The Wonderless Dataset for Serverless Computing

Nafise Eskandani
Technische Universität Darmstadt
Germany
n.eskandani@cs.tu-darmstadt.de

Guido Salvaneschi
University of St.Gallen
Switzerland
guido.salvaneschi@unisg.ch

***Abstract*—Function as a Service *(FaaS)* has grown in popularity in recent years, with an increasing number of applications following the Serverless computing model. Serverless computing supports out of the box autoscaling in a pay-as-you-go manner, letting developers focus on the application logic rather than worrying about resource management. With the increasing adoption of the this model, researchers have started studying a wide variety of aspects of Serverless computing, including communication, security, performance, and cost optimization. Yet, we still know very little of how Serverless computing is used in practice.**

In this paper, we introduce *Wonderless*, a novel dataset of open-source Serverless applications. Wonderless consists of 1,877 real-world Serverless applications extracted from GitHub, and it can be used as a data source for further research in the Serverless ecosystem, such as performance evaluation and software mining. To the best of our knowledge, Wonderless is currently the most diverse and largest dataset for research on Serverless computing.

***Index Terms*—FaaS, Function as a Service, Serverless, Cloud Computing**

I. Introduction

Since the advent of the FaaS model for Serverless computing in 2014 by Amazon, all major cloud service providers, including Google, Microsoft, and IBM, have introduced equivalent services. In addition to these offers, a growing number of open-source platforms such as Apache OpenWhisk, OpenFaas, and Kubeless are being actively developed and maintained to support the FaaS programming model.

In contrast to the traditional cloud offerings where users explicitly provision or configure backend services, in FaaS, infrastructure management is left to the provider, enabling developers to focus on the application logic.

To develop a Serverless application with the FaaS model, programmers upload the code of one or more functions to the cloud and select the trigger events (e.g., a REST request, a file upload) that activate the functions. The cloud provider is then responsible for deployment and resource provisioning. This eliminates the need for over-provisioning to ensure that peak resource requirements can be met. Thus, the developers are charged only for the resources that the application actively requires.

Even though the FaaS approach simplifies programmers' tasks, it also introduces several challenges. First, developers are forced to adopt a programming model that despite superficial similarities to the well known imperative and functional models, significantly departs from them in practice. For example, similar to functional programming, in the Serverless model, programs are required to be stateless to enable autoscaling via automated function parallelization [1]. Yet, function composition, which is the cornerstone of functional programming, is often considered an antipattern in Serverless computing [2]. Consequently, developers resort to a programming model that resembles the imperative model but presents fundamental differences – including the fact that the state of a function is not preserved across several executions and distinct functions that belong to the same application may not even execute on the same machine. As a result, it is a common solution to use external shared storage systems to save intermediate data across functions executions [3].

Second, the performance and cost of Serverless applications are much harder to predict than traditional cloud applications. A number of aspects concur to this issue. For example, one of the consequences of the common programming practice of adopting an external storage system is that cross-function communication is slower and costlier than point-to-point networking. Other issues are inherently due to the characteristics of Serverless systems, including lack of information on data locality [4], delays due to containers startup time [5], complex triggering processes [6], and limited lifetime of functions [1].

Third, we lack tool support for various aspects of Serverless application development, including testing, debugging and continuous integration [7], [8]. Like a small monolithic system, a function can be unit tested locally before deployment. However, system/integration testing and debugging can be more complex when more than a single function is involved. In a Serverless application with several functions, most function dependencies are only available at runtime, making local integration testing and debugging impossible in some cases [9].

The issues above, exacerbated by the vendor lock-in that is currently characterizing vast amount of the Serverless computing market, pose the major challenges that slower the adoption of the FaaS model.

A first step to address these challenges is to achieve a better understanding of how the Serverless computing model is used in practice. Unfortunately, relatively little is known about the characteristics and the behavior of real-world Serverless applications. Existing studies focus on specific aspects, such as evolution of the Serverless vendors [10] or performance of Serverless applications across different providers [11], [12], but they do not provide a general-purpose dataset for research on Serverless computing. Other researchers either applied research methodologies, such as developer interviews and literature surveys [9], or used small datasets containing only

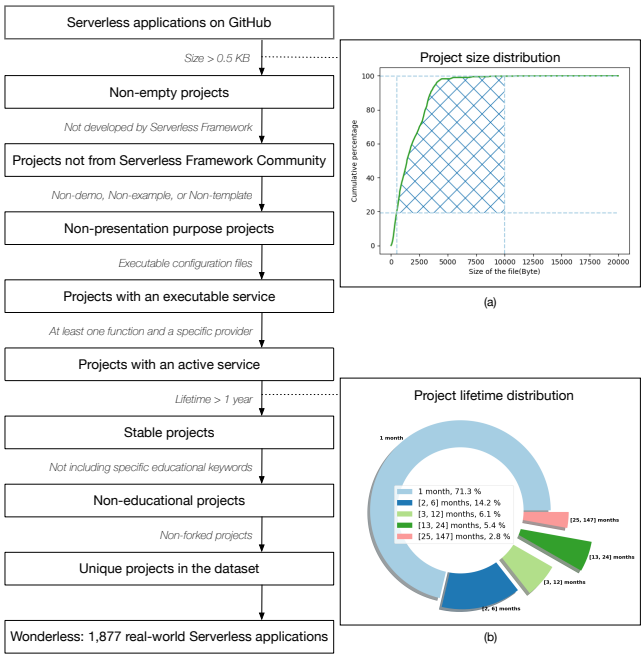

Fig. 1. Overview of creating Wonderless.

tens of applications [13].

In this work, we bridge this gap by providing a dataset of 1,877 real-world Serverless applications which is ready to use for researchers who are interested in investigating Serverless applications. Our dataset, '*Wonderless*', is publicly available[1] and the source code[2] is open for replication as well as for extension.

II. Creating Wonderless

The process of constructing Wonderless consists of two phases, described in the following (Figure 1).

A. Construction of the initial dataset

To construct the initial dataset of Serverless applications, we chose GitHub projects devloped using the *Serverless Framework*[3] – we discuss this decision in section IV. This framework uses a default YAML configuration file, named *serverless.yml*, and it allows developers to deploy applications to cloud providers like AWS, Microsoft Azure, Google Cloud Platform, Apache OpenWhisk, Cloudflare Workers, or a Kubernetes-based solution like Kubeless. Listing 1 provides an example of a *serverless.yml* file. Depending on the provider, this file can list different properties. The example shows the configuration properties for a Telegram bot including name of the provider, runtime of the application, name of the function, and type of the event that triggers the function.

We used the GitHub API to search for all the *serverless.yml* files in GitHub. We discarded files smaller than 0.5 KB which are mostly empty (Figure 1.a). As the result of our search on July 29, 2020, we collected the URLs of 41,862 unique files, corresponded to 30,078 repositories. We cloned the main branch of each repositories to create the initial version of the dataset resulting in about 400 GB of data.

```yaml
service: azure-telegram-bot

provider:
  name: azure
  runtime: nodejs12.x

plugins:
  - serverless-azure-functions

functions:
  hello:
    handler: handler.hello
    events:
      - http: true
        x-azure-settings:
          authLevel: anonymous
```

Listing 1. An example of a serverless.yml configuration file.

B. Real-world Serverless applications

The initial dataset includes a number of spurious data points such as inactive and toy projects. For this reason, we applied several rounds of filtering to the dataset.

As the first step, to remove bias in the dataset, we removed a repository if it was developed by the *Serverless Framework* community. This was done by checking if the application was developed by the *serverless* or *serverless-components* GitHub usernames. Next, to filter out the applications that served just as showcases, we removed a repository if its configuration file was in a directory with one of the labels *example*, *demo*, *template*, or *test*.

Then, we considered I) if the configuration file is executable, II) if the configuration file has the name of the provider in the *provider* property and III) if the configuration file contains the information for at least one function. We removed a repository when the configuration file does not meet all these criteria. At the end of this step, the dataset had 371 GB of data, consisting of 27,812 unique repositories.

To eliminate unstable projects, we removed the repositories with a lifetime (difference between the date of the last and of the first commit) of less than a year. Figure 1.b demonstrates the lifetime distribution of the projects in the dataset: more than 70% of the projects were active for less than a month. Overall, more than 90% of the project did not have a new commit a year after the initial one. This step dramatically reduced the size of the dataset to 2,364 repositories.

To remove educational projects, we applied a keyword search to the labels, topics, and descriptions of repositories. We extracted the description and topics of each repository using the *mercy-preview* media type provided by GitHub API. We removed a repository if the repository contained one of the following keywords: *example, demo, tutorial, playground, learn, teach, exercise, course, practice, template, sample, workshop, lecture, study*.

Finally, to avoid duplications, we removed a repository if it was forked or copied from other repositories in the dataset. To do this, we searched for the projects with the same name and different developers. Then we manually checked if those

[1]https://doi.org/10.5281/zenodo.4451387
[2]https://github.com/prg-grp/wonderless
[3]https://www.serverless.com

repositories were actually the same or they accidentally had the same name. If two repositories were the same, we removed the one with fewer commits. If the number of commits were also the same, we removed the one with fewer contributors.

III. Dataset Overview

The steps in Section II led to the construction of the Wonderless Serverless computing dataset, consisting of 1,877 real-world Serverless applications worthing 44 GB of data. The dataset is available in the two following formats.

First, a CSV file that contains the URL of each repository. With this format one can clone the latest version of the repositories at any time; However, the repositories can change access to private, be removed entirely, or drop the services related to Serverless computing over time.

To overcome these issues, we provide a snapshot of the repositories taken on November 8, 2020. In the snapshot, each repository is assigned a directory named 'a_b' where 'a' is the GitHub username of the developer and 'b' is the name of the repository. The directories are categorized into seven groups based on the provider of the application, retrieved from the $serverless.yml$ configuration file. The categories are *AWS*, *Azure*, *Google*, *OpenWhisk*, *Cloudflare*, *Kubeless*, and *Other*. The *Other* category contains providers supporting less than three applications in the dataset. If an application used multiple providers, we placed the repository in all the related groups.

Wonderless comprises 197,993 files developed in various programming languages. Other than Serverless services, an application can have non-Serverless parts developed in different programming languages. We define the runtime of a Serverless application based on the runtime specified in the configuration file of that application. We used CLOC[4] to count the number of files and lines of code for each language. Table I presents an overview, including the occurrence of the runtimes in the dataset. Node.js, with 72.2% occurrences in Wonderless, is the most popular runtime.

TABLE I
Distribution of programming languages in Wonderless.

Language	Files	Lines of Code	Occurrence%
JavaScript	44,279	7,332,365	72.2%
TypeScript	22,517	1,429,896	
Python	15,875	2,083,442	19%
Java	11,556	415,665	2.7%
Ruby	3,001	91,499	0.5%
Go	2,283	489,636	2.4%
Other	98,482	28,281,135	3.2%
Total	197,993	40,123,638	100%

Wonderless includes the Git history of each repository. According to the results from *git log*, there are 374,922 commits in total, and 13,984 developers contributed to these applications. More than 48% of the repositories have at least one GitHub star, and more than 96% have less than 100 stars.

[4]https://github.com/AlDanial/cloc

IV. Discussion

A. Identifying Serverless applications

The starting point for constructing the Wonderless is identifying Serverless applications. We discuss two alternatives to our approach.

First, one can use services like GHTorrent [14], an offline mirror of the GitHub public event timeline, to search for repositories that contain specific keywords in their descriptions, topics, commits or other related attributes. Yet, the presence of a keyword does not guarantee that an application is Serverless. Analogously, an application may be Serverless without containing a specific keyword in the repository attributes. Our approach is more precise in these regards, but it is limited to a specific framework. *Serverless Framework* is widely used with more than 36 K GitHub stars and 15 M downloads. According to a review of Serverless frameworks [15], *Serverless Framework*, with the highest number of supported providers and programming languages, and with the deployment, testing, monitoring, and security offerings is the most comprehensive existing framework.

Second, one can search for a configuration file that is specific to Serverless applications in the GitHub API. Unfortunately, to consider several vendors, one needs to find specific configuration files for each provider. Worse, the Serverless configuration files are hardly distinguishable from other cloud setups. For example, one of the default Serverless configuration files in Amazon Web Services (AWS) is $index.js$. On January 28, 2021, there were more than 202 M files with this name on GitHub, most not related to Serverless applications. In our approach, instead, we rely on a configuration file that is exclusively for Serverless applications, and has the same default name across different providers and platforms. *Serverless Framework* satisfies both these criteria.

B. Use cases for Wonderless

We envision using Wonderless in future studies concerning different directions.

First, Wonderless can be used to study several aspects that so far have not been considered or have been only marginally touched by researchers. For example, to the best of our knowledge, there is no comprehensive empirical analysis of security of Serverless applications. As a starting point, Hong at al. [16] proposed a catalogue of security patterns for Serverless computing which could be used to assess the security design of the applications in Wonderless.

Recently, Rahman et al. [17] developed a catalogue of antipatterns for Infrastructure as Code. We believe that an analogous study of antipatterns for Serverless computing would be beneficial. Similarly, Obetz et al. have proposed a novel static analysis technique that is specific to Serverless computing and they demonstrated it on seven Serverless applications [18].

Second, as Wonderless includes the full history of each Serverless application, it provides developers the opportunity to study how Serverless computing projects evolve over time [19]–[21]. It would be interesting to compare the results with analogous studies in Wonderless.

Finally, Wonderless, with 1,877 data points, can be used to extend existing studies along several dimensions. Concerning number of applications, studies that have focused on datasets in the order of tens of applications [13] can be extended to a much larger scale. Along the methodology axes, Wonderless can be used to complement the knowledge collected with surveys, and developer interviews with first-hand analysis of Serverless applications [9]. Along the technology axes, since Wonderless does not focus on a specific vendor, it can be used to discover whether existing analyses carried out for one vendor generalize to others [10].

V. Threats to Validity

First, we limited the dataset to public projects available in GitHub; Nevertheless, we believe that such projects are representative of how Serverless computing is adopted in practice. As of December 2020, GitHub with more than 64 M[5] developers and 36 M[6] public repositories, is the home for the largest open-source communities.

Second, Wonderless is restricted to the applications developed with *Serverless Framework*; However, not every developer uses a framework to program Serverless applications. They may directly use a provider, or they may adopt self-hosted solutions. As discussed in section IV-A, *Serverless Framework* is the most popular open-source solution for Serverless applications – based on GitHub stars and the results of a prior study [15]. Hence, we believe that the issue above does not significantly diminish the possibility of generalizing the results obtained from the dataset.

Another issue is that the filtering procedure may not have removed all uninteresting cases, including toy software and stub applications. To this end, we randomly selected 10% of the projects, and we manually checked them. Only 11 of the 180 projects were template or example projects.

VI. Related Work

A. Related datasets

Eismann et al. [13] analyze 32 open-source projects. These projects are a subset of an existing dataset introduced by Pavlov et al. [22]. The initial dataset was extracted from GitHub using GHTorrent based on keyword search and repositories' creation date. Eismann et al. applied several additional filters to this dataset to take out active and real-world Serverless applications resulting in only 32 projects. These filters are based on the number of files, commits, contributors, and watchers of the projects, along with the manual inspection of the repositories.

The Amazon Web Services Serverless Application Repository (AWS SAR)[7] enables developers to store and share reusable applications. This repository can also be used as a dataset to investigate Serverless applications. Yet, it is limited to AWS: it only contains applications provided by AWS and applications developed by AWS verified authors.

[5]https://github.com/search

[6]https://github.com/search?q=is:public

[7]https://aws.amazon.com/serverless/serverlessrepo/

B. Studies on Serverless computing

With the increasing popularity of Serverless applications, more and more studies have been conducted on a broad range of topics in Serverless computing.

Eismann et al. [13] provide a high-level picture of Serverless computing, including company adoption, suitable application context, and implementation of Serverless applications. In the study, the authors analyze 32 open-source Serverless projects along with 57 Serverless sources including industrial sources, academic literature, and scientific computing.

Leitner et al. [9] present the results of a mixed-method study consisting of interviews, a literature review, and a Web-based survey to identify best practices for building Serverless applications. The authors collect the data based on the programmer's experience while developing the application.

Spillner [10] investigates the evolution of Lambda functions through AWS SAR. In this study, the evolution of function-level metadata, code-level metadata, and code-level implementation of Lambda functions is investigated by continuous observation, extraction, mining and conflation of repositories in AWS SAR.

Several studies [11], [12], [23]–[27] have evaluated the performance of popular Serverless solutions by running benchmark functions across platforms. The metrics include time, memory, network, and I/O. Another class of studies investigates the cost of Serverless platforms based on different utilization metrics. Lenarduzzi and Panichella [28] study the migration of two industrial cases of early adopters and show how Lambda deployment architectures reduces hosting costs. Jackson and Clynch [29] study the impact of language runtime on the cost of Serverless functions in AWS Lambda and Azure Functions. Bortolini and Obelheiro [30] investigate the cost variations within and across Serverless platforms based on memory allocation, provider, and programming language.

VII. Summary and Conclusion

In this paper, we introduce Wonderless, a dataset of real-world Serverless applications. Wonderless is available in two formats: a CSV file containing the links to the selected repositories in GitHub and a directory containing a snapshot of the mentioned repositories grouped into seven different categories based on their providers. We believe that Wonderless can enable researchers to explore the characteristics of the Serverless computing model based on a large corpus of Serverless applications.

Acknowledgment

This work has been co-funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center, and by the German Research Foundation (DFG) within projects SA 2918/2-1 and SA 2918/3-1.

References

[1] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," *arXiv preprint arXiv:1812.03651*, 2018.

[2] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, and O. Tardieu, "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*, 2017, pp. 89–103.

[3] A. Klimovic, Y. Wang, C. Kozyrakis, P. Stuedi, J. Pfefferle, and A. Trivedi, "Understanding ephemeral storage for serverless analytics," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 789–794. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/klimovic-serverless

[4] E. Jonas, J. Schleier-Smith, V. Sreekanti, C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. J. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud programming simplified: A Berkeley view on serverless computing," *CoRR*, vol. abs/1902.03383, 2019. [Online]. Available: http://arxiv.org/abs/1902.03383

[5] J. Manner, M. Endreß, T. Heckel, and G. Wirtz, "Cold start influencing factors in function as a service," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 181–188.

[6] I. Pelle, J. Czentye, J. Dóka, and B. Sonkoly, "Towards latency sensitive cloud native applications: A performance study on AWS," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 272–280.

[7] V. Lenarduzzi and A. Panichella, "Serverless testing: Tool vendors and experts point of view," *IEEE Software*, 2020.

[8] J. Nupponen and D. Taibi, "Serverless: What it is, what to do and what not to do," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2020, pp. 49–50.

[9] 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 Software*, vol. 149, pp. 340–359, 2019.

[10] J. Spillner, "Quantitative analysis of cloud function evolution in the AWS serverless application repository," *arXiv preprint arXiv:1905.04800*, 2019.

[11] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 133–146. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/wang-liang

[12] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 159–169.

[13] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. Abad, and A. Iosup, "Serverless applications: Why, when, and how?" *IEEE Software*, 2020.

[14] G. Gousios and D. Spinellis, "GHTorrent: GitHub's data from a firehose," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 12–21.

[15] K. Kritikos and P. Skrzypek, "A review of serverless frameworks," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 161–168.

[16] S. Hong, A. Srivastava, W. Shambrook, and T. Dumitras, "Go serverless: Securing cloud via serverless design patterns," in *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*. Boston, MA: USENIX Association, Jul. 2018. [Online]. Available: https://www.usenix.org/conference/hotcloud18/presentation/hong

[17] A. Rahman, E. Farhana, and L. A. Williams, "The 'as code' activities: development anti-patterns for infrastructure as code," *Empir. Softw. Eng.*, vol. 25, no. 5, pp. 3430–3467, 2020. [Online]. Available: https://doi.org/10.1007/s10664-020-09841-8

[18] M. Obetz, S. Patterson, and A. Milanova, "Static call graph construction in AWS Lambda serverless applications," in *Proceedings of the 11th USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'19. USA: USENIX Association, 2019, p. 20.

[19] L. Sousa, D. Cedrim, A. Garcia, W. Oizumi, A. C. Bibiano, D. Oliveira, M. Kim, and A. Oliveira, "Characterizing and identifying composite refactorings: Concepts, heuristics and patterns," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 186–197.

[20] K. Du, H. Yang, Y. Zhang, H. Duan, H. Wang, S. Hao, Z. Li, and M. Yang, "Understanding promotion-as-a-service on GitHub," in *Annual Computer Security Applications Conference*, 2020, pp. 597–610.

[21] F. Wen, C. Nagy, M. Lanza, and G. Bavota, "An empirical study of quick remedy commits," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 60–71.

[22] I. Pavlov, S. Ali, and T. Mahmud, "Serverless development trends in open source: a mixed-research study," 2019.

[23] K. Figiela, A. Gajek, A. Zima, B. Obrok, and M. Malawski, "Performance evaluation of heterogeneous cloud functions," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 23, p. e4792, 2018.

[24] H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 442–450.

[25] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2017, pp. 405–410.

[26] T. Back and V. Andrikopoulos, "Using a microbenchmark to compare function as a service solutions," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2018, pp. 146–160.

[27] S. K. Mohanty, G. Premsankar, M. Di Francesco *et al.*, "An evaluation of open source serverless computing frameworks." in *CloudCom*, 2018, pp. 115–120.

[28] G. Adzic and R. Chatley, "Serverless computing: economic and architectural impact," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 884–889.

[29] D. Jackson and G. Clynch, "An investigation of the impact of language runtime on the performance and cost of serverless functions," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 154–160.

[30] D. Bortolini and R. R. Obelheiro, "Investigating performance and cost in function-as-a-service platforms," in *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. Springer, 2019, pp. 174–185.