Azure Functions developers guide

10/12/2017 • 6 minutes to read • Contributors all

In this article

In Azure Functions, specific functions share a few core technical concepts and components, regardless of the language or binding you use. Before you jump into learning details specific to a given language or binding, be sure to read through this overview that applies to all of them.

This article assumes that you've already read the Azure Functions overview and are familiar with WebJobs SDK concepts such as triggers, bindings, and the JobHost runtime. Azure Functions is based on the WebJobs SDK.

Function code

A *function* is the primary concept in Azure Functions. You write code for a function in a language of your choice and save the code and configuration files in the same folder. The configuration is named `function.json`, which contains JSON configuration data. Various languages are supported, and each one has a slightly different experience optimized to work best for that language.

The function.json file defines the function bindings and other configuration settings. The runtime uses this file to determine the events to monitor and how to pass data into and return data from function execution. The following is an example function.json file.

JSON Copy

```
{
    "disabled":false,
    "bindings":[
        // ... bindings here
        {
            "type": "bindingType",
            "direction": "in",
            "name": "myParamName",
            // ... more depending on binding
        }
    ]
}
```

Set the `disabled` property to `true` to prevent the function from being executed.

The `bindings` property is where you configure both triggers and bindings. Each binding shares a few common settings and some settings, which are specific to a particular type of binding. Every binding requires the following settings:

Property	Values/Types	Comments
`type`	string	Binding type. For example, `queueTrigger`.
`direction`	'in', 'out'	Indicates whether the binding is for receiving data into the function or sending data from the function.
`name`	string	The name that is used for the bound data in the function. For C#, this is an argument name; for JavaScript, it's the key in a key/value list.

Function app

A function app is comprised of one or more individual functions that are managed together by Azure App Service. All of the functions in a function app share the same pricing plan, continuous deployment and runtime version. Functions written in multiple languages can all share the same function app. Think of a function app as a way to organize and collectively manage your functions.

Runtime (script host and web host)

The runtime, or script host, is the underlying WebJobs SDK host that listens for events, gathers and sends data, and ultimately runs your code.

To facilitate HTTP triggers, there is also a web host that is designed to sit in front of the script host in production scenarios. Having two hosts helps to isolate the script host from the front end traffic managed by the web host.

Folder Structure

The code for all of the functions in a given function app lives in a root folder that contains a host configuration file and one or more subfolders, each of which contain the code for a separate function, as in the following example:

Copy

```
wwwroot
 | - host.json
 | - mynodefunction
 | | - function.json
 | | - index.js
 | | - node_modules
 | | | - ... packages ...
 | | - package.json
 | - mycsharpfunction
 | | - function.json
 | | - run.csx
```

The *host.json* file contains some runtime-specific configuration and sits in the root folder of the function app. For information on settings that are available, see host.json reference.

Each function has a folder that contains one or more code files, the function.json configuration and other dependencies.

When setting-up a project for deploying functions to a function app in Azure App Service, you can treat this folder structure as your site code. You can use existing tools like continuous integration and deployment, or custom deployment scripts for doing deploy time package installation or code transpilation.

Note

Make sure to deploy your `host.json` file and function folders directly to the `wwwroot` folder. Do not include the `wwwroot` folder in your deployments. Otherwise, you end up with `wwwroot\wwwroot` folders.

How to update function app files

The function editor built into the Azure portal lets you update the *function.json* file and the code file for a function. To upload or update other files such as *package.json* or *project.json* or dependencies, you have to use other deployment methods.

Function apps are built on App Service, so all the deployment options available to standard web apps are also available for function apps. Here are some methods you can use to upload or update function app files.

To use App Service Editor

1. In the Azure Functions portal, click **Function app settings**. 3
2. In the **Advanced Settings** section, click **Go to App Service Settings**.
3. Click **App Service Editor** in App Menu Nav under **DEVELOPMENT TOOLS**.
4. click **Go**.

   After App Service Editor loads, you'll see the *host.json* file and function folders under *wwwroot*.
5. Open files to edit them, or drag and drop from your development machine to upload files.

To use the function app's SCM (Kudu) endpoint

1. Navigate to: `https://<function_app_name>.scm.azurewebsites.net`.
2. Click **Debug Console > CMD**.
3. Navigate to `D:\home\site\wwwroot\` to update *host.json* or `D:\home\site\wwwroot\<function_name>` to update a function's files.
4. Drag-and-drop a file you want to upload into the appropriate folder in the file grid. There are two areas in the file grid where you can drop a file. For *.zip* files, a box appears with the label "Drag here to upload and unzip." For other file types, drop in the file grid but outside the "unzip" box.

To use continuous deployment

Follow the instructions in the topic Continuous deployment for Azure Functions.

Parallel execution

When multiple triggering events occur faster than a single-threaded function runtime can process them, the runtime may invoke the function multiple times in parallel. If a function app is using the Consumption hosting plan, the function app could scale out automatically. Each instance of the function app, whether the app runs on the Consumption hosting plan or a regular App Service hosting plan, might process concurrent function invocations in parallel using multiple threads. The maximum number of concurrent function invocations in each function app instance varies based on the type of trigger being used as well as the resources used by other functions within the function app.

Functions runtime versioning

You can configure the version of the Functions runtime using the `FUNCTIONS_EXTENSION_VERSION` app setting. For example, the value "~1" indicates that your Function App will use 1 as its major version. Function Apps are upgraded to each new minor version as they are released. For more information, including how to view the exact version of your function app, see How to target Azure Functions runtime versions.

Repositories

The code for Azure Functions is open source and stored in GitHub repositories:

- Azure Functions runtime
- Azure Functions portal
- Azure Functions templates
- Azure WebJobs SDK
- Azure WebJobs SDK Extensions

Bindings

Here is a table of all supported bindings.

Type	Service	Trigger*	Input	Output
Schedule	Azure Functions	✓		
HTTP (REST or webhook)	Azure Functions	✓		✓**
Blob Storage	Azure Storage	✓	✓	✓
Events	Azure Event Hubs	✓		✓
Queues	Azure Storage	✓		✓
Queues and topics	Azure Service Bus	✓		✓
Storage tables	Azure Storage		✓	✓
SQL tables	Azure Mobile Apps		✓	✓
NoSQL DB	Azure Cosmos DB	✓	✓	✓
Push Notifications	Azure Notification Hubs			✓

Type	Service	Trigger*	Input	Output
Twilio SMS Text	Twilio			✓
SendGrid email	SendGrid			✓
Excel tables	Microsoft Graph		✓	✓
OneDrive files	Microsoft Graph		✓	✓
Outlook email	Microsoft Graph			✓
Microsoft Graph events	Microsoft Graph	✓	✓	✓
Auth tokens	Microsoft Graph		✓	

(* - All triggers have associated input data)

(** - The HTTP output binding requires an HTTP trigger)

Reporting Issues

Item	Description	Link
Runtime	Script Host, Triggers & Bindings, Language Support	File an Issue
Templates	Code Issues with Creation Template	File an Issue
Portal	User Interface or Experience Issue	File an Issue

Next steps

For more information, see the following resources:

- Best Practices for Azure Functions 1
- Azure Functions C# developer reference
- Azure Functions F# developer reference
- Azure Functions NodeJS developer reference
- Azure Functions triggers and bindings
- Azure Functions: The Journey on the Azure App Service team blog. A history of how Azure Functions was developed.