Serverless Development Trends in Open Source: a Mixed-Research Study

Bachelor of Science Thesis in Software Engineering and Management

ILJA PAVLOV
SUSANNE ALI
TAUHID MAHMUD

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

This research provides developers and readers alike on the current trends in Serverless Software Development.

Supervisor: JOEL SCHEUNER
Examiner: Richard Berntsson Svensson

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Serverless Development Trends in Open Source: a Mixed-Research Study

Ilja Pavlov
Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
guspavloil@student.gu.se

Susanne Ali
Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
gussuzal@student.gu.se

Tauhid Mahmud
Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
gusmahmuta@student.gu.se

Abstract— **In the age of modern technology, a new paradigm, Serverless, emerges in the world of cloud computing with which it benefits developers to solely focus on the main objective instead of the maintenance of the infrastructure. This study helps developers and readers alike to have an insight into the current state of serverless software development. For the purpose of the research, an abundant amount of open-source serverless projects in Github has been analyzed with the help of Github bots, crawlers and Code Factor to gather data on common use cases, the complexity of the project and architectural patterns. Primary programming languages used to build serverless components are Javascript, Python, and C#. Furthermore, the common use cases identified in serverless projects are API, Frameworks, Communication, and data processing via Computation. The majority of analyzed projects were deemed dependent on the large vendors, primarily Amazon (72.03%) and Microsoft (21.21%). Only 3.96% of OSS projects were using open source frameworks. However, further studies are required as serverless applications will keep growing bigger in the near future.**

Keywords— serverless, github, data mining, cloud computing, repositories

I. Introduction

Since 2014, a new paradigm has been gaining traction within the cloud computing world. Serverless, or serverless computing, is an execution model where, similar to previous cloud services, the vendor provides and takes care of infrastructural needs. The core difference from other models lies in the openness of infrastructure components to deployed products and, unlike in IaaS, has minimal need to maintain abstracted VMs or containers. The terminology behind "Serverless" went through a series of debates. The study conducted by Fox. et al. [1] pointed out that serverless technology often referred to applications dedicated to third-party services, while Baldini et al. [2] articulated the focus on and concerns with cloud service providers. Following AWS Lambda's successful introduction in 2014, Functions-as-a-Service (FaaS) became one of the most popular implementations of the serverless model, with other providers like Microsoft, IBM and Google releasing equivalent services. Amazon's whitepaper on Serverless Architectures [3] defined best practices and solidified FaaS serverless as stateless compute technology for event-driven solutions. Hence, major implementations of serverless technology have taken a form of utility computing promoting the simplicity of the code deployment process and offering developers an opportunity to focus only (at least in theory) on the business logic of their application.

As reported by Google Trends, the interest in "serverless" has been growing over the last five years with quarterly currents reports conducted by DigitalOcean for Q2 2018 [4] showing growing interest in serverless - an indication of popularity and attention towards serverless services from a broad number of respondents. Yet research papers on serverless computing [2, 5] and FaaS [6] illustrate multiple challenges inherent to serverless model [7], concerns pertaining to vendor lock-in [2, 6], product migration [6], tooling issues (especially when it came to testing and debugging) [4, 6] and more.

Concurrently, surveys conducted on the current cloud computing trends indicate that in comparison to the container-based solutions, "Serverless computing is in a much earlier stage of adoption, with nearly half of developers failing to clearly understand what it is [...]" [4]. 81% of respondents unfamiliar with "serverless" indicated interest in learning more about the technology [4], while 91% of respondents who deployed applications between 2017-2018 used three major serverless platforms: AWS Lambda (58%) Google Cloud Functions (23%), Apache/IBM OpenWhisk (10%) [4].

The purpose of the study is to provide an insight into the current state of serverless software development from the perspective of existing open source communities in GitHub. The descriptive nature of the research will focus on identifying the common use cases, languages, and overall dependence of open source projects on the major platform vendors. By providing an in-depth analysis of Github repositories, the study aims to compare how trends among open serverless projects align with the previous academic studies, as well as assist those new to the serverless paradigm to understand the current trends in serverless computing.

Secondly due to the lack of academic literature that covers the trends in this research area, answering this question will be able to provide a source for future explorations into open-source, public repository-bound serverless projects, creating an academic milestone for the future works within the domain area. For example, the current design properties may be markedly different from how the future of serverless computing and the state of publicly hosted serverless projects will seem in a year or two.

To aid the realization of the proposed study, the candidates broke down the research sequence into objectives that are exemplified by three research questions:

RQ 1: What design properties and practices are prevalent among open-source serverless projects?

RQ 2: What are the common use cases that open serverless projects try to address?

RQ 3: How dependent are the open serverless applications on major platform vendors?

By evaluating the open-source serverless projects in RQ1, the data gathered will be subject to evaluation and interpretation. Analyzing data that is based on an existing, used codebase will provide a parallel observation on the current state of actual serverless applications, their common properties, and differences. This will provide an opportunity to draw a comparison between other academic studies within the domain which elicited information based on the experience of practitioners. Furthermore, through the analysis of RQ2, we intend to derive some of the common challenges that public serverless computing projects are currently trying to address. The publishers of these projects are unlikely to detail their motive for choosing serverless as their architectural and design model of choice without performing a separate study. However, it is through analysis of the use cases, application domain, language choice, target vendor, testing strategies, and other metrics that we can reverse engineer the thought process behind OSS serverless projects. The key shall be grounded in the comparison analysis between the study's findings and comparative case studies and surveys, in order to identify how aligned are the 'technical' and 'human' aspects of this paradigm. Lastly, by approaching RQ3, we intend to identify the extent to which the development of serverless projects is motivated by vendor accessibility. This will especially be approached from the issue of vendor lock-in and open-source deployment platforms in serverless computing. The end result is meant to illustrate the relationship between OSS serverless projects, their existing codebase, and how it reflects the vendor's ecology which might contribute towards this problem.

The report will discuss the originality of the project through a review of current research. The literature review will present current technologies in the research area and the relevance to the thesis project will be assessed. Current commercial products will also be discussed. Finally, the aims and objectives of the project and a project plan will be presented. The project plan will form the most significant part of the report as it will cover the resources required and the techniques to be used for the project as well as a time scale for project milestones and the expected outcomes of the project.

II. Background

As it was noted in the thesis proposal, the serverless infrastructure and services have been in a notable state of expansion and modernity.

A. Github - A key to Serverless

Serverless, in the 21st century, has become a buzzword for the tech industry and has gained unparalleled support and attention based on its appeal of greater productivity and profitability [8]. Considering the fact that serverless has brought some revolutionary changes in the tech industry, studies have unveiled that it has continually changed the previously existing server infrastructure while encouraging businesses to integrate it into their business practices. This might be the reason that leading companies like AOL [9], Reuters [10], and Telenor [11], etc. have integrated serverless computing into their business operations. However, serverless projects require specialization and expertise in reference to its development, which is why GitHub is generally preferred because of its fork and pull model, where developers are presented with an opportunity to create their own copies of repository [12]. Not just this, it creates repositories and changes into the main branch, which further creates an environment, where people can conduct their code reviews. Since GitHub is an open-source software development platform, software developers can rely on the issue tracking system on their repositories; hence providing support to the developers in terms of reporting and discussing the bugs and other related concerns [13].

Social features are worth discussing, as they are integrated into the GitHub. This particular feature has presented the users with an opportunity to watch other projects, and follower their developers; henceforth resulting in a constant stream of updates – not just about the developers, but also about their projects of interest [14]. This makes GitHub the center of attention for software engineering researchers, primarily because of its popularity coupled with the fact that it has integrated social features and metadata that one can access through the API. In the 21st century, there have been a range of studies on GitHub and its community; for instance, the studies conducted by Dabbish et al. [12] and Gousios and Andy [16] exclusively focused on the ways through which the GitHub social features were used by developers for forming impressions and drawing conclusions, while assessing the success, performance, and possible collaboration opportunities. On the other hand, there is also a range of quantitative studies; for instance, the study by Thung et al. [17] and Tsay et al. [18] that shed light on the systematically archiving the publically available data on GitHub and its use in investigating the development practices, in addition to the network structures in the environment.

B. Repositories and Github

According to Gousis and Andy [16], it has been suggested that repository is not necessarily a project, but a relatively new method that helps in collaborating in the distributed software development. This can be attributed to the typical pull request development model of GitHub, and with this particular model; the main repository of the project is not writable by potential contributors, instead, they make changes within the independent repository or on a clone [19]. When the set of changes are submitted by the contributor, a pull request is generated that allows the merging of the contribution in the main repository. However, these changes are not just integrated or merged into the main repository but are reviewed and inspected, which unveils whether the changes are unsatisfactory or satisfactory. In the case of satisfactory changes, the repository is merged into the master branch of the project; however, in the case of unsatisfactory changes, the project can call for further changes. From the latter investigation, it can be argued that repositories can be divided into a forked repository and base repository. In the case of the forked repository, the activities are recorded independently from the base repository. This can be further illustrated through an example of Ruby on Rails, which had approximately 8,327 forks and a total of 8,275 forks were made directly from the base repository, which means that the remainder is the forks of forks [20].

The social feature introduction, in the code hosting sites like GitHub, has drawn significant attention; for instance, the study by Lima et al. [21] suggested that the impressions are formed by GitHub users, which helps them in drawing a conclusion about their activities, while increasing the potential for both projects and their developers. The most prominent aspect here is related to the transparency, especially in reference to the social features that have helped them in maintaining their awareness level, while capitalizing upon this transparency to further organize their work. This can be critically important in the serverless software development, which can be further confirmed with the study of Casalnuovo et al. [23] where it has been suggested that higher visibility of the actions undertaken by developers can further influence their testing behavior. This is important since serverless software development requires constant testing, unlike non-software development apps that are more interested in just testing until a favorable outcome is reached.

In reference to serverless software development, GitHub can be influentially importance, since it has moved beyond just the social features. In particular, the study by Jurado et al. [24] argued that through the GitHub API, research projects have become highly accessible, which means that the data can be easily monitored and recorded based on their occurrence. Not just this, recursive dependency-based retrieval can be further capitalized upon in case of errors. This has even been reflected in the study Jurado and Pilar [24] and Yu et al. [25], where the authors found that even in the standalone run, the users were able to retrieve the history of individual repositories, and can be pulled from GitHub projects.

Since there is a large amount of data available at GitHub and tools like GitHub Archive, GHTorrent, and Gitminer, the developers can capitalise upon these tools for serverless software development, while taking advantage of the fact that this open-source forum can provide some valuable insights into the errors and issue reporting [19], programming languages [26] and project success [27].

C. Github Data Analysis

Being a popular platform amongst developers and coders, [21] argued that the analysis of social activities on the platform is amongst the new trends in software engineering. People have constantly been observing and reviewing the activities on GitHub repositories, which are then analyzed to gain insights into the repository features of the GitHub data. In particular, the study by Hauff et al. [22] focused on the activities of users on the platform, which further helped in conducting a quantitative analysis of the skills and interests of the users in reference to their observations; whereas Casalnuovo et al. [23] focused exclusively on relating the social link between the users and their language experience, while connecting with the productivity of the developers.

Since GitHub repositories are amongst the most important assets of users on GitHub, studies have unveiled that their quality and popularity are amongst the strongest indicators of the capabilities of the owner. This is the reason that the repository analysis on GitHub has gained exceptional importance; for instance, a study was conducted by Jurado et al. [24] in reference to the project issues related to the repositories on GitHub and found that there were some sentimental aspects related to the project issues. Yu et al. [25] studied the pull requests, which helped in discussing the complicated and complex issues related to the pull request evaluation latency, especially on the Git enabled social coding platforms. Another important study was conducted by Avelino et al. [26] that studied the truck factor of the popular repositories on GitHub, where truck factor represents the minimal number of developers that must leave before a project becomes unsustainable. The openness of the GitHub projects was analyzed by Cosentino et al. [27], where three important metrics were discussed; 1) the distribution of the project community, 2) the external contribution's acceptance rates, and 3) the time required for becoming project's official collaborator.

D. Data Mining in GitHub

Data mining in GitHub has remained a central focus in a range of studies, which have presented a meta-analysis in reference to software development practices and influence based on the use of the distributed social coding platform [28, 29]. This can be further illustrated through the following diagram:

Category of use	# of repositories
Software Development	275 (63.4%)
Experimental	53 (12.2%)
Storage	36 (8.3%)
Academic	31 (7.1%)
Web	25 (5.8%)
No longer accessible	11 (2.5%)
Empty	3 (0.7%)

Fig. 1. Mining example by Kalliamvakou et al. [33]

In particular, the study by Kalliamvakou et al. [30] presented some valuable information about the relationship between mining software repositories, data science, and operational data; meanwhile confirming that mining software repository is exclusively focused on extracting knowledge from the software data. The study further notified that Github repositories can be used for software development, storage, web and experimental purposes. In addition, it has further concluded that the mining software repository is actually a data science, as it focuses on the extraction of knowledge from data. This implies that the data mined from open source platforms are generally experimental data that can be analyzed to eliminate bugs and errors from codes and to further make the projects more reliable.

The latter has specifically been addressed in the study by Bird et al. [31], where the focus was to investigate the influence of a biased dataset on the performance of the bug prediction technique. In particular, the study indicated that a biased dataset is generally considered one, where the links between bug trackers and code repositories are missing. More importantly, the study confirmed that professional and experienced users have a critical role in fixing the bugs while establishing a link between bug trackers and code repositories. This is somewhat the case in GitHub, where data is mined by professional and experienced users, who help others by reducing the severity of bugs affecting their projects [32]. A wide range of studies has confirmed the existence of several possibilities of mining software repository. This implies that it allows the users to avoid reprocessing the same data several times since they can use an issue tracker for the collection and mining of the data. The most prominent issue trackers include JIRA, IssueZilla, and Bugzilla.

III. Research Methodology

A. Initial Dataset and GHTorrent

The creation of the initial dataset was driven by the desire to capture an expansive slate of open source repositories with serverless projects. GHTorrent was selected as the primary source of data given several factors associated with the project:

a) ***Data accumulation.*** the GHTorrent service monitors the Github public event timeline via service API. For each announced event, the service retrieves its contents and their dependencies, exhaustively, and then stores the raw JSON responses to several database types: MongoDB and MySQL. GHTorrent works in a distributed manner, using a RabbitMQ message queue which sits between event capturing activities and data dumping phases. This is done to orchestrate the monitoring process between clustered machines in a distributed manner. Most importantly, GHTorrent releases the data collected during the period as expansive downloadable archives which date back to the project's establishment. This became crucial due to the time-limited nature of queries that can be directly executed on the GHTorrent's database cluster. Reconstructing databases out of archived data permits us to run long data filtering queries, as well as preserving generated datasets for reproduction.

b) ***Longevity.*** The project has been active and expanding since 2013, recording 4TB of data as of 2015, in an attempt to capture both the current and future expansion, to peering all the way back to 2011 into Github's history to recreate and preserve the project data from that time. The thoroughness of the data storage process permitted GHTorrent to capture the birth and rapid expansion of serverless (FaaS) services starting with the AWS Lambda's debut in November 2014. This makes GHTorrent a suitable candidate for the study as it provides the majority of data and a roadmap to answer RQs.

c) ***Metadata.*** GHTorrent's database schema is composed out of 21 interconnected tables, with each one containing Github related metadata. For the purposes of the chosen topic, the most important tables are considered: projects, project_languages, project_members, commits, followers and watchers. These tables provide the majority of information necessary for a broad filtering of the project repositories according to description keywords (the project name and/or given description), number of commits (the project's pulse), number of forks, followers and watchers (its popularity according to different metrics), number of branches, commit messages and contributors (similar to previous metrics, but in a different dimension) and more.

d) ***Prolific status:*** GHTorrent's dataset has been used by multiple academics as a part of research papers and by companies to gather and extrapolate useful data. Considering that the topic delves into an area that is under-investigated and lacks clear-cut academic equivalents, the selection of past papers help the current study by providing examples of previously used data gathering and analysis techniques.

B. Static Code Analysis

For the purpose of this research, we use two types of tools: a) repository crawlers and b) source code analysis tools.

Crawlers, or bots, were designed to traverse the repositories in an autonomous or pre-defined manner, and extract valuable information about the projects. The primary use for crawlers in our study is to verify the validity of repositories by checking their status, collecting repository metadata and comparing it against the original set. If the discrepancy is identified, the former data is overridden to provide an updated perspective on the state of the repository. The secondary function of the crawlers is to download the valid repositories to local storage.

After cloned repositories are unpacked to local storage, static code analysis bots are used to extract the source code specific information, identify complexity, lines of code, size, flaws and architectural patterns used in the project.

We have gathered five bots and one GitHub crawler for the purposes of this project. The instruments in use are:

a) ***cloc***: is a command-line program that counts blank lines, comment lines, and physical lines of the source code of projects in many programming languages. It takes files, directory and/or archives names of projects as input, and outputs a table of programming languages used in this specific project with additional information. Cloc is fairly easy to use as it exists as a single file that needs minimum effort to install. With the help of this tool, we are analyzing top programming languages used in serverless projects on GitHub, and as well as the total number of files, blank, comment, and code.

b) ***LocMetrics***: LocMetrics counts the total lines of code (LOC), blank lines of code (BLOC), comment lines of code (CLOC), lines with both code and comments (C&SLOC), logical source lines of code (SLOC-L), McCabe VG complexity (MVG), and the number of comment words (CWORDS). This tool is similar to cloc but provides additional details on the projects.

c) ***Git-sizer***: this bot computes many size-related statistics about GitHub repositories. It provides an overall repository size of each project including the sizes of commits, trees, blobs, annotated tags, references, biggest object, history structure, and the biggest checkout. The instrument integrates with the local git command line and invokes it to analyze the target repository. By knowing each size of serverless projects, we can deduce the size metrics for different serverless projects, averages across repositories, according to languages, and more. New developers who are planning to deploy their applications into the serverless can get a grasp of what to expect.

d) ***Github crawler***: is a collection of scripts based on git-clone.sh that clone the list of repositories to a convenient directory in the home folder. This is used to clone hundreds of serverless project from GitHub to discuss and analyze architectural patterns. We feed the bash script with a list of all the URLs of serverless projects from the filtered GHTorrent list. The crawler also allows us to identify repositories that were deleted, renamed, or otherwise became invalid, and eliminate them out of the candidate pool.

e) ***Code Factor***: is a Static Code Analyser for C#, C, C++, CoffeeScript, CSS, Groovy, GO, JAVA, JavaScript, Less, Python, Ruby, Scala, SCSS, TypeScript. This particular tool performs code reviews, understands code quality issues, collects intelligence about code quality, and also tracks the performance of developers. The most important aspect of this tool is that it will provide us with the complexity of the project, number of methods, grading of each project, and the number of issues present in the project, which are used to understand the details on serverless projects.

C. *Assisted Repository Analysis*

As established in the proposal, the selection of the most popular (by following or contributions) serverless projects will be taken and manually analyzed to identify the purpose of repositories as described in the Table I. This will be used to classify projects according to their use-cases, as well as identify design details that could have eluded us during the previous phases. In addition to manual analysis, static code analysis tools like Codacy and/or Code Factor (in non-script-assisted mode) will be used to capture flaws, problems, complexity and other metrics used by said tools. The analysis within this phase will follow a codified protocol of actions established by the team.

TABLE I. PROTOCOL: ASSISTED REPOSITORY ANALYSIS

Step #	Description
1	Open the repository's GitHub page.
2	<u>Document Git metrics:</u> (1) number of issues, (2) pull requests, (3) # of commits, (4) the date of the last commit.
3	<u>Document social metrics:</u> (1) # of contributors, (2) # of subscriptions, (3) # of comments
4	<u>Document technical metrics:</u> (1) primary programming language, (2) language breakdown in %
5	Analyze the repository's README.md to determine the purpose of the repository. Document the purpose and use case.
6	<u>Add repository link into Code Factor and document further metrics</u>: (1) complexity, (2) duplication, (3) churn, (4) issues, (5) grade, (6) method, (7) LOC.
7	Cross-check inconsistencies between gathered metrics, GHTorrent data, and Bot-gathered data.

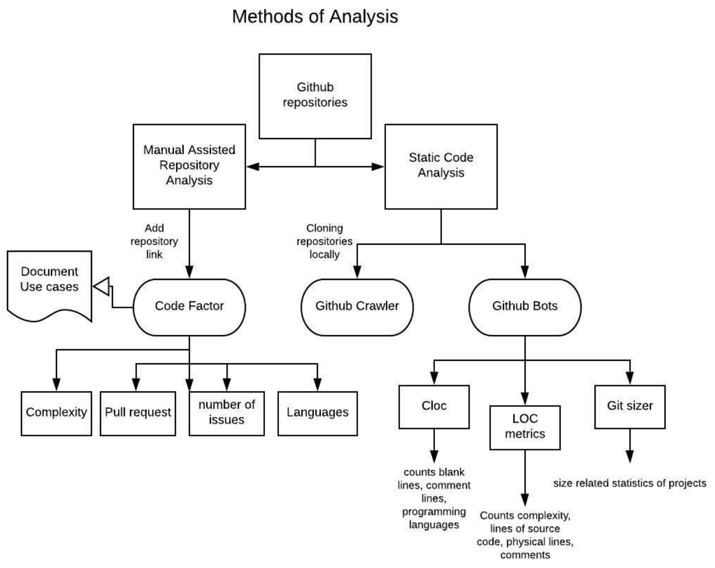
Methods of Analysis
Github repositories
Manual Assisted Repository Analysis
Static Code Analysis
Add repository link
Cloning repositories locally
Document Use cases
Code Factor
Github Crawler
Github Bots
Complexity
Pull request
number of issues
Languages
Cloc
LOC metrics
Git sizer
counts blank lines, comment lines, programming languages
Counts complexity, lines of source code, physical lines, comments
size related statistics of projects

Fig. 2. Research methodology and methods used in the data analysis

D. *Data Collection*

Given the aforementioned specifications, the initial dataset and a traversal map for repository crawlers was created by undertaking archive preparation, data recreation, and filtering the data in preparation of the final dataset.

For the purposes of this study, we have selected a GHTorrent archive dated to 2019-03-01 [34]. The archive was further expanded into a collection of CSV files worth 500 GB of data and representing the relational schema used by GHTorrent [35] to track Github's metadata. A local MySQL server was deployed to host the data, configured using InnoDB storage engine. Each CSV file was imported into the server, recreating GHTorrent's tables, with additional indices created for tables 'project', 'project_languages', 'project_members', 'watchers', ‘followers’, ‘commits’ and ‘pull_requests’ to improve the performance of the search functions.

Further steps included passing the data through several filters to narrow down the scope of repositories to serverless projects. The filtering steps

a) ***1st Filter:*** the initial filtering used keywords associated with serverless projects matching against the project titles and descriptions provided by the developers. While the team acknowledges that not all serverless projects might use said keywords as parameters for their repositories, given the numerical amount of records to parse through (over 3 billion records), this filtering was deemed as a viable tradeoff. Keywords that were used are illustrated in Figure 3. This step narrowed the set down to 750925 unique repositories.

aws, aws lambda, amazon lambda, lambda functions, azure, openwhisk, serverless, google cloud functions, microsoft azure, azure functions, ibm blue mix, bluemix, oracle fn, oracle cloud fn, kubernetes, kubeless, spotinst, ibm cloud functions, fn project, azure data lake, google cloud datastore, faundadb, picloud

Fig. 3. Keywords used in the initial filtering of GHTorrent data

b) ***2nd Filter*:** this filter focused on narrowing down the serverless projects further by defining the timescale and origins of serverless projects. With the Amazon’s Lambda service considered to be the main influencer behind the Function-as-a-Service (FaaS) and modern serverless paradigm [6], we limited the scope to repositories created after AWS Lambda came online - 2014-11-14. Furthermore, we’ve split the resulting set into two categories defined by being either an original source of code, or derivative projects based on original repositories known in GitHub ecosystem as ‘forks’. This resulted in a set of 251823 original repositories and 465900 forks. The main reason behind this separation was to limit the scope to the timescale relevant to serverless projects as well as permit better tracking of derivative projects in the next filters.

c) ***3rd Filter:*** this filtering step focused on filtering the repositories further by multiple criterias: the projects had to have a) more than just 1 commit, as well as b) more than a single contributor unless c) the single owner made more than 1 change in the repository; d) the projects had to have watchers who weren’t participating contributors; and e) weren’t forks made by the original project’s contributors. This step resulted in a set of 2595 original and 1094 forked repositories, with a total of 3689 projects identified in the set.

d) ***4th Filter*:** the final filtering step was concerned with removing projects unrelated to the serverless paradigm as well as repositories which could not be considered as applications. Projects containing only text, images, or other assets without deployable serverless code, as well as software projects which matched the keywords defined in the 1st filter, but had no relation to the serverless applications were removed. Lastly, repositories that were either deleted from Github or were orphaned (forks which had their parent projects deleted) were filtered out.

The final set contains 2194 repositories which were used in the static and assisted analysis as defined in the research methodology to produce results.

IV. Results

In this section, we present the results of our analysis within the three established dimensions, namely: (1), the common properties derived from the mined repositories, (2) the breakdown of use cases identified among analyzed projects, and (3) the analysis of dependencies on the major vendor.

A. RQ1 Prevalent characteristics of Serverless Projects

While performing the study, 3 properties were identified as the primary metrics pointers while analyzing the repositories. The data derived from said identifiers were used to classify the prevalent characteristics between serverless applications. The identified properties are (1) Structure of the repository, (2) Software Languages used in the project, and (3) the size of the project. The characteristics are further described in the following subsections.

a) Structure

One of the key factors that define the footprint of software applications are the components that constitute them. With the total of 2194 repositories examined, we were able to identify 152 distinct software artifacts which were classified into two categories: (1) primary and (2) secondary software artifacts. This distinction was made to separate the parts of the projects which contained the actual executable source code from the configuration files, miscellaneous scripts, documentation, graphical and other assets. The data breakdown is illustrated in Figure 4.

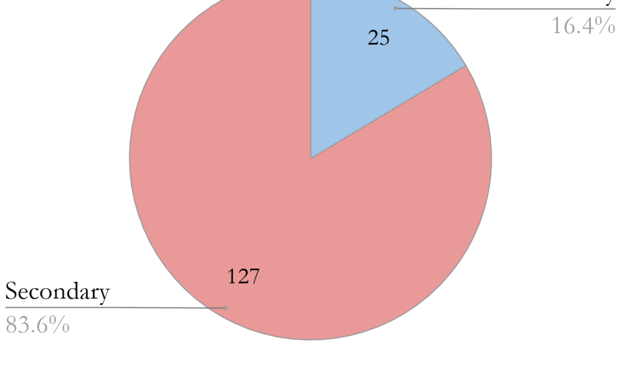

Fig. 4. Distribution of the primary and secondary artifacts

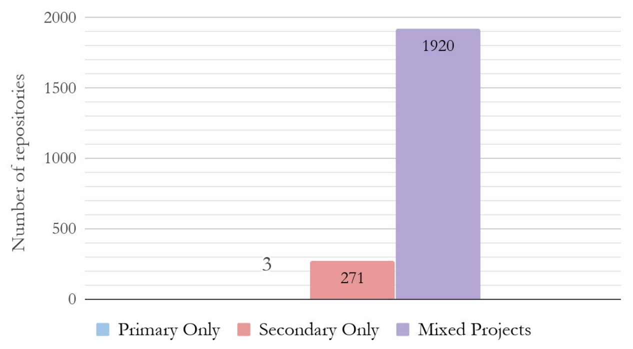

Fig. 5. Distribution of repositories by the artifact exclusivity

The core of source code type artifacts constitutes only a minority (16.4%) of all types. However, this can be offset by the fact that 104 (81.89%) out of 127 secondary artifacts have a lower than 1.00% occurrence across the entire set, relegating the majority of secondary software artifacts to isolated projects.

Further structural differences were derived by analyzing the overall set of serverless repositories against said groups. This yielded three categories of projects, defined by the exclusivity of use of the primary, secondary or a mix of both artifact types. The distribution is listed in Figure 5. The breakdown indicates that the overwhelming majority of the serverless projects (87.51%) contain a mix of primary and secondary types, while projects created using only the secondary artifact types (12.35%) or only source code languages (0.14%) are in the minority.

However, this can be attributed to the near-universal use of 'Markdown' files by GitHub's environment as the default format for documentation as well as 'Text' files for the development documentation. Filtering both out of the set produces a more accurate breakdown as shown in Figure 6. The mixed project still constitute the majority of projects (84.50%), the secondary only projects stay the same (12.35%), but the real number of pure source code based projects increases to (3.15%). The overall distribution of the projects

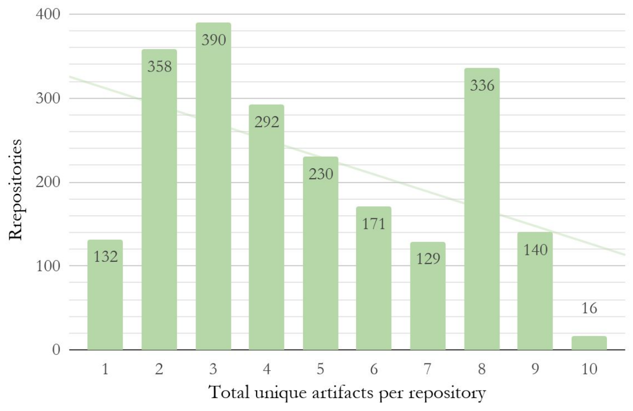

Fig. 7. Number of all unique software artifacts by repository

Lastly, the comparison of serverless projects by the quantity of unique artifacts provides a different indication across the established categories, illustrated in Table II and Figure 8. The data shows that more than half (59.34%) of existing OSS projects within the set use a single primary

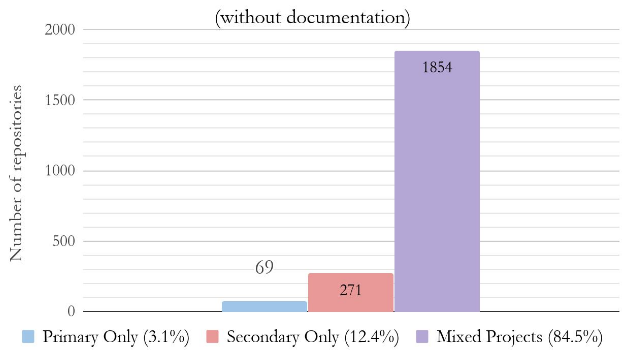

Fig. 6. Distribution of repositories by the artifact exclusivity, excluding secondary text-only artifacts

language, or at least one or two primary languages in 80.85% of projects. Concurrently, the secondary artifacts have a more gradual distribution with the majority (80.86%) of the projects using 2 or more unique types. This indicates that OSS projects favor developing serverless applications using one or two primary programming languages, but often expand the software's structure using secondary assets.

TABLE II. NUMBER OF ARTIFACTS PER PROJECT

#	Primary		Secondary		Total	
	Qty.	%	Qty.	%	Qty.	%
1	1302	59.34	351	15.99	132	6.02
2	472	21.51	446	20.33	358	16.32
3	112	5.1	335	15.27	390	17.78
4	23	1.05	278	12.67	292	13.31
5	12	0.55	252	11.49	230	10.48
6	1	0.05	274	12.48	171	7.79
7	0	0	154	7.02	129	5.88
8	1	0.05	32	1.46	336	15.31
9	0	0	3	0.14	140	6.38
10	0	0	0	0	16	0.73

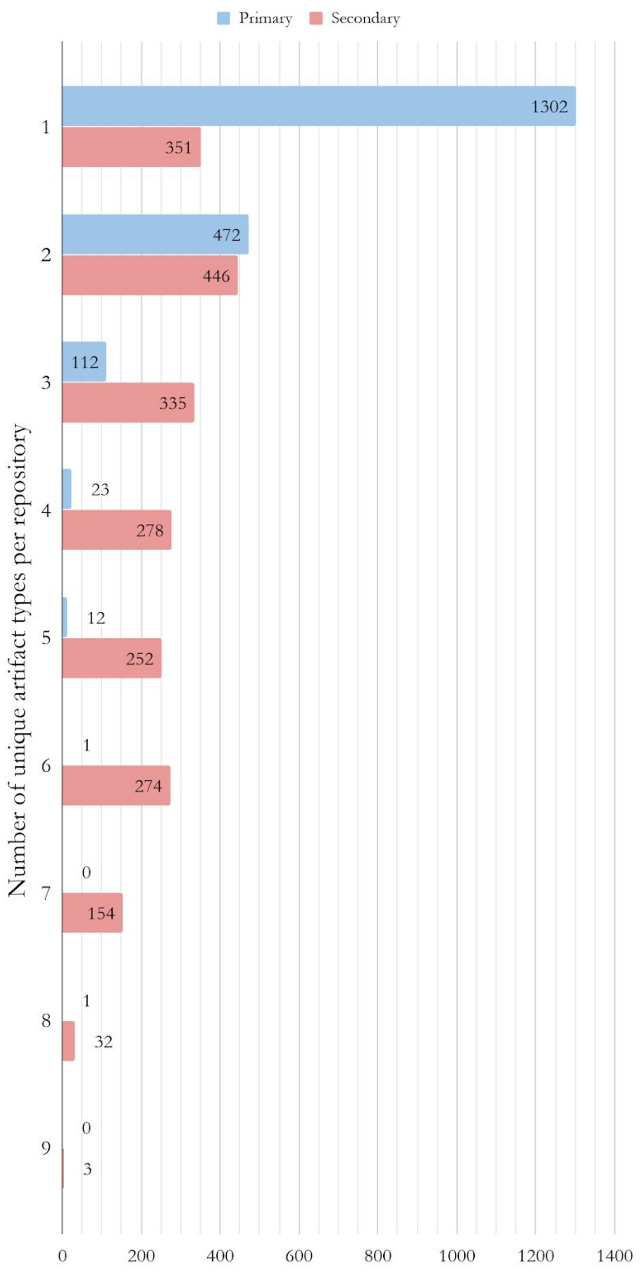

Fig. 8. Serverless repositories grouped by the number of unique software artifacts

TABLE III. PRIMARY LANGUAGES BY OCCURRENCE

Artifact	Occurrence %	# of Files
JavaScript	43.76	193730
Python	23.29	68452
C#	12.03	14064
Java	11.3	8930
Go	8.16	248026
Ruby	7.11	4507
TypeScript	4.24	11225
PHP	3.46	19675
C	3.14	4788
C++	3.1	19280
Perl	1.09	471
Objective-C	1.05	2776
Scala	1.05	502

b) Software Languages

The analysis of 25 Primary artifacts against the repositories yielded 13 programming languages that are used above the 1% threshold of occurrences in the examined projects. The data is showcased in Table III, with JavaScript, Python, and C# being the preferred development languages. The discrepancy between the occurrence and number of files was noted for languages like Go, Typescript, PHP and C++. This irregularity was introduced by the outlier repositories with the numerous quantity of files, such as large projects with the source code written using one language type.

A similar breakdown was performed across the Secondary artifacts, with 19 out of 127 types used above 1% usage threshold and illustrated in Table VI. The analysis identifies Markdown, Text, JSON, and YAML as the secondary data types prevalent in the set.

TABLE VI. SECONDARY LANGUAGES BY OCCURRENCE

Artifact	Occurrence %	# of Files
Markdown	88.15	45477
Text	61.39	24032
JSON	57.20	66216
YAML	49.73	29846
Shell	32.68	10216
HTML	24.93	12733
XML	24.29	28382
CSS	19.74	9056
INI	18.96	1199
Dockerfile	12.90	1595
Makefile	10.26	3064
SVG	9.43	5035
Batch File	7.70	519
Maven_POM	6.61	474
PowerShell	6.06	1506
HCL	4.10	1686
SCSS	3.92	2819
ASP	3.69	273
Ignore_List	3.65	237

c) Size

The total number of files per project was used to assess the general size and complexity of examined projects as well as establish the prevalence of size categories within the set. The data is presented in Figure 9 and Table V. According to the presented data, the majority of serverless are distributed in sizes between 6 to 50 files (57.50%).

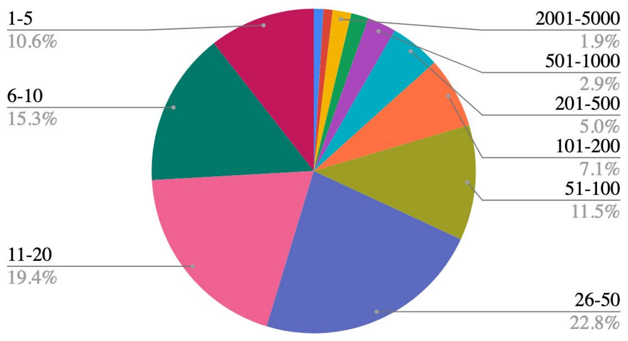

Fig. 9. Distribution of the repositories by the number of files

TABLE V. SERVERLESS PROJECTS BY THE NUMBER OF FILES

Number of Files	Repositories	Occurrence %
10001+	21	0.96
5001-10000	20	0.91
2001-5000	41	1.87
1001-2000	38	1.73
501-1000	63	2.87
201-500	110	5.01
101-200	155	7.07
51-100	252	11.49
26-50	500	22.79
11-20	426	19.42
6-10	336	15.31
1-5	232	10.57

B. *RQ2 Serverless Use Cases*

In total, we have manually analyzed 429 repositories to identify common use cases in serverless projects. In the end, 13 common use cases were derived from 267 repositories, with the occurrence within the subset presented in Figure 10 and Table VI. The remaining 162 repositories either fell into categories with less than 6 occurrences or had unique use cases that were difficult to generalize.

The following subsections contain key observations made by us about the major categories.

API: The largest number of repositories within the set contain APIs, which includes projects such as Key Vault Connector for Logic Apps, Source for the demo app API in Serverless-Stack.com. These API are set of tools that are used to communicate among and link various serverless components.

Framework: The framework is the next most common use case in serverless projects. These serverless frameworks are most used building application on AWS Lambda. An example of such a framework is the Real-time data analysis Framework.

Communication: this third most prevalent category was classified as libraries, extensions, and tools used for networking, communication and transmission of data. Most repositories within this category play a major role in the serverless projects as a number of repositories classified as such used moduled to transfer text, images and other kinds of information from one place to another, eg. a serverless app that posts messages to Slack.

Website: several Websites and web hosting platforms were identified within the set Websites are commonly hosted in the platform as well.

Database: serverless applications within this category fell into the data storage category via databases. However, most often the projects managed or linked components with the database services, such as DynamoDB, MongoDB, or relational databases.

Image Processing: this category relates to repositories that perform tasks related to image processing. For instance, Finpics use AWS Recognition to provide a faces search of finpics.com.

Alexa Skills: "Alexa Skills" is a feature of the Amazon platform used to expand the functionality of an Alexa bot.

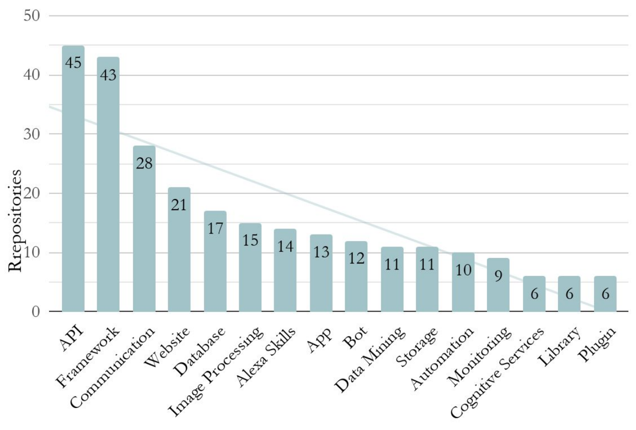

Fig. 10. Distribution of the identified use cases in the repositories

TABLE VI. IDENTIFIED SERVERLESS USE-CASES

Use-Cases	Repositories	Occurrence %
API	45	10.49
Framework	43	10.02
Communication	28	6.53
Website	21	4.9
Database	17	3.96
Image Processing	15	3.5
Alexa Skills	14	3.26
App	13	3.03
Bot	12	2.8
Data Mining	11	2.56
Storage	11	2.56
Automation	10	2.33
Monitoring	9	2.1
Cognitive Services	6	1.4
Library	6	1.4
Plugin	6	1.4
Total	*267*	*62.24*
Between 5 and 2	83	19.35
Unique cases	79	18.41

Most serverless projects within this category were created to demonstrate the commonly used hooks and expandability features of Alexa's platform.

App: this category primarily contains the small scale programs made to perform user-friendly functions as well as containing educational 'toy project' applications. Most of them were adapted to use AWS Lambda.

Bot: bots are used to do a specific task, and a number of repositories that were analyzed contain bot. An example of a Bot is Azure Bot to get information on an Azure Subscription.

Data Mining: all repositories within this category involve serverless tools for collecting and processing data in a semi-autonomous way.

Storage: storage services were noted to be responsible for managing the data preservation between serverless applications which weren't necessarily based on databases.

Automation: repositories within the Automation use-case involved minimizing the user and power-user level input for tasks via serverless schedulers, scripts, and other techniques. For example, a serverless function that automates the enforcement of Multi-Factor Authentication to all AWS IAM users.

Monitoring: repositories within this category are serverless applications that are designed to monitor various serverless components. An example of such application will be stack overflow monitor monitors stack overflow questions and post them in a slack channel.

Cognitive Services: repositories within this category are APIs, SDK and services that help developers to build intelligent applications. For instance, computer vision API where one can test their own image.

Library: repositories within this category represent the collection of useful utilities for serverless applications. An example of such a tool will be JavaScript-based pipelines and utility hooks for NodeJS.

Plugin: software component which add extra features to already existing serverless application and platforms were categorized as plugins, eg. serverless functions which expand the container orchestration modules of Kubernetes.

C. RQ3 Serverless Vendor Dependency

In total, the authors have manually analysed 429 repositories to identify the most popular cloud service provider to deploy serverless applications in. The reason for manually analysing all these repositories is for reliability and accuracy.

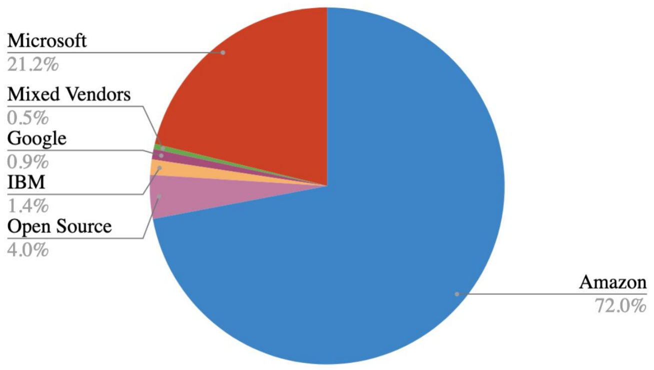

Fig. 11. Distribution of vendor dependencies among the serverless projects

TABLE VII. SERVERLESS VENDOR DEPENDENCIES

Vendor	Repositories	%
Amazon	309	72.03
Microsoft	91	21.21
IBM	6	1.40
Google	4	0.93
Open Source	17	3.96
Mixed Vendors	2	0.47
Total	*429*	*100.00*

V. DISCUSSION

A. RQ1 Prevalent characteristics of Serverless Projects

The analysis of the large serverless dataset has brought several common traits among the examined projects to the authors' attention. The majority of the serverless repositories seem to be based around one or two core programming while using scripts, markdown, configuration files and other assets to expand their functionality. The primary programming languages used to build serverless components are JavaScript, Python and C#. This data correlates with the data gathered in the related literature [6]. The complexity which secondary artifacts introduce varies greatly across the repositories, as unique secondary data types between 2 to 6 per repository are nearly equally represented across the dataset. More so, similar distribution can be noted in regards to the size of serverless projects, as 67.95% of examined projects are no bigger than 50 files, while tiny projects up to 20 files encapsulate nearly 46% of the entire set, thus indicating that serverless OSS application lean towards smaller-scale deployments.

B. RQ2 Serverless Use Cases

The purpose of the study for RQ2 is to provide the common use cases in open-source serverless projects. The significance of the findings for the RQ2 in light of the case study among the developers within the field [6] , indicates that use-cases are similar to an extent in both papers. Database and API are most used in serverless in the published paper, whereas in this research paper API and serverless frameworks leading the chart. A significant amount of open-source serverless projects are analysed to justify the answer. In this study, we have used more categories to distinguish each repository compared to the related literature [6], as a reader can have a wider perspective on what other use cases are present in the serverless projects.

C. RQ3 Serverless Vendor Dependency

The majority of the analyzed repositories were deemed to be dependent on the large vendors, primarily Amazon (72.03%) and Microsoft (21.21%) as shown in Table VII. Only 3.96% of OSS projects were using open source frameworks that were openly available to be instantiated by both commercial vendors and end-users.

D. Threats to validity

Throughout the study we have identified 3 major threats to validity which might affect the results of this study.

Internal: Drawing conclusions between different types of serverless applications or different target platforms can introduce selection bias. To mitigate the bias, the major dataset was selected randomly according to specifications established in the Research Methodology.

External: The filtering methods that were used during the first stages to create the initial dataset can affect the generalizability of outcomes, especially due to the lack of reliability in the unfiltered data from Github repositories. To mitigate this threat, the authors tried to remove unreliable, irrelevant, and empty data, forming the set out of repositories which referenced different serverless platforms.

Reliability: The choice of analysis tools, dataset curation, the state of repositories when GHTorrent captured the snapshot or automated tools analyzed the data can have a decisive effect on the accuracy of the study. The lack of related literature and the previously devised data evaluation protocol that could be applied to our study affects the reliability of data. To promote the replicability we published the set and data breakdown used in the analysis which are accessible online [36].

VI. Conclusion

This research aims to provide readers an insight into the serverless development trends in open source Github projects. Based on the comprehensive qualitative and quantitative analysis of the serverless projects in Github, the study indicates that serverless is gaining popularity within GitHub communities. The existing projects cover a myriad of use cases: from APIs and Backend services to Storage, Automation and Data processing. The numbers indicate that technical alternatives exist to traditional architectural patterns with multiple application fields already explored by the OSS projects. With serverless paradigm slashing the cost of infrastructure maintenance, the developers can solely focus on the main objective, which is creating the product. Moreover, the most common programming languages used to build the components are JavaScript, Python, and C#.

However, while serverless brings benefits to the developers, the data indicates that the majority of the OSS projects are created to fit specific cloud service providers, with the Amazon Web Services and Microsoft Azure's ecosystems dominating the examined applications. While fully independent serverless platforms exist, only a fraction of the projects used them. This might carry both positive and negative consequences as the dominance of AWS and Azure can be linked to the affordable price models, service stability, brand exposure as much as the vendor lock-in, service dependencies, and dealing with inflated costs if applications aren't tailored well to the vendor's ecosystem. Further research can be done to derive changes in the growth and emergence trends between proprietary ecosystems and independent platforms as well as use cases and software artifacts, using the social, timestamps, and code quality metrics accumulated in the study's dataset.

Acknowledgments

The authors of this research would like to thank the supervisor Joel Scheuner for guidance. The help of his supervision sessions allowed authors to achieve the objectives of this study.

References

[1] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, "Status of serverless computing and function-as-a-service (FaaS) in industry and research," in *1st Int. Workshop on Serverless Computing (WoSC)*, Atlanta, GA, USA, Jun. 2017.

[2] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, et al., "Serverless Computing: Current Trends and Open Problems," in Research Advances in Cloud Computing, S. Chaudhary, G. Somani, and R. Buyya, Eds., Singapore: Springer, 2017, pp. 1–20.

[3] A. Baird, G. Huang, C. Munns, and O. Weinstein, "Serverless Architectures with AWS Lambda: Overview and Best Practices", Amazon, p. 44, Nov. 2017. [Online]. Available: https://d1.awsstatic.com/whitepapers/serverless-architectures-with-aws-lambda.pdf [Accessed: 30-Mar-2019]

[4] DigitalOcean, "Currents: A Quarterly Report on Development Trends in the Cloud," p. 22, Jun. 2018. [Online]. Available: https://www.digitalocean.com/assets/media/currents-research/pdf/DigitalOcean-Currents-Q2-2018.pdf [Accessed: 30-Mar-2019]

[5] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, et al., "Serverless Computing: One Step Forward, Two Steps Back", in *9th Conf. on Innovative Data Systems Research (CIDR'19)*, Pacific Grove, CA, USA, Jan. 13 - Jan. 16, 2019.

[6] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A Mixed-method Empirical Study of Function-as-a-Service Software Development in Industrial Practice" in *Journal of Systems and Software*, Vol. 149, pp. 340–359, 2019.

[7] AWS Serverless Application Repository. [Online]. Available: https://aws.amazon.com/serverless/serverlessrepo/ [Accessed: 30-Mar-2019]

[8] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell et al., "Serverless computing: Current trends and open problems." *Research Advances in Cloud Computing*, pp. 1-20. Springer, Singapore, 2017.

[9] A.D. Urso, "A Serverless Instant Messaging Protocol for Mobile Ad Hoc Networks." in 8*th Int. Conf. on Creating, Connecting and Collaborating through Computing*, pp. 71-75. IEEE, 2010.

[10] M. Moula and V. Mancuso, "Experimental performance evaluation of WebRTC video services over mobile networks." in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 541-546. IEEE, 2018.

[11] O. Alqaryouti and N. Siyam, "Serverless Computing and Scheduling Tasks on Cloud: A Review." in *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS) 40*, no. 1, pp 235-247, 2018.

[12] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social Coding in GitHub: transparency and collaboration in an open software repository." in *Proc. of the ACM 2012 conference on computer supported cooperative work*, pp. 1277-1286. ACM, 2012.

[13] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer, and D. M. German, "Open source-style collaborative development practices in commercial projects using GitHub," in *37th International Conference on Software Engineering*, vol 1, pp. 574-585. IEEE Press, 2015.

[14] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," *in Proc. of the 11th working conference on mining software repositories,* pp. 92-101. ACM, 2014.

[15] I. Keivanloo, C. Forbes, A. Hmood, M. Erfani, C. Neal, G. Peristerakis, and J. Rilling, "A linked data platform for mining software repositories." in *Proc. of the 9th IEEE Working Conference on Mining Software Repositories*, pp. 32-35. IEEE Press, 2012.

[16] G. Gousios and A. Zaidman, "A dataset for pull-based development research," in *Proc. of the 11th Working Conference on Mining Software Repositories*, pp. 368-371. ACM, 2014.

[17] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang, "Network structure of social coding in GitHub," in *17th European Conf. on Software Maintenance and Reengineering*, pp. 323-326. IEEE, 2013.

[18] J. Tsay, L. Dabbish, and J. Herbsleb, "Social media and success in open source projects," *in Proc. of the ACM 2012 conference on computer supported cooperative work companion*, pp. 223-226. ACM, 2012.

[19] J. Perkel, "Democratic databases: science on GitHub," in *Nature News 538*, no. 7623, p. 127, 2016.

[20] P. Wagstrom, C. Jergensen, and A. Sarma, "A network of rails: a graph dataset of ruby on rails and associated projects," *in Proc. of the 10th Working Conference on Mining Software Repositories*, pp. 229-232. IEEE Press, 2013.

[21] A. Lima, L. Rossi, and M. Musolesi, "Coding together at scale: GitHub as a collaborative social network," *in 8th Int. AAAI Conference on Weblogs and Social Media*, 2014.

[22] C. Hauff and G. Gousios, "Matching GitHub developer profiles to job advertisements," in *Proc. of the 12th Working Conference on Mining Software Repositories*, pp. 362-366. IEEE Press, 2015.

[23] C. Casalnuovo, B. Vasilescu, P. Devanbu, and V. Filkov, "Developer onboarding in GitHub: the role of prior social links and language experience," *in Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 817-828. ACM, 2015.

[24] F. Jurado and R. Pilar, "Sentiment Analysis in monitoring software development processes: An exploratory case study on GitHub's project issues," in *Journal of Systems and Software*, pp. 82-89,

2015.

[25] Y. Yu, W. Huaimin, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: determinants of pull request evaluation latency on GitHub," in *12th Working Conf. on Mining Software Repositories*, pp. 367-371. IEEE, 2015.

[26] G. Avelino, M. T. Valente, and A. Hora, “What is the Truck Factor of popular GitHub applications? A first assessment,” No. e1683. PeerJ, 2015.

[27] V. Cosentino, J. L. Cánovas Izquierdo, and J. Cabot, "Three metrics to explore the openness of GitHub projects," in *arXiv preprint*,1409.4253, 2014.

[28] G. Gousios and D. Spinellis, "Mining software engineering data from GitHub," in *39th Int. Conf. on Software Engineering Companion*, pp. 501-502. IEEE, 2017.

[29] J. Fowkes and C. Sutton, "Parameter-free probabilistic API mining across GitHub," in *Proc. of the 24th Int. Symposium on Foundations of Software Engineering*, pp. 254-265. ACM, 2016.

[30] A. Mockus, "Is mining software repositories data science," in *Proc. of the 11th Working Conference on Mining Software Repositories,* pp. 1-1, ACM, 2014.

[31] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: bias in bug-fix datasets," in *Proc. of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT Symposium on The foundations of software engineering*, pp. 121-130. ACM, 2009.

[32] R. Hebig, T. Ho Quang, M. Chaudron, G. Robles, and M. A. Fernandez, "The quest for open source projects that use UML: mining GitHub," in *Proc of the 19th International Conference on Model Driven Engineering Languages and Systems*, pp. 173-183. ACM, 2016.

[33] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proc. of the 11th working conference on mining software repositories*, pp. 92-101. ACM, 2014.

[34] G. Gousious, GitHub metadata MySQL datadump, [Online] http://ghtorrent-downloads.ewi.tudelft.nl/mysql/mysql-2019-03-01.tar.gz [Accessed: 31-Mar-2019]

[35] Gousios, Georgios, "The GHTorrent dataset and tool suite," In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 233-236. IEEE Press, 2013.

[36] I. Pavlov, S. Ali, T. Mahmud, Final dataset and analysis data from “Serverless Development trends in Open Source: a mixed-research study”, 2019. [Available]: https://docs.google.com/spreadsheets/d/10FKwBFA4zmROXGU3rTdIgf4BcNj5fIQWkGF5xQuyC-s [Accessed: 10-Sep-2019]