SEMINAR PRESENTATION – SERVERLESS TRADE-OFFS

Laura Pernía Blanco - UO276264

Miguel Cuesta Martínez - UO258220

Nicolás Lozano García - UO271731

1. Introduction

Serverless, contrary to what it suggests, it's not the lack of server. The term is coined from the point of view of the developer, that no longer needs to manage server software and hardware since all the infrastructure is hosted by a third-party service. Besides, it should not be confused with Platform as a Service (PaaS). With PaaS, the application is treated as a whole, it's developed the traditional way and deployed as a single unit to the host provider. In case of needing scalability, it's applied to the entire application. On the other hand, using serverless, also known as Function as a Service (FaaS), applications are fragmented into isolated and autonomous functions. The FaaS provider hosts each of them individually, allowing to allocate more resources only to those functions that receive more calls.

When talking about serverless, sometimes Backend as a Service (BaaS) is also included, that is, hosted backend functionalities such as database services and authentication.

Some of the most popular services offering serverless architecture are AWS Lambda, Microsoft Azure Functions and Google Cloud Functions. Price to pay comes from the number of times functions are called instead of paying to have the application always running waiting for requests.

Serverless is still a relatively new concept, so it's continually evolving and being implemented by more organizations. As the beginning of 2021, functions were invoked on average 3.5 times more often per day than they were two years prior.

2. Advantages of Serverless Integration

From the technical point of view:

- Reduced operational cost: Due to not being managed individually, and instead being run with lots of similar or systems, using serverless options often ends up being less expensive than having your own private database.
- Built-in elasticity: the resources needed to manage any infrastructure can vary al lot, depending for example in the traffic that the web services are experimenting. Using FaaS scales the amount of resources you need dynamically, according to the current requirements of your system. This is good for systems that have inconsistent amounts of traffic.
- Reduced development costs: Using BaaS provides a lot of functionality for applications that is already implemented and can easily be imported to your current project. An

example of this are Google's authentication systems, which can be used in other applications to register accounts.

- Easier management: Backend as a service also reduces the management cost of your application, as it provides you with a good amount of components that you don't need to manage. This can also be seen in the deployment stage of the system, because a system that is managed with FaaS can be deployed without concerns about the state of your servers or your local systems.
- Optimization: Using BaaS and FaaS can also reduce the operational cost of the applications, as they are already tested and optimized.
- Event driven model: Serverless is a perfect fit for event driven architectures. The use of FaaS allows for an easy integration between the different functions and components.
- Isolation: Serverless functions are also more secure, because even if one of the is compromised, the whole of the application is not.
- Language independence: FaaS can also be implemented in different programming languages, as they are all linked by the serverless system. This allows each programmer to work with their preferred language and doesn't restrict them.
- Environmental impact: This is not an advantage of serverless architecture nowadays, but companies like Amazon and Google have expressed plans of hosting their big server centers on building powered by green energy. If this becomes the standard for the big companies, using these servers would be way less polluting than having servers in each company building, as these last ones are almost always powered by fossil carburants and the likes.

From the business point of view:

- Quicker time to market: A good advantage from the point of view of the business, is the quick deployment allowed by serverless systems. Developers don't have to spend time in external problems. This makes serverless a good fit for agile methodologies, as the products are made quickly and are easier to iterate upon.
- Less time spent designing: This is similar to the reduced development costs. Using backend as a service allows the architects to forgo the design of generic stuff that is already implemented by multiple services (like account systems and the likes) and spend their resources on the specific quirks of our current problem.
- Good for small business: Small and medium business, such as start-ups, can really benefit for the reduced costs of using serverless integration, and most often, having their own systems is more of a liability than an advantage.

3. Disadvantages of Serverless Integration

- **Learning curve of mapping languages:** It is very probable most of the development team will have no previous experience with mapping languages. When compared to common programming languages, they behave very differently, and developers will

more likely than not struggle with them lacking typechecking (which difficults error checking) and import (which forces to rewrite a lot of code instead of reusing it).

- **Virtually impossible to test / debug:** When, through debugging in our technology, we locate the definite cause of a bug in step functions, there is little we can do. Debugging and logging are services barely provided by the cloud vendor, and so you either resource to other, external sources to log the behavior of your step functions, or you will be spending a lot of time in it.
- **Complexity increases:** To the developer side, dealing with the implementation of the server will no longer be a problem and so the system has gotten simpler. However, depending on multiple networks, APIs and needing a better coordination in your services might end up not being compensated. As operation intelligence is paramount here, small projects are discouraged from using serverless because of this.
- **Cold Start Problem:** The encapsulation of services in container can prove problematic. When a service hasn't been used for a while, the container will have to be initialized before and therefore performance will momentarily drop. While this is no problem if services are "kept warm" by being required, it's an additional concern to consider.
- **Service limitations:** Serverless philosophy dictates that functions will have a single responsibility, and because of this they are design to run for small amounts of time with a low usage of memory. If you require a long running process with a lot of memory allocation, you might have to consider another architectural pattern.
- **API Hell:** The gateway of the cloud server contains so much logic that it difficults organization, often serving unrealistic purposes such as data processing (which of course the developer himself will want to take full care of).

4. Webography

https://www.twilio.com/docs/glossary/what-is-serverless-architecture

https://medium.com/ssense-tech/the-trade-offs-with-serverless-functions-71ea860d446d

https://www.datadoghq.com/state-of-serverless/

https://martinfowler.com/articles/serverless.html#drawbacks

https://pablo-iorio.medium.com/serverless-architectures-i-iii-design-and-technical-trade-offs-8ca5d637f98e

https://pablo-iorio.medium.com/serverless-architectures-ii-iii-business-benefits-eadd073c8e4d