Contents

What is Azure Key Vault?

5/15/2019 • 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 Secure Sockets Layer/Transport Layer Security (SSL/TLS) 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 validates 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, 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

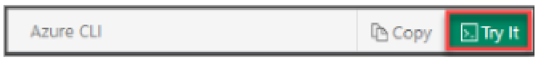
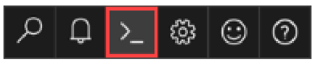
4/25/2019 • 3 minutes to read • Edit Online

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. In this quickstart, you create a key vault. 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.

Open Azure Cloud Shell

Azure Cloud Shell is a free, interactive shell that you can use to run the steps in this article. Common Azure tools are preinstalled and configured in Cloud Shell for you to use with your account. Select **Copy** to copy the code, paste it in Cloud Shell, and then press Enter to run it. There are a few ways to open Cloud Shell:

Select **Try It** in the upper-right corner of a code block.	
Open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu in the upper-right corner of the Azure portal.	

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://ContosoVault.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 have created a Key Vault and stored a secret in it. To learn more about Key Vault and how you can use it with your applications continue to the tutorial for web applications working with Key Vault.

To learn how to read a secret from Key Vault from a web application using managed identities for Azure resources, continue with the following tutorial Configure an Azure web application to read a secret from Key vault

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

4/25/2019 • 3 minutes to read • Edit Online

NOTE

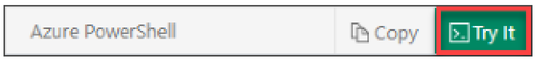
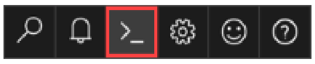
This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For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

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are a few ways to launch the Cloud Shell:

Click **Try It** in the upper right corner of a code block.	
Open Cloud Shell in your browser.	
Click the **Cloud Shell** button on the menu in the upper right of the Azure portal.	

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.

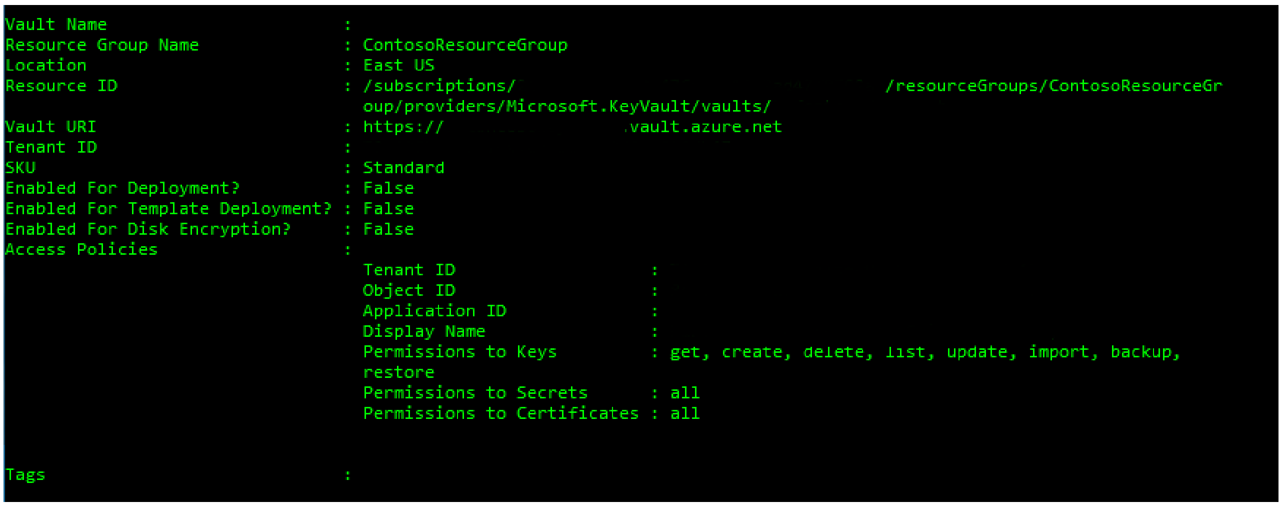
- **Vault name** Contoso-Vault2.
- **Resource group name** ContosoResourceGroup.
- **Location** East US.

```
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://contosokeyvault.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                       :
Resource Group Name              : ContosoResourceGroup
Location                         : East US
Resource ID                      : /subscriptions/                              /resourceGroups/ContosoResourceGr
                                   oup/providers/Microsoft.KeyVault/vaults/
Vault URI                        : https://                    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 'ContosoKeyVault' -Name 'ExamplePassword' -SecretValue $secretvalue
```

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

```
(Get-AzKeyVaultSecret -vaultName "Contosokeyvault" -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 have created a Key Vault and stored a software key in it. To learn more about Key Vault and how you can use it with your applications continue to the tutorial for web applications working with Key Vault.

To learn how to read a secret from Key Vault from a web application using managed identities for Azure resources, continue with the following tutorial

Configure an Azure web application to read a secret from Key vault.

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

4/25/2019 • 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.

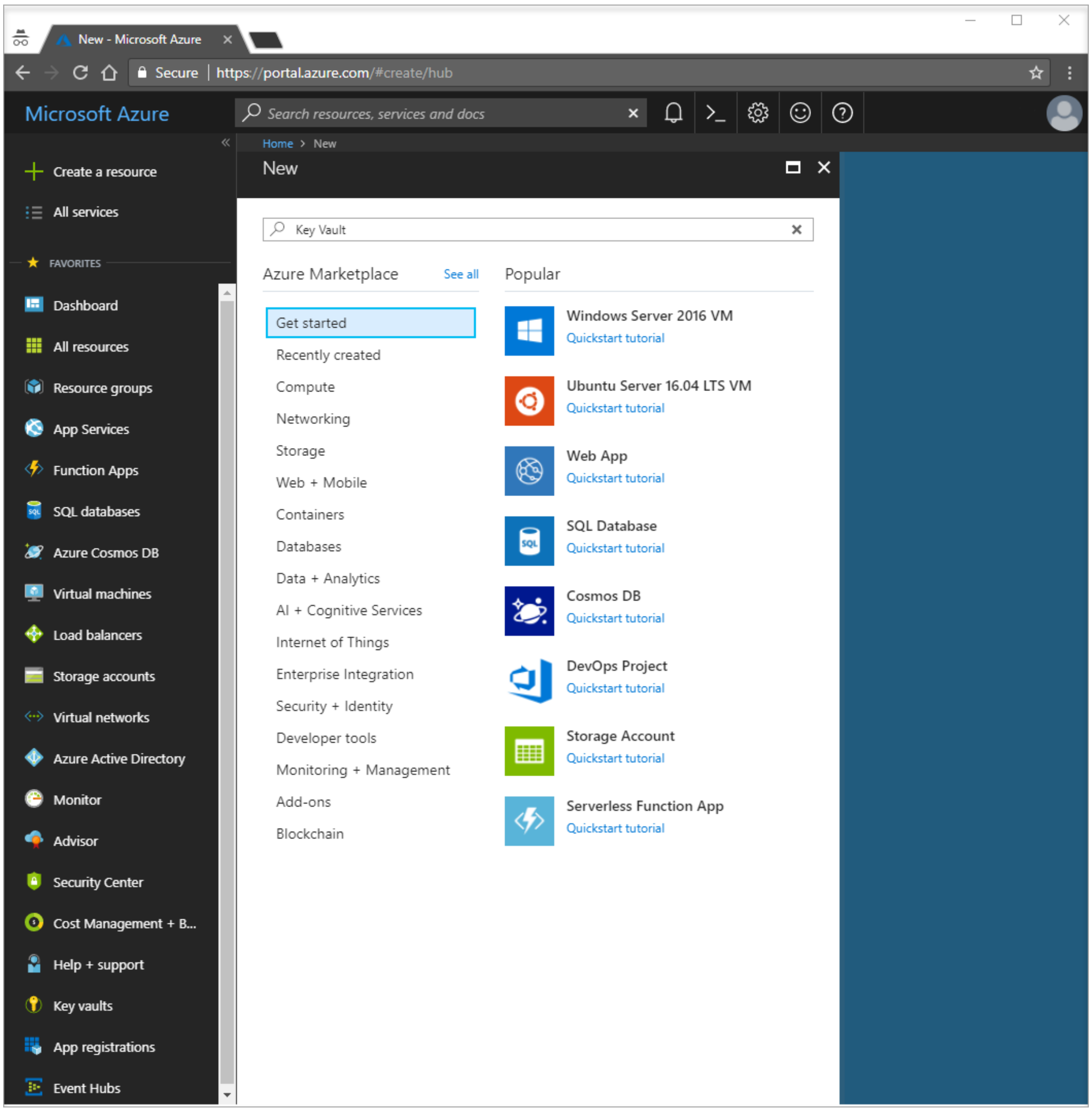
If you don't have an Azure subscription, create a free account before you begin.

Sign in to Azure

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

Create a vault

1. Select the **Create a resource** option on the upper left-hand corner of the Azure portal



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:

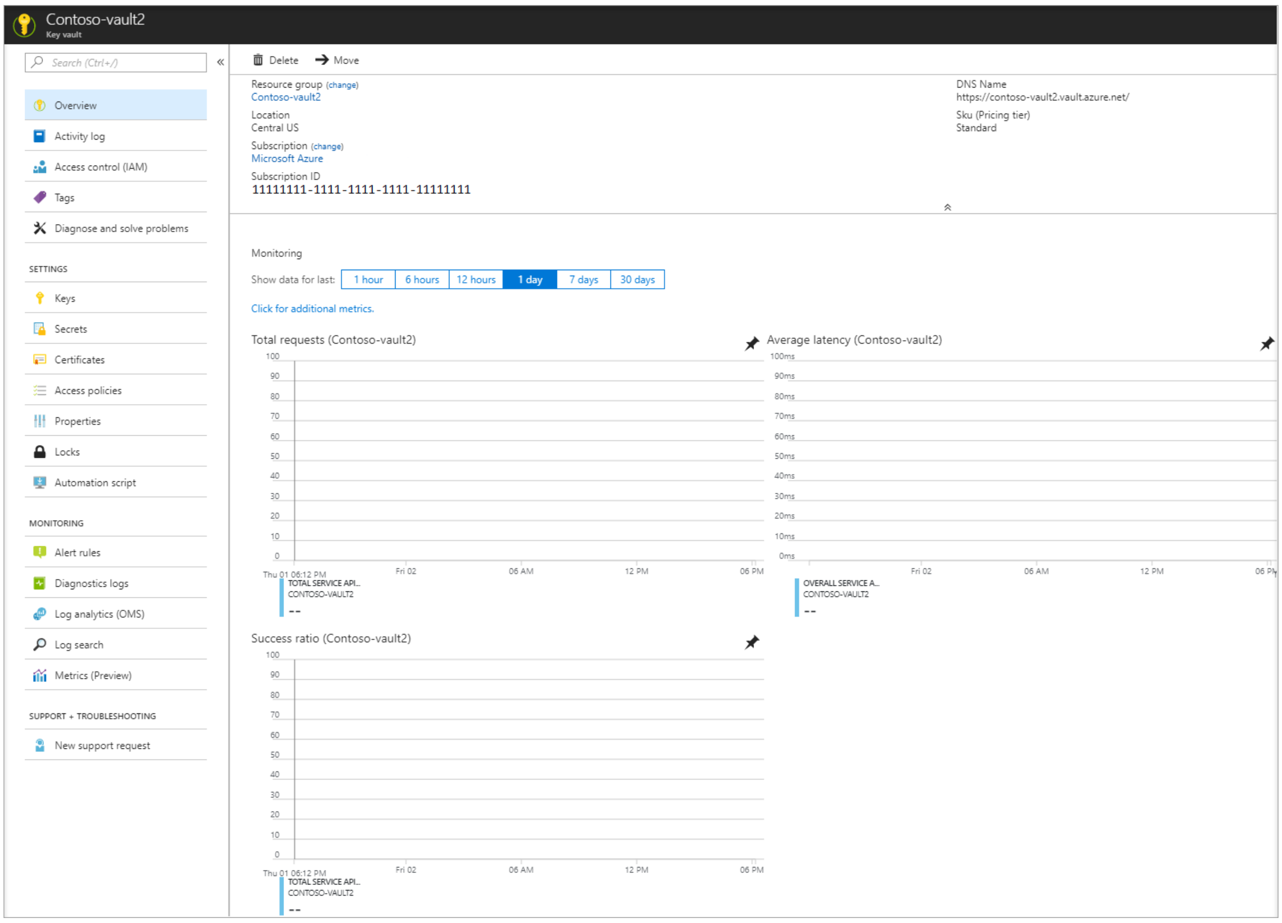
   - **Name**: A unique name is required. For this quickstart we use **Contoso-vault2**.
   - **Subscription**: Choose a subscription.
   - Under **Resource Group** choose **Create new** and enter a resource group name.
   - In the **Location** pull-down menu, choose a location.
   - Leave the other options 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.



Resource group (change)
Contoso-vault2
Location
Central US
Subscription (change)
Microsoft Azure
Subscription ID
11111111-1111-1111-1111-11111111

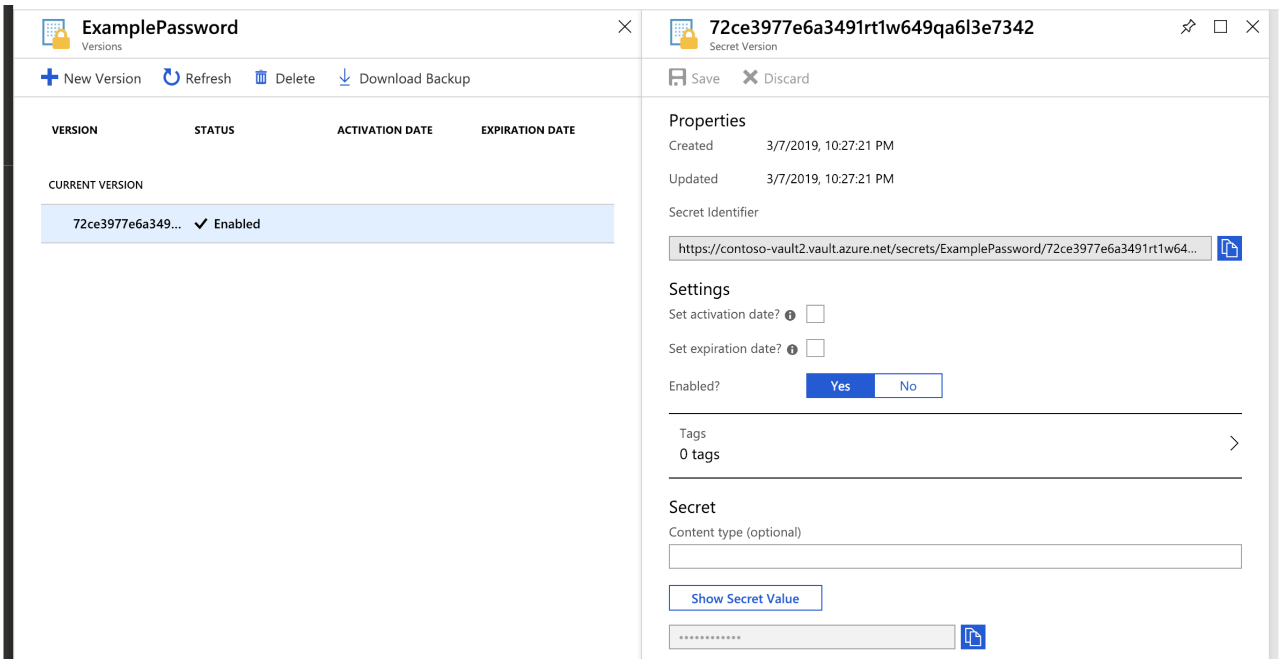
DNS Name
https://contoso-vault2.vault.azure.net/
Sku (Pricing tier)
Standard

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.


ExamplePassword
Versions
New Version | Refresh | Delete | Download Backup

VERSION	STATUS	ACTIVATION DATE	EXPIRATION DATE
CURRENT VERSION			
72ce3977e6a349...	✓ Enabled		

72ce3977e6a3491rt1w649qa6l3e7342
Secret Version
Save | Discard

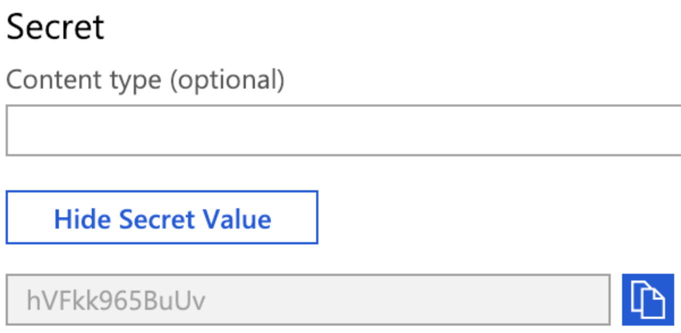
Properties
Created 3/7/2019, 10:27:21 PM
Updated 3/7/2019, 10:27:21 PM
Secret Identifier
https://contoso-vault2.vault.azure.net/secrets/ExamplePassword/72ce3977e6a3491rt1w64...

Settings
Set activation date? ☐
Set expiration date? ☐
Enabled? [✓ Yes] No

Tags
0 tags

Secret
Content type (optional)
Show Secret Value
••••••••••••

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


Secret
Content type (optional)
Hide Secret Value
hVFkk965BuUv

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 have created a Key Vault and stored a secret. To learn more about Key Vault and how you can use it with your applications, continue to the tutorial for web applications working with Key Vault.

To learn how to read a secret from Key Vault from a web application using managed identities for Azure resources, continue with the following tutorial Configure an Azure web application to read a secret from Key vault.

Quickstart: Set and retrieve a secret from Azure Key Vault by using a .NET web app

5/6/2019 • 5 minutes to read • Edit Online

In this quickstart, you follow the steps for getting an Azure web application to read information from Azure Key Vault by using managed identities for Azure resources. Using Key Vault helps keep the information secure. You learn how to:

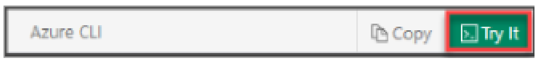
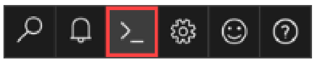
- Create a key vault.
- Store a secret in the key vault.
- Retrieve a secret from the key vault.
- Create an Azure web application.
- Enable a managed service identity for the web app.
- Grant the required permissions for the web application to read data from the key vault.

Before we go any further, please read the basic concepts for Key Vault.

> **NOTE**
>
> Key Vault is a central repository to store secrets programmatically. But to do so, applications and users need to first authenticate to Key Vault--that is, present a secret. In keeping with security best practices, this first secret needs to be rotated periodically.
>
> With managed service identities for Azure resources, applications that run in Azure get an identity that Azure manages automatically. This helps solve the *secret introduction problem* so that users and applications can follow best practices and not have to worry about rotating the first secret.

Open Azure Cloud Shell

Azure Cloud Shell is a free, interactive shell that you can use to run the steps in this article. Common Azure tools are preinstalled and configured in Cloud Shell for you to use with your account. Select **Copy** to copy the code, paste it in Cloud Shell, and then press Enter to run it. There are a few ways to open Cloud Shell:

Select **Try It** in the upper-right corner of a code block.	
Open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu in the upper-right corner of the Azure portal.	

Prerequisites

- On Windows:
  - Visual Studio 2017 version 15.7.3 or later with the following workloads:

        - ASP.NET and web development
        - .NET Core cross-platform development
    - .NET Core 2.1 SDK or later
- On Mac:
    - See What's New in Visual Studio for Mac.
- All platforms:
    - Git (download).
    - 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. This is available for Windows, Mac, and Linux.

Log in to Azure

To log 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.

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

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

The resource group that you just created is used throughout this article.

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 a hyphen (-).
- Resource group name.
- Location: **East US**.

```
az keyvault create --name "<YourKeyVaultName>" --resource-group "<YourResourceGroupName>" --location "East 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 be storing a SQL connection string or any other information that you need to keep securely but make 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"
```

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

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

This command shows 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. You'll need it in a later step.

Clone the repo

Clone the repo to make a local copy where you can edit the source. Run the following command:

```
git clone https://github.com/Azure-Samples/key-vault-dotnet-core-quickstart.git
```

Open and edit the solution

Edit the program.cs file to run the sample with your specific key vault name:

1. Browse to the folder key-vault-dotnet-core-quickstart.
2. Open the key-vault-dotnet-core-quickstart.sln file in Visual Studio 2017.
3. Open the Program.cs file and update the placeholder *YourKeyVaultName* with the name of the key vault that you created earlier.

This solution uses AppAuthentication and KeyVault NuGet libraries.

Run the app

From the main menu of Visual Studio 2017, select **Debug** > **Start** without debugging. When the browser appears, go to the **About** page. The value for **AppSecret** is displayed.

Publish the web application to Azure

Publish this app to Azure to see it live as a web app, and to see that you can fetch the secret value:

1. In Visual Studio, select the **key-vault-dotnet-core-quickstart** project.
2. Select **Publish** > **Start**.
3. Create a new **App Service**, and then select **Publish**.
4. Change the app name to **keyvaultdotnetcorequickstart**.
5. Select **Create**.

Enable a managed identity for the web app

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 overview 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.

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

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

NOTE

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

Assign permissions to your application to read secrets from Key Vault

Make a note of the output when you publish the application to Azure. It should be of the format:

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

Then, run this command by using the name of your key vault and the value of **PrincipalId**:

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

Now when you run the application, you should see your secret value retrieved. In the preceding command, you're giving the identity of the app service permissions to do **get** and **list** operations on your key vault.

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

Learn more about Key Vault

Quickstart: Set and retrieve a secret from Azure Key Vault by using a Node web app

5/6/2019 • 5 minutes to read • Edit Online

This quickstart shows you how to store a secret in Azure Key Vault and how to retrieve it by using a web app. Using Key Vault helps keep the information secure. To see the secret value, you would have to run this quickstart on Azure. The quickstart uses Node.js and managed identities for Azure resources. You learn how to:

- Create a key vault.
- Store a secret in the key vault.
- Retrieve a secret from the key vault.
- Create an Azure web application.
- Enable a managed identity for the web app.
- Grant the required permissions for the web application to read data from the key vault.

Before you proceed, make sure that you're familiar with the basic concepts for Key Vault.

NOTE

Key Vault is a central repository to store secrets programmatically. But to do so, applications and users need to first authenticate to Key Vault--that is, present a secret. In keeping with security best practices, this first secret needs to be rotated periodically.

With managed service identities for Azure resources, applications that run in Azure get an identity that Azure manages automatically. This helps solve the *secret introduction problem* so that users and applications can follow best practices and not have to worry about rotating the first secret.

Prerequisites

- Node.js
- Git
- Azure CLI 2.0.4 or later. This quickstart requires that you run the Azure CLI locally. Run `az --version` to find the version. If you need to install or upgrade the CLI, see Install Azure CLI 2.0.
- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin.

Log in to Azure

To log 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.

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

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

The resource group that you just created is used throughout this article.

Create a key vault

Next you create a key vault by using the resource group that you created in the previous step. Although this article uses “ContosoKeyVault” as the name, you have to use a unique name. Provide the following information:

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

```
az keyvault create --name "<YourKeyVaultName>" --resource-group "<YourResourceGroupName>" --location "East 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 be storing a SQL connection string or any other information that you need to keep securely but make available to your application. In this tutorial, the password will be called **AppSecret** and will store the value of **MySecret** in it.

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"
```

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

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

This command shows 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. You'll need it in a later step.

Clone the repo

Clone the repo to make a local copy where you can edit the source. Run the following command:

```
git clone https://github.com/Azure-Samples/key-vault-node-quickstart.git
```

Install dependencies

Run the following commands to install dependencies:

```
cd key-vault-node-quickstart
npm install
```

This project uses two Node modules: ms-rest-azure and azure-keyvault.

Publish the web app to Azure

Create an Azure App Service plan. You can store multiple web apps in this plan.

~~~
```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup
```
~~~

Next, create a web app. In the following example, replace `<app_name>` with a globally unique app name (valid characters are a-z, 0-9, and -). The runtime is set to NODE|6.9. To see all supported runtimes, run `az webapp list-runtimes`.

~~~
```
# Bash
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app_name> --runtime
"NODE|6.9" --deployment-local-git
```
~~~

When the web app has been created, the Azure CLI shows output similar to the following example:

~~~
```
{
  "availabilityState": "Normal",
  "clientAffinityEnabled": true,
  "clientCertEnabled": false,
  "cloningInfo": null,
  "containerSize": 0,
  "dailyMemoryTimeQuota": 0,
  "defaultHostName": "<app_name>.azurewebsites.net",
  "enabled": true,
  "deploymentLocalGitUrl": "https://<username>@<app_name>.scm.azurewebsites.net/<app_name>.git"
  < JSON data removed for brevity. >
}
```
~~~

Browse to your newly created web app, and you should see that it's functioning. Replace `<app_name>` with a unique app name.

~~~
```
http://<app name>.azurewebsites.net
```
~~~

The preceding command also creates a Git-enabled app that enables you to deploy to Azure from your local Git repository. The local Git repo is configured with this URL: `https://<username>@<app_name>.scm.azurewebsites.net/<app_name>.git`.

After you finish the preceding command, you can add an Azure remote to your local Git repository. Replace `<url>` with the URL of the Git repo.

~~~
```
git remote add azure <url>
```
~~~

Enable a managed identity for the web app

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 overview 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.

Run the assign-identity command to create the identity for this application:

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

This command is the equivalent of going to the portal and switching the **Identity / System assigned** setting to **On** in the web application properties.

Assign permissions to your application to read secrets from Key Vault

Make note of the output of the previous command. It should be in the format:

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

Then, run the following command by using the name of your key vault and the value of **principalId**:

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

Deploy the Node app to Azure and retrieve the secret value

Run the following command to deploy the app to Azure:

```
git push azure master
```

After this, when you browse to `https://<app_name>.azurewebsites.net`, you can see the secret value. Make sure that you replaced the name `<YourKeyVaultName>` with your vault name.

Next steps

Azure SDK for Node

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

4/25/2019 • 2 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. For more information on 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**.

     ```
     echo "Enter your email address that is used to sign in to Azure:" &&
     read upn &&
     az ad user show --upn-or-object-id $upn --query "objectId"
     ```

     ```
     $upn = Read-Host -Prompt "Enter your email address used to sign in to Azure"
     (Get-AzADUser -UserPrincipalName $upn).Id
     ```

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

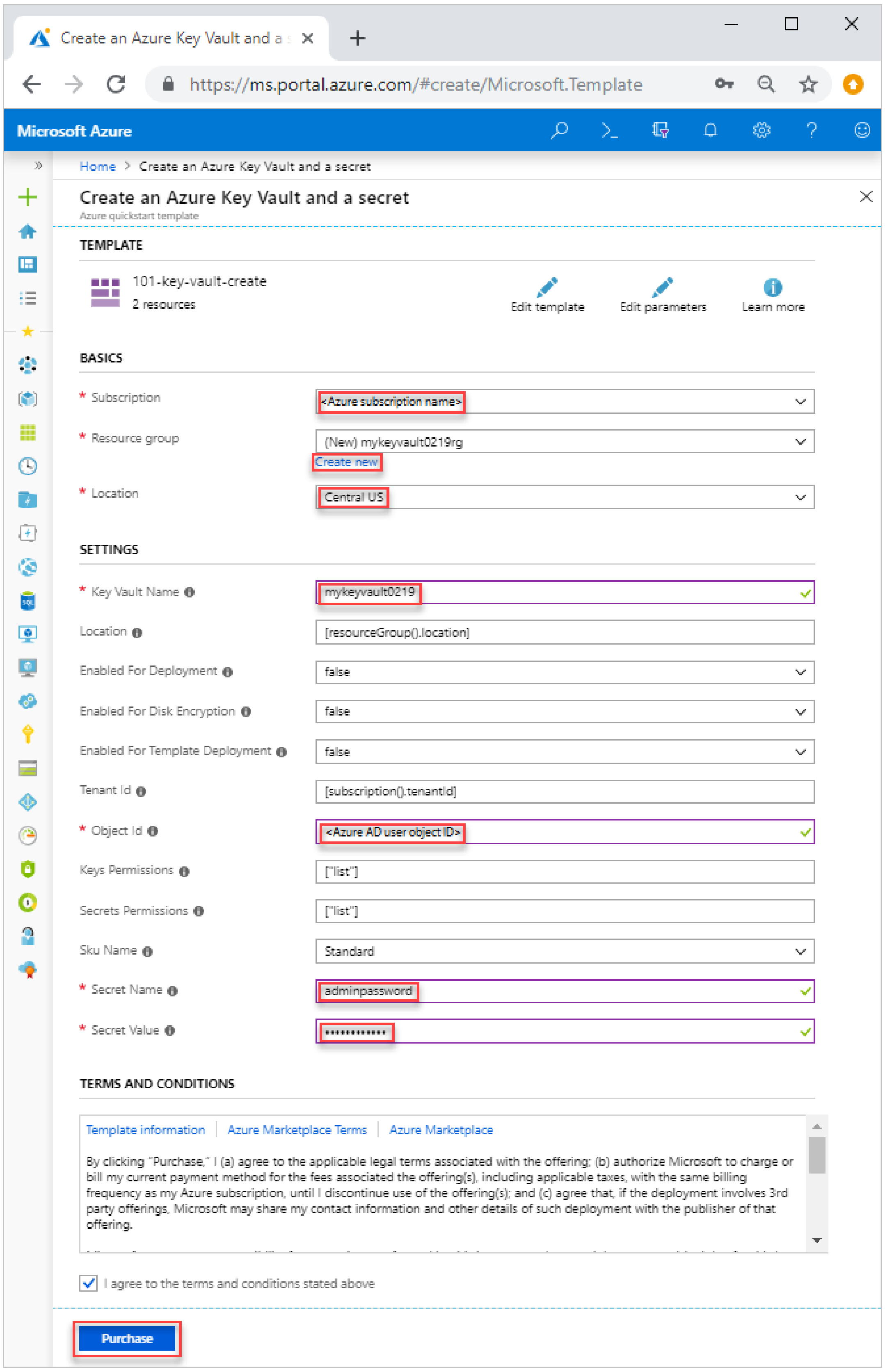
Create a vault and a secret

The template used in this quickstart is from Azure Quickstart templates. More Azure Key Vault template samples can be found here.

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.


Create an Azure Key Vault and a
https://ms.portal.azure.com/#create/Microsoft.Template
Microsoft Azure
Home > Create an Azure Key Vault and a secret

Create an Azure Key Vault and a secret
Azure quickstart template

TEMPLATE

101-key-vault-create
2 resources

Edit template | Edit parameters | Learn more

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 unique name for the key vault.
- **Tenant Id**: the template function automatically retrieve 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**.

Validate the deployment

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.

```
echo "Enter your key vault name:" &&
read keyVaultName &&
az keyvault secret list --vault-name $keyVaultName
```

```
$keyVaultName = Read-Host -Prompt "Enter your key vault name"
Get-AzKeyVaultSecret -vaultName $keyVaultName
```

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:

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az group delete --name $resourceGroupName
```

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
Remove-AzResourceGroup -Name $resourceGroupName
```

Next steps

- Azure Key Vault Home Page
- Azure Key Vault Documentation
- Azure SDK For Node
- Azure REST API Reference

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

5/6/2019 • 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.

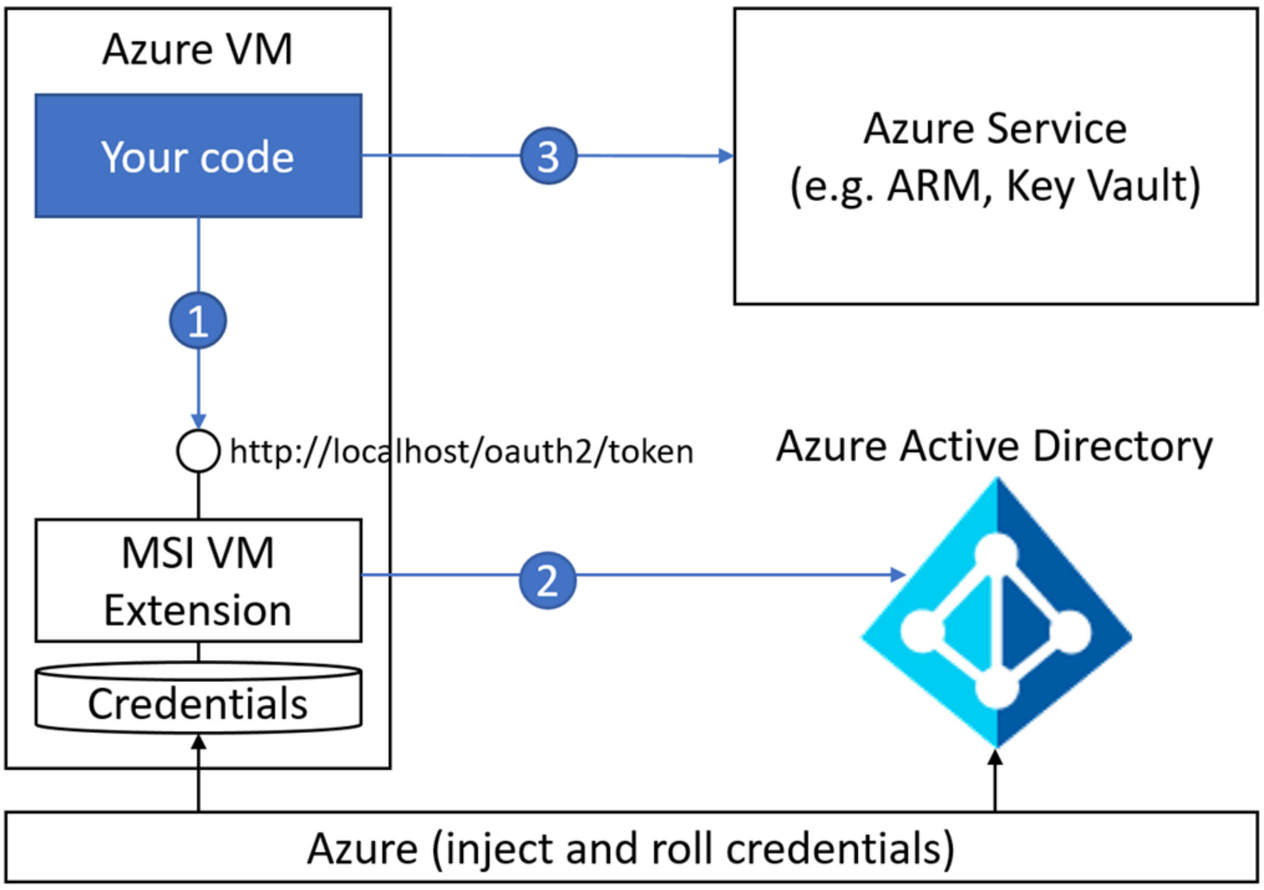
Prerequisites

- 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.


Azure VM
Your code
1
http://localhost/oauth2/token
MSI VM Extension
Credentials
3
Azure Service (e.g. ARM, Key Vault)
Azure Active Directory
2
Azure (inject and roll credentials)

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 2017, 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: How to use Azure Key Vault with Azure Linux Virtual Machine in .NET

5/6/2019 • 5 minutes to read • Edit Online

Azure Key Vault helps you to protect secrets such as API Keys, Database Connection strings needed to access your applications, services, and IT resources.

In this tutorial, you follow the necessary steps for getting a Console application to read information from Azure Key Vault by using managed identities for Azure resources. In the following you learn how to:

- Create a key vault.
- Store a secret in the key vault.
- Retrieve a secret from the key vault.
- Create an Azure 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 secrets from Key Vault

Before we go any further, read the basic concepts.

Prerequisites

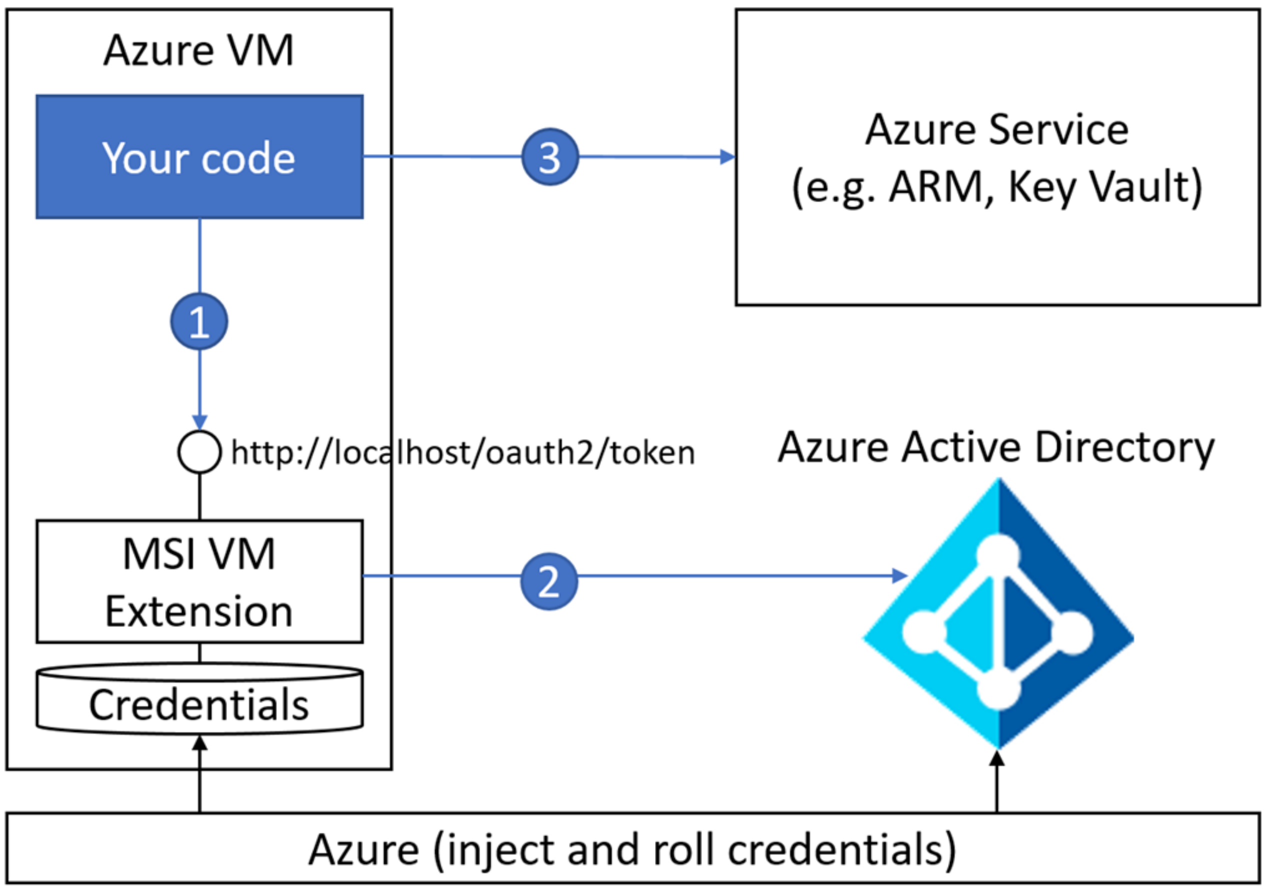
- All platforms:
  - Git (download).
  - An Azure subscription. If you don't have an Azure subscription, create a free account before you begin.
  - Azure CLI version 2.0.4 or later. This is available for Windows, Mac, and Linux.

This tutorial makes use of Managed Service Identity

What is Managed Service Identity and how does it work?

Before we go any further let's understand MSI. 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. To authenticate to Key Vault, you need a credential! A classic bootstrap problem. Through the magic of Azure and Azure AD, MSI provides a “bootstrap identity” that makes it much simpler to get things started.

Here’s how it works! When you enable MSI for an Azure service such as Virtual Machines, App Service, or Functions, Azure creates a Service Principal for the instance of the service in Azure Active Directory, and 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.

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"
```

The resource group that you just created is used throughout this article.

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 -).
- 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 be storing a SQL connection string or any other information that you need to keep securely but make 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 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 us 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 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"
}
```

Note your own `publicIpAddress` in the output from your VM. This address is used to access the VM in the next steps.

Assign identity to Virtual Machine

In this step, we're creating a system assigned identity to the virtual machine by running the following command

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

Note the systemAssignedIdentity shown below. The output of the above command would be

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

Give VM Identity permission to Key Vault

Now we can give the above created identity permission to Key Vault by running the following command

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

Sign in to the Virtual Machine

Now sign in to the Virtual Machine by using a terminal

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

Install Dot Net core on Linux

Register the Microsoft Product key as trusted. Run the following two 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 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 Core

And check .NET Version

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

Create and run Sample Dot Net App

By running the below commands, you should see "Hello World" printed to the console

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

Edit console app

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;
```

Then change the class file to contain the below code. It's a two-step process.

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

```
 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;
        }
    }
```

The above code shows you how to do operations with Azure Key Vault in an Azure Linux Virtual Machine.

Next steps

Azure Key Vault REST API

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

5/6/2019 • 5 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

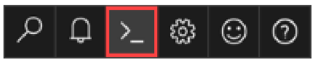
Before you go any further, make sure you understand the basic concepts about Key Vault.

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.

Open Azure Cloud Shell

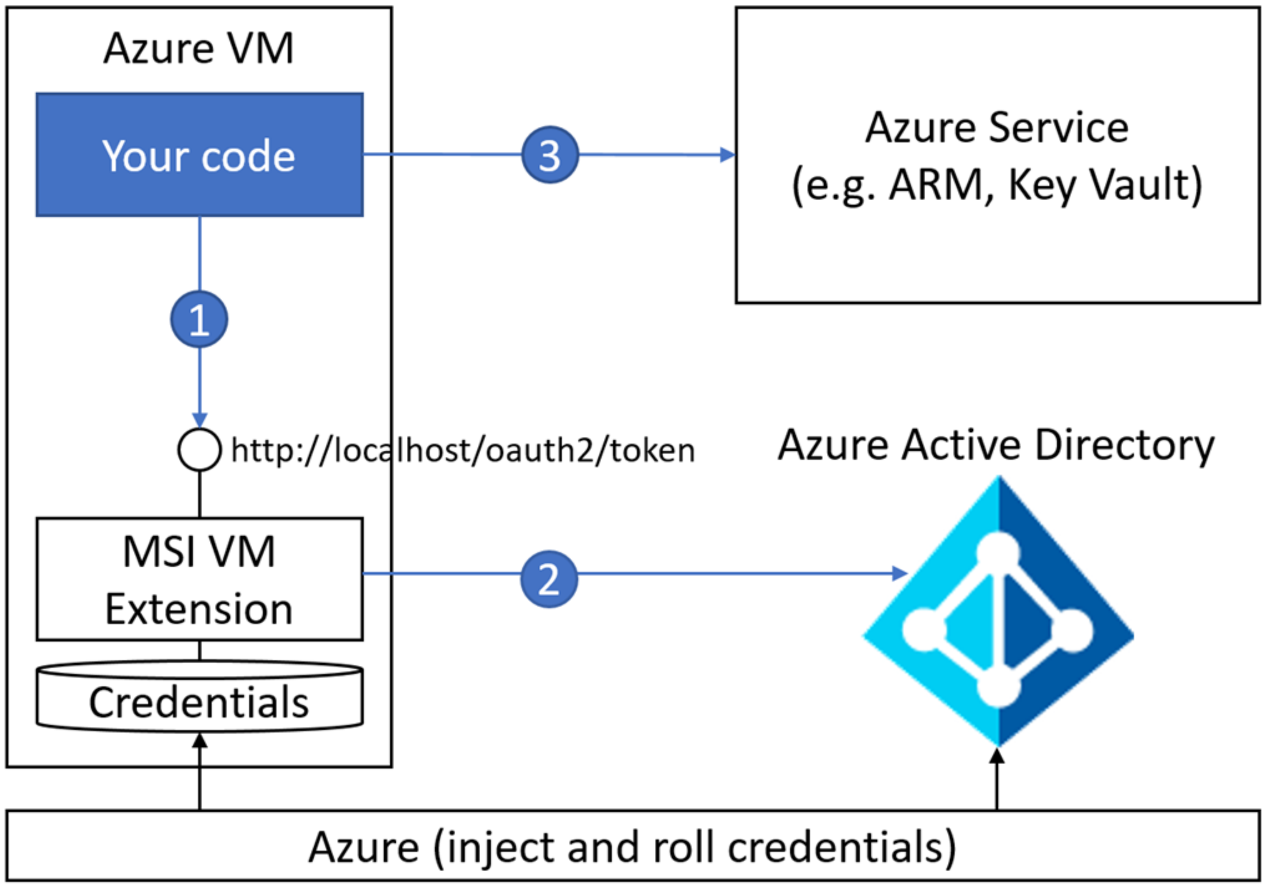
Azure Cloud Shell is a free, interactive shell that you can use to run the steps in this article. Common Azure tools are preinstalled and configured in Cloud Shell for you to use with your account. Select **Copy** to copy the code, paste it in Cloud Shell, and then press Enter to run it. There are a few ways to open Cloud Shell:

Select **Try It** in the upper-right corner of a code block.	
Open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu in the upper-right corner of the Azure portal.	

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://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"])
```

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

5/6/2019 • 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

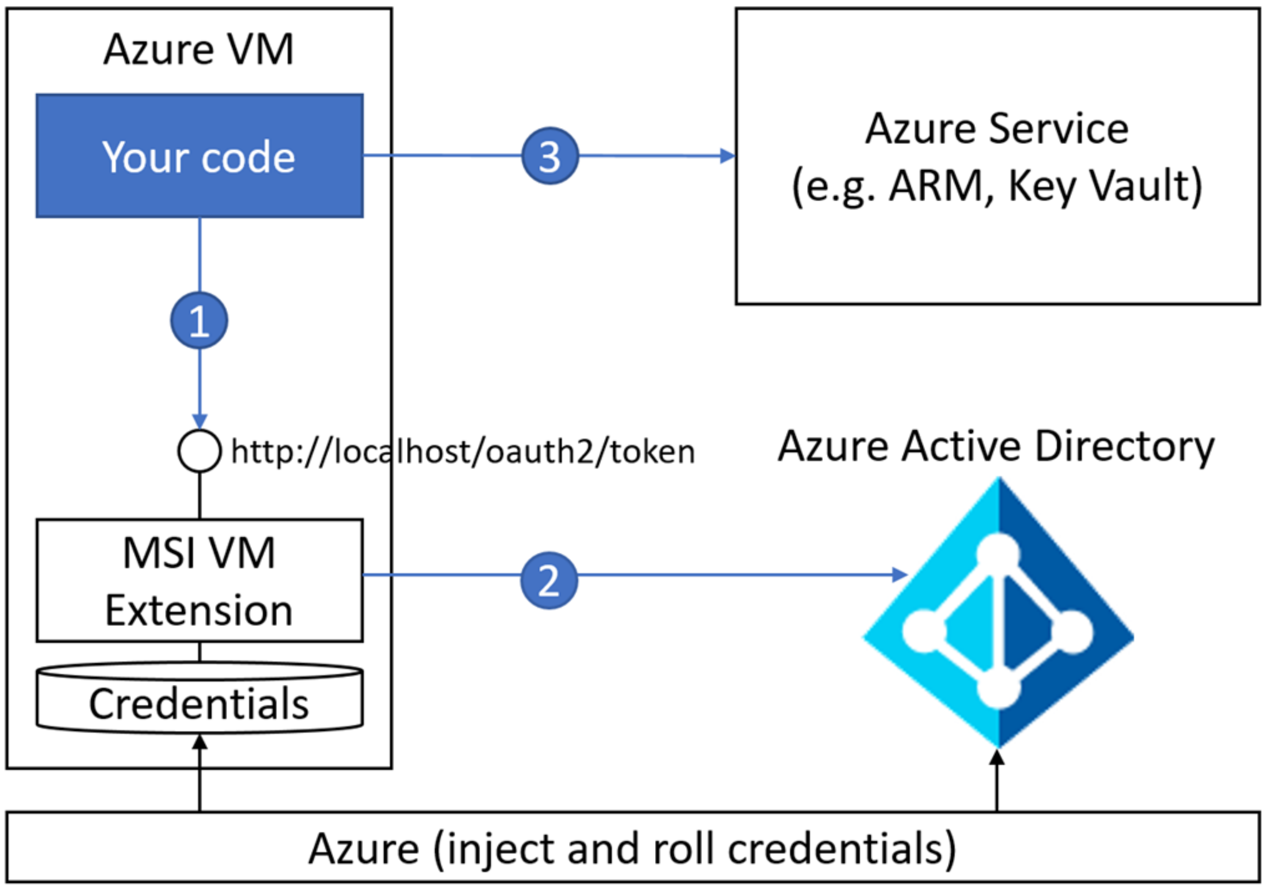
For Windows, Mac, and Linux:

- 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.


Azure VM
Your code
1
http://localhost/oauth2/token
MSI VM Extension
Credentials
3
Azure Service (e.g. ARM, Key Vault)
2
Azure Active Directory
Azure (inject and roll credentials)

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.

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 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.

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
```

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 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.

```
 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)
        {
            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

5/6/2019 • 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

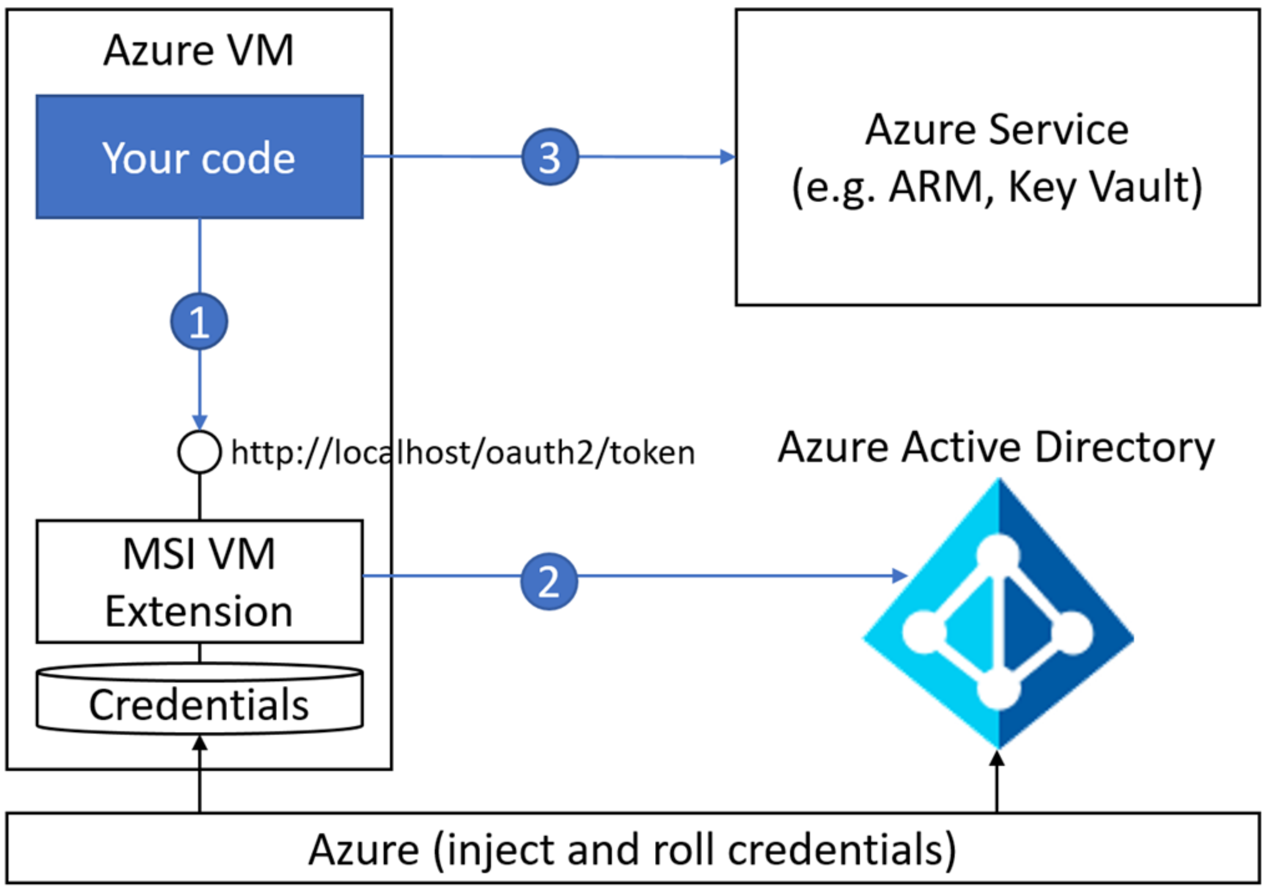
For Windows, Mac, and Linux:

- 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

Azure Key Vault managed storage account - CLI

5/6/2019 • 7 minutes to read • Edit Online

NOTE

[Azure storage integration with Azure Active Directory (Azure AD)] is Microsoft's cloud-based identity and access management service. Azure AD integration is available for the Blob and Queue services. (https://docs.microsoft.com/azure/storage/common/storage-auth-aad). We recommend using Azure AD for authentication and authorization, which provides OAuth2 token-based access to Azure storage, just like Azure Key Vault. This allows you to:

- Authenticate your client application using an application or user identity, instead of storage account credentials.
- Use an Azure AD managed identity when running on Azure. Managed identities remove the need for client authentication all together, and storing credentials in or with your application.
- Use Role Based Access Control (RBAC) for managing authorization, which is also supported by Key Vault.

An Azure storage account uses a credential that consists of an account name and a key. The key is autogenerated, and serves more as a "password" as opposed to a cryptographic key. Key Vault can manage these storage account keys, by storing them as Key Vault secrets.

Overview

The Key Vault managed storage account feature performs several management functions on your behalf:

- Lists (syncs) keys with an Azure storage account.
- Regenerates (rotates) the keys periodically.
- Manages keys for both storage accounts and Classic storage accounts.
- Key values are never returned in response to caller.

When you use the managed storage account key feature:

- **Only allow Key Vault to manage your storage account keys.** Don't attempt to manage them yourself, as you'll interfere with Key Vault's processes.
- **Don't allow storage account keys to be managed by more than one Key Vault object**.
- **Don't manually regenerate your storage account keys**. We recommend that you regenerate them via Key Vault.
- Asking Key Vault to manage your storage account can be done by a User Principal for now and not a Service Principal

The following example shows you how to allow Key Vault to manage your storage account keys.

IMPORTANT

An Azure AD tenant provides each registered application with a **service principal**, which serves as the application's identity. The service principal's Application ID is used when giving it authorization to access other Azure resources, through role-based access control (RBAC). Because Key Vault is a Microsoft application, it's pre-registered in all Azure AD tenants under the same Application ID, within each Azure cloud:

- Azure AD tenants in Azure government cloud use Application ID `7e7c393b-45d0-48b1-a35e-2905ddf8183c`.
- Azure AD tenants in Azure public cloud and all others use Application ID `cfa8b339-82a2-471a-a3c9-0fc0be7a4093`.

Prerequisites

1. Azure CLI Install Azure CLI
2. Create a Storage Account
   - Follow the steps in this document to create a storage account
   - **Naming guidance:** Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase letters only.

Step by step instructions on how to use Key Vault to manage Storage Account Keys

Conceptually the list of steps that are followed are

- We first get a (pre-existing) storage account
- We then fetch a (pre-existing) key vault
- We then add a KeyVault-managed storage account to the vault, setting Key1 as the active key, and with a regeneration period of 180 days
- Lastly we set a storage context for the specified storage account, with Key1

In the below instructions, we are assigning Key Vault as a service to have operator permissions on your storage account

NOTE

Please note that once you've set up Azure Key Vault managed storage account keys they should **NO** longer be changed except via Key Vault. Managed Storage account keys means that Key Vault would manage rotating the storage account key

IMPORTANT

An Azure AD tenant provides each registered application with a **service principal**, which serves as the application's identity. The service principal's Application ID is used when giving it authorization to access other Azure resources, through role-based access control (RBAC). Because Key Vault is a Microsoft application, it's pre-registered in all Azure AD tenants under the same Application ID, within each Azure cloud:

- Azure AD tenants in Azure government cloud use Application ID `7e7c393b-45d0-48b1-a35e-2905ddf8183c`.
- Azure AD tenants in Azure public cloud and all others use Application ID `cfa8b339-82a2-471a-a3c9-0fc0be7a4093`.

> - Currently you can use User Principal to ask Key Vault to manage a storage account and not a Service Principal

1. After creating a storage account run the following command to get the resource ID of the storage account, you want to manage

   ```
   az storage account show -n storageaccountname
   ```

   Copy ID field out of the result of the above command which looks like below

   ```
   /subscriptions/0xxxxxx-4310-48d9-b5ca-
   0xxxxxxxxxx/resourceGroups/ResourceGroup/providers/Microsoft.Storage/storageAccounts/StorageAccountName
   ```

```
    "objectId": "93c27d83-f79b-4cb2-8dd4-4aa716542e74"
```

2. 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."

   ```
   az role assignment create --role "Storage Account Key Operator Service Role"  --assignee-object-id
   <ObjectIdOfKeyVault> --scope 93c27d83-f79b-4cb2-8dd4-4aa716542e74
   ```

   '93c27d83-f79b-4cb2-8dd4-4aa716542e74' is the Object ID for Key Vault in Public Cloud. To get the Object ID for Key Vault in National clouds see the Important section above

3. Create a Key Vault Managed Storage Account.

   Below, we are setting a regeneration period of 90 days. After 90 days, Key Vault will regenerate 'key1' and swap the active key from 'key2' to 'key1'. It will mark Key1 as the active key now.

   ```
   az keyvault storage add --vault-name <YourVaultName> -n <StorageAccountName> --active-key-name key1 --
   auto-regenerate-key --regeneration-period P90D --resource-id <Id-of-storage-account>
   ```

Step by step instructions on how to use Key Vault to create and generate SAS tokens

You can also ask Key Vault to generate SAS (Shared Access Signature) tokens. A shared access signature provides delegated access to resources in your storage account. With a SAS, you can grant clients access to resources in your storage account, without sharing your account keys. This is the key point of using shared access signatures in your applications--a SAS is a secure way to share your storage resources without compromising your account keys.

Once you've completed the steps listed above you can run the following commands to ask Key Vault to generate SAS tokens for you.

The list of things that would be accomplished in the below steps are

- Sets an account SAS definition named `<YourSASDefinitionName>` on a KeyVault-managed storage account `<YourStorageAccountName>` in your vault `<VaultName>`.
- Creates an account SAS token for services Blob, File, Table and Queue, for resource types Service, Container and Object, with all permissions, over https and with the specified start and end dates
- Sets a KeyVault-managed storage SAS definition in the vault, with the template uri as the SAS token created above, of SAS type 'account' and valid for N days
- Retrieves the actual access token from the KeyVault secret corresponding to the SAS definition

1. In this step we will create a SAS Definition. Once this SAS Definition is created, you can ask Key Vault to generate more SAS tokens for you. This operation requires the storage/setsas permission.

```
$sastoken = az storage account generate-sas --expiry 2020-01-01 --permissions rw --resource-types sco --
services bfqt --https-only --account-name storageacct --account-key 00000000
```

You can see more help about the operation above here

When this operation runs successfully, you should see output similar to as shown below. Copy that

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

1. In this step we will use the output ($sasToken) generated above to create a SAS Definition. For more documentation read here

```
az keyvault storage sas-definition create --vault-name <YourVaultName> --account-name <YourStorageAccountName>
-n <NameOfSasDefinitionYouWantToGive> --validity-period P2D --sas-type account --template-uri $sastoken
```

NOTE

In the case that the user does not have permissions to the storage account, we first get the Object-Id of the user

```
az ad user show --upn-or-object-id "developer@contoso.com"

az keyvault set-policy --name <YourVaultName> --object-id <ObjectId> --storage-permissions backup delete list
regeneratekey recover     purge restore set setsas update
```

Fetch SAS tokens in code

In this section we will discuss how you can do operations on your storage account by fetching SAS tokens from Key Vault

In the below section, we demonstrate how to fetch SAS tokens once a SAS definition is created as shown above.

NOTE

There are 3 ways to authenticate to Key Vault as you can read in the basic concepts

- Using Managed Service Identity (Highly recommended)
- Using Service Principal and certificate
- Using Service Principal and password (NOT recommended)

```
// Once you have a security token from one of the above methods, then create KeyVaultClient with vault
credentials
var kv = new KeyVaultClient(new KeyVaultClient.AuthenticationCallback(securityToken));

// Get a SAS token for our storage from Key Vault. SecretUri is of the format
https://<VaultName>.vault.azure.net/secrets/<ExamplePassword>
var sasToken = await kv.GetSecretAsync("SecretUri");

// Create new storage credentials using the SAS 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 SAS token is about to expire, then you would fetch the SAS token again from Key Vault and update the code

```
// If your SAS token is about to expire, get the SAS Token again from Key Vault and update it.
sasToken = await kv.GetSecretAsync("SecretUri");
accountSasCredential.UpdateSASToken(sasToken);
```

Relevant Azure CLI commands

Azure CLI Storage commands

See also

- About keys, secrets, and certificates
- Key Vault Team Blog

Azure Key Vault managed storage account - PowerShell

3/5/2019 • 5 minutes to read • Edit Online

NOTE

Azure storage integration with Azure Active Directory (Azure AD) is now in preview. We recommend using Azure AD for authentication and authorization, which provides OAuth2 token-based access to Azure storage, just like Azure Key Vault. This allows you to:

- Authenticate your client application using an application or user identity, instead of storage account credentials.
- Use an Azure AD managed identity when running on Azure. Managed identities remove the need for client authentication all together, and storing credentials in or with your application.
- Use Role Based Access Control (RBAC) for managing 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.

An Azure storage account uses a credential that consists of an account name and a key. The key is autogenerated, and serves more as a "password" as opposed to a cryptographic key. Key Vault can manage these storage account keys, by storing them as Key Vault secrets.

Overview

The Key Vault managed storage account feature performs several management functions on your behalf:

- Lists (syncs) keys with an Azure storage account.
- Regenerates (rotates) the keys periodically.
- Manages keys for both storage accounts and Classic storage accounts.
- Key values are never returned in response to caller.

When you use the managed storage account key feature:

- **Only allow Key Vault to manage your storage account keys.** Don't attempt to manage them yourself, as you'll interfere with Key Vault's processes.
- **Don't allow storage account keys to be managed by more than one Key Vault object**.
- **Don't manually regenerate your storage account keys**. We recommend that you regenerate them via Key Vault.

The following example shows you how to allow Key Vault to manage your storage account keys.

Authorize Key Vault to access to your storage account

IMPORTANT

An Azure AD tenant provides each registered application with a **service principal**, which serves as the application's identity. The service principal's Application ID is used when giving it authorization to access other Azure resources, through role-based access control (RBAC). Because Key Vault is a Microsoft application, it's pre-registered in all Azure AD tenants under the same Application ID, within each Azure cloud:

- Azure AD tenants in Azure government cloud use Application ID `7e7c393b-45d0-48b1-a35e-2905ddf8183c`.
- Azure AD tenants in Azure public cloud and all others use Application ID `cfa8b339-82a2-471a-a3c9-0fc0be7a4093`.

Before Key Vault can access and manage your storage account keys, you must authorize its access 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 following steps. Be sure to update the `$resourceGroupName`, `$storageAccountName`, `$storageAccountKey`, and `$keyVaultName` variables before you run the script:

```
# TODO: Update with the resource group where your storage account resides, your storage account name, the name
of your active storage account key, and your Key Vault instance name
$resourceGroupName = "rgContoso"
$storageAccountName = "sacontoso"
$storageAccountKey = "key1"
$keyVaultName = "kvContoso"
$keyVaultSpAppId = "cfa8b339-82a2-471a-a3c9-0fc0be7a4093" # See "IMPORTANT" block above for information on Key
Vault Application IDs

# Authenticate your PowerShell session with Azure AD, for use with Azure Resource Manager cmdlets
$azureProfile = Connect-AzAccount

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

# 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/03f0blll-ce69-483a-a092-
d06ea46dfb8z/resourceGroups/rgContoso/providers/Microsoft.Storage/storageAccounts/sacontoso/providers/Microsoft
.Authorization/roleAssignments/189cblll-12fb-406e-8699-4eef8b2b9ecz
Scope              : /subscriptions/03f0blll-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

TIP

Just as Azure AD provides a **service principal** for an application's identity, a **user principal** is provided for a user's identity. The user principal can then be given authorization to access Key Vault, through Key Vault access policy permissions.

Using the same PowerShell session, update the Key Vault access policy for managed storage accounts. This step applies storage account permissions to your user account, ensuring that you can access the managed storage account features:

```
# Give your user principal access to all storage account permissions, on your Key Vault instance

Set-AzKeyVaultAccessPolicy -VaultName $keyVaultName -UserPrincipalName $azureProfile.Context.Account.Id -
PermissionsToStorage get, list, listsas, delete, set, update, regeneratekey, 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

Using the same PowerShell session, 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/03f0blll-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 set a regeneration period. In the following example, we set a regeneration period of three days. After three days, Key Vault will regenerate 'key1' 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/03f0blll-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                :
```

Next steps

- Managed storage account key samples
- About keys, secrets, and certificates
- Key Vault PowerShell reference

How to use managed identities with Azure Container Instances

3/21/2019 • 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 Azure Key Vault and use the managed identity to access 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 are only supported on Linux container instances.

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.

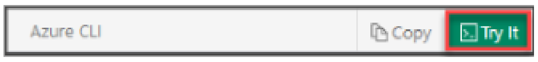
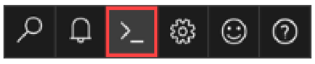
Use a managed identity

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.

Open Azure Cloud Shell

Azure Cloud Shell is a free, interactive shell that you can use to run the steps in this article. Common Azure tools are preinstalled and configured in Cloud Shell for you to use with your account. Select **Copy** to copy the code, paste it in Cloud Shell, and then press Enter to run it. There are a few ways to open Cloud Shell:

Select **Try It** in the upper-right corner of a code block.	
Open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu in the upper-right corner of the Azure portal.	

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 Ubuntu Server. This example provides a single-container group that you can use to interactively access other Azure services. 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 microsoft/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/danlep1018/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 to access the Key Vault within the running container instance. For this example, 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%2F' -H Metadata:true -s
```

Output:

```
{"access_token":"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N
5SEpsWSIsImtpZCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N5SEpsWSJ9......xxxxxxxxxxxxxxxxx","refresh_token":"","expires_i
n":"28799","expires_on":"1539927532","not_before":"1539898432","resource":"https://vault.azure.net/","token_typ
e":"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/...*):

```
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/xxxxxxxxxxx
xxxxxxxxx","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 Ubuntu Server. This example provides a single-container group that you can use to interactively access other Azure services. The `--assign-identity` parameter with no additional value enables a system-assigned managed identity on 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 microsoft/azure-cli --assign-
identity --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. For this example, 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%2F' -H Metadata:true -s
```

Output:

```
{"access_token":"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N
5SEpsWSIsImtpZCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N5SEpsWSJ9......xxxxxxxxxxxxxxxxx","refresh_token":"","expires_i
n":"28799","expires_on":"1539927532","not_before":"1539898432","resource":"https://vault.azure.net/","token_typ
e":"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/...*):

```
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/xxxxxxxxxxx
xxxxxxxxx","attributes":
{"enabled":true,"created":1539965967,"updated":1539965967,"recoveryLevel":"Purgeable"},"tags":{"file-
encoding":"utf-8"}}
```

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 an `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/userAss
ignedIdentities/{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 an `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

5/9/2019 • 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.

Deploy key vaults and secrets

To access a key vault during template deployment, set `enabledForTemplateDeployment` on the key vault to `true`.

The following Azure CLI and Azure PowerShell samples show how to create the key vault, and add a secret.

```
az group create --name $resourceGroupName --location $location
az keyvault create \
  --name $keyVaultName \
  --resource-group $resourceGroupName \
  --location $location \
  --enabled-for-template-deployment true
az keyvault secret set --vault-name $keyVaultName --name "ExamplePassword" --value "hVFkk965BuUv"
```

```
New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzKeyVault `
  -VaultName $keyVaultName `
  -resourceGroupName $resourceGroupName `
  -Location $location `
  -EnabledForTemplateDeployment
$secretvalue = ConvertTo-SecureString 'hVFkk965BuUv' -AsPlainText -Force
$secret = Set-AzKeyVaultSecret -VaultName $keyVaultName -Name 'ExamplePassword' -SecretValue $secretvalue
```

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:

```
az keyvault set-policy \
  --upn $userPrincipalName \
  --name $keyVaultName \
  --secret-permissions set delete get list
```

```
$userPrincipalName = "<Email Address of the deployment operator>"

Set-AzKeyVaultAccessPolicy `
  -VaultName $keyVaultName `
  -UserPrincipalName $userPrincipalName `
  -PermissionsToSecrets 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 Node.js

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 of the user who needs to deploy the templates.

2. Create the new role using the JSON file:

```
az role definition create --role-definition "<PathToRoleFile>"
az role assignment create \
  --role "Key Vault resource manager template deployment operator" \
  --assignee $userPrincipalName \
  --resource-group $resourceGroupName
```

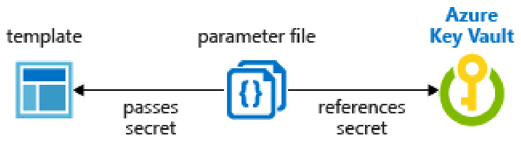
```
New-AzRoleDefinition -InputFile "<PathToRoleFile>"
New-AzRoleAssignment `
  -ResourceGroupName $resourceGroupName `
  -RoleDefinitionName "Key Vault resource manager template deployment operator" `
  -SignInName $userPrincipalName
```

   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": [
    {
      "name": "[parameters('sqlServerName')]",
      "type": "Microsoft.Sql/servers",
      "apiVersion": "2015-05-01-preview",
      "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": "exampleadmin"
        },
        "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:

For Azure CLI, use:

```
az group create --name $resourceGroupName --location $location
az group deployment create \
    --resource-group $resourceGroupName \
    --template-uri <The Template File URI> \
    --parameters <The Parameter File>
```

For PowerShell, use:

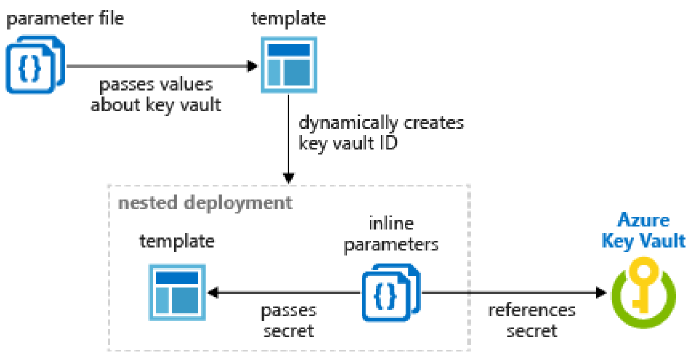
```
New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment `
  -ResourceGroupName $resourceGroupName `
  -TemplateUri <The Template File URI> `
  -TemplateParameterFile <The Parameter File>
```

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 linked 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."
            }
        },
        "_artifactsLocation": {
            "type": "string",
            "metadata": {
                "description": "The base URI where artifacts required by this template are located. When the
template is deployed using the accompanying scripts, a private location in the subscription will be used and
this value will be automatically generated."
            },
            "defaultValue": "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-
key-vault-use-dynamic-id/"
        },
        "_artifactsLocationSasToken": {
            "type": "securestring",
            "metadata": {
                "description": "The sasToken required to access _artifactsLocation.  When the template is
deployed using the accompanying scripts, a sasToken will be automatically generated."
            },
            "defaultValue": ""
        }
    },
    "resources": [
        {
            "apiVersion": "2018-05-01",
            "name": "dynamicSecret",
            "type": "Microsoft.Resources/deployments",
            "properties": {
                "mode": "Incremental",
                "templateLink": {
                    "contentVersion": "1.0.0.0",
                    "uri": "[uri(parameters('_artifactsLocation'), concat('./nested/sqlserver.json',
parameters('_artifactsLocationSasToken')))]"
                },
                "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": {
        "sqlFQDN": {
            "type": "string",
            "value": "[reference('dynamicSecret').outputs.sqlFQDN.value]"
        }
    }
}
```

Deploy the preceding template, and provide values for the parameters. You can use the example template from GitHub, but you must provide parameter values for your environment.

For Azure CLI, use:

```
az group create --name $resourceGroupName --location $location
az group deployment create \
    --resource-group $resourceGroupName \
    --template-uri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-key-vault-
use-dynamic-id/azuredeploy.json \
    --parameters vaultName=$keyVaultName vaultResourceGroupName=examplegroup secretName=examplesecret
```

For PowerShell, use:

```
New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment `
  -ResourceGroupName $resourceGroupName `
  -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-key-vault-use-
dynamic-id/azuredeploy.json `
  -vaultName $keyVaultName -vaultResourceGroupName $keyVaultResourceGroupName -secretName $secretName
```

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

5/13/2019 • 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.

IMPORTANT

The examples in this article are provided for illustration purposes only. They're not intended for production use.

This article walks through:

- An example of using Azure Key Vault to store a secret. In this article, the secret stored is the Azure storage account key accessed by an application.
- How to implement a scheduled rotation of that storage account key.
- How to monitor the key vault audit logs and raise alerts when unexpected requests are made.

NOTE

This article doesn't explain in detail the initial setup of your key vault. For this information, see What is Azure Key Vault?. For cross-platform command-line interface instructions, see Manage Key Vault using the Azure CLI.

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.

Set up Key Vault

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 URI:

```
Get-AzKeyVaultSecret –VaultName <vaultName>
```

Set up the application

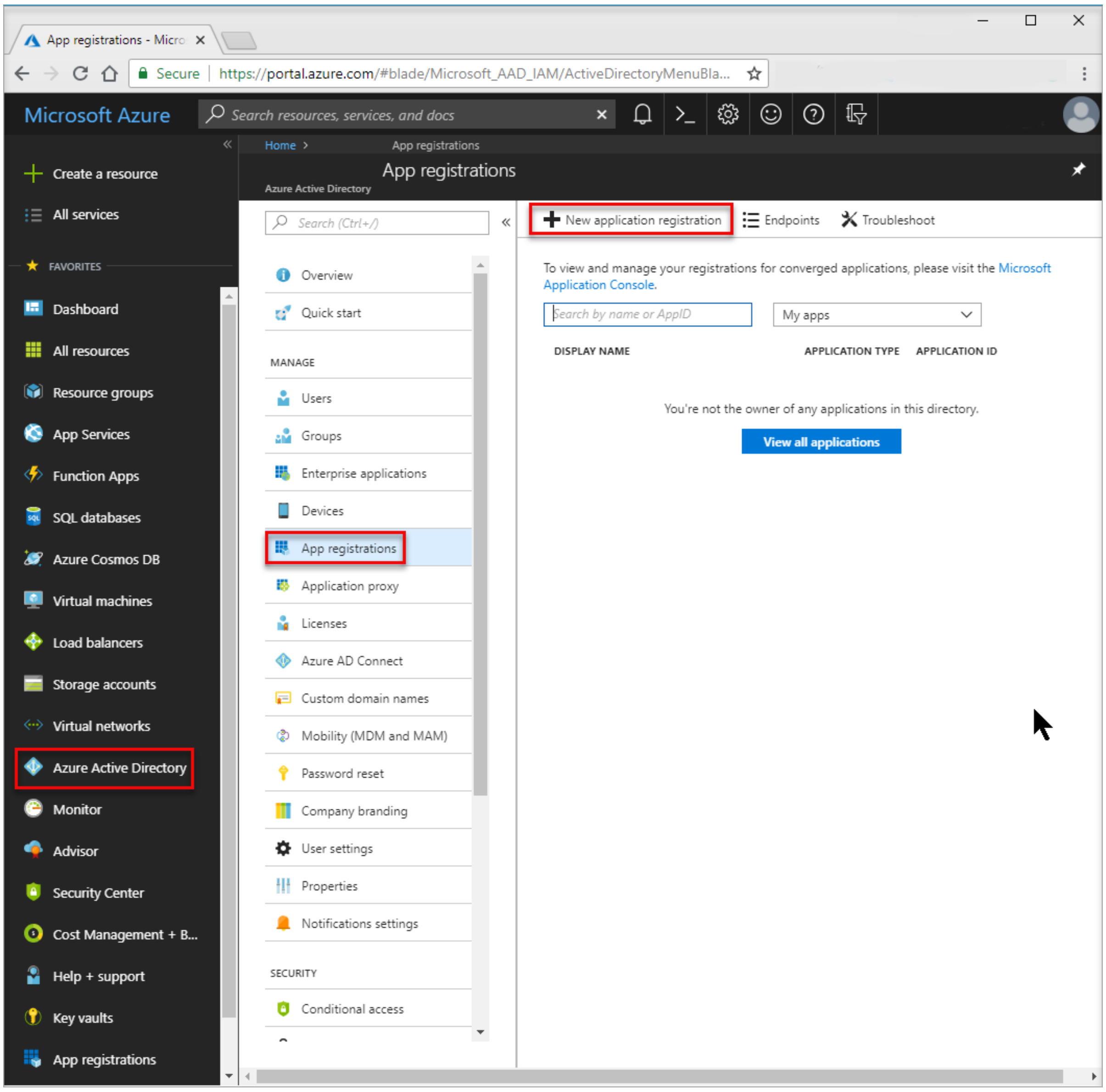
Now that you have a secret stored, you can use code to retrieve and use it after performing a few more steps.

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.

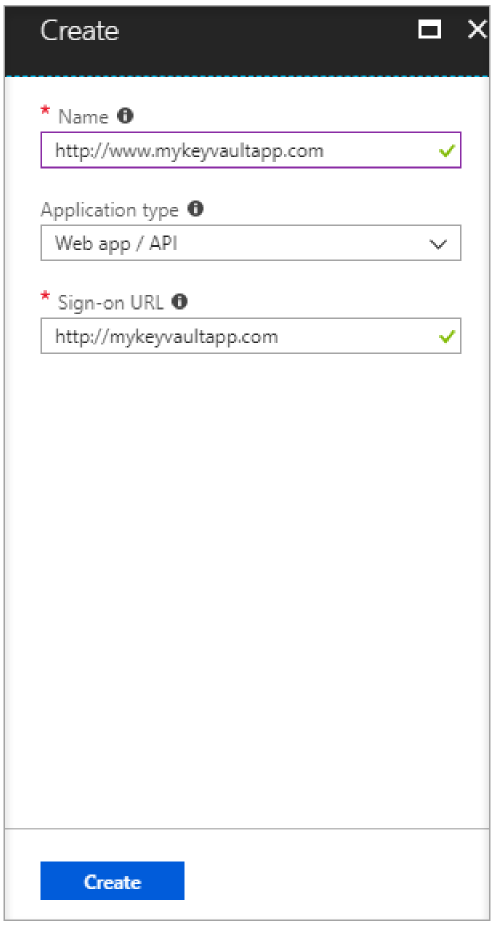
1. Open **Azure Active Directory**.
2. Select **App registrations**.
3. Select **New application registration** to add an application to Azure Active Directory.



To view and manage your registrations for converged applications, please visit the Microsoft Application Console.

You're not the owner of any applications in this directory.

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.



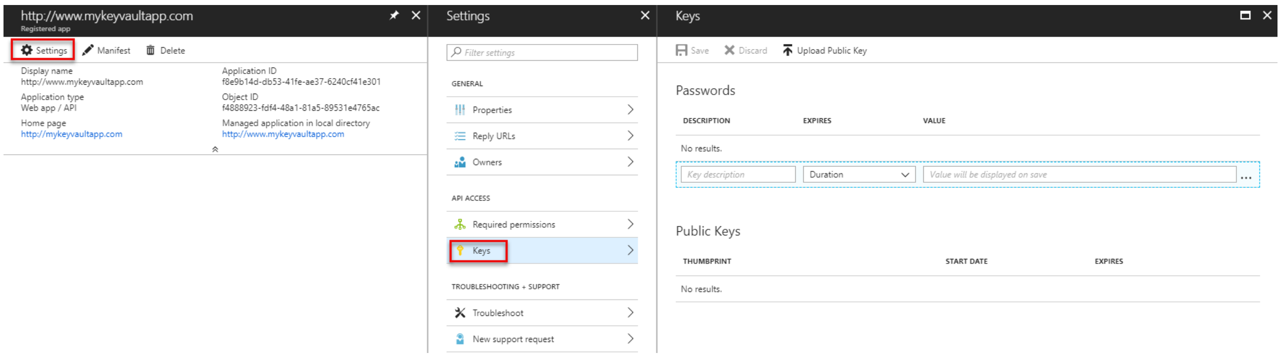
Name: http://www.mykeyvaultapp.com

Application type: Web app / API

Sign-on URL: http://mykeyvaultapp.com

5. 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.



Display name
http://www.mykeyvaultapp.com

Application type
Web app / API

Home page
http://mykeyvaultapp.com

Application ID
f8e9b14d-db53-41fe-ae37-6240cf41e301

Object ID
f4888923-fdf4-48a1-81a5-89531e4765ac

Managed application in local directory
http://www.mykeyvaultapp.com

Passwords

DESCRIPTION	EXPIRES	VALUE
No results.		
Key description	Duration	Value will be displayed on save

Public Keys

THUMBPRINT	START DATE	EXPIRES
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 URI 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.

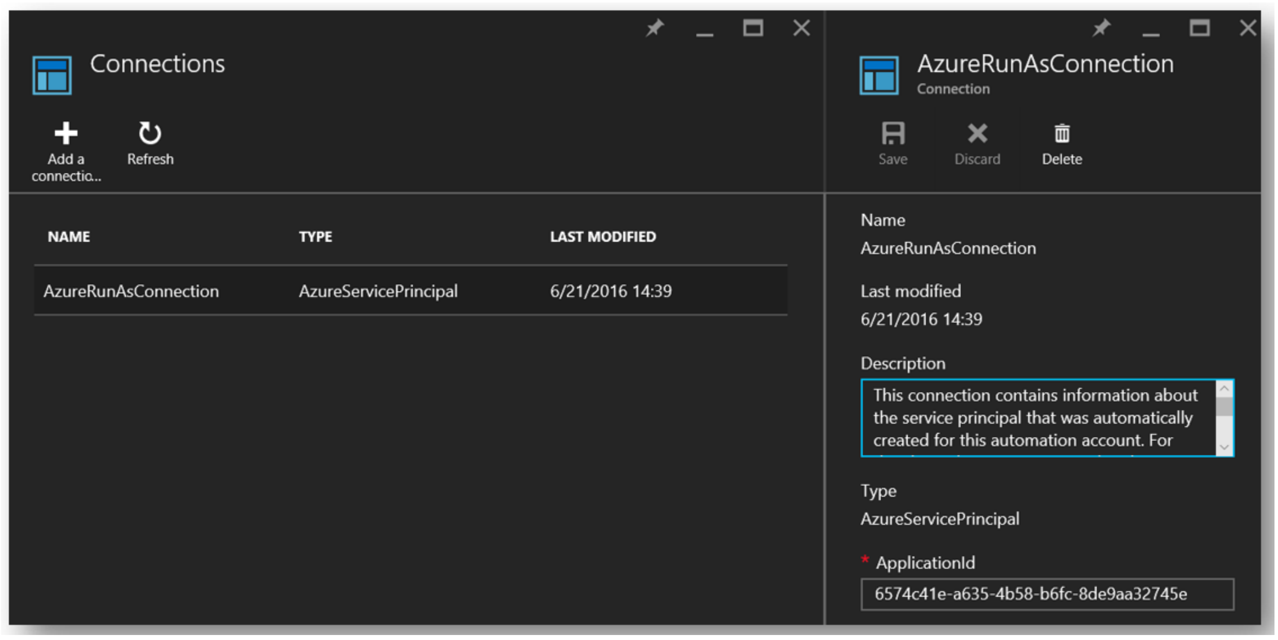
You are now ready to set up a rotation strategy for the values you store as Key Vault secrets. Secrets can be rotated in several ways:

- 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.



NAME	TYPE	LAST MODIFIED
AzureRunAsConnection	AzureServicePrincipal	6/21/2016 14:39

Name
AzureRunAsConnection

Last modified
6/21/2016 14:39

Description
This connection contains information about the service principal that was automatically created for this automation account. For

Type
AzureServicePrincipal

* ApplicationId
6574c41e-a635-4b58-b6fc-8de9aa32745e

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
```

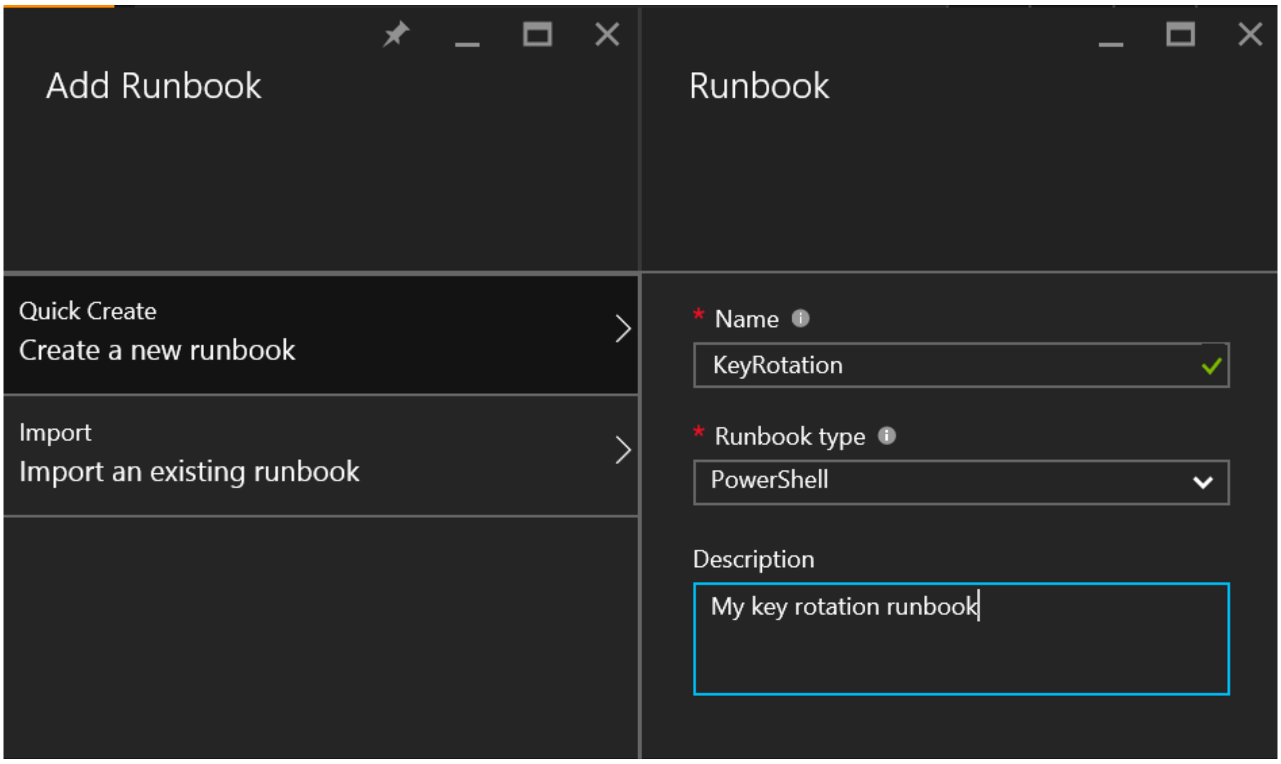
NOTE

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**.


Add Runbook

Quick Create
Create a new runbook

Import
Import an existing runbook

Runbook

* Name
KeyRotation

* Runbook type
PowerShell

Description
My key rotation runbook

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.

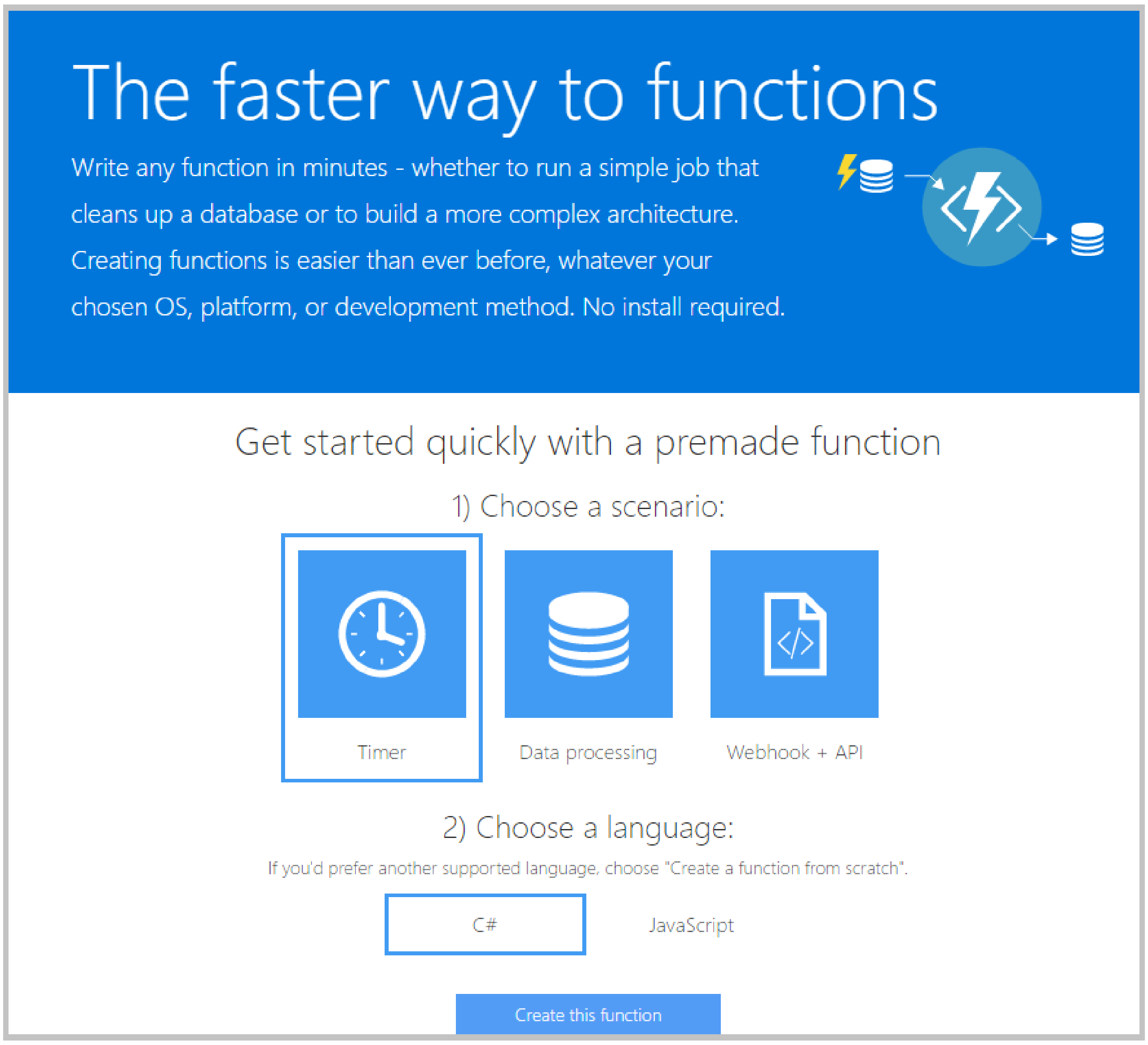
The next step is to create an Azure Service Bus queue. This queue is where key-vault audit logs are pushed. When the audit-log messages are in the queue, the logic app picks them up and acts on them. Create a Service Bus instance with the following steps:

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 txt = inputBlob.ReadToEnd();

        if(!string.IsNullOrEmpty(txt))
        {
            dtPrev = DateTime.Parse(txt);
            log.Verbose($"SyncPoint: {dtPrev.ToString("O")}");
        }
        else
        {
            dtPrev = DateTime.UtcNow;
            log.Verbose($"Sync point file didnt have a date. Setting to now.");
        }
    }

    var now = DateTime.UtcNow;

    string blobPrefix = "insights-logs-
auditevent/resourceId=/SUBSCRIPTIONS/<SUBSCRIPTION_ID>/RESOURCEGROUPS/<RESOURCE_GROUP_NAME>/PROVIDERS/MICROSOF
T.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;
}
```

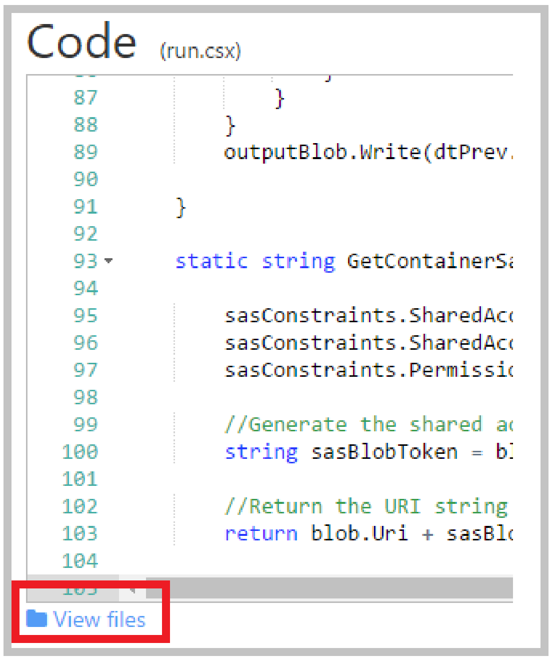
> **NOTE**
>
> Change the variables in the preceding code to point to your storage account where the key vault logs are written, to the Service Bus instance you created earlier, and to the specific path to the key-vault storage logs.

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**.



```
            }
        }
        outputBlob.Write(dtPrev.

    }

    static string GetContainerSa

        sasConstraints.SharedAcc
        sasConstraints.SharedAcc
        sasConstraints.Permissio

        //Generate the shared ac
        string sasBlobToken = bl

        //Return the URI string
        return blob.Uri + sasBl
```

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*.

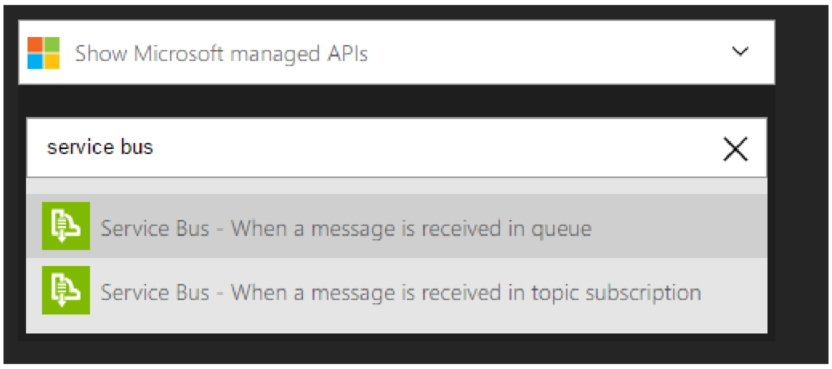
The function is now ready. Make sure to switch back to the **Develop** tab and save the code. Check the output window for any compilation errors and correct them as needed. If the code compiles, then the code should now be checking the key vault logs every minute and pushing any new events into the defined 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.

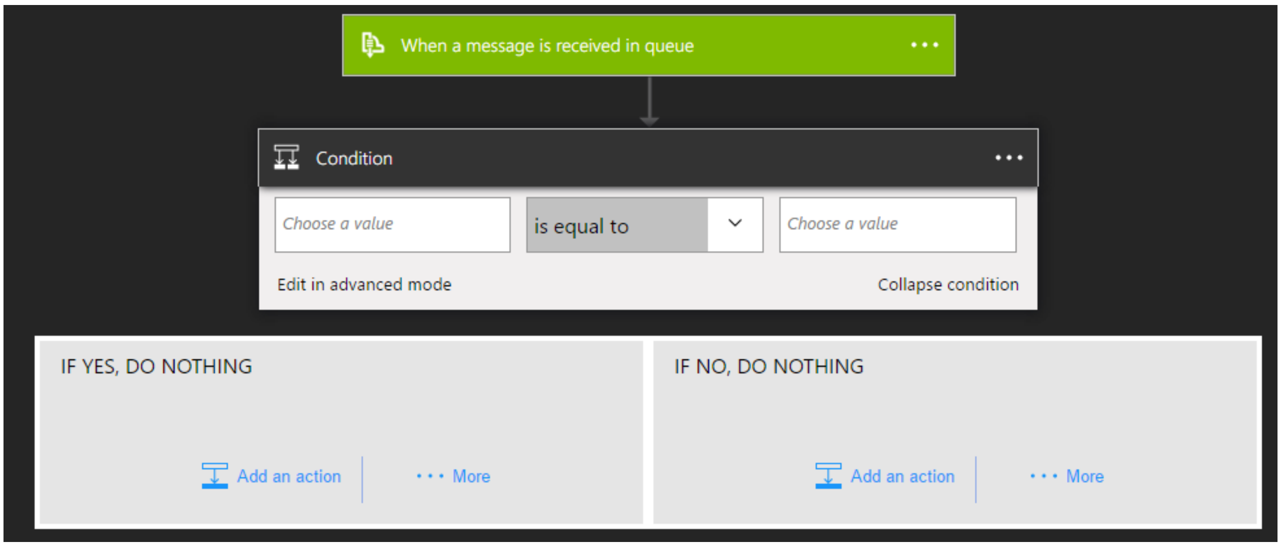

Show Microsoft managed APIs
service bus
Service Bus - When a message is received in queue
Service Bus - When a message is received in topic subscription

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 security

4/25/2019 • 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

Common security attributes for Azure Key Vault

4/25/2019 • 2 minutes to read • Edit Online

Security is integrated into every aspect of an Azure service. This article documents the common security attributes built into Azure Key Vault.

A security attribute is a quality or feature of an Azure service that contributes to the service's ability to prevent, detect, and respond to security vulnerabilities.

Security attributes are categorized as:

- Preventative
- Network segmentation
- Detection
- Identity and access management support
- Audit trail
- Access controls (if used)
- Configuration management (if used)

In each category, we identify if an attribute is used or not (yes/no). For some services, an attribute may not be applicable and is shown as N/A. A note or a link to more information about an attribute may also be provided.

Preventative

SECURITY ATTRIBUTE	YES/NO	NOTES
Encryption at rest: • Server-side encryption • Server-side encryption with customer-managed keys • Other encryption features (such as client-side, always encrypted, etc.)	Yes	All objects are encrypted.
Encryption in transit: • Express route encryption • In VNet encryption • VNet-VNet encryption	Yes	All communication is via encrypted API calls
Encryption key handling (CMK, BYOK, etc.)	Yes	The customer controls all keys in their Key Vault. When hardware security module (HSM) backed keys are specified, a FIPS Level 2 HSM protects the key, certificate, or secret.
Column level encryption (Azure Data Services)	N/A	
API calls encrypted	Yes	Using HTTPS.

Network Segmentation

SECURITY ATTRIBUTE	YES/NO	NOTES
Service endpoint support	Yes	Using Virtual Network (VNet) service endpoints.
VNet injection support	No	
Network isolation and firewalling support	Yes	Using VNet firewall rules.
Forced tunneling support	No	

Detection

SECURITY ATTRIBUTE	YES/NO	NOTES
Azure monitoring support (Log analytics, App insights, etc.)	Yes	Using Log Analytics.

Identity and access management

SECURITY ATTRIBUTE	YES/NO	NOTES
Authentication	Yes	Authentication is through Azure Active Directory.
Authorization	Yes	Using Key Vault Access Policy.

Audit Trail

SECURITY ATTRIBUTE	YES/NO	NOTES
Control/Management plane Logging and Audit	Yes	Using Log Analytics.
Data plane logging and audit	Yes	Using Log Analytics.

Access controls

SECURITY ATTRIBUTE	YES/NO	NOTES
Control/Management plane access controls	Yes	Azure Resource Manager Role-Based Access Control (RBAC)
Data plane access controls (At every service level)	Yes	Key Vault Access Policy

Azure Key Vault security worlds and geographic boundaries

4/25/2019 • 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

4/25/2019 • 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 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:

- **SSL certificate**: Used for 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 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 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 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: back up, 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 (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 URIs.

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 (URIs 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.

What is Azure Key Vault?

4/25/2019 • 6 minutes to read • Edit Online

Cloud applications and services use cryptographic keys and secrets to help keep information secure. Azure Key Vault safeguards these keys and secrets. When you use Key Vault, you can encrypt authentication keys, storage account keys, data encryption keys, .pfx files, and passwords by using keys that are protected by hardware security modules (HSMs).

Key Vault helps solve the following problems:

- **Secret management**: Securely store and tightly control access to tokens, passwords, certificates, API keys, and other secrets.
- **Key management**: Create and control encryption keys that encrypt your data.
- **Certificate management**: Provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with Azure and your internal connected resources.
- **Store secrets backed by HSMs**: Use either software or FIPS 140-2 Level 2 validated HSMs to help protect secrets and keys.

Basic concepts

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 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.

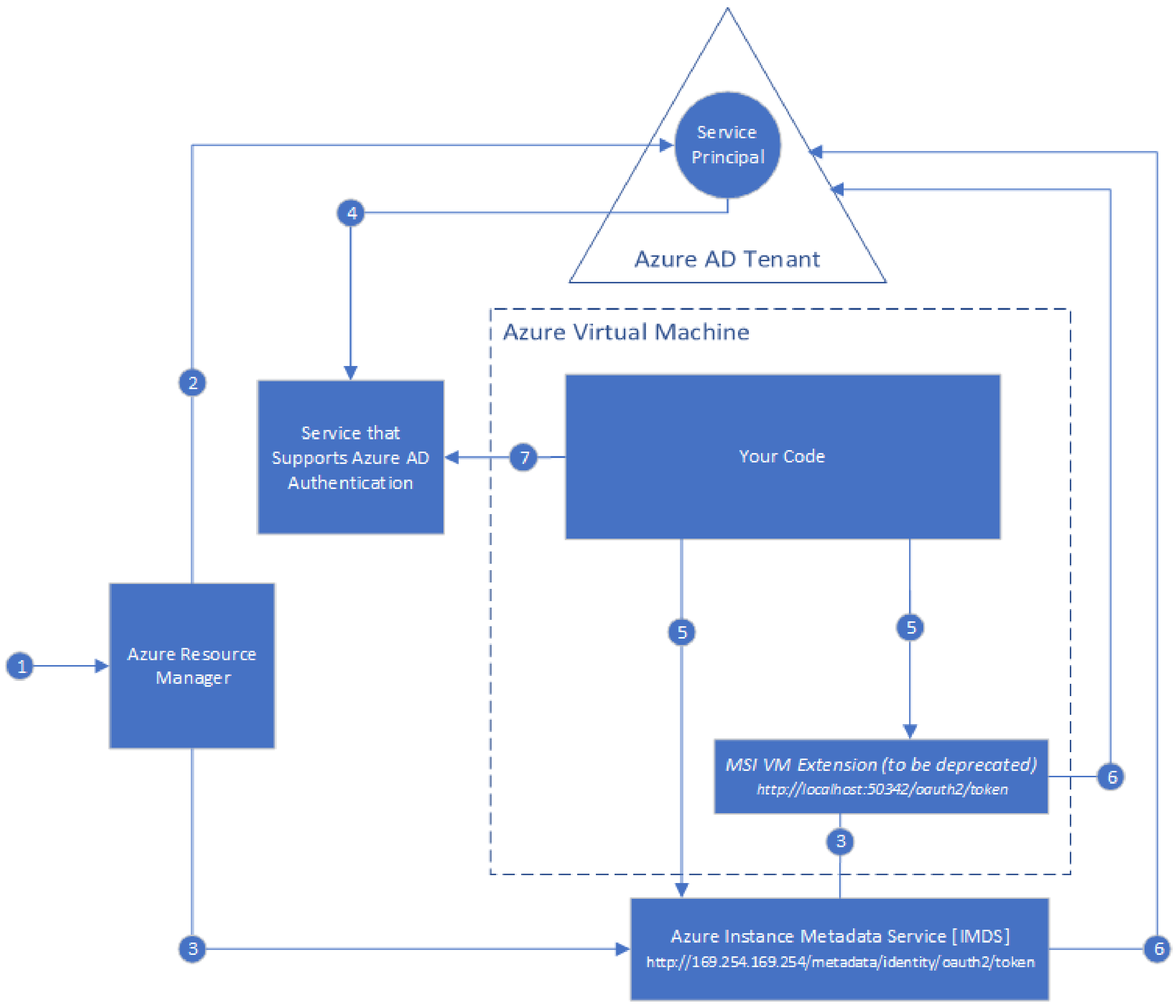
- **Resource group**: A resource group is a container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups, based 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.


Service Principal
Azure AD Tenant
Azure Virtual Machine
Your Code
Service that Supports Azure AD Authentication
Azure Resource Manager
MSI VM Extension (to be deprecated)
http://localhost:50342/oauth2/token
Azure Instance Metadata Service [IMDS]
http://169.254.169.254/metadata/identity/oauth2/token
1
2
3
4
5
6
7

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

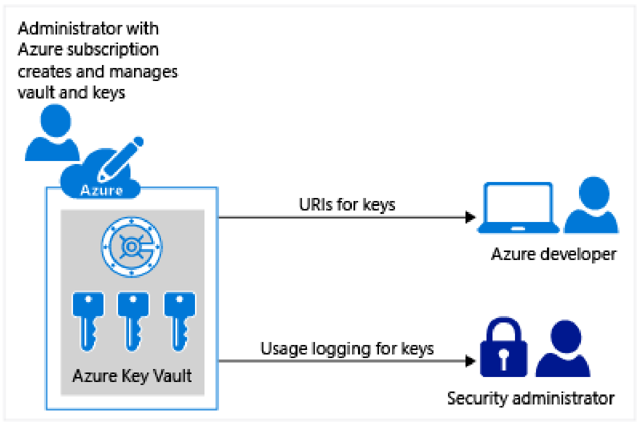
Use the following table to better understand how Key Vault can help to meet the needs of developers and security administrators.

ROLE	PROBLEM STATEMENT	SOLVED BY AZURE KEY VAULT
Developer for an Azure application	"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. 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."	√ 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)	"I don't want the responsibility or potential liability for my customers' tenant keys and secrets. 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."	√ 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)	"I want to know that our applications comply with FIPS 140-2 Level 2 HSMs for secure key management. I want to make sure that my organization is in control of the key lifecycle and can monitor key usage. And although we use multiple Azure services and resources, I want to manage the keys from a single location in Azure."	√ 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 soft-delete overview

4/25/2019 • 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

With this feature, the DELETE operation on a key vault or key vault object is a soft-delete, effectively holding the resources for a given retention period (90 days), while giving the appearance that the object is deleted. The service further provides a mechanism for recovering the deleted object, essentially undoing the deletion.

Soft-delete is an optional Key Vault behavior and is **not enabled by default** in this release. It can be turned on via CLI or Powershell.

Purge protection

When purge protection is on, a vault or an object in deleted state cannot be purged until the retention period of 90 days has passed. These vaults and objects can still be recovered, assuring customers that the retention policy will be followed.

Purge protection is an optional Key Vault behavior and is **not enabled by default**. It can be turned on via CLI or Powershell.

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

Azure Key Vault throttling guidance

4/25/2019 • 3 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 are there to 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. When this happens, Key Vault returns HTTP status code 429 (Too many requests), and the requests fail. Also, failed requests that return a 429 count towards the throttle limits tracked by Key Vault.

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.

```
public sealed class RetryWithExponentialBackoff
{
    private readonly int maxRetries, delayMilliseconds, maxDelayMilliseconds;

    public RetryWithExponentialBackoff(int maxRetries = 50,
        int delayMilliseconds = 200,
        int maxDelayMilliseconds = 2000)
    {
        this.maxRetries = maxRetries;
        this.delayMilliseconds = delayMilliseconds;
        this.maxDelayMilliseconds = maxDelayMilliseconds;
    }

    public async Task RunAsync(Func<Task> func)
    {
        ExponentialBackoff backoff = new ExponentialBackoff(this.maxRetries,
            this.delayMilliseconds,
            this.maxDelayMilliseconds);
        retry:
        try
```

```
            {
                await func();
            }
            catch (Exception ex) when (ex is TimeoutException ||
                ex is System.Net.Http.HttpRequestException)
            {
                Debug.WriteLine("Exception raised is: " +
                    ex.GetType().ToString() +
                    " –Message: " + ex.Message +
                    " -- Inner Message: " +
                    ex.InnerException.Message);
                await backoff.Delay();
                goto retry;
            }
        }
    }

    public struct ExponentialBackoff
    {
        private readonly int m_maxRetries, m_delayMilliseconds, m_maxDelayMilliseconds;
        private int m_retries, m_pow;

        public ExponentialBackoff(int maxRetries, int delayMilliseconds,
            int maxDelayMilliseconds)
        {
            m_maxRetries = maxRetries;
            m_delayMilliseconds = delayMilliseconds;
            m_maxDelayMilliseconds = maxDelayMilliseconds;
            m_retries = 0;
            m_pow = 1;
        }

        public Task Delay()
        {
            if (m_retries == m_maxRetries)
            {
                throw new TimeoutException("Max retry attempts exceeded.");
            }
            ++m_retries;
            if (m_retries < 31)
            {
                m_pow = m_pow << 1; // m_pow = Pow(2, m_retries - 1)
            }
            int delay = Math.Min(m_delayMilliseconds * (m_pow - 1) / 2,
                m_maxDelayMilliseconds);
            return Task.Delay(delay);
        }
    }
```

Using this code in a client C# application is straightforward. The following example shows how, using the HttpClient class.

```
public async Task<Cart> GetCartItems(int page)
{
    _apiClient = new HttpClient();
    //
    // Using HttpClient with Retry and Exponential Backoff
    //
    var retry = new RetryWithExponentialBackoff();
    await retry.RunAsync(async () =>
    {
        // work with HttpClient call
        dataString = await _apiClient.GetStringAsync(catalogUrl);
    });
    return JsonConvert.DeserializeObject<Cart>(dataString);
}
```

Remember that this code is suitable only as a proof of concept.

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

4/25/2019 • 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	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	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

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

4/25/2019 • 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 to use in the authorization request.

See Also

About keys, secrets, and certificates

Common parameters and headers

4/25/2019 • 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

4/25/2019 • 25 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:

`keyvault-name`	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 -.
`object-type`	The type of the object, either "keys" or "secrets".
`object-name`	An `object-name` 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 -.
`object-version`	An `object-version` 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 T 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

- **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:

- 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 (**get**ting) 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.

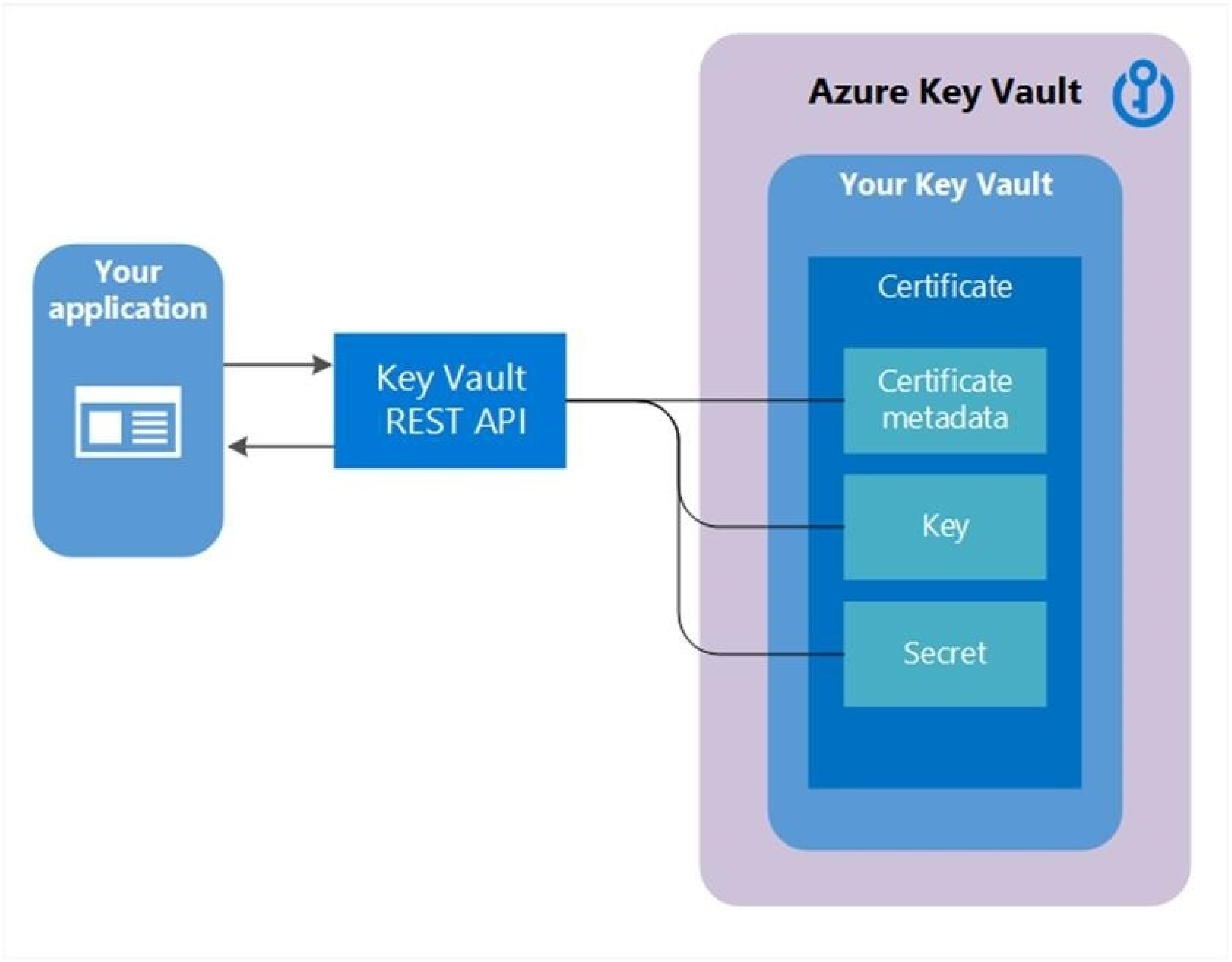
NOTE

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, it's 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
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 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) 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 versions
- Key Vault Developer's Guide

Get started with Key Vault certificates

4/25/2019 • 4 minutes to read • Edit Online

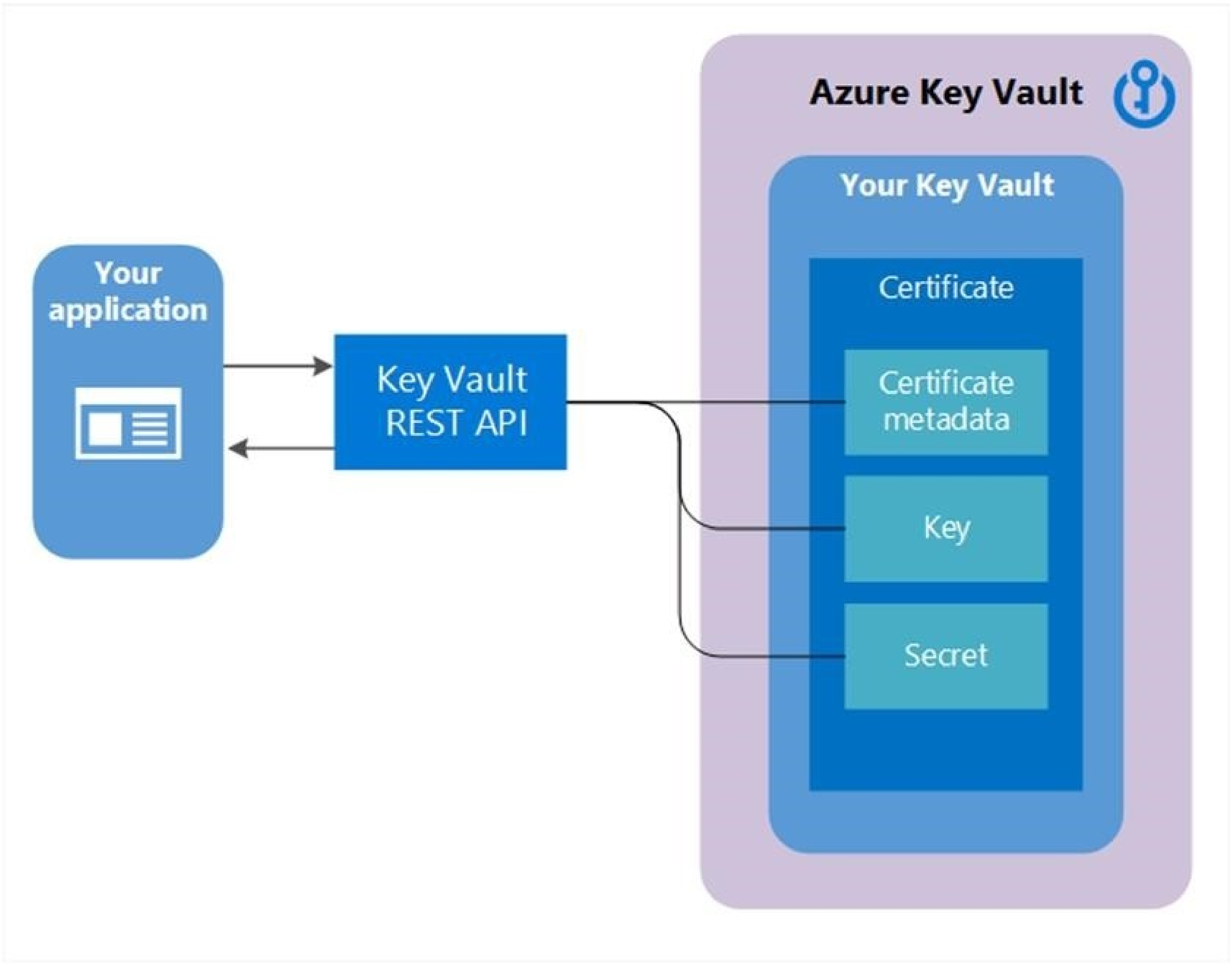
The following scenarios outline several of the primary usages of Key Vault's certificate management service including the additional steps required for creating your first certificate in your key vault.

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 SSL certificates with DigiCert.
  - GlobalSign - Key Vault offers OV SSL certificates with GlobalSign
  - WoSign - Key Vault offers OV SSL or EV SSL certificates with WoSign based on setting configured by customer in their WoSign account on the WoSign portal.

Step 2 - An account admin for a CA provider creates credentials to be used by Key Vault to enroll, renew, and use 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.

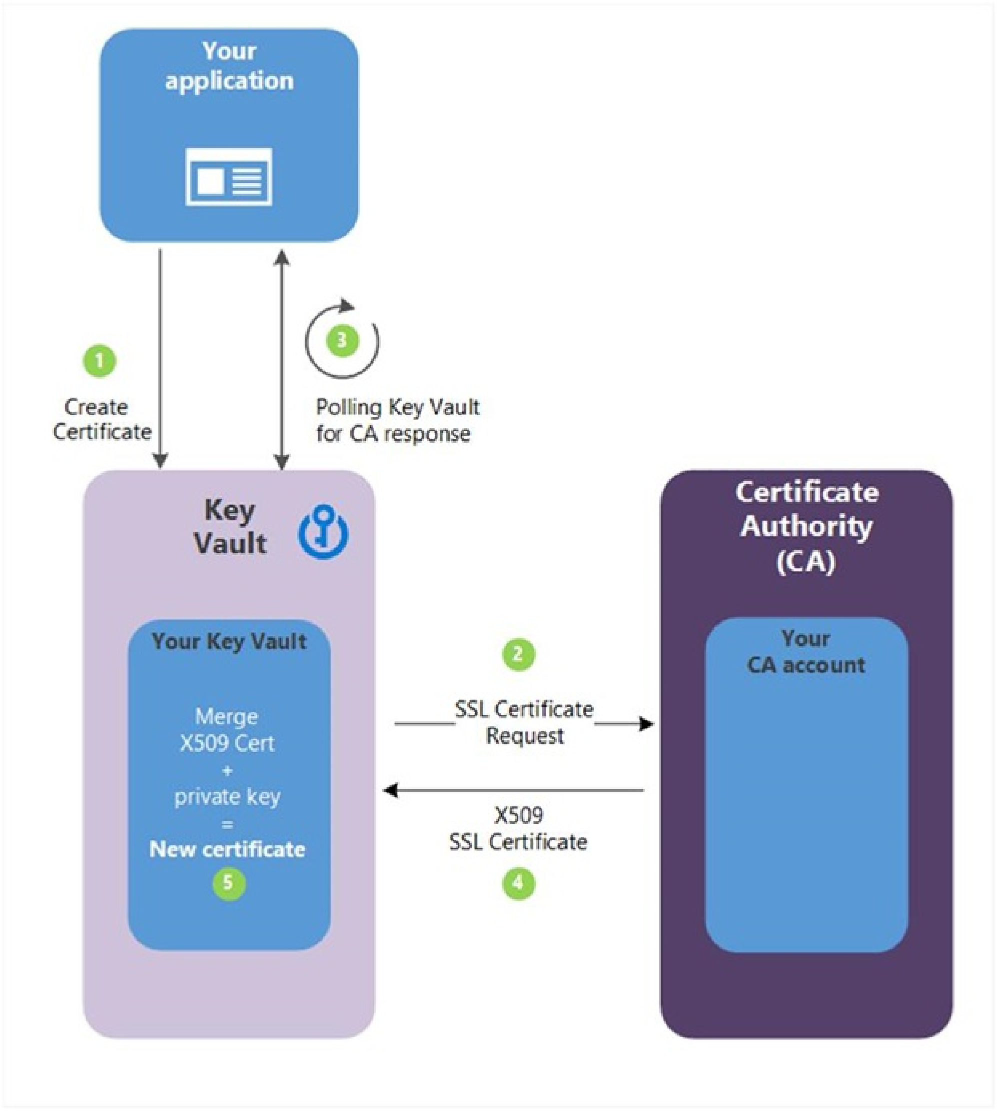
- Begin an add credential operation to a key vault by setting a certificate issuer resource. A certificate issuer is an entity represented in Azure Key Vault (KV) as a CertificateIssuer resource. It is used to provide information about the source of a KV certificate; issuer name, provider, credentials, 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 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 SSL Certificate Request with an X509 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-----

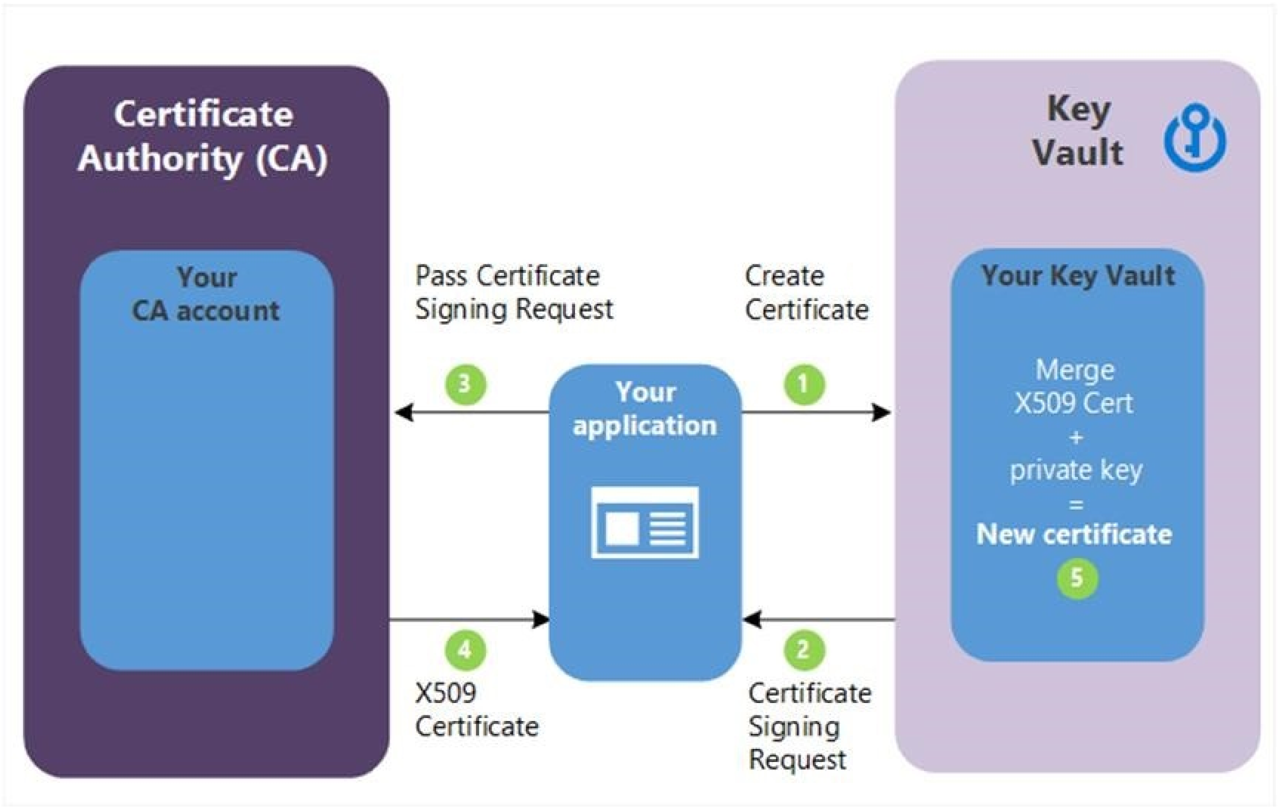
-----BEGIN PRIVATE KEY----- -----END PRIVATE KEY-----

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.


Certificate Authority (CA)
Your CA account
Pass Certificate Signing Request
3
Your application
4
X509 Certificate
Create Certificate
1
2
Certificate Signing Request
Key Vault
Your Key Vault
Merge X509 Cert + private key = New certificate
5

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

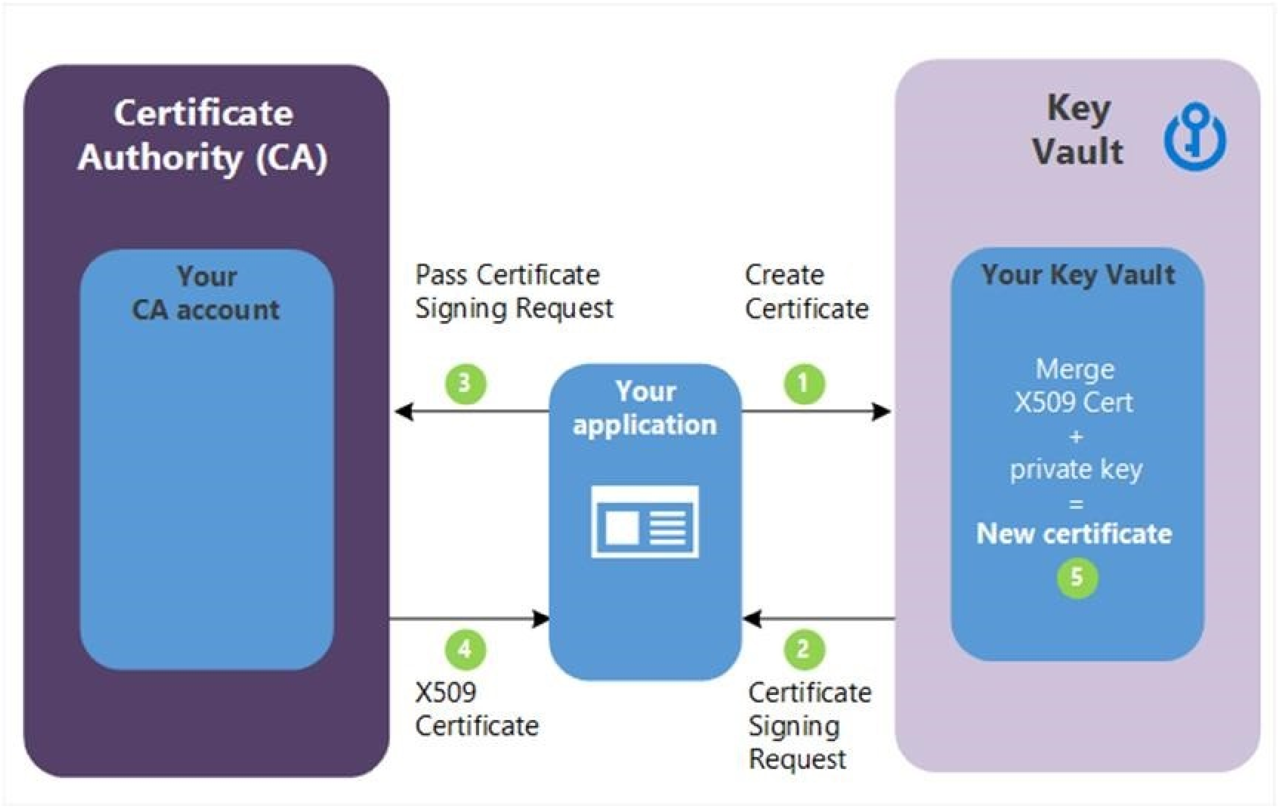
- About keys, secrets, and certificates

Certificate creation methods

4/25/2019 • 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:

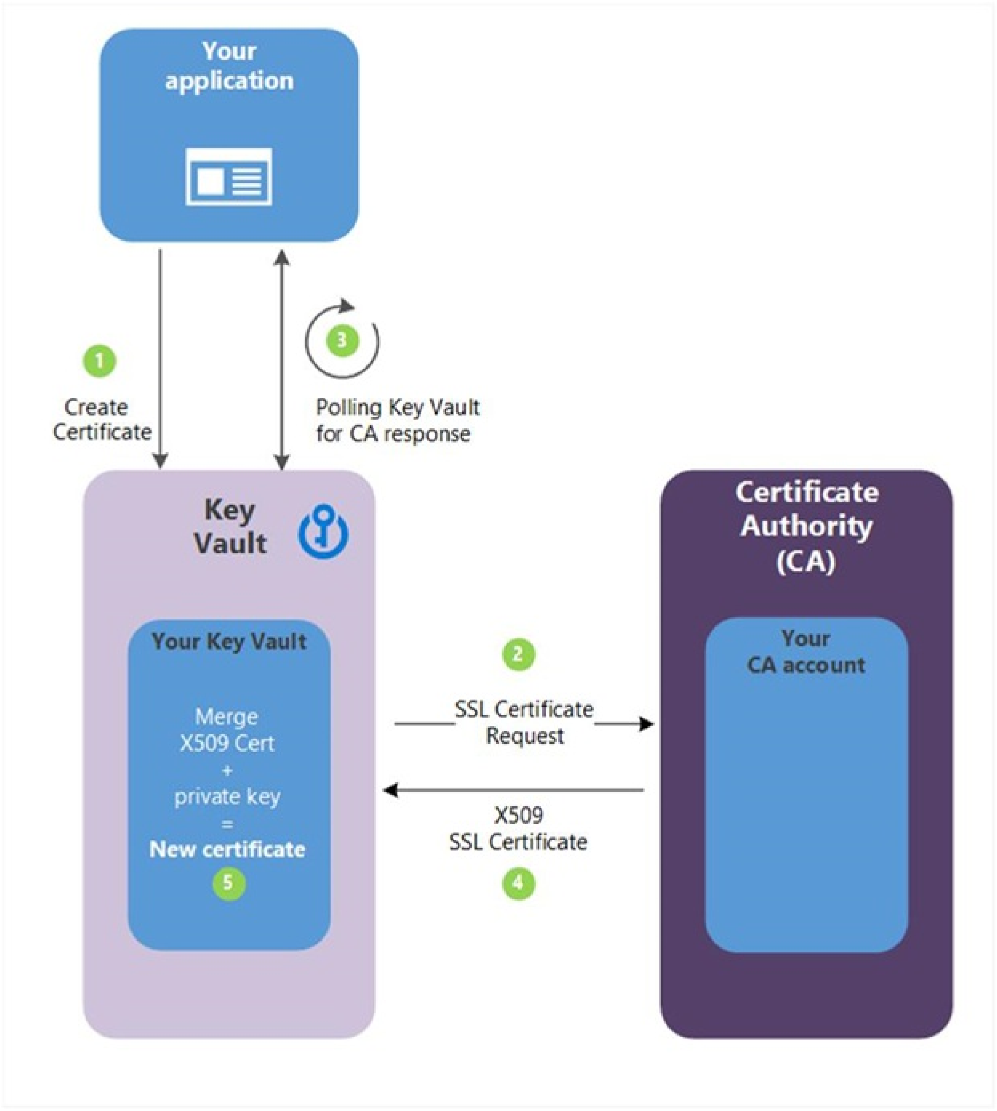
- **Create a self-signed certificate:** This will create a public-private key pair and associate it with a certificate. The certificate will be signed by its own key.
- **Create a new certificate manually:** This will create a public-private key pair and generate an X.509 certificate signing request. 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.


Certificate Authority (CA)
Your CA account
Pass Certificate Signing Request
3
Your application
4
X509 Certificate
Create Certificate
1
2
Certificate Signing Request
Key Vault
Your Key Vault
Merge X509 Cert + private key = New certificate
5

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 you 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 and 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 SSL Certificate Request with an X509 SSL Certificate.
5. Your new certificate creation completes with the merger of the X509 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
GlobalCert	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

4/25/2019 • 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	`https://mykeyvault.vault.azure.net/certificates/mycert1/create?api-version={api-version}`

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	`https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}`

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	`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}"`

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}"`

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	`https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}`

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	`https://mykeyvault.vault.azure.net/certificates/mycert1/create?api-version={api-version}`

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/me` `api-version={api-version}`

Request

```
{
  "x5c": [ "MIICxTCCAbi…………………………trimmed for brevitiy……………………………………………………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 progess."
  }
}
```

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

METHOD	REQUEST URI
PATCH	`https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}`

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	`https://mykeyvault.vault.azure.net/certificates/mycert1/pending?api-version={api-version}`

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/me api-version={api-version}`

Request

```
{
  "x5c": [ "MIICxTCCAbi………………………trimmed for brevitiy………………………………………………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

Common security attributes for Azure Key Vault

4/25/2019 • 2 minutes to read • Edit Online

Security is integrated into every aspect of an Azure service. This article documents the common security attributes built into Azure Key Vault.

A security attribute is a quality or feature of an Azure service that contributes to the service's ability to prevent, detect, and respond to security vulnerabilities.

Security attributes are categorized as:

- Preventative
- Network segmentation
- Detection
- Identity and access management support
- Audit trail
- Access controls (if used)
- Configuration management (if used)

In each category, we identify if an attribute is used or not (yes/no). For some services, an attribute may not be applicable and is shown as N/A. A note or a link to more information about an attribute may also be provided.

Preventative

SECURITY ATTRIBUTE	YES/NO	NOTES
Encryption at rest: • Server-side encryption • Server-side encryption with customer-managed keys • Other encryption features (such as client-side, always encrypted, etc.)	Yes	All objects are encrypted.
Encryption in transit: • Express route encryption • In VNet encryption • VNet-VNet encryption	Yes	All communication is via encrypted API calls
Encryption key handling (CMK, BYOK, etc.)	Yes	The customer controls all keys in their Key Vault. When hardware security module (HSM) backed keys are specified, a FIPS Level 2 HSM protects the key, certificate, or secret.
Column level encryption (Azure Data Services)	N/A	
API calls encrypted	Yes	Using HTTPS.

Network Segmentation

SECURITY ATTRIBUTE	YES/NO	NOTES
Service endpoint support	Yes	Using Virtual Network (VNet) service endpoints.
VNet injection support	No	
Network isolation and firewalling support	Yes	Using VNet firewall rules.
Forced tunneling support	No	

Detection

SECURITY ATTRIBUTE	YES/NO	NOTES
Azure monitoring support (Log analytics, App insights, etc.)	Yes	Using Log Analytics.

Identity and access management

SECURITY ATTRIBUTE	YES/NO	NOTES
Authentication	Yes	Authentication is through Azure Active Directory.
Authorization	Yes	Using Key Vault Access Policy.

Audit Trail

SECURITY ATTRIBUTE	YES/NO	NOTES
Control/Management plane Logging and Audit	Yes	Using Log Analytics.
Data plane logging and audit	Yes	Using Log Analytics.

Access controls

SECURITY ATTRIBUTE	YES/NO	NOTES
Control/Management plane access controls	Yes	Azure Resource Manager Role-Based Access Control (RBAC)
Data plane access controls (At every service level)	Yes	Key Vault Access Policy

Azure Key Vault logging

4/25/2019 • 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","app
id":"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

4/25/2019 • 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)	**Global:** login.microsoftonline.com:443 **Azure China:** login.chinacloudapi.cn:443 **Azure US Government:** login.microsoftonline.us:443 **Azure Germany:** login.microsoftonline.de:443
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	**Global:** management.azure.com:443 **Azure China:** management.chinacloudapi.cn:443 **Azure US Government:** management.usgovcloudapi.net:443 **Azure Germany:** management.microsoftazure.de:443
Azure Active Directory Graph API	**Global:** graph.windows.net:443 **Azure China:** graph.chinacloudapi.cn:443 **Azure US Government:** graph.windows.net:443 **Azure Germany:** graph.cloudapi.de:443

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)	**Global:** <vault-name>.vault.azure.net:443 **Azure China:** <vault-name>.vault.azure.cn:443 **Azure US Government:** <vault-name>.vault.usgovcloudapi.net:443 **Azure Germany:** <vault-name>.vault.microsoftazure.de:443

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

4/25/2019 • 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.

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

4/25/2019 • 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.

Q: My subscription was moved from tenant A to tenant B. How do I change the tenant ID for my existing key vault and set correct ACLs for principals in tenant B?

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. 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. To fix this issue, you need to:

- Change the tenant ID associated with all existing key vaults in this subscription to tenant B.
- Remove all existing access policy entries.
- Add new access policy entries that are 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, here's how to change the tenant ID for this key vault and remove old access policies.

```
Select-AzSubscription -SubscriptionId YourSubscriptionID
$vaultResourceId = (Get-AzKeyVault -VaultName myvault).ResourceId
$vault = Get-AzResource –ResourceId $vaultResourceId -ExpandProperties
$vault.Properties.TenantId = (Get-AzContext).Tenant.TenantId
$vault.Properties.AccessPolicies = @()
Set-AzResource -ResourceId $vaultResourceId -Properties $vault.Properties
```

Because this vault was in tenant A before the move, the original value of **$vault.Properties.TenantId** is tenant A, while **(Get-AzContext).Tenant.TenantId** is tenant B.

Now that your vault is associated with the correct tenant ID and old access policy entries are removed, set new access policy entries with Set-AzKeyVaultAccessPolicy.

Next steps

If you have questions about Azure Key Vault, visit the Azure Key Vault Forums.

Manage Key Vault using the Azure CLI

4/25/2019 • 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] 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] 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 "PaSSWORD"
```

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` command:

```
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

3/21/2019 • 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 resource update --id $(az keyvault show --name ContosoVault -o tsv | awk '{print $1}') --set
properties.enableSoftDelete=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

3/22/2019 • 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.

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?.ate=Succeeded}

Grant several applications access to a key vault

4/25/2019 • 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.

Access control policy can be used to grant several applications access to a key vault. An access control policy can support up to 1024 applications, and is configured as follows:

1. Create an Azure Active Directory security group.
2. Add all of the applications' associated service principals to the security group.
3. Grant the security group access to your Key Vault.

Prerequisites

Here are the prerequisites:

- Install Azure PowerShell.
- Install the Azure Active Directory V2 PowerShell module.
- Permissions to create/edit groups in the Azure Active Directory tenant. If you don't have permissions, you may need to contact your Azure Active Directory administrator. See About Azure Key Vault keys, secrets and certificates for details on Key Vault access policy permissions.

Granting Key Vault access to applications

Run the following commands in PowerShell:

```
# Connect to Azure AD
Connect-AzureAD

# Create Azure Active Directory Security Group
$aadGroup = New-AzureADGroup -Description "Contoso App Group" -DisplayName "ContosoAppGroup" -MailEnabled 0 -
MailNickName none -SecurityEnabled 1

# Find and add your applications (ServicePrincipal ObjectID) as members to this group
$spn = Get-AzureADServicePrincipal –SearchString "ContosoApp1"
Add-AzureADGroupMember –ObjectId $aadGroup.ObjectId -RefObjectId $spn.ObjectId

# You can add several members to this group, in this fashion.

# Set the Key Vault ACLs
Set-AzKeyVaultAccessPolicy –VaultName ContosoVault –ObjectId $aadGroup.ObjectId `
-PermissionsToKeys
decrypt,encrypt,unwrapKey,wrapKey,verify,sign,get,list,update,create,import,delete,backup,restore,recover,purge
`
–PermissionsToSecrets get,list,set,delete,backup,restore,recover,purge `
–PermissionsToCertificates
get,list,delete,create,import,update,managecontacts,getissuers,listissuers,setissuers,deleteissuers,manageissue
rs,recover,purge,backup,restore `
-PermissionsToStorage
get,list,delete,set,update,regeneratekey,getsas,listsas,deletesas,setsas,recover,backup,restore,purge

# Of course you can adjust the permissions as required
```

If you need to grant a different set of permissions to a group of applications, create a separate Azure Active Directory security group for such applications.

Next steps

Learn more about how to Secure your key vault.

Configure Azure Key Vault firewalls and virtual networks

4/25/2019 • 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 **Firewalls and virtual networks**.
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

Best practices to use Key Vault

5/6/2019 • 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

4/25/2019 • 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 to use in the authorization request.

See Also

About keys, secrets, and certificates

Common parameters and headers

4/25/2019 • 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

4/25/2019 • 7 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.
- As of the September 2016 release of Key Vault, your applications can now 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.

Storage Account Keys - July 10, 2017

> **NOTE**
> For this update of Azure Key Vault only the **Storage Account Keys** feature is in preview.

This preview includes our new Storage Account Keys feature, available through these interfaces; .NET/C#, REST and PowerShell.

For more information on the new Storage Account Keys feature, see Azure Key Vault storage account keys overview.

Videos

This video shows you how to create your own key vault and how to use it from the 'Hello Key Vault' sample application.

- Key Vault developer - quick start guide

Resources mentioned in above video:

- Azure PowerShell
- Azure Key Vault Sample Code

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 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 Node.js

Python

Azure Key Vault libraries for Python

Azure CLI 2

Azure CLI for Key Vault

Azure PowerShell

Azure PowerShell for Key Vault

Quick start guides

- Create Key Vault
- Getting started with Key Vault in Node.js

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.

How to generate and transfer HSM-protected keys for Azure Key Vault

5/13/2019 • 16 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. The HSMs are FIPS 140-2 Level 2 validated. Azure Key Vault uses nCipher nShield family of HSMs 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 American or Asia.

More information about nCipher HSMs and Microsoft services

nCipher Security is a leading global provider of data encryption and cyber security solutions to the financial services, high technology, manufacturing, government, and technology sectors. With a 40-year track record of protecting corporate and government information, nCipher Security cryptographic solutions are used by four of the five largest energy and aerospace companies. Their solutions are also used by 22 NATO countries/regions, and secure more than 80 per cent of worldwide payment transactions.

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

India:

KeyVault-BYOK-Tools-India.zip

49EDCEB3091CF1DF7B156D5B495A4ADE1CFBA77641134F61B0E0940121C436C8

France:

KeyVault-BYOK-Tools-France.zip

5C9D1F3E4125B0C09E9F60897C9AE3A8B4CB0E7D13A14F3EDBD280128F8FE7DF

United Kingdom:

KeyVault-BYOK-Tools-UnitedKingdom.zip

432746BD0D3176B708672CCFF19D6144FCAA9E5EB29BB056489D3782B3B80849

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 the 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

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 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
```

> **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 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
```

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_xferacId_<contosokey>.

You may inspects 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 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
```

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.

Azure Key Vault .NET 2.0 - Release Notes and Migration Guide

4/25/2019 • 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, string []* changed to *IList*
- *NextList* changed to *NextPageLink*

Return types

- **KeyList** and **SecretList** now returns *IPage* 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.Error.Message*.
- 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

5/9/2019 • 4 minutes to read • Edit Online

Overview

This article describes how to securely save secret application configuration settings for Azure applications.

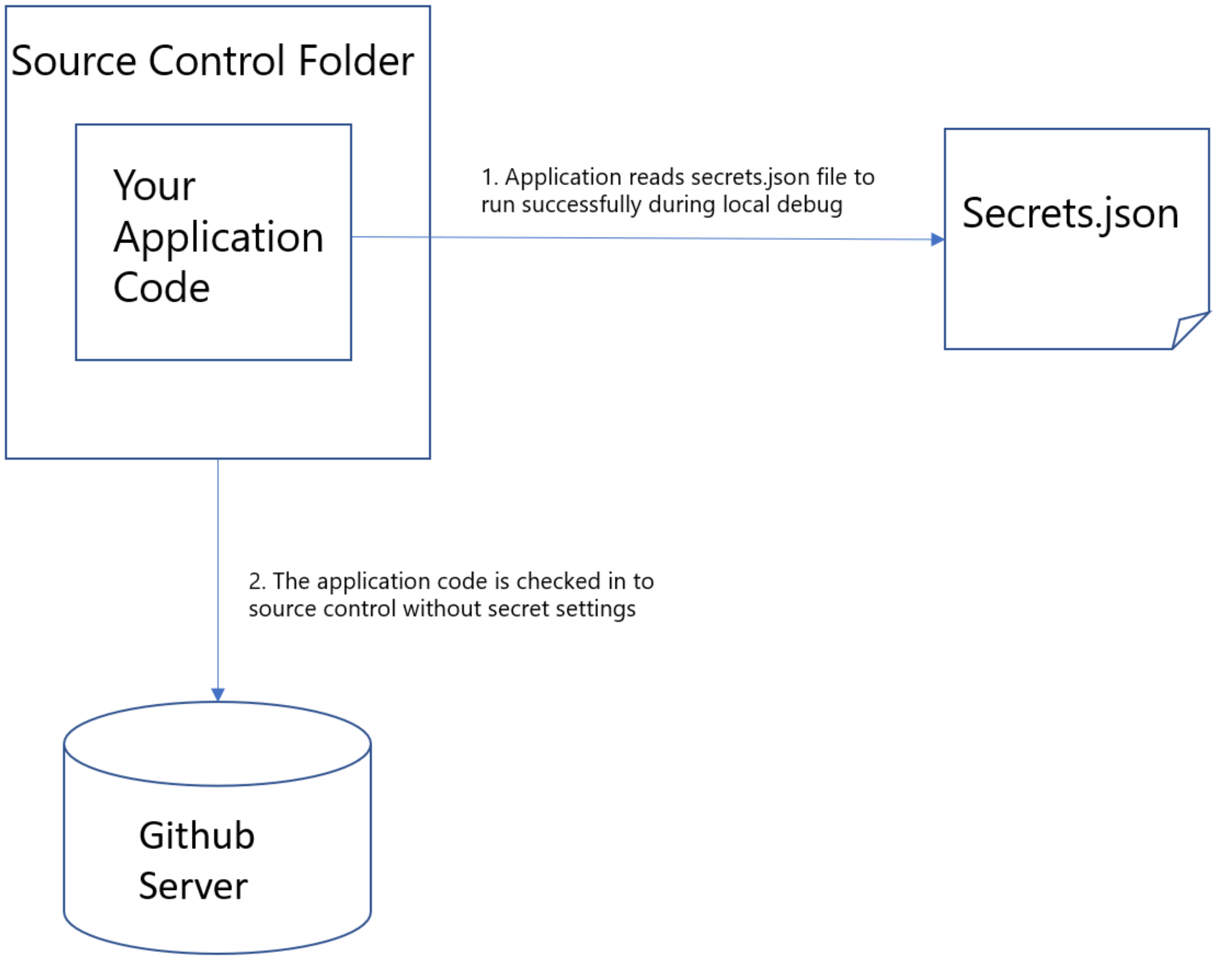
Traditionally all web application configuration settings are saved in configuration files such as Web.config. This practice leads to checking in secret settings such as Cloud credentials to public source control systems like GitHub. Meanwhile, it could be hard to follow security best practice because of the overhead required to change source code and reconfigure development settings.

To make sure 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.

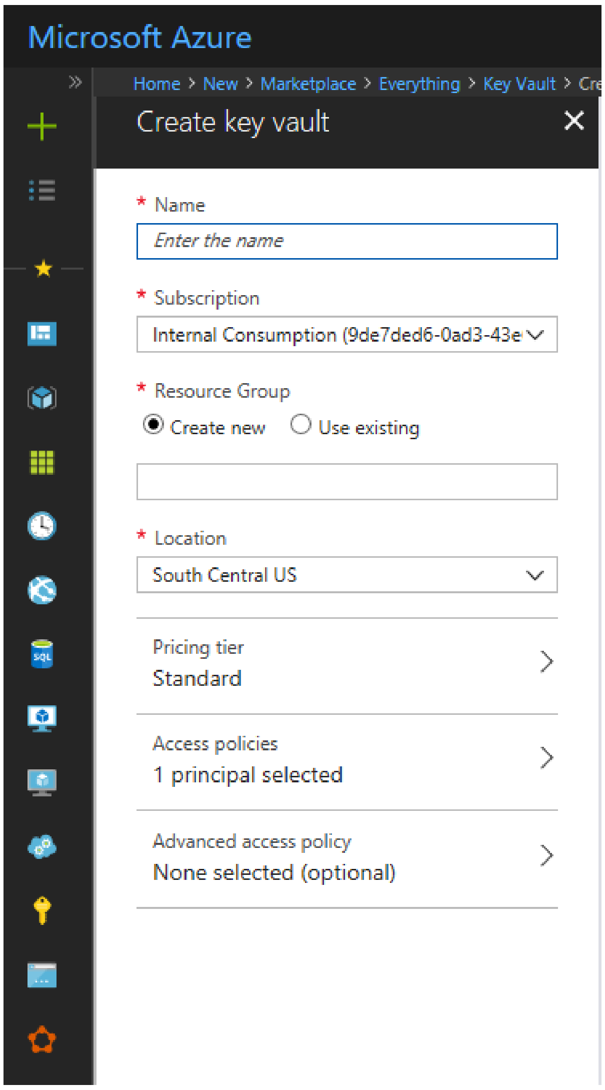

Source Control Folder
Your Application Code
1. Application reads secrets.json file to run successfully during local debug
Secrets.json
2. The application code is checked in to source control without secret settings
Github Server

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



* Name
Enter the name

* Subscription
Internal Consumption (9de7ded6-0ad3-43e

* Resource Group
● Create new ○ Use existing

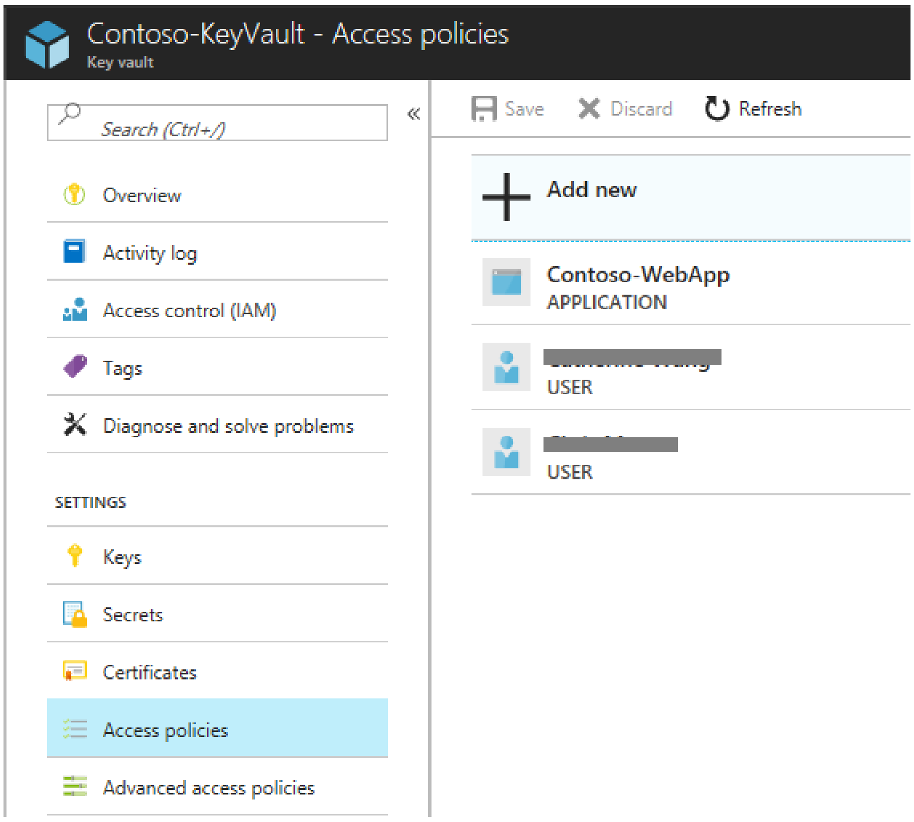
* Location
South Central US

Pricing tier
Standard

Access policies
1 principal selected

Advanced access policy
None selected (optional)

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*.

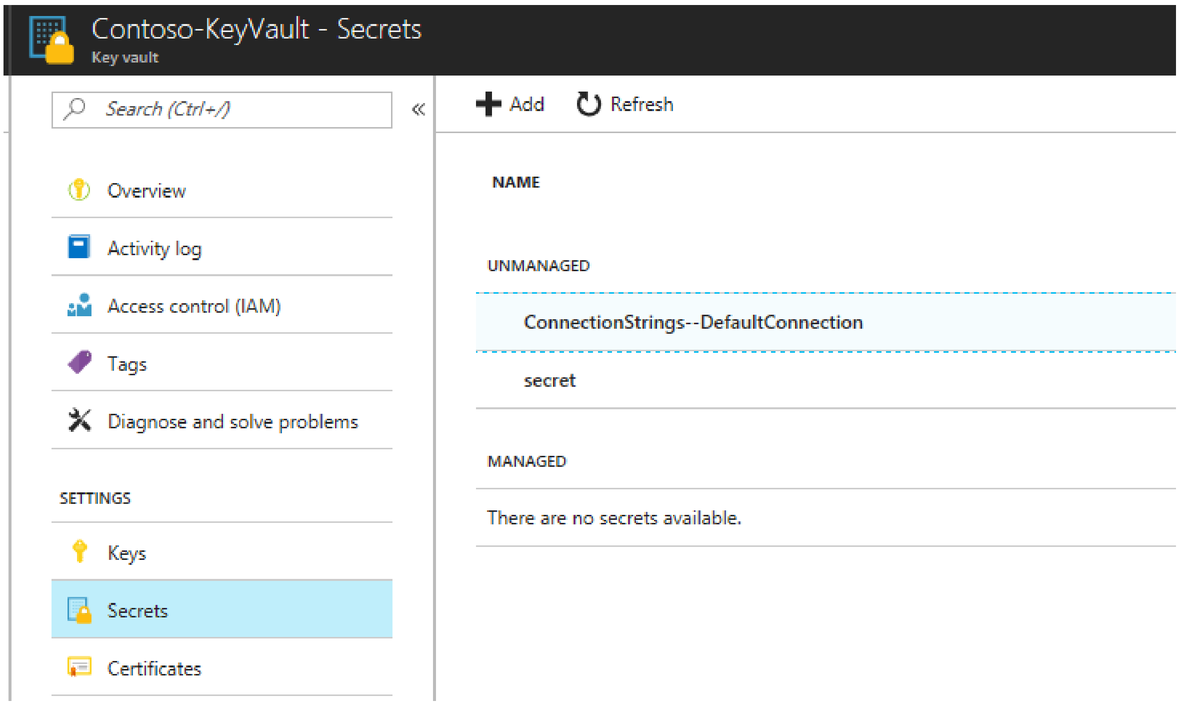


Contoso-WebApp
APPLICATION

[redacted]
USER

[redacted]
USER

3. Add your secret to Key Vault on 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.



NAME

UNMANAGED

ConnectionStrings--DefaultConnection

secret

MANAGED

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 funcionality 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:

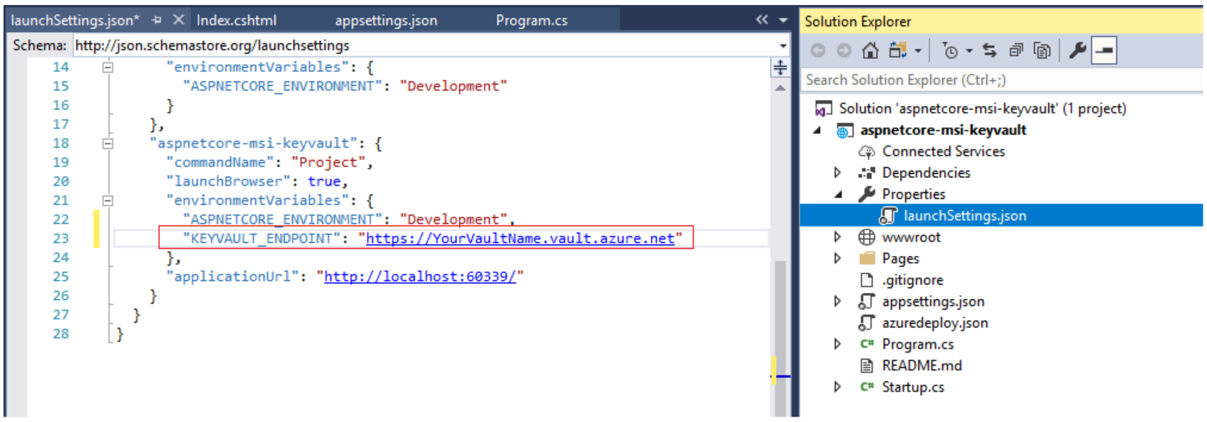
```
Microsoft.Azure.Services.AppAuthentication
```

5. Add the following code to Program.cs file:

```
public static IWebHost BuildWebHost(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>()
                .Build();

    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.



```
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "aspnetcore-msi-keyvault": {
      "commandName": "Project",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development",
        "KEYVAULT_ENDPOINT": "https://YourVaultName.vault.azure.net"
      },
      "applicationUrl": "http://localhost:60339/"
    }
  }
}
```

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.Basic
```

2. Create a file that's similar to the follow. 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 any 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

4/25/2019 • 7 minutes to read • Edit Online

To authenticate to Azure Key Vault, you need an Azure Active Directory (AD) credential, either a shared secret or a certificate. Managing such credentials can be difficult and 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 do not need to create Azure AD credentials or share credentials between developers.

The `Microsoft.Azure.Services.AppAuthentication` library manages authentication automatically, which in turn allows you to focus on your solution, rather than your credentials.

The `Microsoft.Azure.Services.AppAuthentication` library 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 is not available, or when the developer's security context cannot be determined during local development.

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. Add references to the Microsoft.Azure.Services.AppAuthentication and Microsoft.Azure.KeyVault NuGet packages to your application.

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. Consequently, 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, see which Azure services support managed identities for Azure resources.

Samples

The following samples show the `Microsoft.Azure.Services.AppAuthentication` library in action:

1. Use a managed identity to retrieve a secret from Azure Key Vault at runtime

2. Programmatically deploy an Azure Resource Manager template from an Azure VM with a managed identity.

3. Use .NET Core sample and a managed identity to call Azure services from an Azure Linux VM.

Local development authentication

For local development, there are two primary authentication scenarios:

- Authenticating to Azure services
- Authenticating to custom services

Authenticating to Azure Services

Local machines do not 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 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

Authenticating with Visual Studio has the following prerequisites:

1. Visual Studio 2017 v15.5 or later.

2. 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.

Sign in to Visual Studio and use **Tools** > **Options** > **Azure Service Authentication** to select an account for local development.

If you run into problems using Visual Studio, such as errors regarding the token provider file, carefully review these steps.

It may also be necessary to reauthenticate your developer token. To do so, go to **Tools** > **Options**>**Azure Service Authentication** and 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:

1. Install Azure CLI v2.0.12 or later. Upgrade earlier versions.

2. Use **az login** to sign in to Azure.

Use `az account get-access-token` to verify access. If you receive an error, verify that Step 1 completed successfully.

If Azure CLI is not installed to the default directory, you may receive an error reporting that

`AzureServiceTokenProvider` cannot 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.

If you are signed in to Azure CLI using multiple accounts or your account has access to multiple subscriptions, you need to specify the specific subscription to be used. To do so, use:

```
az account set --subscription <subscription-id>
```

This command generates output only on failure. To verify the current account settings, use:

```
az account list
```

Authenticating with Azure AD Integrate authentication

To use Azure AD authentication, verify that:

- Your on-premises active directory syncs to Azure AD.
- Your code is running on a domain-joined machine.

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:

1. Use a service principal to sign into Azure:

   a. Create a service principal.

   b. Use Azure CLI to sign in:

   ```
   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.

2. Use environment variables to specify service principal details. For details, see Running the application using a service principal.

Once you've signed in to Azure, `AzureServiceTokenProvider` uses the service principal to retrieve a token for local development.

This 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.

Alternatively, you may authenticate with a user-assigned identity. For more information on user-assigned identities, see About Managed Identities for Azure resources. The connection string is specified in the Connection String Support section below.

Running the application using a Service Principal

It may be necessary to create an Azure AD Client credential to authenticate. Common examples include:

1. 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.
2. Your code runs on a local development environment and you authenticate to a custom service, so you can't use your developer identity.
3. Your code is running on an Azure compute resource that does not yet support managed identities for Azure resources, such as Azure Batch.

To use a certificate to sign into Azure AD:

1. Create a service principal certificate.
2. Deploy the certificate to either the *LocalMachine* or *CurrentUser* store.
3. Set an environment variable named **AzureServicesAuthConnectionString** to:

   ```
   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.
4. Run the application.

To sign in using an Azure AD shared secret credential:

1. Create a service principal with a password and grant it access to the Key Vault.
2. Set an environment variable named **AzureServicesAuthConnectionString** to:

   ```
   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.

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
`RunAs=Developer; DeveloperTool=AzureCli`	Local development	AzureServiceTokenProvider uses AzureCli to get token.
`RunAs=Developer; DeveloperTool=VisualStudio`	Local development	AzureServiceTokenProvider uses Visual Studio to get token.
`RunAs=CurrentUser`	Local development	AzureServiceTokenProvider uses Azure AD Integrated Authentication to get token.
`RunAs=App`	Managed identities for Azure resources	AzureServiceTokenProvider uses a managed identity to get token.
`RunAs=App;AppId={ClientId of user-assigned identity}`	User-assigned identity for Azure resources	AzureServiceTokenProvider uses a user-assigned identity to get token.
`RunAs=App;AppId={AppId};TenantId={TenantId};CertificateThumbprint={Thumbprint};CertificateStoreLocation={LocalMachine or CurrentUser}`	Service principal	`AzureServiceTokenProvider` uses certificate to get token from Azure AD.
`RunAs=App;AppId={AppId};TenantId={TenantId};CertificateSubjectName={Subject};CertificateStoreLocation={LocalMachine or CurrentUser}`	Service principal	`AzureServiceTokenProvider` uses certificate to get token from Azure AD
`RunAs=App;AppId={AppId};TenantId={TenantId};AppKey={ClientSecret}`	Service principal	`AzureServiceTokenProvider` uses secret to get token from Azure AD.

Next steps

- 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

4/25/2019 • 5 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.

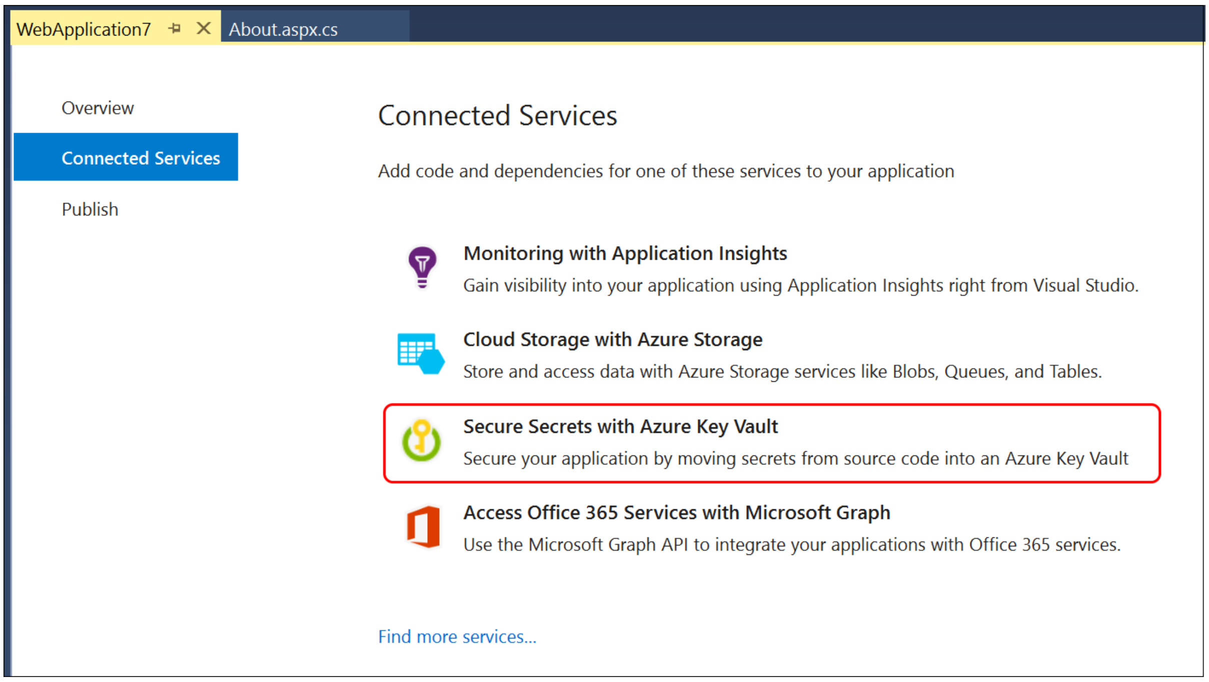
Prerequisites

- **An Azure subscription**. If you do not have one, you can sign up for a free account.
- **Visual Studio 2019** 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 web project open.

Add Key Vault support to your project

1. In **Solution Explorer**, 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**.



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.

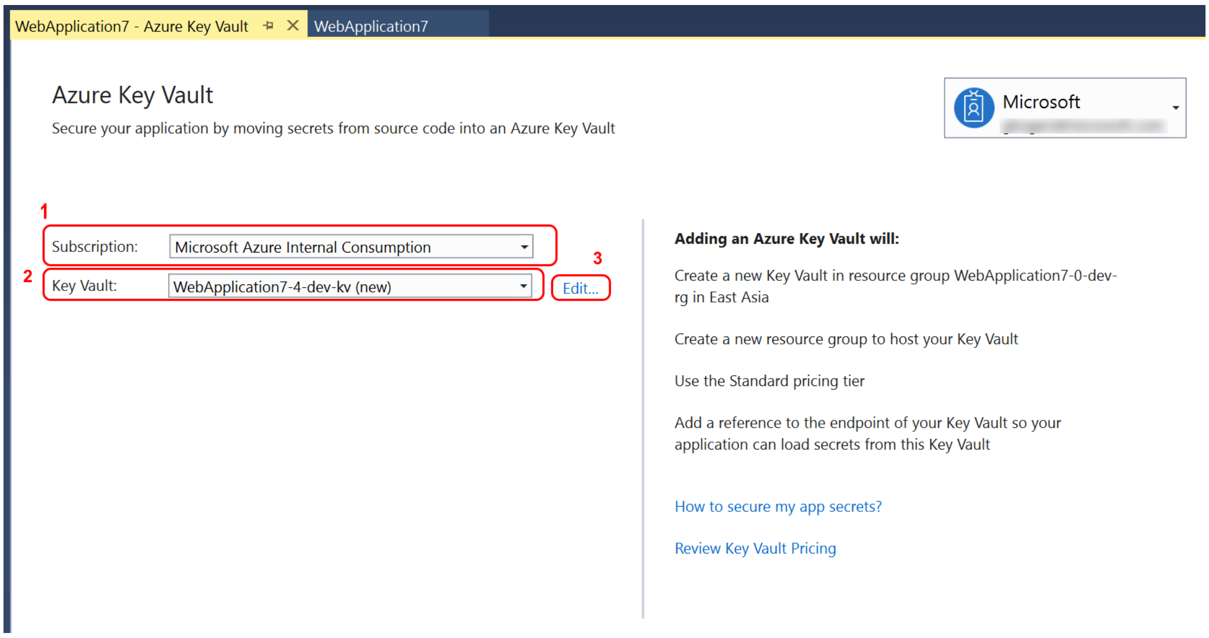
Secure Secrets with Azure Key Vault
Secure your application by moving secrets from source code into an Azure Key Vault

Access Office 365 Services with Microsoft Graph
Use the Microsoft Graph API to integrate your applications with Office 365 services.

Find more services...

If you've signed into Visual Studio, and have an Azure subscription associated with your account, a page appears with a dropdown list with your subscriptions. Make sure that you're signed into Visual Studio, and that the account you're signed in with is the same account that you use for your Azure subscription.

3. Select the subscription you want to use, and then choose a new or existing Key Vault, or choose the Edit link to modify the automatically generated name.


WebApplication7 - Azure Key Vault
WebApplication7
Microsoft
1
2
3
Edit...
How to secure my app secrets?
Review Key Vault Pricing

Azure Key Vault

Secure your application by moving secrets from source code into an Azure Key Vault

Subscription: Microsoft Azure Internal Consumption

Key Vault: WebApplication7-4-dev-kv (new)

Adding an Azure Key Vault will:

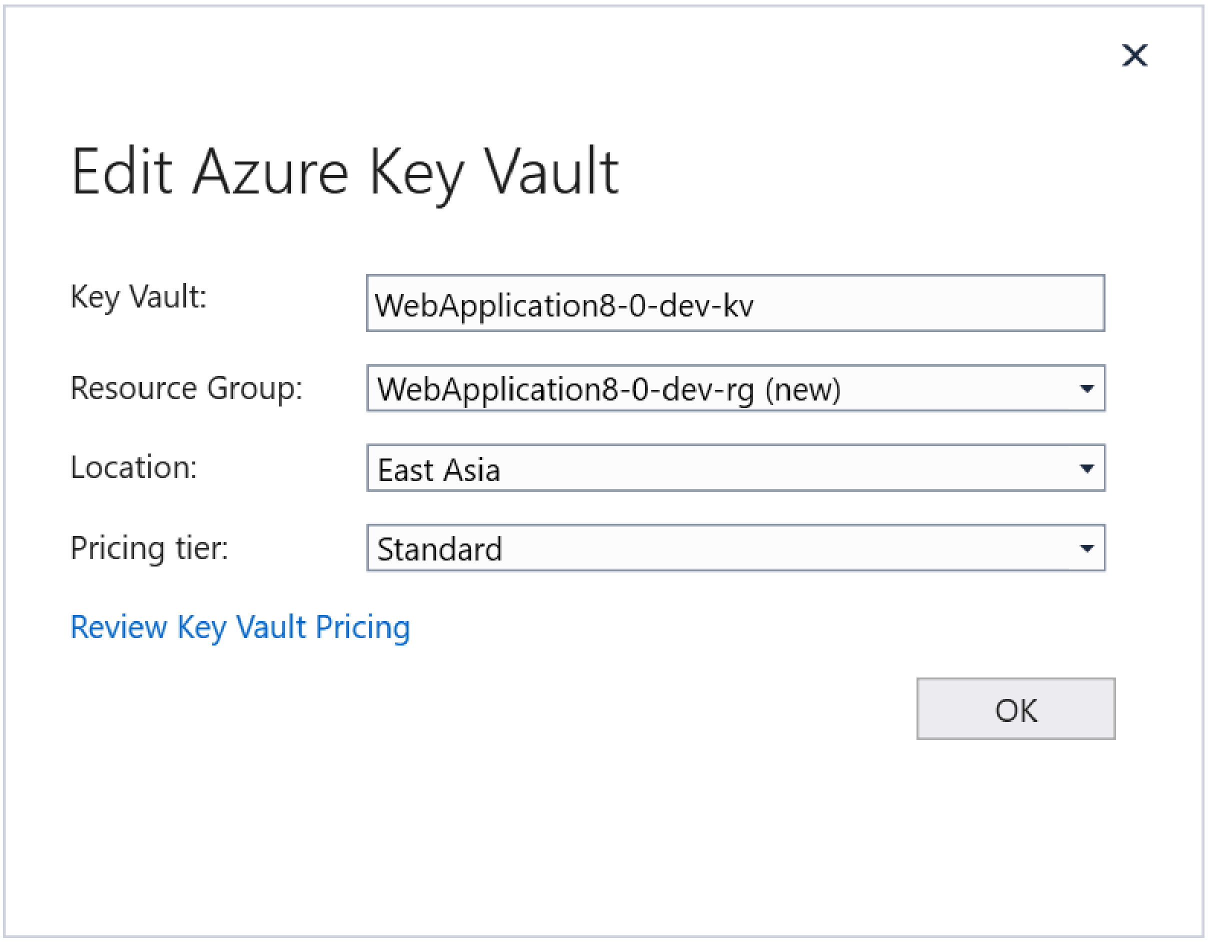
Create a new Key Vault in resource group WebApplication7-0-dev-rg in East Asia

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

4. Type the name you want to use for the Key Vault.



Edit Azure Key Vault

Key Vault: WebApplication8-0-dev-kv

Resource Group: WebApplication8-0-dev-rg (new)

Location: East Asia

Pricing tier: Standard

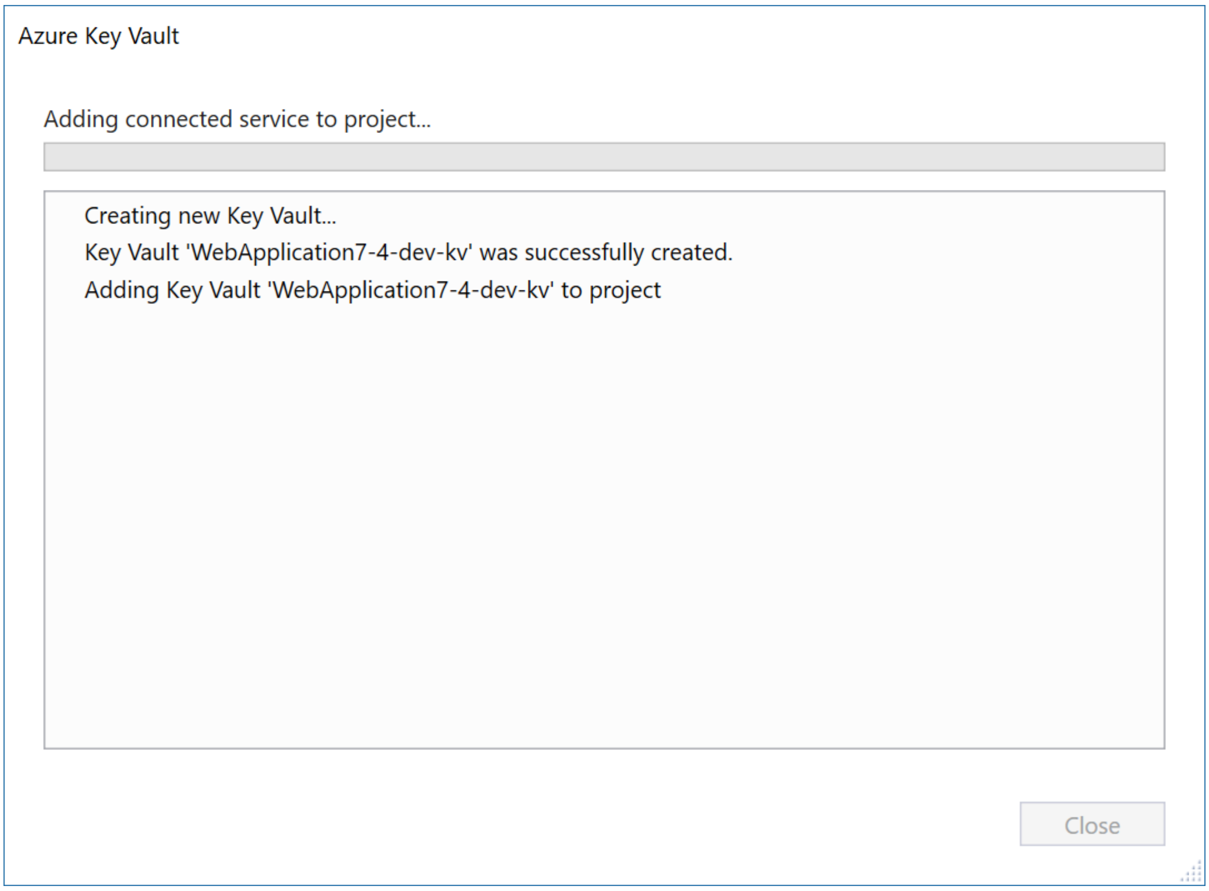
5. Select an existing Resource Group, or choose to create a new one with an automatically generated unique name. If you want to create a new group with a different name, you can use the Azure Portal, and then close the page and restart to reload the list of resource groups.

6. Choose the region 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 model. For details, see Key Vault Pricing.

8. Choose OK to accept the configuration choices.

9. Choose **Add** to create the Key Vault. The create process might fail if you choose a name that was already used. If that happens, use the **Edit** link to rename the Key Vault and try again.

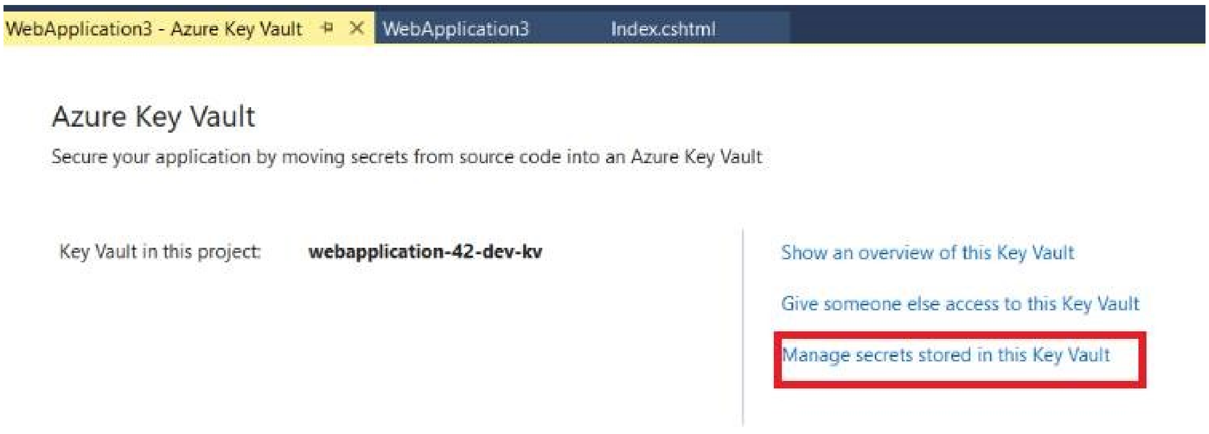


Azure Key Vault

Adding connected service to project...

```
Creating new Key Vault...
Key Vault 'WebApplication7-4-dev-kv' was successfully created.
Adding Key Vault 'WebApplication7-4-dev-kv' to project
```

10. Now, add a secret in your Key Vault in Azure. To get to the right place in the portal, click on the link for Manage secrets stored in this Key Vault. If you closed the page or the project, you can navigate to it in the Azure portal by choosing **All Services**, under **Security**, choose **Key Vault**, then choose the Key Vault you created.



WebApplication3 - Azure Key Vault
WebApplication3
Index.cshtml

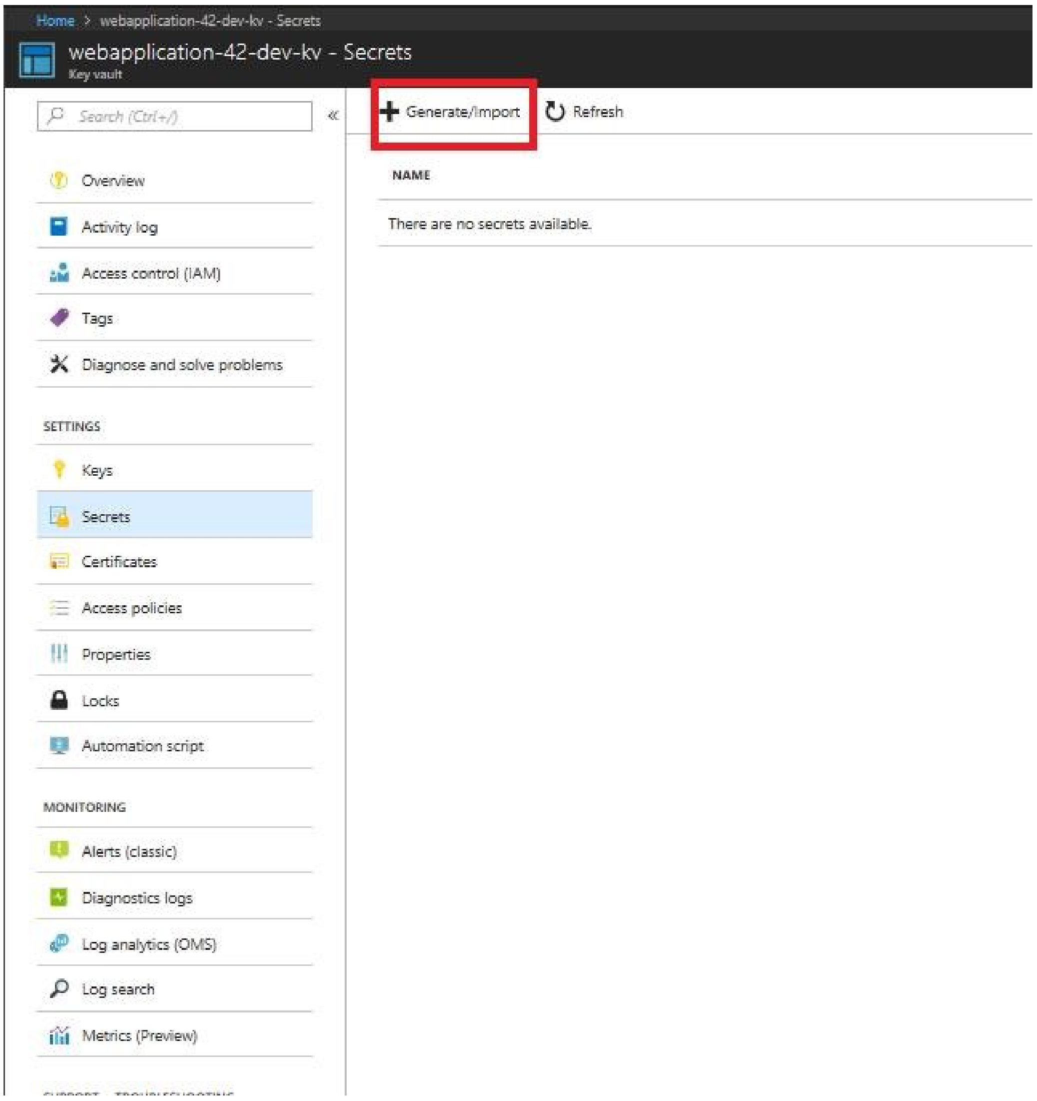
Azure Key Vault

Secure your application by moving secrets from source code into an Azure Key Vault

Key Vault in this project: **webapplication-42-dev-kv**

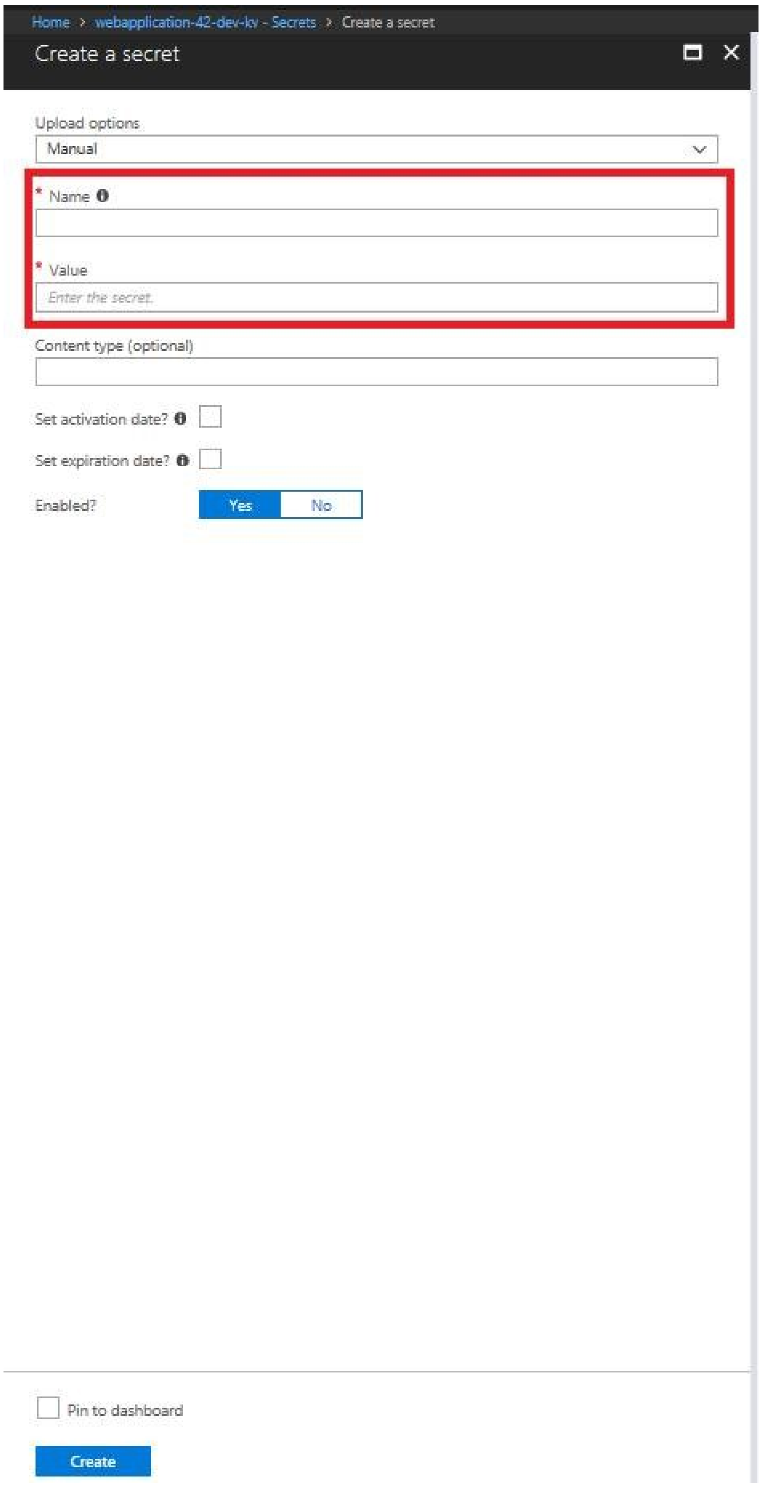
Show an overview of this Key Vault
Give someone else access to this Key Vault
Manage secrets stored in this Key Vault

11. In the Key Vault section for the key vault you created, choose **Secrets**, then **Generate/Import**.



There are no secrets available.

12. Enter a secret, such as "MySecret" and give it any string value as a test, then choose the **Create** button.


Home > webapplication-42-dev-kv - Secrets > Create a secret

Create a secret

Upload options: Manual

* Name

* Value: Enter the secret.

Content type (optional)

Set activation date?

Set expiration date?

Enabled? [✓ Yes] No

Pin to dashboard

Create

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

1. Install these two nuget packages AppAuthentication and KeyVault NuGet libraries.

2. Open Program.cs file and update the code with the following code:

```
public class Program
{
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }

    public static IWebHost BuildWebHost(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>()
         .Build();

    private static string GetKeyVaultEndpoint() => "https://<YourKeyVaultName>.vault.azure.net";
}
```

3. Next open About.cshtml.cs file and write the following code:

   a. Include a reference to Microsoft.Extensions.Configuration by this using statement:

   ```
   using Microsoft.Extensions.Configuration
   ```

   b. Add this constructor:

   ```
   public AboutModel(IConfiguration configuration)
   {
       _configuration = configuration;
   }
   ```

   c. Update the OnGet method. Update the placeholder value shown here with the secret name you created in the above commands.

   ```
   public void OnGet()
   {
       //Message = "Your application description page.";
       Message = "My key val = " + _configuration["<YourSecretNameThatWasCreatedAbove>"];
   }
   ```

Run the app locally by browsing to the About page. You should see your secret value retrieved.

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 type in the name of the resource group and select **Delete**.

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

Affects the project file .NET references and NuGet package references.

TYPE	REFERENCE
NuGet	Microsoft.AspNetCore.AzureKeyVault.HostingStartup

Added files

- ConnectedService.json added, which records some information about the Connected Service provider, version, and a link the documentation.

Project file changes

- Added the Connected Services ItemGroup and ConnectedServices.json file.

launchsettings.json changes

- 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__CONFIGURATIONVAULT": "<your keyvault URL>"
}
```

Changes on Azure

- 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

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

- ConnectedService.json added, which records some information about the Connected Service provider, version, and a link to the documentation.

Project file changes

- 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

- Created a resource group (or used an existing one).
- Created a Key Vault in the specified resource group.

Next steps

Learn more about Key Vault development by reading the Key Vault Developer's Guide

Authentication, requests and responses

4/25/2019 • 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 to use in the authorization request.

See Also

About keys, secrets, and certificates

Common parameters and headers

4/25/2019 • 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

4/25/2019 • 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

Key Vault versions

4/25/2019 • 2 minutes to read • Edit Online

2016-10-01 - Managed Storage Account Keys

Summer 2017 - Storage Account Keys feature added easier integration with Azure Storage. See the overview topic for more information, Managed Storage Account Keys overview.

2016-10-01 - Soft-delete

Summer 2017 - soft-delete feature added for improved data protection of your key vaults and key vault objects. See the overview topic for more information, Soft-delete overview.

2015-06-01 - Certificate management

Certificate management is added as a feature to the GA version 2015-06-01 on September 26, 2016.

2015-06-01 - General availability

General Availability version 2015-06-01, announced on June 24, 2015.

The following changes were made at this release:

- Delete a key - “use” field removed.
- Get information about a key - “use” field removed.
- Import a key into a vault - “use” field removed.
- Restore a key - “use” field removed.
- Changed “RSA_OAEP” to “RSA-OAEP” for RSA Algorithms. See About keys, secrets, and certificates.

2015-02-01-preview

Second preview version 2015-02-01-preview, announced on April 20, 2015. For more information, see REST API Update blog post.

The following tasks were updated:

- List the keys in a vault - added pagination support to operation.
- List the versions of a key - added operation to list the versions of a key.
- List secrets in a vault - added pagination support.
- List versions of a secret - add operation to list the versions of a secret.
- All operations - Added created/updated timestamps to attributes.
- Create a secret - added Content-Type to secrets.
- Create a key - added tags as optional information.
- Create a secret - added tags as optional information.
- Update a key - added tags as optional information.
- Update a secret - added tags as optional information.
- Changed max size for secrets from 10 K to 25 K Bytes. See, About keys, secrets, and certificates.

2014-12-08-preview

First preview version 2014-12-08-preview, announced on January 8, 2015.

See also

- About keys, secrets, and certificates

Azure Key Vault service limits

4/25/2019 • 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 region[1]):

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.