CANRIGHT

EXECUTIVE TECHNOLOGY BRIEF

Serverless Architecture:

Increasing Business Competitiveness and Developer Productivity with FaaS Infrastructure

Serverless computing is a method whose time has arrived. Increasingly adopted by software-development organizations moving away from traditional hosted models to cloud-based, virtualized environments, this new architectural and development framework provides several methods that allow developers to build full application stacks without the need to manage servers, speeding new-feature development while reducing infrastructure costs.

Serverless application development takes advantage of the low-cost and high-scale attributes of the public cloud and its pay-as-you-go model. Indeed, serverless development is one of Gartner Group's Top 10 Trends Impacting Infrastructure and Operations for 2019.

This executive technology brief provides an overview of the pros and cons of a serverless software development framework. It focuses on the utility of an cloud architecture models that creates infrastructure on the fly, with a enphasis on AWS Lambda and how it can make software development teams more effective.

What is Serverless Architecture?

Serverless architecture, also known as serverless computing or function as a service (FaaS), is a software design pattern wherein applications are hosted by a third-party service, eliminating the need for server software and hardware management by developers. Applications are broken up into individual functions representing business logic that can be invoked and scaled individually.

For a platform to be considered truly serverless, it should provide the following capabilities:

- No infrastructure provisioning or management necessary on the developer side.
- Application-centric, with development focused on managing APIs and SLAs rather than physical infrastructure.
- Built to run in stateless compute containers that are event-triggered, ephemeral (may only last for a single invocation), and fully managed by a third party.
- Flexible scaling, where applications scale capacity automatically through toggling units of consumption rather than allocating server units—cost is calculated based on what runs rather than servers allocated.

The following terms are relevant to the discussion:

Platform as a Service (PaaS). PaaS is about the outsourced purchase of an entire application server. PaaS brings most of the drawbacks of a hosted server farm—

lack of ability to bring down pieces of the application in response to an event, needs to consider scaling in terms of allocation—while also bringing new drawbacks unique to serverless implementations.

Infrastructure as a Service (IaaS). IaaS is a step even closer to a traditional implementation: it's simply the purchase of remote infrastructure, virtual machines, and other resources, replicating a traditional server farm on servers whose uptime is ultimately in another company's control.

Function as a Service (FaaS). FaaS is about running back-end code without managing your own server systems or your own long-lived server applications. In a FaaS environment, all developers do is deploy code; the cloud provider does everything else necessary for provisioning resources, instantiating VMs, and managing processes. No cluster or VM management is required by the user.

Why Go Serverless?

In traditional, hosted server solutions, a significant portion of development time must necessarily be dedicated to the provisioning and management of physical server infrastructure. This pushes developers to focus not only on business application, but lower-level, largely invisible structural tasks that provide no benefit to the end user.

A serverless infrastructure, by nature increases the potential for innovation: developers can focus most of their efforts on visible application work and business-related features. In brief, serverless architecture outsources those parts of implementation that are not core to delivering business value.

In addition, this sort of turnkey setup drastically reduces lead time for testing new ideas and new innovations. A focus on calling individual functions means that pieces of the application can be tested, implemented, and pushed to market quickly without the need for overarching development to tie these pieces together. This kind of development focus can be described as prioritizing choreography over orchestration. Each coded component plays a more architecturally aware role than in traditionally hosted services—an idea also more common in a microservices approach.

The ultimate decision is therefore what is most important for the needs of the business. There are now many alternatives to on-premises data centers, leaving leaders with a question that brings the decision to focus: are there strategic needs for on-premises servers, or would the business be better served buying rather than building, and taking on the significant investment involved with building a data center?

FaaS also meets ten of the twelve steps in the Twelve-Factor App development method by default—it automatically handles horizontal scaling and security—which builds in portability and resiliency from the front end. This makes it easier to separate data from individual functions, which can then be used to code future functions that serve new products.

Benefits of a Serverless Implementation

Serverless implementation can:

- **Reduce costs, complexity, and engineering lead time.** Because you only pay for the compute you need, there is great support in a serverless system for a granular breakdown of components—a microservices-style approach that can be cost-prohibitive in a more traditional hosting solution. Such cost benefits mean that there are very small operational costs for experimenting with new features; compute times measured in minutes do not make a significant impact on monthly fees.
- **Provide scale-to-zero pricing.** In a FaaS implementation, scaling is automatically managed, transparent, and fine-grained. This differs from container hosting services like Docker and ultimately saves money, as payment in AWS is determined by resources used, with no need to allocate in advance. Serverless architecture, in this way, can be said to scale to zero—if you use zero resources, you pay nothing.
- **Outsource architecture-level development.** This embrace of division of labor results in dramatically reduced costs. The simple reality of shared infrastructure is one element of these savings; the second comes in strong labor-cost gains associated with what would have been in-house infrastructure deployment and long-term maintenance.
- **Minimize lock-in costs.** One of the biggest fears in outsourcing infrastructure is an increased reliance on your service provider: if terms change, it is often thought it could be difficult or sometimes impossible to switch hosts without a total rewrite of all applications. This is not necessarily the case, however. Because FaaS is operation-oriented, an AWS-centric implementation can be built in such a way as to minimize this sort of lock-in. Lambdas are the dominant form of serverless functions, and a good architecture pattern—such as ensuring you choose a cross-platform programming language like NodeJS, Python, or Go—will make any future migrations easier.

Drawbacks of a Serverless Implementation

Drawbacks include:

- **Lack of vendor control.** With any outsourcing strategy, you are giving some control over server functionality.

While AWS features reliable uptime, with any service there is the potential for downtime, and recovery is entirely reliant on the vendor. In addition, security services are entirely dependent on AWS.

- **No in-server state.** FaaS functions have significant restrictions when it comes to local state. You cannot assume that state from one invocation of a function will be available to another invocation of the same function. There is no control over when host containers for functions start and stop. As such, any data that needs to persist must be stored in a stateful backing service, typically in services such as a NoSQL database, out-of-process cache, or an external object/file store. These are slower than in-memory or on-machine persistence.
- **Difficulties in debugging.** All debugging must be handled by running FaaS functions locally. Lambda currently offers no support for debugging functions running in a production environment—nor for debugging functions that trigger on production events.
- **Invocation limitations.** At present, the "timeout" for an AWS Lambda function to respond to an event is at most five minutes before being terminated. Other serverless FaaS implementations have similar limitations.

Implementation in AWS Lambda

AWS Lambda, first introduced in 2014, is the most mature offering in the serverless space. It presents significant advantages to increase the speed of development time, reduce costs, and unify business value with technology.

Lambdas are the dominant form of serverless functions, blobs of code that AWS will run for you in a virtualized environment without having to do any configuration.

As FaaS implementation doesn't rely on an underlying application, to make use of Lambdas you write code in one of several languages and deploy it to the cloud with your Lambda requests designed in. AWS starts your code in a virtualized environment, complete with whatever software packages you required.

Lambdas are like containers—you don't manage storage or the file system directly, that's all set up by initial configuration. However, unlike a container, you also don't directly manage Lambda startup, responses, or routing directly; you leave all of that to AWS.

This offers a lot of flexibility; you can set up your code to automatically trigger from other AWS services, or call it directly from any web or mobile app. AWS offerings can also be implemented "first class" in JavaScript, Python, Go, any JVM language, or any .NET language

There are some design challenges unique to a serverless approach. As noted, in any outsourced implementation there's no local control over response to downtime. Thus, the best practice for reliable implementation is to make your code truly serverless: pieces of your app must be spread across AWS availability zones, minimizing the possibility that more than a few individual functions of your overall application can be unresponsive at any given time.

AWS Lambda v. Other Cloud FaaS Providers

Across the board, the theme is the same: AWS simply has the most mature offering on the market. Microsoft and Google are both playing catch-up to features that AWS has had integrated and connected for years. That maturity leaves AWS in a position of spending its next five years adding new features, whereas the others will have to spend at least a portion of that time simply seeking parity.

Based on a private analysis given by a consulting firm that works with the leading cloud providers, it appears that both Microsoft and Google are shifting their serverless strategies. AWS is so far ahead that its competitors are expected to focus on providing unintegrated open-source building blocks that can be tied together, as opposed to the integrated function approach that AWS has taken.

Another major difference between AWS and other cloud providers is that AWS has done a lot to integrate enterprise authentication levels. You bring your own provider and they can plug and play and bridge into their own permissions systems making it more possible for enterprise to use their own public cloud services without losing operational control. With others, the decision comes down to whether their authentication systems are a fit; there is little flexibility on importing a different system without heavy modifications.

Microsoft's Azure Functions is tightly integrated with Visual Studio, like its other developer-focused products. Given this integration, Azure is widely regarded to be a "simpler" implementation in comparison to AWS Lambda, but that simplicity also translates to a comparative lack of flexibility. AWS is highly configurable, with a host of features that Azure simply lacks.

Storage cost comparisons—and flexibility in options—also breaks in favor of AWS. Both services make use of basic block storage, but AWS's pricing tiers are more elastic, with flat fees of $.045 per GB for hard-disk drive or solid-state drive at $.10 per GB. Azure charges $.05 per GB for HDD, or a flat fee of $19.71 for 128 GB per month on SDD. The price difference for

HDD is already noticeable over many operations, but it's this comparative lack of flexibility in SDD storage that really breaks in Amazon's favor.

Indeed, AWS cost savings have been significant for one company—and will be even more so with serverless operations. In one month, the company paid AWS $1,380 for services. The team estimates that eliminating AWS services that its serverless implementation would not require (EC2, Tomcat, and Beanstalk) could cut that nearly in half.

AWS also features stronger DevOps support than Azure. Azure's offering, which is relatively young—launched in September of 2018—is still far from feature-complete, and focused on the Microsoft method of development.

Most critically, and perhaps least surprisingly, Azure also lags AWS in terms of open-source support. While Azure made some uncharacteristic early strides in this area—the Kubernetes container orchestration system is open source—AWS has that philosophy at its core. Its services are largely built on open source (MySQL, Tomcat), and it has contributed large amounts of code to projects in Xen, Linux, Docker, Chromium, MXNet, etc.

Components of a Successful Serverless Implementation

FaaS development is a radical departure from traditional hosted development structures. While this offers major opportunities in terms of potential innovations, it also requires a different approach to team structure to maximize the potential benefits.

A well-balanced structure should include programmers to write functions and manage their source code, cloud professionals to assign and control the resources these functions use, and operations and security to deploy these stacks to the correct environments.

Critically, these teams have interlocking responsibilities; siloing is actively counterproductive, as interaction and coordination will be needed for releases, updates, and emergencies.

Conclusion

Despite the drawbacks inherent to any outsourced approach, serverless architecture's reduced operational and development costs, easier operational management, and reduced feedback loop for creating new application components represent significant advantages over a hosted solution.

The shift to a pay-as-you-go model represents significant—and immediate—short- and long-term savings. Even beyond the advantages of only paying for used compute cycles, the benefits of outsourcing the long-term costs in development, maintenance, and replacement of components in a traditional hosted server farm are potentially enormous.

The most critical value-add, however, lies in the fundamentally different approach it supports for developers. Bringing developer focus exclusively to the application side brings business value and technology together: it is a streamlining of the development model to focus exclusively on adding business value to applications.

Schedule a consultation for an Executive Tech Brief about your software solution. Contact Collin Canright at collin@canrightcomunications.com.