Implement Task Processing Logic by using Azure Functions
(LAB-204-06-01)

Lab scenario

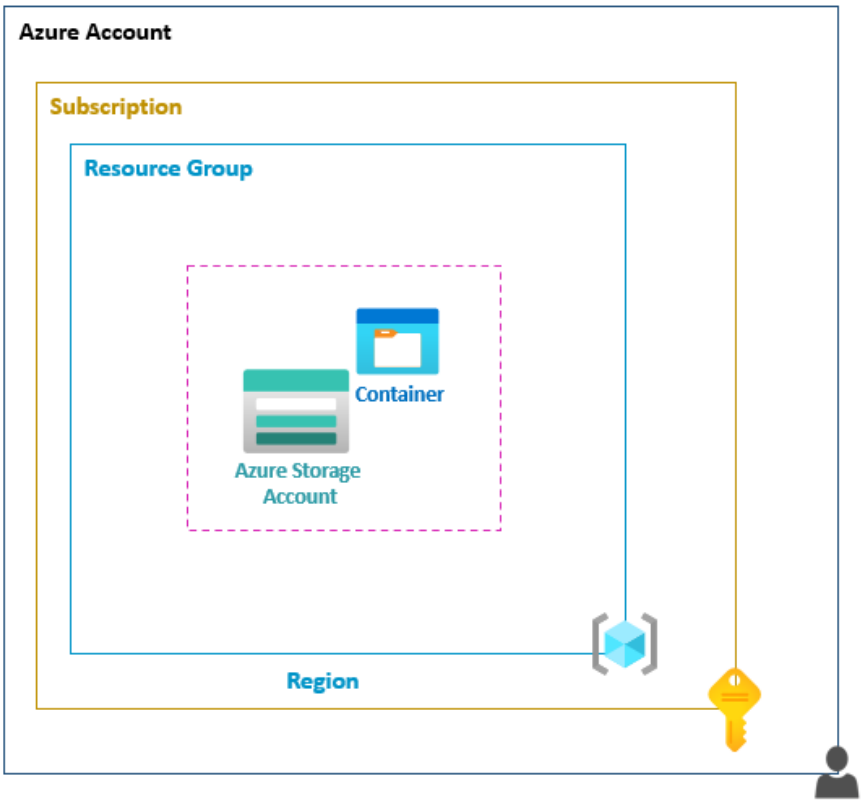
Your company has built a desktop software tool that parses a local JavaScript Object Notation (JSON) file for its configuration settings. During its latest meeting, your team decided to reduce the number of files that are distributed with your application by serving your default configuration settings from a URL instead of from a local file. As the new developer on the team, you've been tasked with evaluating Microsoft Azure Functions as a solution to this problem.

Objectives

After you complete this lab, you will be able to:

- Create a Functions app.
- Create various functions by using built-in triggers.
- Configure function app triggers and input integrations.

Task 1: Create and configure Azure Storage accounts



Step 1: Create Azure Storage Account

1. In the Azure portal, **go to the left** side, Select **Storage accounts**
2. Select **Add** and **Configure**:
   a. **Subscription**: Select your **Default Subscription**
   b. **Resource group**: Select **new** resource group **Az-204-06-01-RG**
   c. **Storage account name**: Write **funcstore123**

Note: **Replace 123** to make account name unique.

   d. **Location**: Select **East US**
   e. **Performance**: Select **Standard**
   f. **Replication**: Select **Locally-redundant storage (LSR)**
   g. **Select Next: Advanced**

Note: Leave all the details as default.

   h. Select **Next: Networking**

Note: Leave all the details as default.

   i. Select **Next: Data Protection**

Note: Leave all the details as default.

   j. Select **Next: Tags**

Note: Leave all the details as default.

   k. Select **Next: Review + Create**

Note: Leave all the details as default.

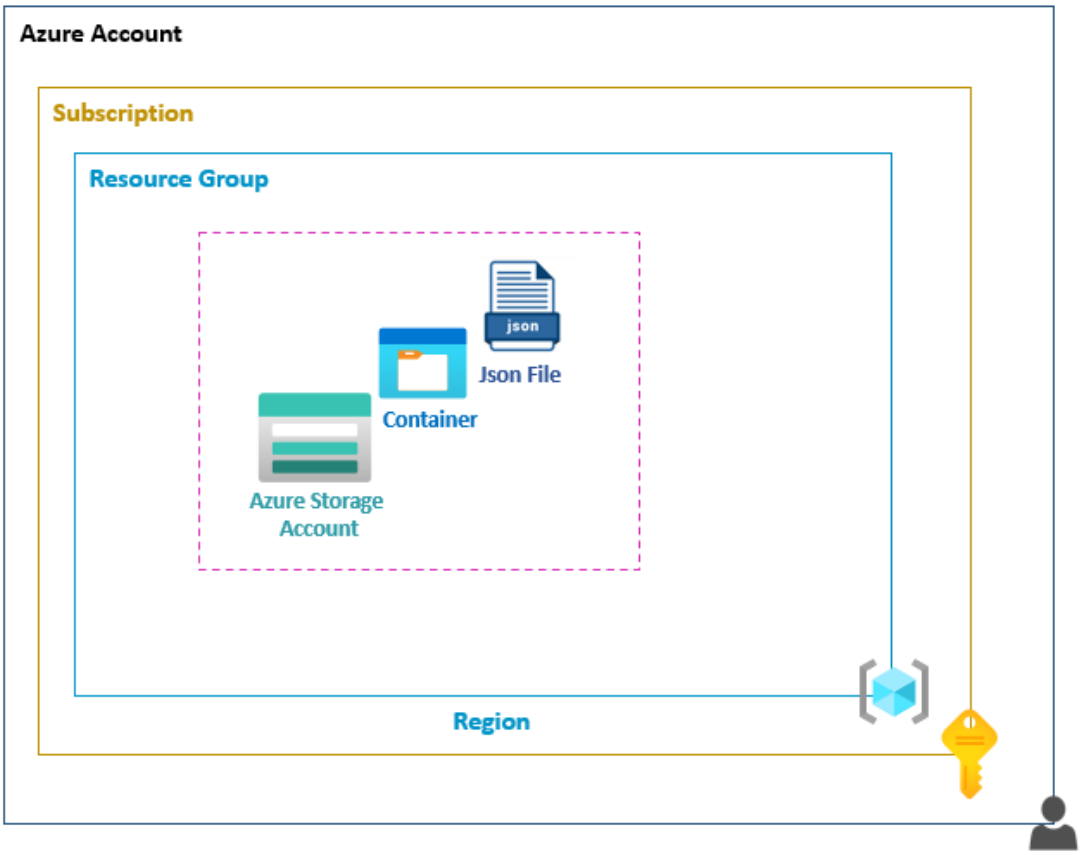
   l. Select **Create**

Note: **Wait**, until storage account gets **created**.

Step 2: Create a Container

3. In the Azure portal, go to the left side, Select **Storage accounts**
4. Open Storage account **funcstore123**
5. Select **Containers** under **Data storage**.
6. Select the **Container**
    a. **Name**: Write **content**
    b. **Public access level**: Select **Private (no anonymous access)**
    c. Select **Create**

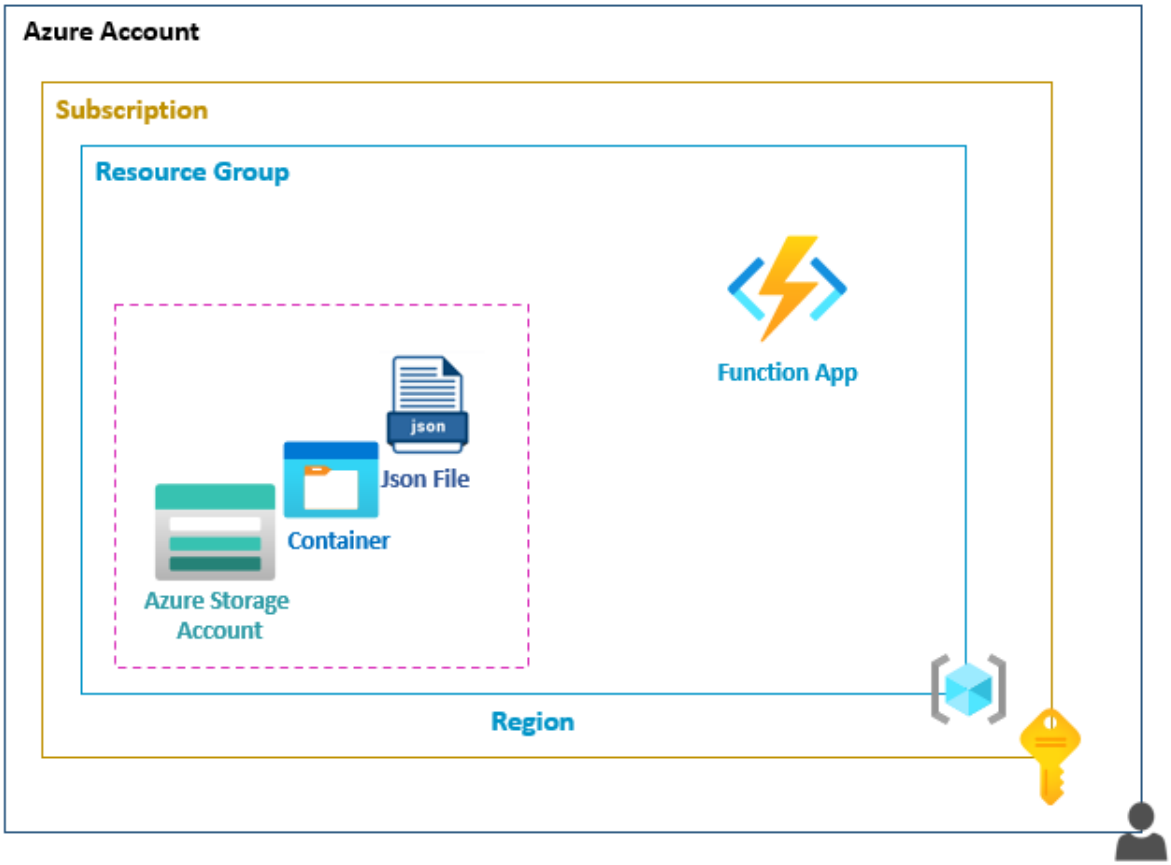
Step 3: Upload a JSON Files



7. **Open** the container **content**.
    a. Select the **Upload** button to upload files to the container.
    b. Select the **settings.json** files and select **Upload**

Note: **settings.json** file is provided with the Lab manual.

Task 2: Create a Function that's triggered by an HTTP request



Step 1: Create Function App Resource

8. ***Go to the left*** side, Select **Create a Resource**
9. Search & Select **Function App**
10. Select **Create** & configure
    a. **Subscription**: Select your **Default subscription**
    b. **Resource group**: Select **existing** resource group **Az-204-06-01-RG**
    c. **Function App Name**: Write **funclogic-123**

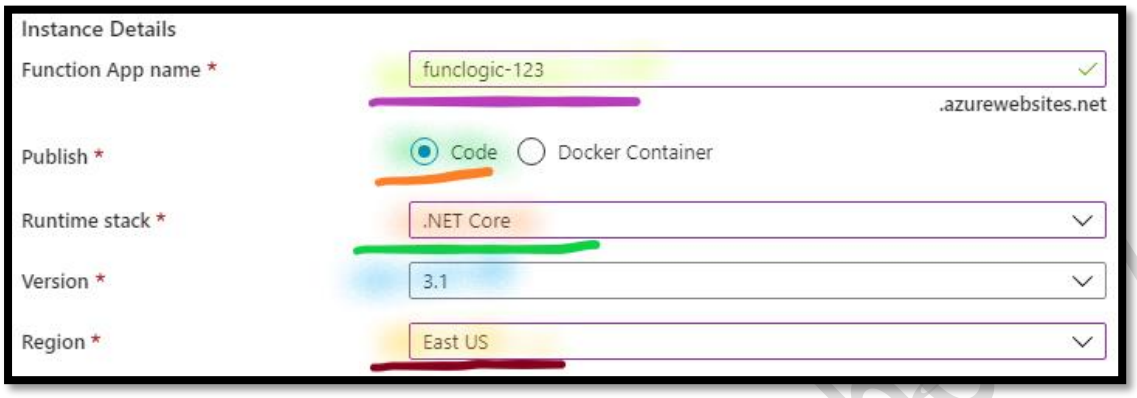
Note: **Replace 123** to make account name unique.

    d. **Publish**: Select **Code**

e. **Runtime stack**: Dropdown and select **.Net**

f. **Version**: Dropdown and select **3.1**

g. **Region**: Select **East US**

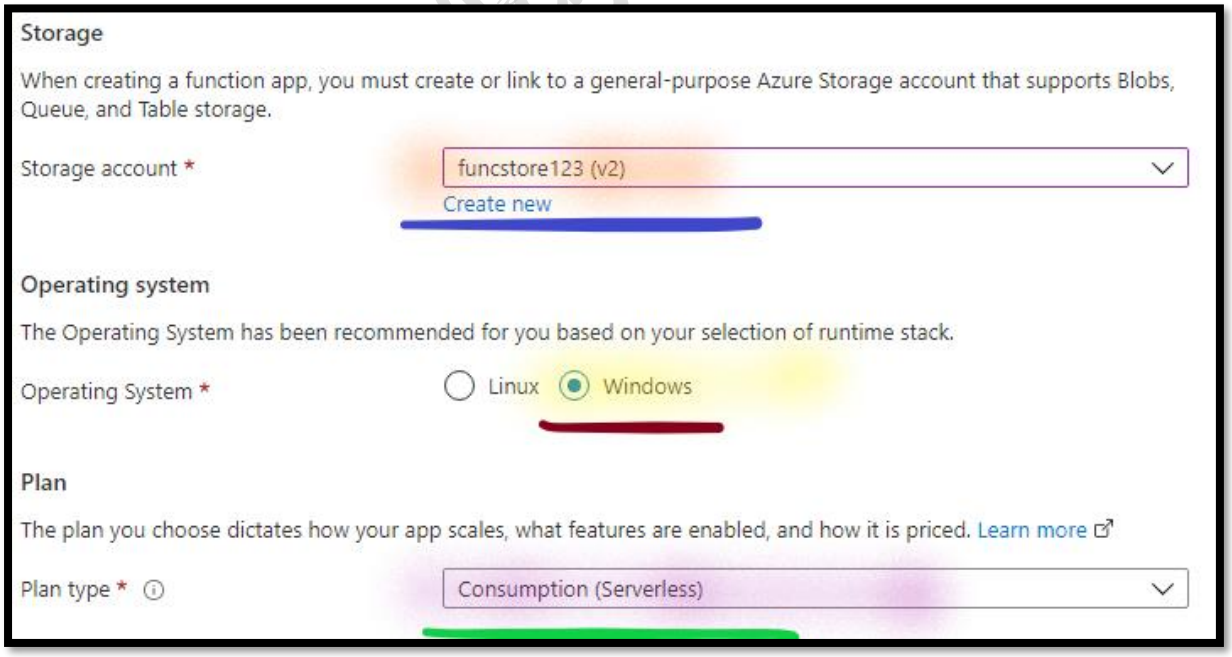


h. Select **Next: Hosting**

i. **Storage account**: Dropdown and select the **funcstor123** storage account that you created earlier in this lab.

ii. **Operating System**: Select **Windows**.

iii. **Plan type**: Dropdown and select **Consumption (Serverless)**.



Note: Leave other details as Default.

i. Select **Next: Monitoring**

Note: Leave all the details as Default.

j. Select **Next: Tags**

Note: Leave all the details as Default.

k. Select **Next: Review + Create**

l. Select **Create**

Note: **Wait**, till resource gets **deployed**.

Step 2: Verify the Configuration settings

11. ***Go to left*** side, click on **Resource Group**

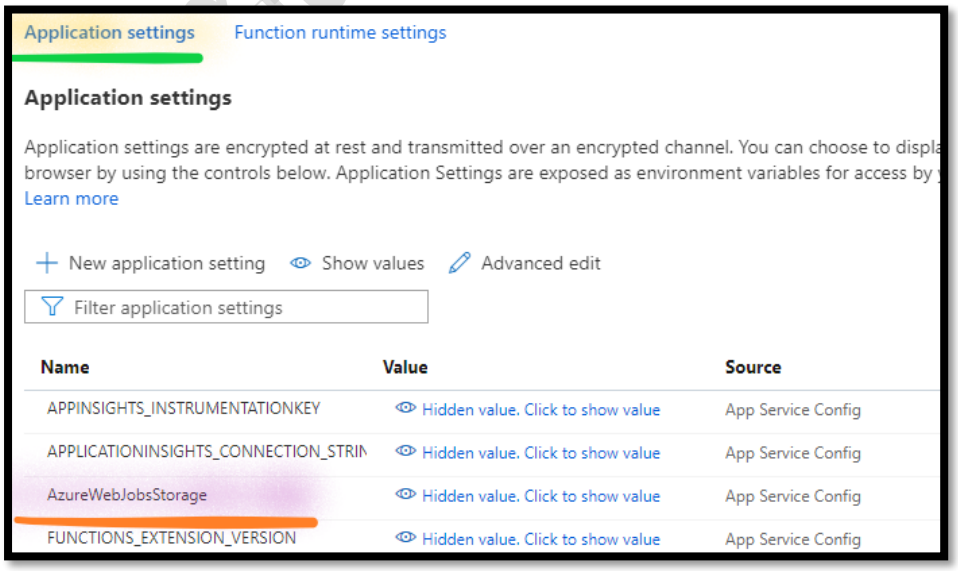
12. Open **Az-204-06-01-RG** resource group

13. Open the **funclogic-123** function app

    a. **Go to left**, Select **Configuration**.

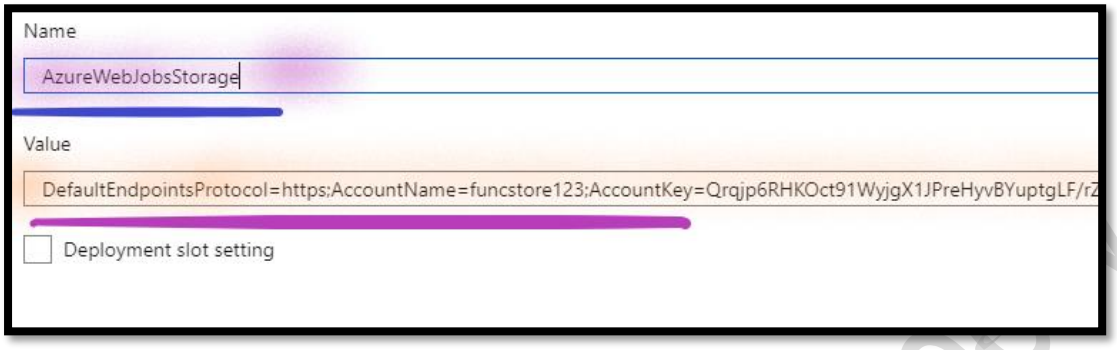
    b. Select **Application settings**.

Note: You can see the **AzureWebJobsStorage** settings.



c. Open the **AzureWebJobsStorage** settings.

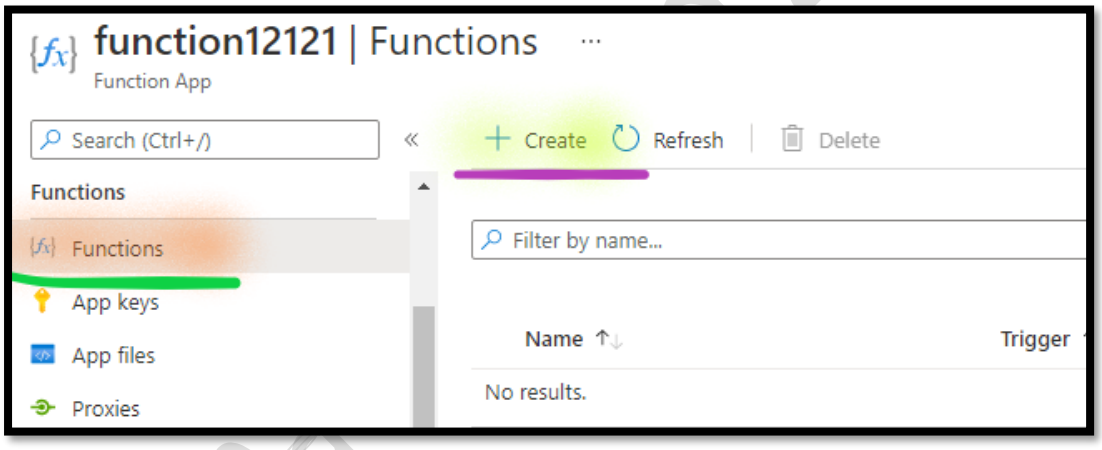
Note: You can see the **Connection String** of Azure Storage account.



Step 3: Create an HTTP-triggered function

14. **Go to left** side, Select **functions** under **Functions.**

15. Select **+Create**



a. **Development environment**: Dropdown and Select **Develop in portal**.

b. Select **HTTP trigger** under **templates**.

c. Select **Create**.

Template	Description
HTTP trigger	A function that will be run whenever it receives an HTTP request, responding based on data in the b query string
Timer trigger	A function that will be run on a specified schedule
Azure Queue Storage trigger	A function that will be run whenever a message is added to a specified Azure Storage queue
Azure Service Bus Queue trigger	A function that will be run whenever a message is added to a specified Service Bus queue
Azure Service Bus Topic trigger	A function that will be run whenever a message is added to the specified Service Bus topic
Azure Blob Storage trigger	A function that will be run whenever a blob is added to a specified container
Azure Event Hub trigger	A function that will be run whenever an event hub receives a new event

Create Cancel

Note: It will Open the Function Section.

Step 4: Update the function code

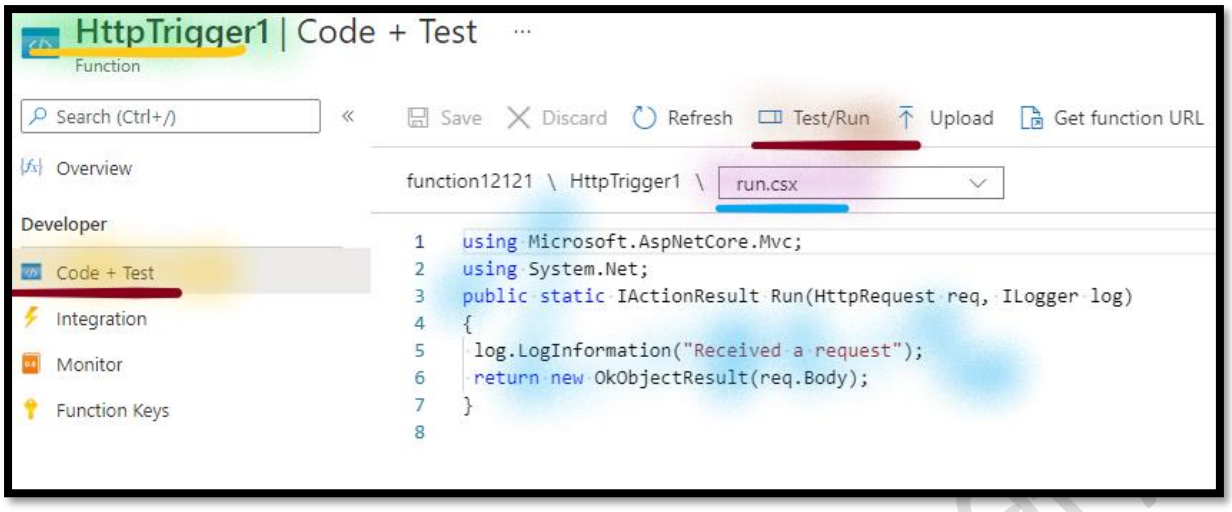
16. Select the **Code + Test** option from the **Developer** section.

    a. In the **Function editor** **Delete** ***the example code*** in the **run.csx** function script.

    b. **Add** the **following using directives** for libraries that will be referenced by the application:

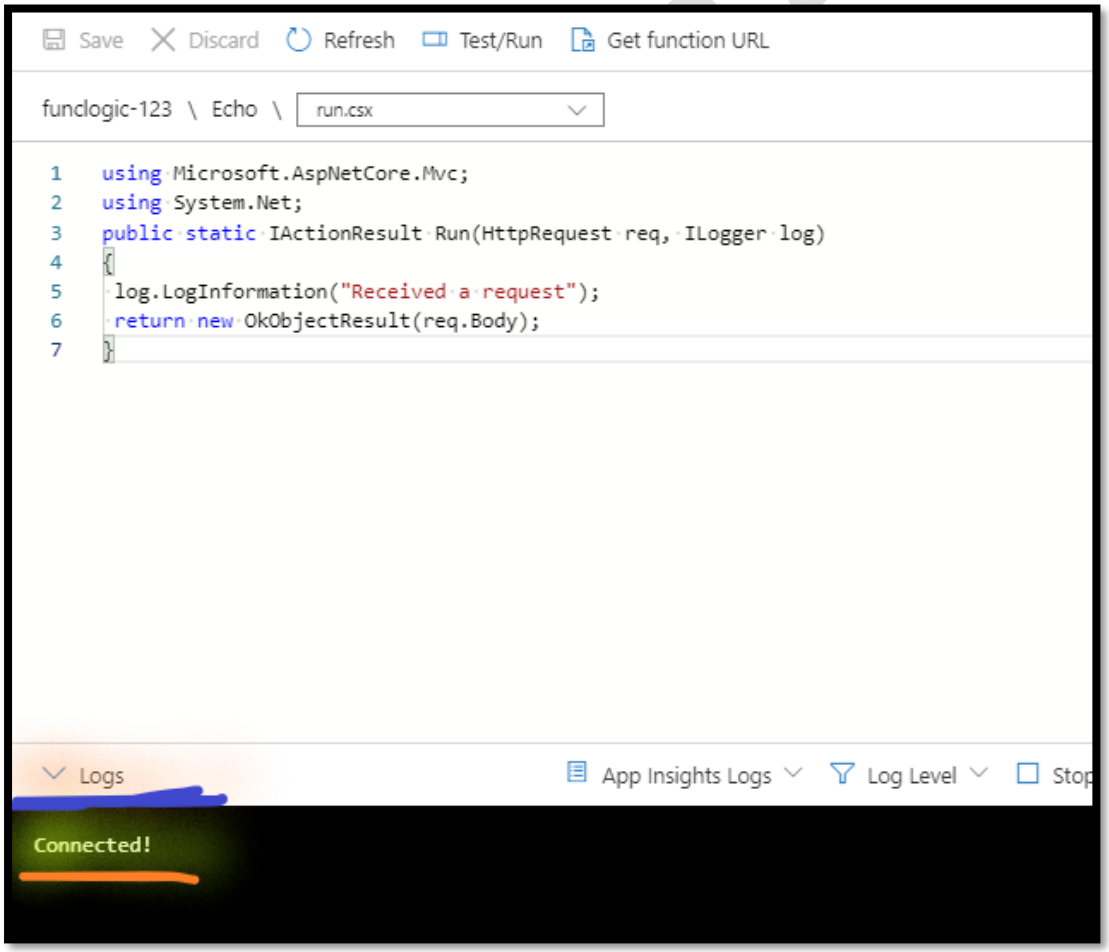
```
using Microsoft.AspNetCore.Mvc;
using System.Net;
public static IActionResult Run(HttpRequest req, ILogger log)
{
 log.LogInformation("Received a request");
 return new OkObjectResult(req.Body);
}
```

    c. Select the **Save**



Note: **Copy** the Function name (like **HttpTrigger1**) in **Notepad**.

d. Under the **Logs**, it shows as **Connected**.



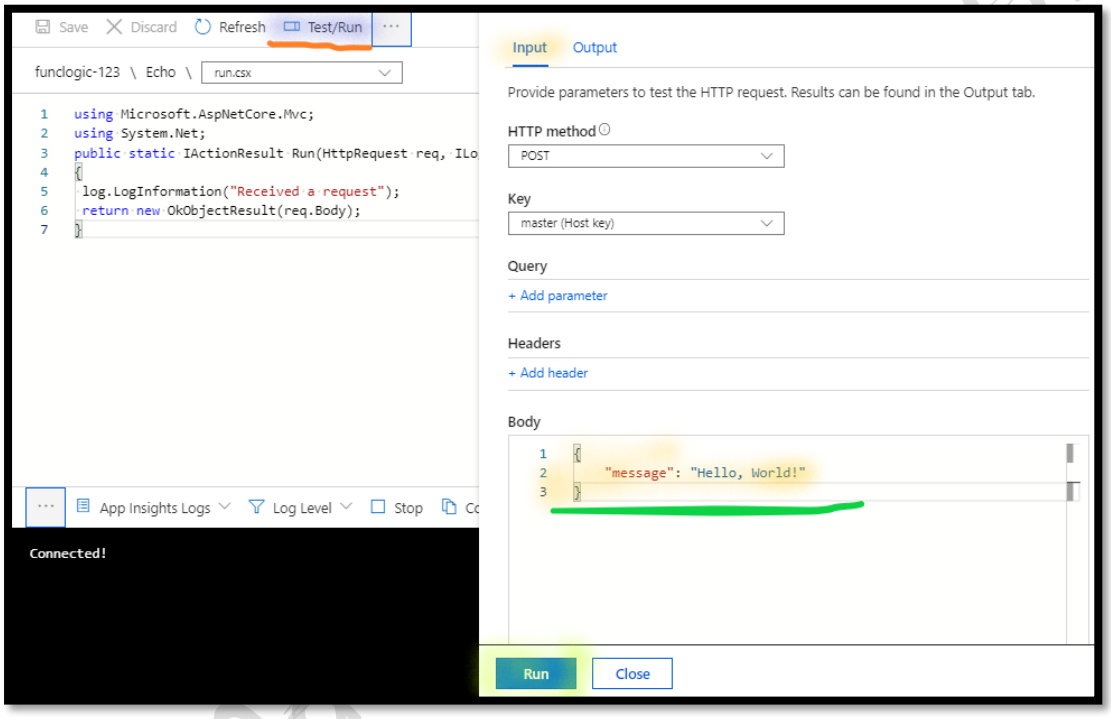
Step 5: Test function run in the portal

17. Select **Test/Run**.

    a. Within the **Body** section, ***remove the existing input*** & **Replace** it with the following **JSON** request body:

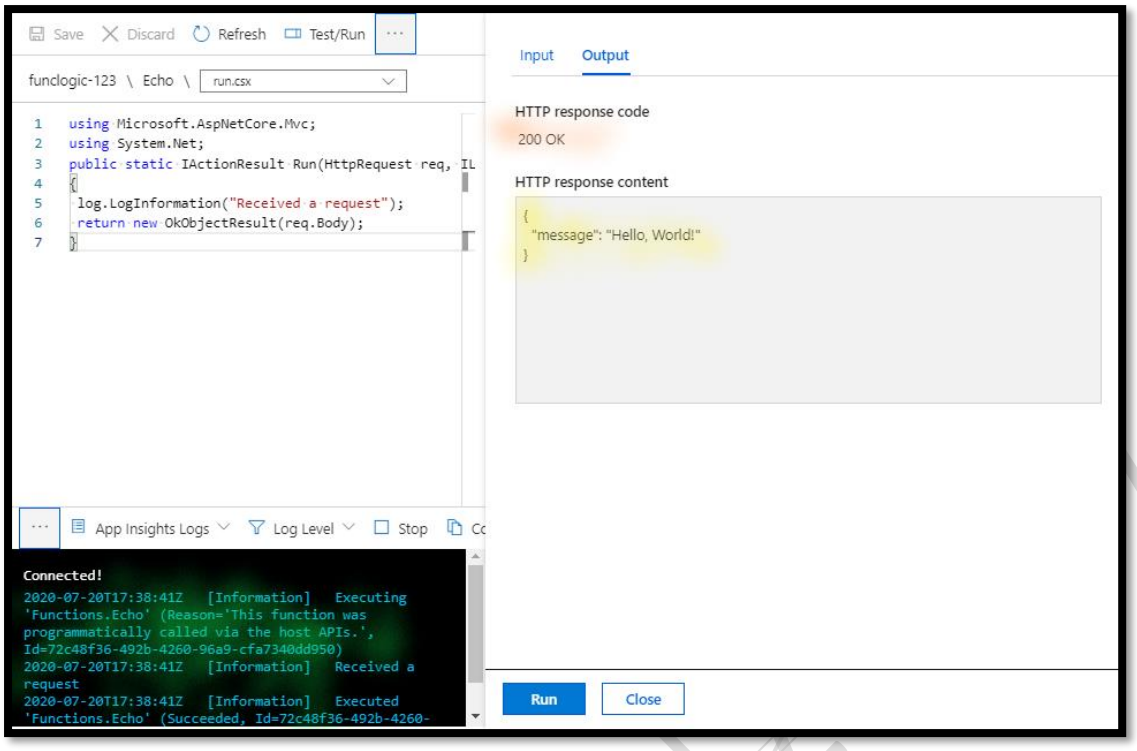
```
{
   "message": "Hello, World!"
}
```

    b. Select **Run** to test the function.



Note: Observe the results of the test run. The results should echo the original request body exactly.

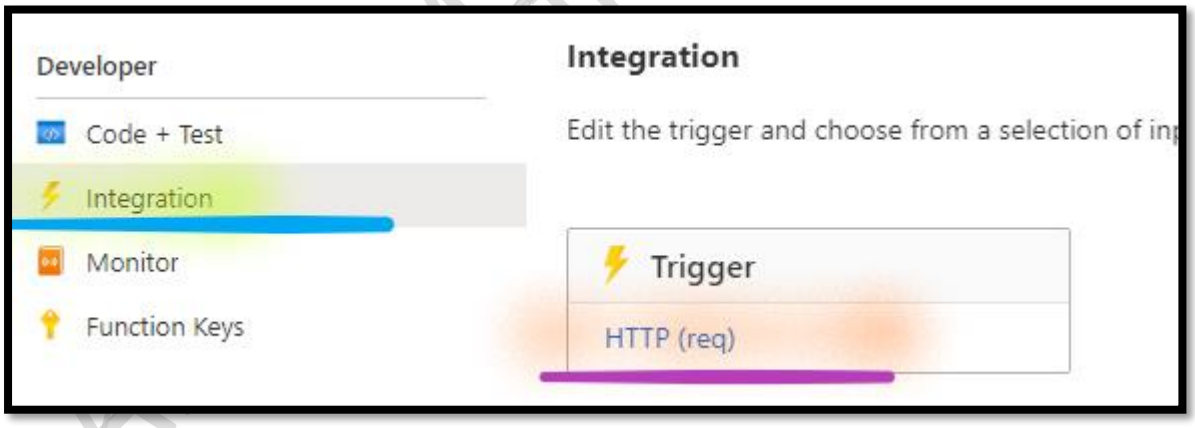
In the Logs, you can also the detail logs of execution.



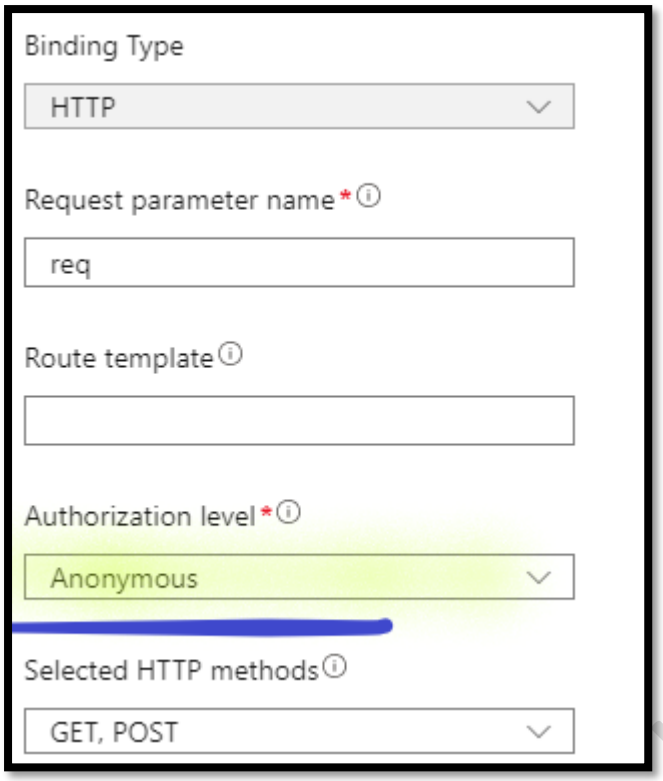
Step 5: Update the Trigger Request

18. Select the **Integration** option from the **Developer** section.

    a. Open the **HTTP (req)**.



    b. **Authorisation level**: Dropdown and Select **Anonymous**.



c. Select **Save**.

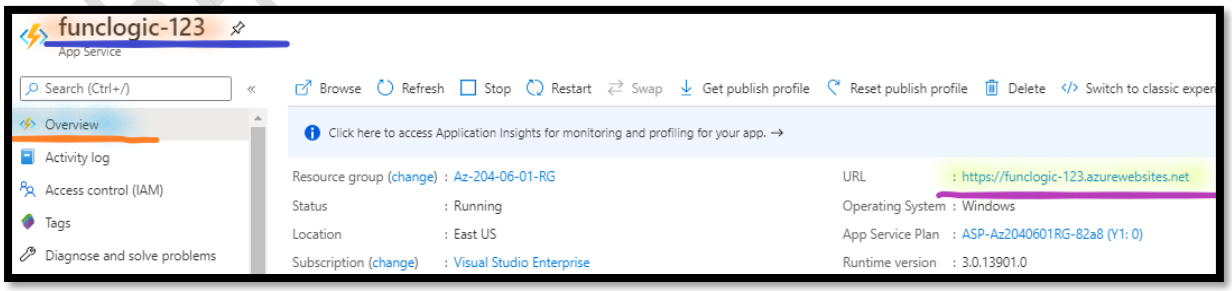
Step 6: Copy the Base Function URL

19. Go to left side, click on **Resource Group**

20. Open **Az-204-06-01-RG** resource group

21. Open the **funclogic-123** function app

    a. **Copy** the **URL** in **Notepad**.



Task 3: Deploy Virtual Machine with Dot Net SDK

Step 1: Create Windows Virtual Machine

22. Click the **virtual machines** link in the left-hand navigation bar.

23. Click the **Create** button to start the creation process.

24. You will be required to **fill in specific information** regarding your virtual machine, including:

    a. **Subscription**: Select **Default subscription**

    b. **Resource Group**: Use **existing** resource group **Az-204-06-01-RG**

    c. **Name**: Write **LAB-204-06-VM**

    d. **Region**: Select region **East US**

    e. **Image**: Dropdown and Select **Windows Server 2016 Datacenter**

    f. **Size**:

        i. Select **Change size**

        ii. Search & **Select** **B2ms** virtual machine

    g. **Administrator Account**:

        i. **Username**: Write **master**

        ii. **Password**: Write **Lab@password**

    h. **Inbound Port Rules**:

        i. **Public inbound ports**: Select **Allow selected ports**

        ii. **Select inbound ports**:

            a. Dropdown and select **RDP (3389)**

25. Click the **Next: Disks** to continue

26. Click the **Next: Networking** to continue.

27. Click the **Next: Management** to continue.

28. Click on the **Next: Advanced** to continue

29. Click the **Next: Tags** to continue.

30. Click the **Next: Review + create** button to continue.

31. Click the **Create** button

Note: The deployment process may take a few minutes. Don't wait, go to the next step.

Step 2: Connect to Windows 2019 Virtual Machine

32. Go to the left side of the menu, select **virtual machines**.

33. Select & Open the **LAB-204-06-VM** virtual machine.

34. On the right side of the page copy **Public IP Address**.

35. **Connect** to **LAB-204-06-VM** virtual machine via **RDP**.

Step 3: Install Dot Net SDK

36. From the **LAB-204-06-VM** server, Go to **Start menu**, open **Server manager**

    a. Select **Local Server**

    b. Click in **ON** showing against IE Enhanced Security Configuration

    c. Select **Off** in Administrator and Select **Ok**.

    d. **Refresh** you screen & now you can see **OFF** showing against IE Enhanced Security Configuration.

37. **Download** and **Install** **.Net Core SDK 3.1** or ***above*** Edition.

https://download.visualstudio.microsoft.com/download/pr/4e88f517-196e-4b17-a40c-2692c689661d/eed3f5fca28262f764d8b650585a7278/dotnet-sdk-3.1.301-win-x64.exe

Note: **Wait** for installation **completion**.

38. From the **LAB-204-06-VM** server, right click on **Start** & **Run**

   a. In **Open** write **CMD**

   b. From **command line**, write **dotnet --version**

Note: You can see the **dotnet version 3.1**.

39. **Download** and **Install** the **HTTP REPL**, run the following command:

   a. From **command line**, write:

   dotnet tool install -g Microsoft.dotnet-httprepl

Note: You can see the output, httprepl installed succesfully.

Info: The HTTP Read-Eval-Print Loop (REPL) is A lightweight, cross-platform command-line tool, used for making HTTP requests to test web APIs and view their results.

```
C:\Users\azureadmin>dotnet tool install -g Microsoft.dotnet-httprepl

Welcome to .NET Core 3.1!
---------------------
SDK Version: 3.1.301

Telemetry
---------
The .NET Core tools collect usage data in order to help us improve your experience. The data is anonymous. It is coll
ed by Microsoft and shared with the community. You can opt-out of telemetry by setting the DOTNET_CLI_TELEMETRY_OPTOU
nvironment variable to '1' or 'true' using your favorite shell.

Read more about .NET Core CLI Tools telemetry: https://aka.ms/dotnet-cli-telemetry

----------------
Explore documentation: https://aka.ms/dotnet-docs
Report issues and find source on GitHub: https://github.com/dotnet/core
Find out what's new: https://aka.ms/dotnet-whats-new
Learn about the installed HTTPS developer cert: https://aka.ms/aspnet-core-https
Use 'dotnet --help' to see available commands or visit: https://aka.ms/dotnet-cli-docs
Write your first app: https://aka.ms/first-net-core-app
--------------------------------------------------------------------------------------
Since you just installed the .NET Core SDK, you will need to reopen the Command Prompt window before running the tool
u installed.
You can invoke the tool using the following command: httprepl
Tool 'microsoft.dotnet-httprepl' (version '3.0.47301') was successfully installed.
```

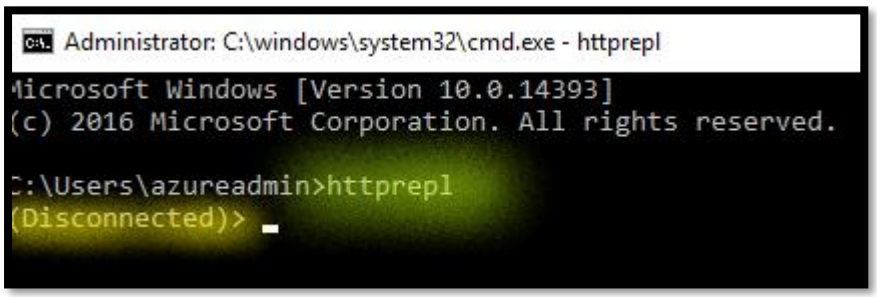
   b. Close the **CMD** tool

40. **From** the **LAB-204-06-VM** server, right click on **Start** & **Run**

   a. In **Open** write **CMD**

   b. **Test** the **HTTPREPL** command, **run** the following **command**:
   httprepl

Note: You can see the output, shown as **disconnected**.



c. Write **Exit**

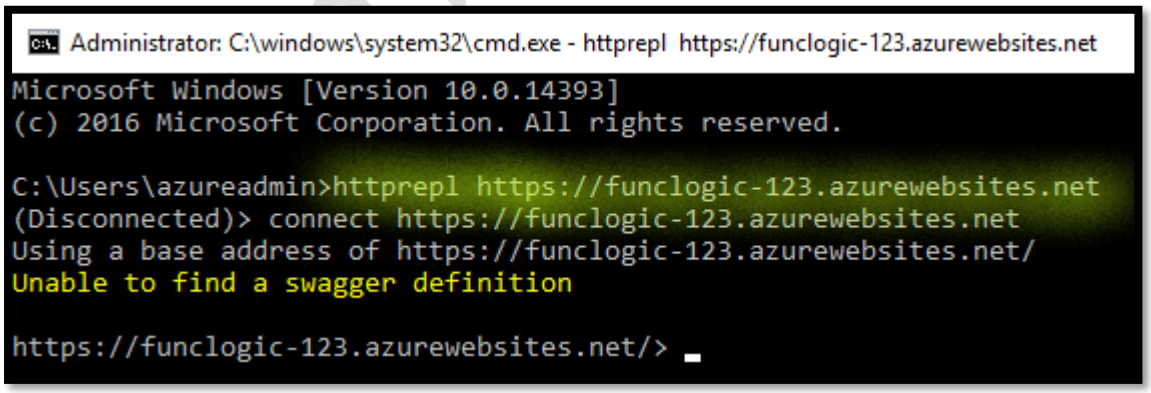
Step 4: Test function run by using Httprepl

41. From the **LAB-204-06-VM** server, right click on **Start** & **Run**

a. In **Open** write **CMD**

b. Start the **httprepl** tool, and set the base Uniform Resource Identifier (URI) to the value of the Request URL for the API operation. Execute the below command:
httprepl <Function-App -URL>

Note: **Replace** the **Function-App-URL**, with the Request URL you have copied in the previous step.



Note: Observe the error message 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 Logic App does not produce a Swagger definition file, you will need to traverse the API manually.

Info: Swagger is a set of open-source tools built around the OpenAPI Specification that can help you design, build, document and consume REST APIs.

c. At the tool prompt, **browse** to the relative **api/ HttpTrigger1 directory**:
cd api

cd HttpTrigger1

Info: Change the **HttpTrigger1** to other name, as you have copied in the previous step.

```
C:\Users\azureadmin>httprepl https://function12121.azurewebsites.net
(Disconnected)> co
Using a base address of https://function12121.azurewebsites.net/
Unable to find an OpenAPI description
For detailed tool info, see https://aka.ms/http-repl-doc

https://function12121.azurewebsites.net/> cd api

https://function12121.azurewebsites.net/api> cd HttpTrigger1

https://function12121.azurewebsites.net/api/HttpTrigger1>
```

i. **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

Note: Observe the response content.

```
https://function12121.azurewebsites.net/api/HttpTrigger1> post --content 3
HTTP/1.1 200 OK
 2021 11:05:32 GMT
Request-Context: appId=cid-v1:9f6f1e1e-baf6-460b-888a-2463748d8f93
Transfer-Encoding: chunked

3
```

ii. **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

iii. **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"

iv. **Run** the post command sending in an HTTP request body set to a JSON value of {"msg": "Successful"} by using the --content option:

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

v. **Exit** the **httprepl** application:

Exit

Task 4: Create a Function that's triggers on Schedule

Step 1: Create Function

42. Go to left side, click on **Resource Group**
43. Open **Az-204-06-01-RG** resource group
44. Open the **funclogic-123** function app
45. Select **Functions** under **functions**
46. Select **+Create**

    a. **Development environment**: Dropdown and Select **Develop in portal**.

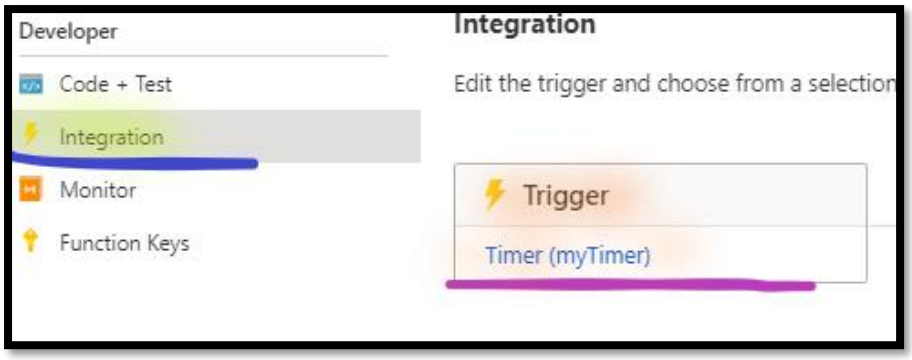
    b. Select **Timer trigger** under **templates**

    c. Select **Create function**

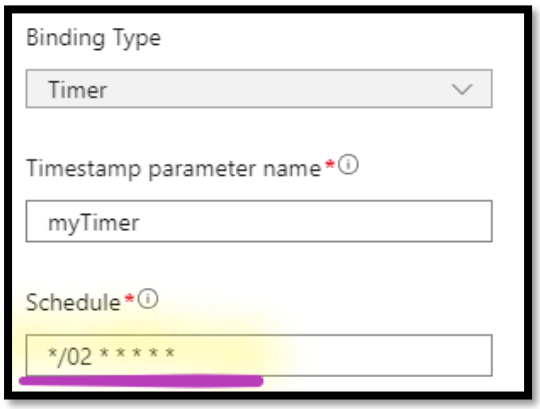
> **Note**: It will Open the Function Section.

    d. ***Go to left***, Select **Integration** under **Developer** section:

    e. **Open** the **Timer (myTimer)**



f. **Schedule**: Write ***/02 * * * * ****



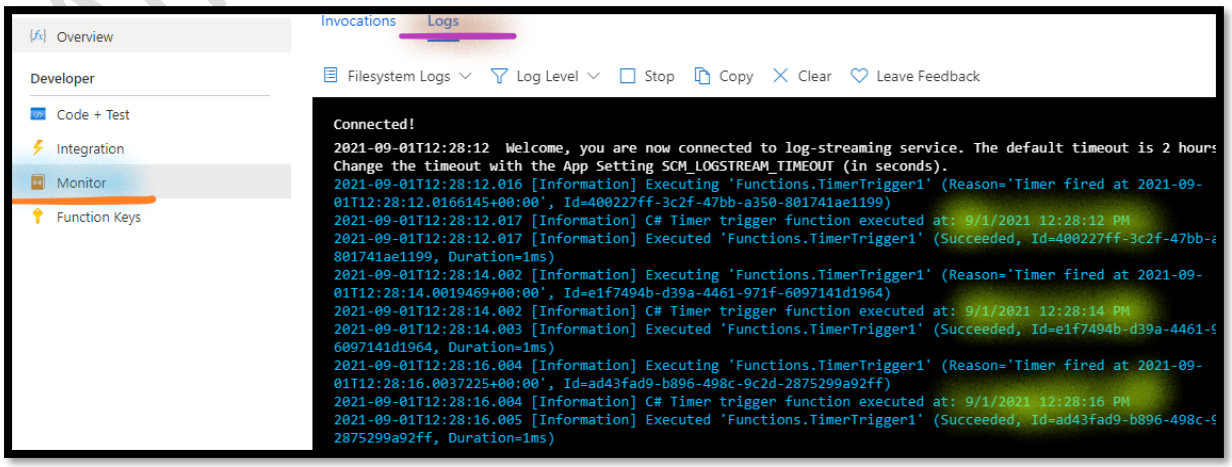
g. Select **Save**.

Step 2: Observe Function Runs

47. **Go to left** side, Select **Monitor**

a. Select **Logs**

Note: Observe the **function run** that occurs about **every two minutes**.



Step 3: Update the Function Integration Configuration

48. ***Go to left***, Select **Integration** under **Developer** section:

    a. **Open** the **Timer (myTimer)**

    b. **Schedule**: Update ***/30 * * * * ****

49. Select the **Save**

Step 4: Observe Function Runs

50. **Go to left** side, Select **Monitor**

    a. Select **Logs**

> **Note**: Observe the **function run** that occurs about **every two minutes**.

Task 5: Create a Function that's that Integrates with other services

Step 1: Create Function

51. Go to left side, click on **Resource Group**

52. Open **Az-204-06-01-RG** resource group

53. Open the **funclogic-123** function app

54. Select **Functions** under **functions**

55. Select **+Create**

    a. **Development environment**: Dropdown and Select **Develop in portal**.

    b. Select **HTTP trigger** under **templates**.

    c. Select **Create**.

Note: It will Open the **HttpTrigger2 Function** Section.

Step 2: Update the Trigger Function

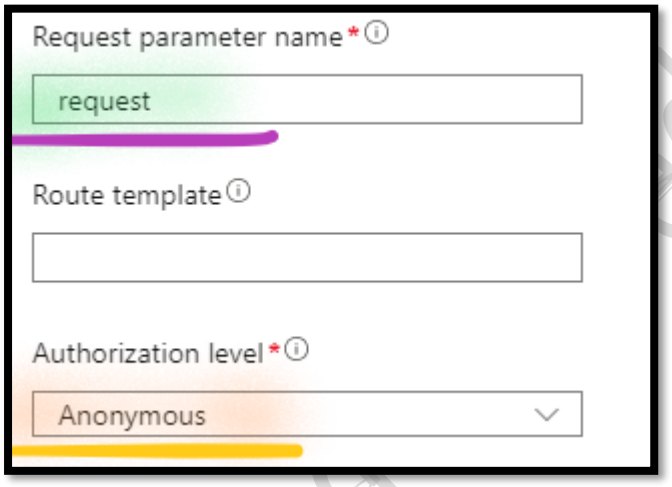
56. Select the **Integration** option from the **Developer** section.

    a. Open the **HTTP (req)**.

        i. **Request parameter name**: Write **request**

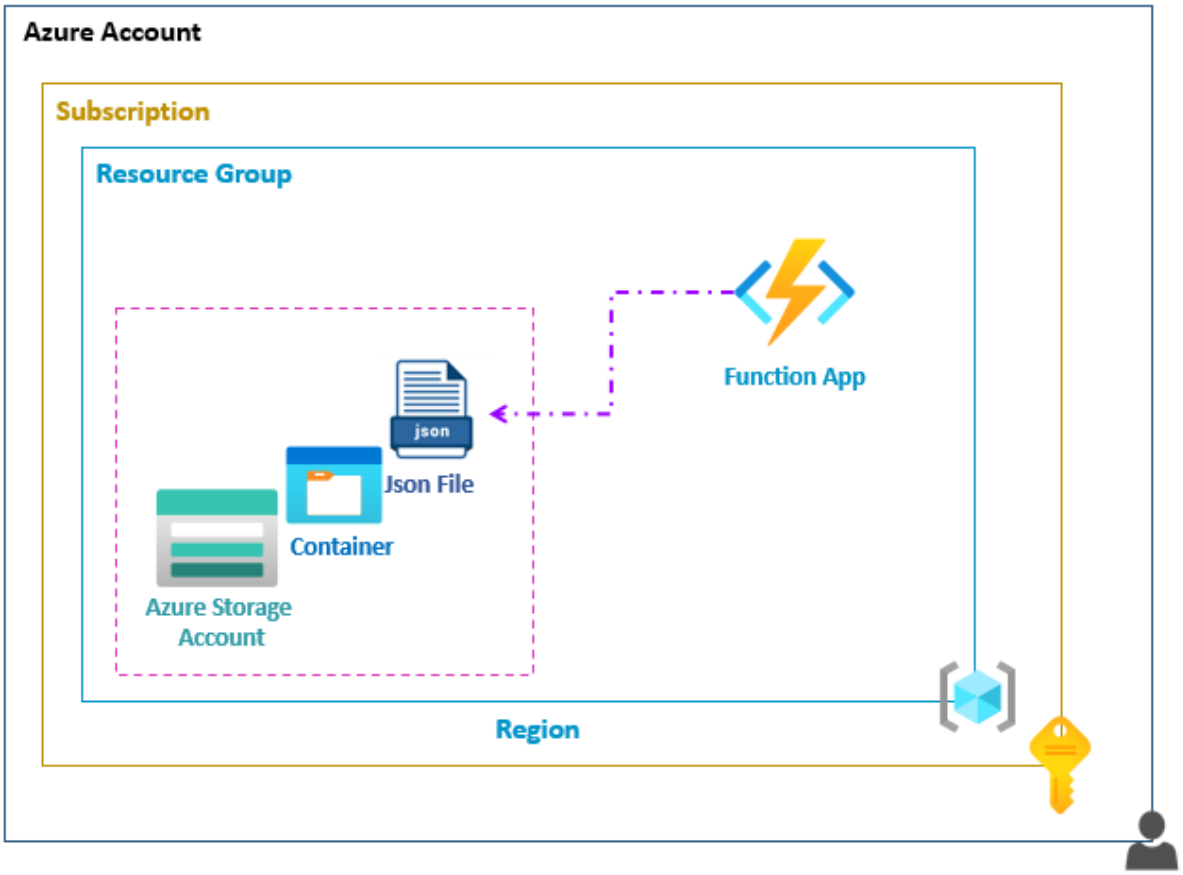
        ii. **Authorisation level**: Dropdown and Select **Anonymous**.

Note: Leave other details as default.



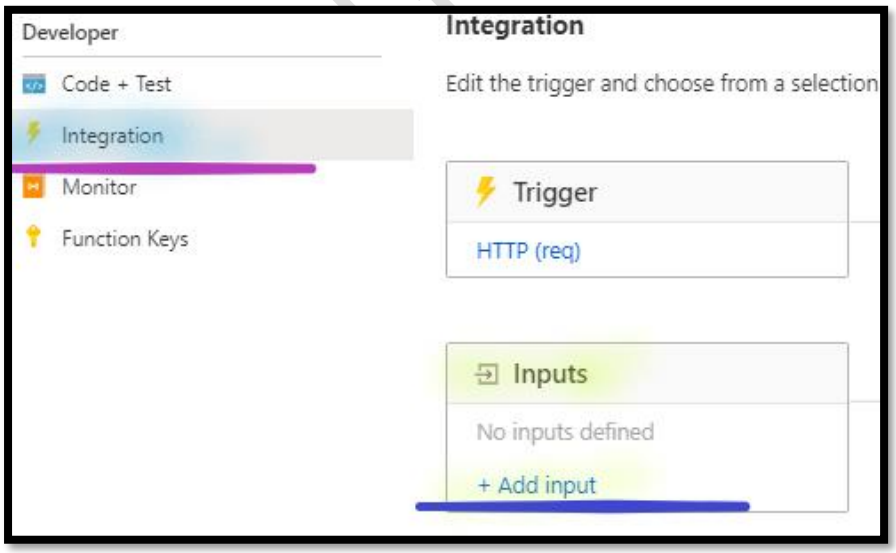
        iii. Select **Save**.

Step 3: Configure an HTTP-triggered function



57. **From** the **Integration** section.

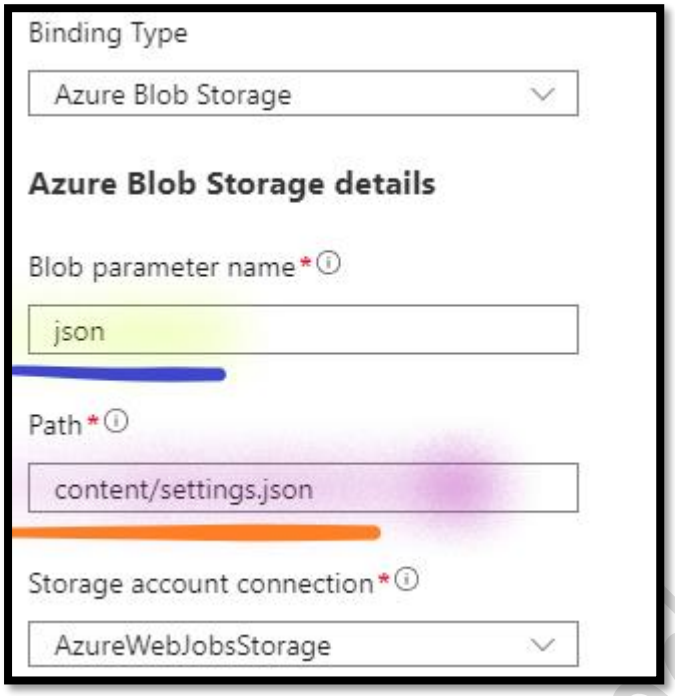
   a. Select the **Add input** under **Input**



   i. **Blob parameter name**: Write **json**

   ii. **Path**: Write **content/settings.json**

Note: Leave other details as default.



iii. Select **Ok**.

Step 4: Update the function code

58. Select the **Code + Test** option from the **Developer** section.

a. In the **Function editor**, **Delete** the ***example code*** in the **run.csx** function script.

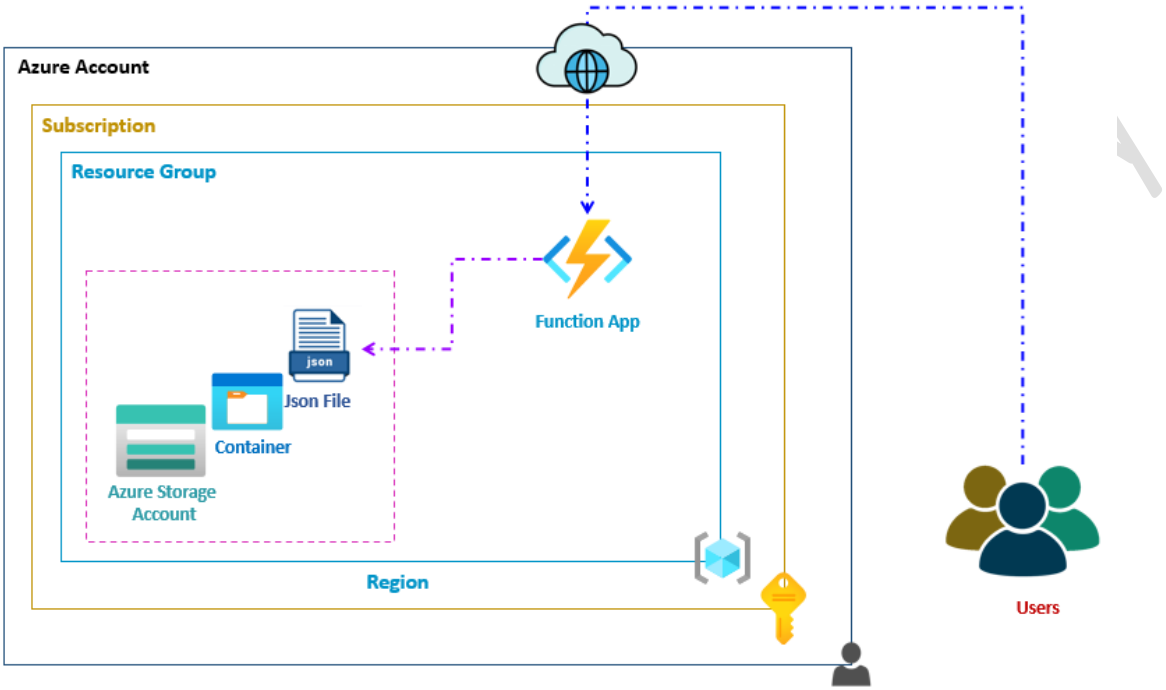
b. **Add** the **following** using directives for libraries that will be referenced by the application:

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

c. Select the **Save**

Note: **Copy** the Function name (like **HttpTrigger2**) in **Notepad**.

Step 4: Test function run by using Httprepl



59. **Return** to the **LAB-204-06-VM** server, right click on **Start** & **Run**

    a. In **Open** write **CMD**

    b. **Start** the **httprepl** tool and set the base Uniform Resource Identifier (URI) to the value of the Request URL for the API operation. Execute the below command:

    httprepl <Function-App -URL>

Note: Replace the **Function-App-URL**, with the Request URL you have copied in the previous step.

    c. At the tool prompt, **browse** to the relative **api/ HttpTrigger2 directory**:

    cd api

    cd HttpTrigger2

Info: Change the **HttpTrigger2** to other name, as you have copied in the previous step.

d. **Run** the **get** command for the current endpoint:

```
get
```

Note: **Observe** the JSON content of the **response** from the **function app**.

Note: In the Response you can see the **settings.json** file content you have uploaded in the container in the previous step.

```
https://funclogic-123.azurewebsites.net/api/getsettinginfo> get
HTTP/1.1 200 OK
Content-Length: 245
Content-Type: text/plain; charset=utf-8
Date: Mon, 20 Jul 2020 19:55:59 GMT
Request-Context: appId=cid-v1:2964f261-319d-4e65-96fa-510d584620f3
Set-Cookie: ARRAffinity=2a72c92982c7768e24d5bed66a92ca45ca4bda9d1295ce244a4243
c-123.azurewebsites.net

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

e. **Exit** the httprepl application:

```
exit
```

Task 6: Delete the Environment

Step 1: Delete the resource groups

60. Delete **Az-204-06-01-RG** resource group