# Lab 04: Access resource secrets securely across services

V 1.4

## Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure user interface (UI) changes after the development of this training content. These changes might cause the lab instructions and lab steps to not match up.

Microsoft updates this training course as soon as the community brings needed changes to our attention. However, because cloud updates occur frequently, you might encounter UI changes before this training content is updated. **If this occurs, adapt to the changes and work through them in the labs as needed.**

## Instructions

### Before you start

#### Review installed applications

* Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications you will use in this lab:
  + Microsoft Edge
  + File Explorer

### Lab files

* Lab files located in the folder **Starter**

### Exercise 1: Create Azure resources

#### Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, navigate to the **Azure portal** ([portal.azure.com](https://portal.azure.com)).
3. Enter the **email address** for your Microsoft account.
4. Select **Next**.
5. Enter the **password** for your Microsoft account.
6. Select **Sign in**.

Note: If this is your first time signing into the **Azure Portal**, a dialog box will appear offering a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

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

1. In the navigation pane on the left side of the Azure portal, select **All services**.
2. In the **All services** blade, select **Storage Accounts**.
3. In the **Storage accounts** blade, view your list of Storage instances.
4. At the top of the **Storage accounts** blade, select **Add**.
5. In the **Create storage account** blade, observe the tabs at the top of the blade, such as **Basics**.

Note: Each tab represents a step in the workflow to create a new **storage account**. At any time, you can select **Review + create** to skip the remaining tabs.

1. In the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **Create new**, enter **SecureFunction**, and then select **OK**.
   3. In the **Storage account** **name** text box, enter **securestor[your name in lowercase]**.
   4. In the **Location** drop-down list, select the **East US** region.
   5. In the **Performance** section, select **Standard**.
   6. In the **Account kind** drop-down list, select **StorageV2 (general purpose v2)**.
   7. In the **Replication** drop-down list, select **Locally-redundant storage (LRS)**.
   8. In the **Access tier** section, ensure that **Hot** is selected.
   9. Select **Review + Create**.
2. In the **Review + Create** tab, review the options that you selected during the previous steps.
3. Select **Create** to create the storage account by using your specified configuration.
4. Wait for the creation task to complete before you move forward with this lab.
5. In the navigation pane on the left side of the Azure portal, select **All services**.
6. In the **All services** blade, select **Storage Accounts**.
7. In the **Storage accounts** blade, select the storage account instance with the prefix **securestor**.
8. In the **Storage account** blade, locate the **Settings** section on the left side of the blade and select the **Access keys** link.
9. In the **Access keys** blade, select any one of the keys and record the value in either of the **Connection string** fields. You will use this value later in this lab.

**Note**: It does not matter which connection string you choose to use. They are interchangeable.

#### Task 3: Create an Azure Key Vault

1. On the navigation menu located on the left side of the portal, select the **+ Create a resource** link.
2. At the top of the **New** blade, locate the **Search the Marketplace** text box above the list of featured services.
3. In the search text box, enter **Vault** and then press Enter.
4. In the **Everything** search results blade, select the **Key Vault** result.
5. In the **Key Vault** blade, select **Create**.
6. In the **Create key vault** blade, perform the following actions:
   1. In the **Name** text box, enter **securevault[your name in lowercase]**.
   2. Leave the **Subscription** text box set to its default value.
   3. In the **Resource group** section, select **Use existing**, and then select **SecureFunction** from the list.
   4. In the **Location** drop-down list, select **East US**.
   5. Leave the **Pricing tier** text box set to its default value.
   6. Leave the **Access policies** text box set to its default value.
   7. Leave the **Virtual Network Access** text box set to its default value.
   8. Select **Create**.
7. Wait for the creation task to complete before you move forward with this lab.

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

1. On the navigation menu located on the left side of the portal, select the **+ Create a resource** link.
2. At the top of the **New** blade, locate the **Search the Marketplace** text box above the list of featured services.
3. In the search text box, enter **Function** and then press Enter.
4. In the **Everything** search results blade, select the **Function App** result.
5. In the **Function App** blade, select **Create**.
6. In the **Create function app** blade, perform the following actions:
   1. In the **App name** text box, enter **securefunc[your name in lowercase]**.
   2. Leave the **Subscription** text box set to its default value.
   3. In the **Resource group** section, select **Use existing**, and then select **SecureFunction** from the list.
   4. In the **OS** section, select **Windows**.
   5. In the **Hosting Plan** drop-down list, select **Consumption Plan**.
   6. In the **Location** drop-down list, select **East US**.
   7. In the **Runtime Stack** drop-down list, select **.NET Core**.
   8. In the **Storage** section, select **Use existing**, and then select **securestor[your name in lowercase]** from the list.
   9. Leave the **Application Insights** text box set to its default value.
   10. Select **Create**.
7. Wait for the creation task to complete before you move forward with this lab.

#### Review

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

### Exercise 2: Configure secrets and identities

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

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securefunc\*** function app that you created earlier in this lab.
4. In the **Function Apps** blade, select the **Platform features** tab.
5. In the **Platform features** tab, select the **Identity** link located in the **Networking** section.
6. In the **Identity** blade, locate the **System assigned** tab and then perform the following actions:
   1. In the **Status** section, select **On**.
   2. Select **Save**.
   3. Select **Yes** in the confirmation dialog.
7. Wait for the system-assigned managed identity to be created before you move forward with this lab.

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

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securevault\*** Key Vault that you created earlier in this lab.
4. In the **Key Vault** blade, select the **Secrets** link located in the **Settings** section.
5. In the **Secrets** pane, select **Generate/Import**.
6. In the **Create a secret** blade, perform the following actions:
   1. In the **Upload options** drop-down list, select **Manual**.
   2. In the **Name** text box, enter **storagecredentials**.
   3. In the **Value** text box, enter the storage account **Connection String** that you recorded earlier in this lab.
   4. Leave the **Content Type** text box set to its default value.
   5. Leave the **Set activation date** text box set to its default value.
   6. Leave the **Set expiration date** text box set to its default value.
   7. In the **Enabled** section, select **Yes**.
   8. Select **Create**.
7. Wait for the secret to be created before you move forward with this lab.
8. Back in the **Secrets** pane, select the **storagecredentials** item in the list.
9. In the **Versions** pane, select the latest version of the **storagecredentials** secret.
10. In the **Secret Version** pane, perform the following actions.
    1. Observe the metadata for the latest version of the secret.
    2. Select **Show secret value** to view the value of the secret.
    3. Record the value of the **Secret Identifier** text box because you will use this later in the lab.

Note: You are recording the value of the **Secret Identifier** field, not the **Secret Value** field.

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

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securevault\*** Key Vault that you created earlier in this lab.
4. In the **Key Vault** blade, select the **Access policies** link located in the **Settings** section.
5. In the **Access policies** pane, select **Add new**.
6. In the **Add access policy** blade, perform the following actions:
   1. Select the **Select principal** link.
   2. In the **Principal** blade, locate and select the service principal named **securefunc[your name in lowercase]**, and then select **Select**.
   3. Leave the **Key permissions** list set to its default value.
   4. In the Secret permissions drop-down list, select the GET permission.
   5. Leave the **Certificate permissions** list set to its default value.
   6. Leave the **Authorized application** text box set to its default value.
   7. Select **OK**.
7. Back in the **Access policies** pane, select **Save**.
8. Wait for your changes to the access policies to be saved before you move forward with this lab.

#### 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 will use within your function app.

### Exercise 3: Write function app code

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

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securefunc\*** function app that you created earlier in this lab.
4. In the **Function App** blade, select the **Platform features** tab.
5. In the **Platform features** tab, select the **Configuration** link located in the **General Settings** section.
6. In the **Application Settings** tab, perform the following actions:
   1. Scroll down until you see the **Application settings** section.
   2. Select **+ New application setting**.
   3. In the **Enter a name** text box, enter **StorageConnectionString**.
   4. In the **Enter a value** text box, construct a value by using the following syntax: **@Microsoft.KeyVault(SecretUri=\)**

Note: You will need to build a reference to your **Secret Identifier** by using the above 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. Leave the **Slot Setting** text box set to its default value.
2. Scroll back to the top of the tab and then select **Save**.
3. Wait for your application settings to persist before you move forward with this lab.

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

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securefunc\*** function app that you created earlier in this lab.
4. In the **Function App** blade, select **+ New function**.
5. In the **New Azure Function** getting started, perform the following actions:
   1. Select **In-Portal** function
   2. Under the **Choose a Function** header, select **More templates…**.
   3. Select **Finish and view templates**.
   4. In the **Templates** drop-down list, select **HTTP trigger**.
   5. In the **New Function** pop-up, locate the **Name** text box and enter **FileParser**.
   6. In the **New Function** pop-up, locate the **Authorization level** list and select **Anonymous**.
   7. In the **New Function** pop-up, select **Create**.
6. In the function editor, observe the example function script:

#r "Newtonsoft.Json"

using System.Net;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Primitives;

using Newtonsoft.Json;

public static async Task Run(HttpRequest req, ILogger log)

{

log.LogInformation("C# HTTP trigger function processed a request.");

string name = req.Query["name"];

string requestBody = await new StreamReader(req.Body).ReadToEndAsync();

dynamic data = JsonConvert.DeserializeObject(requestBody);

name = name ?? data?.name;

return name != null

? (ActionResult)new OkObjectResult($"Hello, {name}")

: new BadRequestObjectResult("Please pass a name on the query string or in the request body");

}

1. **Run the function and make sure that function executed without exception.**
2. On the right side of the editor, select **View files** to open the tab.
3. In the **View files** tab, select **+ Add**.
4. Create new file named **function.proj**
5. Provide following code in the file

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<TargetFramework>netstandard2.0</TargetFramework>

<AzureFunctionsVersion>v2</AzureFunctionsVersion>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Microsoft.AspNetCore.Mvc" Version="2.2.0" />

<PackageReference Include="Microsoft.Azure.Storage.Common" Version="10.0.3" />

<PackageReference Include="Microsoft.Azure.Storage.Blob" Version="10.0.3" />

</ItemGroup>

</Project>

1. From the **console** in the bottom where is **log** located type: **dotnet restore**
2. Wait until restoring will be completed. You will see the packages loaded. It should not fail with error.
3. Select the **run.csx** file to return to the editor for the **FileParser** function.
4. Minimize the **View files** tab.

Note: You can minimize the tab by selecting the arrow immediately to the right of the tab header.

1. **Delete** all the example code.
2. Within the function editor, copy and paste the following placeholder function:

using System.Net;

using Microsoft.AspNetCore.Mvc;

public static async Task<OkObjectResult> Run(HttpRequest req)

{

return new OkObjectResult("Test Successful");

}

1. Select **Save and Run** to save the script and perform a test execution of the function.
2. The **Test** and **Logs** panes will automatically appear when the script executes for the first time.
3. Observe the **Output** text box in the **Test** pane. You should now see the value **Test Successful** returned from the function.

#### Task 3: Test an application setting

1. Delete the existing code within the **Run** method of the script.
2. The **Run** method should now look like this:

using System.Net;

using Microsoft.AspNetCore.Mvc;

public static async Task<OkObjectResult> Run(HttpRequest req)

{

}

1. Add the following line of code to get the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method:

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

1. Add the following line of code to return the value of the **connectionString** variable by using the **OkObjectResult** class constructor:

return new OkObjectResult(connectionString);

1. The **Run** method should now look like this:

using System.Net;

using Microsoft.AspNetCore.Mvc;

public static async Task Run(HttpRequest req)

{

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

return new OkObjectResult(connectionString);

}

1. Select **Save and run** to save the script and perform a test execution of the function.
2. Observe the **Output** text box in the **Test** pane. You should now see the connection string returned from the function.

#### Review

In this exercise, you securely used a service identity to read the value of a secret stored in **Azure Key Vault** and return that value as the result of an **Azure Function**.

### Exercise 4: Access Storage Account blobs

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

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securestor\*** storage account that you created earlier in this lab.
4. In the **Storage account** blade, select the **Blobs** link located in the **Blob service** section on the left side of the blade.
5. In the **Blobs** section, select **+ Container**.
6. In the **New container** pop-up, perform the following actions:
   1. In the **Name** text box, enter **drop**.
   2. In the **Public access level** drop-down list, select **Blob (anonymous read access for blobs only)**.
   3. Select **OK**.
7. Back in the **Blobs** section, select the newly created **drop** container.
8. In the **Container** blade, select **Upload**.
9. In the **Upload blob** pop-up, perform the following actions:
   1. In the **Files** section, select the **Folder** icon.
   2. In the File Explorer dialog box, go to \**Starter**, select the **records.json** file, and then select **Open**.
   3. Ensure that **Overwrite if files already exist** is selected.
   4. Select **Upload**.
10. Wait for the blob to be uploaded before you continue with this lab.
11. Back in the **Container** blade, select the **records.json** blob from the list of blobs.
12. In the **Blob** blade, view the blob metadata.
13. Copy the **URL** for the blob.
14. On the taskbar, right-select the **Microsoft Edge** icon and then select **New window**.
15. In the new browser window, navigate to the **URL** that you copied for the blob.
16. You should now see the **JSON** contents of the blob. Close the browser window showing the **JSON** contents.
17. Return to the browser window with the **Azure portal.**
18. Close the **Blob** blade.
19. Back in the **Container** blade, select **Change access level policy** located at the top of the blade.
20. In the **Change access level** pop-up that appears, perform the following actions:
    1. In the **Public access level** drop-down list, select **Private (no anonymous access)**.
    2. Select **OK**.
21. On the taskbar, right-select the **Microsoft Edge** icon and then select **New window**.
22. In the new browser window, navigate to the **URL** that you copied for the blob.
23. You should now see an error message indicating that the resource was not found.

Note: If you do not see the error message, your browser might have cached the file. Use **Ctrl+F5** to refresh the page until you see the error message.

#### Task 2: Configure a Storage Account SDK

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securefunc\*** function app that you created earlier in this lab.
4. In the **Function App** blade, locate and select the existing **FileParser** function to open the editor for the function.

Note: You might need to expand the **Functions** option in the menu on the left side of the blade.

1. On the right side of the editor, select **View files** to open the tab.
2. Minimize the **View files** tab.

Note: You can minimize the tab by selecting the arrow immediately to the right of the tab header.

1. Within the editor, delete the existing code within the **Run** method of the script.
2. At the top of the code file, add the following line of code to create a **using** block for the **Microsoft.WindowsAzure.Storage** namespace:

using Microsoft.Azure.Storage;

1. Add the following line of code to create a **using** block for the **Microsoft.WindowsAzure.Storage.Blob** namespace:

using Microsoft.Azure.Storage.Blob;

1. The **Run** method should now look like this:

using System;

using System.Net;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.Storage;

using Microsoft.Azure.Storage.Blob;

public static async Task Run(HttpRequest req)

{

}

#### Task 3: Write storage account code

1. Add the following line of code within the **Run** method to get the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method:

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

1. Add the following line of code to create a new instance of the **CloudStorageAccount** class by using the **CloudStorageAccount.Parse** static method, passing in your connectionString variable:

CloudStorageAccount account = CloudStorageAccount.Parse(connectionString);

1. Add the following line of code to use the **CloudStorageAccount.CreateCloudBlobClient** method to create a new instance of the **CloudBlobClient** class that will give you access to blobs in your storage account:

CloudBlobClient blobClient = account.CreateCloudBlobClient();

1. Add the following line of code to use the **CloudBlobClient.GetContainerReference** method, while passing in the **drop** container name to create a new instance of the **CloudBlobContainer** class that references the container that you created earlier in this lab:

CloudBlobContainer container = blobClient.GetContainerReference("drop");

1. Add the following line of code to use the **CloudBlobContainer.GetBlockBlobReference** method, while passing in the **records.json** blob name to create a new instance of the **CloudBlockBlob** class that references the blob that you uploaded earlier in this lab:

CloudBlockBlob blob = container.GetBlockBlobReference("records.json");

1. The **Run** method should now look like this:

using System;

using System.Net;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.Storage;

using Microsoft.Azure.Storage.Blob;

public static async Task Run(HttpRequest req)

{

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

CloudStorageAccount account = CloudStorageAccount.Parse(connectionString);

CloudBlobClient blobClient = account.CreateCloudBlobClient();

CloudBlobContainer container = blobClient.GetContainerReference("drop");

CloudBlockBlob blob = container.GetBlockBlobReference("records.json");

}

#### Task 4: Download a blob

1. Add the following line of code to use the **CloudBlockBlob.DownloadTextAsync** method to download the contents of the referenced blob asynchronously and store the result in a string variable named content:

string content = await blob.DownloadTextAsync();

1. Add the following line of code to return the value of the content variable by using the **OkObjectResult** class constructor. Also change the return type of the Run function to Task<OkObjectResult>:

return new OkObjectResult(content);

1. The **Run** method should now look like this:

using System;

using System.Net;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.Storage;

using Microsoft.Azure.Storage.Blob;

public static async Task<OkObjectResult> Run(HttpRequest req)

{

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

CloudStorageAccount account = CloudStorageAccount.Parse(connectionString);

CloudBlobClient blobClient = account.CreateCloudBlobClient();

CloudBlobContainer container = blobClient.GetContainerReference("drop");

CloudBlockBlob blob = container.GetBlockBlobReference("records.json");

string content = await blob.DownloadTextAsync();

return new OkObjectResult(content);

}

1. Select **Save and run** to save the script and perform a test execution of the function.
2. Observe the **Output** text box in the **Test** pane. You should now see the content of the **$/drop/records.json** blob stored in your **storage account**.

#### Task 5: Generate a Shared Access Signature (SAS)

1. **Replace** the following lines of code:

string content = await blob.DownloadTextAsync();

return new OkObjectResult(content);

1. With the following line of code to create a new instance of the **SharedAccessPolicy** class with the following settings:
   1. **Permission**: Read
   2. **Service Scope**: Blob
   3. **Resource Types**: Object
   4. **Expiration Time**: 2 Hours
   5. **Protocol:** HTTPS Only

SharedAccessAccountPolicy policy = new SharedAccessAccountPolicy()

{

Permissions = SharedAccessAccountPermissions.Read,

Services = SharedAccessAccountServices.Blob,

ResourceTypes = SharedAccessAccountResourceTypes.Object,

SharedAccessExpiryTime = DateTime.UtcNow.AddHours(2),

Protocols = SharedAccessProtocol.HttpsOnly

};

1. Add the following line of code to use the **CloudStorageAccount.GetSharedAcessSignature** method to generate a new **Shared Access Signature (SAS) token** by using the provided policy, and then store the result in a string variable named sasToken:

string sasToken = account.GetSharedAccessSignature(policy);

1. Add the following line of code to concatenate the value of the **CloudBlockBlob.Uri** property and the sasToken string variable, and store the result in a new variable named secureBlobUrl:

string secureBlobUrl = $"{blob.Uri}{sasToken}";

1. Add the following line of code to return the value of the secureBlobUrl variable by using the **OkObjectResult** class constructor:

return new OkObjectResult(secureBlobUrl);

1. The **Run** method should now look like this:

using System;

using System.Net;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.Storage;

using Microsoft.Azure.Storage.Blob;

public static async Task<OkObjectResult> Run(HttpRequest req)

{

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

CloudStorageAccount account = CloudStorageAccount.Parse(connectionString);

CloudBlobClient blobClient = account.CreateCloudBlobClient();

CloudBlobContainer container = blobClient.GetContainerReference("drop");

CloudBlockBlob blob = container.GetBlockBlobReference("records.json");

SharedAccessAccountPolicy policy = new SharedAccessAccountPolicy()

{

Permissions = SharedAccessAccountPermissions.Read,

Services = SharedAccessAccountServices.Blob,

ResourceTypes = SharedAccessAccountResourceTypes.Object,

SharedAccessExpiryTime = DateTime.UtcNow.AddHours(2),

Protocols = SharedAccessProtocol.HttpsOnly

};

string sasToken = account.GetSharedAccessSignature(policy);

string secureBlobUrl = $"{blob.Uri}{sasToken}";

return new OkObjectResult(secureBlobUrl);

}

1. Select **Save and run** to save the script and perform a test execution of the function.
2. Observe the **Output** text box in the **Test** pane. You should now see a unique **Secure URL** that can be used to access the secured blob. Record this **URL** because you will need to use it in the next step of this lab.
3. On the taskbar, right-select the **Microsoft Edge** icon and then select **New window**.
4. In the new browser window, navigate to the **Secure** **URL** that you copied for the blob.
5. You should now see the **JSON** contents of the blob. Close the browser window showing the **JSON** contents.
6. Return to the browser window with the **Azure portal.**

#### Review

In this exercise, you used C# code to access a storage account securely, download the contents of a blob, and then generate a SAS token that can be used to access the blob securely on another client.

### Exercise 5: Clean up subscription

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

1. In the top navigation bar in the Azure Portal, select the **Cloud Shell** icon to open a new shell instance.
2. In the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press Enter to list all resource groups in the subscription:

az group list

1. In the prompt, type the following command and press Enter to view a list of possible commands to delete a resource group:

az group delete --help

#### Task 2: Delete resource group

1. In the prompt, type the following command and press Enter to delete the **SecureFunction** resource group:

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

1. Close the **Cloud Shell** pane at the bottom of the portal.

#### Task 3: Close active application

Close the currently running **Microsoft Edge** application.

#### Review

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