# Serverless applications

Serverless compute can be thought of as a function as a service (FaaS), or a microservice that is hosted on a cloud platform. Your business logic runs as functions and you don't have to manually provision or scale infrastructure. The cloud provider manages infrastructure. Your app is automatically scaled out or down depending on load. Azure has several ways to build this sort of architecture. The two most common approaches are Azure Logic Apps and Azure Functions

## Technology options

Azure includes four different technologies that you can use to build and implement workflows that integrate multiple systems. There is no requirement for you to use the same technology for different workflows.

### Design-first technologies

which include user interfaces in which you can draw out the workflow. Both technologies allow custom connectors.

* **Logic Apps**: (technical staff) Is a service within Azure that you can use to automate, orchestrate, and integrate disparate components of a distributed application. You can draw out complex workflows that model complex business processes. It provides hundreds of pre-built connectors that you can use to create your apps.
* **Microsoft Power Automate**: (non-technical staff) You can create workflows that integrate and orchestrate many different components by using the website or the Microsoft Power Automate mobile app.

### Code-first technologies

Use this approach when you need more control over the performance of your workflow or need to write custom code as part of the business process

* **WebJobs**: are a part of the Azure App Service that you can use to run a program or script automatically. There are two kinds of WebJob: *Continuous* (run in a loop) and *Triggered* (manually start or schedule). This technology has only two advantages over AZ functins: *Can be part of an App Service application* and *Provides close control of JobHost*
* **Azure Functions**: is a simple way for you to run small pieces of code in the cloud, without having to worry about the infrastructure required to host that code. It has four trigger options: *HTTPTrigger*, *TimerTrigger*, *BlobTrigger* and *CosmosDBTrigger*. **IMPORTANT: it can run on a consumption plan!!!** You can write your function code in the language of your choice, including C#, F#, JavaScript, Python, and PowerShell Core. Support for package managers like NuGet and NPM is also included.

## Function app

Functions are hosted in an execution context called a function app. You define function apps to logically group and structure your functions and a compute resource in Azure.

### Service plan

Function apps may use one of two types of service plans

* **Consumption service plan**: it provides automatic scaling and bills you when your functions are running. It has a configurable timeout as long as 10 minutes (by default has 5 minutes).
* **Azure App Service plan**: It allows to avoid timeout periods by having the function run continuously on a VM, so this is technically not a serverless plan. It may be a better choice if the functions are used continuously or if the functions require more processing power or execution time than the Consumption plan can provide.

### Storage account

The function app uses a storage account for internal operations such as logging function executions and managing execution triggers. On the Consumption service plan, this is also where the function code and configuration file are stored.

### Triggers

You must configure a function with exactly one trigger

* **Blob storage**: Start a function when a new or updated blob is detected.
* **Azure Cosmos DB**: Start a function when inserts and updates are detected.
* **Event Grid**: Start a function when an event is received from Event Grid.
* **HTTP**: Start a function with an HTTP request.
* **Microsoft Graph Events**: Start a function in response to an incoming webhook from the Microsoft Graph. Each instance of this trigger can react to one Microsoft Graph resource type.
* **Queue storage**: Start a function when a new item is received on a queue. The queue message is provided as input to the function.
* **Service Bus**: Start a function in response to messages from a Service Bus queue.
* **Timer**: Start a function on a schedule.

#### Timer trigger

A timer trigger is a trigger that executes a function at a consistent interval. To create a timer trigger, you need to supply two pieces of information.

* A Timestamp parameter name, which is simply an identifier to access the trigger in code.
* A Schedule, which is a CRON expression that sets the interval for the timer.

A CRON expression is a string that consists of six fields that represent a set of times. The order of those fields in Azure is: **{second} {minute} {hour} {day} {month} {day of the week}**.

For example, to execute a function **every 30 seconds, between 10:00 PM and 11:00 PM, from Tuesdays to Wednesday and just in January and March** the CRON must be: **\*/30 \* 22 \* 1,3 2-5**

|  |  |  |
| --- | --- | --- |
| **Special character** | **Meaning** | **Example** |
| \* | Selects every value in a field | An asterisk "\*" in the day of the week field means *every* day. |
| , | Separates items in a list | A comma "1,3" in the day of the week field means just Mondays (day 1) and Wednesdays (day 3). |
| - | Specifies a range | A hyphen "10-12" in the hour field means a range that includes the hours 10, 11, and 12. |
| / | Specifies an increment | A slash "\*/10" in the minutes field means an increment of every 10 minutes. |

#### HTTP trigger

An HTTP trigger is a trigger that executes a function when it receives an HTTP request. Azure Functions provides an Authorization level for its functions which are:

* **Function**: ("key" based) For this you can use a function key (specific to a function) or a host key (apply to all functions inside the function app).
* **Anonymous**: no authentication required.
* **Admin**: ("key" based) For this you must use a host key.

These kinds of functions can receive data from query strings or request body. Also, can returns data back to the caller.

#### Blob trigger

A blob trigger is a trigger that executes a function when a file is uploaded or updated in Azure Blob storage. To create a blob trigger, you create an Azure Storage account and provide a location that the trigger monitors. The **Path** tells the blob trigger where to monitor to see if a blob is uploaded or updated. By default, the **Path** value is ***samples-workitems/{name}***

* **samples-workitems**: The first part represents the blob container that the trigger monitors.
* **{name}**: The name represents a parameter in your Azure function that receives the name of the added file. Also after that you could set the filters, for example {fileName}.png

### Bindings

In Azure Functions, bindings provide a declarative way to connect to data from within your code. They make it easier to integrate with data streams consistently in a function. You can have multiple bindings providing access to different data elements. This is powerful because you can connect to your data sources without having to code specific connection logic.

Each binding has a direction - your code reads data from input bindings and writes data to output bindings. Each function can have zero or more bindings to manage the input and output data processed by the function ([Supported bindings](https://docs.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings#supported-bindings)).

#### Binding properties

Three properties are required in all bindings. You may have to supply additional properties based on the type of binding and storage you are using.

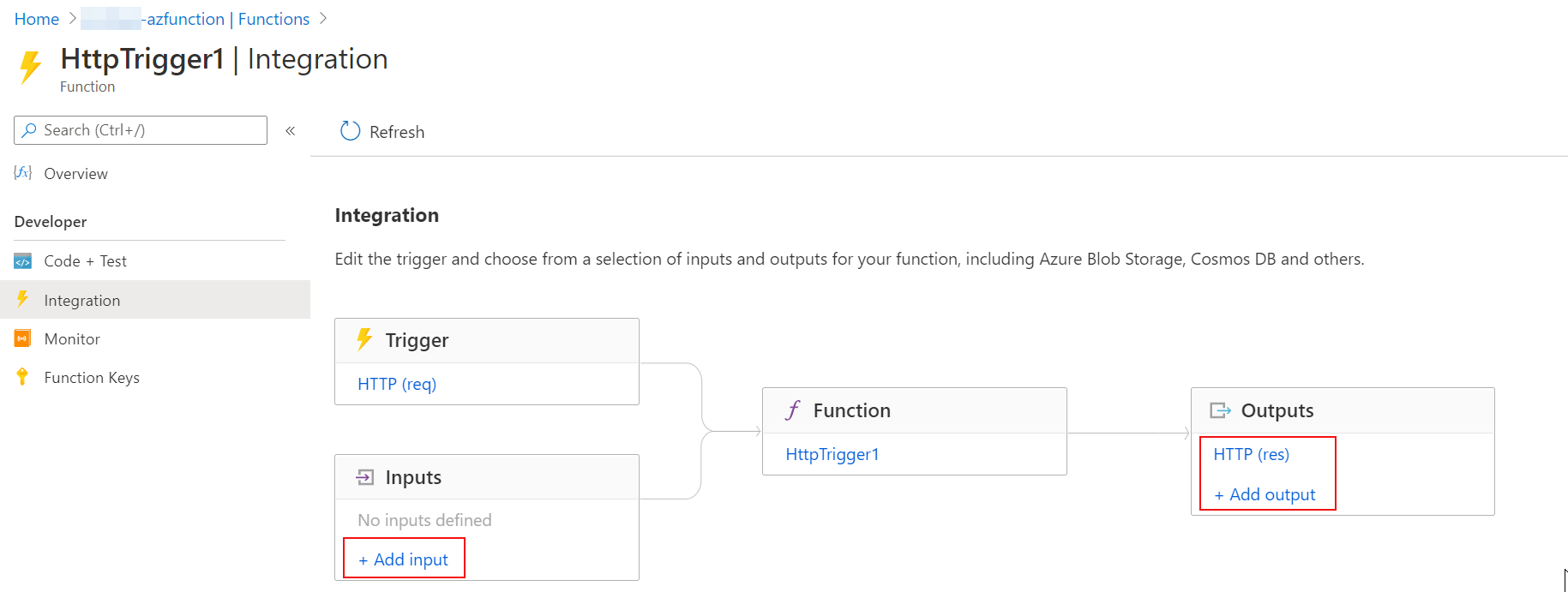
1. **Name** - Defines the function parameter through which you access the data. For example, in a queue input binding, this is the name of the function parameter that receives the queue message content.
2. **Type** - Identifies the type of binding, i.e., the type of data or service we want to interact with.
3. **Direction** - Indicates the direction data is flowing, i.e., is it an input or output binding?

Additionally, most binding types also need a fourth property:

1. **Connection** - Provides the name of an app setting key that contains the connection string. Bindings use connection strings stored in app settings to keep secrets out of the function code. This makes your code more configurable and secure.

#### Explore bindings

You can manage the bindings functions from the Integration section. There you can add/edit inputs and outputs bindings.



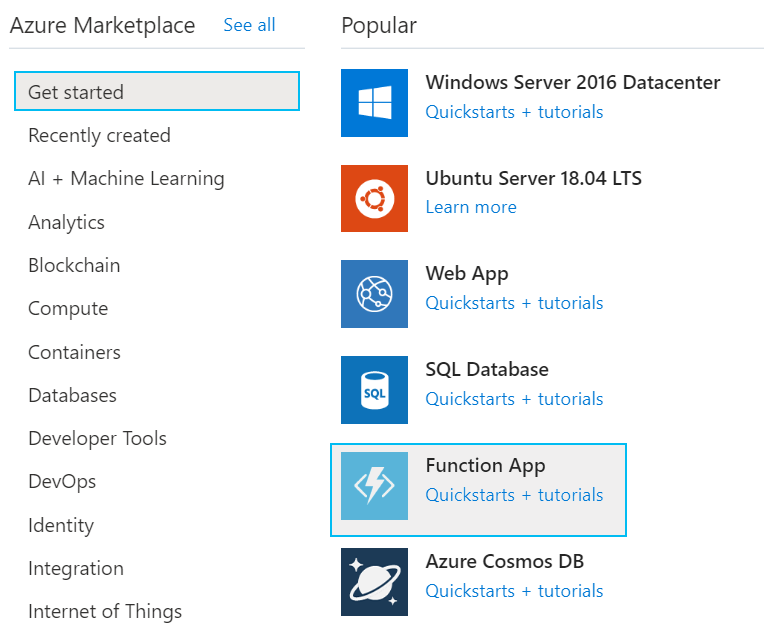
### Log window

You are able to add logging statements to your function for debugging in the Azure portal. The called methods for each language are passed a "logging" object, which may be used to log information to the log window located in a tabbed flyout menu located at the bottom of the code window.

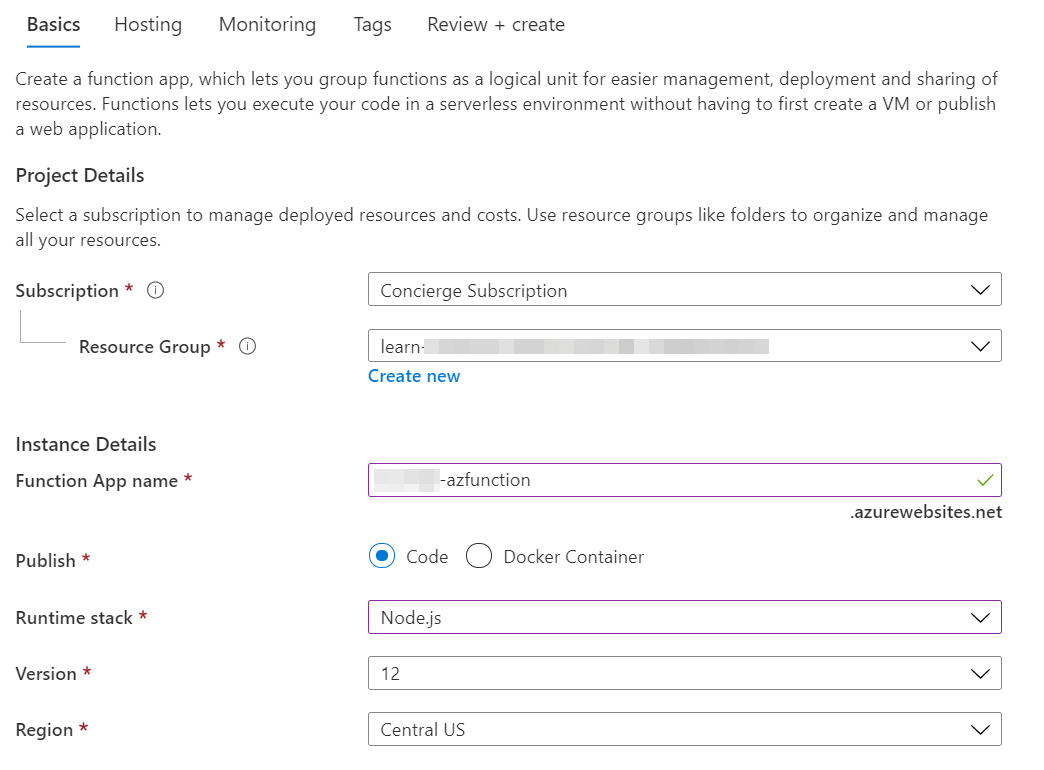
* **JavaScript**: context.log('Enter your logging statement here');
* **C#**: log.Info("Enter your logging statement here");
* **PowerShell**: Write-Host "Enter your logging statement here"

### Create a Function app

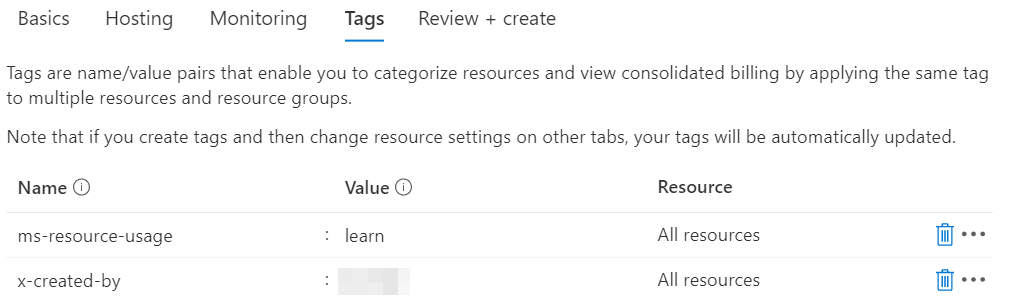
This is a summary to create e Function APP. For it you should go into the Resource-group and then choose Add (at the top-left of the page). After that choose Function App in the resources list.



Complete the Basics tab with a Function App name and the Runtime stack (.NET Core, Node.js, Python, Java or Powershell Core).

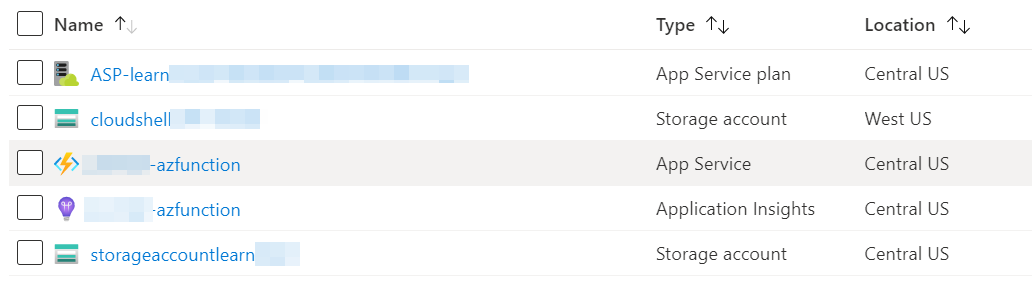


In the Tags section, you could add any tags as you need. This is not mandatory but is very recommended, because this way the resources are easily identifiable.

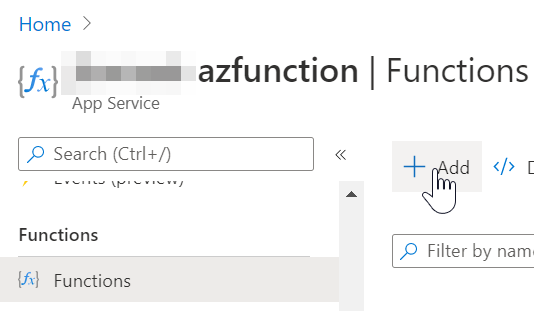


### Create an Azure Function

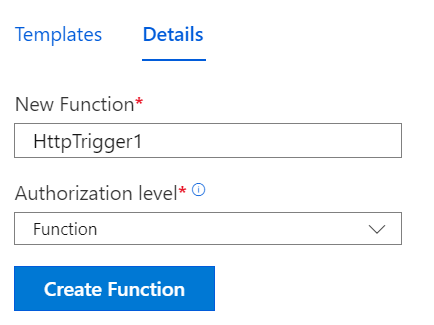
Once the function app is created, go into the App Service resource, and then go to Functions



Once in there, in the top of the page you could see the button to add new Azure Functions



After that you should choose the Trigger (in this case will be HTTP trigger, but there are a lot of options), the name and the Authorization level.



If you select **function** as authorization level, the button **Get Function Url** will provide you an authorization code to call this endpoint

