International Journal of Current Advanced Research
ISSN: O: 2319-6475, ISSN: P: 2319-6505, Impact Factor: 6.614
Available Online at www.journalijcar.org
Volume 7; Issue 5(F); May 2018; Page No. 12609-12612
DOI: http://dx.doi.org/10.24327/ijcar.2018.12612.2223

Research Article

SERVERLESS ARCHITECTURE THE FUTURE OF COMPUTATION

Udith Shankar M and Vanishree K*

Information Science & Engineering Department, RV College of Engineering Bengaluru, India

A R T I C L E I N F O

Article History:

Received 6th February, 2018
Received in revised form 20th March, 2018 Accepted 8th April, 2018
Published online 28th May, 2018

Key words:

Serverless, microservices, AWS lambda, Azure, google functions, PaaS, FaaS.

A B S T R A C T

Serverless is next evolution in cloud computing. It helps in development of application without the worry of managing the server, hardware and software. The developer has to deal with the business logic. They help run business logic as microservices. It provides high availability virtually at any scale. By embracing serverless designs, clients can reconsider their cutting edge products from ideation to creation, without sitting tight for, or stressing over, framework. The benefits are noteworthy, producing efficiencies, bringing down expenses, and accelerating time to market. In this paper we try to analyze how Serverless architecture is an upgrade to PaaS (Platform as a Service), their economic impact on application development and compare services available in the market such as AWS lambda, Azure and google functions.

INTRODUCTION

Platform as a Service (PaaS) is a class of cloud computing service that gives an opportunity enabling clients to create, run, and oversee applications without the multifaceted nature of building and keeping up the framework ordinarily connected with creating and propelling an application[6].

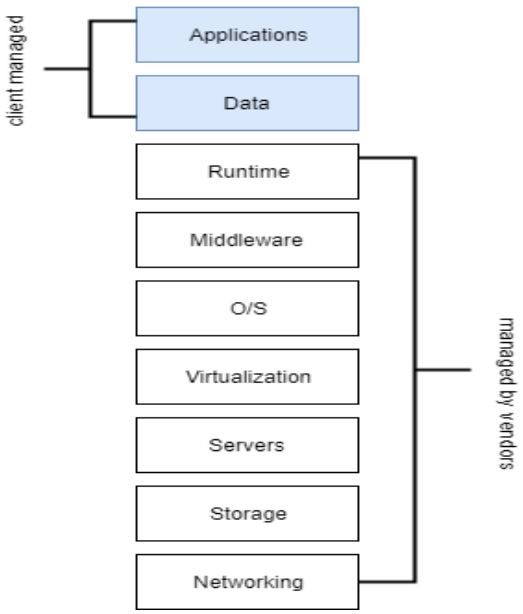

Figure 1 PaaS architecture

Corresponding author:* **Vanishree K
Information Science & Engineering Department, RV College of Engineering Bengaluru, India

Platform as-a-Service streamlines the deployment procedure of applications. It enables you to deploy your application and the cloud stresses over how to deploy the servers to run it. Most PaaS hosting options can even auto-scale the quantity of servers to deal with workloads and spare you cash amid times of low use. When deployed as PaaS, an application is typically running on at least on the server at all times. In PaaS there is an application thread always running and handles many requests. Serverless computing or FaaS totally overcomes the shortcomings of PaaS model [11] [12]. FaaS gives the capacity to deploy a single function or part of an application and is intended to possibly be a serverless design. It may not keep running at all until the point that the function should be executed. It starts the function inside a couple of milliseconds and the close it down. The Serverless term is often misunderstood to be there are no servers in background. It's not true. There are indeed servers behind the scenes but there are no dedicated servers running all the time. Whenever an event is triggered the function runs on servers and you pay only for that computational time [13].

'Serverless' alludes to another age of stage platform-as-a-service offerings where the framework supplier assumes liability for receiving customer requests and reacting to them, capacity planning, and task scheduling and operational monitoring. Developers need to worry just over the business logic. This is a huge change from the application facilitating platform as-a-service age of suppliers. As opposed to ceaselessly running servers, we deploy 'functions' that work as event handlers, and pay for CPU time when these capacities are executing [3].

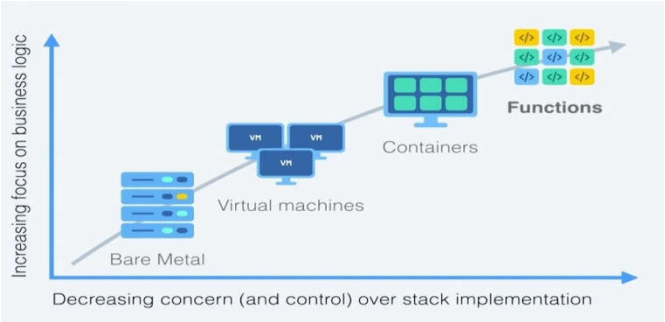

Figure 2 Evolution of Serverless

LITERATURE SURVEY

Adam Eivey says in "Be Wary of the Economics of "Serverless" Cloud Computing" that serverless has the potential to be a great abstraction offering economic advantages for simple workflows, but beware of throwing everything into it too hastily. It means that cost is often calculated on per hit basis. If your APIs are having broader execution time or accessed frequently then buying a on demand instance is quite cheap when compared to serverless. So choosing serverless should be solely based on the nature of the app.[1]

Gojko Adzic and Robert Chately in their paper "Serverless Computing: Economic and Architectural Impact" says that serverless platforms today are useful for important tasks, where high-throughput is key, rather than very low latency, and where individual requests can be completed in a relatively short time window. The economics of hosting such tasks in a serverless environment make it a compelling way to reduce hosting costs significantly, and to speed up time to market for delivery of new features. They take up case study of startups such as Mindmup and Yubl and show how adopting serverless has reduced their cost upto 66% to 95%. They shed light on opportunities created by the serverless platform such as removing incentives for bundling and removing barriers to versioning. They also discuss about the potential disadvantages such as vendor lock in.[2]

Geoffrey C. Fox, Vatche Ishakian, Vinod Muthusamy, Aleksander Slominski in their whitepaper "First International Workshop on Serverless Computing (WoSC) 2017" explains current state of serverless. Serverless is the next generation of computing supporting centralized and edge computing with a common event-driven programming model. Serverless will build the long dreamed infinite limitless computing fabric. The future of serverless has also been discussed in this paper. Serverless will be applied to general purpose computing and it will grow in capability and limitations such as the "5 minute kill limit" will disappear. It will be great for end developers as they will not need to know scaling and distributed computing. They discuss the need for serverless development community [3].

Erwin van Eyk, Simon Sief and Markus Thommes in their paper "The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures" addresses the community problem suffered by the Serverless and FaaS platforms. They proposed four perspectives that aim to capture the direction of the serverless field, and in particular of FaaS architectures. They are A Cloud-Native Programming Model, Separation between Business and Operational Logic, Hybrid Clouds and Focus on Cost/Performance. They have defined serverless as an abstract model of cloud software development, where, with FaaS, cloud functions are executed as services, with operations delegated to infrastructure operating transparently to cloud developers and their customers. We have introduced the concept of simple and composite, workflow-like cloud functions, and proposed as layered model for FaaS-based software architectures [4].

Serverless Architecture

Functions are the unit of scale in serverless that theoretically abstract the language runtime. We don't discuss the amount CPU or RAM or some other asset we require for a function to run. We speak pretty much the time it takes to run the function. Every single other metric ought not to trouble us. We compose our functions, deploy them to the cloud, and pay just for time these functions ran [4].

A serverless solution consists of a web server, FaaS layer, security token service (STS), user authentication and database.

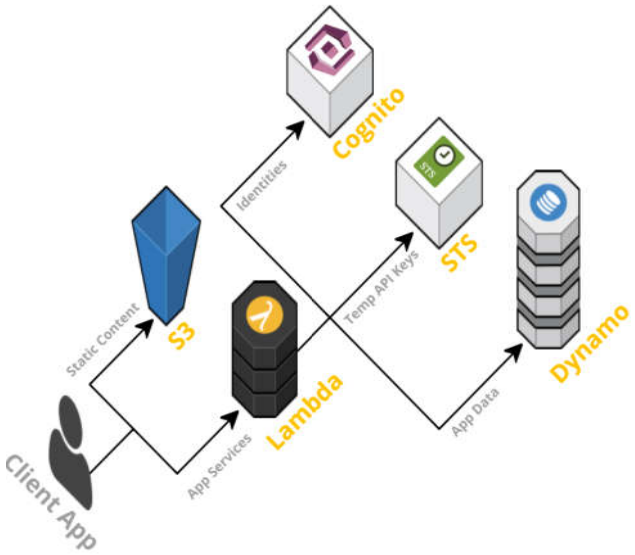

Figure 3 General Serverless Architecture

Client Application – The UI of your application is best-rendered customer side in Javascript which enables you to utilize a straightforward, static web server. Web Server – Amazon S3 provides a robust and simple web server. All of the static HTML, CSS and JS files for your application can be served from S3. FaaS solution – It is the key empowering agent in serverless design[10]. Some well-known providers of FaaS are AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions. AWS Lambda is utilized as a part of this system. The application administrations for signing in and getting to information will be worked as Lambda capacities. These capacities will read and compose from your database and give JSON responses. Security Token Service (STS) will generate temporary AWS credentials (API key and secret key) for users of the application. These temporary credentials are used by the client application to invoke the AWS API (and thus invoke Lambda). User Authentication – AWS Cognito is an identity service which is integrated with AWS Lambda. With Amazon Cognito, you can easily add user sign-up and sign-in to your mobile and web apps. It also has the options to authenticate users through social identity providers such as Facebook, Twitter, or Amazon, with SAML Identity solutions, or by using your own identity system. Database – AWS Dynamo DB provides a fully managed

NoSQL database. Dynamo DB is not essential for a serverless application but is used as an example here [8].

Serverless design doesn't entirely indicate what our functions actually should be. It is only some unit of work that we need to be finished. Functions can be activated numerous ways. It can be a clock that runs a capacity intermittently, however it can likewise be HTTP ask for or some event in some related service. On a very basic level, FaaS is tied in with running back-end code without dealing with your own particular server frameworks or your own particular server applications. Since we have no server applications to run, deploying is altogether different from conventional frameworks – we transfer the code to the FaaS supplier, and it does everything else. Functions in FaaS are activated by events specified characterized by the supplier. Most suppliers additionally enable functions to be activated as a reaction to inbound HTTP asks for, normally in some sort of API Gateway.

Cost of Serverless

Amazon Web Services was the primary cloud supplier to present serverless with "Lambda functions" in 2014[13], however comparative offerings from Google, Microsoft, IBM, and others have been presented from that point forward. Numerous small players are additionally arranging offerings, anxious to compete in the market. There are numerous likenesses and a few contrasts between estimating models and genuine costs, and cloud evaluating is famously subject to revisions, but we investigate one that is extremely illustrative—Amazon Web Services Lambda.

Serverless lambda costs us in two ways [14]. One is visible costs and second is hidden costs. Visible costs consist of requests and CPU & RAM usages. The hidden costs comprises of API requests and networking costs. Some unknown costs such as code maintenance also adds up. An AWS Lambda work with 512 MB of memory costs $0.030024 contrasted with an On-Demand server with the same details costing $0.0059. So when your CPU is completely used constantly, running on Serverless may not be taken a toll productive for your workload. Serverless may prove costly if data and networking are prime contenders for the cost. API Gateway has a tendency to be a tremendous piece of your Serverless costs when you associate with a ton of APIs. The more API asks for you make per trigger, the less savings you'll see from switching to Serverless [1].

	IBM OpenWhisk	AWS Lambda	Azure Functions	GCP Functions
Requests	N/A	$0.2 per 1M requests	$0.2 per 1M requests	$0.4 Per 1M requests
GB/s(compute time)	$0.000017 per GB-s	$0.000016 per GB-s	$0.000016 per GB-s	$0.00000231/GB-s
Data Transfer (IN)	Free	$0.1-$0.2 between VPCs/regions	Free	Free
Data Transfer (OUT)	$0.05- $0.09 per GB	$0.05- $0.09 per GB	$0.05- $0.09 per GB	$0.12 per GB
API Gateway	Free	$3.50 per 1M calls	Free	$3.00 per 1M calls

Table 1 Serverless cost comparision

It is important to know how much time our functions takes to execute. Currently Lambda supports function runtime of 300 seconds. So larger function have to split up to accommodate this constraint. Also results of function might be different on each platform based in congestion levels, noisy neighbors and different generation of hardware. While balancing this difference we may increase the execution time which adds up to the serverless bill.

Use Case

For the use case we take a look on Heavywater Inc [15]. The organization is centered around utilizing artificial intelligence virtual assistants to empower business process outsourcing, and the framework is constructed totally on Amazon Web Services (AWS). A major component of the product involved processing batch files, and their orchestration infrastructure was built using SWF and EC2 instances. Their approach had some drawback such as their batch processing jobs controlled by SWF were being executed and monitored 24x7and relied on the same EC2 instances used by their microservices. They were using t2.micro EC2 instances but still within 4 months their bill increased from $10K to $30K with over 1,000 EC2 instances running Indeed, even with all the spend, throughput was as yet an issue-with a normal handling rate of just 4000 documents at regular intervals. To exacerbate the situation, SWF would come up short various circumstances due to a nondeterministic issue. They had lot of brain storming sessions with AWS support and finally AWS suggested them to use step functions. AWS Step Functions makes it easy to coordinate the components of distributed applications and microservices using visual workflows. As a first step they converted all of their microservices into lambda functions and they also converted their workflows into step functions.

Instead of using the decider in SWF to transitions of workflow transitions, they built their own decider using Step Functions. Whenever a workflow was completed, it would invoke the starter Lambda of Step Function with parameters of which workflow needs to be triggered next.

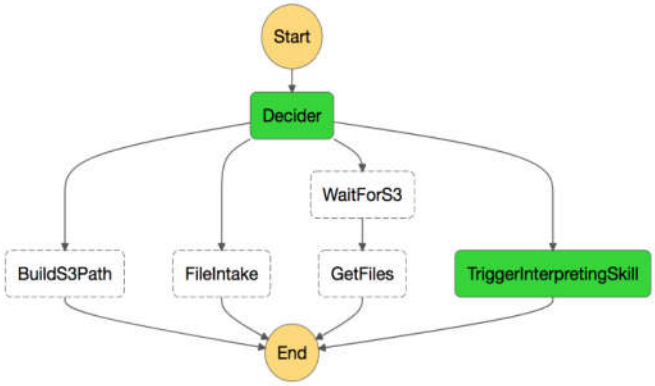

Figure 4 Sample Step function

The benefits were evident. Costs reduced to considerable amount. Number of EC2 instances reduced to 211. Human work reduced to 16 hours from usual 24 hours. By adopting this method they got a bill of around $4k only which is $26k less than previous approach. Switching into Serverless has been a major turning point in the success of the company which directly helped them to sustain in the market.

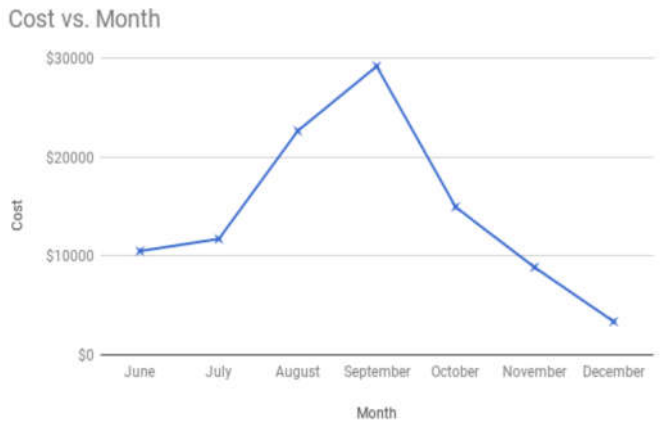

Figure 5 Cost chart

From figure 5 we can clearly see how Heavywater saved their expenses by adopting serverless computation.

CONCLUSION

In conclusion, serverless platform today are valuable for imperative assignments, where high-through put is vital, instead of low inactivity, and where singular requests can be finished in a moderately brief time window. The financial aspects of facilitating such errands in a serverless domain make it a convincing method to decrease facilitating costs fundamentally, and to accelerate time to market the new future. It gives low operational and development cost and also reduces complexity of software because of use of microservices [9]. But as every good thing has, even serverless suffers from drawbacks. It is not efficient for long running application. Vendor lock in is another serious drawback [2]. Your application is totally reliant on an outside supplier. You don't have a full control of your application. Undoubtedly, you can't change your stage or supplier without rolling out critical improvements to your application. Additionally, you are reliant on platform availabilty, and the platforms's API and expenses can change [5]. In spite of the complete documentation and community resources, you may soon discover that the expectation to learn and adapt for FaaS instruments is quite steep. Likewise, to effortlessly move to serverless, you should need to split your monolith into microservices, another difficult assignment to handle. That is the reason it's desirable to get assistance from experts experienced in serverless tools. Serverless is suitable for application like IoT, Virtual assistants and chatbots, Image-rich applications and Agile and Continuous Integration pipelines etc.

References

1. Adam Eivy. 2017. Be Wary of the Economics of "Serverless" Cloud Computing. IEEE Cloud Computing, published by the IEEE computer society.
2. Gjko Adzic, Robert Chatley. 2017. Serverless Computing: Econmic and Architectural Impact. ESEC/FSE'17, September 4-8, 2017, Paderborn, Germany
3. Goffrey C. Fox, Vatche Ishakian, Vinod Muthusamy, Aleksander Slominski. 2017. First International Workshop on Serveless Computing. WoSC.
4. ErwinvanEyk, Alexandru Iosup, Simon Sief, Markus Thommes. 2017. The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures WoSC'17, 2017, Las Vegas, NV, USA.
5. Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, Philippe Suter. 2017. Serverless Computing: Current Trends and Open Problems. arXiv:1706.03178v1 [cs.DC] 10 Jun 2017.
6. Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Mate iZaharia.2010. A View of Cloud Computing. Commun. ACM 53, 4(April 2010),50–58.
7. Nick Gottlieb. 2016. State of the Serverless Community Survey Results.
8. Alexandru Iosup and Dick H.J. Epema. 2011. Grid Computing Workloads. IEEE Internet Computing.
9. Sameer Limaye and Asif Khan. 2017. Serverless Computing: A Compelling Opportunity for Today's Digital Enterprise. TCS Whitepaper.
10. Chard R. 2017. FaaS: The future of computing [Internet]. First International Workshop on Serverless Computing. (WOSC).
11. Barga RS. 207. Serverless Computing: Redefining the Cloud [Internet]. First International Workshop on Serverless Computing (WoSC).
12. McGrath G. 2017. Provider-Side Serverless Opportunities [Internet]. First International Workshop on Serverless Computing (WoSC).
13. "AWS Lambda," February 2017; https://aws.amazon .com/lambda/
14. "AWS Lambda Pricing," February 2017; https:// aws.amazon.com/lambda/pricing.
15. "ServerLess Framework" February 2017; https:// serverless.com.
16. GojkoAdzic.2017. The key lesson from our serverless migration. https://gojko. net/2017/02/23/serverless-migration-lesson.html.(2017). Accessed:2017-04-20.
17. Mohsiur Rahman. 2017. How serverless reduced our costs by 70%. https://read.acloud.guru/how-going-serverless-helped-us-reduce-costs-by-70-255adb87b093

How to cite this article:

Udith Shankar M and Vanishree K(2018) 'Serverless Architecture The Future of Computation', *International Journal of Current Advanced Research,* 07(5), pp. 12609-12612. DOI: http://dx.doi.org/10.24327/ijcar.2018.12612.2223

****