What is a Microservice and what is a Serverless Computing Platform?

David Pearson

1050197

2023-03-26

Both microservices and serverless computing have become victim of misconception regarding their function as a result of their names. In the same way cloud computing is not really in the cloud, serverless does not entail that it runs without servers and microservices does not necessarily mean the application it supports is indeed ‘micro’. Certainly, when considering AWS Lambda, many may consider that Lambda functions only serve to support microservices. Afterall, Lambda functions are visibly ‘microservices’ – that is, micro functions that support a single service that could be apart of a larger architecture. This report, however, will dispel this misconception regarding AWS Lambda and defend the position that it is just a serverless computing platform. It is possible then, with this definition, some could argue that Lambda being just a serverless computing platform would make it equivalent to other serverless computing services like Fargate. This report will make the distinction between the two serverless services and make it clear each serves a specific purpose and are not interchangeable as that statement may make it seem. How is it then that microservices fit into this conversation? Are microservices serverless? If they are not made up of Lambda functions, what is a microservice then? This report will also make clear distinctions surrounding microservices and by the end of this report, it should be clear that serverless computing and microservices are very much independent concepts that are becoming more and more intertwined according to trends. Finally, this report will consider these trends and analyze how the future of the industry could be shaped as a result.

Microservices have gained a lot of traction in recent years as more and more companies announce their departure from traditional monolithic architectures. Despite the gaining popularity of the concept, it seems that much confusion still revolves around what a microservice actually is in practice. Microservices are an architecture pattern where a larger application is comprised of many individual and independent services. The best way to consider this concept is to imagine an application made out of *LEGO* pieces – each piece is a service that embodies a singular purpose. Individually, these components are fairly limited, but these ‘bricks’ can be connected together using lightweight communication methods, HTTP API calls for example, forming a fully functioning application – an application that is the sum of its parts. Microservices is a more modular approach to complex applications. Historically, the monolithic architecture was commonplace where a single massive application was designed to handle everything. As these application grew however, companies began to notice the glaring scalability issues related to this architecture. One of the most famous examples of this is *Netflix*, who, is largely considered to be a pioneer of enterprise level adoption of the microservice architecture.

In August of 2008, *Netflix* faced a data corruption problem which limited customer access to their DVD shipping service for three days. This sparked interest among company executives to pivot from self-managed datacenters, vertically scaled with single points of failure, to a cloud solution. In a press release from 2016, Yury Izrailevsky, Vice President, Cloud Computing and Platform Engineering at *Netflix*, discussed the companies journey over their seven-year transition to AWS.[1] Seven full years to make the transition to cloud may seem excessive, but *Netflix* wanted to fundamentally shift their entire architecture to a ‘cloud native’ solution. Yury outlined that the company could have accelerated the transition considerably if their operations were simply ‘forklift’ from their datacenters to AWS solutions – but this would have just moved the problems and limitations that came with that design. Over the course of their seven-year transition, *Netflix* fundamentally changed their architecture from the monolithic model to hundreds of microservices. Moving to this architecture resolved single points of failure experienced with their old model. It also vastly improved their overall development process, now

[1] https://about.netflix.com/en/news/completing-the-netflix-cloud-migration

allowing independent teams to make their own decisions and the typical mutli-week cycle made way for continuous delivery. *Netflix* spent the better part of a decade transitioning to a cloud native solution and as a result, they were able to maintain reliable operations as their user base increased by three orders of magnitude in the same time frame. Yury claimed had they still been on self managed datacenters they would not have been able to rack servers fast enough to support the growth.

Netflix utilizes AWS EC2 instances to support their microservice infrastructure, which they claim has since expanded to thousands of independent services. This leads to the important distinction that microservices are not necessarily serverless. AWS EC2 instances are self managed clusters, this additional overhead makes *Netflix*'s microservice architecture not serverless. It is possible that certain services could run on serverless instances, but it is important to note that a microservices architecture does not equate to serverless in all cases.

Serverless computing is a fairly new concept but is very rapidly gaining popularity as more companies transition to a cloud based microservice architecture.[2] Serverless computing is truly the next evolution in cloud-based computing and maximizes what cloud has to offer. Serverless is a bit of a misnomer, that is, they still require servers to operate. Serverless implies the lack of direct hardware management required by the developer to create a cloud application. Serverless cloud solutions abstract away any hardware management that would otherwise be required from solutions like AWS EC2 or Google Cloud Compute Engine. Serverless computing offers the developers the ability to simply have their code be deployed, and AWS or other cloud

[2] Baldini I. et al. (2017) Serverless Computing: Current Trends and Open Problems. In: Chaudhary S., Somani G., Buyya R. (eds) Research Advances in Cloud Computing. Springer, Singapore

providers manage all the rest of the infrastructure setup. This massively reduces the overall complexity of designing cloud applications and improves the general scalability of the application. Within this form of cloud computing, you pay for resources that the application actually uses, whereas a more traditional cloud offering like EC2 pricing is dependent on the configuration. In practice, this means that in instances of large unpredictable growth for an application, a serverless compute system would automatically scale to meet the required demand with no additional configurations required. The opposite of this is also true, in cases where there is no demand, and compute resources are not required, the serverless system will have no associated costs - unlike EC2 where even if its not being used you are still footing the bill for the resources. This model for computing offers developers the ability to focus entirely on the application without concern for infrastructure, scaling, and even latency caused by region selection.

Not to say that serverless comes without any drawbacks however, despite all the potential benefits. Serverless compute models are very much vendor dependent, and it can prove very difficult to transfer a new provider. Developing the Lambda functions for assignment four also proved incredibly challenging to correctly debug. As the serverless interface abstracts away many of the tools’ developers have become reliant upon for immediate feedback on code execution, the debugging process can become rather time consuming. In the instance of this assignment, debugging the Lambda functions, particularly the docker image consisted of pushing a new image, replacing the Lambda image with the new one, triggering the function, and navigating to the CloudWatch logs to inspect any console output. The necessary extra steps in this process as a result of the serverless abstraction made debugging far more time consuming.

AWS Lambda is one such example of a serverless product designed to execute functions. Lambda is entirely triggered based and is not designed for sustained execution times. In fact, AWS limits the total execution time of a function to 15 minutes. Lambda makes it fairly easy to deploy what is essentially a callback function that can respond to changes within other AWS products or triggers from external HTTP API calls. This can make AWS Lambda a great addition to a microservices architecture where low latency, efficient responses are necessary. Importantly however, Lambda functions do not necessarily need to be contained within a larger microservices architecture. While this service can certainly facilitate some functionality within a microservices system, this is not its sole purpose. If Lambda is not designed exclusively for microservices, then it is just another serverless product offered by AWS – this does not mean Lambda is equivalent to other serverless offerings such as Fargate, however.

AWS Fargate is another serverless offering, but it is entirely distinct from Lambda. Where AWS Lambda works well for limited execution time, event-based function, Fargate is designed for persistent applications that either are not trigger reliant or need more than 15 minutes of execution time. Fargate is much more akin to AWS EC2 but is the serverless equivalent. This service allows developers to deploy containers to an environment where infrastructure maintenance is not required, and scaling is handled automatically by AWS. AWS Boasts that Fargate could be the more economical option for projects as you are only billed on resources that are actually required by the system. If a developer is running a container-based web application with low traffic, they do not need to pay for resources that are not being used. Nor would this developer need to handle scaling the servers should there be a sudden uptick in site visitors. All of this is handled automatically by the AWS serverless model that has abstracted away infrastructure management. The persistence of Fargate instances is one of the most glaring differences from Lambda. If there are no triggers, the Lambda function will sit dormant, not using any resources and not costing the developer any resource time. Fargate on the other hand will always have some level of resource allocation, but the serverless model will handle scaling of resources allocated depending on system requirements.

Lambda is a fantastic service if the developer is hoping for a low latency, efficient model for executing event driven functions. If the developer was hoping to have a web application, this would not be possible through Lambda with the function timeout of fifteen minutes. Lambda functions are not meant to be persistent instances. Of course, what was developed along side assignment four serves as an example of Lambda in action - FTP requests from S3 buckets. While this could have been developed to run on a Fargate instance, this would not have made much sense as we would be paying for a baseline level of resources even when the code was not in execution. Coca-Cola utilizes AWS Lambda to facilitate their *Freestyle* beverage machines – allowing the user to scan a QR code and make a personalized drink mix. Coca-Cola said in a press release they selected the AWS serverless architecture as it allowed them to release a prototype in one week instead of the months it would have taken on a self-managed system. Serverless also allowed them to scale the product from prototype to 10,000 machines in just 150 days.[3] Lambda specifically was selected for the extremely low latency the product offers. The API gateway allows the Lambda function to take inventory of syrup available in the machine which is returned to the webapp. Fargate on the other hand is much more tailored to more complex systems that require persistent access. Samsung describe their transition from ECS to Fargate and the benefits it has provided their *SmartThings* IoT portal. Their blog discusses that the *SmartThings* portal had historically been developed with ECS but has transitioned to Fargate

[3] https://aws.amazon.com/solutions/case-studies/coca-cola-freestyle/?pg=ln&sec=c

after seeing the potential benefits. Samsung claims that by migrating their systems to Fargate they no longer required a system admin and overall operational efficiency greatly increasing as engineering teams could focus exclusively on container development.[4]

Lambda can still play an important role in the microservices architectural pattern – it just cannot be responsible for the entire architecture. In the Coca-Cola pop machine example, their webapp uses Lambda to make the API request so the user's phone can have near instantaneous response for products in stock, this is one minor component of the larger system, but that is where Lambda is strongest. This service will not run a full web app container, but it can help facilitate some useful event handlers in a cost-effective manner. Despite not being exclusively used for microservices, Lambda can still play an important role in this architecture.

[4] https://aws.amazon.com/blogs/architecture/samsung-builds-a-secure-developer-portal-with-fargate-and-ecr/?pg=ln&sec=c