PowerShell & Azure Functions - Part 1

27 Nov 2019

I have been using Azure Functions for a little while and I published my initial thoughts on Azure Functions and their use cases here.

In this post I want to cover how to get started with Functions and a few lessons learned and I wanted to collect together the documentation I used to get up and running which at present lives in various places on the Microsoft Docs site.

What are Azure Functions

Simply put they are a serverless solution in Azure which allow you to run code without having to worry about where it is running or maintaining the underlying infrastructure.

You can read a whole lot more about Azure Functions in the documentation and it can explain a lot better than I can.

Writing Your First Function

I will go over how to write an Azure Function using Visual Studio Code as that is by the best development experience

Pre-Requisites

You can find the pre-requisites and installation instructions in the Microsoft Docs.

- PowerShell 6+
- Visual Studio Code (obviously)
- Node.js which includes npm
- Azure Functions Core Tools
- .NET Core SDK - *I have this installed but you can avoid that with the extension bundles*

You can also use the Remote Development Extension and containers to run and debug your functions but there is no official docker image for PowerShell available. I have however created a post to show you how to build your own.

Developing PowerShell Azure Functions In Docker And VsCode

Create A Local Function Project

Create a folder where your Function will be contained

```
New-Item -ItemType Directory -Name MyFirstFunction

cd MyFirstFunction
```

Now that you're in your project folder initialise your function. Select PowerShell from the list and it will

```
PS /root/MyFirstFunction> func init

Select a worker runtime:
dotnet
node
python
powershell

Choose option: 4
powershell

Writing profile.ps1
Writing requirements.psd1
Writing .gitignore
Writing host.json
Writing local.settings.json
Writing /root/MyFirstFunction/.vscode/extensions.json
```

You then need to create your first PowerShell function within your initialised folder and select the trigger type.

```
PS /root/MyFirstFunction> func new

Select a template:
1. Azure Blob Storage trigger
2. Azure Cosmos DB trigger
3. Azure Event Grid trigger
4. Azure Event Hub trigger
5. HTTP trigger
6. IoT Hub (Event Hub)
7. Azure Queue Storage trigger
8. SendGrid
9. Azure Service Bus Queue trigger
10. Azure Service Bus Topic trigger
11. Timer trigger

Choose option: 5
HTTP trigger

Function name: [HttpTrigger] HttpTriggerFunction
Writing /root/MyFirstFunction/MyFirstFunction/run.ps1
Writing /root/MyFirstFunction/MyFirstFunction/function.json
The function "MyFirstFunction" was created successfully from the "HTTP trigger" template.
```

Your folder structure will now have a sub-folder with the name you gave to your function.

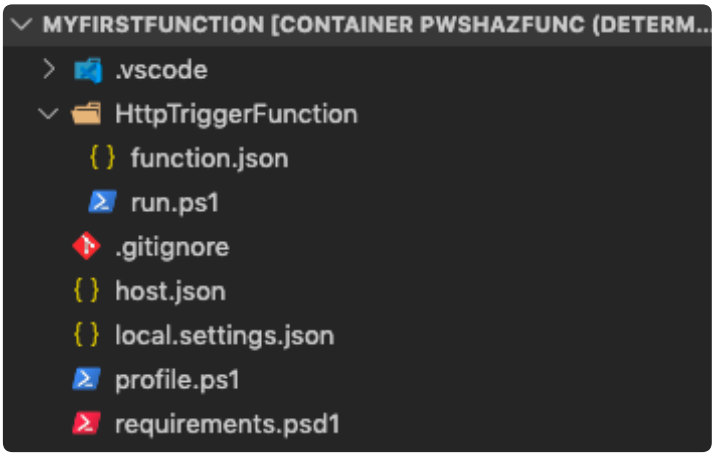
```
Directory: /root/MyFirstFunction

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          11/26/19  4:26 PM                HttpTriggerFunction
------          11/26/19  4:26 PM             72 host.json
------          11/26/19  4:26 PM            142 local.settings.json
------          11/26/19  4:26 PM            882 profile.ps1
------          11/26/19  4:26 PM            273 requirements.psd1
```

Your first PowerShell Azure Function is ready to play with 😁

```
# Open the current folder in VSCode
code .
```

Function Layout Overview



Scaffolding

The majority of the contents of the folder are used to assist with deploying and configuring of your Function.

- .gitignore
  - This is a templated gitignore file which contains the common files which you don't want to commit to git. Things like `local.settings.json` which is used for storing secrets for local debugging.
- host.json
  - This is for setting global configuration settings for your functions. There are lots of options to configure there but not much you have to worry about at first. Host.json reference documentation is here and covers all of the possible options.
- local.settings.json
  - When running and debugging your function locally, app settings are read from the `local.settings.json` file. The possible settings for local development are covered in this article.
- profile.ps1
  - this is only required if your function is going to require authentication for interaction with other Azure resources within your tenant. The comments in the file are self-explanatory.
- requirements.psd1
  - This is specified to manage your dependencies and the host will automatically download dependencies from the PowerShell Gallery for you so you don't have to ship modules with your code. It is recommended to pin to a specific major version of a module to prevent breaking changes. Dependency Management Docs

Your Function - The Fun Stuff

Within your `HttpTriggerFunction` folder you have your actual code file and information about bindings for this particular function.

- function.json
  - This is where your binding information and parameters are defined. You may need to modify these if you're using something like Azure Service Bus Queue trigger. Triggers & Bindings Docs
- run.ps1
  - This is the powershell code that makes up your function and where you will spend most of your time.
  - It consists of a param block where content is passed into your function, the main body which interacts with the content, and an output which utilises an Azure Functions specific cmdlet called `Push-OutputBinding` for passing content back out of your function. Push-Outputbinding Docs

```powershell
# Input bindings are passed in via param block.
param(
    $Request,
    $TriggerMetadata
)

# Write to the Azure Functions log stream.
Write-Host "PowerShell HTTP trigger function processed a request."

# Interact with query parameters or the body of the request.
$name = $Request.Query.Name
if (-not $name) {
    $name = $Request.Body.Name
}

if ($name) {
    $status = [HttpStatusCode]::OK
    $body = "Hello $name"
}
else {
    $status = [HttpStatusCode]::BadRequest
    $body = "Please pass a name on the query string or in the request body."
}

# Associate values to output bindings by calling 'Push-OutputBinding'.
Push-OutputBinding -Name Response -Value ([HttpResponseContext]@{
    StatusCode = $status
    Body = $body
})
```

Running & Debugging Your Function

One of the best things about developing Azure Functions in Visual Studio Code is the fact that all of the tools that you need to write, test, debug and deploy your Azure Function is available to you out of the box.

Running Your Function Locally

You can check that your function works as intended locally with the Function Core Tools. The local runtime is the same runtime that hosts your function app in Azure so it saves a tonne of time having to redeploy to test minor changes.

To run your function locally navigate to your function project folder and start the runtime.

```
PS /root/MyFirstFunction> func start

                  %%%%%%
                 %%%%%%
            @   %%%%%%    @
          @@   %%%%%%      @@
       @@@    %%%%%%%%%%    @@@
     @@      %%%%%%%%%%        @@
       @@         %%%%       @@
         @@      %%%       @@
           @@    %%      @@
                 %%
                 %

Azure Functions Core Tools (2.7.1846 Commit hash: 458c671341fda1c52bd46e1aa8943cb26e467830)
Function Runtime Version: 2.0.12858.0
Skipping 'FUNCTIONS_WORKER_RUNTIME' from local settings as it's already defined in current envir

Hosting environment: Production
Content root path: /root/MyFirstFunction
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.

Http Functions:

        HttpTriggerFunction: [GET,POST] http://localhost:7071/api/HttpTriggerFunction
```

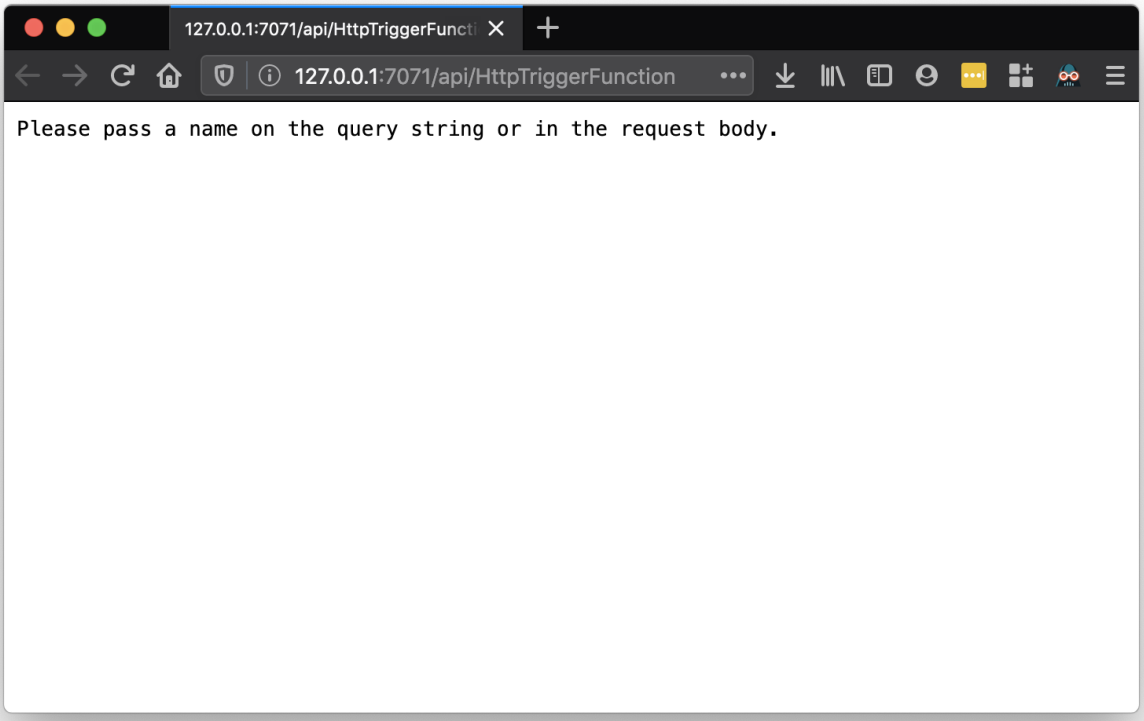
Alongside from pretty trick ASCII art you get some information about your Function Runtime, the address and port the host is listening on, the methods your function supports and the URI that you can use to trigger the function.

Any logs from the function are then output to the console so if you need to see what is happening within your code you can use the age old tactic of littering it with output.

```
[11/27/19 2:24:11 PM] INFORMATION: PowerShell HTTP trigger function processed a request.
```

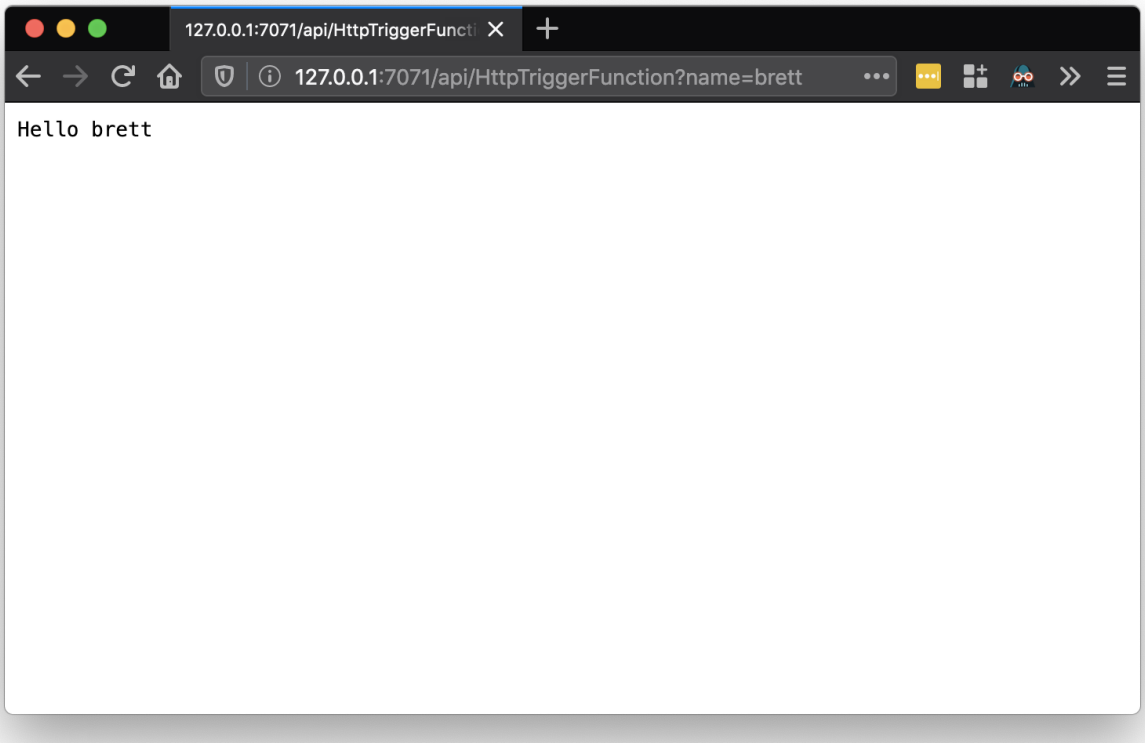
Triggering your Function

You can open the URI in your browser which will trigger the function and you should see the output below.



You can see from the output that this is the `else` block of the code telling you to provide a query parameter to your function.

If you change the URI to `http://127.0.0.1:7071/api/HttpTriggerFunction?name=brett` you will get the following output.



Testing this in the browser is fine when your Azure Function supports GET method but in order to test a POST method then you will need to use PowerShell or Postman so you can pass a JSON payload.

Azure Functions Methods

So if you've used PowerShell to interact with API's then you'll likely be aware of the HTTP Methods that are used for RESTful services. It's the same principle for Azure Functions and you should take this into account when designing your Functions as the way you handle your input parameters will differ.

When your function is triggered with the GET method then the `Query` property of your `$Request` payload will be populated but if the POST method is used then the `Body` property will be used.

The methods are defined in the `function.json` and because the example project has both GET and POST then it checks for each of the properties accordingly

```
{
  "bindings": [
    {
      "authLevel": "function",
      "type": "httpTrigger",
      "direction": "in",
      "name": "Request",
      "methods": [
        "get",
        "post"
      ]
    }
  ]
}
```

```
# Check to see if the payload has a query value - GET method
$name = $Request.Query.Name

# If query is not populated then check the Body value - POST method
if (-not $name) {
    $name = $Request.Body.Name
}
```

Summary

Working with PowerShell in Azure Functions has been an awesome experience. Being able to develop and run them locally without having to constantly push my changes to Azure made the learning process so easy.

The way you work with Functions is very familiar so you can bring what you already know about writing a PowerShell Function and with very little modification, start working with your cloud workloads in a much leaner way. Something that would previously have been a scheduled script in Azure Automation can now be reactive and if you're on a consumption plan then it will cost nothing at all.

One of the most difficult parts of getting to grips with Azure Functions in general is the sheer amount of documentation that resides on slightly different pages dependent on what information you're looking for. It was very easy to overlook parts of the documentation or quite frequently had to piece together parts from the examples in other programming languages as PowerShell documentation within Azure Functions is lacking a little.

What's Next?

I have another follow up post in the making which will cover local debugging, deploying your function and some more in depth parts of PowerShell in Azure Functions as well as some of the edge cases I encountered.

References & Links

- An introduction to Azure Functions
- Work with Azure Functions Core Tools
- Host.json reference documentation
- Local settings file
- Dependency Management
- Azure Functions Bindings
- Writing Output Data

PowerShell Azure Functions Serverless

Comments

Load Comments

Related Posts

Developing PowerShell Azure Functions Locally in a Container 27 Nov 2019

Azure Functions - Event Driven, Serverless Functions 26 Nov 2019

Tab Completion for Your Azure Subscriptions 11 Sep 2019