Microservice Applications and Their Workloads on GitHub

Author Version. Final Version published by ACM.

Yannik Lubas
University of Würzburg
Würzburg, Germany
yannik.lubas@uni-wuerzburg.de

Martin Straesser
University of Würzburg
Würzburg, Germany
martin.straesser@uni-wuerzburg.de

Ivo Rohwer
University of Würzburg
Würzburg, Germany
ivo.rohwer@uni-wuerzburg.de

Samuel Kounev
University of Würzburg
Würzburg, Germany
samuel.kounev@uni-wuerzburg.de

André Bauer
Illinois Institute of Technology
Chicago, IL, USA
andre.bauer@iit.edu

Abstract

Many cloud applications employ a microservice architecture to reap its various benefits, such as independent scalability and development. However, microservices' distributed nature comes with its own challenges, which the literature tries to combat with various approaches in domains, such as autoscaling or service placement. Researchers, especially performance engineers, need representative microservice applications as a basis to develop and evaluate their approaches. While the literature proposes several reference and benchmark applications, the set of commonly used applications is rather limited. Further, the representativeness of reference applications for industry practices has been challenged, raising questions about the applicability of existing approaches. To investigate possible alternatives to and inspirations for microservice testbeds, we conduct an exploratory search of microservice applications and their workloads by mining GitHub repositories. Our work provides two datasets containing 553 applications and 8 workload dataset repositories, respectively. In addition, we provide a first analysis of the collected data that future work can build upon.

CCS Concepts

• **Software and its engineering → Software performance**; • **Computer systems organization → Cloud computing**.

Keywords

GitHub, microservice applications, microservice workloads, dataset, repository mining

ACM Reference Format:
Yannik Lubas, Martin Straesser, Ivo Rohwer, Samuel Kounev, and André Bauer. 2025. Microservice Applications and Their Workloads on GitHub: Author Version. Final Version published by ACM.. In *Companion of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '25), May 5–9, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3680256.3721321

ICPE Companion '25, May 5–9, 2025, Toronto, ON, Canada

ACM ISBN 979-8-4007-1130-5/2025/05
https://doi.org/10.1145/3680256.3721321

1 Introduction

In recent years, the microservice architecture has been widely used to develop and operate applications in the cloud. Large companies, such as Meta [14], Microsoft [22], or Alibaba [24], have been successful in utilizing microservice applications to offer their products to customers. Similarly, researchers have also taken an interest in microservice architectures. Although microservices' distributed nature offers many advantages to application development and runtime, their increased complexity inherently creates challenges.

The literature proposes various approaches in domains, such as autoscaling [10], root cause localization [21], or service placement [2] to combat these challenges. Researchers, especially performance engineers, need representative microservice applications as a basis to develop and evaluate these approaches. Hence, the literature commonly uses a set of reference or benchmark applications (e.g., DeathStarBench [11], TrainTicket [26], or AcmeAir [1]). However, most benchmarks have been discussed to reflect microservice applications from the industry inaccurately [19]. The set of commonly used application testbeds is fairly small, leading to a limited evaluation scope of new approaches.

Existing reference applications are slowly evolving, as researchers most often do not actively maintain and develop them after their initial release. For instance, AcmeAir has not been updated for almost a decade; however, it is still used for evaluation in recent approaches [6]. Additionally, the applications used in the literature mirror laboratory-like environments, as they were purposefully developed for microservice research. In contrast, our experience shows that many applications in the wild are often more diverse due to the dynamic nature of software projects. Consequently, reference applications do not capture new trends and the evolvement and variety of emerging technologies. This observation challenges the applicability of existing approaches in real-world scenarios.

We conduct an exploratory search of microservice applications and their workloads by mining GitHub repositories. We see multiple use cases enabled by our data. First, researchers can investigate possible alternatives to established microservice testbeds. Second, the dataset can serve as inspiration for future microservice research.

We aim to give researchers an overview of publicly available microservice applications and related workload datasets on GitHub. The contributions of our work are as follows:

- A dataset of publicly available microservice applications on GitHub that researchers can use as a starting point for future research
- An analysis of the applications in the dataset, including their number of microservices, used programming languages, and common technology stacks
- A dataset of publicly available cloud and cluster workload traces on GitHub that researchers can use as a reference for finding real-world trace data

2 Related Work

Baresi et al. [8] analyze microservice architectures on GitHub based on Docker configuration files. Their study uses the information from `Dockerfiles` and `docker-compose` files to extract application sizes, static dependencies, and technology stacks. After a manual filtering step, their dataset contains 24 repositories.

Amoroso d'Aragona et al. [3] manually analyze 3,804 projects obtained from the World of Code (WoC) dataset. Their search includes different types of projects, such as microservice applications, libraries for building microservices, and tools for monitoring applications. The final dataset comprises 378 repositories.

Previous works heavily rely on information from `Dockerfiles` and `docker-compose` files. However, our manual analysis showed that the state of the overall project and, e.g., `docker-compose` files often is misaligned. Furthermore, microservice projects not using `docker-compose` in their deployment stack are excluded from such approaches.

Baresi et al. [7] analyze 651 microservice GitHub repositories regarding their activity (e.g., pull requests, stars, or watchers) and main programming languages. Their findings show that Java is the most common language, with Go and JavaScript being second and third.

Assunção et al. [4] analyze the evolution of eleven microservice repositories. Their results indicate that the driving factor for the evolution of microservice applications is technicalities rather than business logic.

Nasab et al. [16] extract 28 security practices from ten GitHub open-source microservice applications and 308 Stack Overflow posts. They validate the practices' usefulness by surveying 74 microservice practitioners.

Waseem et al. [23] categorize 1,345 issues from five open-source GitHub repositories to highlight the types of problems and their triggers that developers encounter during microservice development. They find that technical dept from legacy versions and dependencies, build problems from invalid configuration and communication, security problems from poor security management, and service execution errors from general programming errors are the most prevalent issues.

While related work focuses on different aspects, such as security, project evolution, and common issues during development, our work specifically targets performance engineers. Hence, we search for repositories with executable applications potentially useful for benchmarking. Unlike other approaches, we do not include non-application projects, such as libraries for building microservice applications. In addition, our data includes real-world workloads that performance researchers can use to conduct more realistic experiments.

3 Methodology and Research Questions

Figure 1 illustrates the three phases of our methodology, namely *search*, *preprocessing*, and *analysis*. In the first phase, we query the official GitHub Rest API[1] with various query terms to find repositories that either contain microservice applications or workload datasets. For microservice applications, we construct query terms from different spellings of the keyword *microservice* in combination with *application* and *system*. For workload datasets, we build query terms from different combinations and spellings of the keywords *cluster*, *data*, *workload*, *trace*, *microservice*, *cloud*, *serverless*, and *tracing*. The complete list of query terms is included in our replication package[2]. We further restrict the results to public, non-archived repositories that were last pushed to between January 1st, 2020 and January 15th, 2025. In total, we find 47,655 application and 53,701 workload repositories.

During the *preprocessing* phase, we aggregate the results for the application and workload projects, filtering out duplicates, forks, and template repositories. The GitHub API also reports the primary programming language of each project. Thus, we filter out application repositories for which GitHub cannot detect a programming language, e.g., repositories only containing Markdown files. Lastly, we define a minimum popularity threshold based on *stars* or *forks* count as an indicator of active usage [9]. *Staring* a repository allows GitHub users to keep up to date with projects and is similar to a *like* on other social media platforms. In our case, we set the popularity threshold to a minimum of five *stars* or *forks*. With this, 940 application projects and 1,716 workload repositories remain.

In the next step, we manually filter the remaining repositories to include only relevant entries in our analysis. While filtering application repositories, we exclude projects that do not contain a microservice application (e.g., libraries or monitoring tools). Similarly, we only retain workload repositories that contain or provide ready-to-use datasets.

Two experienced researchers independently evaluate repositories regarding their relevance. If they are unsure about a project, they mark it for peer review. In the peer review, we discuss and agree on whether to filter the project. As a result, we came up with application projects to 553 and workload repositories to 8.

In the last phase, we analyze the final 553 repositories both manually and automatically. We manually count the number of microservices and supporting components of every application. In our understanding, a supporting component is a deployable software part of the application that does not contain business logic specific to the application. The components might not necessarily be microservices but must be part of the standard deployment of the application. For example, we consider databases, API gateways,

[1] GitHub Rest API: https://docs.github.com/en/rest
[2] Replication Package: https://github.com/DescartesResearch/microservice-application-and-workload-dataset

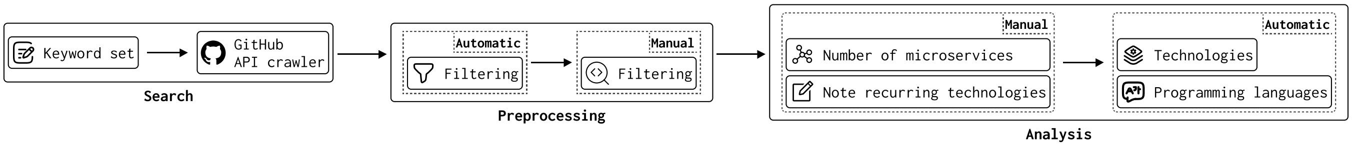

Figure 1: Overview of our methodology for searching microservice applications

Table 1: The 51 technologies and their respective categories

Category	Technology
Gateway	nginx, Zuul, Kong, Envoy, Traefik, Ocelot
Data Storage	MongoDB, MySQL, PostgreSQL, SnowflakeDB, OracleDB, MsSQL, Redis, Cassandra, MariaDB, ElasticSearch, MinIO, Other Databases
Observability	Prometheus, Jaeger, Zipkin, OpenTelemetry, Logstash, Filebeat, Hystrix, Kiali, Grafana Kibana, Akhq, Portainer
Message Queue	Kafka, RabbitMQ, Nats
Authentication	Keycloak, Vault
Benchmark Tooling	Locust, K6, JMeter
Frontend	React, NextJS, Svelte, SvelteKit, VueJS, Nuxt, AngularJS, AnalogJS
Communication	Dapr, Istio, Consul
Other	Eureka, Zookeeper

discovery servers, message queues, observability tools, and dashboards, such as Grafana and Kibana, as supporting components.

During this process, the researchers make note of recurring technologies, such as specific databases, frontend frameworks, or message queues. We compile a list of 51 technologies microservice applications might use in their technology stack. Using the GitHub Code Search API, we automatically search the application repositories for these technologies by searching for the technology name in specific files or paths or matching necessary parts of configuration files for a particular technology. Our replication package[2] includes the concrete query terms for each technology. Table 1 lists the set of technologies that we searched for. We group the technologies into categories that we later use in our analysis.

Lastly, we clone every project and use the `github-linguist`[3] tool to detect the used programming languages for each repository automatically. We filter out certain languages, such as configuration and template languages. In addition, we only consider languages with a usage of more than 1% in a particular project. We provide all processing steps, raw API query responses, and the final datasets in our replication package.[2]

Research Questions: We define the following research questions for our analysis:

- **RQ 1:** What are popular programming languages for microservice applications? Are there any trends noticeable in language usage over time?
- **RQ 2:** How many microservices and supporting components are used per project?
- **RQ 3:** What are popular supporting components?
- **RQ 4:** Are there any supporting components and programming languages commonly used together?

[3]GitHub Linguist: https://github.com/github-linguist/linguist

4 Findings

This section presents our findings to answer the research questions from the previous section. Moreover, we describe the workload repositories in more detail.

4.1 Programming Languages

To address RQ1, we examine the programming languages used across all repositories in our dataset. We identify 24 different languages, with their distribution shown in Figure 2. We select the six most common languages for display and group the remaining 18 as *Other*. The x-axis represents the languages, while the y-axis indicates the number of repositories that use the particular language. The most used language is Java, closely followed by JavaScript. Both languages are used in more than 200 repositories. TypeScript and C# were found in more than 100 repositories. Over 50 repositories include microservices programmed in Python or Go. Notably, 297 repositories use exactly one programming language, while 170 use exactly two languages. Only 31 projects use four or more programming languages, with one repository employing the maximum of ten languages. JavaScript is often combined with other languages. The top three combinations of languages are JavaScript and Java (98 repositories), JavaScript and TypeScript (82), and JavaScript and Python (47).

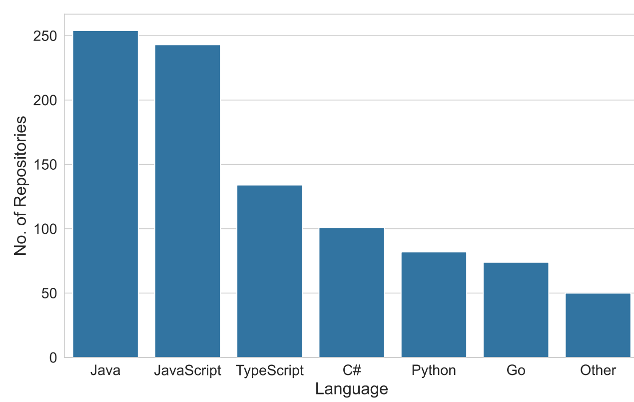

Figure 2: Language usage distribution

Besides this distribution, we are interested in whether and how language usage has changed over time. To this end, we analyze the distribution of programming languages in newly created projects per year from 2015 to 2024, as shown in Figure 3. Note that the years before 2020 do not necessarily give a representative view due to our query filters. The x-axis represents the years, while the y-axis indicates the percentage of repositories created in that year using a particular language. Note that the sum of percentages may exceed 100%, as repositories may use multiple programming languages at once. In all years except 2019, 2021, and 2022, Java is the most used language in newly created projects, with JavaScript ranking second. Since 2020, C# usage has declined, dropping from the third most used language to roughly 14% usage in new projects by 2024.

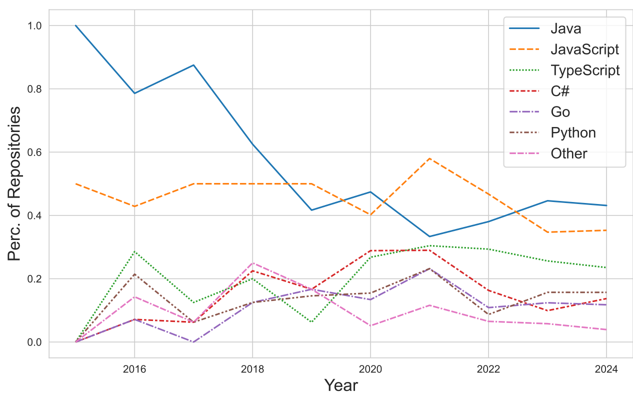

Figure 3: Language usage trend by creation year

4.2 Microservices and Supporting Components

As mentioned in Section 3, we consider two types of application components: microservices and supporting components. Our analysis reveals that the number of microservices ranges from a single microservice to a complex system with 14 services. The average number of microservices per application was 4.04, while the median was 3. The number of supporting components exhibits a similar range, from 0 to 21, with a median of 3 and an average of 3.58. When considering the combined count of microservices and supporting components within each application, the range increases from 1 to 33, with a median of 7 and an average of 7.62.

The distribution of the components is illustrated in Figure 4, with the horizontal axis representing the number of components and the vertical axis reflecting the cumulative distribution function. Half of the repositories contain three or fewer microservices; 80% have at most five microservices. Similarly, about 50% of repositories employ at most three supporting components, and 90% have at most eight, while the remaining 10% use up to 21. Considering the total component count, 55% of repositories contain at most seven components, and 90% have fewer than 14.

4.3 Technologies and Languages

To answer RQ3, we first categorize the 51 supporting components into nine categories as shown in Table 1 and visualize their distribution in Figure 5, with the x-axis showing the different categories and the y-axis the percentage of all repositories using at least one technology of that particular type. The most common category is

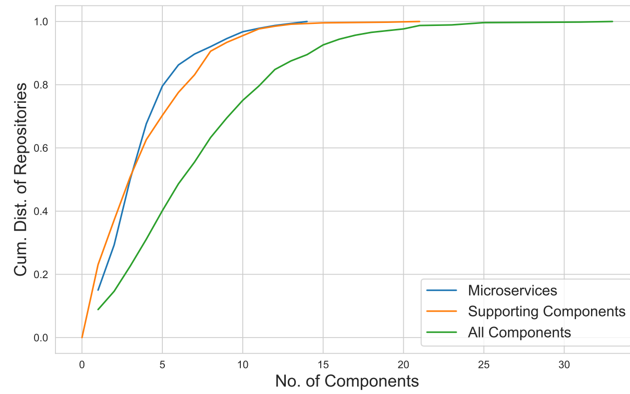

Figure 4: Cumulative distribution of microservices as well as supporting and total components

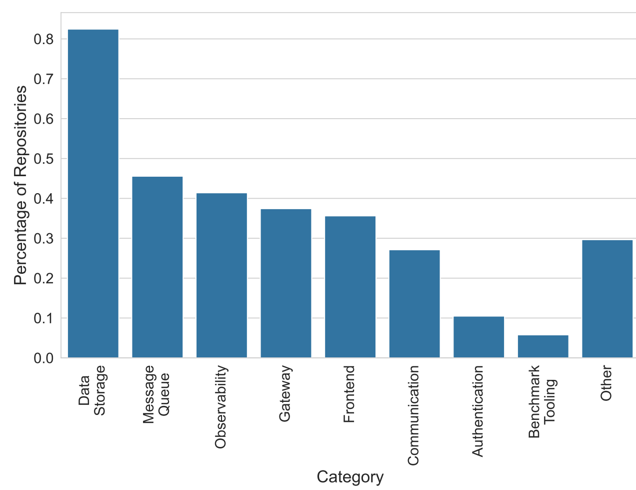

Figure 5: Supporting component categories distribution

Data Storage, used in 82% of the analyzed repositories. *Message Queue* usage ranks second, closely followed by *Observability* and *Gateway*. *Benchmark Tooling* is only supported by fewer than 10% of repositories.

In the following, we describe the distribution of technologies from two specific categories as illustrative examples in more detail. Figure 6 shows the usage of specific databases. The x-axis depicts the different databases, and the y-axis marks the number of repositories employing each database. *MongoDB* is the most widely used, appearing in 193, followed by *Redis* in 176 repositories. *PostgreSQL* and *MySQL* rank third and fourth, respectively. These top four databases appear in at least 100 more repositories than any other database.

Similarly, we examine the most used gateways, as shown in Figure 7, where the x-axis represents the different gateways and the y-axis indicates the number of repositories utilizing each one. *Nginx* is by far the most commonly used gateway, appearing in 130 repositories. *Zuul* ranks second, followed by *Ocelot*.

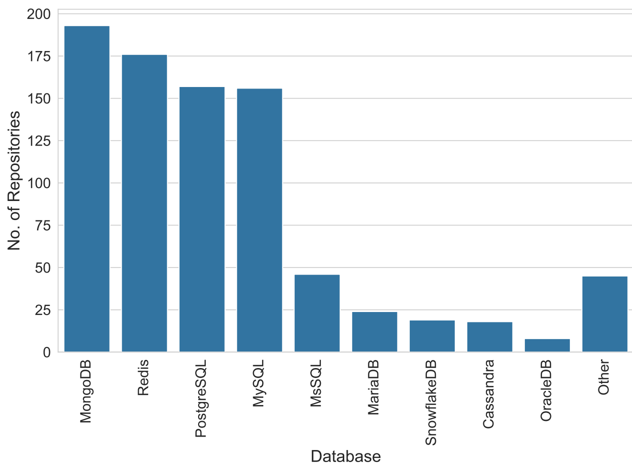

Figure 6: Database usage distribution

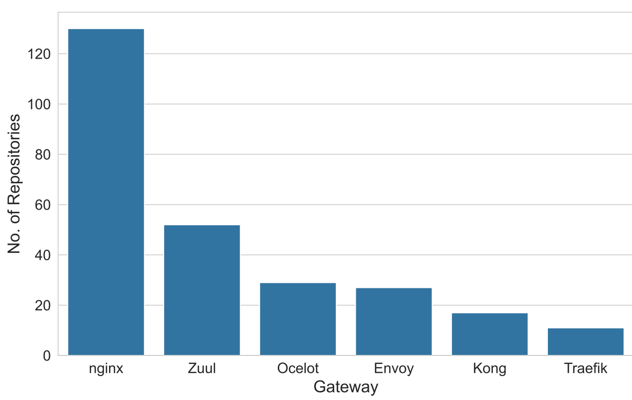

Figure 7: Gateway usage distribution

To identify the usage of containerization technologies, we looked at Docker, `docker-compose`, and Kubernetes usage. With 59%, the usage of Docker is fairly high among the repositories. Just under half (46%) of projects include `docker-compose` files. In our data, about 23% of repositories provide Kubernetes manifests for deploying their applications.

To address RQ4, we examine the associations between programming languages and supporting components, as illustrated in Figures 8 and 9. In Figure 8, the horizontal axis presents the six most common languages, with the remaining ones grouped as *Other*, while the vertical axis features 19 selected supporting technologies. Each heatmap cell represents the relative usage of a supporting component across repositories that use a given language. For instance, 47% of projects using TypeScript also use MongoDB as a database, and 50% of TypeScript repositories in our data use AngularJS for their frontend. JavaScript, C#, Go, TypeScript, and Python projects also commonly use Redis, indicating a broad popularity across languages. Interestingly, Go projects seem to use a wide variety of databases with almost equal usage of MongoDB, MySQL, and PostgreSQL.

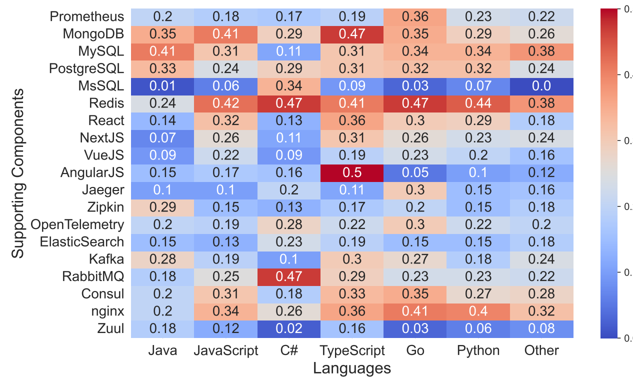

Figure 8: Supporting component usage across languages

In Figure 9, we analyze the inverse relationship. Here, each heatmap cell represents the relative usage of a language across repositories that use a particular supporting technology. The figure shows that most projects that use MsSQL also use C# as a programming language. Further, 87% of Zuul and 79% of Zipkin usage is in Java repositories. Unsurprisingly, all React, NextJS, VueJS, or AngularJS repositories contain JavaScript or TypeScript code.

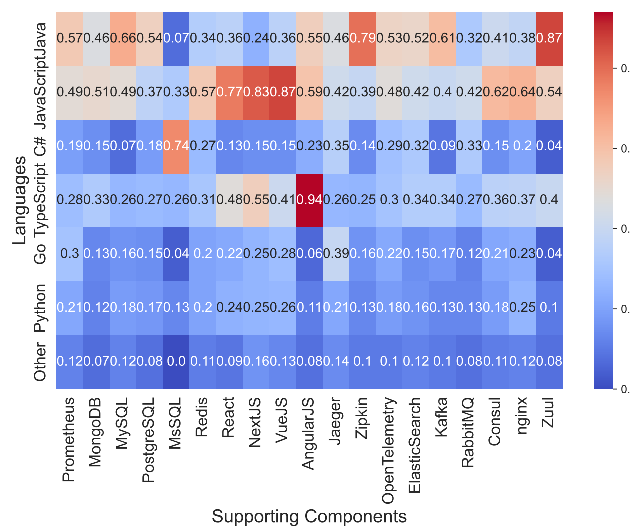

Figure 9: Language usage across supporting components

4.4 Workload Dataset

In the following, we briefly describe the datasets in the workload repositories. The `Alibaba Clusterdata` repository comprises trace datasets collected between 2017 and 2023 from the production cluster of the Alibaba Group [13]. The datasets include cluster traces containing the collocation of services and batch workloads and GPU traces containing the machine-learning (ML) workloads from Alibaba's ML-as-a-service platform. In addition, the data contains microservices and microservice architecture traces comprising runtime metrics, call dependencies, call rates, memory bandwidth contention, and more.

Shanghai AI Lab publishes the `Acme Trace` repository that encompasses job traces of two independent GPU clusters collected between March 2023 and August 2023 [15]. The dataset is about 80GB and contains more than 880, 000 jobs, of which over 470, 000 are GPU jobs.

The `Prism` repository includes anonymized communication and resource usage traces from Huawei Cloud [17]. The communication data comprises the network packet transmission records of 1019 service instances, while the resource usage data contains the resource usage of eleven metrics of these instances sampled at five-minute intervals.

The `Google Clusterdata` repository comprises three types of datasets [12]. Firstly, it includes cluster workload traces from 2009, 2011, and 2019 of workloads running on Google compute cells. For instance, the 2019 dataset focuses on the resource requests and usage of the jobs in the traces. Due to its size (about 2.4TiB compressed), the data is only available via Google BigQuery. Secondly, the repository provides traces of power utilization information from Google data centers, which partially align with the 2019 workload trace dataset. Lastly, Google publishes execution traces of its Exploratory Testing Architecture framework, which aims to explore the execution of distributed applications.

Meta publishes the `Distributed Traces` repository that provides tracing data of the architecture and workflows from their microservice architecture [18]. The architecture data includes information, such as the history of the total number of services and service instances. Also, the communication and endpoint characteristics per service are recorded. The workflow data contains call traces, which specify the call depth and number of unique services in the trace, among other things.

Microsoft Azure publishes multiple virtual machine (VM), Azure Function, and Azure large language model (LLM) inference trace datasets in their `Azure Public Dataset` repository [5]. The VM trace datasets comprise two representative traces of Azure's VM workload collected in 2017 and 2019 and one VM request trace for investigating packing algorithms. The Azure Function dataset includes traces of serverless function invocations over two weeks in 2019 and two months of serverless function blob accesses in 2020. Lastly, the LLM inference traces provide invocations of LLM inferences, including their input and output tokens collected in November 2023 and May 2024.

The Systems Infrastructure Research Lab provides trace data from Huawei Cloud's public and private serverless cloud platforms [20]. The dataset from Huawei's private serverless cloud platform contains information on 200 functions, such as resource usage and limits and the number of invocations per second. The public cloud dataset only includes the invocations per second for over 5000 functions. The newest dataset comprises cold start events and individual request traces collected from five public cloud regions.

The `SNESet` repository contains nine million records of quality of service and quality of experience metrics collected from 798 edge sites across 30 cities and three internet service providers in one country [25].

4.5 Summary

Our analysis revealed several interesting findings. For instance, our data indicates that C# is widely used for developing microservice applications, being the fourth most used language among the analyzed repositories. However, C# is largely underrepresented in microservice testbeds like TrainTicket, AcmeAir, and DeathStarBench. While most microservice benchmarks suffer from limited application sizes, the applications in our dataset are also fairly small, with 80% of applications having less than six microservices. During our manual analysis, we noticed that many applications have been developed for exemplary purposes, which might explain the low number of microservices. Overall, our dataset is a starting point for future research. For instance, future work might extend the benchmark tooling support for more applications, as the current support is very low (less than 10%).

5 Threats to Validity

Our work is based on the assumption that GitHub is a representative source for microservice applications. While it is the largest platform for open-source projects, many applications in our dataset are developed as examples to showcase certain technologies or aspects of microservice systems. Thus, our findings might not necessarily be representative of industry practices.

Furthermore, we cannot validate the accuracy of our methodology, as there is no ground truth for our data. While some features, such as the programming languages used, have rather clear answers, other aspects, such as manually counting the number of microservices, are not as clear-cut. However, we argue that our peer review process in uncertain cases mitigates the error-prone nature of manual steps to some extent. Additionally, our findings are not meant to determine an exact average number of microservices but rather to derive an approximate range.

Lastly, our keyword-based query terms might not capture every microservice application project on GitHub. Notably, TrainTicket as a forked repository is not included in our dataset. Within the scope of this work, we cannot consider every possible edge case. We intentionally decided against manually adding known repositories or artificially modifying our search and filter criteria for handling these edge cases. However, the aim of this work is not to be complete but to provide a sufficiently large amount of initial data for researchers to analyze and extend in future work.

6 Conclusion

Our work lays the foundation for increasing the understanding of the differences between microservice benchmarks and applications, which are publicly available on GitHub. While our approach specifically aims to help performance engineers find new applications for benchmark experiments, our data might also be useful for other researchers. We used the GitHub REST API to collect open-source microservice and workload dataset repositories and provided our findings as a publicly available dataset. While we provided a first analysis of our findings, future work may expand it. In conclusion, our datasets are a first step toward a source for performance engineers and other researchers that can be used as a starting point for finding suitable projects to evaluate new approaches. In the future, we envision other researchers to collectively extend the dataset by including novel application and workload traces. This would further increase the value of the collected data.

Acknowledgments

This work was mainly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 536362030 and partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 510552229.

References

[1] AcmeAir. 2013. *AcmeAir*. AcmeAir. Retrieved January 15, 2025 from https://github.com/acmeair/acmeair

[2] Işıl Karabey Aksakalli, Turgay Çelik, Ahmet Burak Can, and Bedir Tekinerdoğan. 2021. Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software* 180 (2021), 111014. https://doi.org/10.1016/j.jss.2021.111014

[3] Dario Amoroso d'Aragona, Alexander Bakhtin, Xiaozhou Li, Ruoyu Su, Lauren Adams, Ernesto Aponte, Francis Boyle, Patrick Boyle, Rachel Koerner, Joseph Lee, Fangchao Tian, Yuqing Wang, Jesse Nyyssölä, Ernesto Quevedo, Shahidur Md Rahaman, Amr S. Abdelfattah, Mika Mäntylä, Tomas Cerny, and Davide Taibi. 2024. A Dataset of Microservices-based Open-Source Projects. In *Proceedings of the 21st International Conference on Mining Software Repositories* (Lisbon, Portugal) *(Msr '24)*. Association for Computing Machinery, New York, NY, USA, 504–509. https://doi.org/10.1145/3643991.3644890

[4] Wesley KG Assunção, Jacob Krüger, Sébastien Mosser, and Sofiane Selaoui. 2023. How do microservices evolve? An empirical analysis of changes in open-source microservice repositories. *Journal of Systems and Software* 204 (2023), 111788. https://doi.org/10.1016/j.jss.2023.111788

[5] Microsoft Azure. 2024. *AzurePublicDataset*. Microsoft Azure. Retrieved January 30, 2025 from https://github.com/Azure/AzurePublicDataset

[6] Alexandru Baluta, Yar Rouf, Joydeep Mukherjee, Zhen Ming Jiang, and Marin Litoiu. 2024. Disambiguating Performance Anomalies from Workload Changes in Cloud-Native Applications. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering* (London, United Kingdom) *(Icpe '24)*. Association for Computing Machinery, New York, NY, USA, 286–297. https://doi.org/10.1145/3629526.3645046

[7] Luciano Baresi and Martin Garriga. 2020. *Microservices: The Evolution and Extinction of Web Services?* Springer International Publishing, Cham, 3–28. https://doi.org/10.1007/978-3-030-31646-4_1

[8] Luciano Baresi, Giovanni Quattrocchi, and Damian Andrew Tamburri. 2022. Microservice Architecture Practices and Experience: a Focused Look on Docker Configuration Files. arXiv:2212.03107 [cs.SE] https://arxiv.org/abs/2212.03107

[9] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the Factors That Impact the Popularity of GitHub Repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Ieee, New York, NY, 334–344. https://doi.org/10.1109/icsme.2016.31

[10] Javad Dogani, Reza Namvar, and Farshad Khunjush. 2023. Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey. *Computer Communications* 209 (2023), 120–150. https://doi.org/10.1016/j.comcom.2023.06.010

[11] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(Asplos '19)*. Association for Computing Machinery, New York, NY, USA, 3–18. https://doi.org/10.1145/3297858.3304013

[12] Google. 2019. *cluster-data*. Google. Retrieved January 30, 2025 from https://github.com/google/cluster-data

[13] Alibaba Group. 2022. *clusterdata*. Alibaba Group. Retrieved January 30, 2025 from https://github.com/alibaba/clusterdata

[14] Darby Huye, Yuri Shkuro, and Raja R. Sambasivan. 2023. Lifting the veil on Meta's microservice architecture: Analyses of topology and request workflows. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 419–432. https://www.usenix.org/conference/atc23/presentation/huye

[15] InternLM. 2023. *AcmeTrace*. Shanghai AI Lab. Retrieved January 30, 2025 from https://github.com/InternLM/AcmeTrace

[16] Ali Rezaei Nasab, Mojtaba Shahin, Seyed Ali Hoseyni Raviz, Peng Liang, Amir Mashmool, and Valentina Lenarduzzi. 2023. An empirical study of security practices for microservices systems. *Journal of Systems and Software* 198 (2023), 111563. https://doi.org/10.1016/j.jss.2022.111563

[17] Ops PAI. 2023. *Prism*. Ops PAI. Retrieved January 30, 2025 from https://github.com/OpsPAI/Prism

[18] Facebook Research. 2022. *distributed_traces*. Facebook Research. Retrieved January 30, 2025 from https://github.com/facebookresearch/distributed_traces

[19] Vishwanath Seshagiri, Darby Huye, Lan Liu, Avani Wildani, and Raja R Sambasivan. 2022. [SoK] Identifying Mismatches Between Microservice Testbeds and Industrial Perceptions of Microservices. *Journal of Systems Research* 2, 1 (March 2022), 27 pages. https://doi.org/10.5070/sr32157839

[20] SIR-Lab. 2024. *data-release*. Huawei. Retrieved January 30, 2025 from https://github.com/sir-lab/data-release

[21] Jacopo Soldani and Antonio Brogi. 2022. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *ACM Comput. Surv.* 55, 3, Article 59 (feb 2022), 39 pages. https://doi.org/10.1145/3501297

[22] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2024. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. arXiv:2408.00741 [cs.AI] https://arxiv.org/abs/2408.00741

[23] Muhammad Waseem, Peng Liang, Mojtaba Shahin, Aakash Ahmad, and Ali Rezaei Nassab. 2021. On the Nature of Issues in Five Open Source Microservices Systems: An Empirical Study. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering* (Trondheim, Norway) *(Ease '21)*. Association for Computing Machinery, New York, NY, USA, 201–210. https://doi.org/10.1145/3463274.3463337

[24] Qizhen Weng, Lingyun Yang, Yinghao Yu, Wei Wang, Xiaochuan Tang, Guodong Yang, and Liping Zhang. 2023. Beware of Fragmentation: Scheduling GPU-Sharing Workloads with Fragmentation Gradient Descent. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 995–1008. https://www.usenix.org/conference/atc23/presentation/weng

[25] YananLi18. 2023. *SNESet*. Alibaba Group. Retrieved January 30, 2025 from https://github.com/YananLi18/SNESet

[26] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chenjie Xu, Chao Ji, and Wenyun Zhao. 2018. Benchmarking microservice systems for software engineering research. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (Gothenburg, Sweden) *(Icse '18)*. Association for Computing Machinery, New York, NY, USA, 323–324. https://doi.org/10.1145/3183440.3194991