

PowerShell in Azure Functions

This post is part of the series 'azureapi':

1. [Build a Serverless API in Azure](#)
2. PowerShell in Azure Functions
3. [GitHub Integration with Azure Functions](#)
4. [Add an API spec in Azure Functions](#)
5. [Azure Functions and Azure API Management](#)
6. [Serverless API Series - Conclusion](#)

Just hopping on the [serverless](#) train in 2018? Join the club! Lately, I've been a little obsessed with [Azure Functions](#) and the doors it opens.

However, PowerShell in Azure Functions is still a bit unintuitive when you're getting started. PowerShell is currently available only as an unsupported "experimental" language, and you should be aware of the expectations Azure Functions casts onto your code.

This post will show you:

- How to create your first Azure Function in PowerShell
- What the default code is actually doing
- What to apply from the example to your code
- Finished sample code to return JSON objects

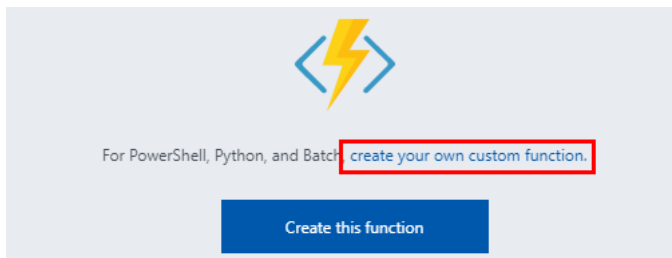


Table of Contents

- [Start with the official docs](#)
 - [1\) Log in to Azure](#)
 - [2\) Create a function app](#)

- [3\) Favorite Functions in the portal](#)
 - [4\) Create an HTTP triggered function](#)
 - [5\) Test the function](#)
 - [Explaining the default PowerShell function code](#)
 - [POST and GET](#)
 - [Where are these variables coming from?](#)
 - [Query parameters](#)
 - [A sample script](#)
 - ["Best Practices" from Microsoft](#)
 - [Next steps \(and diverging paths\)](#)
-

Start with the official docs

The [official walkthrough](#) is useful. I'll mirror the steps here, because we have to deviate for PowerShell halfway through. (Hopefully the doc doesn't change tomorrow...)

1) Log in to Azure

You just need a free, personal Azure account to get started.

Azure Functions is [mostly free](#) for personal use...you won't exceed the free tier execution/bandwidth limits without trying, so it's just a question of whether you need to store data for your use case.

The walkthrough in this post isn't storing anything in Azure.

2) Create a function app

Function App

Create

*

App name

bbpowershellapi

.azurewebsites.net

*

Subscription

Pay-As-You-Go (cfb2a

*

Resource Group

Create new

Use existing

rgbbpowershellapi

*

OS

Windows

Linux (Preview)

*

Hosting Plan

Consumption Plan

*

Location

West US 2

*

Storage

Create new

Use existing

sabbpowershellapi

Application Insights

On

Off

☒

Pin to dashboard

Create

Automation options

Your "app name" becomes a public subdomain: APPNAME.azurewebsites.net

3) Favorite Functions in the portal

Uh, sure. Ok, on to the good stuff.

4) Create an HTTP triggered function

Follow these screenshots to create a not-yet-supported PowerShell function, instead of the default C# walkthrough.



Get started quickly with a premade function

1. Choose a scenario

Webhook + API

Timer

Data processing

2. Choose a language

☒ CSharp ☐ JavaScript ☐ FSharp ☐ Java

For PowerShell, Python, and Batch [create your own custom function.](#)

Create this function

bbpowershellapi
Function Apps

bbpowershellapi

Pay-As-You-Go

Function Apps

bbpowershellapi

Functions

Proxies

Slots (preview)

Choose a template below or [go to the quickstart](#)

Search by trigger, language, or desc

Language: All

Scenario: All

Experimental Language Support: ☒ Enabled

HTTP trigger


A function that will be run whenever it receives an HTTP request, responding based on data in the body or query string

Batch C# F# JavaScript PowerShell Python TypeScript

Timer trigger

A function that will be run on a specified schedule

C# F# JavaScript PowerShell TypeScript

 HTTP trigger

New Function

Language:

PowerShell

Name:

ScriptingGuysGet

HTTP trigger

Authorization level ⓘ

Function

This language is experimental and does not yet have full support. If you run into issues, please file a bug on our [GitHub repository](#).

Create Cancel

Spoiler alert on the function name: Yes, this means you will `/ScriptingGuysGet -Method Get`, which is silly. I thought I could mask this at the end of the series, but instead I'll briefly cover why I didn't. Name retained as my mark of shame and your warning.

5) Test the function

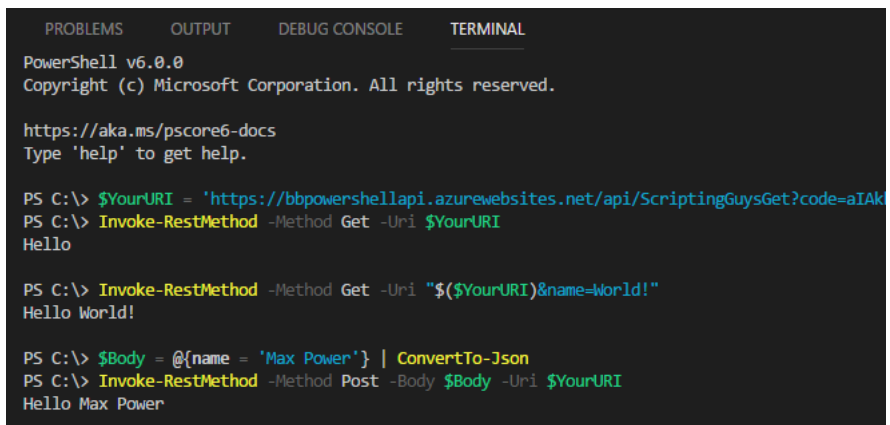
You have a working PowerShell Azure Function! (We'll look at the code in a sec.) First, use PowerShell on your local workstation to test the new "HTTP trigger":



```
# Replace this with yours ;)
$YourURI = 'https://bbpowershellapi.azurewebsites.net/api/ScriptingGuysGet?code=blahblahKEYblahblah'
```

```
# Try a normal GET
Invoke-RestMethod -Method Get -Uri $YourURI
# GET using the "name" query parameter
Invoke-RestMethod -Method Get -Uri "$($YourURI)&name=World!"

# Use the POST method provided
$Body = @{name = 'Max Power'} | ConvertTo-Json
Invoke-RestMethod -Method Post -Body $Body -Uri $YourURI
```



The screenshot shows a PowerShell terminal window with the following content:

```
PowerShell v6.0.0
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/pscore6-docs
Type 'help' to get help.

PS C:\> $YourURI = 'https://bbpowershellapi.azurewebsites.net/api/ScriptingGuysGet?code=aIAk
PS C:\> Invoke-RestMethod -Method Get -Uri $YourURI
Hello

PS C:\> Invoke-RestMethod -Method Get -Uri "$($YourURI)&name=World!"
Hello World!

PS C:\> $Body = @{name = 'Max Power'} | ConvertTo-Json
PS C:\> Invoke-RestMethod -Method Post -Body $Body -Uri $YourURI
Hello Max Power
```

Explaining the default PowerShell function code

Here's the default code when you create a PowerShell function:

```
# POST method: $req
$requestBody = Get-Content $req -Raw | ConvertFrom-Json
$name = $requestBody.name

# GET method: each querystring parameter is its own variable
if ($req_query_name)
{
    $name = $req_query_name
}

Out-File -Encoding Ascii -FilePath $res -inputObject "Hello $name"
```

That code in English:

1. Accepts an optional `$name`
 - via POST
 - or via GET
2. Outputs the string `Hello $name`

If you wanted to always output the same string, your Azure Function could be one line:

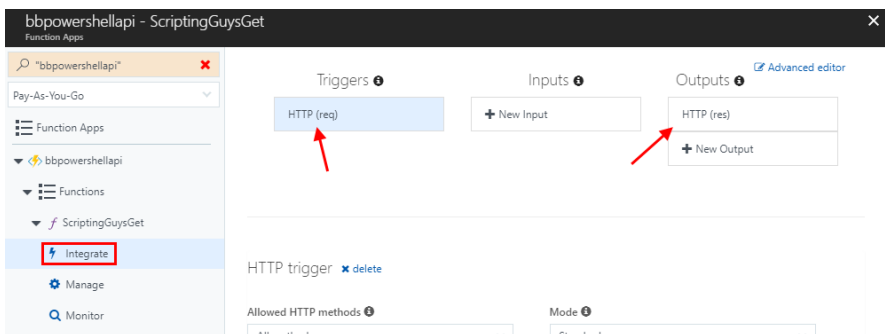
```
'Hello World!' | Out-File -Encoding Ascii -FilePath $res
```

POST and GET

As you saw in our test section, POST and GET are two standard [REST API verbs](#). This default script supports providing a name via both operations. (Which is weird. But that's ok; it's still a good, concise example.)

Where are these variables coming from?

`$req` and `$res` are Azure Functions-specific automatic variables. They're just file paths, local to the currently executing function. If you start digging around, you'll see them referenced here:



- **\$req:** The body of a POST request ends up in `$req`. This example code gets the `$req` file's contents, converts from JSON into an object, and then extracts the `name` property to set `$name`.
- **\$res:** Azure Functions returns the contents of `$res`, and nothing else. If you see output in your log, but don't get a return in your local console, you may have forgotten to `Out-File $res`.

If you're developing a suite of Azure Functions, it's a good idea to have a Pester test file ensure you remember to `Out-File $res` in each individual function.

For more info on the Azure Functions runtime environment, see Stefan Stranger's [TechNet post](#).

Query parameters

The GET method uses `$req_query_name`. When we did our testing, we used the GET method with `name` as a [query parameter](#).

```
Invoke-RestMethod -Method Get -Uri "$($YourURI)&name=World!"
```

Query parameters are used *everywhere*. `?` denotes the first parameter, and `&` allows you to append additional parameters.

As an example, <https://www.google.com/search?q=test&hl=en> hits the /search endpoint, then sets your query = "test" and language = "en" (English).

In Azure Functions, variables are automatically created at runtime for each query parameter supplied. What does that mean?

The user can do this

```
Invoke-RestMethod -Method Get -Uri "$($YourURI)?food=yes&drink=yes&quantity=yes"
```

Whether your Azure Function uses none, one, or all of the following variables:

`$req_query_food`

`$req_query_drink`

`$req_query_quantity`

In my limited testing, `$req_query_*` variables seem to be read-only once inside your PowerShell Azure Function.

A sample script

The code below:

1. Optionally accepts `tag` as a query parameter
2. Queries the [Scripting Guys blog](#)'s RSS feed for either:
 - The ten most recent posts, or

- The ten most recent posts containing the given `tag`

3. Returns each post's Date/Title/Description/Link in a JSON collection

What can you learn from this example?

- Param blocks are (apparently?) useless in Azure Functions
 - But it's nice to still notate any expected parameters, somehow
- Reaches outside of Azure, querying content outside of your control
- Proper output of a collection of objects in JSON format
- I can't get Verbose to work
 - So potential debugging is currently a frustrating endeavor

```

# Supported query parameters:
# tag

# Create an empty list to append results into
$ResultList = New-Object System.Collections.Generic.List[object]

# Set the URI, with or without a user-supplied tag
$BaseURI = 'https://blogs.technet.microsoft.com/heyscriptingguy'
If ($req_query_tag) {
    $BaseURI = "$BaseURI/tag/$req_query_tag"
}

Try {
    # Get the latest 10 posts (RSS feed default is 10 per page)
    $iwr = Invoke-WebRequest -Uri "$BaseURI/feed" -UseBasicParsing
} Catch {
    # Invoke-WebRequest is weird. Just silently fail
}

If ($iwr) {
    # Cast the RSS feed as XML
    [xml]$xml = $iwr.Content

    ForEach ($post in $xml.rss.channel.item) {
        # Assemble the most useful properties in an object
        $newObject = [PSCustomObject]@{
            Date       = $post.pubDate -as [DateTime]
            Title       = $post.title
            Description = $post.description.'#cdata-section'
            Link        = $post.link
        }

        # Append the object into the collection
        [void]$ResultList.Add($newObject)
    }

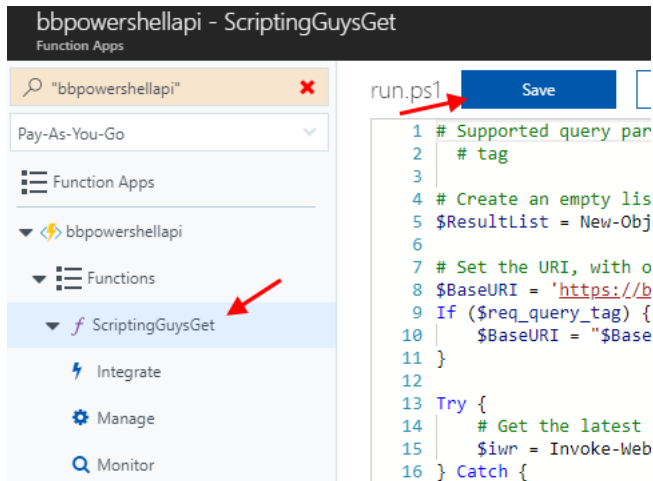
    # Return the objects in JSON format. Azure Functions likes Out-String
    $return = $ResultList | ConvertTo-Json | Out-String

    # By default, Azure Functions wants to output the contents of $res
    Out-File -Encoding Ascii -FilePath $res -inputObject $return
} Else {
    # Can't get Azure Functions to respect -Verbose, even trying this:
    # https://justingrote.github.io/2017/12/25/Powershell-Azure-Function
    # s-The-Missing-Manual.html#logs-panel-and-verbose-debug-output
    # help, haha
}

```

```
}  
    Write-Verbose 'Invoke-WebRequest returned no results' 4>&1  
}
```

Drop that in your “ScriptingGuysGet” function and Save.



Then, calling it from your local workstation again, just as we did before:

```
PS C:\> Invoke-RestMethod -Method Get -Uri "$YourURI&tag=registry"
```

Date	Title	Description
8/21/2015 11:59:00 AM	PowerTip: Use PowerShell Tab Expansion to Navigate Registry	Summary: Use the win
8/21/2015 12:01:00 AM	New PowerShell 5 Feature Provides Better Registry Support	Summary: Ed Wilson, M
4/24/2015 12:01:00 AM	Use PowerShell to Move User Files and Update Registry	Summary: Microsoft S
4/2/2015 11:59:00 AM	PowerTip: Use PowerShell to Read Registry Key Property Value	Summary: Use Windows
4/2/2015 12:01:00 AM	Update or Add Registry Key Value with PowerShell	Summary: Microsoft S
2/6/2015 11:59:00 AM	PowerTip: Use PowerShell to Display Registry Keys	Summary: Learn how to
2/6/2015 12:01:00 AM	Use PowerShell DSC to Configure the Registry	Summary: Microsoft S
2/5/2015 12:01:00 AM	Registry Cmdlets: Complete the Registry CDXML Module	Summary: Richard Side
2/4/2015 12:01:00 AM	Registry Cmdlets: Advanced CDXML	Summary: Richard Side
2/3/2015 11:59:00 AM	PowerTip: Use PowerShell to Find Parameters for CIM Class Method	Summary: Learn how to

Note that `Invoke-RestMethod` automatically *deserializes* JSON; that is, it has implicitly converted the JSON response back into the standard PowerShell objects we know and love.

“Best Practices” from Microsoft

If you're still here, hey! I assume you're pretty interested in writing your own function. You should know Microsoft has some general recommendations on being smart and responsible with your Azure Functions.

<https://docs.microsoft.com/en-us/azure/azure-functions/functions-best-practices>

It includes things like “You can pass the HTTP trigger payload into a queue to be processed by a queue trigger function,” which seems like something I would do here, had I bothered to spend 10 minutes looking into it.

I have not. (Yet!)

Next steps (and diverging paths)

We have a working PowerShell Azure Function! With this, we’ve completed our first step toward building a Serverless REST API in Azure.

If building a full API doesn’t sound interesting or fun...congratulations, you’re normal! This post still stands on its own, and there’s a lot of awesome directions you can take your first PowerShell-backed Azure Function that don’t need a full, formal API front end.

For the abnormal among us, I’m really enjoying building my first formal REST API. Step two is setting up continuous deployment to our new Azure Function App from GitHub...I hope you’ll stick around!
