The spec cloud group's research vision on faas and serverless architectures

Van Eyk, Erwin; Iosup, Alexandru; Seif, Simon; Thömmes, Markus

published in
WOSC 2017 - Proceedings of the 2nd International Workshop on Serverless Computing, Part of Middleware 2017

2017

DOI (link to publisher)
10.1145/3154847.3154848

document version
Publisher's PDF, also known as Version of record

The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures

Erwin van Eyk[1,3] and Alexandru Iosup[2,1]
[1]TU Delft and [2]VU, NL, and
[3]Platform9 Systems Inc., USA
E.vanEyk@atlarge-research.com

Simon Seif
SAP SE, Germany
simon.seif@sap.com

Markus Thömmes
IBM, Germany
markus.thoemmes@de.ibm.com

ABSTRACT

Cloud computing enables an entire ecosystem of developing, composing, and providing IT services. An emerging class of cloud-based software architectures, serverless, focuses on providing software architects the ability to execute arbitrary functions with small overhead in server management, as Function-as-a-service (FaaS). However useful, serverless and FaaS suffer from a community problem that faces every emerging technology, which has indeed also hampered cloud computing a decade ago: lack of clear terminology, and scattered vision about the field. In this work, we address this community problem. We clarify the term serverless, by reducing it to cloud functions as programming units, and a model of executing simple and complex (e.g., workflows of) functions with operations managed primarily by the cloud provider. We propose a research vision, where 4 key directions (perspectives) present 17 technical opportunities and challenges.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Applied computing → Service-oriented architectures**; **Event-driven architectures**; • **Software and its engineering →** *Extra-functional properties*;

KEYWORDS

Serverless, FaaS, Cloud computing, Software architecture, Vision.

ACM Reference format:
Erwin van Eyk[1,3] and Alexandru Iosup[2,1], Simon Seif, and Markus Thömmes. 2017. The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures. In *Proceedings of WoSC'17: Workshop on Serverless Computing, Las Vegas, NV, USA, December 11–15, 2017 (WoSC'17),* 4 pages.
https://doi.org/10.1145/3154847.3154848

1 INTRODUCTION

Cloud computing enables developers of cloud-based software to increasingly abstract away hardware and operational concerns [1]. After a decade of maturation, developers can rely on Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) when implementing their cloud-based software architectures. Following a durable trend of miniaturization and commoditization of software services, we see the emergence of an architecture (model) for cloud-based software that focuses on executing arbitrary functions without much server- and resource-management burden put on the cloud developer or customer. This architecture, which is now commonly associated with emerging terms such as *serverless* and *Function-as-a-Service (FaaS)*, addresses needs for which PaaS and SaaS leave many aspects unanswered or insufficiently addressed. Previous work has focused on making serverless and FaaS technically feasible, and there currently exist tens of platforms offering FaaS services with various degrees of capability and maturity. In contrast, in this work we address a *community problem*: the lack of clearly defined terminology and models, and of a vision regarding promising research opportunities (challenges).

We see the emerging model based on running individual *cloud functions* as a consequence of the slow but sustained evolution of computing. For many decades we have witnessed a transition from relatively large, monolithic applications, to smaller or more structured applications with smaller execution units (e.g, workflows with many small tasks). This transition is captured qualitatively by various software architectures, for example, the Service-Oriented Architecture, and quantitatively by various workload-characterization and modeling studies, for example of scientific workloads running on supercomputers and grids between 1990 and 2010 [8].

A function-based model is particularly fitting for bursty, CPU-intensive, granular workloads. Currently, the use cases for FaaS are widely varying, including data processing (ETL), stream processing, edge computing (IoT), and scientific computing [5, 9]. With the current widespread experimentation around FaaS continuing, it is likely that other use cases will become apparent in the near future.

However, the FaaS field has a *community problem* that could inhibit further adoption and limit growth. Most urgently, the field lacks a clear terminology and model, and an agenda of research challenges. Early efforts, such as the seminal analysis of serverless architectures by Roberts [13], an analysis of possible applications and a description of technical challenges by Baldini et al. [2], and ongoing work by CNCF Serverless WG [3] do not comprehensively address the community problem.

The SPEC RG Cloud Group [6] focuses on terminology, methodology, quantitative evaluation, and experimental analysis, in areas of combined industry and academic interest. An activity within this group focuses on the emerging cloud-field of serverless and FaaS,

WoSC'17, December 11–15, 2017, Las Vegas, NV, USA

ACM ISBN 978-1-4503-5434-9/17/12…$15.00
https://doi.org/10.1145/3154847.3154848

with the goal of enabling performance evaluation studies of current and future FaaS platforms. Our main contribution is threefold:

(1) We clarify the terms serverless and FaaS, and provide an abstract operational model for them (Section 2). Our description includes cloud functions, to be executed as a service (FaaS) by using operations managed as much as possible by the cloud provider, and providing means to integrate multiple functions into a complex function (a workflow).
(2) We propose perspectives of where the FaaS field is heading (Section 3). Our 4 perspectives are: FaaS emerging as a native cloud-programming model, the increasing separation of business and operational logic, the move toward vendor-agnostic technology, and the increasing focus on non-functional aspects (e.g., cost and performance).
(3) Starting from the perspectives, we identify many open challenges for FaaS platforms (Section 4). We focus on challenges for software engineering, application and system operation, and performance engineering.

2 DEFINING SERVERLESS AND FAAS

In this section, we focus on clarifying "serverless" and "FaaS" through an architectural framework, using high-level abstractions for the key concepts. Our approach opens up the field for technological growth and scientific research, for the next decade. It contrasts with the industry-led efforts, such as the CNCF Serverless WG [3], which focus on current technology and aim for standardization.

2.1 Overview of Serverless Architectures

We see serverless architectures as making possible the idea of running complex, distributed applications built from relatively simple functions, without requiring the developer (the cloud customer) to manage servers (e.g., through elastic scheduling or auto-scaling) or complex operational aspects (e.g., authentication, authorization, and accounting). Specifically, we see serverless cloud architectures as defined by three key characteristics: (1) *Granular billing*: the user of a serverless model is charged only when the application is actually executing; (ii) *(Almost) no operational logic*: operational logic, such as resource management and autoscaling, is delegated to the infrastructure, making those concerns of the infrastructure operator; and (iii) *Event-Driven*: interactions with serverless applications are designed to be short-lived, allowing the infrastructure to deploy serverless applications to respond to events, when needed.

Where does serverless fit among the NIST definition of architectures (models) for cloud computing services [11]? To answer this question, we focus on who manages the operation of the cloud-based software, and depict the spectrum of answers in Figure 1: management by both the cloud user and the cloud provider (as in PaaS), to only the cloud provider (as in SaaS). From this perspective, serverless fits in the gap left between PaaS and SaaS, and partially overlaps with them. Platform-as-a-Service (Paas) and Software-as-a-Service (SaaS), although generally not considered serverless, contain examples of serverless services, such as Auth0 (serverless authentication) and Fauna (serverless database), which in our view exhibit the main characteristics of serverless.

The gap between PaaS and SaaS is large, which explains the emergence in this space of FaaS architectures (discussed in Section 2.3) and *Backend-as-a-Service (BaaS)*. We see FaaS as tightly

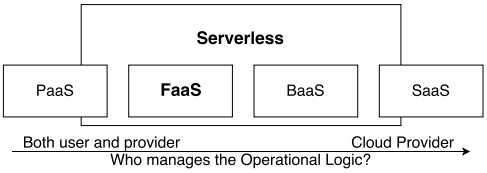

Figure 1: Serverless and FaaS vs. PaaS and SaaS.

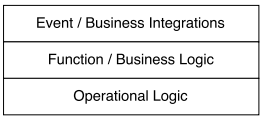

Figure 2: The FaaS model for Cloud Function deployment.

integrated with the notion of executing cloud functions, and, as such, a core component of serverless architectures. Useful especially for applications with narrow functionality, e.g., mobile applications, the (Mobile-)Backend-as-a-Service ((M)BaaS) focuses more than FaaS on operational logic, providing a more configurable, vendor-specific framework with many built-in components that simplify operation, such as user management and data-storage[13]. Thus, we see BaaS as a more tailored, vendor-specific approach to FaaS, and focus in the remainder of this work on FaaS.

2.2 Cloud Function: Simple to Workflow-Like

We define a *cloud function* as a small, stateless, on-demand service with a single functional responsibility. The function implements specific *business logic*, depending on the goal of the application. For example, for a file-sharing system, file-compression could be implemented as a cloud function; however, for an enterprise system, compressing files is part of how the system is operated.

We identify three main characteristics of cloud functions. First, cloud functions are *short-lived*: each function takes in typically small input and produces output after a typically short amount of time, which means they can be easily operated automatically (for example, they can be easily auto-scaled). Second, a cloud function is *devoid of operational logic*: any operational concerns are *delegated to the platform* (operational, cloud-managed) layer, which allows the cloud function to be platform-agnostic. Third, a cloud function is *context-agnostic*, unaware of how or why it is used.

A cloud function can quickly become complex. To overcome complexity, we envision *composed (integrated) cloud functions*, similar to classic *workflows*. The workflow structure is a concession made to facilitate operation, but preserves the integrity of the cloud function abstraction and also facilitates development.

2.3 Function-as-a-Service

We see FaaS as a cloud architecture, enabling developers with little to no experience of operational logic to create, monitor, and invoke cloud functions. Current FaaS platforms (such as AWS Lambda, Google Cloud Functions, Fission, and OpenWhisk) deploy, monitor, and manage cloud functions, tending to operational concerns such as resource auto-scaling, traffic routing, and log aggregation.

Unlike the abstract concept of cloud functions, FaaS cannot completely abstract away all operation logic from the user. The FaaS user can still change parameters and configurations, such as the suggested memory size or number of CPUs of the underlying function host, which influence the operation of the deployed cloud function. To address this situation, we propose *a layered model for*

FaaS and depict it in Figure 2, with the cloud function (business logic) represented as the middle layer.

The *Operational Logic* layer focuses on deploying, executing, and monitoring the cloud function. This could include *routing* and *throttling* incoming requests to functions, *isolating* functions using virtualization, *provisioning* (auto-scaling), and *allocating* appropriate resources to the functions. This layer is currently platform-specific; without a common standard, operational logic is not transferable from platform to platform, or from cloud provider to cloud provider.

The *Event / Business Integration* layer defines how, why, and when the business logic is executed. It integrates the business logic and couples it to events sources, triggering the cloud function on the occurrence of arbitrary events. These events can originate from timers to execute the cloud function periodically, from incoming messages from message queues, from changes to files and other system objects, or from HTTP requests to a specific endpoint. This layer is therefore context-specific, as specific events sources are directly coupled to the required cloud functions.

3 PERSPECTIVES ON SERVERLESS AND FAAS

In this section, we propose four perspectives that aim to capture the direction of the serverless field, and in particular of FaaS architectures. Because the field is growing rapidly, and many commercial and community-driven products already exist, we propose diverse perspectives that cover non-exhaustively aspects of software engineering (SE), systems (Sys), and performance engineering (PE).

(1) A Cloud-Native Programming Model: FaaS architectures could provide support for a programming model that is native to the cloud, that is, it considers services as naturally as any other programming concepts, to be developed, composed, and shared with ease. This approach has its background in the 1970s flow-based programming [12], and should offer similar operational benefits to what modern workflow management systems offer to scientific computing applications, albeit at less finer granularity than functions [10] and thus with different SE and Sys challenges.

(2) Separation between Business and Operational Logic: A characteristic of cloud functions is the delegation of operational logic to the cloud provider (see Section 2.2). As more systems are ported to FaaS architectures, the separation between operational and business logic will require a clarification of the Dev and Ops roles in DevOps teams, and a re-assessment of the principles of system design. Additionally, because serverless leads to the near-absence of CAPEX, companies will focus on managing the OPEX costs of running applications, raising interesting PE challenges.

(3) Hybrid Clouds: A vendor-agnostic FaaS layer allows a function to be deployed to different clouds (a hybrid-cloud deployment). For example, a composition of functions could reside partially in an on-premise cluster executing proprietary code, whereas the other parts of the application are hosted on public clouds, or even on sensors at the edge of the cloud. Hybrid-cloud deployment still has to overcome many challenges, across SE, Sys, and PE.

(4) Focus on Cost/Performance: The stateless, short-running cloud functions enable fine-grained control over performance. A deployed function can be migrated from one cloud to another seamlessly, if the new cloud offers a better fit between requirements and achieved performance. An example of this is the recent rise of spot markets [15], in which the price for different resource types varies over time and across resource-providers, triggering financial incentives to migrate functions across providers. We forecast that for individual functions absolute performance will start to matter less, whereas the overall performance/cost ratio will start to matter more. This raises interesting challenges in PE.

4 CHALLENGES FOR SERVERLESS AND FAAS

We identify in this section key SE, Sys, and PE challenges (opportunities) in realizing the perspectives we propose in this work.

4.1 Software Engineering Challenges

One of the most pressing issues with the serverless paradigm is the developer experience [13]. To overcome the SE challenges posed by the FaaS model, we envision advances in testing, tooling, functionality, and training and education.

(1) Simplicity vs. Completeness: The perceived simplicity of the FaaS model partly explains its popularity, especially in contrast to the perceived complexity of distributed systems. However, future FaaS platforms will need to support many more use-cases than today. What is the trade-of between simplicity and expressiveness?

(2) Interoperability: Cloud functions are devoid of operational logic, theoretically allowing the functions to easily be moved to other FaaS platforms. However, currently each cloud vendor requires a different definition of a cloud function. There is a need for a vendor-agnostic definition of both the basic cloud-function and of composite functions, to allow functions to be cloud-agnostic.

(3) Complex Workflows: Allowing developers to compose functions increases application-modularity and encourages the reuse of functions. Research will need to focus on what composition models would fit FaaS, on ways to express these compositions of functions (e.g., AWS Step Functions, Apache OpenWhisk function composition, Fission Workflows), and on how to support (frequent) function-updates and hybrid-cloud deployment.

(4) Testing: By delegating operational logic to the infrastructure, testing can be focused on business-critical logic. There are several challenges in this area, including how testing FaaS deployments differs from traditional deployments, how to orchestrate integration tests in a near-production environment, and how to replicate locally for testing the third-party, often closed-source cloud platforms.

(5) Education: A key consequence of large-scale adoption of FaaS is that software engineers do not need vast knowledge of distributed systems, because the operational concerns, such as autoscaling and resource management, are handled by the infrastructure. New questions arise: what knowledge of distributed systems and networking does a FaaS user still need to know?, what other notions apply to FaaS specifically that need to be taught?, and how to efficiently teach students and train engineers new to FaaS?

(6) Migrating legacy systems: As noted by Spillner et al. [14], legacy Java-based systems have a notion of functions. This makes migration to cloud functions intuitive. Further research should investigate how to optimize the migration process, and to what extend it is possible to automate the extraction of functions from legacy systems to cloud functions.

(7) Local development environment: Because cloud-functions will reside in different environments and interact with third-party services, the local development processes will also change. What is an effective development process for applications of cloud-functions? What tools and IDE features are needed?

4.2 System (Operational) Challenges

The operational side of FaaS includes both cloud users deploying, monitoring and maintaining their functions, and cloud providers managing these functions and the underlying FaaS platform. The highly dynamic nature of cloud functions calls for advances in security, cost predictability, and cloud function lifecycle management.

(8) Security: Centralizing operational logic in the infrastructure reduces the attack surface, compared to services maintaining their own servers and resources. However, research needs to understand the new security issues introduced by FaaS. For example, because the infrastructure can share resources among cloud-functions, what is the ideal security vs. performance/cost trade-off?

(9) Division of concerns: Our layered model for FaaS introduces an explicit division between business logic for the function, and operational logic delegated to the infrastructure. However, it is unlikely that the developer will avoid operational concerns completely. Thus: what operational concerns still matter to the developer?, and which concerns matter only to the infrastructure?

(10) Lifecycle management: FaaS platforms are responsible for most of the lifecycle of cloud functions. We envision that new versions of functions will constantly be deployed, facilitated by the small granularity of these functions. Lifecycle management concerns, such as versioning at unprecedented scale, remain open.

(11) Management of Non-Functional Requirements (NFRs), such as performance and availability: We envision that, motivated by OPEX cost and by open-competition, FaaS platforms will allow developers and even users to (dynamically) fine-tune the behavior of deployed cloud-functions. Research in this area should focus on specifying (dynamic) NFRs, developing schedulers with support NFRs, and scheduling policies focusing on NFRs within the FaaS context.

(12) Reference architecture: A reference architecture would provide developers and researchers with an understanding of the main components shared by FaaS platforms, facilitating new deployments and enabling comparisons of designs. Although desirable, for the moment this remains an open challenge.

4.3 Performance Engineering Challenges

Particularly of interest to the SPEC RG Cloud Group are the challenges related to PE, engineering and evaluation of FaaS platforms:

(13) Performance isolation: Current FaaS platforms make use of containerization combined with various performance-optimization techniques. However, these practices reduce the performance isolation between the functions and the associated invocations. Ways to reduce the performance isolation, and understanding the performance-isolation trade-off, are open challenges.

(14) Optimizing overhead: Reducing the overhead of FaaS platforms would help further user adoption, as currently FaaS platforms can lead to long-tail response times [7]. Investigating ways to reduce the overhead, for example by eliminating wait-time due to cold starts, is a research opportunity.

(15) Engineering for cost-performance: The granular billing (pay-per-use) characteristic of serverless, where costs are directly related to business-critical functionality, means there is a direct incentive to optimize *simultaneously* performance and OPEX costs [4].

(16) Fair comparison of FaaS platforms: A comparison between FaaS platforms on performance, availability, and other common cloud characteristics, would allow users to select the FaaS platform that fits their needs and platform developers to further improve their products. To objectively measure the performance of FaaS platforms, we need: representative workloads, relevant metrics, and fair, tractable, and meaningful performance evaluation methodologies.

(17) Scheduling policies: In the context of resource management, FaaS offers opportunities for scheduling policies focusing on individual function-invocations. This raises new challenges for granular, function-oriented scheduling. To make data-driven decisions in-time, a challenge of scheduling scalability arises.

5 CONCLUSION AND ONGOING WORK

The emerging field of serverless and Function-as-a-Service (FaaS) suffers from a community challenge: defining the main terms and formulating a long-term vision. Addressing this challenge, in this work we have defined serverless as an abstract model of cloud software development. With FaaS, cloud functions are executed as services, with operations delegated to infrastructure operating transparently to cloud developers and their customers. We further envision 4 perspectives of how serverless and FaaS could evolve, and propose 17 technical challenges to be overcome by the next generation of FaaS platforms.

This paper is the first step toward the broader goal of the SPEC RG Cloud Group [6], to develop a general performance evaluation methodology, and an objective benchmark of FaaS platforms. Our on-going plans are to help build up a community of interested researchers and practitioners.

ACKNOWLEDGMENTS

This work is partially supported by the Dutch STW/NWO personal grant Vidi MagnaData (#14826) and national program COMMIT.

REFERENCES

[1] Michael Armbrust et al. 2010. A view of cloud computing. *Commun. ACM* 53, 4 (2010), 50–58.
[2] Ioana Baldini et al. 2017. Serverless Computing: Current Trends and Open Problems. *CoRR* abs/1706.03178 (2017).
[3] CNCF. 2017. Serverless WG. https://github.com/cncf/wg-serverless. (2017).
[4] Adam Eivy. 2017. Be Wary of the Economics of "Serverless" Cloud Computing. *IEEE Cloud Computing* 4, 2 (2017), 6–12.
[5] Nick Gottlieb. 2016. State of the Serverless Community Survey Results. https://serverless.com/blog/state-of-serverless-community/. (Nov 2016).
[6] Cloud Group. 2017. SPEC Research Group's Cloud chapter. https://research.spec.org/working-groups/rg-cloud.html. (2017).
[7] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. 2016. Serverless computation with openlambda. *Elastic* 60 (2016), 80.
[8] Alexandru Iosup and Dick H. J. Epema. 2011. Grid Computing Workloads. *IEEE Internet Computing* 15, 2 (2011), 19–26.
[9] Eric Jonas, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the Cloud: Distributed Computing for the 99%. In *SoCC*. 1:1. (In print, available as CoRR article http://arxiv.org/abs/1702.04024).
[10] Gideon Juve, Ann L. Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. 2013. Characterizing and profiling scientific workflows. *Future Generation Comp. Syst.* 29, 3 (2013), 682–692.
[11] Peter M. Mell and Timothy Grance. 2011. *SP 800-145. The NIST Definition of Cloud Computing.* Technical Report. Gaithersburg, MD, United States.
[12] J Paul Morrison. 2010. *Flow-Based Programming: A new approach to application development.* CreateSpace.
[13] Mike Roberts. 2016. Serverless Architectures. https://martinfowler.com/articles/serverless.html. (2016).
[14] Josef Spillner and Serhii Dorodko. 2017. Java Code Analysis and Transformation into AWS Lambda Functions. *arXiv preprint arXiv:1702.05510* (2017).
[15] Qi Zhang, Quanyan Zhu, and Raouf Boutaba. 2011. Dynamic resource allocation for spot markets in cloud computing environments. In *UCC*. 178–185.