Serverless Computing and Cloud Function-based Applications

Josef Spillner
Zurich University of Applied Sciences
School of Engineering, Service Prototyping Lab (blog.zhaw.ch/splab/)
Winterthur, Switzerland
josef.spillner@zhaw.ch

ABSTRACT

Serverless computing is a growing industry trend with corresponding rise in interest by scholars and tinkerers. Increasingly, open source and academic system prototypes are being proposed especially in relation with cloud, edge and fog computing among other distributed computing specialisations. Due to the strict separation between elastically scalable stateless microservices bound to stateful backend services prevalent in this computing paradigm, the resulting applications are inherently distributed with favourable characteristics such as elastic scalability and disposability.

Still, software application developers are confronted with a multitude of different methods and tools to build, test and deploy their function-based applications in today's serverless ecosystems. The logical next step is therefore a methodical development approach with key enablers based on a classification of languages, tools, systems, system behaviours, patterns, pitfalls, application architectures, compositions and cloud services around the serverless application development process.

CCS CONCEPTS

• **Networks → Cloud computing**; • **Software and its engineering →** *Automatic programming*; *Software as a service orchestration system.*

KEYWORDS

serverless computing, cloud functions, artefact quality, tutorial

ACM Reference Format:
Josef Spillner. 2019. Serverless Computing and Cloud Function-based Applications. In *IEEE/ACM 12th International Conference on Utility and Cloud Computing Companion (UCC'19 Companion), December 2–5, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3368235.3370269

1 INTRODUCTION

In the past five years of work on serverless computing, around 200 research articles and around 70 research-inspired technologies have been published as evidenced by the continuously curated Serverless Literature Dataset [6]. Most of the works focus on experiments and system proposals, whereas few are concerned with optimising the development lifecycle for serverless applications in the context of given private or public platform offerings.

In practice, many applications – both purely based on cloud functions and hybrid models where functions are merely used as glue or extension code – are built in an ad-hoc way. Developers possess a modest amount of knowledge and perform little to no regular assessment of cloud function implementation and runtime characteristics. As a consequence, function implementation is tedious, application design on top of functions requires manual effort, consistency issues creep into the resulting function artefacts, and the function execution shows subtle variations in performance and scalability.

Overcoming these issues requires the production and exploitation of knowledge in smart tools. Classifications, databases and ontologies need to be authored by the research community so that advanced enablers for better serverless application engineering can be introduced to software developers. Even details such as the choice of programming language and the corresponding language version will benefit from knowledge about the implications of that choice. By integrating knowledge a developer typically possesses about his or her productivity and the resulting code maintenance effort with metrics on performance and memory ramifications, developers can make more informed decisions at each phase of the engineering lifecycle.

2 KEY ENABLERS

Four key enablers are particularly suited to improve the situation: First, FaaSification will speed up development. Second, artefact checks will protect against deploying low-quality functions. Third, a knowledge base allows for dynamic placement, migration and general adaptation decisions at runtime. Fourth, mixed-technology compositions allow for using cloud functions wherever appropriate without having to give up on existing proven application designs. While these enablers are promising, they are all in early stages and will benefit from more research. Hence, each enabler is presented along with a brief description of maturity and practical usefulness along with link for download and further information.

2.1 FaaSification and Portability

FaaSification tools analyse conventional code and allow for its semi-automated transformation into cloud functions suitable for Function-as-a-Service (FaaS) hosting, guided by annotations. Researchers have produced FaaSification tools for Python, Java and JavaScript, all with limited applicability in large-scale practical projects [2]. However, the utility of such tools can greatly increase when they become aware of the different syntax formats for function definition and API calls for function management. Thus, a classification of heterogeneous function concepts across serverless computing vendors becomes a key enabler for rapid portable function development. The most recent version of the Lambada FaaSifier for Python[1] implements initial multi-FaaS support, albeit the representation of limits such as the maximum code size or maximum parameter size remains an open challenge.

2.2 Function Implementation Artefact Checks

In a serverless software engineering context, multiple specific artefact types have emerged within few years. Among them are function code files, function archives or packages including multiple code files and dependency libraries, function container images, workflow specifications, compositions such as Serverless Application Model descriptors and complete applications such as those offered on the Serverless Application Repository. In contrast to conventional code quality checks, the determination of how well-done these artefacts are is still under discussion in the community. The presence of quality issues is however undoubtful an issue, ranging from consistency issues to underspecified events on how to invoke a cloud function [5]. There is a need for both artefact type-specific checks and frameworks to coordinate those checks and run them as needed with appropriate follow-up actions in case of quality discrepancies. The Microservice Artefact Observatory (MAO) framework[2] tackles this issue with a global observatory, gathering information about artefacts in a reliable way on a global scale while still allowing for local real-time checks for individual developers upon commits or builds.

2.3 FaaS Knowledge Base

Awareness and artificial intelligence about the hosting environment, its characteristics and capabilities is widely seen as giving a boost to dynamic and adaptive applications. In a serverless context, cloud function-based applications will benefit economically from exploiting the available free tier and from closing memory allocation and execution duration gaps. Especially when the processing granularity permits, such as in bag-of-task processing applications, reducing idle periods (going below the currently dominant 100 ms intervals) and unnecessary memory allocations (going in between the current coarse-grained allocation steps) is feasible. Developers will further benefit strategically when understanding the evolution of offerings. A first knowledge base about FaaS exists[3], but is sporadically maintained. Open issues thus include the representation of such knowledge, the maintenance and governance model, fair and standardised benchmarks, and inclusion of metrics from automated FaaS provider measurements [3].

2.4 Mixed-Technology Composition and Workflows

Composability has contributed to the proliferation of microservices in software technology. In practice, there are mostly homogeneous formats for artefact and service composition. While cloud function sequences and flows can be expressed in particular workflow languages, the expression of an interaction with a non-function service is not easily possible within the same language [4]. Furthermore, the structure of an application consisting of functions, containers and other artefacts is non-trivial to express for deployments and other activities with most existing languages, in particular when discarding vendor-specific languages. Very basic research prototypes exist to overcome this limitation[4] but require clearly more work even to be usable in purely educational scenarios. In particular, such composition tools can benefit from knowledge about volatilities, execution path probabilities, redundancy and dispersal schemes, and other properties ranging from deployment to coordinated execution.

3 APPLICATION DOMAINS

Serverless computing concepts are known to be used for a lot of diverse application domains. From chat bots to real estate listings, from translation and localisation services to energy management, from video processing to intrusion detection, the benefits of an event-driven execution of short-lived scalable code units is often acknowledged. Yet the extent of use differs between the domains. This knowledge is currently confined to architecture diagrams and publications intended to be consumed by humans. Thinking further ahead, capturing the domain-specific patterns and relevant limitations would allow for the automated generation of designs and blueprints of suitable serverless applications optimised for use in the respective domains. For instance, video processing and compilation farms both in need massive parallelisation would benefit from common structures and helper functionality to make serverless application development easier. Such a move would further be aligned with the original motivation to free the developer from infrastructural and security concerns and give more space to concentrate on the core processing logic [1]. The preliminary nature of such concepts is made evident by the absence of even initial prototypes. Therefore, this long-term aspect will have to be discussed in the cloud and post-cloud computing and software engineering communities to find a suitable conceptual bridge.

REFERENCES

[1] Stefan Brenner and Rüdiger Kapitza. 2019. Trust more, serverless. In *Proceedings of the 12th ACM International Conference on Systems and Storage, SYSTOR 2019, Haifa, Israel, June 3-5, 2019.* 33–43. https://doi.org/10.1145/3319647.3325825

[2] L. Carvalho and Aletéia Patrícia Favacho de Araújo. 2019. Framework Node2FaaS: Automatic NodeJS Application Converter for Function as a Service. In *Proceedings of the 9th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,.* INSTICC, SciTePress, 271–278. https://doi.org/10.5220/0007677902710278

[3] V. Giménez-Alventosa, Germán Moltó, and Miguel Caballer. 2019. A framework and a performance assessment for serverless MapReduce on AWS Lambda. *Future Generation Comp. Syst.* 97 (2019), 259–274. https://doi.org/10.1016/j.future.2019.02.057

[4] Pedro García López, Marc Sánchez Artigas, Gerard París, Daniel Barcelona Pons, Álvaro Ruiz Ollobarren, and David Arroyo Pinto. 2018. Comparison of Production Serverless Function Orchestration Systems. *CoRR* abs/1807.11248 (2018). arXiv:1807.11248 http://arxiv.org/abs/1807.11248

[5] Louis Racicot, Nicolas Cloutier, Julien Abt, and Fábio Petrillo. 2019. Quality Aspects of Serverless Architecture: An Exploratory Study on Maintainability. In *Proceedings of the 14th International Conference on Software Technologies, ICSOFT 2019, Prague, Czech Republic, July 26-28, 2019.* 60–70. https://doi.org/10.5220/0007842000600070

[6] Josef Spillner, Mohammed Al-Ameen, and Daiana Boruta. 2019. Serverless Literature Dataset. Zenodo dataset (4th revision) at https://doi.org/10.5281/zenodo.1175423. https://doi.org/10.5281/zenodo.1175423

[1]Lambada download: https://pypi.org/project/lambadatransformer/

[2]MAO download: https://mao-mao-research.github.io/

[3]FaaS Characteristics and Constraints Knowledge Base: https://zenodo.org/record/1236763

[4]Composeless download: https://github.com/serviceprototypinglab/composeless