AZ-204

Developing solutions for Microsoft Azure

Lab 02

Implement task processing logic by using Azure Functions

Table of Contents

1 Pre-requisites

1.1 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.

1.2 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
- Azure CLI
- Windows PowerShell
- Visual Studio Code

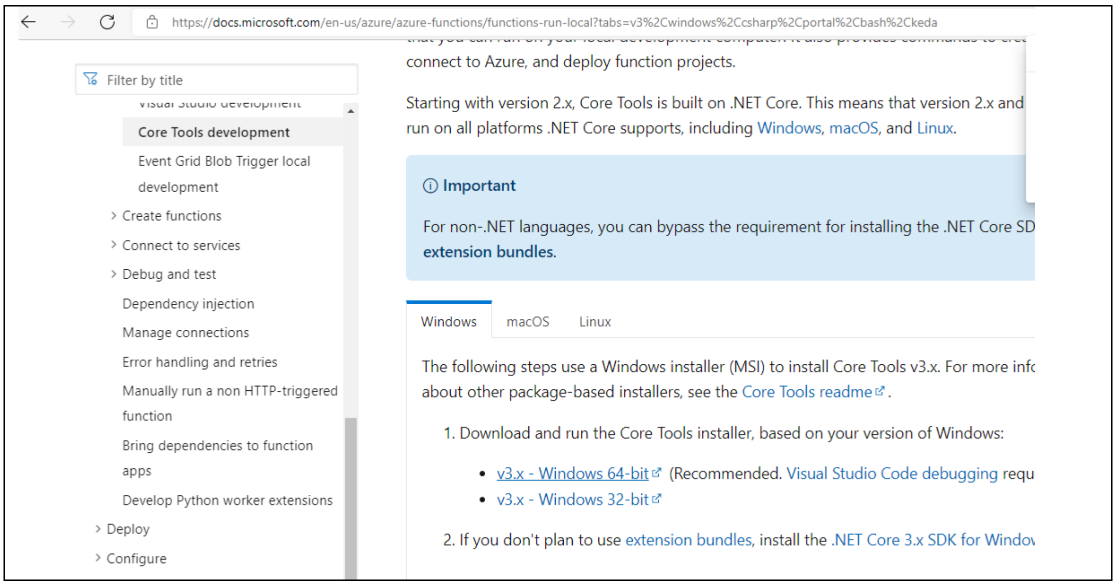
1.3 Installing Azure Functions Core tools

1.3.1 Download the Tools

1. Install NodeJS
2. Instal the Azure Function Core Tools globally using NPM :

   ```
   npm install -g azure-functions-core-tools@3 --unsafe-perm true
   ```

3. Download the Tools MSI Package



https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local?tabs=v3%2Cwindows%2Ccsharp%2Cportal%2Cbash%2Ckeda

Filter by title

Core Tools development
Event Grid Blob Trigger local development
› Create functions
› Connect to services
› Debug and test
Dependency injection
Manage connections
Error handling and retries
Manually run a non HTTP-triggered function
Bring dependencies to function apps
Develop Python worker extensions
› Deploy
› Configure

connect to Azure, and deploy function projects.

Starting with version 2.x, Core Tools is built on .NET Core. This means that version 2.x and run on all platforms .NET Core supports, including Windows, macOS, and Linux.

ⓘ Important

For non-.NET languages, you can bypass the requirement for installing the .NET Core SD **extension bundles.**

Windows | macOS | Linux

The following steps use a Windows installer (MSI) to install Core Tools v3.x. For more infc about other package-based installers, see the Core Tools readme.

1. Download and run the Core Tools installer, based on your version of Windows:
   - v3.x - Windows 64-bit (Recommended. Visual Studio Code debugging requ
   - v3.x - Windows 32-bit
2. If you don't plan to use extension bundles, install the .NET Core 3.x SDK for Windov

4. Install the MSI Package

2 Exercise 1: Create Azure resources

2.1 Task 1: Open the Azure portal

5. On the taskbar, select the **Microsoft Edge** icon.
6. In the open browser window, go to the Azure portal (https://portal.azure.com).
7. Enter the email address for your Microsoft account, and then select **Next**.
8. Enter the password for your Microsoft account, and then select **Sign in**.

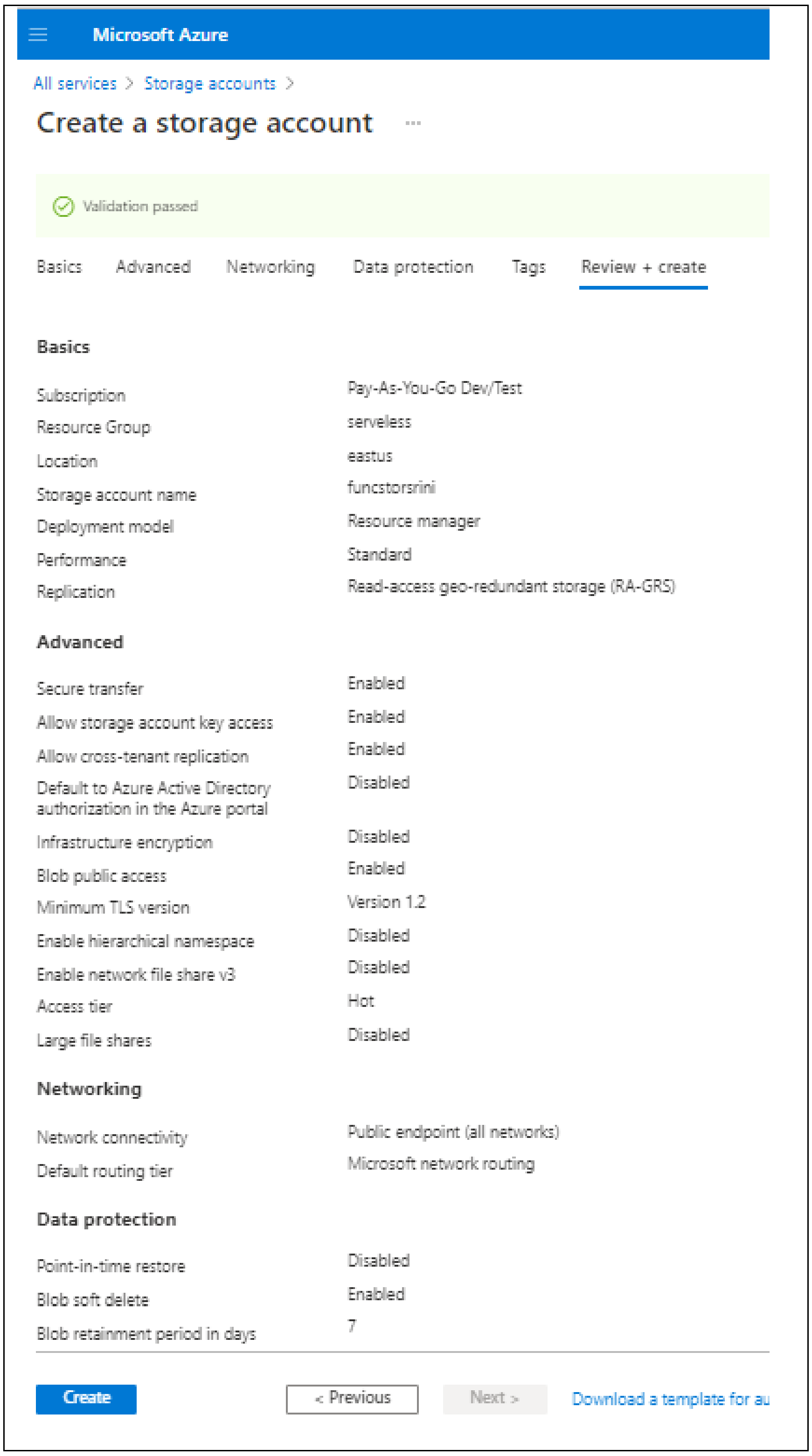
> **Note**: If this is your first time signing in to the Azure portal, you'll be offered a tour of the portal. If you prefer to skip the tour, select **Get Started** to begin using the portal.

2.2 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 **+ Create**.
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.

6. 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 **Region** list, select the **(US) East US** region.
   5. In the **Performance** section, select **Standard**.
   6. In the **Redundancy** list, select **Locally-redundant storage (LRS)**.
   7. Select **Review + Create**.
7. On the **Review + Create** tab, review the options that you specified in the previous steps.


Microsoft Azure
All services > Storage accounts >
Create a storage account
Validation passed
Basics Advanced Networking Data protection Tags Review + create
Create
< Previous
Next >
Download a template for au

Basics

Subscription	Pay-As-You-Go Dev/Test
Resource Group	serveless
Location	eastus
Storage account name	funcstorsrini
Deployment model	Resource manager
Performance	Standard
Replication	Read-access geo-redundant storage (RA-GRS)

Advanced

Secure transfer	Enabled
Allow storage account key access	Enabled
Allow cross-tenant replication	Enabled
Default to Azure Active Directory authorization in the Azure portal	Disabled
Infrastructure encryption	Disabled
Blob public access	Enabled
Minimum TLS version	Version 1.2
Enable hierarchical namespace	Disabled
Enable network file share v3	Disabled
Access tier	Hot
Large file shares	Disabled

Networking

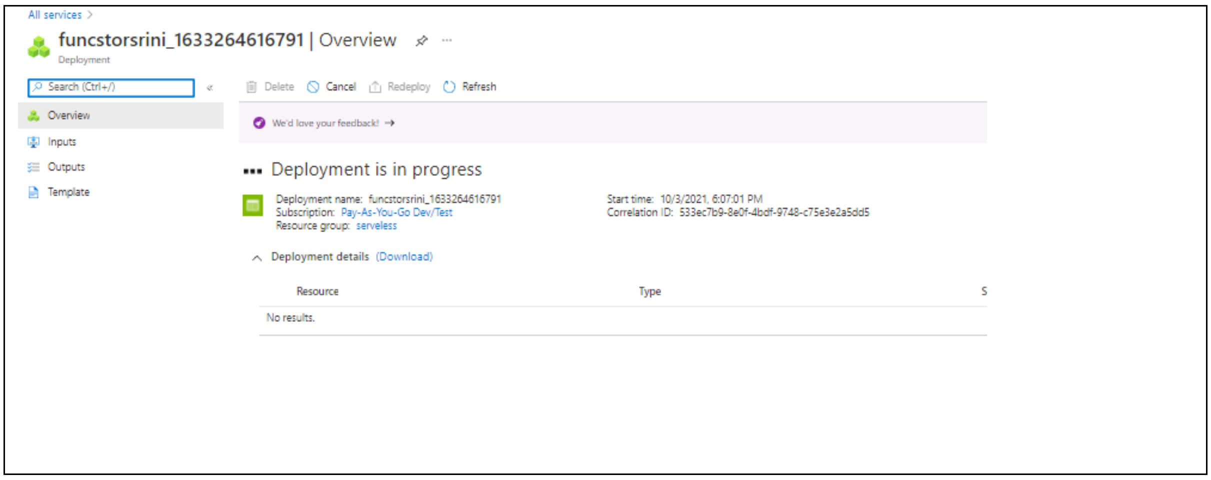
Network connectivity	Public endpoint (all networks)
Default routing tier	Microsoft network routing

Data protection

Point-in-time restore	Disabled
Blob soft delete	Enabled
Blob retainment period in days	7

8. 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.



All services >

funcstorsrini_1633264616791 | Overview

Deployment

Deployment is in progress

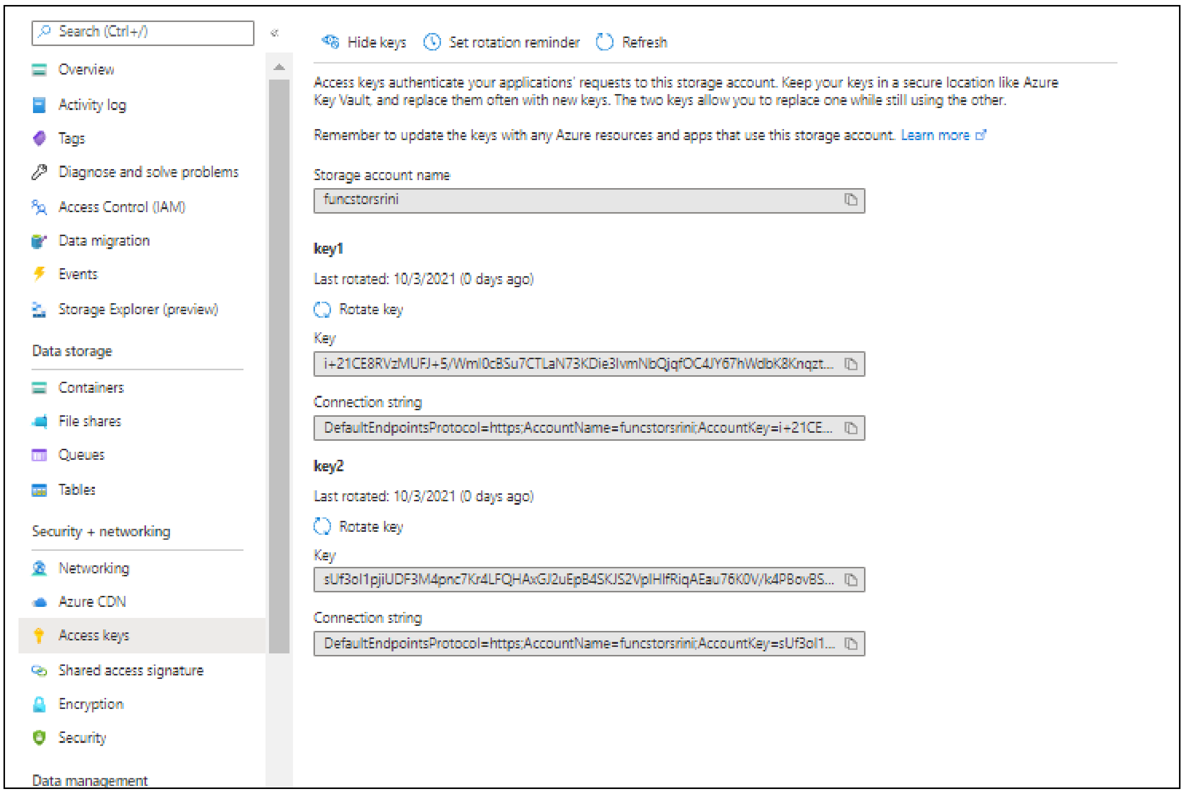
Deployment name: funcstorsrini_1633264616791
Subscription: Pay-As-You-Go Dev/Test
Resource group: serveless

Start time: 10/3/2021, 6:07:01 PM
Correlation ID: 533ec7b9-8e0f-4bdf-9748-c75e3e2a5dd5

Deployment details (Download)

Resource	Type	S
No results.		

9. In the Azure portal's navigation pane, select **All services**.
10. On the **All services** blade, select **Storage Accounts**.
11. On the **Storage accounts** blade, select the **funcstor[yourname]** storage account instance.
12. From the **Storage account** blade, find the **Security + networking** section, and then select **Access keys**.
13. From the **Access keys** blade, select **Show keys**.
14. Select any one of the keys, and then record the value of either of the **Connection string** boxes.



Access keys authenticate your applications' requests to this storage account. Keep your keys in a secure location like Azure Key Vault, and replace them often with new keys. The two keys allow you to replace one while still using the other.

Remember to update the keys with any Azure resources and apps that use this storage account. Learn more

Storage account name
funcstorsrini

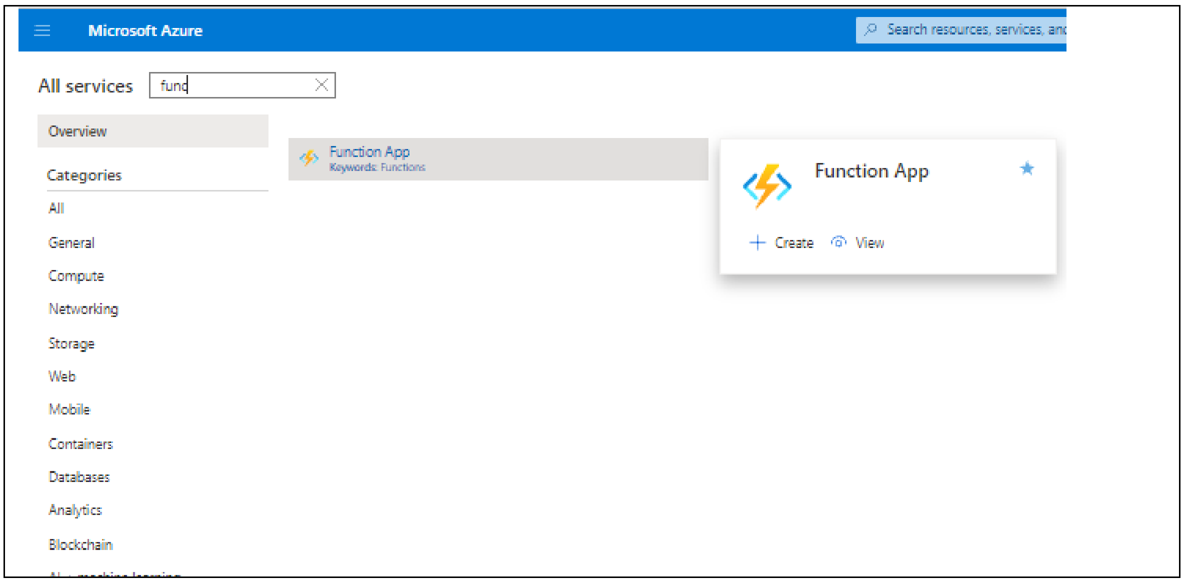
key1
Last rotated: 10/3/2021 (0 days ago)
Rotate key
Key
i+21CE8RVzMUFJ+5/Wml0cBSu7CTLaN73KDie3lvmNbQjqfOC4JY67hWdbK8Knqzt...
Connection string
DefaultEndpointsProtocol=https;AccountName=funcstorsrini;AccountKey=i+21CE...

key2
Last rotated: 10/3/2021 (0 days ago)
Rotate key
Key
sUf3ol1pjiUDF3M4pnc7Kr4LFQHAxGJ2uEpB4SKJS2VplHlfRiqAEau76K0V/k4PBovBS...
Connection string
DefaultEndpointsProtocol=https;AccountName=funcstorsrini;AccountKey=sUf3ol1...

Note: You'll use this value later in the lab. It doesn't matter which connection string you choose. They are interchangeable.

2.3 Task 3: Create a Function app

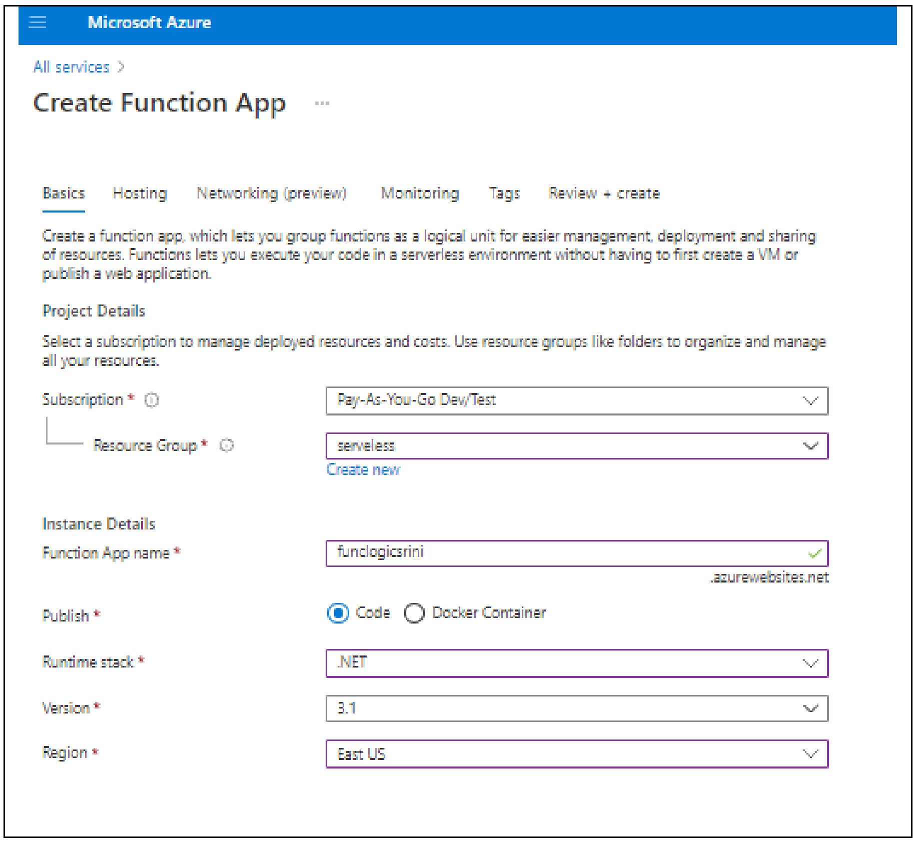
1. In the Azure portal’s navigation pane, select the **Create a resource** link.
2. From the **Create a resource** blade, find the **Search services and marketplace** text box.
3. In the search box, enter **Function**, and then select Enter.
4. On the **Everything** search results blade, select the **Function 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.

7. 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**.
   6. In the **Version** drop-down list, select **3.1**.
   7. In the **Region** drop-down list, select the **East US** region.
   8. Select **Next: Hosting**.



Create a function app, which lets you group functions as a logical unit for easier management, deployment and sharing of resources. Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

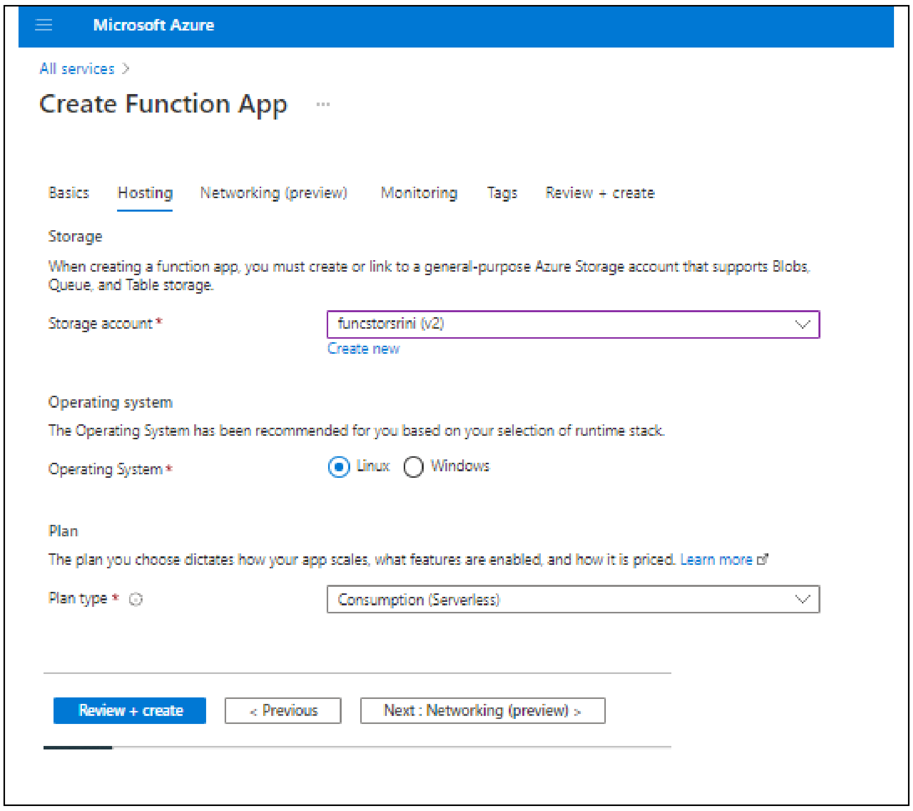
Subscription *	Pay-As-You-Go Dev/Test
Resource Group *	serveless

Create new

Instance Details

Function App name *	funclogicsrini .azurewebsites.net
Publish *	[✓ Code] Docker Container
Runtime stack *	.NET
Version *	3.1
Region *	East US

8. 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 **Linux**.
    3. In the **Plan type** drop-down list, select the **Consumption (Serverless)** option.



Storage

When creating a function app, you must create or link to a general-purpose Azure Storage account that supports Blobs, Queue, and Table storage.

Storage account *	funcstorsrini (v2)

Create new

Operating system

The Operating System has been recommended for you based on your selection of runtime stack.

Operating System *	[✓ Linux] Windows

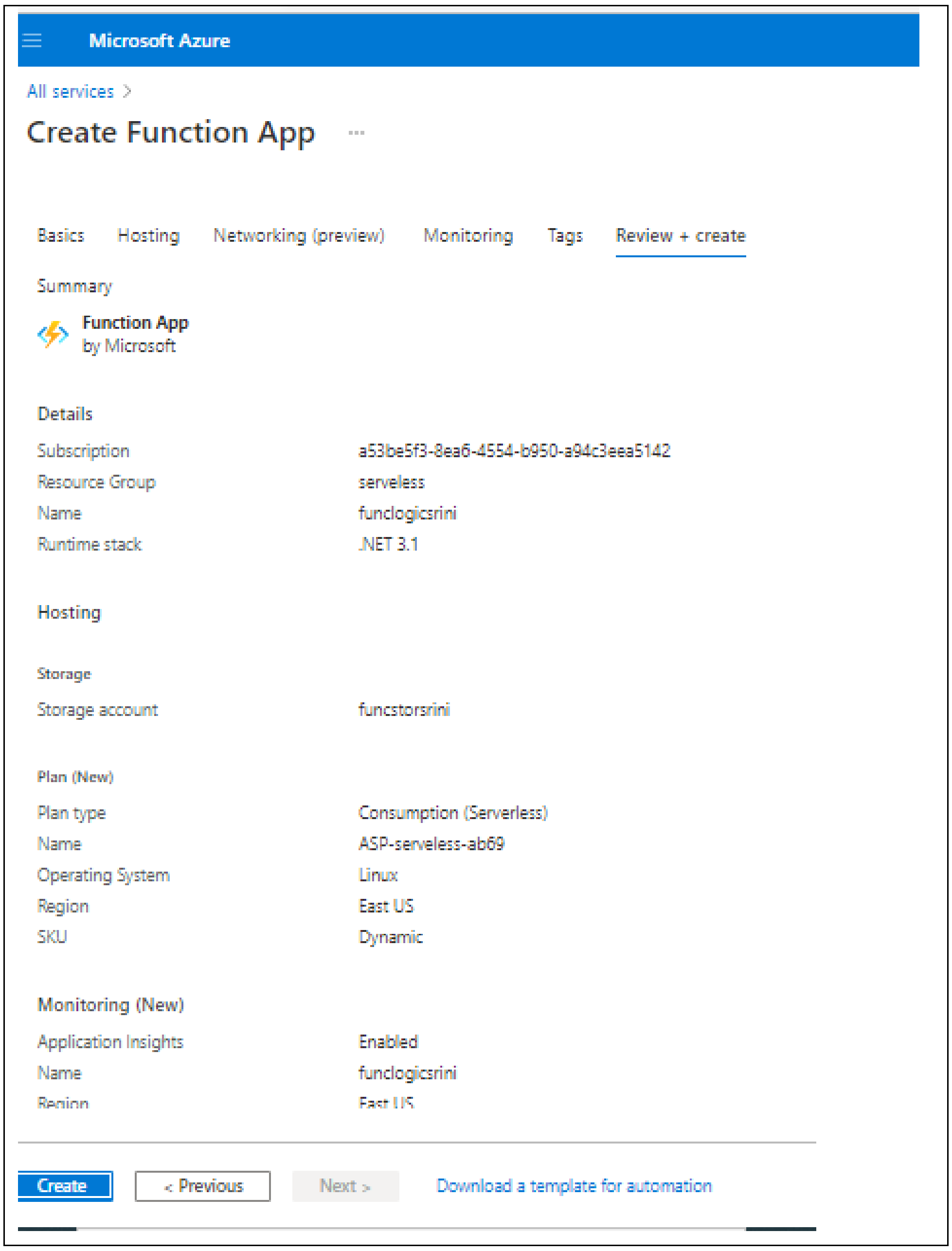
Plan

The plan you choose dictates how your app scales, what features are enabled, and how it is priced. Learn more

Plan type *	Consumption (Serverless)

4. Select **Review + Create**.

9. On the **Review + Create** tab, review the options that you selected during the previous steps.



Summary

Function App
by Microsoft

Details

Subscription	a53be5f3-8ea6-4554-b950-a94c3eea5142
Resource Group	serveless
Name	funclogicsrini
Runtime stack	.NET 3.1

Hosting

Storage

Storage account	funcstorsrini

Plan (New)

Plan type	Consumption (Serverless)
Name	ASP-serveless-ab69
Operating System	Linux
Region	East US
SKU	Dynamic

Monitoring (New)

Application Insights	Enabled
Name	funclogicsrini
Region	East US

10. 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.

3 Exercise 2: Configure a local Azure Functions project

3.1 Task 1: Initialize a function project

1. Naviage to Terminal.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

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

When you receive the open command prompt, enter the following command, and then select Enter to use the **Azure Functions Core Tools** to create a new local Azure Functions project in the current directory using the **dotnet** runtime:

```
func init --worker-runtime dotnet --force
```

3. **Note**: You can review the documentation to create a new project using the **Azure Functions Core Tools**.
4. Close the currently running **Windows Terminal** application.

3.2 Task 2: Configure connection string

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From 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. In the Explorer pane of the **Visual Studio Code** window, open the **local.settings.json** file.
5. Observe the current value of the **AzureWebJobsStorage** setting:

```
"AzureWebJobsStorage": "UseDevelopmentStorage=true",
```

5. Update the value of the **AzureWebJobsStorage** by setting it to the **connection string** of the storage account that you recorded earlier in this lab.

```
{
    "IsEncrypted": false,
    "Values": {
        "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=funcstorsrini;AccountKey=i+21CE8RVzMUFJ+5/WmI0cBSu7CTLaN73KDie3lvmNbQjqfOC4JY67hWdbK8KnqztXDcgl4EeLPzsy4oTJJ48Q==;EndpointSuffix=core.windows.net",
        "FUNCTIONS_WORKER_RUNTIME": "dotnet"
    }
}
```

13. Save the **local.settings.json** file.

3.3 Task 3: Build and validate a project

1. Naviage to the Terminal.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

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

When you receive the open command prompt, enter the following command, and then select Enter to **build** the .NET Core 3.1 project:

```
dotnet build
```

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

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

4.1 Task 1: Create an HTTP-triggered function

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

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

3. When you receive the open command prompt, enter the following command, and then select Enter to use the Azure Functions Core Tools to create a new function named Echo using the HTTP trigger template:

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

   3. **Note**: You can review the documentation to create a new function using the **Azure Functions Core Tools**.
4. Close the currently running **Windows Terminal** application.

4.2 Task 2: Write HTTP-triggered function code

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From 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. In the Explorer pane of the **Visual Studio Code** window, open the **Echo.cs** file.
5. In the code editor, observe the example implementation:

```
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)
        {
            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;

            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);
        }
    }
}
```

6. Delete all the content within the Echo.cs file.

   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:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
```

7. Create a new public static class named Echo:

```
public static class Echo{



}
```

8. 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:

9. 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:

```
[FunctionName("Echo")]
public static class Echo
{
    public static IActionResult Run(HttpRequest request, ILogger logger)
    {
    }
}
```

10. 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:

```
  [FunctionName("Echo")]
    public static IActionResult
    Run([HttpTrigger("POST")] HttpRequest request, ILogger logger)
    {

    }
```

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

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;

[FunctionName("Echo")]
public static class Echo
{
    [FunctionName("Echo")]
    public static IActionResult
    Run([HttpTrigger("POST")] HttpRequest request, ILogger logger)
    {

    }
}
```

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

```
logger.LogInformation("Received a request");
```

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

```
return new OkObjectResult(request.Body);
```

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

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;

[FunctionName("Echo")]
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);
   }
}
```

14. Select Save to save your changes to the Echo.cs file.

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

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

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

3. When you receive the open command prompt, enter the following command, and then select Enter to run the function app project:

```
func start --build
```

Note: You can review the documentation to start the function app project locally using the **Azure Functions Core Tools**.

4. On the taskbar, select the Windows Terminal icon again to open a new instance of the Windows Terminal application.

When you receive 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 http://localhost:7071:

```
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 functio projectp does not produce a Swagger definition file, you’ll need to traverse the API manually.

5. When you receive the tool prompt, enter the following command, and then select Enter to browse to the relative api directory:

   ```
   cd api
   ```

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

   ```
   cd echo
   ```

7. 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
   ```

8. 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
   ```

9. 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"
   ```

10. 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"}"
    ```

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

    ```
    exit
    ```

12.

12. Close all currently running instances of the Windows Terminal application.

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

5 Exercise 4: Create a function that triggers on a schedule

5.1 Task 1: Create a schedule-triggered function

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

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

3. When you receive the open command prompt, enter the following command, and then select Enter to use the Azure Functions Core Tools to create a new function named Recurring using the Timer trigger template:

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

Note: You can review the documentation to create a new function using the **Azure Functions Core Tools**.

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

5.2 Task 2: Observe function code

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From 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. In the Explorer pane of the **Visual Studio Code** window, open the **Recurring.cs** file.
5. In the code editor, observe the implementation:

```
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)
        {
            log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
        }
    }
}
```

```
6.
```

5.3 Task 3: Observe function runs

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

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

When you receive the open command prompt, enter the following command, and then select Enter to run the function app project:

```
func start --build
```

3. **Note**: You can review the documentation to start the function app project locally using the **Azure Functions Core Tools**.
4. Observe the function run that occurs about every five minutes. Each function run should render a simple message to the log.
5. Close the currently running **Windows Terminal** application.

5.4 Task 4: Update the function integration configuration

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From 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. In 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:

```
[FunctionName("Recurring")]

    public static void Run([TimerTrigger("0 */5 * * * *")]TimerInfo myTimer, ILogger log)
    {
        log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
    }
```

6. Update the Run method signature code block to change the schedule to execute once every 30 seconds:

```
[FunctionName("Recurring")]
      public static void Run(
          [TimerTrigger("0 */5 * * * *")] TimerInfo myTimer,
          ILogger log
      )
      {
          log.LogInformation(DateTime.Now.ToString());
      }
```

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

5.5 Task 5: Observe function runs

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

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

When you receive the open command prompt, enter the following command, and then select Enter to run the function app project:

```
func start –build
```

Note: You can review the documentation to start the function app project locally using the **Azure Functions Core Tools**.

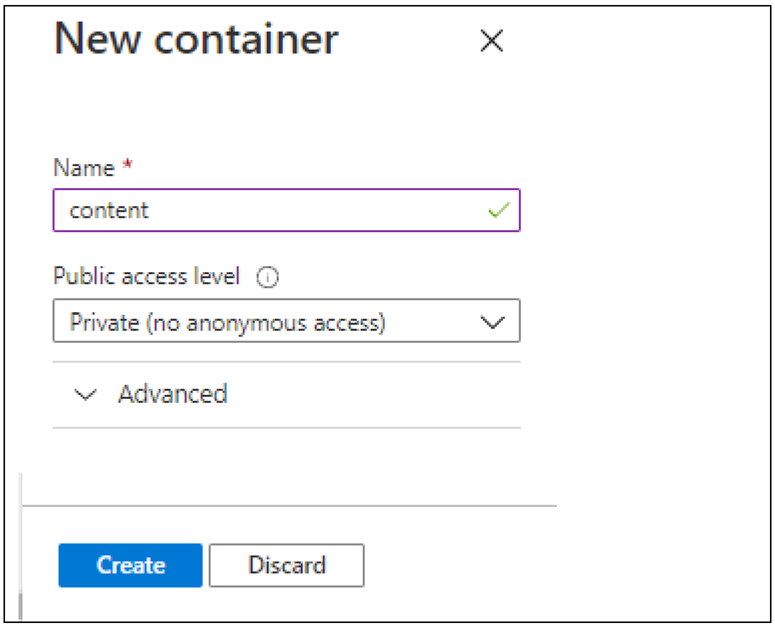
3. Observe the function run that occurs about every 30 seconds. Each function run should render a simple message to the log.
4. Close the currently running **Windows Terminal** application.

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

6 Exercise 5: Create a function that integrates with other services

6.1 Task 1: Upload sample content to Azure Blob Storage

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 **Data storage** 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 **Create**.



New container

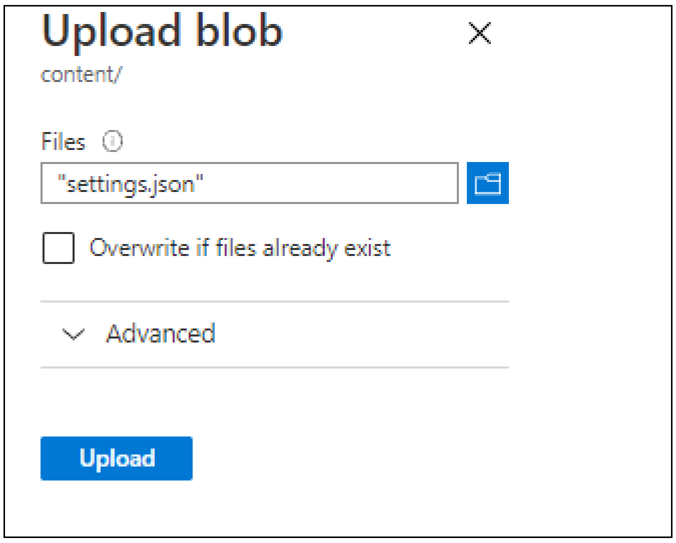
Name *

content

Public access level

Private (no anonymous access)

7. Return to the **Containers** section, and then 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.

6.2 Task 2: Create a HTTP-triggered function

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

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

When you receive the open command prompt, enter the following command, and then select Enter to use the **Azure Functions Core Tools** to create a new function named **GetSettingInfo** using the **HTTP trigger** template:

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

Note: You can review the documentation to create a new function using the **Azure Functions Core Tools**.

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

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

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From 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. In the Explorer pane of the **Visual Studio Code** window, open the **GetSettingInfo.cs** file.
5. In the code editor, observe the example implementation:

```
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)] HttpRequ
est 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;

            string responseMessage = string.IsNullOrEmpty(name)
                ? "This HTTP triggered function executed successfully. Pass a name in the qu
ery string or in the request body for a personalized response."
                : $"Hello, {name}. This HTTP triggered function executed successfully.";

            return new OkObjectResult(responseMessage);
        }
    }
}
6.
```

7. Delete all the content within the GetSettingInfo.cs file.

Add the following lines of code to add **using directives** for the **Microsoft.AspNetCore.Http**, **Microsoft.AspNetCore.Mvc**, and **Microsoft.Azure.WebJobs** namespaces:

```
 using Microsoft.AspNetCore.Http;
 using Microsoft.AspNetCore.Mvc;
 using Microsoft.Azure.WebJobs
```

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

8. Observe the GetSettingInfo.cs file again, which should now include:

```
using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs
```

9. 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:

```
  public static IActionResult Run(

     HttpRequest request,
     string json)
     => null;
```
Note: You are only temporarily setting the return value to **null**.

10. 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:

```
public static IActionResult Run(
    HttpRequest request,
    string json)
    => null;
```

11. 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:

```
[FunctionName("GetSettingInfo")]
 public static IActionResult Run(
     HttpRequest request,
     string json)
     => null;
```

12. 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:

```
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
    ) => null;
}
```
;

13. 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:

```
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);
}
```

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

6.4 Task 4: Register Azure Storage blob extensions

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:
3. When you receive the open command prompt, enter the following command, and then select Enter to register the Microsoft.Azure.WebJobs.Extensions.Storage extension:

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

4. Enter the following command, and then select Enter to validate the extensions were installed correctly by building the .NET project:

```
dotnet build
```

4.
5. Close all currently running instances of the **Windows Terminal** application.

6.5 Task 5: Test the function by using httprepl

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

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

3. When you receive the open command prompt, enter the following command, and then select Enter to run the function app project:

```
func start --build
```

Note: You can review the documentation to start the function app project locally using the **Azure Functions Core Tools**.

4. On the taskbar, select the Windows Terminal icon again to open a new instance of the Windows Terminal application.
5. When you receive 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 http://localhost:7071:

```
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 functio project does not produce a Swagger definition file, you’ll need to traverse the API manually.

6. When you receive the tool prompt, enter the following command, and then select Enter to browse to the relative api endpoint:

```
cd api
```

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

```
cd getsettinginfo
```

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

```
get
```

9. 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": "joseph.price@contoso.com",
       "phone": "(425) 555-0162 x4151"
   }
}
```

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

```
exit
```

10. 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.

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

7.1 Task 1: Deploy using the Azure Functions Core Tools

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

   code

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

When you receive the open command prompt, enter the following command, and then select Enter to login to the Azure Command-Line Interface (CLI):

```
az login
```

In the **Microsoft Edge** browser window, perform the following actions:

1. Enter the email address for your Microsoft account, and then select **Next**.
2. Enter the password for your Microsoft account, and then select **Sign in**.

Return to the currently open **Windows Terminal** window. Wait for the sign-in process to finish.

3. Enter the following command, and then select Enter to publish the function app project:

```
func azure functionapp publish <function-app-name>
```

6. **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 using the **Azure Functions Core Tools**.

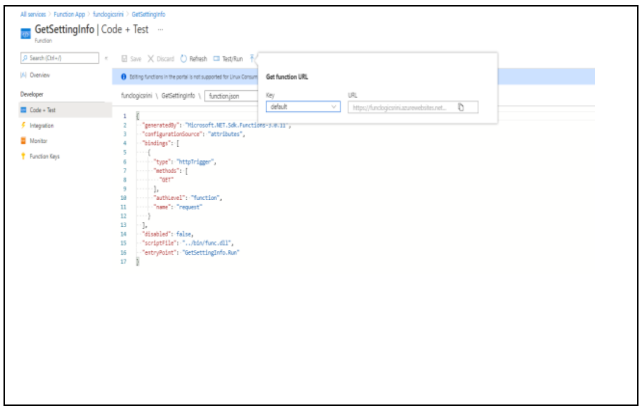
7. Wait for the deployment to finalize before you move forward with the lab.
8. Close the currently running **Windows Terminal** application.

7.2 Task 2: Validate deployment

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. In the Azure portal's navigation pane, select the **Resource groups** link.
4. On the **Resource groups** blade, find and then select the **Serverless** resource group that you created earlier in this lab.
5. On the **Serverless** blade, select the **funclogic[yourname]** function app that you created earlier in this lab.
6. From the **App Service** blade, select the **Functions** option from the **Functions** section.
7. In the **Functions** pane, select the the existing **GetSettingInfo** function.
8. In the **Function** blade, select the **Code + Test** option from the **Developer** section.
9. In the function editor, select **Test/Run**.
10. Select **Get Function URL**

11. In the popup dialog that appears, perform the following actions:

    Copy the URL as shown below



Editing functions in the portal is not supported for Linux Consum

funclogicsrini \ GetSettingInfo \ function.json

Key: default

URL: https://funclogicsrini.azurewebsites.net...

```
{
  "generatedBy": "Microsoft.NET.Sdk.Functions-3.0.11",
  "configurationSource": "attributes",
  "bindings": [
    {
      "type": "httpTrigger",
      "methods": [
        "GET"
      ],
      "authLevel": "function",
      "name": "request"
    }
  ],
  "disabled": false,
  "scriptFile": "../bin/func.dll",
  "entryPoint": "GetSettingInfo.Run"
}
```

12. Select **Run** to test the function.
13. Observe the results of the test run. The JSON content should now include:

```
{
    "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.

8 Exercise 7: Clean up your subscription

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

1. In the Azure portal, 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 (_).

2. 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 configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If Cloud Shell configuration options don't display, 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.

8.2 Task 2: Delete a resource group

1. When you receive 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.

8.3 Task 3: Close the active application

1. 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.