# Lab: Implement task processing logic by using Azure Functions

# Student lab answer key

## Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure 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 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 virtual machine

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

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

**Note**: Instructions to connect to the virtual lab environment will be provided by your instructor.

#### Review the installed applications

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

* Microsoft Edge
* File Explorer
* Windows Terminal

### 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, go to the Azure portal (<https://portal.azure.com>).
3. Enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

* **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 an Azure Storage account

1. In the Azure portal's navigation pane, select **All services**.
2. On the **All services** blade, select **Storage Accounts**.
3. On the **Storage accounts** blade, get your list of storage account instances.
4. On the **Storage accounts** blade, select **Add**.
5. On the **Create storage account** blade, observe the tabs on the blade, such as **Basics**, **Tags**, and **Review + Create**.

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

1. Select the **Basics** tab, and then in the tab area, 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 **Serverless**, and then select **OK**.
   3. In the **Storage account name** text box, enter **funcstor*[yourname]***.
   4. In the **Location** list, select the **(US) East US** region.
   5. In the **Performance** section, select **Standard**.
   6. In the **Account kind** list, select **StorageV2 (general purpose v2)**.
   7. In the **Replication** list, select **Locally-redundant storage (LRS)**.
   8. In the **Access tier (default)** section, ensure that **Hot** is selected.
   9. Select **Review + Create**.
2. On the **Review + Create** tab, review the options that you specified in the previous steps.
3. Select **Create** to create the storage account by using your specified configuration.

* **Note**: On the **Deployment** blade, wait for the creation task to complete before moving forward with this lab.

#### Task 3: Create an Azure Functions app

1. In the Azure portal's navigation pane, select the **Create a resource** link.
2. From the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **Function**, and then select Enter.
4. On the **Everything** search results blade, select the **Function App** result.
5. On the **Function App** blade, select **Create**.
6. Find the tabs on the **Function App** blade, such as **Basics**.

* **Note**: Each tab represents a step in the workflow to create a new function app. You can select **Review + Create** at any time to skip the remaining tabs.

1. On 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 **Use existing**, and then select **Serverless** in the list.
   3. In the **Function app name** text box, enter **funclogic*[yourname]***.
   4. In the **Publish** section, select **Code**.
   5. In the **Runtime stack** drop-down list, select **.NET Core**.
   6. In the **Region** drop-down list, select the **East US** region.
   7. Select **Next: Hosting**.
2. On the **Hosting** tab, perform the following actions:
   1. In the **Storage account** drop-down list, select the **funcstor*[yourname]*** storage account that you created earlier in this lab.
   2. In the **Operating System** section, select **Windows**.
   3. In the **Plan type** drop-down list, select the **Consumption** option.
   4. Select **Next: Monitoring**.
3. On the **Monitoring** tab, perform the following actions:
   1. In the **Enable Application Insights** section, select **No**.
   2. Select **Review + Create**.
4. On the **Review + Create** tab, review the options that you selected during the previous steps.
5. 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 for this lab.

### Exercise 2: Create a function that's triggered by an HTTP request

#### Task 1: Create an HTTP-triggered function

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **Serverless** resource group that you created earlier in this lab.
3. On the **Serverless** blade, select the **funclogic*[yourname]*** function app that you created earlier in this lab.
4. On the **Function Apps** blade, select the plus sign (**+**) next to the **Functions** drop-down list.
5. In the **New Azure Function** quickstart, perform the following actions:
   1. Under the **Choose a Development Environment** header, select **In-Portal**, and then select **Continue**.
   2. Under the **Create a Function** header, select **More templates**, and then select **Finish and view templates**.
   3. In the list of templates, select **HTTP trigger**.
   4. In the **New Function** pop-up window, find the **Name** text box, and then enter **Echo**.
   5. In the **New Function** pop-up window, find the **Authorization level** list, and then select **Anonymous**.
   6. In the **New Function** pop-up window, select **Create**.

#### Task 2: Write function code

1. In the function editor, find the example **run.csx** function script:

* #r "Newtonsoft.Json"  
    
  using System.Net;  
  using Microsoft.AspNetCore.Mvc;  
  using Microsoft.Extensions.Primitives;  
  using Newtonsoft.Json;  
    
  public static async Task<IActionResult> 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. Delete all the example code.
2. Add the following line of code to import the **Microsoft.AspNetCore.Mvc** namespace:

* using Microsoft.AspNetCore.Mvc;

1. Add the following line of code to import the **System.Net** namespace:

* using System.Net;

1. 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 *req* and *log*:

* public static IActionResult Run(HttpRequest req, ILogger log)  
  {  
  }

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

* log.LogInformation("Received a request");

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

* return new OkObjectResult(req.Body);

1. Observe the **run.csx** file, which should now include:

* using System.Net;  
  using Microsoft.AspNetCore.Mvc;  
    
  public static IActionResult Run(HttpRequest req, ILogger log)  
  {  
   log.LogInformation("Received a request");  
    
   return new OkObjectResult(req.Body);  
  }

1. Select **Save** to save the script.

#### Task 3: Test function run in the portal

1. Select **Logs**.
2. Observe the compilation results. The results should include a "Compilation succeeded" message.
3. Select **Run** to test the function.
4. Observe the results of the test run. The results should echo the original request body exactly.

#### Task 4: Get a base function URL

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **Serverless** resource group that you created earlier in this lab.
3. On the **Serverless** blade, select the **funclogic*[yourname]*** function app that you created earlier in this lab.
4. On the **Function Apps** blade, copy the value of the **URL** text box. You'll use this value later in the lab.

#### Task 5: Test function run by using httprepl

1. On the taskbar, select the **Windows Terminal** icon.
2. At the open command prompt, enter the following command, and then select Enter to start the **httprepl** tool setting the base Uniform Resource Identifier (URI) to the value of the URL that you copied earlier in this lab.

* httprepl <function-app-url>
* **Note**: For example, if your URL is **https://funclogicstudent.azurewebsites.net**, your command would be **httprepl https://funclogicstudent.azurewebsites.net**.

1. At the tool prompt, enter the following command, and then select Enter to browse to the relative **api** directory:

* cd api

1. Enter the following command, and then select Enter to browse to the relative **echo** directory:

* cd echo

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

* post --content 3

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

* post --content 5

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

* post --content "Hello"

1. Enter the following command, and then select Enter 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:

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

1. Enter the following command, and then select Enter to exit the **httprepl** application:

* exit

1. Close the currently running Windows Terminal application.
2. Return to the browser window with the Azure portal.

#### Review

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

### Exercise 3: Create a function that triggers on a schedule

#### Task 1: Create a schedule-triggered function

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **Serverless** resource group that you created earlier in this lab.
3. On the **Serverless** blade, select the **funclogic*[yourname]*** function app that you created earlier in this lab.
4. On the **Function Apps** blade, select the plus sign (**+**) next to the **Functions** drop-down list.
5. In the **New Azure Function** quickstart, perform the following actions:
   1. In the list of templates, select **Timer trigger**.
   2. In the **New Function** pop-up window, find the **Name** text box, and then enter **Recurring**.
   3. In the **New Function** pop-up window, find the **Schedule** text box, and then enter **0 \* \* \* \* \***.
   4. In the **New Function** pop-up window, select **Create**.

#### Task 2: Observe function runs

1. In the function editor, select **Save** to persist the default function implementation.
2. Select **Logs**.
3. Observe the function run that occurs about every minute. Each function run should render a simple message to the log.

#### Task 3: Update the function integration configuration

1. Back on the **Function Apps** blade, perform the following actions:
   1. Expand the node for the **funclogic*[yourname]*** function app that you created earlier in this lab.
   2. Expand the **Functions** node.
   3. Expand the **Recurring** node for that specific function.
   4. Select the **Integrate** node.
2. In the **Integrate** section, perform the following actions:
   1. Select the **Timer (myTimer)** option in the **Trigger** section.
   2. In the **Schedule** text box, enter the value **\*/30 \* \* \* \* \***.
   3. Select **Save**.

#### Task 4: Observe function runs

1. Back on the **Function Apps** blade, perform the following actions:
   1. Expand the node for the **funclogic*[yourname]*** function app that you created earlier in this lab.
   2. Expand the **Functions** node.
   3. Select the **Recurring** node for that specific function.
2. In the function editor, select **Logs**.
3. Observe the function run that now occurs about every 30 seconds. Each function run should render a simple message to the log.

#### Review

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

### Exercise 4: Create a function that integrates with other services

#### Task 1: Create an HTTP-triggered function

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **Serverless** resource group that you created earlier in this lab.
3. On the **Serverless** blade, select the **funclogic*[yourname]*** function app that you created earlier in this lab.
4. On the **Function Apps** blade, select the plus sign (**+**) next to the **Functions** drop-down list.
5. In the **New Azure Function** quickstart, perform the following actions:
   1. In the list of templates, select **HTTP trigger**.
   2. In the **New Function** pop-up window, find the **Name** text box, and then enter **GetSettingInfo**.
   3. In the **New Function** pop-up window, find the **Authorization level** list, and then select **Anonymous**.
   4. In the **New Function** pop-up window, select **Create**.

#### Task 2: Upload sample content

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **Serverless** resource group that you created earlier in this lab.
3. On the **Serverless** blade, select the **funcstor*[yourname]*** storage account that you created earlier in this lab.
4. On the **Storage account** blade, select the **Containers** link in the **Blob service** section.
5. In the **Containers** section, select **+ Container**.
6. In the **New container** pop-up window, perform the following actions:
   1. In the **Name** text box, enter **content**.
   2. In the **Public access level** drop-down list, select **Private (no anonymous access)**.
   3. Select **OK**.
7. Back in the **Containers** section, select the recently created **content** container.
8. On the **Container** blade, select **Upload**.
9. In the **Upload blob** window, perform the following actions:
   1. In the **Files** section, select the **Folder** icon.
   2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\02\Starter**, select the **settings.json** file, and then select **Open**.
   3. Ensure that the **Overwrite if files already exist** check box is selected, and then select **Upload**.

* **Note**: Wait for the blob to upload before you continue with this lab.

#### Task 3: Configure an HTTP-triggered function

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **Serverless** resource group that you created earlier in this lab.
3. On the **Serverless** blade, select the **funclogic*[yourname]*** function app that you created earlier in this lab.
4. On the **Function Apps** blade, perform the following actions:
   1. Expand the node for the **funclogic*[yourname]*** function app that you created earlier in this lab.
   2. Expand the **Functions** node.
   3. Expand the **GetSettingInfo** node for that specific function.
   4. Select the **Integrate** node.
5. In the **Integrate** section, perform the following actions to create a new input of type **Azure Blob Storage**:
   1. Select **New Input**.
   2. Select **Azure Blob Storage**.
   3. Select **Select**.
6. In the **Azure Blob Storage input** section, perform the following actions:
   1. In the **Blob parameter name** text box, enter the value **json**.
   2. In the **Path** text box, enter the value **content/settings.json**.
   3. In the **Storage account connection** list, select **AzureWebJobsStorage**.
   4. Select **Save**.
7. Back in the **Integrate** section, select the existing **HTTP trigger**.
8. In the **HTTP trigger** section, perform the following actions:
   1. In the **Allowed HTTP methods** list, select **Selected methods**.
   2. In the **Request parameter name** text box, enter the value **request**.
   3. In the **Selected HTTP methods** check box group, ensure that only the **GET** option is selected.
   4. Select **Save**.

#### Task 4: Write function code

1. On the **Function Apps** blade, perform the following actions:
   1. Expand the node for the **funclogic*[yourname]*** function app that you created earlier in this lab.
   2. Expand the **Functions** node.
   3. Select the **GetSettingInfo** node for that specific function.
2. In the function editor, find the example **run.csx** function script:

* #r "Newtonsoft.Json"  
    
  using System.Net;  
  using Microsoft.AspNetCore.Mvc;  
  using Microsoft.Extensions.Primitives;  
  using Newtonsoft.Json;  
    
  public static async Task<IActionResult> 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. **Delete** all the example code.
2. Add the following line of code to import the **Microsoft.AspNetCore.Mvc** namespace:

* using Microsoft.AspNetCore.Mvc;

1. Add the following line of code to import the **System.Net** namespace:

* using System.Net;

1. 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 **string** as parameters named *request* and *json*:

* public static IActionResult Run(HttpRequest request, string json)  
  {  
  }

1. In the **Run** method, enter the following line of code to return the content of the *json* parameter as the HTTP response of the function:

* return new OkObjectResult(json);

1. Observe the **run.csx** file, which should now include:

* using Microsoft.AspNetCore.Mvc;  
  using System.Net;  
    
  public static IActionResult Run(HttpRequest request, string json)  
  {  
   return new OkObjectResult(json);  
  }

1. Select **Save** to save the script.

#### Task 5: Test function run

1. On the taskbar, select the **Windows Terminal** icon.
2. At the open command prompt, enter the following command, and then select Enter to start the **httprepl** tool setting the base URI to the value of the URL that you copied earlier in this lab.

* httprepl <function-app-url>
* **Note**: For example, if your URL is **https://funclogicstudent.azurewebsites.net**, your command would be **httprepl https://funclogicstudent.azurewebsites.net**.

1. At the tool prompt, enter the following command, and then select Enter to browse to the relative **api** endpoint:

* cd api

1. Enter the following command, and then select Enter to browse to the relative **getsettinginfo** endpoint:

* cd getsettinginfo

1. Enter the following command, and then select Enter to run the **get** command for the current endpoint:

* get

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

* {  
   "version": "0.2.4",  
   "root": "/usr/libexec/mews\_principal/",  
   "device": {  
   "id": "21e46d2b2b926cba031a23c6919"  
   },  
   "notifications": {  
   "email": "Anais85@outlook.com",  
   "phone": "751.757.2014 x4151"  
   }  
  }

1. Enter the following command, and then select Enter to exit the **httprepl** application:

* exit

1. Close the currently running Windows Terminal application.
2. Return to the browser window with the Azure portal.

#### Review

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

### Exercise 5: Clean up your subscription

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

1. In the Azure portal's navigation pane, select the **Cloud Shell** icon to open a new shell instance.

* **Note**: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (\_).

1. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:
   1. A dialog box prompts you to create a new storage account to begin using the shell. Accept the default settings, and then select **Create storage**.

* **Note**: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If you don't notice Cloud Shell configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

1. At the **Cloud Shell** command prompt in the portal, enter the following command, and then select Enter to list all resource groups in the subscription:

* az group list

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

* az group delete --help

#### Task 2: Delete a resource group

1. At the command prompt, enter the following command, and then select Enter to delete the **Serverless** resource group:

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

1. Close the Cloud Shell pane in the portal.

#### Task 3: Close the active application

1. the currently running Microsoft Edge application.

#### Review

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

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

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