FEATURE

JAVAWORLD

ng: Freedom for devs at last

Strip away your infrastructure headaches with our clear-eyed guide to serverless and the public cloud and on-premises options fueling its possibilities

By Todd Hoff

InfoWorld |

MAR 6, 2017 3:00 AM PST

Serverless computing provides a great opportunity for developers seeking relief from the burden of infrastructure. By abstracting away everything but a block of code, the serverless model makes it faster for developers to iterate and deploy new code, enabling smaller teams with smaller budgets to do things that only big companies could do before. Or, as Mat Ellis, founder and CEO of Cloudability, recently said in a CloudCast episode, “Serverless attempts to industrialize developer impact.”

Of course, in the background, servers still exist, humming away. But serverless architectures are stateless. They do their work by executing a bit of logic -- a function -- and calling out to other services to do whatever else they need. If you’re a developer who primarily builds applications using services via APIs or needs to respond to events, serverless architecture may prove to be the easiest, fastest, and least risky means for getting the job done.

In this article, we break down what serverless architecture really means, offer a detailed comparison of the major public cloud options, and shed light on a real-world serverless project under way at Coca-Cola.

With serverless, it’s all about the function

Serverless is a cloud computing service model that -- like IaaS, PaaS, SaaS, BaaS, and CaaS -- depends on ubiquitous, convenient, on-demand access to a dynamic shared pool of configurable network, storage, and compute resources. Serverless, however, takes a different approach to using those resources. There is no agreed-upon definition of serverless computing, but a manifesto about the concept has emerged. With serverless, functions are the unit of deployment. No machines, VMs, or containers are visible to the programmer.

Among the chief services available for serverless in the cloud are AWS Lambda, Azure Functions and Google Cloud Functions. In the cloud, "FaaS" (functions as a service) may be

say "function," we really mean a function. Here's an AWS

ode.js:

```
exports.myHandler = function(event, context, callback) {
     console.log("value1 = " + event.key1);
     // do stuff
     callback(null, "some success message");
}
```

That's it. Upload the function and wire it to a request or event. When your serverless hosting provider (in this case, AWS) detects a request has been made (a REST invocation, for example) or an event has occurred (say, a file is added to an S3 bucket), your function will be called with the passed-in arguments and your return parameters will be passed along as the result. It's more complicated in practice, of course, as you can add security restrictions, for example, but that's the essence.

Your function can be written in any language supported by your provider. All you need to do is map an incoming request or event into a function call. Each provider has its own set of supported languages, conventions, procedures, costs, capabilities, and restrictions (see table below). But this is what the Serverless Manifesto means by "bring your own code."

[Enterprise architects lead digital transformations. This PluralSight intro course explains the fundamentals of EA and its latest practices.]

Your function can be arbitrarily complex, and it can call out to included libraries or external services through an API. To be scalable, a serverless function must only use services that are themselves scalable.

Depending on the provider, code can be written directly in an online editor or be uploaded as a code file, a .zip or .jar file, or a container. This is also a downside of serverless because the tooling for uploading and managing your code through the release cycle is still not very good, though numerous frameworks are stepping in to fill the void.

Of course, the code can't truly be arbitrarily complex. There are provider-dependent restrictions. Each host has a max code upload size (50MB for AWS Lambda, for example). Each host has a max function execution time (between 1 and 300 seconds for AWS

Lambda). Each function is also limited in the amount of memory it has available and the more memory, a better CPU, or longer execution time? You difference between providers, as we'll see below.

Behind the curtain: No idle time

Behind the scenes there are servers, of course, but as a developer, you never need to think about them. All you should know is your function. You no longer have to deal with capacity planning, deployment, scaling, installation, patches, and so on. This is often seen as NoOps or LessOps, but really it's DifferentOps. Code still needs to be organized, developed, built, tested, versioned, released, logged, and monitored.

Since all you see is the function, your provider is in charge of activating your functions in response to any requests or events. When a request comes in and a free instance of the function isn't available, the code must be allocated to a server and started. As a developer you do nothing. It's on your provider to ensure there is enough capacity to handle the load. Of course, in the instance of a cold-startup scenario, the latency hit is one of the downsides of serverless.

With serverless, you are paying for consumption and are charged only when your service runs. You never pay for idle compute time. The implications of this are vast. You can build out an entire infrastructure without paying a penny. Operational, development, and scaling costs are all reduced.

Contrast this with PaaS/IaaS, where you pay for capacity. A PaaS provider will manage servers for you, but your application has long-running processes that you're always paying for, even if they sit idle. Capacity can be managed by building a load-sensitive autoscaling system, but you will always be paying for some excess capacity.

What about state?

Since all you see is a function, there's no place to store state. When a function is done executing, its compute resources can be garbage-collected and reused. State must be stored in some other service like a database or a cache.

To control billing costs, providers place a maximum execution time on a function. Currently this is five minutes for AWS Lambda and Azure Functions. There's a downside to execution
ou are upgrading a database? Or your function makes a lot of
rn a result? Or your function has a lot of input to process? These operations can take a lot longer than five minutes in some instances.

Unless your function is simple, functions need to be coded as stateless, idempotent, event-driven services driven by a state machine that hold data in a persistent store between invocations. This can get complicated. Your normal cloud approaches still work. Divide inputs into batches and put them in a work queue, for example.

To help here, AWS has created a new service called Step Functions that implements a state machine and allows functions to last indefinitely. Really, AWS Lambda should be better rather than requiring a completely different and expensive service be used.

Metrics and logging: Challenging at best

Although serverless narrows down the service interface to a single function, sometimes called a *nanoservice*, complexity is always conserved. With PaaS, the atomic unit of compute is the application. This has pluses and minuses. Since an application is large it can be slow to start and stop; it can be hard to scale in response to demand; it can be hard to bill at a granular level. But PaaS is also easy to manage, monitor, and debug.

By spreading out code across a potentially large number of functions, serverless can be really hard to debug. The trace for logic flows is spread across a number of different log files, and there's no connection between the calls. Logging and metrics are required, but it's not enough. Tooling in this area needs to get better.

Serverless providers

How to get started? There are a number of options to begin rolling with serverless. The two broad buckets: using the public cloud and going with an on-premises solution.

Serverless has become table stakes for any public cloud, so the all major providers are developing their own product. Amazon has AWS Lambda, which has been GA since April 2015. Microsoft has Azure Functions, which has been GA since November 2016. Google has Google Cloud Functions, which is in closed alpha.

In terms of deciding between the three, if you are already on the public cloud, it's best to use the product from your current provider. A big part of the usefulness of serverless is the ices and events you can consume. Those choices are best e getting started in the cloud and stability is important, then AWS Lambda has been around the longest and is the safest choice thus far.

For AWS Lambda, functions are independent, though in practice they are often managed in groups using a framework like Serverless. You are charged based on the number of requests for your functions and the time your code executes. If you want more CPU you have to allocate more memory.

With Azure Functions, a Function App is composed of one or more individual functions that are managed together by Azure App Service. A Function App can use either a Consumption hosting plan or an App Service hosting plan. With a Consumption plan, a Function App scales automatically and you pay by memory size and total execution time across all functions as measured in gigabyte-seconds. Using an App Service hosting plan billing is more like EC2.

As Google Cloud Functions is still in closed alpha, much about how functions operate and are billed remains unknown. We know there are two types of functions: HTTP Functions and Background Functions. HTTP Functions are directly invoked via HTTP(S). Background Functions are invoked indirectly via a message on a Google Cloud Pub/Sub topic or a change in a Google Cloud Storage bucket.

	AWS Lambda	**Azure Functions**	**Google Cloud Functions**
Operating system	Linux	Windows	Linux
Suported languages	JavaScript/Node.js, Python, Java, C#	JavaScript, C#, F#, Python, PHP, Bash, Batch, PowerShell	JavaScript/Node.js
Pricing	Free tier (first 1M requests or 400,000GB-seconds of compute time free per month); additional request charges: 20 cents per 1M requests; additional compute charges: 0.001667 cents per GB-second; memory and CPU are linked in Lambda	Free tier (first 1M requests or 400,000GB-seconds of compute time free per month); additional request charges: 20 cents per 1M requests; additional compute charges: 0.0016 cents per GB-second	Unknown
Licensing	Closed source	Open source	Unknown

	AWS Lambda	Azure Functions	Google Cloud Functions
		5 minutes (consumption compute plan)	No limit
Event sources	S3, DynamoDB, Kinesis Streams, SNS, SES, Cognito, Cloud Formation, Cloud Watch, Code Commit, Schedule Events, Config, Echo, Alexa, API Gateway	HTTP (webhook), GitHub (webhooks), Azure Storage (blob, queues, tables), Azure Event Hubs, Azure Service Bus (queues and topics), Azure DocumentDB, Azure Notification Hubs, On-premises (using Service Bus)	HTTP Triggers (HTTP requests), Cloud Pub/Sub Triggers, Cloud Storage Triggers, Cloud Logging, Gmail
Web dashboard	CLI, works well enough	CLI, excellent (Visual Studio Online included)	Cloud Platform Console, CLI (seems to be primary interface)
Log management	Cloud watch	Live event stream (basic)	Stackdriver Logging, Stackdriver Monitoring, Stackdriver Debugger, local emulator
Granular IAM	IAM policy can be associated with Lambda	Not yet	Unknown
Persistent storage	None	Environment variables	Unknown
HTTP invocation	AWS API Gateway (has great flexibility, complex to use)	HTTP Webhooks (automatic, one per function, very simple)	HTTP Trigger (very simple)
Function versioning	Versioning and aliases	No	Cloud Source branch/tag
Memory allocation	Per function	Per App Service	Unknown
Maximum number of functions	No limit	20 per project	20 per project (during alpha)
Concurrent executions	100 parallel executions per account per region (can be raised)	10 instances (which aren't functions; there may be dozens or even hundreds of concurrent executions on a single instance)	No limit
Dependency management	Compile all external packages and Zip the source code	package.json / project.json	NPM package.json
Authentication	AWS Cognito (flexible but complex)	Can authenticate from Microsoft auth and third parties (Facebook, Twitter, Google)	Unknown

	AWS Lambda	Azure Functions	Google Cloud Functions
		5TB, persistent	Unknown
Deployment	Online editing (primitive); Zip upload from local disk or S3	Online editing (primitive); Git (GitHub, Bitbucket); Dropbox; Visual Studio Online (very cool); OneDrive; Kudu Console; Visual Studio Team Services	Zip upload; Cloud Storage; Cloud Source Repositories (a Git platform); GitHub; Bitbucket

Page 1 of 2