



Microsoft Azure

Azure Tips and Tricks

azuredev.tips

ISBN 978-1-7327041-2-1

www.dbooks.org



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 eBook 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. I've grouped the top tips and landed on four categories that cover web, data, serverless, and productivity. 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)

Table of Contents

1 WEB

2 DATA

3 SERVERLESS

4 PRODUCTIVITY

WEB

If you've used Azure, you've more than likely used Azure App Service to easily host web applications, REST APIs, and mobile back ends. In this set of tips, I've pulled out the [top 6 tips](#) since the creation of Azure Tips and Tricks for Azure App Service. They include easily working with files in the console, easily setting up staging environments and swapping between them, and routing traffic to different versions of your app to "Test in Production". I'll also cover how you can implement performance testing, best practices for App Settings in Azure App Service, and cloning a web app that is especially helpful if you have customers all over the world.



[Back to Table of Contents](#)

Working with Files in Azure App Service



You can learn more about Azure App Service [here](#)

We'll take a look at the files inside an Azure App Service web site and how you can easily work with them.

Console Access to my App Service

Go to the Azure Portal and select my App Service. Click on **Console** under **Development Tools** to have a command prompt to quickly work with my Azure App Service.

The screenshot shows the Azure App Service console interface for the 'MyQuizApplication' web app. The left sidebar lists various development tools: Clone app, **Console** (which is selected and highlighted with a red box), Advanced Tools, App Service Editor (Preview), Performance test, Resource explorer, Testing in production, and Extensions. The main right pane is a terminal window showing a directory tree and a command prompt. The command prompt shows the path `D:\home\site\wwwroot` and a ready prompt `> |`.

```
MyQuizApplication - Console
App Service

Search (Ctrl+ /)

Clone app
Console
Advanced Tools
App Service Editor (Preview)
Performance test
Resource explorer
Testing in production
Extensions

MOBILE

D:\home\site\wwwroot
> |
```

As you can tell from the screenshot, I start in `D:\home\site\wwwroot`. I can type `dir` to see a current directory listing.

```
Volume in drive D is Windows  
Volume Serial Number is FE33-4717
```

```
Directory of D:\home\site\wwwroot
```

```
09/21/2017  08:35 PM    <DIR>          .  
09/21/2017  08:35 PM    <DIR>          ..  
09/20/2017  09:03 PM    <DIR>          css  
09/20/2017  09:03 PM            5,351 Default.html  
09/20/2017  09:03 PM    <DIR>          js  
09/20/2017  09:03 PM            1,950 jsQuizEngine.sln  
09/20/2017  09:03 PM            304 jsQuizEngine.userprefs  
09/20/2017  09:03 PM            31,744 jsQuizEngine.v12.suo  
09/20/2017  09:03 PM    <DIR>          PrecompiledWeb  
09/20/2017  09:03 PM    <DIR>          quiz  
                           4 File(s)        39,349 bytes  
                           7 Dir(s)   1,072,893,952 bytes free
```



Quick Tip You can type **help** from the console window for a list of available commands.

I can do basic commands here and even use **TYPE <FILENAME>** to parse the output of a file to the screen. You can make directory and so forth, but keep in mind that this is a sandbox environment and some commands which require elevated permissions may not work.



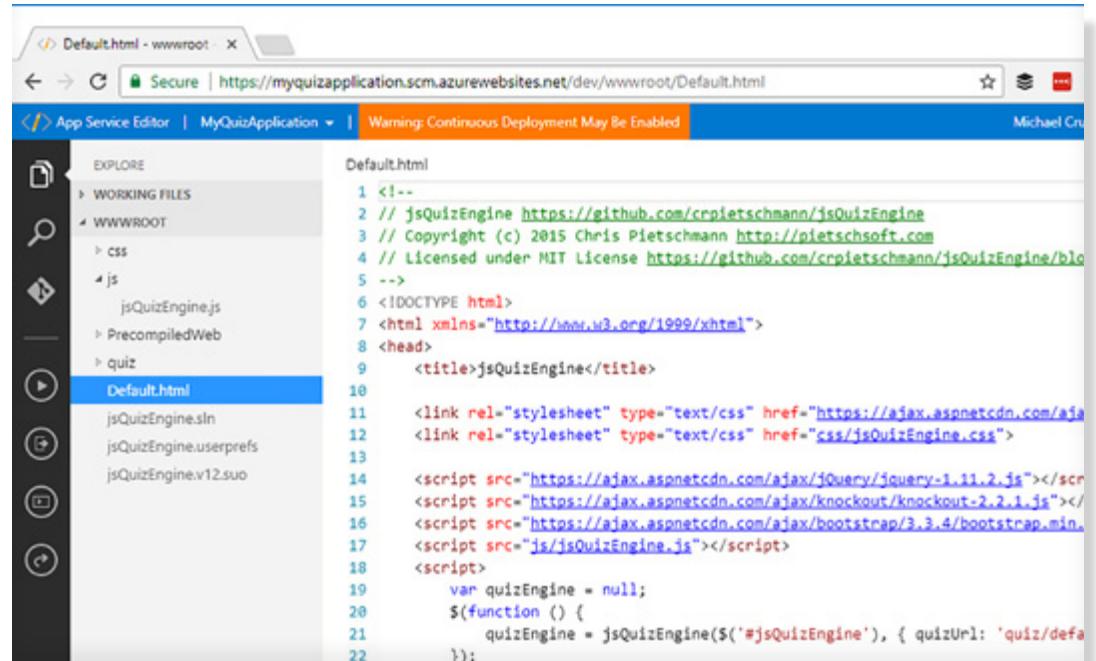
If you're familiar with VS Code, then you'll be right at home as you can explore, search and add to Git.

You can also manipulate files from within the window. This makes it easy to add, edit or delete files.

A VS Code Experience to an Azure App Service

A VS Code Experience to an Azure App Service

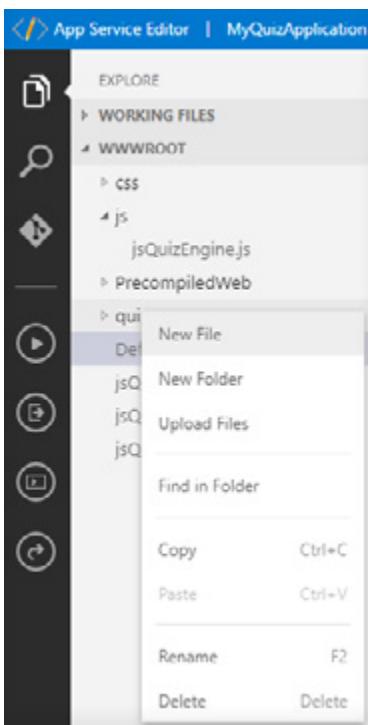
There is also another option that is called "[App Service Editor](#)" located just two items down from "[Console](#)" that you picked before.



The screenshot shows the Azure App Service Editor interface. The left sidebar has icons for file operations like Open, Save, Find, Copy, Paste, and Delete. The "EXPLORE" section shows the "WORKING FILES" tree, with "WWWROOT" expanded to show "css", "js", "PrecompiledWeb", and "quiz". "Default.html" is selected and highlighted in blue. The main area displays the content of "Default.html":

```
1 <!--
2 // jsQuizEngine https://github.com/cpietschmann/jsQuizEngine
3 // Copyright (c) 2015 Chris Pietschmann http://pietschsoft.com
4 // Licensed under MIT License https://github.com/cpietschmann/jsQuizEngine/blob/master/LICENSE
5 -->
6 <!DOCTYPE html>
7 <html xmlns="http://www.w3.org/1999/xhtml">
8 <head>
9   <title>jsQuizEngine</title>
10  <link rel="stylesheet" type="text/css" href="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.2.js"></scr
11  <link rel="stylesheet" type="text/css" href="css/jsQuizEngine.css">
12
13
14  <script src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.2.js"></scr
15  <script src="https://ajax.aspnetcdn.com/ajax/knockout/knockout-2.2.1.js"></scr
16  <script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.4/bootstrap.min.
17  <script src="js/jsQuizEngine.js"></script>
18
19    var quizEngine = null;
20    $(function () {
21      quizEngine = jsQuizEngine($('#jsQuizEngine'), { quizUrl: 'quiz/defa
22    });


```



Just like in VS Code, you can modify your settings and even change your theme.

Kudu Diagnostic Console

No App Service tutorial is complete without mentioning Kudu Diagnostic Console. You can access it from within the [App Service Editor](#) under your [app name](#) -> [Open Kudu Console](#) or through the portal under [Advanced Tools](#).

/ + | 4 items |

Name	Modified	Size
ssh	9/20/2017, 2:03:36 PM	
data	9/21/2017, 1:31:04 PM	
LogFiles	9/21/2017, 1:31:06 PM	
site	9/21/2017, 1:31:06 PM	



[Use old console](#)

Kudu Remote Execution Console
Type 'exit' then hit 'enter' to get a new CMD process.
Type 'cls' to clear the console

Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

D:\home>



Quick Tip The App Service Editor is a great choice if ever in doubt and you can access it directly [here](#)

You can just click on the folder name to navigate or type in the command. You can also easily manipulate the files, but I like the App Service Editor better for that functionality.

Editor is perfect for lightweight work such as editing files whereas Kudu puts you deep into the weeds with debugging information, file manipulation and more.

The main reason that I typically come to the Kudu Diagnostic Console is to download files.

Test Web Apps in Production with Azure App Service



You can learn more about Azure Deployment Slots [here](#)

We'll take a look at the files inside an Azure App Service web site and how you can easily work with them.

Creating Deployment Slot

Deployment slots let you deploy different versions of your web app to different URLs. You can test a certain version and then swap content and configuration between slots.

Go to the Azure Portal and select my App Service and click on **Deployment Slots** under **Deployment** to get started. Then click on the **Add Slots** button. Give it a name such as staging then use an existing configuration source. We'll use our "production" web app. You know, the cool quiz application. [Aka.ms/azuretips/myquizapp](https://aka.ms/azuretips/myquizapp)

Add a slot □ X

Deployment slots let you deploy different versions of your web app to different URLs. You can test a certain version and then swap content and configuration between slots.

* Name ?
 ✓

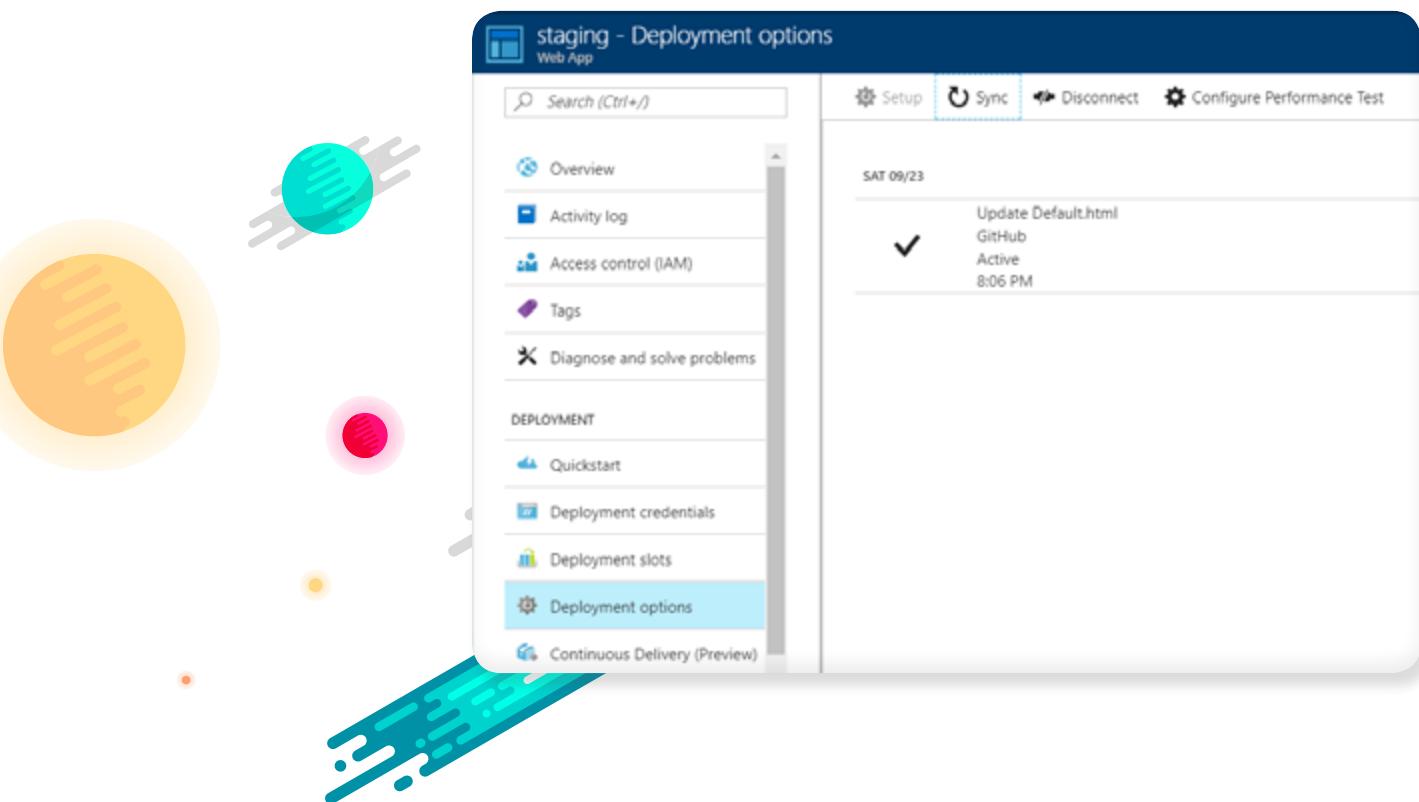
Configuration Source
 ▼

Great, now if we go back to Deployment Slots, we should see it running.

NAME	STATUS	APP SERVICE PLAN
myquizapplication-staging	Running	StaticAppServicePlan

Click on the new staging site that we just created and you'll notice that it has appended the word **staging**. You'll also notice we have a new site: Aka.ms/azuretips/quizsourcegit

We need to push a new version of our existing quiz application to this staging slot. Go to **Deployment Options** and select **External Repository**. Give it the following URL: Aka.ms/azuretips/quizsource and hit OK. You might have to hit Sync, and you'll eventually see the following:





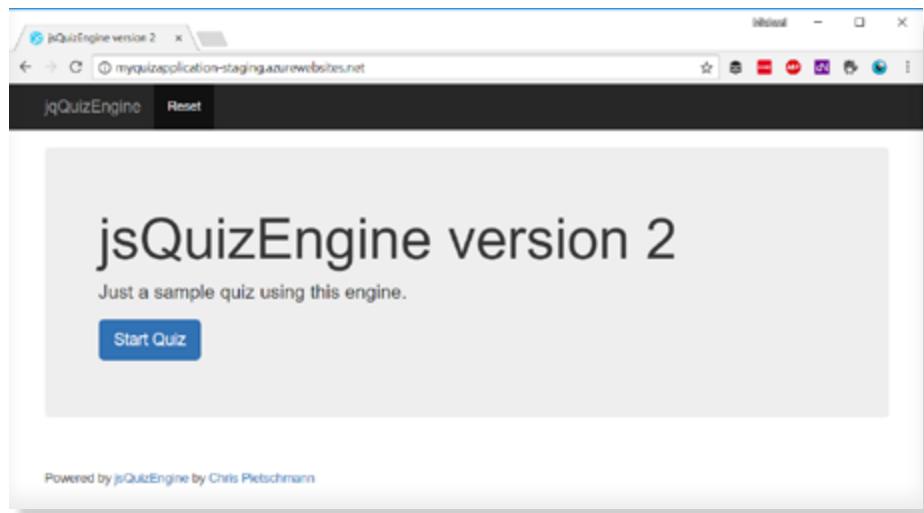
We could now return to the original app service that we created and swap between the two sites that we have.

For example, you might want to move the **staging** site over to the **production** site and vice versa. The power of this is that your users don't experience a downtime and you can continue working in your preferred space until ready to move to production.



Source Code The source code to the staging environment can be found [here](#)

Give it a couple of minutes until you see that it has completed pulling down your code from Git and then go to the new URL of your site. You can find the URL on your overview page. In my case it is, <http://myquizapplication-staging.azurewebsites.net/>



Success! This is our new site as indicated by the awesome large font that says **jsQuizEngine version 2**.

In this tip, we'll look at a feature called Testing in Production which allows you to test your application in production. Not scary at all!



Hold up! You'll want to take a look at the deployment slots in the previous tip if you haven't worked with deployment slots before.



What is Static Routing This section lets you control how traffic is distributed between your production and other slots. This is useful if you want to try out a new change with a small percentage of requests and then gradually increase the percentage of requests that get the new behavior.

Testing Web Apps in Production with Azure App Service

Go to the Azure Portal and select my App Service and click on Testing in Production under **Development Tools** to get started. The first thing you'll see is **Static Routing** and you'll notice that it's looking for a deployment slot and traffic percentage.

We'll want to split the traffic to our site into two groups to test our new site and see if customers like it. Since this is just a demo, I want to send a large number of folks to our new **staging** site as shown below.

Slot	Traffic %
staging	75%
Choose deployment slot	Traffic %
production	25%

Great! Now keep in mind that we have two versions of our site: one that is **production** and one that is **staging**. They are identical except for the staging site has a large font that says **jsQuizEngine version 2**.

We don't want to **swap** sites, we just want to **distribute** traffic between the two sites.

I can test this by going to my production URL and refreshing the site until the staging site is shown with the production URL.

A screenshot of a Microsoft Edge browser window showing a quiz application. The title bar says "jsQuizEngine". The address bar shows "myquizapplication.azurewebsites.net". The page has a dark header with "jqQuizEngine" and "Reset" buttons. The main content area has a light gray background. It features a large "Default Quiz" heading, a subtext "Just a sample quiz using this engine.", and a blue "Start Quiz" button. Below the main content, a footer says "Powered by jsQuizEngine by Chris Pietschmann". The browser's status bar at the bottom shows the URL "myquizapplication.azurewebsites.net/#".



Success! It works, but what happens when they leave the site? We actually store a cookie that keeps track of it. You can find this cookie yourself by inspecting the site and looking for the cookie shown on the next page.

Screenshot of the Microsoft Edge DevTools Application tab showing cookies. A cookie named 'x-ms-routing-name' with the value 'staging' is selected and highlighted with a red border.

Name	Value	Domain	Path	Expires / M...	Size	HTTP	Secure	SameSite
test	noojgilkidnpfjb...	/data/sha...	/	Session	4			
ARRAffinity	d1590768...	.myquizapplic...	/	Session	75	✓		
TiPMix	86.110841...	.myquizapplic...	/	Session	22			
x-ms-routing-name	staging	.myquizapplic...	/	Session	24			

You could actually force the **old production** site by setting the **x-ms-routing-name** cookie to **self** or providing it in the URL query string such as <http://myquizapplication.azurewebsites.net/?x-ms-routing-name=self> You could even use the URL to let your users test different versions of your site. For example, I could use <http://myquizapplication.azurewebsites.net/?x-ms-routing-name=staging> to let users try my new website before I push it live. This is very neat stuff, folks!



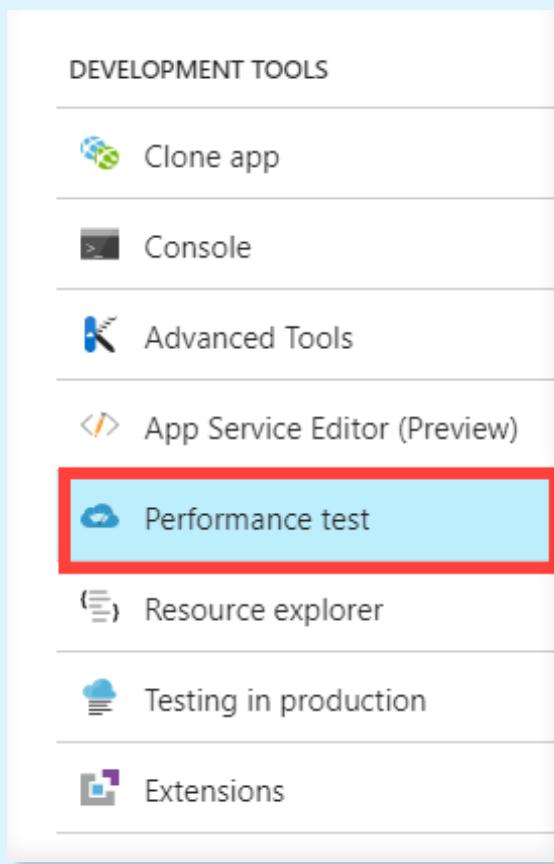
Learn more about load testing at
[Aka.ms/azuretips/vsts](http://aka.ms/azuretips/vsts)

In this tip, we'll look at a simple and quick way to perform load testing of your web app.

Load Testing web apps with Azure App Services

Load Testing allows you to test your web app's performance and determine if your app can handle increased traffic during peak times. You can find this tool by logging into your Azure account, going to your App service that you created, and looking under [Development Tools](#).

Inside the blade, select New and you will see the following options:



New performance test

PREVIEW

CONFIGURE TEST USING ⓘ
Test type: ManualTest 1 Url

NAME
PerfTest01

GENERATE LOAD FROM ⓘ
West US (Web app Location)

USER LOAD ⓘ
250

DURATION (MINUTES) ⓘ
5

Configure test using

PREVIEW

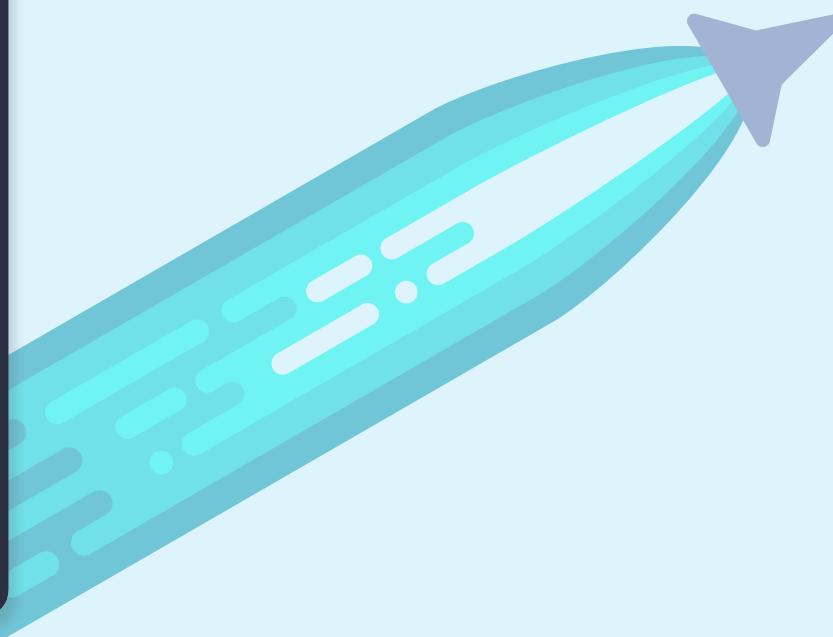
TEST TYPE ⓘ
Manual Test

URL ⓘ
http://myquizapplication.azurewebsites.net



Use Case Scenario Suppose you have a web app and you have something for sale. You have an upcoming promo that last year had 175 users connected for 5 minutes. Users complained that the site was slow and since your site has grown, you want to improve customer satisfaction by reducing the page load time and test your web app with a load of 250 users for 5 minutes. Let the test run and you'll be presented with the following information once it has completed:

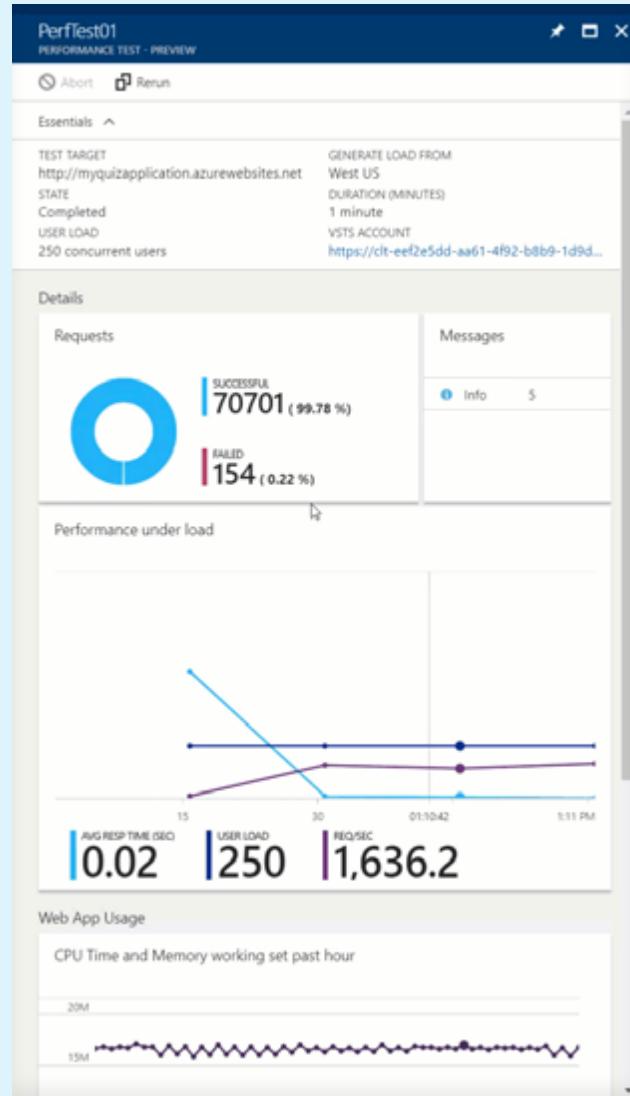
You have the option to [Configure Test](#) and you can leave this as [Manual Test](#) or [Visual Studio Web Test](#). The main difference between the two is that with the latter you can select multiple URLs and even use a HTTP Archive file (such as one created by Fiddler). Leave the testing option as manual and select a name and location, and make sure you leave the defaults as 250 users for 5 minutes.





Look out! Keep in mind that there is a charge for performing a load test in terms of virtual users as indicated in the screenshot.

We were able to do this without writing code and with just a couple of clicks in the portal.



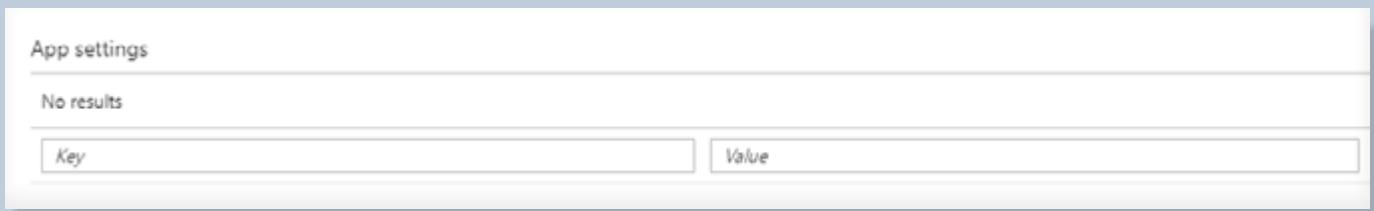
Learn more about App Settings at
[Aka.ms/azuretips/
appservconfig](http://Aka.ms/azuretips/appservconfig)

In this post, we'll take advantage of App Settings to store a Key/Value pair securely in Azure and access it in your web app.

Working with App Settings and Azure App Services

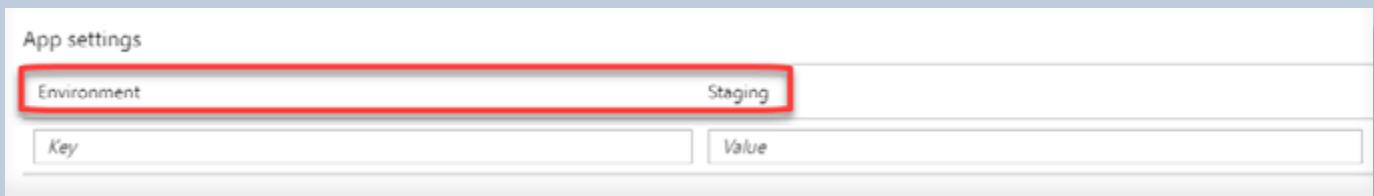
App Settings are used to store configurable items without making any changes to the code. The key/value pairs are stored behind the scenes in a configuration store, which is nice because sensitive information never shows up in a web.config, etc. file. In order to take advantage of this, you'll need to log into your Azure account and go to your App Service that you created and look under **Development Tools** then you will see **Application Settings**.

Open it and scroll down and you'll see **App Settings** as shown below.



The screenshot shows a web interface titled "App settings". Below the title, it says "No results". There are two input fields: "Key" and "Value".

We're going to add an App Setting in Azure. I added one with the key of Environment and the value is set to **Staging**.



The screenshot shows the same "App settings" interface. A new row has been added with the "Key" field containing "Environment" and the "Value" field containing "Staging". The "Environment" entry is highlighted with a red border.

Open or create your ASP.NET MVC app and modify the appSettings section of the [web.config](#) file to add our **Environment** key/value pair as shown below:

```
<appSettings>
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="Environment" value="Production" />
</appSettings>
````
```

Now it is in our web configuration of our app. In order to see the value, we'll need to add it to a page to display the value.

If you're using ASP.NET MVC (for example), then navigate to your **\*\*appname/Views/Home/Index.cshtml\*\*** file and add the following **\*\*using\*\*** statement followed by a call to **\*\* ConfigurationManager\*\*** as shown below :

```
```html
@using System.Configuration
@{
    ViewBag.Title = "Home Page";
}

<div class="jumbotron">
    <h1>Testing App Settings</h1>
    @ConfigurationManager.AppSettings["Environment"]
</div>
```



If you run the application locally, then you'll see **Production** as it is coming from the [web.config file](#), but if you run it inside of Azure, then you'll see **Staging** as it is coming from the [Apps Settings](#) configuration store located in Azure. Neat stuff!

Testing App Settings

Production

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

[Learn more »](#)

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

[Learn more »](#)

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

[Learn more »](#)



Connection Strings vs. App Settings You may have noticed **Connection Strings** right below the **App Settings** option and wonder when to use it. A general rule of thumb is to use Connection Strings for database **connection strings** and **App Settings** for key/value pair application settings. If you examine your **web.config** file, then you'll see there is also a section for **connectionStrings** just as there is a section for **appSettings**.

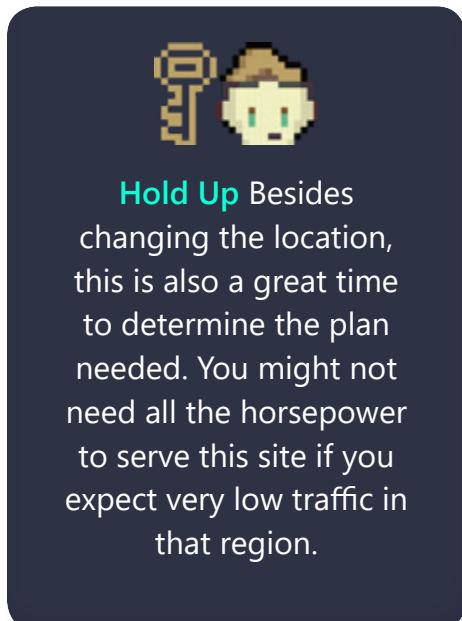
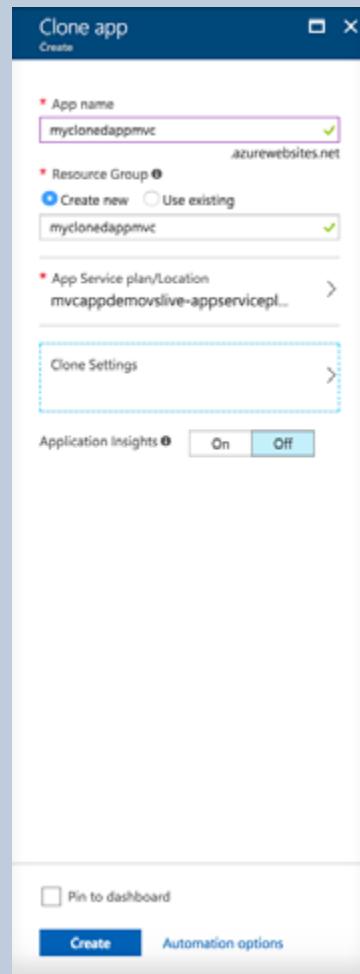


Cloning is the ability to duplicate an existing Web App to a newly created app that is often in a different region. This will enable customers to deploy a number of apps across different regions quickly and easily.

Cloning Web Apps Using and Azure App Services

Scenario: A company has an existing web app in **West US**, they would like to clone the app to **East US** to serve folks that live on that site with better performance such as latency. To do this, log into your Azure account and go to your App Service that you created. Look under **Development Tools** and find **Clone App**.

Open it and
you'll see the following:



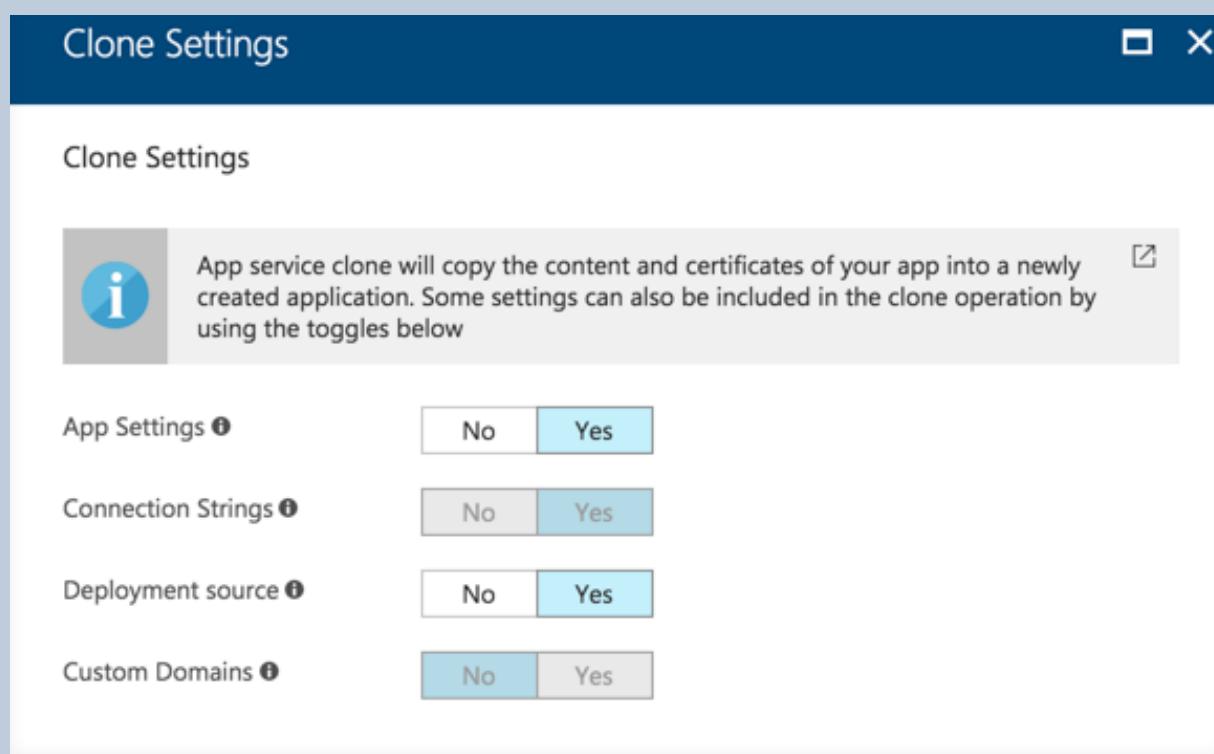
Ensure you give it an:

- **App Name** - Something unique as this site will live in something.azurewebsites.net
- **Resource Group** - Create a new one or use an existing one
- **App Service Plan/Location** - This is a good time to associate a new plan that will determine the location, features, and cost, and compute resources associated with your app.

- **Application Insights** - You can turn it on or off to help you detect and diagnose issues with .NET apps.
- **Clone Settings** - Clone will copy the content and certificates of your app into a newly created application. You can also copy things like App Settings, Connection Strings, Deployment Source, and Custom Domains.

Finally, there is **Automation Options** which brings you to the Azure Resource Manager templates that are so valuable.

[Aka.ms/azuretips/resourcemanager](http://aka.ms/azuretips/resourcemanager)





What is a Azure Resource Manager again? Azure

Resource Manager enables you to work with the resources in your solution as a group. You can deploy, update, or delete all the resources for your solution in a single, coordinated operation. You use a template for deployment and that template can work for different environments such as testing, staging, and production.

Resource Manager provides security, auditing, and tagging features to help you manage your resources after deployment.

[Aka.ms/azuretips/
appservdeploy](http://Aka.ms/azuretips/appservdeploy)

Once everything is set up then press **Create** and you'll see the **Deployment in Progress** begin. You can click on it while deploying to see details as shown:

CloneAppe7924f0c-8a6d Deployment

Delete Cancel Refresh Redeploy View template

Deploying

DEPLOYMENT DATE	10/7/2017, 8:40:26 PM
STATUS	Deploying
DURATION	2 minutes 34 seconds
RESOURCE GROUP	myclonedappmvc
RELATED	Events

Outputs

NO DEPLOYMENT OUTPUTS

Inputs

NAME	myclonedappmvc
HOSTINGPLANNAME	mvcappdemovslive-appserviceplan
LOCATION	South Central US
HOSTINGENVIRONMENT	
SERVERFARMRESOURCEGROUP	mvcappdemovslive-resourcegroup
SUBSCRIPTIONID	d1ecc7ac-c1d8-40dc-97d6-2507597e7404

Operation details

RESOURCE	TYPE	STATUS	TIMESTAMP
myclonedappmvc	Microsoft.Web/sites	Accepted	2017-10-08T03:40:2...

Data

Data and application development go hand in hand. It's no wonder that these four data tips were the best of all time for developers. In this section, we'll discover an easy way to configure backups to create copies of your content, configuration, and database. We'll also take a look at how to work with streams in Azure Blob storage. In addition, you'll learn how to work with our command line and tools with Azure Storage, and discover a data migration tool that you can use to move data into Azure Cosmos DB.

Configure a Backup for your Azure App Service and Database

Most folks don't realize how easy it is to configure a backup copy of your Azure App Service to ensure you have restorable archive copies of your app and database. In order to take advantage of this, you'll need to log into your Azure account and go to your App Service that you created. Look under **Settings** and you will see **Backup**.

The screenshot shows the 'Backup' section of the Azure App Service settings. At the top, there are three buttons: 'Configure', 'Refresh', and 'Reset'. Below them is a large blue cloud icon with the word 'Backup'. A gear icon indicates that backup is not configured. A message says: 'Backup is not configured. Click here to configure backup for your app.' There is also a link to 'Learn more'.

Open it and select **Configure** and you'll see the following screen:

The screenshot shows the 'Backup Storage' configuration page. It includes sections for 'Backup Storage' (selecting a target container), 'Backup Schedule' (configuring scheduled backups), and 'Backup Database' (selecting databases to include). On the left, a sidebar provides a link to learn more about Azure App Service.

Backup Storage
Select the target container to store your app backup.
Storage Settings
Storage not configured

Backup Schedule
Configure the schedule for your app backup.

Scheduled backup On Off

* Backup Every Days

* Start backup schedule from UTC -07:00

* Retention (Days)

Keep at least one backup No Yes

Backup Database
Select the databases you to include with your backup. The backup database list is based on the apps configured connection strings.

INCLUDE IN BACKUP	CONNECTION STRING NAME	DATABASE TYPE
<input checked="" type="checkbox"/> Included	DefaultConnection	Sql Database

Create storage account

* Name
myappstoragemvc 
.core.windows.net

Performance 
 Standard Premium

Replication 
 Locally-redundant storage (LRS) 

* Location
 East US 

I provided a Name, selected the Standard under the Performance option, and used Locally-Redundant Storage (LRS) for Replication. For the location field, please use the closest location nearest you.



Quick Tip You can type **help** from the console window for a list of available commands.

Containers

myappstoragemvc

 Container  Refresh

New container

* Name
mymvcappcontainer 

Public access level 
 Private (no anonymous access) 

OK **Cancel**



Once completed, you can click on the backup and see a feature called **Snapshot** which automatically creates periodic restore points of your app when hosted in a Premium App Service plan. You can even download a zip of the app.

Next, you'll want to make sure that **Scheduled backup** is set to **On**. You'll want to configure the Days and Hours and then the current schedule that it should back up from. I set mine to back up every **seven** days, and starting from now. You'll also want to set the retention and by default it will keep at least one backup. If you have a database, then you can also add it with just a checkmark.

Once everything is set, you can see that the next backup is configured and can either force it manually or restore from an existing backup with just a visit to the Azure Portal. You typically want to use "manual" restore when you want to look at your backup at the current point in time vs "restore" a backup at a different time that occurred in the past.

 Snapshot (Preview)

Snapshots automatically create periodic restore points of your app when hosted in a Premium App Service plan.

Backup Id
3077

Status
Created

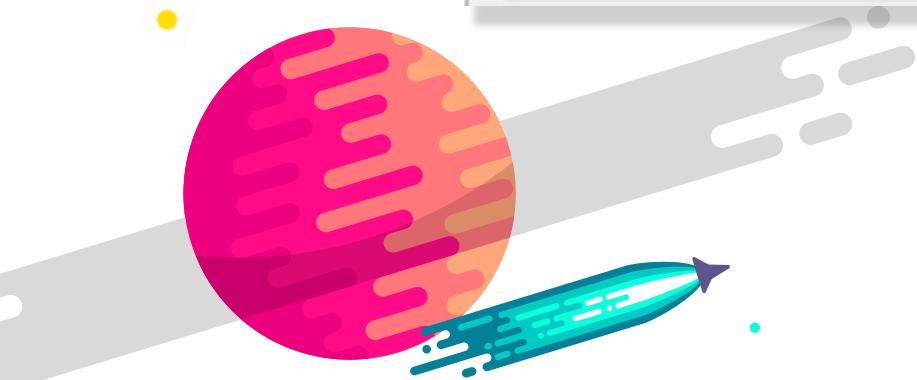
Created Time
Wednesday, October 4, 2017, 10:47:27 PM PDT

Download Backup Zip

Zip contains the backup of an app

[Download](#) 

Size Of Backup
0MB





Learn more about Azure Cosmos DB [here](#)

Using the Data Migration Tool with Cosmos DB

Migrating data from one format to another is a common task for application developers (even if it is just for testing). I was recently building out an API and needed to dump some data into Cosmos DB. The tool that made short work of this was the [Azure DocumentDB Data Migration Tool](#). In my case, I needed to dump a large JSON file into Cosmos DB. Here is how I did it.



Grab whatever sample file that you'd like to experiment with. For this exercise, I selected a file that is public domain and contains a large set of data.

I'm using the [en_kjv.json](#) JSON file from [here](#)

Now we're ready to begin work!

The Tools + Public Domain Sample Data

Get to work

Ensure you have a Cosmos DB database id and collection. You can learn how to create a Cosmos DB by going to <https://docs.microsoft.com/en-us/azure/cosmos-db/>. I'm using the following:

Add Collection

* Database id: bible

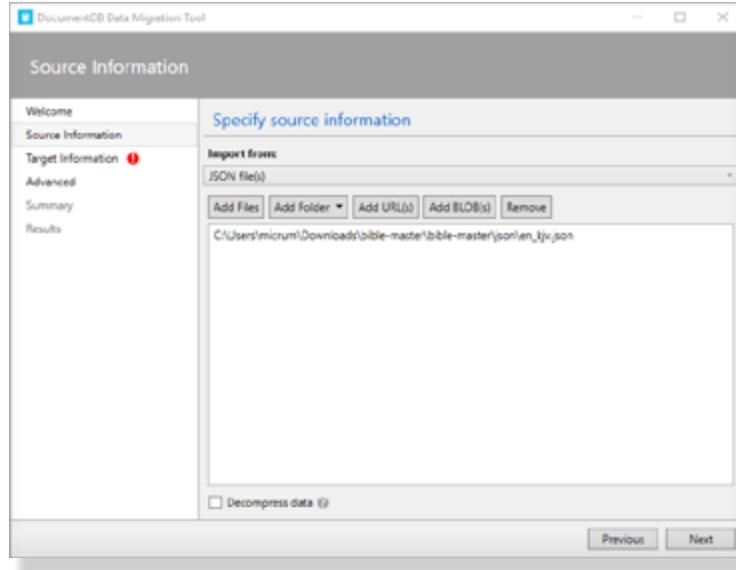
* Collection Id: verses

* Storage capacity: Fixed (10 GB) Unlimited

* Throughput (400 - 10,000 RU/s): 400

Estimated spend (USD): \$0.032 hourly / \$0.77 daily.
Choose unlimited storage capacity for more than 10,000 RU/s.

Open the Data Migration Tool and under **Source Information**, point to the local JSON file as shown below.

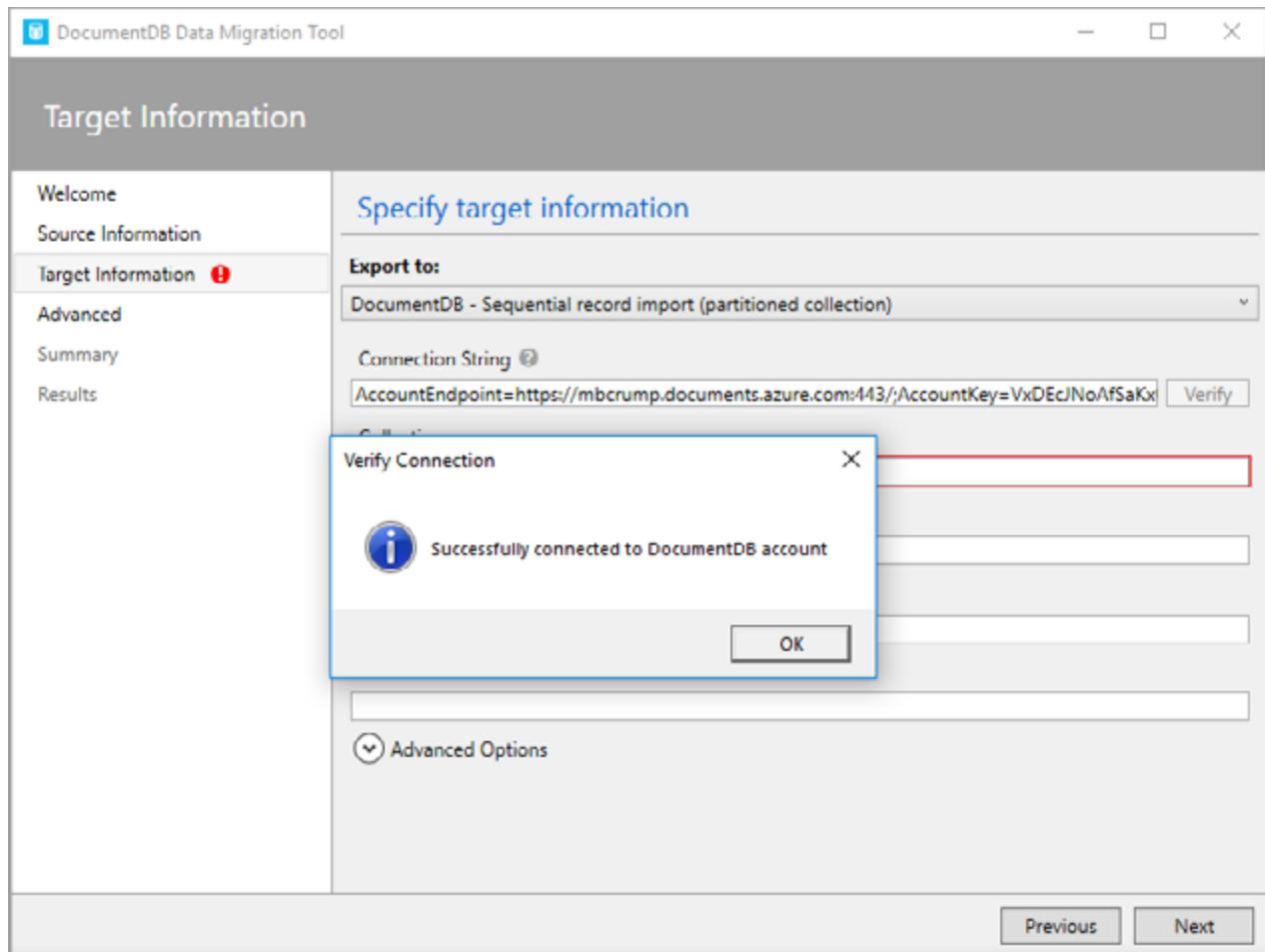


Go to **Keys** (inside your Cosmos DB blade in the portal) to copy the **Primary Connection String**.

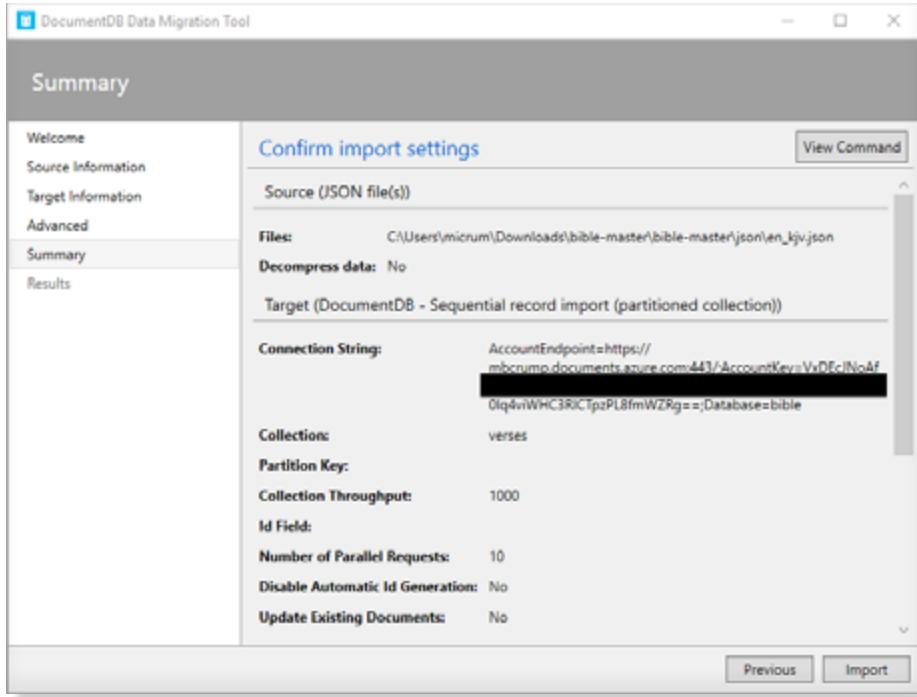
A screenshot of the 'Keys' section in the Azure portal. It shows two tabs: 'Read-write Keys' (selected) and 'Read-only Keys'. Under 'Read-write Keys', there are four fields: 'URI' (redacted), 'PRIMARY KEY' (redacted), 'SECONDARY KEY' (redacted), and 'PRIMARY CONNECTION STRING' (redacted). The 'PRIMARY CONNECTION STRING' field is highlighted with a red rectangular box. Below it is a 'SECONDARY CONNECTION STRING' field (redacted) with a blue icon. To the right of each field are small blue icons for copy and refresh.

You'll need to append the Database name to the end of the string.
For example: **Database=bible** will be appended to the string
AccountEndpoint=https://mbcrump.documents.azure.com:443/;Account-Key=VxDEcJblah==;Database=bible that I copied out of the portal.
Now press **Verify Connection**.

Give it a couple of minutes until you see that it has completed pulling down your code from Git and then go to the new URL of your site. You can find the URL on your overview page. In my case it is, <http://myquizapplication-staging.azurewebsites.net/>

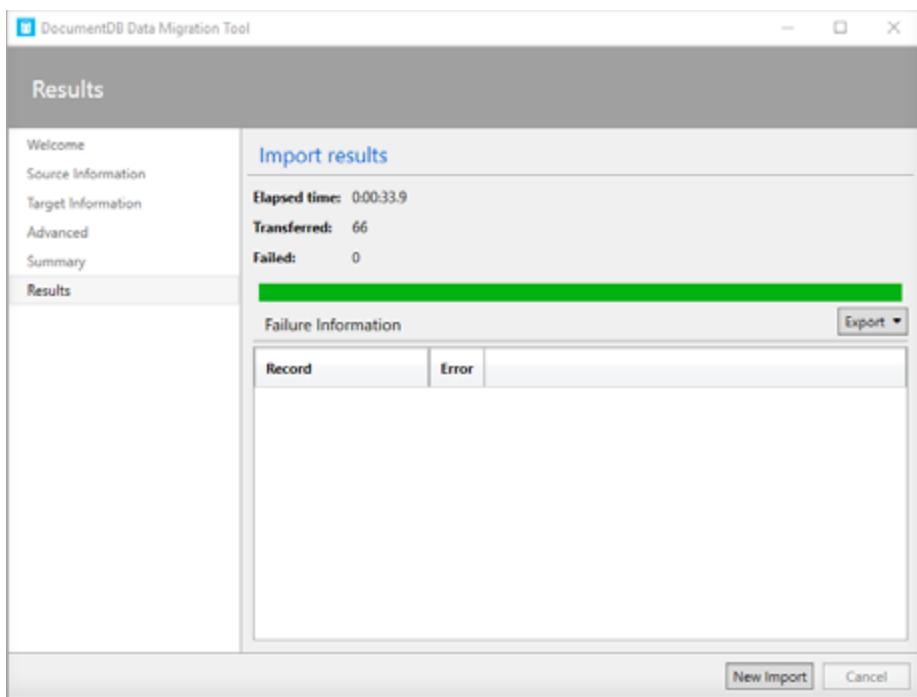


You'll need to add the **Collection** and in my case it is **verses**. We'll take the defaults on the next two screens and you'll finally see a **Confirm import settings** page.



You can even click on [View Command](#) to see the command that will be used to migrate your data. This is helpful to just learn the syntax.

You'll finally see the Import has completed with 66 transferred.



If you go back to the Azure Portal, open Cosmos DB, and look under [Data Explorer](#), you'll see the data has been imported successfully into our collection.

The screenshot shows the Azure portal interface. At the top, there are several navigation links: New Collection, New SQL Query, New Stored Procedure, New UDF, New Trigger, Delete Collection, Delete Database, and Feedback. Below the header, the 'COLLECTIONS' section is visible, showing a tree structure with 'bible' expanded to show 'verses'. Under 'verses', there are 'Documents', 'Scale & Settings', 'Stored Procedures', 'User Defined Functions', and 'Triggers'. The main area displays a 'Documents' view with a table. The table has columns for 'id' and a preview of the document content. One document is selected, showing its full JSON content. The JSON is a list of chapters from the Exodus book of the Bible, starting with chapter 1 and ending with chapter 22.

```

1   "abbrev": "ex",
2   "book": "Exodus",
3   "chapters": [
4     {
5       "1": {
6         "1": "Now these (are) the names of the children of Israel, i",
7         "2": "Reuben, Simeon, Levi, and Judah,",
8         "3": "Issachar, Zebulun, and Benjamin,",
9         "4": "Dan, and Naphtali, Gad, and Asher.",
10        "5": "And all the souls that came out of the loins of Jacob",
11        "6": "And Joseph died, and all his brethren, and all that ga",
12        "7": "And the children of Israel were fruitful, and increase",
13        "8": "Now there arose up a new king over Egypt, which knew n",
14        "9": "And he said unto his people, Behold, the people of the",
15        "10": "Come on, let us deal wisely with them; lest they multi",
16        "11": "Therefore they did set over them taskmasters to affl",
17        "12": "But the more they afflicted them, the more they multi",
18        "13": "And the Egyptians made the children of Israel to serv",
19        "14": "And they made their lives bitter with hard bondage, i",
20        "15": "And the king of Egypt spake to the Hebrew midwives, c",
21        "16": "And he said, When ye do the office of a midwife to th",
22        "17": "But the midwives feared God, and did not as the king",
23        "18": "And the king of Egypt called for the midwives, and sa",
24        "19": "And the midwives said unto Pharaoh, Because the Hebrew",
25        "20": "Therefore God dealt well with the midwives: and the p",
26        "21": "And it came to pass, because the midwives feared God",
27        "22": "And Pharaoh charged all his people, saying, Every son"
28      },
29    },
30  },
31  {
32    "2": {
33      "1": "And there went a man of the house of Levi, and took (t",
34      "2": "And the woman conceived, and bare a son: and when she",
35      "3": "And when she could not longer hide him, she took for i",
36      "4": "And his sister stood afar off, to wit what would be do",
37      "5": "And the daughter of Pharaoh came down to wash (herseli",
38      "6": "And when she had opened {it}, she saw the child: and",
39      "7": "Then said his sister to Pharaoh&x27;s daughter, Shall"
40    }
41  }

```



Learn more about our variety of data options [here](#)

Uploading and Downloading a Stream into an Azure Storage Blob

Azure Storage is described as a service that provides storage that is available, secure, durable, scalable, and redundant. Azure Storage consists of **1) Blob storage**, **2) File Storage**, and **3) Queue storage**. In this tip, we'll take a look at how to upload and download a stream into an Azure Storage Blob with C#.



If you need help setting up a project for the code below then go [here](#)

Upload a File

Now that we've created the Azure Storage Blob Container, we'll upload a file to it. We'll build off our last code snippet and add the following lines of code to upload a file off our local hard disk:

```
static void Main(string[] args)
{
    var storageAccount =
CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("StorageConnection"))
);
    var myClient = storageAccount.CreateCloudBlobClient();
    var container = myClient.GetContainerReference("images-backup");
    container.CreateIfNotExists(BlobContainerPublicAccessType.Blob);

//lines modified
    var blockBlob = container.GetBlockBlobReference("mikepic.png");
    using (var fileStream = System.IO.File.OpenRead(@"c:\mikepic.png"))
    {
        blockBlob.UploadFromStream(fileStream);
    }
//lines modified

    Console.ReadLine();
}
```

If we switch over to our Storage Account and navigate inside the container, we'll see our new file has been added:

NAME	MODIFIED	BLOB TYPE	SIZE	LEASE STATE	...
mikepic.png	1/1/2018, 12:34:21 PM	Block blob	182.88 KiB	Available	...

Download a File

Now that we've uploaded a file to the Azure Storage Blob Container, we'll download a file from it.

We'll build off our last code snippet and add the following lines of code to download a file from our local hard disk and give it new name:

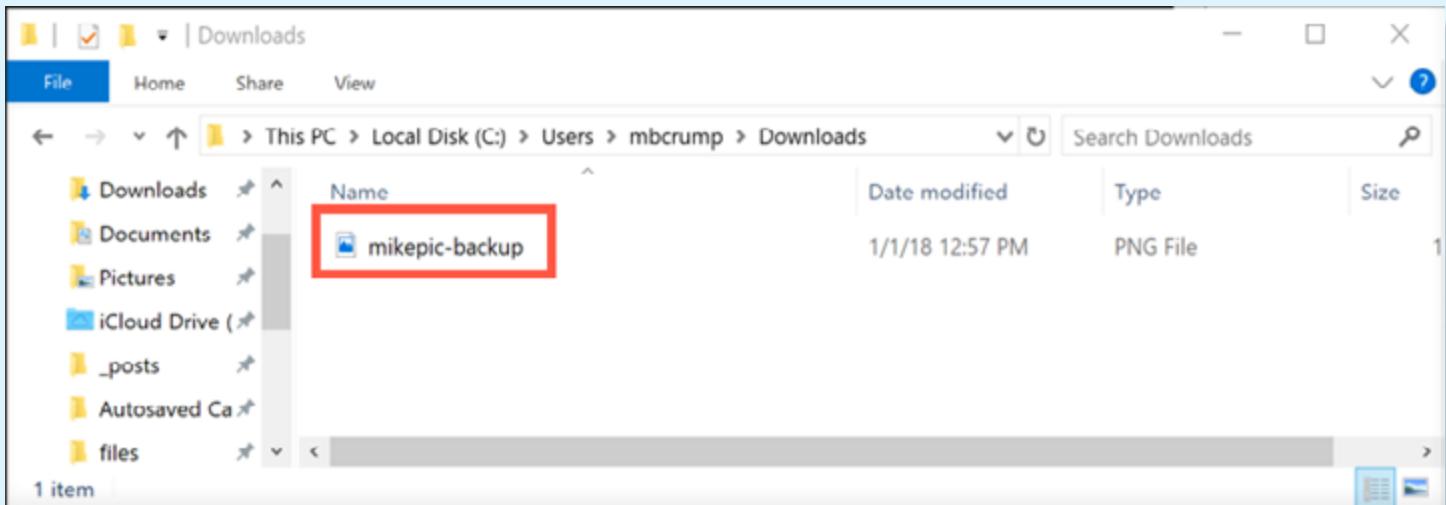
```
static void Main(string[] args)
{
    var storageAccount =
CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("StorageConnection"))
;
    var myClient = storageAccount.CreateCloudBlobClient();
    var container = myClient.GetContainerReference("images-backup");
    container.CreateIfNotExists(BlobContainerPublicAccessType.Blob);

//lines modified
    var blockBlob = container.GetBlockBlobReference("mikepic.png");
    using (var fileStream =
System.IO.File.OpenWrite(@"C:\Users\mbcrump\Downloads\mikepic-backup.png"))
    {
        blockBlob.DownloadToStream(fileStream);
    }
//lines modified

    Console.ReadLine();
}
```

Note that are now using the [OpenWrite](#) method and specifying a new name. We are also taking advantage of the [DownloadToStream](#) method. If we run the application, our new file should be in the downloads folder.





Working with AzCopy and Azure Storage



What is AzCopy?

AzCopy is a command line utility designed for copying data to/from Microsoft Azure Blob, File, and Table storage, using simple commands designed for optimal performance.

You can copy data between a file system and a storage account, or between storage accounts.
(courtesy of docs)

You can easily work with AzCopy to manipulate Azure Storage containers and more. In this tip, we'll explore AzCopy in the context of Azure Storage containers.

For this example, I'm going to use Windows. After I downloaded and installed the utility, I navigated inside my command prompt to the following folder **%ProgramFiles(x86)%\Microsoft SDKs\Azure\AzCopy** and ran the **azcopy.exe** command to ensure everything was working properly.



```
C:\Windows\System32\cmd.exe
C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy>azcopy
AzCopy 7.1.0 Copyright (c) 2017 Microsoft Corp. All Rights Reserved.
-----
# AzCopy is designed for high-performance uploading, downloading, and copying
data to and from Microsoft Azure Blob, File, and Table storage.

# Command Line Usage:
  AzCopy /Source:<source> /Dest:<destination> [options]

# Options:
  [/SourceKey:] [/DestKey:] [/SourceSAS:] [/DestSAS:] [/V:] [/Z:] [/@:] [/Y]
  [/NC:] [/SourceType:] [/DestType:] [/S] [/Pattern:] [/CheckMD5] [/L] [/MT]
  [/XN] [/X0] [/A] [/IA] [/XA] [/SyncCopy] [/SetContentType] [/BlobType:]
  [/Delimiter:] [/Snapshot] [/PKRS:] [/SplitSize:] [/EntityOperation:]
  [/Manifest:] [/PayloadFormat:]

-----
For AzCopy command-line help, type one of the following commands:
# Detailed command-line help for AzCopy      ---  AzCopy /?
# Detailed help for any AzCopy option        ---  AzCopy /?:SourceKey
# Command line samples                      ---  AzCopy /?:Sample
You can learn more about AzCopy at http://aka.ms/azcopy

C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy>
```

You can download either the latest version of AzCopy on [Windows](#) or [Linux](#).

You may be wondering if you need to do the device login as we did with the Azure CLI. The answer is no, we'll be using our Azure Storage Access Key.



Getting the Azure Storage Access Key

Go ahead and open the Azure Portal and navigate to the Azure Storage account that we worked with [earlier](#).

Look under **Settings**, then **Access Keys** and copy the key1.

The screenshot shows the 'Access keys' section of the Azure Storage account 'mbcrumpsa'. On the left, there's a sidebar with links like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. Under 'SETTINGS', 'Access keys' is selected and highlighted in blue. The main area displays two access keys:

NAME	KEY	CONNECTION STRING
key1	TWZajZIBijqEEo5UKQ+Ai5SHNiX0UxBy9S+KTH34... (highlighted with a red box)	DefaultEndpointsProtocol=https;AccountName=mbcrumpsa...
key2	d/q08eKSh6MMKosIUQbwIhwq4rZnQhz1emyYKQ...	DefaultEndpointsProtocol=https;AccountName=mbcrumpsa...

Store the key1 somewhere that you can retrieve it again.

Kick the tires with a couple of commands.

We can easily download a file from our Azure Storage Blob Container that we've been working with by using the following command:

```
AzCopy /Source:https://mbcrumpsa.blob.core.windows.net/images-backup  
/Dest:C:\mytest /SourceKey:thekeyyoucopiedearlier /Pattern:"mikepic.png"
```





Keep in mind: The main difference between these two commands is the use of **SourceKey** for downloading and **DestKey** for uploading. The key that is being used is identical (named key1 from the example above).

We can do the reverse and upload a file from our hard disk to Azure Storage Blob Container with the following command:

```
AzCopy /Source:C:\mytest  
/Dest:https://mbcrumpsa.blob.core.windows.net/images-backup  
/DestKey:thekeyyoucopiedearlier /Pattern:"mikepicnew.png"
```

Finally, you can copy from one Azure Storage account to another one with the following command:

```
AzCopy /Source:https://mbcrumpsa.blob.core.windows.net/images-backup  
/Dest:https://mbcrumpsa.blob.core.windows.net/images  
/SourceKey:thekeyyoucopiedearlier /DestKey:thekeyyoucopiedearlier  
/Pattern:"mikepicnew.png"
```

In this case, I am copying a file named **mikepicnew.png** from **images-backup** to **images** and then I'll refresh the container.

The screenshot shows the Azure Storage Explorer interface on the left and a Command Prompt window on the right.

Azure Storage Explorer (Left):

- Essentials:** Shows containers **images** and **images-backup**.
- Location:** **images**
- Blobs:** Shows a single blob **mikepic.png** with details:

NAME	MODIFIED	BLOB TYPE	SIZE	LEASE STATE
mikepic.png	12/30/2017, 10:36:40 AM	Block blob	182.88 KB	Available

Command Prompt (Right):

```
C:\Windows\System32\cmd.exe  
C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy>
```

Serverless

In this set of tips, I've pulled out 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.



[Back to Table of Contents](#)

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

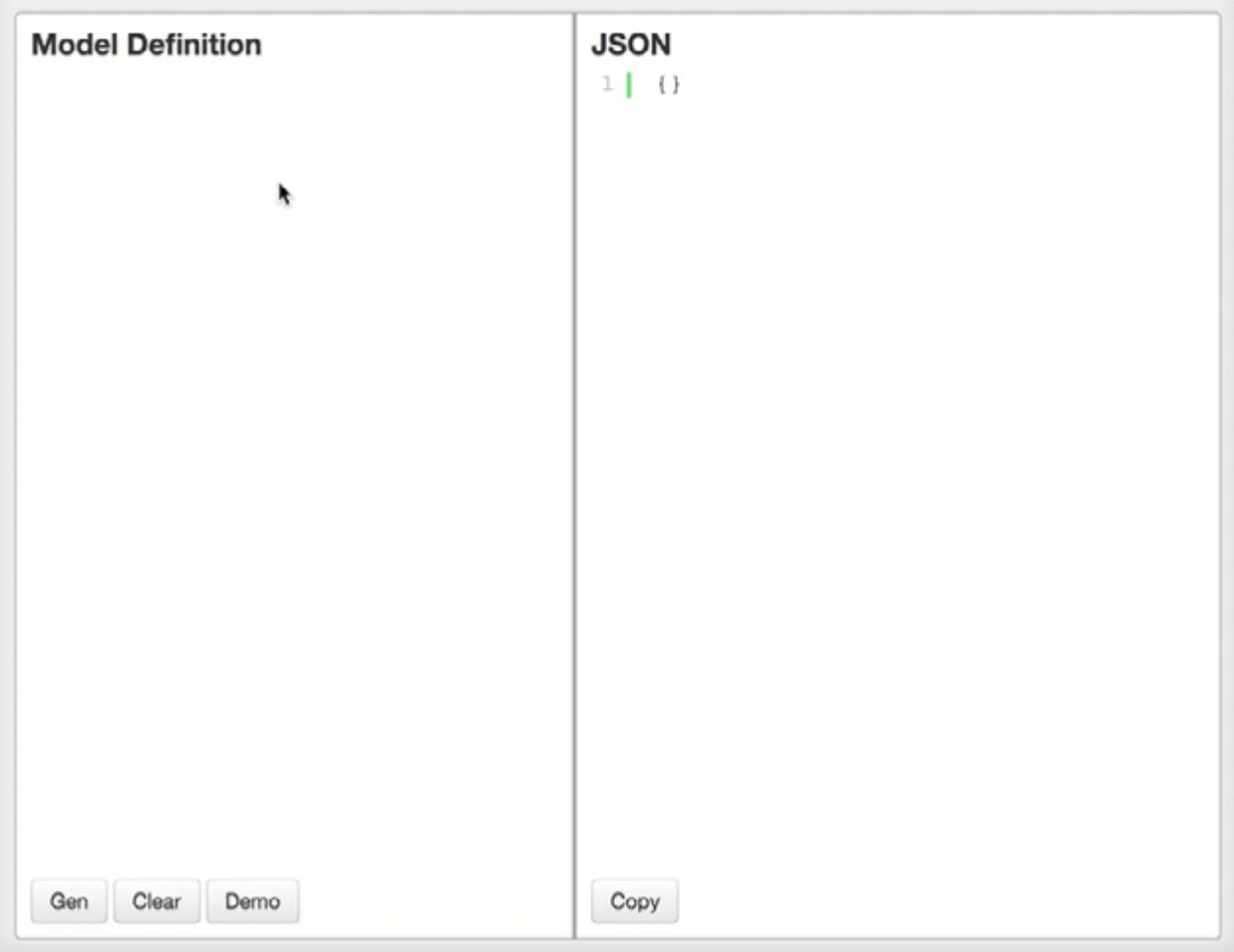
Now that you have your mailbox created and the parser engine knows what data to extract, we can connect the app to the [Zapier Editor](#). But first let's review the pieces of data that we wanted to extract and why.

- 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.



The screenshot shows a web-based tool for generating JSON schemas. The interface is divided into two main sections: 'Model Definition' on the left and 'JSON' on the right. The 'JSON' section contains a single character '1' followed by a cursor and an empty brace '{}'. At the bottom of the interface, there are three buttons: 'Gen', 'Clear', and 'Demo' on the left, and a 'Copy' button on the right. A small orange dot is visible on the far left edge of the slide.

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.

The screenshot shows the jsonschema.net interface. On the left, there's a JSON editor with a red box highlighting the input JSON. The input JSON is:

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

On the right, the generated JSON schema is displayed in a code editor:

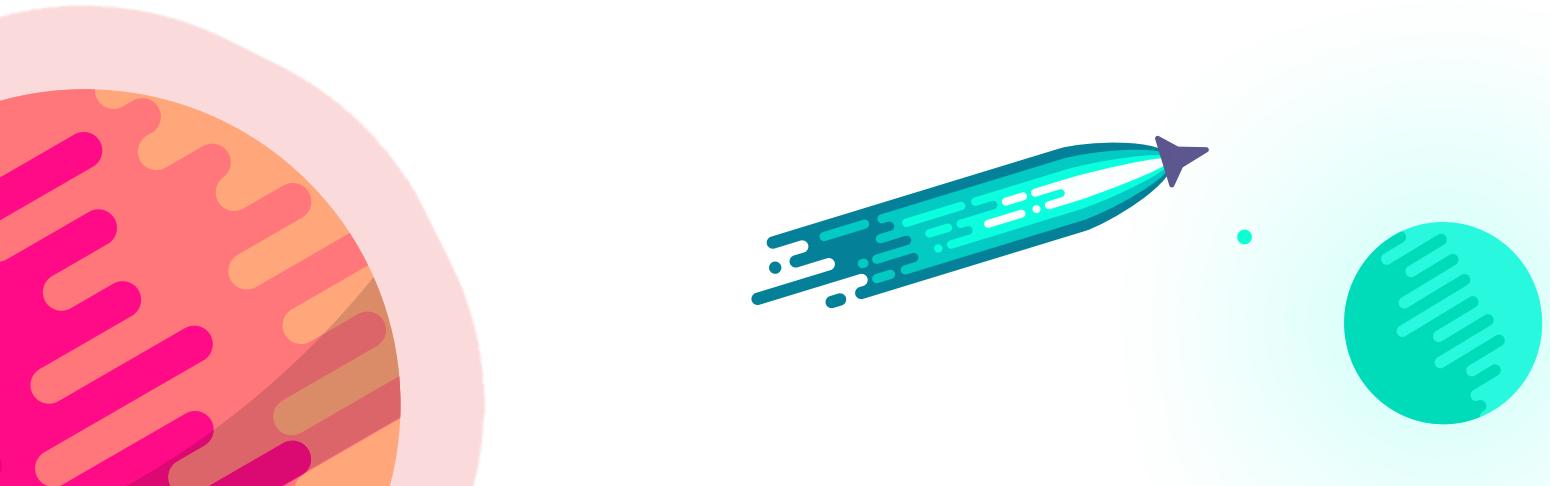
```
1 {  
2   "$$schema": "http://json-schema.org/draft-06/schema#",  
3   "definitions": {},  
4   "id": "http://example.com/example.json",  
5   "properties": {  
6     "csv": {  
7       "id": "/properties/csv",  
8       "type": "string"  
9     },  
10    "filename": {  
11      "id": "/properties/filename",  
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 }
```

Below the JSON editor are two buttons: "RESET" and "SUBMIT".

```
{  
  "$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"  
}
```

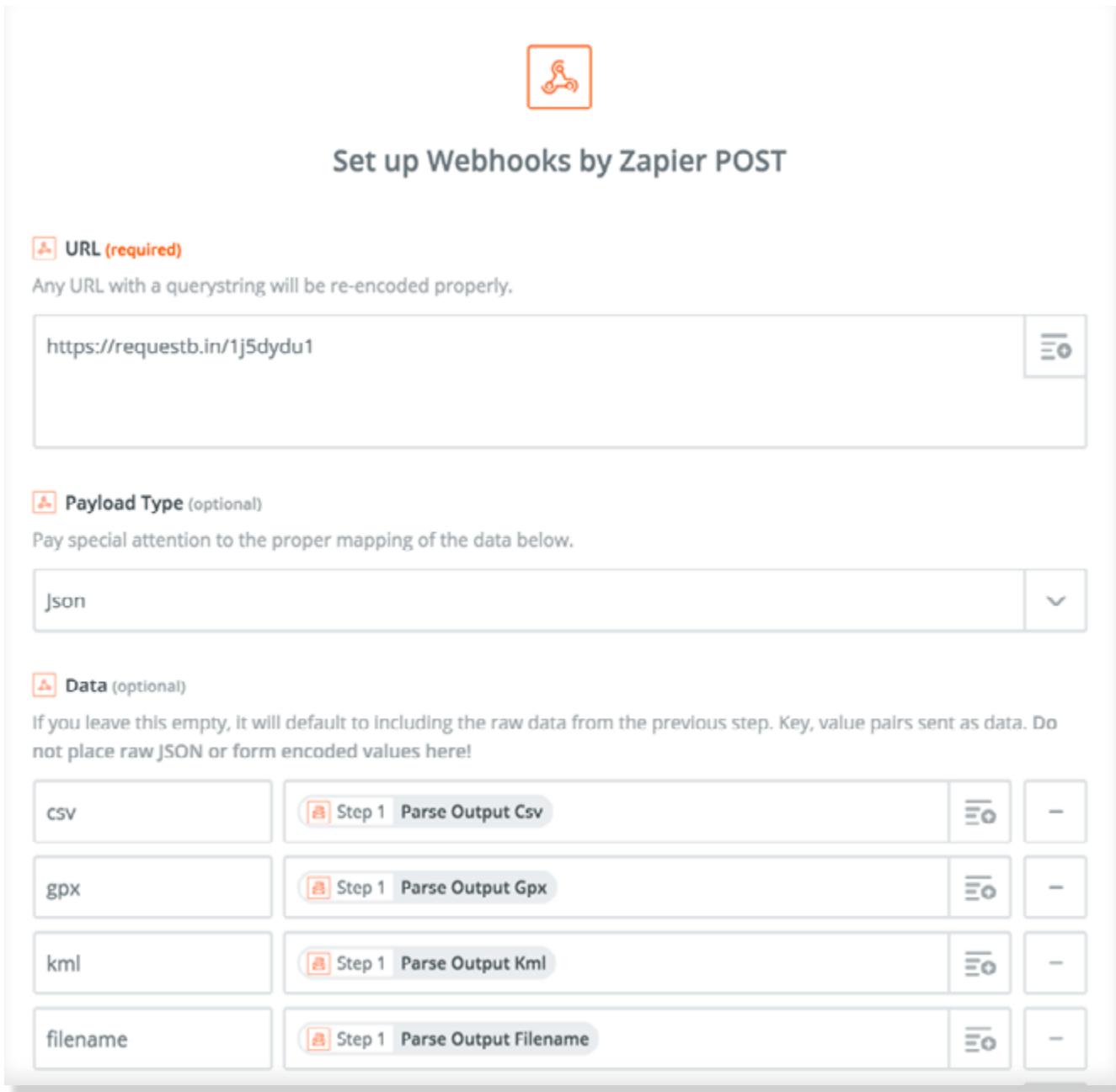
Too easy! Now head over to the [Zapier Editor](#) and create a new app.

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:



The screenshot shows the configuration interface for a 'Webhooks by Zapier POST' action. At the top, there's a placeholder icon for a logo or app icon. Below it, the title 'Set up Webhooks by Zapier POST' is centered.

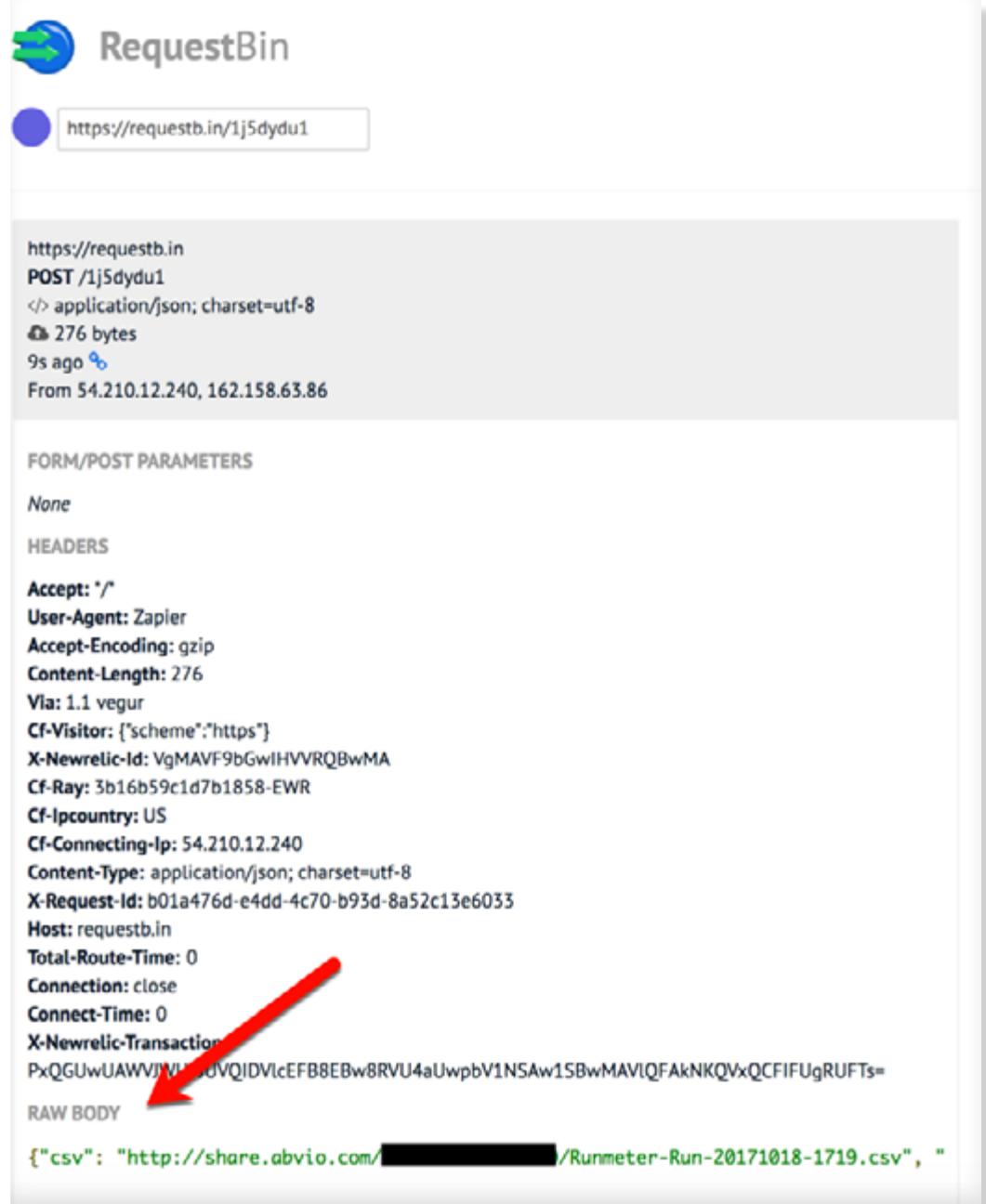
URL (required)
Any URL with a querystring will be re-encoded properly.
 More options

Payload Type (optional)
Pay special attention to the proper mapping of the data below.
 More options

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	More options	-
gpx	Step 1 Parse Output Gpx	More options	-
kml	Step 1 Parse Output Kml	More options	-
filename	Step 1 Parse Output Filename	More options	-

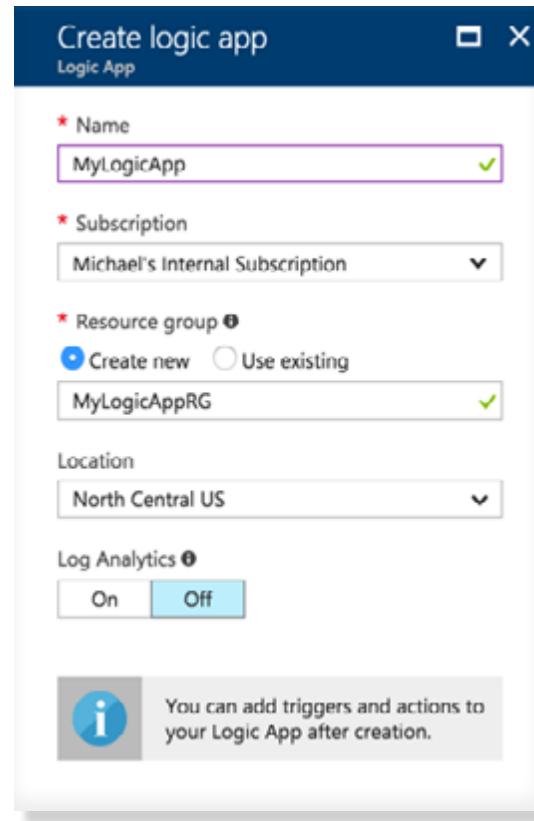
Go ahead and save and run the test. After you switch over to your requestb.in you should see the output that matches the parsed data from the email.



The screenshot shows the RequestBin interface. At the top, there's a blue circular icon with two arrows and the text "RequestBin". Below it is a purple circular icon with a white question mark and the URL "https://requestb.in/1j5dydu1". The main content area has a grey header bar with the URL "https://requestb.in" and the ID "1j5dydu1". The bar also shows "POST /1j5dydu1", "application/json; charset=utf-8", "276 bytes", "9s ago", and "From 54.210.12.240, 162.158.63.86". Below this is a section titled "FORM/POST PARAMETERS" with the value "None". Under "HEADERS", there is a long list of HTTP headers. A red arrow points from the bottom of the "HEADERS" list down towards the "RAW BODY" section. The "RAW BODY" section contains the JSON object: {"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.

Start with a common trigger
Pick from one of the most commonly used triggers, then orchestrate any number of actions using the rich collection of connectors

 When a message is received in a Service Bus queue	 When a HTTP request is received	 When a new tweet is posted	 When a Event Grid event occurs
 Recurrence	 When a new email is received in Outlook.com	 When a new file is created on OneDrive	 When a file is added to FTP server

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 a 'When a HTTP request is received' trigger in Azure Logic Apps. The 'HTTP GET URL' field is highlighted with a red box and contains the URL: <https://prod-21.northcentralus.logic.azure.com:443/workflows/996d879417a04689a1a4fc07dc7...>. Below it, the 'Request Body JSON Schema' field displays a JSON schema definition:

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

Below the schema, there's a link 'Use sample payload to generate schema'. The 'Method' dropdown is set to 'POST'. The 'Relative path' field is empty. At the bottom, there's a 'Hide advanced options ^' link.

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.

The screenshot shows the Zapier interface for setting up a webhook. On the left, under the 'ACTION' section for step 2, 'Webhooks by Zapier' is selected. The 'Edit Template' option is highlighted with a blue background. The main panel displays the configuration for the 'Set up Webhooks by Zapier POST' step. It includes fields for 'URL (required)' containing 'https://prod-43.westus.logic.azure.com:443' and 'Payload Type (optional)' set to 'json'. A note at the bottom states: '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 include your JSON in form encoded unless you have to.'

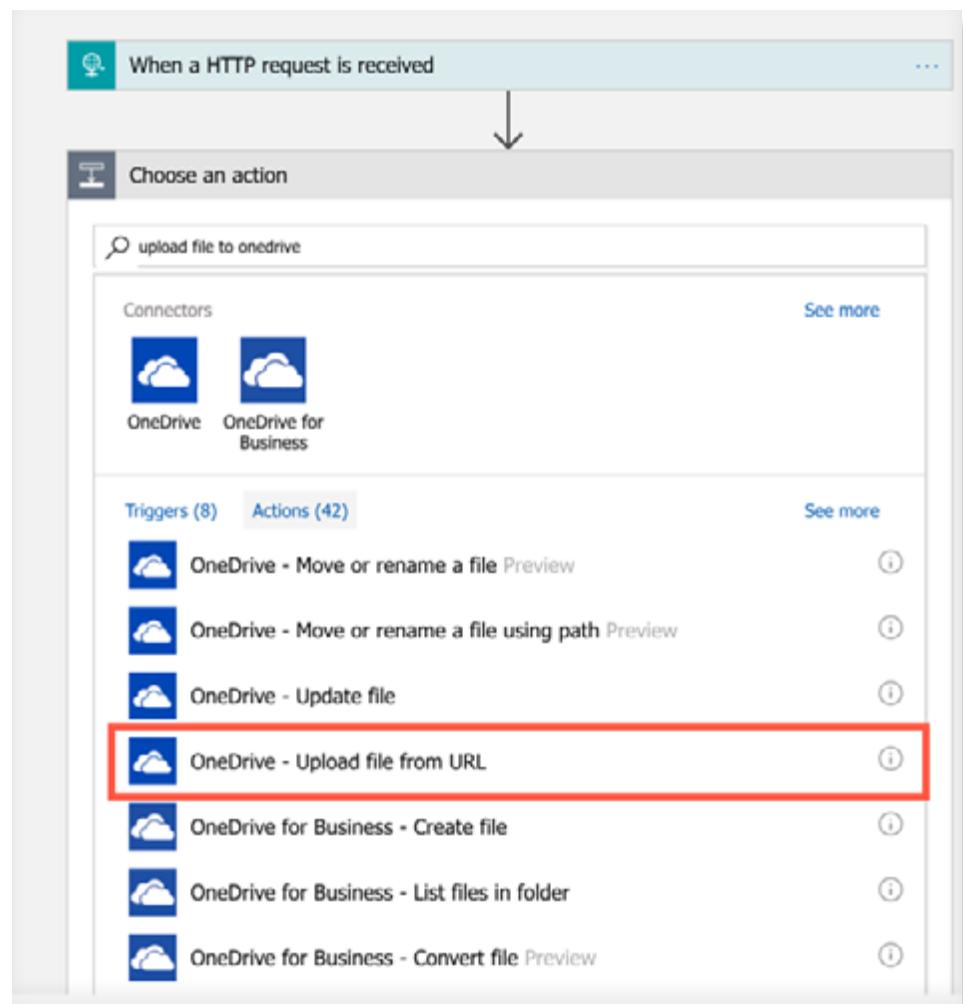
Upload Files from a URL with Azure Logic Apps

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

The screenshot shows the Azure Logic App builder interface. At the top, there is a header bar with a globe icon and the text 'When a HTTP request is received'. Below the header is a blue navigation bar with three items: '+ New step' (highlighted with a white background), 'Add an action', 'Add a condition', and 'More'. The main workspace is currently empty, indicating the start of a new logic app.

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

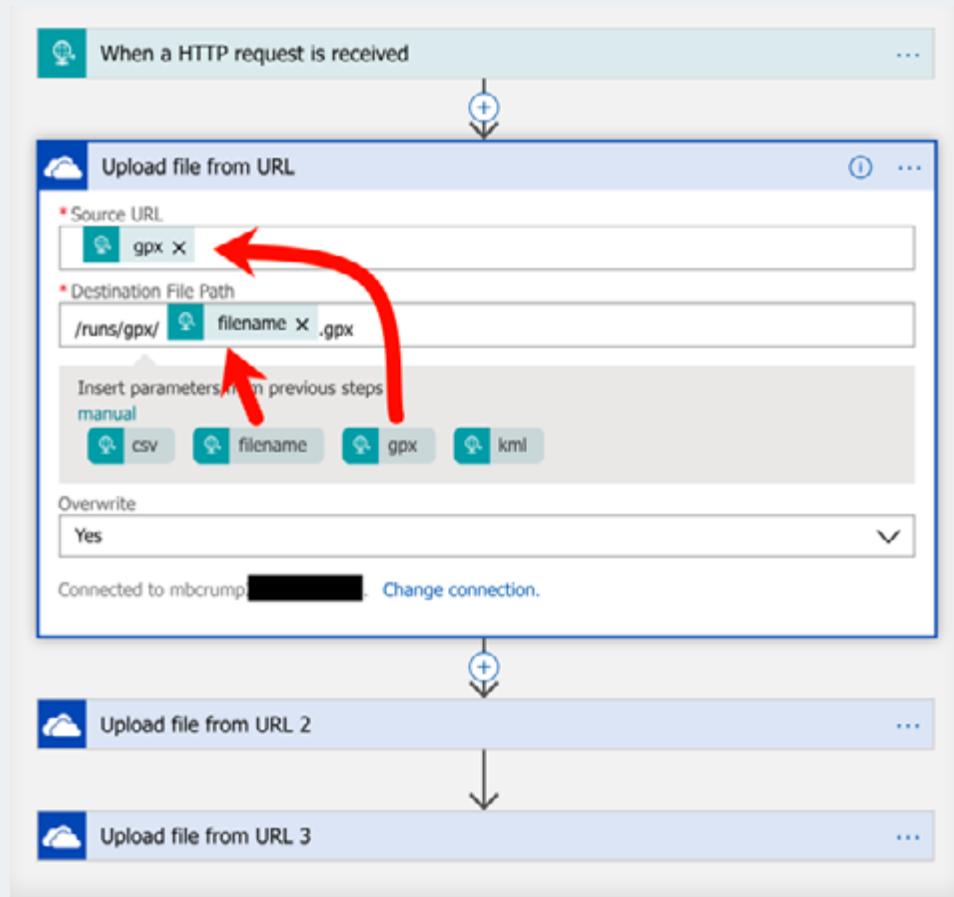
We'll select an **Action** as we want it to run every time vs. a **Condition** which would use "If..then.." logic after the HTTP request comes in. Select **Action** and search for "upload file to onedrive" and you'll see the following is available to use.



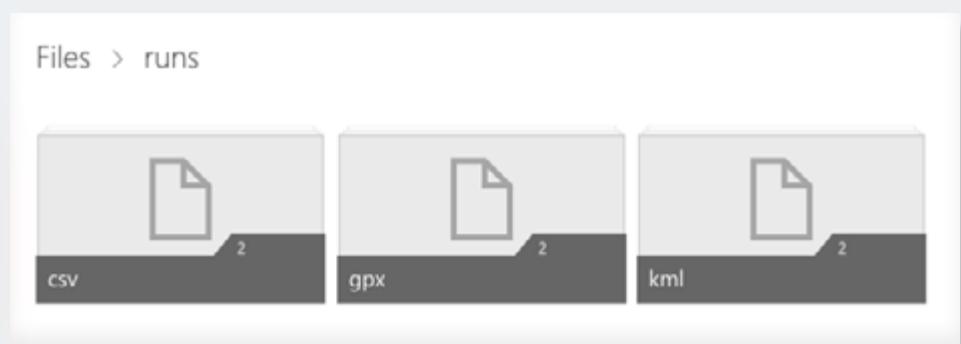
You'll have to sign in to your OneDrive account.

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.



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+ /)

Run Trigger Refresh Edit Delete Disable Update Schema Clone Export

Overview

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 time... 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 time... 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...	10/19... Fired



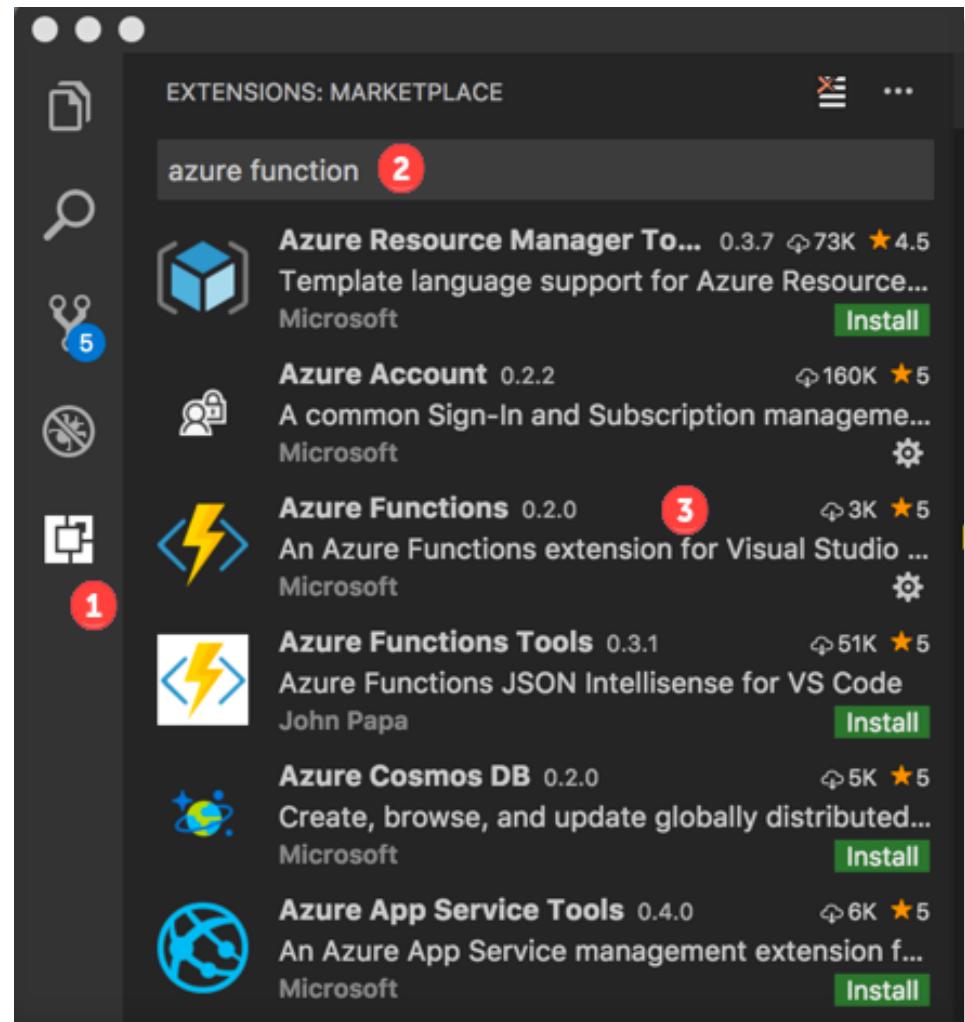
I was able to create the app in less time than it took to write this up!

Success! Our application is working properly.

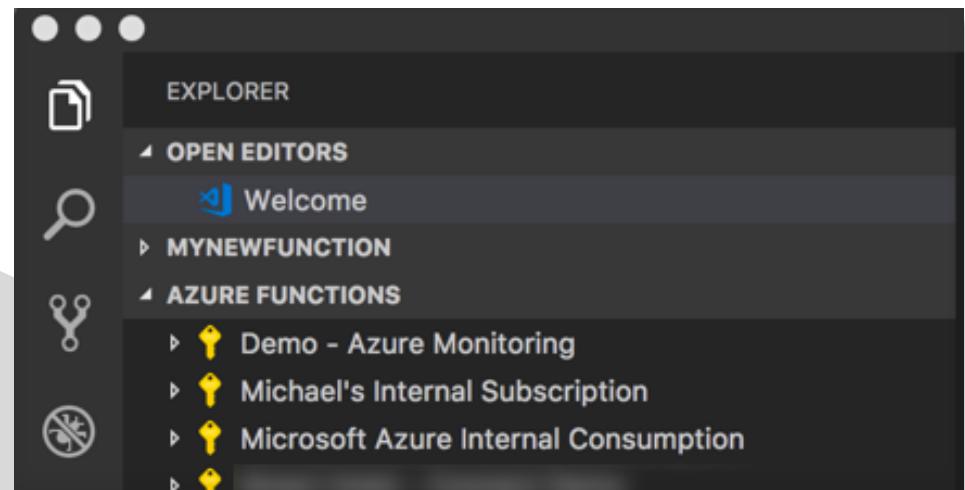
Create an Azure Functions Project with Visual Studio Code

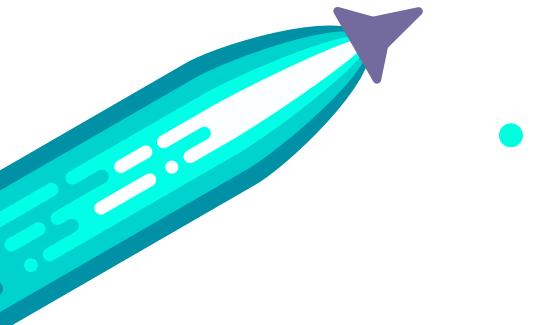
[Visual Studio Code](#) is the best thing since coffee for developers and if you pair it with Azure Functions... well, more awesome happens. In this post, we'll look at adding an Azure Function project to Visual Studio Code.

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 :



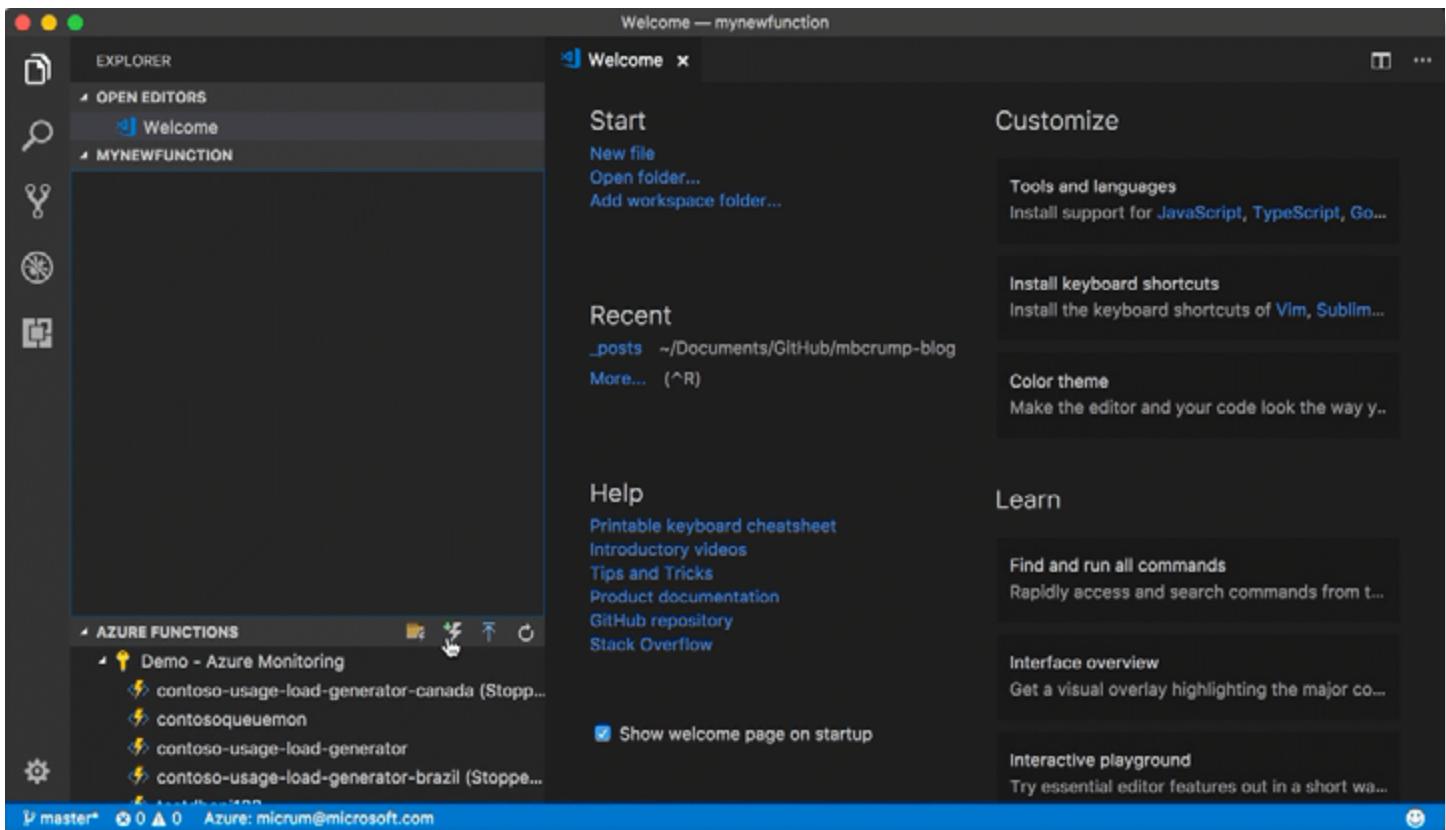
Once installed, you'll need to reload the extension and you should see your subscriptions.





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.



Remember this! You can also add Azure Cloud Shell to Visual Studio Code with this [tip](#)!

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\(Kudu\)](#).

Overview Platform features

Search features

GENERAL SETTINGS

- Function app settings
- Application settings
- Properties
- Backups
- All settings

CODE DEPLOYMENT

- Deployment options
- Deployment credentials

DEVELOPMENT TOOLS

- Logic Apps
- Console
- Advanced tools (Kudu)**
- App Service Editor
- Resource Explorer
- Extensions

NETWORKING

- Networking
- SSL
- Custom domains
- Authentication / Authorization
- Managed service identity
- Push notifications

MONITORING

- Diagnostic logs
- Log streaming
- Process explorer

API

- API definition
- CORS

APP SERVICE PLAN

- App Service plan
- Quotas

RESOURCE MANAGEMENT

- Activity log
- Access control (IAM)
- Tags
- Locks
- Automation script

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

Kudu Environment Debug console Process explorer Tools Site extensions

... / wwwroot + | 2 items | [Home](#) [Logs](#) [Console](#)

Name
HttpTriggerCSharp1
host.json

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](#).

A screenshot of a web browser window. The address bar shows a secure connection to <https://mynewapimc.azurewebsites.net/HttpTriggerCSharp1?code=7CGSgjglgxTaeYrglbUu4nF>. The page content displays an XML response: "This XML file does not appear to have any style information associated with it. The document tree is shown below." followed by the XML code: <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/"> Please pass a name on the query string or in the request body </string>. A red arrow points from the text "Simply restart your Azure Function and now my URL is accessible without [api](#)." to the word "api" in the XML response.

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

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

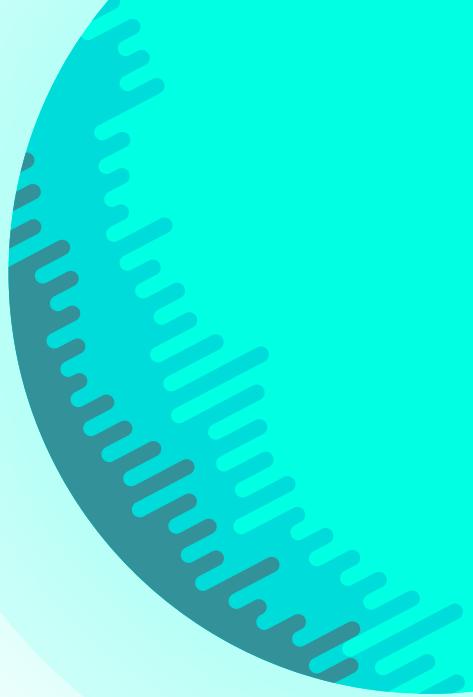
A screenshot of a web browser window. The address bar shows a secure connection to <https://mynewapimc.azurewebsites.net/myroute/HttpTriggerCSharp1?code=7CGSgjglgxTaeYrglbUu4nF>. The page content displays an XML response: "This XML file does not appear to have any style information associated with it. The document tree is shown below." followed by the XML code: <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/"> Please pass a name on the query string or in the request body </string>. A red arrow points from the text "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." to the word "api" in the XML response.

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.



Productivity

If you jumped straight to this section, then you certainly understand the spirit of what I originally wanted to achieve with Azure Tips and Tricks - simply to be more productive with Azure. In this set of tips, I've gone back to the first tip that I ever wrote describing how you can use keyboard shortcuts within the Azure Portal to navigate more effectively. We'll also cover how you can apply tags to your Azure resources to logically organize them by categories. We'll wrap up with using Azure Cloud Shell, which provides an interactive, browser-accessible shell for managing Azure resources, and how you can quickly take advantage of it with Visual Studio Code in the browser or on your local development machine.



Azure Portal Keyboard Shortcuts

Developers love keyboard shortcuts and there are plenty of keyboard shortcuts in the Azure platform. You can see a list by logging into the Azure Portal, clicking on the question mark (or help icon), and selecting Keyboard Shortcuts.

You will see that you have the following keyboard shortcuts available:

Actions

CTRL+ /	Search blade menu items
ALT+SHIFT+Up	Move favorites up
ALT+SHIFT+Down	Move favorites down
G+ /	Search resources (global)
G+N	Create a new resource
G+B	Open the 'More services' pane

Navigation

G+,	Move focus to command bar
G+.	Toggle focus between top bar and side bar

Go to

G+D	Go to dashboard
G+A	Move favorites up
G+R	Move favorites down
G+number	Search resources (global)



Continue checking the site as new ones are being added all the time!

Use Tags to Quickly Organize Azure Resources

You can utilize tags to quickly organize Azure Resources. For example, if you'd like to have a set of Resources for "Production" and another for "Dev", then you can quickly do that.



Remember this!

Tags are user-defined key/value pairs which can be placed directly on a resource or a resource group.

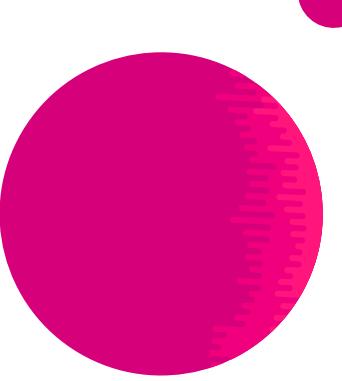
Head over to the Azure Portal and select a service. In my example, I'm going to select a Web App that I want to tag as a Production App. Select the Tags menu and provide a Name and Value as shown below.

The screenshot shows the Azure Portal interface for managing tags. On the left, there's a sidebar with links: Overview, Activity log, Access control (IAM), Tags (which is selected and highlighted in blue), and Diagnose and solve problems. The main area is titled 'Save' and contains an information icon with text about tags. It has fields for 'Name' (set to 'Environment') and 'Value' (set to 'Production'). Below these fields, there's a link to 'No tags'.

I selected **Environment** and gave it the value of **Production**. I then clicked **Save**. I could also do this for other Production resources, and even tag the appropriate ones with **Dev**. I can now take advantage of this ability by going to **More Services**, typing **Tags**, and clicking on the Environment: Production as shown below.

The screenshot shows the 'Tags' blade in the Azure Portal. At the top, it says 'Subscriptions: 2 of 5 selected - Don't see a subscription? Switch directories' and shows a dropdown for '2 subscriptions'. Below this is an information card about tags. The main area is divided into two sections: 'Environment : Dev' and 'Environment : Production'. The 'Production' section is highlighted with a red number '2'. Under 'Environment : Production', there's a table with columns 'NAME' and 'SUBSCRIPTION'. It lists two items: 'mc-webstarter' under 'Visual Studio Enterprise' and 'mvcappdemovslive' under 'Michael's Internal Subscription'. Each item has a three-dot ellipsis button to its right.

NAME	SUBSCRIPTION
mc-webstarter	Visual Studio Enterprise
mvccappdemovslive	Michael's Internal Subscription

- 
1. Results from searching “Tags”
 2. Our Production Environment we just setup
 3. List all the Web Apps with the Production Environment Tag
 4. Pin the Blade to our Azure Portal Main Page

If you pin the blade (by pressing the pin in step 4) you'll see the following on your Azure Portal dashboard:



Recap: Make your life easier by applying tags to your Azure resources to logically organize them by categories.



Environment : Production

You can even interact with **Tags** using Azure CLI 2.0. For example, I can type `az tag list -o json` to list all the tags associated with an account.

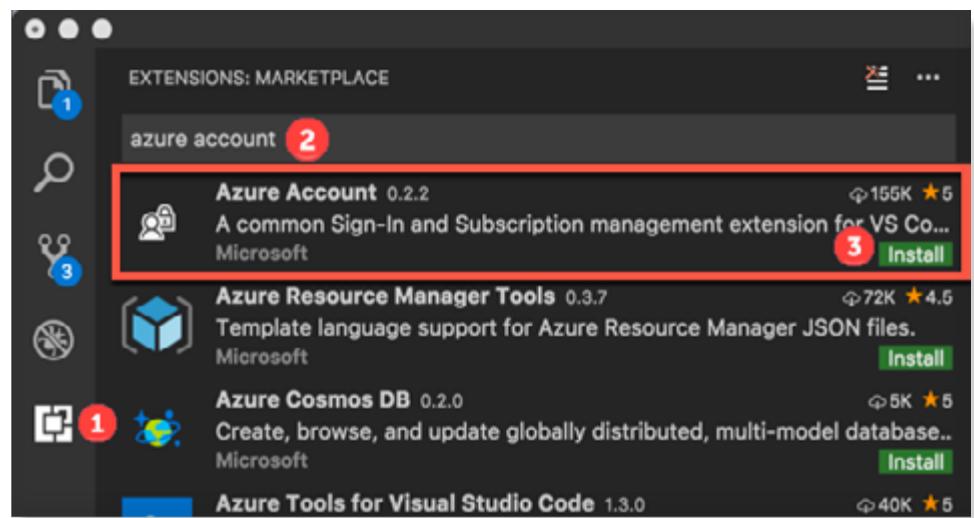
```
michael@Azure:~$ az tag list
[
  {
    "count": {
      "type": "Total",
      "value": 2
    },
    "id": "/subscriptions/c0e5fb0f-7461-4b04-9720-63fe407b1bdb/tagNames/Environment",
    "tagName": "Environment",
    "values": [
      {
        "count": {
          "type": "Total",
          "value": 1
        },
        "id": "/subscriptions/c0e5fb0f-7461-4b04-9720-63fe407b1bdb/tagNames/Environment/tagValues/Dev",
        "tagValue": "Dev"
      },
      {
        "count": {
          "type": "Total",
          "value": 1
        },
        "id": "/subscriptions/c0e5fb0f-7461-4b04-9720-63fe407b1bdb/tagNames/Environment/tagValues/Production",
        "tagValue": "Production"
      }
    ]
  }
]
```



Azure Cloud Shell
is an interactive,
browser-accessible shell for
managing Azure resources.
Linux users can opt for
a Bash experience, while
Windows users can
opt for PowerShell.

Add Azure Cloud Shell to Visual Studio Code

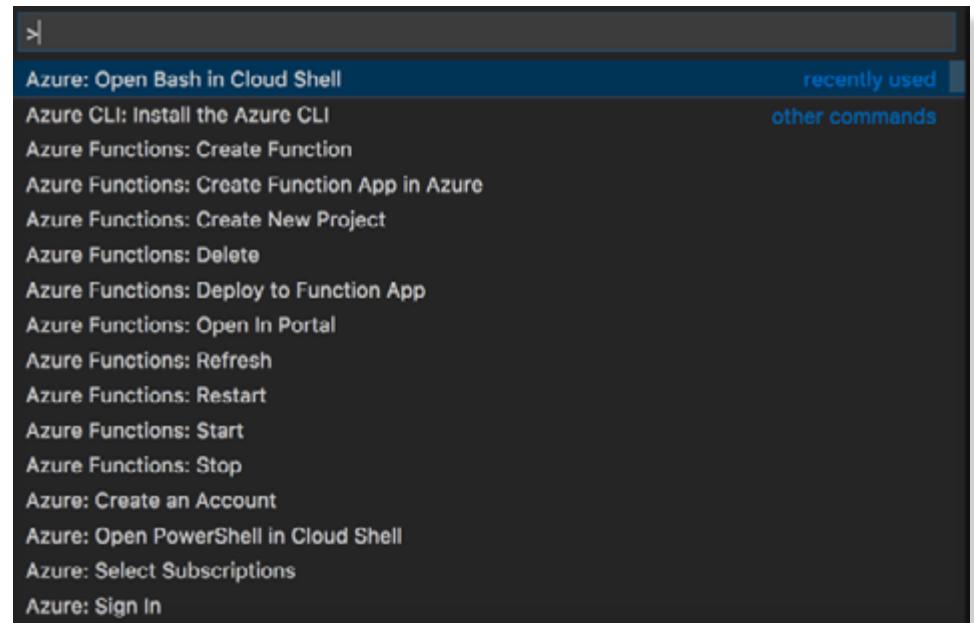
To add Azure Cloud Shell to VS Code, click on Extensions and search for [azure account](#).
Install it as shown below.



Once installed, go to View ->
Command Palette and type [Open Bash in Cloud Shell](#).



Note: You can also open
PowerShell in Cloud Shell
with this extension!





You'll need to sign in first, and Visual Studio Code makes that simple by opening the browser and copying your device authentication code. Once that is complete, you'll see:

Visual Studio Code

You have signed in to the Visual Studio Code application on your device. You may now close this window.

Go back to View -> Command Palette and select [Open Bash in Cloud Shell](#) again and it should spin up as shown below.

The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal output is as follows:

```
2017-11-12-azure-tips-and-tricks49.md — _posts
1 ----
2 layout: post
3 title: "Azure Tips and Tricks Part 49 – Add Azure Cloud Shell to Visual Studio Code"
4 excerpt: "Learn how to add Azure Cloud Shell to Visual Studio Code"
5 tags: [azure, windows, portal, cloud, developers, tipsandtricks]
6 share: true
7 comments: true
8 ---
9
10 ## Intro
11

Connecting terminal...
Welcome to Azure Cloud Shell (Preview)

Type "az" to use Azure CLI 2.0
Type "help" to learn about Cloud Shell

michael@Azure:~$ 
```

Three numbered callouts point to specific elements:

- Callout 1 points to the terminal tab in the bottom navigation bar.
- Callout 2 points to the terminal output showing the connection to Azure Cloud Shell.
- Callout 3 points to the status bar at the bottom, which displays the email address "Azure: micrum@microsoft.com".

Very cool! If you want to see the source code for the app it can be found [here](#).

Quickly Edit Files Within Azure Cloud Shell Using Visual Studio Code That You Know and Love

Did you know that you can access Visual Studio Code within a Cloud Shell instance?

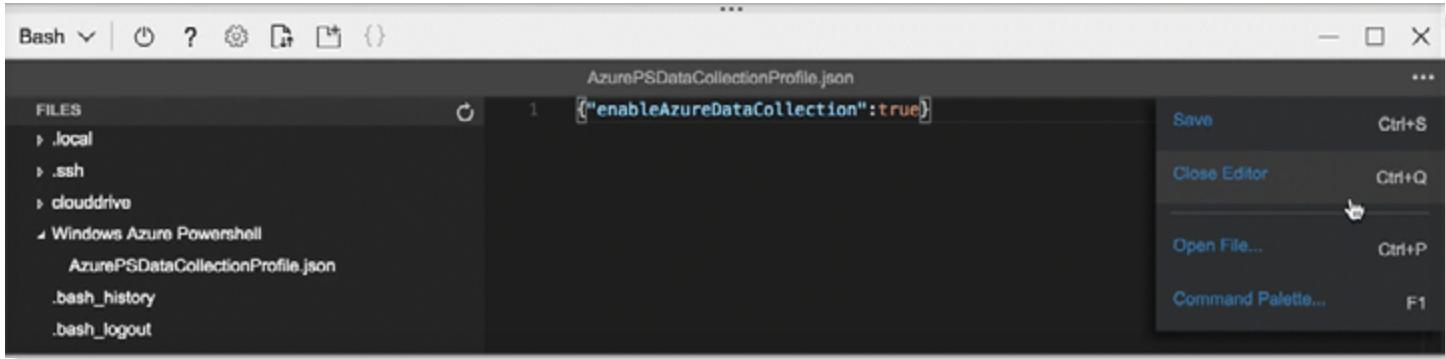
I'm sure by now everyone has used the lovely [Visual Studio Code editor](#) in some application before, but you may not be aware that you can use the editor within Cloud Shell without installing anything. To give this a spin, open up Cloud Shell and type `code .` and you'll see the following:



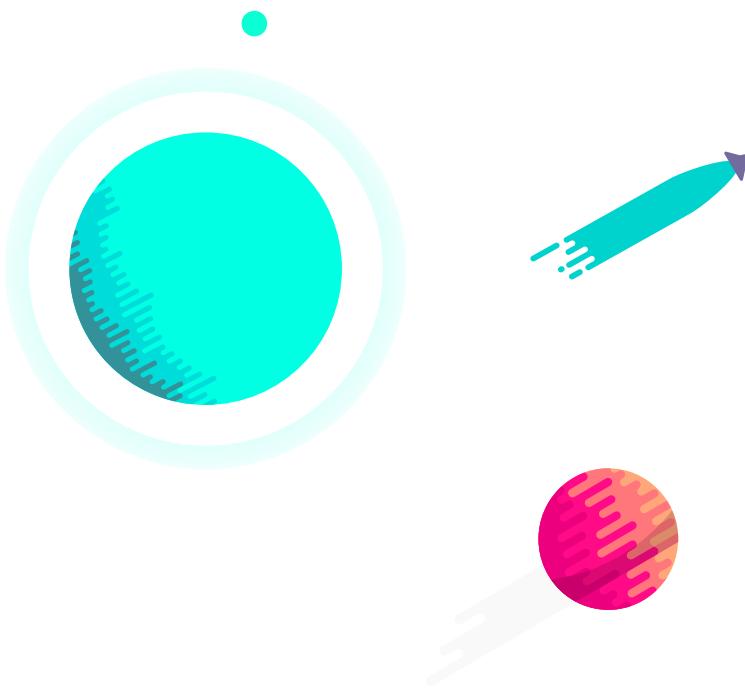
A screenshot of the Azure Cloud Shell terminal window. The title bar says "Bash". The command line shows "michael@Azure:~\$ code ." being typed. The rest of the terminal window is black, indicating the editor is running in the background.

Notice that you can do things such as navigate directories, and also view files with the same syntax used in VS Code. You can easily save and close the editor, open a file outside the current working directory, and open the command palette.

If you open the command palette you'll see a very familiar list of commands that you've probably used in the editor on your desktop.



And since this is based upon the open-source Monaco project that powers Visual Studio Code, you can expect we'll see more features added over time. As of the publication time of this eBook, it automatically includes authorization for pre-installed open source tools like Terraform, Ansible, and InSpec. So what are you waiting for? [Go check out now!](#)



Conclusion

Thanks for reading and I hope that you enjoyed the top tips of Azure Tips & Tricks since the creation of the series. While we've discussed four broad sections that covered web, data, serverless & productivity, 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:

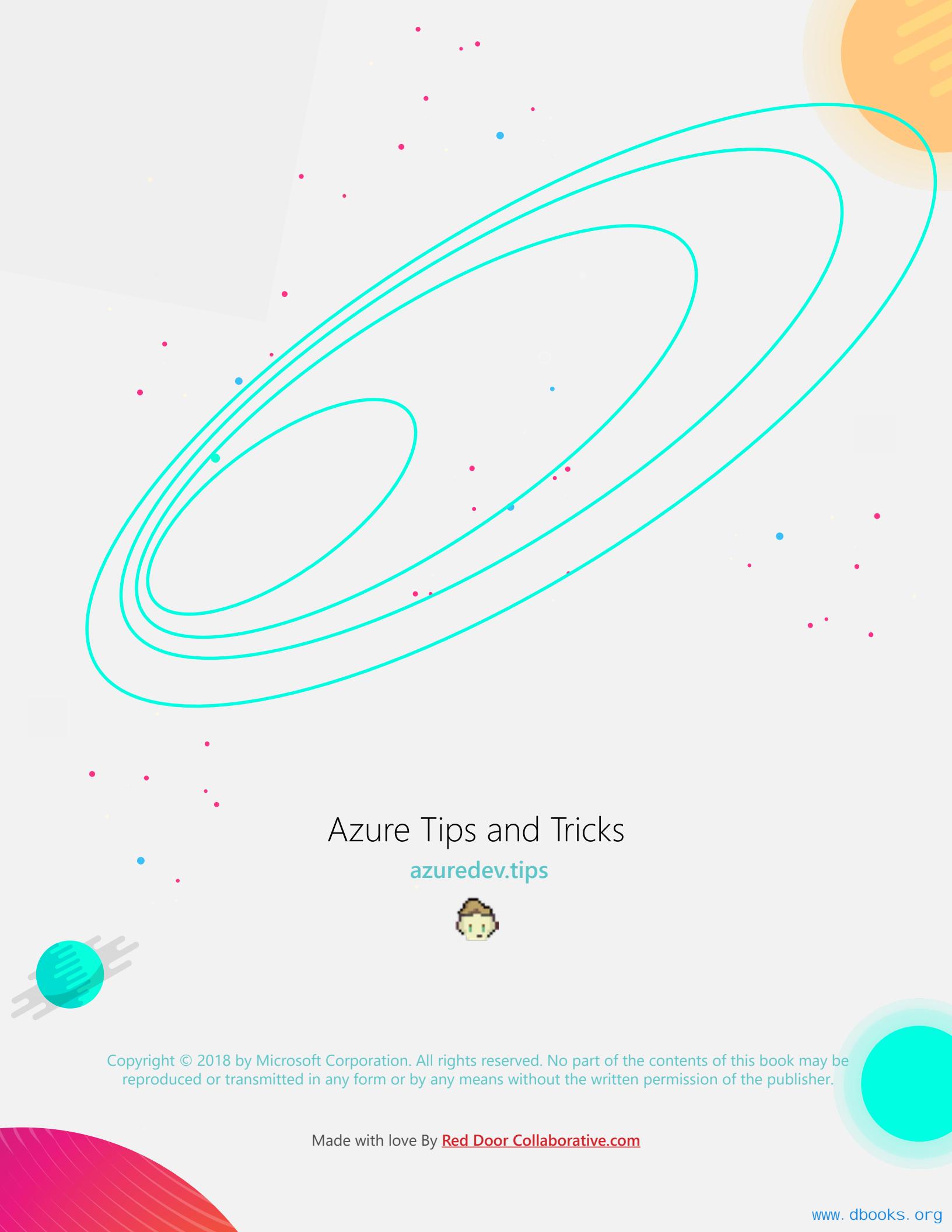
- \$200 credit toward use of any Azure service
- 12 months of free services—includes compute, storage, network, and database
- 25+ always-free services—includes serverless, containers, and artificial intelligence

[Start free](#)

Until next time,

Michael Crump [@mbccrump](https://twitter.com/mbccrump)

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)