When to use FaaS? - Influencing technical factors for and against using serverless functions

Robin Lichtenthäler, Stefan Winzinger, Johannes Manner, and Guido Wirtz

Distributed Systems Group, University of Bamberg
robin.lichtenthaeler@uni-bamberg.de
stefan.winzinger@uni-bamberg.de
johannes.manner@uni-bamberg.de
guido.wirtz@uni-bamberg.de

Abstract. Cloud computing offerings evolve continuously. A recent trend is the Function as a Service paradigm which confronts developers with the decision whether adopting this new paradigm can be beneficial for parts of their application. However, many factors influence this decision or even prevent the usage of FaaS. Therefore, this paper provides a structured overview of relevant technical factors to guide the decision process.

Keywords: FaaS, serverless, serverless function, cloud computing, decision

1 Introduction

Building applications which run in a cloud environment is the norm for many enterprises today [9]. However, the possibilities how applications are designed and run in the cloud are manifold and evolve with new offerings of cloud providers. A recent trend is serverless computing with the Function as a Service (FaaS) offering at its core [4]. Developers implement a *serverless function* by writing code, potentially combined with further code artifacts (e.g., libraries), which processes an input and produces an output according to a predefined interface. The serverless function can then be deployed to a *FaaS platform* which enables its execution. The FaaS platform stores the function code and registers it to be executable via one or more triggers. The available triggers depend on the specific platform [17]. Typical triggers are messages from a messaging system, database events or an HTTP request to a URL associated with the function by the FaaS platform. When a serverless function is invoked via such a trigger, the platform manages the actual execution transparently. That means, the platform starts up the execution environment with the required resources, typically a container, and executes the function with the given input [4]. For multiple invocations, running containers can be reused or the platform can scale accordingly by starting or stopping additional containers. Whether a platform can reuse a container (warm start) or has to start up a new container (cold start) can have a significant impact on the execution time [19].

J. Manner, S. Haarmann, S. Kolb, O. Kopp (Eds.): 12th ZEUS Workshop, ZEUS 2020, Potsdam, Germany, 20-21 February 2020, published at http://ceur-ws.org/Vol-2575

FaaS platforms are typically administered by cloud providers as in the case of AWS Lambda[1] or Azure Functions[2]. These cloud providers also offer direct integration with their respective cloud services, like database services or messaging services to serve as triggers or dependencies for functions. But also open-source platforms like OpenFaaS[3] or Knative[4] are available.

FaaS enables a fine-granular billing in a commercial setting where customers only pay when a serverless function is actually invoked and executed [2, 10]. Since the FaaS platform starts and stops function containers transparently in order to scale the system dynamically and to enable this billing model, serverless functions should be stateless. This statelessness is one technical aspect in the decision for or against using serverless functions. Although such aspects are already known, they were not summarized yet and are often discussed in a different context [4, 10, 15]. Therefore, the aim of this paper is to provide a structured and comprehensive overview of criteria relevant for the decision process. The focus is on criteria which result from the technical characteristics of FaaS platforms. Since it can make sense to use serverless functions only for parts of an application, the starting point for the criteria we consider is a so-called functionality.

A *functionality* represents a specific part of business logic which can be implemented and potentially deployed as a serverless function. An *application* is a composition of multiple functionalities. An application primarily consisting of serverless functions is called a *serverless application*.

In the following, section 2 provides a short overview of our approach for the identification of the criteria which are presented in a catalog as the main contribution of this paper in section 3. We discuss characteristics of the criteria and how they can be practically applied in section 4. Section 5 gives an overview of related work whereas we conclude the paper in section 6.

2 Approach

The initial identification of relevant technical criteria was not based on a formal approach but our prior experience with studying FaaS as a recent trend in cloud computing. In order to refine and validate the identified criteria, we then decided to build upon the work of Spillner et al. [23] who maintain a comprehensive list of research on serverless computing. By relying on their collection of relevant literature, we looked for insights that either substantiated or rebutted our identified criteria. The result is the criteria catalog in section 3. The basic assumption is that each criterion can be evaluated on the basis of a functionality which is considered to be implemented as a serverless function.

[1] https://aws.amazon.com/lambda
[2] https://azure.microsoft.com/en-gb/services/functions/
[3] https://www.openfaas.com/
[4] https://knative.dev/

3 Criteria catalog

Statelessness: A functionality should be stateless since the FaaS platform cannot guarantee a reexecution on the same instance [15, 21]. This means that a functionality should not hold state (e.g., data stored in memory or on the disk of the function instance) of previous invocations which is required for further invocations. Thus, a serverless function is an example of the *stateless component* pattern [12] and all state required should be externalized.

Idempotency: A functionality should be idempotent in the sense of the *idempotent processor* pattern [12] meaning that it can be reexecuted multiple times with the same input and produce the same output. This mostly applies to how a serverless function handles externalized state. Since FaaS platforms typically reexecute a function in case of an error, a functionality should be idempotent or at least provide the possibility to be made idempotent [15]. Otherwise, a reexecution can lead to unexpected behavior, for example data inconsistencies because of repeated database transactions.

Synchronous dependencies: Synchronous dependencies express the frequency a functionality requires synchronous communication with other services during its execution. This means that a request is sent to an external system, e.g., a data storage or an external API, and the process is blocked until a response arrives. Because a serverless function is also charged for the time in which it is blocked, each synchronous dependency potentially creates unnecessary cost [3, 5]. This effect can even be increased when multiple function instances are executed on the same host and therefore also have to share the available network bandwidth for their synchronous dependencies [13]. A functionality having many synchronous dependencies should therefore not be implemented as a serverless function. Asynchronous interactions with other services where no response is expected are not problematic.

Event-driven architecture: An event-driven architecture aligns well with the FaaS paradigm because serverless functions are well-suited as event processors [5, 10, 20]. Therefore, if an event-driven architecture is used for the application the functionality belongs to, a serverless function is a suitable implementation. Additionally, several FaaS platforms provide a built-in support (in the form of specific triggers) for the consumption of events from various sources.

Algorithmic computing resource efficiency: The computing resources required for the execution of a functionality depend on its algorithmic characteristics and the input. Since the computing resources have to be defined upfront for serverless functions, a computing resource configuration has to be chosen which is able to handle the input requiring the most computing resources. Consequently, potential inputs need to be known or a worst-case assumption for the input has to be made. Memory and CPU power are typically not assigned independently but change in conjunction [18] for most FaaS platforms. Therefore, the configuration has to be chosen so that enough CPU power and memory are assigned meaning that usually one of the two is oversupplied for most inputs.
Oversupplied CPU power does not inevitably result in higher cost because less complex inputs are often processed faster [16]. If additional cores are assigned

with more CPU power, it depends on the ability of the functionalities to be processed by several cores in parallel. However, oversupplied memory does not result in a faster execution. It leads to unnecessary cost if memory is provisioned and has to be paid for, although not needed for all inputs [7, 8]. Thus, an inefficient usage of the memory resources contradicts the implementation as a serverless function from an economical point of view.

Maximum execution time: The execution time for a serverless function has an upper limit enforced by the FaaS platform [4, 17] because serverless functions should be short-lived [10, 13]. The maximum execution time for a functionality, even with the highest computing resource configuration, has to be lower than this limit. Otherwise, the execution will abort with a timeout.

Maximum memory consumption: FaaS platforms limit the memory which can be assigned to the execution of a function [4, 17]. The maximum memory consumption of a function for all inputs has to be smaller than the maximum possible memory configuration. Otherwise, the execution will abort with an out of memory error.

Availability of execution environments: Available execution environments are managed by the FaaS platform [15]. Implementing a functionality as a serverless function is therefore only possible if the desired execution environment is supported by the chosen FaaS platform. While the available execution environments of AWS Lambda are limited to common execution environments like Java, Python or NodeJS [17], OpenFaaS accepts custom Docker images enabling the usage of arbitrary environments [22].

Deployment artifact size: FaaS platforms limit the maximum size of the deployment artifact (code archive, container image) for a serverless function [17] to prevent too large functions and excessive storage usage. If the size of a serverless function implementation in the form of its deployment artifact exceeds this limit, it is not possible to implement the functionality as a serverless function.

Latency: If a functionality strictly requires a very low latency, a FaaS platform might not be able to provide such a latency because of the start-up overhead for starting a function instance. This start-up overhead is the time needed by the FaaS platform to provide the required computing resources and to start the infrastructure (e.g., the container). Since the FaaS platform manages transparently when function instances are started and stopped, the start-up overhead occurs frequently [21]. In general, this leads to a longer response time for functionalities implemented as serverless functions [1–3]. Although the overhead is also influenced by the runtime environment, it is inherent to the FaaS paradigm. Therefore, the suitability of serverless functions depends on the latency requirements for a functionality, as discussed in detail by Aditya et al. [1].

Update frequency: Running systems have to be updated frequently, e.g., to implement new features or to fix bugs. Using FaaS offers benefits regarding the speed with which functionalities can be updated. These benefits are based on two specific aspects of FaaS: externalization of operational tasks and the small size of independent deployment units. Because tasks like hardware installation, operating system maintenance and container orchestration are externalized

[10, 11], no additional efforts are required and developers can focus solely on the actual functionality [15]. All serverless functions of an application can be deployed independently. If a functionality has to be changed, only the serverless function implementing this functionality has to be redeployed which saves time in contrast to other approaches.

Vendor independence: In order to be vendor-independent, the possibility to implement and deploy a functionality in different environments is needed. FaaS platforms expect that serverless functions are implemented according to an interface predefined by the corresponding platform provider. Once a functionality is implemented as a serverless function for a specific FaaS platform, its code has to be adapted if it has to be transferred to another FaaS platform [2]. Furthermore, FaaS platforms provide platform-specific services which are often used by the functions. If these services are used, additional efforts have to be made to transfer and provide these services on another platform [15, 24]

Workload type: In general, different workload types can be distinguished which can be used to classify specific usage profiles of functionalities. According to Fehling [12], there are static, periodic, once-in-a-lifetime, unpredictable and continuously changing workloads. If the workload is unpredictable, i.e., bursty, or continuously changing, FaaS is well-suited since its provision of resources can be adapted dynamically [4]. Under- or over-provisioning does not occur compared to other deployment options like VMs. For a once-in-a-lifetime workload (e.g., for a migration) FaaS makes sense if the resources required cannot be provided otherwise, but it is not the typical use case for FaaS. Handling a static workload can usually be implemented more cost-efficiently in a more traditional cloud model since the resources needed are known upfront and are used most of the time which is cost-efficient [15]. If workload is produced periodically, FaaS can make sense but offers no specific advantage compared to other deployment models. Apart from this broad assessment, a thorough cost calculation for a specific usage profile is required to definitively evaluate the suitability of FaaS [7].

4 Discussion

The criteria presented in section 3 cover important aspects of FaaS in a compact way without being too detailed. Therefore, other criteria were excluded from our collection which are not exclusive to FaaS. E.g., for a cloud service like AWS Lambda, it might not be possible to determine where the serverless function is actually executed. Howevere, there can be regulations for sensitive date that they can only be processed in certain countries. Thus, the usage of FaaS for sensitive data might be prohibited. Despite being an important concern, it is not exclusive to FaaS because it also applies to other cloud deployment models like Platform as a Service (PaaS). Another often mentioned advantage of FaaS is the automated horizontal scalability of serverless functions. While it can be a convincing argument for FaaS, it is not an exclusive characteristic of FaaS. Other deployment models like a Kubernetes[5] deployment with a horizontal pod

[5] https://kubernetes.io/

autoscaler or a PaaS deployment can offer the same horizontal scalability, although lacking the possibility to scale to zero instances. Additional criteria in this regard are availability of SLAs and resilience. It has to be noted that the decision for or against using FaaS therefore also depends on the alternatives to which FaaS is compared to. Alternative deployment models could for example be an on-premises deployment on dedicated hardware, a virtual machine (VM) deployment within an Infrastructure as a Service (IaaS) offering, a PaaS deployment, or a deployment in a Kubernetes cluster. For some of those alternatives the mentioned excluded criteria can make a significant difference. But, as already stated, to keep the catalog comprehensible we decided to focus on criteria exclusive to FaaS and a more comprehensive comparison of criteria with selected deployment alternatives could be done in future work.

Furthermore, our catalog is focused on criteria resulting from the technical characteristics of FaaS. Other non-technical criteria can be relevant in a decision process as well, for example the effort to train developers to work with the new paradigm and the specific FaaS platforms. But criteria which cannot be regarded on the basis of a single functionality are out of the scope for this work because they are less helpful for the decision whether FaaS is applicable to parts of an application.

An additional aspect regarding the criteria included in our catalog is that their relevance differs depending on whether an administered FaaS platform of a cloud provider or a self-hosted open source platform is used. This should be kept in mind when applying the criteria in a decision process since operational concerns are not completely externalized when a self-hosted platform is used.

Table 1. Decision guidance for incorporating serverless functions

Category	**Criterion**	**Dimension**
Mandatory	Statelessness	Yes/No
	Executable in maximum execution time	Yes/No
	Executable with desired memory setting	Yes/No
	Availability of desired execution environment	Yes/No
	Deployment artifact size not exceeded	Yes/No
Cost-Efficiency	Synchronous dependencies	Number of dependencies
	Algorithmic computing resource efficiency	% of Utilization
	Expected usage profile	static/periodic/once-in-a-lifetime/unpredictable
Individual	Effort to achieve idempotency	high/medium/low
	Event driven architecture	Yes/No
	Update frequency	Number of deployments per month
	Vendor independence important	Yes/No

We have summarized the criteria as a structured questionnaire presented in Table 1 to enable a practical application. The criteria are adjusted to be used as questions. Possible answers are provided in column *Dimension*. The table is structured into three categories. *Mandatory* criteria are knock-out questions where a single *No* indicates that FaaS is not an option.

Cost-Efficiency criteria impact the billing and are therefore relevant from a business perspective. Since FaaS is the first *real* pay-as-you-go billing model, evaluating the potential cost is a significant factor.

The impacts of *individual* criteria depend on the specific situation. Regardless of the answers given to these questions, an implementation with FaaS is not immediately prevented. Each developer or organization has to decide individually how important these aspects are. If, for example, vendor independence is important, it generally makes FaaS less suitable. But if the disadvantages can be dealt with, it might nevertheless be possible to implement a functionality using FaaS.

This questionnaire helps assess the potential of implementing functionalities as serverless functions by substantiating the decision process. The criteria are intended to be assessed on the basis of a predefined functionality. However, finding the right scope for a functionality or decide whether several functionalities could be fused into one serverless function is another important topic which has to be addressed in research.

5 Related Work

Criteria to consider in the decision process for or against using FaaS have so far not been discussed in combination but individually as suitable for the respective purpose. To our knowledge, this is the first work collecting these criteria in a catalog. The only comparable approach is a flow chart created by Bolscher as part of his Master's thesis [6], but it considers all criteria to be yes or no criteria.

Research on decision making in the broader topic of cloud computing, however, is already a mature field of research. The main focus is often on multiple-criteria decision making to select a specific cloud service or cloud provider [14]. In comparison to this, our work has a considerably smaller scope and is focused only on a single outcome, namely whether or not to use FaaS. It could however be combined with multiple-criteria decision making to select a suitable deployment model for an application.

6 Conclusion and Outlook

This paper presents a catalog of criteria which can be practically used with the presented questionnaire for the decision of whether to implement a functionality as a serverless function or not. To evaluate their comprehensiveness and usefulness, in future work the criteria should be applied to an actual use case or be empirically validated with the help of practitioners. Furthermore, specific criteria are worth to be considered in more detail such as the computing resource efficiency or the impact of different workload types.

References

1. Aditya, P., Akkus, I.E., Beck, A., Chen, R., Hilt, V., Rimac, I., Satzke, K., Stein, M.: Will Serverless Computing Revolutionize NFV. Proceedings of the IEEE 107(4), 667–678 (2019)
2. Adzic, G., Chatley, R.: Serverless Computing: Economic and Architectural Impact. In: Proceedings of 11th Joint Meeting on Foundations of Software Engineering. ACM, Paderborn, Germany (2017)
3. Albuquerque Jr, L.F., Ferraz, F.S., Oliveira, R.F.A.P., Galdino, S.M.L.: Function-as-a-Service X Platform-as-a-Service: Towards a Comparative Study on FaaS and PaaS. In: The Twelfth International Conference on Software Engineering Advances (ICSEA). p. 217. IARIA (2017)
4. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., Suter, P.: Serverless Computing: Current Trends and Open Problems. In: Research Advances in Cloud Computing, pp. 1–20. Springer Singapore (2017)
5. Baldini, I., Cheng, P., Fink, S.J., Mitchell, N., Muthusamy, V., Rabbah, R., Suter, P., Tardieu, O.: The serverless trilemma: function composition for serverless computing. In: Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software - Onward! 2017. ACM Press (2017)
6. Bolscher, R.: Leveraging Serverless Cloud Architectures. Master's thesis, University of Twente (2019)
7. Eivy, A.: Be Wary of the Economics of "Serverless" Cloud Computing. IEEE Cloud Computing 4(2), 6–12 (2017)
8. Elgamal, T., Sandur, A., Nahrstedt, K., Agha, G.: Costless: Optimizing Cost of Serverless Computing through Function Fusion and Placement. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC). pp. 300–312. IEEE (2018)
9. Eurostat: Cloud computing services used by more than one out of four enterprises in the eu. Online (2018), `https://ec.europa.eu/eurostat/documents/2995521/9447642/9-13122018-BP-EN.pdf`
10. van Eyk, E., Iosup, A., Seif, S., Thömmes, M.: The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures. In: Proceedings of the 2nd International Workshop on Serverless Computing. pp. 1–4. WoSC '17, ACM, New York, NY, USA (2017)
11. van Eyk, E., Toader, L., Talluri, S., Versluis, L., Uta, A., Iosup, A.: Serverless is More: From PaaS to Present Cloud Computing. IEEE Internet Computing 22(5), 8–17 (2018)
12. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns. Springer Vienna (2014)
13. Hellerstein, J.M., Faleiro, J., Gonzalez, J.E., Schleier-Smith, J., Sreekanti, V., Tumanov, A., Wu, C.: Serverless Computing: One Step Forward, Two Steps Back. In: 9th Conference on Innovative Data Systems Research (CIDR) (2019)
14. Lee, S., Seo, K.K.: A hybrid multi-criteria decision-making model for a cloud service selection problem using BSC, fuzzy delphi method and fuzzy AHP. Wireless Personal Communications 86(1), 57–75 (2015)
15. Leitner, P., Wittern, E., Spillner, J., Hummer, W.: A mixed-method empirical study of Function-as-a-Service software development in industrial practice. Journal of Systems and Software 149, 340–359 (2019)

16. Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L., Pallickara, S.: Serverless Computing: An Investigation of Factors Influencing Microservice Performance. In: 2018 IEEE International Conference on Cloud Engineering (IC2E). pp. 159–169. IEEE (2018)
17. Lynn, T., Rosati, P., Lejeune, A., Emeakaroha, V.: A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms. In: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). pp. 162–169. IEEE (2017)
18. Malawski, M., Figiela, K., Gajek, A., Zima, A.: Benchmarking Heterogeneous Cloud Functions. In: Heras, D.B., Bougé, L. (eds.) Euro-Par 2017: Parallel Processing Workshops. pp. 415–426. Springer International Publishing (2018)
19. Manner, J., Endreß, M., Heckel, T., Wirtz, G.: Cold Start Influencing Factors in Function as a Service. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion, 4th Workshop on Serverless Computing (WoSC). IEEE (2018)
20. McGrath, G., Brenner, P.R.: Serverless Computing: Design, Implementation, and Performance. In: 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). pp. 405–410. IEEE (2017)
21. McGrath, G., Short, J., Ennis, S., Judson, B., Brenner, P.: Cloud Event Programming Paradigms: Applications and Analysis. In: 2016 IEEE 9th International Conference on Cloud Computing. pp. 400–406. CLOUD, IEEE (2016)
22. Mohanty, S.K., Premsankar, G., di Francesco, M.: An evaluation of open source serverless computing frameworks. In: 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). pp. 115–120. IEEE (2018)
23. Spillner, J., Al-Ameen, M., Boruta, D.: Serverless literature dataset. Online (2019), `https://zenodo.org/record/3517819`
24. Yussupov, V., Breitenbücher, U., Leymann, F., Müller, C.: Facing the unplanned migration of serverless applications. In: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing - UCC'19. pp. 273–283. ACM Press, New York, NY, USA (2019)

All links were last followed on January 10, 2020.