



Tom Maiaroto

Follow

Full Stack Product Architect

Apr 4, 2016 · 11 min read

Azure Cloud Functions vs. AWS Lambda

What you Need to Know



Photo by Olivier Miche.

Microsoft announced their answer to AWS Lambda at their *Build 2016* conference. Brace yourself for the next cloud hosting change; because, it now seems like everyone is going “serverless.” Amazon Web Service’s Lambda, Google’s Cloud Functions, IBM’s OpenWhisk, and now Azure’s Functions.

There’s too much to compare in one article, so I’ll write about the differences in a series of articles. I’ve had the opportunity to play with some. Google’s Cloud Functions are in private alpha, so I can’t share much about them, but Microsoft’s are in public preview.

I’ve been using Lambda a lot lately and it’s been a **game changer**.

While Azure’s cloud function usage is billed the same way, the service is *radically* different than Lambda under the hood.

Before you make any major decisions, keep in mind that these services are evolving.

Supported Languages

This will change over time for every cloud function service.

Azure Functions supports Node.js, C#, F#, Python and PHP. They also list Java, bash, and batch but it's not clear how to use these (again, it's in preview).

AWS Lambda supports Node.js, Python, and Java for now.

I think PHP support is a very strategic choice given its popularity, but I wish one of these services would better support Go.

Why Go? Because it runs fast and its code is easy to read. It is a language of brevity and therefore fits well for the cloud and (micro)services. Add Rust and Swift to the list too.

It's important to note that you can execute binaries in both services through one of these languages. This is the method in which Go works in Lambda. Both Apex and also Sparta are serverless frameworks that let you use Go in Lambda as well.

It would be great for a cloud function service to support Go natively. Swift for mobile developers who often need simple back-end services would also be a smart move.

Web Dashboard

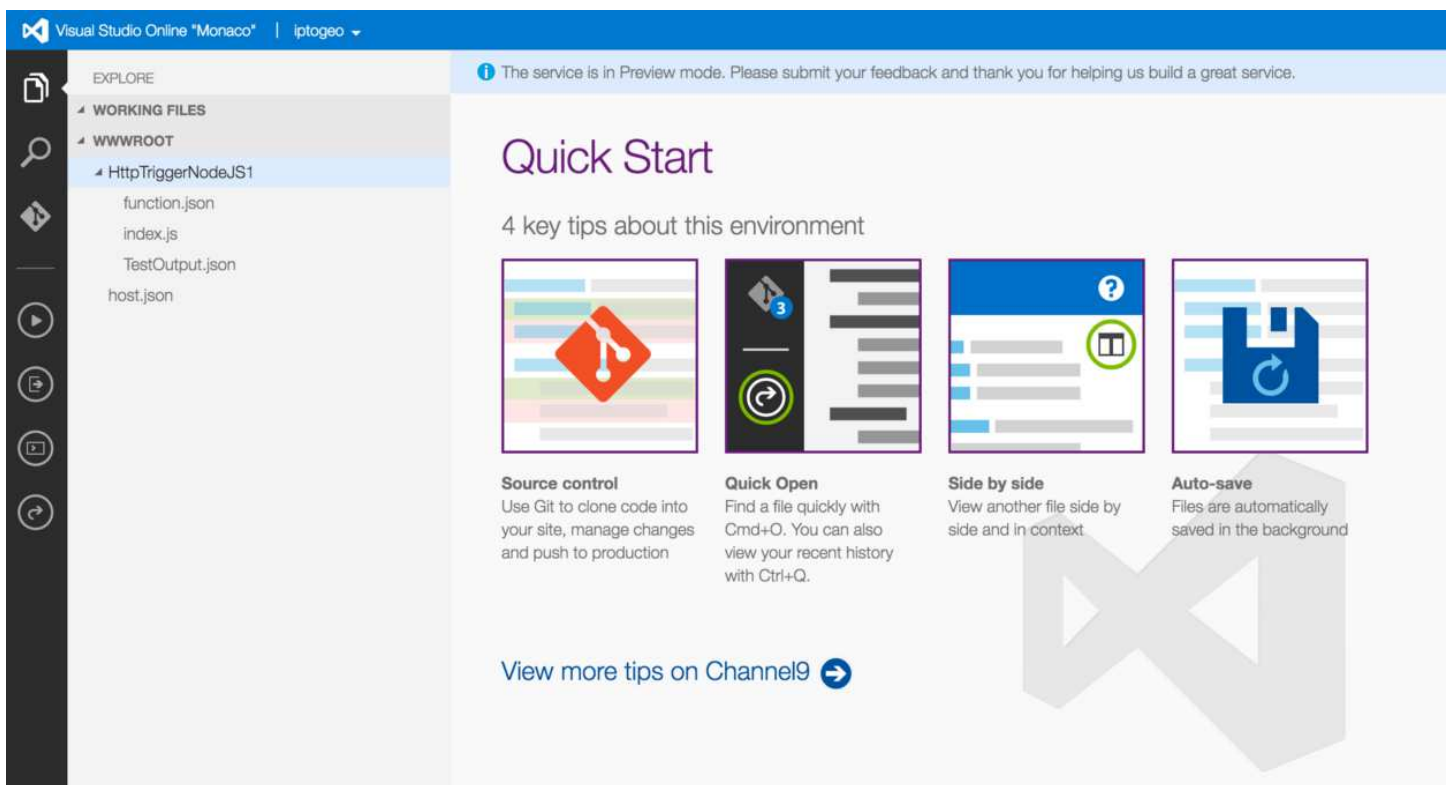
No amount of fancy devops tricks or tooling will save you from having to use the web dashboard at least every now and then.

Azure's Function usage charts are still being developed, but they will be there and at least on par with Lambda's.

Azure's new dashboard is beautiful.

Once you understand the organizational structure of Azure Function it becomes a little easier. The same can be said for AWS Lambda too.

One important difference is the editor. Both are on par when it comes to the function's settings screen. However, Azure has the more robust Visual Studio Online which can also be used.



Azure Functions can be authored with Visual Studio Online.

- Visual Studio Online is found under the Function's App Service's tools section. It's worth noting that there are some other cool tools here too, including a console.

Most of your development time should be outside of dashboards anyway. I don't consider them a huge factor in my decision making process but they can offer convenience.

System Structure

Let's dive into the architecture of Azure Functions compared to Lambda.

An "App Service" is a container, or environment, for a set of Azure Functions. This is much different than Lambda. In fact, the two services couldn't be more different.

Lambdas are organizationally independent, where Azure Functions are grouped logically into an "application."

This App Service can either be "dynamic" or "classic." The dynamic option is where you pay only for the time and memory your functions use. This is the biggest similarity between Lambda and Azure Functions.

It's important to note that the memory you allocate is per app service (with potentially many functions), not per function like Lambda.

I think of the "classic" App Service more like Amazon ECS where you have EC2 instances running to handle the tasks or functions. The pricing model for this is then like EC2.

Azure is actually more like a blend between ECS Tasks and Lambda. For example, you can set environment variables on App Services which are then available for your Azure Functions. AWS Lambda cannot persist environment variables, but ECS Tasks can (correction/update: Lambda now can as pointed out by [Jeremy Axmacher](#), thank you).

The entire container architecture is different. Lambdas provision a brand new one and deploy your code (from a zip file) on a cold request. Subsequent requests *can* be subject to container re-use and be handled much faster. However, you need to understand there is no persistence and with Node.js Lambdas you need to watch your variable scope because the container *can* be re-used.

However, Azure Functions are less subject to the cold/warm request effects. Azure still provisions resources as needed, but your files aren't "frozen" somewhere in a zip file. They run on top of Azure's WebJobs.

Azure Functions are also supposed to be accessible via FTP by connecting to the App Service's FTP. Though I couldn't get it to work so far during my review (no matter how many times I reset my credentials).

Azure Functions of course is a Windows system (32bit for dynamic services) while Lambda is a Linux system. Though I do hope that changes since Ubuntu can now run on Windows. In fact, that would put Azure in a good position with cloud hosting.

Continuous Deployment

Speaking of working with the files, you can deploy your Azure Functions in a variety of ways:

- Git (you can connect GitHub, Bitbucket, etc.)

- Dropbox

- OneDrive

- Visual Studio Team Services

- Visual Studio Online editor

- Cloud function's own editor under its settings area (limited to just the code and the function.json)

You have many more inherent options than AWS Lambda, but I didn't explore any sort of CLI deployment. Lambda has some great 3rd party tools for deploying Lambdas and both cloud services have SDKs to make just about anything possible.

I do have to hand it to Microsoft here though. You can hook up GitHub, even with your favorite CI tool, and easily deploy your cloud functions to Azure. Release managers rejoice.

Though it's important to note that Lambdas can be versioned whereas Azure Functions can not (at least not yet). So if you wanted to rollback, you'll need to rely on Git or some other version control system.

Run Your Functions Anywhere

One challenge with Lambda is development workflow. People have created tools to test your Lambda locally so you can write code, run, and then even deploy if all looks good.

There are also several “serverless frameworks” that help with this process too.

Regardless, Lambda is closed source. However, Azure Function’s runtime is open-source. I would expect some clever CI integrations in the future.

Function Triggers & Integrations

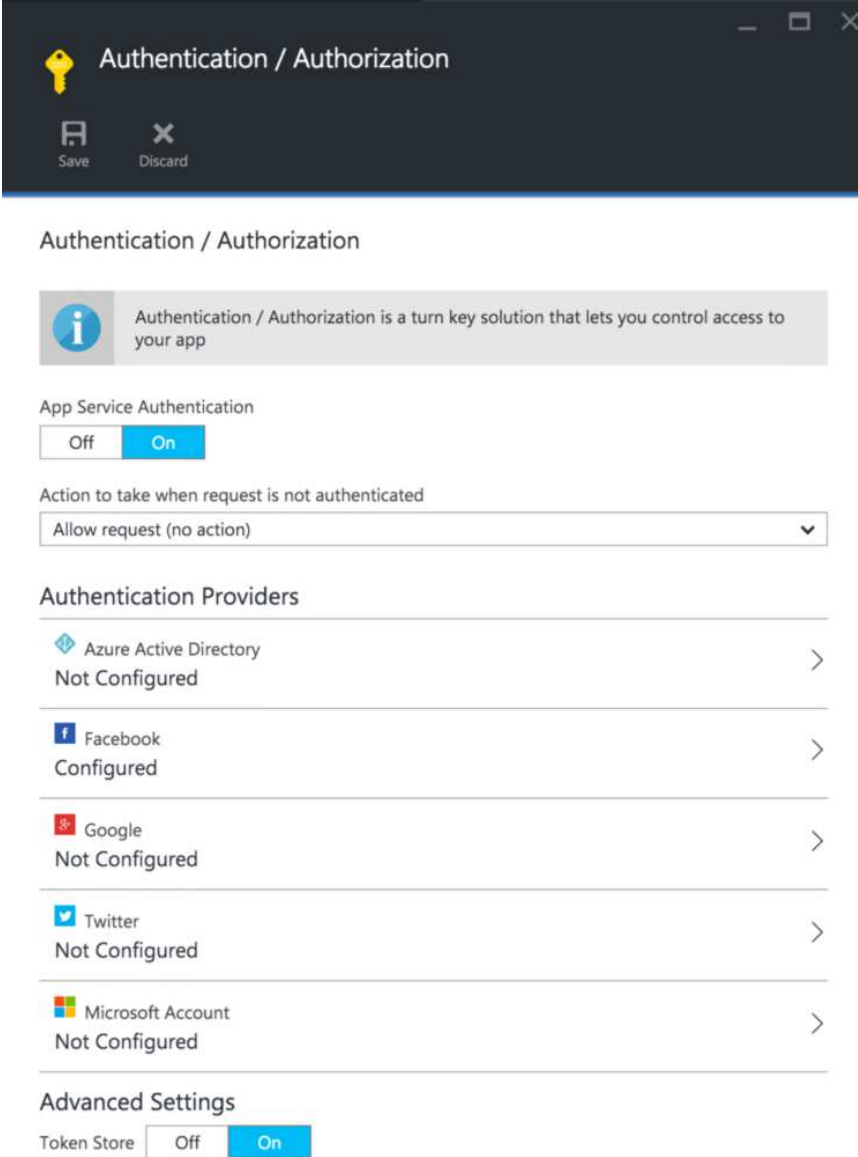
I also have to concede that Azure Functions make things a bit easier in this department too. Every function automatically maps to an HTTP endpoint if enabled. Whereas with Lambda, you must configure API Gateway separately.

API Gateway is fine, but complex and time consuming. Again, some serverless frameworks ease this pain point by automatically setting up an API for Lambdas.

Microsoft gets points for UX because you have a *lot less* to configure. I’m not sure you can run Azure Functions from multiple triggers though. It doesn’t appear that way, but maybe by editing the `function.json` directly you can.

I would say that AWS Lambda has a slight edge in flexibility here, but both services are constantly changing.

The Azure app and function names are reflected in the URLs to trigger the cloud functions. This helps avoid confusion, but might also be limiting.



The screenshot shows the 'Authentication / Authorization' settings page in the Azure Portal. At the top, there's a dark header with a yellow key icon and the title 'Authentication / Authorization'. Below the header are 'Save' and 'Discard' buttons. The main content area has a sub-header 'Authentication / Authorization' followed by an information box stating: 'Authentication / Authorization is a turn key solution that lets you control access to your app'. Under 'App Service Authentication', the 'On' toggle is selected. A dropdown menu for 'Action to take when request is not authenticated' is set to 'Allow request (no action)'. The 'Authentication Providers' section lists five providers: Azure Active Directory (Not Configured), Facebook (Configured), Google (Not Configured), Twitter (Not Configured), and Microsoft Account (Not Configured). Each provider has a right-pointing chevron. The 'Advanced Settings' section at the bottom has a 'Token Store' toggle with 'On' selected.

Authentication / Authorization

Authentication / Authorization is a turn key solution that lets you control access to your app

App Service Authentication

Off On

Action to take when request is not authenticated

Allow request (no action)

Authentication Providers

- Azure Active Directory
Not Configured
- Facebook
Configured
- Google
Not Configured
- Twitter
Not Configured
- Microsoft Account
Not Configured

Advanced Settings

Token Store Off On

Apparently all HTTP methods are supported (GET, POST, etc.) through the endpoint assigned to each cloud function. With API Gateway, you need to configure each method manually.

One thing I don't see is path variables. It seems that everything must be passed as a querystring or in the request body.

To be fair, it's called an "HTTP trigger" so when thinking about it in those terms, it makes sense. If you need more robust settings, then you're talking about "API configuration and management."

Azure Functions also have super easy authentication. You can protect

the cloud function endpoints with 3rd party authentication; this includes Facebook, Twitter, Google, and of course Microsoft's auth.

I've never setup Facebook auth for an HTTP endpoint so quickly and easily before in my life.

You can configure CORS and you can also manage your own domain name for the endpoints. Though I couldn't see if there was a way to limit the HTTP request methods. I imagine with Azure's API management service more will be possible, but then you're going through as much work as API Gateway.

I don't think there's a clear "winner" here. Both Microsoft and Amazon have powerful features for triggering cloud functions with HTTP requests. Though I definitely think Amazon could learn a thing or two about UX from Microsoft.

Azure Solves a Few Pain Points

I was thoroughly impressed by Azure and I think it's in part due to its new dashboard. The new dashboard and the process of setting up cloud functions in a logical group called an "app" makes complete sense.

To illustrate how obvious this pain point is, look no further than Serverless (formerly JAWS) and Apex. These are two serverless frameworks that address this specific organizational concern. They help a developer logically group functions.

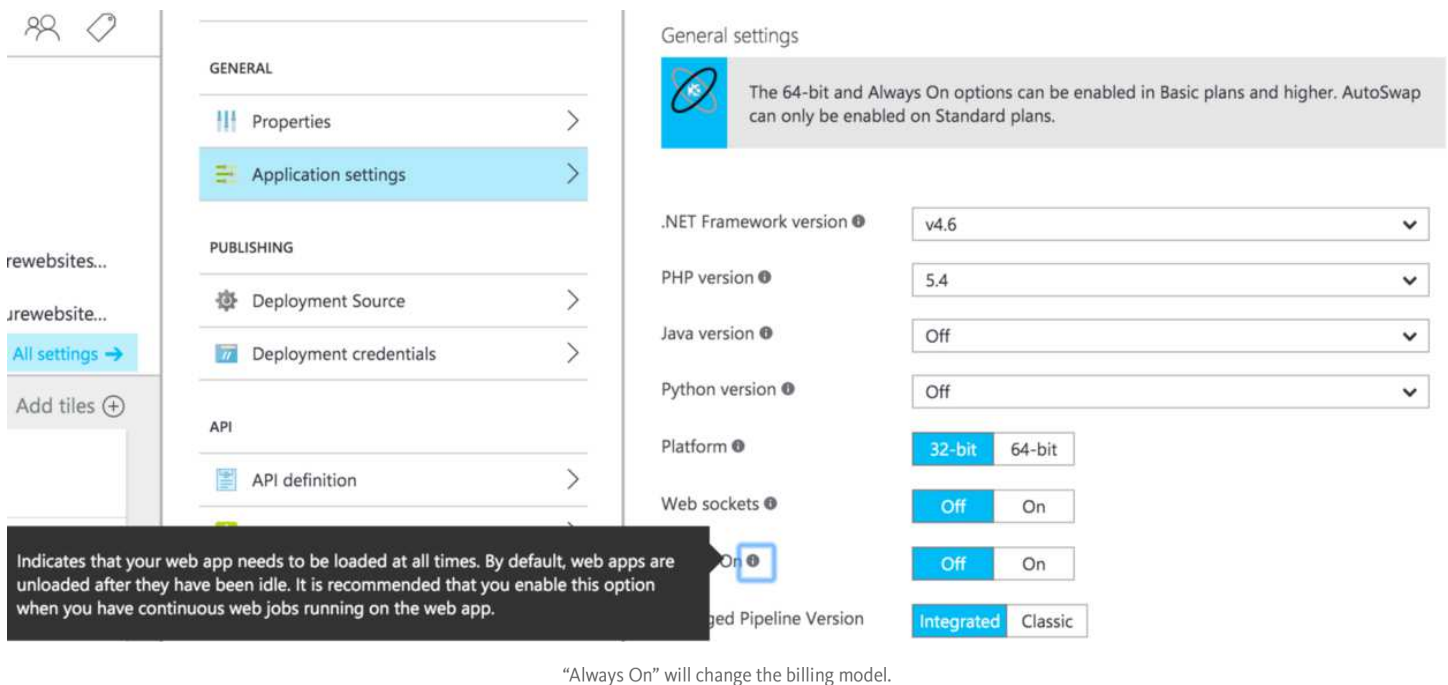
Another pain point with Lambda is the maximum execution time limit. It wasn't always clear and many people never thought their functions would run that long (at least in my experience).

The time limit used to be 1 minute for Lambda, but increased to 5 to help with "ETL" operations according to Amazon. I'm happy for the time limit increase, but even 5 minutes may not be sufficient for all operations.

Iron.io pointed out that their workers had no time limits in a comparison with Lambda. Iron workers are another option, though I stayed the course with AWS Lambda. I even found a way to re-purpose your Lambda code as ECS Tasks to cope with the 5 minute limit.

While that solved my own problems with the time limit, I understand that it's not a solution for everyone. It also requires EC2 instances which are billed in a different fashion. So it's not a perfect solution if "pay as you go" is a top priority.

Azure's Functions really are the perfect blend of Lambdas and ECS Tasks. You can write and execute the same exact code regardless of the environment and billing model. There are apparently no time limits either(so long as your function is not idle).



On the other hand, I imagine there could be some surprises in billing too. Lambda's 5 minute execution time limit does serve to protect you from expense. Will Microsoft keep billing you for some accidental loop in a cloud function? Assuming the same amount of resources were used, will the cost work out to be about the same regardless of

billing models? It's perhaps too early to say since it's still in preview and there is no pricing information available yet.

There is some chatter about this concern though if you follow the GitHub issue.

Other Observations

As I explored Azure Functions and it's underlying architecture (as best I could), I realized a few other things of note.

First, since there is persistence and no execution time limits, working with large files and ETL is easier. *Just watch out for how cost effective this ends up being or not being.*

With Lambda + API Gateway, the content-type of the response is set via API Gateway. With Azure Functions, the content-type is set by the code. This can be very useful, but could also lead to some bloated functions.

Remember, your goal is not to build a complete application within a cloud function. It's not really possible to do with Lambda's limitations, but it is with Azure. So be careful not to fall into any bad practices.

Watch out for Azure Function's limitation — only 10 concurrent executions per function.

Again, Azure Functions run on a server with a persistent filesystem and webroot. It's just not accessible to the outside world. However, it is accessible to you while logged in.

All of your code and assets can be seen via a URL like this:

`https://yourAppName.scm.azurewebsites.net/dev/api/files/wwwroot/HttpTriggerNodeJS1/index.js`

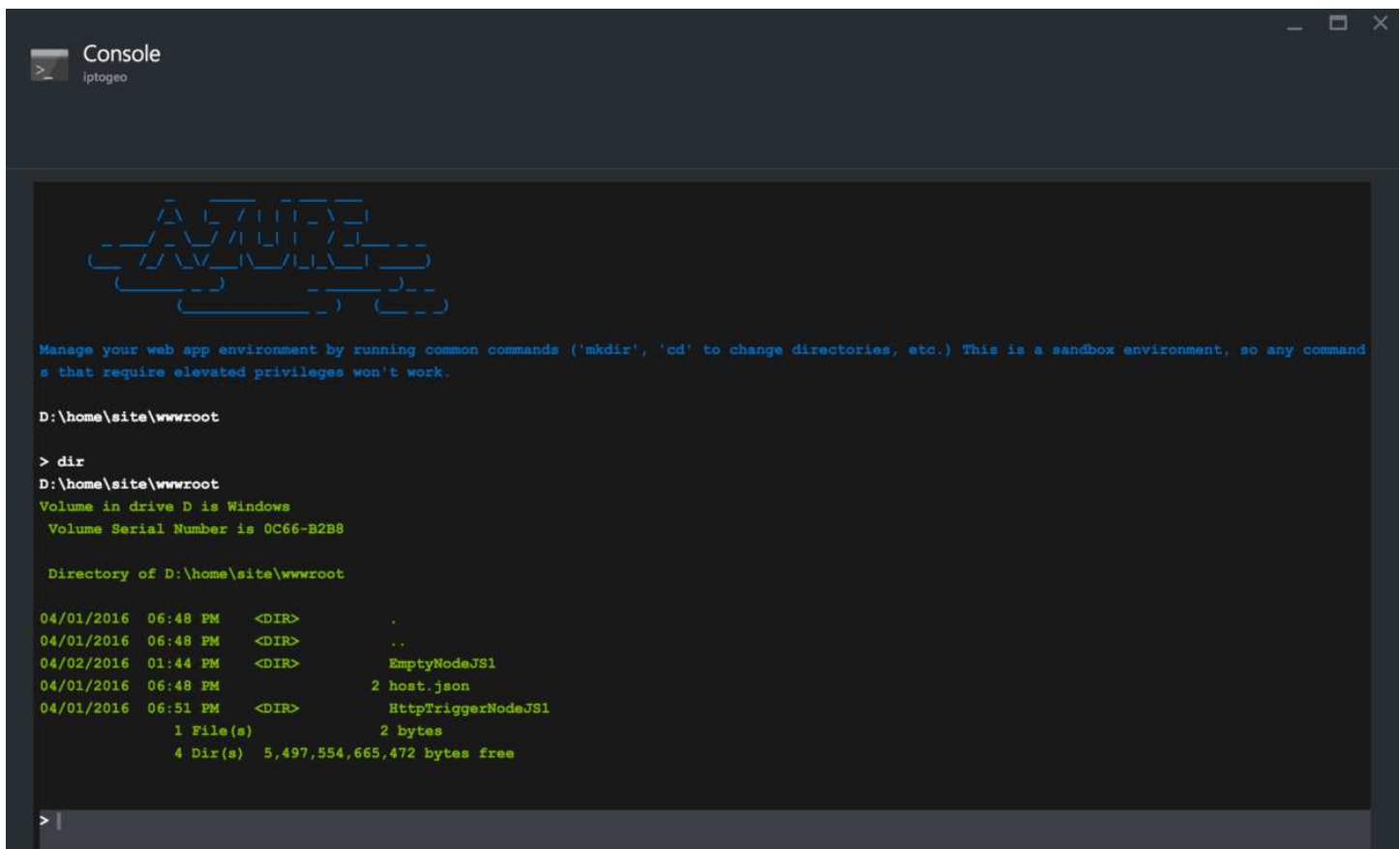
This maps this location on disk: `/home/site/wwwroot/HttpTriggerNodeJS1`

Is there a cost to request these files this way? Can you somehow open this to the public? I don't know. It definitely might raise an eye-brow or two though.

You have more insight, access to and control over the environment running your cloud function than you do with AWS Lambda...For better or worse.

Speaking of insight, when listing the directory of `D:\home\site\wwwroot` where your files are, it indicates that there is over **5 terabytes** of disk space available!

This is much more than Lambda's *default ephemeral* disk space of **500MB**. There isn't as much info about Azure's Cloud Functions limitations just yet, but it is important to note that the instance limit is **10** while using "dynamic" service apps (up from 4). The concurrent execution limit per instance is a bit unknown. Lambda's limit is **100** concurrent executions total (which is a soft limit).



The screenshot shows a console window titled "Console" with the user "iptogeo". It displays a ASCII art logo for "Azure Functions". Below the logo, a message states: "Manage your web app environment by running common commands ('mkdir', 'cd' to change directories, etc.) This is a sandbox environment, so any commands that require elevated privileges won't work." The current directory is "D:\home\site\wwwroot". The user has entered the command "> dir". The output shows the directory listing for "D:\home\site\wwwroot". It includes the volume information "Volume in drive D is Windows" and "Volume Serial Number is 0C66-B2B8". The directory listing shows files and directories: ". .", "EmptyNodeJS1", "host.json", and "HttpTriggerNodeJS1". It also shows the file size and free space: "1 File(s) 2 bytes" and "4 Dir(s) 5,497,554,665,472 bytes free".

```
Console
iptogeo

Manage your web app environment by running common commands ('mkdir', 'cd' to change directories, etc.) This is a sandbox environment, so any commands that require elevated privileges won't work.

D:\home\site\wwwroot

> dir
D:\home\site\wwwroot
Volume in drive D is Windows
Volume Serial Number is 0C66-B2B8

Directory of D:\home\site\wwwroot

04/01/2016  06:48 PM  <DIR>          .
04/01/2016  06:48 PM  <DIR>          ..
04/02/2016  01:44 PM  <DIR>          EmptyNodeJS1
04/01/2016  06:48 PM             2 host.json
04/01/2016  06:51 PM  <DIR>          HttpTriggerNodeJS1
               1 File(s)                2 bytes
               4 Dir(s)  5,497,554,665,472 bytes free

> |
```

Your Azure Functions are but a drop in a very large bucket.

Is everyone's code on the same storage volume? Or does each Azure account get its own, very large, storage volume? I'm not familiar enough with what's going on to know, but exploring is fun.

So Which Path?

I would encourage everyone to explore and try each of these services. Understand why and when they can help you. The end result can be a game changer for you too.

The cloud has powerful magic, but not all clouds are the same. Get to know the environment your code lives in as well as the capabilities of the services available to you. Read the fine print. Understand the limitations and tradeoffs.

Only with this information can you make an informed decision. Remember, your choice will be the path less taken here.

My choice? The jury is still out. I use a variety of services because I believe there is strength and opportunity in diversity.

I primarily use Lambda and ECS because it's well established. I love AWS and will continue to use it happily, but I definitely think these companies can learn a thing or two from each other.

There's a lot of opportunity in this fast changing landscape. There aren't many books. Few best practices. Barely any frameworks. Less than ideal monitoring and debugging. There will be trial and error. You are subject to the possibility of having to re-build code and move your operations. If it gives you any solace, you will be in good company. Some major companies and smart developers are on board with cloud services and the (micro)service movement. This is the new frontier on the web. Buy the ticket, take the ride.

