Technical Report 3: Serverless Architecture

Student Project Management Dashboard

Team: Online Learning
Hayden Baker, Keshav Seshadri, Elise Wiltshire, Minh Vu

Table of Contents

1. Serverless Architecture Overview

1.1 What is a serverless architecture?

Serverless architecture is a term used to describe two similar ways in which applications handle their underlying logic. The first is the Backend as a Service (BaaS) model, where applications entrust most or all of their server-side logic to a third-party host [1]. The other model that comes under the term serverless architecture is Functions as a Service (FaaS). FaaS takes advantage of stateless compute containers. The application is broken down by the developer into atomised functions that are then hosted by a third-party provider [2]. When the client calls one of these functions, they are run through a temporary compute container. By removing the need for a central server to process all requests, FaaS allows for great scalability of its solutions. This model also changes the way in which third-parties price their services. Since there is no need to keep a server on constantly, the provider needs to only charge for the amount of times the functions are executed lowering the costs for the client [3].

1.2 How are serverless architectures different from alternatives, such as microservice architectures and containers?

Microservice architecture involves breaking an application down into loosely coupled components that are run in a container environment. It allows for easy deployment as updates to one component only require that component to be redeployed, not the whole system. With a serverless architecture, the burden of intensively planning the system design is lifted leaving developers more time to focus on the actual application itself. Additionally, the scaling of a serverless architecture is much more straightforward, as the third-party providers take care of that for you. Serverless architectures also tend to be more cost-effective as they don't require an "always-on" server to host the application unlike a microservice system [4]. The key difference between serverless architectures and the alternatives is in the name, it's serverless. As mentioned earlier this leads to many benefits such as the ease of scaling and a more cost-effective pricing scheme.

1.3 How do serverless architectures relate to historical trends in software architecture?

Originally, a person had to request a physical server to run their application. The load on the server varied among different applications, but people often bought greater and better hardware than what was needed. This cost individuals and companies a considerable amount of capital expense which was unnecessary as applications never used the full extent of their server.

Big companies such as Google understood this problem and solved it by creating a more efficient way to "scale out" by adding low-cost hardware side-by-side which distributed the load all across it [5]. This made building and running the application easier and cost-effective as one could scale according to what was needed.

Next, virtualization became popular around the world where a software-based representation of the server was created. This saved money on IT costs and reduced the number of servers needed which cut down energy consumption and provided affordable web hosting [6].

The Cloud was introduced next with the primary focus on IaaS (Infrastructure-as-a-service). Iaas provides a pool of virtual servers that helps users avoid the cost and complexity of using and managing their own physical servers. The cloud service providers manage the servers infrastructure while the user/company can focus on their software [7]. Cloud PasS (platform-as-a-service) quickly followed which is a complete development and deployment environment in the cloud with resources that users could use to build applications of different sizes and scales [8]. This made deploying and scaling much easier because the users did not have to worry about the physical servers in use.

Finally came the serverless architecture which was an evolution of PaaS where the virtual machines were replaced with ephemeral compute power that comes into existence on request and disappears immediately after use [9]. The user still does not need to worry about the server infrastructure but now can design an application using non-server building blocks. The cloud provider handles the environment configurations, runs the user's code only when invoked and only bills for actual usage while hiding scaling complexity. Scaling is now automatic which makes it easy to deploy, flexible and extremely cost-effective. Serverless architecture was first introduced in 2012 but gained popularity in 2015 with the introduction of AWS Lambda launch [1]. By 2016, the serverless architecture became extremely popular with a lot of companies and vendors using it.

2. Analysis

2.1 What are the costs and benefits of serverless architectures?

2.1.1 Benefits

- Lower Cost: Since the serverless architecture provider is responsible for managing servers and databases, going serverless can reduce the cost of building and maintaining an application tremendously. There are lesser operational, scaling and development costs and the users and companies involved can primarily focus on the application code [10].

- Better UX: Since serverless handles the architecture, developers can now focus more on the front-end design making it more appealing to the end user [11].

- Ease of Deployment: Serverless architecture facilitates the developer to only focus on the code and can thereby release it faster and quicker. The developer does not have to worry about scalability as it happens automatically [11].

- Better Scalability: Serverless architecture automatically scales based on the applications needs and requirements. This makes the application more cost-effective and efficient for both the developers and the end-users [12].

- Greater Efficiency: Users only have to pay per request when using serverless architectures. Compared to traditional servers that have to be kept running at any instance, users only have to pay for what is used in serverless architecture [12].

- More Flexibility: Users can easily change application functionality or features, and can easily pivot in situations where restructuring of the application is required [12].

- Improved Latency: Serverless architectures have access nodes all around the world making it easy for end-users to utilize the application from different parts of the globe [12].

- Greener Computing: Data centres require physical resources to be built and maintained. Idle servers require a lot of unnecessary energy and this is one of the main reasons why a lot of the huge technology companies have big data centres around the world. Serverless architecture requires companies to buy servers based on only what's required which reduces resources needed to maintain extra servers [10].

2.1.2 Cost

Serverless architectures offer users the ability to pay only for the amount they use. A request is counted each time the application starts executing a response to an event notification or invoke call. Services like AWS Lambda and Azure function have a price per function request at $0.0000002 (20 cents for one million invocations), but this alone does not reflect the total cost a user has to spend on a serverless architecture. The execution of functions utilizes computer resources and AWS and Azure functions charge an additional amount for the allocated memory and time taken to execute the function. AWS Lambda charges $0.00001667 for every GB-second used whereas Azure functions charge $0.000016 for every GB-second [13]. Thus, taking everything into consideration, AWS Lambda will roughly charge 1.25 USD for 1 million requests with the average time of 500ms and 128MB for memory. A similar function would cost around $6 if it was to be run continuously on a server. AWS and Azure also offer a free tier with 1 million invocations and 400,000GB-seconds free each month for users to get a taste of the true capabilities of their serverless architecture [14].

Applications require more than functions in the real-world, they need features through which the functionality can be exposed to the end user such as through a web-application or an API. AWS Lambda charges $3.50 per 1 million executions if an API gateway is being used [13]. Assuming it takes 3 seconds to serve a web page with a total memory allocation of 1GB, the total cost of AWS Lambda would be $11.15 which is way cheaper than running servers continuously [14].

However, a serverless architecture is not always cheaper than IaaS because certain workloads require a high amount of resources and computation, making the model less cost-effective. Numerous cloud service providers provide a serverless cost calculator that can be used to calculate the cost according to the number of executions, execution time and memory size. This calculator can be used to compare the cost between using an IaaS and serverless architecture. It should also be noted that there are no operational costs in hiring DevOps specialists and engineers as the serverless architecture manages the application infrastructure.

2.2 How serverless architecture is changing DevOps

Serverless architecture simplified the operations aspect of DevOps and made the entire architecture easy to scale where users pay only for what they use. Cloud providers are now responsible for most of the operational tasks such as server maintenance, OS updates, fault tolerance, scalability and monitoring which once required DevOps specialists [15]. The pay-as-you-go model of serverless architecture replaced on-premise self-managed data with simpler resources managed by a third party [16]. Serverless architecture makes use of third party services to carry out normal app operations. For example, instead of developing and maintaining code for an authentication system, a company or a developer can use AWS

Cognito or Okta to do this for them. This greatly reduced the complexity of the code and now the developer can focus on other aspects of the application.

Serverless architecture lets teams to get started with development and testing tight away and smoothly scale as the load increases with minimal or no effort of operations. Serverless Architecture enables DevOps automation where companies can deploy their pipelines without hosted solutions and are only responsible for the glue code [17]. In serverless architectures, developers can send software application code to the third party provider without worrying about scaling, security patching or the underlying OS. Thus, there is no Ops work to do while releasing new features and now the company can focus on developing new features.

AWS and Azure also provide fully managed DevOps Services as Saas (Software as a service) models. This service manages the underlying infrastructure and ranges from a semi-managed service to a fully managed service. This makes it possible to implement the build, test and deployment pipeline by writing only the glue code using serveless solutions [18].

However, even though a lot of the operational work is shifted to the third party provider, companies still need to understand how the different services provided work. This could be considered as 'Ops' and hence although DevOps has been greatly reduced, it has not been mitigated. Furthermore, serverless architectures work best in a distributed system which is complex and hence require teams with the relevant expertise [16].

2.3 What kinds of projects are suited to a serverless approach?

Serverless architecture can be useful for a multitude of projects, much the way they're not suited for a multitude of other projects.

Serverless architectures are good for projects where you either don't know the scalability or you get spike times [1]. The joy of serverless is that the functions and code only get run when someone is using it. As such, it will only run as little (or as much) as needed. There is no need to keep the server on 24/7 when it's not getting used that often and there's no need to make sure the server can handle a spike of triple the usual traffic even when the software isn't clocking those numbers.
A good example that would benefit from this feature, that we as Monash Software Engineering students are unfortunately familiar with, is the Monash IT GitLab server (though maybe not the best example as why have a serverless server… Either way, it has happened on occasion with Moodle so the point still sticks, regardless of the feasibility). The GitLab server gets used a reasonable amount most of the time and students have no troubles but then you get to the weekend before a project is due and all of a sudden, everyone's committing and pushing and using the GitLab server, creating spike traffic and then the server crashes. And everyone is sad, until we get assignment deadlines and then we're still sad, but with a pushed deadline. With serverless architecture, this increase in activity isn't an issue since it has scalability inbuilt into it.

Projects that also have a focus on features or need a quick turnaround coding time also benefit from serverless architectures [1]. Since the architecture itself has inbuilt services and features, this means we, as software engineers, can code features and the like in a shorter time frame since we have less things to think about. We don’t need to worry about server limits and instead can focus on the actual code itself and give more features quicker.

2.4 What risks do serverless architecture pose to the organization

In a serverless architecture, the serverless providers are responsible for securing all the cloud components including data center, servers, network, operating systems and its configurations [19]. Although this reduces the security burden developers have to deal with but security risks are still very considerable and have serious consequences to the organization. Some of the most significant risks when going serverless are:

- Function Event-Data injection: serverless architectures provide a multitude of event sources, which increases the potential attack surface and its complexity [19].

- Broken Authentication: applications built for serverless architectures contain many distinct functions which may expose public web APIs or provide access control to other functions [20].

- Insecure Serverless Deployment Configuration: Misconfiguring critical configuration settings can lead to catastrophic data losses, sensitive data exposure to unwanted personnel [19].

- Over-Privileged Function Permissions and Roles: Serverless functions should only be given necessary privileges to perform intended logic, providing over-privileged permissions to functions can give them unwanted read/write access to other components [20].

- Inadequate Function Monitoring and Logging: Processes, tools and procedures for security event monitoring and logging are inapt in serverless architectures for the purpose of providing a full security event audit trail [20].

- Insecure 3rd Party Dependencies: serverless functions depend on 3rd party software packages, libraries and even remote web services. These dependencies could be vulnerable and can make the applications susceptible to cyber attacks [19].

3. Recommendation

3.1 Would you recommend that we consider using serverless architectures in a future FIT3170 project?

While some FIT3170 projects may lend themselves to a serverless approach, we wouldn't recommend taking a serverless approach for all projects at this time.
As mentioned above, not all projects are suited for serverless architecture, and in our time as software engineers in the future, we will probably all come across a project where we can't use a serverless approach, in which case we will have to fall back on other approaches, such as the ones we currently looked into for our FIT3170 projects. As such, having that experience up our belts will be beneficial to our learning, especially since most of the pros of serverless architecture are to do with improved functionality, compared to a different approach. This means the skills we learn in this unit will be transferable to a serverless approach, whereas it doesn't quite work the same in reverse. An example of this would be server limitations. If you generate too much traffic on your server, it will crash. You can mitigate it, and you'll have to learn to mitigate it. Serverless architecture doesn't have the limitation, since it's a per-usage basis and hence, if you started with serverless and didn't have to worry about it, it may comes as a bit of a slap-in-the-face when suddenly having to switch to a server and worry about it for the first time.

3.2 Would you recommend that we consider discussing serverless architectures in other units, such as FIT3077?

Serverless architecture undeniably brings about a lot of benefits in terms of scalability as well as efficiency to the product and can be a sufficient option for developers to take into consideration when analysing and designing the product. However, there are also very considerable risks to a serverless architecture design. These risks are more likely to happen and will cause greater security damage if the developers have insufficient knowledge about the deployment and configuration of serverless architectures. In a unit like FIT3077 which focuses on high-level architecture design, its foundations, industrial practices, discussing serverless architecture will certainly help students have a better grasp at all the different approaches that are popular in the industry. Though the students should be able to recognise the risks of a serverless approach and should be familiar with a conventional server architecture.

References

[1] M Roberts. “Serverless Architectures”. MartinFowler. https://martinfowler.com/articles/serverless.html (Accessed November 12, 2020)

[2] "What is Serverless Architecture?". Twilio.com https://www.twilio.com/docs/glossary/what-is-serverless-architecture (Accessed November 12, 2020)

[3] What is Serverless Architecture? What are its Pros and Cons? | Hacker Noon". Hackernoon.com. https://hackernoon.com/what-is-serverless-architecture-what-are-its-pros-and-cons-cc4b804022e9 (Accessed November 12, 2020)

[4] "Microservices vs. Serverless", Fathomtech.io. https://fathomtech.io/blog/microservices-vs-serverless/ (Accessed November 12, 2020)

[5] Static.googleusercontent.com. https://static.googleusercontent.com/media/research.google.com/en//archive/googlecluster-ieee.pdf (Accessed November 12, 2020)

[6] L McCabe. “What is Virtualization , and Why Should You Care”. Small Business Computing. https://www.smallbusinesscomputing.com/testdrive/article.php/3819231/What-is-Virtualization-and-Why-Should-You-Care.htm (Accessed November 12, 2020)

[7] B Noble. “What is IaaS”. MilesWeb. https://www.milesweb.com/au/hosting/cloud-hosting/what-is-iaas (Accessed November 12, 2020)

[8] “What is PaaS”. Microsoft Azure. https://azure.microsoft.com/en-au/overview/what-is-paas/ (Accessed November 12, 2020)

[9] “Serverless Architecture”. ThoughtWorks. https://www.thoughtworks.com/radar/techniques/serverless-architecture (Accessed November 12, 2020)

[10] M Roberts. “Serverless Architectures”. MartinFowler. https://martinfowler.com/articles/serverless.html#benefits (Accessed November 13, 2020)

[11] N Fee. “What is Serverless Architecture”. NewRelic. https://blog.newrelic.com/engineering/what-is-serverless-architecture/ (Accessed November 13, 2020)

[12] D Bird. “10 Amazing Benefits of Serverless Technology”. DashBird. https://dashbird.io/blog/business-benefits-of-serverless/ (Accessed November 13, 2020)

[13] “AWS Lambda Pricing”. Amazon. https://aws.amazon.com/lambda/pricing/ (Accessed November 13, 2020)

[14] R Gancarz. “The Economics of Serverless Computing”. TechBeacon. https://techbeacon.com/enterprise-it/economics-serverless-computing-real-world-test (Accessed November 13, 2020)

[15] H Dhadhuk. “Why you should go serverless for DevOps”. Stackify. https://stackify.com/why-you-should-go-serverless-for-devops/ (Accessed November 13, 2020)

[16] B Ellerby. “Is Serverless the end of Ops?”. Medium. https://medium.com/serverless-transformation/is-serverless-the-end-of-ops-devops-vs-noops-nativeops-7997889f9a9c (Accessed November 13, 2020)

[17] L Wainstein. “Serverless Computing: Taking DevOps to the next level”. DevOps. https://devops.com/serverless-computing-taking-devops-to-the-next-level/ (Accessed November 13, 2020)

[18] D McAllister. “How Serverless will change DevOps”. Scalyr. https://www.scalyr.com/blog/how-serverless-will-change-devops (Accessed November 12, 2020)

[19] I Thomas, "Top 10 Security Risks In Serverless". We45. https://we45.com/blog/top-10-security-risks-in-serverless (Accessed November 13, 2020)

[20] Puresec.io, “The Ten Most Critical Security Risks in Serverless Architectures“. Puresec. https://www.puresec.io/hubfs/SAS-Top10-2018/PureSec%20-%20SAS%20Top%2010%20-%202018.pdf (Accessed November 13, 2020)