application accesses Key Vault on behalf of a signed-in user. Examples of this type of access include Azure PowerShell and the Azure portal. User access is granted in two ways. Users can access Key Vault from any application, or they must use a specific application (referred to as *compound identity*).
- **Application-only access:** The application runs as a daemon service or background job. The application identity is granted access to the key vault.

For both types of access, the application authenticates with Azure AD. The application uses any [supported authentication method](#) based on the application type. The application acquires a token for a resource in the plane to grant access. The resource is an endpoint in the management or data plane, based on the Azure environment. The application uses the token and sends a REST API request to Key Vault. To learn more, review the [whole authentication flow](#).

The model of a single mechanism for authentication to both planes has several benefits:

- Organizations can control access centrally to all key vaults in their organization.

- If a user leaves, they instantly lose access to all key vaults in the organization.
- Organizations can customize authentication by using the options in Azure AD, such as to enable multi-factor authentication for added security.

Resource endpoints

Applications access the planes through endpoints. The access controls for the two planes work independently. To grant an application access to use keys in a key vault, you grant data plane access by using a Key Vault access policy. To grant a user read access to Key Vault properties and tags, but not access to data (keys, secrets, or certificates), you grant management plane access with RBAC.

The following table shows the endpoints for the management and data planes.

ACCESS PLANE	ACCESS ENDPOINTS	OPERATIONS	ACCESS CONTROL MECHANISM
Management plane	Global: management.azure.com:443 Azure China 21Vianet: management.chinacloudapi.cn:443 Azure US Government: management.usgovcloudapi.net:443 Azure Germany: management.microsoftazure.de:443	Create, read, update, and delete key vaults Set Key Vault access policies Set Key Vault tags	Azure Resource Manager RBAC
Data plane	Global: <vault-name>.vault.azure.net:443 Azure China 21Vianet: <vault-name>.vault.azure.cn:443 Azure US Government: <vault-name>.vault.usgovcloudapi.net:443 Azure Germany: <vault-name>.vault.microsoftazure.de:443	Keys: decrypt, encrypt, unwrap, wrap, verify, sign, get, list, update, create, import, delete, backup, restore Secrets: get, list, set, delete	Key Vault access policy

Management plane and RBAC

In the management plane, you use RBAC(Role Based Access Control) to authorize the operations a caller can execute. In the RBAC model, each Azure subscription has an instance of Azure AD. You grant access to users, groups, and applications from this directory. Access is granted to manage resources in the Azure subscription that use the Azure Resource Manager deployment model. To grant access, use the [Azure portal](#), the [Azure CLI](#), [Azure PowerShell](#), or the [Azure Resource Manager REST APIs](#).

You create a key vault in a resource group and manage access by using Azure AD. You grant users or groups the ability to manage the key vaults in a resource group. You grant the access at a specific scope level by assigning appropriate RBAC roles. To grant access to a user to manage key vaults, you assign a predefined

`key vault Contributor` role to the user at a specific scope. The following scopes levels can be assigned to an RBAC role:

- **Subscription:** An RBAC role assigned at the subscription level applies to all resource groups and resources within that subscription.
- **Resource group:** An RBAC role assigned at the resource group level applies to all resources in that resource group.
- **Specific resource:** An RBAC role assigned for a specific resource applies to that resource. In this case, the resource is a specific key vault.

There are several predefined roles. If a predefined role doesn't fit your needs, you can define your own role. For more information, see [RBAC: Built-in roles](#).

IMPORTANT

If a user has `Contributor` permissions to a key vault management plane, the user can grant themselves access to the data plane by setting a Key Vault access policy. You should tightly control who has `Contributor` role access to your key vaults. Ensure that only authorized persons can access and manage your key vaults, keys, secrets, and certificates.

Data plane and access policies

You grant data plane access by setting Key Vault access policies for a key vault. To set these access policies, a user, group, or application must have `Contributor` permissions for the management plane for that key vault.

You grant a user, group, or application access to execute specific operations for keys or secrets in a key vault. Key Vault supports up to 1,024 access policy entries for a key vault. To grant data plane access to several users, create an Azure AD security group and add users to that group.

Key Vault access policies grant permissions separately to keys, secrets, and certificate. You can grant a user access only to keys and not to secrets. Access permissions for keys, secrets, and certificates are at the vault level. Key Vault access policies don't support granular, object-level permissions like a specific key, secret, or certificate. To set access policies for a key vault, use the [Azure portal](#), the [Azure CLI](#), [Azure PowerShell](#), or the [Key Vault Management REST APIs](#).

IMPORTANT

Key Vault access policies apply at the vault level. When a user is granted permission to create and delete keys, they can perform those operations on all keys in that key vault.

You can restrict data plane access by using [virtual network service endpoints for Azure Key Vault](#). You can configure [firewalls and virtual network rules](#) for an additional layer of security.

Example

In this example, we're developing an application that uses a certificate for TLS/SSL, Azure Storage to store data, and an RSA 2,048-bit key for sign operations. Our application runs in an Azure virtual machine (VM) (or a virtual machine scale set). We can use a key vault to store the application secrets. We can store the bootstrap certificate that's used by the application to authenticate with Azure AD.

We need access to the following stored keys and secrets:

- **TLS/SSL certificate:** Used for TLS/SSL.
- **Storage key:** Used to access the Storage account.
- **RSA 2,048-bit key:** Used for sign operations.

- **Bootstrap certificate:** Used to authenticate with Azure AD. After access is granted, we can fetch the storage key and use the RSA key for signing.

We need to define the following roles to specify who can manage, deploy, and audit our application:

- **Security team:** IT staff from the office of the CSO (Chief Security Officer) or similar contributors. The security team is responsible for the proper safekeeping of secrets. The secrets can include TLS/SSL certificates, RSA keys for signing, connection strings, and storage account keys.
- **Developers and operators:** The staff who develop the application and deploy it in Azure. The members of this team aren't part of the security staff. They shouldn't have access to sensitive data like TLS/SSL certificates and RSA keys. Only the application that they deploy should have access to sensitive data.
- **Auditors:** This role is for contributors who aren't members of the development or general IT staff. They review the use and maintenance of certificates, keys, and secrets to ensure compliance with security standards.

There's another role that's outside the scope of our application: the subscription (or resource group) administrator. The subscription admin sets up initial access permissions for the security team. They grant access to the security team by using a resource group that has the resources required by the application.

We need to authorize the following operations for our roles:

Security team

- Create key vaults.
- Turn on Key Vault logging.
- Add keys and secrets.
- Create backups of keys for disaster recovery.
- Set Key Vault access policies to grant permissions to users and applications for specific operations.
- Roll the keys and secrets periodically.

Developers and operators

- Get references from the security team for the bootstrap and TLS/SSL certificates (thumbprints), storage key (secret URI), and RSA key (key URI) for signing.
- Develop and deploy the application to access keys and secrets programmatically.

Auditors

- Review the Key Vault logs to confirm proper use of keys and secrets, and compliance with data security standards.

The following table summarizes the access permissions for our roles and application.

ROLE	MANAGEMENT PLANE PERMISSIONS	DATA PLANE PERMISSIONS
Security team	Key Vault Contributor	Keys: backup, create, delete, get, import, list, restore Secrets: all operations
Developers and operators	Key Vault deploy permission Note: This permission allows deployed VMs to fetch secrets from a key vault.	None

ROLE	MANAGEMENT PLANE PERMISSIONS	DATA PLANE PERMISSIONS
Auditors	None	Keys: list Secrets: list Note: This permission enables auditors to inspect attributes (tags, activation dates, expiration dates) for keys and secrets not emitted in the logs.
Application	None	Keys: sign Secrets: get

The three team roles need access to other resources along with Key Vault permissions. To deploy VMs (or the Web Apps feature of Azure App Service), developers and operators need `contributor` access to those resource types. Auditors need read access to the Storage account where the Key Vault logs are stored.

For more information about how to deploy certificates, access keys, and secrets programmatically, see these resources:

- Learn how to [deploy certificates to VMs from a customer-managed key vault](#) (blog post).
- Download the [Azure Key Vault client samples](#). This content illustrates how to use a bootstrap certificate to authenticate to Azure AD to access a key vault.

You can grant most of the access permissions by using the Azure portal. To grant granular permissions, you can use Azure PowerShell or the Azure CLI.

The PowerShell snippets in this section are built with the following assumptions:

- The Azure AD administrator has created security groups to represent the three roles: Contoso Security Team, Contoso App DevOps, and Contoso App Auditors. The admin has added users to their respective groups.
- All resources are located in the **ContosoAppRG** resource group.
- The Key Vault logs are stored in the **contosologstorage** storage account.
- The **ContosoKeyVault** key vault and the **contosologstorage** storage account are in the same Azure location.

The subscription admin assigns the `key vault Contributor` and `User Access Administrator` roles to the security team. These roles allow the security team to manage access to other resources and key vaults, both of which in the **ContosoAppRG** resource group.

```

New-AzRoleAssignment -ObjectId (Get-AzADGroup -SearchString 'Contoso Security Team')[0].Id -
  RoleDefinitionName "key vault Contributor" -ResourceGroupName ContosoAppRG
New-AzRoleAssignment -ObjectId (Get-AzADGroup -SearchString 'Contoso Security Team')[0].Id -
  RoleDefinitionName "User Access Administrator" -ResourceGroupName ContosoAppRG
  
```

The security team creates a key vault and sets up logging and access permissions. For details about Key Vault access policy permissions, see [About Azure Key Vault keys, secrets, and certificates](#).

```

# Create a key vault and enable logging
$sa = Get-AzStorageAccount -ResourceGroup ContosoAppRG -Name contosologstorage
$kv = New-AzKeyVault -Name ContosoKeyVault -ResourceGroup ContosoAppRG -SKU premium -Location 'westus' -
EnabledForDeployment
Set-AzDiagnosticSetting -ResourceId $kv.ResourceId -StorageAccountId $sa.Id -Enabled $true -Category
AuditEvent

# Set up data plane permissions for the Contoso Security Team role
Set-AzKeyVaultAccessPolicy -VaultName ContosoKeyVault -ObjectId (Get-AzADGroup -SearchString 'Contoso
Security Team')[0].Id -PermissionsToKeys backup,create,delete,get,import,list,restore -PermissionsToSecrets
get,list,set,delete,backup,restore,recover,purge

# Set up management plane permissions for the Contoso App DevOps role
# Create the new role from an existing role
$devopsrole = Get-AzRoleDefinition -Name "Virtual Machine Contributor"
$devopsrole.Id = $null
$devopsrole.Name = "Contoso App DevOps"
$devopsrole.Description = "Can deploy VMs that need secrets from a key vault"
$devopsrole.AssignableScopes = @("/subscriptions/<SUBSCRIPTION-GUID>")

# Add permissions for the Contoso App DevOps role so members can deploy VMs with secrets deployed from key
# vaults
$devopsrole.Actions.Add("Microsoft.KeyVault/vaults/deploy/action")
New-AzRoleDefinition -Role $devopsrole

# Assign the new role to the Contoso App DevOps security group
New-AzRoleAssignment -ObjectId (Get-AzADGroup -SearchString 'Contoso App Devops')[0].Id -RoleDefinitionName
"Contoso App Devops" -ResourceGroupName ContosoAppRG

# Set up data plane permissions for the Contoso App Auditors role
Set-AzKeyVaultAccessPolicy -VaultName ContosoKeyVault -ObjectId (Get-AzADGroup -SearchString 'Contoso App
Auditors')[0].Id -PermissionsToKeys list -PermissionsToSecrets list

```

Our defined custom roles are assignable only to the subscription where the **ContosoAppRG** resource group is created. To use a custom role for other projects in other subscriptions, add other subscriptions to the scope for the role.

For our DevOps staff, the custom role assignment for the key vault `deploy/action` permission is scoped to the resource group. Only VMs created in the **ContosoAppRG** resource group are allowed access to the secrets (TLS/SSL and bootstrap certificates). VMs created in other resource groups by a DevOps member can't access these secrets, even if the VM has the secret URLs.

Our example describes a simple scenario. Real-life scenarios can be more complex. You can adjust permissions to your key vault based on your needs. We assumed the security team provides the key and secret references (URLs and thumbprints), which are used by the DevOps staff in their applications. Developers and operators don't require any data plane access. We focused on how to secure your key vault. Give similar consideration when you secure [your VMs, storage accounts](#), and other Azure resources.

NOTE

This example shows how Key Vault access is locked down in production. Developers should have their own subscription or resource group with full permissions to manage their vaults, VMs, and the storage account where they develop the application.

We recommend that you set up additional secure access to your key vault by [configuring Key Vault firewalls and virtual networks](#).

Resources

- [Azure AD RBAC](#)

- [RBAC: Built-in roles](#)
- [Understand Resource Manager deployment and classic deployment](#)
- [Manage RBAC with Azure PowerShell](#)
- [Manage RBAC with the REST API](#)
- [RBAC for Microsoft Azure](#)

This 2015 Microsoft Ignite conference video discusses access management and reporting capabilities in Azure. It also explores best practices for securing access to Azure subscriptions by using Azure AD.

- [Authorize access to web applications by using OAuth 2.0 and Azure AD](#)
- [Key Vault Management REST APIs](#)

The reference for the REST APIs to manage your key vault programmatically, including setting Key Vault access policy.

- [Key Vault REST APIs](#)
- [Key access control](#)
- [Secret access control](#)
- [Set and remove Key Vault access policy by using PowerShell.](#)

Next steps

Configure [Key Vault firewalls and virtual networks](#).

For a getting-started tutorial for an administrator, see [What is Azure Key Vault?](#).

For more information about usage logging for Key Vault, see [Azure Key Vault logging](#).

For more information about using keys and secrets with Azure Key Vault, see [About keys and secrets](#).

If you have questions about Key Vault, visit the [forums](#).

Azure Key Vault soft-delete overview

5 minutes to read • [Edit Online](#)

Key Vault's soft delete feature allows recovery of the deleted vaults and vault objects, known as soft-delete. Specifically, we address the following scenarios:

- Support for recoverable deletion of a key vault
- Support for recoverable deletion of key vault objects (ex. keys, secrets, certificates)

Supporting interfaces

The soft-delete feature is initially available through the [REST](#), [CLI](#), [PowerShell](#) and [.NET/C#](#) interfaces.

Scenarios

Azure Key Vaults are tracked resources, managed by Azure Resource Manager. Azure Resource Manager also specifies a well-defined behavior for deletion, which requires that a successful DELETE operation must result in that resource not being accessible anymore. The soft-delete feature addresses the recovery of the deleted object, whether the deletion was accidental or intentional.

1. In the typical scenario, a user may have inadvertently deleted a key vault or a key vault object; if that key vault or key vault object were to be recoverable for a predetermined period, the user may undo the deletion and recover their data.
2. In a different scenario, a rogue user may attempt to delete a key vault or a key vault object, such as a key inside a vault, to cause a business disruption. Separating the deletion of the key vault or key vault object from the actual deletion of the underlying data can be used as a safety measure by, for instance, restricting permissions on data deletion to a different, trusted role. This approach effectively requires quorum for an operation which might otherwise result in an immediate data loss.

Soft-delete behavior

When soft-delete is enabled, resources marked as deleted resources are retained for a specified period (90 days by default). The service further provides a mechanism for recovering the deleted object, essentially undoing the deletion.

When creating a new key vault, soft-delete is on by default. You can create a key vault without soft-delete through the [Azure CLI](#) or [Azure Powershell](#). Once soft-delete is enabled on a key vault it cannot be disabled

The default retention period is 90 days but, during key vault creation, it is possible to set the retention policy interval to a value from 7 to 90 days through the Azure portal. The purge protection retention policy uses the same interval. Once set, the retention policy interval cannot be changed.

You cannot reuse the name of a key vault that has been soft-deleted until the retention period has passed.

Purge protection

Purge protection is an optional Key Vault behavior and is **not enabled by default**. It can be turned on via [CLI](#) or [Powershell](#).

When purge protection is on, a vault or an object in the deleted state cannot be purged until the retention period has passed. Soft-deleted vaults and objects can still be recovered, ensuring that the retention policy will be followed.

The default retention period is 90 days, but it is possible to set the retention policy interval to a value from 7 to 90

days through the Azure portal. Once the retention policy interval is set and saved it cannot be changed for that vault.

Permitted purge

Permanently deleting, purging, a key vault is possible via a POST operation on the proxy resource and requires special privileges. Generally, only the subscription owner will be able to purge a key vault. The POST operation triggers the immediate and irrecoverable deletion of that vault.

Exceptions are:

- When the Azure subscription has been marked as *undeletable*. In this case, only the service may then perform the actual deletion, and does so as a scheduled process.
- When the --enable-purge-protection flag is enabled on the vault itself. In this case, Key Vault will wait for 90 days from when the original secret object was marked for deletion to permanently delete the object.

Key vault recovery

Upon deleting a key vault, the service creates a proxy resource under the subscription, adding sufficient metadata for recovery. The proxy resource is a stored object, available in the same location as the deleted key vault.

Key vault object recovery

Upon deleting a key vault object, such as a key, the service will place the object in a deleted state, making it inaccessible to any retrieval operations. While in this state, the key vault object can only be listed, recovered, or forcefully/permanently deleted.

At the same time, Key Vault will schedule the deletion of the underlying data corresponding to the deleted key vault or key vault object for execution after a predetermined retention interval. The DNS record corresponding to the vault is also retained for the duration of the retention interval.

Soft-delete retention period

Soft deleted resources are retained for a set period of time, 90 days. During the soft-delete retention interval, the following apply:

- You may list all of the key vaults and key vault objects in the soft-delete state for your subscription as well as access deletion and recovery information about them.
 - Only users with special permissions can list deleted vaults. We recommend that our users create a custom role with these special permissions for handling deleted vaults.
- A key vault with the same name cannot be created in the same location; correspondingly, a key vault object cannot be created in a given vault if that key vault contains an object with the same name and which is in a deleted state
- Only a specifically privileged user may restore a key vault or key vault object by issuing a recover command on the corresponding proxy resource.
 - The user, member of the custom role, who has the privilege to create a key vault under the resource group can restore the vault.
- Only a specifically privileged user may forcibly delete a key vault or key vault object by issuing a delete command on the corresponding proxy resource.

Unless a key vault or key vault object is recovered, at the end of the retention interval the service performs a purge of the soft-deleted key vault or key vault object and its content. Resource deletion may not be rescheduled.

Billing implications

In general, when an object (a key vault or a key or a secret) is in deleted state, there are only two operations possible: 'purge' and 'recover'. All the other operations will fail. Therefore, even though the object exists, no operations can be performed and hence no usage will occur, so no bill. However there are following exceptions:

- 'purge' and 'recover' actions will count towards normal key vault operations and will be billed.

- If the object is an HSM-key, the 'HSM Protected key' charge per key version per month charge will apply if a key version has been used in last 30 days. After that, since the object is in deleted state no operations can be performed against it, so no charge will apply.

Next steps

The following two guides offer the primary usage scenarios for using soft-delete.

- [How to use Key Vault soft-delete with PowerShell](#)
- [How to use Key Vault soft-delete with CLI](#)

Monitoring Key Vault with Azure Event Grid (preview)

2 minutes to read • [Edit Online](#)

Key Vault integration with Event Grid is currently in preview. It allows users to be notified when the status of a secret stored in key vault has changed. A status change is defined as a secret that is about to expire (within 30 days of expiration), a secret that has expired, or a secret that has a new version available. Notifications for all three secret types (key, certificate, and secret) are supported.

Applications can react to these events using modern serverless architectures, without the need for complicated code or expensive and inefficient polling services. Events are pushed through [Azure Event Grid](#) to event handlers such as [Azure Functions](#), [Azure Logic Apps](#), or even to your own Webhook, and you only pay for what you use. For information about pricing, see [Event Grid pricing](#).

Key Vault events and schemas

Event grid uses [event subscriptions](#) to route event messages to subscribers. Key Vault events contain all the information you need to respond to changes in your data. You can identify a Key Vault event because the eventType property starts with "Microsoft.KeyVault".

For more information, see the [Key Vault event schema](#).

WARNING

Notification events are triggered only on new versions of secrets, keys and certificates, and you must first subscribe to the event on your key vault in order to receive these notifications.

You will receive notification events on certificates only when the certificate is automatically renewed according to the policy you have specified for your certificate.

Practices for consuming events

Applications that handle Key Vault events should follow a few recommended practices:

- Multiple subscriptions can be configured to route events to the same event handler. It is important not to assume events are from a particular source, but to check the topic of the message to ensure that it comes from the key vault you are expecting.
- Similarly, check that the eventType is one you are prepared to process, and do not assume that all events you receive will be the types you expect.
- Ignore fields you don't understand. This practice will help keep you resilient to new features that might be added in the future.
- Use the "subject" prefix and suffix matches to limit events to a particular event.

Next steps

- [Azure Key Vault overview](#)
- [Azure Event Grid overview](#)
- [How to: Route Key Vault Events to Automation Runbook \(preview\)](#).
- [How to: Receive email when a key vault secret changes](#)
- [Azure Event Grid event schema for Azure Key Vault \(preview\)](#)
- [Azure Automation overview](#)

Azure Key Vault throttling guidance

5 minutes to read • [Edit Online](#)

Throttling is a process you initiate that limits the number of concurrent calls to the Azure service to prevent overuse of resources. Azure Key Vault (AKV) is designed to handle a high volume of requests. If an overwhelming number of requests occurs, throttling your client's requests helps maintain optimal performance and reliability of the AKV service.

Throttling limits vary based on the scenario. For example, if you are performing a large volume of writes, the possibility for throttling is higher than if you are only performing reads.

How does Key Vault handle its limits?

Service limits in Key Vault prevent misuse of resources and ensure quality of service for all of Key Vault's clients. When a service threshold is exceeded, Key Vault limits any further requests from that client for a period of time, returns HTTP status code 429 (Too many requests), and the request fails. Failed requests that return a 429 count towards the throttle limits tracked by Key Vault.

Key Vault was originally designed to be used to store and retrieve your secrets at deployment time. The world has evolved, and Key Vault is being used at run-time to store and retrieve secrets, and often apps and services want to use Key Vault like a database. Current limits do not support high throughput rates.

Key Vault was originally created with the limits specified in [Azure Key Vault service limits](#). To maximize your Key Vault through put rates, here are some recommended guidelines/best practices for maximizing your throughput:

1. Ensure you have throttling in place. Client must honor exponential back-off policies for 429's and ensure you are doing retries as per the guidance below.
2. Divide your Key Vault traffic amongst multiple vaults and different regions. Use a separate vault for each security/availability domain. If you have five apps, each in two regions, then we recommend 10 vaults each containing the secrets unique to app and region. A subscription-wide limit for all transaction types is five times the individual key vault limit. For example, HSM-other transactions per subscription are limited to 5,000 transactions in 10 seconds per subscription. Consider caching the secret within your service or app to also reduce the RPS directly to key vault and/or handle burst based traffic. You can also divide your traffic amongst different regions to minimize latency and use a different subscription/vault. Do not send more than the subscription limit to the Key Vault service in a single Azure region.
3. Cache the secrets you retrieve from Azure Key Vault in memory, and reuse from memory whenever possible. Re-read from Azure Key Vault only when the cached copy stops working (e.g. because it got rotated at the source).
4. Key Vault is designed for your own services secrets. If you are storing your customers' secrets (especially for high-throughput key storage scenarios), consider putting the keys in a database or storage account with encryption, and storing just the master key in Azure Key Vault.
5. Encrypt, wrap, and verify public-key operations can be performed with no access to Key Vault, which not only reduces risk of throttling, but also improves reliability (as long as you properly cache the public key material).
6. If you use Key Vault to store credentials for a service, check if that service supports AAD Authentication to authenticate directly. This reduces the load on Key Vault, improves reliability and simplifies your code since Key Vault can now use the AAD token. Many services have moved to using AAD Auth. See the current list at [Services that support managed identities for Azure resources](#).
7. Consider staggering your load/deployment over a longer period of time to stay under the current RPS limits.
8. If your app comprises multiple nodes that need to read the same secret(s), then consider using a fan out pattern, where one entity reads the secret from Key Vault, and fans out to all nodes. Cache the retrieved secrets

only in memory. If you find that the above still does not meet your needs, please fill out the below table and contact us to determine what additional capacity can be added (example put below for illustrative purposes only).

VAULT NAME	VAULT REGION	OBJECT TYPE (SECRET, KEY, OR CERT)	OPERATION(S)*	KEY TYPE	KEY LENGTH OR CURVE	HSM KEY?	STEADY STATE RPS NEEDED	PEAK RPS NEEDED
https://mykeyvault.vault.azure.net/		Key	Sign	EC	P-256	No	200	1000

* For a full list of possible values, see [Azure Key Vault operations](#).

If additional capacity is approved, please note the following as result of the capacity increases:

1. Data consistency model changes. Once a vault is allow listed with additional throughput capacity, the Key Vault service data consistency guarantee changes (necessary to meet higher volume RPS since the underlying Azure Storage service cannot keep up). In a nutshell:
2. **Without allow listing:** The Key Vault service will reflect the results of a write operation (eg. SecretSet, CreateKey) immediately in subsequent calls (eg. SecretGet, KeySign).
3. **With allow listing:** The Key Vault service will reflect the results of a write operation (eg. SecretSet, CreateKey) within 60 seconds in subsequent calls (eg. SecretGet, KeySign).
4. Client code must honor back-off policy for 429 retries. The client code calling the Key Vault service must not immediately retry Key Vault requests when it receives a 429 response code. The Azure Key Vault throttling guidance published here recommends applying exponential backoff when receiving a 429 Http response code.

If you have a valid business case for higher throttle limits, please contact us.

How to throttle your app in response to service limits

The following are **best practices** you should implement when your service is throttled:

- Reduce the number of operations per request.
- Reduce the frequency of requests.
- Avoid immediate retries.
 - All requests accrue against your usage limits.

When you implement your app's error handling, use the HTTP error code 429 to detect the need for client-side throttling. If the request fails again with an HTTP 429 error code, you are still encountering an Azure service limit. Continue to use the recommended client-side throttling method, retrying the request until it succeeds.

Code that implements exponential backoff is shown below.

```
SecretClientOptions options = new SecretClientOptions()
{
    Retry =
    {
        Delay= TimeSpan.FromSeconds(2),
        MaxDelay = TimeSpan.FromSeconds(16),
        MaxRetries = 5,
        Mode = RetryMode.Exponential
    }
};

var client = new SecretClient(new Uri("https://keyVaultName.vault.azure.net"), new DefaultAzureCredential(),options);

//Retrieve Secret
secret = client.GetSecret(secretName);
```

Using this code in a client C# application is straightforward.

Recommended client-side throttling method

On HTTP error code 429, begin throttling your client using an exponential backoff approach:

1. Wait 1 second, retry request
2. If still throttled wait 2 seconds, retry request
3. If still throttled wait 4 seconds, retry request
4. If still throttled wait 8 seconds, retry request
5. If still throttled wait 16 seconds, retry request

At this point, you should not be getting HTTP 429 response codes.

See also

For a deeper orientation of throttling on the Microsoft Cloud, see [Throttling Pattern](#).

Virtual network service endpoints for Azure Key Vault

4 minutes to read • [Edit Online](#)

The virtual network service endpoints for Azure Key Vault allow you to restrict access to a specified virtual network. The endpoints also allow you to restrict access to a list of IPv4 (internet protocol version 4) address ranges. Any user connecting to your key vault from outside those sources is denied access.

There is one important exception to this restriction. If a user has opted-in to allow trusted Microsoft services, connections from those services are let through the firewall. For example, these services include Office 365 Exchange Online, Office 365 SharePoint Online, Azure compute, Azure Resource Manager, and Azure Backup. Such users still need to present a valid Azure Active Directory token, and must have permissions (configured as access policies) to perform the requested operation. For more information, see [Virtual network service endpoints](#).

Usage scenarios

You can configure [Key Vault firewalls and virtual networks](#) to deny access to traffic from all networks (including internet traffic) by default. You can grant access to traffic from specific Azure virtual networks and public internet IP address ranges, allowing you to build a secure network boundary for your applications.

NOTE

Key Vault firewalls and virtual network rules only apply to the [data plane](#) of Key Vault. Key Vault control plane operations (such as create, delete, and modify operations, setting access policies, setting firewalls, and virtual network rules) are not affected by firewalls and virtual network rules.

Here are some examples of how you might use service endpoints:

- You are using Key Vault to store encryption keys, application secrets, and certificates, and you want to block access to your key vault from the public internet.
- You want to lock down access to your key vault so that only your application, or a short list of designated hosts, can connect to your key vault.
- You have an application running in your Azure virtual network, and this virtual network is locked down for all inbound and outbound traffic. Your application still needs to connect to Key Vault to fetch secrets or certificates, or use cryptographic keys.

Configure Key Vault firewalls and virtual networks

Here are the steps required to configure firewalls and virtual networks. These steps apply whether you are using PowerShell, the Azure CLI, or the Azure portal.

1. Enable [Key Vault logging](#) to see detailed access logs. This helps in diagnostics, when firewalls and virtual network rules prevent access to a key vault. (This step is optional, but highly recommended.)
2. Enable **service endpoints for key vault** for target virtual networks and subnets.
3. Set firewalls and virtual network rules for a key vault to restrict access to that key vault from specific virtual networks, subnets, and IPv4 address ranges.
4. If this key vault needs to be accessible by any trusted Microsoft services, enable the option to allow **Trusted Azure Services** to connect to Key Vault.

For more information, see [Configure Azure Key Vault firewalls and virtual networks](#).

IMPORTANT

After firewall rules are in effect, users can only perform Key Vault [data plane](#) operations when their requests originate from allowed virtual networks or IPv4 address ranges. This also applies to accessing Key Vault from the Azure portal. Although users can browse to a key vault from the Azure portal, they might not be able to list keys, secrets, or certificates if their client machine is not in the allowed list. This also affects the Key Vault Picker by other Azure services. Users might be able to see list of key vaults, but not list keys, if firewall rules prevent their client machine.

NOTE

Be aware of the following configuration limitations:

- A maximum of 127 virtual network rules and 127 IPv4 rules are allowed.
- Small address ranges that use the "/31" or "/32" prefix sizes are not supported. Instead, configure these ranges by using individual IP address rules.
- IP network rules are only allowed for public IP addresses. IP address ranges reserved for private networks (as defined in RFC 1918) are not allowed in IP rules. Private networks include addresses that start with **10.**, **172.16-31**, and **192.168..**
- Only IPv4 addresses are supported at this time.

Trusted services

Here's a list of trusted services that are allowed to access a key vault if the **Allow trusted services** option is enabled.

TRUSTED SERVICE	SUPPORTED USAGE SCENARIOS
Azure Virtual Machines deployment service	Deploy certificates to VMs from customer-managed Key Vault.
Azure Resource Manager template deployment service	Pass secure values during deployment.
Azure Disk Encryption volume encryption service	Allow access to BitLocker Key (Windows VM) or DM Passphrase (Linux VM), and Key Encryption Key, during virtual machine deployment. This enables Azure Disk Encryption .
Azure Backup	Allow Backup and restore of relevant keys and secrets during Azure Virtual Machines backup, by using Azure Backup .
Exchange Online & SharePoint Online	Allow access to customer key for Azure Storage Service Encryption with Customer Key .
Azure Information Protection	Allow access to tenant key for Azure Information Protection .
Azure App Service	Deploy Azure Web App Certificate through Key Vault.
Azure SQL Database	Transparent Data Encryption with Bring Your Own Key support for Azure SQL Database and Data Warehouse.
Azure Storage	Storage Service Encryption using customer-managed keys in Azure Key Vault.

TRUSTED SERVICE	SUPPORTED USAGE SCENARIOS
Azure Data Lake Store	Encryption of data in Azure Data Lake Store with a customer-managed key.
Azure Databricks	Fast, easy, and collaborative Apache Spark-based analytics service
Azure API Management	Deploy certificates for Custom Domain from Key Vault using MSI
Azure Data Factory	Fetch data store credentials in Key Vault from Data Factory
Azure Event Hubs	Allow access to a key vault for customer-managed keys scenario
Azure Service Bus	Allow access to a key vault for customer-managed keys scenario
Azure Import/Export	Use customer-managed keys in Azure Key Vault for Import/Export service

NOTE

You must set up the relevant Key Vault access policies to allow the corresponding services to get access to Key Vault.

Next steps

- [Secure your key vault](#)
- [Configure Azure Key Vault firewalls and virtual networks](#)

Authentication, requests and responses

3 minutes to read • [Edit Online](#)

Azure Key Vault supports JSON formatted requests and responses. Requests to the Azure Key Vault are directed to a valid Azure Key Vault URL using HTTPS with some URL parameters and JSON encoded request and response bodies.

This topic covers specifics for the Azure Key Vault service. For general information on using Azure REST interfaces, including authentication/authorization and how to acquire an access token, see [Azure REST API Reference](#).

Request URL

Key management operations use HTTP DELETE, GET, PATCH, PUT and HTTP POST and cryptographic operations against existing key objects use HTTP POST. Clients that cannot support specific HTTP verbs may also use HTTP POST using the X-HTTP-REQUEST header to specify the intended verb; requests that do not normally require a body should include an empty body when using HTTP POST, for example when using POST instead of DELETE.

To work with objects in the Azure Key Vault, the following are example URLs:

- To CREATE a key called TESTKEY in a Key Vault use - `PUT /keys/TESTKEY?api-version=<api_version> HTTP/1.1`
- To IMPORT a key called IMPORTEDKEY into a Key Vault use -
`POST /keys/IMPORTEDKEY/import?api-version=<api_version> HTTP/1.1`
- To GET a secret called MYSECRET in a Key Vault use -
`GET /secrets/MYSECRET?api-version=<api_version> HTTP/1.1`
- To SIGN a digest using a key called TESTKEY in a Key Vault use -
`POST /keys/TESTKEY/sign?api-version=<api_version> HTTP/1.1`

The authority for a request to a Key Vault is always as follows, `https://<keyvault-name>.vault.azure.net/`

Keys are always stored under the /keys path, Secrets are always stored under the /secrets path.

API Version

The Azure Key Vault Service supports protocol versioning to provide compatibility with down-level clients, although not all capabilities will be available to those clients. Clients must use the `api-version` query string parameter to specify the version of the protocol that they support as there is no default.

Azure Key Vault protocol versions follow a date numbering scheme using a {YYYY}.{MM}.{DD} format.

Request Body

As per the HTTP specification, GET operations must NOT have a request body, and POST and PUT operations must have a request body. The body in DELETE operations is optional in HTTP.

Unless otherwise noted in operation description, the request body content type must be application/json and must contain a serialized JSON object conformant to content type.

Unless otherwise noted in operation description, the Accept request header must contain the application/json media type.

Response Body

Unless otherwise noted in operation description, the response body content type of both successful and failed operations will be application/json and contains detailed error information.

Using HTTP POST

Some clients may not be able to use certain HTTP verbs, such as PATCH or DELETE. Azure Key Vault supports HTTP POST as an alternative for these clients provided that the client also includes the "X-HTTP-METHOD" header to specific the original HTTP verb. Support for HTTP POST is noted for each of the API defined in this document.

Error Responses

Error handling will use HTTP status codes. Typical results are:

- 2xx – Success: Used for normal operation. The response body will contain the expected result
- 3xx – Redirection: The 304 "Not Modified" may be returned to fulfill a conditional GET. Other 3xx codes may be used in the future to indicate DNS and path changes.
- 4xx – Client Error: Used for bad requests, missing keys, syntax errors, invalid parameters, authentication errors, etc. The response body will contain detailed error explanation.
- 5xx – Server Error: Used for internal server errors. The response body will contain summarized error information.

The system is designed to work behind a proxy or firewall. Therefore, a client might receive other error codes.

Azure Key Vault also returns error information in the response body when a problem occurs. The response body is JSON formatted and takes the form:

```
{  
  "error":  
  {  
    "code": "BadArgument",  
    "message":  
      "'Foo' is not a valid argument for 'type'."  
  }  
}
```

Authentication

All requests to Azure Key Vault MUST be authenticated. Azure Key Vault supports Azure Active Directory access tokens that may be obtained using OAuth2 [[RFC6749](#)].

For more information on registering your application and authenticating to use Azure Key Vault, see [Register your client application with Azure AD](#).

Access tokens must be sent to the service using the HTTP Authorization header:

```
PUT /keys/MYKEY?api-version=<api_version> HTTP/1.1
Authorization: Bearer <access_token>
```

When an access token is not supplied, or when a token is not accepted by the service, an HTTP 401 error will be returned to the client and will include the WWW-Authenticate header, for example:

```
401 Not Authorized
WWW-Authenticate: Bearer authorization="...", resource="..."
```

The parameters on the WWW-Authenticate header are:

- **authorization:** The address of the OAuth2 authorization service that may be used to obtain an access token for the request.
- **resource:** The name of the resource (<https://vault.azure.net>) to use in the authorization request.

See Also

[About keys, secrets, and certificates](#)

Common parameters and headers

2 minutes to read • [Edit Online](#)

The following information is common to all operations that you might do related to Key Vault resources:

- Replace `{api-version}` with the api-version in the URI.
- Replace `{subscription-id}` with your subscription identifier in the URI
- Replace `{resource-group-name}` with the resource group. For more information, see [Using Resource groups to manage your Azure resources](#).
- Replace `{vault-name}` with your key vault name in the URI.
- Set the Content-Type header to application/json.
- Set the Authorization header to a JSON Web Token that you obtain from Azure Active Directory (AAD). For more information, see [Authenticating Azure Resource Manager requests](#).

Common error response

The service will use HTTP status codes to indicate success or failure. In addition, failures contain a response in the following format:

```
{
  "error": {
    "code": "BadRequest",
    "message": "The key vault sku is invalid."
  }
}
```

ELEMENT NAME	TYPE	DESCRIPTION
code	string	The type of error that occurred.
message	string	A description of what caused the error.

See Also

[Azure Key Vault REST API Reference](#)

About keys, secrets, and certificates

26 minutes to read • [Edit Online](#)

Azure Key Vault enables Microsoft Azure applications and users to store and use several types of secret/key data:

- Cryptographic keys: Supports multiple key types and algorithms, and enables the use of Hardware Security Modules (HSM) for high value keys.
- Secrets: Provides secure storage of secrets, such as passwords and database connection strings.
- Certificates: Supports certificates, which are built on top of keys and secrets and add an automated renewal feature.
- Azure Storage: 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 general information about Key Vault, see [What is Azure Key Vault?](#)

Azure Key Vault

The following sections offer general information applicable across the implementation of the Key Vault service.

Supporting standards

The JavaScript Object Notation (JSON) and JavaScript Object Signing and Encryption (JOSE) specifications are important background information.

- [JSON Web Key \(JWK\)](#)
- [JSON Web Encryption \(JWE\)](#)
- [JSON Web Algorithms \(JWA\)](#)
- [JSON Web Signature \(JWS\)](#)

Data types

Refer to the JOSE specifications for relevant data types for keys, encryption, and signing.

- **algorithm** - a supported algorithm for a key operation, for example, RSA1_5
- **ciphertext-value** - cipher text octets, encoded using Base64URL
- **digest-value** - the output of a hash algorithm, encoded using Base64URL
- **key-type** - one of the supported key types, for example RSA (Rivest-Shamir-Adleman).
- **plaintext-value** - plaintext octets, encoded using Base64URL
- **signature-value** - output of a signature algorithm, encoded using Base64URL
- **base64URL** - a Base64URL [RFC4648] encoded binary value
- **boolean** - either true or false
- **Identity** - an identity from Azure Active Directory (AAD).
- **IntDate** - a JSON decimal value representing the number of seconds from 1970-01-01T0:0:0Z UTC until the specified UTC date/time. See RFC3339 for details regarding date/times, in general and UTC in particular.

Objects, identifiers, and versioning

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. When an object is first created, it's given a unique version identifier and marked as the current version of the object. Creation of a new instance with the same object name gives the

new object a unique version identifier, causing it to become the current version.

Objects in Key Vault can be addressed using the current identifier or a version-specific identifier. For example, given a Key with the name `MasterKey`, performing operations with the current identifier causes the system to use the latest available version. Performing operations with the version-specific identifier causes the system to use that specific version of the object.

Objects are uniquely identified within Key Vault using a URL. No two objects in the system have the same URL, regardless of geo-location. The complete URL to an object is called the Object Identifier. The URL consists of a prefix that identifies the Key Vault, object type, user provided Object Name, and an Object Version. The Object Name is case-insensitive and immutable. Identifiers that don't include the Object Version are referred to as Base Identifiers.

For more information, see [Authentication, requests, and responses](#)

An object identifier has the following general format:

```
https://{{keyvault-name}}.vault.azure.net/{{object-type}}/{{object-name}}/{{object-version}}
```

Where:

<code>keyvault-name</code>	The name for a key vault in the Microsoft Azure Key Vault service. Key Vault names are selected by the user and are globally unique. Key Vault name must be a 3-24 character string, containing only 0-9, a-z, A-Z, and -.
<code>object-type</code>	The type of the object, either "keys" or "secrets".
<code>object-name</code>	An <code>object-name</code> is a user provided name for and must be unique within a Key Vault. The name must be a 1-127 character string, containing only 0-9, a-z, A-Z, and -.
<code>object-version</code>	An <code>object-version</code> is a system-generated, 32 character string identifier that is optionally used to address a unique version of an object.

Key Vault keys

Keys and key types

Cryptographic keys in Key Vault are represented as JSON Web Key [JWK] objects. The base JWK/JWA specifications are also extended to enable key types unique to the Key Vault implementation. For example, importing keys using HSM vendor-specific packaging, enables secure transportation of keys that may only be used in Key Vault HSMs.

- **"Soft" keys:** A key processed in software by Key Vault, but is encrypted at rest using a system key that is in an HSM. Clients may import an existing RSA or EC (Elliptic Curve) key, or request that Key Vault generate one.
- **"Hard" keys:** A key processed in an HSM (Hardware Security Module). These keys are protected in one of the Key Vault HSM Security Worlds (there's one Security World per geography to maintain isolation). Clients may import an RSA or EC key, in soft form or by exporting from a compatible HSM device. Clients may also request Key Vault to generate a key. This key type adds the `key_hsm` attribute

to the JWK obtain to carry the HSM key material.

For more information on geographical boundaries, see [Microsoft Azure Trust Center](#)

Key Vault supports RSA and Elliptic Curve keys only.

- **EC**: "Soft" Elliptic Curve key.
- **EC-HSM**: "Hard" Elliptic Curve key.
- **RSA**: "Soft" RSA key.
- **RSA-HSM**: "Hard" RSA key.

Key Vault supports RSA keys of sizes 2048, 3072 and 4096. Key Vault supports Elliptic Curve key types P-256, P-384, P-521, and P-256K (SECP256K1).

Cryptographic protection

The cryptographic modules that Key Vault uses, whether HSM or software, are FIPS (Federal Information Processing Standards) validated. You don't need to do anything special to run in FIPS mode. Keys **created** or **imported** as HSM-protected are processed inside an HSM, validated to FIPS 140-2 Level 2. Keys **created** or **imported** as software-protected, are processed inside cryptographic modules validated to FIPS 140-2 Level 1. For more information, see [Keys and key types](#).

EC algorithms

The following algorithm identifiers are supported with EC and EC-HSM keys in Key Vault.

Curve Types

- **P-256** - The NIST curve P-256, defined at [DSS FIPS PUB 186-4](#).
- **P-256K** - The SEC curve SECP256K1, defined at [SEC 2: Recommended Elliptic Curve Domain Parameters](#).
- **P-384** - The NIST curve P-384, defined at [DSS FIPS PUB 186-4](#).
- **P-521** - The NIST curve P-521, defined at [DSS FIPS PUB 186-4](#).

SIGN/VERIFY

- **ES256** - ECDSA for SHA-256 digests and keys created with curve P-256. This algorithm is described at [RFC7518](#).
- **ES256K** - ECDSA for SHA-256 digests and keys created with curve P-256K. This algorithm is pending standardization.
- **ES384** - ECDSA for SHA-384 digests and keys created with curve P-384. This algorithm is described at [RFC7518](#).
- **ES512** - ECDSA for SHA-512 digests and keys created with curve P-521. This algorithm is described at [RFC7518](#).

RSA algorithms

The following algorithm identifiers are supported with RSA and RSA-HSM keys in Key Vault.

WRAPKEY/UNWRAPKEY, ENCRYPT/DECRYPT

- **RSA1_5** - RSAES-PKCS1-V1_5 [RFC3447] key encryption
- **RSA-OAEP** - RSAES using Optimal Asymmetric Encryption Padding (OAEP) [RFC3447], with the default parameters specified by RFC 3447 in Section A.2.1. Those default parameters are using a hash function of SHA-1 and a mask generation function of MGF1 with SHA-1.

SIGN/VERIFY

- **PS256** - RSASSA-PSS using SHA-256 and MGF1 with SHA-256, as described in [RFC7518](#).
- **PS384** - RSASSA-PSS using SHA-384 and MGF1 with SHA-384, as described in [RFC7518](#).
- **PS512** - RSASSA-PSS using SHA-512 and MGF1 with SHA-512, as described in [RFC7518](#).
- **RS256** - RSASSA-PKCS-v1_5 using SHA-256. The application supplied digest value must be computed using SHA-256 and must be 32 bytes in length.

- **RS384** - RSASSA-PKCS-v1_5 using SHA-384. The application supplied digest value must be computed using SHA-384 and must be 48 bytes in length.
- **RS512** - RSASSA-PKCS-v1_5 using SHA-512. The application supplied digest value must be computed using SHA-512 and must be 64 bytes in length.
- **RSNULL** - See [RFC2437], a specialized use-case to enable certain TLS scenarios.

Key operations

Key Vault supports the following operations on key objects:

- **Create**: Allows a client to create a key in Key Vault. The value of the key is generated by Key Vault and stored, and isn't released to the client. Asymmetric keys may be created in Key Vault.
- **Import**: Allows a client to import an existing key to Key Vault. Asymmetric keys may be imported to Key Vault using a number of different packaging methods within a JWK construct.
- **Update**: Allows a client with sufficient permissions to modify the metadata (key attributes) associated with a key previously stored within Key Vault.
- **Delete**: Allows a client with sufficient permissions to delete a key from Key Vault.
- **List**: Allows a client to list all keys in a given Key Vault.
- **List versions**: Allows a client to list all versions of a given key in a given Key Vault.
- **Get**: Allows a client to retrieve the public parts of a given key in a Key Vault.
- **Backup**: Exports a key in a protected form.
- **Restore**: Imports a previously backed up key.

For more information, see [Key operations in the Key Vault REST API reference](#).

Once a key has been created in Key Vault, the following cryptographic operations may be performed using the key:

- **Sign and Verify**: Strictly, this operation is "sign hash" or "verify hash", as Key Vault doesn't support hashing of content as part of signature creation. Applications should hash the data to be signed locally, then request that Key Vault sign the hash. Verification of signed hashes is supported as a convenience operation for applications that may not have access to [public] key material. For best application performance, verify that operations are performed locally.
- **Key Encryption / Wrapping**: A key stored in Key Vault may be used to protect another key, typically a symmetric content encryption key (CEK). When the key in Key Vault is asymmetric, key encryption is used. For example, RSA-OAEP and the WRAPKEY/UNWRAPKEY operations are equivalent to ENCRYPT/DECRYPT. When the key in Key Vault is symmetric, key wrapping is used. For example, AES-KW. The WRAPKEY operation is supported as a convenience for applications that may not have access to [public] key material. For best application performance, WRAPKEY operations should be performed locally.
- **Encrypt and Decrypt**: A key stored in Key Vault may be used to encrypt or decrypt a single block of data. The size of the block is determined by the key type and selected encryption algorithm. The Encrypt operation is provided for convenience, for applications that may not have access to [public] key material. For best application performance, encrypt operations should be performed locally.

While WRAPKEY/UNWRAPKEY using asymmetric keys may seem superfluous (as the operation is equivalent to ENCRYPT/DECRYPT), the use of distinct operations is important. The distinction provides semantic and authorization separation of these operations, and consistency when other key types are supported by the service.

Key Vault doesn't support EXPORT operations. Once a key is provisioned in the system, it cannot be extracted or its key material modified. However, users of Key Vault may require their key for other use cases, such as after it has been deleted. In this case, they may use the BACKUP and RESTORE operations to export/import the key in a protected form. Keys created by the BACKUP operation are not usable outside Key Vault. Alternatively, the IMPORT operation may be used against multiple Key Vault instances.

Users may restrict any of the cryptographic operations that Key Vault supports on a per-key basis using the `key_ops` property of the JWK object.

For more information on JWK objects, see [JSON Web Key \(JWK\)](#).

Key attributes

In addition to the key material, the following attributes may be specified. In a JSON Request, the attributes keyword and braces, '{ }', are required even if there are no attributes specified.

- `enabled`: boolean, optional, default is **true**. Specifies whether the key is enabled and useable for cryptographic operations. The `enabled` attribute is used in conjunction with `nbf` and `exp`. When an operation occurs between `nbf` and `exp`, it will only be permitted if `enabled` is set to **true**. Operations outside the `nbf` / `exp` window are automatically disallowed, except for certain operation types under [particular conditions](#).
- `nbf`: IntDate, optional, default is now. The `nbf` (not before) attribute identifies the time before which the key MUST NOT be used for cryptographic operations, except for certain operation types under [particular conditions](#). The processing of the `nbf` attribute requires that the current date/time MUST be after or equal to the not-before date/time listed in the `nbf` attribute. Key Vault MAY provide for some small leeway, normally no more than a few minutes, to account for clock skew. Its value MUST be a number containing an IntDate value.
- `exp`: IntDate, optional, default is "forever". The `exp` (expiration time) attribute identifies the expiration time on or after which the key MUST NOT be used for cryptographic operation, except for certain operation types under [particular conditions](#). The processing of the `exp` attribute requires that the current date/time MUST be before the expiration date/time listed in the `exp` attribute. Key Vault MAY provide for some small leeway, typically no more than a few minutes, to account for clock skew. Its value MUST be a number containing an IntDate value.

There are additional read-only attributes that are included in any response that includes key attributes:

- `created`: IntDate, optional. The `created` attribute indicates when this version of the key was created. The value is null for keys created prior to the addition of this attribute. Its value MUST be a number containing an IntDate value.
- `updated`: IntDate, optional. The `updated` attribute indicates when this version of the key was updated. The value is null for keys that were last updated prior to the addition of this attribute. Its value MUST be a number containing an IntDate value.

For more information on IntDate and other data types, see [Data types](#)

Date-time controlled operations

Not-yet-valid and expired keys, outside the `nbf` / `exp` window, will work for **decrypt**, **unwrap**, and **verify** operations (won't return 403, Forbidden). The rationale for using the not-yet-valid state is to allow a key to be tested before production use. The rationale for using the expired state is to allow recovery operations on data that was created when the key was valid. Also, you can disable access to a key using Key Vault policies, or by updating the `enabled` key attribute to **false**.

For more information on data types, see [Data types](#).

For more information on other possible attributes, see the [JSON Web Key \(JWK\)](#).

Key tags

You can specify additional application-specific metadata in the form of tags. Key Vault supports up to 15 tags, each of which can have a 256 character name and a 256 character value.

NOTE

Tags are readable by a caller if they have the *list* or *get* permission to that object type (keys, secrets, or certificates).

Key access control

Access control for keys managed by Key Vault is provided at the level of a Key Vault that acts as the container of keys. The access control policy for keys is distinct from the access control policy for secrets in the same Key Vault. Users may create one or more vaults to hold keys, and are required to maintain scenario appropriate segmentation and management of keys. Access control for keys is independent of access control for secrets.

The following permissions can be granted, on a per user / service principal basis, in the keys access control entry on a vault. These permissions closely mirror the operations allowed on a key object. Granting access to an service principal in key vault is a onetime operation, and it will remain same for all Azure subscriptions. You can use it to deploy as many certificates as you want.

- Permissions for key management operations

- *get*: Read the public part of a key, plus its attributes
- *list*: List the keys or versions of a key stored in a key vault
- *update*: Update the attributes for a key
- *create*: Create new keys
- *import*: Import a key to a key vault
- *delete*: Delete the key object
- *recover*: Recover a deleted key
- *backup*: Back up a key in a key vault
- *restore*: Restore a backed up key to a key vault

- Permissions for cryptographic operations

- *decrypt*: Use the key to unprotect a sequence of bytes
- *encrypt*: Use the key to protect an arbitrary sequence of bytes
- *unwrapKey*: Use the key to unprotect wrapped symmetric keys
- *wrapKey*: Use the key to protect a symmetric key
- *verify*: Use the key to verify digests
- *sign*: Use the key to sign digests

- Permissions for privileged operations

- *purge*: Purge (permanently delete) a deleted key

For more information on working with keys, see [Key operations in the Key Vault REST API reference](#). For information on establishing permissions, see [Vaults - Create or Update](#) and [Vaults - Update Access Policy](#).

Key Vault secrets

Working with secrets

From a developer's perspective, Key Vault APIs accept and return secret values as strings. Internally, Key Vault stores and manages secrets as sequences of octets (8-bit bytes), with a maximum size of 25k bytes each. The Key Vault service doesn't provide semantics for secrets. It merely accepts the data, encrypts it, stores it, and returns a secret identifier ("id"). The identifier can be used to retrieve the secret at a later time.

For highly sensitive data, clients should consider additional layers of protection for data. Encrypting data using a separate protection key prior to storage in Key Vault is one example.

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. There are no pre-defined values. The suggested usage is as a hint for interpreting the secret data. For instance, an implementation may store both passwords and certificates as secrets, then use this field to differentiate. There are no predefined values.

Secret attributes

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

- *exp*: IntDate, optional, default is **forever**. The *exp* (expiration time) attribute identifies the expiration time on or after which the secret data SHOULD NOT be retrieved, except in [particular situations](#). This field is for **informational** purposes only as it informs users of key vault service that a particular secret may not be used. Its value MUST be a number containing an IntDate value.
- *nbf*: IntDate, optional, default is **now**. The *nbf* (not before) attribute identifies the time before which the secret data SHOULD NOT be retrieved, except in [particular situations](#). This field is for **informational** purposes only. Its value MUST be a number containing an IntDate value.
- *enabled*: boolean, optional, default is **true**. This attribute specifies whether the secret data can be retrieved. The enabled attribute is used in conjunction with *nbf* and *exp* when an operation occurs between *nbf* and *exp*, it will only be permitted if *enabled* is set to **true**. Operations outside the *nbf* and *exp* window are automatically disallowed, except in [particular situations](#).

There are additional read-only attributes that are included in any response that includes secret attributes:

- *created*: IntDate, optional. The created attribute indicates when this version of the secret was created. This value is null for secrets created prior to the addition of this attribute. Its value must be a number containing an IntDate value.
- *updated*: IntDate, optional. The updated attribute indicates when this version of the secret was updated. This value is null for secrets that were last updated prior to the addition of this attribute. Its value must be a number containing an IntDate value.

Date-time controlled operations

A secret's **get** operation will work for not-yet-valid and expired secrets, outside the *nbf* / *exp* window. Calling a secret's **get** operation, for a not-yet-valid secret, can be used for test purposes. Retrieving (**getting**) an expired secret, can be used for recovery operations.

For more information on data types, see [Data types](#).

Secret access control

Access Control for secrets managed in Key Vault, is provided at the level of the Key Vault that contains those secrets. The access control policy for secrets, is distinct from the access control policy for keys in the same Key Vault. Users may create one or more vaults to hold secrets, and are required to maintain scenario appropriate segmentation and management of 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

For more information on working with secrets, see [Secret operations in the Key Vault REST API reference](#). For information on establishing permissions, see [Vaults - Create or Update](#) and [Vaults - Update Access Policy](#).

Secret tags

You can specify additional application-specific metadata in the form of tags. Key Vault supports up to 15 tags, each of which can have a 256 character name and a 256 character value.

NOTE

Tags are readable by a caller if they have the *list* or *get* permission to that object type (keys, secrets, or certificates).

Key Vault Certificates

Key Vault certificates support provides for management of your x509 certificates and the following behaviors:

- Allows a certificate owner to create a certificate through a Key Vault creation process or through the import of an existing certificate. Includes both self-signed and Certificate Authority generated certificates.
- Allows a Key Vault certificate owner to implement secure storage and management of X509 certificates without interaction with private key material.
- Allows a certificate owner to create a policy that directs Key Vault to manage the life-cycle of a certificate.
- Allows certificate owners to provide contact information for notification about life-cycle events of expiration and renewal of certificate.
- Supports automatic renewal with selected issuers - Key Vault partner X509 certificate providers / certificate authorities.

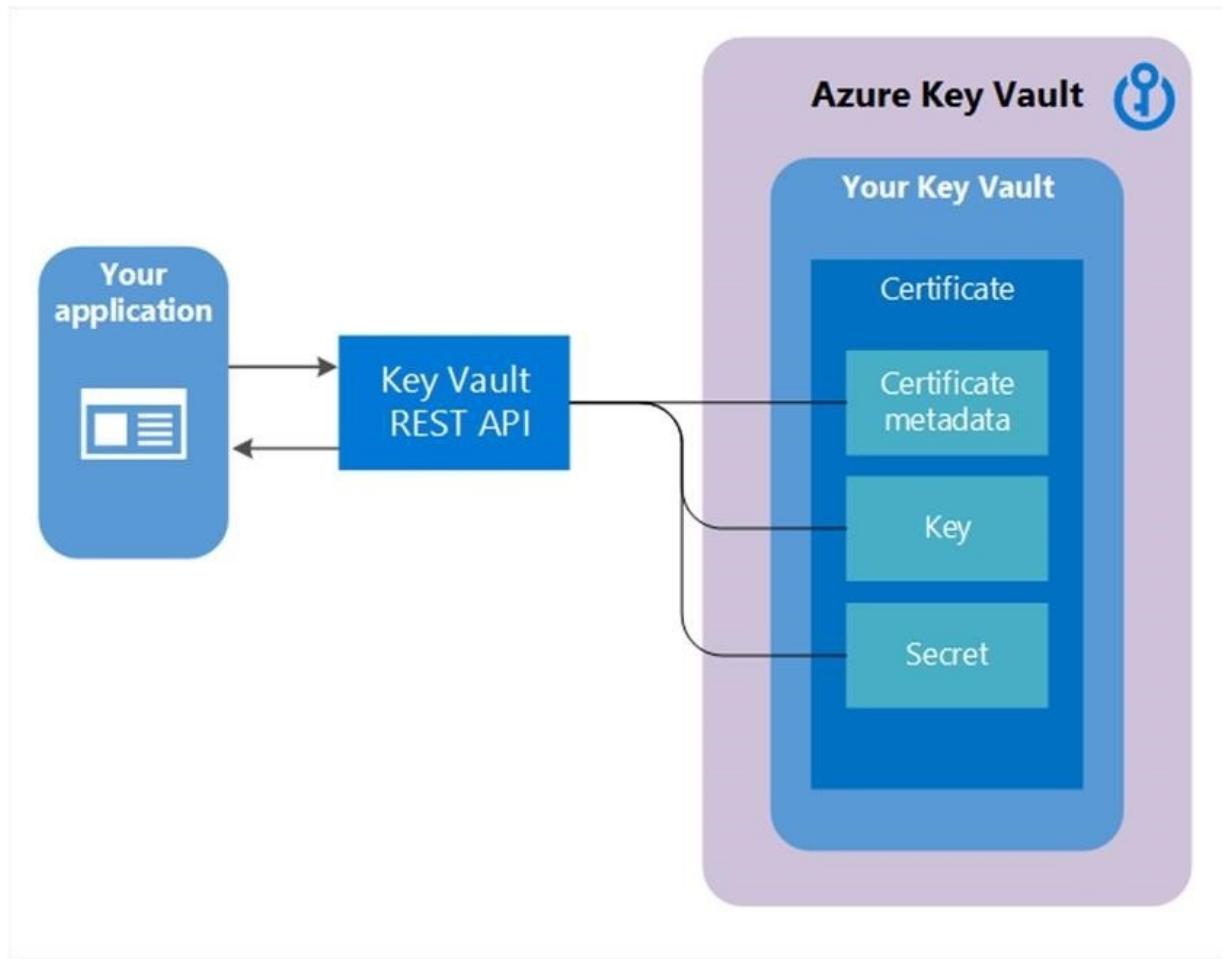
NOTE

Non-partnered providers/authorities are also allowed but, will not support the auto renewal feature.

Composition of a Certificate

When a Key Vault certificate is created, an addressable key and secret are also created with the same name. The Key Vault key allows key operations and the Key Vault secret allows retrieval of the certificate value as a secret. A Key Vault certificate also contains public x509 certificate metadata.

The identifier and version of certificates is similar to that of keys and secrets. A specific version of an addressable key and secret created with the Key Vault certificate version is available in the Key Vault certificate response.



Exportable or Non-exportable key

When a Key Vault certificate is created, it can be retrieved from the addressable secret with the private key in either PFX or PEM format. The policy used to create the certificate must indicate that the key is exportable. If the policy indicates non-exportable, then the private key isn't a part of the value when retrieved as a secret.

The addressable key becomes more relevant with non-exportable KV certificates. The addressable KV key's operations are mapped from *keyusage* field of the KV certificate policy used to create the KV Certificate.

Two types of key are supported – *RSA* or *RSA HSM* with certificates. Exportable is only allowed with RSA, not supported by RSA HSM.

Certificate Attributes and Tags

In addition to certificate metadata, an addressable key and addressable secret, a Key Vault certificate also contains attributes and tags.

Attributes

The certificate attributes are mirrored to attributes of the addressable key and secret created when KV certificate is created.

A Key Vault certificate has the following attributes:

- *enabled*: boolean, optional, default is **true**. Can be specified to indicate if the certificate data can be retrieved as secret or operable as a key. Also used in conjunction with *nbf* and *exp* when an operation occurs between *nbf* and *exp*, and will only be permitted if *enabled* is set to true. Operations outside the *nbf* and *exp* window are automatically disallowed.

There are additional read-only attributes that are included in response:

- *created*: IntDate: indicates when this version of the certificate was created.
- *updated*: IntDate: indicates when this version of the certificate was updated.

- *exp*: IntDate: contains the value of the expiry date of the x509 certificate.
- *nbf*: IntDate: contains the value of the date of the x509 certificate.

NOTE

If a Key Vault certificate expires, its addressable key and secret become inoperable.

Tags

Client specified dictionary of key value pairs, similar to tags in keys and secrets.

NOTE

Tags are readable by a caller if they have the *list* or *get* permission to that object type (keys, secrets, or certificates).

Certificate policy

A certificate policy contains information on how to create and manage lifecycle of a Key Vault certificate. When a certificate with private key is imported into the key vault, a default policy is created by reading the x509 certificate.

When a Key Vault certificate is created from scratch, a policy needs to be supplied. The policy specifies how to create this Key Vault certificate version, or the next Key Vault certificate version. Once a policy has been established, it isn't required with successive create operations for future versions. There's only one instance of a policy for all the versions of a Key Vault certificate.

At a high level, a certificate policy contains the following information:

- X509 certificate properties: Contains subject name, subject alternate names, and other properties used to create an x509 certificate request.
- Key Properties: contains key type, key length, exportable, and reuse key fields. These fields instruct key vault on how to generate a key.
- Secret properties: contains secret properties such as content type of addressable secret to generate the secret value, for retrieving certificate as a secret.
- Lifetime Actions: contains lifetime actions for the KV Certificate. Each lifetime action contains:
 - Trigger: specified via days before expiry or lifetime span percentage
 - Action: specifying action type – *emailContacts* or *autoRenew*
- Issuer: Parameters about the certificate issuer to use to issue x509 certificates.
- Policy Attributes: contains attributes associated with the policy

X509 to Key Vault usage mapping

The following table represents the mapping of x509 key usage policy to effective key operations of a key created as part of a Key Vault certificate creation.

X509 KEY USAGE FLAGS	KEY VAULT KEY OPS	DEFAULT BEHAVIOR
DataEncipherment	encrypt, decrypt	N/A
DecipherOnly	decrypt	N/A

X509 KEY USAGE FLAGS	KEY VAULT KEY OPS	DEFAULT BEHAVIOR
DigitalSignature	sign, verify	Key Vault default without a usage specification at certificate creation time
EncipherOnly	encrypt	N/A
KeyCertSign	sign, verify	N/A
KeyEncipherment	wrapKey, unwrapKey	Key Vault default without a usage specification at certificate creation time
NonRepudiation	sign, verify	N/A
crlsign	sign, verify	N/A

Certificate Issuer

A Key Vault certificate object holds a configuration used to communicate with a selected certificate issuer provider to order x509 certificates.

- Key Vault partners with following certificate issuer providers for TLS/SSL certificates

PROVIDER NAME	LOCATIONS
DigiCert	Supported in all key vault service locations in public cloud and Azure Government
GlobalSign	Supported in all key vault service locations in public cloud and Azure Government

Before a certificate issuer can be created in a Key Vault, following prerequisite steps 1 and 2 must be successfully accomplished.

1. Onboard to Certificate Authority (CA) Providers

- An organization administrator must on-board their company (ex. Contoso) with at least one CA provider.

2. Admin creates requester credentials for Key Vault to enroll (and renew) TLS/SSL certificates

- Provides the configuration to be used to create an issuer object of the provider in the key vault

For more information on creating Issuer objects from the Certificates portal, see the [Key Vault Certificates blog](#)

Key Vault allows for creation of multiple issuer objects with different issuer provider configuration. Once an issuer object is created, its name can be referenced in one or multiple certificate policies. Referencing the issuer object instructs Key Vault to use configuration as specified in the issuer object when requesting the x509 certificate from CA provider during the certificate creation and renewal.

Issuer objects are created in the vault and can only be used with KV certificates in the same vault.

Certificate contacts

Certificate contacts contain contact information to send notifications triggered by certificate lifetime events. The contacts information is shared by all the certificates in the key vault. A notification is sent to all the

specified contacts for an event for any certificate in the key vault.

If a certificate's policy is set to auto renewal, then a notification is sent on the following events.

- Before certificate renewal
- After certificate renewal, stating if the certificate was successfully renewed, or if there was an error, requiring manual renewal of the certificate.

When a certificate policy that is set to be manually renewed (email only), a notification is sent when it's time to renew the certificate.

Certificate Access Control

Access control for certificates is managed by Key Vault, and is provided by the Key Vault that contains those certificates. The access control policy for certificates is distinct from the access control policies for keys and secrets in the same Key Vault. Users may create one or more vaults to hold certificates, to maintain scenario appropriate segmentation and management of certificates.

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

- Permissions for certificate management operations
 - *get*: Get the current certificate version, or any version of a certificate
 - *list*: List the current certificates, or versions of a certificate
 - *update*: Update a certificate
 - *create*: Create a Key Vault certificate
 - *import*: Import certificate material into a Key Vault certificate
 - *delete*: Delete a certificate, its policy, and all of its versions
 - *recover*: Recover a deleted certificate
 - *backup*: Back up a certificate in a key vault
 - *restore*: Restore a backed-up certificate to a key vault
 - *managecontacts*: Manage Key Vault certificate contacts
 - *manageissuers*: Manage Key Vault certificate authorities/issuers
 - *getissuers*: Get a certificate's authorities/issuers
 - *listissuers*: List a certificate's authorities/issuers
 - *setissuers*: Create or update a Key Vault certificate's authorities/issuers
 - *deleteissuers*: Delete a Key Vault certificate's authorities/issuers
- Permissions for privileged operations
 - *purge*: Purge (permanently delete) a deleted certificate

For more information, see the [Certificate operations in the Key Vault REST API reference](#). For information on establishing permissions, see [Vaults - Create or Update](#) and [Vaults - Update Access Policy](#).

Azure Storage account key management

Key Vault can manage Azure storage account keys:

- Internally, Key Vault can list (sync) keys with an Azure storage account.
- Key Vault regenerates (rotates) the keys periodically.
- Key values are never returned in response to caller.
- Key Vault manages keys of both storage accounts and classic storage accounts.

For more information, see [Azure Key Vault Storage Account Keys](#)

Storage account access control

The following permissions can be used when authorizing a user or application principal to perform operations on a managed storage account:

- Permissions for managed storage account and SAS-definition operations
 - *get*: Gets information about a storage account
 - *list*: List storage accounts managed by a Key Vault
 - *update*: Update a storage account
 - *delete*: Delete a storage account
 - *recover*: Recover a deleted storage account
 - *backup*: Back up a storage account
 - *restore*: Restore a backed-up storage account to a Key Vault
 - *set*: Create or update a storage account
 - *regeneratekey*: Regenerate a specified key value for a storage account
 - *getsas*: Get information about a SAS definition for a storage account
 - *listsas*: List storage SAS definitions for a storage account
 - *deletesas*: Delete a SAS definition from a storage account
 - *setsas*: Create or update a new SAS definition/attributes for a storage account
- Permissions for privileged operations
 - *purge*: Purge (permanently delete) a managed storage account

For more information, see the [Storage account operations in the Key Vault REST API reference](#). For information on establishing permissions, see [Vaults - Create or Update](#) and [Vaults - Update Access Policy](#).

See Also

- [Authentication, requests, and responses](#)
- [Key Vault Developer's Guide](#)

Get started with Key Vault certificates

4 minutes to read • [Edit Online](#)

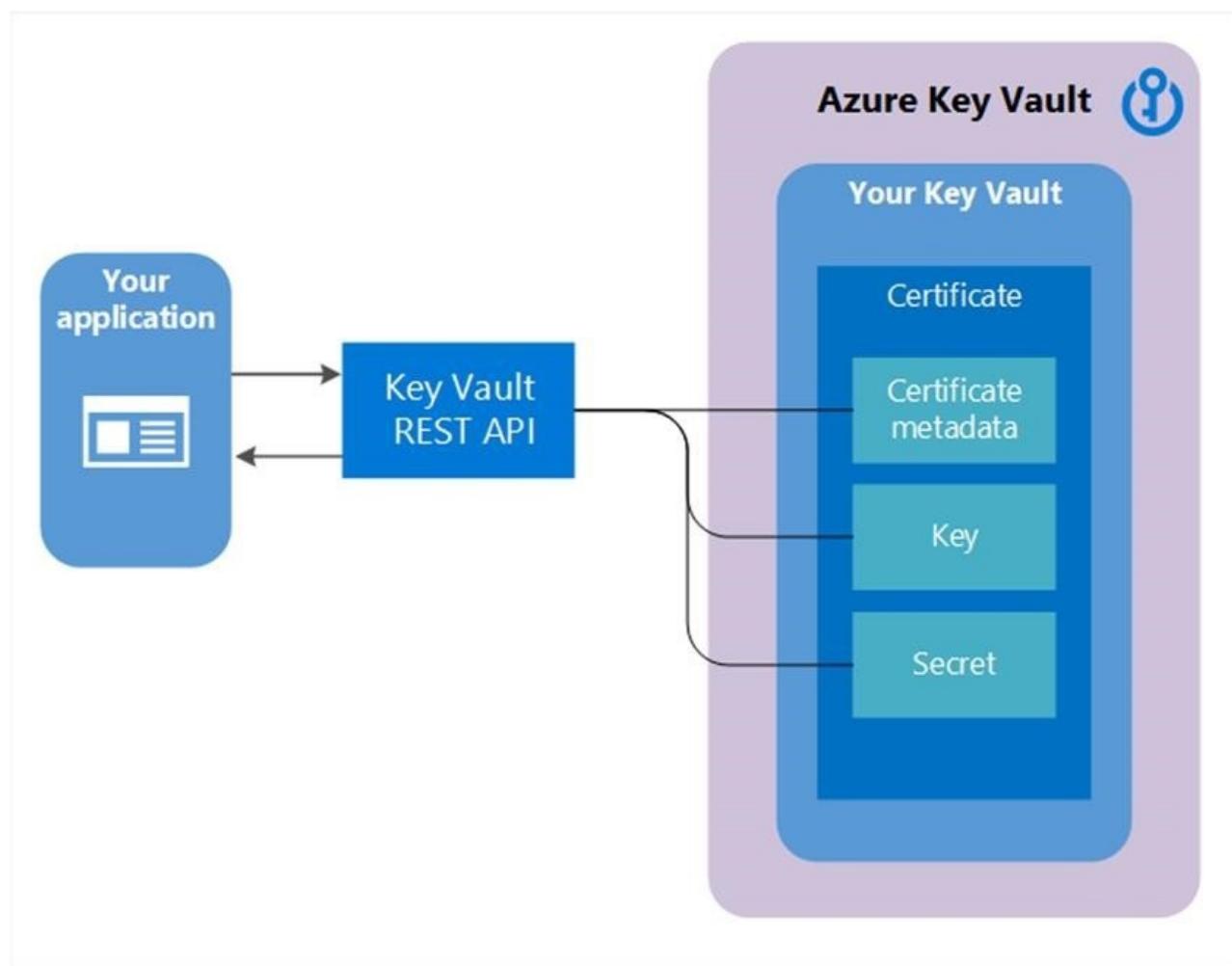
The following scenarios outline several of the primary usages of Key Vault's certificate management service including the additional steps required for creating your first certificate in your key vault.

The following are outlined:

- Creating your first Key Vault certificate
- Creating a certificate with a Certificate Authority that is partnered with Key Vault
- Creating a certificate with a Certificate Authority that is not partnered with Key Vault
- Import a certificate

Certificates are complex objects

Certificates are composed of three interrelated resources linked together as a Key Vault certificate; certificate metadata, a key, and a secret.



Creating your first Key Vault certificate

Before a certificate can be created in a Key Vault (KV), prerequisite steps 1 and 2 must be successfully accomplished and a key vault must exist for this user / organization.

Step 1 - Certificate Authority (CA) Providers

- On-boarding as the IT Admin, PKI Admin or anyone managing accounts with CAs, for a given company (ex. Contoso) is a prerequisite to using Key Vault certificates.

The following CAs are the current partnered providers with Key Vault:

- DigiCert - Key Vault offers OV TLS/SSL certificates with DigiCert.
- GlobalSign - Key Vault offers OV TLS/SSL certificates with GlobalSign.

Step 2 - An account admin for a CA provider creates credentials to be used by Key Vault to enroll, renew, and use TLS/SSL certificates via Key Vault.

Step 3 - A Contoso admin, along with a Contoso employee (Key Vault user) who owns certificates, depending on the CA, can get a certificate from the admin or directly from the account with the CA.

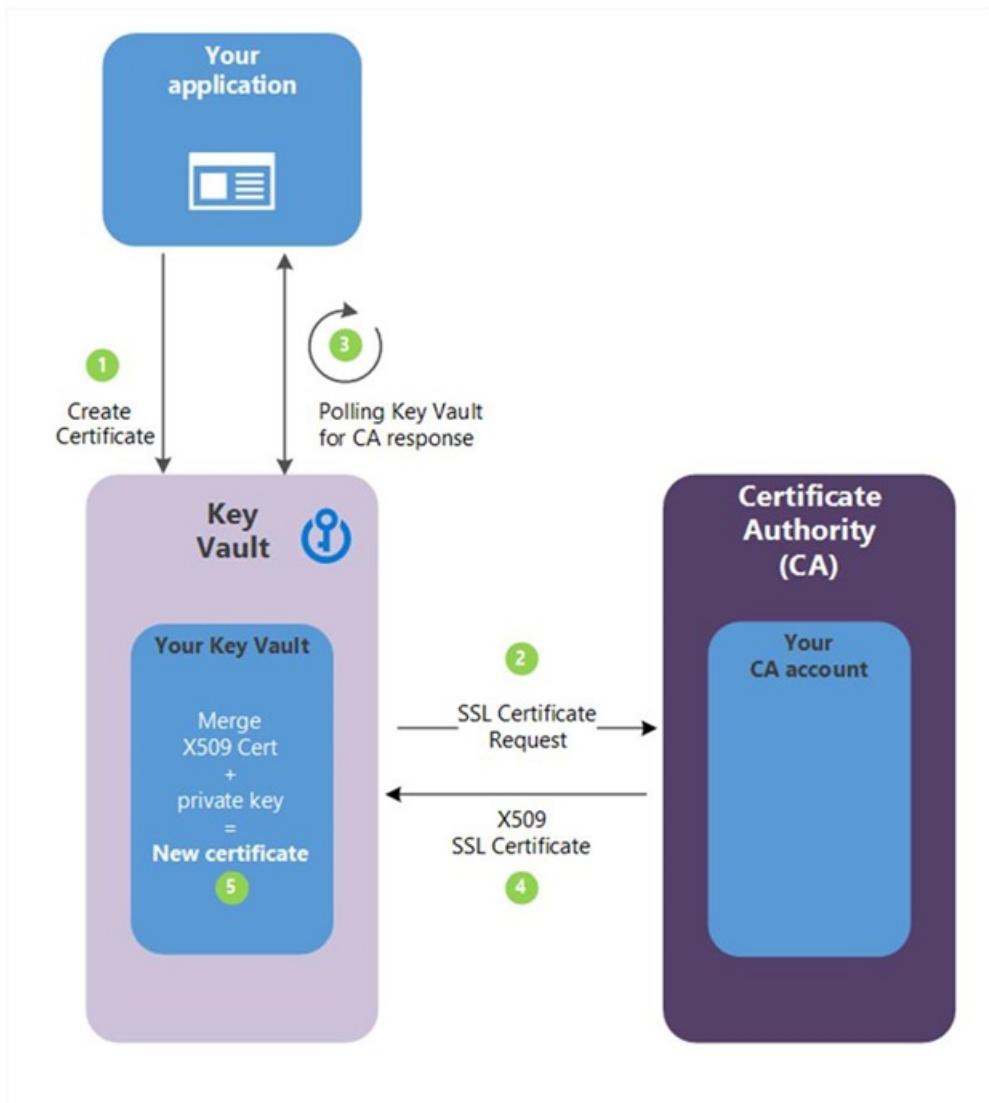
- Begin an add credential operation to a key vault by [setting a certificate issuer](#) resource. A certificate issuer is an entity represented in Azure Key Vault (KV) as a CertificateIssuer resource. It is used to provide information about the source of a KV certificate; issuer name, provider, credentials, and other administrative details.
 - Ex. MyDigiCertIssuer
 - Provider
 - Credentials – CA account credentials. Each CA has its own specific data.

For more information on creating accounts with CA Providers, see the related post on the [Key Vault blog](#).

Step 3.1 - Set up [certificate contacts](#) for notifications. This is the contact for the Key Vault user. Key Vault does not enforce this step.

Note - This process, through step 3.1, is a onetime operation.

Creating a certificate with a CA partnered with Key Vault



Step 4 - The following descriptions correspond to the green numbered steps in the preceding diagram.

- (1) - In the diagram above, your application is creating a certificate which internally begins by creating a key in your key vault.
- (2) - Key Vault sends an TLS/SSL Certificate Request to the CA.
- (3) - Your application polls, in a loop and wait process, for your Key Vault for certificate completion. The certificate creation is complete when Key Vault receives the CA's response with x509 certificate.
- (4) - The CA responds to Key Vault's TLS/SSL Certificate Request with an X509 TLS/SSL Certificate.
- (5) - Your new certificate creation completes with the merger of the X509 Certificate for the CA.

Key Vault user – creates a certificate by specifying a policy

- Repeat as needed
- Policy constraints
 - X509 properties
 - Key properties
 - Provider reference - > ex. MyDigiCertIssure
 - Renewal information - > ex. 90 days before expiry
- A certificate creation process is usually an asynchronous process and involves polling your key vault for the state of the create certificate operation.

[Get certificate operation](#)

- Status: completed, failed with error information or, canceled
- Because of the delay to create, a cancel operation can be initiated. The cancel may or may not be effective.

Import a certificate

Alternatively – a cert can be imported into Key Vault – PFX or PEM.

For more information on PEM format, see the certificates section of [About keys, secrets, and certificates](#).

Import certificate – requires a PEM or PFX to be on disk and have a private key.

- You must specify: vault name and certificate name (policy is optional)
- PEM / PFX files contains attributes that KV can parse and use to populate the certificate policy. If a certificate policy is already specified, KV will try to match data from PFX / PEM file.
- Once the import is final, subsequent operations will use the new policy (new versions).
- If there are no further operations, the first thing the Key Vault does is send an expiration notice.
- Also, the user can edit the policy, which is functional at the time of import but, contains defaults where no information was specified at import. Ex. no issuer info

Formats of Import we support

We support the following type of Import for PEM file format. A single PEM encoded certificate along with a PKCS#8 encoded, unencrypted key which has the following

-----BEGIN CERTIFICATE----- -----END CERTIFICATE-----

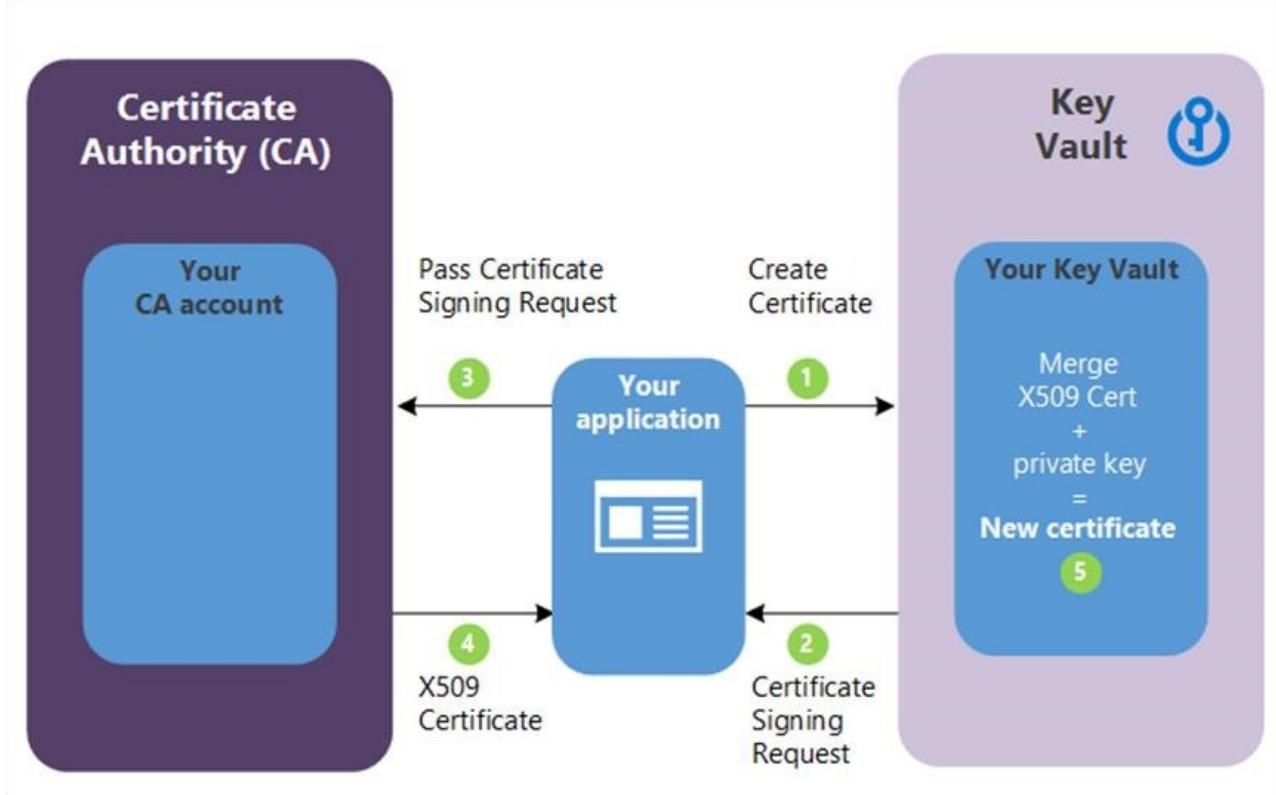
-----BEGIN PRIVATE KEY----- -----END PRIVATE KEY-----

On certificate merge we support 2 PEM based formats. You can either merge a single PKCS#8 encoded certificate or a base64 encoded P7B file. -----BEGIN CERTIFICATE----- -----END CERTIFICATE-----

We currently don't support EC keys in PEM format.

Creating a certificate with a CA not partnered with Key Vault

This method allows working with other CAs than Key Vault's partnered providers, meaning your organization can work with a CA of its choice.



The following step descriptions correspond to the green lettered steps in the preceding diagram.

- (1) - In the diagram above, your application is creating a certificate, which internally begins by creating a key in your key vault.
- (2) - Key Vault returns to your application a Certificate Signing Request (CSR).
- (3) - Your application passes the CSR to your chosen CA.
- (4) - Your chosen CA responds with an X509 Certificate.
- (5) - Your application completes the new certificate creation with a merger of the X509 Certificate from your CA.

See Also

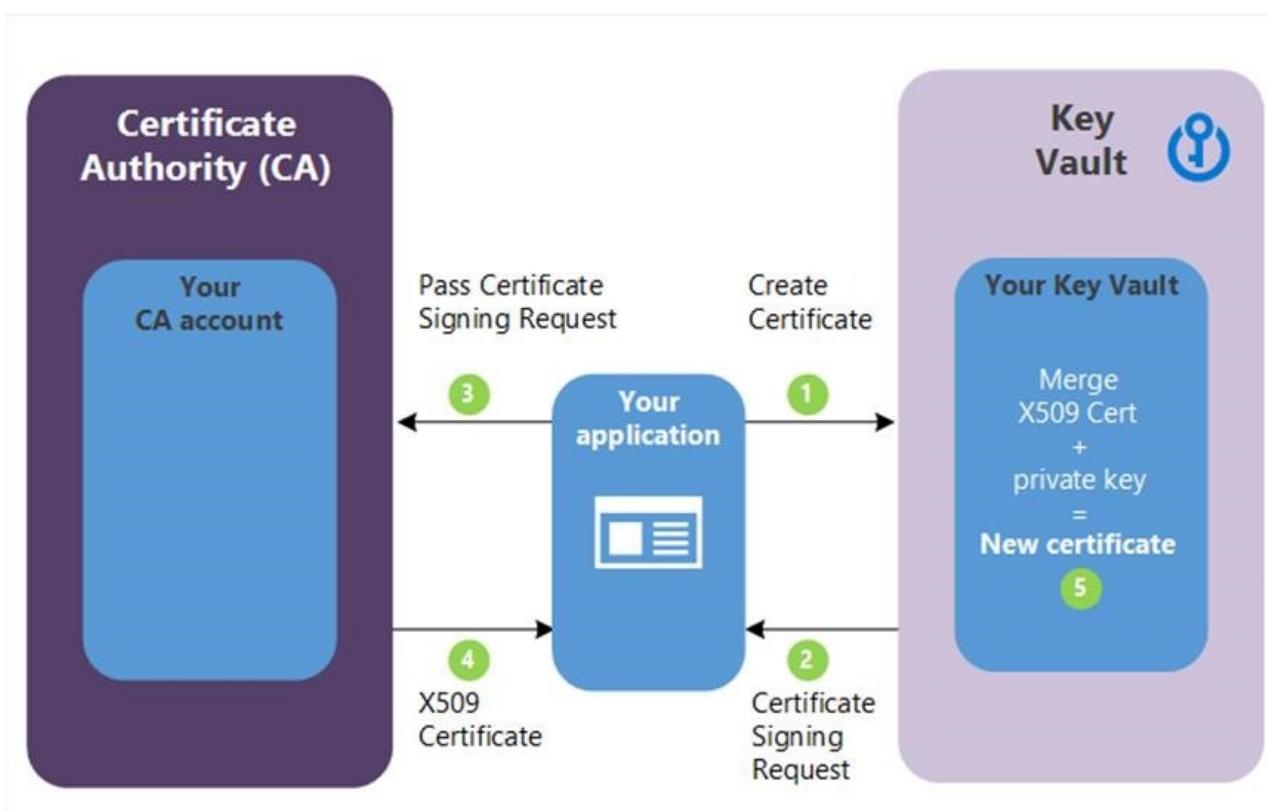
- [About keys, secrets, and certificates](#)

Certificate creation methods

4 minutes to read • [Edit Online](#)

A Key Vault (KV) certificate can be either created or imported into a key vault. When a KV certificate is created the private key is created inside the key vault and never exposed to certificate owner. The following are ways to create a certificate in Key Vault:

- **Create a self-signed certificate:** This will create a public-private key pair and associate it with a certificate. The certificate will be signed by its own key.
- **Create a new certificate manually:** This will create a public-private key pair and generate an X.509 certificate signing request. The signing request can be signed by your registration authority or certification authority. The signed x509 certificate can be merged with the pending key pair to complete the KV certificate in Key Vault. Although this method requires more steps, it does provide you with greater security because the private key is created in and restricted to Key Vault. This is explained in the diagram below.

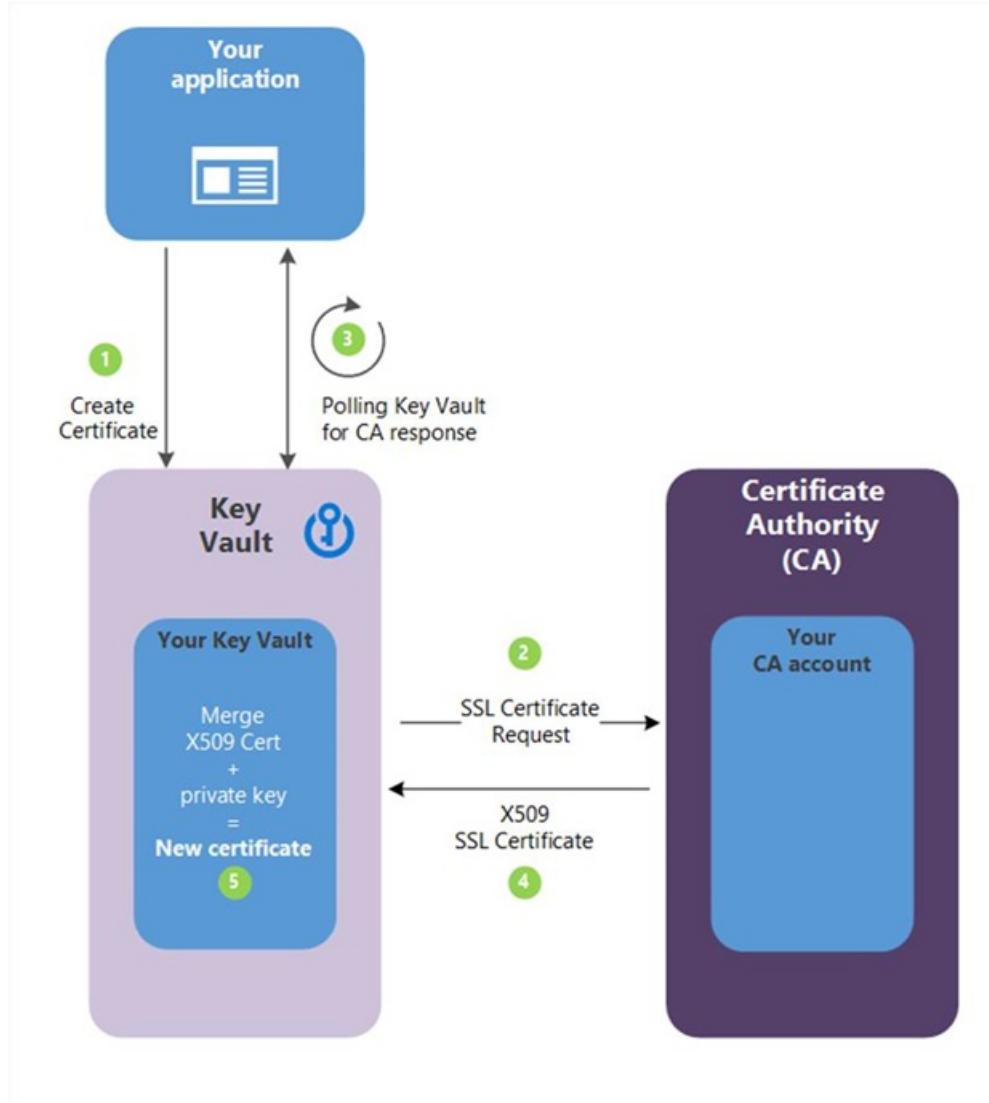


The following descriptions correspond to the green lettered steps in the preceding diagram.

1. In the diagram above, your application is creating a certificate which internally begins by creating a key in your key vault.
2. Key Vault returns to your application a Certificate Signing Request (CSR)
3. Your application passes the CSR to your chosen CA.
4. Your chosen CA responds with an X509 Certificate.
5. Your application completes the new certificate creation with a merger of the X509 Certificate from your CA.

- **Create a certificate with a known issuer provider:** This method requires you to do a one-time task of creating an issuer object. Once an issuer object is created in your key vault, its name can be referenced in the policy of the KV certificate. A request to create such a KV certificate will create a key pair in the vault and communicate with the issuer provider service using the information in the referenced issuer object to get an

x509 certificate. The x509 certificate is retrieved from the issuer service and is merged with the key pair to complete the KV certificate creation.



The following descriptions correspond to the green lettered steps in the preceding diagram.

1. In the diagram above, your application is creating a certificate which internally begins by creating a key in your key vault.
2. Key Vault sends an TLS/SSL Certificate Request to the CA.
3. Your application polls, in a loop and wait process, for your Key Vault for certificate completion. The certificate creation is complete when Key Vault receives the CA's response with x509 certificate.
4. The CA responds to Key Vault's TLS/SSL Certificate Request with an TLS/SSL X.509 certificate.
5. Your new certificate creation completes with the merger of the TLS/SSL X.509 certificate for the CA.

Asynchronous process

KV certificate creation is an asynchronous process. This operation will create a KV certificate request and return an http status code of 202 (Accepted). The status of the request can be tracked by polling the pending object created by this operation. The full URI of the pending object is returned in the LOCATION header.

When a request to create a KV certificate completes, the status of the pending object will change to "completed" from "inprogress", and a new version of the KV certificate will be created. This will become the current version.

First creation

When a KV certificate is created for the first time, an addressable key and secret is also created with the same name

as that of the certificate. If the name is already in use, then the operation will fail with an http status code of 409 (conflict). The addressable key and secret get their attributes from the KV certificate attributes. The addressable key and secret created this way are marked as managed keys and secrets, whose lifetime is managed by Key Vault. Managed keys and secrets are read-only. Note: If a KV certificate expires or is disabled, the corresponding key and secret will become inoperable.

If this is the first operation to create a KV certificate then a policy is required. A policy can also be supplied with successive create operations to replace the policy resource. If a policy is not supplied, then the policy resource on the service is used to create a next version of KV certificate. Note that while a request to create a next version is in progress, the current KV certificate, and corresponding addressable key and secret, remain unchanged.

Self-issued certificate

To create a self-issued certificate, set the issuer name as "Self" in the certificate policy as shown in following snippet from certificate policy.

```
"issuer": {  
    "name": "Self"  
}
```

If the issuer name is not specified, then the issuer name is set to "Unknown". When issuer is "Unknown", the certificate owner will have to manually get a x509 certificate from the issuer of his/her choice, then merge the public x509 certificate with the key vault certificate pending object to complete the certificate creation.

```
"issuer": {  
    "name": "Unknown"  
}
```

Partnered CA Providers

Certificate creation can be completed manually or using a "Self" issuer. Key Vault also partners with certain issuer providers to simplify the creation of certificates. The following types of certificates can be ordered for key vault with these partner issuer providers.

PROVIDER	CERTIFICATE TYPE
DigiCert	Key Vault offers OV or EV SSL certificates with DigiCert
GlobalSign	Key Vault offers OV or EV SSL certificates with GlobalSign

A certificate issuer is an entity represented in Azure Key Vault (KV) as a CertificateIssuer resource. It is used to provide information about the source of a KV certificate; issuer name, provider, credentials, and other administrative details.

Note that when an order is placed with the issuer provider, it may honor or override the x509 certificate extensions and certificate validity period based on the type of certificate.

Authorization: Requires the certificates/create permission.

See Also

- [About keys, secrets and certificates](#)

- Monitor and manage certificate creation

Monitor and manage certificate creation

5 minutes to read • [Edit Online](#)

Applies To: Azure

The following

The scenarios / operations outlined in this article are:

- Request a KV Certificate with a supported issuer
- Get pending request - request status is "inProgress"
- Get pending request - request status is "complete"
- Get pending request - pending request status is "canceled" or "failed"
- Get pending request - pending request status is "deleted" or "overwritten"
- Create (or Import) when pending request exists - status is "inProgress"
- Merge when pending request is created with an issuer (DigiCert, for example)
- Request a cancellation while the pending request status is "inProgress"
- Delete a pending request object
- Create a KV certificate manually
- Merge when a pending request is created - manual certificate creation

Request a KV Certificate with a supported issuer

METHOD	REQUEST URI
POST	<code>https://mykeyvault.vault.azure.net/certificates/mycert1/create?api-version={api-version}</code>

The following examples require an object named "mydigicert" to already be available in your key vault with the issuer provider as DigiCert. The certificate issuer is an entity represented in Azure Key Vault (KV) as a CertificateIssuer resource. It is used to provide information about the source of a KV certificate; issuer name, provider, credentials, and other administrative details.

Request

```
{  
  "policy": {  
    "x509_props": {  
      "subject": "CN=MyCertSubject1"  
    },  
    "issuer": {  
      "name": "mydigicert",  
      "cty": "OV-SSL",  
    }  
  }  
}
```

Response

```

StatusCode: 202, ReasonPhrase: 'Accepted'
Location: "https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}&request_id=a76827a18b63421c917da80f28e9913d"
{
  "id": "https://mykeyvault.vault.azure.net/certificates/mycert1/pending",
  "issuer": {
    "name": "mydigicert"
  },
  "csr": "MIICq.....DD5Lp5cqXg==",
  "cancellation_requested": false,
  "status": "InProgress",
  "status_details": "Pending certificate created. Certificate request is in progress. This may take some time based on the issuer provider. Please check again later",
  "request_id": "a76827a18b63421c917da80f28e9913d"
}

```

Get pending request - request status is "inProgress"

METHOD	REQUEST URI
GET	<code>https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}</code>

Request

GET

```
"https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}&request_id=a76827a18b63421c917da80f28e9913d"
```

OR

GET `"https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}"`

NOTE

If `request_id` is specified in the query, it acts like a filter. If the `request_id` in the query and in the pending object are different, an http status code of 404 is returned.

Response

```

StatusCode: 200, ReasonPhrase: 'OK'
{
  "id": "https://mykeyvault.vault.azure.net/certificates/mycert1/pending",
  "issuer": {
    "name": "{issuer-name}"
  },
  "csr": "MIICq.....DD5Lp5cqXg==",
  "cancellation_requested": false,
  "status": "InProgress",
  "status_details": "...",
  "request_id": "a76827a18b63421c917da80f28e9913d"
}

```

Get pending request - request status is "complete"

Request

METHOD	REQUEST URI
GET	<code>https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}</code>

GET

```
"https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}&request_id=a76827a18b63421c917da80f28e9913d"
```

OR

GET ["https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}"](https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version})

Response

```
StatusCode: 200, ReasonPhrase: 'OK'  
{  
  "id": "https://mykeyvault.vault.azure.net/certificates/mycert1/pending",  
  "issuer": {  
    "name": "{issuer-name}"  
  },  
  "csr": "MIICq.....DD5Lp5cqXg==",  
  "cancellation_requested": false,  
  "status": "completed",  
  "request_id": "a76827a18b63421c917da80f28e9913d",  
  "target": "https://mykeyvault.vault.azure.net/certificates/mycert1?api-version={api-version}"  
}
```

Get pending request - pending request status is "canceled" or "failed"

Request

METHOD	REQUEST URI
GET	https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}

GET

```
"https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}&request_id=a76827a18b63421c917da80f28e9913d"
```

OR

GET ["https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}"](https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version})

Response

```
StatusCode: 200, ReasonPhrase: 'OK'  
{  
  "id": "https://mykeyvault.vault.azure.net/certificates/mycert1/pending",  
  "issuer": {  
    "name": "{issuer-name}"  
  },  
  "csr": "MIICq.....DD5Lp5cqXg==",  
  "cancellation_requested": false,  
  "status": "failed",  
  "status_details": "",  
  "request_id": "a76827a18b63421c917da80f28e9913d",  
  "error": {  
    "code": "<errorcode>",  
    "message": "<message>"  
  }  
}
```

NOTE

The value of the *errorcode* can be "Certificate issuer error" or "Request rejected" based on issuer or user error respectively.

Get pending request - pending request status is "deleted" or "overwritten"

A pending object can be deleted or overwritten by a create/import operation when its status is not "inProgress."

METHOD	REQUEST URI
GET	<code>https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}</code>

Request

GET

```
"https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}&request_id=a76827a18b63421c917da80f28e9913d"
```

OR

GET `"https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}"`

Response

```
StatusCode: 404, ReasonPhrase: 'Not Found'  
{  
    "error": {  
        "code": "PendingCertificateNotFound",  
        "message": "..."  
    }  
}
```

Create (or Import) when pending request exists - status is "inProgress"

A pending object has four possible states; "inprogress", "canceled", "failed", or "completed."

When a pending request's state is "inprogress", create (and import) operations will fail with an http status code of 409 (conflict).

To fix a conflict:

- If the certificate is being manually created, you can either complete the KV certificate by doing a merge or delete on the pending object.
- If the certificate is being created with an issuer, you can wait until the certificate completes, fails or is canceled. Alternatively, you can delete the pending object.

NOTE

Deleting a pending object may or may not cancel the x509 certificate request with the provider.

METHOD	REQUEST URI
POST	<code>https://mykeyvault.vault.azure.net/certificates/mycert1/create?api-version={api-version}</code>

Request

```
{
  "policy": {
    "x509_props": {
      "subject": "CN=MyCertSubject1"
    },
    "issuer": {
      "name": "mydigicert"
    }
  }
}
```

Response

```
StatusCode: 409, ReasonPhrase: 'Conflict'
{
  "error": {
    "code": "Forbidden",
    "message": "A new key vault certificate can not be created or imported while a pending key vault certificate's status is inProgress."
  }
}
```

Merge when pending request is created with an issuer

Merge is not allowed when a pending object is created with an issuer but is allowed when its state is "inProgress."

If the request to create the x509 certificate fails or cancels for some reason, and if an x509 certificate can be retrieved by out-of-band means, a merge operation can be done to complete the KV certificate.

METHOD	REQUEST URI
POST	https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}

Request

```
{
  "x5c": [ "MIICxTCCAbi.....trimmed for brevity.....EPAQj8=" ]
}
```

Response

```
StatusCode: 403, ReasonPhrase: 'Forbidden'
{
  "error": {
    "code": "Forbidden",
    "message": "Merge is forbidden on pending object created with issuer : <issuer-name> while it is in progress."
  }
}
```

Request a cancellation while the pending request status is "inProgress"

A cancellation can only be requested. A request may or may not be canceled. If a request is not "inProgress", an http status of 400 (Bad Request) is returned.

METHOD	REQUEST URI
DELETE	https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}

METHOD	REQUEST URI
PATCH	<code>https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}</code>

Request

PATCH

```
"https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}&request_id=a76827a18b63421c917da80f28e9913d"
```

OR

PATCH `"https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}"`

```
{
  "cancellation_requested": true
}
```

Response

```
StatusCode: 200, ReasonPhrase: 'OK'
{
  "id": "https://mykeyvault.vault.azure.net/certificates/mycert1/pending",
  "issuer": {
    "name": "{issuer-name}"
  },
  "csr": "MIICq.....DD5Lp5cqXg==",
  "cancellation_requested": true,
  "status": "inProgress",
  "status_details": "...",
  "request_id": "a76827a18b63421c917da80f28e9913d"
}
```

Delete a pending request object

NOTE

Deleting the pending object may or may not cancel the x509 certificate request with the provider.

METHOD	REQUEST URI
DELETE	<code>https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}</code>

Request

DELETE

```
"https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}&request_id=a76827a18b63421c917da80f28e9913d"
```

OR

DELETE `"https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}"`

Response

```

StatusCode: 200, ReasonPhrase: 'OK'
{
  "id": "https://mykeyvault.vault.azure.net/certificates/mycert1/pending",
  "issuer": {
    "name": "{issuer-name}"
  },
  "csr": "MIICq.....DD5Lp5cqXg==",
  "cancellation_requested": false,
  "status": "inProgress",
  "request_id": "a76827a18b63421c917da80f28e9913d",
}

```

Create a KV certificate manually

You can create a certificate issued with a CA of your choice through a manual creation process. Set the name of the issuer to "Unknown" or do not specify the issuer field.

METHOD	REQUEST URI
POST	https://mykeyvault.vault.azure.net/certificates/mycert1/create? api-version={api-version}

Request

```
{
  "policy": {
    "x509_props": {
      "subject": "CN=MyCertSubject1"
    },
    "issuer": {
      "name": "Unknown"
    }
  }
}
```

Response

```

StatusCode: 202, ReasonPhrase: 'Accepted'
Location: "https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-
version}&request_id=a76827a18b63421c917da80f28e9913d"
{
  "id": "https://mykeyvault.vault.azure.net/certificates/mycert1/pending",
  "issuer": {
    "name": "Unknown"
  },
  "csr": "MIICq.....DD5Lp5cqXg==",
  "status": "inProgress",
  "status_details": "Pending certificate created. Please Perform Merge to complete the request.",
  "request_id": "a76827a18b63421c917da80f28e9913d"
}

```

Merge when a pending request is created - manual certificate creation

METHOD	REQUEST URI
POST	https://mykeyvault.vault.azure.net/certificates/mycert1/pending/ api-version={api-version}

Request

```
{
  "x5c": [ "MIICxTCCAbi.....trimmed for brevity.....EPAQj8=" ]
}
```

ELEMENT NAME	REQUIRED	TYPE	VERSION	DESCRIPTION
x5c	Yes	array	<introducing version>	X509 certificate chain as base 64 string array.

Response

```
StatusCode: 201, ReasonPhrase: 'Created'
Location: "https://mykeyvault.vault.azure.net/certificates/mycert1?api-version={api-version}"
{
  "id": "https://mykeyvault.vault.azure.net/certificates/mycert1/f366e1a9dd774288ad84a45a5f620352",
  "kid": "https://mykeyvault.vault.azure.net/keys/mycert1/f366e1a9dd774288ad84a45a5f620352",
  "sid": "mykeyvault.vault.azure.net/secrets/mycert1/f366e1a9dd774288ad84a45a5f620352",
  "cer": "...de34534....",
  "x5t": "n14q2wbvyXr71Pcb58NivuiwJKk",
  "attributes": {
    "enabled": true,
    "exp": 1530394215,
    "nbf": 1435699215,
    "created": 1435699919,
    "updated": 1435699919
  },
  "pending": {
    "id": "https://mykeyvault.vault.azure.net/certificates/mycert1/pending"
  },
  "policy": {
    "id": "https://mykeyvault.vault.azure.net/certificates/mycert1/policy",
    "key_props": {
      "exportable": false,
      "kty": "RSA",
      "key_size": 2048,
      "reuse_key": false
    },
    "secret_props": {
      "contentType": "application/x-pkcs12"
    },
    "x509_props": {
      "subject": "CN=Mycert1",
      "ekus": ["1.3.6.1.5.5.7.3.1", "1.3.6.1.5.5.7.3.2"],
      "validity_months": 12
    },
    "lifetime_actions": [
      {
        "trigger": {
          "lifetime_percentage": 80
        },
        "action": {
          "action_type": "EmailContacts"
        }
      }
    ],
    "issuer": {
      "name": "Unknown"
    },
    "attributes": {
      "enabled": true,
      "created": 1435699811,
      "updated": 1435699811
    }
  }
}
```

See Also

- [About keys, secrets, and certificates](#)

Security recommendations for Azure Key Vault

2 minutes to read • [Edit Online](#)

This article contains security recommendations for Azure Key Vault. Implementing these recommendations will help you fulfill your security obligations as described in our shared responsibility model. For more information on what Microsoft does to fulfill service provider responsibilities, read [Shared responsibilities for cloud computing](#).

Some of the recommendations included in this article can be automatically monitored by Azure Security Center. Azure Security Center is the first line of defense in protecting your resources in Azure. It periodically analyzes the security state of your Azure resources to identify potential security vulnerabilities. It then provides you with recommendations on how to address them.

- For more information on Azure Security Center recommendations, see [Security recommendations in Azure Security Center](#).
- For information on Azure Security Center see the [What is Azure Security Center?](#)

Data protection

RECOMMENDATION	COMMENTS	SECURITY CENTER
Enable soft delete	Soft delete allows you to recover deleted vaults and vault objects	-
Limit access to vault data	Follow the principle of least privilege and limit which members of your organization have access to vault data	-

Identity and access management

RECOMMENDATION	COMMENTS	SECURITY CENTER
Limit the number of users with contributor access	If a user has Contributor permissions to a key vault management plane, the user can grant themselves access to the data plane by setting a Key Vault access policy. You should tightly control who has Contributor role access to your key vaults. Ensure that only those with a need for access authorized persons can access and manage your vaults. You can read Secure access to a key vault	-

Monitoring

RECOMMENDATION	COMMENTS	SECURITY CENTER
Diagnostics logs in Key Vault should be enabled	Enable logs and retain them up to a year. This enables you to recreate activity trails for investigation purposes when a security incident occurs or your network is compromised.	Yes

RECOMMENDATION	COMMENTS	SECURITY CENTER
Restrict who can access your Azure Key vault logs	Key Vault logs save information about the activities performed on your vault such as creation or deletion of vaults, keys, secrets and may be used during an investigation	-

Networking

RECOMMENDATION	COMMENTS	SECURITY CENTER
Limit network exposure	Network access should be limited to the virtual networks used by solutions requiring vault access. Review information on Virtual network service endpoints for Azure Key Vault	-

Next steps

Check with your application provider to see if there are additional security requirements. For more information on developing secure applications, see [Secure Development Documentation](#).

Provide Key Vault authentication with a managed identity

3 minutes to read • [Edit Online](#)

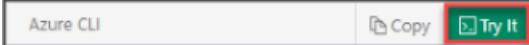
A managed identity from Azure Active Directory allows your app to easily access other Azure AD-protected resources. The identity is managed by the Azure platform and does not require you to provision or rotate any secrets. For more information, see [Managed identities for Azure resources](#).

This article shows you how to create a managed identity for an App Service application and use it to access Azure Key Vault. For applications hosted in Azure VMs, see [Use a Windows VM system-assigned managed identity to access Azure Key Vault](#).

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

Prerequisites

To complete this guide, you must have the following resources.

- A key vault. You can use an existing key vault, or create a new one by following the steps in one of these quickstarts:
 - [Create a key vault with the Azure CLI](#)

- [Create a key vault with Azure PowerShell](#)
- [Create a key vault with the Azure portal](#).
- An existing App Service application to which to grant key vault access. You can quickly create one by following the steps in the [App Service documentation](#).
- [Azure CLI](#) or [Azure PowerShell](#). Alternatively, you can use the [Azure portal](#).

Adding a system-assigned identity

First, you must add a system-assigned identity to an application.

Azure portal

To set up a managed identity in the portal, you will first create an application as normal and then enable the feature.

1. If using a function app, navigate to **Platform features**. For other app types, scroll down to the **Settings** group in the left navigation.
2. Select **Managed identity**.
3. Within the **System assigned** tab, switch **Status** to **On**. Click **Save**.

Azure CLI

This quickstart requires the Azure CLI version 2.0.4 or later. Run `az --version` to find your current version. If you need to install or upgrade, see [Install the Azure CLI](#).

To sign in with Azure CLI, use the `az login` command:

```
az login
```

For more information on login options with the Azure CLI, see [Sign in with Azure CLI](#).

To create the identity for this application, use the Azure CLI `az webapp identity assign` command or `az functionapp identity assign` command:

```
az webapp identity assign --name myApp --resource-group myResourceGroup
```

```
az functionapp identity assign --name myApp --resource-group myResourceGroup
```

Make a note of the `PrincipalId`, which will be needed in next section.

```
{
  "principalId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "tenantId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "type": "SystemAssigned"
}
```

Grant your app access to Key Vault

Azure portal

1. Navigate to Key Vault resource.
2. Select **Access policies** and click **Add Access Policy**.
3. In **Secret permissions**, select **Get, List**.
4. Choose **Select Principal**, and in the search field enter the name of the app. Select the app in the result list and click **Select**.
5. Click **Add** to finish adding the new access policy.

The screenshot shows the Azure portal interface for adding an access policy to a Key Vault. On the left, the 'Add access policy' dialog is open, showing fields for 'Configure from template (optional)', 'Key permissions' (0 selected), 'Secret permissions' (2 selected), 'Certificate permissions' (0 selected), 'Select principal' (None selected), 'Authorized application' (None selected), and an 'Add' button. On the right, a 'Principal' selection modal is displayed, listing four options: 'systemass' (selected), 'systemassigned-app', 'systemAssignedIdentity-APP', and 'systemassigned-linux'. Below the list is a 'Selected member:' section showing 'systemassigned-app' with a 'Remove' link. At the bottom of the modal is a 'Select' button.

Azure CLI

To grant your application access to your key vault, use the Azure CLI `az keyvault set-policy` command, supplying the **ObjectId** parameter with the **principalId** you noted above.

```
az keyvault set-policy --name myKeyVault --object-id <PrincipalId> --secret-permissions get list
```

Next steps

- [Azure Key Vault security: Identity and access management](#)
- [Provide Key Vault authentication with an access control policy](#)
- [About keys, secrets, and certificates](#)
- [Secure your key vault.](#)
- [Azure Key Vault developer's guide](#)
- Review [Azure Key Vault best practices](#)

Provide Key Vault authentication with an access control policy

7 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

The simplest way to authenticate a cloud-based application to Key Vault is with a managed identity; see [Use an App Service managed identity to access Azure Key Vault](#) for details. If you are creating an on-prem application, doing local development, or otherwise unable to use a managed identity, you can instead register a service principal manually and provide access to your key vault using an access control policy.

Key vault supports up to 1024 access policy entries, with each entry granting a distinct set of permissions to a "principal": For example, this is how the console app in the [Azure Key Vault client library for .NET quickstart](#) accesses the key vault.

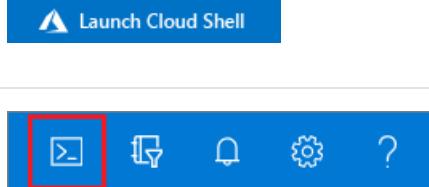
For full details on Key Vault access control, see [Azure Key Vault security: Identity and access management](#). For full details on [Keys, Secrets, and Certificates](#) access control, see:

- [Keys access control](#)
- [Secrets access control](#)
- [Certificates access control](#)

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.

2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

Prerequisites

- A key vault. You can use an existing key vault, or create a new one by following the steps in one of these quickstarts:
 - [Create a key vault with the Azure CLI](#)
 - [Create a key vault with Azure PowerShell](#)
 - [Create a key vault with the Azure portal](#).
- The [Azure CLI](#) or [Azure PowerShell](#). Alternatively, you can use the [Azure portal](#).

Grant access to your key vault

Each key vault access policy entry grants a distinct set of permissions to a principal:

- **An application** If the application is cloud-based, you should instead [Use an managed identity to access Azure Key Vault](#), if possible
- **An Azure AD group** Although key vault only supports 1024 access policy entries, you can add multiple applications and users to a single Azure AD group, and then add that group as a single entry to your access control policy.
- **A User** Giving users direct access to a key vault is **discouraged**. Ideally, users should be added to an Azure AD group, which is in turn given access to the key vault. See [Azure Key Vault security: Identity and access management](#).

Get the objectId

To give an application, Azure AD group, or user access to your key vault, you must first obtain its objectId.

Applications

The objectId for an applications corresponds with its associated service principal. For full details on service principals. see [Application and service principal objects in Azure Active Directory](#).

There are two ways to obtain an objectId for an application. The first is to register your application with Azure Active Directory. To do so, follow the steps in the quickstart [Register an application with the Microsoft identity platform](#). When registration is complete, the objectId will be listed as the "Application (client) ID".

The second is to create a service principal in a terminal window. With the Azure CLI, use the [az ad sp create-for-rbac](#) command.

```
az ad sp create-for-rbac -n "http://mySP"
```

The objectId will be listed in the output as `clientId`.

With Azure PowerShell, use the [New-AzADServicePrincipal](#) cmdlet.

```
New-AzADServicePrincipal -DisplayName mySP
```

The objectId will be listed in the output as `Id` (not `ApplicationId`).

Azure AD Groups

You can add multiple applications and users to an Azure AD group, and then give the group access to your key

vault. For more details, see the [Creating and adding members to an Azure AD group](#) section, below.

To find the objectId of an Azure AD group with the Azure CLI, use the [az ad group list](#) command. Because of the large number of groups that may be in your organization, you should also provide a search string to the `--display-name` parameter.

```
az ad group list --display-name <search-string>
```

The objectId will be returned in the JSON:

```
"objectId": "48b21bfb-74d6-48d2-868f-ff9eeaf38a64",
"objectType": "Group",
"odata.type": "Microsoft.DirectoryServices.Group",
```

To find the objectId of an Azure AD group with Azure PowerShell, use the [Get-AzADGroup](#) cmdlet. Because of the large number of groups that may be in your organization, you will probably wish to also provide a search string to the `-SearchString` parameter.

```
Get-AzADGroup -SearchString <search-string>
```

In the output, the objectId is listed as `Id`:

```
...
Id : 1cef38c4-388c-45a9-b5ae-3d88375e166a
...
```

Users

You can also add an individual user to an key vault's access control policy. **We do not recommend this.** We instead encourage you to add users to an Azure AD group, and add the group on the policies.

If you nonetheless wish to find a user with the Azure CLI, use the [az ad user show](#) command, passing the users email address to the `--id` parameter.

```
az ad user show --id <email-address-of-user>
```

The user's objectId will be returned in the output:

```
...
objectId": "f76a2a6f-3b6d-4735-9abd-14dccbf70fd9",
"objectType": "User",
...
```

To find a user with Azure PowerShell, use the [Get-AzADUser](#) cmdlet, passing the users email address to the `-UserPrincipalName` parameter.

```
Get-AzAdUser -UserPrincipalName <email-address-of-user>
```

The user's objectId will be returned in the output as `Id`.

```
...  
Id : f76a2a6f-3b6d-4735-9abd-14dccb70fd9  
Type :
```

Give the principal access to your key vault

Now that you have an objectID of your principal, you can create an access policy for your key vault that gives it get, list, set, and delete permissions for both keys and secrets, plus any additional permissions you wish.

With the Azure CLI, this is done by passing the objectId to the [az keyvault set-policy](#) command.

```
az keyvault set-policy -n <your-unique-keyvault-name> --spn <ApplicationID-of-your-service-principal> --  
secret-permissions get list set delete --key-permissions create decrypt delete encrypt get list unwrapKey  
wrapKey
```

With Azure PowerShell, this is done by passing the objectId to the [Set-AzKeyVaultAccessPolicy](#) cmdlet.

```
Set-AzKeyVaultAccessPolicy -VaultName <your-key-vault-name> -PermissionsToKeys  
create,decrypt,delete,encrypt,get,list,unwrapKey,wrapKey -PermissionsToSecrets get,list,set,delete -  
ApplicationId <Id>
```

Creating and adding members to an Azure AD group

You can create an Azure AD group, add applications and users to the group, and give the group access to your key vault. This allows you to add a number of applications to a key vault as a single access policy entry, and eliminates the need to give users direct access to your key vault (which we discourage). For more details, see [Manage app and resource access using Azure Active Directory groups](#).

Additional prerequisites

In addition to the [prerequisites above](#), you will need permissions to create/edit groups in your Azure Active Directory tenant. If you don't have permissions, you may need to contact your Azure Active Directory administrator.

If you intend to use PowerShell, you will also need the [Azure AD PowerShell module](#)

Create an Azure Active Directory group

Create a new Azure Active Directory group using the Azure CLI [az ad group create](#) command, or the Azure PowerShell [New-AzureADGroup](#) cmdlet.

```
az ad group create --display-name <your-group-display-name> --mail-nickname <your-group-mail-nickname>
```

```
New-AzADGroup -DisplayName <your-group-display-name> -MailNickname <your-group-mail-nickname>
```

In either case, make note on the newly created groups GroupId, as you will need it for the steps below.

Find the objectIds of your applications and users

You can find the objectIds of your applications using the Azure CLI with the [az ad sp list](#) command, with the `--show-mine` parameter.

```
az ad sp list --show-mine
```

Find the objectIds of your applications using Azure PowerShell with the [Get-AzADServicePrincipal](#) cmdlet, passing a search string to the `-SearchString` parameter.

```
Get-AzADServicePrincipal -SearchString <search-string>
```

To find the objectIds of your Users, follow the steps in the [Users](#) section, above.

Add your applications and users to the group

Now, add the objectIds to your newly created Azure AD group.

With the Azure CLI, use the [az ad group member add](#), passing the objectId to the `--member-id` parameter.

```
az ad group member add -g <groupId> --member-id <objectId>
```

With Azure PowerShell, use the [Add-AzADGroupMember](#) cmdlet, passing the objectId to the `-MemberObjectId` parameter.

```
Add-AzADGroupMember -TargetGroupObjectId <groupId> -MemberObjectId <objectId>
```

Give the AD group access to your key vault

Lastly, give the AD group permissions to your key vault using the Azure CLI [az keyvault set-policy](#) command, or the Azure PowerShell [Set-AzKeyVaultAccessPolicy](#) cmdlet. For examples, see the [Give the application, Azure AD group, or user access to your key vault](#) section, above.

The application also needs at least one Identity and Access Management (IAM) role assigned to the key vault. Otherwise it will not be able to login and will fail with insufficient rights to access the subscription.

Next steps

- [Azure Key Vault security: Identity and access management](#)
- [Provide Key Vault authentication with an App Service managed identity](#)
- [About keys, secrets, and certificates](#)
- [Secure your key vault.](#)
- [Azure Key Vault developer's guide](#)
- Review [Azure Key Vault best practices](#)

Import HSM-protected keys to Key Vault

2 minutes to read • [Edit Online](#)

For added assurance, when you use Azure Key Vault, you can import or generate keys in hardware security modules (HSMs) that never leave the HSM boundary. This scenario is often referred to as *bring your own key*, or BYOK. Azure Key Vault uses nCipher nShield family of HSMs (FIPS 140-2 Level 2 validated) to protect your keys.

This functionality is not available for Azure China 21Vianet.

NOTE

For more information about Azure Key Vault, see [What is Azure Key Vault?](#)

For a getting started tutorial, which includes creating a key vault for HSM-protected keys, see [What is Azure Key Vault?](#).

Supported HSMs

Transferring HSM-protected keys to Key Vault is supported via two different methods depending on the HSMs you use. Use the table below to determine which method should be used for your HSMs to generate, and then transfer your own HSM-protected keys to use with Azure Key Vault.

VENDOR NAME	VENDOR TYPE	SUPPORTED HSM MODELS	SUPPORTED HSM-KEY TRANSFER METHOD
nCipher	Manufacturer	<ul style="list-style-type: none">nShield family of HSMs	Use legacy BYOK method
Thales	Manufacturer	<ul style="list-style-type: none">SafeNet Luna HSM 7 family with firmware version 7.3 or newer	Use new BYOK method (preview)
Fortanix	HSM as a Service	<ul style="list-style-type: none">Self-Defending Key Management Service (SDKMS)	Use new BYOK method (preview)

Next steps

Follow [Key Vault Best Practices](#) to ensure security, durability and monitoring for your keys.

Import HSM-protected keys to Key Vault (preview)

5 minutes to read • [Edit Online](#)

NOTE

This feature is in preview and available only in the Azure regions *East US 2 EUAP* and *Central US EUAP*.

For added assurance when you use Azure Key Vault, you can import or generate a key in a hardware security module (HSM); the key will never leave the HSM boundary. This scenario often is referred to as *bring your own key* (BYOK). Key Vault uses the nCipher nShield family of HSMs (FIPS 140-2 Level 2 validated) to protect your keys.

Use the information in this article to help you plan for, generate, and transfer your own HSM-protected keys to use with Azure Key Vault.

NOTE

This functionality is not available for Azure China 21Vianet.

This import method is available only for [supported HSMs](#).

For more information, and for a tutorial to get started using Key Vault (including how to create a key vault for HSM-protected keys), see [What is Azure Key Vault?](#).

Overview

Here's an overview of the process. Specific steps to complete are described later in the article.

- In Key Vault, generate a key (referred to as a *Key Exchange Key* (KEK)). The KEK must be an RSA-HSM key that has only the `import` key operation. Only Key Vault Premium SKU supports RSA-HSM keys.
- Download the KEK public key as a .pem file.
- Transfer the KEK public key to an offline computer that is connected to an on-premises HSM.
- In the offline computer, use the BYOK tool provided by your HSM vendor to create a BYOK file.
- The target key is encrypted with a KEK, which stays encrypted until it is transferred to the Key Vault HSM. Only the encrypted version of your key leaves the on-premises HSM.
- A KEK that's generated inside a Key Vault HSM is not exportable. HSMs enforce the rule that no clear version of a KEK exists outside a Key Vault HSM.
- The KEK must be in the same key vault where the target key will be imported.
- When the BYOK file is uploaded to Key Vault, a Key Vault HSM uses the KEK private key to decrypt the target key material and import it as an HSM key. This operation happens entirely inside a Key Vault HSM. The target key always remains in the HSM protection boundary.

Prerequisites

The following table lists prerequisites for using BYOK in Azure Key Vault:

Requirement	More Information
An Azure subscription	To create a key vault in Azure Key Vault, you need an Azure subscription. Sign up for a free trial .
A Key Vault Premium SKU to import HSM-protected keys	For more information about the service tiers and capabilities in Azure Key Vault, see Key Vault Pricing .
An HSM from the supported HSMs list and a BYOK tool and instructions provided by your HSM vendor	You must have permissions for an HSM and basic knowledge of how to use your HSM. See Supported HSMs .
Azure CLI version 2.1.0 or later	See Install the Azure CLI .

Supported HSMs

Vendor Name	Vendor Type	Supported HSM Models	More Information
Thales	Manufacturer	SafeNet Luna HSM 7 family with firmware version 7.3 or later	SafeNet Luna BYOK tool and documentation
Fortanix	HSM as a Service	Self-Defending Key Management Service (SDKMS)	Exporting SDKMS keys to Cloud Providers for BYOK - Azure Key Vault

Note

To import HSM-protected keys from the nCipher nShield family of HSMs, use the [legacy BYOK procedure](#).

Supported key types

Key Name	Key Type	Key Size	Origin	Description
Key Exchange Key (KEK)	RSA	2,048-bit 3,072-bit 4,096-bit	Azure Key Vault HSM	An HSM-backed RSA key pair generated in Azure Key Vault
Target key	RSA	2,048-bit 3,072-bit 4,096-bit	Vendor HSM	The key to be transferred to the Azure Key Vault HSM

Generate and transfer your key to the Key Vault HSM

To generate and transfer your key to a Key Vault HSM:

- [Step 1: Generate a KEK](#)
- [Step 2: Download the KEK public key](#)
- [Step 3: Generate and prepare your key for transfer](#)
- [Step 4: Transfer your key to Azure Key Vault](#)

Step 1: Generate a KEK

A KEK is an RSA key that's generated in a Key Vault HSM. The KEK is used to encrypt the key you want to import (the *target key*).

The KEK must be:

- An RSA-HSM key (2,048-bit; 3,072-bit; or 4,096-bit)
- Generated in the same key vault where you intend to import the target key
- Created with allowed key operations set to `import`

NOTE

The KEK must have 'import' as the only allowed key operation. 'import' is mutually exclusive with all other key operations.

Use the `az keyvault key create` command to create a KEK that has key operations set to `import`. Record the key identifier (`kid`) that's returned from the following command. (You will use the `kid` value in [Step 3](#).)

```
az keyvault key create --kty RSA-HSM --size 4096 --name KEKforBYOK --ops import --vault-name ContosoKeyVaultHSM
```

Step 2: Download the KEK public key

Use `az keyvault key download` to download the KEK public key to a .pem file. The target key you import is encrypted by using the KEK public key.

```
az keyvault key download --name KEKforBYOK --vault-name ContosoKeyVaultHSM --file KEKforBYOK.publickey.pem
```

Transfer the KEKforBYOK.publickey.pem file to your offline computer. You will need this file in the next step.

Step 3: Generate and prepare your key for transfer

Refer to your HSM vendor's documentation to download and install the BYOK tool. Follow instructions from your HSM vendor to generate a target key, and then create a key transfer package (a BYOK file). The BYOK tool will use the `kid` from [Step 1](#) and the KEKforBYOK.publickey.pem file you downloaded in [Step 2](#) to generate an encrypted target key in a BYOK file.

Transfer the BYOK file to your connected computer.

NOTE

Importing RSA 1,024-bit keys is not supported. Currently, importing an Elliptic Curve (EC) key is not supported.

Known issue: Importing an RSA 4K target key from SafeNet Luna HSMs is only supported with firmware 7.4.0 or newer.

Step 4: Transfer your key to Azure Key Vault

To complete the key import, transfer the key transfer package (a BYOK file) from your disconnected computer to the internet-connected computer. Use the `az keyvault key import` command to upload the BYOK file to the Key Vault HSM.

```
az keyvault key import --vault-name ContosoKeyVaultHSM --name ContosoFirstHSMkey --byok-file KeyTransferPackage-ContosoFirstHSMkey(byok)
```

If the upload is successful, Azure CLI displays the properties of the imported key.

Next steps

You can now use this HSM-protected key in your key vault. For more information, see [this price and feature comparison](#).

Import HSM-protected keys for Key Vault (legacy)

17 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

For added assurance, when you use Azure Key Vault, you can import or generate keys in hardware security modules (HSMs) that never leave the HSM boundary. This scenario is often referred to as *bring your own key*, or BYOK. Azure Key Vault uses nCipher nShield family of HSMs (FIPS 140-2 Level 2 validated) to protect your keys.

Use the information in this topic to help you plan for, generate, and then transfer your own HSM-protected keys to use with Azure Key Vault.

This functionality is not available for Azure China.

NOTE

For more information about Azure Key Vault, see [What is Azure Key Vault?](#)

For a getting started tutorial, which includes creating a key vault for HSM-protected keys, see [What is Azure Key Vault?](#).

More information about generating and transferring an HSM-protected key over the Internet:

- You generate the key from an offline workstation, which reduces the attack surface.
- The key is encrypted with a Key Exchange Key (KEK), which stays encrypted until it is transferred to the Azure Key Vault HSMs. Only the encrypted version of your key leaves the original workstation.
- The toolset sets properties on your tenant key that binds your key to the Azure Key Vault security world. So after the Azure Key Vault HSMs receive and decrypt your key, only these HSMs can use it. Your key cannot be exported. This binding is enforced by the nCipher HSMs.
- The Key Exchange Key (KEK) that is used to encrypt your key is generated inside the Azure Key Vault HSMs and is not exportable. The HSMs enforce that there can be no clear version of the KEK outside the HSMs. In addition, the toolset includes attestation from nCipher that the KEK is not exportable and was generated inside a genuine HSM that was manufactured by nCipher.
- The toolset includes attestation from nCipher that the Azure Key Vault security world was also generated on a genuine HSM manufactured by nCipher. This attestation proves to you that Microsoft is using genuine hardware.
- Microsoft uses separate KEKs and separate Security Worlds in each geographical region. This separation ensures that your key can be used only in data centers in the region in which you encrypted it. For example, a key from a European customer cannot be used in data centers in North America or Asia.

More information about nCipher HSMs and Microsoft services

nCipher Security, an Entrust Datacard company, is a leader in the general purpose HSM market, empowering world-leading organizations by delivering trust, integrity and control to their business critical information and applications. nCipher's cryptographic solutions secure emerging technologies – cloud, IoT, blockchain, digital payments – and help meet new compliance mandates, using the same proven technology that global organizations

depend on today to protect against threats to their sensitive data, network communications and enterprise infrastructure. nCipher delivers trust for business critical applications, ensuring the integrity of data and putting customers in complete control – today, tomorrow, at all times.

Microsoft has collaborated with nCipher Security to enhance the state of art for HSMs. These enhancements enable you to get the typical benefits of hosted services without relinquishing control over your keys. Specifically, these enhancements let Microsoft manage the HSMs so that you do not have to. As a cloud service, Azure Key Vault scales up at short notice to meet your organization's usage spikes. At the same time, your key is protected inside Microsoft's HSMs: You retain control over the key lifecycle because you generate the key and transfer it to Microsoft's HSMs.

Implementing bring your own key (BYOK) for Azure Key Vault

Use the following information and procedures if you will generate your own HSM-protected key and then transfer it to Azure Key Vault—the bring your own key (BYOK) scenario.

Prerequisites for BYOK

See the following table for a list of prerequisites for bring your own key (BYOK) for Azure Key Vault.

Requirement	More Information
A subscription to Azure	To create an Azure Key Vault, you need an Azure subscription: Sign up for free trial
The Azure Key Vault Premium service tier to support HSM-protected keys	For more information about the service tiers and capabilities for Azure Key Vault, see the Azure Key Vault Pricing website.
nCipher nShield HSMs, smartcards, and support software	You must have access to a nCipher Hardware Security Module and basic operational knowledge of nCipher nShield HSMs. See nCipher nShield Hardware Security Module for the list of compatible models, or to purchase an HSM if you do not have one.
The following hardware and software: 1. An offline x64 workstation with a minimum Windows operation system of Windows 7 and nCipher nShield software that is at least version 11.50. If this workstation runs Windows 7, you must install Microsoft .NET Framework 4.5 . 2. A workstation that is connected to the Internet and has a minimum Windows operating system of Windows 7 and Azure PowerShell minimum version 1.1.0 installed. 3. A USB drive or other portable storage device that has at least 16 MB free space.	For security reasons, we recommend that the first workstation is not connected to a network. However, this recommendation is not programmatically enforced. In the instructions that follow, this workstation is referred to as the disconnected workstation. In addition, if your tenant key is for a production network, we recommend that you use a second, separate workstation to download the toolset, and upload the tenant key. But for testing purposes, you can use the same workstation as the first one. In the instructions that follow, this second workstation is referred to as the Internet-connected workstation.

Generate and transfer your key to Azure Key Vault HSM

You will use the following five steps to generate and transfer your key to an Azure Key Vault HSM:

- [Step 1: Prepare your Internet-connected workstation](#)
- [Step 2: Prepare your disconnected workstation](#)

- [Step 3: Generate your key](#)
- [Step 4: Prepare your key for transfer](#)
- [Step 5: Transfer your key to Azure Key Vault](#)

Step 1: Prepare your Internet-connected workstation

For this first step, do the following procedures on your workstation that is connected to the Internet.

Step 1.1: Install Azure PowerShell

From the Internet-connected workstation, download and install the Azure PowerShell module that includes the cmdlets to manage Azure Key Vault. For installation instructions, see [How to install and configure Azure PowerShell](#).

Step 1.2: Get your Azure subscription ID

Start an Azure PowerShell session and sign in to your Azure account by using the following command:

```
Connect-AzAccount
```

In the pop-up browser window, enter your Azure account user name and password. Then, use the [Get-AzSubscription](#) command:

```
Get-AzSubscription
```

From the output, locate the ID for the subscription you will use for Azure Key Vault. You will need this subscription ID later.

Do not close the Azure PowerShell window.

Step 1.3: Download the BYOK toolset for Azure Key Vault

Go to the Microsoft Download Center and [download the Azure Key Vault BYOK toolset](#) for your geographic region or instance of Azure. Use the following information to identify the package name to download and its corresponding SHA-256 package hash:

United States:

KeyVault-BYOK-Tools-UnitedStates.zip

2E8C00320400430106366A4E8C67B79015524E4EC24A2D3A6DC513CA1823B0D4

Europe:

KeyVault-BYOK-Tools-Europe.zip

9AAA63E2E7F20CF9BB62485868754203721D2F88D300910634A32DFA1FB19E4A

Asia:

KeyVault-BYOK-Tools-AsiaPacific.zip

4BC14059BF0FEC562CA927AF621DF665328F8A13616F44C977388EC7121EF6B5

Latin America:

KeyVault-BYOK-Tools-LatinAmerica.zip

E7DFAFF579AFE1B9732C30D6FD80C4D03756642F25A538922DD1B01A4FACB619

Japan:

KeyVault-BYOK-Tools-Japan.zip

3933C13CC6DC06651295ADC482B027AF923A76F1F6BF98B4D4B8E94632DEC7DF

Korea:

KeyVault-BYOK-Tools-Korea.zip

71AB6BCFE06950097C8C18D532A9184BEF52A74BB944B8610DDDA05344ED136F

South Africa:

KeyVault-BYOK-Tools-SouthAfrica.zip

C41060C5C0170AAAAD896DA732E31433D14CB9FC83AC3C67766F46D98620784A

UAE:

KeyVault-BYOK-Tools-UAE.zip

FADE80210B06962AA0913EA411DAB977929248C65F365FD953BB9F241D5FC0D3

Australia:

KeyVault-BYOK-Tools-Australia.zip

CD0FB7365053DEF8C35116D7C92D203C64A3D3EE2452A025223EEB166901C40A

Azure Government:

KeyVault-BYOK-Tools-USGovCloud.zip

F8DB2FC914A7360650922391D9AA79FF030FD3048B5795EC83ADC59DB018621A

US Government DOD:

KeyVault-BYOK-Tools-USGovernmentDoD.zip

A79DD8C6DFFF1B00B91D1812280207A205442B3DDF861B79B8B991BB55C35263

Canada:

KeyVault-BYOK-Tools-Canada.zip

61BE1A1F80AC79912A42DEBBCC42CF87C88C2CE249E271934630885799717C7B

Germany:

KeyVault-BYOK-Tools-Germany.zip

5385E615880AAFC02AFD9841F7BADD025D7EE819894AA29ED3C71C3F844C45D6

Germany Public:

KeyVault-BYOK-Tools-Germany-Public.zip

54534936D0A0C99C8117DB724C34A5E50FD204CF CBD75C78972B785865364A29

India:

KeyVault-BYOK-Tools-India.zip

49EDCEB3091CF1DF7B156D5B495A4ADE1CFBA77641134F61B0E0940121C436C8

France:

KeyVault-BYOK-Tools-France.zip

5C9D1F3E4125B0C09E9F60897C9AE3A8B4CB0E7D13A14F3EDBD280128F8FE7DF

United Kingdom:

KeyVault-BYOK-Tools-UnitedKingdom.zip

432746BD0D3176B708672CCFF19D6144FCAA9E5EB29BB056489D3782B3B80849

Switzerland:

KeyVault-BYOK-Tools-Switzerland.zip

88CF8D39899E26D456D4E0BC57E5C94913ABF1D73A89013FCE3BBD9599AD2FE9

To validate the integrity of your downloaded BYOK toolset, from your Azure PowerShell session, use the [Get-FileHash](#) cmdlet.

```
Get-FileHash KeyVault-BYOK-Tools-*.zip
```

The toolset includes:

- A Key Exchange Key (KEK) package that has a name beginning with **BYOK-KEK-pkg-**.
- A Security World package that has a name beginning with **BYOK-SecurityWorld-pkg-**.
- A python script named **verifykeypackage.py**.
- A command-line executable file named **KeyTransferRemote.exe** and associated DLLs.
- A Visual C++ Redistributable Package, named **vcredist_x64.exe**.

Copy the package to a USB drive or other portable storage.

Step 2: Prepare your disconnected workstation

For this second step, do the following procedures on the workstation that is not connected to a network (either the Internet or your internal network).

Step 2.1: Prepare the disconnected workstation with nCipher nShield HSM

Install the nCipher support software on a Windows computer, and then attach a nCipher nShield HSM to that computer.

Ensure that the nCipher tools are in your path (**%nfast_home%\bin**). For example, type the following:

```
set PATH=%PATH%;"%nfast_home%\bin"
```

For more information, see the user guide included with the nShield HSM.

Step 2.2: Install the BYOK toolset on the disconnected workstation

Copy the BYOK toolset package from the USB drive or other portable storage, and then do the following:

1. Extract the files from the downloaded package into any folder.

2. From that folder, run vcredist_x64.exe.
3. Follow the instructions to install the Visual C++ runtime components for Visual Studio 2013.

Step 3: Generate your key

For this third step, do the following procedures on the disconnected workstation. To complete this step your HSM must be in initialization mode.

Step 3.1: Change the HSM mode to 'I'

If you are using nCipher nShield Edge, to change the mode: 1. Use the Mode button to highlight the required mode. 2. Within a few seconds, press and hold the Clear button for a couple of seconds. If the mode changes, the new mode's LED stops flashing and remains lit. The Status LED might flash irregularly for a few seconds and then flashes regularly when the device is ready. Otherwise, the device remains in the current mode, with the appropriate mode LED lit.

Step 3.2: Create a security world

Start a command prompt and run the nCipher new-world program.

```
new-world.exe --initialize --cipher-suite=Dlf3072s256mRijndael --module=1 --acs-quorum=2/3
```

This program creates a **Security World** file at %NFAST_KMDATA%\local\world, which corresponds to the C:\ProgramData\nCipher\Key Management Data\local folder. You can use different values for the quorum but in our example, you're prompted to enter three blank cards and pins for each one. Then, any two cards give full access to the security world. These cards become the **Administrator Card Set** for the new security world.

NOTE

If your HSM does not support the newer cypher suite DLF3072s256mRijndael, you can replace --cipher-suite=DLF3072s256mRijndael with --cipher-suite=DLf1024s160mRijndael

Security world created with new-world.exe that ships with nCipher software version 12.50 is not compatible with this BYOK procedure. There are two options available:

1. Downgrade nCipher software version to 12.40.2 to create a new security world.
2. Contact nCipher support and request them to provide a hotfix for 12.50 software version, which allows you to use 12.40.2 version of new-world.exe that is compatible with this BYOK procedure.

Then do the following:

- Back up the world file. Secure and protect the world file, the Administrator Cards, and their pins, and make sure that no single person has access to more than one card.

Step 3.3: Change the HSM mode to 'O'

If you are using nCipher nShield Edge, to change the mode: 1. Use the Mode button to highlight the required mode. 2. Within a few seconds, press and hold the Clear button for a couple of seconds. If the mode changes, the new mode's LED stops flashing and remains lit. The Status LED might flash irregularly for a few seconds and then flashes regularly when the device is ready. Otherwise, the device remains in the current mode, with the appropriate mode LED lit.

Step 3.4: Validate the downloaded package

This step is optional but recommended so that you can validate the following:

- The Key Exchange Key that is included in the toolset has been generated from a genuine nCipher nShield HSM.
- The hash of the Security World that is included in the toolset has been generated in a genuine nCipher nShield HSM.

- The Key Exchange Key is non-exportable.

NOTE

To validate the downloaded package, the HSM must be connected, powered on, and must have a security world on it (such as the one you've just created).

To validate the downloaded package:

1. Run the verifykeypackage.py script by typing one of the following, depending on your geographic region or instance of Azure:

- For North America:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-NA-1 -w BYOK-SecurityWorld-pkg-NA-1
```

- For Europe:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-EU-1 -w BYOK-SecurityWorld-pkg-EU-1
```

- For Asia:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-AP-1 -w BYOK-SecurityWorld-pkg-AP-1
```

- For Latin America:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-LATAM-1 -w BYOK-SecurityWorld-pkg-LATAM-1
```

- For Japan:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-JPN-1 -w BYOK-SecurityWorld-pkg-JPN-1
```

- For Korea:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-KOREA-1 -w BYOK-SecurityWorld-pkg-KOREA-1
```

- For South Africa:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-SA-1 -w BYOK-SecurityWorld-pkg-SA-1
```

- For UAE:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-UAE-1 -w BYOK-SecurityWorld-pkg-UAE-1
```

- For Australia:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-AUS-1 -w BYOK-SecurityWorld-pkg-AUS-1
```

- For [Azure Government](#), which uses the US government instance of Azure:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-USGOV-1 -w BYOK-SecurityWorld-pkg-USGOV-1
```

- For US Government DOD:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-USDOD-1 -w BYOK-SecurityWorld-pkg-USDOD-1
```

- For Canada:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-CANADA-1 -w BYOK-SecurityWorld-pkg-CANADA-1
```

- For Germany:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-GERMANY-1 -w BYOK-SecurityWorld-pkg-GERMANY-1
```

- For Germany Public:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-GERMANY-1 -w BYOK-SecurityWorld-pkg-GERMANY-1
```

- For India:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-INDIA-1 -w BYOK-SecurityWorld-pkg-INDIA-1
```

- For France:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-FRANCE-1 -w BYOK-SecurityWorld-pkg-FRANCE-1
```

- For United Kingdom:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-UK-1 -w BYOK-SecurityWorld-pkg-UK-1
```

- For Switzerland:

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-SUI-1 -w BYOK-SecurityWorld-pkg-SUI-1
```

TIP

The nCipher nShield software includes python at %NFAST_HOME%\python\bin

2. Confirm that you see the following, which indicates successful validation: **Result: SUCCESS**

This script validates the signer chain up to the nShield root key. The hash of this root key is embedded in the script and its value should be **59178a47 de508c3f 291277ee 184f46c4 f1d9c639**. You can also confirm this value separately by visiting the [nCipher website](#).

You're now ready to create a new key.

Step 3.5: Create a new key

Generate a key by using the nCipher nShield **generatekey** program.

Run the following command to generate the key:

```
generatekey --generate simple type=RSA size=2048 protect=module ident=contosokey plainname=contosokey nvram=no pubexp=
```

When you run this command, use these instructions:

- The parameter *protect* must be set to the value **module**, as shown. This creates a module-protected key. The BYOK toolset does not support OCS-protected keys.
- Replace the value of *contosokey* for the **ident** and **plainname** with any string value. To minimize administrative overheads and reduce the risk of errors, we recommend that you use the same value for both. The **ident** value must contain only numbers, dashes, and lower case letters.
- The pubexp is left blank (default) in this example, but you can specify specific values. For more information, see the [nCipher documentation](#).

This command creates a Tokenized Key file in your %NFAST_KMDATA%\local folder with a name starting with **key_simple_**, followed by the **ident** that was specified in the command. For example: **key_simple_contosokey**. This file contains an encrypted key.

Back up this Tokenized Key File in a safe location.

IMPORTANT

When you later transfer your key to Azure Key Vault, Microsoft cannot export this key back to you so it becomes extremely important that you back up your key and security world safely. Contact [nCipher](#) for guidance and best practices for backing up your key.

You are now ready to transfer your key to Azure Key Vault.

Step 4: Prepare your key for transfer

For this fourth step, do the following procedures on the disconnected workstation.

Step 4.1: Create a copy of your key with reduced permissions

Open a new command prompt and change the current directory to the location where you unzipped the BYOK zip file. To reduce the permissions on your key, from a command prompt, run one of the following, depending on your geographic region or instance of Azure:

- For North America:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-NA-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-NA-1
```

- For Europe:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-EU-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-EU-1
```

- For Asia:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-AP-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-AP-1
```

- For Latin America:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-LATAM-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-LATAM-1
```

- For Japan:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-JPN-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-JPN-1
```

- For Korea:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-KOREA-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-KOREA-1
```

- For South Africa:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-SA-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-SA-1
```

- For UAE:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-UAE-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-UAE-1
```

- For Australia:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-AUS-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-AUS-1
```

- For [Azure Government](#), which uses the US government instance of Azure:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-USGOV-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-USGOV-1
```

- For US Government DOD:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-USDOD-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-USDOD-1
```

- For Canada:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-CANADA-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-CANADA-1
```

- For Germany:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-GERMANY-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-GERMANY-1
```

- For Germany Public:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-GERMANY-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-GERMANY-1
```

- For India:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-INDIA-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-INDIA-1
```

- For France:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-FRANCE-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-FRANCE-1
```

- For United Kingdom:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-UK-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-UK-1
```

- For Switzerland:

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier contosokey -ExchangeKeyPackage  
BYOK-KEK-pkg-SUI-1 -NewSecurityWorldPackage BYOK-SecurityWorld-pkg-SUI-1
```

When you run this command, replace *contosokey* with the same value you specified in **Step 3.5: Create a new key** from the [Generate your key](#) step.

You are asked to plug in your security world admin cards.

When the command completes, you see **Result: SUCCESS** and the copy of your key with reduced permissions are in the file named `key_xferacl_<contosokey>`.

You may inspect the ACLS using following commands using the nCipher nShield utilities:

- `aclprint.py`:

```
"%nfast_home%\bin\preload.exe" -m 1 -A xferacld -K contosokey "%nfast_home%\python\bin\python"  
"%nfast_home%\python\examples\aclprint.py"
```

- kmfile-dump.exe:

```
"%nfast_home%\bin\kmfile-dump.exe" "%NFAST_KMDATA%\local\key_xferacld_contosokey"
```

When you run these commands, replace contosokey with the same value you specified in **Step 3.5: Create a new key** from the [Generate your key](#) step.

Step 4.2: Encrypt your key by using Microsoft's Key Exchange Key

Run one of the following commands, depending on your geographic region or instance of Azure:

- For North America:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-NA-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-NA-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For Europe:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-EU-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-EU-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For Asia:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-AP-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-AP-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For Latin America:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-LATAM-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-LATAM-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For Japan:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-JPN-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-JPN-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For Korea:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-KOREA-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-KOREA-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For South Africa:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-SA-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-SA-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For UAE:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-UAE-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-UAE-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For Australia:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-AUS-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-AUS-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For [Azure Government](#), which uses the US government instance of Azure:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-USGOV-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-USGOV-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For US Government DOD:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-USDOD-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-USDOD-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For Canada:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-CANADA-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-CANADA-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For Germany:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-GERMANY-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-GERMANY-1 -SubscriptionId SubscriptionID -  
KeyFriendlyName ContosoFirstHSMkey
```

- For Germany Public:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-GERMANY-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-GERMANY-1 -SubscriptionId SubscriptionID -  
KeyFriendlyName ContosoFirstHSMkey
```

- For India:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-INDIA-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-INDIA-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For France:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-France-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-France-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For United Kingdom:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-UK-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-UK-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

- For Switzerland:

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -ExchangeKeyPackage BYOK-KEK-pkg-SUI-1 -  
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-SUI-1 -SubscriptionId SubscriptionID -KeyFriendlyName  
ContosoFirstHSMkey
```

When you run this command, use these instructions:

- Replace *contosokey* with the identifier that you used to generate the key in **Step 3.5: Create a new key** from the [Generate your key](#) step.
- Replace *SubscriptionID* with the ID of the Azure subscription that contains your key vault. You retrieved this value previously, in **Step 1.2: Get your Azure subscription ID** from the [Prepare your Internet-connected workstation](#) step.
- Replace *ContosoFirstHSMKey* with a label that is used for your output file name.

When this completes successfully, it displays **Result: SUCCESS** and there is a new file in the current folder that has the following name: KeyTransferPackage-ContosoFirstHSMkey.byok

Step 4.3: Copy your key transfer package to the Internet-connected workstation

Use a USB drive or other portable storage to copy the output file from the previous step (KeyTransferPackage-ContosoFirstHSMkey.byok) to your Internet-connected workstation.

Step 5: Transfer your key to Azure Key Vault

For this final step, on the Internet-connected workstation, use the [Add-AzKeyVaultKey](#) cmdlet to upload the key transfer package that you copied from the disconnected workstation to the Azure Key Vault HSM:

```
Add-AzKeyVaultKey -VaultName 'ContosoKeyVaultHSM' -Name 'ContosoFirstHSMkey' -KeyFilePath  
'c:\KeyTransferPackage-ContosoFirstHSMkey.byok' -Destination 'HSM'
```

If the upload is successful, you see displayed the properties of the key that you just added.

Next steps

You can now use this HSM-protected key in your key vault. For more information, see this price and feature [comparison](#).

Receive and respond to key vault notifications with Azure Event Grid (preview)

5 minutes to read • [Edit Online](#)

Azure Key Vault integration with Azure Event Grid (currently in preview) enables user notification when the status of a secret stored in a key vault has changed. For an overview of this feature, see [Monitoring Key Vault with Event Grid](#).

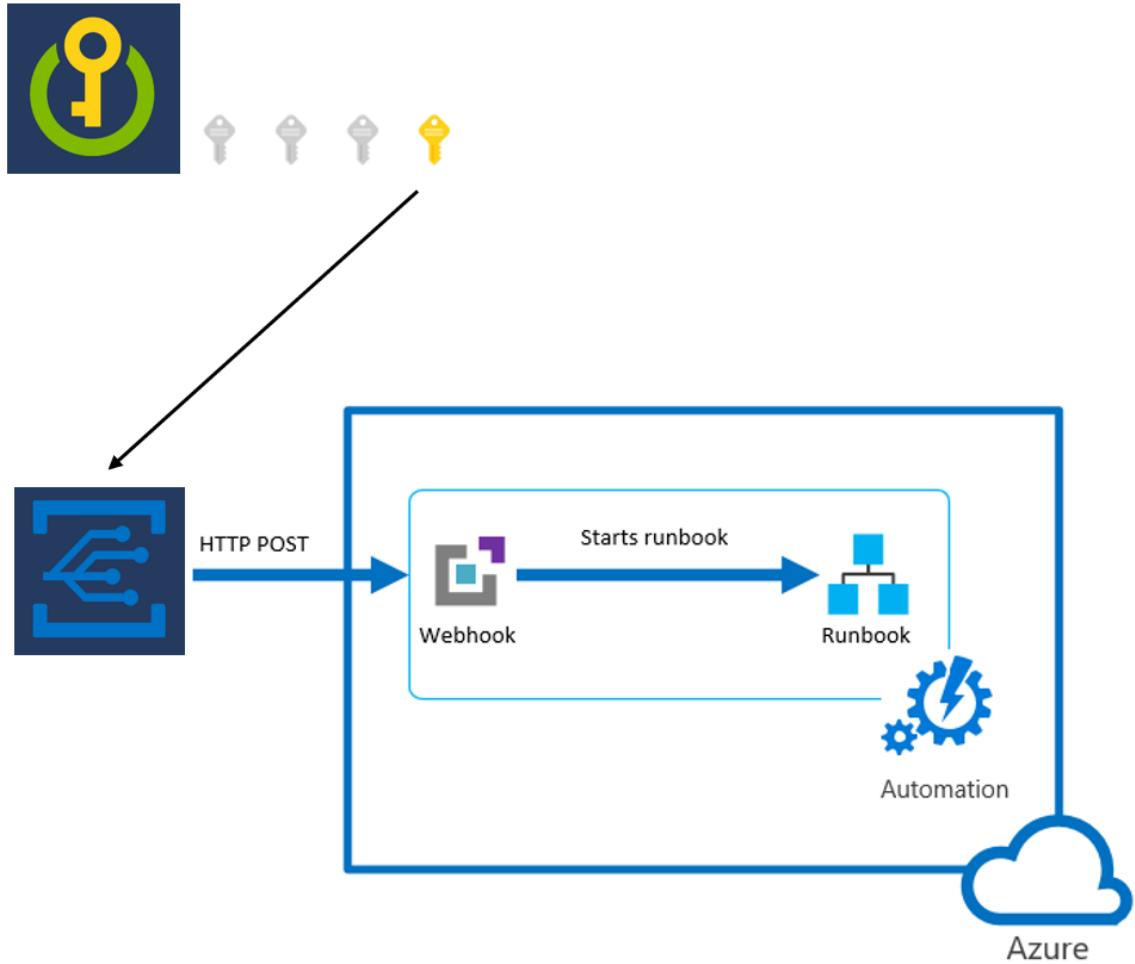
This guide describes how to receive Key Vault notifications through Event Grid, and how to respond to status changes through Azure Automation.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- A key vault in your Azure Subscription. You can quickly create a new key vault by following the steps in [Set and retrieve a secret from Azure Key Vault using Azure CLI](#).

Concepts

Event Grid is an eventing service for the cloud. By following the steps in this guide, you'll subscribe to events for Key Vault and route events to Automation. When one of the secrets in the key vault is about to expire, Event Grid is notified of the status change and makes an HTTP POST to the endpoint. A web hook then triggers an Automation execution of a PowerShell script.



Create an Automation account

Create an Automation account through the [Azure portal](#):

1. Go to portal.azure.com and log in to your subscription.
2. In the search box, enter **Automation Accounts**.
3. Under the **Services** section of the drop-down list on the search bar, select **Automation Accounts**.
4. Select **Add**.

5. Enter the required information in the **Add Automation Account** pane and then select **Create**.

Create a runbook

After your Automation account is ready, create a runbook.

The screenshot shows the Azure portal's Automation Accounts page. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Configuration Management (Inventory, Change tracking, State configuration (DSC)), Update management, and Process Automation (Runbooks, Jobs, Runbooks gallery, Hybrid worker groups, Watcher tasks). The 'Runbooks' section is currently selected. In the main pane, there's a search bar at the top. Below it, a table lists several runbooks:

NAME	AUTHORIZING STATUS	RUNBOOK TYPE
AzureAutomationTutorial	✓ Published	Graphical Runbook
AzureAutomationTutorialPython2	✗ In edit	Python 2 Runbook
AzureAutomationTutorialScript	✓ Published	PowerShell Runbook
AzureClassicAutomationTutorial	✓ Published	Graphical Runbook
AzureClassicAutomationTutorialSc...	✓ Published	PowerShell Runbook
notificationstest	✓ Published	PowerShell Runbook
shane-notifications-runbook	✗ In edit	PowerShell Runbook

1. Select the Automation account you just created.
2. Select **Runbooks** under **Process Automation**.
3. Select **Create a runbook**.
4. Name your runbook and select **PowerShell** as the runbook type.
5. Select the runbook you created and then select the **Edit** button.
6. Enter the following code (for testing purposes) and select the **Publish** button. This action returns the result of the POST request received.

```

param
(
[Parameter (Mandatory = $false)]
[object] $WebhookData
)

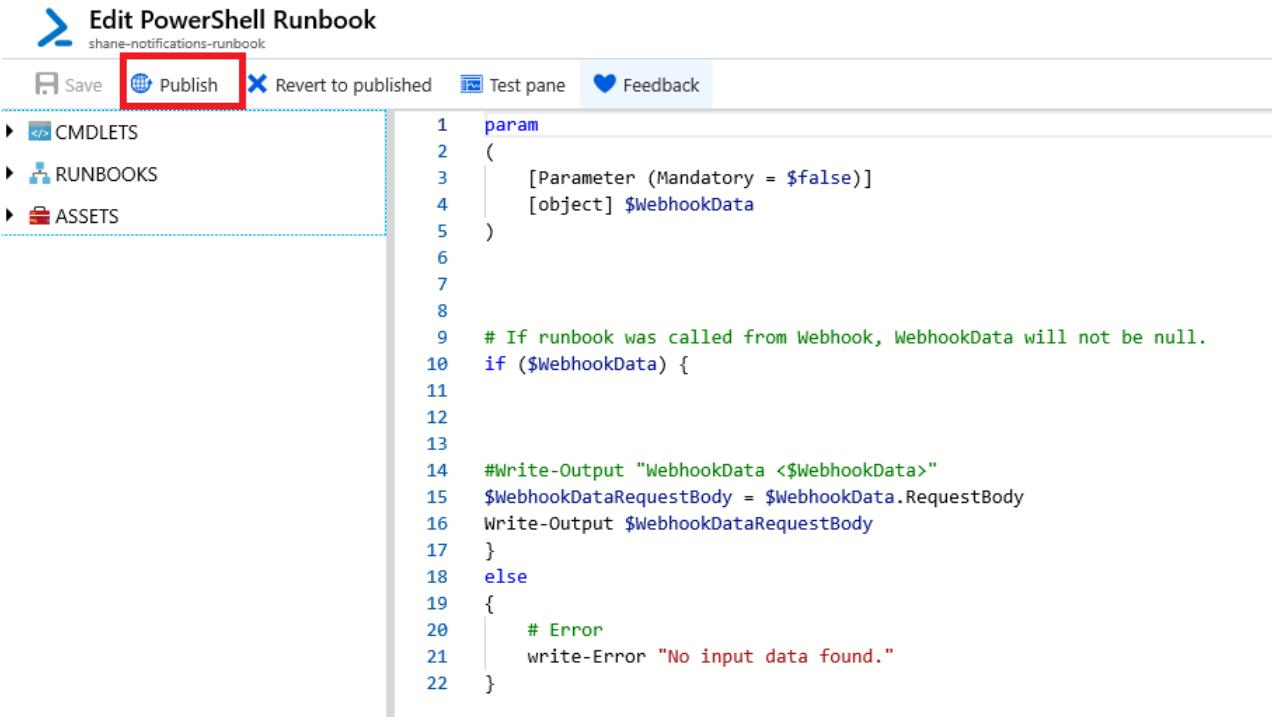
#If runbook was called from Webhook, WebhookData will not be null.
if ($WebhookData) {

#rotate secret:
#generate new secret version in key vault
#update db/service with generated secret

#Write-Output "WebhookData <$WebhookData>"
Write-Output $WebhookData.RequestBody
}

else
{
# Error
write-Error "No input data found."
}

```



The screenshot shows the 'Edit PowerShell Runbook' page. At the top, there are buttons for Save, Publish (which is highlighted with a red box), Revert to published, Test pane, and Feedback. On the left, there's a sidebar with links to CMDLETS, RUNBOOKS, and ASSETS. The main area contains the PowerShell script:

```

1 param
2 (
3     [Parameter (Mandatory = $false)]
4     [object] $WebhookData
5 )
6
7
8
9 # If runbook was called from Webhook, WebhookData will not be null.
10 if ($WebhookData) {
11
12
13
14 #Write-Output "WebhookData <$WebhookData>"
15 $WebhookDataRequestBody = $WebhookData.RequestBody
16 Write-Output $WebhookDataRequestBody
17 }
18 else
19 {
20     # Error
21     write-Error "No input data found."
22 }

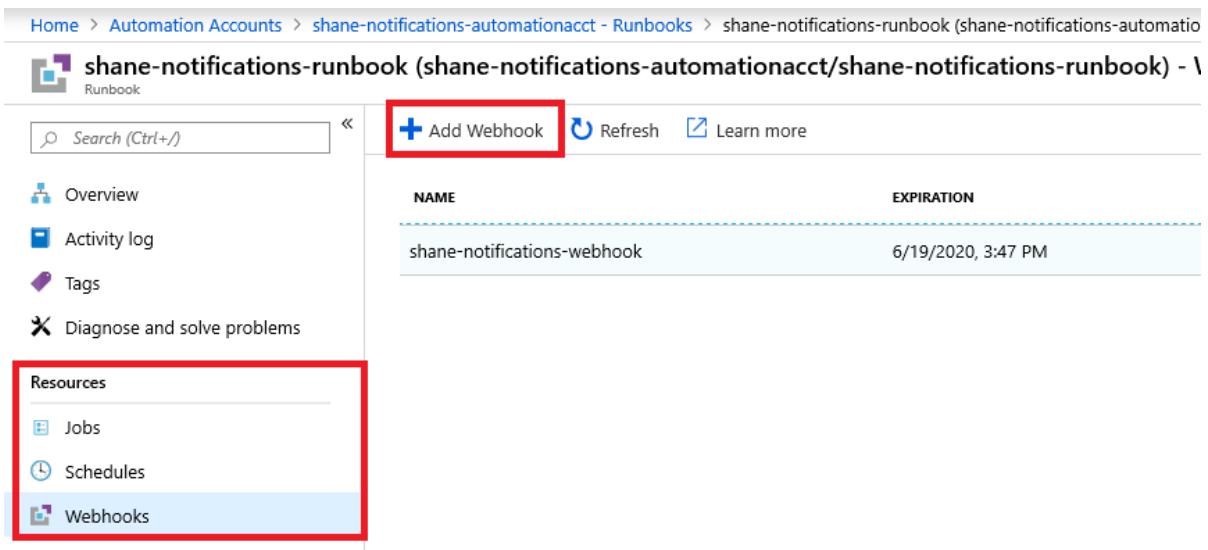
```

Create a webhook

Create a webhook to trigger your newly created runbook.

1. Select **Webhooks** from the **Resources** section of the runbook you just published.

2. Select **Add Webhook**.



The screenshot shows the 'shane-notifications-runbook' runbook details page. The 'Add Webhook' button is highlighted with a red box. The 'Webhooks' option in the 'Resources' sidebar is also highlighted with a red box. The main table shows one webhook entry:

NAME	EXPIRATION
shane-notifications-webhook	6/19/2020, 3:47 PM

3. Select **Create new Webhook**.

4. Name the webhook, set an expiration date, and copy the URL.

IMPORTANT

You can't view the URL after you create it. Make sure you save a copy in a secure location where you can access it for the remainder of this guide.

5. Select **Parameters and run settings** and then select **OK**. Don't enter any parameters. This will enable the **Create** button.

- Select **OK** and then select **Create**.

The screenshot shows two side-by-side windows in the Azure portal:

- Add Webhook**: A list of options including "Webhook" and "Create new webhook".
- Create a new webhook**: A form with fields:
 - Name**: A text input field with placeholder "Enter the webhook name...".
 - Enabled**: A switch between "Yes" (selected) and "No".
 - Expires**: A date and time picker set to "2020-06-20 1:59:39 PM".
 - URL**: An input field containing "https://s15events.azure-automation.ne..." with a copy icon.

Create an Event Grid subscription

Create an Event Grid subscription through the [Azure portal](#).

- Go to your key vault and select the **Events** tab. If you can't see it, make sure you're using the [preview version of the portal](#).

The screenshot shows the "notifications-kv - Events" page in the Azure portal:

- Left sidebar (Events tab):**
 - Overview
 - Activity log
 - Access control (IAM)
 - Tags
 - Diagnose and solve problems
 - Events** (highlighted with a red box)
 - Settings
 - Keys
 - Secrets
 - Certificates
- Top navigation:**
 - + Event Subscription
 - Refresh
- Main content area:**
 - Event Subscriptions** tab is selected.
 - A search bar: "Search to filter items..."
 - A table with columns: Name, Endpoint, Prefix Filter.
 - A message: "No Event Subscriptions found."

- Select the **Event Subscription** button.
- Create a descriptive name for the subscription.
- Choose **Event Grid Schema**.
- Topic Resource** should be the key vault you want to monitor for status changes.
- For **Filter to Event Types**, leave all options selected (**9 selected**).
- For **Endpoint Type**, select **Webhook**.
- Choose **Select an endpoint**. In the new context pane, paste the webhook URL from the [Create a webhook](#)

step into the **Subscriber Endpoint** field.

9. Select **Confirm Selection** on the context pane.

10. Select **Create**.

The screenshot shows two windows side-by-side. On the left is the 'Create Event Subscription' page under 'Event Grid'. It has tabs for 'Basic', 'Filters', and 'Additional Features', with 'Basic' selected. A red box highlights the 'EVENT SUBSCRIPTION DETAILS' section where 'Name' is set to 'notifications-demo' and 'Event Schema' is set to 'Event Grid Schema'. Another red box highlights the 'TOPIC DETAILS' section where 'Topic Resource' is set to 'notifications-kv'. A third red box highlights the 'EVENT TYPES' section where 'Filter to Event Types' shows '9 selected'. A fourth red box highlights the 'ENDPOINT DETAILS' section where 'Endpoint Type' is set to 'Web Hook (change)' and 'Endpoint' is set to 'Select an endpoint'. On the right is a 'Select Web Hook' modal window titled 'Event Grid'. It has a single input field labeled 'Subscriber Endpoint *' with a placeholder 'Enter endpoint URL' and a red border around it.

Test and verify

Verify that your Event Grid subscription is properly configured. This test assumes you have subscribed to the "Secret New Version Created" notification in the [Create an Event Grid subscription](#), and that you have the necessary permissions to create a new version of a secret in a key vault.

The screenshot shows the 'Secrets' blade in the 'notifications-kv - Secrets' key vault. The left sidebar has links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings, Keys, Secrets (which is highlighted with a red box), and Certificates. The main area has a search bar, a 'Generate/Import' button, and refresh and restore backup buttons. A table lists secrets with columns for Name, Type, Status, and Expiration Date. The table shows the message 'There are no secrets available.'

Create a secret

Upload options

Manual

Name * ⓘ

notifications-test



Value *



Content type (optional)

Set activation date? ⓘ

Set expiration date? ⓘ

Expiration Date

06/22/2019



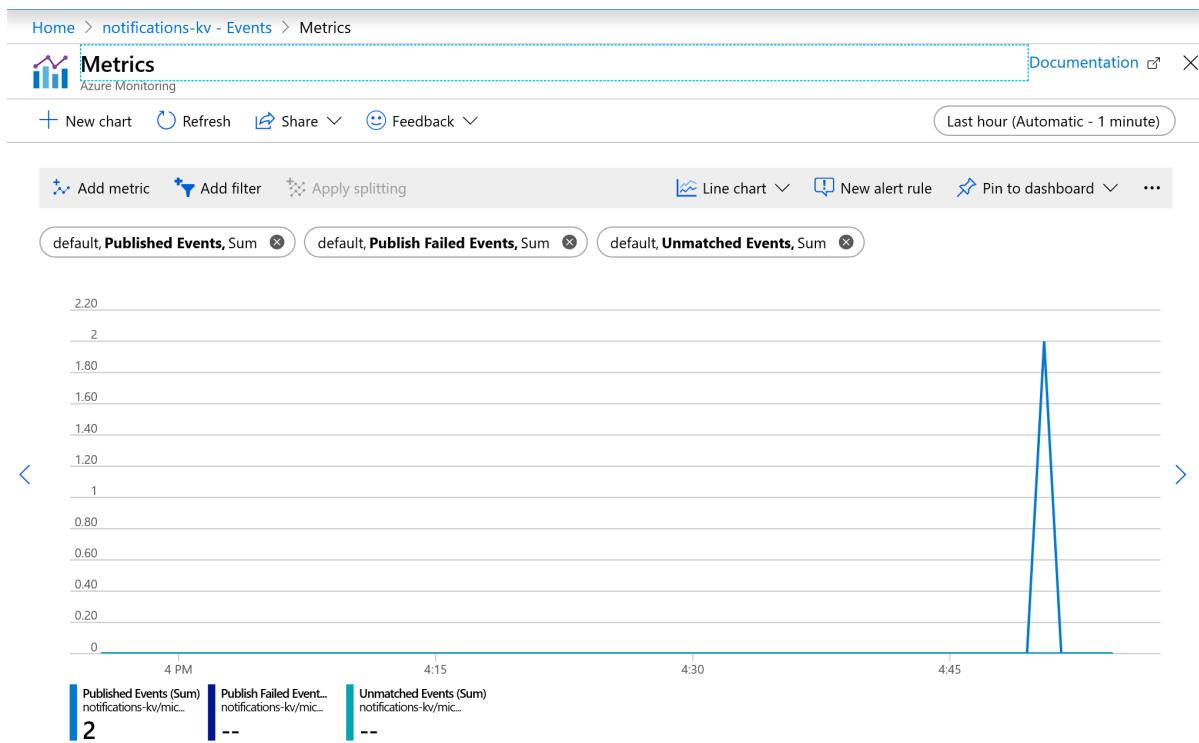
4:12:07 PM

(UTC-07:00) --- Current Time Zone ---



Enabled? Yes No

1. Go to your key vault on the Azure portal.
2. Create a new secret. For testing purposes, set the expiration to date to the next day.
3. On the **Events** tab in your key vault, select the Event Grid subscription you created.
4. Under **Metrics**, check whether an event was captured. Two events are expected: SecretNewVersion and SecretNearExpiry. These events validate that Event Grid successfully captured the status change of the secret in your key vault.



5. Go to your Automation account.
6. Select the **Runbooks** tab, and then select the runbook you created.
7. Select the **Webhooks** tab, and confirm that the "last triggered" time stamp is within 60 seconds of when you created the new secret. This result confirms that Event Grid made a POST to the webhook with the event details of the status change in your key vault and that the webhook was triggered.

The screenshot shows the Azure Runbook blade for the "shane-notifications-runbook". The left sidebar shows the following sections:

- Overview
- Activity log
- Tags
- Diagnose and solve problems
- Resources (Jobs, Schedules, Webhooks)

The main area shows the Webhooks table:

NAME	EXPIRATION	LAST TRIGGERED	STATUS
shane-notifications-webhook	6/19/2020, 3:47 PM	6/20/2019, 9:43 AM	✓ Enabled

8. Return to your runbook and select the **Overview** tab.
9. Look at the **Recent Jobs** list. You should see that a job was created and that the status is complete. This confirms that the webhook triggered the runbook to start executing its script.

The screenshot shows the Azure Runbook blade for the "shane-notifications-runbook" with the Overview tab selected. The left sidebar shows the following sections:

- Overview
- Activity log
- Tags
- Diagnose and solve problems
- Resources (Jobs, Schedules, Webhooks)
- Runbook settings (Properties, Description, Logging and tracing)

The main area shows the Recent Jobs table:

STATUS	CREATED	LAST UPDATED
✓ Completed	6/20/2019, 9:43:44 AM	6/20/2019, 9:43:52 AM
✓ Completed	6/19/2019, 3:53:42 PM	6/19/2019, 3:53:52 PM
✓ Completed	6/19/2019, 3:53:27 PM	6/19/2019, 3:53:35 PM

10. Select the recent job and look at the POST request that was sent from Event Grid to the webhook. Examine the JSON and make sure that the parameters for your key vault and event type are correct. If the "event type" parameter in the JSON object matches the event that occurred in the key vault (in this example, Microsoft.KeyVault.SecretNearExpiry), the test was successful.

Troubleshooting

You can't create an event subscription

Reregister Event Grid and the key vault provider in your Azure subscription resource providers. See [Azure resource providers and types](#).

Next steps

Congratulations! If you've correctly followed all these steps, you're now ready to programmatically respond to status changes of secrets stored in your key vault.

If you've been using a polling-based system to search for status changes of secrets in your key vaults, you can now start using this notification feature. You can also replace the test script in your runbook with code to programmatically renew your secrets when they're about to expire.

Learn more:

- Overview: [Monitoring Key Vault with Azure Event Grid \(preview\)](#)
- How to: [Receive email when a key vault secret changes](#)
- [Azure Event Grid event schema for Azure Key Vault \(preview\)](#)
- [Azure Key Vault overview](#)
- [Azure Event Grid overview](#)
- [Azure Automation overview](#)

Use Logic Apps to receive email about status changes of key vault secrets

2 minutes to read • [Edit Online](#)

In this guide you will learn how to respond to Azure Key Vault events that are received via [Azure Event Grid](#) by using [Azure Logic Apps](#). By the end, you will have an Azure logic app set up to send a notification email every time a secret is created in Azure Key Vault.

For an overview of Azure Key Vault / Azure Event Grid integration, see [Monitoring Key Vault with Azure Event Grid \(preview\)](#).

Prerequisites

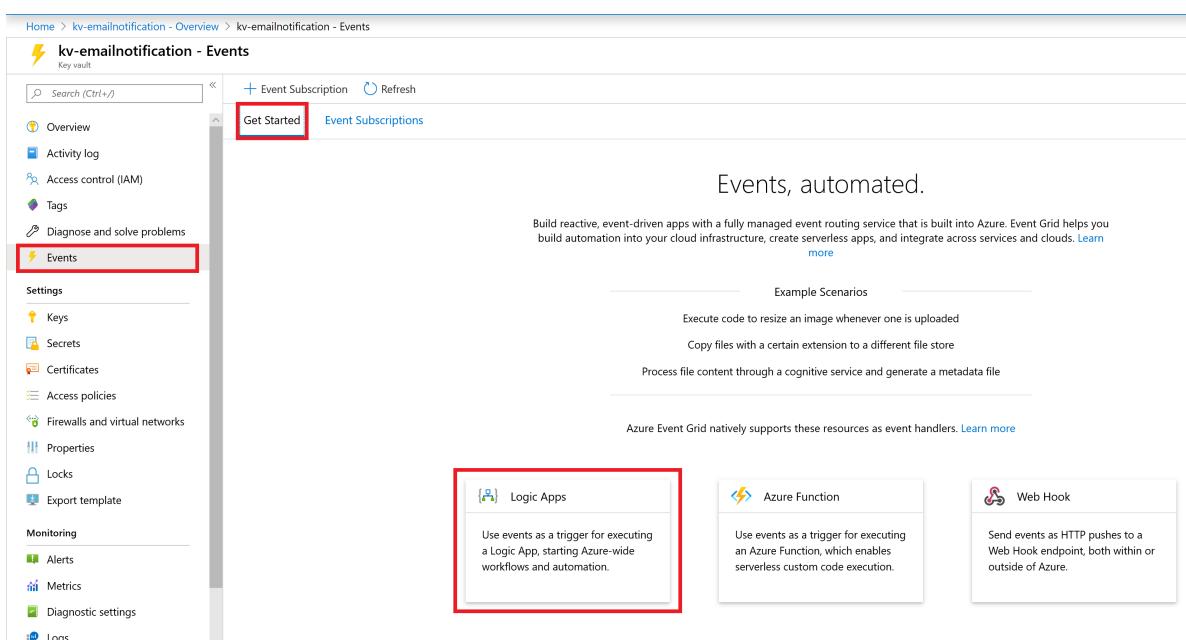
- An email account from any email provider that is supported by Azure Logic Apps (such as Office 365 Outlook). This email account is used to send the event notifications. For a complete list of supported Logic App connectors, see the [Connectors overview](#)
- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- A key vault in your Azure Subscription. You can quickly create a new key vault by following the steps in [Set and retrieve a secret from Azure Key Vault using Azure CLI](#).

Create a Logic App via Event Grid

First, create Logic App with event grid handler and subscribe to Azure Key Vault "SecretNewVersionCreated" events.

To create an Azure Event Grid subscription, follow these steps:

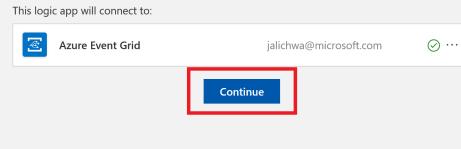
1. In the Azure portal, go to your key vault, select **Events > Get Started** and click **Logic Apps**



2. On **Logic Apps Designer** validate the connection and click **Continue**

Logic Apps Designer

Save As Discard Designer Code view Templates Connectors Help

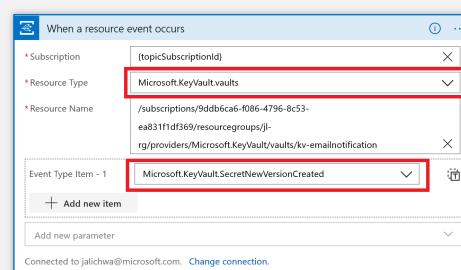


3. On the **When a resource event occurs** screen, do the following:

- Leave **Subscription** and **Resource Name** as default.
- Select **Microsoft.KeyVault.vaults** for the **Resource Type**.
- Select **Microsoft.KeyVault.SecretNewVersionCreated** for **Event Type Item - 1**.

Logic Apps Designer

Save As Discard Designer Code view Templates Connectors Help



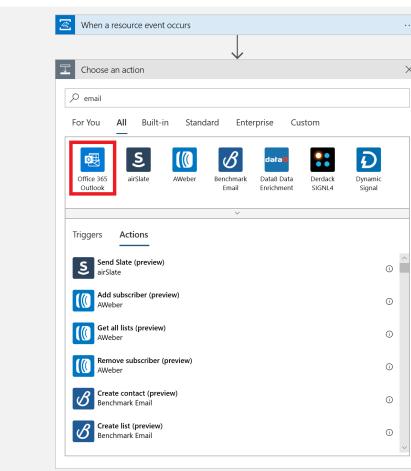
4. Select **+ New Step** This will open a window to Choose an action.

5. Search for **Email**. Based on your email provider, find and select the matching connector. This tutorial uses **Office 365 Outlook**. The steps for other email providers are similar.

6. Select the **Send an email (V2)** action.

Logic Apps Designer

Save As Discard Designer Code view Templates Connectors Help



7. Build your email template:

- To:** Enter the email address to receive the notification emails. For this tutorial, use an email account that you can access for testing.
- Subject** and **Body**: Write the text for your email. Select JSON properties from the selector tool to

include dynamic content based on event data. You can retrieve the data of the event using

```
@{triggerBody()?'[ 'Data' ]} .
```

Your email template may look like this example.

The screenshot shows the Logic Apps Designer interface. A workflow step titled "Send an email (V2)" is selected. The "To" field is set to "jalichwa@microsoft.com" and the "Subject" field is set to "New Secret Version Created". In the "Body" section, there is a rich text editor with a "Font" dropdown and a toolbar. Below the toolbar, dynamic content is being added to the body. The "Event Id" placeholder is highlighted with a blue box. The "Event Data" placeholder contains the expression "@{triggerBody()?'['Data']}
". To the right of the main canvas, a sidebar titled "Dynamic content" lists several options: Event Time, Event Type, ID, Subject, and Topic. Each option has a brief description and a small icon next to it. At the bottom right of the sidebar, there is a "Create" button.

8. Click **Save as**.

9. Enter a **name** for new logic app and click **Create**.

The screenshot shows the "Logic App" creation dialog. On the left, the logic app definition is visible, showing the "When a resource event occurs" trigger and the "Send an email (V2)" action with its configuration. On the right, the "Logic App" form is displayed. The "Name" field is filled with "logicapp-emailnotification" and is highlighted with a red box. Below it, other fields include "Subscription" (set to "jalichwa Projects"), "Resource group" (set to "jlg-rg" with "Use existing" checked), "Location" (set to "Central US"), "Logic App Status" (set to "Enabled"), and "Log Analytics" (set to "Off"). At the bottom right of the dialog, there is a large blue "Create" button, which is also highlighted with a red box.

Test and verify

1. Go to your key vault on the Azure portal and select **Events > Event Subscriptions**. Verify that a new subscription created



2. Go to your key vault, select **Secrets**, and select **+ Generate/Import**. Create a new secret for testing purposes name the key and keep the remaining parameters in their default settings.

Home > EventGrid-LogicApps - Overview > EventGrid-LogicApps - Secrets

EventGrid-LogicApps - Secrets

Key vault

Search (Ctrl+ /)

+ Generate/Import Refresh Restore Backup

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

Settings

- Keys
- Secrets** (circled in red)
- Certificates

Name

There are no secrets available.

3. On the **Create a secret** screen provide any name, any value, and select **Create**.

When the secret is created, an email will be received at the configured addresses.

Next steps

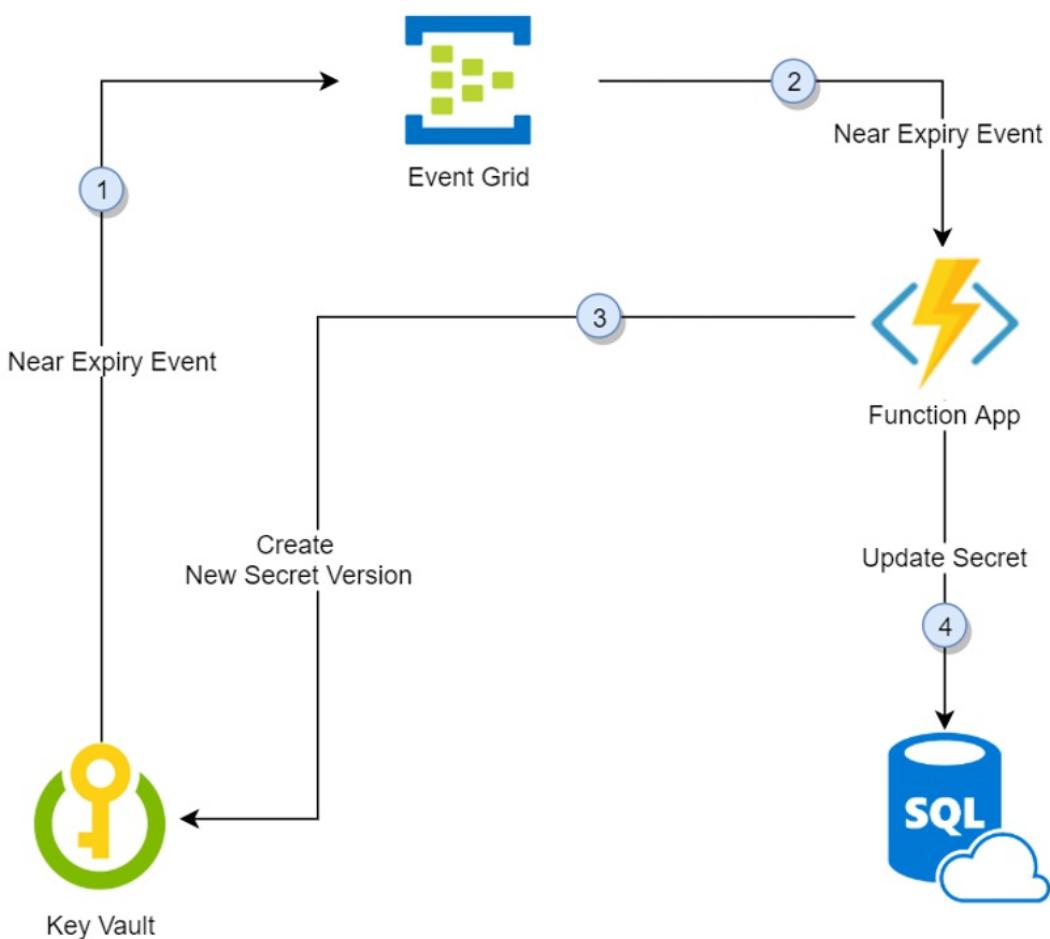
- Overview: [Monitoring Key Vault with Azure Event Grid \(preview\)](#)
- How to: [Route key vault notifications to Azure Automation](#).
- [Azure Event Grid event schema for Azure Key Vault \(preview\)](#)
- Learn more about [Azure Event Grid](#).
- Learn more about the [Logic Apps feature of Azure App Service](#).

Automate the rotation of a secret for resources with single user/password authentication

5 minutes to read • [Edit Online](#)

Although the best way to authenticate to Azure services is by using an [managed identity](#), there are some scenarios where this is not an option. In these cases, access keys or secrets are used. Access keys or secrets should be periodically rotated.

This tutorial demonstrates how to automate the periodic rotation of secrets for databases and services with single user/password authentication. Specifically, this scenario rotates SQL server passwords stored in key vault using a function triggered by Event Grid notification:



1. Thirty days before the expiration date of a secret, Key Vault publish the "near expiry" event to Event Grid.
2. Event Grid checks the event subscriptions and, using http post, calls the Function App endpoint subscribed to this event.
3. The function App receives the secret information, generates a new random password, and creates a new version for the secret with a new password in Key Vault.
4. The function App updates SQL with new password.

NOTE

There could be a lag between step 3 and 4 and during that time secret in Key Vault would not be valid to authenticate to SQL. In case of failure in any of the steps Event Grid retries for 2 hours.

Setup

Create a key vault and SQL server

Before we begin, we must create a Key Vault, create a SQL Server and database, and store the SQL Server admin password in Key Vault.

This tutorial uses a pre-created Azure Resource Manager template to create components. You can find entire code here: [Basic Secret Rotation Template Sample](#).

1. Click Azure template deployment link:



2. For "Resource Group", select "Create New" and give it the name "simplerotation".

3. Select "Purchase".

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

Home > Custom deployment

Custom deployment

Deploy from a custom template

TEMPLATE

Customized template
4 resources

Edit template Edit param... Learn more

BASICS

Subscription * jalichwa Projects

Resource group * Select a resource group Create new

Location *

SETTINGS

Resource Name Prefix ⓘ

TERMS AND CONDITIONS

Azure Marketplace Terms | Azure Marketplace Terms OK Cancel

By clicking "Purchase," I (a) agree to the applicable legal terms associated with the offering; (b) authorize Microsoft to charge or bill my current payment method for the fees associated with the offering(s), including applicable taxes, with the same billing frequency as my Azure subscription, until I discontinue use of the offering(s); and (c) agree that, if the deployment involves 3rd party offerings, Microsoft may share my contact information and other details of such deployment with the publisher of that offering.

I agree to the terms and conditions stated above

Purchase

After completing these steps, you will have a key vault, a SQL server, and a SQL database. You can verify this in an Azure CLI terminal by running:

```
az resource list -o table
```

The results will look something like this:

Name	ResourceGroup	Location	Type	Status
simplerotation-kv	simplerotation	eastus	Microsoft.KeyVault/vaults	
simplerotation-sql	simplerotation	eastus	Microsoft.Sql/servers	
simplerotation-sql/master	simplerotation	eastus	Microsoft.Sql/servers/databases	

Create Function App

Create a Function App with a system-managed identity, as well as the additional required components:

Function app requires below components and configuration:

- App Service Plan
- Storage Account
- Access policy to access secrets in Key Vault using Function App Managed Identity

1. Click Azure template deployment link:



2. For "Resource Group", select "simplerotation".

3. Select "Purchase".

The screenshot shows the 'Custom deployment' page in the Azure portal. The 'TEMPLATE' section shows a customized template with 5 resources. The 'BASICS' section has fields for Subscription (jalichwa Projects), Resource group (simplerotation, highlighted with a red box), and Location ((US) East US). The 'SETTINGS' section shows a Resource Name Prefix of [resourceGroup().name]. In the 'TERMS AND CONDITIONS' section, there's a link to Azure Marketplace Terms and a checkbox for agreeing to terms. A large red box highlights the 'Purchase' button at the bottom.

After completing the steps above, you will have a storage account, a server farm, and a Function App. You can verify this in an Azure CLI terminal by running:

```
az resource list -o table
```

The results will look something like this:

Name	ResourceGroup	Location	Type	Status
simplerotation-kv	simplerotation	eastus	Microsoft.KeyVault/vaults	
simplerotation-sql	simplerotation	eastus	Microsoft.Sql/servers	
simplerotation-sql/master	simplerotation	eastus	Microsoft.Sql/servers/databases	
simplerotationstrg	simplerotation	eastus	Microsoft.Storage/storageAccounts	
simplerotation-plan	simplerotation	eastus	Microsoft.Web/serverFarms	
simplerotation-fn	simplerotation	eastus	Microsoft.Web/sites	

For information how to create Function App and using Managed Identity to access Key Vault, see [Create a function app from the Azure portal](#) and [Provide Key Vault authentication with a managed identity](#)

Rotation function and deployment

Function is using event as trigger and perform rotation of a secret updating Key Vault and SQL database.

Function Event Trigger Handler

Below Function reads event data and executes rotation logic

```
public static class SimpleRotationEventHandler
{
    [FunctionName("SimpleRotation")]
    public static void Run([EventGridTrigger]EventGridEvent eventGridEvent, ILogger log)
    {
        log.LogInformation("C# Event trigger function processed a request.");
        var secretName = eventGridEvent.Subject;
        var secretVersion = Regex.Match(eventGridEvent.Data.ToString(), "Version\"":\[a-z0-9\]*").Groups[1].ToString();
        var keyVaultName = Regex.Match(eventGridEvent.Topic, ".vaults.(.*)").Groups[1].ToString();
        log.LogInformation($"Key Vault Name: {keyVaultName}");
        log.LogInformation($"Secret Name: {secretName}");
        log.LogInformation($"Secret Version: {secretVersion}");

        SeretRotator.RotateSecret(log, secretName, secretVersion, keyVaultName);
    }
}
```

Secret Rotation Logic

This rotation method reads database information from the secret, create a new version of the secret, and updates the database with a new secret.

```

public class SecretRotator
{
    private const string UserIdTagName = "UserID";
    private const string DataSourceTagName = "DataSource";
    private const int SecretExpirationDays = 31;

    public static void RotateSecret	ILogger log, string secretName, string secretVersion, string keyVaultName)
    {
        //Retrieve Current Secret
        var kvUri = "https://" + keyVaultName + ".vault.azure.net";
        var client = new SecretClient(new Uri(kvUri), new DefaultAzureCredential());
        KeyVaultSecret secret = client.GetSecret(secretName, secretVersion);
        log.LogInformation("Secret Info Retrieved");

        //Retrieve Secret Info
        var userId = secret.Properties.Tags.ContainsKey(UserIdTagName) ?
            secret.Properties.Tags[UserIdTagName] : "";
        var datasource = secret.Properties.Tags.ContainsKey(DataSourceTagName) ?
            secret.Properties.Tags[DataSourceTagName] : "";
        log.LogInformation($"Data Source Name: {datasource}");
        log.LogInformation($"User Id Name: {userId}");

        //create new password
        var randomPassword = CreateRandomPassword();
        log.LogInformation("New Password Generated");

        //Check db connection using existing secret
        CheckServiceConnection(secret);
        log.LogInformation("Service Connection Validated");

        //Create new secret with generated password
        CreateNewSecretVersion(client, secret, randomPassword);
        log.LogInformation("New Secret Version Generated");

        //Update db password
        UpdateServicePassword(secret, randomPassword);
        log.LogInformation("Password Changed");
        log.LogInformation($"Secret Rotated Succesffuly");
    }
}

```

You can find entire code here: <https://github.com/jlichwa/azure-keyvault-basicrotation-tutorial/tree/master/rotation-function>

Function deployment

1. Download function app zip file: <https://github.com/jlichwa/azure-keyvault-basicrotation-tutorial/raw/master/simplerotationsample-fn.zip>
2. Upload file simplerotationsample-fn.zip to Azure Cloud Shell.
3. Use below CLI command to deploy zip file to function app:

```
az functionapp deployment source config-zip -g simplerotation -n simplerotation-fn --src /home/{firstname e.g jack}/simplerotationsample-fn.zip
```

```
PS /home/jack> az function app config-zip -g simplerotation -n simplerotation-fn --src /home/jack/simplerotationsample-fn.zip
```

Upload
Download
Manage file share

Upload destination: /home/jack
simplerotationsample-fn.zip COMPLETE

After deployment you should notice two functions under simplerotation-fn:

Home > simplerotation > simplerotation-fn

simplerotation-fn

Function Apps

jalichwa Projects

Function Apps

simplerotation-fn

- Functions (Read Only)
 - SimpleRotation
 - SimpleRotationHttpTest
- Proxies (Read Only)
- Slots

Application Insights is not connected

Overview

Status: Running

Availability: Not applicable

Configured features

Function app settings

Configuration

Add event subscription for "SecretNearExpiry" event

Copy the function app eventgrid_extension key.

jalichwa Projects

Function Apps

simplerotation-fn

- Functions (Read Only)
 - SimpleRotation
 - Integrate
 - Manage
 - Monitor
 - SimpleRotationHttpTest
- Proxies (Read Only)
- Slots

Configured features

Function app settings

Configuration

Use the copied eventgrid extension key and your subscription ID in below command to create an event grid subscription for SecretNearExpiry events.

```
az eventgrid event-subscription create --name simplerotation-eventsSubscription --source-resource-id
"/subscriptions/<subscription-
id>/resourceGroups/simplerotation/providers/Microsoft.KeyVault/vaults/simplerotation-kv" --endpoint
"https://simplerotation-fn.azurewebsites.net/runtime/webhooks/EventGrid?functionName=SimpleRotation&code=<extension-key>" --endpoint-type WebHook --included-event-types "Microsoft.KeyVault.SecretNearExpiry"
```

Add secret to Key Vault

Set your access policy to give "manage secrets" permission to users.

```
az keyvault set-policy --upn <email-address-of-user> --name simplerotation-kv --secret-permissions set delete
get list
```

Now create a new secret with tags containing sql database datasource and user ID, with the expiration date set for tomorrow.

```
$tomorrowDate = (get-date).AddDays(+1).ToString("yyy-MM-ddThh:mm:ssZ")
az keyvault secret set --name sqluser --vault-name simplerotation-kv --value "Simple123" --tags
"UserID=azureuser" "DataSource=simplerotation-sql.database.windows.net" --expires $tomorrowDate
```

Creating a secret with a short expiration date will immediately publish a SecretNearExpiry event, which will in turn trigger the function to rotate the secret.

Test and verify

After few minutes, sqluser secret should automatically rotate.

To verify secret rotation verification, go to Key Vault > Secrets

Name	Type	Status	Expiration Date
sqluser		✓ Enabled	2/9/2020

Open the "sqluser" secret and view the original and rotated version

Version	Status	Activation Date	Expiration Date
CURRENT VERSION 65af27e40b7d4e49be3ef6dbcad3b822	✓ Enabled		2/9/2020
OLDER VERSIONS d1891540d0b44b8b89c6d7d0be545ac7	✓ Enabled		1/10/2020

Create Web App

To verify SQL credentials, create a web application. This web application will get the secret from key vault, extract sql database information and credentials from the secret, and test the connection to sql.

The web app requires below components and configuration:

- Web App with System-Managed Identity
- Access policy to access secrets in Key Vault using Web App Managed Identity

1. Click Azure template deployment link:



2. Select the **simplerotation** resource group

3. Click Purchase

Deploy Web App

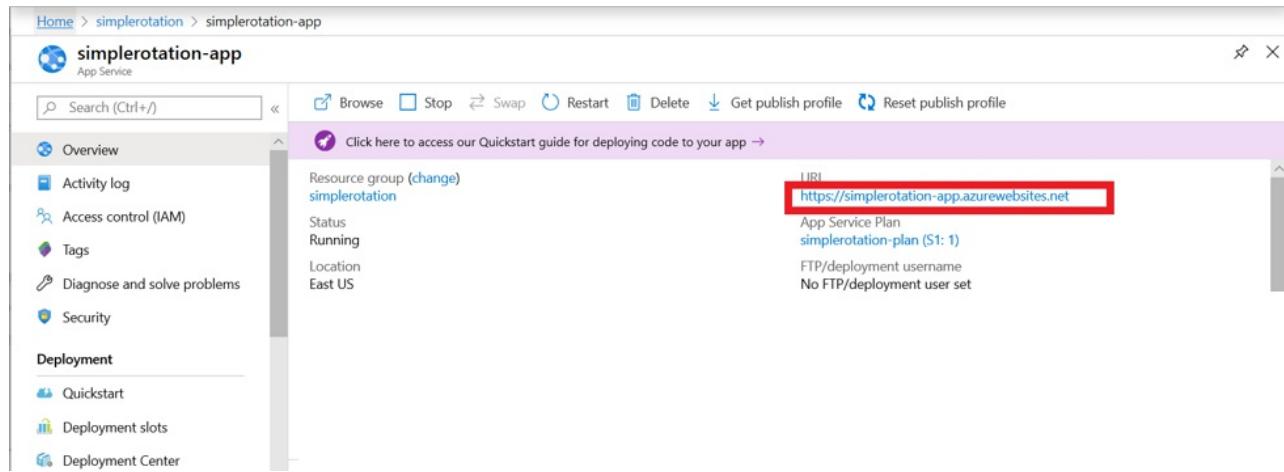
Source code for the web app you can find at <https://github.com/jlichwa/azure-keyvault-basicrotation-tutorial/tree/master/test-webapp> For deployment of the web app, do the following:

1. Download the function app zip file from <https://github.com/jlichwa/azure-keyvault-basicrotation-tutorial/raw/master/simplerotationsample-app.zip>
2. Upload the file `simplerotationsample-app.zip` to Azure Cloud Shell.
3. Use this Azure CLI command to deploy the zip file to the function app:

```
az webapp deployment source config-zip -g simplerotation -n simplerotation-app --src /home/{firstname}  
e.g jack}/simplerotationsample-app.zip
```

Open web Application

Go to the deployed application and click "URL":



The screenshot shows the Azure portal interface for an App Service named 'simplerotation-app'. The left sidebar has sections for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment (Quickstart, Deployment slots, Deployment Center), and Resource group (simplerotation). The main content area displays the app's status as 'Running' in 'East US'. On the right, there are links for 'Get publish profile' and 'Reset publish profile'. A prominent purple bar at the top says 'Click here to access our Quickstart guide for deploying code to your app'. Below it, the 'URI' is listed as <https://simplerotation-app.azurewebsites.net>, which is highlighted with a red box.

The Generated Secret Value should be shown with Database Connected as true.

Learn more:

- Overview: [Monitoring Key Vault with Azure Event Grid \(preview\)](#)
- How to: [Receive email when a key vault secret changes](#)
- [Azure Event Grid event schema for Azure Key Vault \(preview\)](#)

Integrate Azure Key Vault with Azure Policy

8 minutes to read • [Edit Online](#)

Azure Policy is a governance tool that gives users the ability to audit and manage their Azure environment at scale. Azure Policy provides the ability to place guardrails on Azure resources to ensure they are compliant with assigned policy rules. It allows users to perform audit, real-time enforcement, and remediation of their Azure environment. The results of audits performed by policy will be available to users in a compliance dashboard where they will be able to see a drilldown of which resources and components are compliant and which are not. For more information, see the [Overview of the Azure Policy service](#).

Example Usage Scenarios:

- You want to improve the security posture of your company by implementing requirements around minimum key sizes and maximum validity periods of certificates in your company's key vaults but you don't know which teams will be compliant and which are not.
- You currently don't have a solution to perform an audit across your organization, or you are conducting manual audits of your environment by asking individual teams within your organization to report their compliance. You are looking for a way to automate this task, perform audits in real time, and guarantee the accuracy of the audit.
- You want to enforce your company security policies and stop individuals from creating self-signed certificates, but you don't have an automated way to block their creation.
- You want to relax some requirements for your test teams, but you want to maintain tight controls over your production environment. You need a simple automated way to separate enforcement of your resources.
- You want to be sure that you can roll-back enforcement of new policies in the event of a live-site issue. You need a one-click solution to turn off enforcement of the policy.
- You are relying on a 3rd party solution for auditing your environment and you want to use an internal Microsoft offering.

Types of policy effects and guidance

Audit: When the effect of a policy is set to audit, the policy will not cause any breaking changes to your environment. It will only alert you to components such as certificates that do not comply with the policy definitions within a specified scope, by marking these components as non-compliant in the policy compliance dashboard. Audit is default if no policy effect is selected.

Deny: When the effect of a policy is set to deny, the policy will block the creation of new components such as certificates as well as block new versions of existing components that do not comply with the policy definition. Existing non-compliant resources within a key vault are not affected. The 'audit' capabilities will continue to operate.

Available "Built-In" Policy Definitions

Key Vault has created a set of policies, which you can assign for common scenarios to manage certificates. These policies are 'Built-In', which means they don't require you to write any custom JSON to enable them and they are available in the Azure portal for you to assign. You can still customize certain parameters to fit your organization's needs.

The eight preview policies are as follows.

Manage certificate validity period (preview)

This policy allows you to manage the maximum validity period of your certificates stored in key vault. It is a good

security practice to limit the maximum validity period of your certificates. If a private key of your certificate were to become compromised without detection, using short lived certificates minimizes the time frame for ongoing damage and reduces the value of the certificate to an attacker.

Manage allowed certificate key types (preview)

This policy allows you to restrict the type of certificates that can be in your key vault. You can use this policy to make sure that your certificate private keys are RSA, ECC, or are HSM backed. You can choose from the following list which certificate types are allowed.

- RSA
- RSA - HSM
- ECC
- ECC - HSM

Manage certificate lifetime action triggers (preview)

This policy allows you to manage the lifetime action specified for certificates that are either within a certain number of days of their expiration or have reached a certain percentage of their usable life.

Manage certificates issued by an integrated CA (preview)

If you use a Key Vault integrated certificate authority (Digicert or GlobalSign) and you want users to use one or either of these providers, you can use this policy to audit or enforce your selection. This policy can also be used to audit or deny the creation of self-signed certificates in key vault.

Manage certificates issued by an integrated CA (preview)

If you use an internal certificate authority or a certificate authority not integrated with key vault and you want users to use a certificate authority from a list you provide, you can use this policy to create an allowed list of certificate authorities by issuer name. This policy can also be used to audit or deny the creation of self-signed certificates in key vault.

Manage allowed curve names for elliptic curve cryptography certificates (preview)

If you use elliptic curve cryptography or ECC certificates, you can customize an allowed list of curve names from the list below. The default option allows all the following curve names.

- P-256
- P-256K
- P-384
- P-521

Manage minimum key size for RSA certificates (preview)

If you use RSA certificates, you can choose a minimum key size that your certificates must have. You may select one option from the list below.

- 2048 bit
- 3072 bit
- 4096 bit

Manage certificates that are within a specified number of days of expiration (preview)

Your service can experience an outage if a certificate that is not being adequately monitored is not rotated prior to its expiration. This policy is critical to making sure that your certificates stored in key vault are being monitored. It is recommended that you apply this policy multiple times with different expiration thresholds, for example, at 180, 90, 60, and 30-day thresholds. This policy can be used to monitor and triage certificate expiration in your organization.

Example Scenario

You manage a key vault used by multiple teams that contains 100 certificates, and you want to make sure that none of the certificates in the key vault are valid for longer than 2 years.

1. You assign the [Manage certificate validity period](#) policy, specify that the maximum validity period of a certificate is 24 months, and set the effect of the policy to "audit".
2. You view the [compliance report on the Azure portal](#), and discover that 20 certificates are non-compliant and valid for > 2 years, and the remaining certificates are compliant.
3. You contact the owners of these certificates and communicate the new security requirement that certificates cannot be valid for longer than 2 years. Some teams respond and 15 of the certificates were renewed with a maximum validity period of 2 years or less. Other teams do not respond, and you still have 5 non-compliant certificates in your key vault.
4. You change the effect of the policy you assigned to "deny". The 5 non-compliant certificates are not revoked, and they continue to function. However, they cannot be renewed with a validity period that is greater than 2 years.

Enabling and managing a Key Vault policy through the Azure portal

Select a Policy Definition

1. Log in to the Azure portal.
2. Search "Policy" in the Search Bar and Select **Policy**.

Azure services

- Create a resource
- Policy
- Storage accounts

Recent resources

NAME
Shane Bala Enterprise
Shane-Bala-Vault
Contoso IT - demo
shane-notifications-automationacct
shane-notificationsdemo

Services

- Policy
- WAF policies
- WAF policies
- Firewall Policies
- Service endpoint policies
- Time Series Insights access policies
- Application security groups
- Bastions

Resources

No results were found.

Marketplace

- Firewall Policy
- Service endpoint policy
- Data Protection and Access Policy Management
- Web Application Firewall (WAF)

Documentation

- All 1000+ results
- App configuration policies for Microsoft Intune ...
- App protection policies overview - Microsoft Intune ...
- Create and use password policies in Azure AD Domain ...
- Azure API Management policies | Microsoft Docs

Resource Groups

No results were found.

3. In the Policy window, select **Definitions**.

Name	Definition location	Policies	Type	Definition type	Category
azuresecuritypackautoupdate_3.1	Non Production	3	Custom	Initiative	Initiative
azuresecuritypackautoupdate_3.0	Non Production	3	Custom	Initiative	Initiative
azuresecuritypackautoupdate_1.0	Non Production	2	Custom	Initiative	Initiative
antimalwaresignatureautoupdate_1.0	Non Production	3	Custom	Initiative	Initiative
azuresecuritypackautoupdate_3.0	Non Production	3	Custom	Initiative	Initiative
azuresecuritypackautoupdate_3.1	Non Production	3	Custom	Initiative	Initiative
azuresecuritypackautoupdate_3.0	Se116433-8b65-49ec-8d2c-85ab367037ad	3	Custom	Initiative	Initiative
azuresecuritypackautoupdate_3.1	Se116433-8b65-49ec-8d2c-85ab367037ad	3	Custom	Initiative	Initiative
Audit Windows VMs in which the Administrators gro...		2	Built-in	Initiative	Guest Configuration

4. In the Category Filter, Unselect **Select All** and select **Key Vault**.

Initiative definition + Policy definition Refresh

Scope	Definition type	Type	Category
4 selected	All definition types	All types	1 categories
Name	Definition location	Policies	Type
[Key Vault objects should be recoverable]		Built-in	Policy
[Diagnostic logs in Key Vault should be enabled]		Built-in	Policy
[Deploy Diagnostic Settings for Key Vault to Event Hub]		Built-in	Policy
[Preview]: Manage certificate validity period		Built-in	Policy
[Preview]: Manage allowed certificate key types		Built-in	Policy
[Preview]: Manage certificate lifetime action triggers		Built-in	Policy
[Preview]: Manage certificates issued by an integrated CA		Built-in	Policy
[Preview]: Manage certificates issued by a non-integrat...		Built-in	Policy
[Preview]: Manage allowed curve names for elliptic curv...		Built-in	Policy
[Preview]: Manage minimum key size for RSA certificates		Built-in	Policy
[Preview]: Manage certificates that are within a specific...		Built-in	Policy

Category

- Select all
- App Service
- Automation
- Backup
- Batch
- Cache
- Compute
- Cosmos DB
- Custom Provider
- Data Lake
- Event Hub
- General
- Guest Configuration
- Internet of Things
- Key Vault
- Kubernetes
- Kubernetes service
- Lighthouse
- Logic Apps
- Managed Application
- Monitoring
- Network

- Now you should be able to see all the policies available for Public Preview, for Azure Key Vault. Make sure you have read and understood the policy guidance section above and select a policy you want to assign to a scope.

Initiative definition + Policy definition Refresh

Scope	Definition type	Type	Category
4 selected	All definition types	All types	1 categories
Name	Definition location	Policies	Type
[Key Vault objects should be recoverable]		Built-in	Policy
[Diagnostic logs in Key Vault should be enabled]		Built-in	Policy
[Deploy Diagnostic Settings for Key Vault to Event Hub]		Built-in	Policy
[Preview]: Manage certificate validity period		Built-in	Policy
[Preview]: Manage allowed certificate key types		Built-in	Policy
[Preview]: Manage certificate lifetime action triggers		Built-in	Policy
[Preview]: Manage certificates issued by an integrated CA		Built-in	Policy
[Preview]: Manage certificates issued by a non-integrated CA		Built-in	Policy
[Preview]: Manage allowed curve names for elliptic curve cryptography certificates		Built-in	Policy
[Preview]: Manage minimum key size for RSA certificates		Built-in	Policy
[Preview]: Manage certificates that are within a specified number of days of expiration		Built-in	Policy

Assign a Policy to a Scope

- Select a policy you wish to apply, in this example, the **Manage Certificate Validity Period** policy is shown. Click the assign button in the top-left corner.

[Preview]: Manage certificate validity period

Policy definition

Assign Edit definition Duplicate definition Delete definition

Name	: [Preview]: Manage certificate validity period	Definition location :	--
Description	: This policy manages the maximum validity period for certificates in months.	Definition ID	: /providers/Microsoft.Authorization/policyDefinitions/0a075868-4c26-42ef-914c-5bc007359560
Available Effects	: audit, deny, disabled	Type	: Built-in
Category	: Key Vault	Mode	: Microsoft.KeyVault.Data

Definition Assignments (0) Parameters

```

1  {
2   "properties": {
3    "displayName": "[Preview]: Manage certificate validity period",
4    "policyType": "BuiltIn",
5    "mode": "Microsoft.KeyVault.Data",
6    "description": "This policy manages the maximum validity period for certificates in months.",
7    "metadata": {
8      "category": "Key Vault",
9      "preview": true
10     },
11    "parameters": {
12      "maximumValidityInMonths": {
13        "type": "Integer",
14        "metadata": {
15          "displayName": "The maximum validity in months",
16          "description": "The limit to how long a certificate may be valid for. Certificates with lengthy validity periods aren't best practice."
17        }
18      },
19      "effect": {
20        "type": "String",
21        "metadata": {
22          "displayName": "Effect",
23          "description": "Enable or disable the execution of the policy"
24        }
25      },
26      "allowedValues": [
27        "audit",
28        "deny",
29        "disabled"
30      ],
31      "defaultValue": "audit"
32    }
33  }

```

- Select the subscription where you want the policy to be applied. You can choose to restrict the scope to only a single resource group within a subscription. If you want to apply the policy to the entire subscription and exclude some resource groups, you can also configure an exclusion list. Set the policy enforcement selector to **Enabled** if you want the effect of the policy (audit or deny) to occur or **Disabled** to turn the effect (audit or deny) off.

The screenshot shows the 'Basics' tab of the Azure Policy Definition creation interface. The 'Scope' section is expanded, showing the 'Subscription' dropdown set to 'Shane Bala Enterprise'. The 'Assignment name' field contains '[Preview]: Manage certificate validity period'. The 'Policy enforcement' section shows 'Enabled' selected. The 'Assigned by' field is populated with 'Shane Bala'. At the bottom, there are 'Review + create', 'Cancel', 'Previous', and 'Next' buttons, along with a 'Select' button and a 'Clear All Selections' link.

- Click on the parameters tab at the top of the screen in order to specify the maximum validity period in months that you want. Select **audit** or **deny** for the effect of the policy following the guidance in the sections above. Then select the review + create button.

Microsoft Azure (Preview) Report a bug Search resources, services, and docs (G+/)

Home > Policy - Definitions > [Preview]: Manage certificate validity period > [Preview]: Manage certificate validity period

[Preview]: Manage certificate validity period

Assign policy

Basics Parameters Remediation Review + create

Specify parameters for this policy assignment.

The maximum validity in months * ⓘ

Effect * ⓘ

audit

Review + create Cancel Previous Next

View Compliance Results

1. Go back to the Policy blade and select the compliance tab. Click on the policy assignment you wish to view compliance results for.

Microsoft Azure (Preview) Report a bug Search resources, services, and docs (G+/)

Home > Policy - Compliance

Policy - Compliance

Assign policy Assign initiative Refresh

Scope: 4 selected Type: All definition types Compliance state: All compliance states

Search: certificate

Overall resource compliance: 87% (158 out of 182) Non-compliant initiatives: 1 (out of 1) Non-compliant policies: 14 (out of 100) Non-compliant resources: 24 (out of 182)

Name	Scope	Compliance state	Compliance	Non-Compliant Resources	Non-compliant policies
[Preview] Certificate Expiration Policy	Shane Bala Enterprise	Non-compliant	62%	10	1
DEMO Certificate Expiration AUDIT 90 Days	Shane Bala Enterprise	Compliant	100%	0	0

2. From this page you can filter results by compliant or non-compliant vaults. Here you can see a list of non-compliant key vaults within the scope of the policy assignment. A vault is considered non-compliant if any of the components (certificates) in the vault are non-compliant. You can select an individual vault to view the individual non-compliant components (certificates).

Home > Policy - Compliance > Assignment Details

Assignment Details

Policy compliance

[View definition](#) [Edit assignment](#) [Delete assignment](#) [Create Remediation Task](#)

Name : [Preview] Certificate Expiration Policy	Scope : Shane Bala Enterprise
Description : Mark vault as non-compliant if it contains a certificate that will expire in < 45 days	Excluded scopes : 0
Assignment ID : /subscriptions/4f3a88a1-3580-4102-a464-6cef63216ae9/providers/Microsoft.Authorization/policyAssignments/e60bee1ea4c4d08a0052c92	Definition : [Preview] Certificate Expiration Policy

Selected Scopes (4 selected subscriptions)

Compliance state Non-compliant (62% out of 26)

Overall resource compliance 62% (16 out of 26)

Non-compliant resources 10 out of 26

Events (last 7 days)

- Audit 4
- Append 0
- Modify 0
- Deny 0
- Deploy 0

Details

Effect Type Audit
Parent Initiative <>NONE>

Resource compliance [Events](#)

Name	Parent Resource	Compliance state	Resource Type	Location	Scope
non-compliant-vault	resourcegroups/shane-non-compliant-rg	Non-compliant	/Microsoft.KeyVault/vaults	North Central US	Shane Bala Enterprise/shane-non-compliant-rg
shane-bala-vault	resourcegroups/shane-bala-rg	Non-compliant	/Microsoft.KeyVault/vaults	Central US	Shane Bala Enterprise/shane-bala-rg
shane-compliant-vault	resourcegroups/shane-compliant-rg	Non-compliant	/Microsoft.KeyVault/vaults	North Central US	Shane Bala Enterprise/shane-compliant-rg
shane-bala-rg	resourcegroups/shane-bala-rg	Non-compliant	/Microsoft.Security/policies	--	Shane Bala Enterprise/shane-bala-rg
88bbc99c-e5af-ddd7-6105-6150b2bfa519	microsoft.keyvault/vaults/non-compliant-va...	Non-compliant	/Microsoft.Security/assessments	--	Shane Bala Enterprise/shane-non-compliant-rg
88bbc99c-e5af-ddd7-6105-6150b2bfa519	microsoft.keyvault/vaults/shane-compliant-...	Non-compliant	/Microsoft.Security/assessments	--	Shane Bala Enterprise/shane-compliant-rg
88bbc99c-e5af-ddd7-6105-6150b2bfa519	microsoft.keyvault/vaults/shane-bala-vault	Non-compliant	/Microsoft.Security/assessments	--	Shane Bala Enterprise/shane-bala-rg
shane-non-compliant-rg	subscriptions/4f3a88a1-3580-4102-a464-6c...	Non-compliant	/Microsoft.Resources/subscriptions/resour...	North Central US	Shane Bala Enterprise/shane-non-compliant-rg
shane-compliant-rg	subscriptions/4f3a88a1-3580-4102-a464-6c...	Non-compliant	/Microsoft.Resources/subscriptions/resour...	Central US	Shane Bala Enterprise/shane-compliant-rg
shane-bala-rg	subscriptions/4f3a88a1-3580-4102-a464-6c...	Non-compliant	/Microsoft.Resources/subscriptions/resour...	Central US	Shane Bala Enterprise/shane-bala-rg

3. View the name of the components within a vault that are non-compliant

Microsoft Azure (Preview) [Report a bug](#)

Home > Policy > Assignment Details > shane-non-compliant-kv

shane-non-compliant-kv Resource Compliance

[View Resource](#) [Activity Logs](#)

Name : shane-non-compliant-kv
Type : /Microsoft.KeyVault/vaults

Selected Scopes (4 selected subscriptions)

Compliance state Non-compliant (1 out of 1)

Non-compliant policies 1 out of 1

Events (last 7 days)

- Audit 0
- Append 0
- Modify 0
- Deny 0
- Deploy 0

Non-compliant components 1 out of 1

Component Events (last 7 days)

- Audit 0
- Deny 2

Component Compliance (preview) [Policies](#) [Events](#) [Component Events \(preview\)](#) [Change History \(preview\)](#)

Component Name	Component Id	Compliance state	Type	Timestamp
maxValidityNonCompliantCert	maxValidityNonCompliantCert	Non-compliant	Certificate	11/7/2019, 11:20 AM

4. If you need to check whether users are being denied the ability to create resources within key vault, you can click on the **Component Events (preview)** tab to view a summary of denied certificate operations with the requestor and timestamps of requests.

Microsoft Azure (Preview) [Report a bug](#)

Home > Policy > Assignment Details > shane-non-compliant-kv

shane-non-compliant-kv Resource Compliance

[View Resource](#) [Activity Logs](#)

Name : shane-non-compliant-kv
Type : /Microsoft.KeyVault/vaults

Selected Scopes (4 selected subscriptions)

Compliance state Non-compliant (1 out of 1)

Non-compliant policies 1 out of 1

Events (last 7 days)

- Audit 0
- Append 0
- Modify 0
- Deny 0
- Deploy 0

Non-compliant components 1 out of 1

Component Events (last 7 days)

- Audit 0
- Deny 1

Component Compliance (preview) [Policies](#) [Events](#) [Component Events \(preview\)](#) [Change History \(preview\)](#)

Initiated by	Effect Type	Event count	Last event
Shane Bala	deny	1	Thursday, November 7, 2019, 10:20:09 AM

Feature Limitations

Assigning a policy with a "deny" effect may take up to 30 mins (average case) and 1 hour (worst case) to start denying the creation of non-compliant resources. The policy evaluation of existing components in a vault may take up to 1 hour (average case) and 2 hours (worst case) before compliance results are viewable in the portal UI. If the compliance results show up as "Not Started" it may be due to the following reasons:

- The policy valuation has not completed yet. Initial evaluation latency can take up to 2 hours in the worst-case

scenario.

- There are no key vaults in the scope of the policy assignment.
- There are no key vaults with certificates within the scope of the policy assignment.

Next Steps

- Learn more about the [Azure Policy service](#)
- See Key Vault samples: [Key Vault built-in policy definitions](#)

Integrate Key Vault with Azure Private Link

7 minutes to read • [Edit Online](#)

Azure Private Link Service enables you to access Azure Services (for example, Azure Key Vault, Azure Storage, and Azure Cosmos DB) and Azure hosted customer/partner services over a Private Endpoint in your virtual network.

An Azure Private Endpoint is a network interface that connects you privately and securely to a service powered by Azure Private Link. The private endpoint uses a private IP address from your VNet, effectively bringing the service into your VNet. All traffic to the service can be routed through the private endpoint, so no gateways, NAT devices, ExpressRoute or VPN connections, or public IP addresses are needed. Traffic between your virtual network and the service traverses over the Microsoft backbone network, eliminating exposure from the public Internet. You can connect to an instance of an Azure resource, giving you the highest level of granularity in access control.

For more information, see [What is Azure Private Link?](#)

Prerequisites

To integrate a key vault with Azure Private Link, you will need the following:

- A key vault.
- An Azure virtual network.
- A subnet in the virtual network.
- Owner or contributor permissions for both the key vault and the virtual network.

Your private endpoint and virtual network must be in the same region. When you select a region for the private endpoint using the portal, it will automatically filter only virtual networks that are in that region. Your key vault can be in a different region.

Your private endpoint uses a private IP address in your virtual network.

Establish a private link connection to Key Vault using the Azure portal

First, create a virtual network by following the steps in [Create a virtual network using the Azure portal](#)

You can then either create a new key vault, or establish a private link connection to an existing key vault.

Create a new key vault and establish a private link connection

You can create a new key Vault by following the steps in [Set and retrieve a secret from Azure Key Vault using the Azure portal](#)

After configuring the key vault basics, select the Networking tab and follow these steps:

1. Select the Private Endpoint radio button in the Networking tab.
2. Click the "+ Add" Button to add a private endpoint.

Create key vault

Basics Access policy Networking Tags Review + create

Network connectivity

You can connect to this key vault either publically, via public IP addresses or service endpoints, or privately, using a private endpoint.

Connectivity method

- Public endpoint (all networks)
- Public endpoint (selected networks)
- Private endpoint (preview)

Private endpoint

Create a private endpoint to allow a private connection to this resource. Additional private endpoint connections can be created within the key vault or private link center. [Learn more](#)

+ Add

Name	Subscription	Resource Group
Click on add button to add private endpoint		

3. In the "Location" field of the Create Private Endpoint Blade, select the region in which your virtual network is located.
4. In the "Name" field, create a descriptive name that will allow you to identify this private endpoint.
5. Select the virtual network and subnet you want this private endpoint to be created in from the dropdown menu.
6. Leave the "integrate with the private zone DNS" option unchanged.
7. Select "Ok".

Create private endpoint

Subscription * ⓘ Contoso IT - demo

Resource group * ⓘ (New) contoso-privatelink
Create new

Location * (US) East US

Name * ⓘ contoso-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 * ⓘ vnet-demo

Subnet * ⓘ demo (10.0.0.0/24)

Tip: If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement.

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 No

Private DNS Zone * ⓘ (New) privatelink.vaultcore.azure.net

You will now be able to see the configured private endpoint. You now have the option to delete and edit this private endpoint. Select the "Review + Create" button and create the key vault. It will take 5-10 minutes for the deployment to complete.

Establish a private link connection to an existing key vault

If you already have a key vault, you can create a private link connection by following these steps:

1. Sign in to the Azure portal.
2. In the search bar, type in "key vaults"
3. Select the key vault from the list to which you want to add a private endpoint.
4. Select the "Networking" tab under Settings
5. Select the Private endpoint connections tab at the top of the page
6. Select the "+ Private Endpoint" button at the top of the page.

The screenshot shows the Azure Key Vault Networking settings page. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events (preview), Keys, Secrets, Certificates, Access policies, Networking (which is selected and highlighted in grey), Properties, and Locks. The main area has a header with a search bar, a toolbar with buttons for Private endpoint, Approve, Reject, Remove, and Refresh, and filters for Connection name and Connection state. Below that, there's a table with columns for Provisioning state and Private endpoint. A message at the bottom says "No results".

Create a private endpoint (Preview)

1 Basics 2 Resource 3 Configuration 4 Tags 5 Review + create

Use private endpoints to privately connect to a service or resource. Your private endpoint must be in the same region as your virtual network, but can be in a different region from the private link resource that you are connecting to. [Learn more](#)

Project details

Subscription *

Resource group * [Create new](#)

Instance details

Name * ✓

Region * ✓

You can choose to create a private endpoint for any Azure resource in using this blade. You can either use the dropdown menus to select a resource type and select a resource in your directory, or you can connect to any Azure resource using a resource ID. Leave the "integrate with the private zone DNS" option unchanged.

Search (Ctrl+Shift+F)

Firewalls and virtual networks Private endpoint connections (preview)

+ Private endpoint Approve Reject Remove Refresh

Filter by name... All connection states

Connection name	Connection state	Provisioning state	Private endpoint
No results			

Create a private endpoint (Preview)

1 Basics 2 Resource 3 Configuration 4 Tags 5 Review + create

Use private endpoints to privately connect to a service or resource. Your private endpoint must be in the same region as your virtual network, but can be in a different region from the private link resource that you are connecting to. [Learn more](#)

Project details

Subscription *

Resource group * [Create new](#)

Instance details

Name * ✓

Region * ✓

Establish a private link connection to Key Vault using CLI

Login to Azure CLI

```
az login
```

Select your Azure Subscription

```
az account set --subscription {AZURE SUBSCRIPTION ID}
```

Create a new Resource Group

```
az group create -n {RG} -l {AZURE REGION}
```

Register Microsoft.KeyVault as a provider

```
az provider register -n Microsoft.KeyVault
```

Create a new Key Vault

```
az keyvault create --name {KEY VAULT NAME} --resource-group {RG} --location {AZURE REGION}
```

Turn on Key Vault Firewall

```
az keyvault update --name {KEY VAULT NAME} --resource-group {RG} --location {AZURE REGION} --default-action deny
```

Create a Virtual Network

```
az network vnet create --resource-group {RG} --name {vNet NAME} --location {AZURE REGION}
```

Add a subnet

```
az network vnet subnet create --resource-group {RG} --vnet-name {vNet NAME} --name {subnet NAME} --address-prefixes {addressPrefix}
```

Disable Virtual Network Policies

```
az network vnet subnet update --name {subnet NAME} --resource-group {RG} --vnet-name {vNet NAME} --disable-private-endpoint-network-policies true
```

Add a Private DNS Zone

```
az network private-dns zone create --resource-group {RG} --name privatelink.vaultcore.azure.net
```

Link Private DNS Zone to Virtual Network

```
az network private-dns link vnet create --resource-group {RG} --virtual-network {vNet NAME} --zone-name privatelink.vaultcore.azure.net --name {dnsZoneLinkName} --registration-enabled true
```

Create a Private Endpoint (Automatically Approve)

```
az network private-endpoint create --resource-group {RG} --vnet-name {vNet NAME} --subnet {subnet NAME} --name {Private Endpoint Name} --private-connection-resource-id "/subscriptions/{AZURE SUBSCRIPTION ID}/resourceGroups/{RG}/providers/Microsoft.KeyVault/vaults/ {KEY VAULT NAME}" --group-ids vault --connection-name {Private Link Connection Name} --location {AZURE REGION}
```

Create a Private Endpoint (Manually Request Approval)

```
az network private-endpoint create --resource-group {RG} --vnet-name {vNet NAME} --subnet {subnet NAME} --name {Private Endpoint Name} --private-connection-resource-id "/subscriptions/{AZURE SUBSCRIPTION ID}/resourceGroups/{RG}/providers/Microsoft.KeyVault/vaults/ {KEY VAULT NAME}" --group-ids vault --connection-name {Private Link Connection Name} --location {AZURE REGION} --manual-request
```

Show Connection Status

```
az network private-endpoint show --resource-group {RG} --name {Private Endpoint Name}
```

Manage private link connection

When you create a private endpoint, the connection must be approved. If the resource for which you are creating a private endpoint is in your directory, you will be able to approve the connection request provided you have sufficient permissions; if you are connecting to an Azure resource in another directory, you must wait for the owner of that resource to approve your connection request.

There are four provisioning states:

SERVICE PROVIDER ACTION	SERVICE CONSUMER PRIVATE ENDPOINT STATE	DESCRIPTION
None	Pending	Connection is created manually and is pending approval from the Private Link resource owner.
Approve	Approved	Connection was automatically or manually approved and is ready to be used.
Reject	Rejected	Connection was rejected by the private link resource owner.
Remove	Disconnected	Connection was removed by the private link resource owner, the private endpoint becomes informative and should be deleted for cleanup.

How to manage a private endpoint connection to Key Vault using the Azure portal

1. Log in to the Azure portal.
2. In the search bar, type in "key vaults"
3. Select the key vault that you want to manage.
4. Select the "Networking" tab.
5. If there are any connections that are pending, you will see a connection listed with "Pending" in the provisioning state.

6. Select the private endpoint you wish to approve
7. Select the approve button.
8. If there are any private endpoint connections you want to reject, whether it is a pending request or existing connection, select the connection and click the "Reject" button.

The screenshot shows the Azure Key Vault Networking blade for a vault named 'privatelink-vault-shane'. On the left, the 'Networking' tab is selected under the 'Settings' section. On the right, the 'Private endpoint connections (preview)' tab is selected. It displays two connections:

Connection name	Connection state	Provisioning state	Private endpoint
8789e66f5993493fa0b5617d2ac64979	Approved	Succeeded	private_endpoint-test-share
7347b07fb354bfff8ecd7365a26b3af	Approved	Succeeded	private-endpoint-test2

How to manage a private endpoint connection to Key Vault using Azure CLI

Approve a Private Link Connection Request

```
az keyvault private-endpoint-connection approve --approval-description {"OPTIONAL DESCRIPTION"} --resource-group {RG} --vault-name {KEY VAULT NAME} -name {PRIVATE LINK CONNECTION NAME}
```

Deny a Private Link Connection Request

```
az keyvault private-endpoint-connection reject --rejection-description {"OPTIONAL DESCRIPTION"} --resource-group {RG} --vault-name {KEY VAULT NAME} -name {PRIVATE LINK CONNECTION NAME}
```

Delete a Private Link Connection Request

```
az keyvault private-endpoint-connection delete --resource-group {RG} --vault-name {KEY VAULT NAME} --name {PRIVATE LINK CONNECTION NAME}
```

Validate that the private link connection works

You should validate that the resources within the same subnet of the private endpoint resource are connecting to your key vault over a private IP address, and that they have the correct private DNS zone integration.

First, create a virtual machine by following the steps in [Create a Windows virtual machine in the Azure portal](#)

In the "Networking" tab:

1. Specify Virtual network and Subnet. You can create a new virtual network or select an existing one. If selecting an existing one, make sure the region matches.
2. Specify a Public IP resource.
3. In the "NIC network security group", select "None".
4. In the "Load balancing", select "No".

Open the command line and run the following command:

```
nslookup <your-key-vault-name>.vault.azure.net
```

If you run the ns lookup command to resolve the IP address of a key vault over a public endpoint, you will see a result that looks like this:

```
c:\>nslookup <your-key-vault-name>.vault.azure.net  
  
Non-authoritative answer:  
Name:  
Address: (public IP address)  
Aliases: <your-key-vault-name>.vault.azure.net
```

If you run the ns lookup command to resolve the IP address of a key vault over a private endpoint, you will see a result that looks like this:

```
c:\>nslookup your_vault_name.vault.azure.net  
  
Non-authoritative answer:  
Name:  
Address: 10.1.0.5 (private IP address)  
Aliases: <your-key-vault-name>.vault.azure.net  
         <your-key-vault-name>.privatelink.vaultcore.azure.net
```

Limitations and Design Considerations

Pricing: For pricing information, see [Azure Private Link pricing](#).

Limitations: Private Endpoint for Azure Key Vault is only available in Azure public regions.

Maximum Number of Private Endpoints per Key Vault: 64.

Maximum Number of Key Vaults with Private Endpoints per Subscription: 64.

For more, see [Azure Private Link service: Limitations](#)

Next Steps

- Learn more about [Azure Private Link](#)
- Learn more about [Azure Key Vault](#)

Azure Key Vault logging

10 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

After you create one or more key vaults, you'll likely want to monitor how and when your key vaults are accessed, and by whom. You can do this by enabling logging for Azure Key Vault, which saves information in an Azure storage account that you provide. A new container named **insights-logs-auditevent** is automatically created for your specified storage account. You can use this same storage account for collecting logs for multiple key vaults.

You can access your logging information 10 minutes (at most) after the key vault operation. In most cases, it will be quicker than this. It's up to you to manage your logs in your storage account:

- Use standard Azure access control methods to secure your logs by restricting who can access them.
- Delete logs that you no longer want to keep in your storage account.

Use this tutorial to help you get started with Azure Key Vault logging. You'll create a storage account, enable logging, and interpret the collected log information.

NOTE

This tutorial does not include instructions for how to create key vaults, keys, or secrets. For this information, see [What is Azure Key Vault?](#). Or, for cross-platform Azure CLI instructions, see [this equivalent tutorial](#).

This article provides Azure PowerShell instructions for updating diagnostic logging. You can also update diagnostic logging by using Azure Monitor in the **Diagnostic logs** section of the Azure portal.

For overview information about Key Vault, see [What is Azure Key Vault?](#). For information about where Key Vault is available, see the [pricing page](#).

Prerequisites

To complete this tutorial, you must have the following:

- An existing key vault that you have been using.
- Azure PowerShell, minimum version of 1.0.0. To install Azure PowerShell and associate it with your Azure subscription, see [How to install and configure Azure PowerShell](#). If you have already installed Azure PowerShell and don't know the version, from the Azure PowerShell console, enter `$PSVersionTable.PSVersion`.
- Sufficient storage on Azure for your Key Vault logs.

Connect to your key vault subscription

The first step in setting up key logging is to point Azure PowerShell to the key vault that you want to log.

Start an Azure PowerShell session and sign in to your Azure account by using the following command:

```
Connect-AzAccount
```

In the pop-up browser window, enter your Azure account user name and password. Azure PowerShell gets all the subscriptions that are associated with this account. By default, PowerShell uses the first one.

You might have to specify the subscription that you used to create your key vault. Enter the following command to see the subscriptions for your account:

```
Get-AzSubscription
```

Then, to specify the subscription that's associated with the key vault you'll be logging, enter:

```
Set-AzContext -SubscriptionId <subscription ID>
```

Pointing PowerShell to the right subscription is an important step, especially if you have multiple subscriptions associated with your account. For more information about configuring Azure PowerShell, see [How to install and configure Azure PowerShell](#).

Create a storage account for your logs

Although you can use an existing storage account for your logs, we'll create a storage account that will be dedicated to Key Vault logs. For convenience for when we have to specify this later, we'll store the details in a variable named **sa**.

For additional ease of management, we'll also use the same resource group as the one that contains the key vault. From the [getting-started tutorial](#), this resource group is named **ContosoResourceGroup**, and we'll continue to use the East Asia location. Replace these values with your own, as applicable:

```
$sa = New-AzStorageAccount -ResourceGroupName ContosoResourceGroup -Name contosokeyvaultlogs -Type Standard_LRS -Location 'East Asia'
```

NOTE

If you decide to use an existing storage account, it must use the same subscription as your key vault. And it must use the Azure Resource Manager deployment model, rather than the classic deployment model.

Identify the key vault for your logs

In the [getting-started tutorial](#), the key vault name was **ContosoKeyVault**. We'll continue to use that name and store the details in a variable named **kv**:

```
$kv = Get-AzKeyVault -VaultName 'ContosoKeyVault'
```

Enable logging

To enable logging for Key Vault, we'll use the **Set-AzDiagnosticSetting** cmdlet, together with the variables that we created for the new storage account and the key vault. We'll also set the **-Enabled** flag to **\$true** and set the category to **AuditEvent** (the only category for Key Vault logging):

```
Set-AzDiagnosticSetting -ResourceId $kv.ResourceId -StorageAccountId $sa.Id -Enabled $true -Category AuditEvent
```

The output looks like this:

```
StorageAccountId    : /subscriptions/<subscription-GUID>/resourceGroups/ContosoResourceGroup/providers/Microsoft.Storage/storageAccounts/ContosoKeyVaultLogs
ServiceBusRuleId   :
StorageAccountName :
    Logs
        Enabled      : True
        Category     : AuditEvent
        RetentionPolicy
            Enabled : False
            Days      : 0
```

This output confirms that logging is now enabled for your key vault, and it will save information to your storage account.

Optionally, you can set a retention policy for your logs such that older logs are automatically deleted. For example, set retention policy by setting the **-RetentionEnabled** flag to **\$true**, and set the **-RetentionInDays** parameter to **90** so that logs older than 90 days are automatically deleted.

```
Set-AzDiagnosticSetting -ResourceId $kv.ResourceId -StorageAccountId $sa.Id -Enabled $true -Category AuditEvent -RetentionEnabled $true -RetentionInDays 90
```

What's logged:

- All authenticated REST API requests, including failed requests as a result of access permissions, system errors, or bad requests.
- Operations on the key vault itself, including creation, deletion, setting key vault access policies, and updating key vault attributes such as tags.
- Operations on keys and secrets in the key vault, including:
 - Creating, modifying, or deleting these keys or secrets.
 - Signing, verifying, encrypting, decrypting, wrapping and unwrapping keys, getting secrets, and listing keys and secrets (and their versions).
- Unauthenticated requests that result in a 401 response. Examples are requests that don't have a bearer token, that are malformed or expired, or that have an invalid token.

Access your logs

Key Vault logs are stored in the **insights-logs-auditevent** container in the storage account that you provided. To view the logs, you have to download blobs.

First, create a variable for the container name. You'll use this variable throughout the rest of the walkthrough.

```
$container = 'insights-logs-auditevent'
```

To list all the blobs in this container, enter:

```
Get-AzStorageBlob -Container $container -Context $sa.Context
```

The output looks similar to this:

```
Container Uri: https://contosokeyvaultlogs.blob.core.windows.net/insights-logs-auditevent
```

Name

```
resourceId=/SUBSCRIPTIONS/361DA5D4-A47A-4C79-AFDD-
XXXXXXXXXXXX/RESOURCEGROUPS/CONTOSORESOURCEGROUP/PROVIDERS/MICROSOFT.KEYVAULT/VAULTS/CONTOSOKEYVAULT/y=2016/m
=01/d=05/h=01/m=00/PT1H.json
```

```
resourceId=/SUBSCRIPTIONS/361DA5D4-A47A-4C79-AFDD-
XXXXXXXXXXXX/RESOURCEGROUPS/CONTOSORESOURCEGROUP/PROVIDERS/MICROSOFT.KEYVAULT/VAULTS/CONTOSOKEYVAULT/y=2016/m
=01/d=04/h=02/m=00/PT1H.json
```

```
resourceId=/SUBSCRIPTIONS/361DA5D4-A47A-4C79-AFDD-
XXXXXXXXXXXX/RESOURCEGROUPS/CONTOSORESOURCEGROUP/PROVIDERS/MICROSOFT.KEYVAULT/VAULTS/CONTOSOKEYVAULT/y=2016/m
=01/d=04/h=18/m=00/PT1H.json
```

As you can see from this output, the blobs follow a naming convention:

```
resourceId=<ARM resource ID>/y=<year>/m=<month>/d=<day of month>/h=<hour>/m=<minute>/filename.json
```

The date and time values use UTC.

Because you can use the same storage account to collect logs for multiple resources, the full resource ID in the blob name is useful to access or download just the blobs that you need. But before we do that, we'll first cover how to download all the blobs.

Create a folder to download the blobs. For example:

```
New-Item -Path 'C:\Users\username\ContosoKeyVaultLogs' -ItemType Directory -Force
```

Then get a list of all blobs:

```
$blobs = Get-AzStorageBlob -Container $container -Context $sa.Context
```

Pipe this list through **Get-AzStorageBlobContent** to download the blobs to the destination folder:

```
$blobs | Get-AzStorageBlobContent -Destination C:\Users\username\ContosoKeyVaultLogs'
```

When you run this second command, the / delimiter in the blob names creates a full folder structure under the destination folder. You'll use this structure to download and store the blobs as files.

To selectively download blobs, use wildcards. For example:

- If you have multiple key vaults and want to download logs for just one key vault, named CONTOSOKEYVAULT3:

```
Get-AzStorageBlob -Container $container -Context $sa.Context -Blob '/VAULTS/CONTOSOKEYVAULT3'
```

- If you have multiple resource groups and want to download logs for just one resource group, use -Blob '/RESOURCEGROUPS/<resource group name>/*' :

```
Get-AzStorageBlob -Container $container -Context $sa.Context -Blob
'/RESOURCEGROUPS/CONTOSORESOURCEGROUP3/*'
```

- If you want to download all the logs for the month of January 2019, use -Blob '/year=2019/m=01/*' :

```
Get-AzStorageBlob -Container $container -Context $sa.Context -Blob '*/year=2016/m=01/*'
```

You're now ready to start looking at what's in the logs. But before we move on to that, you should know two more commands:

- To query the status of diagnostic settings for your key vault resource:

```
Get-AzDiagnosticSetting -ResourceId $kv.ResourceId
```

- To disable logging for your key vault resource:

```
Set-AzDiagnosticSetting -ResourceId $kv.ResourceId -StorageAccountId $sa.Id -Enabled $false -Category AuditEvent
```

Interpret your Key Vault logs

Individual blobs are stored as text, formatted as a JSON blob. Let's look at an example log entry. Run this command:

```
Get-AzKeyVault -VaultName 'contosokeyvault'
```

It returns a log entry similar to this one:

```
{
  "records": [
    {
      "time": "2016-01-05T01:32:01.2691226Z",
      "resourceId": "/SUBSCRIPTIONS/361DA5D4-A47A-4C79-AFDD-
XXXXXXXXXXXX/RESOURCEGROUPS/CONTOSOGROUP/PROVIDERS/MICROSOFT.KEYVAULT/VAULTS/CONTOSOKEYVAULT",
      "operationName": "VaultGet",
      "operationVersion": "2015-06-01",
      "category": "AuditEvent",
      "resultType": "Success",
      "resultSignature": "OK",
      "resultDescription": "",
      "durationMs": "78",
      "callerIpAddress": "104.40.82.76",
      "correlationId": "",
      "identity": {"claim": {
        "http://schemas.microsoft.com/identity/claims/objectidentifier": "d9da5048-2737-4770-bd64-
XXXXXXXXXXXX", "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn": "live.com#username@outlook.com", "ap-
pid": "1950a258-227b-4e31-a9cf-XXXXXXXXXXXX"}, "properties": {"clientInfo": "azure-resource-manager/2.0", "requestUri": "https://control-prod-
wus.vaultcore.azure.net/subscriptions/361da5d4-a47a-4c79-afdd-
XXXXXXXXXXXX/resourcegroups/contosoresourcegroup/providers/Microsoft.KeyVault/vaults/contosokeyvault?api-
version=2015-06-01", "id": "https://contosokeyvault.vault.azure.net/", "httpStatusCode": 200}}
    }
  ]
}
```

The following table lists the field names and descriptions:

FIELD NAME	DESCRIPTION
time	Date and time in UTC.
resourceId	Azure Resource Manager resource ID. For Key Vault logs, this is always the Key Vault resource ID.

FIELD NAME	DESCRIPTION
operationName	Name of the operation, as documented in the next table.
operationVersion	REST API version requested by the client.
category	Type of result. For Key Vault logs, AuditEvent is the single, available value.
resultType	Result of the REST API request.
resultSignature	HTTP status.
resultDescription	Additional description about the result, when available.
durationMs	Time it took to service the REST API request, in milliseconds. This does not include the network latency, so the time you measure on the client side might not match this time.
callerIpAddress	IP address of the client that made the request.
correlationId	An optional GUID that the client can pass to correlate client-side logs with service-side (Key Vault) logs.
identity	Identity from the token that was presented in the REST API request. This is usually a "user," a "service principal," or the combination "user+appId," as in the case of a request that results from an Azure PowerShell cmdlet.
properties	Information that varies based on the operation (operationName). In most cases, this field contains client information (the user agent string passed by the client), the exact REST API request URI, and the HTTP status code. In addition, when an object is returned as a result of a request (for example, KeyCreate or VaultGet), it also contains the key URI (as "id"), vault URI, or secret URI.

The **operationName** field values are in *ObjectVerb* format. For example:

- All key vault operations have the `Vault<action>` format, such as `VaultGet` and `VaultCreate`.
- All key operations have the `Key<action>` format, such as `KeySign` and `KeyList`.
- All secret operations have the `Secret<action>` format, such as `SecretGet` and `SecretListVersions`.

The following table lists the **operationName** values and corresponding REST API commands:

OPERATIONNAME	REST API COMMAND
Authentication	Authenticate via Azure Active Directory endpoint
VaultGet	Get information about a key vault
VaultPut	Create or update a key vault
VaultDelete	Delete a key vault

OPERATIONNAME	REST API COMMAND
VaultPatch	Update a key vault
VaultList	List all key vaults in a resource group
KeyCreate	Create a key
KeyGet	Get information about a key
KeyImport	Import a key into a vault
KeyBackup	Back up a key
KeyDelete	Delete a key
KeyRestore	Restore a key
KeySign	Sign with a key
KeyVerify	Verify with a key
KeyWrap	Wrap a key
KeyUnwrap	Unwrap a key
KeyEncrypt	Encrypt with a key
KeyDecrypt	Decrypt with a key
KeyUpdate	Update a key
KeyList	List the keys in a vault
KeyListVersions	List the versions of a key
SecretSet	Create a secret
SecretGet	Get a secret
SecretUpdate	Update a secret
SecretDelete	Delete a secret
SecretList	List secrets in a vault
SecretListVersions	List versions of a secret

Use Azure Monitor logs

You can use the Key Vault solution in Azure Monitor logs to review Key Vault **AuditEvent** logs. In Azure Monitor logs, you use log queries to analyze data and get the information you need.

For more information, including how to set this up, see [Azure Key Vault solution in Azure Monitor logs](#). This article also contains instructions if you need to migrate from the old Key Vault solution that was offered during the Azure Monitor logs preview, where you first routed your logs to an Azure storage account and configured Azure Monitor logs to read from there.

Next steps

For a tutorial that uses Azure Key Vault in a .NET web application, see [Use Azure Key Vault from a web application](#).

For programming references, see [the Azure Key Vault developer's guide](#).

For a list of Azure PowerShell 1.0 cmdlets for Azure Key Vault, see [Azure Key Vault cmdlets](#).

For a tutorial on key rotation and log auditing with Azure Key Vault, see [Set up Key Vault with end-to-end key rotation and auditing](#).

Access Azure Key Vault behind a firewall

2 minutes to read • [Edit Online](#)

What ports, hosts, or IP addresses should I open to enable my key vault client application behind a firewall to access key vault?

To access a key vault, your key vault client application has to access multiple endpoints for various functionalities:

- Authentication via Azure Active Directory (Azure AD).
- Management of Azure Key Vault. This includes creating, reading, updating, deleting, and setting access policies through Azure Resource Manager.
- Accessing and managing objects (keys and secrets) stored in Key Vault itself, going through the Key Vault-specific endpoint (for example, <https://yourvaultname.vault.azure.net>).

Depending on your configuration and environment, there are some variations.

Ports

All traffic to a key vault for all three functions (authentication, management, and data plane access) goes over HTTPS: port 443. However, there will occasionally be HTTP (port 80) traffic for CRL. Clients that support OCSP shouldn't reach CRL, but may occasionally reach <http://cdp1.public-trust.com/CRL/Omniroot2025.crl>.

Authentication

Key vault client applications will need to access Azure Active Directory endpoints for authentication. The endpoint used depends on the Azure AD tenant configuration, the type of principal (user principal or service principal), and the type of account--for example, a Microsoft account or a work or school account.

PRINCIPAL TYPE	ENDPOINT:PORT
User using Microsoft account (for example, user@hotmail.com)	Global: login.microsoftonline.com:443 Azure China: login.chinacloudapi.cn:443 Azure US Government: login.microsoftonline.us:443 Azure Germany: login.microsoftonline.de:443 and login.live.com:443

PRINCIPAL TYPE	ENDPOINT:PORT
User or service principal using a work or school account with Azure AD (for example, user@contoso.com)	<p>Global: login.microsoftonline.com:443</p> <p>Azure China: login.chinacloudapi.cn:443</p> <p>Azure US Government: login.microsoftonline.us:443</p> <p>Azure Germany: login.microsoftonline.de:443</p>
User or service principal using a work or school account, plus Active Directory Federation Services (AD FS) or other federated endpoint (for example, user@contoso.com)	All endpoints for a work or school account, plus AD FS or other federated endpoints

There are other possible complex scenarios. Refer to [Azure Active Directory Authentication Flow](#), [Integrating Applications with Azure Active Directory](#), and [Active Directory Authentication Protocols](#) for additional information.

Key Vault management

For Key Vault management (CRUD and setting access policy), the key vault client application needs to access an Azure Resource Manager endpoint.

TYPE OF OPERATION	ENDPOINT:PORT
Key Vault control plane operations via Azure Resource Manager	<p>Global: management.azure.com:443</p> <p>Azure China: management.chinacloudapi.cn:443</p> <p>Azure US Government: management.usgovcloudapi.net:443</p> <p>Azure Germany: management.microsoftazure.de:443</p>
Microsoft Graph API	<p>Global: graph.microsoft.com:443</p> <p>Azure China: graph.chinacloudapi.cn:443</p> <p>Azure US Government: graph.microsoft.com:443</p> <p>Azure Germany: graph.cloudapi.de:443</p>

Key Vault operations

For all key vault object (keys and secrets) management and cryptographic operations, the key vault client needs to access the key vault endpoint. The endpoint DNS suffix varies depending on the location of your key vault. The key vault endpoint is of the format *vault-name.region-specific-dns-suffix*, as described in the following table.

TYPE OF OPERATION	ENDPOINT:PORT
Operations including cryptographic operations on keys; creating, reading, updating, and deleting keys and secrets; setting or getting tags and other attributes on key vault objects (keys or secrets)	<p>Global: <vault-name>.vault.azure.net:443</p> <p>Azure China: <vault-name>.vault.azure.cn:443</p> <p>Azure US Government: <vault-name>.vault.usgovcloudapi.net:443</p> <p>Azure Germany: <vault-name>.vault.microsoftazure.de:443</p>

IP address ranges

The Key Vault service uses other Azure resources like PaaS infrastructure. So it's not possible to provide a specific range of IP addresses that Key Vault service endpoints will have at any particular time. If your firewall supports only IP address ranges, refer to the [Microsoft Azure Datacenter IP Ranges](#) document. Authentication and Identity (Azure Active Directory) is a global service and may fail over to other regions or move traffic without notice. In this scenario, all of the IP ranges listed in [Authentication and Identity IP Addresses](#) should be added to the firewall.

Next steps

If you have questions about Key Vault, visit the [Azure Key Vault Forums](#).

Azure Key Vault availability and redundancy

2 minutes to read • [Edit Online](#)

Azure Key Vault features multiple layers of redundancy to make sure that your keys and secrets remain available to your application even if individual components of the service fail.

The contents of your key vault are replicated within the region and to a secondary region at least 150 miles away but within the same geography. This maintains high durability of your keys and secrets. See the [Azure paired regions](#) document for details on specific region pairs.

If individual components within the key vault service fail, alternate components within the region step in to serve your request to make sure that there is no degradation of functionality. You do not need to take any action to trigger this. It happens automatically and will be transparent to you.

In the rare event that an entire Azure region is unavailable, the requests that you make of Azure Key Vault in that region are automatically routed (*failed over*) to a secondary region. When the primary region is available again, requests are routed back (*failed back*) to the primary region. Again, you do not need to take any action because this happens automatically.

Through this high availability design, Azure Key Vault requires no downtime for maintenance activities.

There are a few caveats to be aware of:

- In the event of a region failover, it may take a few minutes for the service to fail over. Requests that are made during this time may fail until the failover completes.
- After a failover is complete, your key vault is in read-only mode. Requests that are supported in this mode are:
 - List key vaults
 - Get properties of key vaults
 - List secrets
 - Get secrets
 - List keys
 - Get (properties of) keys
 - Encrypt
 - Decrypt
 - Wrap
 - Unwrap
 - Verify
 - Sign
 - Backup
- After a failover is failed back, all request types (including read *and* write requests) are available.

Change a key vault tenant ID after a subscription move

2 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

When you create a new key vault in a subscription, it is automatically tied to the default Azure Active Directory tenant ID for that subscription. All access policy entries are also tied to this tenant ID.

If you move your Azure subscription from tenant A to tenant B, your existing key vaults are inaccessible by the principals (users and applications) in tenant B. To fix this issue, you need to:

- Change the tenant ID associated with all existing key vaults in the subscription to tenant B.
- Remove all existing access policy entries.
- Add new access policy entries associated with tenant B.

For example, if you have key vault 'myvault' in a subscription that has been moved from tenant A to tenant B, you can use Azure PowerShell to change the tenant ID and remove old access policies.

```
Select-AzSubscription -SubscriptionId <your-subscriptionId>          # Select your Azure Subscription
$vaultResourceId = (Get-AzKeyVault -VaultName myvault).ResourceId        # Get your key vault's Resource ID
$vault = Get-AzResource -ResourceId $vaultResourceId -ExpandProperties    # Get the properties for your key
vault
$vault.Properties.TenantId = (Get-AzContext).Tenant.TenantId               # Change the Tenant that your key
vault resides in
$vault.Properties.AccessPolicies = @()                                       # Access policies can be updated
with real
it does not need to be                                                 # applications/users/rights so that
# done after this whole activity. Here we are not setting
# any access policies.
Set-AzResource -ResourceId $vaultResourceId -Properties $vault.Properties   # Modifies the key vault's
properties.
```

Or you can use the Azure CLI.

```
az account set -s <your-subscriptionId>
tenantId=$(az account show --query tenantId)
az keyvault update -n myvault --remove Properties.accessPolicies
az keyvault update -n myvault --set Properties.tenantId=$tenantId
# Select your Azure Subscription
# Get your tenantId
# Remove the access policies
# Update the key vault tenantID
```

Now that your vault is associated with the correct tenant ID and old access policy entries are removed, set new access policy entries with the Azure PowerShell [Set-AzKeyVaultAccessPolicy](#) cmdlet or the Azure CLI [az keyvault set-policy](#) command.

If you are using a managed identity for Azure resources, you will need to update it to the new Azure AD tenant as well. For more information on managed identities, see [Provide Key Vault authentication with a managed identity](#).

If you are using MSI, you'll also have to update the MSI identity since the old identity will no longer be in the correct AAD tenant.

Next steps

If you have questions about Azure Key Vault, visit the [Azure Key Vault Forums](#).

Manage Key Vault using the Azure CLI

9 minutes to read • [Edit Online](#)

This article covers how to get started working with Azure Key Vault using the Azure CLI. You can see information on:

- How to create a hardened container (a vault) in Azure
- Adding a key, secret, or certificate to the key vault
- Registering an application with Azure Active Directory
- Authorizing an application to use a key or secret
- Setting key vault advanced access policies
- Working with Hardware security modules (HSMs)
- Deleting the key vault and associated keys and secrets
- Miscellaneous Azure Cross-Platform Command-line Interface Commands

Azure Key Vault is available in most regions. For more information, see the [Key Vault pricing page](#).

NOTE

This article does not include instructions on how to write the Azure application that one of the steps includes, which shows how to authorize an application to use a key or secret in the key vault.

For an overview of Azure Key Vault, see [What is Azure Key Vault?](#) If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

To use the Azure CLI commands in this article, you must have the following items:

- A subscription to Microsoft Azure. If you don't have one, you can sign up for a [free trial](#).
- Azure Command-Line Interface version 2.0 or later. To install the latest version, see [Install the Azure CLI](#).
- An application that will be configured to use the key or password that you create in this article. A sample application is available from the [Microsoft Download Center](#). For instructions, see the included Readme file.

Getting help with Azure Cross-Platform Command-Line Interface

This article assumes that you're familiar with the command-line interface (Bash, Terminal, Command prompt).

The --help or -h parameter can be used to view help for specific commands. Alternately, The Azure help [command] [options] format can also be used too. When in doubt about the parameters needed by a command, refer to help. For example, the following commands all return the same information:

```
az account set --help  
az account set -h
```

You can also read the following articles to get familiar with Azure Resource Manager in Azure Cross-Platform Command-Line Interface:

- [Install Azure CLI](#)
- [Get started with Azure CLI](#)

How to create a hardened container (a vault) in Azure

Vaults are secured containers backed by hardware security modules. Vaults help reduce the chances of accidental loss of security information by centralizing the storage of application secrets. Key Vaults also control and log the access to anything stored in them. Azure Key Vault can handle requesting and renewing Transport Layer Security (TLS) certificates, providing the features required for a robust certificate lifecycle management solution. In the next steps, you will create a vault.

Connect to your subscriptions

To sign in interactively, use the following command:

```
az login
```

To sign in using an organizational account, you can pass in your username and password.

```
az login -u username@domain.com -p password
```

If you have more than one subscription and need to specify which to use, type the following to see the subscriptions for your account:

```
az account list
```

Specify a subscription with the subscription parameter.

```
az account set --subscription <subscription name or ID>
```

For more information about configuring Azure Cross-Platform Command-Line Interface, see [Install Azure CLI](#).

Create a new resource group

When using Azure Resource Manager, all related resources are created inside a resource group. You can create a key vault in an existing resource group. If you want to use a new resource group, you can create a new one.

```
az group create -n "ContosoResourceGroup" -l "East Asia"
```

The first parameter is resource group name and the second parameter is the location. To get a list of all possible locations type:

```
az account list-locations
```

Register the Key Vault resource provider

You may see the error "The subscription is not registered to use namespace 'Microsoft.KeyVault'" when you try to create a new key vault. If that message appears, make sure that Key Vault resource provider is registered in your subscription. This is a one-time operation for each subscription.

```
az provider register -n Microsoft.KeyVault
```

Create a key vault

Use the `az keyvault create` command to create a key vault. This script has three mandatory parameters: a resource group name, a key vault name, and the geographic location.

To create a new vault with the name **ContosoKeyVault**, in the resource group **ContosoResourceGroup**, residing in the **East Asia** location, type:

```
az keyvault create --name "ContosoKeyVault" --resource-group "ContosoResourceGroup" --location "East Asia"
```

The output of this command shows properties of the key vault that you've created. The two most important properties are:

- **name**: In the example, the name is ContosoKeyVault. You'll use this name for other Key Vault commands.
- **vaultUri**: In the example, the URI is <https://contosokeyvault.vault.azure.net>. Applications that use your vault through its REST API must use this URI.

Your Azure account is now authorized to perform any operations on this key vault. As of yet, nobody else is authorized.

Adding a key, secret, or certificate to the key vault

If you want Azure Key Vault to create a software-protected key for you, use the `az key create` command.

```
az keyvault key create --vault-name "ContosoKeyVault" --name "ContosoFirstKey" --protection software
```

If you have an existing key in a .pem file, you can upload it to Azure Key Vault. You can choose to protect the key with software or HSM. This example imports the key from the .pem file and protect it with software, using the password "hVFkk965BuUv":

```
az keyvault key import --vault-name "ContosoKeyVault" --name "ContosoFirstKey" --pem-file "./softkey.pem" --pem-password "hVFkk965BuUv" --protection software
```

You can now reference the key that you created or uploaded to Azure Key Vault, by using its URI. Use <https://ContosoKeyVault.vault.azure.net/keys/ContosoFirstKey> to always get the current version. Use [https://\[keyvault-name\].vault.azure.net/keys/\[keyname\]/\[key-unique-id\]](https://[keyvault-name].vault.azure.net/keys/[keyname]/[key-unique-id]) to get this specific version. For example, <https://ContosoKeyVault.vault.azure.net/keys/ContosoFirstKey/cgacf4f763ar42ffb0a1gca546aygd87>.

Add a secret to the vault, which is a password named SQLPassword, and that has the value of "hVFkk965BuUv" to Azure Key Vaults.

```
az keyvault secret set --vault-name "ContosoKeyVault" --name "SQLPassword" --value "hVFkk965BuUv"
```

Reference this password by using its URI. Use <https://ContosoVault.vault.azure.net/secrets/SQLPassword> to always get the current version, and [https://\[keyvault-name\].vault.azure.net/secret/\[secret-name\]/\[secret-unique-id\]](https://[keyvault-name].vault.azure.net/secret/[secret-name]/[secret-unique-id]) to get this specific version. For example,

<https://ContosoVault.vault.azure.net/secrets/SQLPassword/90018dbb96a84117a0d2847ef8e7189d>.

Import a certificate to the vault using a .pem or .pfx.

```
az keyvault certificate import --vault-name "ContosoKeyVault" --file "c:\cert\cert.pfx" --name "ContosoCert" --password "hVFkk965BuUv"
```

Let's view the key, secret, or certificate that you created:

- To view your keys, type:

```
az keyvault key list --vault-name "ContosoKeyVault"
```

- To view your secrets, type:

```
az keyvault secret list --vault-name "ContosoKeyVault"
```

- To view certificates, type:

```
az keyvault certificate list --vault-name "ContosoKeyVault"
```

Registering an application with Azure Active Directory

This step would usually be done by a developer, on a separate computer. It isn't specific to Azure Key Vault but is included here, for awareness. To complete the app registration, your account, the vault, and the application need to be in the same Azure directory.

Applications that use a key vault must authenticate by using a token from Azure Active Directory. The owner of the application must register it in Azure Active Directory first. At the end of registration, the application owner gets the following values:

- An **Application ID** (also known as the AAD Client ID or appId)
- An **authentication key** (also known as the shared secret).

The application must present both these values to Azure Active Directory, to get a token. How an application is configured to get a token will depend on the application. For the [Key Vault sample application](#), the application owner sets these values in the app.config file.

For detailed steps on registering an application with Azure Active Directory you should review the articles titled [Integrating applications with Azure Active Directory](#), [Use portal to create an Azure Active Directory application and service principal that can access resources](#), and [Create an Azure service principal with the Azure CLI](#).

To register an application in Azure Active Directory:

```
az ad sp create-for-rbac -n "MyApp" --password "hVFkk965BuUv" --skip-assignment  
# If you don't specify a password, one will be created for you.
```

Authorizing an application to use a key or secret

To authorize the application to access the key or secret in the vault, use the `az keyvault set-policy` command.

For example, if your vault name is ContosoKeyVault, the application has an appId of 8f8c4bbd-485b-45fd-98f7-ec6300b7b4ed, and you want to authorize the application to decrypt and sign with keys in your vault, use the following command:

```
az keyvault set-policy --name "ContosoKeyVault" --spn 8f8c4bbd-485b-45fd-98f7-ec6300b7b4ed --key-permissions  
decrypt sign
```

To authorize the same application to read secrets in your vault, type the following command:

```
az keyvault set-policy --name "ContosoKeyVault" --spn 8f8c4bbd-485b-45fd-98f7-ec6300b7b4ed --secret-  
permissions get
```

Setting key vault advanced access policies

Use [az keyvault update](#) to enable advanced policies for the key vault.

Enable Key Vault for deployment: Allows virtual machines to retrieve certificates stored as secrets from the vault.

```
az keyvault update --name "ContosoKeyVault" --resource-group "ContosoResourceGroup" --enabled-for-deployment "true"
```

Enable Key Vault for disk encryption: Required when using the vault for Azure Disk encryption.

```
az keyvault update --name "ContosoKeyVault" --resource-group "ContosoResourceGroup" --enabled-for-disk-encryption "true"
```

Enable Key Vault for template deployment: Allows Resource Manager to retrieve secrets from the vault.

```
az keyvault update --name "ContosoKeyVault" --resource-group "ContosoResourceGroup" --enabled-for-template-deployment "true"
```

Working with Hardware security modules (HSMs)

For added assurance, you can import or generate keys from hardware security modules (HSMs) that never leave the HSM boundary. The HSMs are FIPS 140-2 Level 2 validated. If this requirement doesn't apply to you, skip this section and go to [Delete the key vault and associated keys and secrets](#).

To create these HSM-protected keys, you must have a vault subscription that supports HSM-protected keys.

When you create the keyvault, add the 'sku' parameter:

```
az keyvault create --name "ContosoKeyVaultHSM" --resource-group "ContosoResourceGroup" --location "East Asia" --sku "Premium"
```

You can add software-protected keys (as shown earlier) and HSM-protected keys to this vault. To create an HSM-protected key, set the Destination parameter to 'HSM':

```
az keyvault key create --vault-name "ContosoKeyVaultHSM" --name "ContosoFirstHSMKey" --protection "hsm"
```

You can use the following command to import a key from a .pem file on your computer. This command imports the key into HSMs in the Key Vault service:

```
az keyvault key import --vault-name "ContosoKeyVaultHSM" --name "ContosoFirstHSMKey" --pem-file "./softkey.pem" --protection "hsm" --pem-password "PaASSWORD"
```

The next command imports a "bring your own key" (BYOK) package. This lets you generate your key in your local HSM, and transfer it to HSMs in the Key Vault service, without the key leaving the HSM boundary:

```
az keyvault key import --vault-name "ContosoKeyVaultHSM" --name "ContosoFirstHSMKey" --byok-file "./ITByok.byok" --protection "hsm"
```

For more detailed instructions about how to generate this BYOK package, see [How to use HSM-Protected Keys with Azure Key Vault](#).

Deleting the key vault and associated keys and secrets

If you no longer need the key vault and its keys or secrets, you can delete the key vault by using the

```
az keyvault delete
```

```
az keyvault delete --name "ContosoKeyVault"
```

Or, you can delete an entire Azure resource group, which includes the key vault and any other resources that you included in that group:

```
az group delete --name "ContosoResourceGroup"
```

Miscellaneous Azure Cross-Platform Command-line Interface Commands

Other commands that you might find useful for managing Azure Key Vault.

This command lists a tabular display of all keys and selected properties:

```
az keyvault key list --vault-name "ContosoKeyVault"
```

This command displays a full list of properties for the specified key:

```
az keyvault key show --vault-name "ContosoKeyVault" --name "ContosoFirstKey"
```

This command lists a tabular display of all secret names and selected properties:

```
az keyvault secret list --vault-name "ContosoKeyVault"
```

Here's an example of how to remove a specific key:

```
az keyvault key delete --vault-name "ContosoKeyVault" --name "ContosoFirstKey"
```

Here's an example of how to remove a specific secret:

```
az keyvault secret delete --vault-name "ContosoKeyVault" --name "SQLPassword"
```

Next steps

- For complete Azure CLI reference for key vault commands, see [Key Vault CLI reference](#).
- For programming references, see [the Azure Key Vault developer's guide](#)
- For information on Azure Key Vault and HSMs, see [How to use HSM-Protected Keys with Azure Key Vault](#).

How to use Key Vault soft-delete with CLI

7 minutes to read • [Edit Online](#)

Azure Key Vault's soft delete feature allows recovery of deleted vaults and vault objects. Specifically, soft-delete addresses the following scenarios:

- Support for recoverable deletion of a key vault
- Support for recoverable deletion of key vault objects; keys, secrets, and certificates

Prerequisites

- Azure CLI - If you don't have this setup for your environment, see [Manage Key Vault using Azure CLI](#).

For Key Vault specific reference information for CLI, see [Azure CLI Key Vault reference](#).

Required permissions

Key Vault operations are separately managed via role-based access control (RBAC) permissions as follows:

OPERATION	DESCRIPTION	USER PERMISSION
List	Lists deleted key vaults.	Microsoft.KeyVault/deletedVaults/read
Recover	Restores a deleted key vault.	Microsoft.KeyVault/vaults/write
Purge	Permanently removes a deleted key vault and all its contents.	Microsoft.KeyVault/locations/deletedVaults/purge/action

For more information on permissions and access control, see [Secure your key vault](#).

Enabling soft-delete

You enable "soft-delete" to allow recovery of a deleted key vault, or objects stored in a key vault.

IMPORTANT

Enabling 'soft delete' on a key vault is an irreversible action. Once the soft-delete property has been set to "true", it cannot be changed or removed.

Existing key vault

For an existing key vault named ContosoVault, enable soft-delete as follows.

```
az keyvault update -n ContosoVault --enable-soft-delete true
```

New key vault

Enabling soft-delete for a new key vault is done at creation time by adding the soft-delete enable flag to your create command.

```
az keyvault create --name ContosoVault --resource-group ContosoRG --enable-soft-delete true --location westus
```

Verify soft-delete enablement

To verify that a key vault has soft-delete enabled, run the *show* command and look for the 'Soft Delete Enabled?' attribute:

```
az keyvault show --name ContosoVault
```

Deleting a soft-delete protected key vault

The command to delete a key vault changes in behavior, depending on whether soft-delete is enabled.

IMPORTANT

If you run the following command for a key vault that does not have soft-delete enabled, you will permanently delete this key vault and all its content with no options for recovery!

```
az keyvault delete --name ContosoVault
```

How soft-delete protects your key vaults

With soft-delete enabled:

- A deleted key vault is removed from its resource group and placed in a reserved namespace, associated with the location where it was created.
- Deleted objects such as keys, secrets, and certificates, are inaccessible as long as their containing key vault is in the deleted state.
- The DNS name for a deleted key vault is reserved, preventing a new key vault with same name from being created.

You may view deleted state key vaults, associated with your subscription, using the following command:

```
az keyvault list-deleted
```

- *ID* can be used to identify the resource when recovering or purging.
- *Resource ID* is the original resource ID of this vault. Since this key vault is now in a deleted state, no resource exists with that resource ID.
- *Scheduled Purge Date* is when the vault will be permanently deleted, if no action is taken. The default retention period, used to calculate the *Scheduled Purge Date*, is 90 days.

Recovering a key vault

To recover a key vault, you specify the key vault name, resource group, and location. Note the location and the resource group of the deleted key vault, as you need them for the recovery process.

```
az keyvault recover --location westus --resource-group ContosoRG --name ContosoVault
```

When a key vault is recovered, a new resource is created with the key vault's original resource ID. If the original resource group is removed, one must be created with same name before attempting recovery.

Deleting and purging key vault objects

The following command will delete the 'ContosoFirstKey' key, in a key vault named 'ContosoVault', which has soft-delete enabled:

```
az keyvault key delete --name ContosoFirstKey --vault-name ContosoVault
```

With your key vault enabled for soft-delete, a deleted key still appears like it's deleted except, when you explicitly list or retrieve deleted keys. Most operations on a key in the deleted state will fail except for listing a deleted key, recovering it or purging it.

For example, to request to list deleted keys in a key vault, use the following command:

```
az keyvault key list-deleted --vault-name ContosoVault
```

Transition state

When you delete a key in a key vault with soft-delete enabled, it may take a few seconds for the transition to complete. During this transition, it may appear that the key isn't in the active state or the deleted state.

Using soft-delete with key vault objects

Just like key vaults, a deleted key, secret, or certificate, remains in deleted state for up to 90 days, unless you recover it or purge it.

Keys

To recover a soft-deleted key:

```
az keyvault key recover --name ContosoFirstKey --vault-name ContosoVault
```

To permanently delete (also known as purging) a soft-deleted key:

IMPORTANT

Purging a key will permanently delete it, and it will not be recoverable!

```
az keyvault key purge --name ContosoFirstKey --vault-name ContosoVault
```

The **recover** and **purge** actions have their own permissions associated in a key vault access policy. For a user or service principal to be able to execute a **recover** or **purge** action, they must have the respective permission for that key or secret. By default, **purge** isn't added to a key vault's access policy, when the 'all' shortcut is used to grant all permissions. You must specifically grant **purge** permission.

Set a key vault access policy

The following command grants user@contoso.com permission to use several operations on keys in *ContosoVault* including **purge**:

```
az keyvault set-policy --name ContosoVault --key-permissions get create delete list update import backup  
restore recover purge
```

NOTE

If you have an existing key vault that has just had soft-delete enabled, you may not have **recover** and **purge** permissions.

Secrets

Like keys, secrets are managed with their own commands:

- Delete a secret named SQLPassword:

```
az keyvault secret delete --vault-name ContosoVault -name SQLPassword
```

- List all deleted secrets in a key vault:

```
az keyvault secret list-deleted --vault-name ContosoVault
```

- Recover a secret in the deleted state:

```
az keyvault secret recover --name SQLPassword --vault-name ContosoVault
```

- Purge a secret in deleted state:

IMPORTANT

Purging a secret will permanently delete it, and it will not be recoverable!

```
az keyvault secret purge --name SQLPassword --vault-name ContosoVault
```

Purging a soft-delete protected key vault

IMPORTANT

Purging a key vault or one of its contained objects, will permanently delete it, meaning it will not be recoverable!

The purge function is used to permanently delete a key vault object or an entire key vault, that was previously soft-deleted. As demonstrated in the previous section, objects stored in a key vault with the soft-delete feature enabled, can go through multiple states:

- **Active**: before deletion.
- **Soft-Deleted**: after deletion, able to be listed and recovered back to active state.
- **Permanently-Deleted**: after purge, not able to be recovered.

The same is true for the key vault. In order to permanently delete a soft-deleted key vault and its contents, you must purge the key vault itself.

Purging a key vault

When a key vault is purged, its entire contents are permanently deleted, including keys, secrets, and certificates. To purge a soft-deleted key vault, use the `az keyvault purge` command. You can find the location your subscription's deleted key vaults using the command `az keyvault list-deleted`.

```
az keyvault purge --location westus --name ContosoVault
```

Purge permissions required

- To purge a deleted key vault, the user needs RBAC permission to the *Microsoft.KeyVault/locations/deletedVaults/purge/action* operation.
- To list a deleted key vault, the user needs RBAC permission to the *Microsoft.KeyVault/deletedVaults/read* operation.
- By default only a subscription administrator has these permissions.

Scheduled purge

Listing deleted key vault objects also shows when they're scheduled to be purged by Key Vault. *Scheduled Purge Date* indicates when a key vault object will be permanently deleted, if no action is taken. By default, the retention period for a deleted key vault object is 90 days.

IMPORTANT

A purged vault object, triggered by its *Scheduled Purge Date* field, is permanently deleted. It is not recoverable!

Enabling Purge Protection

When purge protection is turned on, a vault or an object in deleted state cannot be purged until the retention period of 90 days has passed. Such vault or object can still be recovered. This feature gives added assurance that a vault or an object can never be permanently deleted until the retention period has passed.

You can enable purge protection only if soft-delete is also enabled.

To turn on both soft delete and purge protection when creating a vault, use the [az keyvault create](#) command:

```
az keyvault create --name ContosoVault --resource-group ContosORG --location westus --enable-soft-delete true  
--enable-purge-protection true
```

To add purge protection to an existing vault (that already has soft delete enabled), use the [az keyvault update](#) command:

```
az keyvault update --name ContosoVault --resource-group ContosORG --enable-purge-protection true
```

Other resources

- For an overview of Key Vault's soft-delete feature, see [Azure Key Vault soft-delete overview](#).
- For a general overview of Azure Key Vault usage, see [What is Azure Key Vault?](#).

How to use Key Vault soft-delete with PowerShell

7 minutes to read • [Edit Online](#)

Azure Key Vault's soft delete feature allows recovery of deleted vaults and vault objects. Specifically, soft-delete addresses the following scenarios:

- Support for recoverable deletion of a key vault
- Support for recoverable deletion of key vault objects; keys, secrets, and certificates

Prerequisites

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

- Azure PowerShell 1.0.0 or later - If you don't have this already setup, install Azure PowerShell and associate it with your Azure subscription, see [How to install and configure Azure PowerShell](#).

NOTE

There is an outdated version of our Key Vault PowerShell output formatting file that **may** be loaded into your environment instead of the correct version. We are anticipating an updated version of PowerShell to contain the needed correction for the output formatting and will update this topic at that time. The current workaround, should you encounter this formatting problem, is:

- Use the following query if you notice you're not seeing the soft-delete enabled property described in this topic:

```
$vault = Get-AzKeyVault -VaultName myvault; $vault.EnableSoftDelete
```

For Key Vault specific reference information for PowerShell, see [Azure Key Vault PowerShell reference](#).

Required permissions

Key Vault operations are separately managed via role-based access control (RBAC) permissions as follows:

OPERATION	DESCRIPTION	USER PERMISSION
List	Lists deleted key vaults.	Microsoft.KeyVault/deletedVaults/read
Recover	Restores a deleted key vault.	Microsoft.KeyVault/vaults/write
Purge	Permanently removes a deleted key vault and all its contents.	Microsoft.KeyVault/locations/deletedVaults/purge/action

For more information on permissions and access control, see [Secure your key vault](#).

Enabling soft-delete

You enable "soft-delete" to allow recovery of a deleted key vault, or objects stored in a key vault.

IMPORTANT

Enabling 'soft delete' on a key vault is an irreversible action. Once the soft-delete property has been set to "true", it cannot be changed or removed.

Existing key vault

For an existing key vault named ContosoVault, enable soft-delete as follows.

```
($resource = Get-AzResource -ResourceId (Get-AzKeyVault -VaultName "ContosoVault").ResourceId).Properties |  
Add-Member -MemberType "NoteProperty" -Name "enableSoftDelete" -Value "true"  
  
Set-AzResource -resourceid $resource.ResourceId -Properties $resource.Properties
```

New key vault

Enabling soft-delete for a new key vault is done at creation time by adding the soft-delete enable flag to your create command.

```
New-AzKeyVault -Name "ContosoVault" -ResourceGroupName "ContosoRG" -Location "westus" -EnableSoftDelete
```

Verify soft-delete enablement

To verify that a key vault has soft-delete enabled, run the *show* command and look for the 'Soft Delete Enabled?' attribute:

```
Get-AzKeyVault -VaultName "ContosoVault"
```

Deleting a soft-delete protected key vault

The command to delete a key vault changes in behavior, depending on whether soft-delete is enabled.

IMPORTANT

If you run the following command for a key vault that does not have soft-delete enabled, you will permanently delete this key vault and all its content with no options for recovery!

```
Remove-AzKeyVault -VaultName 'ContosoVault'
```

How soft-delete protects your key vaults

With soft-delete enabled:

- A deleted key vault is removed from its resource group and placed in a reserved namespace, associated with the location where it was created.
- Deleted objects such as keys, secrets, and certificates, are inaccessible as long as their containing key vault is in the deleted state.
- The DNS name for a deleted key vault is reserved, preventing a new key vault with same name from being created.

You may view deleted state key vaults, associated with your subscription, using the following command:

```
Get-AzKeyVault -InRemovedState
```

- *ID* can be used to identify the resource when recovering or purging.
- *Resource ID* is the original resource ID of this vault. Since this key vault is now in a deleted state, no resource exists with that resource ID.
- *Scheduled Purge Date* is when the vault will be permanently deleted, if no action is taken. The default retention period, used to calculate the *Scheduled Purge Date*, is 90 days.

Recovering a key vault

To recover a key vault, you specify the key vault name, resource group, and location. Note the location and the resource group of the deleted key vault, as you need them for the recovery process.

```
Undo-AzKeyVaultRemoval -VaultName ContosoVault -ResourceGroupName ContosoRG -Location westus
```

When a key vault is recovered, a new resource is created with the key vault's original resource ID. If the original resource group is removed, one must be created with same name before attempting recovery.

Deleting and purging key vault objects

The following command will delete the 'ContosoFirstKey' key, in a key vault named 'ContosoVault', which has soft-delete enabled:

```
Remove-AzKeyVaultKey -VaultName ContosoVault -Name ContosoFirstKey
```

With your key vault enabled for soft-delete, a deleted key still appears to be deleted, unless you explicitly list deleted keys. Most operations on a key in the deleted state will fail, except for listing, recovering, purging a deleted key.

For example, the following command lists deleted keys in the 'ContosoVault' key vault:

```
Get-AzKeyVaultKey -VaultName ContosoVault -InRemovedState
```

Transition state

When you delete a key in a key vault with soft-delete enabled, it may take a few seconds for the transition to complete. During this transition, it may appear that the key is not in the active state or the deleted state.

Using soft-delete with key vault objects

Just like key vaults, a deleted key, secret, or certificate, remains in deleted state for up to 90 days, unless you recover it or purge it.

Keys

To recover a soft-deleted key:

```
Undo-AzKeyVaultKeyRemoval -VaultName ContosoVault -Name ContosoFirstKey
```

To permanently delete (also known as purging) a soft-deleted key:

IMPORTANT

Purging a key will permanently delete it, and it will not be recoverable!

```
Remove-AzKeyVaultKey -VaultName ContosoVault -Name ContosoFirstKey -InRemovedState
```

The **recover** and **purge** actions have their own permissions associated in a key vault access policy. For a user or service principal to be able to execute a **recover** or **purge** action, they must have the respective permission for that key or secret. By default, **purge** isn't added to a key vault's access policy, when the 'all' shortcut is used to grant all permissions. You must specifically grant **purge** permission.

Set a key vault access policy

The following command grants user@contoso.com permission to use several operations on keys in *ContosoVault* including **purge**:

```
Set-AzKeyVaultAccessPolicy -VaultName ContosoVault -UserPrincipalName user@contoso.com -PermissionsToKeys get,create,delete,list,update,import,backup,restore,recover,purge
```

NOTE

If you have an existing key vault that has just had soft-delete enabled, you may not have **recover** and **purge** permissions.

Secrets

Like keys, secrets are managed with their own commands:

- Delete a secret named SQLPassword:

```
Remove-AzKeyVaultSecret -VaultName ContosoVault -name SQLPassword
```

- List all deleted secrets in a key vault:

```
Get-AzKeyVaultSecret -VaultName ContosoVault -InRemovedState
```

- Recover a secret in the deleted state:

```
Undo-AzKeyVaultSecretRemoval -VaultName ContosoVault -Name SQLPassword
```

- Purge a secret in deleted state:

IMPORTANT

Purging a secret will permanently delete it, and it will not be recoverable!

```
Remove-AzKeyVaultSecret -VaultName ContosoVault -InRemovedState -name SQLPassword
```

Purging a soft-delete protected key vault

IMPORTANT

Purging a key vault or one of its contained objects, will permanently delete it, meaning it will not be recoverable!

The purge function is used to permanently delete a key vault object or an entire key vault, that was previously soft-deleted. As demonstrated in the previous section, objects stored in a key vault with the soft-delete feature enabled, can go through multiple states:

- **Active**: before deletion.
- **Soft-Deleted**: after deletion, able to be listed and recovered back to active state.
- **Permanently-Deleted**: after purge, not able to be recovered.

The same is true for the key vault. In order to permanently delete a soft-deleted key vault and its contents, you must purge the key vault itself.

Purging a key vault

When a key vault is purged, its entire contents are permanently deleted, including keys, secrets, and certificates. To purge a soft-deleted key vault, use the `Remove-AzKeyVault` command with the option `-InRemovedState` and by specifying the location of the deleted key vault with the `-Location location` argument. You can find the location of a deleted vault using the command `Get-AzKeyVault -InRemovedState`.

```
Remove-AzKeyVault -VaultName ContosoVault -InRemovedState -Location westus
```

Purge permissions required

- To purge a deleted key vault, the user needs RBAC permission to the `Microsoft.KeyVault/locations/deletedVaults/purge/action` operation.
- To list a deleted key vault, the user needs RBAC permission to the `Microsoft.KeyVault/deletedVaults/read` operation.
- By default only a subscription administrator has these permissions.

Scheduled purge

Listing deleted key vault objects also shows when they're scheduled to be purged by Key Vault. *Scheduled Purge Date* indicates when a key vault object will be permanently deleted, if no action is taken. By default, the retention period for a deleted key vault object is 90 days.

IMPORTANT

A purged vault object, triggered by its *Scheduled Purge Date* field, is permanently deleted. It is not recoverable!

Enabling Purge Protection

When purge protection is turned on, a vault or an object in deleted state cannot be purged until the retention period of 90 days has passed. Such vault or object can still be recovered. This feature gives added assurance that a vault or an object can never be permanently deleted until the retention period has passed.

You can enable purge protection only if soft-delete is also enabled.

To turn on both soft delete and purge protection when creating a vault, use the `New-AzKeyVault` cmdlet:

```
New-AzKeyVault -Name ContosoVault -ResourceGroupName ContosoRG -Location westus -EnableSoftDelete -EnablePurgeProtection
```

To add purge protection to an existing vault (that already has soft delete enabled), use the [Get-AzKeyVault](#), [Get-AzResource](#), and [Set-AzResource](#) cmdlets:

```
($resource = Get-AzResource -ResourceId (Get-AzKeyVault -VaultName "ContosoVault").ResourceId).Properties |  
Add-Member -MemberType "NoteProperty" -Name "enablePurgeProtection" -Value "true"  
  
Set-AzResource -resourceid $resource.ResourceId -Properties $resource.Properties
```

Other resources

- For an overview of Key Vault's soft-delete feature, see [Azure Key Vault soft-delete overview](#).
- For a general overview of Azure Key Vault usage, see [What is Azure Key Vault?](#).

Configure Azure Key Vault firewalls and virtual networks

3 minutes to read • [Edit Online](#)

This article provides step-by-step instructions to configure Azure Key Vault firewalls and virtual networks to restrict access to your key vault. The [virtual network service endpoints for Key Vault](#) allow you to restrict access to a specified virtual network and set of IPv4 (internet protocol version 4) address ranges.

IMPORTANT

After firewall rules are in effect, users can only perform Key Vault [data plane](#) operations when their requests originate from allowed virtual networks or IPv4 address ranges. This also applies to accessing Key Vault from the Azure portal. Although users can browse to a key vault from the Azure portal, they might not be able to list keys, secrets, or certificates if their client machine is not in the allowed list. This also affects the Key Vault Picker by other Azure services. Users might be able to see list of key vaults, but not list keys, if firewall rules prevent their client machine.

Use the Azure portal

Here's how to configure Key Vault firewalls and virtual networks by using the Azure portal:

1. Browse to the key vault you want to secure.
2. Select **Networking**, and then select the **Firewalls and virtual networks** tab.
3. Under **Allow access from**, select **Selected networks**.
4. To add existing virtual networks to firewalls and virtual network rules, select **+ Add existing virtual networks**.
5. In the new blade that opens, select the subscription, virtual networks, and subnets that you want to allow access to this key vault. If the virtual networks and subnets you select don't have service endpoints enabled, confirm that you want to enable service endpoints, and select **Enable**. It might take up to 15 minutes to take effect.
6. Under **IP Networks**, add IPv4 address ranges by typing IPv4 address ranges in [CIDR \(Classless Inter-domain Routing\) notation](#) or individual IP addresses.
7. Select **Save**.

You can also add new virtual networks and subnets, and then enable service endpoints for the newly created virtual networks and subnets, by selecting **+ Add new virtual network**. Then follow the prompts.

Use the Azure CLI

Here's how to configure Key Vault firewalls and virtual networks by using the Azure CLI

1. [Install Azure CLI](#) and [sign in](#).
2. List available virtual network rules. If you haven't set any rules for this key vault, the list will be empty.

```
az keyvault network-rule list --resource-group myresourcegroup --name mykeyvault
```

3. Enable a service endpoint for Key Vault on an existing virtual network and subnet.

```
az network vnet subnet update --resource-group "myresourcegroup" --vnet-name "myvnet" --name "mysubnet" --service-endpoints "Microsoft.KeyVault"
```

4. Add a network rule for a virtual network and subnet.

```
subnetid=$(az network vnet subnet show --resource-group "myresourcegroup" --vnet-name "myvnet" --name "mysubnet" --query id --output tsv)
az keyvault network-rule add --resource-group "demo9311" --name "demo9311premium" --subnet $subnetid
```

5. Add an IP address range from which to allow traffic.

```
az keyvault network-rule add --resource-group "myresourcegroup" --name "mykeyvault" --ip-address "191.10.18.0/24"
```

6. If this key vault should be accessible by any trusted services, set `bypass` to `AzureServices`.

```
az keyvault update --resource-group "myresourcegroup" --name "mykeyvault" --bypass AzureServices
```

7. Turn the network rules on by setting the default action to `Deny`.

```
az keyvault update --resource-group "myresourcegroup" --name "mekeyvault" --default-action Deny
```

Use Azure PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Here's how to configure Key Vault firewalls and virtual networks by using PowerShell:

1. Install the latest [Azure PowerShell](#), and [sign in](#).

2. List available virtual network rules. If you have not set any rules for this key vault, the list will be empty.

```
(Get-AzKeyVault -VaultName "mykeyvault").NetworkAcls
```

3. Enable service endpoint for Key Vault on an existing virtual network and subnet.

```
Get-AzVirtualNetwork -ResourceGroupName "myresourcegroup" -Name "myvnet" | Set-AzVirtualNetworkSubnetConfig -Name "mysubnet" -AddressPrefix "10.1.1.0/24" -ServiceEndpoint "Microsoft.KeyVault" | Set-AzVirtualNetwork
```

4. Add a network rule for a virtual network and subnet.

```
$subnet = Get-AzVirtualNetwork -ResourceGroupName "myresourcegroup" -Name "myvnet" | Get-AzVirtualNetworkSubnetConfig -Name "mysubnet"
Add-AzKeyVaultNetworkRule -VaultName "mykeyvault" -VirtualNetworkResourceId $subnet.Id
```

5. Add an IP address range from which to allow traffic.

```
Add-AzKeyVaultNetworkRule -VaultName "mykeyvault" -IpAddressRange "16.17.18.0/24"
```

6. If this key vault should be accessible by any trusted services, set `bypass` to `AzureServices`.

```
Update-AzKeyVaultNetworkRuleSet -VaultName "mykeyvault" -Bypass AzureServices
```

7. Turn the network rules on by setting the default action to `Deny`.

```
Update-AzKeyVaultNetworkRuleSet -VaultName "mykeyvault" -DefaultAction Deny
```

References

- Azure CLI commands: [az keyvault network-rule](#)
- Azure PowerShell cmdlets: [Get-AzKeyVault](#), [Add-AzKeyVaultNetworkRule](#), [Remove-AzKeyVaultNetworkRule](#), [Update-AzKeyVaultNetworkRuleSet](#)

Next steps

- [Virtual network service endpoints for Key Vault](#)
- [Secure your key vault](#)

Azure Key Vault REST API Error Codes

5 minutes to read • [Edit Online](#)

The following error codes could be returned by an operation on an Azure Key Vault web service.

HTTP 401: Unauthenticated Request

401 means that the request is unauthenticated for Key Vault.

A request is authenticated if:

- The key vault knows the identity of the caller; and
- The caller is allowed to try to access Key Vault resources.

There are several different reason why a request may return 401.

No authentication token attached to the request.

Here is an example PUT request, setting the value of a secret:

```
PUT https://putreqexample.vault.azure.net/secrets/DatabaseRotatingPassword?api-version=7.0 HTTP/1.1
x-ms-client-request-id: 03d275a2-52a4-4bed-82c8-6fe15165affb
accept-language: en-US
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Im5iQ3dXMTF3M1hrQi14VWFYd0tSU0xqTUhHUSisImtpZCI6Im5iQ3dXMTF3M1hrQi14VWFYd0tSU0xqTUhHUSj9eyJhdWQiOiJodHRwczovL3ZhdWx0LmF6dXJ1Lm5ldCIsImlzcyI6Imh0dBzOi8vc3RzLnDpbmRvd3MubmV0LzcyZjk40GJmLTg2ZjEtNDFhZi05MWFiltJkn2NkMDExZGI0Ny8iLCJpYXQi0jE1NDg20Tc1MTMsIm5iZiI6MTU0ODV5NzUxMywiZXhwIjoxNTQ4NzAxNDEzLCJhaw8i0i0MkpnuWohoDVqaVBnZHF5ZlRGZE5TdHY3bGUvQkFBPSIsImFwcGlkIjoiZmFkN2Q1YjMtNjlkNi00YjQ4LTkyNTktOGQxMjEyNGUXY2YxIwiYXBwaWRhY3Ii0iIxIiwiawRwIjoiHR0cHM6Ly9zdHMud2luZG93cy5uZXQvNzJmOTg4YmYtODZmMS00MWfmlTkxYWItMmQ3Y2QwMTFkYjQ3LyIsIm9pZCI6IjM5NzVhZWVkLTdkMDgtNDUzYi1iNmY0LTQ0NWYzMjY50DA5MSIsInN1Yi6IjM5NzVhZWVkLTdkMDgtNDUzYi1iNmY0LTQ0NWYzMjY50DA5MSIsInRpZCI6IjcyZjk40GJmLTg2ZjEtNDFhZi05MWFiltJkn2NkMDExZGI0NyIsInV0aSI6IjItZ3JoUmt1SwS2QmVZLUxuNDJtQUEiLCJ2ZXi0iIxLjAifQ.fgubiz1MKqTJTXI8dHIV7t9Fle6FdHrkaGYKcBeVRX1WtLVuk1QVxzIFD1ZKLXJ7QPNS0KwpeIwQI9IWIRK-8w038yCqKTfD1fHOiNWGOpkKdd1G729KFqakVf2w0GPyGPFCONRDAR5wjQarN9Bt8I8YbHwZQz_M1hztlnv-Lmsk1jBmech9ujD9-1TMBmSffBHCqqquev119V7sneI-
zxBZLf8C0pIDkaXf1t8y6Xr8CUJDm1Ls1cf3pBCNIOy65_TyGvy4Z4AJryTPBarNBPwOkNAtjCfZ4BDc2KqUZM5QN_VK4foP64sVzUL6mSr0Gh7lQJIL5b1qIpJxjxyQ
User-Agent: FxVersion/4.7.3324.0 OSName/Windows OSVersion/6.2.9200.0
Microsoft.Azure.KeyVault.KeyVaultClient/3.0.3.0
Content-Type: application/json; charset=utf-8
Host: putreqexample.vault.azure.net
Content-Length: 31

{
  "value": "m*gBJ7$Zuoz"
}
```

The "Authorization" header is the access token that is required with every call to the Key Vault for data-plane operations. If the header is missing, then the response must be 401.

The token lacks the correct resource associated with it.

When requesting an access token from the Azure OAUTH endpoint, a parameter called "resource" is mandatory. The value is important for the token provider because it scopes the token for its intended use. The resource for **all** tokens to access a Key Vault is <https://vault.keyvault.net> (with no trailing slash).

The token is expired

Tokens are base64 encoded and the values can be decoded at websites such as <http://jwt.calebb.net>. Here is the above token decoded:

```

{
  typ: "JWT",
  alg: "RS256",
  x5t: "nbCwW11w3XkB-xUaXwKRSLjMHGQ",
  kid: "nbCwW11w3XkB-xUaXwKRSLjMHGQ"
}.
{
  aud: "https://vault.azure.net",
  iss: "https://sts.windows.net/72f988bf-86f1-41af-91ab-2d7cd011db47/",
  iat: 1548697513,
  nbf: 1548697513,
  exp: 1548701413,
  aio: "42JgYHh85jiPgdqyfTFdNSTv7le/BAA=",
  appid: "fad7d5b3-69d6-4b48-9259-8d12124e1cf1",
  appidacr: "1",
  idp: "https://sts.windows.net/72f988bf-86f1-41af-91ab-2d7cd011db47/",
  oid: "3975aeed-7d08-453b-b6f4-445f32698091",
  sub: "3975aeed-7d08-453b-b6f4-445f32698091",
  tid: "72f988bf-86f1-41af-91ab-2d7cd011db47",
  uti: "2-grhRkeIk6BeY-Ln42mAA",
  ver: "1.0"
}.
[signature]

```

We can see many important parts in this token:

- aud (audience): The resource of the token. Notice that this is <https://vault.azure.net>. This token will NOT work for any resource that does not explicitly match this value, such as graph.
- iat (issued at): The number of ticks since the start of the epoch when the token was issued.
- nbf (not before): The number of ticks since the start of the epoch when this token becomes valid.
- exp (expiration): The number of ticks since the start of the epoch when this token expires.
- appid (application ID): The GUID for the application ID making this request.
- tid (tenant ID): The GUID for the tenant ID of the principal making this request

It is important that all of the values be properly identified in the token in order for the request to work. If everything is correct, then the request will not result in 401.

Troubleshooting 401

401s should be investigated from the point of token generation, before the request is made to the key vault. Generally code is being used to request the token. Once the token is received, it is passed into the Key Vault request. If the code is running locally, you can use Fiddler to capture the request/response to

<https://login.microsoftonline.com>. A request looks like this:

```

POST https://login.microsoftonline.com/<key vault tenant ID>/oauth2/token HTTP/1.1
Accept: application/json
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: login.microsoftonline.com
Content-Length: 192

resource=https%3A%2F%2Fvault.azure.net&client_id=<registered-app-ID>&client_secret=<registered-app-secret>&client_info=1&grant_type=client_credentials

```

The following user-supplied information must be correct:

- The key vault tenant ID
- Resource value set to https%3A%2F%2Fvault.azure.net (URL encoded)
- Client ID
- Client secret

Ensure the rest of the request is nearly identical.

If you can only get the response access token, you can decode it (as shown above) to ensure the tenant ID, the client ID (app ID), and the resource.

HTTP 403: Insufficient Permissions

HTTP 403 means that the request was authenticated (it knows the requesting identity) but the identity does not have permission to access the requested resource. There are two causes:

- There is no access policy for the identity.
- The IP address of the requesting resource is not whitelisted in the key vault's firewall settings.

HTTP 403 often occurs when the customer's application is not using the client ID that the customer thinks it is. That usually means that the access policies is not correctly set up for the actual calling identity.

Troubleshooting 403

First, turn on logging. For instructions on how to do so, see [Azure Key Vault logging](#).

Once logging is turned on, you can determine if the 403 is due to access policy or firewall policy.

Error due to firewall policy

"Client address (00.00.00.00) is not authorized and caller is not a trusted service"

There is a limited list of "Azure Trusted Services". Azure Web Sites are **not** a Trusted Azure Service. For more information, see the blog post [Key Vault Firewall access by Azure App Services](#).

You must add the IP address of the Azure Web Site to the Key Vault in order for it to work.

If due to access policy: find the object ID for the request and ensure that the object ID matches the object to which the user is trying to assign the access policy. There will often be multiple objects in the AAD which have the same name, so choosing the correct one is very important. By deleting and re-adding the access policy, it is possible to see if multiple objects exist with the same name.

In addition, most access policies do not require the use of the "Authorized application" as shown in the portal. Authorized application are used for "on-behalf-of" authentication scenarios, which are rare.

HTTP 429: Too Many Requests

Throttling occurs when the number of requests exceeds the stated maximum for the timeframe. If throttling occurs, the Key Vault's response will be HTTP 429. There are stated maximums for types of requests made. For instance: the creation of an HSM 2048-bit key is 5 requests per 10 seconds, but all other HSM transactions have a 1000 request/10 seconds limit. Therefore it is important to understand which types of calls are being made when determining the cause of throttling. In general, requests to the Key Vault are limited to 2000 requests/10 seconds. Exceptions are Key Operations, as documented in [Key Vault service limits](#)

Troubleshooting 429

Throttling is worked around using these techniques:

- Reduce number of requests made to the Key Vault by determining if there are patterns to a requested resource and attempting to cache them in the calling application.
- When Key Vault throttling occurs, adapt the requesting code to use a exponential backoff for retrying. The algorithm is explained here: [How to throttle your app](#)
- If the number of requests cannot be reduced by caching and timed backoff does not work, then consider splitting the keys up into multiple Key Vaults. The service limit for a single subscription is 5x the individual Key Vault limit. If using more than 5 Key Vaults, consideration should be given to using multiple

subscriptions.

Detailed guidance including request to increase limits, can be find here: [Key Vault throttling guidance](#)

Best practices to use Key Vault

2 minutes to read • [Edit Online](#)

Control Access to your vault

Azure Key Vault is a cloud service that safeguards encryption keys and secrets like certificates, connection strings, and passwords. Because this data is sensitive and business critical, you need to secure access to your key vaults by allowing only authorized applications and users. This [article](#) provides an overview of the Key Vault access model. It explains authentication and authorization, and describes how to secure access to your key vaults.

Suggestions while controlling access to your vault are as follows:

1. Lock down access to your subscription, resource group and Key Vaults (RBAC)
2. Create Access policies for every vault
3. Use least privilege access principal to grant access
4. Turn on Firewall and [VNET Service Endpoints](#)

Use separate Key Vault

Our recommendation is to use a vault per application per environment (Development, Pre-Production and Production). This helps you not share secrets across environments and also reduces the threat in case of a breach.

Backup

Make sure you take regular back ups of your [vault](#) on update/delete/create of objects within a Vault.

Turn on Logging

[Turn on logging](#) for your Vault. Also set up alerts.

Turn on recovery options

1. Turn on [Soft Delete](#).
2. Turn on purge protection if you want to guard against force deletion of the secret / vault even after soft delete is turned on.

Authentication, requests and responses

3 minutes to read • [Edit Online](#)

Azure Key Vault supports JSON formatted requests and responses. Requests to the Azure Key Vault are directed to a valid Azure Key Vault URL using HTTPS with some URL parameters and JSON encoded request and response bodies.

This topic covers specifics for the Azure Key Vault service. For general information on using Azure REST interfaces, including authentication/authorization and how to acquire an access token, see [Azure REST API Reference](#).

Request URL

Key management operations use HTTP DELETE, GET, PATCH, PUT and HTTP POST and cryptographic operations against existing key objects use HTTP POST. Clients that cannot support specific HTTP verbs may also use HTTP POST using the X-HTTP-REQUEST header to specify the intended verb; requests that do not normally require a body should include an empty body when using HTTP POST, for example when using POST instead of DELETE.

To work with objects in the Azure Key Vault, the following are example URLs:

- To CREATE a key called TESTKEY in a Key Vault use - `PUT /keys/TESTKEY?api-version=<api_version> HTTP/1.1`
- To IMPORT a key called IMPORTEDKEY into a Key Vault use -
`POST /keys/IMPORTEDKEY/import?api-version=<api_version> HTTP/1.1`
- To GET a secret called MYSECRET in a Key Vault use -
`GET /secrets/MYSECRET?api-version=<api_version> HTTP/1.1`
- To SIGN a digest using a key called TESTKEY in a Key Vault use -
`POST /keys/TESTKEY/sign?api-version=<api_version> HTTP/1.1`

The authority for a request to a Key Vault is always as follows, `https://<keyvault-name>.vault.azure.net/`

Keys are always stored under the /keys path, Secrets are always stored under the /secrets path.

API Version

The Azure Key Vault Service supports protocol versioning to provide compatibility with down-level clients, although not all capabilities will be available to those clients. Clients must use the `api-version` query string parameter to specify the version of the protocol that they support as there is no default.

Azure Key Vault protocol versions follow a date numbering scheme using a {YYYY}.{MM}.{DD} format.

Request Body

As per the HTTP specification, GET operations must NOT have a request body, and POST and PUT operations must have a request body. The body in DELETE operations is optional in HTTP.

Unless otherwise noted in operation description, the request body content type must be application/json and must contain a serialized JSON object conformant to content type.

Unless otherwise noted in operation description, the Accept request header must contain the application/json media type.

Response Body

Unless otherwise noted in operation description, the response body content type of both successful and failed operations will be application/json and contains detailed error information.

Using HTTP POST

Some clients may not be able to use certain HTTP verbs, such as PATCH or DELETE. Azure Key Vault supports HTTP POST as an alternative for these clients provided that the client also includes the "X-HTTP-METHOD" header to specific the original HTTP verb. Support for HTTP POST is noted for each of the API defined in this document.

Error Responses

Error handling will use HTTP status codes. Typical results are:

- 2xx – Success: Used for normal operation. The response body will contain the expected result
- 3xx – Redirection: The 304 "Not Modified" may be returned to fulfill a conditional GET. Other 3xx codes may be used in the future to indicate DNS and path changes.
- 4xx – Client Error: Used for bad requests, missing keys, syntax errors, invalid parameters, authentication errors, etc. The response body will contain detailed error explanation.
- 5xx – Server Error: Used for internal server errors. The response body will contain summarized error information.

The system is designed to work behind a proxy or firewall. Therefore, a client might receive other error codes.

Azure Key Vault also returns error information in the response body when a problem occurs. The response body is JSON formatted and takes the form:

```
{  
  "error":  
  {  
    "code": "BadArgument",  
    "message":  
      "'Foo' is not a valid argument for 'type'."  
  }  
}
```

Authentication

All requests to Azure Key Vault MUST be authenticated. Azure Key Vault supports Azure Active Directory access tokens that may be obtained using OAuth2 [[RFC6749](#)].

For more information on registering your application and authenticating to use Azure Key Vault, see [Register your client application with Azure AD](#).

Access tokens must be sent to the service using the HTTP Authorization header:

```
PUT /keys/MYKEY?api-version=<api_version> HTTP/1.1
Authorization: Bearer <access_token>
```

When an access token is not supplied, or when a token is not accepted by the service, an HTTP 401 error will be returned to the client and will include the WWW-Authenticate header, for example:

```
401 Not Authorized
WWW-Authenticate: Bearer authorization="...", resource="..."
```

The parameters on the WWW-Authenticate header are:

- **authorization:** The address of the OAuth2 authorization service that may be used to obtain an access token for the request.
- **resource:** The name of the resource (<https://vault.azure.net>) to use in the authorization request.

See Also

[About keys, secrets, and certificates](#)

Common parameters and headers

2 minutes to read • [Edit Online](#)

The following information is common to all operations that you might do related to Key Vault resources:

- Replace `{api-version}` with the api-version in the URI.
- Replace `{subscription-id}` with your subscription identifier in the URI
- Replace `{resource-group-name}` with the resource group. For more information, see [Using Resource groups to manage your Azure resources](#).
- Replace `{vault-name}` with your key vault name in the URI.
- Set the Content-Type header to application/json.
- Set the Authorization header to a JSON Web Token that you obtain from Azure Active Directory (AAD). For more information, see [Authenticating Azure Resource Manager requests](#).

Common error response

The service will use HTTP status codes to indicate success or failure. In addition, failures contain a response in the following format:

```
{
  "error": {
    "code": "BadRequest",
    "message": "The key vault sku is invalid."
  }
}
```

ELEMENT NAME	TYPE	DESCRIPTION
code	string	The type of error that occurred.
message	string	A description of what caused the error.

See Also

[Azure Key Vault REST API Reference](#)

Azure Key Vault Developer's Guide

6 minutes to read • [Edit Online](#)

Key Vault allows you to securely access sensitive information from within your applications:

- Keys and secrets are protected without having to write the code yourself and you are easily able to use them from your applications.
- You are able to have your customers own and manage their own keys so you can concentrate on providing the core software features. In this way, your applications will not own the responsibility or potential liability for your customers' tenant keys and secrets.
- Your application can use keys for signing and encryption yet keeps the key management external from your application, allowing your solution to be suitable as a geographically distributed app.
- Manage Key Vault certificates. For more information, see [About keys, secrets, and certificates](#).

For more general information on Azure Key Vault, see [What is Key Vault](#).

Public Previews

Periodically, we release a public preview of a new Key Vault feature. Try out these and let us know what you think via azurekeyvault@microsoft.com, our feedback email address.

Creating and Managing Key Vaults

Azure Key Vault provides a way to securely store credentials and other keys and secrets, but your code needs to authenticate to Key Vault to retrieve them. Managed identities for Azure resources makes solving this problem simpler by giving Azure services an automatically managed identity in Azure Active Directory (Azure AD). You can use this identity to authenticate to any service that supports Azure AD authentication, including Key Vault, without having any credentials in your code.

For more information on managed identities for Azure resources, see [the managed identities overview](#). For more information on working with AAD, see [Integrating applications with Azure Active Directory](#).

Before working with keys, secrets or certificates in your key vault, you'll create and manage your key vault through CLI, PowerShell, Resource Manager Templates or REST, as described in the following articles:

- [Create and manage Key Vaults with CLI](#)
- [Create and manage Key Vaults with PowerShell](#)
- [Create and manage Key Vaults with the Azure port](#)
- [Create and manage Key Vaults with Python](#)
- [Create and manage Key Vaults with Java](#)
- [Create and manage Key Vaults with Nodejs](#)
- [Create and manage Key Vaults with .NET \(v4 SDK\)](#)
- [Create a key vault and add a secret via an Azure Resource Manager template](#)
- [Create and manage Key Vaults with REST](#)

Coding with Key Vault

The Key Vault management system for programmers consists of several interfaces. This section contains links to all of the languages as well as some code examples.

Supported programming and scripting languages

REST

All of your Key Vault resources are accessible through the REST interface; vaults, keys, secrets, etc.

[Key Vault REST API Reference](#).

.NET

[.NET API reference for Key Vault](#).

For more information on the 2.x version of the .NET SDK, see the [Release notes](#).

Java

[Java SDK for Key Vault](#)

Node.js

In Node.js, the Key Vault management API and the Key Vault object API are separate. The following overview article gives you access to both.

[Azure Key Vault modules for Nodejs](#)

Python

[Azure Key Vault libraries for Python](#)

Azure CLI

[Azure CLI for Key Vault](#)

Azure PowerShell

[Azure PowerShell for Key Vault](#)

Code examples

For complete examples using Key Vault with your applications, see:

- [Azure Key Vault code samples](#) - Code Samples for Azure Key Vault.
- [Use Azure Key Vault from a Web Application](#) - tutorial to help you learn how to use Azure Key Vault from a web application in Azure.

How-tos

The following articles and scenarios provide task-specific guidance for working with Azure Key Vault:

- [Change key vault tenant ID after subscription move](#) - When you move your Azure subscription from tenant A to tenant B, your existing key vaults are inaccessible by the principals (users and applications) in tenant B. Fix this using this guide.
- [Accessing Key Vault behind firewall](#) - To access a key vault your key vault client application needs to be able to access multiple end-points for various functionalities.
- [How to Generate and Transfer HSM-Protected Keys for Azure Key Vault](#) - This will help you plan for, generate and then transfer your own HSM-protected keys to use with Azure Key Vault.
- [How to pass secure values \(such as passwords\) during deployment](#) - When you need to pass a secure value (like a password) as a parameter during deployment, you can store that value as a secret in an Azure Key Vault and reference the value in other Resource Manager templates.
- [How to use Key Vault for extensible key management with SQL Server](#) - The SQL Server Connector for Azure Key Vault enables SQL Server and SQL-in-a-VM to leverage the Azure Key Vault service as an Extensible Key Management (EKM) provider to protect its encryption keys for applications link; Transparent Data Encryption, Backup Encryption, and Column Level Encryption.
- [How to deploy Certificates to VMs from Key Vault](#) - A cloud application running in a VM on Azure needs a certificate. How do you get this certificate into this VM today?
- [How to set up Key Vault with end to end key rotation and auditing](#) - This walks through how to set up key

- rotation and auditing with Azure Key Vault.
- [Deploying Azure Web App Certificate through Key Vault](#) provides step-by-step instructions for deploying certificates stored in Key Vault as part of [App Service Certificate](#) offering.
- [Grant permission to many applications to access a key vault](#) Key Vault access control policy supports up to 1024 entries. However you can create an Azure Active Directory security group. Add all the associated service principals to this security group and then grant access to this security group to Key Vault.
- For more task-specific guidance on integrating and using Key Vaults with Azure, see [Ryan Jones' Azure Resource Manager template examples for Key Vault](#).
- [How to use Key Vault soft-delete with CLI](#) guides you through the use and lifecycle of a key vault and various key vault objects with soft-delete enabled.
- [How to use Key Vault soft-delete with PowerShell](#) guides you through the use and lifecycle of a key vault and various key vault objects with soft-delete enabled.

Integrated with Key Vault

These articles are about other scenarios and services that use or integrate with Key Vault.

- [Azure Disk Encryption](#) leverages the industry standard [BitLocker](#) feature of Windows and the [DM-Crypt](#) feature of Linux to provide volume encryption for the OS and the data disks. The solution is integrated with Azure Key Vault to help you control and manage the disk encryption keys and secrets in your key vault subscription, while ensuring that all data in the virtual machine disks are encrypted at rest in your Azure storage.
- [Azure Data Lake Store](#) provides option for encryption of data that is stored in the account. For key management, Data Lake Store provides two modes for managing your master encryption keys (MEKs), which are required for decrypting any data that is stored in the Data Lake Store. You can either let Data Lake Store manage the MEKs for you, or choose to retain ownership of the MEKs using your Azure Key Vault account. You specify the mode of key management while creating a Data Lake Store account.
- [Azure Information Protection](#) allows you to manager your own tenant key. For example, instead of Microsoft managing your tenant key (the default), you can manage your own tenant key to comply with specific regulations that apply to your organization. Managing your own tenant key is also referred to as bring your own key, or BYOK.

Key Vault overviews and concepts

- [Key Vault soft-delete behavior](#) describes a feature that allows recovery of deleted objects, whether the deletion was accidental or intentional.
- [Key Vault client throttling](#) orients you to the basic concepts of throttling and offers an approach for your app.
- [Key Vault storage account keys overview](#) describes the Key Vault integration Azure Storage Accounts keys.
- [Key Vault security worlds](#) describes the relationships between regions and security areas.

Social

- [Key Vault Blog](#)
- [Key Vault Forum](#)

Supporting Libraries

- [Microsoft Azure Key Vault Core Library](#) provides **IKey** and **IKeyResolver** interfaces for locating keys from identifiers and performing operations with keys.
- [Microsoft Azure Key Vault Extensions](#) provides extended capabilities for Azure Key Vault.

Azure Key Vault .NET 2.0 - Release Notes and Migration Guide

2 minutes to read • [Edit Online](#)

The following information helps migrating to the 2.0 version of the Azure Key Vault library for C# and .NET. Apps written for earlier versions need to be updating to support the latest version. These changes are needed to fully support new and improved features, such as **Key Vault certificates**.

Key Vault certificates

Key Vault certificates manage x509 certificates and supports the following behaviors:

- Create certificates through a Key Vault creation process or import existing certificate. This includes both self-signed and Certificate Authority (CA) generated certificates.
- Securely store and manage x509 certificate storage without interaction using private key material.
- Define policies that direct Key Vault to manage the certificate lifecycle.
- Provide contact information for lifecycle events, such as expiration warnings and renewal notifications.
- Automatically renew certificates with selected issuers (Key Vault partner X509 certificate providers and certificate authorities).* Support certificate from alternate (non-partner) provides and certificate authorities (does not support auto-renewal).

.NET support

- **.NET 4.0** is not supported by the 2.0 version of the Azure Key Vault .NET library
- **.NET Framework 4.5.2** is supported by the 2.0 version of the Azure Key Vault .NET library
- **.NET Standard 1.4** is supported by the 2.0 version of the Azure Key Vault .NET library

Namespaces

- The namespace for **models** is changed from **Microsoft.Azure.KeyVault** to **Microsoft.Azure.KeyVault.Models**.
- The **Microsoft.Azure.KeyVault.Internal** namespace is dropped.
- The following Azure SDK dependencies namespaces have
 - **Hyak.Common** is now **Microsoft.Rest**.
 - **Hyak.Common.Internals** is now **Microsoft.Rest.Serialization**.

Type changes

- *Secret* changed to *SecretBundle*
- *Dictionary* changed to *IDictionary*
- *List<T>, string []* changed to *IList<T>*
- *NextList* changed to *NextPageLink*

Return types

- **KeyList** and **SecretList** now returns *IPage<T>* instead of *ListKeysResponseMessage*

- The generated **BackupKeyAsync** now returns *BackupKeyResult*, which contains *Value* (back-up blob). Previously, the method was wrapped and returned just the value.

Exceptions

- KeyVaultClientException* is changed to *KeyVaultErrorException*
- The service error changed from *exception.Error* to *exception.Body.ErrorMessage*.
- Removed additional info from the error message for **[JsonExtensionData]**.

Constructors

- Instead of accepting an *HttpClient* as a constructor argument, the constructor only accepts *HttpClientHandler* or *DelegatingHandler[]*.

Downloaded packages

When a client processes a Key Vault dependency, the following packages are downloaded:

Previous package list

- package id="Hyak.Common" version="1.0.2" targetFramework="net45"
- package id="Microsoft.Azure.Common" version="2.0.4" targetFramework="net45"
- package id="Microsoft.Azure.Common.Dependencies" version="1.0.0" targetFramework="net45"
- package id="Microsoft.Azure.KeyVault" version="1.0.0" targetFramework="net45"
- package id="Microsoft.Bcl" version="1.1.9" targetFramework="net45"
- package id="Microsoft.Bcl.Async" version="1.0.168" targetFramework="net45"
- package id="Microsoft.Bcl.Build" version="1.0.14" targetFramework="net45"
- package id="Microsoft.Net.Http" version="2.2.22" targetFramework="net45"

Current package list

- package id="Microsoft.Azure.KeyVault" version="2.0.0-preview" targetFramework="net45"
- package id="Microsoft.Rest.ClientRuntime" version="2.2.0" targetFramework="net45"
- package id="Microsoft.Rest.ClientRuntime.Azure" version="3.2.0" targetFramework="net45"

Class changes

- UnixEpoch** class has been removed.
- Base64UrlConverter** class is renamed to **Base64UrlJsonConverter**.

Other changes

- Support for the configuration of KV operation retry policy on transient failures has been added to this version of the API.

Microsoft.Azure.Management.KeyVault NuGet

- For the operations that returned a *vault*, the return type was a class that contained a **Vault** property. The return type is now *Vault*.
- PermissionsToKeys* and *PermissionsToSecrets* are now *Permissions.Keys* and *Permissions.Secrets*
- Certain return types changes apply to the control-plane as well.

Microsoft.Azure.KeyVault.Extensions NuGet

- The package is broken up to **Microsoft.Azure.KeyVault.Extensions** and **Microsoft.Azure.KeyVault.Cryptography** for the cryptography operations.

Securely save secret application settings for a web application

4 minutes to read • [Edit Online](#)

Overview

This article describes how to securely save secret application configuration settings for Azure applications.

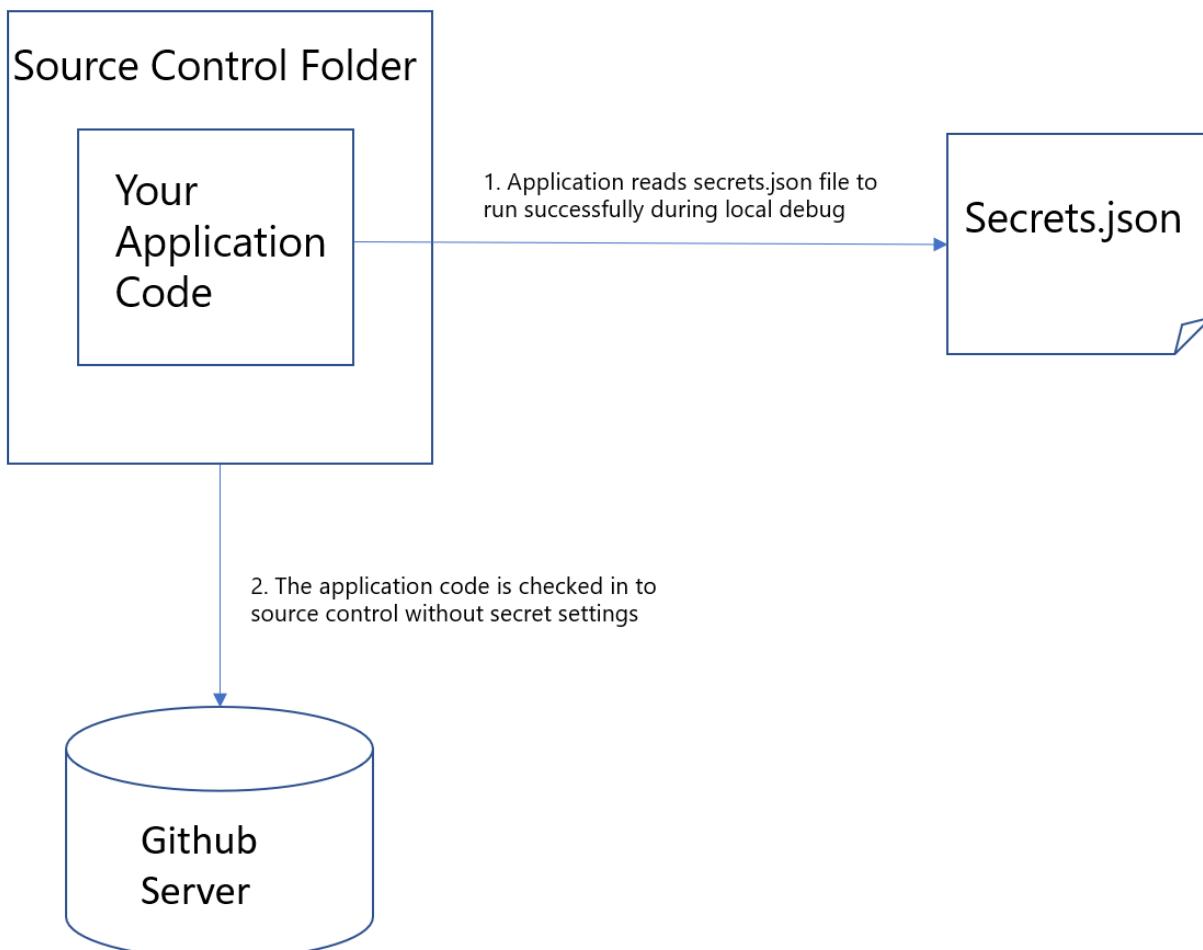
Traditionally all web application configuration settings are saved in configuration files such as Web.config. This practice leads to checking in secret settings such as Cloud credentials to public source control systems like GitHub. Meanwhile, it could be hard to follow security best practice because of the overhead required to change source code and reconfigure development settings.

To make sure the development process is secure, tooling and framework libraries are created to save application secret settings securely with minimal or no source code change.

ASP.NET and .NET Core applications

Save secret settings in User Secret store that is outside of source control folder

If you are doing a quick prototype or you don't have internet access, start with moving your secret settings outside of source control folder to User Secret store. User Secret store is a file saved under user profiler folder, so secrets are not checked in to source control. The following diagram demonstrates how [User Secret](#) works.

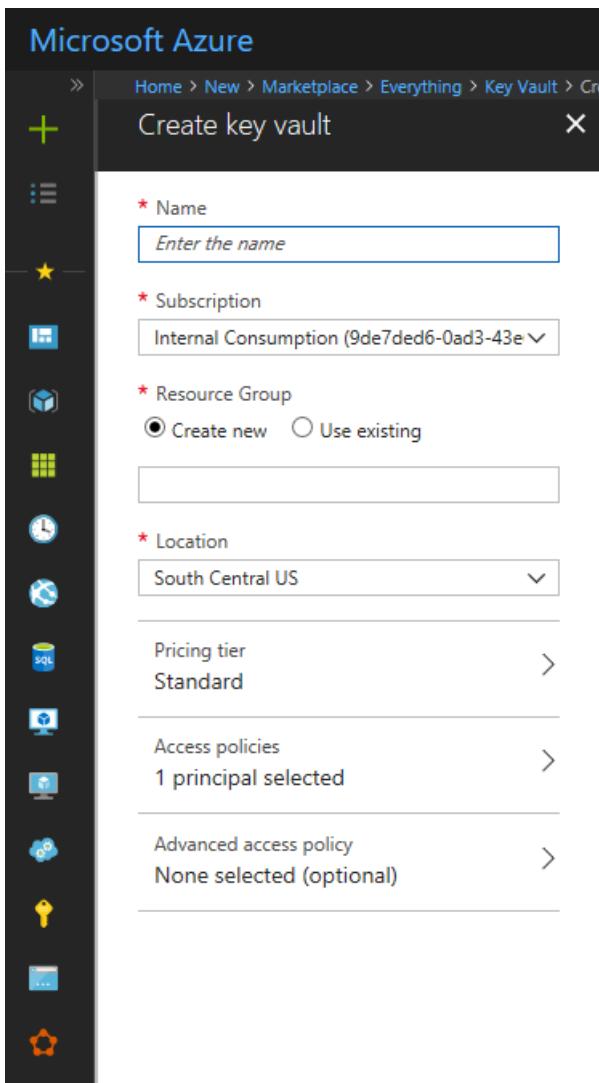


If you are running .NET core console application, use Key Vault to save your secret securely.

Save secret settings in Azure Key Vault

If you are developing a project and need to share source code securely, use [Azure Key Vault](#).

1. Create a Key Vault in your Azure subscription. Fill out all required fields on the UI and click *Create* on the bottom of the blade



2. Grant you and your team members access to the Key Vault. If you have a large team, you can create an [Azure Active Directory group](#) and add that security group access to the Key Vault. In the *Secret Permissions* dropdown, check *Get* and *List* under *Secret Management Operations*. If you already have your web app created, grant the web app access to the Key Vault so it can access the key vault without storing secret configuration in App Settings or files. Search for your web app by its name and add it the same way you grant users access.

The screenshot shows the 'Access policies' section of the Azure Key Vault 'Contoso-KeyVault'. On the left, a sidebar lists various management options like Overview, Activity log, and Settings. Under Settings, 'Access policies' is selected and highlighted in blue. The main pane displays a list of access policies for the 'Contoso-WebApp' application. Two entries are shown: one for a user named 'Contoso-WebApp' with the role 'USER', and another for a user with a blurred name also with the role 'USER'. A large '+' button at the top right is labeled 'Add new'.

3. Add your secret to Key Vault on the Azure portal. For nested configuration settings, replace ':' with '--' so the Key Vault secret name is valid. ':' is not allowed to be in the name of a Key Vault secret.

The screenshot shows the 'Secrets' section of the Azure Key Vault 'Contoso-KeyVault'. The sidebar shows 'Secrets' is selected. The main pane lists secrets under two categories: 'UNMANAGED' and 'MANAGED'. In the 'UNMANAGED' section, there is one secret named 'ConnectionString--DefaultConnection'. In the 'MANAGED' section, it says 'There are no secrets available.'

NOTE

Prior to Visual Studio 2017 V15.6 we used to recommend installing the Azure Services Authentication extension for Visual Studio. But it is deprecated now as the functionality is integrated within the Visual Studio . Hence if you are on an older version of visual Studio 2017 , we suggest you to update to at least VS 2017 15.6 or up so that you can use this functionality natively and access the Key-vault from using the Visual Studio sign-in Identity itself.

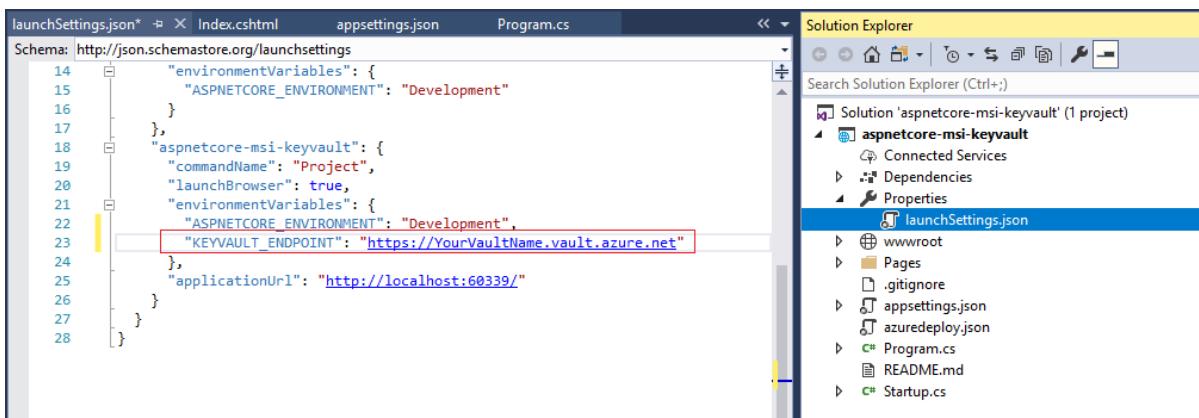
4. Add the following NuGet packages to your project:

```
Microsoft.Azure.KeyVault  
Microsoft.Azure.Services.AppAuthentication  
Microsoft.Extensions.Configuration.AzureKeyVault
```

5. Add the following code to Program.cs file:

```
public static IHostBuilder CreateHostBuilder(string[] args) =>  
    Host.CreateDefaultBuilder(args)  
        .ConfigureAppConfiguration((ctx, builder) =>  
        {  
            var keyVaultEndpoint = GetKeyVaultEndpoint();  
            if (!string.IsNullOrEmpty(keyVaultEndpoint))  
            {  
                var azureServiceTokenProvider = new AzureServiceTokenProvider();  
                var keyVaultClient = new KeyVaultClient(  
                    new KeyVaultClient.AuthenticationCallback(  
                        azureServiceTokenProvider.KeyVaultTokenCallback));  
                builder.AddAzureKeyVault(  
                    keyVaultEndpoint, keyVaultClient, new DefaultKeyVaultSecretManager());  
            }  
        })  
        .ConfigureWebHostDefaults(webBuilder =>  
        {  
            webBuilder.UseStartup<Startup>();  
        });  
  
    private static string GetKeyVaultEndpoint() =>  
        Environment.GetEnvironmentVariable("KEYVAULT_ENDPOINT");
```

6. Add your Key Vault URL to launchsettings.json file. The environment variable name *KEYVAULT_ENDPOINT* is defined in the code you added in step 6.



7. Start debugging the project. It should run successfully.

ASP.NET and .NET applications

.NET 4.7.1 supports Key Vault and Secret configuration builders, which ensures secrets can be moved outside of source control folder without code changes. To proceed, [download .NET 4.7.1](#) and migrate your application if it's using an older version of .NET framework.

Save secret settings in a secret file that is outside of source control folder

If you are writing a quick prototype and don't want to provision Azure resources, go with this option.

1. Install the following NuGet package to your project

```
Microsoft.Configuration.ConfigurationBuilders.Base
```

2. Create a file that's similar to the following. Save it under a location outside of your project folder.

```
<root>
  <secrets ver="1.0">
    <secret name="secret1" value="foo_one" />
    <secret name="secret2" value="foo_two" />
  </secrets>
</root>
```

3. Define the secret file to be a configuration builder in your Web.config file. Put this section before *appSettings* section.

```
<configBuilders>
  <builders>
    <add name="Secrets"
      secretsFile="C:\Users\AppData\MyWebApplication1\secret.xml"
      type="Microsoft.Configuration.ConfigurationBuilders.UserSecretsConfigBuilder,
        Microsoft.Configuration.ConfigurationBuilders, Version=1.0.0.0, Culture=neutral" />
  </builders>
</configBuilders>
```

4. Specify *appSettings* section is using the secret configuration builder. Make sure there is an entry for the *secret* setting with a dummy value.

```
<appSettings configBuilders="Secrets">
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  <add key="secret" value="" />
</appSettings>
```

5. Debug your app. It should run successfully.

Save secret settings in an Azure Key Vault

Follow instructions from ASP.NET core section to configure a Key Vault for your project.

1. Install the following NuGet package to your project

```
Microsoft.Configuration.ConfigurationBuilders.UserSecrets
```

2. Define Key Vault configuration builder in Web.config. Put this section before *appSettings* section. Replace *vaultName* to be the Key Vault name if your Key Vault is in public Azure, or full URI if you are using Sovereign cloud.

```
<configSections>
  <section name="configBuilders" type="System.Configuration.ConfigurationBuildersSection,
System.Configuration, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
restartOnExternalChanges="false" requirePermission="false" />
</configSections>
<configBuilders>
  <builders>
    <add name="AzureKeyVault" vaultName="Test911"
type="Microsoft.Configuration.ConfigurationBuilders.AzureKeyVaultConfigBuilder, ConfigurationBuilders,
Version=1.0.0.0, Culture=neutral" />
  </builders>
</configBuilders>
```

3. Specify appSettings section is using the Key Vault configuration builder. Make sure there is any entry for the secret setting with a dummy value.

```
<appSettings configBuilders="AzureKeyVault">
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  <add key="secret" value="" />
</appSettings>
```

4. Start debugging the project. It should run successfully.

Service-to-service authentication to Azure Key Vault using .NET

12 minutes to read • [Edit Online](#)

To authenticate to Azure Key Vault, you need an Azure Active Directory (Azure AD) credential, either a shared secret or a certificate.

Managing such credentials can be difficult. It's tempting to bundle credentials into an app by including them in source or configuration files. The `Microsoft.Azure.Services.AppAuthentication` for .NET library simplifies this problem. It uses the developer's credentials to authenticate during local development. When the solution is later deployed to Azure, the library automatically switches to application credentials. Using developer credentials during local development is more secure because you don't need to create Azure AD credentials or share credentials between developers.

The `Microsoft.Azure.Services.AppAuthentication` library manages authentication automatically, which in turn lets you focus on your solution, rather than your credentials. It supports local development with Microsoft Visual Studio, Azure CLI, or Azure AD Integrated Authentication. When deployed to an Azure resource that supports a managed identity, the library automatically uses [managed identities for Azure resources](#). No code or configuration changes are required. The library also supports direct use of Azure AD [client credentials](#) when a managed identity isn't available, or when the developer's security context can't be determined during local development.

Prerequisites

- [Visual Studio 2019](#) or [Visual Studio 2017 v15.5](#).
- The App Authentication extension for Visual Studio, available as a separate extension for Visual Studio 2017 Update 5 and bundled with the product in Update 6 and later. With Update 6 or later, you can verify the installation of the App Authentication extension by selecting Azure Development tools from within the Visual Studio installer.

Using the library

For .NET applications, the simplest way to work with a managed identity is through the `Microsoft.Azure.Services.AppAuthentication` package. Here's how to get started:

1. Select **Tools > NuGet Package Manager > Manage NuGet Packages for Solution** to add references to the `Microsoft.Azure.Services.AppAuthentication` and `Microsoft.Azure.KeyVault` NuGet packages to your project.
2. Add the following code:

```
using Microsoft.Azure.Services.AppAuthentication;
using Microsoft.Azure.KeyVault;

// Instantiate a new KeyVaultClient object, with an access token to Key Vault
var azureServiceTokenProvider1 = new AzureServiceTokenProvider();
var kv = new KeyVaultClient(new
    KeyVaultClient.AuthenticationCallback(azureServiceTokenProvider1.KeyVaultTokenCallback));

// Optional: Request an access token to other Azure services
var azureServiceTokenProvider2 = new AzureServiceTokenProvider();
string accessToken = await
    azureServiceTokenProvider2.GetAccessTokenAsync("https://management.azure.com/").ConfigureAwait(false);
```

The `AzureServiceTokenProvider` class caches the token in memory and retrieves it from Azure AD just before expiration. So, you no longer have to check the expiration before calling the `GetAccessTokenAsync` method. Just call the method

when you want to use the token.

The `GetAccessTokenAsync` method requires a resource identifier. To learn more about Microsoft Azure services, see [What is managed identities for Azure resources](#).

Local development authentication

For local development, there are two primary authentication scenarios: [authenticating to Azure services](#), and [authenticating to custom services](#).

Authenticating to Azure Services

Local machines don't support managed identities for Azure resources. As a result, the `Microsoft.Azure.Services.AppAuthentication` library uses your developer credentials to run in your local development environment. When the solution is deployed to Azure, the library uses a managed identity to switch to an OAuth 2.0 client credential grant flow. This approach means you can test the same code locally and remotely without worry.

For local development, `AzureServiceTokenProvider` fetches tokens using **Visual Studio, Azure command-line interface (CLI), or Azure AD Integrated Authentication**. Each option is tried sequentially and the library uses the first option that succeeds. If no option works, an `AzureServiceTokenProviderException` exception is thrown with detailed information.

Authenticating with Visual Studio

To authenticate by using Visual Studio:

1. Sign in to Visual Studio and use **Tools > Options** to open **Options**.
2. Select **Azure Service Authentication**, choose an account for local development, and select **OK**.

If you run into problems using Visual Studio, such as errors that involve the token provider file, carefully review the preceding steps.

You may need to reauthenticate your developer token. To do so, select **Tools > Options**, and then select **Azure Service Authentication**. Look for a **Re-authenticate** link under the selected account. Select it to authenticate.

Authenticating with Azure CLI

To use Azure CLI for local development, be sure you have version [Azure CLI v2.0.12](#) or later.

To use Azure CLI:

1. Search for Azure CLI in the Windows Taskbar to open the **Microsoft Azure Command Prompt**.
2. Sign in to the Azure portal: `az login` to sign in to Azure.
3. Verify access by entering `az account get-access-token --resource https://vault.azure.net`. If you receive an error, check that the right version of Azure CLI is correctly installed.

If Azure CLI isn't installed to the default directory, you may receive an error reporting that `AzureServiceTokenProvider` can't find the path for Azure CLI. Use the **AzureCLIPath** environment variable to define the Azure CLI installation folder. `AzureServiceTokenProvider` adds the directory specified in the **AzureCLIPath** environment variable to the **Path** environment variable when necessary.

4. If you're signed in to Azure CLI using multiple accounts or your account has access to multiple subscriptions, you need to specify the subscription to use. Enter the command `az account set --subscription`.

This command generates output only on failure. To verify the current account settings, enter the command `az account list`.

Authenticating with Azure AD authentication

To use Azure AD authentication, verify that:

- Your on-premises Active Directory syncs to Azure AD. For more information, see [What is hybrid identity with Azure Active Directory?](#)

- Your code is running on a domain-joined computer.

Authenticating to custom services

When a service calls Azure services, the previous steps work because Azure services allow access to both users and applications.

When creating a service that calls a custom service, use Azure AD client credentials for local development authentication. There are two options:

- Use a service principal to sign into Azure:

1. Create a service principal. For more information, see [Create an Azure service principal with Azure CLI](#).
2. Use Azure CLI to sign in with the following command:

```
az login --service-principal -u <principal-id> --password <password> --tenant <tenant-id> --allow-no-subscriptions
```

Because the service principal may not have access to a subscription, use the `--allow-no-subscriptions` argument.

- Use environment variables to specify service principal details. For more information, see [Running the application using a service principal](#).

After you've signed in to Azure, `AzureServiceTokenProvider` uses the service principal to retrieve a token for local development.

This approach applies only to local development. When your solution is deployed to Azure, the library switches to a managed identity for authentication.

Running the application using managed identity or user-assigned identity

When you run your code on an Azure App Service or an Azure VM with a managed identity enabled, the library automatically uses the managed identity. No code changes are required, but the managed identity must have *get* permissions for the key vault. You can give the managed identity *get* permissions through the key vault's *Access Policies*.

Alternatively, you may authenticate with a user-assigned identity. For more information on user-assigned identities, see [About Managed Identities for Azure resources](#). To authenticate with a user-assigned identity, you need to specify the Client ID of the user-assigned identity in the connection string. The connection string is specified in [Connection String Support](#).

Running the application using a Service Principal

It may be necessary to create an Azure AD Client credential to authenticate. This situation may happen in the following examples:

- Your code runs on a local development environment, but not under the developer's identity. Service Fabric, for example, uses the [NetworkService account](#) for local development.
- Your code runs on a local development environment and you authenticate to a custom service, so you can't use your developer identity.
- Your code is running on an Azure compute resource that doesn't yet support managed identities for Azure resources, such as Azure Batch.

There are three primary methods of using a Service Principal to run your application. To use any of them, you must first create a service principal. For more information, see [Create an Azure service principal with Azure CLI](#).

Use a certificate in local keystore to sign into Azure AD

1. Create a service principal certificate using the Azure CLI `az ad sp create-for-rbac` command.

```
az ad sp create-for-rbac --create-cert
```

This command creates a .pem file (private key) that's stored in your home directory. Deploy this certificate to either the *LocalMachine* or *CurrentUser* store.

IMPORTANT

The CLI command generates a .pem file, but Windows only provides native support for PFX certificates. To generate a PFX certificate instead, use the PowerShell commands shown here: [Create service principal with self-signed certificate](#). These commands automatically deploy the certificate as well.

2. Set an environment variable named **AzureServicesAuthConnectionString** to the following value:

```
RunAs=App;AppId={AppId};TenantId={TenantId};CertificateThumbprint={Thumbprint};  
CertificateStoreLocation={CertificateStore}
```

Replace *{AppId}*, *{TenantId}*, and *{Thumbprint}* with values generated in Step 1. Replace *{CertificateStore}* with either *LocalMachine`* or *CurrentUser*, based on your deployment plan.

3. Run the application.

Use a shared secret credential to sign into Azure AD

1. Create a service principal certificate with a password using the Azure CLI `az ad sp create-for-rbac` command with the `--sdk-auth` parameter.

```
az ad sp create-for-rbac --sdk-auth
```

2. Set an environment variable named **AzureServicesAuthConnectionString** to the following value:

```
RunAs=App;AppId={AppId};TenantId={TenantId};AppKey={ClientSecret}
```

Replace *{AppId}*, *{TenantId}*, and *{ClientSecret}* with values generated in Step 1.

3. Run the application.

Once everything's set up correctly, no further code changes are necessary. `AzureServiceTokenProvider` uses the environment variable and the certificate to authenticate to Azure AD.

Use a certificate in Key Vault to sign into Azure AD

This option lets you store a service principal's client certificate in Key Vault and use it for service principal authentication. You may use this option for the following scenarios:

- Local authentication, where you want to authenticate using an explicit service principal, and want to keep the service principal credential securely in a key vault. Developer account must have access to the key vault.
- Authentication from Azure where you want to use explicit credential and want to keep the service principal credential securely in a key vault. You might use this option for a cross-tenant scenario. Managed identity must have access to key vault.

The managed identity or your developer identity must have permission to retrieve the client certificate from the Key Vault. The AppAuthentication library uses the retrieved certificate as the service principal's client credential.

To use a client certificate for service principal authentication:

1. Create a service principal certificate and automatically store it in your Key Vault. Use the Azure CLI `az ad sp create-for-rbac --keyvault <keyvaultname> --cert <certificatename> --create-cert --skip-assignment` command:

```
az ad sp create-for-rbac --keyvault <keyvaultname> --cert <certificatename> --create-cert --skip-assignment
```

The certificate identifier will be a URL in the format

```
https://<keyvaultname>.vault.azure.net/secrets/<certificatename>
```

2. Replace `{KeyVaultCertificateSecretIdentifier}` in this connection string with the certificate identifier:

```
RunAs=App;AppId={TestAppId};KeyVaultCertificateSecretIdentifier={KeyVaultCertificateSecretIdentifier}
```

For instance, if your key vault was called *myKeyVault* and you created a certificate called *myCert*, the certificate identifier would be:

```
RunAs=App;AppId={TestAppId};KeyVaultCertificateSecretIdentifier=https://myKeyVault.vault.azure.net/secrets/myCert
```

Connection String Support

By default, `AzureServiceTokenProvider` uses multiple methods to retrieve a token.

To control the process, use a connection string passed to the `AzureServiceTokenProvider` constructor or specified in the `AzureServicesAuthConnectionString` environment variable.

The following options are supported:

CONNECTION STRING OPTION	SCENARIO	COMMENTS
<code>RunAs=Developer;DeveloperTool=AzureCli</code>	Local development	<code>AzureServiceTokenProvider</code> uses AzureCli to get token.
<code>RunAs=Developer;DeveloperTool=VisualStudio</code>	Local development	<code>AzureServiceTokenProvider</code> uses Visual Studio to get token.
<code>RunAs=CurrentUser</code>	Local development	<code>AzureServiceTokenProvider</code> uses Azure AD Integrated Authentication to get token.
<code>RunAs=App</code>	Managed identities for Azure resources	<code>AzureServiceTokenProvider</code> uses a managed identity to get token.
<code>RunAs=App;AppId={ClientId of user-assigned identity}</code>	User-assigned identity for Azure resources	<code>AzureServiceTokenProvider</code> uses a user-assigned identity to get token.
<code>RunAs=App;AppId={TestAppId};KeyVaultCertificateSecretIdentifier={KeyVaultCertificateSecretIdentifier}</code>	Custom services authentication	<code>KeyVaultCertificateSecretIdentifier</code> is the certificate's secret identifier.
<code>RunAs=App;AppId={AppId};TenantId={TenantId};CertificateThumbprint={Thumbprint};CertificateStoreLocation={LocalMachine or CurrentUser}</code>	Service principal	<code>AzureServiceTokenProvider</code> uses certificate to get token from Azure AD.
<code>RunAs=App;AppId={AppId};TenantId={TenantId};CertificateSubjectName={Subject};CertificateStoreLocation={LocalMachine or CurrentUser}</code>	Service principal	<code>AzureServiceTokenProvider</code> uses certificate to get token from Azure AD
<code>RunAs=App;AppId={AppId};TenantId={TenantId};AppKey={ClientSecret}</code>	Service principal	<code>AzureServiceTokenProvider</code> uses secret to get token from Azure AD.

CONNECTION STRING OPTION	SCENARIO	COMMENTS
--------------------------	----------	----------

Samples

To see the `Microsoft.Azure.Services.AppAuthentication` library in action, refer to the following code samples.

- [Use a managed identity to retrieve a secret from Azure Key Vault at runtime](#)
- [Programmatically deploy an Azure Resource Manager template from an Azure VM with a managed identity.](#)
- [Use .NET Core sample and a managed identity to call Azure services from an Azure Linux VM.](#)

AppAuthentication Troubleshooting

Common issues during local development

Azure CLI is not installed, you're not logged in, or you don't have the latest version

Run `az account get-access-token` to see if Azure CLI shows a token for you. If it says **no such program found**, install the [latest version of the Azure CLI](#). You may be prompted to sign in.

AzureServiceTokenProvider can't find the path for Azure CLI

AzureServiceTokenProvider looks for Azure CLI at its default install locations. If it can't find Azure CLI, set environment variable **AzureCLIPath** to the Azure CLI installation folder. AzureServiceTokenProvider will add the environment variable to the Path environment variable.

You're logged into Azure CLI using multiple accounts, the same account has access to subscriptions in multiple tenants, or you get an Access Denied error when trying to make calls during local development

Using Azure CLI, set the default subscription to one that has the account you want to use. The subscription must be in the same tenant as the resource you want to access: `az account set --subscription [subscription-id]`. If no output is seen, it succeeded. Verify the right account is now the default using `az account list`.

Common issues across environments

Unauthorized access, access denied, forbidden, or similar error

The principal used doesn't have access to the resource it's trying to access. Grant either your user account or the App Service's MSI "Contributor" access to a resource. Which one depends on whether you're running the sample on your local computer or deployed in Azure to your App Service. Some resources, like key vaults, also have their own [access policies](#) that you use grant access to principals, such as users, apps, and groups.

Common issues when deployed to Azure App Service

Managed identity isn't set up on the App Service

Check the environment variables `MSI_ENDPOINT` and `MSI_SECRET` exist using [Kudu debug console](#). If these environment variables don't exist, Managed Identity isn't enabled on the App Service.

Common issues when deployed locally with IIS

Can't retrieve tokens when debugging app in IIS

By default, AppAuth runs in a different user context in IIS. That's why it doesn't have access to use your developer identity to retrieve access tokens. You can configure IIS to run with your user context with the following two steps:

- Configure the Application Pool for the web app to run as your current user account. See more information [here](#)
- Configure "setProfileEnvironment" to "True". See more information [here](#).
 - Go to `%windir%\System32\inetsrv\config\applicationHost.config`
 - Search for "setProfileEnvironment". If it's set to "False", change it to "True". If it's not present, add it as an attribute to the `processModel` element
`(/configuration/system.applicationHost/applicationPools/applicationPoolDefaults/processModel/@setProfileEnvironment)`, and set it to "True".

- Learn more about [managed identities for Azure resources](#).
- Learn more about [Azure AD authentication scenarios](#).

Add Key Vault to your web application by using Visual Studio Connected Services

8 minutes to read • [Edit Online](#)

In this tutorial, you will learn how to easily add everything you need to start using Azure Key Vault to manage your secrets for web projects in Visual Studio, whether you are using ASP.NET Core or any type of ASP.NET project. By using the Connected Services feature in Visual Studio, you can have Visual Studio automatically add all the NuGet packages and configuration settings you need to connect to Key Vault in Azure.

For details on the changes that Connected Services makes in your project to enable Key Vault, see [Key Vault Connected Service - What happened to my ASP.NET 4.7.1 project](#) or [Key Vault Connected Service - What happened to my ASP.NET Core project](#).

Prerequisites

- **An Azure subscription.** If you don't have a subscription, sign up for a [free account](#).
- **Visual Studio 2019 version 16.3 or later, or Visual Studio 2017 version 15.7** with the **Web Development** workload installed. [Download it now](#).
- For ASP.NET (not Core) with Visual Studio 2017, you need the .NET Framework 4.7.1 or later Development Tools, which are not installed by default. To install them, launch the Visual Studio Installer, choose **Modify**, and then choose **Individual Components**, then on the right-hand side, expand **ASP.NET and web development**, and choose **.NET Framework 4.7.1 Development Tools**.
- An ASP.NET 4.7.1 or later, or ASP.NET Core 2.0 or later web project open.

Add Key Vault support to your project

Before you begin, make sure that you're signed into Visual Studio. Sign in with the same account that you use for your Azure subscription. Then open an ASP.NET 4.7.1 or later, or ASP.NET Core 2.0 web project, and do the follow steps:

1. In **Solution Explorer**, right-click the project that you want to add the Key Vault support to, and choose **Add > Connected Service**. The Connected Service page appears with services you can add to your project.
2. In the menu of available services, choose **Secure Secrets With Azure Key Vault**.

WebApplication7 About.aspx.cs

Overview Connected Services Publish

Connected Services

Add code and dependencies for one of these services to your application

- Monitoring with Application Insights**
Gain visibility into your application using Application Insights right from Visual Studio.
- Cloud Storage with Azure Storage**
Store and access data with Azure Storage services like Blobs, Queues, and Tables.
- Secure Secrets with Azure Key Vault**
Secure your application by moving secrets from source code into an Azure Key Vault
- Access Office 365 Services with Microsoft Graph**
Use the Microsoft Graph API to integrate your applications with Office 365 services.

[Find more services...](#)

3. Select the subscription you want to use, and then choose a new or existing Key Vault. If you choose the new Key Vault, an **Edit** link appears. Select it to configure your new Key Vault.

WebApplicationASPD... - Azure Key Vault WebApplicationASPDotNETTest

Azure Key Vault
Secure your application by moving secrets from source code into an Azure Key Vault

Contoso meganb@contoso.com

Subscription:	Contoso Subscription
Key Vault:	WebApplicationASPDotNETTest-0-kv (new)

Adding an Azure Key Vault will:

- Create a new Key Vault in resource group 'WebApplicationASPDotNETTest-0-rg' in East Asia
- Create a new resource group to host your Key Vault
- Use the Standard pricing tier
- Add a reference to the endpoint of your Key Vault so your application can load secrets from this Key Vault

[How to secure my app secrets?](#)
[Review Key Vault pricing](#)

Add

4. In **Edit Azure Key Vault**, enter the name you want to use for the Key Vault.
5. Select an existing **Resource Group**, or choose to create a new one with an automatically generated unique name. If you want to create a new group with a different name, you can use the [Azure portal](#), and then close the page and restart to reload the list of resource groups.
6. Choose the **Location** in which to create the Key Vault. If your web application is hosted in Azure, choose the region that hosts the web application for optimum performance.
7. Choose a **Pricing tier**. For details, see [Key Vault Pricing](#).
8. Choose **OK** to accept the configuration choices.
9. After you select an existing Key Vault or have configured a new Key Vault, in the **Azure Key Vault** tab of Visual Studio, select **Add** to add the Connected Service.
10. Select the **Manage secrets stored in this Key Vault** link to open the **Secrets** page for your Key Vault. If

you closed the page or the project, you can navigate to it in the [Azure portal](#) by choosing **All Services** and, under **Security**, choosing **Key Vault**, then choose your Key Vault.

11. In the Key Vault section for the Key Vault you created, choose **Secrets**, then **Generate/Import**.

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation menu is open, showing various service categories like Home, Dashboard, All services, and Favorites. Under Favorites, 'KeyVaultConnectedService' is listed. The main content area shows the 'KeyVaultConnectedService - Secrets' blade. At the top right of this blade, there is a red box highlighting the '+ Generate/Import' button. Below the button is a search bar and a refresh/restore backup link. The main table area displays a message: 'There are no secrets available.' The sidebar on the left lists other management options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (with Keys selected), Certificates, Access policies, Firewalls and virtual networks, Properties, Locks, Export template, Monitoring, and Alerts.

12. Enter a secret, such as *MySecret* and give it any string value as a test, then select the **Create** button.

The screenshot shows the 'Create a secret' form within the Azure portal. The form has a header 'Home > KeyVaultConnectedService - Secrets > Create a secret'. The main area contains the following fields:

- Upload options:** A dropdown menu set to 'Manual'.
- Name:** A required field labeled '*' containing the value 'MySecret'.
- Value:** A required field labeled '*' containing the value '*****'.
- Content type (optional):** An empty input field.
- Set activation date?**: A checkbox followed by a help icon.
- Set expiration date?**: A checkbox followed by a help icon.
- Enabled?**: A toggle switch with 'Yes' and 'No' options.

At the bottom of the form is a large blue 'Create' button.

13. (optional) Enter another secret, but this time put it into a category by naming it *Secrets--MySecret*. This syntax specifies a category "Secrets" that contains a secret "MySecret".

Now, you can access your secrets in code. The next steps are different depending on whether you are using ASP.NET 4.7.1 or ASP.NET Core.

Access your secrets in code (ASP.NET Core)

1. In Solution Explorer, right-click on your project, and select **Manage NuGet Packages**. In the **Browse** tab, locate and install these two NuGet packages: [Microsoft.Azure.Services.AppAuthentication](#) and for .NET Core 2, add [Microsoft.Azure.KeyVault](#) or for .NET Core 3, add [Microsoft.Azure.KeyVault.Core](#).
2. For .NET Core 2, select the `Program.cs` tab and change the `BuildWebHost` definition in the Program class to the following:

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((ctx, builder) =>
    {
        var keyVaultEndpoint = GetKeyVaultEndpoint();
        if (!string.IsNullOrEmpty(keyVaultEndpoint))
        {
            var azureServiceTokenProvider = new AzureServiceTokenProvider();
            var keyVaultClient = new KeyVaultClient(
                new KeyVaultClient.AuthenticationCallback(
                    azureServiceTokenProvider.KeyVaultTokenCallback));
            builder.AddAzureKeyVault(
                keyVaultEndpoint, keyVaultClient, new DefaultKeyVaultSecretManager());
        }
    })
    .UseStartup<Startup>();

private static string GetKeyVaultEndpoint() => "https://<YourKeyVaultName>.vault.azure.net";
}
```

For .NET Core 3, use the following code.

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((context, config) =>
    {
        var keyVaultEndpoint = GetKeyVaultEndpoint();
        if (!string.IsNullOrEmpty(keyVaultEndpoint))
        {
            var azureServiceTokenProvider = new AzureServiceTokenProvider();
            var keyVaultClient = new KeyVaultClient(
                new KeyVaultClient.AuthenticationCallback(
                    azureServiceTokenProvider.KeyVaultTokenCallback));
            config.AddAzureKeyVault(keyVaultEndpoint, keyVaultClient, new
DefaultKeyVaultSecretManager());
        }
    })
    .ConfigureWebHostDefaults(webBuilder =>
    {
        webBuilder.UseStartup<Startup>();
    });
private static string GetKeyVaultEndpoint() => "https://<YourKeyVaultName>.vault.azure.net";
```

3. Next open one of the page files, such as `Index.cshtml.cs` and write the following code:

- a. Include a reference to `Microsoft.Extensions.Configuration` by this using directive:

```
using Microsoft.Extensions.Configuration;
```

- b. Add the configuration variable.

```
private static readonly IConfiguration _configuration;
```

- c. Add this constructor or replace the existing constructor with this:

```
public IndexModel(IConfiguration configuration)
{
    _configuration = configuration;
}
```

- d. Update the `OnGet` method. Update the placeholder value shown here with the secret name you created in the above commands.

```
public void OnGet()
{
    ViewData["Message"] = "My key val = " + _configuration["
<YourSecretNameThatWasCreatedAbove>"];
}
```

- e. To confirm the value at runtime, add code to display `ViewData["Message"]` to the `.cshtml` file to display the secret in a message.

```
<p>@ViewData["Message"]</p>
```

You can run the app locally to verify that the secret is obtained successfully from the Key Vault.

Access your secrets (ASP.NET)

You can set up the configuration so that the `web.config` file has a dummy value in the `appSettings` element that is replaced by the true value at runtime. You can then access this via the `ConfigurationManager.AppSettings` data structure.

1. Edit your `web.config` file. Find the `appSettings` tag, add an attribute `configBuilders="AzureKeyVault"`, and add a line:

```
<add key="mysecret" value="dummy"/>
```

2. Edit the `About` method in `HomeController.cs`, to display the value for confirmation.

```
public ActionResult About()
{
    ViewBag.Message = "Key vault value = " + ConfigurationManager.AppSettings["mysecret"];
}
```

3. Run the app locally under the debugger, switch to the **About** tab, and verify that the value from the Key Vault is displayed.

Clean up resources

When no longer needed, delete the resource group. This deletes the Key Vault and related resources. To delete the resource group through the portal:

1. Enter the name of your resource group in the Search box at the top of the portal. When you see the resource

group used in this quickstart in the search results, select it.

2. Select **Delete resource group**.
3. In the **TYPE THE RESOURCE GROUP NAME:** box, enter in the name of the resource group and select **Delete**.

Troubleshooting

If your Key Vault is running on a different Microsoft account than the one you're logged in to Visual Studio (for example, the Key Vault is running on your work account, but Visual Studio is using your private account) you get an error in your Program.cs file, that Visual Studio can't get access to the Key Vault. To fix this issue:

1. Go to the [Azure portal](#) and open your Key Vault.
2. Choose **Access policies**, then **Add Access Policy**, and choose the account you are logged in with as Principal.
3. In Visual Studio, choose **File > Account Settings**. Select **Add an account** from the **All account** section. Sign in with the account you have chosen as Principal of your access policy.
4. Choose **Tools > Options**, and look for **Azure Service Authentication**. Then select the account you just added to Visual Studio.

Now, when you debug your application, Visual Studio connects to the account your Key Vault is located on.

How your ASP.NET Core project is modified

This section identifies the exact changes made to an ASP.NET project when adding the Key Vault connected service using Visual Studio.

Added references for ASP.NET Core

Affects the project file .NET references and NuGet package references.

TYPE	REFERENCE
NuGet	Microsoft.AspNetCore.AzureKeyVault.HostingStartup

Added files for ASP.NET Core

- `ConnectedService.json` added, which records some information about the Connected Service provider, version, and a link the documentation.

Project file changes for ASP.NET Core

- Added the Connected Services ItemGroup and `ConnectedServices.json` file.

launchsettings.json changes for ASP.NET Core

- Added the following environment variable entries to both the IIS Express profile and the profile that matches your web project name:

```
"environmentVariables": {  
    "ASPNETCORE_HOSTINGSTARTUP__KEYVAULT__CONFIGURATIONENABLED": "true",  
    "ASPNETCORE_HOSTINGSTARTUP__KEYVAULT__CONFIGURATIONVULT": "<your keyvault URL>"  
}
```

Changes on Azure for ASP.NET Core

- Created a resource group (or used an existing one).
- Created a Key Vault in the specified resource group.

How your ASP.NET Framework project is modified

This section identifies the exact changes made to an ASP.NET project when adding the Key Vault connected service using Visual Studio.

Added references for ASP.NET Framework

Affects the project file .NET references and `packages.config` (NuGet references).

TYPE	REFERENCE
.NET; NuGet	Microsoft.Azure.KeyVault
.NET; NuGet	Microsoft.Azure.KeyVault.WebKey
.NET; NuGet	Microsoft.Rest.ClientRuntime
.NET; NuGet	Microsoft.Rest.ClientRuntime.Azure

Added files for ASP.NET Framework

- `ConnectedService.json` added, which records some information about the Connected Service provider, version, and a link to the documentation.

Project file changes for ASP.NET Framework

- Added the Connected Services ItemGroup and `ConnectedServices.json` file.
- References to the .NET assemblies described in the [Added references](#) section.

web.config or app.config changes

- Added the following configuration entries:

```
<configSections>
  <section
    name="configBuilders"
    type="System.Configuration.ConfigurationBuildersSection, System.Configuration, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
    restartOnExternalChanges="false"
    requirePermission="false" />
</configSections>
<configBuilders>
  <builders>
    <add
      name="AzureKeyVault"
      vaultName="vaultname"
      type="Microsoft.Configuration.ConfigurationBuilders.AzureKeyVaultConfigBuilder,
Microsoft.Configuration.ConfigurationBuilders.Azure, Version=1.0.0.0, Culture=neutral"
      vaultUri="https://vaultname.vault.azure.net" />
  </builders>
</configBuilders>
```

Changes on Azure for ASP.NET Framework

- Created a resource group (or used an existing one).
- Created a Key Vault in the specified resource group.

Next steps

If you followed this tutorial, your Key Vault permissions are set up to run with your own Azure subscription, but that might not be desirable for a production scenario. You can create a managed identity to manage Key Vault

access for your app. See [Provide Key Vault authentication with a managed identity](#).

Learn more about Key Vault development by reading the [Key Vault Developer's Guide](#).

Authentication, requests and responses

3 minutes to read • [Edit Online](#)

Azure Key Vault supports JSON formatted requests and responses. Requests to the Azure Key Vault are directed to a valid Azure Key Vault URL using HTTPS with some URL parameters and JSON encoded request and response bodies.

This topic covers specifics for the Azure Key Vault service. For general information on using Azure REST interfaces, including authentication/authorization and how to acquire an access token, see [Azure REST API Reference](#).

Request URL

Key management operations use HTTP DELETE, GET, PATCH, PUT and HTTP POST and cryptographic operations against existing key objects use HTTP POST. Clients that cannot support specific HTTP verbs may also use HTTP POST using the X-HTTP-REQUEST header to specify the intended verb; requests that do not normally require a body should include an empty body when using HTTP POST, for example when using POST instead of DELETE.

To work with objects in the Azure Key Vault, the following are example URLs:

- To CREATE a key called TESTKEY in a Key Vault use -
`PUT /keys/TESTKEY?api-version=<api_version> HTTP/1.1`
- To IMPORT a key called IMPORTEDKEY into a Key Vault use -
`POST /keys/IMPORTEDKEY/import?api-version=<api_version> HTTP/1.1`
- To GET a secret called MYSECRET in a Key Vault use -
`GET /secrets/MYSECRET?api-version=<api_version> HTTP/1.1`
- To SIGN a digest using a key called TESTKEY in a Key Vault use -
`POST /keys/TESTKEY/sign?api-version=<api_version> HTTP/1.1`

The authority for a request to a Key Vault is always as follows, `https://<keyvault-name>.vault.azure.net/`

Keys are always stored under the /keys path, Secrets are always stored under the /secrets path.

API Version

The Azure Key Vault Service supports protocol versioning to provide compatibility with down-level clients, although not all capabilities will be available to those clients. Clients must use the `api-version` query string parameter to specify the version of the protocol that they support as there is no default.

Azure Key Vault protocol versions follow a date numbering scheme using a {YYYY}.{MM}.{DD} format.

Request Body

As per the HTTP specification, GET operations must NOT have a request body, and POST and PUT operations must have a request body. The body in DELETE operations is optional in HTTP.

Unless otherwise noted in operation description, the request body content type must be application/json and must contain a serialized JSON object conformant to content type.

Unless otherwise noted in operation description, the Accept request header must contain the application/json

media type.

Response Body

Unless otherwise noted in operation description, the response body content type of both successful and failed operations will be application/json and contains detailed error information.

Using HTTP POST

Some clients may not be able to use certain HTTP verbs, such as PATCH or DELETE. Azure Key Vault supports HTTP POST as an alternative for these clients provided that the client also includes the "X-HTTP-METHOD" header to specific the original HTTP verb. Support for HTTP POST is noted for each of the API defined in this document.

Error Responses

Error handling will use HTTP status codes. Typical results are:

- 2xx – Success: Used for normal operation. The response body will contain the expected result
- 3xx – Redirection: The 304 "Not Modified" may be returned to fulfill a conditional GET. Other 3xx codes may be used in the future to indicate DNS and path changes.
- 4xx – Client Error: Used for bad requests, missing keys, syntax errors, invalid parameters, authentication errors, etc. The response body will contain detailed error explanation.
- 5xx – Server Error: Used for internal server errors. The response body will contain summarized error information.

The system is designed to work behind a proxy or firewall. Therefore, a client might receive other error codes.

Azure Key Vault also returns error information in the response body when a problem occurs. The response body is JSON formatted and takes the form:

```
{  
  "error":  
  {  
    "code": "BadArgument",  
    "message":  
      "'Foo' is not a valid argument for 'type'."  
  }  
}
```

Authentication

All requests to Azure Key Vault MUST be authenticated. Azure Key Vault supports Azure Active Directory access tokens that may be obtained using OAuth2 [[RFC6749](#)].

For more information on registering your application and authenticating to use Azure Key Vault, see [Register your client application with Azure AD](#).

Access tokens must be sent to the service using the HTTP Authorization header:

```
PUT /keys/MYKEY?api-version=<api_version> HTTP/1.1
Authorization: Bearer <access_token>
```

When an access token is not supplied, or when a token is not accepted by the service, an HTTP 401 error will be returned to the client and will include the WWW-Authenticate header, for example:

```
401 Not Authorized
WWW-Authenticate: Bearer authorization="...", resource="..."
```

The parameters on the WWW-Authenticate header are:

- **authorization:** The address of the OAuth2 authorization service that may be used to obtain an access token for the request.
- **resource:** The name of the resource (<https://vault.azure.net>) to use in the authorization request.

See Also

[About keys, secrets, and certificates](#)

Common parameters and headers

2 minutes to read • [Edit Online](#)

The following information is common to all operations that you might do related to Key Vault resources:

- Replace `{api-version}` with the api-version in the URI.
- Replace `{subscription-id}` with your subscription identifier in the URI
- Replace `{resource-group-name}` with the resource group. For more information, see [Using Resource groups to manage your Azure resources](#).
- Replace `{vault-name}` with your key vault name in the URI.
- Set the Content-Type header to application/json.
- Set the Authorization header to a JSON Web Token that you obtain from Azure Active Directory (AAD). For more information, see [Authenticating Azure Resource Manager requests](#).

Common error response

The service will use HTTP status codes to indicate success or failure. In addition, failures contain a response in the following format:

```
{
  "error": {
    "code": "BadRequest",
    "message": "The key vault sku is invalid."
  }
}
```

ELEMENT NAME	TYPE	DESCRIPTION
code	string	The type of error that occurred.
message	string	A description of what caused the error.

See Also

[Azure Key Vault REST API Reference](#)

Azure Key Vault customer data features

2 minutes to read • [Edit Online](#)

Azure Key Vault receives customer data during creation or update of vaults, keys, secrets, certificates, and managed storage accounts. This Customer data is directly visible in the Azure portal and through the REST API. Customer data can be edited or deleted by updating or deleting the object that contains the data.

System access logs are generated when a user or application accesses Key Vault. Detailed access logs are available to customers using Azure Insights.

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

Identifying customer data

The following information identifies customer data within Azure Key Vault:

- Access policies for Azure Key Vault contain object-IDs representing users, groups, or applications
- Certificate subjects may include email addresses or other user or organizational identifiers
- Certificate contacts may contain user email addresses, names, or phone numbers
- Certificate issuers may contain email addresses, names, phone numbers, account credentials, and organizational details
- Arbitrary tags can be applied to Objects in Azure Key Vault. These objects include vaults, keys, secrets, certificates, and storage accounts. The tags used may contain personal data
- Azure Key Vault access logs contain object-IDs, [UPNs](#), and IP addresses for each REST API call
- Azure Key Vault diagnostic logs may contain object-IDs and IP addresses for REST API calls

Deleting customer data

The same REST APIs, Portal experience, and SDKs used to create vaults, keys, secrets, certificates, and managed storage accounts, are also able to update and delete these objects.

Soft delete allows you to recover deleted data for 90 days after deletion. When using soft delete, the data may be permanently deleted prior to the 90 days retention period expires by performing a purge operation. If the vault or subscription has been configured to block purge operations, it is not possible to permanently delete data until the scheduled retention period has passed.

Exporting customer data

The same REST APIs, portal experience, and SDKs that are used to create vaults, keys, secrets, certificates, and managed storage accounts also allow you to view and export these objects.

Azure Key Vault access logging is an optional feature that can be turned on to generate logs for each REST API call. These logs will be transferred to a storage account in your subscription where you apply the retention policy that meets your organization's requirements.

Azure Key Vault diagnostic logs that contain personal data can be retrieved by making an export request in the User Privacy portal. This request must be made by the tenant administrator.

Next steps

- [Azure Key Vault Logging](#)
- [Azure Key Vault soft-delete overview](#)
- [Azure Key Vault key operations](#)
- [Azure Key Vault secret operations](#)
- [Azure Key Vault certificates and policies](#)
- [Azure Key Vault storage account operations](#)

Azure Key Vault service limits

2 minutes to read • [Edit Online](#)

Here are the service limits for Azure Key Vault.

Key transactions (maximum transactions allowed in 10 seconds, per vault per region¹):

KEY TYPE	HSM KEY CREATE KEY	HSM KEY ALL OTHER TRANSACTIONS	SOFTWARE KEY CREATE KEY	SOFTWARE KEY ALL OTHER TRANSACTIONS
RSA 2,048-bit	5	1,000	10	2,000
RSA 3,072-bit	5	250	10	500
RSA 4,096-bit	5	125	10	250
ECC P-256	5	1,000	10	2,000
ECC P-384	5	1,000	10	2,000
ECC P-521	5	1,000	10	2,000
ECC SECP256K1	5	1,000	10	2,000

NOTE

In the previous table, we see that for RSA 2,048-bit software keys, 2,000 GET transactions per 10 seconds are allowed. For RSA 2,048-bit HSM-keys, 1,000 GET transactions per 10 seconds are allowed.

The throttling thresholds are weighted, and enforcement is on their sum. For example, as shown in the previous table, when you perform GET operations on RSA HSM-keys, it's eight times more expensive to use 4,096-bit keys compared to 2,048-bit keys. That's because $1,000/125 = 8$.

In a given 10-second interval, an Azure Key Vault client can do *only one* of the following operations before it encounters a [429](#) throttling HTTP status code:

- 2,000 RSA 2,048-bit software-key GET transactions
- 1,000 RSA 2,048-bit HSM-key GET transactions
- 125 RSA 4,096-bit HSM-key GET transactions
- 124 RSA 4,096-bit HSM-key GET transactions and 8 RSA 2,048-bit HSM-key GET transactions

Secrets, managed storage account keys, and vault transactions:

TRANSACTIONS TYPE	MAXIMUM TRANSACTIONS ALLOWED IN 10 SECONDS, PER VAULT PER REGION ¹
All transactions	2,000

For information on how to handle throttling when these limits are exceeded, see [Azure Key Vault throttling guidance](#).

¹ A subscription-wide limit for all transaction types is five times per key vault limit. For example, HSM-other

transactions per subscription are limited to 5,000 transactions in 10 seconds per subscription.