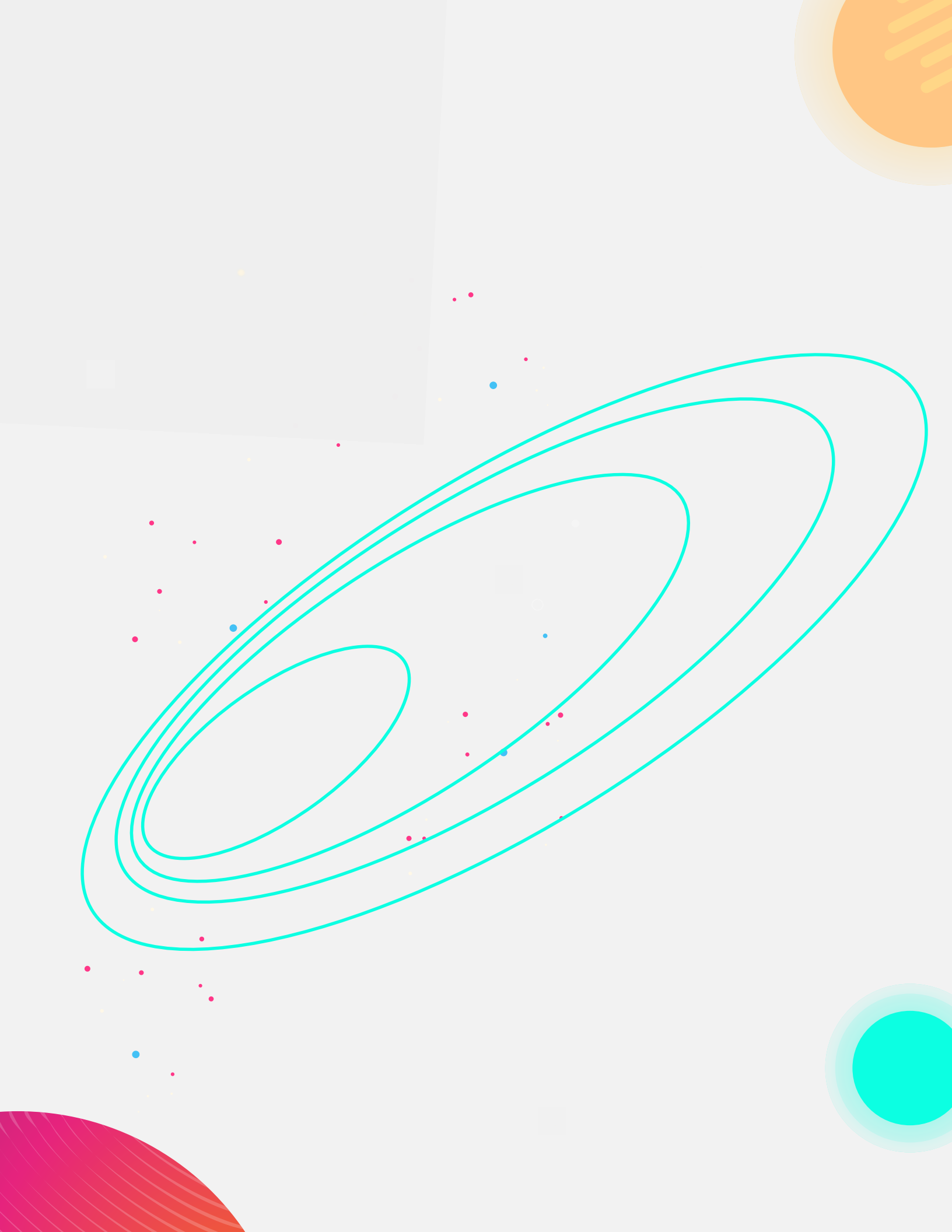




Azure Tips and Tricks: Serverless

azuredev.tips



Introduction

Hi, folks!



When I reflect back on Azure Tips and Tricks a year ago, I was only thinking that I'd write a couple of posts and move on. Fast-forward to today, the collection has grown to over 150+ tips, as well as videos, conference talks, and now an e-book spanning the entire universe of the Azure platform. What you are currently reading is a special collection of tips based on page views of the entire series over the last year. Before we dive in, you'll notice my pixelated form as you turn each page.

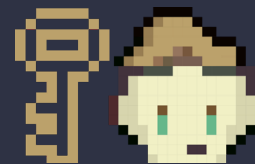
These represent:



Something I found interesting and you may too.



Additional resources to get the most out of this tip.



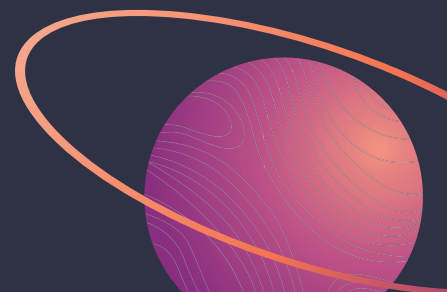
A key takeaway from the tip.

You can stay up to date with the latest Azure Tips and Tricks at:

- Blog - azuredev.tips
- Videos - videos.azuredev.tips
- eBook - ebook.azuredev.tips
- Survey - survey.azuredev.tips

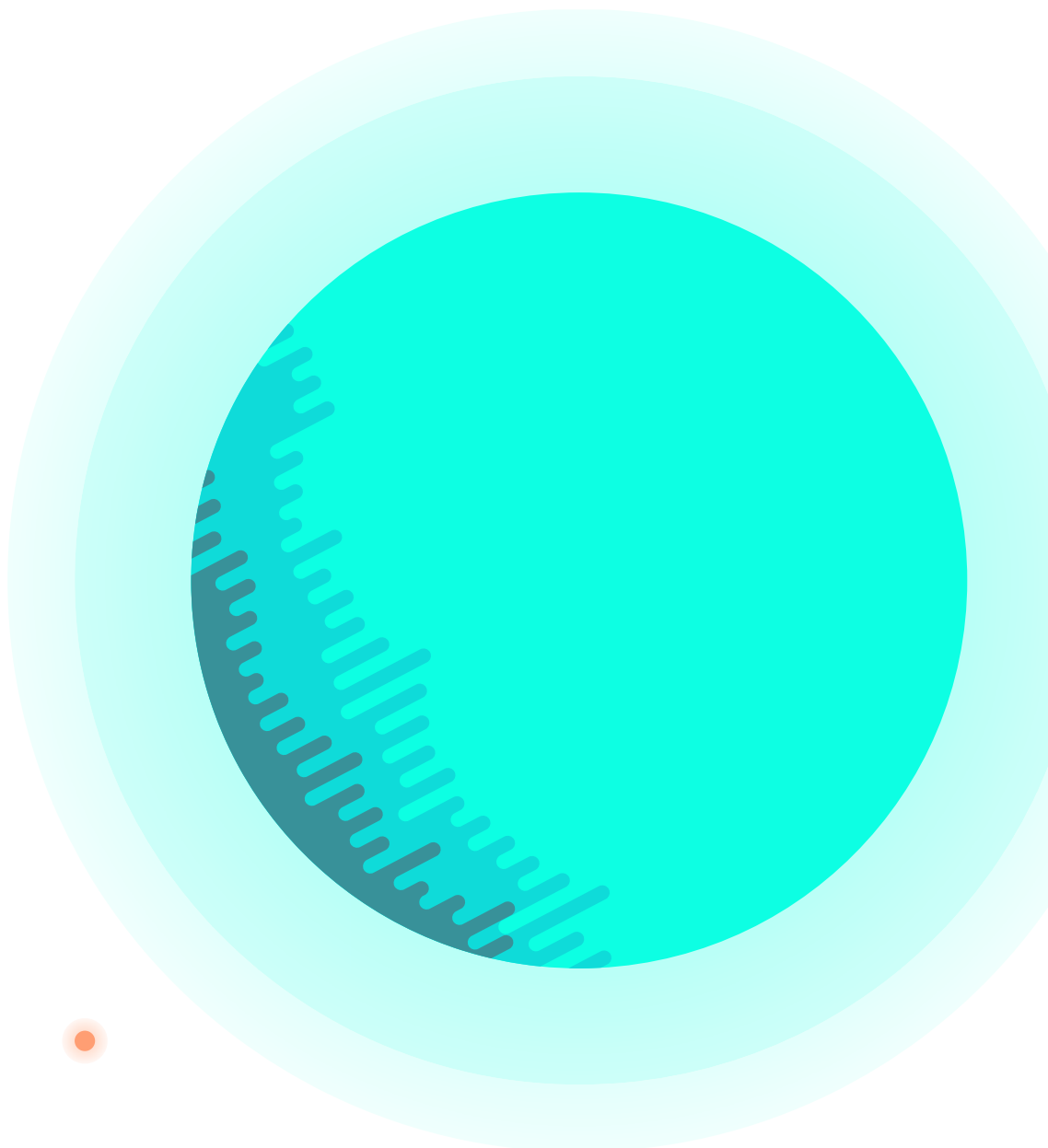
I hope you enjoy reading the eBook as much as I did writing it.

Thanks,
Michael Crump (@[mbcrump](https://twitter.com/mbc Crump))



Serverless

Here you'll find the [top 4 tips](#) from the serverless topics and it is no surprise that they include Azure Logic Apps and Azure Functions. We'll begin with two tips that show how I used Azure to help me track my running data with Azure Logic Apps and OneDrive. Next, we'll look at how I create Azure Functions projects in Visual Studio Code. Then we'll wrap up with a way to use a different route prefix with Azure Functions.



Tracking Run Data with Azure

I'd like to share a practical example of how I am using Azure in my daily life. I've started running outdoors and would like to extract several bits of information that the app on my phone generates and sends via email once the run is complete. Currently I open the email and save the [kml](#), [gpx](#), [csv](#) files to my OneDrive for historical purposes. There is a better way with Azure.

Parse Emails to Be Used in a Azure Logic Apps

Once a run is complete, the app that I use (Runmeter) generates an email with a link to the run data (GPX, CSV, KML File) in the following format:

```
Finished Run: Oct 19, 2017 at 8:46:32 PM
Route: New Route
Explorer Link: http://runmeter.com/xxx/Run-20171019-2045
Import Link: http://share.abvio.com/xxx/Runmeter-Run-20171019-2045.kml
Run Time: 1:04
Stopped Time: 0:00
Distance: 0.00 miles
Average: 0:00 /mile
Fastest Pace: 0:00 /mile
Calories: 4
GPX Link: http://share.abvio.com/xxx/Runmeter-Run-20171019-2045.gpx
CSV Link: http://share.abvio.com/xxx/Runmeter-Run-20171019-2045.csv
```

The pieces of data that we'd like to extract are the [kml](#), [gpx](#), [csv](#) URLs and the last piece of the Explorer Link URL. After we have the URLs we are going to download them automatically into a OneDrive folder.

Fire up [parser.Zapier.com](#) and create a mailbox. You'll need to send an email to it as it will be your starting template. Once you've sent an email, select the pieces of data that you want to use and give them a name. In the example below, I've already selected four pieces of data and show how to create a new one.

Extra Template:

Highlight the text you would like extracted and give it a name!

```
Runmeter Run Oct 19, 2017 at 12:22:02 PM

Finished Run: Oct 19, 2017 at 12:24:45 PM
Route: New Route
Explorer Link: http://runmeter.com/1a19e948504b98eedza9b9/
{{filename}}
Import Link: {{kml}}
Run Time: 2:12
Stopped Time: 0:08
Distance: 0.12 miles
Average: 18:42 /mile
Fastest Pace: 15:07 /mile
GPX Link: {{gpx}}
CSV Link: {{csv}}

http://www.runmeter.com
```

Save Extra Template

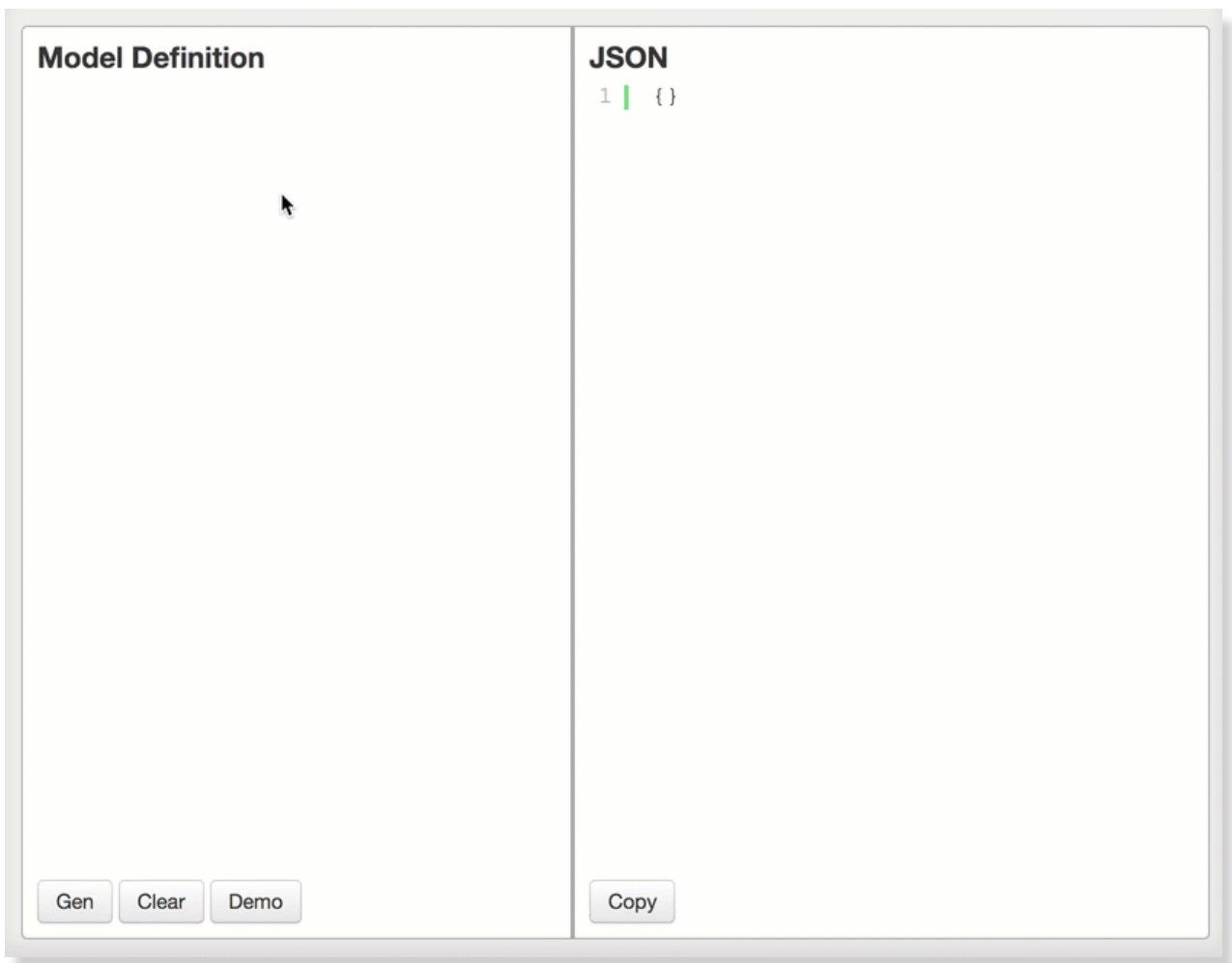
Delete Extra Template

Now that you have your mailbox created and the parser engine knows what data to extract, we can connect the app to the [Zapier Editor](#). But first let's review the pieces of data that we wanted to extract and why.

- Filename - This is the general filename that the app uses, and I think it's a piece of data we want to store.
- CSV URL - A URL to the CSV File that we'll be posting to OneDrive.
- GPX URL - A URL to the GPX File that we'll be posting to OneDrive.
- KML URL - A URL to the KML File that we'll be posting to OneDrive.

Create JSON Schema to Be Used in Azure Logic Apps

We need to create the JSON body which we'll use to create the schema. I used objgen.com/json to quickly create this piece, but you can just manually type it if you want.



Here is the JSON payload with some sample data:

```
{
  "filename": "myfilename",
  "gpx": "http://www.someurl.com",
  "csv": "http://www.someurl.com",
  "kml": "http://www.someurl.com"
}
```

Now I've clicked the "Copy" Button, headed over to jsonschema.net, pasted it in, and my [JSON schema](#) was generated.

JSON

Root ID
http://example.com/example.json

JSON

```
{
  "filename": "myfilename",
  "gpx": "http://www.someurl.com",
  "csv": "http://www.someurl.com",
  "kml": "http://www.someurl.com"
}
```

RESET SUBMIT

Pretty Plain Edit

JSON

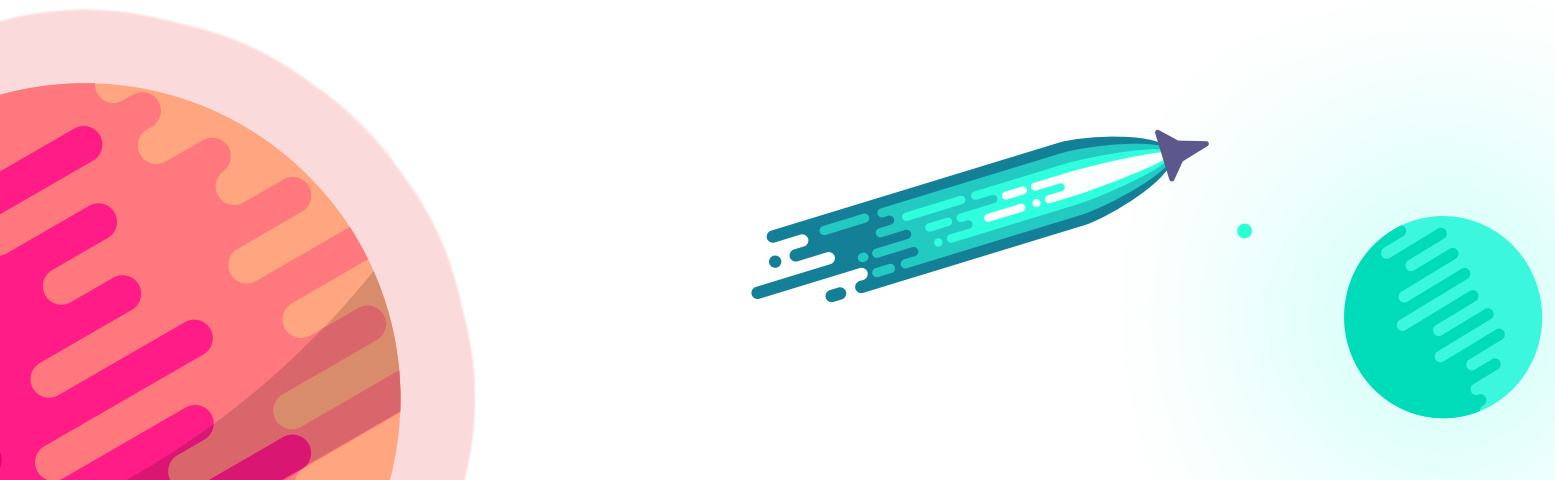
```
1 {
2   "$schema": "http://json-schema.org/draft-06/schema#",
3   "definitions": {},
4   "id": "http://example.com/example.json",
5   "properties": {
6     "csv": {
7       "id": "/properties/csv",
8       "type": "string"
9     },
10    "filename": {
11      "id": "/properties/filename",
12      "type": "string"
13    },
14    "gpx": {
15      "id": "/properties/gpx",
16      "type": "string"
17    },
18    "kml": {
19      "id": "/properties/kml",
20      "type": "string"
21    }
22  },
23   "type": "object"
24 }
```




```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "definitions": {},
  "id": "http://example.com/example.json",
  "properties": {
    "csv": {
      "id": "/properties/csv",
      "type": "string"
    },
    "filename": {
      "id": "/properties/filename",
      "type": "string"
    },
    "gpx": {
      "id": "/properties/gpx",
      "type": "string"
    },
    "kml": {
      "id": "/properties/kml",
      "type": "string"
    }
  },
  "type": "object"
}
```

Too easy! Now head over to the [Zapier Editor](#) and create a new app.


You'll want to use the [New Email](#) Trigger and use the [Email](#) Parser by [Zapier](#) and allow it to connect to your mailbox that you created earlier.




For the next step, you'll want to use an **Action** that is a **POST** request that uses **Webhooks by Zapier**. When you get to the point to where it asks you for a URL, use requestb.in to see what your HTTP client is sending or to inspect and debug webhook requests. Now you have a URL that you can use for testing. Ensure your payload is set to **JSON** and now you can select the data from your parsed email (filename, csv, kml, gpx). You can leave the rest of the fields as they are. When you finish your screen should look like the following:




Set up Webhooks by Zapier POST


 **URL (required)**


Any URL with a querystring will be re-encoded properly.











 **Payload Type (optional)**

Pay special attention to the proper mapping of the data below.

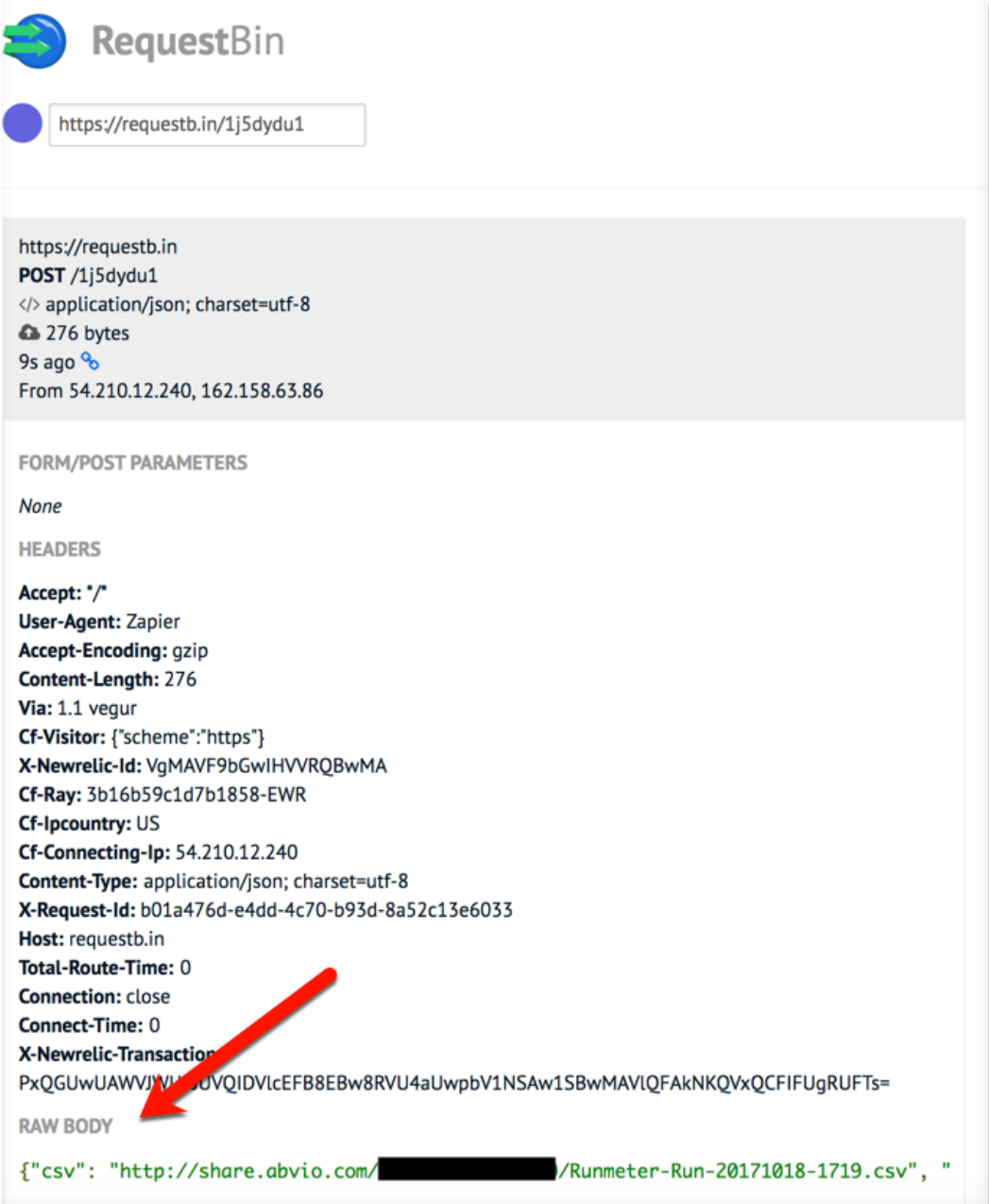


 **Data (optional)**

If you leave this empty, it will default to including the raw data from the previous step. Key, value pairs sent as data. Do not place raw JSON or form encoded values here!

csv	 Step 1 Parse Output Csv 	-
gpx	 Step 1 Parse Output Gpx 	-
kml	 Step 1 Parse Output Kml 	-
filename	 Step 1 Parse Output Filename 	-

Go ahead and save and run the test. After you switch over to your requestb.in you should see the output that matches the parsed data from the email.



RequestBin

<https://requestb.in/1j5dydu1>

`https://requestb.in`
POST /1j5dydu1
</> application/json; charset=utf-8
276 bytes
9s ago
From 54.210.12.240, 162.158.63.86

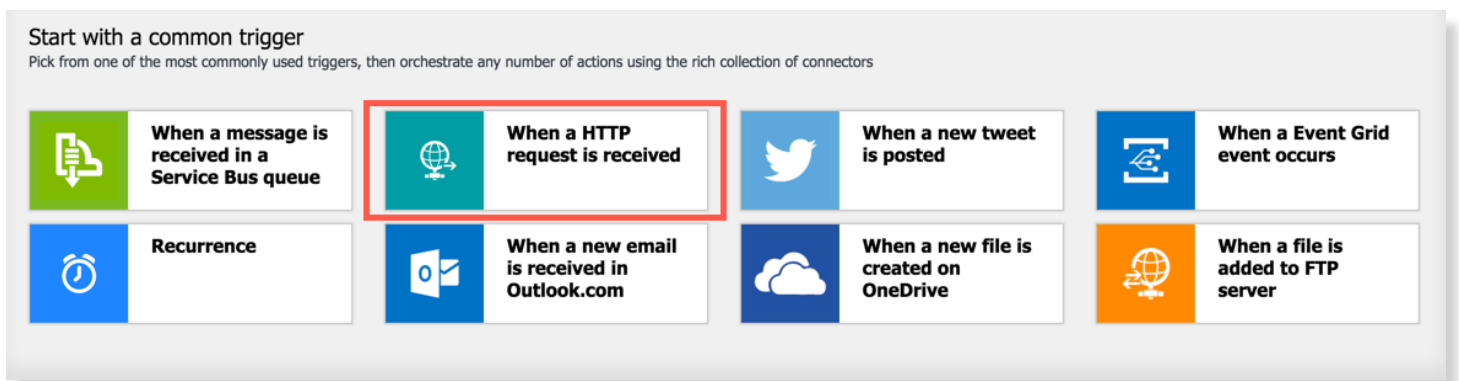
FORM/POST PARAMETERS
None

HEADERS
Accept: */*
User-Agent: Zapier
Accept-Encoding: gzip
Content-Length: 276
Via: 1.1 vegur
Cf-Visitor: {"scheme":"https"}
X-Newrelic-Id: VgMAVF9bGwIHVVRQBwMA
Cf-Ray: 3b16b59c1d7b1858-EWR
Cf-Ipcountry: US
Cf-Connecting-Ip: 54.210.12.240
Content-Type: application/json; charset=utf-8
X-Request-Id: b01a476d-e4dd-4c70-b93d-8a52c13e6033
Host: requestb.in
Total-Route-Time: 0
Connection: close
Connect-Time: 0
X-Newrelic-Transaction: PxQGUwUAWVJ...
RAW BODY
`{"csv": "http://share.abvio.com/[REDACTED]/Runmeter-Run-20171018-1719.csv", "}`


Set up an HTTP Request Trigger that is used in Azure Logic Apps

Create a new Azure Logic App by going to the Azure Portal and create a new resource

After the resource is ready, we're going to need to trigger an action when an HTTP request comes in. Thankfully, this is one of the [Common Triggers](#) and we can select it to begin.




Note that the URL isn't generated until we provide the parameters.

 When a HTTP request is received ...

HTTP POST URL

URL will be generated after save



Using the default values for the parameters. [Edit](#)

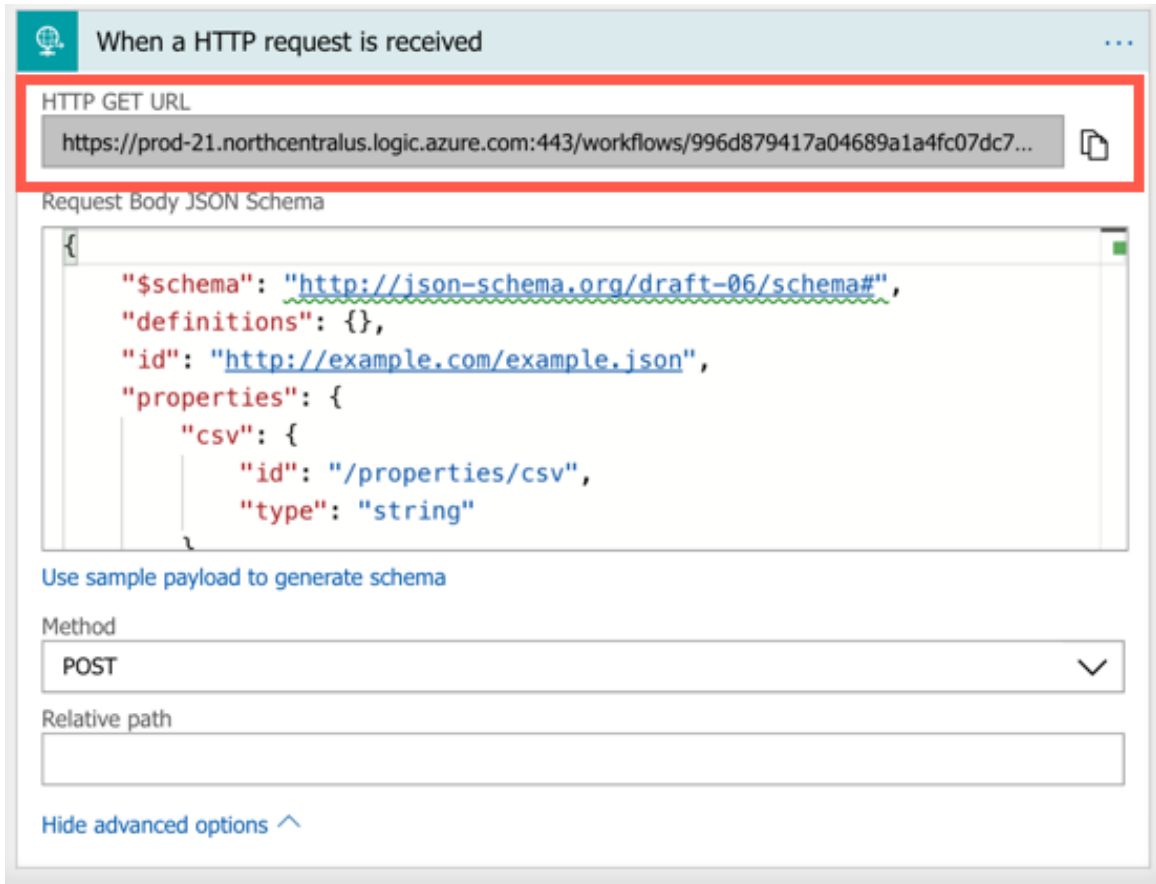
Go ahead and press [Edit](#). Remember the JSON Schema from the [last post](#)? Well, now is the time to paste it in. I'll also include it below:

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "definitions": {},
  "id": "http://example.com/example.json",
  "properties": {
    "csv": {
      "id": "/properties/csv",
      "type": "string"
    },
    "filename": {
      "id": "/properties/filename",
      "type": "string"
    },
    "gpx": {
      "id": "/properties/gpx",
      "type": "string"
    },
    "kml": {
      "id": "/properties/kml",
      "type": "string"
    }
  },
  "type": "object"
}
```



Note: You can use the "Use sample payload to generate schema" option, but I prefer the additional meta data that JSON Schema can provide.

You'll now have a GET URL that you can put in Zapier and replace the [requestb.in](#) that we stubbed out earlier.

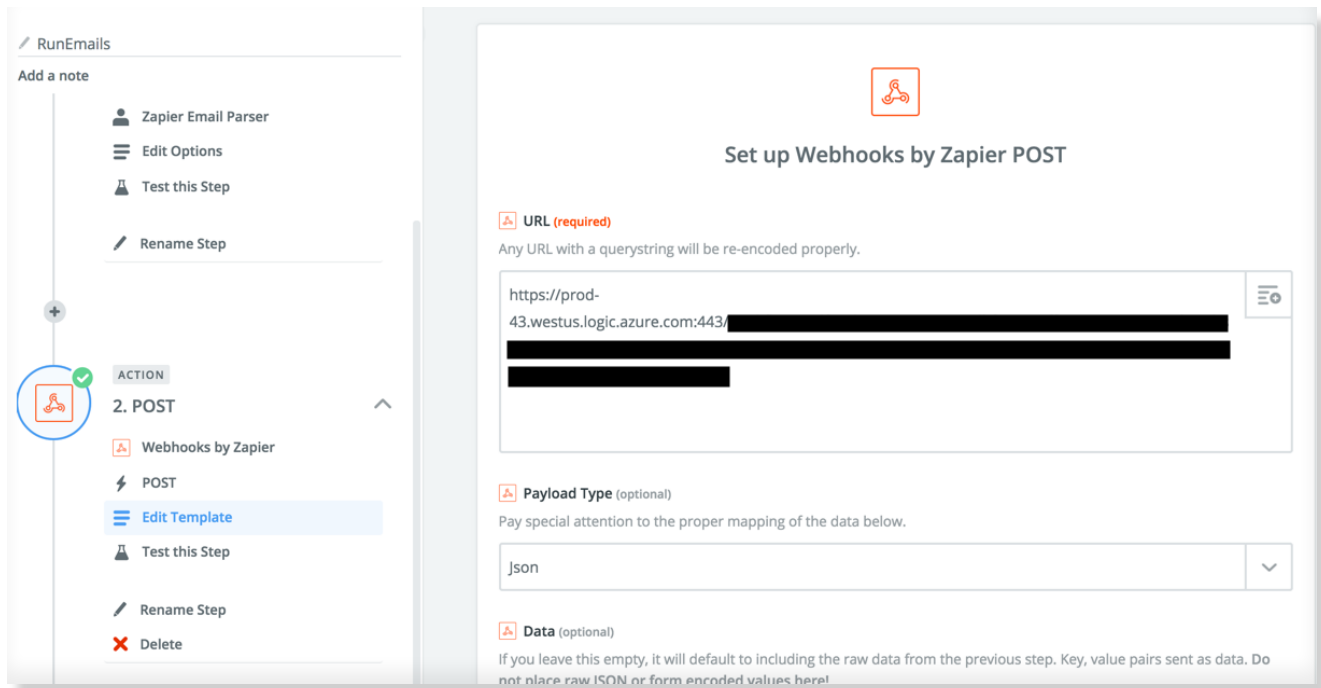


The screenshot shows the configuration for the 'When a HTTP request is received' trigger in Zapier. The 'HTTP GET URL' field is highlighted with a red border and contains the URL: `https://prod-21.northcentralus.logic.azure.com:443/workflows/996d879417a04689a1a4fc07dc7...`. Below this, the 'Request Body JSON Schema' is displayed as a JSON object:

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "definitions": {},
  "id": "http://example.com/example.json",
  "properties": {
    "csv": {
      "id": "/properties/csv",
      "type": "string"
    }
  }
}
```

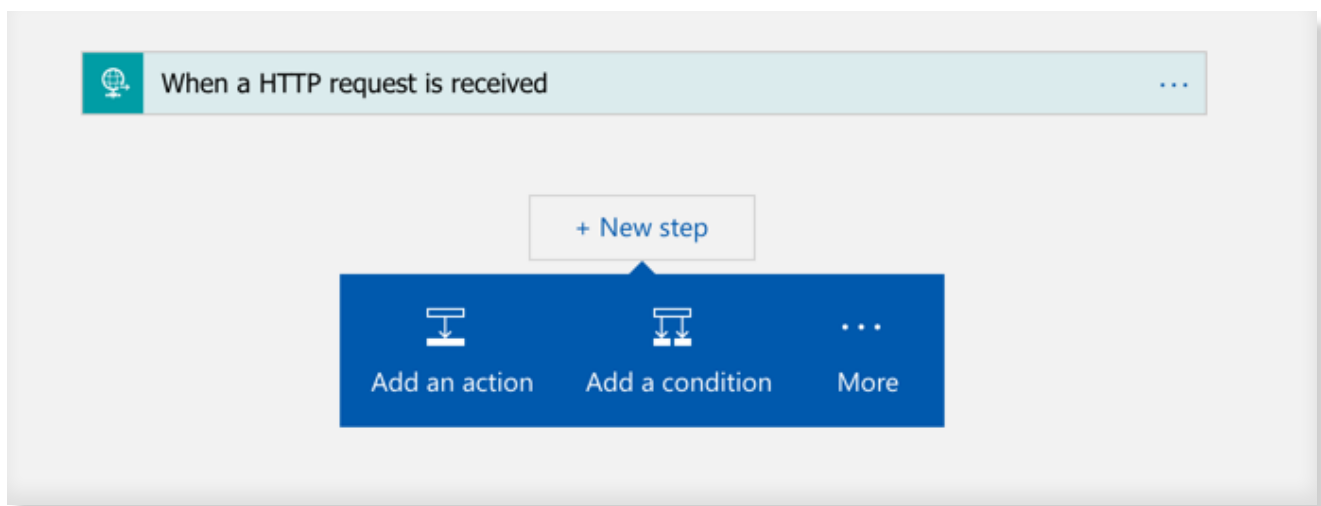
. Further down, the 'Method' is set to 'POST' and the 'Relative path' field is empty. A link 'Use sample payload to generate schema' is visible above the method dropdown. At the bottom, there is a link 'Hide advanced options' with an upward arrow.

Head back over to [Zapier Editor](#) and modify your Zap by editing the template and replacing the requestb.in URL with your live Azure Logic Apps ones.



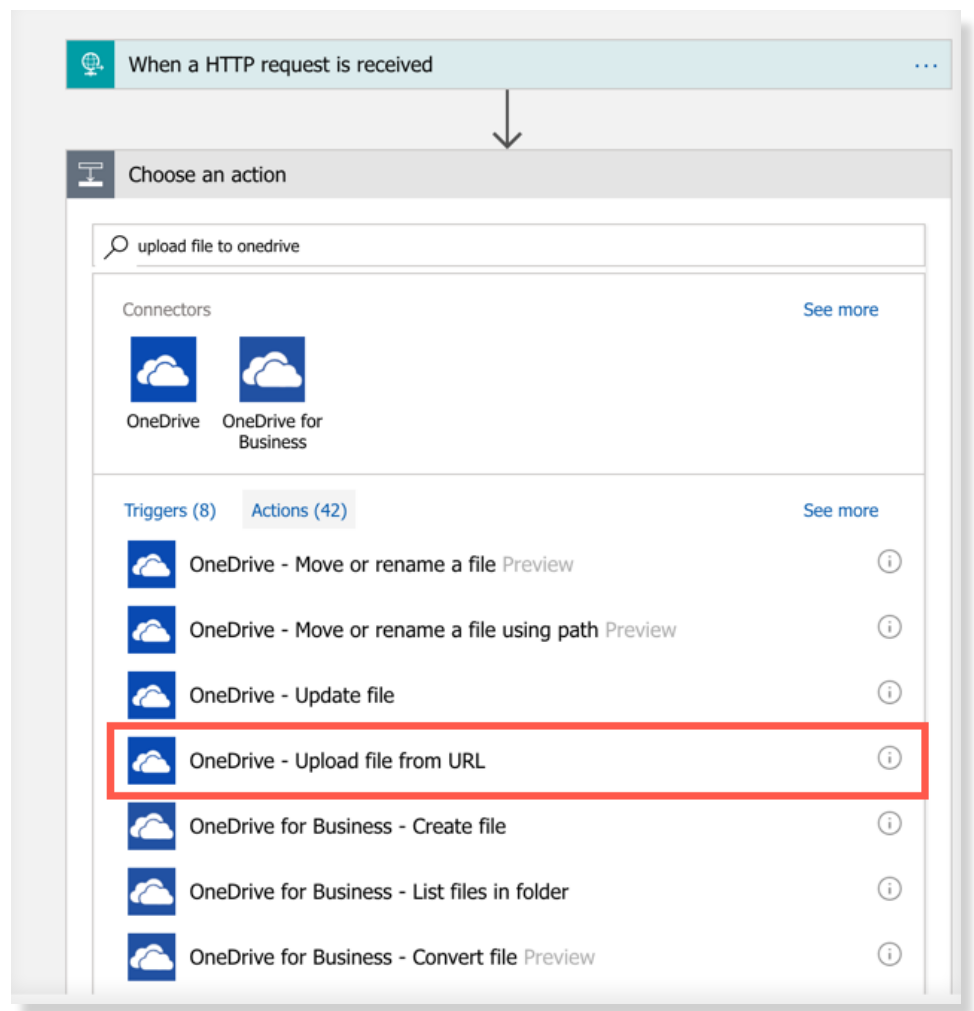
Upload Files from a URL with Azure Logic Apps

Open our existing Azure Logic App and we'll use OneDrive to automatically upload the files to my personal OneDrive account.



Typically, you'll add an **Action** or **Condition** to trigger once the HTTP request is complete.

We'll select an **Action** as we want it to run every time vs. a **Condition** which would use "If..then.." logic after the HTTP request comes in. Select **Action** and search for "upload file to onedrive" and you'll see the following is available to use.



You'll have to sign in to your OneDrive account.

Now you can pull the fields that we captured and use them as dynamic content. For example, the GPX file contains the full URL, so we can just use that dynamic field. For the destination URL, we'll construct the location we want it to go in our OneDrive account. Note that I've also setup 2 additional OneDrive actions for the KML and CSV file.

Now you'd want to send an email to your Zapier mailbox to test all the pieces to this app. Now you can switch over to your OneDrive account. If everything goes well and worked successfully you will see your new files in your OneDrive folder.

When a HTTP request is received

Upload file from URL

* Source URL
gpx

* Destination File Path
/runs/gpx/ filename .gpx

Insert parameters from previous steps
manual
csv filename gpx kml

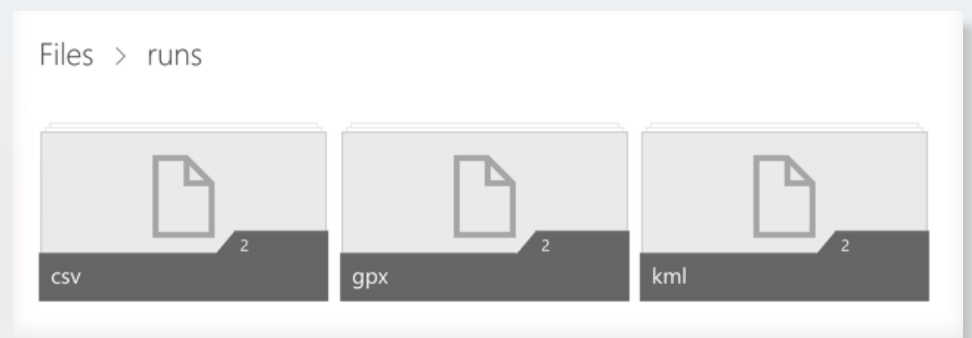
Overwrite
Yes

Connected to mbcrrump [redacted] [Change connection.](#)

Upload file from URL 2

Upload file from URL 3

If it doesn't appear to be working, you should start by looking at the [Overview](#) section, then the [Run History](#) as shown below.



Search (Ctrl+)

Overview

Activity log

Access control (IAM)

Tags

DEVELOPMENT TOOLS

Logic App Designer

Logic App Code View

Versions

API Connections

Quick Start Guides

Release notes

SETTINGS

Integration account

Access control configuration

Access keys

Properties

Locks

Automation script

Run Trigger

Refresh

Edit

Delete

Disable

Update Schema

Clone

Export

Resource group (change)

EmailToStorageRG

Location

West US

Subscription (change)

Michael's Internal Subscription

Subscription ID

Definition

1 trigger, 3 actions

Status

Enabled

Runs last 24 hours

1 successful, 2 failed

Integration Account

-- --

Plan

Consumption

Runs history

All

Start tim...

Pick a date

Pick a time

Specify the run identifier to open monitor view directly

STATUS	START TIME	IDENTIFIER	DURATION
Failed	10/21/2017, 6:0...		1.69 Minut...
Succ...	10/21/2017, 5:1...		14.58 Seco...
Failed	10/21/2017, 4:4...		1.68 Minut...
Succ...	10/21/2017, 12:...		17.24 Seco...
Succ...	10/19/2017, 8:4...		42.17 Seco...
Succ...	10/19/2017, 8:4...		1.97 Minut...
Failed	10/19/2017, 12:...		7.67 Secon...
Failed	10/19/2017, 12:...		9.6 Seconds
Failed	10/19/2017, 11:...		1.78 Minut...
Failed	10/19/2017, 11:...		1.79 Minut...

Trigger History

All

Start tim...

Pick a date

Pick a time

manual

Callback url [POST]

https://prod-43.westus.logic.azure.com:443/workflows/6d...

STATUS	START ...	FIRED
Succee...	10/21...	Fired
Succee...	10/21...	Fired
Succee...	10/21...	Fired
Succee...	10/21...	Fired
Succee...	10/21...	Fired
Succee...	10/19...	Fired
Succee...	10/19...	Fired
Succee...	10/19...	Fired
Succee...	10/19...	Fired
Succee...	10/19...	Fired

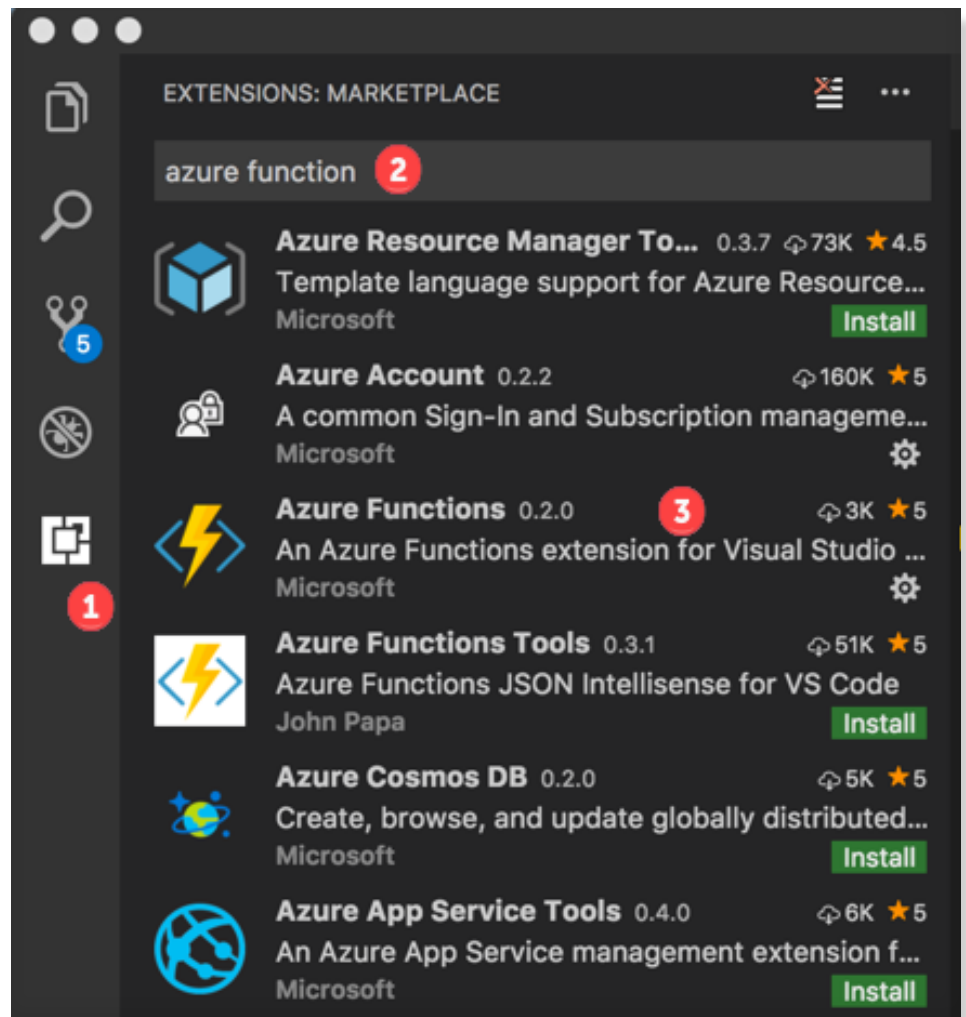
I was able to create the app in less time than it took to write this up!

Success! Our application is working properly.

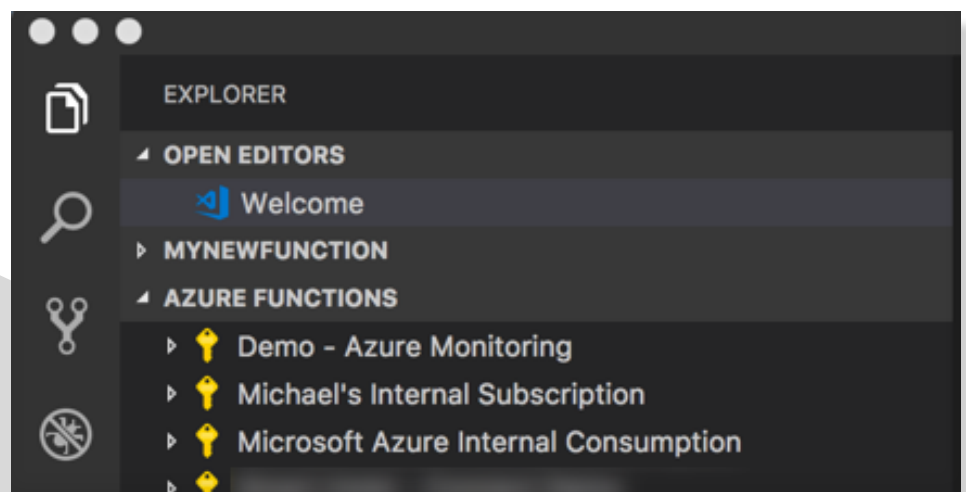
Create an Azure Functions Project with Visual Studio Code

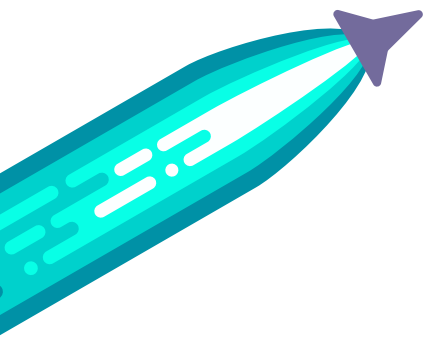
Visual Studio Code is the best thing since coffee for developers and if you pair it with Azure Functions... well, more awesome happens. In this post, we'll look at adding an Azure Function project to Visual Studio Code.

It is fairly easy as all you need to do is open VS Code, click on Extensions, search for **azure function**, and install it as shown below :



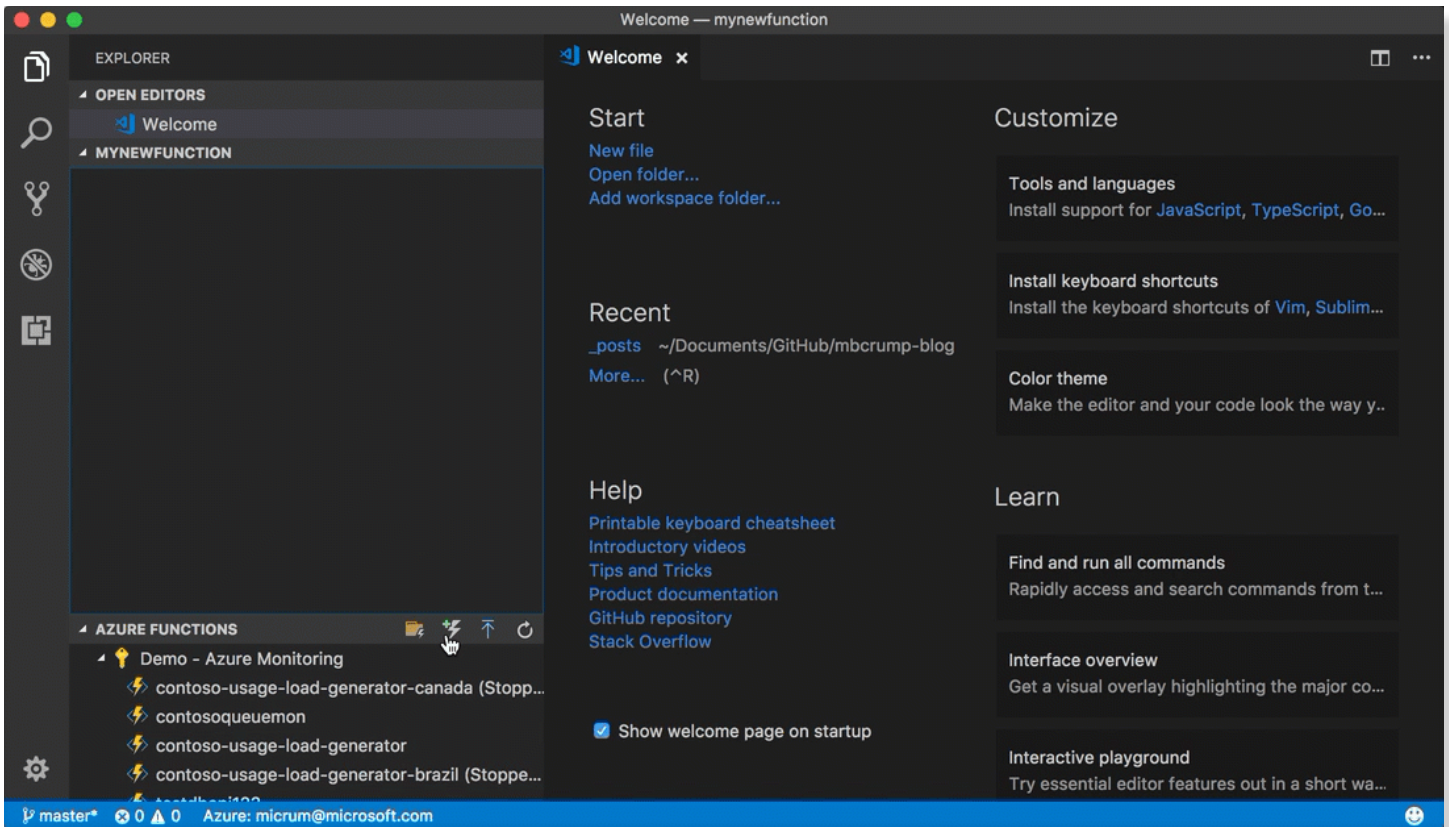
Once installed, you'll need to reload the extension and you should see your subscriptions.





You may need to sign in if Visual Studio Code hasn't already been authenticated.

Now you should create a project, then a function app, and select which template that you want to use. After you select a template, you'll need to provide a name and an authorization level.



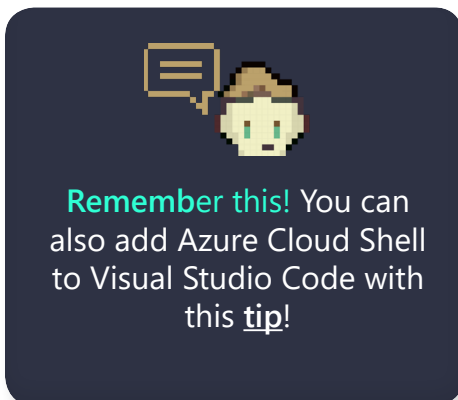
Just hit **F5** and you have a local Azure Function running in Visual Studio Code.

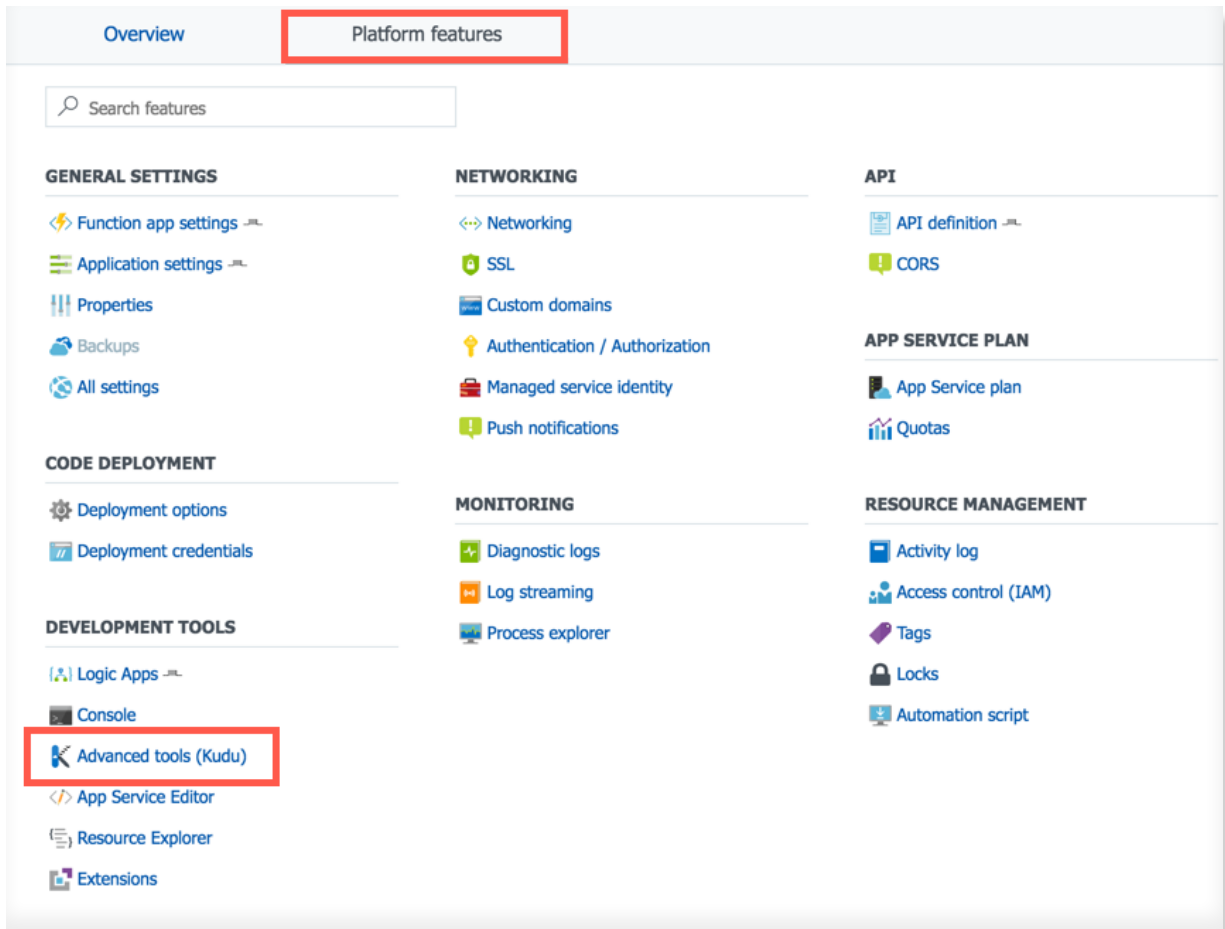
Using a different route prefix with Azure Functions

Sometimes you have the requirement to use a different route prefix than the one that Azure Functions auto-generates

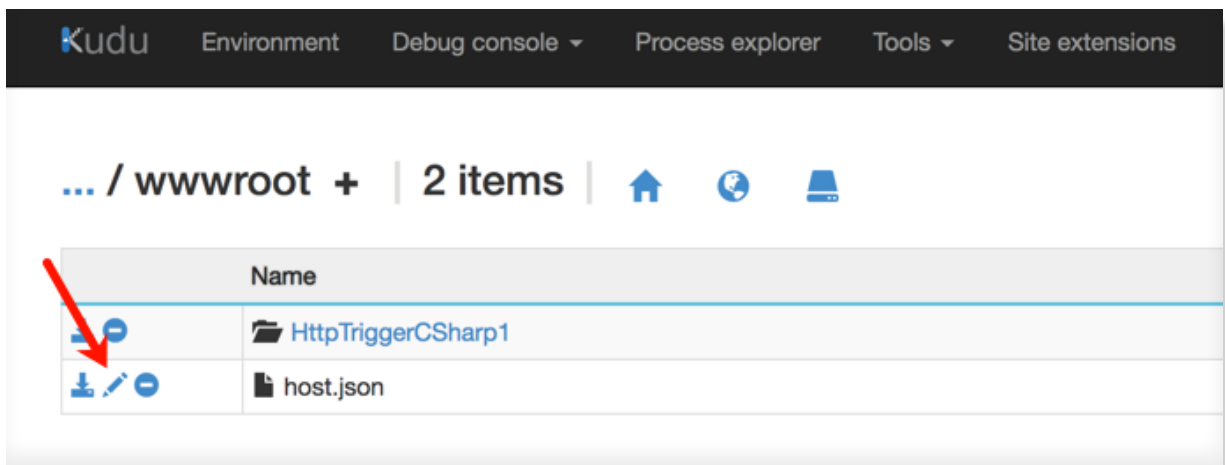
For example: <https://mynewapimc.azurewebsites.net/api/HttpTriggerCSharp1> uses **api** before the function name. You might want to either remove `api` or change it to another name.

I typically fix this by going into the Azure Portal and clicking on my Azure Function. I then click on **Platform Features** and **Advanced tools(Kudu)**.





I then navigate to [wwwroot](#) and hit edit on the [host.json](#) file.



Inside the editor, add the [routePrefix](#) to define the route prefix. So if I wanted the route prefix to be blank, then I'd use the following:

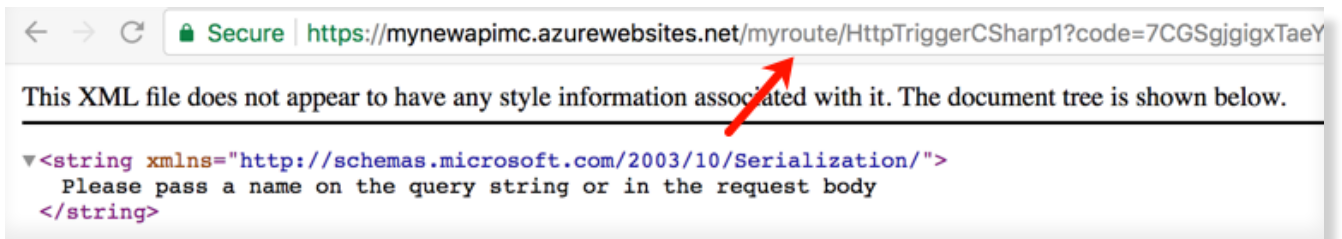
```
{
  "http": {
    "routePrefix": ""
  }
}
```

Simply restart your Azure Function and now my URL is accessible without [api](#).



On the flip side, if I wanted a route prefix, then I'd just add the following:

```
{
  "http": {
    "routePrefix": "myroute"
  }
}
```



Keep in mind that best practice (as far as I can tell) is to use [api](#), but wanted to flag this as only you can make your design decisions.

Conclusion

There are 130+ additional tips waiting on you that cover additional topics such as :

- App Services
- CLI
- Cloud Shell
- Cognitive Services
- Containers
- Cosmos DB
- Functions
- IoT
- Logic Apps
- Portal
- PowerShell
- Productivity
- Storage
- SQL and Search

Find all of these and more at azuredev.tips

Don't forget that if you are modernizing an existing application or building a new app, you can get started Azure for free and get:

- \$200 credit toward use of any Azure service
- 12 months of free services—including compute, storage, network, and database
- 25+ always-free services—including serverless, containers, and artificial intelligence

[Start free](#)

Until next time,

Michael Crump [@mbcrump](#)

signing off...



Azure Tips and Tricks

azuredev.tips



Copyright © 2018 by Microsoft Corporation. All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Made with love By [Red Door Collaborative.com](http://RedDoorCollaborative.com)