Serverless: What it Is, What to Do and What Not to Do

Jussi Nupponen
Gofore
Tampere, Finland
jussi.nupponen@gofore.com

Davide Taibi
Tampere University
Tampere, Finland
davide.taibi@tuni.fi

***Abstract*—Serverless, the new buzzword, has been gaining a lot of attention from the developers and industry. Cloud vendors such as AWS and Microsoft have hyped the architecture almost everywhere, from practitioners' conferences to local events, to blog posts. In this work, we introduce serverless functions (also known as Function-as-a-Service or FaaS), together with on bad practices experienced by practitioners, members of the Tampere Serverless Meetup group.**

***Index Terms*—Serverless, Function-as-a-service, FaaS**

I. Introduction

Serverless computing provides a platform to efficiently develop and deploy applications to the market without having to manage any underlying infrastructure [1]. Different serverless computing platforms such as AWS Lambda, Microsoft Azure Functions, and Google Functions have been proposed by the main cloud providers. Such platforms facilitate and enable developers to focus more on business logic, without the overhead of scaling and provisioning the infrastructure as the program technically runs on external servers with the support of cloud service providers [2].

The most prominent implementation of serverless computing is Function-as-a-Service (FaaS) (also named "serverless functions"). When using FaaS, developers only need to deploy the source code of short-running functions and define triggers for executing them. The FaaS provider then, on-demand, executes and bills functions as isolated instances and scales their execution. In the remainder of this work, we refer to FaaS with the term "Serverless Functions".

The contribution of this work, targeted to practitioners and researchers, are the following:

- Introduction to Serverless and Function-As-a-Service (Section II)
- Six bad practices that we experienced while implementing serverless-based applications, together with the solutions we adopted to overcome them (Section III).
- Open issues, and research directions (Section IV)

II. What is Serverless

In serverless, the cloud provider dynamically allocates and provisions servers. The code is executed in almost-stateless containers that are event-triggered, and ephemeral (may last for one invocation). Serverless covers a wide range of technologies, that can be grouped into two categories: Backend-as-a-Service (BaaS) and Functions-as-a-Service (FaaS).

Backend-as-a-Service enables to replace server-side components with off-the-shelf services. BaaS enables developers to outsource all the aspects behind a scene of an application so that developers can choose to write and maintain all application logic in the frontend. Examples are remote authentication systems, database management, cloud storage, and hosting.

An example of BaaS can be Google Firebase, a fully managed database that can be directly used from an application. In this case, Firebase (the BaaS services) manage data components on our behalf.

Function-as-a-Service is an environment within which is possible to run software. Serverless applications are event-driven cloud-based systems where application development relies solely on a combination of third-party services, client-side logic, and cloud-hosted remote procedure calls [2].

FaaS allows developers to deploy code that, upon being triggered, is executed in an isolated environment. Each function typically describe a small part of an entire application. The execution time of functions is typically limited (e.g. 15 minutes for AWS Lambda). Functions are not constantly active. Instead, the FaaS platforms listen for events that instantiate the functions. Therefore, functions must be triggered by events, such as client requests, events produced by any external systems, data streams, or others. The FaaS provider is then responsible to horizontally scale function executions in response to the number of incoming events.

Serverless applications can be developed in several contexts while, because of its limitations, it might have some issues in other contexts. As an example, long-running functions, such as machine learning training or long-running algorithms might have timeout problems, while constant workloads might result in higher costs compared to indefinitely running on-demand compute services like virtual machines or container runtimes.

III. What to do and what not to do

Because of the recent introduction of serverless, good and best practices are very limited [2]. Leitner et al. [1] identified five patterns for composing and triggering serverless functions.

The bad practices presented in this work were elicited with the same design adopted in our previous studies on

978-1-7281-4659-1/20/$31.00 ©2020 IEEE
DOI 10.1109/ICSA-C50368.2020.00016

Authorized licensed use limited to: Oulu University. Downloaded on April 17,2023 at 10:35:34 UTC from IEEE Xplore. Restrictions apply.

microservices issues and bad smells [3] [4]. In May 2019, we surveyed 21 members of the Tampere Serverless Meetup Group[1] to collect their experience on the bad practices they applied when developing serverless applications. Then, during the next Meetup in September 2019, we organized a focus group to group similar practices with the help of the same practitioners that replied to the survey.

In this Section we describe the six bad practices identified, together with the solutions that we adopted to overcome them.

Asynchronous calls: Asynchronous calls to and between Serverless Functions increase complexity of the system. Usually remote API calls follow request response model and are easier to implemented with synchronous Serverless Function calls.

- *Problems*: Increased complexity, requires alternate response channel
- *Adopted Solution*: Use synchronous calls when applicable. Use integration on message queue (Pub-Subscribe) to allow notification to caller of the success of the operation if asynchronous calls are used. However, asynchronous calls are viable solution to one-off jobs like triggering long running backup process.

Functions calling other functions

- *Problems*: Complex debugging, loose isolation of features. Extra costs if functions are called synchronously as we need to pay for two functions running at the same time.
- *Adopted Solution*: Avoid calling functions from another function. Merge the functions, when possible.

Shared code between functions

- *Problems*: Might break existing Serverless Functions that depend on the shared code that is changed. Risk to hit the image size limit (50MB in Lambda), warmup-time (the bigger the image, the longer it takes to start).
- *Adopted Solution*: Write independent and decoupled Functions. Use clean architecture and depend shared code only via well defined and tested interfaces.

Usage of too many libraries

- *Problems*: Increased space used by the libraries increase the risk to hit the image size limit and increase the warmup-time.
- *Adopted Solution*: Import only the libraries that are really needed, avoiding overloading the system. A possible workaround in AWS Lambda, if the warm-up time does not matter, is to load the library at startup from external storage in a non-persistent temporary directory.

Adoption of too many technologies such as libraries, frameworks, languages.

- *Problems:* Adds maintenance complexity and increases skill requirements for people working within the project.
- *Solutions:* Limit the number of technologies adopted in the project.

Too many functions Creation of functions without reusing the existing one. Non-active Serverless Functions doesn't cost anything so there is temptation to create new functions instead of altering existing functionality to match changed requirements.

- *Problems:* Decreased maintainability and lower system understandability.
- *Solutions* Carefully consider if there is a need to create a new function. Group functions into "microservices" so as other services will see only the microservice interface instead of the detailed implementation of the individual functions.

IV. Open Issues

Several issues are still open in serverless, mainly because of the youth of this technology:

- Lack of *understanding of the event-driven paradigm*, especially for developers used to develop with different approaches.
- Lack of *solid tools* for deploying and developing functions. Deployment tools are not yet stable and development tools and IDEs do not yet provide a matured specific support.
- *Confusion between functions and microservices*. Some participants claimed that functions should only do only one thing, for a specific business logic. However, a microservice can be composed of one or more functions, but a single function should not be confused as a microservice.
- *Testing*. Since functions are triggered by well-defined interfaces, unit tests can be easily developed. However, system-level and integration testing become much more complex. One of the reasons, also reported by [1] and [2], might be the reduced system observability.

Providers recently developed tracing tools (e.g. AWS CloudWatch). However, the debug of the code is still very far from the debugging of a monolithic system. This issue created opportunities for tool providers (e.g. Thundra[2]) that are currently working to partially fill this gap.

Practitioners proposed several patterns to compose, orchestrate, and trigger functions. However, every few months new practitioners propose new patterns that invalidate some of the previous ones, making even more complex to understand which pattern should be used.

References

[1] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of function-as-a-service software development in industrial practice," *Journal of Systems and Softw.*, vol. 149, pp. 340 – 359, 2019.

[2] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, *Serverless Computing: Current Trends and Open Problems*, 2017.

[3] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, 2017.

[4] D. Taibi and V. Lenarduzzi, "On the definition of microservice bad smells," *IEEE Software*, vol. 35, no. 3, pp. 56–62, 2018.

[1] Serverless Tampere Meetup https://www.meetup.com/Tampere-Serverless/

[2] Thundra http://thundra.io

Authorized licensed use limited to: Oulu University. Downloaded on April 17,2023 at 10:35:34 UTC from IEEE Xplore. Restrictions apply.