# **Agend****a: Implement Azure Functions**

* About Azure Functions.
* Azure Webjobs vs Azure Functions
* Create and Configure an Azure Function
* Create an event processing function
  + Timer Triggered Function
  + Blog Storage Triggered Function
* Implement an Azure-connected functions
  + Adding message to Storage Queue
  + Connecting to SQL Database
* Creating Azure Functions through Visual Studio

**About Azure Functions**

A picture containing diagram, text, plan, screenshot

Description automatically generated

* Azure Functions is a **serverless compute service** that enables you to run code on-demand without having to explicitly provision or manage infrastructure.
* Use Azure Functions to run a script or piece of code in response to a **variety of events**.
* Functions can make development even more productive, and you can use your development language of choice, such as C#, Java, Node.js, Python or PHP.

The following service integrations are supported by Azure Functions:

|  |  |
| --- | --- |
| Azure Cosmos DBAzure Event HubsAzure Event GridAzure Notification HubsAzure Service Bus (queues and topics)Azure Storage (blob, queues, and tables)On-premises (using Service Bus) |  |

The following are a common, but by no means exhaustive, set of scenarios for Azure Functions.

|  |  |
| --- | --- |
| **If you want to...** | **then...** |
| **Build a web API** | Implement an endpoint for your web applications using the **HTTP trigger** |
| **Process file uploads** | Run code when a file is uploaded or changed in **blob storage.** |
| **Build a serverless workflow** | Chain a series of functions together using **durable functions.** |
| **Respond to database changes** | Run custom logic when a document is created or updated in Cosmos DB |
| **Run scheduled tasks** | Execute code on pre-defined timed intervals (Timer trigger) |
| **Create reliable message queue systems** | Process message queues using Queue Storage, Service Bus, or Event Hubs |
| **Analyze IoT data streams** | Collect and process data from IoT devices |
| **Process data in real time** | Use Functions and SignalR to respond to data in the moment |

## **How much does Functions cost?**

Azure Functions has following kinds of pricing plans:

* **App Service plan**
  + Run your functions just like your web, mobile, and API apps.
  + When you are already using App Service for your other applications, you can run your functions on the same plan at no additional cost.
* **Consumption plan**
  + Billing is based on number of executions, execution time, and memory used. Billing is aggregated across all functions within a function app.
  + When your function runs, Azure provides all of the necessary computational resources.
  + You don't have to worry about resource management, and you only pay for the time that your code runs.
  + Your app scales additional instances of the Functions host when needed to handle load, and scaled down when code stops running.
  + Every execution can use max up to **1.5GB Memory** and **One CPU.**
* **Premium plan**
  + Premium plan provides features, such as premium compute instances(one core, two core, and four core instances), Predictable Pricing, the ability to keep instances warm indefinitely, Unlimited execution duration, and VNet connectivity.
  + In the premium plan, your plan size will determine the available memory and CPU for all apps in that plan on that instance.

**Creating an Azure Function in Portal**

**Create a Function App**

1. Azure Portal 🡪 Search 🡪 Function App
2. App name = <YourName>Demo-funcApp, Hosting Plan: Consumption Plan, Runtime Stack = .NET, Version **= 6**, Operating System=Windows 🡪 Create
3. Storage Tab: Select existing storage if existing or create a new one.
4. Monitoring: Enable Application Insights: Select **Yes**
5. Leave default for other Tabs 🡪 Review + create

A screenshot of a computer

Description automatically generated with medium confidence

**Create a Function**

1. Function App 🡪 Click **Functions 🡪**  +Create

A screenshot of a computer

Description automatically generated

1. Development environment = Develop in portal
   1. Select a template = **HTTP trigger**,
   2. New Function = SayHello,
   3. Authorization level = Function / Anonymous / Admin
2. Create

A screenshot of a computer

Description automatically generated with medium confidence

Authorization level controls whether the function requires an **API key** and which key to use; Function uses a function key; Admin uses your master key. The function and master keys are found in the **'keys'** management panel on the portal, when your function is selected.

1. Switch to **Code + Test** Blade

**Following is the function auto generated in C#**

#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;

    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);

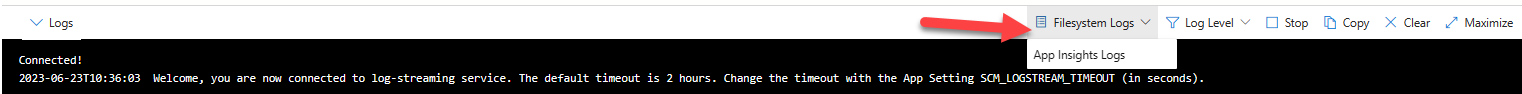
}

**Test the function:**

1. In your new function toolbar, click **Get function URL** and copy the **Function URL.**

https://dssdemofunc.azurewebsites.net/api/HttpTriggerJS1?**code**=HaAojLFnYG05Fa0hMtocj7ymoCExasRWW64BaWtbQGx/O9DvSoDv8A==

1. Paste the URL for the HTTP request into your browser's address bar. Append the query string **&name=<yourname>** to this URL and execute the request.
2. View the function logs at the bottom of the screen
   1. click up arrow at the bottom of the screen
   2. Change from App Insights Logs to **Filesystem Logs**



1. You can make changes (specially to **context.res.body**) to the function as needed and test the same.
2. You can also test the function from Right Handside panel: Test (expand to test)

**Blob Trigger**

1. Function App 🡪 Click **Functions 🡪**  +Create
2. Development environment = Develop in portal
   1. Select a template = **Azure Blob Storage trigger**,
   2. New Function = **ProcessFile**,
   3. Storage account connection = AzureWebJobsStorage
3. 🡪 Create

**Configuring the Function**

1. Click **Integration** 🡪 In Trigger: Click Azure Blob Storage (myBlob)
   1. Blob parameter **name**: myBlob.
   2. **Path**: A path that specifies the container to monitor, and optionally a blob name pattern = **"resumes/{name}"**

**Examples of Path:**

* + 1. **input/original-{name}**: This path would find a blob named ***original-Blob1.txt*** in the ***input*** container, and the value of the name variable (myBlob) in function code would be **Blob1.txt**
    2. **samples/{name}.png**: only .png blobs in the samples container will trigger the function. If **demo.png** is uploaded into the samples container, value of name parameter(myBlob) would be **demo**

1. Code + Test 🡪 Edit function as below

using System.IO;

public static void Run(**String** myBlob, string name, ILogger log)

{

    log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n ----------------- \n Content: {myBlob} Bytes");

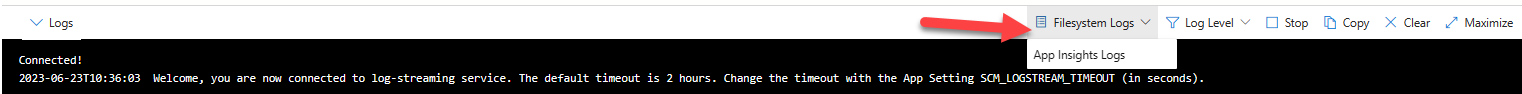
}

1. Function App 🡪 Configuration 🡪 App Settings 🡪 Edit **AzureWebJobsStorage** 🡪 Note the name of Storage Account (Same as the one selected while creating the Azure Function App)

A screenshot of a computer

Description automatically generated with low confidence

1. Function App 🡪 Functions 🡪 Select your Function 🡪 Code + Test 🡪 Expand **Logs** 🡪 Switch to Filesystem Logs



1. In New Tab of Browser 🡪 Azure Portal 🡪 Select Storage Account (dssdemostorage468)🡪
   1. Create a container, Name=**resumes,** Access Level=**Private**
2. Test the function:
   1. Storage account 🡪 C**ontainers** 🡪 **resumes 🡪 Upload a File**
   2. Go back to your function **logs** and verify that the blob has been read.