SERVERLESS CLOUD COMPUTING: A COMPARISON BETWEEN "FUNCTION AS A SERVICE" PLATFORMS

Víctor Juan Expósito Jiménez and Herwig Zeiner

JOANNEUM RESEARCH
Forschungsgesellschaft mbH, Steyrergasse, Graz, Austria

ABSTRACT

Cloud computing is moving fast and continually progressing. Beyond the microservices architecture, a new paradigm appears to be evolving and complementing it. By using a serverless computing architecture, faster and more reliable developments are possible in several fields such as the Internet of Things, industrial or mobility applications. In this paper, a serverless computing architecture is described and, in addition, a comparison of the most important "Function as a Service" platforms is given.

KEYWORDS

Serverless Architecture, Function as a Service (FaaS), Cloud Computing, Microservices

1. INTRODUCTION

The Internet of Things (IoT) has changed the way applications are designed. Billions of devices will be connected to the network in a near future. The next generation of connected applications has to change the way they have been designed to support this exponential growth of connected devices and their corresponding services.

At the moment, a microservices architecture can fulfill these requirements where the combination of different services is a main issue. In this kind of cloud computing architecture, all tasks and the logic are split in small services, where each is able to serve just one specialized purpose instead of one monolithic component for all purposes. This allows a more versatile development in which maintenance and updates can be done only to the needed components without changing the whole core system. Every service is isolated and all information and all controllers are accessible through external APIs. The popularization and the standardization of RESTful APIs give developers the ability to use some external services which can take too many resources for a self-implementation. Moreover, reliability, security or scalability are some of the few constrains modern applications have to handle.

Going a step forward, what if there was a way to build a pure native cloud application in which clients (i.e. mobile applications) make all requests and handle the logic of the result without any central server? This type of design is called a serverless computing architecture. This concept will be possible when more cloud services are available. In addition, cloud computing services have grown exponentially in the last decade. For example, Amazon Web Services (AWS) and Microsoft Azure are constantly updating their cloud services to offer more possibilities to satisfy all requirements that developers could be needing to design and develop applications. In this paper, a detailed analysis of the current status of serverless cloud computing architecture and Function as a Services (FaaS) platforms is given.

The next section gives an overview of existing related work in the field of cloud computing architectures. Section 3 explains the basics of serverless architecture and Function as a Services (FaaS) platforms. In section 4, a brief description of each provider is given. The results of the evaluation are discussed in detail in section 5. Finally, section 6 concludes this work.

2. Related Work

There are many publications which deal with modern architectures and their usage to build cloud-based applications in different fields. Adrian Cockfort [1] explains how modern cloud platforms help companies to speed up the developments and why Netflix decided to move to cloud services to increase the availability and the reliability of the platform. Alan Sill [2] gives some guidelines for designing an application by using a microservices architecture. The authors describe in [3] how the microservices architecture can be used to build a web of things platform. This platform uses some of the principles that define a serverless architecture. Moreover, the authors in this paper are focused on the time constraints and the effect of the delays between microservices on the platform behaviour [4] describes in detail the process and the advantages of using this kind of architectures for an Internet of Things application. Finally, [5] characterizes the implementation of microservices for critical applications in which the sensitive data is handled, integrity and confidentiality are key points.

An important point to consider when designing a cloud application is the cost of the different external services. Andy Singleton [6] discusses how microservices can help companies to avoid errors when a program is too big to be successfully maintained, and the costs that can be saved once a microservices architecture is implemented. He also explains in which cases moving to a microservices architecture makes sense. [7] explains all parameters and costs that developers have to keep in mind when building resilient cloud services.

3. Serverless Computing

A serverless architecture expands the concept where developers do not have to worry about a central server. Some scenarios where this kind of architecture can be successfully implemented are shown in book [8]. Developers can design a complete application with the combination and the communication between third-party services, native cloud services and self-enabled components that give them the flexibility to choose the best service which fulfills the requirements of the application as well as faster development, since an existing service can be instead of developing a one.

On the other hand, the usage of third-party services makes development more dependable on external conditions which are not under the developer's influence. For example, if a provider changes the API of a service, the developer has to modify this aspect on the application or the latter will not be able to connect in future releases. Moreover, all major companies such as

Microsoft or Amazon, which already have a complete cloud services suite, are trying to integrate all services to their own platforms. Therefore, the migration from one provider to another can also be difficult for the interconnection of services from different providers.

In the context of a serverless architecture, two scenarios can be identified:

Backend as a Service (BaaS): Usually a third-party component which provides complete service to the application. In this case, all of the business logic is carried out by the service and the client just gets the results of the task.

Function as a Service (FaaS): The service just runs a small piece of code and gives the result of the function to the client.

In this paper, the main focus is on the analysis and the comparison of FaaS platforms. The usage of FaaS has some advantages in comparison to other types of applications. In traditional applications, developers have to consider all aspects of the environment in which the code is running, like hardware or operating system versions. Once a FaaS platform is used, this procedure is completely transparent to them. Furthermore, no additional time is needed to consider the amount of resources the code needs to be successfully executed on a computer. For instance, the application's design keeps in mind average requests per second to assure the needed hardware is able to handle them with a minimum performance. At this point, there are always some request peaks that may compromise the performance of the whole system due to a lack of resources to handle such an unexpected amount of requests. When it comes to this kind of situation, a FaaS platform comes in handy since the platform automatically handles these request peaks in a transparent way to guarantee a good performance. So developers only have to worry about the application performing well.

When comparing FaaS and PaaS (Platform as a Service), both seem similar at first glance, but on closer look, one separating key aspect becomes evident. According the article [9], Mike Roberts explains that the main difference between both services is the scalability. When PaaS is used, the developer has to keep in mind the resources in case the application receives a request peaks. This process is transparent, however, if FaaS is used.

4. PLATFORMS

This section gives an overview of the principal FaaS platforms as well as a description of the functionality and the special features each one implements.

4.1. AWS Lambda

The Amazon cloud provider was the first to offer a FaaS platform at the end of 2014. Lambda is offered as an isolate service from other AWS services and the price is calculated on the basis of two parameters: the number of executions and the execution computing time according to a defined memory. Amazon also includes a free tier per month before charging costs. AWS Lambda is able to run code of the following programming languages: Java, JavaScript, Python and, since November 2016, C#. The service is strongly integrated with the Amazon Web Services and functions can be triggered by other services too, like Kinesis, S3 or DynamoDB. Unfortunately, a HTTP trigger is limited to the Amazon API Gateway Service which may add complexity and some delays. One of the flaws of Lambda is the way in which dependencies are handled. There are no configuration files in which developers can version a function or define its dependencies. Therefore, a complete package must be uploaded every time the function changes.

Table 1. Comparison between Function as a Service platforms.

	AWS Lambda	**Azure Functions**	**Google Cloud Functions**	**IBM OpenWhisk**
Resease Date	Nov 2014	Nov 2016	Beta	Feb 2016
Price	$0.00001667/GB-s and $0.20 million execs	Azure subscription plan or $0.000016/GB-s and $0.20 Million Execs	$0.0000025 GB-s and $0.40 Million Execs	IBM Bluemix plan or $0.000017/GB-s
Monthly Free Tier	400,000 GB-s and 1 million execs	400,000 GB-s and 1 million execs	400,000 GB-s and 2 million execs	400,000 GB-s
Maximum time to execute	300 seconds	N/A	540 seconds	300 seconds
Compatibility	Java, JavaScript, Python	JavaScript, Python, PHP, C#, F#, bash, batch	JavaScript	JavaScript, Python, Swift, Docker
Available memory usage	128MB-1536MB	128MB-1536MB	128MB-2048MB	128MB-512MB
HTTP trigger	API Gateway	Native	Native	Native
Open source	No	Yes (Runtime)	No	Yes (Runtime)

4.2. Microsoft Functions

Previously known as a part of Azure WebJobs, Azure Functions was released in November 2016 as an isolated service within the Azure cloud suite. With the release, the related runtime was also published as open source under MIT license and is available in its GitHub repository [10]. The fees of the service follows the same calculation rates as AWS Lambda and it also includes a free grant with the same features. Unlike Lambda, Azure Functions service can be purchased as a pay per demand model or as a part of an Azure subscription plan. Besides the compatibility with common programming languages such as C#, F#, and JavaScript; Functions is also able to execute scripts that use the Windows Command Line, the Power Shell syntax as well as the more common PHP and Python functions which provide more flexibility to the developers; unfortunately, these are still in an experimental status. Moreover, the functions are accessible through HTTP without using any API gateway.

4.3. Google Cloud Functions

Like the other big cloud players, Google has also developed its own FaaS platform called Google Cloud Functions. This service is still a beta release, so its possible functionality and features will be extended in the release version. Unlike other providers, Google Cloud Functions currently supports only JavaScript. On the other hand it is different to the other services. The service allows the usage of more memory, in this case, a maximum of 2GB, and also the free tier gives 1 million executions more than the Azure Functions or AWS Lambda. Moreover, the maximum time of execution for a function increases until 9 minutes. Another interesting aspect of Google Cloud Functions is the possibility of calculating the bill of the service by using GHzseconds rather than the otherwise commonly used GB-seconds.

4.4. IBM OpenWhisk

Starting in February 2016, IBM OpenWhisk was the first open source FaaS, followed by Microsoft with its Azure Functions later. The runtime code is available at GitHub[11] under Apache 2.0 license. IBM OpenWhisk has two price modes, per demand or associated to an IBM BlueMix plan. OpenWhisk supports JavaScript, Python and also adds a compatibility with Swift, the Apple programming language. So far, it is the only service that implements this language and it could be interesting for iOS developers. Another feature that excels this service is the possibility of using Docker containers to run any function or the native implementation of artificial intelligence service, IBM Watson.

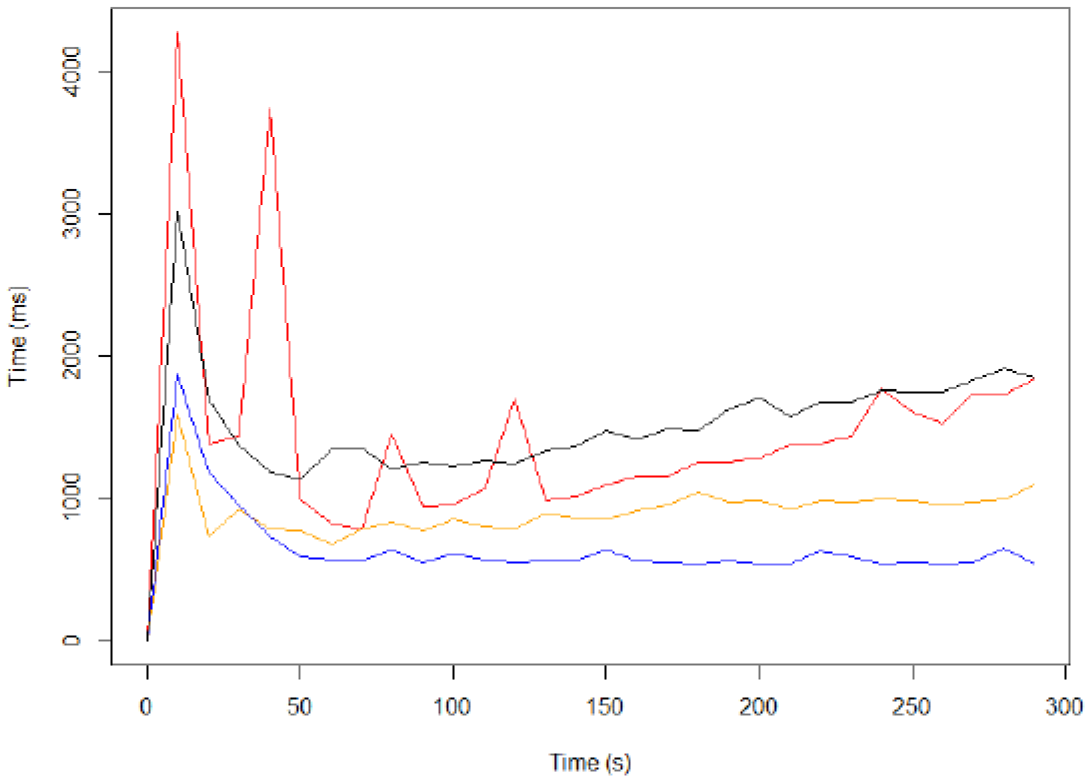


Figure 1. Mean RTTs when the response includes the random generated string by the function. Azure Functions: blue line, IBM OpenWhisk: black line, AWS Lambda: orange line, Google Cloud Functions: red line

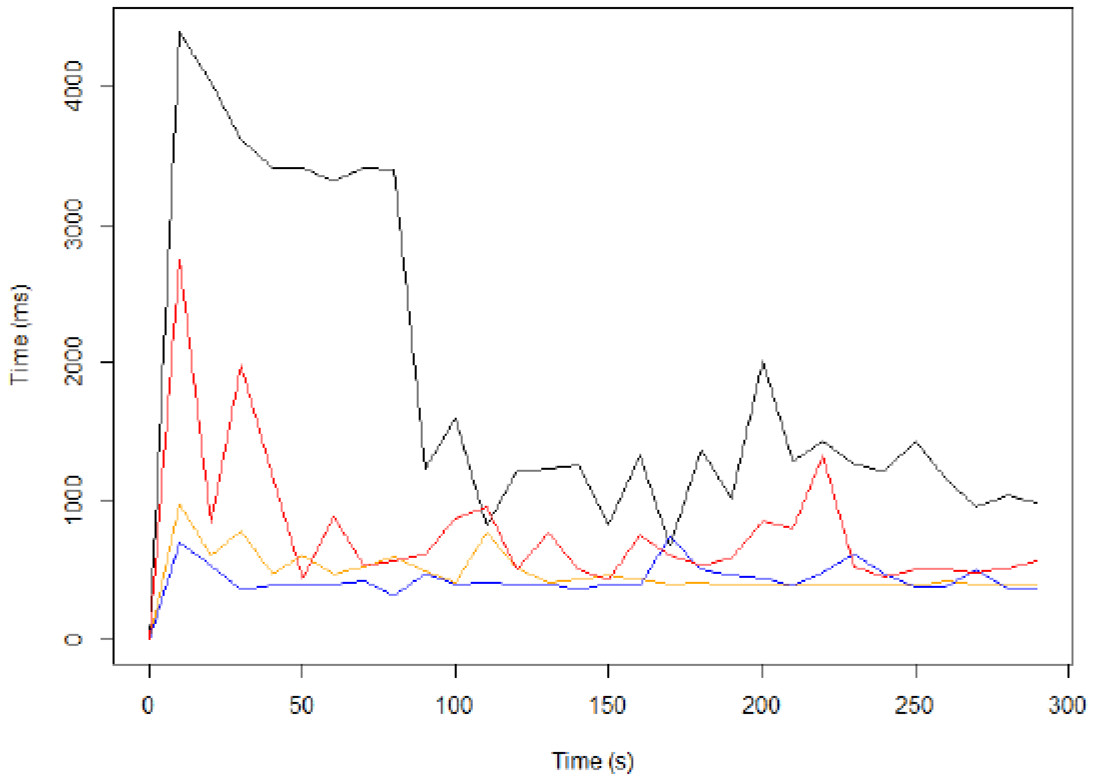


Figure 2. Mean RTTs when the response has an empty body. Azure Functions: blue line, IBM OpenWhisk: black line, AWS Lambda: orange line, Google Cloud Functions: red line

5. EVALUATION

Table 1 summarizes all features of all services that have been described in this paper; for a more detailed evaluation, a simulation has been done. Therefore, a small JavaScript function which generates some computer processing by the calculation of random strings was programmed. JavaScript, being the most supported programming language, was chosen as function programming language. Besides, one point has to be considered during the simulation. At the beginning, we considered as one of the simulation´s parameters the usage of the closest server to our location (i.e. Austria). Unfortunately, OpenWhisk and Google Cloud Functions are still not available in Europe, so all platform simulations had to be done by using servers located in the United States (US). Azure Functions, Google Cloud Functions and IBM OpenWhisk are located in the south central US, Amazon Lambda in the east.

The simulation consists of a client which sends HTTP requests to trigger the FaaS function. Starting with no requests, the requests rate per second increases every ten seconds by adding ten more requests per second. At the end of the simulation (after five minutes), the request rate is 300 req/s which represents a linear increase rate. The reason for increasing the simulation requests is to see how the platform scales when more simultaneous requests trigger the function. The figures show the mean time of the Round-Trip Times (RTT) the platforms give in two determined cases. Figure 1 depicts the mean RTTs when the response includes the random generated string by the JavaScript function. Figure 2, on the other hand, shows the mean RTTs when an empty response body is given. These two cases have been chosen to see how the body (approximately 80KB) response affects to delays.

All four platforms depict stable behavior with just some peaks at the moment before the platforms scale themselves. The peaks are especially visible in the case of Google Could Function when it scales itself as being clear. Azure Function gives the best performance, both with and without response body, which is always around 500ms; on the other hand, IBM OpenWhisk provides the worst values which are always higher than 1000ms. Amazon Lambda provides a good RTT performance when there is no response body in the request, but it is worse once the randomly generated string is included in the response body and more requests are sent. This may be the result of the required configuration of the API Gateway to enable the HTTP trigger on the Lambda platform. Google Cloud Functions does not present a good performance once the body is included in the response, but the behavior is similar to Azure or Lambda when the response body is empty. One point is especially interesting: all figures show a peak when the simulation starts but stable themself once the simulation continues, which could mean that the platforms need a small period of time to initiate.

According to all analyzed data, the Azure Function platform offers a better performance and supports more programming languages. Moreover, the price is the same compared to the closest competitor, AWS Lambda. IBM OpenWhisk yielded a worse performance but is cheaper than the other platforms and the compatibility with Docker containers, Swift and IBM Watson could be interesting features for some developers. Finally, Google Cloud Functions presents a middle point in features, performance, and price to the others competitors.

6. CONCLUSION

The paper gives an overview and analyses on the current state-of-the-art of serverless cloud architecture. Afterwards, a comparison between the most used FaaS platforms was described in which some parameters were analyzed. These parameters will be used in the future to choose the most suited platform for our requirements. This paper is a basis which will be continuously updated as an extended live document to reflect the changes that may occur in the future on these platforms. The next step will be a more technical comparison of the platforms in a real life scenario, where more precise requirements in delays, programming languages and price are given.

ACKNOWLEDGMENT

This research was partly funded by the Comet-Project DeSSnet- Dependable, secure and timeaware sensor networks. The K-Project DeSSnet is funded within the context of COMET – Competence Centers for Excellent Technologies by the Austrian Ministry for Transport, Innovation and Technology (BMVIT), the Federal Ministry for Digital and Economic Affairs (BMDW), and the federal states of Styria and Carinthia. The programme is conducted by the Austrian Research Promotion Agency (FFG).

REFERENCES

[1] S. Tilkov, “The modern cloud-based platform,” IEEE Software, vol. 32, no. 2, pp. 116–116, Mar 2015.

[2] A. Sill, “The design and architecture of microservices,” IEEE Cloud Computing, vol. 3, no. 5, pp.76–80, Sept 2016.

[3] H. Zeiner, M. Goller, V. J. Expósito Jiménez, F. Salmhofer, and W. Haas, “Secos: Web of things platform based on a microservices architecture and support of time awareness,” e & i Elektrotechnik und Informationstechnik, vol. 133, no. 3, pp. 158–162, 2016. [Online]. Available: http://dx.doi.org/10.1007/s00502-016-0404-z

[4] B. Butzin, F. Golatowski, and D. Timmermann, “Microservices approach for the internet of things,” in 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Sept 2016, pp. 1–6.

[5] C. Fetzer, “Building critical applications using microservices,” IEEE Security Privacy, vol. 14, no.6, pp. 86–89, Nov 2016.

[6] A. Singleton, “The economics of microservices,” IEEE Cloud Computing, vol. 3, no. 5, pp. 16–20, Sept 2016.

[7] B. Wagner and A. Sood, “Economics of resilient cloud services,” in 2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Aug 2016, pp. 368–374.

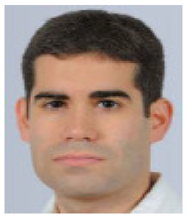
[8] P. Sbarski and S. Kroonenburg, ”Serverless Architectures on AWS”. Shelter Island, New York: Manning Publications Co.,2017.

[9] M. Roberts. (2016, Aug.) “Serverless architectures”.ThoughtWorks, Inc. [Online]. Available: http://martinfowler.com/articles/serverless.html [Accessed: 02- 05- 2018]

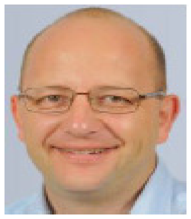
[10] Microsoft Corporation, “WebJobs.Script”. [Online]. Available: https://github.com/Azure/azurewebjobs-sdk-script. [Accessed: 02- 05- 2018]

[11] IBM Inc., “OpenWhisk”. [Online]. Available: https://github.com/openwhisk/openwhisk. [Accessed: 02- 05- 2018]

AUTHOR



Víctor J. Expósito Jiménez studied telecommunications engineering at the University of Seville (Spain) and Graz University of Technology (Austria). Since 2012, he works at JOANNEUM RESEARCH in which his main topics have been related to Internet of Things and Cloud Computing. During these years he was one of the people in charge of the development of a 'Web of Things' application platform, designing the architecture and core components. Moreover, he has worked in several projects in different topics such as RFID, wireless sensor networks, and robotics.



Herwig Zeiner is key researcher for Industrial Internet. He was project manager and coordinator for JOANNEUM RESEARCH of several projects. His research interests are time-aware data-analytics in industrial applications, distributed service-oriented and event driven applications, data driven applications by using stream analytics, as well as intelligent feature extraction techniques and different data mining-methods for event streams. He is also an evaluator of the European Commission.