# Lab: Access resource secrets more securely across services

# Student lab manual

## Lab scenario

Your company has a data-sharing business-to-business (B2B) agreement with another local business in which you're expected to parse a file that's dropped off nightly. To keep things simple, the second company has decided to drop the file as a Microsoft Azure Storage blob every night. You're now tasked with devising a way to access the file and generate a secure URL that any internal system can use to access the blob without exposing the file to the internet. You have decided to use Azure Key Vault to store the credentials for the storage account and Azure Functions to write the code necessary to access the file without storing credentials in plaintext or exposing the file to the internet.

## Objectives

After you complete this lab, you will be able to:

* Create an Azure key vault and store secrets in the key vault.
* Create a server-assigned managed identity for an Azure App Service instance.
* Create a Key Vault access policy for an Azure Active Directory identity or application.
* Use the Storage .NET SDK to download a blob.

## Lab setup

* Estimated time: **45 minutes**

## Instructions

### Before you start

#### Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine by using the following credentials:

* Username: **Admin**
* Password: **Pa55w.rd**

#### Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

* Microsoft Edge
* File Explorer

### Exercise 1: Create Azure resources

#### Task 1: Open the Azure portal

1. Sign in to the Azure portal (<https://portal.azure.com>).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select **Get Started** to skip the tour.

#### Task 2: Create an Azure Storage account

1. Create a new storage account with the following details:
   * New resource group: **SecureFunction**
   * Name: **securestor*[yourname]***
   * Location: **East US**
   * Performance: **Standard**
   * Account kind: **StorageV2 (general purpose v2)**
   * Replication: **Locally-redundant storage (LRS)**
   * Access tier: **Hot**

* **Note**: Wait for Azure to finish creating the storage account before you move forward with the lab. You'll receive a notification when the account is created.

1. Open the **Access Keys** blade of your newly created storage account instance.
2. Record the value in the **Connection string** text box. You'll use this value later in this lab.

#### Task 3: Create an Azure key vault

1. Create a new key vault with the following details:
   * Existing resource group: **SecureFunction**
   * Name: **securevault*[yourname]***
   * Region: **East US**
   * Pricing tier: **Standard**

* **Note**: Wait for Azure to finish creating the key vault before you move forward with the lab. You'll receive a notification when the vault is created.

#### Task 4: Create an Azure Functions app

1. Create a new function app with the following details:
   * Existing resource group: **SecureFunction**
   * App name: **securefunc*[yourname]***
   * Publish: **Code**
   * Runtime Stack: **.NET Core**
   * Version: **3.1**
   * Region: **East US**
   * Storage account: **securestor*[yourname]***
   * Operating system: **Windows**
   * Plan: **Consumption (Serverless)**
   * Enable Application Insights: **No**

* **Note**: Wait for Azure to finish creating the function app before you move forward with the lab. You'll receive a notification when the app is created.

#### Review

In this exercise, you created all the resources that you'll use for this lab.

### Exercise 2: Configure secrets and identities

#### Task 1: Configure a system-assigned managed service identity

1. Access the **securefunc*[yourname]*** function app that you created earlier in this lab.
2. Browse to the **Identity** settings from the **Platform features** tab.
3. Enable the system-assigned managed identity, and then save your changes.

#### Task 2: Create a Key Vault secret

1. Access the **securevault*[yourname]*** key vault that you created earlier in this lab.
2. Select the **Secrets** link in the **Settings** section.
3. Create a new secret with the following settings:
   * Name: **storagecredentials**
   * Value: ***Storage connection string***
   * Enabled: **Yes**

* **Note**: Use the storage account connection string that you recorded earlier in this lab for the value of this secret.

1. Select through the secret to find the metadata for its latest version.
2. Record the value of the **Secret Identifier** text box because you'll use this later in the lab.

#### Task 3: Configure a Key Vault access policy

1. Access the **securevault*[yourname]*** key vault that you created earlier in this lab.
2. Browse to the **Access Policies** link in the **Settings** section.
3. Create a new access policy with the following settings:
   * Principal: **securefunc*[yourname]***
   * **Note**: The system-assigned managed identity you created earlier in this lab will have the same name as the Azure Function resource.
   * Key permissions: **None**
   * Secret permissions: **GET**
   * Certificate permissions: **None**
   * Authorized application: **None**
4. Save your changes to the list of **Access Policies**.

#### Review

In this exercise, you created a server-assigned managed service identity for your function app and then gave that identity the appropriate permissions to get the value of a secret in your key vault. Finally, you created a secret that you'll use within your function app.

### Exercise 3: Write function app code

#### Task 1: Create a Key Vault-derived application setting

1. Access the **securefunc*[yourname]*** function app that you created earlier in this lab.
2. Browse to the **Configuration** settings from the **Platform features** tab.
3. Create a new application setting by using the following details:
   * Name: **StorageConnectionString**
   * Value: **@Microsoft.KeyVault(SecretUri=*Secret Identifier*)**
   * Deployment slot setting: **Not selected**

* **Note**: You'll need to build a reference to your ***Secret Identifier*** by using the previous syntax. For example, if your ***Secret Identifier*** is **https://securevaultstudent.vault.azure.net/secrets/storagecredentials/17b41386df3e4191b92f089f5efb4cbf**, then your value would be **@Microsoft.KeyVault(SecretUri=https://securevaultstudent.vault.azure.net/secrets/storagecredentials/17b41386df3e4191b92f089f5efb4cbf)**

1. Save your changes to the application settings.

#### Task 2: Create an HTTP-triggered function

1. Access the **securefunc*[yourname]*** function app that you created earlier in this lab.
2. Create a new function by using the following settings:
   * Development environment: **In-portal**
   * Template: **HTTP trigger**
   * Name: **FileParser**
   * Authorization level: **Anonymous**
3. In the function editor, replace the example function script with the following placeholder C# code:

* using System.Net;  
  using Microsoft.AspNetCore.Mvc;  
    
  public static async Task<IActionResult> Run(HttpRequest req)  
  {  
   return new OkObjectResult("Test Successful");   
  }

1. Select **Save and run** to perform a test of the function. The output from the should be "Test Successful".

* **Note**: You may notice warning content during compilation in the log. This can be safely ignored.

#### Task 3: Test the Key Vault-derived application setting

1. Delete all the existing code within the **Run** method.
2. Get the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method:

* string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");

1. Return the value of the *connectionString* variable by using the **OkObjectResult** class constructor:

* return new OkObjectResult(connectionString);

1. Select **Save and run** to perform a test of the function. The output should be your storage account connection string stored in Key Vault.

* **Note**: You may notice a warning message. This is simply a C# compiler warning that says you are using an asynchronous method without asynchronous code. This warning will no longer show up later in the lab.

#### Review

In this exercise, you used a service identity to read the value of a secret stored in Key Vault and returned that value as the result of a function.

### Exercise 4: Access storage account blobs

#### Task 1: Upload a sample Storage blob

1. Access the **securestor*[yourname]*** storage account that you created earlier in this lab.
2. Select the **Containers** link in the **Blob service** section, and then create a new container with the following settings:
   * Name: **drop**
   * Public access level: **Blob (anonymous read access for blobs only)**
3. Browse to the new **drop** container, and then select **Upload** to upload the **records.json** file in the **Allfiles (F): \Allfiles\Labs\07\Starter** folder on your lab virtual machine.

* **Note:** We recommend that you enable the **Overwrite if files already exist** option.

1. Find the metadata for the **records.json** blob by selecting the blob entry in the list of blobs.
2. Using a new browser tab, go to to the URL for the blob, and then find the blob’s contents.
3. Update the container’s access level by changing the **Public access level** to **Private (no anonymous access)**.
4. Using a new browser window or tab, go to to the URL for the blob, and then find the blob’s contents. You should receive an error message indicating that the resource wasn't found.

* **Note**: If you don't receive the error message, your browser might have cached the file. Refresh the page until you receive the error message.

#### Task 2: Pull the storage account SDK from NuGet

1. Access the **securefunc*[yourname]*** function app that you created earlier in this lab.
2. Open the editor for the **FileParser** function.
3. Using the **View files** tab, create a new **function.proj** file with the following content:

* <Project Sdk="Microsoft.NET.Sdk">  
   <PropertyGroup>  
   <TargetFramework>netstandard2.0</TargetFramework>  
   </PropertyGroup>  
   <ItemGroup>  
   <PackageReference Include="Azure.Storage.Blobs" Version="12.4.0" />  
   </ItemGroup>  
  </Project>

1. **Save** the newly created **function.proj** file.

* **Note**: This .proj file contains the NuGet package reference necessary to import the [Azure.Storage.Blobs](https://www.nuget.org/packages/Azure.Storage.Blobs/12.4.0) package.

1. Find the contents of the **function.proj** file by selecting the file from the **View files** tab.
2. Return to the editor for the **FileParser** function by selecting the **run.csx** file from the **View files** tab.
3. Add three **using** directives for the **Azure.Storage**, **Azure.Storage.Blobs**, and **Azure.Storage.Blobs.Models** namespaces.
4. Delete all the existing code within the **Run** method.

#### Task 3: Write storage account code

1. Get the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method:

* string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");

1. Create a new instance of the **BlobServiceClient** class by passing in your *connectionString* variable to the constructor:

* BlobServiceClient serviceClient = new BlobServiceClient(connectionString);

1. Use the **BlobServiceClient.GetBlobContainerClient** method while passing in the **drop** container name to create a new instance of the **BlobContainerClient** class that references the container that you created earlier in this lab:

* BlobContainerClient containerClient = serviceClient.GetBlobContainerClient("drop");

1. Use the **BlobContainerClient.GetBlobClient** method while passing in the **records.json** blob name to create a new instance of the **BlobClient** class that references the blob that you uploaded earlier in this lab:

* BlobClient blobClient = containerClient.GetBlobClient("records.json");

#### Task 4: Download a blob

1. Use the **BlobClient.DownloadAsync** method to download the contents of the referenced blob asynchronously and store the result in a variable named *response*:

* var response = await blobClient.DownloadAsync();

1. Return the value of the various content stored in the *content* variable by using the **FileStreamResult** class constructor:

* return new FileStreamResult(response?.Value?.Content, response?.Value?.ContentType);

1. Select **Save and run** to perform a test of the function. The output should be the content of the **$/drop/records.json** blob stored in your storage account.

#### Review

In this exercise, you used C# code to access a storage account, and then download the contents of a blob.

### Exercise 5: Clean up your subscription

#### Task 1: Open Azure Cloud Shell and list resource groups

1. In the portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for Bash by using the default settings.
3. At the **Cloud Shell** command prompt in the portal, enter the following command, and then select Enter to list all resource groups in the subscription:

* az group list

1. Enter the following command, and then select Enter to find a list of possible commands to delete a resource group:

* az group delete --help

#### Task 2: Delete a resource group

1. Enter the following command, and then select Enter to delete the **SecureFunction** resource group:

* az group delete --name SecureFunction --no-wait --yes

1. Close the Cloud Shell pane from the portal.

#### Task 3: Close the active application

1. the currently running Microsoft Edge application.

#### Review

In this exercise, you cleaned up your subscription by removing the resource groups that were used in this lab.

©2019 Microsoft Corporation. All rights reserved. The text in this document is available under the [Creative Commons Attribution 3.0 License](https://creativecommons.org/licenses/by/3.0/legalcode), additional terms may apply. All other content contained in this document (including, without limitation, trademarks, logos, images, etc.) are ***not*** included within the Creative Commons license grant. This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it. Some examples are for illustration only and are fictitious. No real association is intended or inferred. Microsoft makes no warranties, express or implied, with respect to the information provided here.