An Overview of Microservice-Based Systems Used for Evaluation in Testing and Monitoring: A Systematic Mapping Study

Stefan Fischer
Pirmin Urbanke
Rudolf Ramler
Software Competence Center Hagenberg GmbH (SCCH)
Hagenberg, Austria
{firstname.lastname}@scch.at

Monika Steidl
Monika.Steidl@uibk.ac.at
University of Innsbruck
Innsbruck, Austria

Michael Felderer
Michael.Felderer@dlr.de
German Aerospace Center (DLR), Institute for Software Technology
Cologne, Germany

ABSTRACT

Microservice-based systems have emerged as an effective architecture for countless industry applications. They provide applications as small, independent, and modular services. With the increasing interest in such systems, it is important to tackle challenges related to their quality assurance. However, to advance research in this area, systems are required to evaluate new approaches and tools. In this paper, we perform a systematic literature search for systems used in research for testing and monitoring microservice-based systems to aid future research. We provide an overview of the found studies and the systems used in their evaluation. We compose a list of publicly available systems and their characteristics, like size, available tests, and technologies used. Finally, we investigated the context in which these systems were used to provide insights in their usage and additional data that is available for them.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; *Monitors.*

KEYWORDS

microservice-based systems, mapping study, testing, monitoring

ACM Reference Format:
Stefan Fischer, Pirmin Urbanke, Rudolf Ramler, Monika Steidl, and Michael Felderer. 2024. An Overview of Microservice-Based Systems Used for Evaluation in Testing and Monitoring: A Systematic Mapping Study. In *5th ACM/IEEE International Conference on Automation of Software Test (AST 2024) (AST '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3644032.3644445

1 INTRODUCTION

Microservice-based systems enable the organization of distributed applications as a collection of possibly stateless services, to achieve scalability, separation of concerns, and maintainability [7]. Such systems are widely used by companies such as Netflix and Amazon [25]. Although microservice-based systems have many advantages over monolithic architectures, such as scalability and independent deploying and managing of services, they come with challenges that stem from the complex integration of numerous services [19].

Quality assurance of microservice-based systems must grapple with these challenges. Hence testing needs to understand the concurrent behaviors of the various microservices and interactions between them [24]. Moreover, to improve testing processes of microservice-based systems, monitoring of microservices in operation is an important capability [7]. In addition to testing, research indicates encouraging outcomes in the identification of faults within microservice-based systems through the analysis of monitoring data [18]. Therefore, we consider testing and monitoring the two most important practices for quality assurance.

Although microservice-based systems are becoming increasingly prevalent, and testing and monitoring them is considered crucial, research results in these areas are still limited. Partly, this is due to the lack of suitable and widely available microservice-based benchmark systems and data for research. Moreover, existing research in these areas is often performed separately on testing and monitoring with little overlap. However, we believe that these two fields can benefit from each other's work, and bringing testing and monitoring together can lead to more reliable systems. To support this goal, in this paper, we perform a systematic literature search for testing and monitoring of microservice-based systems. We are specifically interested in research on software testing of microservice-based systems and utilizing monitoring data to identify faults. In our work, thus, we identify microservice-based systems used for evaluation in the analyzed scientific literature. We compile a list of systems and additional datasets, valuable for testing as well as monitoring research. This outcome is intended as basis for selecting benchmark systems for future research, guided by a description of their characteristics, additional data sets, as well as the context in which they have been used before. The contributions of our work are:

- A comprehensive overview of publicly available (open-source) systems found in research related to testing and monitoring of microservice-based systems.
- An annotated list of open-source microservice-based systems suitable for evaluation, replication, and benchmarking in future testing and monitoring research.

- A taxonomy of the research performing evaluations with commonly used microservice-based systems to enable the identification of related artifacts and data-sets.

2 BACKGROUND

In this section, we discuss the necessary background for our work and the motivation behind performing this search.

2.1 Microservice-Based Systems

A microservice-based system comprises several microservices — compact, self-contained software units that collaborate with one another and can be independently upgraded and replaced. A microservice should be focused on a single responsibility and have fine granularity [3]. Additionally, a microservice should be "autonomous" in that it should (i) be largely independent from other services and (ii) be independent of technologies used [19]. A microservice runs in its own process and communicates with other microservices through lightweight protocols, like RESTful or RPC-based APIs [25]. It is important to distinguish microservice-based systems from Service-Oriented Architectures (SOA) [19]. Microservice-based systems emerged from SOA as a composition of lightweight and independent services, but they differ in terms of containing more fine-grained services, that communicate through an API layer and not through an Enterprise Service Bus (ESB). Additionally, microservices tend to provide better scalability, decoupling, and control over application implementation than SOA [24].

2.2 Testing and Monitoring of Microservice-Based Systems

Due to their complex nature and dynamic behavior microservice-based systems pose significant challenges for testing [24]. Complex deployment of numerous diverse services makes test automation challenging. The inter-communication between these services poses additional challenges, due to the complexity of many interoperable services [24]. Other challenges are due to unreliable feedback from testing frameworks or the need to manually analyze logs across numerous microservices to localize faults [24].

Many of these challenges can be addressed with monitoring and tracing solutions, which enables developer to understand the concurrent behaviors of the various microservices and interactions between them [4]. For instance, anomaly detection on system monitoring has been used to identify issues with microservice-based systems early on [20]. Moreover, monitoring data from operation may be used to improve in-house testing [7]. Finally, research has shown that field faults are inevitable, no matter how advanced the in-house testing is [8]. Therefore, monitoring the system in operation is an essential part of quality assurance.

2.3 Benchmark Systems for Evaluation

Benchmarks are useful to evaluate and compare different approaches, techniques, or tools for empirical software engineering [10]. Additionally, the use of benchmarks in research communities have shown to increase technical progress and cohesiveness in the community [17]. To this end, several benchmarks or benchmark applications have been proposed in different research communities. For instance, Arcuri et al. proposed a benchmark for testing RESTful application interfaces [1]. Artho et al. performed a literature search to identify benchmarks used to evaluate test generation techniques and reported their characteristics [2]. For research on microservice-based systems, Zhou et al. developed TrainTicket, which is a fictitious railway ticketing system consisting of 41 microservices in a layered architecture [27]. Similarly, Kistowski et al. introduced TeaStore as an artificial microservice-based reference system to perform experiments with [22]. Gan et al. proposed DeathStarBench, an open-source benchmark suite consisting of six microservice-based systems to use for analysis [6].

Wilkinson et al. introduced the FAIR Guiding Principles for researchers to enhance the reusability of their reported data [26]. FAIR stands for Findable, Accessible, Interoperable, and Reusable, which are the principles they describe in their work to make research data useful for other researchers and practitioners. Hirsch et al. applied the FAIR principles to evaluate benchmarks for debugging approaches [11]. They evaluated the principles in the following manner:

- **Findable:** based on the storage location and how permanent it is. Additionally, they considered the likelihood that it might be deleted in the future, for instance, if space in a Google drive was needed. Moreover, they checked the data that is stored if everything required to use the benchmark is contained or if they link to other sources that might be deleted or moved in the future.
- **Accessible:** was evaluated based on the public availability of the benchmark or if it is only available upon request.
- **Interoperable:** based on the file formats used in the benchmark data. A higher rating is given to standardized formats that can be easily processed for future research. Conversely, if formats require substantial manual effort to convert data into a machine-readable form a lower rating is assigned.
- **Reusable:** pertains to the documentation of the benchmark and the license used to publish or modify the benchmark.
- **Reproducible:** evaluated if the benchmark is provided in a version control system with a full version history to reproduce experiments on a specific version of the benchmark. Note: this principle was added to the FAIR Principles by Hirsch et al. in their work [11].

3 METHOD

In this section, we discuss our research aims, by means of our research questions, and describe our search and data extraction processes.

3.1 Research Questions

First, we set to review the scientific literature for references to microservice-based systems (RQ1) and, based on our findings, we identify commonly used systems suitable for experimentation (RQ2).

3.1.1 Systems used for Experimentation. Initially, our focus is on the systems documented in the literature. The goal here is to provide an overview over all the relevant systems we identified in our literature search and describe their most important characteristics. To facilitate further research, our emphasis lies in systems that are publicly available and widely utilized to provide a strong basis for the evaluation of future testing and monitoring approaches. The final goal of this set of research questions is a list of publicly available microservice-based systems, which are useful for researchers in their future endeavors, by supporting them to identify the most appropriate choice of systems to evaluate their research.

RQ1. Which systems are used in experiments for testing and monitoring of microservice-based systems?
RQ1.1 Which systems are available with an open source license?
RQ1.2 What is the distribution of publicly available systems across the primary studies?
RQ1.3 Which publicly available systems use a microservice architecture?

3.1.2 Common Microservice-Based Systems. Next, we want to filter out the systems that are most commonly used in experiments and we assess as the most relevant for future work in the research areas. For an easier selection of relevant systems for future researcher we also want to provide the system's characteristics and technologies used. To evaluate the collection of systems we identified in our search we apply the FAIR Guiding Principles, similar to [11]. This allows us to provide qualitative criteria on the systems used in literature, to enable readers to select the correct systems for their research easily. Finally, we want to provide the context in which these most relevant systems were used in and create a taxonomy of this literature. Additionally, we check these publications for links to additional data-sets that might be useful for future research, using these systems.

RQ2. Which microservice-based systems are most commonly used for experimentation?
RQ2.1. What are characteristics (e.g., size, technology, etc.) of microservice systems used for experimentation?
RQ2.2. What FAIR Guiding Principles are honored by microservice systems used for experimentation?
RQ2.3. In what context were these systems used for testing and monitoring in the respective experiments?

3.2 Overview of Literature Search and Mapping

Figure 1 shows the overall process of our research approach based on the process for performing systematic review and mapping [13]. In order to cover the two areas of testing and monitoring thoroughly and to compare the results from both areas we conduct one search process for each area. After the initial search (Step 1) and filtering (Step 2), we apply back- and forward snowballing (Step 3) and filter (Step 4) to obtain our final set of studies. We perform the snowballing steps until we find no further relevant studies. From both sets of final primary studies, we extract the systems used (Step 5), which we merge and filter (Step 6) to finally obtain a list of microservice-based systems. We provide the exact search strings and intermediate results in our online appendix[1]. The following subsections provide details on each of these steps.

[1]https://github.com/software-competence-center-hagenberg/2024-AST-Microservices-QA

3.3 Apply Search (Step 1)

Based on our research questions (Section 3.1), we defined two sets of keywords:

Testing: “software test", “software testing", “test automation", “test oracle", “test generation", “system test", “system tests", “system testing"

Monitoring: “benchmark", “fault localization", “root cause analysis", “tracing", “anomaly", “dynamic analysis", “telemetry data", “case study", “fault injection"

We combined these keywords with various spellings of the term “microservice" in order to create the final search strings. We performed the queries on well established sources for scientific literature, i.e., *Scopus*, *IEEE*, *ACM* and *Springer*. For *Scopus*, *IEEE*, *ACM* we filtered the studies based on title, abstract and keywords. For ACM we searched in the *ACM Guide to Computing Literature* which offers a larger search space than the *ACM Full-Text Collection*. As of writing the search engine of *Springer* only supports either searching in the publication titles or the full text. We choose to search in the full text at the cost of irrelevant results instead of missing out relevant studies. We conducted the search in the end of June and beginning of July 2023 without limiting the publication year, which returned a total of 349 studies for the testing search and 507 for the monitoring search.

3.4 Deduplicate & Filter Results (Step 2)

We first deduplicated the raw results based on the digital object identifier (DOI) and title, which removed 41 studies for testing and 80 for monitoring. Next, we imported the result sets into a spreadsheet solution for applying inclusion and exclusion criteria.

Inclusion Criteria. We *included* a study if the following criteria were fulfilled:

- Conference papers, journal/magazine articles
- For testing: deals with testing of microservice systems or has a relation to testing of such systems
- For monitoring: deals with or has a relation with fault localization or root cause analysis in microservice systems
- Only for snowballing: Reference publications, i.e., publications presenting a benchmark system

Exclusion Criteria. We *excluded* a study if one of the following criteria is applicable:

- Not written in English
- Conference summaries, talks, books, master doctoral thesis
- Does not have any evaluation with a system i.e., we exclude studies, which only present a concept or only use a set of data for evaluation
- For Testing: testing of SOA systems or GUI testing. Studies included by the "monitoring search"
- For Monitoring: does not have any relation with fault localization or root cause analysis in microservice systems. Studies included by the "testing search"

Note that the last inclusion criterion contradicts the third exclusion criterion. However, we argue that this exception enables us to find further studies using such benchmark systems and thereby giving a more accurate picture of the popularity of the systems. One of the authors evaluated the criteria based on title and abstract

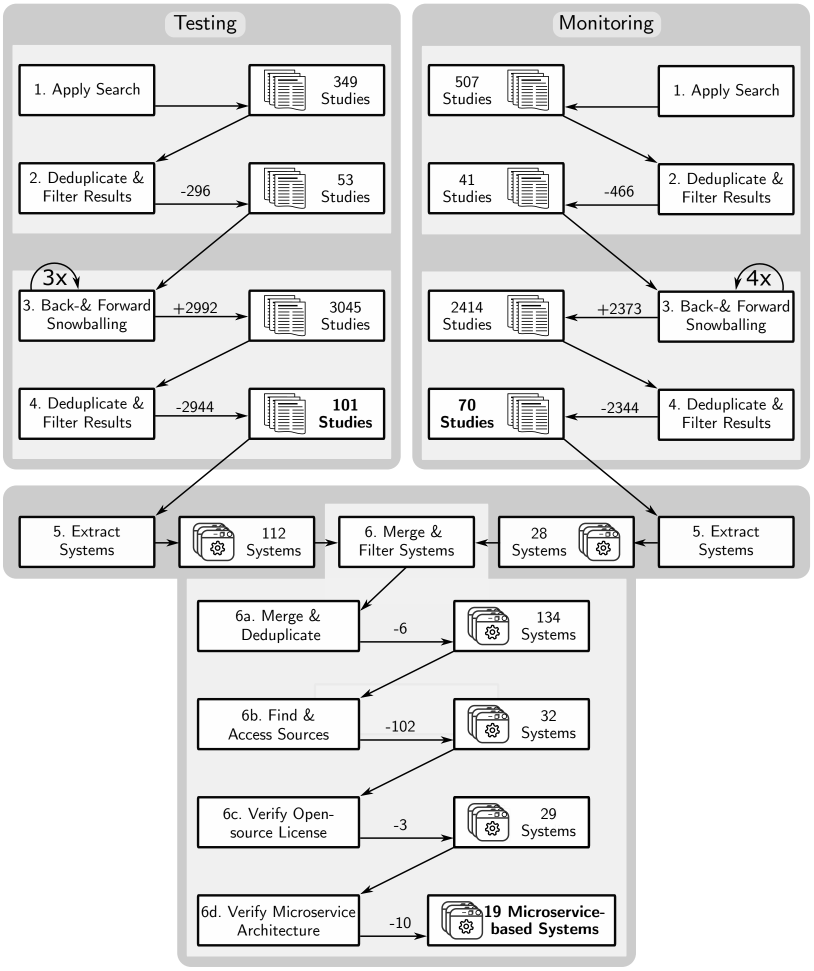
Testing
1. Apply Search
349 Studies
2. Deduplicate & Filter Results
-296
53 Studies
3x
3. Back-& Forward Snowballing
+2992
3045 Studies
4. Deduplicate & Filter Results
-2944
101 Studies
Monitoring
507 Studies
1. Apply Search
41 Studies
-466
2. Deduplicate & Filter Results
2414 Studies
+2373
3. Back-& Forward Snowballing
4x
70 Studies
-2344
4. Deduplicate & Filter Results
5. Extract Systems
112 Systems
6. Merge & Filter Systems
28 Systems
5. Extract Systems
6a. Merge & Deduplicate
-6
134 Systems
6b. Find & Access Sources
-102
32 Systems
6c. Verify Open-source License
-3
29 Systems
6d. Verify Microservice Architecture
-10
19 Microservice-based Systems

Figure 1: Overview on the search and filtering process.

of the results. When in doubt about including or excluding, the author consulted the full text of the study. This step left us with 53 publications for testing and 41 for monitoring.

3.5 Backward & Forward Snowballing (Step 3)

Since relevant literature might refer to further important studies, we used the references included in the 53 and 41 studies for backward snowballing via Scopus. During backward snowballing we only included studies published after 2011. We made this restriction by searching for "microservice" in different spellings in Scopus and found it to first appear in this year. The selected studies might also be cited by other relevant studies, hence we also performed forward snowballing, by using Scopus to find studies, which cite one of the initial studies. We applied back- and forward snowballing in multiple iterations until we did not find any more relevant studies. Overall the snowballing added total of 2992 raw studies in three iterations to the testing result and 2373 raw studies in four iteration to the monitoring result.

3.6 Deduplicate & Filter Results (Step 4)

After each round of snowballing, we deduplicated and filtered the results and only used the new relevant studies for the next round of snowballing. For the sake of brevity, we did not depict this process in detail in Figure 1. By deduplication and application of in- and exclusion criteria, we reduced the set down to 101 studies for testing and 70 studies for monitoring.

3.7 Extract Systems (Step 5)

From the full text of final sets of primary studies for testing and monitoring, we extracted the systems used in experimentation and evaluation. We extracted the names of the systems and when possible, references or URLs pointing to publications or websites of these systems. This left us with 112 systems used in the testing studies and 28 systems from the monitoring studies. These systems include proprietary systems used in industry, publicly available APIs, demonstrators, and dedicated benchmark systems. Note that some studies from the testing search test web APIs (mostly REST APIs) and mine API repositories such as www.apis.guru to perform tests on a large number of subjects. For each such study we only increase the system count by one, because they would significantly inflate and thereby distort the system count and the concrete APIs is mostly not available. Ultimately these APIs are not that relevant in this work, because we search for systems where the researchers can have complete control. We count studies using one or more systems from the Evo Master Benchmark (EMB) [1] to the usage of EMB to properly reflect the popularity of this benchmark suite.

3.8 Merge & Filter Systems (Step 6)

We merged (Step 6a) the found systems based on their names, which reduced the set of systems from 140 by 6 to 134. Then we applied the following four-tiered filter, where each layer is a Boolean question:

Findable (Step 6b): Is there a reference or link to the system in the paper? 82 of 134 systems fulfilled this criterion.
Accessible (Step 6b): Is the reference actually working such that we have access to the system and its implementation (i.e., source code)? 32 of 82 systems fulfilled this criterion.
Open-source (Step 6c): Is the system released under a license that permits usage in experimentation (i.e., open-source licenses like MIT, Apache 2.0)? 29 of 32 systems fulfilled this criterion.
Microservice-based (Step 6d): Is the system described as a microservice system? 19 of 29 systems fulfilled this criterion.

For the 19 microservice systems we gather information from the source code repositories on the characteristics of the system and the used technologies. In the following sections we use both the results from the OSS and the Microservice filter for further analysis.

3.9 Data Extraction

To answer *RQ2.3* we performed a detailed screening of the literature that contained experiments using the most relevant microservice-based systems. We extracted the research aims and system data used in the studies to create two taxonomies in an iterative process. Moreover, during the screening of the studies we looked for links to replication packages or other supplementary data for the used microservice-based systems.

4 RESULTS

In this section, we discuss the results of our literature search and the extracted data.

4.1 Found Primary Studies

First, we present an overview of the identified studies in our search results. Figure 2 depicts the number of studies per year they have been published. The first paper in our results was published in 2011 and focused on testing of web services. Studies with the focus on monitoring of microservice-based systems appear first in 2018 in our results, which appears to be the year that started a general uptick in research in testing and monitoring of such systems. Note that the numbers for 2023 are not complete, due to the search being performed in that year.

4.2 Found Systems

We identified 134 systems used in primary studies for experiments and evaluation. Out of these, references and links to find the actual system were provided in 82 cases and 29 of them are available under

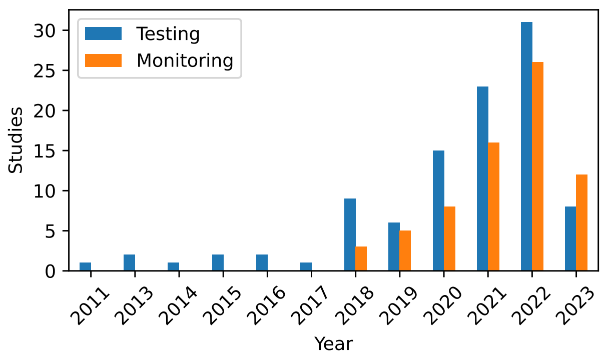

Figure 2: Amount of primary studies per year.

an open source license. Table 1 lists these 29 systems along with the number of publications in testing and monitoring, the years when they were used, and their classification as a microservice-based systems. Moreover, we show the last commit, the forks and stars of the GitHub repository (as of July 2023), and the license under which the systems are published. In the table, the names of the systems are linked to the source repositories. The most frequently used systems (top three) highlighted for testing (blue) and monitoring (orange).

We were not able to report forks and stars for the system *ElasticPress*, which is not hosted on GitHub, and for systems that are distributed over multiple repositories (*Corona Warn App* and *OpenMRS*) or which are part of a larger repository (*Book Info*).

5 DISCUSSION

In this section, we discuss our findings in relation to our initial goals and answer our research questions.

5.1 RQ1: Systems used for Experimentation

We checked all identified systems in the relevant studies if they are publicly available under an open source license.

Answering RQ1.1 Open source systems: Our search resulted in 29 open source systems that have been used in the relevant literature for experiments. These systems are listed in Table 1.

We analyzed the frequency and consistency over the years that a system is used in research studies, as well as the research area. Figure 3 shows the distribution of systems over the years. *Train Ticket* and *Sock Shop* are the most common systems, both consistently used in studies over the last few years. They are mostly used in studies about monitoring, but some studies also use them in testing research. The third most common systems is *EMB*, the EvoMaster Benchmark, which is exclusively used in testing research. *Tea Store* and *Hipster Shop* are also used in both research areas, even though in fewer studies. Finally, *DeathStarBench* has been used six times, but only in monitoring research. The remaining systems have been used less frequently, with 20 out of the 29 systems only appearing in one or two studies.

Table 1: Available systems used in evaluations (as of July 2023)

System	Usage in			Years	Micro-	Last	Forks	Star	License
	T	M	SUM	of use	Service	Commit			
AWS demo	1	-	1	2020	✓	2022-03	175	413	MIT
Book Info	1	1	2	2021	✓	2023-05	-	-	Apache 2.0
ChaosEcho	-	1	1	2021		2021-11	0	1	MIT
CloudStore	-	1	1	2022		2016-08	3	8	EPL 1.0
Corona Warn App	1	-	1	2022	✓	2023-07	-	-	Apache 2.0
customers and orders	1	-	1	2020	✓	2023-05	224	464	Apache 2.0
DeathStarBench [6]	-	6	6	2021-2022	✓	2023-06	324	555	Apache 2.0
ElasticPress	1	-	1	2016		2023-07	-	-	GPLv2
Elgg	-	1	1	2019		2023-07	686	1600	MIT
EMB [1]	16	-	16	2017-2023		2023-04	12	18	Apache 2.0
FTGO	1	-	1	2022	✓	2022-09	1200	3100	Apache 2.0
Full Teaching	2	-	2	2021-2023		2020-04	27	29	Apache 2.0
Hipster Shop	1	5	6	2021-2023	✓	2023-05	20	37	Apache 2.0
Jenkins	1	-	1	2020		2023-07	8200	21200	MIT
OnlineBoutique	-	2	2	2021-2022	✓	2023-07	5200	14500	Apache 2.0
OpenMRS	-	1	1	2022		2023-07	-	-	MPL 2.0
Petstore	3	-	3	2021-2023		2023-03	262	172	Apache 2.0
Piggy Metrics	2	1	3	2020-2023	✓	2021-11	5900	12500	Apache 2.0
Prestashop	1	-	1	2022		2023-07	4600	7300	OSL-3.0
PyMicro	-	4	4	2018-2023	✓	2015-11	8	34	MIT
restaurant management	1	-	1	2020	✓	2017-01	107	206	Apache 2.0
Rideshare	2	-	2	2019-2022	✓	2019-04	0	1	MIT
RobotShop	-	1	1	2021	✓	2023-03	1100	708	Apache 2.0
Sock Shop	4	20	24	2018-2023	✓	2021-08	2600	3500	Apache 2.0
SockPong	-	1	1	2021	✓	2021-07	0	0	MIT
T2	-	1	1	2022	✓	2022-09	0	0	Apache 2.0
Tea Store [22]	4	3	7	2019-2022	✓	2022-08	116	101	Apache 2.0
Train Ticket [27]	8	20	28	2018-2023	✓	2022-11	256	567	Apache 2.0
Transaction service	1	-	1	2020	✓	2017-01	957	3000	Apache 2.0

Answering RQ1.2 Distribution of systems: We observed varying usage of systems across studies and years. Some systems were frequently utilized over several years, while most (20 out of 29) were used only in one or two years with limited study representation. Moreover, 6 systems have been used in both research areas of testing and monitoring, while others are used only in testing (13 systems) or only in monitoring (10 systems).

We further analyzed the pool of systems for those that use a microservice architecture. We found 19 microservice-based systems.

Answering RQ1.3 Microservice-based systems: We identified 19 microservice-based systems released under an open source license. These systems are marked with a check-mark in Table 1.

5.2 RQ2: Common Microservice-Based Systems

To compile a list of most commonly used systems for research in testing and monitoring, we filtered for systems that appear in at least two studies and consist of more than four services. The resulting list of 9 system is described below:

- **Train Ticket:** is a dedicated benchmark system for research on microservice-based systems. It is a platform for booking train tickets and provides extensive documentation with a dedicated Wiki. Additionally, it provides 22 optional faulty services that can be used for research.
- **Sock Shop:** is a web shop for demonstration and testing of microservice-based systems. Implementations of services are individual Github repositories with one repository dedicated to documenting the complete system.
- **Tea Store:** is a benchmark system in the form of a web shop for buying tea. It was created for the purpose of testing and benchmarking for microservice research.
- **DeathStarBench:** is a collection of microservice applications with various sizes, developed for use in research. Currently three systems are released, with three more being reported as *in progress.*
- **Hipstershop:** is a fork of the web shop *OnlineBoutique* for demonstrating monitoring and tracing with OpenCensus, Prometheus and Jaeger.
- **PyMicro:** is a very rudimentary microservice-based system implemented in Python. It can be configured in a single file and the topology and number of services can be adapted easily.
- **Piggy Metrics:** is a financial advisor application for demonstrating microservice architecture. The project was intended as a tutorial for a microservice-based system using Spring technology.

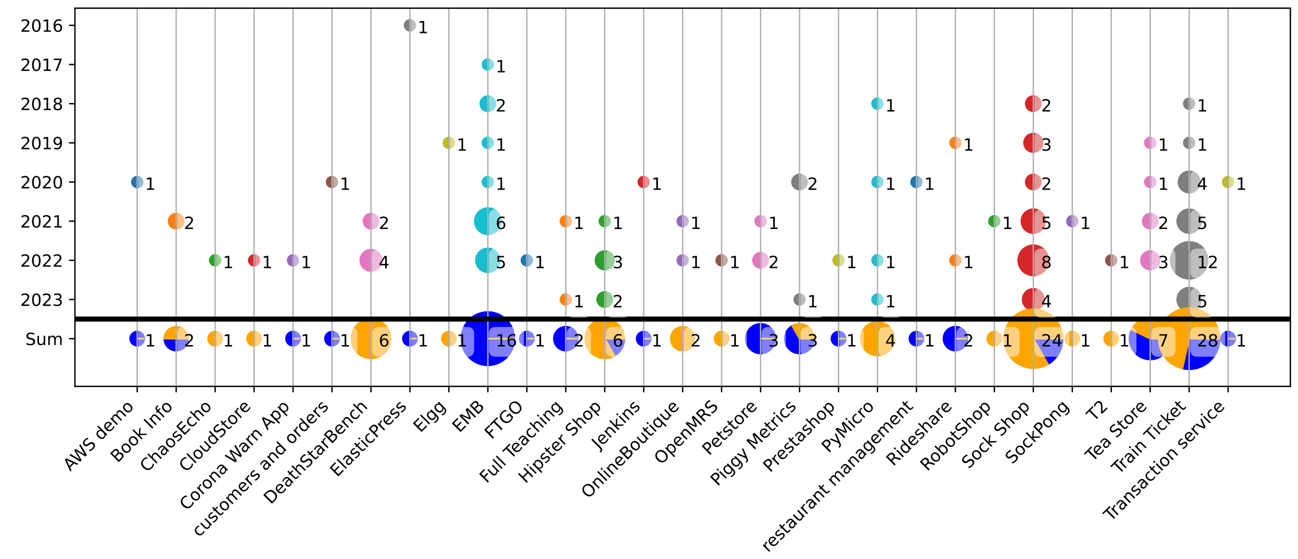

Figure 3: Identified open source systems and their usages over years. The bottom row shows the sum of usages, the segments of the pies show the distribution of usages between the testing (blue) and monitoring (orange) studies.

- **OnlineBoutique:** is a microservice demonstration application in form of a web shop. Beside the use in research, this system is used by Google to demonstrate cloud related technologies.
- **Rideshare:** is a ride-sharing platform developed for comparing testing tools. Implementation and documentation is spread across different repositories and publications by its' authors.

Table 2 provides further details about each of these nine most commonly used systems (in case of *DeathStarBench*, details are given for each of the three released systems it contains). We show the number of services reported for each system. In case this number was not provided in the related documentation or publications, we analyzed the related repositories (*Sock Shop*, *Hotel Reservation*, *Piggy Metrics*, and *Rideshare*). We depict the main development languages, contained testing and monitoring technologies, the provided container deployment technologies, communication techniques between services, assessments of their documentation and if their repository provides version tags.

Answering RQ2.1 System characteristics and technologies: Our search identified systems of different size (from 5 to 41 microservices), using a variety of languages (e.g., Java, JS, Go) and related testing and monitoring technologies (see Table 2).

To further evaluate the potential for reusability of the systems, we adopted the FAIR Guiding Principles similar to Hirsch et al. [11]. We base our following evaluation on Table 1 and 2.

- **Findable:** The top nine commonly used microservice-based systems are hosted on GitHub, lacking a globally unique and persistent identifier. However, Hirsch et al. [11] assume that the storage remains secure and accessible in the foreseeable future. We assess if the source code is complete and self-contained. We found that nearly all systems contain all source files in one repository, whereas *Rideshare* and *Sock Shop* only provide deployment configurations in their repository, and the code is spread over several repositories.
- **Accessible:** All nine systems are hosted on GitHub and publicly accessible, allowing direct access to the source code.
- **Interoperable:** Hirsch et al. [11] investigated the used data formats. In analogy, we investigated containerization methods for system deployment, ensuring compatibility across environments. Seven systems employ either docker or docker-compose. Additionally, six systems opt for Kubernetes, a robust orchestration platform.
- **Reusable:** Regarding documentation, we verified that all systems provide a README file containing set-up information. *Train Ticket* and *Sock Shop* additionally provide a wiki with detailed information. Most systems include a microservice description covering at least the architecture (*Tea Store*, *Social Network*, *Hipstershop*, *PyMicro*, *OnlineBoutique*). For systems using REST communication we found an API documentation in their repositories except for *Tea Store*. Documentation regarding monitoring is mostly related to information on how to access tracing and infrastructure monitoring data, with the exception of *PyMicro*, which omits monitoring information. *Train Ticket*, *Sock Shop*, and *Tea Store* further provide information about their testing approach.
All nine systems are published under open source licenses (see Table 1); seven systems use the Apache 2.0 license and two (*PyMicro* and *Rideshare*) use the MIT license.
- **Reproducible:** While each of the nine systems offers version control through their GitHub commits, determining the

Table 2: The most commonly used microservice-based systems in literature

		Technologies																																
		Language						Testing								Monitoring							Container					Co.		Docs.				
System	**Reported Services**	Java	JS	Python	C#	C/C++	Go	Load Gen.	JUnit / xUnit	Python Tests	Go Tests	Mocha	Cypress	Karma	Protractor	Jaeger	Prometheus	Zipkin	OpenTelemetry	Elastic	Kieker	Stackdriver	docker	docker-compose	kubernetes	skaffold	OpenShift	REST	RPC	Testing	Monitoring	Microservices	Wiki	**Version Tags**
Train Ticket	41	■	■						■	■						■	■	■		■				■	■			■		■	■	■	■	■
Sock Shop	9	■	■				■	■	■			■				■	■							■	■			■		■	■	■	■	■
Tea Store	5	■						■	■				■						■		■		■	■	■			■		■	■	■		■
DeathStarBench																																		
Social Network	36					■				■						■								■			■		■		■	■		■
Media Service	38					■				■						■								■			■		■		■	■		
Hotel Reservation	18						■			■						■								■	■		■		■		■			■
Hipstershop	10	■	■	■	■		■	■			■					■	■		■			■			■	■			■		■	■		
PyMicro	16			■																■			■						■			■		
Piggy Metrics	8	■	■						■											■				■				■			■	■		■
OnlineBoutique	11	■	■	■	■		■	■	■		■											■			■	■			■		■	■		■
Rideshare	13	■	■				■		■		■			■	■					■				■	■			■			■			

precise version for referencing can be unclear. Specific systems utilize version tags or releases to address this, ensuring a clear indication of the intended version. The systems that do not provide version tags or releases are *Media Service*, *HipsterShop*, *PyMicro*, *Rideshare*.

Answering RQ2.2 FAIR Guiding Principles: All systems honor the FAIR Guiding Principles [11], with some shortcomings related to GitHub hosting and versioning, as well as documentation.

Finally, we looked at the studies that use these systems to determine in what context they were used in and to identify some additional data-sets. The first thing that became clear when doing this was that the systems have been used in more monitoring research than testing research. Of the 73 paper that use at least one of the eleven systems, 18 are from the testing search and 55 are from the monitoring search results.

Table 3 shows the taxonomy of testing research the identified systems have been used in. We group the research into their research methods, by solution proposals (i.e., research proposing novel solutions for problems) and validation research (i.e., performing experiments to validate existing research or tool, or investigating existing issues) [13]. The most papers we identified here, propose solutions for test case generation and we therefore split them up into smaller groups to better differentiate them. We distinguish generation from field data (i.e., ex-vivo testing) or with combinatorial approaches and generation for reliability or performance testing. The first thing we notice by doing this is that the top three systems in our list have been used in various different research areas within testing. On the other hand, some systems have only been used in some specific use cases. For instance, *Rideshare* has been used only in two studies comparing testing tools, by the same group of researchers [T19], [T33]. Similarly, *Piggy Metrics* was only used in two studies by the same researches, proposing and approach to generate test cases from field data [T37], [T43].

We show our taxonomy of the monitoring research using the systems in Table 4. Similar to the testing taxonomy, we grouped the research into their research methods, but found that all but

Table 3: Taxonomy of system usage for testing

	Research Goals	**Studies**	**Systems**
solution	test case generation		
	ex-vivo testing	[T37], [T39], [T43]	Train Ticket, Piggy Metrics
	reliability testing	[T7], [T39], [T353], [T549]	Train Ticket, Hipstershop
	performance testing	[T7], [T256]	Train Ticket, Sock Shop
	combinatorial	[T45], [T402]	Train Ticket, Sock Shop
	test case selection	[T4], [T30]	Train Ticket, Tea Store
	debugging	[T1625]	Train Ticket
validation	comparing tools	[T19], [T33], [T193]	Sock Shop, Rideshare
	fault tolerance	[T65], [T74]	Sock Shop, Tea Store
	performance testing	[T85]	Tea Store

one paper are proposing a solution. The only one that we consider validation research is comparing different approaches for anomaly detection in monitoring data and defining requirements for such approaches from their results [M10]. The remaining studies propose solutions to identify faults in microservice-based systems. The majority of these proposed approaches focus on detecting anomalies in monitoring data and identifying the root-causes of these anomalies and faults. To accomplish this, different types of monitoring data are analyzed. Commonly metrics (i.e., performance and communication metrics) are monitored as a data source for these approaches. Other approaches use topology information, logs, or trace data to identify faults and their root-causes. Therefore, we classified the research in the types of faults that they aim to identify in our taxonomy in Table 4. The fault types we identified are:

- **Performance:** abnormal increases in service response times, typically resulting from resource anomalies (e.g., CPU, memory, disk, network).
- **Availability:** failed service invocations typically caused by defects in the service or anomalies in the operating environment.
- **Bugs:** caused by faulty requests (e.g., 4XX errors), or errors in the business logic like incorrect response values.
- **Communication:** a significant increase in the number of service requests or an increase in service packet loss, which is typically caused by network anomalies between services.

Table 4: Taxonomy of system usage for monitoring

	Research Goals	**Studies**	**Systems**
solution	**Performance**	[M7], [M591], [M640], [M670], [M1559]	Tea Store, DeathStarBench
	|-*CPU hog*		
	| |–Toolkit	[M8], [M11], [M37], [M46], [M49], [M52], [M53], [M55], [M56], [M62], [M66], [M93], [M102], [M109], [M405], [M478], [M498], [M541], [M592], [M626], [M679], [M731], [M1319]	Train Ticket, Sock Shop, DeathStarBench, Hipstershop, OnlineBoutique
	| |–change sources	[M80], [M1320]	Train Ticket, Sock Shop, Tea Store
	|-*memory usage*	[M15], [M485], [M487], [M560]	Train Ticket, Sock Shop, Hipstershop
	| |–Toolkit	[M8], [M11], [M49], [M55], [M66], [M102], [M109], [M478], [M498], [M541], [M592], [M626], [M731], [M1319]	Train Ticket, Sock Shop, DeathStarBench
	| |-change sources	[M53], [M80]	Train Ticket, Sock Shop, Social Network (DSB)
	|-*disk usage*	[M15], [M485]	Sock Shop, Hipstershop
	| |–Toolkit	[M46], [M109], [M498], [M679]	Train Ticket, DeathStarBench, Hipstershop
	| |–change sources	[M1320]	Tea Store
	|-*network response delay*	[M5], [M15], [M57], [M405], [M450], [M485], [M487], [M560], [M582]	Train Ticket, Sock Shop, Hipstershop, PyMicro, OnlineBoutique
	| |–Toolkit	[M11], [M46], [M52], [M56], [M62], [M66], [M93], [M102], [M109], [M478], [M498], [M541], [M592], [M679]	Train Ticket, Sock Shop, DeathStarBench, Hipstershop, OnlineBoutique
	| |–change data records	[M584]	Train Ticket
	| |–change sources	[M464], [M785], [M1320]	Train Ticket, Sock Shop, Tea Store
	|-*DB delay*	[M582]	OnlineBoutique
	Availability	[M15], [M17]	Train Ticket, Sock Shop
	|-*service*	[M53], [M57], [M487], [M560], [M582]	Train Ticket, Sock Shop, Social Network (DSB), OnlineBoutique
	| |–Toolkit	[M8], [M14], [M62], [M112], [M592], [M626]	Train Ticket, Sock Shop
	| |–container shutdown/shutdown	[M52], [M405], [M760]	Sock Shop, PyMicro, OnlineBoutique
	| |–change data records	[M584]	Train Ticket
	|-*resource*		
	| |–permission change	[M785]	Piggy Metrics
	| |–change sources	[M14], [M80]	Train Ticket, Sock Shop
	Bugs	[M15], [M17], [M485]	Train Ticket, Sock Shop, Hipstershop
	|-*request error*		
	| |–change sources	[M14], [M626]	Train Ticket
	|-*business logic errors*	[M560]	
	| |–Toolkit	[M56]	Train Ticket
	| |–change sources	[M498], [M626]	Train Ticket
	Communication		
	|-*package loss*	[M8], [M15], [M57], [M485], [M487], [M592], [M626], [M707], [M785]	Train Ticket, Sock Shop, Hipstershop, Piggy Metrics
	|-*call frequency*	[M8], [M53], [M450], [M560]	Sock Shop, Social Network (DSB)
	Operations		
	|-*swap or add calls*	[M584]	Train Ticket
	Architecture	[M12], [M29]	Train Ticket
	Unclear	[M3], [M21], [M74], [M84], [M620], [M1173]	Sock Shop, Hipstershop, PyMicro
v	**Comparison of approaches**	[M10]	Train Ticket

- **Operations:** a change in operation sequences like swapping call orders or additional calls between services.
- **Architecture:** anti patterns in the system that reveal architectural smells.

We further distinguished by the method used to inject faults in the microservice-based systems in the research evaluation. Most commonly the research uses Chaos Engineering Toolkits (like *ChaosBlad*[2] or *Chaos Mesh*[3]), stress testing tools (like *stress-ng*[4]), or network emulation tools (like *tc-netem*[5]) to introduce anomalies in the recorded data from monitoring. Other experiments introduce faults directly into the system's source code or change configurations to lead to anomalies in the data records.

Finally, we checked the papers for links to additional data-sets. Many of these links contained source code of the approaches or for evaluation, additional result data, or plots the authors could not fit into the publication. However, some papers link to replication packages containing data we considered potentially useful for other research. We list these data-sets in Table 5. We identified trace data for *Train Ticket*, test scenarios used to generate a load on systems, faults that were injected into systems to evaluate approaches, and other monitoring data like performance and access logs.

Answering RQ2.3 Usage context: We identified a variety of research using the systems to identify faults in them, either by testing (Table 3) or monitoring (Table 4). Some systems (esp. *Train Ticket*) were used in a wide variety of contexts and studies. From our taxonomy, we were able to identify data-sets (e.g., traces, faults) useful for future research (see Table 5).

5.3 Threats to Validity

We face similar threats to validity as other systematic mapping studies. The first threat to validity is the completeness of the list of studies and systems. To address this, we searched for literature in the most well-known digital libraries to get a representative pool of studies. Moreover, we selected our search terms by checking related

[2]https://github.com/chaosblade-io/chaosblade
[3]https://github.com/chaos-mesh/chaos-mesh
[4]https://aur.archlinux.org/packages/stress-ng
[5]https://man7.org/linux/man-pages/man8/tc-netem.8.html

Table 5: Additional data-sets for our list of systems

Study	Data	Link
Train Ticket		
[M10]	trace data	https://zenodo.org/records/6979726
[M46]	trace data	https://github.com/NetManAIOps/TraceAnomaly
[M56]	trace data	https://github.com/NetManAIOps/TraceRCA
[M464]	trace data	https://github.com/SEALABQualityGroup/replication_delag
[M626]	trace data	https://fudanselab.github.io/PUTraceAD
[T43]	test scenarios	https://gitlab.com/learnERC/exvivomicrotest
[M592]	performance data, injected faults	https://zenodo.org/records/6955909
Sock Shop		
[M478]	performance data	https://github.com/AXinx/CausalRCA_code
[M1319]	performance data	https://github.com/azamikram/rcd
Tea Store		
[M1320]	access logs	https://doi.org/10.5281/zenodo.5659008
Hipstershop		
[T353]	injected faults	https://github.com/rkarn/automated-testing-resciliency
Piggy Metrics		
[T43]	test scenarios	https://gitlab.com/learnERC/exvivomicrotest
OnlineBoutique		
[M582]	injected faults, topology, service calls	https://github.com/IntelligentDDS/GIED

literature [4, 24, 25] and refined them by checking the search results for relevant papers. A second threat to validity is the selection of research questions. To minimize this threat we had several discussions about the questions and goals of our search. We argue the research questions reflect the goals of our work. A third threat to validity is related to the extraction of data from the studies and for the systems. During data extraction we recorded the data in spreadsheets and when necessary studies were reread to clarify some doubt about the data. We had many meetings and discussions about the extraction of the data. The fourth threat to validity is the evaluation of the systems using the FAIR evaluation criteria. This evaluation is based on subjective judgement and some of the rating criteria might be outdated in a few years as best practices and technology continue to evolve. To minimize this, we used the same FAIR evaluation criteria as in related research [11].

6 RELATED WORK

Several mapping studies have been performed on research for microservice-based systems [5, 9, 12, 23]. Most of the publications found in these studies focuses on the architecture and design of such systems [12, 23]. As a result of these research interests, additional mapping studies have been published, centering their attention on the architecture of microservice-based systems [21, 25]. In contrast to our work, these studies did not focus on the systems used for experiments, but rather to summarize existing research and identify trends and gaps in existing research.

For research specific to testing microservice-based systems, Waseem et al. performed a systematic mapping study [24]. Their pool of primary sources overlaps with the studies we identified in our search, but their study was performed in 2019 and therefore does not contain the newer publications we identified. Moreover, they did not focus on the systems used in the evaluations of their identified studies. Additionally, a mapping study about testing microservice-based systems by Panahandeh et al. is currently available as a pre-print [14]. They compiled a list of research on testing microservice-based systems along with a taxonomy further breaking down the actual research performed. One of the research questions in this study also pertained to the systems used in the evaluation of the publications. We differ from their work in the identified systems and the analysis performed on the available technologies and documentation.

Rahman et al. compiled microservice-based systems from GitHub [15], contrasting with our systematic search in digital libraries. Despite some overlap between their dataset and our findings, differences in search methods yield divergent results, allowing their data to complement our search results.

Silva et al. [16] performed a literature search of faults in microservice systems. They created a taxonomy classifying 117 different fault types, grouped into different subcategories. These categories are similar to the fault types we identified in our taxonomy over the monitoring research. However, the faults identified by Silva et al. are more detailed, which was out of the scope of our work.

Hirsch et al. conducted a systematic review on debugging benchmarks, comparing their size and data availability [11]. They also assessed these benchmarks using FAIR principles, akin to our study. Unlike our work, which sought research proposing and evaluating solutions for quality issues in microservice-based systems, they specifically searched for publications for benchmarks or datasets.

7 CONCLUSIONS

We performed a systematic literature search for systems used in research for testing and monitoring of microservice-based systems. The goal of our work was to provide a list of publicly available microservice-based systems that can be used for experimentation in future research in both fields, testing and monitoring.

A large number of systems have been used in research over the years. In total, we found 134 systems mentioned in the literature; 112 systems in the testing studies and 28 systems in monitoring studies, with 6 of them related to both fields. However, the vast majority of these systems are proprietary or lack relevant information how to get access for using them in research. In our review we were able to identify only 29 systems (22%) that provide access to the source code released under an open-source license that permits using them freely in experiments. Out of these, only 19 systems (14%) are actually based on microservices.

From the pool of microservice-based systems suitable for research we selected the nine most commonly used ones and describe them in detail to support their selection for experiments in future research. Additionally, we mapped these systems to existing research on testing and monitoring to highlight previous results as well as available data sets for benchmarking.

ACKNOWLEDGMENTS

This work has been funded by the Austrian Research Promotion Agency FFG (FFG grant no. 888127), as well as the Austrian ministries BMK and BMAW, and the State of Upper Austria in the frame of the SCCH competence center INTEGRATE (FFG grant no. 892418) part of the FFG COMET Competence Centers for Excellent Technologies Programme.

DATA AVAILABILITY STATEMENT

The used search queries and results of our search process are openly available in the GitHub repository at https://github.com/software-competence-center-hagenberg/2024-AST-Microservices-QA.

REFERENCES

[1] Andrea Arcuri, Man Zhang, Amid Golmohammadi, Asma Belhadi, Juan P. Galeotti, Bogdan Marculescu, and Susruthan Seran. 2023. EMB: A Curated Corpus of Web/Enterprise Applications And Library Support for Software Testing Research. In *IEEE Conference on Software Testing, Verification and Validation, ICST 2023, Dublin, Ireland, April 16-20, 2023*. IEEE, 433–442. https://doi.org/10.1109/ICST57152.2023.00047

[2] Cyrille Artho, Adam Benali, and Rudolf Ramler. 2021. Test Benchmarks: Which One Now and in Future?. In *21st IEEE International Conference on Software Quality, Reliability and Security, QRS 2021, Hainan, China, December 6-10, 2021*. IEEE, 328–336. https://doi.org/10.1109/QRS54544.2021.00044

[3] Sasa Baskarada, Vivian Nguyen, and Andy Koronios. 2020. Architecting Microservices: Practical Opportunities and Challenges. *J. Comput. Inf. Syst.* 60, 5 (2020), 428–436. https://doi.org/10.1080/08874417.2018.1520056

[4] Antonia Bertolino, Guglielmo De Angelis, Micael Gallego, Boni García, Francisco Gortázar, Francesca Lonetti, and Eda Marchetti. 2019. A Systematic Review on Cloud Testing. *ACM Comput. Surv.* 52, 5 (2019), 93:1–93:42. https://doi.org/10.1145/3331447

[5] Roberta Capuano and Henry Muccini. 2022. A Systematic Literature Review on Migration to Microservices: a Quality Attributes perspective. In *IEEE 19th International Conference on Software Architecture Companion, ICSA Companion 2022, Honolulu, HI, USA, March 12-15, 2022*. IEEE, 120–123. https://doi.org/10.1109/ICSA-C54293.2022.00030

[6] 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, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*. ACM, 3–18. https://doi.org/10.1145/3297858.3304013

[7] Luca Gazzola, Maayan Goldstein, Leonardo Mariani, Marco Mobilio, Itai Segall, Alessandro Tundo, and Luca Ussi. 2023. ExVivoMicroTest: ExVivo Testing of Microservices. *J. Softw. Evol. Process.* 35, 4 (2023). https://doi.org/10.1002/smr.2452

[8] Luca Gazzola, Leonardo Mariani, Fabrizio Pastore, and Mauro Pezzè. 2017. An Exploratory Study of Field Failures. In *28th IEEE International Symposium on Software Reliability Engineering, ISSRE 2017, Toulouse, France, October 23-26, 2017*. IEEE Computer Society, 67–77. https://doi.org/10.1109/ISSRE.2017.10

[9] Sara Hassan, Rami Bahsoon, and Rick Kazman. 2020. Microservice transition and its granularity problem: A systematic mapping study. *Softw. Pract. Exp.* 50, 9 (2020), 1651–1681. https://doi.org/10.1002/spe.2869

[10] Wilhelm Hasselbring. 2021. Benchmarking as Empirical Standard in Software Engineering Research. In *EASE 2021: Evaluation and Assessment in Software Engineering, Trondheim, Norway, June 21-24, 2021*. ACM, 365–372. https://doi.org/10.1145/3463274.3463361

[11] Thomas Hirsch and Birgit Hofer. 2022. A systematic literature review on benchmarks for evaluating debugging approaches. *J. Syst. Softw.* 192 (2022), 111423. https://doi.org/10.1016/j.jss.2022.111423

[12] Claus Pahl and Pooyan Jamshidi. 2016. Microservices: A Systematic Mapping Study. In *CLOSER 2016 - Proceedings of the 6th International Conference on Cloud Computing and Services Science, Volume 1, Rome, Italy, April 23-25, 2016*. SciTePress, 137–146. https://doi.org/10.5220/0005785501370146

[13] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

[14] Mohammad Imranur Rahman and James Miller. 2023. A Systematic Review on Microservice Testing. *PREPRINT (Version 1) available at Research Square* (2023). https://doi.org/10.21203/rs.3.rs-3158138/v1

[15] Mohammad Imranur Rahman, Sebastiano Panichella, and Davide Taibi. 2019. A curated Dataset of Microservices-Based Systems. *CoRR* abs/1909.03249 (2019). arXiv:1909.03249 http://arxiv.org/abs/1909.03249

[16] Francisco Silva, Valéria Lelli, Ismayle de Sousa Santos, and Rossana M. de Castro Andrade. 2022. Towards a Fault Taxonomy for Microservices-Based Applications. In *SBES 2022: XXXVI Brazilian Symposium on Software Engineering, Virtual Event Brazil, October 5 - 7, 2022*, Marcelo de Almeida Maia, Fabiano A. Dorça, Rafael Dias Araújo, Christina von Flach, Elisa Yumi Nakagawa, and Edna Dias Canedo (Eds.). ACM, 247–256. https://doi.org/10.1145/3555228.3555245

[17] Susan Elliott Sim, Steve M. Easterbrook, and Richard C. Holt. 2003. Using Benchmarking to Advance Research: A Challenge to Software Engineering. In *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*. IEEE Computer Society, 74–83. https://doi.org/10.1109/ICSE.2003.1201189

[18] Jacopo Soldani and Antonio Brogi. 2023. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *ACM Comput. Surv.* 55, 3 (2023), 59:1–59:39. https://doi.org/10.1145/3501297

[19] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan van den Heuvel. 2018. The pains and gains of microservices: A Systematic grey literature review. *J. Syst. Softw.* 146 (2018), 215–232. https://doi.org/10.1016/j.jss.2018.09.082

[20] Monika Steidl, Marko Gattringer, Michael Felderer, Rudolf Ramler, and Mostafa Shahriari. 2022. Requirements for Anomaly Detection Techniques for Microservices. In *Product-Focused Software Process Improvement - 23rd International Conference, PROFES 2022, Jyväskylä, Finland, November 21-23, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13709)*. Springer, 37–52. https://doi.org/10.1007/978-3-031-21388-5_3

[21] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2018. Architectural Patterns for Microservices: A Systematic Mapping Study. In *Proceedings of the 8th International Conference on Cloud Computing and Services Science, CLOSER 2018, Funchal, Madeira, Portugal, March 19-21, 2018*. SciTePress, 221–232. https://doi.org/10.5220/0006798302210232

[22] Jóakim von Kistowski, Simon Eismann, Norbert Schmitt, André Bauer, Johannes Grohmann, and Samuel Kounev. 2018. TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In *26th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2018, Milwaukee, WI, USA, September 25-28, 2018*. IEEE Computer Society, 223–236. https://doi.org/10.1109/MASCOTS.2018.00030

[23] Hulya Vural, Murat Koyuncu, and Sinem Guney. 2017. A Systematic Literature Review on Microservices. In *Computational Science and Its Applications - ICCSA 2017 - 17th International Conference, Trieste, Italy, July 3-6, 2017, Proceedings, Part VI (Lecture Notes in Computer Science, Vol. 10409)*. Springer, 203–217. https://doi.org/10.1007/978-3-319-62407-5_14

[24] Muhammad Waseem, Peng Liang, Gastón Márquez, and Amleto Di Salle. 2020. Testing Microservices Architecture-Based Applications: A Systematic Mapping Study. In *27th Asia-Pacific Software Engineering Conference, APSEC 2020, Singapore, December 1-4, 2020*. IEEE, 119–128. https://doi.org/10.1109/APSEC51365.2020.00020

[25] Muhammad Waseem, Peng Liang, and Mojtaba Shahin. 2020. A Systematic Mapping Study on Microservices Architecture in DevOps. *J. Syst. Softw.* 170 (2020), 110798. https://doi.org/10.1016/j.jss.2020.110798

[26] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3, 1 (2016), 1–9.

[27] 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, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. ACM, 323–324. https://doi.org/10.1145/3183440.3194991