Hi. Today we're going to learn the basics of about the server less offerings of 3 different providers, Google, Amazon & Microsoft. Before I start, I wanted to do a little housekeeping. Throughout this conference you've been learning lots of cool things and I wanted to make sure that you don't need to spend my talk frantically taking notes, and instead focus on laughing at my jokes. Everything I am covering today will be on my website. You'll find links to the slides and the repos under the "Speaking" tab.

I also want everyone to feel comfortable asking questions. Let’s make this a conversation. If you have a question during the talk, feel free to raise your hand. Also, if you don’t feel comfortable raising your hand, you can tweet at me and I’ll address it at the end.

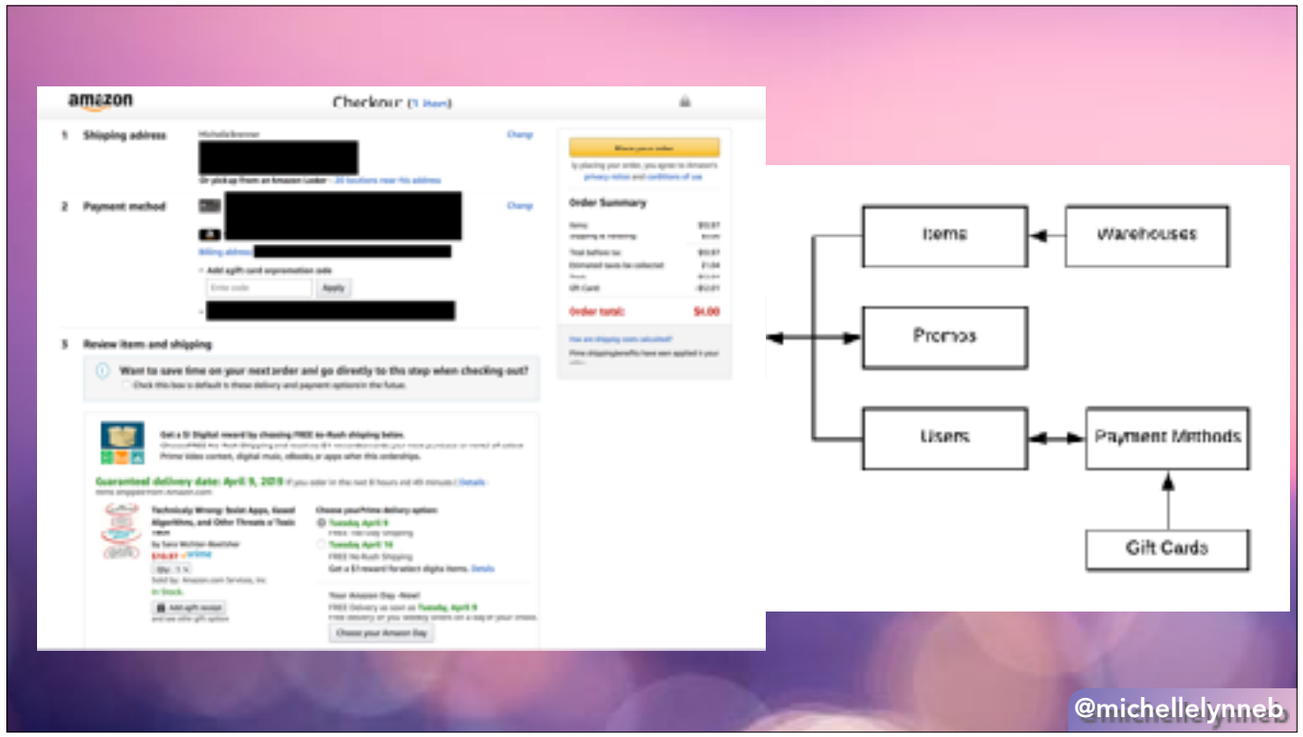
So why did I want to give this talk? Being a self-taught developer means your education is ad-hoc. Most of the things I learned were from other people suggesting them, or trying to solve a direct need on a job. When I was working more and more in the cloud I realized most people aren't going to have the chance to work with different cloud providers. Changing cloud providers can be very difficult and not worth a company's time. And if you do get to choose a new provider on the next job, it's much easier to go with the one you are familiar with. Because despite all the documentation and demos, cloud services have a steep learning curve. They all accomplish similar tasks, but with different terminology and design philosophies. After today, I hope you will have the knowledge and confidence to start creating your own projects.

8 years ago, I got an extremely practical degree in 3D Animation. Luckily for me, I had some experience coding, so when a job came up to work in a tech department at a VFX company, I jumped at it. Working in entertainment technology is very different than other large technology companies. Technology is not their business, it is just a means to an end. Their product is art. Anything that will get the artists producing faster or higher quality images will be used. The work cycle is also not in sprints, quarters and years, but instead is counted in projects and film shots. My first job was focused on reducing the company's biggest expenditure, artist time. If you were working on a tool and it was not done fast enough to actually help on the project you were working on, it was useless. This taught me to be practical and put business needs first, not technology. I like serverless apps because they help me solve problems faster and easier.

Finally, before we dive into the tech, I want you to keep in mind my motto for all my tech talks. I want you to learn how to use these tools so you can do less and accomplish

I am going to quote here from AWS. “Serverless computing allows you to build and run applications and services without thinking about servers. Serverless applications don't require you to provision, scale, and manage any servers. You can build them for nearly any type of application or backend service, and everything required to run and scale your application with high availability is handled for you.” -- https://aws.amazon.com/serverless/
This is also often called function as a service. You provide the business logic, they take care of everything else. Sortof. But we’ll get to that.

Serverless works best as small, self contained microservices. Each service does exactly one thing, so you don’t have to create complex interactions in each one. Instead, they all talk to each other to create larger features. As an example, let’s talk about an ecommerce site like, say, amazon and it’s checkout page. When a customer is trying to checkout, there are numerous questions that need to be answered to provide a checkout service.



Here I am answering questions like are the items available and what warehouses they are in (to check shipping speed). Are there promo codes to be checked, and what payment methods does the user have? Do they have a gift card? By separating these out into different services it means only the service needed is running at any given time. Remember with a serverless app the service spins itself up when needed. If the user never enters a promo code, the promo service never has to come up. It also allows for independence. Internally the services can do anything as long as the input & output stays the same. When looking at this chart it should remind you of is relational tables or endpoints in a RESTful API. Can anyone think of a service we might want to add to give data to this checkout page?

Keeping this process in mind, I wanted to create a template that can be used to make all of these services. I wanted an endpoint that allows you to add data to persistent storage, get that data, edit that data and delete that data. I wanted to be able to test that endpoint, deploy updates and have it secured. I want my template to define all the services I use, so if they all get destroyed, I can easily spin everything up again.

These are the questions I wanted to answer about each one of the cloud providers. I not only wanted to answer them, but test them myself and save it in the repo.

What are each of the services needed to create my microservice? How do they work and do they work well together?

How fast can I get started coding? How do I structure my code? What’s the best quickstart tutorial?

Is there a capability for creating a template to describing the services and how they interact? How difficult is it to understand and create? What were the roadblocks in understanding how it works?

I’m a big fan of TDD and I like to get started right away. Until I start testing, everything I do is theoretical. So for testing I wanted to answer:
Was I able to setup local endpoints and test them? How difficult was it?
Was I able to add unit tests? Was I able to mock the infrastructure so the tests were self contained? How hard was it to create test payloads so unit tests were as accurate as possible?

Was I able to successfully deploy? As in, is there a url I can give out that returns the data I want.

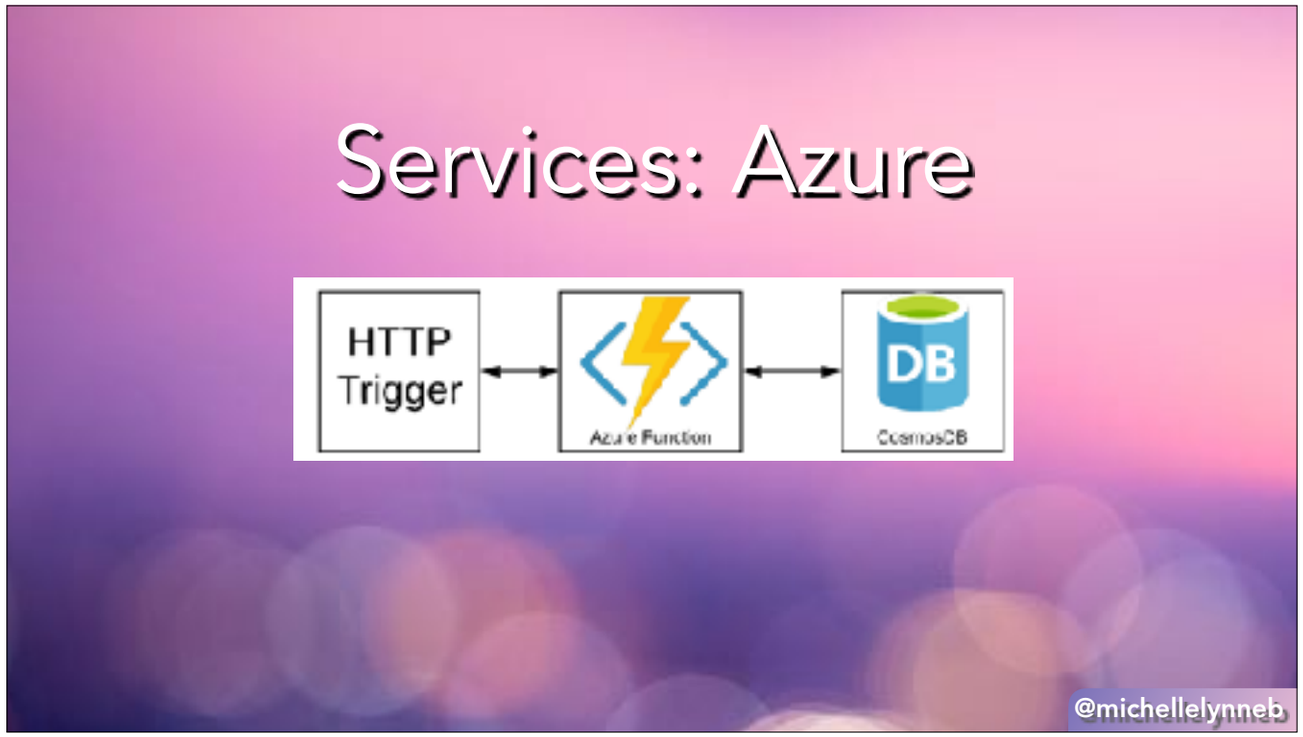
Is the endpoint secure? Can I limit who is using it and how much? How do I secure the services between eachother and limit who can update the service?

How hard is it to figure out when something is going wrong?

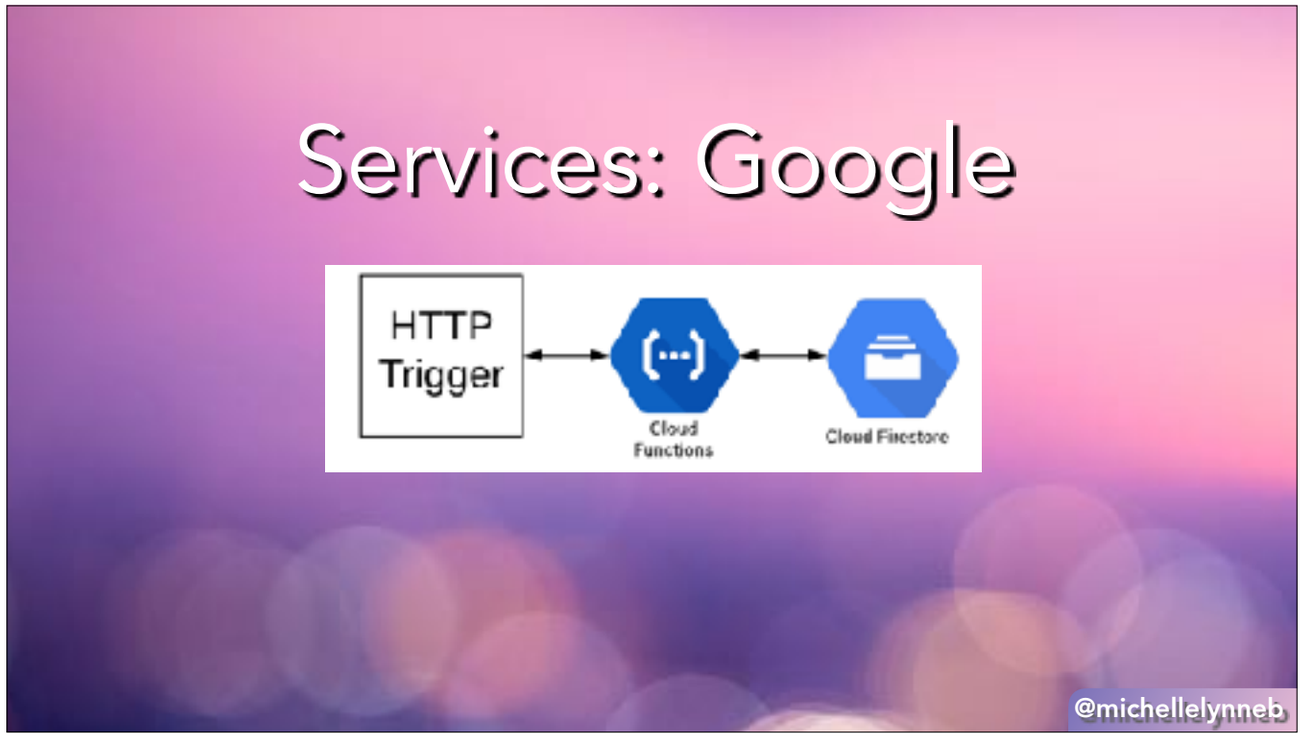
Not to discourage you, but my first lesson with all of these services is that they were harder than they looked. Setting up a quick example in the UI is fairly straightforward. Just a matter of filling in forms and deciding on names. But even then it might not always work. Setting up the app in a sustainable way so it could be production way, much more difficult. There are a myriad of options on everything you do, and error messaging can be obtuse. Hopefully you are a little sympathetic as an engineer that has tried to write an error message for someone to understand. But I am telling you this so you do persevere. There is a steep learning curve, but sustaining and scaling the system makes it worth it. What I will go over today won’t give you all the answers, but will give you a step up.

My approach on figuring out which services I needed started with the core, knowing I want to store data for an item. I asked myself what's the easiest way to store an item with an attribute for each service. What's the serverless tool to access that data and how do I create an endpoint for that service. These are what I decided on.

For AWS I used API Gateway, Lambda & DynamoDB. API Gateway is the dedicated service to creating the routing. Lambda is the function where it parses the request and accesses the database via an SDK. DynamoDB is a non-relational document storage. I chose this database because it is the best integrated with the serverless services on AWS. If you are used to SQL you have to be careful with this because it can quickly get messy. I use it here because it's fast and easy to get setup and with these one task microservices, it tends not to have too much complexity. I worked on setting up a MySQL database but it required a whole slew of services, and wasn't worth it when I need a single table.



For Azure I used an HTTP Trigger, Azure Function and CosmosDB. You'll notice Azure named its Function to represent what it is, which is points in its favor. I also want to point out that the entry point to a Function is not another service, like with the API Gateway. Instead Azure has this concept called triggers and bindings. A trigger is anything that can call a function. In this case we're going to use an HTTP trigger, which will handle the routing. You can also trigger with their other services, like a file landing in storage, or a row changing in the database. A binding is the output of a function. In this case, the binding output is going to be CosmosDB. This datastore is flexible and hold a variety of collections, though I used it as a table and accessed my data via SQL queries. I chose CosmosDB because it was the builtin, recommended service for Functions. I hope you're starting to notice a trend here.



For Google Cloud I used an HTTP Trigger, Cloud Function and Cloud Firestore. They also get a point for naming their function, Function. They get negative points because the combination of documentation combining App Engine and Cloud Function was confusing. It took me a while to realize they were two different services that did slightly similar things. There's a good takedown in the readme, but the basics is Cloud Function is a simpler version of App Engine, which is why I went with it. Cloud functions also have triggers like Azure. It can be other services, but it does not use the binding concept for output. Instead the function accesses the database via SDK, like AWS. Google doesn't use its own routing, like Azure or AWS. Instead, it uses flask, a common python framework. If you are used to using flask, this is great because you already know how to use it. It also means the routing is tightly tied with the language and function, instead of decoupled like API Gateway. In practice for me it did not make a huge difference, but if you are using multiple languages on different services it could be annoying to have to define routes in different ways instead of say, one YAML or JSON file.

Now let’s talk about the quickstart. If you’re anything like me, you want to code as soon as possible. Deciding on folder structure, typing out config files, not fun at all. So I wanted to test each provider on how fast I was writing code. A nice feature of all 3 of the providers is the ability to try things out in the UI before you even have to create a repo. This is the best way to start because you can see how the services connect, get things launched, play around without having to worry about syntax. It’s definitely possible to make an entire project that can be used, but then you’re not using the source control or IDEs that make collaboration possible. So I made a fresh repository in GitHub and got started.

For AWS, I used SAM. SAM is a multi-use open source tool. For this section doing a sam --init does the initial directory setup and included a sample Hello World. It includes a READme that doubles as the tutorial for setup. I was able to go into app.py and start working. I only started running into trouble when I had to add a library (like simplejson). For authentication to my AWS account, I had to set up a config file and a profile (because I have multiple AWS accounts). I then set the profile as an environment variable. I setup all my secrets/configuration as environment variables. It wasn't bulit-in, so I had to handle storing and deploying that myself. One interesting step is that I had to confirm my identity via a phone call, which took a few hours to get to me. I was confused why my account wasn't working right, and that's when I realized it was my identity confirmation, so be on the lookout for that if you are just setting up your account.

For Azure, I used func. It’s an all purpose tool like SAM that has an init that sets up everything. It was straightforward and walks you through setting up your first project. One warning is all the initial installations can take a very long time, so I would start installing before you want to work. I used the “Create an HTTP triggered function in Azure” tutorial and everything worked. The actual Python looks tricky because it has some formatting that seems more like Java but once you understand the trigger/binding dynamic it makes more sense. The authentication uses a shell script that redirects to the website. I’m sure it is storing something on my machine, but I thought it was neat that it handled it for more. Another nicely handled feature is the local settings. By default it is already in your project so you can see how to set it up. Anything you put in there is available as part of the deployment, like the database name, or in your function. When you deploy manually you just have to specify add local settings for it to pass it along

For Google, there wasn’t a single tool that got me up and running. Instead I had to cobble together a bunch of tutorials. I started with the how to create it in the UI. Then I downloaded the function to see what it looked like. It did not include a configuration file for when I wanted to deploy it. I’ll talk about that more in the next section. In the meantime they have a Python Samples repo where I was able to pull how to setup and test functions. Instead of launching a local instance like the other providers, you tested using the built in Flask tools. For authentication you setup a key, then download a json file that you add to your environment settings. For other environment settings, it is similar to AWS where it goes into the config file or you store it in their version of secret manager called NDB, which is stored in their Datastore.

In any cloud service you'll want to programmatically define the infrastructure as much as possible. This means all the configuration for the app is written out and can be stored in version control, with the code. For example, Amazon uses a tool called Cloudformation. Cloudformation reads either YAML or JSON. In that file you can say I want to make a DynamoDB table, call it this and give it these settings. Say you want to add an index or change something about the database setup. If you do it in the console, it's not easily trackable. If someone wants to know what you did they can try to puzzle it out, but it's an investigation. Instead, you'll want to change your cloudformation file, commit it to a repository, then have Amazon read and deploy your changes. This not only documents changes, it allows for multiple environments and emergency recovery. For a professional level project you'll want atleast 1 staging environment to test things out before you release to production. By having this config file you can make as many versions of your setup as you want, with different names.

As I mentioned, the Amazon tool is called Cloudformation. But there is also a slightly easier way. The SAM tool includes a simple template file you can use. It helps with things like connecting the services and creating all the needed permissions and roles. Upon deploy, SAM converts that template file into a Cloudformation document. SAM template does not provide all the functionality you might need for a service. For example, I needed more options when I was working on a MySql database. In that case you can add Cloudformation settings directly into the template. Even with this template it took me a bunch of tries to figure out the formatting. Every AWS service is built to be separate, so you have to make sure you define everything.

For an Azure function, you're not going to need a dedicated tool. Instead you have a json file. When you initialize it's created for you with an http trigger and http out. I added in a cosmosdb binding so my data goes in there for the post. There is another file connected called local settings. If you add settings in there the function.json file can find it, as well as the function itself. When you deploy there's a flag to make sure the parameters go with it. I found this to be a very helpful feature.

For the Google Function, there is not a config file. If you see any references in the documentation to app.yaml, that is for App Engine. The only connection to another tool is through the SDK inside the function. If you need something more complex than what's provided, that's when you'll want to move to app engine.

One feature all the providers have in common is OpenAPI. You might know this as swagger. This is what you'll want to use to design your APIs. It lets you define endpoints and their parameters. This is especially crucial for all the databases I picked because it helps with payload validation. Since they all thankfully use the same thing, there's really no comparison there, but it's good to know it's there.

Now that I knew where the code was to go, I wanted to do some testing. Especially when I am working with brand new tools, setting up tests helps me learn them. I use the Python Debugger tool to stop the code and evaluate different scenarios on the spot.

Unit tests were fairly similar to other Python projects. For any outside calls, like calls to the database, I mocked. If you don’t mock them, you are actually calling a database, which can be dangerous unless you specifically setup a local environment database. You could have multiple people calling the same database and that will get confusing. For Amazon I was able to get the example payloads from the UI. If you go to the lambda there is a create tests section. Scroll to API Gateway and they have a sample payload you can base your mocked calls on. They are JSON payloads. For Azure, I just played around with the functions until I was able to get a mocked request setup.

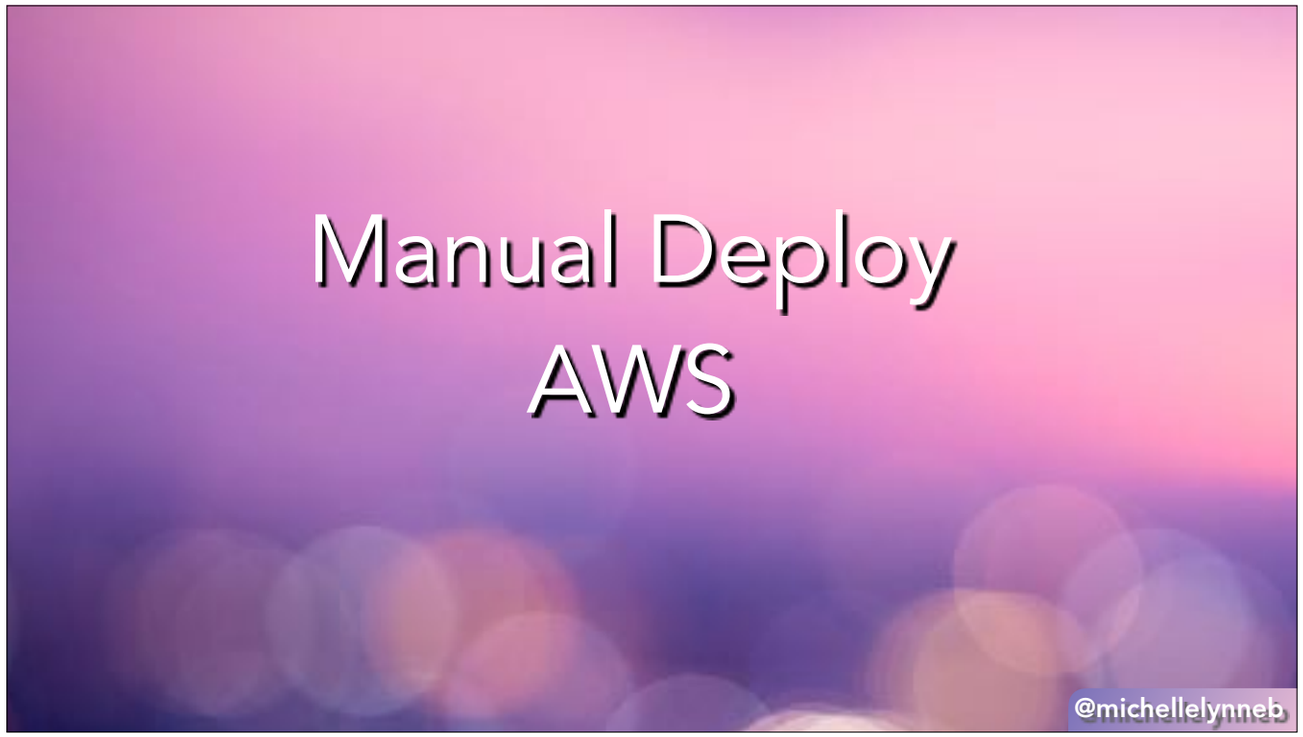
For integration tests, I had to get a local version of the API up and running. Both Amazon and Azure use the same SAM and func tools, but Amazon has an extra step. You have to “build” the app. Building the app means putting the executable file and the dependencies in a hidden directory. Then SAM runs the gateway from there. This is critical to know, because SAM actually says “no need to reload code when you make changes”. But that’s a trick. You have to edit the hidden file. Then you have to be careful to copy any changes you made to the original file, otherwise it will disappear when you rebuild. You also have to rebuild if you add any packages. With Azure, there is no build step and it will automatically handle the packages and pulling in new code.

Once I had them all up and running, I treated the endpoints like they were a 3rd party url and did a request. I let the localhost run and then saved the output using VCR. VCR is a Python library that blocks requests as long as the parameters are the same. For example, in the Azure repository I have a test get on a localhost. I did it once, saved the output, and now when I do it again it returns the same cached result. If there are changes made, you'll want to delete the cache and try again.

For Google, it’s the same as any flask app. I used the examples I found in the Google Cloud Python samples repo and was able to create unit and integration tests that use pytest.

All of them suffered from poor error messaging while doing integration testing of one degree or another. Amazon has a really good one where it just “times out” without telling you what took so long or why. But eventually I got them all to go through.

Now we've got some code working and tested, it's time to deploy. First step is to deploy manually, then see what the options are for automation. Amazon and Azure both used the same tools we've been using, SAM & func, to package and deploy the applications.



For Amazon the first thing to know is that you are deploying a Cloudformation Stack. When you want to find the deployment logs, that's the service you'll go to. You can pull them in via the command line, but it's much easier to read in the UI. The problem will be in the middle somewhere because the top of the logs show the rollback of everything that succeeded. If you fail to deploy repeatedly, don't get too discouraged, it takes a while to interpret the errors and connect all the pieces correctly. So I finally got the Cloudformation stack deployed, but it didn't 100% work. I had to reconnect the API Gateway to the Lambda each time, or it wouldn't be able to find it. But once I did that I was able to test the endpoints in the UI.

Next I tried automating this deploy. The recommended tool is a connection between CodePipeline and GitHub. I made a hook so when I merge to a specific branch, the CodePipeline kicks off. The pipeline I set up does all the copy and deploy I was doing manually. It's also just a series of commands, so it can be used to add features like automatically running tests and code quality checks. This can be used to update different environments by using a different branch with different variables. For example, a staging branch that has "staging" in all of the endpoint names.

For Azure, you deploy the function app itself. A warning here is that logging is not automatically set up for functions. I haven't yet figured out a way to add it to the config, so I had to do it manually. It's called Application Insights and you subscribe, then connect it to your function. There is also a 5 minute delay, so when I started testing the endpoint and got nothing in the logs I thought I had set it up wrong, but really just needed a 5 minute break. When you deploy using func it gives you your new endpoint on azure to try out, which was great. Sso I was able to create Postman tests, which is my preferred method of testing APIs. First thing I noticed is that Azure SDKs were not included in the pre-installed packages. When I did add it, I got strange errors I could not figure out, and had to submit a help desk ticket. I am pretty sure it's because Python Functions are still in preview mode, but as of this presentation haven't gotten it fixed yet.

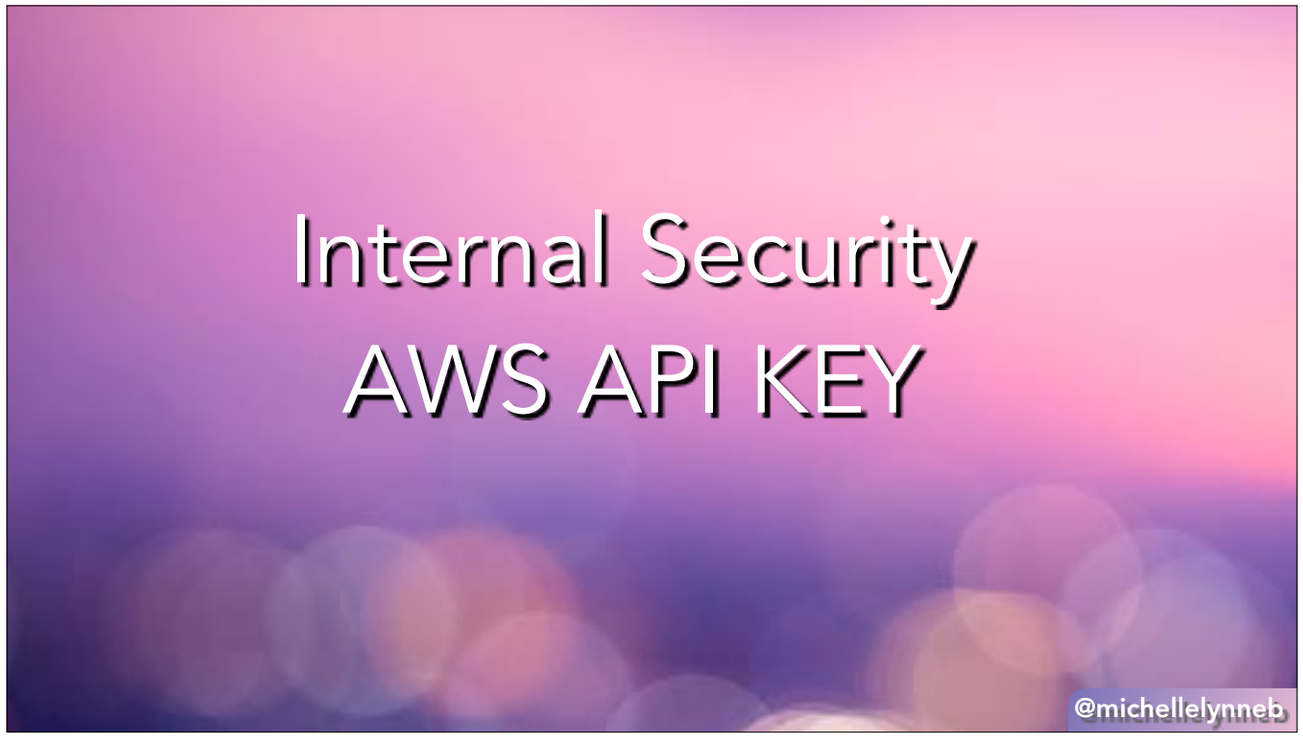
To automate the deployment, I used Azure DevOps. It is a similar style as Amazon. You connect your master branch from the GitHub repository and it launches every time that branch is updated. Then you have to create two different things, a pipeline and a release. The pipeline is where you put things like the python version, tests and the code quality tests. You also create the artifact for the release. I found a template for this and used that. I created a new release and this is another place where I had trouble. This whole section was confusing and could use the same simplification the “func” command provided me. Func knows I just want to deploy my application, release wants me to set a whole bunch of settings that aren’t very clear.

For Google first you have to mirror your repository into the Cloud Source Repo. That way any changes you make to your GitHub repo will be reflected there. Then you have to make CloudBuild yaml file. Most of the documentation talks about docker artifacts, but you just need the correct format for Cloud Functions. You can test this build using the command line. My build took a bunch of tries to succeed because the Service Account that runs the build did not have the correct permissions. Why isn't there a button to just add the correct permissions? Who knows! It was fun that the error message said "run this command to fix it" and the command was mis-formatted and I never got it to work. Eventually I made a custom role with the correct permissions and added it to the service account. IAM is never easy.

After the build succeed I setup a trigger for automated deploy. This was so easy it was almost anticlimactic after the CloudBuild issues. It has some nice custom features I didn’t see in the other providers, like the ability to trigger on a certain file in a certain branch.

Now let’s talk about security. As of now I have been making an API anyone can access, but we’ll want to limit that. Whether this will be public API or a bunch of internal services, you’ll want to be sure you can control who accesses it and when. You don’t want to allow unlimited access even from internal services because you could have a domino effect of failures. If one service goes haywire, you want to do as much as you can to limit it to that single service. It’s also helpful for compliance to limit data access. Finally, you want to limit developer access to change the infrastructure.

For Amazon, there are a few different ways you can go. For internal service to service security and developer access I liked IAM. If you're unfamiliar with IAM, first, I'm sorry for what you're about to go through. It's an extremely granular permission system, which is great for when you understand every detail of your system, but incredibly frustrating when you just want one tool to be able to access another tool. My method is to create roles with all the permissions and try out the action. Then I slowly remove permissions until only the actually necessary ones are there and the action still works. It can be tedious, but once you have it working you have a well locked down service.



Another option is to generate API Keys and share them throughout the services. This is basically the brute force method, but will get you going faster. You can use AWS Secrets Manager to store and rotate the keys. You'll want to create a seperate authorizer service to check the keys so you don't have to re-write the same logic for a bunch of services. And after you've done all that you'll think, maybe I should have just figured out the IAM permissions. But it's good to know your options.

So there’s also the user security if this is a public facing API. You could use the same API Key method I already described for this as well. But the way I liked better is AWS Cognito. Cognito lets you setup user pools with different permissions and automatically assign users as they signup. It also integrates social media logins for you. That was a big selling point for me since I hate making new logins and always want to make products I would enjoy using.

For Azure, the security features are all of similar designs. The public security tool that includes social media login is Azure App Service. One thing to note is that this is in preview mode like many of the Azure tools I researched. The Azure version of Secrets Manager is Key Vault, which you can use to add and access API Keys. It does not currently have an automated rotation tool, so you'll have to do that on your own. There's also the ability to create internal user and system identities to limit access, which is similar to IAM. One more security feature is the ClaimsPrincipal, which I did not dig too much into because it is currently only available for .NET. It looks like a promising way to access the user permission information.

For Google, the 3 areas to secure use the same tools. Those 3 areas again are service to service, developer access and end user access. This is accomplished via a combination of IAM & Oauth. There is also a tool called Firebase that lets you authenticate via other social logins or email/password. Oauth is basically a way of letting the user login and agree to the permissions you've asked for. You've seen this in action when you are using mobile apps. Luckily for us, Google makes it as easy as possible. You don't need to setup any webhooks or custom code. You just designate an oauth code for the application and add the javascript Google generates for you to your site. I always enjoy a good copy/paste installation.

One question that comes up often with serverless functions is, how do I know it's running? With a traditional server you can have a ping letting you know, hey I'm still here. With a serverless architecture you should only be running when you are computing, so that doesn't work. It's important that you write your code and setup your services in a way that you know if something is going wrong. All of the providers have logging tools, but for Azure and Amazon you have to make sure to turn them on. When I realized that, I was confused because I was like who doesn't want logging? Why is this not automatically turned on? Then I realized you have to think of cloud providers like Spirit Airlines. They get a bad rap, but I kinda like them. You get what you pay for and not a thing more. They won't secretly charge you for things you don't ask for. So you don't want to be charged for logging? Great, logging is not turned on automatically.

Anyway, once you have logging turned on you can see what's going on. For this portion I recommend thinking about the tools you already like to use for alerting and debugging and plug those in. I personally like Sentry, so I use that for all the tracebacks. You can also set up all sorts of rules and alerts within the providers themselves. I have not taken any of these apps to scale, so this is definitely an area for more research.

Google was the only one who turned on logging by default and it was pretty clear what was happening. It also has a feature where Python print() statements go into the logs. So if you just want a quickstart and not have to worry about defining a logger with levels, that’s there.

So one thing I wanted to do for this section is to tell you how much this app would cost you between providers. But sometimes your abstract is too aspirational. My current costs for all of these projects is $0. Since I am only testing and have low throughput I am still on the free tiers. Putting in my credit card number was scary, but the first thing I did was make sure I put in billing alerts so I didn’t accidentally leave something expensive running.

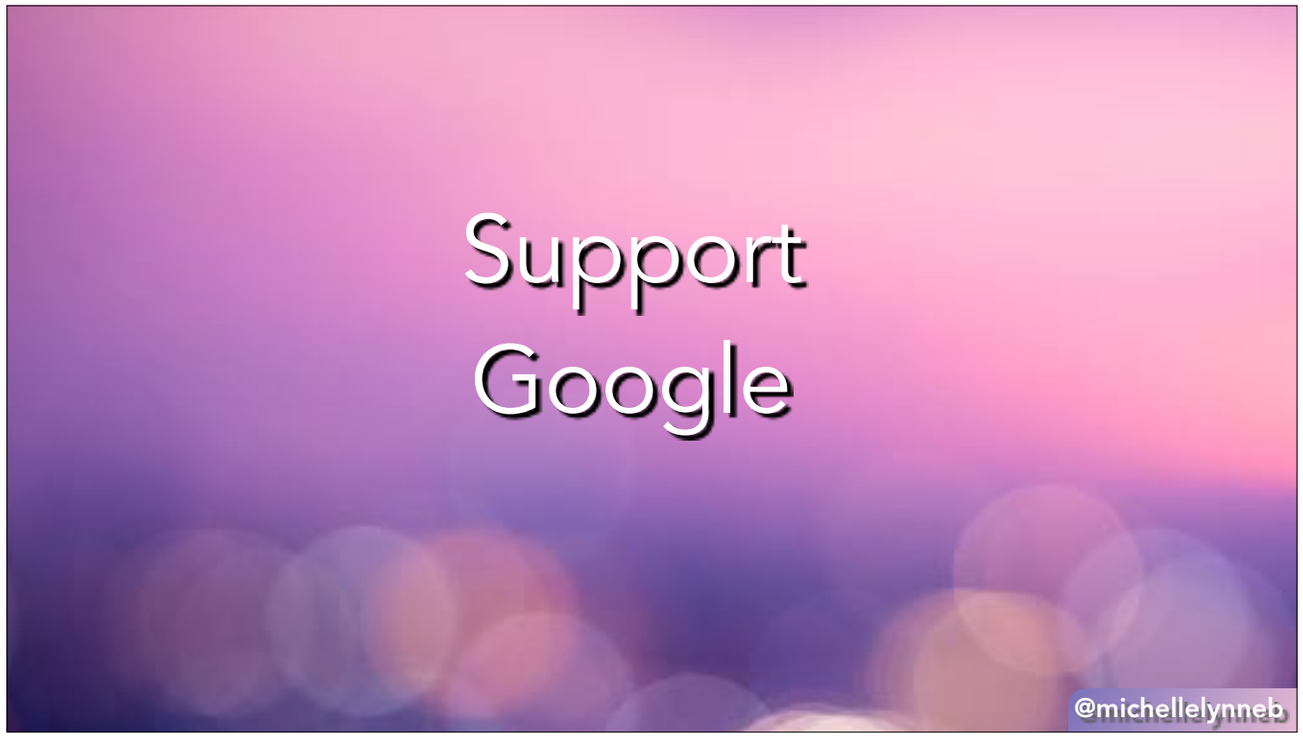
So instead of going over how much this project would cost you, I wanted to talk about estimating costs for your own project. All of these providers layout exactly how much each service will cost to the minute detail, but that actually makes it harder to figure out. What you need to do to estimate costs is to create a detailed map of exactly what services you use and what calls they make. Then you need to go into each service and check the costs both in usage (per call) and any ongoing costs. For example, having an encrypted key costs this much per month, but also this much how many times it's used. That's something to keep in mind when designing your app. Are you wasting money by calling secrets manager 5 times instead of caching and re-using the data? Another thing to keep in mind is all these providers want your business. They want you to grow and expand your project so you're both making lots of money. So they're invested in your success. They all have programs designed for startups to get them going and if you are starting a business I would make sure to reach out and see what your options are.

Speaking of reaching out, let’s talk about support. In order to learn and make what you need you’ll want to use a variety of resources. The first step is the provider documentation itself. They generally have some quickstart guides for the services you’ll want to try out. But in general they tend to suffer from over-explanation. I’ll want to do a single thing, but here’s the 10 different variables you could set to do that one thing. It can be maddening in the beginning.

So the next step is to look for tutorials for that very specific thing. There is a growing community of people like me who realized, wow, this is harder than it should be, how about I show people exactly how to do that one thing. This can be great, but it’s hit or miss. What if no one happened to write the one thing tutorial you are looking for? What if the service is in beta mode, so the critical mass of users haven’t even used the one thing you are trying to do, let alone write about it? Well, the next step is asking the providers directly for support.

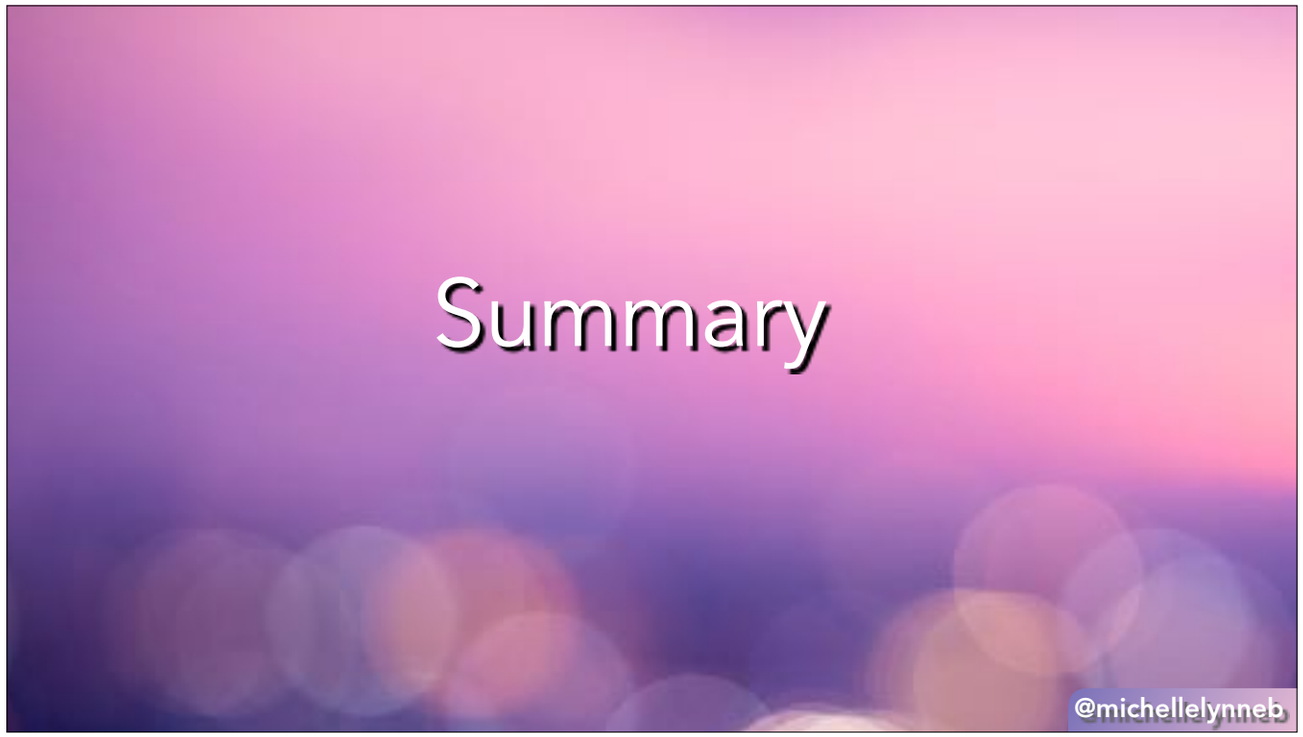
This is where I was let down. I asked on twitter, I asked on forums and I put in tickets. Not alot of traction. This is definitely an area where the providers can do better about answering questions quickly. You can lose all momentum when you don’t get an answer about an error in 2 weeks. It’s important to try and find others who are working on similar projects, either through social media or in-person meetups.

If I could request more support from a cloud provider, I'd ask for a slack group for developers with dedicated newbie channels and support engineers who just love to answer my questions. I did find community slack groups. No answers yet, but I'm hoping they'll get more lively. If anyone has recommendations for where they like to get cloud support, please lmk so I can add to my list.



There was one bright spot of support I found right at the end. I put in a ticket with Google Support and got a response within hours! Apparently the time you spend on the free tier includes support, which is a huge plus for them.

A quick mention on IDE integration. I love using PyCharm and doing everything through there. It’s convenient not to leave the IDE. When a cloud provider integration was announced it makes it even more convenient. However, one thing to keep in mind is once you are depending on your IDE to do things, you have less flexibility. When I had errors in deployment, it was hard to capture what was happening. Another thing to think about is your team. Can you convince everyone you work with to use the same IDE? If you can, what’s your secret, I want to know. For me, it was a nice-to-have when starting up, but you don’t want to plan your work around it. You always want to have READmes that explain all the provider commands you need to run, so it is easier for everyone to use.



And we are finally near the end. I hope this has been as epic a journey for you as it has been for me. As part of this talk, I wanted to make it easier for you to decide which service to use. I've come to some conclusions, but I don't want you to take it as dogma. Services are improving all the time. My goal instead is to give you a framework for deciding for yourself. As a reminder, here's the sections I wanted answers on.

What is available and how do they connect?
How fast can I get started?
How do I programmatically define the infrastructure?
How do I test everything?
How is their CI/CD pipeline?
How do I secure the app?
How can I monitor what is going on?

As you heard, I was able to answer all these questions for all of these services. They share some of the same design patterns and architecture, even if they implement in different ways. So how to decide what to use? Let’s start with some technical questions.

Which language are you using? I used Python because that's the language I am most familiar with, but I bet if I was using .NET, Azure would be the big winner. Go to the provider and see how many examples are in the language you want to use. How many services support it?

Do you prefer tried and true or are you comfortable with a tool in beta? It seems more risky, but then you get more input on the final product. It really depends on how fast you need to go and how confident you are debugging issues.

Next, let’s talk some end goals. How do you want to use it?
Do you work at a company and what to test the waters with a serverless microservice? Then your answer is, whatever provider they already use. I’ve found when trying to convince a team to adopt a new thing, it’s best if its only one new thing. You wouldn’t submit a pull request with 5 new features in it, don’t think you convince your CTO to use a new provider and a new architecture.

Are you instead looking to grow your skills to be more marketable? Look at the companies you are interested in working for, and see what they’re using. AWS has the biggest market share, so if the company uses a different provider, you’d be even more in demand. And of course if you want to work for Amazon, Microsoft or Google, your choices are clear.

Are you looking to create a new business? I would contact each of them directly and ask about their startup programs. How much credit will they give you? Can they slide you some free support to get going?

Are you looking to grow your skills just for fun? Then I say use Azure because their Developer Advocates are the most fun. Also because Microsoft is always the unexpected choice for fun.

Amazon is the OG of the cloud providers and it shows. It has an extensive array of tools and is used by most of the marketplace. But in the fast moving tech world that can also be a downside. The other providers were able to see how they worked and build on them.

Azure has a really interesting design. I like the trigger bindings concept and once I realized how it worked everything else fell into place. It also had the easiest quickstart. But it’s beta for Python and I still haven’t entirely gotten my app to work.

Google uses Flask instead of its own routing system, which was nice to see because it will be a familiar tool for those who already make RESTful APIs in Python. Their error messages were unhelpful, but the fact that I actually got a response to my ticket was a big plus. Their documentation was also outlined with similar sections that I used here, like monitoring and deployment.

It can get very easy to get overwhelmed by the cloud offerings. This talk was much more difficult to work on than I expected. I don't want you to have like 15 tabs of documentation open and think you will never understand this. You will! From working on this talk, I have found most of the documentation is detailed on every single feature, but not great on user story. If you want to find a specific flag on a specific function, it's there, but if you have a story like I want to create an api endpoint with a database you're in for a slog. Hopefully over time more of these user stories get written so it's easier to solve problems.

Here's the repos I started for my experimentation. Like all software projects, they are a work in progress. As I mentioned, this was more difficult than I expected, so there is not as much detail as I like right now. When I have the time, I'll continue to update them. If you're interested in helping with this project, feel free to reach out and I'll consider turning it into an open source project.

Whew. I almost thought we'd never get here. Any final questions?

Finally, if you somehow did not get enough of hearing me talk, I host a podcast called From The Source where I interview people in tech to answer the question of what tech jobs are really like, both the good and the boring.