Module 02 (CSR): Implement Azure Functions

58 Minutes Remaining

Instructions Resources Help  100%

Required Lab Setup

Sign into [AZ-204T00A-SEA-DEV](https://labclient.labondemand.com/Instructions/7d2d33f8-070a-45a8-abc1-6771be986f91?rc=10) with Username **Admin** and Password Pa55w.rd

Prepare cloud shell for later use

Mount storage in Azure

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

|  |  |
| --- | --- |
| Username | LabUser-23749630@cloudslice.onmicrosoft.com |
| Password | hJ1qNa!B2# |

1. In the toolbar at the top of the Azure portal, select the **Cloud Shell** icon.
2. In the Welcome to Azure Cloud Shell dialog, select **Bash**.
3. On the you have no storage mounted screen select **Show advanced settings**.
4. In the advanced settings screen, fill in the following fields, then click **Create Storage**:

|  |  |
| --- | --- |
| Resource Group | Use existing (**Serverless-lod23749630**) |
| Cloud Shell Region | **East US** |
| Storage account (Create new) | cloudshell23749630 |
| File share (create new) | shellstorage |

1. After the cloud shell initializes and puts you at a text prompt, exit the shell.

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.

Sign in to the lab virtual machine

1. Hello Altaf Hussain, log on to [AZ-204T00A-SEA-DEV](https://labclient.labondemand.com/Instructions/7d2d33f8-070a-45a8-abc1-6771be986f91?rc=10) lick [Ctrl+Alt+Delete](https://labclient.labondemand.com/Instructions/7d2d33f8-070a-45a8-abc1-6771be986f91?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 inside the VM.

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-23749630@cloudslice.onmicrosoft.com
* Password: hJ1qNa!B2#

Sites used:

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

All the resource groups you require in this lab have been created for you as part of the lab setup. You can safely ignore any steps that ask you to create a resource group. Please use or select a pre-existing resource group that has a similar name to the one you are asked to create. If no similarly named resource group exists, use any existing resource group.

For access to Azure, use the following credentials:

* Username: LabUser-23749630@cloudslice.onmicrosoft.com
* Password: hJ1qNa!B2#

You can see your available Resource Groups by checking the Resources tab above.

Lab 02: Implement task processing logic by using Azure Functions

Microsoft Azure user interface

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

Microsoft updates this training course when the community alerts us to needed changes. However, cloud updates occur frequently, so 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**

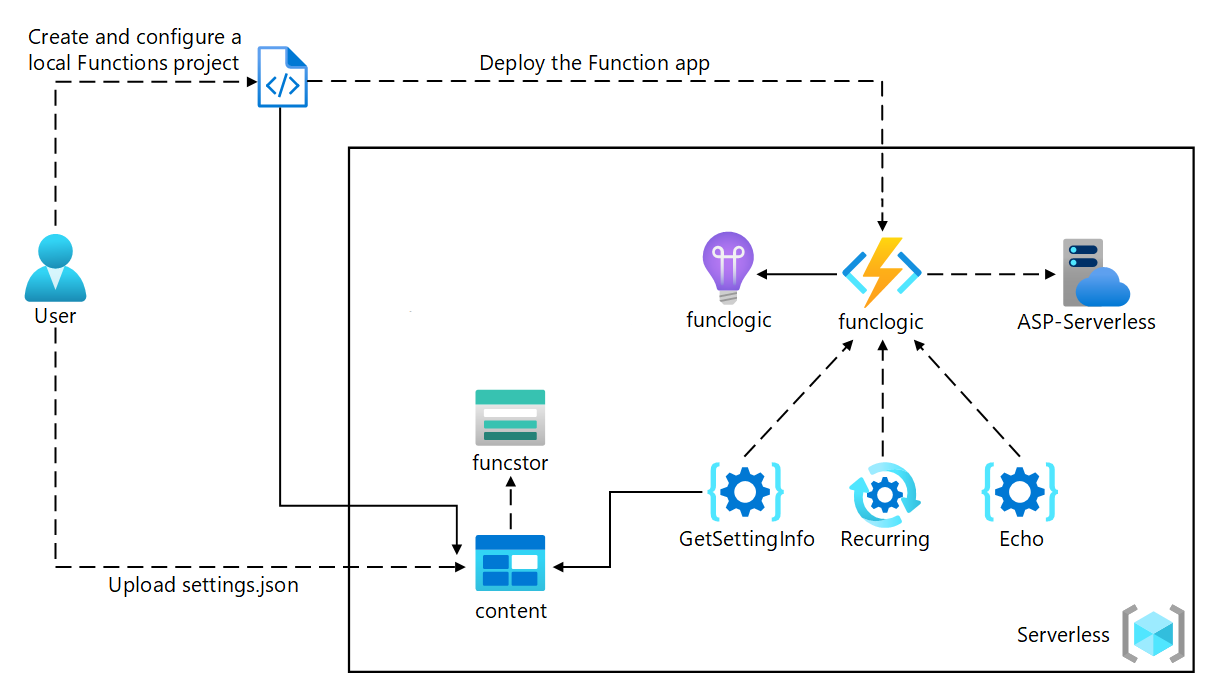
**Note**: Your instructor will provide instructions to connect to the virtual lab environment.

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

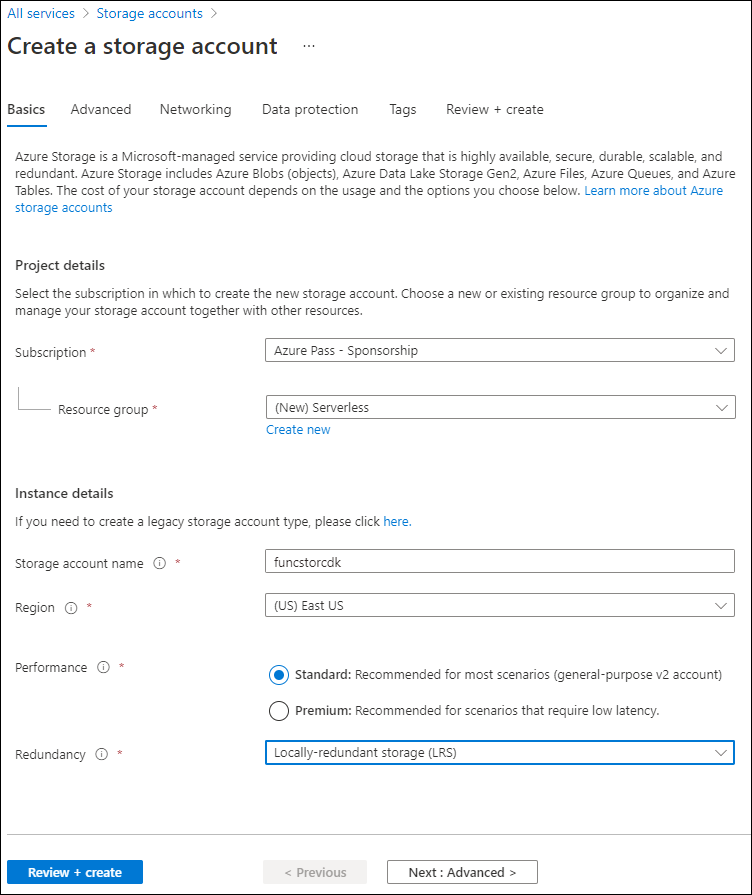
1. On the taskbar, select the **Microsoft Edge** icon.
2. In the browser window, browse to the Azure portal https://portal.azure.com, and then sign in with the account you'll be using for this lab.

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

Task 2: Create an Azure 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 then select **Review + create**:

| **Setting** | **Action** |
| --- | --- |
| **Subscription** drop-down list | Retain the default value. |
| **Resource group** section | Use existing, and select **Serverless-lod23749630**. |
| **Storage account name** text box | Enter **funcstor***[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 in 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 proceed with this lab.

1. On the **Overview** blade, select the **Go to resource** button to navigate to the blade of the newly created storage account.
2. On the **Storage account** blade, in the **Security + networking** section, select **Access keys**.
3. On the **Access keys** blade, select **Show keys**.
4. Review any one of the keys, and then copy the value of either of the **Connection string** boxes to the clipboard.

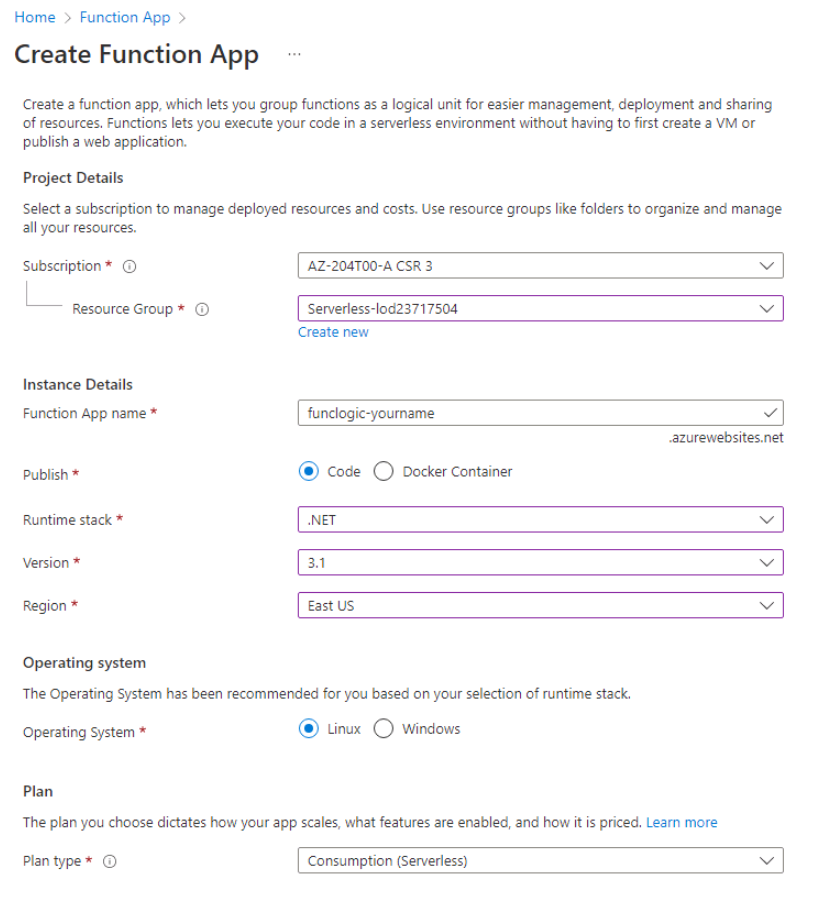
**Note**: It doesn't matter which connection string you choose. They are interchangeable.

1. Open Notepad, and then paste the copied connection string value to Notepad. You'll use this value later in this lab.

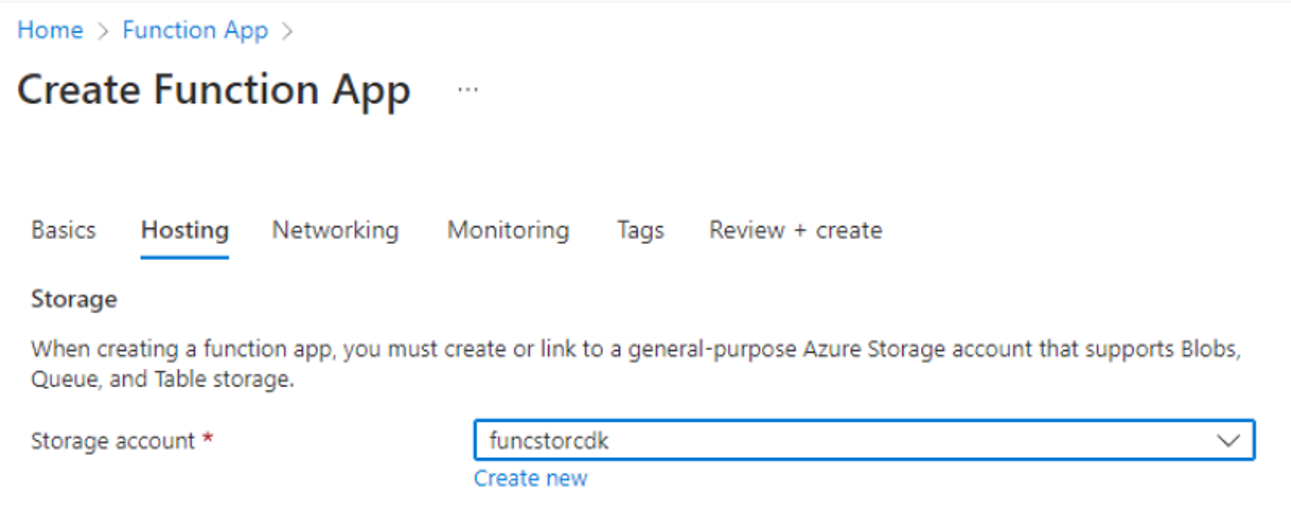
Task 3: Create a function app

1. On the Azure portal's navigation pane, select the **Create a resource** link.
2. On the **Create a resource** blade, in the **Search services and marketplace** text box, enter **Function**, and then select Enter.
3. On the **Marketplace** search results blade, select the **Function App** result.
4. On the **Function App** blade, select **Create**.
5. 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** section | Select **Serverless-lod23749630**. |
| **Function App name** text box | Enter **funclogic***[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** section | Select **Linux**. |
| **Plan type** drop-down list | Select **Consumption (Serverless-lod23749630)**. |

1. The following screenshot displays the configured settings in the **Create Function App** blade.
2. 
3. On the **Hosting** tab, perform the following actions, and then select **Review + create**:

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

1. The following screenshot displays the configured settings on the **Hosting** tab on the **Create Function App** blade.
2. 
3. On the **Review + create** tab, review the options that you selected during the previous steps.
4. 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 a local Azure Functions project

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\02\Starter\func** empty directory:

powershell

cd F:\Allfiles\Labs\02\Starter\func

**Note**: In Windows Explorer remove the **Read-only** attribute from **F:\Allfiles\Labs\02\Starter\func\.gitignore** file.

1. Run the following command to use the **Azure Functions Core Tools** to create a new local Azure 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/7d2d33f8-070a-45a8-abc1-6771be986f91?rc=10) using the **Azure Functions Core Tools**.

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

Task 2: Configure a connection string

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\02\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 **AzureWebJobsStorage** setting:

json

"AzureWebJobsStorage": "UseDevelopmentStorage=true",

1. Update the value of the **AzureWebJobsStorage** by setting it to the **connection string** of the storage account that you recorded earlier in this lab.
2. Save the **local.settings.json** file.

Task 3: Build and validate a 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\02\Starter\func** directory:

powershell

cd F:\Allfiles\Labs\02\Starter\func

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

powershell

dotnet build

Review

In this exercise, you created a local project that you'll use for Azure Functions development.

Exercise 3: Create a function that's triggered by an HTTP request

Task 1: Create an HTTP-triggered 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\02\Starter\func** directory:

powershell

cd F:\Allfiles\Labs\02\Starter\func

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

powershell

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

**Note**: You can review the documentation to [create a new function](https://labclient.labondemand.com/Instructions/7d2d33f8-070a-45a8-abc1-6771be986f91?rc=10) using the **Azure Functions Core Tools**.

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

Task 2: Write HTTP-triggered function code

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\02\Starter\func**, and then select **Select Folder**.
4. On the **Explorer** pane of the **Visual Studio Code** window, open the **Echo.cs** file.
5. 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 Echo

{

[FunctionName("Echo")]

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 the content within the **Echo.cs** file.
2. Add the following lines of code to add **using directives** for the **Microsoft.AspNetCore.Mvc**, **Microsoft.Azure.WebJobs**, **Microsoft.AspNetCore.Http**, and **Microsoft.Extensions.Logging** namespaces:

csharp

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Logging;

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

csharp

public static class Echo

{ }

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

csharp

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Logging;

public static class Echo

{ }

1. Within the **Echo** class, add the following code block to create a new **public static** method named **Run** that returns a variable of type **IActionResult** and that also takes in variables of type **HttpRequest** and **ILogger** as parameters named *request* and *logger*:

csharp

public static IActionResult Run(

HttpRequest request,

ILogger logger)

{ }

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 **Echo**:

csharp

[FunctionName("Echo")]

public static IActionResult Run(

HttpRequest request,

ILogger logger)

{ }

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 **POST**:

csharp

[FunctionName("Echo")]

public static IActionResult Run(

[HttpTrigger("POST")] HttpRequest request,

ILogger logger)

{ }

1. Observe the **Echo.cs** file again, which should now include the following code:

csharp

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Logging;

public static class Echo

{

[FunctionName("Echo")]

public static IActionResult Run(

[HttpTrigger("POST")] HttpRequest request,

ILogger logger)

{ }

}

1. In the **Run** method, enter the following line of code to log a fixed message:

csharp

logger.LogInformation("Received a request");

1. Enter the following line of code to echo the body of the HTTP request as the HTTP response:

csharp

return new OkObjectResult(request.Body);

1. Observe the **Echo.cs** file again, which should now include the following code:

csharp

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Logging;

public static class Echo

{

[FunctionName("Echo")]

public static IActionResult Run(

[HttpTrigger("POST")] HttpRequest request,

ILogger logger)

{

logger.LogInformation("Received a request");

return new OkObjectResult(request.Body);

}

}

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

Task 3: Test the HTTP-triggered function by using httprepl

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\02\Starter\func** directory:

powershell

cd F:\Allfiles\Labs\02\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://docs.microsoft.com/azure/azure-functions/functions-develop-local) using the **Azure Functions Core Tools**.

1. On the taskbar, select the **Windows Terminal** icon again to open a new instance of the application. Run the following command to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

powershell

cd F:\Allfiles\Labs\02\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 **echo** directory:

powershell

cd echo

1. Run the following command to run the **post** command, sending in an HTTP request body set to a numeric value of **3** by using the **--content** option:

powershell

post --content 3

1. Run the following command to run the **post** command, sending in an HTTP request body set to a numeric value of **5** by using the **--content** option:

powershell

post --content 5

1. Run the following command to run the **post** command, sending in an HTTP request body set to a string value of **Hello** by using the **--content** option:

powershell

post --content "Hello"

1. Run the following command to run the **post** command, sending in an HTTP request body set to a JavaScript Object Notation (JSON) value of **{"msg": "Successful"}** by using the **--content** option:

powershell

post --content "{"msg": "Successful"}"

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

powershell

exit

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

Review

In this exercise, you created a basic function that echoes the content sent through an HTTP POST request.

Exercise 4: Create a function that triggers on a schedule

Task 1: Create a schedule-triggered 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\02\Starter\func** directory:

powershell

cd F:\Allfiles\Labs\02\Starter\func

1. When you receive the open command prompt, run the following command to use the **Azure Functions Core Tools** to create a new function named **Recurring**, using the **Timer trigger** template:

powershell

func new --template "Timer trigger" --name "Recurring"

**Note**: You can review the documentation to [create a new function](https://labclient.labondemand.com/Instructions/7d2d33f8-070a-45a8-abc1-6771be986f91?rc=10) using the **Azure Functions Core Tools**.

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

Task 2: Observe function code

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\02\Starter\func**, and then select **Select Folder**.
4. On the **Explorer** pane of the **Visual Studio Code** window, open the **Recurring.cs** file.
5. In the code editor, observe the implementation:

csharp

using System;

using Microsoft.Azure.WebJobs;

using Microsoft.Azure.WebJobs.Host;

using Microsoft.Extensions.Logging;

namespace func

{

public static class Recurring

{

[FunctionName("Recurring")]

public static void Run([TimerTrigger("0 \*/5 \* \* \* \*")]TimerInfo myTimer, ILogger log)

{

1. []LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");

}

}

}

Task 3: Observe function runs

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\02\Starter\func** directory:

powershell

cd F:\Allfiles\Labs\02\Starter\func

1. When you receive the open command prompt, 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/7d2d33f8-070a-45a8-abc1-6771be986f91?rc=10) using the **Azure Functions Core Tools**.

1. Observe the function run that occurs about every five minutes. Each function run should render a simple message to the log.
2. Close the currently running **Windows Terminal** application.

Task 4: Update the function integration configuration

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\02\Starter\func**, and then select **Select Folder**.
4. On the **Explorer** pane of the **Visual Studio Code** window, open the **Recurring.cs** file.
5. In the code editor, observe the existing **Run** method signature:

csharp

[FunctionName("Recurring")]

public static void Run([TimerTrigger("0 \*/5 \* \* \* \*")]TimerInfo myTimer, ILogger log)

1. Update the **Run** method signature code block to change the schedule to run once every **30 seconds**:

csharp

[FunctionName("Recurring")]

public static void Run([TimerTrigger("\*/30 \* \* \* \* \*")]TimerInfo myTimer, ILogger log)

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

Task 5: Observe function runs

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\02\Starter\func** directory:

powershell

cd F:\Allfiles\Labs\02\Starter\func

1. When you receive the open command prompt, 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/7d2d33f8-070a-45a8-abc1-6771be986f91?rc=10) using the **Azure Functions Core Tools**.

1. Observe the function run that occurs about every 30 seconds. Each function run should render a simple message to the log.
2. Close the currently running **Windows Terminal** application.
3. Close the Visual Studio Code window.

Review

In this exercise, you created a function that runs automatically based on a fixed schedule.

Exercise 5: Create a function that integrates with other services

Task 1: Upload sample content to Azure Blob Storage

1. On the Azure portal's **navigation** pane, select the **Resource groups** link.
2. On the **Resource groups** blade, select the **Serverless-lod23749630** resource group that you created previously in this lab.
3. On the **Serverless-lod23749630** blade, select the **funcstor***[yourname]* storage account that you created previously in this lab.
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 **content**. |
| **Public access level** drop-down list | Select **Private (no anonymous access)**. |

1. Return to the **Containers** section, and then select the recently created **content** 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\02\Starter**, select the **settings.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.

Task 2: Create an HTTP-triggered 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\02\Starter\func** directory:

powershell

cd F:\Allfiles\Labs\02\Starter\func

1. When you receive the open command prompt, run the following command to use the **Azure Functions Core Tools** to create a new function named **GetSettingInfo**, using the **HTTP trigger** template:

powershell

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

**Note**: You can review the documentation to [create a new function](https://labclient.labondemand.com/Instructions/7d2d33f8-070a-45a8-abc1-6771be986f91?rc=10) using the **Azure Functions Core Tools**.

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

Task 3: Write HTTP-triggered and blob-inputted function code

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\02\Starter\func**, and then select **Select Folder**.
4. On the **Explorer** pane of the **Visual Studio Code** window, open the **GetSettingInfo.cs** file.
5. 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 GetSettingInfo

{

[FunctionName("GetSettingInfo")]

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 the content within the **GetSettingInfo.cs** file.
2. Add the following lines of code to add **using directives** for the **Microsoft.AspNetCore.Http**, **Microsoft.AspNetCore.Mvc**, and **Microsoft.Azure.WebJobs** namespaces:

csharp

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

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

csharp

public static class GetSettingInfo

{ }

1. Observe the **GetSettingInfo.cs** file again, which should now include the following code:

csharp

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

public static class GetSettingInfo

{ }

1. Within the **GetSettingInfo** class, add the following code block to create a new **public static** expression-bodied method named **Run** that returns a variable of type **IActionResult** and that also takes in variables of type **HttpRequest** and **string** as parameters named *request* and *json*:

csharp

public static IActionResult Run(

HttpRequest request,

string json)

=> null;

**Note**: You are only temporarily setting the return value to **null**.

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 **GetSettingInfo**:

csharp

[FunctionName("GetSettingInfo")]

public static IActionResult Run(

HttpRequest request,

string json)

=> null;

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("GetSettingInfo")]

public static IActionResult Run(

[HttpTrigger("GET")] HttpRequest request,

string json)

=> null;

1. Add the following code to append an attribute to the **json** parameter of type **BlobAttribute** that has its **blobPath** parameter set to a value of **content/settings.json**:

csharp

[FunctionName("GetSettingInfo")]

public static IActionResult Run(

[HttpTrigger("GET")] HttpRequest request,

[Blob("content/settings.json")] string json)

=> null;

1. Add the following code to update the **Run** expression-bodied method to return a new instance of the **OkObjectResult** class passing in the value of the **json** method parameter as the sole constructor parameter:

csharp

[FunctionName("GetSettingInfo")]

public static IActionResult Run(

[HttpTrigger("GET")] HttpRequest request,

[Blob("content/settings.json")] string json)

=> new OkObjectResult(json);

1. Observe the **GetSettingInfo.cs** file again, which should now include the following code:

csharp

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

public static class GetSettingInfo

{

[FunctionName("GetSettingInfo")]

public static IActionResult Run(

[HttpTrigger("GET")] HttpRequest request,

[Blob("content/settings.json")] string json)

=> new OkObjectResult(json);

}

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

Task 4: Register Azure Storage Blob extensions

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\02\Starter\func** directory:

powershell

cd F:\Allfiles\Labs\02\Starter\func

1. When you receive the open command prompt, run the following command to register the [Microsoft.Azure.WebJobs.Extensions.Storage](https://www.nuget.org/packages/Microsoft.Azure.WebJobs.Extensions.Storage/4.0.4" \t "_blank) extension:

powershell

func extensions install --package Microsoft.Azure.WebJobs.Extensions.Storage --version 4.0.4

1. Run the following command to build the .NET project and to validate the extensions were installed correctly:

powershell

dotnet build

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

Task 5: Test the function by using httprepl

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\02\Starter\func** directory:

powershell

cd F:\Allfiles\Labs\02\Starter\func

1. When you receive the open command prompt, 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/7d2d33f8-070a-45a8-abc1-6771be986f91?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.
2. 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** endpoint:

powershell

cd api

1. Run the following command to browse to the relative **getsettinginfo** endpoint:

powershell

cd getsettinginfo

1. Run the following command to run the **get** command for the current endpoint:

powershell

get

1. Observe the JSON content of the response from the function app, which should now include:

json

{

"version": "0.2.4",

"root": "/usr/libexec/mews\_principal/",

"device": {

"id": "21e46d2b2b926cba031a23c6919"

},

"notifications": {

"email": "joseph.price@contoso.com",

"phone": "(425) 555-0162 x4151"

}

}

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

powershell

exit

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

Review

In this exercise, you created a function that returns the content of a JSON file in Storage.

Exercise 6: Deploy a local function project to an Azure Functions app

Task 1: Deploy 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\02\Starter\func** directory:

powershell

cd F:\Allfiles\Labs\02\Starter\func

1. When you receive the open command prompt, run the following command to login 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**: For example, if your **Function App name** is **funclogicstudent**, your command would be func azure functionapp publish funclogicstudent. You can review the documentation to [publish the local function app project](https://labclient.labondemand.com/Instructions/7d2d33f8-070a-45a8-abc1-6771be986f91?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 2: Validate deployment

1. On the taskbar, select the **Microsoft Edge** icon, and 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 **Serverless-lod23749630** resource group that you created previously in this lab.
4. On the **Serverless-lod23749630** blade, select the **funclogic***[yourname]* function app that you created previously in this lab.
5. On the **Function App** blade, select the **Functions** option in the **Functions** section.
6. On the **Functions** pane, select the existing **GetSettingInfo** function.
7. In the **Function** blade, select the **Code + Test** option in the **Developer** section.
8. In the function editor, select **Test/Run**.
9. In the popup dialog that appears, perform the following actions:
   * In the **HTTP method** list, select **GET**.
10. Select **Run** to test the function.
11. Observe the results of the test run. The JSON content should now include the following code:

json

{

"version": "0.2.4",

"root": "/usr/libexec/mews\_principal/",

"device": {

"id": "21e46d2b2b926cba031a23c6919"

},

"notifications": {

"email": "joseph.price@contoso.com",

"phone": "(425) 555-0162 x4151"

}

}

Review

In this exercise, you deployed a local function project to Azure Functions and validated that the functions work in Azure.

Exercise 7: 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 and list resource groups

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 **Serverless-lod23749630** resource group:

powershell

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

**Note**: The command executes 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 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 group that was used in this lab.

Congratulations!

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

18% Tasks Complete

End

Live Chat