ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/346675531

Serverless Where are we now and where are we heading?

Article *in* IEEE Software · January 2021

DOI: 10.1109/MS.2020.3028708

CITATIONS
32

READS
1,446

3 authors, including:

Davide Taibi
Tampere University
215 PUBLICATIONS **3,828** CITATIONS

SEE PROFILE

All content following this page was uploaded by Davide Taibi on 07 December 2020.

The user has requested enhancement of the downloaded file.

Serverless Where are we now and where are we heading?

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

Josef Spillner
Zurich University of Applied Sciences, Switzerland. spillner@zhaw.ch

Konrad Wawruch
7bulls.com, Poland. kwaw@7bulls.com

Serverless is getting more and more popular, being adopted from a large number of companies. Serverless computing is a new paradigm that provides a platform to efficiently develop and deploy applications to the market without having to manage any underling infrastructure [1][2]. However, serverless introduces several new issues that companies need to tackle. In this work we introduce the serverless technology, and its evolution, we discuss its benefits and issues and we describe future challenges that researchers should address.

INTRODUCTION

In the last few years [3], developers started thinking about operating their systems instead of operating their servers, considering applications as workflows, distributed logic, and externally managed data stores. This way of working can be considered ”serverless”, not because servers aren't running, but because developers do not need to think about them anymore.

The major cloud providers (e.g. Amazon Lambda, Microsoft Azure Functions, and Google Functions) have introduced serverless computing platforms. These platforms facilitate and enable developers to focus solely on business logic. That is excluding factors such as overhead scaling, provisioning, and infrastructure. In this case the program technically runs on external servers with the support of cloud service providers [4].

In serverless technologies, the cloud provider dynamically allocates the servers. That is, the code is executed in event-triggered stateless containers which may last for more than one invocation. In addition, serverless includes different technologies, that can be classified as Backend-as-a-Service (BaaS) and Functions-as-a-Service (FaaS).

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

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

Function-as-a-Service is an environment used for running software. Serverless applications are event-driven cloud-based systems where application development relies solely on a combina-

© IEEE

tion of third-party services, client-side logic, and cloud-hosted remote procedure calls [4]. FaaS allows developers to deploy code that, upon being triggered, is executed in an isolated environment. Each function typically describes 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 listens 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 however may be be limited in other contexts. For example, long-running functions, such as machine learning training or long-running algorithms might have timeout problems. Such constant workloads might result in higher costs compared to indefinitely running on-demand compute services like virtual machines or container runtimes.

Serverless and Microservices

Serverless computing has several characteristics in common with microservices, but also several differences. In some cases, serverless is more suited for the job than microservices. In order to understand which to use, developers should understand how these two technologies compare and contrast.

Microservices originated from the evolution of the software architectures 1, to reduce the complexity of monolithic systems and service-oriented-architectures.

Microservices are small and autonomous services with a single and clearly defined purpose [5] aimed at reducing the complexity of monolithic applications [6].

Microservices enable to vertically decompose applications into a subset of business-driven independent services connected with lightweight communication protocols — usually HTTP API interfaces. Each service can be developed, deployed, and tested independently by different development teams and using different technology stacks. Microservices have a variety of different advantages. They can be developed in different programming languages, can scale independently from other services, and can be deployed on the hardware that best suits their needs.

Microservices are commonly responsive to their interface, the primary mechanisms of interaction with the logic. Serverless, instead, are supported to be responsive to events, and the APIs are mainly the mechanisms adopted for generating the events. As an example, it is possible to trigger a function from different event sources (e.g. a database, or a message bus) without using an API Gateway. The key difference between microservices and serverless is the smaller granularity of serverless functions and the adoption of the event-driven paradigm. While it is possible to develop event-driven microservices, it is highly recommended to develop event-driven serverless applications. It is possible to compose different serverless functions to create a microservice using different patterns, such as the SAGA pattern, the gatekeeper and others. More details on serverless patterns for microservices can be found in [7].

When the first serverless platforms came on the market, practitioners started to build microservices using serverless functions. As an example, it is possible to build a microservice with an API Gateway as service interface, a serverless function behind it and a switch statement to act as a router. This very simple approach enable applications to scale independently, but it is still not properly leveraging the benefits of serverless computing.

Microservices are best suited for long-running, complex applications, that have significant resource and management requirements.

Serverless functions are event-driven and therefore executed only when needed. When the execution is over, the computing instance that runs the function is decommissioned. Serverless is more suited for event processing and non resource-intensive tasks.

However, serverless scales automatically, without needing a specific infrastructure, and therefore is it more suitable when developers need automatic scaling and lower runtime costs, while microservices can be adopted when flexibility is required.

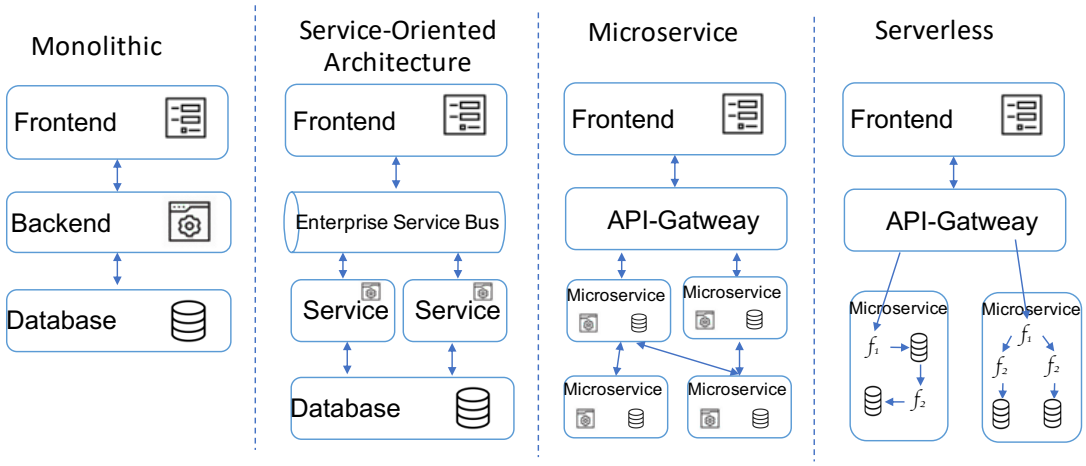

Figure 1. The Evolution of Software Architectures

Serverless benefits and issues

Serverless has several benefits compared to the traditional server-full model:

- **NoOps**
  From the development point of view, when using Serverless functions, developers simply need to deploy the code. There is no need to think of scaling, or orchestrating the system. From the Ops point of view, serverless and especially BaaS reduce the amount of resources owned by the teams, outsourcing the management of the infrastructures (e.g. databases). Therefore, developers do not need to deal with the servers, that are managed by the cloud provider.
- **Fast and independent deployment**
  Developers can deploy, and in particular redeploy a single serverless function without the need of deploying the whole system. While this is possible also for microservice, serverless allow to upload a smaller function being even more independent than with microservices.
- **No infrastructural costs**
  When using traditional cloud services, companies usually pay for generously large machines, to avoid to run out of resources. Costs are paid independently form the actual usage of the system. With serverless, there is no need to buy servers or to pay fixed fees. Code is executed only when triggered by an event, and the infrastructure is billed only for the effective time that the code is executed. In case of short-running tasks, this might reduce dramatically the infrastructural costs. However, in case of continuously running task, this might incur extra costs.
- **Auto-scaling**
  Serverless functions scale automatically, independently from the load. Serverless application can handle very high number of requests as well as a single request from a single client.

Serverless-based systems are distributed systems and introduce several complexities, including different development paradigms (e.g. the event-driven paradigm), the problem of communication between services, but also the complexity of monitoring, testing and security.

- **Testing and debugging**
  The replication of serverless environments is not always feasible and developers need to test serverless functions in production. Debugging is even more complex, because of the distributed nature of serverless projects and because the back-end processes are hidden to the developers [2][8].
- **Long-running processes**
  Long running processes might cost more on serverless compared to a traditional server-full infrastructure.
- **Performance**
  Serverless functions are not always active, and need to be activated. As a result, such ”cold start” requires some time and might introduce delays in the execution of the applications. Some workaround have been proposed to

"keep the functions warm" and avoid the cold start. However, some cloud providers already provided solutions to reduce the cold start (e.g. cloudflare.com)

- **Vendor lock-in**
  Adopting a serverless-based solution with one vendor makes very difficult to switch to another vendor, if needed. In particular, the adoption of vendor-specific databases, message buses API-Gateways or other native technologies, increases even more the vendor-lock-in. Some solutions for cross-provider deployment have been proposed. As an example, the Serverless Application Framework[1] allows to deploy the same code on different serverless providers. However, once the code is deployed and the proprietary technologies are adopted, it is very hard to move the data to another provider if needed.
- Adoption of the **even-driven paradigm** can be complex, especially if developers are used to different approaches.

Serverless evolution

Technological perspective

The perceived change of mindset and the democratisation of building large-scale applications [9] has been accompanied by evolving serverless technologies. Initially, the notion of serverless application execution was tightly bound to Function-as-a-Service offerings by public cloud providers, primarily the hyperscalers with AWS Lambda, Google Cloud Functions and their competitors. Since then, the technological choices have expanded. First, public FaaS interfaces have been replicated in several open source equivalents (Kubeless, Fission) and rather non-equivalent alternative designs mostly provided as prototypes by researchers. Further, the concepts of a serverless database was introduced with FaunaDB, and event-driven function processing in messaging middleware was introduced with Apache Pulsar, bringing serverless closer to big data and enterprise communications needs [10]. Just like in the wider field of cloud computing, multi-vendor support was added on the research side (Snafu) and in commercial tools (Triggermesh). With increasing container options (in IBM Cloud Functions/OpenWhisk, Google Cloud Run/Knative or AWS Fargate), serverless applications can now also include long-running components. Is it all history repeating with a number of choices too high for rational decision-making, or are there systematic ways to design and engineer serverless applications systematically with the best choice of technologies? Or are conventional computing offerings increasingly branded as serverless? The articles selected by us leave the answer open, but hint at best practices for serverless architectures whose path towards implementation affects adjusted engineering methods.

[1]The Serverless Application Framework www.serverless.com

Architectural perspective

Due to the novelty of serverless applications, there is little empirical data on how complex applications are composed on the architectural level, beyond what could be learned so far from interviews and mixed-method studies [11]. Typical architectures encompass cloud functions as glue code to automate processes based on events; cloud functions as short-lived services exposed to the Internet bound to backend services (BaaS), avoiding the need to restart servers and apply security patches to web servers; and functions deployed at various stages in data processing pipelines, for instance to handle sensor data streams. Now, you might be curious about whether this is changing over time, perhaps driven by new technologies. Understanding the evolution requires long-term observation studies on publicly available data. The AWS Serverless Application Repository is a prime source of knowledge on how cloud functions are composed with each other and with backend services, and luckily the evolution of its content have been recorded for the past two years. From that observation, we know that JavaScript and Python are by far the most popular programming languages (on AWS Lambda). We know further that the execution models differ among simple request-reply functions, iterators working over multiple inputs (e.g. bag-of-tasks functions), dispatchers and other forms. Finally, we understand that half of all applications are single-function implementations, and less than 15% bind the stateless function to a stateful backend, effectively yielding a stateful application.

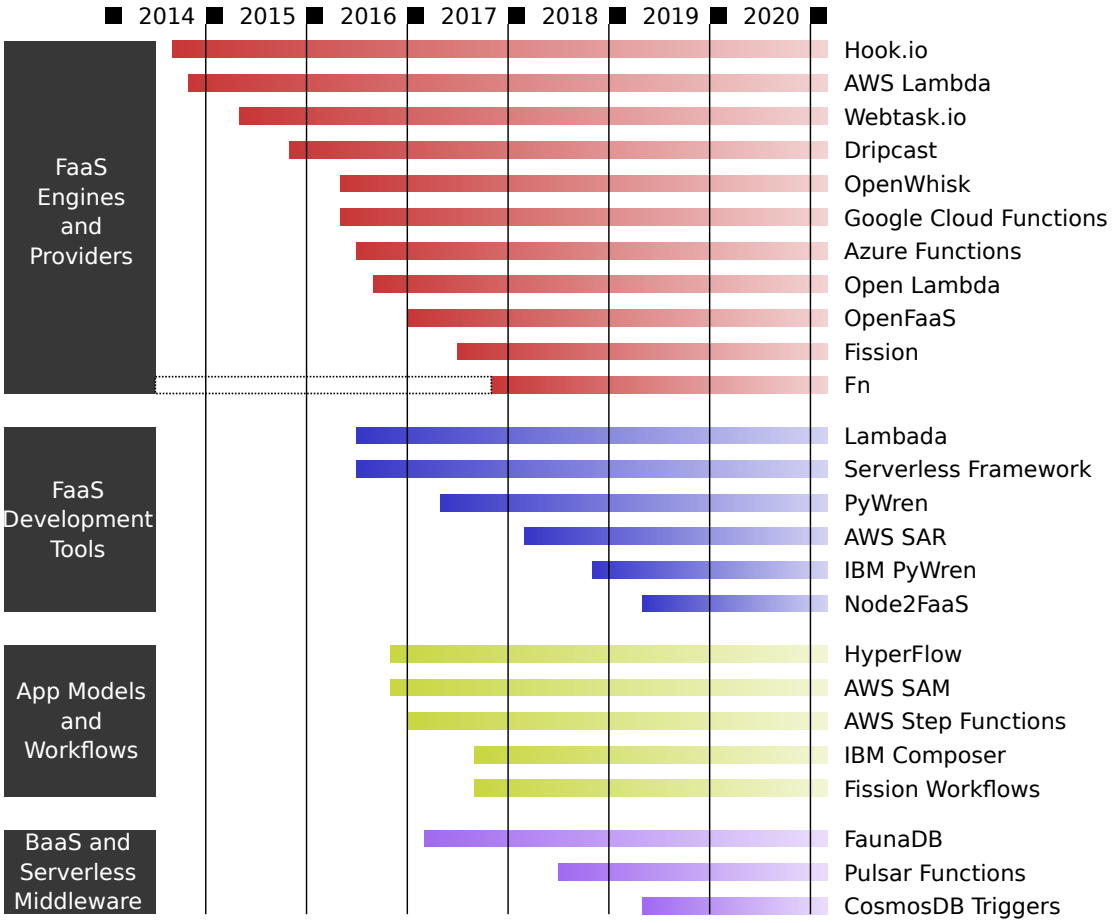

Figure 2. Growth of serverless software ecosystem over the past six years

Taking the idea of event-driven microservices further, not only can we deploy declaratively specified compositions of cloud functions and backend services in a GitOps-style workflow, but we can also describe with workflow models how they interact in causal order. It is in this space where further innovation in serverless applications engineering will happen.

Future Challenges

Trends emerge from the convergence of serverless architectures with others to further free application engineers from infrastructural concerns. One challenge is capturing rich portable deployment information so that flexible data processing across edge-cloud continuum becomes easy through optimised deployments. This means departing from the instruction-centric Serverless Framework towards more goal- or intent-centric ones where runtime behaviour knowledge is taken into account. Better tracing and profiling is another one, freeing the engineer from the need to specify runtime version or memory configuration – surely, in the age of AI it should be possible to detect that automatically and avoid effort and inconsistencies. Another challenge is the identification of optimal architectural patterns. Developers often apply workarounds to overcome the serverless limitations [2]. As an example, developers might try to find workarounds to avoid the problem of double billing when having a synchronous call waiting for the output of another call. From this point of view, the major challenge is to change the developers mindset, enabling them to reason using an event-driven paradigm. The long-term evolution is another important challenge. The lack of clear patterns, but also the vendor lock-in increases the maintenance complexity, due to the need to using only the technologies provided by one vendor and the

complexity of migration to others. Community solutions are upcoming at faasification.com.

Acknowledgment

We sincerely thank the authors and reviewers of all of the high-quality submissions we received for this theme issue. We also thank Editor in Chief Ipek Ozkaya and the IEEE Software crew for its guidance and support. Happy reading.

REFERENCES

1. W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara. Serverless computing: An investigation of factors influencing microservice performance. In *Int. Conf. on Cloud Engineering (IC2E)*, pages 159–169, 2018.
2. J. Nupponen and D. Taibi. Serverless: What it is,what to do and what not to do. In *International Conference on Software Architecture (ICSA 2020)*, 2020.
3. M. Roberts and J. Chapin. *What Is Serverless?* O'Reilly, 2017.
4. Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. *Serverless Computing: Current Trends and Open Problems*, pages 1–20. Springer Singapore, Singapore, 2017.
5. James Lewis and Martin Fowler. Microservices. www.martinfowler.com/articles/microservices.html,, March 2014.
6. D. Taibi, V. Lenarduzzi, and C. Pahl. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, 4(5):22–32, 2017.
7. C. Pahl D. Taibi, N. El Ioini and J. R. Schmid Niederkofler. Patterns for serverless functions (function-as-a-service): A multivocal literature review. In *10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, 2020.
8. V. Lenarduzzi and A. Panichella. Serverless testing: Tool vendors and experts point of view. *IEEE Software*, 2021.
9. Eric Jonas, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. Occupy the cloud: Distributed computing for the 99%. *CoRR*, abs/1702.04024, 2017.
10. Hendrik Makait. Rethinking message brokers on rdma and nvm. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 2833–2835, New York, NY, USA, 2020. Association for Computing Machinery.
11. Philipp Leitner, Erik Wittern, Josef Spillner, and Waldemar Hummer. A mixed-method empirical study of Function-as-a-Service software development in industrial practice. *Journal of Systems and Software*, 146:340–359, March 2019.

Davide Taibi is an associate professor at the Tampere University, Finland. His research activities are focused on software quality in cloud-based systems, supporting companies in keeping Technical Debt under control while migrating to cloud-native architectures. Moreover, he is interested in patterns, anti-patterns and "bad smells" that can help companies to avoid issues during the development process both in monolithic systems and in cloud-native ones. Formerly, he worked at the Free University of Bolzano, Technical University of Kaiserslautern, Germany, Fraunhofer IESE - Kaiserslautern, Germany, and Università degli Studi dell'Insubria, Italy. In 2011 he was one of the co-founders of OpenSoftEngineering s.r.l., a spin-off company of the Università degli Studi dell'Insubria. He is a member of the IEEE and the IEEE Computer Society. Contact him at davide.taibi@tuni.fi.

Josef Spillner is associate professor/senior lecturer affiliated with Zurich University of Applied Sciences since 2015 and heading the university's efforts in Distributed Application Computing Paradigms. His applied research activities encompass multi-cloud, serverless and continuum computing with many successful transfers into industry innovation projects. Since 2020, he also holds a Fellowship of the digitalisation initiative of the canton of Zurich, responsible for cloud service enablement for smart cities and regions.

Konrad Wawruch is serial entrepreneur in ICT with successful exits. Co-founder and vice president of 7bulls.com, Polish software house and government-recognized R&D center that develops innovative business solutions and technology frameworks, among others, in cloud and AI space. He is personally involved in distributed computing since Beowulf clusters and computing grids, nowadays helping to develop open source frameworks that aid to avoid vendor lock-in through multicloud approach and building complex ML-based solutions in fog/edge/cloud environments. As member of the Council of Coalition for Polish Innovations and partner of technology-focused VC funds in Europe and Asia, he helps to grow CEE innovation ecosystem. Formerly, VP of Polish Linux Users Group and researcher & supercomputing/cluster software development head at ICM University of Warsaw. He

is a member of the IEEE and the IEEE Computer Society. Contact him at kwaw@7bulls.com.

View publication stats