Module 07 (CSS): Implement secure cloud solutions

1 Hr 5 Min Remaining

Instructions Resources Help  100%

Required Lab Setup

Sign in to the lab virtual machine

1. Hello Altaf Hussain, log on to [AZ-204T00A-SEA-DEV](https://labclient.labondemand.com/Instructions/f77ce42e-9785-4e77-b9af-37d2de53c47d?rc=10) click [Ctrl+Alt+Delete](https://labclient.labondemand.com/Instructions/f77ce42e-9785-4e77-b9af-37d2de53c47d?rc=10) to activate the **Ctrl + Alt + Delete** sequence and bring up the logon page.
2. Sign in as Admin with the password Pa55w.rd

Any links like the one above will send Ctrl+Alt+Delete to the selected machine. This can also be done the **Commands** menu (lightning bolt) in the upper-left hand corner of the screen.

The **Lightning Bolt** in the upper left hand of the screen can also be used to **Copy /Paste** strings and sentences from the Instructions **into the VM** rather than typing them out.

* + To accomplish this simply Highligh\*\*t the sentence in the instructions you wish to type text. **CTRL + C** (Copy)
  + Click the lightning bolt icon, in the Dropdown menu select **Type Text** > **Type Clipboard Text**
  + Click into the Type Text window, **CTRL + V** (Paste) to Paste the sentence into the window
  + Select OK

This Lab supports **Redirect Clipboard** functionality in addition to TypeText.

You can use **Redirect Clipboard** to quickly input code blocks and other strings from the Instructions and elsewhere into the virtual machine using standard copy and paste from your local machine's Clipboard directly into the VM. You may need to click **Allow** in your browser to allow access to your local Clipboard.

Note that due to an issue within **Azure Cloudshell**, use your mouse: **Right-click + Paste** instead of **CTRL+V** when using **Cloudshell** inside the VM.

Sign in to Azure

1. Sign in to the Azure Portal https://portal.azure.com using the below credentials:

|  |  |
| --- | --- |
| Username | LabUser-23843094@LODSPRODMCA.onmicrosoft.com |
| Password | gGpA4e!0D@ |

At the end of this lab, you can skip the **Clean Up** exercise directing you to remove the resources from your Subscription or Resource Group(s). The clean up is handled automatically, after ending your lab.

Download the lab files

Download Files

This will copy the latest lab files from GitHub and configure the directories as needed for the remainder of the lab exercises. You will see a 'Success!' message above, once the process is complete.

Lab Credentials:

* Username: LabUser-23843094@LODSPRODMCA.onmicrosoft.com
* Password: gGpA4e!0D@

Sites used:

* https://portal.azure.com/#home
* https://shell.azure.com

Lab 07: Access resource secrets more securely across services

Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes that occur after this training content's development. As a result, the lab instructions and lab steps might not align correctly.

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

Instructions

Before you start

Sign in to the lab environment

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

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

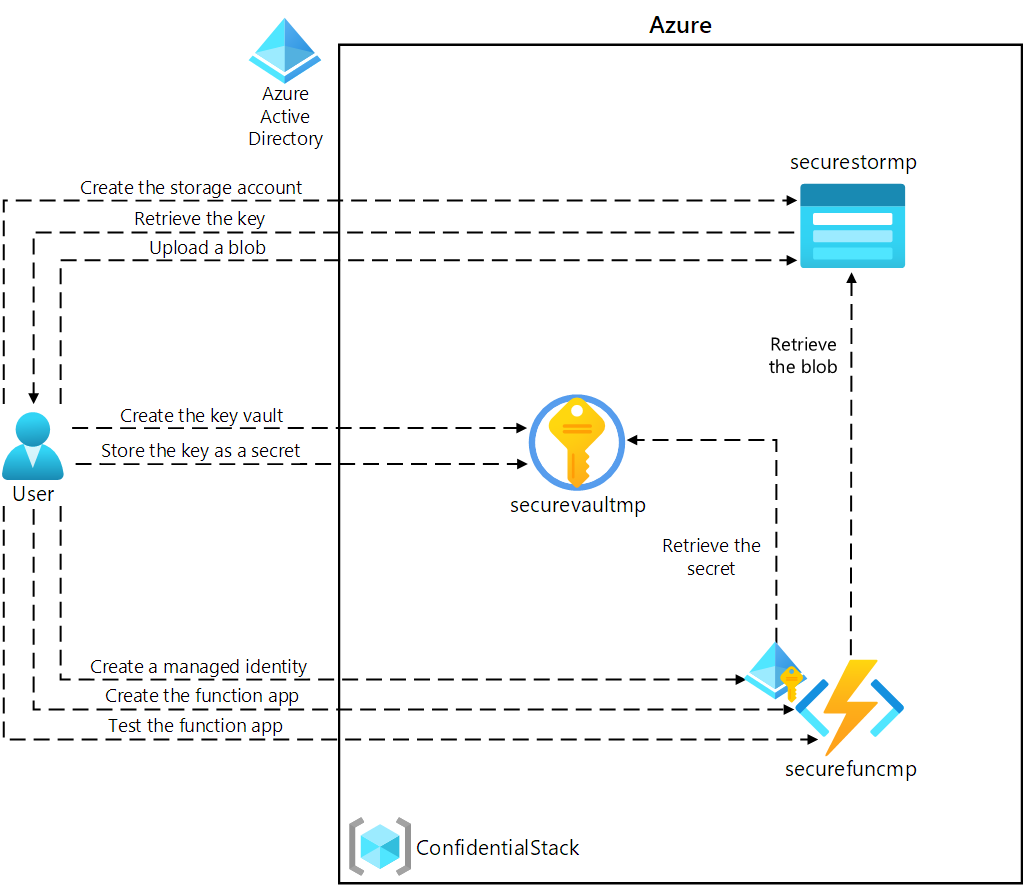
**Note**: Your instructor will provide instructions to connect to the virtual lab environment.

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, including:

* Microsoft Edge
* File Explorer
* Windows Terminal
* Visual Studio Code

Architecture diagram



Exercise 1: Create Azure resources

Task 1: Open the Azure portal

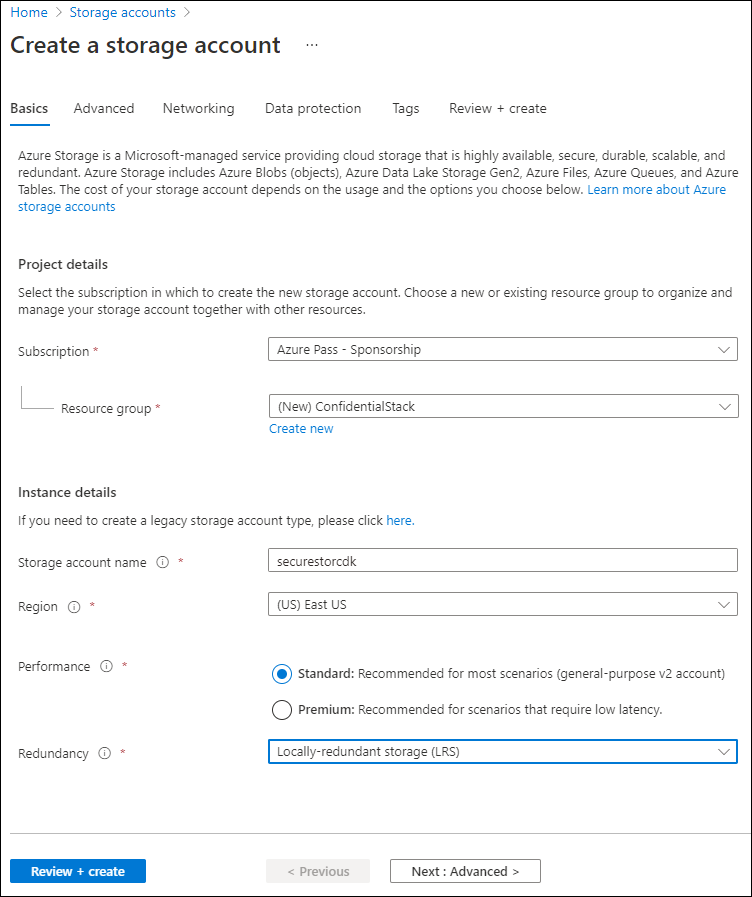
1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, browse to the Azure portal https://portal.azure.com, and then sign in with the account you'll be using for this lab.

**Note**: If this is your first time signing in to the Azure portal, you'll be offered a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

Task 2: Create a Storage account

1. In the Azure portal, use the **Search resources, services, and docs** text box to search for **Storage Accounts**, and then in the list of results, select **Storage accounts**.
2. On the **Storage accounts** blade, select **+ Create**.
3. On the **Create a storage account** blade, on the **Basics** tab, perform the following actions, and select **Review + create**:

| **Setting** | **Action** |
| --- | --- |
| **Subscription** drop-down list | Retain the default value |
| **Resource group** section | Select **Create new**, enter **ConfidentialStack**, and then select **OK** |
| **Storage account name** text box | Enter **securestor***[yourname]* |
| **Region** drop-down list | Select **(US) East US** |
| **Performance** section | Select the **Standard** option |
| **Redundancy** drop-down list | select **Locally-redundant storage (LRS)** |

1. The following screenshot displays the configured settings on the **Create a storage account** blade.
2. 
3. On the **Review + create** tab, review the options that you selected during the previous steps.
4. Select **Create** to create the storage account by using your specified configuration.

**Note**: Wait for the creation task to complete before you move forward with this lab.

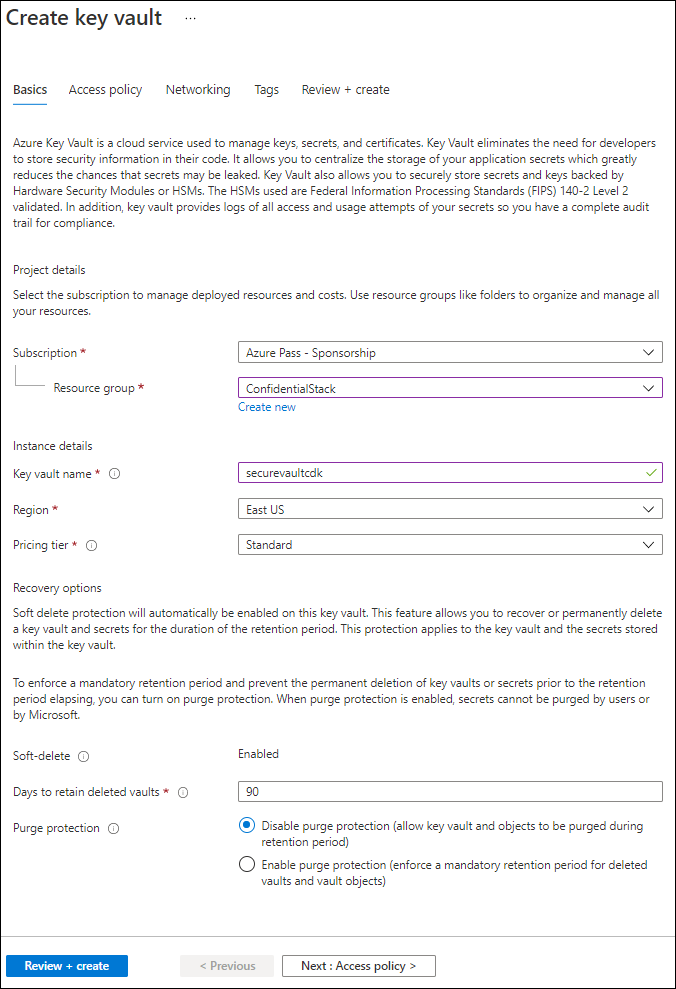
1. On the **Deployment Overview** blade, select **Go to resource**.
2. On the **Storage account** blade, in the **Security + networking** section, select the **Access keys** link.
3. In the **Access keys** section, select **Show keys**.
4. Select any one of the keys and record the value in either of the **Connection string** boxes. You'll use this value later in this lab.

**Note**: It doesn't matter which connection string you choose. They're interchangeable.

Task 3: Create an Azure Key Vault

1. In the Azure portal, use the **Search resources, services, and docs** text box to search for **Key vaults**, and then in the list of results, select **Key vaults**.
2. On the **Key vaults** blade, select **Create**.
3. On the **Create key vault** blade, on the **Basics** tab, perform the following actions, and then select **Review + create**:

| **Setting** | **Action** |
| --- | --- |
| **Subscription** drop-down list | Retain the default value. |
| **Resource group** drop-down list | Select **ConfidentialStack** in the list. |
| **Key vault name** text box | Enter **securevault***[yourname]*. |
| **Region** drop-down list | Select **East US**. |
| **Pricing tier** drop-down list | select **Standard**. |

1. The following screenshot displays the configured settings on the **Create key vault** blade.
2. 
3. On the **Review + create** tab, review the options that you selected during the previous steps.
4. Select **Create** to create the key vault by using your specified configuration.

**Note**: Wait for the creation task to complete before you move forward with this lab.

Task 4: Create a Function app

1. In the Azure portal, use the **Search resources, services, and docs** text box to search for **Function App**, and then in the list of results, select **Function App**.
2. On the **Function App** blade, select **Create**.
3. On the **Create Function App** blade, on the **Basics** tab, perform the following actions, and then select **Next: Hosting**:

| **Setting** | **Action** |
| --- | --- |
| **Subscription** drop-down list | Retain the default value |
| **Resource group** drop-down list | Select **ConfidentialStack** |
| **Function App name** text box | Enter **securefunc***[yourname]* |
| **Publish** section | Select **Code** |
| **Runtime stack** drop-down list | Select **.NET** |
| **Version** drop-down list | Select **3.1** |
| **Region** drop-down list | Select the **East US** region. |
| **Operating system** radio buttons | **Select Linux** |
| **Plan type** drop-down list | Select **Consumption (Serverless)** |

1. On the **Hosting** tab, perform the following actions, and then select **Review + create**:

| **Setting** | **Action** |
| --- | --- |
| **Storage account** drop-down list | Select the **securestor***[yourname]* storage account |

1. On the **Review + create** tab, review the options that you selected during the previous steps.
2. Select **Create** to create the function app by using your specified configuration.

**Note**: 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'll use in this lab.

Exercise 2: Configure secrets and identities

Task 1: Configure a system-assigned managed service identity

1. On the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, select the **ConfidentialStack** resource group.
3. On the **ConfidentialStack** blade, select the **securefunc***[yourname]* function app.

**Note**: There will be two resources, a function app and application insights resource, with the same name. Make sure you select the function app resource.

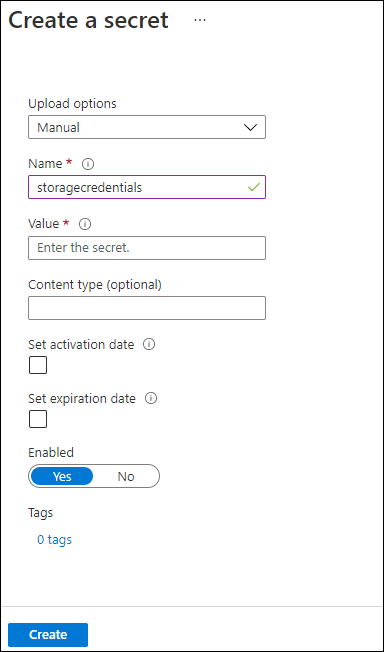
1. On the **Function App** blade, select the **Identity** option from the **Settings** section.
2. On the **Identity** pane, on the **System assigned** tab, set the **Status** to **On**, and then select **Save**.
3. Select **Yes** to confirm the setting.

**Note**: 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 Azure portal's **navigation** pane, select the **Resource groups** link.
2. On the **Resource groups** blade, select the **ConfidentialStack** resource group.
3. On the **ConfidentialStack** blade, select the **securevault***[yourname]* key vault.
4. On the **Key Vault** blade, select the **Secrets** link in the **Settings** section.
5. On the **Secrets** pane, select **+ Generate/Import**.
6. On the **Create a secret** blade, perform the following actions, and then select **Create**:

| **Setting** | **Action** |
| --- | --- |
| **Upload options** drop-down list | Select **Manual** |
| **Name** text box | Enter **storagecredentials** |
| **Value** text box | Enter the storage account connection string that you recorded previously in this lab. |
| **Content type** text box | Leave blank. |
| **Set activation date** check box | Not selected. |
| **Set expiration date** check box | Not selected. |
| **Enabled** option | Select **Yes**. |

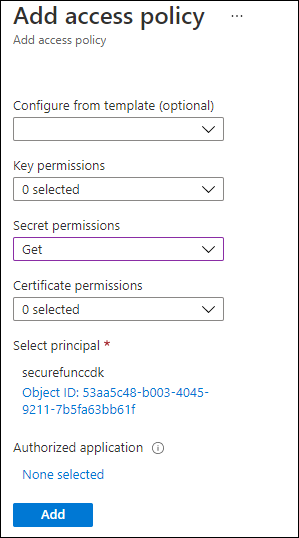
1. The following screenshot displays the configured settings on the **Create a secret** blade.
2. 
3. **Note**: Wait for the secret to be created before you move forward with this lab.
4. Return to the **Secrets** pane, and then select the **storagecredentials** item in the list.
5. On the **Versions** pane, select the latest version of the **storagecredentials** secret.
6. On the **Secret Version** pane, perform the following actions:
   1. Select **Show secret value** to find the value of the secret.
   2. Record the value of the **Secret Identifier** text box because you'll use this later in the lab.

**Note**: You're recording the value of the **Secret Identifier** text box, not the **Secret Value** text box.

Task 3: Configure a Key Vault access policy

1. On the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, select the **ConfidentialStack** resource group.
3. On the **ConfidentialStack** blade, select the **securevault[yourname]** key vault.
4. On the **Key vault** blade, select the **Access policies** link in the **Settings** section.
5. On the **Access policies** pane, select **Add Access Policy**.
6. On the **Add access policy** blade, perform the following actions, and then select **Add**:

| **Setting** | **Action** |
| --- | --- |
| **Configure from template** drop-down list | Leave blank |
| **Key permissions** drop-down list | 0 selected |
| **Secret permissions** drop-down list | Select the **GET** permission |
| **Certificate permissions** drop-down list | 0 selected |
| **Select principal** link | Find and then select the service principal named **securefunc***[yourname]*. The system-assigned managed identity you created previously in this lab will have the same name as the Azure Function resource. |
| **Authorized application** link | None selected |

1. The following screenshot displays the configured settings in the **Add access policy** blade.
2. 
3. On the **Access policies** pane, select **Save**.

**Note**: Wait for your changes to the access policies to save before you continue with this lab.

Task 4: Create a Key Vault-derived application setting

1. On the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, select the **ConfidentialStack** resource group.
3. On the **ConfidentialStack** blade, select the **securefunc[yourname]** function app.
4. On the **Function App** blade, select the **Configuration** option from the **Settings** section.
5. On the **Configuration** pane, on the **Application settings** tab, select **New application setting**.
6. In the **Add/Edit application setting** pop-up window, in the **Name** text box, enter **StorageConnectionString**.
7. In the **Value** text box, construct a value by using the following syntax: @Microsoft.KeyVault(SecretUri=\*Secret Identifier\*)

**Note**: You'll 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, your value would be @Microsoft.KeyVault(SecretUri=https://securevaultstudent.vault.azure.net/secrets/storagecredentials/17b41386df3e4191b92f089f5efb4cbf).

1. Leave the **deployment slot setting** check box set to its default value (not selected), and then select **OK** to close the pop-up window and to return to the **Configuration** section.
2. Select **Save** to save your settings, and then in the **save Changes** confirmation pop-up dialog box, select **Continue**.

**Note**: Wait for your application settings to save before you continue with the lab.

Review

In this exercise, you created a system-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 referenced within your function app's configuration settings.

Exercise 3: Build an Azure Functions app

Task 1: Initialize a function project

1. On the taskbar, select the **Windows Terminal** icon.
2. Run the following command to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

powershell

cd F:\Allfiles\Labs\07\Starter\func

**Note**: In Windows Explorer remove the Read-only attribute from F:\Allfiles\Labs\07\Starter\func.gitignore file.

1. Run the following command to use the **Azure Functions Core Tools** to create a new local Functions project in the current directory using the **dotnet** runtime:

powershell

func init --worker-runtime dotnet --force

**Note**: You can review the documentation to [create a new project](https://labclient.labondemand.com/Instructions/f77ce42e-9785-4e77-b9af-37d2de53c47d?rc=10) using the **Azure Functions Core Tools**.

1. Run the following command to **build** the .NET Core 3.1 project:

powershell

dotnet build

Task 2: Create an HTTP-triggered function

1. Run the following command to use the **Azure Functions Core Tools** to create a new function named **FileParser** using the **HTTP trigger** template:

powershell

func new --template "HTTP trigger" --name "FileParser"

**Note**: You can review the documentation to [create a new function](https://labclient.labondemand.com/Instructions/f77ce42e-9785-4e77-b9af-37d2de53c47d?rc=10) using the **Azure Functions Core Tools**.

1. Close the currently running **Windows Terminal** application.

Task 3: Configure and read an application setting

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. On the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\07\Starter\func**, and then select **Select Folder**.
4. On the **Explorer** pane of the **Visual Studio Code** window, open the **local.settings.json** file.
5. Observe the current value of the **Values** object:

json

"Values": {

"AzureWebJobsStorage": "UseDevelopmentStorage=true",

"FUNCTIONS\_WORKER\_RUNTIME": "dotnet"

}

1. Update the value of the **Values** object by adding a new setting named **StorageConnectionString**, and then assigning it a string value of **[TEST VALUE]**:

json

"Values": {

"AzureWebJobsStorage": "UseDevelopmentStorage=true",

"FUNCTIONS\_WORKER\_RUNTIME": "dotnet",

"StorageConnectionString": "[TEST VALUE]"

}

1. The **local.settings.json** file should now include:

json

{

"IsEncrypted": false,

"Values": {

"AzureWebJobsStorage": "UseDevelopmentStorage=true",

"FUNCTIONS\_WORKER\_RUNTIME": "dotnet",

"StorageConnectionString": "[TEST VALUE]"

}

}

1. Select **Save** to save your changes to the **local.settings.json** file.
2. On the **Explorer** pane of the **Visual Studio Code** window, open the **FileParser.cs** file.
3. In the code editor, observe the example implementation:

csharp

using System;

using System.IO;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.Azure.WebJobs.Extensions.Http;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Logging;

using Newtonsoft.Json;

namespace func

{

public static class FileParser

{

[FunctionName("FileParser")]

public static async Task<IActionResult> Run(

[HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,

ILogger log)

{

1. []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;

string responseMessage = string.IsNullOrEmpty(name)

? "This HTTP triggered function executed successfully. Pass a name in the query string or in the request body for a personalized response."

: $"Hello, {name}. This HTTP triggered function executed successfully.";

return new OkObjectResult(responseMessage);

}

}

}

1. Delete all of the content within the **FileParser.cs** file.
2. Add the following lines of code to add **using directives** for the **Microsoft.AspNetCore.Mvc**, **Microsoft.Azure.WebJobs**, **Microsoft.AspNetCore.Http**, **System**, and **System.Threading.Tasks** namespaces:

csharp

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.AspNetCore.Http;

using System;

using System.Threading.Tasks;

1. Create a new **public static** class named **FileParser**:

csharp

public static class FileParser

{ }

1. Observe the **FileParser.cs** file again, which should now include:

csharp

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.AspNetCore.Http;

using System;

using System.Threading.Tasks;

public static class FileParser

{ }

1. Within the **FileParser** class, add the following code block to create a new **public static** *asynchronous* method named **Run**. This method returns a variable of type **Task\** and also takes in a variable of type **HttpRequest** named *request*:

csharp

public static async Task<IActionResult> Run(

HttpRequest request)

{ }

1. Add the following code to append an attribute to the **Run** method of type **FunctionNameAttribute** that has its **name** parameter set to a value of **FileParser**:

csharp

[FunctionName("FileParser")]

public static async Task<IActionResult> Run(

HttpRequest request)

{ }

1. Add the following code to append an attribute to the **request** parameter of type **HttpTriggerAttribute** that has its **methods** parameter array set to a single value of **GET**:

csharp

[FunctionName("FileParser")]

public static async Task<IActionResult> Run(

[HttpTrigger("GET")] HttpRequest request)

{ }

1. Observe the **FileParser.cs** file again, which should now include:

csharp

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.AspNetCore.Http;

using System;

using System.Threading.Tasks;

public static class FileParser

{

[FunctionName("FileParser")]

public static async Task<IActionResult> Run(

[HttpTrigger("GET")] HttpRequest request)

{ }

}

1. In the **Run** method, enter the following line of code to retrieve the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method and to store the result in a **string** variable named **connectionString**:

csharp

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

1. Enter the following line of code to return the value of the **connectionString** variable as the HTTP response:

csharp

return new OkObjectResult(connectionString);

1. Observe the **FileParser.cs** file again, which should now include:

csharp

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.AspNetCore.Http;

using System;

using System.Threading.Tasks;

public static class FileParser

{

[FunctionName("FileParser")]

public static async Task<IActionResult> Run(

[HttpTrigger("GET")] HttpRequest request)

{

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

return new OkObjectResult(connectionString);

}

}

1. Select **Save** to save your changes to the **FileParser.cs** file.

Task 4: Validate the local function

1. On the taskbar, select the **Windows Terminal** icon.
2. Run the following command to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

powershell

cd F:\Allfiles\Labs\07\Starter\func

1. Run the following command to run the function app project:

powershell

func start --build

**Note**: You can review the documentation to [start the function app project locally](https://labclient.labondemand.com/Instructions/f77ce42e-9785-4e77-b9af-37d2de53c47d?rc=10) using the **Azure Functions Core Tools**.

1. On the taskbar, select the **Windows Terminal** icon again to open a new instance of the **Windows Terminal** application. Run the following command to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

powershell

cd F:\Allfiles\Labs\07\Starter\func

1. When you receive the open command prompt, run the following command to start the **httprepl** tool, setting the base Uniform Resource Identifier (URI) to http://localhost:7071:

powershell

httprepl http://localhost:7071

**Note**: An error message is displayed by the **httprepl** tool. This message occurs because the tool is searching for a Swagger definition file to use to traverse the API. Because your function project doesn't produce a Swagger definition file, you'll need to traverse the API manually.

1. When you receive the tool prompt, run the following command to browse to the relative **api** directory:

powershell

cd api

1. Run the following command to browse to the relative **fileparser** directory:

powershell

cd fileparser

1. Run the following command to run the **get** command:

powershell

get

1. Observe the **[TEST VALUE]** value of the **StorageConnectionString** being returned as the result of the HTTP request:

powershell

HTTP/1.1 200 OK

Content-Type: text/plain; charset=utf-8

Date: Tue, 01 Sep 2020 23:35:39 GMT

Server: Kestrel

Transfer-Encoding: chunked

[TEST VALUE]

1. Run the following command to exit the **httprepl** tool:

powershell

exit

1. Close all currently running instances of the **Windows Terminal** application.

Task 5: Deploy the function using the Azure Functions Core Tools

1. On the taskbar, select the **Windows Terminal** icon.
2. Run the following command to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

powershell

cd F:\Allfiles\Labs\07\Starter\func

1. Run the following command to sign in to the Azure Command-Line Interface (CLI):

powershell

az login

1. In the **Microsoft Edge** browser window, enter the email address and password for your Microsoft account, and then select **Sign in**.
2. Return to the currently open **Windows Terminal** window. Wait for the sign-in process to finish.
3. Run the following command to publish the function app project:

powershell

func azure functionapp publish <function-app-name>

**Note**: As an example, if your **Function App name** is **securefuncstudent**, your command would be func azure functionapp publish securefuncstudent. You can review the documentation to [publish the local function app project](https://labclient.labondemand.com/Instructions/f77ce42e-9785-4e77-b9af-37d2de53c47d?rc=10) using the **Azure Functions Core Tools**.

1. Wait for the deployment to finalize before you move forward with the lab.
2. Close the currently running **Windows Terminal** application.

Task 6: Test the Key Vault-derived application setting

1. On the taskbar, select the **Microsoft Edge** icon, and then select the tab that contains the Azure portal https://portal.azure.com.
2. On the Azure portal's navigation pane, select the **Resource groups** link.
3. On the **Resource groups** blade, select the **ConfidentialStack** resource group.
4. On the **ConfidentialStack** blade, select the **securefunc[yourname]** function app.
5. On the **Function App** blade, select the **Functions** option in the **Functions** section.
6. On the **Functions** pane, select the existing **FileParser** function.
7. On the **Function** blade, select the **Code + Test** option in the **Developer** section.
8. In the function editor, select **Test/Run**.
9. In the pop-up dialog box that displays, in the **HTTP method** list, select **GET**.
10. Select **Run** to test the function.
11. Observe the results of the test run. The result should be your Azure Storage connection string.

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 app.

Exercise 4: Access Azure Blob Storage data

Task 1: Upload a sample storage blob

1. On the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, select the **ConfidentialStack** resource group.
3. On the **ConfidentialStack** blade, select the **securestor***[yourname]* storage account.
4. On the **Storage account** blade, select the **Containers** link in the **Data storage** section.
5. In the **Containers** section, select **+ Container**.
6. In the **New container** pop-up window, perform the following actions, and then select **Create**:

| **Setting** | **Action** |
| --- | --- |
| **Name** text box | Enter **drop** |
| **Public access level** drop-down list | Select **Blob (anonymous read access for blobs only)**. |

1. Return to the **Containers** section, and then select the newly created **drop** container.
2. On the **Container** blade, select **Upload**.
3. In the **Upload blob** window, perform the following actions, and then select **Upload**:

| **Setting** | **Action** |
| --- | --- |
| **Files** section | Select the **Folder** icon. |
| **File Explorer** window | Browse to **Allfiles (F):\Allfiles\Labs\07\Starter**, select the **records.json** file, and then select **Open**. |
| **Overwrite if files already exist** check box | Ensure that this check box is selected. |

1. **Note**: Wait for the blob to upload before you continue with this lab.
2. Return to the **Container** blade, and then select the **records.json** blob in the list of blobs.
3. On the **Blob** blade, find the blob metadata, and then copy the URL for the blob.
4. On the taskbar, activate the shortcut menu for the **Microsoft Edge** icon, and then select **New window**.
5. In the new browser window, refer to the URL that you copied for the blob.
6. The JavaScript Object Notation (JSON) contents of the blob should now display. Close the browser window with the JSON contents.
7. Return to the browser window with the Azure portal, and then close the **Blob** blade.
8. Return to the **Container** blade, and then select **Change access level**.
9. In the **Change access level** pop-up window, perform the following actions:
   1. In the **Public access level** drop-down list, select **Private (no anonymous access)**.
   2. Select **OK**.
10. On the taskbar, activate the shortcut menu for the **Microsoft Edge** icon, and then select **New window**.
11. In the new browser window, refer to the URL that you copied for the blob.
12. An error message indicating that the resource wasn't found should now display.

**Note**: If the error message doesn't display, your browser might have cached the file. Select Ctrl+F5 to refresh the page until the error message displays.

Task 2: Pull and configure the Azure SDK for .NET

1. On the taskbar, select the **Windows Terminal** icon.
2. Run the following command to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

powershell

cd F:\Allfiles\Labs\07\Starter\func

1. Run the following command to add version **12.6.0** of the **Azure.Storage.Blobs** package from NuGet:

powershell

dotnet add package Azure.Storage.Blobs --version 12.6.0

**Note**: The [Azure.Storage.Blobs](https://www.nuget.org/packages/Azure.Storage.Blobs/12.6.0) NuGet package references the subset of the Azure SDK for .NET required to write code for Azure Blob Storage.

1. Close the currently running **Windows Terminal** application.
2. On the **Start** screen, select the **Visual Studio Code** tile.
3. On the **File** menu, select **Open Folder**.
4. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\07\Starter\func**, and then select **Select Folder**.
5. On the **Explorer** pane of the **Visual Studio Code** window, open the **FileParser.cs** file.
6. Add a **using directive** for the **Azure.Storage.Blobs** namespace:

csharp

using Azure.Storage.Blobs;

1. Observe the **FileParser.cs** file, which should now include:

csharp

using Azure.Storage.Blobs;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.AspNetCore.Http;

using System;

using System.Threading.Tasks;

public static class FileParser

{

[FunctionName("FileParser")]

public static async Task<IActionResult> Run(

[HttpTrigger("GET")] HttpRequest request)

{

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

return new OkObjectResult(connectionString);

}

}

Task 3: Write Azure Blob Storage code using the Azure SDK for .NET

1. Within the **Run** method of the **FileParser** class, delete the following line of code:

csharp

return new OkObjectResult(connectionString);

1. Still within the **Run** method, add the following code block to create a new instance of the **BlobClient** class by passing in your *connectionString* variable, a "drop" string value, and a "records.json" string value to the constructor:

csharp

BlobClient blob = new BlobClient(connectionString, "drop", "records.json");

1. Still within the **Run** method, add the following code block to use the **BlobClient.DownloadAsync** method to download the contents of the referenced blob asynchronously, and then store the result in a variable named *response*:

csharp

var response = await blob.DownloadAsync();

1. Still within the **Run** method, add the following code block to return the value of the various content stored in the *content* variable by using the **FileStreamResult** class constructor:

csharp

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

1. Observe the **FileParser.cs** file again, which should now include:

csharp

using Azure.Storage.Blobs;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.AspNetCore.Http;

using System;

using System.Threading.Tasks;

public static class FileParser

{

[FunctionName("FileParser")]

public static async Task<IActionResult> Run(

[HttpTrigger("GET")] HttpRequest request)

{

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

BlobClient blob = new BlobClient(connectionString, "drop", "records.json");

var response = await blob.DownloadAsync();

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

}

}

1. Select **Save** to save your changes to the **FileParser.cs** file.

Task 4: Deploy and validate the Azure Functions app

1. On the taskbar, select the **Windows Terminal** icon.
2. Run the following command to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

powershell

cd F:\Allfiles\Labs\07\Starter\func

1. Run the following command to sign in to the Azure CLI:

powershell

az login

1. In the **Microsoft Edge** browser window, enter the email address and password for your Microsoft account, and then select **Sign in**.
2. Return to the currently open **Windows Terminal** window. Wait for the sign-in process to finish.
3. Run the following command to publish the function app project again:

powershell

func azure functionapp publish <function-app-name>

**Note**: As an example, if your **Function App name** is **securefuncstudent**, your command would be func azure functionapp publish securefuncstudent. You can review the documentation to [publish the local function app project](https://labclient.labondemand.com/Instructions/f77ce42e-9785-4e77-b9af-37d2de53c47d?rc=10) using the **Azure Functions Core Tools**.

1. Wait for the deployment to finalize before you move forward with the lab.
2. Close the currently running **Windows Terminal** application.
3. On the taskbar, select the **Microsoft Edge** icon, and then refer to the Azure portal https://portal.azure.com.
4. On the Azure portal's navigation pane, select the **Resource groups** link.
5. On the **Resource groups** blade, select the **ConfidentialStack** resource group.
6. On the **ConfidentialStack** blade, select the **securefunc[yourname]** function app.
7. On the **App Service** blade, select the **Functions** option in the **Functions** section.
8. On the **Functions** pane, select the existing **FileParser** function.
9. On the **Function** blade, select the **Code + Test** option in the **Developer** section.
10. In the function editor, select **Test/Run**.
11. In the pop-up dialog box that displays, in the **HTTP method** list, select **GET**.
12. Select **Run** to test the function.
13. Observe the results of the test run. The output will contain the content of the **$/drop/records.json** blob stored in your Azure Storage account.

Review

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

Exercise 5: Clean up your subscription

You can skip the **Clean Up** exercise directing you to remove the resources from your Subscription or Resource Group(s). The clean up is handled automatically, after ending your lab.

Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon Cloud Shell icon to open a new Bash session. If Cloud Shell defaults to a PowerShell session, select **PowerShell**, and in the drop-down menu, select **Bash**.

**Note**: If this is the first time you're starting **Cloud Shell**, when prompted to select either **Bash** or **PowerShell**, select **PowerShell**. When you're presented with the **You have no storage mounted** message, select the subscription you're using in this lab, and then select **Create storage**.

Task 2: Delete a resource group

1. On the **Cloud Shell** pane, run the following command to delete the **ConfidentialStack** resource group:

bash

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

**Note**: The command runs asynchronously (as determined by the *--no-wait* parameter), so while you'll be able to run another Azure CLI command immediately afterwards within the same Bash session, it'll take a few minutes before the resource groups are actually removed.

1. Close the **Cloud Shell** pane in the portal.

Task 3: Close the 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.

Congratulations!

You have successfully completed this **Lab** press **End** to end your lab.

33% Tasks Complete

End

Live Chat