

Blog / Google Cloud Platform

Google Cloud Functions VS AWS Lambda: fight for serverless cloud domination begins



[Alex Casalboni](#)

| 22 Mar, 2016

[AWS](#)

[Azure](#)

[Google Cloud](#)



A not-entirely-fair comparison between alpha-release Google Cloud Functions and mature AWS Lambda. My insights into the game-changing future of serverless clouds.

Serverless computing lands on Google

Cloud: welcome to Google Cloud Functions

The Alpha release of [Google Cloud Functions](#) officially released in February, as part of the Google Cloud Platform solution.

This new Cloud service aims at relieving most of the pain caused by server maintenance, deployments, and scalability. It perfectly aligns with the [serverless revolution](#) started by AWS Lambda back in 2014.

Serverless means that you can focus on your application logic without dealing with infrastructure at all (almost). Painless development, deployment, and maintenance of a web API is still not turn-key, although modern web application frameworks have improved dramatically in the last 5 years.

Serverless computing is definitely a game changer. Consider the endless possibilities offered by the [event-driven approach](#) combined with the rich Cloud ecosystem offered by the main Cloud vendors, AWS, Microsoft Azure, and Google Cloud Platform.

Here I would like to discuss the upcoming features of **Google Cloud Functions** and compare them with the current status of AWS Lambda. I'll provide some basic examples of how you'll be able to migrate, and then test your Lambda functions on Google within minutes. I want to explore what serverless computing may look like in just a few months.

Google Cloud Functions & AWS Lambda

First of all, I have to admit that, comparing an alpha release with a 2-year-old stable product is not completely fair. That said, I believe some of the functionalities already offered by Cloud Functions will make a substantial positive difference — especially from a development point of view.

Here is a quick recap of the main functionalities of both products:

FUNCTIONALITY	AWS LAMBDA	CLOUD FUNCTIONS
Scalability & availability	Automatic scaling (transparently)	Automatic scaling
Max # of functions	Unlimited functions	20 functions per project (during alpha)
Concurrent executions	100 parallel executions per account per region (default safety throttle)	No limit
Max execution time	300 seconds (5 minutes)	No limit
Supported Languages	JavaScript, Java and Python	Only Javascript

Dependencies	Deployment Packages	npm package.json
Deployments	Only ZIP upload (to Lambda or S3)	ZIP upload, Cloud Storage or Cloud Source Repositories
Versioning	Versions and aliases	Cloud Source branch/tag
Event-driven	Event Sources (S3, SNS, SES, DynamoDB, Kinesis, CloudWatch)	CloudPub/Sub or Cloud Storage Object Change Notifications
HTTP(S) invocation	API Gateway	HTTP trigger
Logs management	CloudWatch	Cloud Logging
In-browser code editor	Only if you don't have dependencies	Only with Cloud Source Repositories
Granular IAM	IAM roles	Not yet
Pricing	1M requests for free (Free Tier), then \$0.20/1M requests	Unknown until open beta

Let's dig deeper on each functionality.

Scalability, availability, and resource limits

Of course, this is the primary focus of both services. The key feature promises that you no longer need to worry about maintenance, downtime or bottlenecks.

As far as AWS Lambda, scalability is completely and transparently handled by the system, meaning that you don't know how many instances or machine your functions are running on at a given time. You can monitor your Lambda functions usage anytime, but your visibility of the underlying architecture is limited.

At a different extreme, Google Cloud Functions explicitly creates a set of instances in the Cloud. This way you can always check the number of machines created and monitor the load of your cluster.

Besides scaling and monitoring, there are some other AWS Lambda limitations. For example, in AWS Lambda you can create an unlimited number of functions, but each execution can't exceed 5 minutes (it used to be much shorter!) and you are limited to [100 parallel executions](#) per account per region, as a **default safety throttle** which can be increased upon request. Furthermore, your zipped deployment packages can't exceed **50MB** (250MB when uncompressed). There are a few more [AWS Lambda Limits](#), but I think they actually affect only very specific scenarios so I won't name them here.

On the other hand, Google Cloud Functions doesn't seem to impose such limitations (yet?), in spite of having a hard limit of **20 functions per project**. I would expect this limit to eventually disappear

though.

Supported Languages and Dependencies management

The first version of Lambda only supported JavaScript, then later it included Java (Jun 2015) and Python (Oct 2015). Currently, you can even write your functions in Ruby (with [JRuby](#)), or any unsupported language by [running arbitrary executables](#) (i.e. by spawning child processes).

Google Cloud Functions currently supports only JavaScript. Although there seems to be no public roadmap. I would expect Python and Java to be supported sometimes later this year.

As far as **dependencies management** and deployment, I think Deployment Packages are the only real weakness of AWS Lambda. In practice, you can use external dependencies only by including them within your zipped source code, and I find this is inconvenient for many reasons.

First of all, you are forced to compile and install these external packages on the same OS used by AWS Lambda internally. After this you must upload all together every time you need to change something in your own code. Second, this is not the way modern dependency management works: web developers are now used to declaring and versioning their code dependencies, rather than providing local compiled libraries.

Of course, the whole process can be automated, and **wouldn't a configuration file be easier and safer to maintain?**

Yes. :)

In fact, Google Cloud Functions allows you to define a simple ***package.json*** file to declare and version your npm dependencies. As soon as Python is also supported, I expect the ability to simply deploy a *pip requirements.txt* file. Let's see what happens.

Deployments and Versioning

As I mentioned in the previous section, I don't personally like the way AWS Lambda handles deployment packages and dependencies. It forces you to re-deploy a (potentially huge) deployment package every time you change your code or update a dependency.

On the other hand, I love the possibility of having **multiple versions** of the same Lambda function. This makes deploying and testing a new version very easy – even from the AWS Console UI. The real trick is **binding versions to aliases**, so that you can easily switch to new versions (or rollback to older ones) with just a couple of clicks. Linking stable versions of your functions to API Gateway stages – such as `dev`, `stage`, `prod`, etc. – requires a little bit of manual configuration, but it's totally worth it. I recommend setting-up an API Gateway stage variable and using it to invoke given AWS Lambda

aliases.

On this front, Google Cloud Function chose not re-invent the wheel and devised a developer-friendly solution: **you can achieve versioning with git** (i.e. a given branch or tag), even though you would need to host your repo on **Cloud Source Repositories**. I'm looking forward to a more general solution, hopefully including other mainstream git solutions, such as GitHub, BitBucket, etc.

Invocations, events, and logging

Both AWS Lambda and Google Cloud Functions support the event-driven approach, meaning that you can trigger a function whenever something interesting happens within the Cloud environment. They also support a simple HTTP approach.

AWS Lambda can be invoked by nearly every other AWS service, including S3, SNS, SES, DynamoDB, Kinesis, Cognito, CloudWatch, etc. You can configure API Gateway to invoke a given Lambda function and obtain a RESTful interface for free (almost), including authentication, caching, parameters mapping, etc.

Google Cloud Functions currently only supports internal events by Google Cloud Storage (i.e. Object Change Notifications) and through Google Cloud Pub/Sub topics (Google's globally distributed message bus that automatically scales as you need it). **HTTP invocations** are already natively supported.

You simply need to deploy your function with a *trigger-http* flag. Currently, you need to explicitly configure and deploy your Google Cloud Functions for each different trigger type.

As far as logging, both services are well integrated with their corresponding logging management services: **Amazon CloudWatch** and **Google Cloud Logging**. I personally find CloudWatch better integrated, better document and with easy-to-configure charts (kind-of).

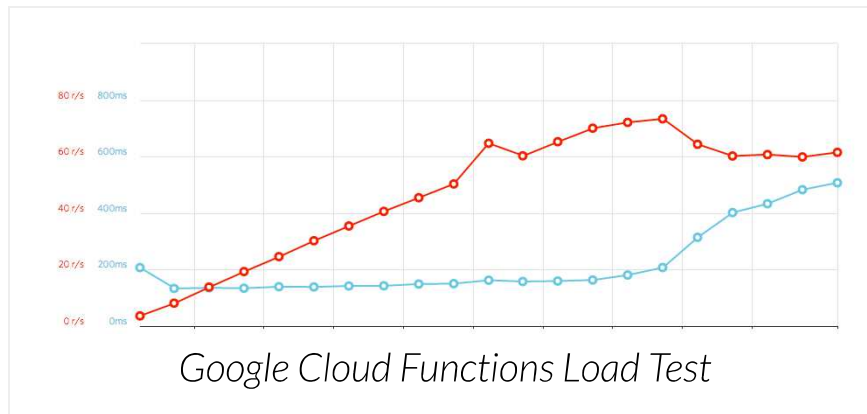
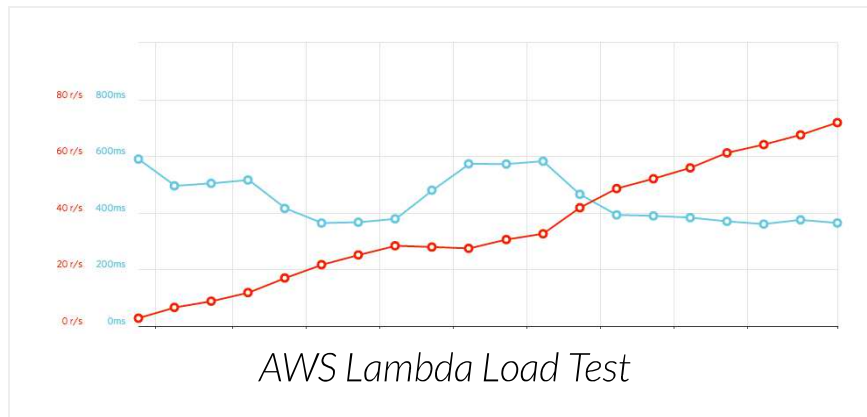
Load testing and statistics

I took the time to perform some load tests on [arbitrary JavaScript](#) involving pure computation (i.e. generate one thousand md5 hashes for each invocation). This presented me the opportunity of playing with the two different dependencies management systems because I needed to include the *md5 npm* module.

I configured a linearly incremental load of 5 minutes, up to about 70 requests per second.

Below I have plotted the two charts, representing the **average response time** and the average number of **requests per second**.

Please note, that these two charts use the same scale for both dimensions. Also, I've deployed and tested both functions in the corresponding EU-west region (Ireland).



As you can see, there is a **noticeable difference in the average response time**: Google Cloud Functions consistently keeps it between 130 and 200ms, with a strange increase during the last minutes of my test (maybe due to the decreasing load?).

On the other hand, AWS Lambda's response time is much higher and reveals an interesting rectangular pattern: the service seems to internally scale up after the load reached 20 req/s. When the load stabilizes around 30 req/s, it seems to scale down (i.e. response time raised to 600ms) and then it scales up again with a load of 40+ req/s.

Since each function invocation is returning a relatively **heavy JSON response** (almost 50KB), I assumed network performance had an impact on the resulting response time, independently of the actual

computation.

I quickly verified this assumption and modified both functions to return a simple “OK” message. I noticed a consistent improvement for the new AWS Lambda function, whose response time dropped between 200 and 300ms. The new Google Cloud Function was not drastically affected, but its response time dropped to only 100ms.

Given these results, I would say that the computational difference is still relevant, but network probably has a bigger impact if your application involves heavy HTTP responses. Apparently – [as we already discussed back in 2014](#) – Google’s networking just works better, and I would assume the native HTTP integration is faster as well – with respect to Amazon API Gateway. Google’s networking is great but still lacks critical features such as authentication and caching.

Function Code compatibility between AWS and Google

Unfortunately for us, AWS Lambda and Google Cloud Functions are not directly compatible with each other. As I mentioned before, Google Cloud Functions is still in alpha and things can change, but I assume Google won’t make the effort of being compatible with AWS Lambda without compelling reasons.

If you already have a few Lambda functions, in most

cases, they are also interacting with at least another AWS service. Your Lambda functions are probably using some IAM roles, and plenty of AWS details so you wouldn't easily migrate to another Cloud vendor anyway.

In plenty of other cases though, your Lambda functions involve pure computation or simple input/output logics (i.e. read from a queue, write into a database, process an image, etc.). In these cases you may be tempted to try your Lambda functions on Google Cloud Functions as well — even if just to evaluate the service, or shrink your costs. [Here](#) you can request your account to be whitelisted.

What about an automated conversion tool?

Luckily, I took the time to develop a simple [conversion tool](#) that will definitely speed up the porting process of your JavaScript Lambda functions. It correctly handles the event/context functionalities mapping, and automatically comments incompatible attributes and methods. I really hate manual refactoring or porting tasks, so I hope it will be useful for some of you.

For example, a very simple function like the following:

```
1 exports.myHandler = function(event, context) {  
2  
3     console.log("input data: " + event);  
4  
5     if (!event.name) {  
6         return context.fail("No name");  
7     }  
8  
9     context.succeed("Hello " + event.name);  
10  
11 }
```

would become something very similar to this:

```
1 exports.myHandler = function(context, data) {  
2  
3     console.log("input data: " + data);  
4  
5     if (!data.name) {  
6         return context.failure("No name");  
7     }  
8  
9     context.success("Hello " + data.name);  
10  
11 }
```

As you can guess, the conversion is quite intuitive and shouldn't take you more than a couple of minutes, although things start to get way more complicated and time-consuming if you have a very complex function (especially if you defined additional utility functions that require both the original event and context objects).

Alpha testing Conclusions

Google Cloud Functions looks very promising, and I am looking forward to the long list of upcoming features. I will continue running tests and monitoring the Cloud Functions trusted testers group, which already contains plenty of suggestions, improvements, and feedback. I personally hope to see many more tools that will enable cross-platform development in a serverless fashion.

If you enjoyed the article, feel free to drop a comment and let us know what you think of the serverless revolution. We are happily using Lambda Functions in the Cloud Academy platform as well, and we can't wait to see what will happen in the near future (if you don't know it yet, have a look at [the Serverless Framework](#)).



Alex Casalboni Alex is a Software Engineer with a great passion for music and web technologies, experienced in web development and software design, with a particular focus on frontend and UX. His sound and music engineering background allows him to deal with multimedia, signal processing, machine learning, AI and a lot of interesting tools, which are even more powerful when you merge them with the Cloud.





Posted on: 22 Mar , 2016

[Comments](#)[Community](#)[Login](#) ¹[Recommend](#)[Share](#)[Sort by Best](#)

Be the first to comment.

[Subscribe](#)[Add Disqus to your site](#) [Add Disqus](#) [Add](#)

Browse Courses, Quizzes, and Hands-on Labs

PATH	COURSE	COURSE	46n	COURSE	62n	LAB	19n	COURSE	45n	LEARNII
Security Fundamentals for AWS	Security Fundamentals for AWS	Security Fundamentals for AWS	Security Fundamentals for AWS	Networking Fundamentals for AWS	Networking Fundamentals for AWS	Private Cloud (VPC)	Private Cloud (VPC)	Private Cloud (VPC)	Private Cloud (VPC)	Private Cloud (VPC)
 Andrew Larkins	 Andrew Larkins	 David Clinton	10 GUIDED STEPS	 Ben Lambert						
★★★★☆	★★★★★	★★★★☆	★★★★☆	★★★★★						
AWS	AWS	AWS	AWS	Google						



AWS Popular Articles



Choosing the Right AWS Certification for Your Career



[Sam Alapati](#)

AWS Lambda: an introduction and practical walkthrough



[Vineet Badola](#)

Amazon Machine Learning: Use Cases and a Real Example in

[Browse Blog](#) [How it works](#) [Content Library](#)
[Learning Paths](#) [Cloud Academy for Teams](#)



© Copyright 2017 Cloud Academy Blog