Received 25 April 2023, accepted 29 May 2023, date of publication 1 June 2023, date of current version 21 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3281860

SURVEY

Toward the Observability of Cloud-Native Applications: The Overview of the State-of-the-Art

JOANNA KOSIŃSKA[1,2], (Member, IEEE), BARTOSZ BALIŚ[1,2], MAREK KONIECZNY[1], (Member, IEEE), MACIEJ MALAWSKI[1,2], AND SŁAWOMIR ZIELIŃSKI[1]
[1]Institute of Computer Science, AGH University of Krakow, 30-059 Kraków, Poland
[2]Sano Centre for Computational Medicine, Extreme-Scale Data and Computing Team, 30-054 Kraków, Poland

Corresponding author: Joanna Kosińska (kosinska@agh.edu.pl)

This work was supported in part by the European Union's Horizon 2020 Research and Innovation Program through Sano under Grant 857533; and in part by the Sano Project carried out within the International Research Agendas Program of the Foundation for Polish Science, co-financed by the European Union under the European Regional Development Fund.

ABSTRACT The Cloud-native model, established to enhance the Twelve-Factor patterns, is an approach to developing and deploying applications according to DevOps concepts, Continuous Integration/Continuous Delivery, containers, and microservices. The notion of observability can help us cope with the complexity of such applications. We present a Systematic Mapping Study (SMS) in the observability of Cloud-native applications. We have chosen 56 studies published between 2018 and 2022. The selected studies were thoroughly analyzed, compared, and classified according to the chosen comparative criteria. The presented SMS assesses engineering approaches, maturity, and efficiency of observability by deliberating around four research questions: 1) What provides the motivations for equipping Cloud-native applications with observability capabilities? 2) Which research areas are addressed in the related literature? 3) How are observability approaches implemented? 4) What are the future trends in the Cloud-native applications observability research?

INDEX TERMS Cloud-native, microservice architecture, observability, monitoring, logging, tracing, systematic mapping study.

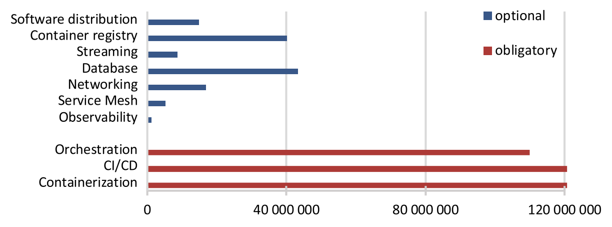
I. INTRODUCTION

The architectural style known as Cloud-native [1], [2], [3] has been established to augment the Twelve Factor [4] patterns of designing modern applications. Opposed to on-premise applications are new applications built in a Cloud-native manner fully exploiting the Cloud Computing (CC) model. Cloud-native is an approach to developing and deploying applications according to DevOps concepts [5], [6], Continuous Integration/Continuous Delivery (CI/CD) [7], [8], [9], containers [10], [11], and microservices [12], [13]. Cloud-native is a philosophy of programming. Adopting it requires a lot of effort and imposes new engineering challenges that organizations face.

The associate editor coordinating the review of this manuscript and approving it for publication was Massimo Cafaro.

Cloud-native Computing Foundation (CNCF) [14] in its trail map [15] among optional steps lists Observability & analysis. Observability is a concept that originated in Control Theory [16]. According to this theory, a system is observable if the current state can be determined in finite time using only the outputs. Measurement of overall microservice performance imposes the application's Quality of Service (QoS) metrics. The system has to properly externalize its state through instrumentation techniques to attain negotiated Service Level Agreement (SLA) parameters.

Observability is often defined as consisting of (i) metrics (collected while monitoring activities), (ii) logging, and (iii) tracing. Observability is an indispensable feature of every Cloud-native execution environment [17]. Its importance in the software lifecycle allows us to call all its activities observability engineering. Observability engineering focuses on



Results returned by Google search engine (12.2022).

FIGURE 1. Interest in components of cloud-native.

understanding multi-layered and complex architectures. The principal aim of observability is to make the Cloud-native system more understandable in numerous planes. The benefits of observability features underline gained capabilities allowing one to identify behavior deviations, perform root cause analysis, forecast failures, alert on events, optimize performance, proactively improve customer experience, etc.

Although recent publications indicate that Cloud-native observability is a significant topic [18], [19], there is a compelling demand for a systematic study that assesses the engineering approaches of observability, its maturity, efficiency, tools and also shows future research directions. The contribution of this work is the Systematic Mapping Study (SMS) [20] in the area of Cloud-native Application (CNApp) observability. SMS addresses all the research questions asked. As a result, it provides a map that is an overview of the state-of-the-art. Observability is an optional step of Cloud-native environments. Probably due to this option, this research area feels the scarcity of high-quality primary studies, especially papers related to logging and tracing. [21] recommended the SMS for areas with poor coverage. This fact also justifies our research.

In the first place, we have made a Systematic Literature Review (SLR). It helps prepare SMS by performing a more in-depth analysis of the literature. The SLR also considers the quality of the studies. Unlike usual, our SMS methodology includes the analysis of the full text of the selected studies. It is not based only on abstracts. Our SMS analyzes 56 primary studies from the last five years. The studies were selected based on the queries from the four defined research questions. Finally, we formed three categories related to Concepts, Management, and Supporters, with 12 entries. We gathered our data in summary tables and provided a set of plots representing the category entries.

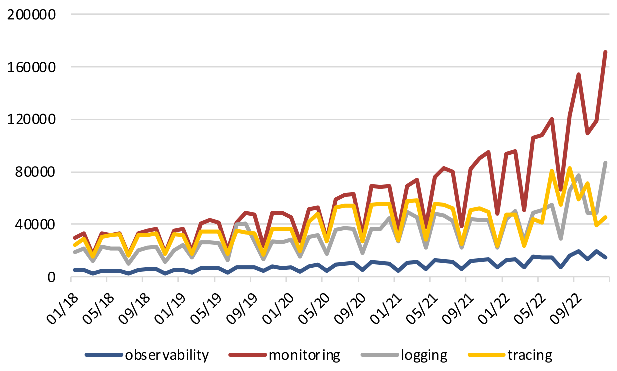
The structure of this paper is as follows. Section II on a few topics discusses key design patterns. It also presents related work. Section III describes the details of our methodology, including the selection process, categories, and the mapping of primary studies. The results with all the summary tables and graphs are discussed in Section IV, while the conclusions and possible future directions summarize Section V.

II. DOMAINS OF THE OBSERVABILITY

The term *observability* has been known for decades [16]. However, in the context of other technologies, this word has disappeared, or other substitutes are used. Fig. 1 shows

TABLE 1. Hits of observability domains returned by the google search engine.

Term	Monitoring	Logging	Tracing
No search hits	31 300 000	17 000 000	30 900 000



Results returned by Google search engine (01.01.2018 until 01.01.2023).

FIGURE 2. Trends of cloud-native observability and its domains.

Google trends for Cloud-native terminologies, which have distinguished CNCF on its landscape map. It has to be stressed that the Google search engine is not a good source of research findings, rather its hits are industry-directed. However, the results give a basic understanding of the researched field. The Google search engine results confirm the highest interest in the Cloud-native obligatory steps. Little attention is paid to the optional steps. However, in the case of the term *observability*, the absolute values are the lowest. The reason is the popularity of the word *observability*. Instead, its domains are often used, such as monitoring, logging, or tracing. The particular hits returned by the search engine are the following:

To restrict the context, we have decorated each word with the Cloud-native adjective. However, in the case of tracing, it is more often described as distributed than Cloud-native. Fig. 2 confirms that the word *observability* is less used than its domains. This word has maximally only about 20.000 hits. The most popular among businesses is monitoring. We can deduce that logging and tracing are underinvested domains. Therefore, it is probably worth further work. A similar situation is noticed in the scientific field, as resulting from our SMS.

Fig. 3 shows a high-level architecture of a modern observability ecosystem. Telemetry Data is collected from the underlying Cloud-native Applications and their Execution Environment. The definition of the Execution Environment is introduced in [17]. In short, it contains microservices, an orchestration system, containers, and a computing infrastructure. Data collected from standard logs, automatic instrumentation, and metric sampling are usually insufficient for understanding a complex CNApp with its execution environment. It is valuable to continuously add instrumentation to the application to collect more rich telemetry data. Telemetry data should be structured and time-stamped events containing

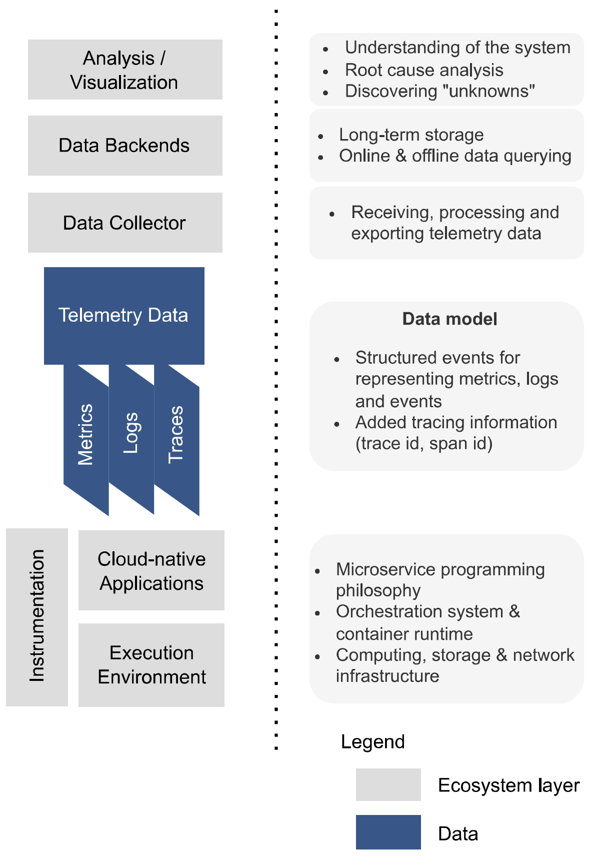

Analysis / Visualization
Understanding of the system
Root cause analysis
Discovering "unknowns"
Data Backends
Long-term storage
Online & offline data querying
Data Collector
Receiving, processing and exporting telemetry data
Telemetry Data
Metrics
Logs
Traces
Data model
Structured events for representing metrics, logs and events
Added tracing information (trace id, span id)
Instrumentation
Cloud-native Applications
Execution Environment
Microservice programming philosophy
Orchestration system & container runtime
Computing, storage & network infrastructure
Legend
Ecosystem layer
Data

FIGURE 3. The observability ecosystem.

metrics or log entries, along with trace information. Data Collector(s) is a layer that receives telemetry data from its sources and delivers them to various destinations. For example Open-Telemetry [22] divides this layer into Receivers (consuming data), Processors (transforming data), and Exporters (exporting data to backends). *Data Backends* provide long-term storage for telemetry data and advanced queries, used by the *Analysis/Visualization* tools.

Cloud-native applications gain the advantage if their execution environment supports observability features. Some benefits of the observability ecosystem are as follows:

- Increased visibility allows for seeing more details on what, where, how, and when things happen in the system. For example, what services are running, how they perform, where the application components are deployed when specific events occur, or notice unknown unknowns [23].
- Improved alerting and earlier issue detection – it allows early detection of issues even before they arise. Hence, it is possible to detect problems not only in the application performance but also in the architecture and design of the system.
- Improved development workflow and DevOps processes – it can drive the development process because of more valuable insights into the system behavior. As Cloud-native applications can rely on many cloud services, it is not always obvious to make some design decisions without observing the running system. Therefore, the insights from these observations can influence re-design or change in the architecture.
- Improved scalability – while designing the system, it is possible to ensure that all subsystems, particularly observability subsystems, are designed with scalability in mind, which allows adapting to the growing data demands when the system grows.
- Enabling automation – observability data can automate scaling, self-healing, and, in general, the autonomous control of the whole application.
- Valuable data for mining – collecting data about the holistic view of the system allows for more advanced analysis using statistics and machine learning, which in turn can provide additional information on the behavior of a CNApp and its execution environment.
- Ultimately, the observability of the application leads to a better user experience, customer satisfaction, and other business goals of the application under observation.

Including the observability ecosystem to support the above features is not always straightforward. Architectural designers should be aware of some challenges:

- Diverse and separate data sources – observability requires gathering data that are hard to integrate and correlate due to the differences in origin, type, and concern from which they come. These are challenges to the data model and its analysis, which needs building a complete picture of these separate and fragmented views.
- Big data challenges – combining telemetry data from a multitude of services in a distributed system leads to challenges related to, e.g., volume, velocity, variety, etc. These problems are typical for large data processing systems. Hence, running an observability subsystem will require solving data engineering and data science problems, as in large-scale analytic applications.
- Instrumentation, connectors, and overall configuration – these technical challenges need accounting for when planning and implementing observability solutions. They have costs in development, maintenance, and operations efforts.
- Performance overheads and interference – any active measurements on the running system always influence the performance of the observed system. Hence, it is significant not only to provide appropriate technology which is as non-invasive as possible but also to tune the observability parameters, e.g., the sampling frequency or the level of accuracy.

Typically, observability is focused on the performance of systems and can be defined as the ability to accurately capture, analyze, and present (collectively) information on the performance of a computer system [24]. The parallel computer systems of the 1990s were becoming so complex that measurement and monitoring were challenging due to the lack of appropriate methods and tools. We can notice that complex systems, e.g., physical or biological systems, pose challenges to observability. The ideas of observability and controllability known from Control Theory have become particularly relevant in the context of other technologies, such as, e.g., Autonomous Computing (AC) [25]. Today, as the complexity of computer systems, in particular Cloud Computing technologies, increases, and while many details of the computing infrastructure are hidden from the cloud provider, the observability of CNApps is an increasingly important and challenging topic.

A. RELATED WORK

There are many examples of SMSs and SLRs in the domains of Cloud Computing, microservices, and in CNApps in particular. One of the examples is the overview of Cloud-native applications after ten years of Cloud Computing [3]. It evaluates 49 papers on CNApps from a software engineering perspective. It discusses the areas of focus and trends that deal with CNApp engineering, and finally, it defines the Cloud-native. However, the term observability is not mentioned in the discussion, and neither logging, monitoring, nor tracing domains. A similar systematic study of CNApp design and engineering presents [26]. Although it analyzes 110 papers, it does not include any discussion of observability.

The paper [27] presents a systematic study of microservice architecture. The study selected 88 of the 2754 initial sets of papers and, finally, after manual check, included 42 articles in the survey. The authors examine how various architecture patterns depend on diverse migration, orchestration, storage, and deployment settings. One of the often mentioned advantages of the microservice architecture, as reported in the survey, was that a microservice architecture helps visualize the health status of all services in the system. Therefore, it allows for rapidly locating and responding to any problem.

An example of SMS applied to serverless computing [28] acknowledges testing and observability as one of the main categories included in the analysis, confirming that observability is an important topic. But in that work, only five papers were identified as belonging to this class. In addition, a related systematic study of microservice architecture [29] shows that monitoring (e.g., logging, profiling) has increased interest in the research community, but no detailed discussion of observability is provided.

There are also many papers focusing on cloud monitoring, e.g., [30] and [31], which provide an overview of problems, approaches, and tools for cloud monitoring. Some also propose taxonomies and discuss open questions, as in [32] and [33]. However, these papers do not cover all aspects of observability and do not focus on CNApps.

The conclusion from the related work, which motivates our research is there are no SMS that address the observability and its domains.

III. RESEARCH METHODOLOGY

Our methodology is based on the systematic mapping procedure [20] and the preparation of systematic reviews of the

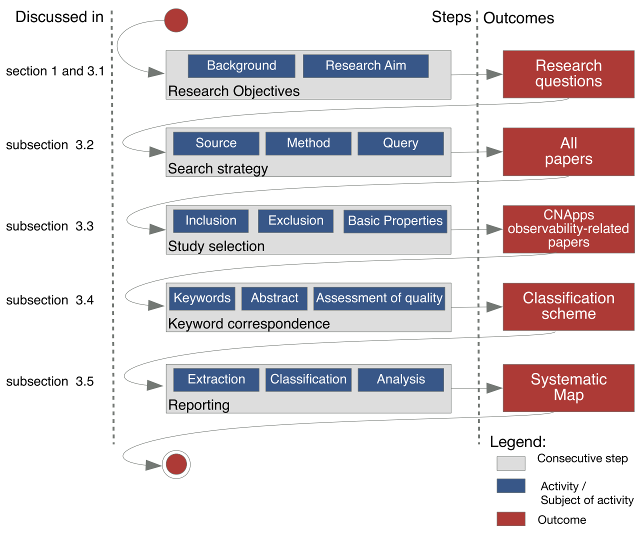


FIGURE 4. The procedure of observability SMS.

literature [21]. Both are in the software engineering field. We have adjusted the proposed concepts to our research area (Fig. 4). We propose to combine the approaches SLR and SMS. Our research resulted in the synthesis of evidence and provided this secondary study.

The following subsections form the successive outcomes of the proposed observability SMS that we have depicted in Fig. 4.

A. RESEARCH OBJECTIVES

The set of Research Questions (RQs) (Table 2) is an initial outcome of the observability SMS process. The further activities and their outcomes distinguished in the SMS procedure relate to the defined RQs. RQs cover diverse areas of the investigated field.

The generality of the defined RQs gives a realistic level of coverage for the Cloud-native observability domain. Furthermore, the focus of the research is not too narrow and is free from bias.

B. SEARCH STRATEGY

The search strategy (Fig. 5) helps to select studies for inclusion in the SMS. The activities are based mainly on the [20] guidelines and respond to the defined RQs. Identifying primary studies includes a combination of automated (over a set of databases and indexing services) and manual (over the citations) searching. We used both techniques. The findings of automated search answered RQ2, RQ3, and RQ4. To answer RQ1 we manually searched through the citations.

The search strategy also involves a suitably prepared query. The query consisted of search strings built in terms of the Population, Intervention, Comparision, Outcome (PICO) approach [21]. Their structure is depicted in Fig. 5 that is in the rightmost table. The particular search strings are composed while answering our research questions (listed in

TABLE 2. **Research questions.**

Concern	Research question (RQ)	Supplemental questions
Motivation	**RQ1** What provides the motivations for equipping CNApps with observability capabilities?	• Are the distinguished fields mature enough?
Structure	**RQ2** Which research areas are addressed?	• What is the established meaning of the observability term? • How many articles cover the different areas, and how has this popularity changed over time?
Methods and techniques	**RQ3** How are observation approaches implemented?	• What tools and techniques are used most frequently?
Directions	**RQ4** What are the recommended future trends in CNapps observability research?	• What challenges are distinguished for adopting observability? • What novel features do CNApps gain while equipping the execution environment with observability?

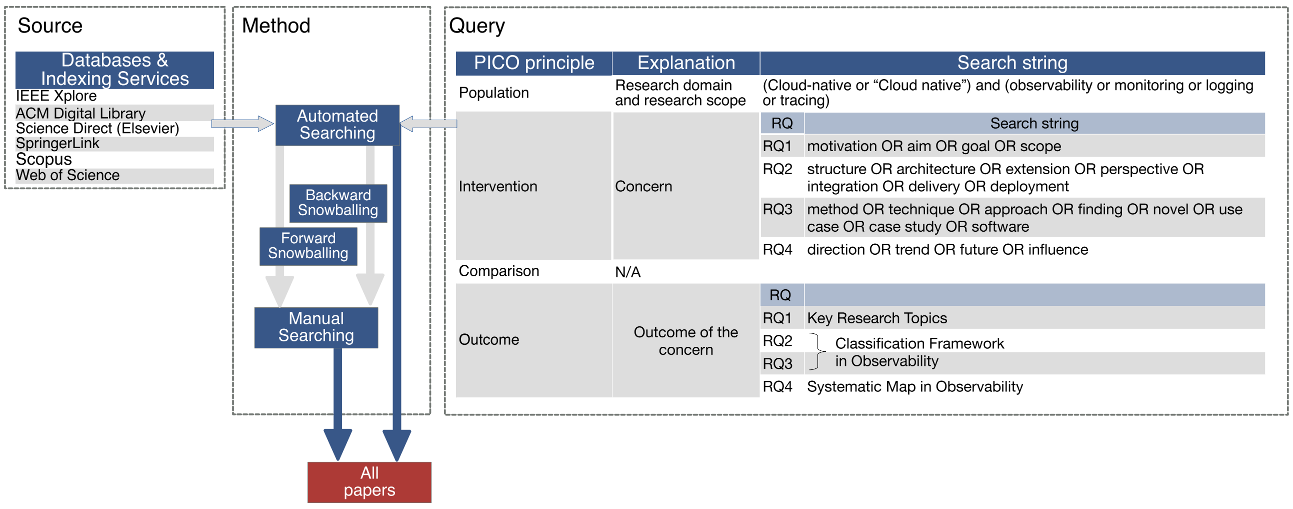

Source
Databases & Indexing Services
IEEE Xplore
ACM Digital Library
Science Direct (Elsevier)
SpringerLink
Scopus
Web of Science
Method
Automated Searching
Backward Snowballing
Forward Snowballing
Manual Searching
All papers
Query
PICO principle
Explanation
Search string
Population
Research domain and research scope
(Cloud-native or "Cloud native") and (observability or monitoring or logging or tracing)
Intervention
Concern
RQ
Search string
RQ1
motivation OR aim OR goal OR scope
RQ2
structure OR architecture OR extension OR perspective OR integration OR delivery OR deployment
RQ3
method OR technique OR approach OR finding OR novel OR use case OR case study OR software
RQ4
direction OR trend OR future OR influence
Comparison
N/A
Outcome
Outcome of the concern
RQ
RQ1
Key Research Topics
RQ2
RQ3
Classification Framework in Observability
RQ4
Systematic Map in Observability

FIGURE 5. **Search strategy.**

TABLE 3. **Sources of research studies.**

Database or Indexing Service	IEEE Xplore	ACM Digital Library	Science Direct (Elsevier)	Springer Link	Scopus	Web of Science
	21	14	20	34	28	11

Table 2). The *Explanation* column regards elements formulated in that table. The *Population* part, mainly reflecting the research domain of this study, is the same for all RQs. The *Intervention* part includes all elements of the *Concern* column of Table 2 and its synonyms. The constructed search question does not touch on the principle *Comparison* of the PICO approach. Agreeably, we establish that the *Outcome* part will constitute the results of the proposed methodology. Restricting the search string to that part of the table does not result in findings confirming that our research is unique.

The number of all primary studies found in the distinguished sources is presented in Table 3.

We also checked the sources as DBLP, Google Scholar, Semantic Scholar, or CiteSeerX, but they returned results as already in the previous searches. Therefore we did not list them as the primary sources. The total number of all primary studies is 147 and, without duplicates, 138. The next step of the SMS observability procedure is to choose primary studies. This step presents the following subsection.

C. STUDY SELECTION

We selected the primary studies on the observability of CNApp based on the inclusion and exclusion criteria listed below in a tabular form. Purposefully, we ranked them in the order shown. The order indicates the direction of the selection procedure. If the study meets the given criterion, this procedure stops, which results in the exclusion or inclusion of the study in the result of this step (Fig. 4). Additionally, in Table 4, we ended all criteria with the number of papers that fulfilled it. Finally, we have selected peer-reviewed studies on Cloud-native application observability published in the last five years. Table 5 presents selected studies.

In addition to the title, the table contains some basic parameters regarding the study. The associated reference number serves as a reference for citation purposes in the remainder of this manuscript.

D. CLASSIFICATION SCHEME

First, we used the authors' keywords associated with particular studies to develop a classification scheme. The

TABLE 4. Summary of selection criteria.

Exclusion criteria: 1) Not in English (8 papers). 2) Inappropriate publication title, for example, the paper was published in *Animal* journal (16 papers). 3) Too short study < 6 pages (13 papers). 4) The main focus is not on observability. This term is only mentioned in the introduction or the abstract (36 papers). 5) The study presents the same theme as the already selected study (9 papers).
Inclusion criteria: 6) Only research in the form of papers, books or chapters, and technical reports (56 papers).

keywording procedure is described in [20]. However, if the authors did not connect keywords with the study, we added some based on its Abstract. Second, a group of independent reviewers classified each study according to multiple criteria included in the designed entry form.[1] The goal of the form was to systematize the review and group the papers into respective categories. Note that the initial form evolved during subsequent reviews according to the reviewers' feedback. Third, the number of reviewers reduced the risk of entering routine and laconic responses. The results of the analyzes give a high-level view of the contribution to Cloud-native applications observability research.

The keywording process helped us distinguish representative categories:

Contribution – This category covers the type of study and the method that helped achieve the type. We also distinguished some properties that helped to match the study with its contribution. The categorization of study types is proposed in [89]. On its basis, we propose the entries in the Contribution category. They are presented in Table 6.

Most of the selected studies are considered evaluation and validation research papers. The knowledge introduced in these papers is interesting, and the methodology used is justified. Their abstracts encourage the reader to read on. The remaining categories are specific to observability and correspond to the research questions defined in Table 2.

Concepts – This category concentrates on Cloud-native and observability foundations. First, the studies distinguish various communities interested in equipping the solutions with an observability paradigm. Then they point to the research domain they deliberate on. Each study refers to the Cloud-native background compliant with the proposed observability ecosystem (Fig. 3). We propose the entries in the Concepts category in Table 7.

Management – The entries in the Management category we divided according to the components of the Application Lifecycle Management (ALM) [90]. Note that a single study does not address all aspects

[1]The form is available at https://tinylink.net/KXVJO

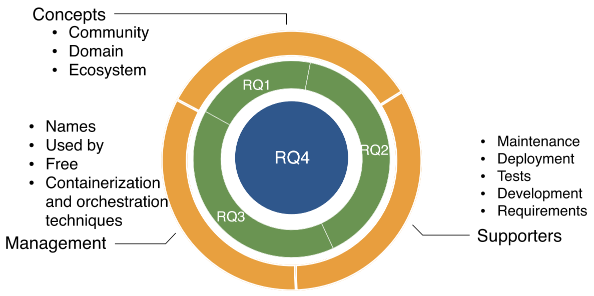


FIGURE 6. Mapping RQs into the feedback on observability-related category entries.

of ALM. ALM is an umbrella term that covers different aspects of application delivery, from planning, developing, testing, deploying, and finally maintaining. The ALM process (and hence our category) consists of several phases presented in Table 8.

Supporters – This is the last category and regards the technology, tools, and platforms used in the given observability domain. In this category, we propose the entries presented in Table 9.

The distinguished categories can be classified according to two different perspectives:

- General – covering the research contribution type, the methodology, and some properties as, e.g., background.
- Observability-related – covering the observability concepts and, particularly, the technology stack, the management techniques through the CNApps lifecycle, and last but not least, the proposed supporters, i.e., tools, platforms, and technologies.

In summary, in this subsection, we distinguished four categories that describe the selected studies. They establish two different perspectives on the classification of the studies. The general perspective regards all studies and classifies them generally. The observability-related categories with their entries aim to answer research questions. To recall, RQs cover diverse areas of the investigated field and give a realistic level of coverage for the Cloud-native observability domain. This Is illustrated in the following graph (Fig. 6). The four RQs map to the feedback on observability-related concerns in the following way:

- About RQ1 queries the category Concepts,
- about RQ2 queries the categories Concepts and Management,
- about RQ3 queries the categories Management and Supporters,
- RQ4 is a core of the mapping. Feedback in all categories answers this research question.

E. SYSTEMATIC MAP

Table 10 presents some final content of the analysis of primary studies related to observability of CNApps. This table and the rest form responses, constitute the input

TABLE 5. Selected studies.

Ref	Title	Domain			Type				Year
		Monitoring	Logging	Tracing	Journal Article	Conference Paper	Chapter	Book	
S1	A Black-Box Monitoring Approach to Measure Microservices Runtime Performance [34].	x			x				2020
S2	A layered framework for root cause diagnosis of microservices [35].	x	x	x		x			2021
S3	A Microservices Platform for Monitoring and Analysis of IoT Traffic Data in Smart Cities [36].	x				x			2019
S4	A Multitenant Container Platform with OKD, Harbor Registry and ELK [37].	x	x			x			2019
S5	A protocol-independent container network observability analysis system based on eBPF [38].	x				x			2020
S6	A Scalable Monitoring Framework for Network Slicing in 5G and Beyond Mobile Networks [39].	x			x				2021
S7	A Survey of Software Log Instrumentation [40].		x		x				2021
S8	A Survey on Observability of Distributed Edge & Container-Based Microservices [41].	x	x	x	x				2022
S9	A Tracing Based Model to Identify Bottlenecks in Physically Distributed Applications [42].			x		x			2022
S10	Adaptive processing rate based container provisioning for meshed Microservices in Kubernetes Clouds [43].	x			x				2022
S11	AI-based Autonomic & Scalable Security Management Architecture for Secure Network Slicing in B5G [44].	x			x				2022
S12	Ananke: A framework for Cloud-Native Applications smart orchestration [45].	x				x			2020
S13	Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks [46].			x	x				2022
S14	Automated Analysis of Distributed Tracing: Challenges and Research Directions [47].			x	x				2021
S15	Autonomic Management Framework for Cloud-Native Applications [17].	x			x				2020
S16	Benchmarking for Observability: The Case of Diagnosing Storage Failures [48].		x	x	x				2021
S17	Deploying the observability of the SigSaude system using service mesh [49].	x	x	x		x			2020
S18	Design, monitoring, and testing of microservices systems: The practitioners' perspective [50].	x			x				2021
S19	Detecting anomalies in microservices with execution trace comparison [51].			x	x				2021
S20	DevOps for IoT Systems: Fast and Continuous Monitoring Feedback of System Availability [52].	x			x				2020
S21	Distributed monitoring system for microservices-based IoT middleware system [53].	x	x	x		x			2018
S22	Does migrating a monolithic system to microservices decrease the technical debt? [54]	x			x				2020
S23	Efficient resourceful mobile cloud architecture (mRARSA) for resource demanding applications [55].	x			x				2020
S24	Enjoy your observability: an industrial survey of microservice tracing and analysis [56].		x	x	x				2021
S25	Graph-Based Trace Analysis for Microservice Architecture Understanding and Problem Diagnosis [57].			x		x			2020
S26	Improving observability in Event Sourcing systems [58].			x	x				2021
S27	JCallGraph: Tracing microservices in very large scale container cloud platforms [59].			x		x	x		2019
S28	Kmon: An In-kernel Transparent Monitoring System for Microservice Systems with eBPF [60].	x		x		x			2022
S29	Kubernetes Security and Observability: A Holistic Approach to Securing Containers and Cloud Native Applications [61].	x	x	x				x	2021
S30	Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs [62].			x		x			2019
S31	Localizing and Explaining Faults in Microservices Using Distributed Tracing [63].			x		x			2022
S32	LongTale: Toward Automatic Performance Anomaly Explanation in Microservices [64].	x		x		x			2022
S33	Metrics selection for load monitoring of service-oriented system [65].	x				x			2021
S34	MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments [66].			x		x			2021
S35	Microservices in IoT Security: Current Solutions, Research Challenges, and Future Directions [67].	x				x			2021
S36	Monitoring Network Flows in Containerized Environments [68].	x					x		2022
S37	Monitoring solution for cloud-native DevSecOps [69].					x			2021
S38	Observability, Microservices for the Enterprise [70].	x	x	x			x		2018
S39	Observability, Cloud Native Architecture and Design [71].	x	x	x			x		2021
S40	Offline Trace Generation for Microservice Observability [72].			x		x			2021
S41	On Observability and Monitoring of Distributed Systems – An Industry Interview Study [73].	x	x	x		x	x		2019
S42	Performance characterization of containerization for HPC workloads on InfiniBand clusters: an empirical study [74].	x			x				2021
S43	Predicting cloud-native application failures based on monitoring data of cloud infrastructure [75].	x				x			2021
S44	Prometheus and AIOps for the orchestration of Cloud-native applications in Ananke [76].	x				x			2021
S45	Refining microservices placement employing workload profiling over multiple kubernetes clusters [77].	x			x				2020
S46	Rusty: Runtime Interference-Aware Predictive Monitoring for Modern Multi-Tenant Systems [78].	x			x				2020
S47	Secure end-to-end processing of smart metering data [79].	x			x				2019
S48	Service Candidate Identification from Monolithic Systems Based on Execution Traces [80].			x	x				2019
S49	Service Mesh Based Distributed Tracing System [81].			x		x			2021
S50	SLATE: Monitoring Distributed Kubernetes Clusters [82].	x				x			2020
S51	SRv6-PM: A Cloud-Native Architecture for Performance Monitoring of SRv6 Networks [83].	x			x				2021
S52	T-Rank:A Lightweight Spectrum based Fault Localization Approach for Microservice Systems [84].			x		x			2021
S53	The Kaiju Project: Enabling Event-Driven Observability [85]	x	x	x		x			2020
S54	Using Service Dependency Graph to Analyze and Test Microservices [86].			x		x			2018
S55	ViperProbe: Rethinking Microservice Observability with eBPF [87].	x				x			2020
S56	X-MAN: A Non-Intrusive Power Manager for Energy-Adaptive Cloud-Native Network Functions [88].				x				2021
	Sum	38	13	28	24	28	5	1	

of the researched SMS. The SMS is an interpretation of all responses in the form. Every response is calculated. The frequencies obtained in the distinguished categories allow us to identify gaps and possibilities for future research.

IV. DISCUSSION OF RESULTS

The analysis of the annual CNCF report leads to a list of technologies that are considered stable and mature. Those technologies form the basis for most observability-related tools. In this section, we analyze the selected primary studies

TABLE 6. Contribution category.

Type	of the study
Evaluation research	This type of study sounds like an investigation. The study practically assesses the identified observability problem or the observability implementation technique.
Solution proposal	The study proposes a new solution for observability. The solution is based on observability concepts or is an extension of existing observability techniques.
Validation research	The study is in the experimental phase. It proposes a novel solution for observability that has not yet been implemented in practice.
Philosophical paper	The study proposes a new way to look at observability concepts, a new conceptual framework, taxonomy, etc.
Opinion paper	The study presents a subjective point of view on an observability topic. The statements are isolated from related work and other research methodologies. It states whether an approach is good or bad.
Experience paper	The study presents personal experiences from, e.g., a project and the learned lessons. The presented point of view comes from people who have used the approach (tools) in practice and report the experience.
Method	to achieve the type
Case study	It is an in-depth study that analyzes things to find patterns and causes of a particular behavior. A case study is used in finished products.
Example	It presents a single situation and is used to explain the approach.
Simulation	It differs from the case study, as it analyzes ideas and concepts in approaches that are still not production-ready.
Experiment	The study depicts two variables that test a given hypothesis.
Proof	The study has a solid theoretical background with mathematical definitions, theorems, etc.
Report	It presents the personal experiences from using the technique and covers the configuration details that help to achieve the same context. It is relevant for practitioners.
Application	It is a computer software.
Framework	It is a building block of observability paradigm covering the concepts and rules. It is independent of any implementation details.
Property	of the study that helps to assign the contribution type and method
Novelty	a significant improvement
Background	discussion of related work are sufficient
Technical	the research is used in production
Provokable	the presented research is surprising and provokes discussion

TABLE 7. Concepts category.

Community	The computing field apart from Cloud-native addressed in the study. It can be distributed computing, peer-to-peer computing, cloud computing, edge computing, fog computing, internet of things (IoT), or autonomic computing.
Subject areas	They can be computing or non-computing. The examples are big data, virtualization, HPC, medicine, physics, education, etc.
Domain	The domain (monitoring, logging, tracing) of the observability elaborated in the study.
Ecosystem	The entry comprises the technology stack that builds the observability ecosystem. The distinguished building blocks are shown in Fig. 3. Starting from the top these are Analysis/Visualization, Data backends, Data Collector, Telemetry data (which are already included in Domain entry), and CNApps with their Execution Environment. Lastly, there is the Instrumentation of CNApps-related blocks. The selected studies especially concentrate on the Execution Environment, covering the Microservices, Oerchestrator, Image and Infrastructure categories. All of them have following entries: construction, communication, management, execution.

with the CNCF defined taxonomy in mind. Note that the first CNCF project graduation (Prometheus) took place in 2018, a date that directly corresponds to the range of our SMS.

A. GENERAL FINDINGS

Cloud-native observability has not been extensively researched yet (Section II). In our SMS, only three papers [S21], [S38], [S54] have been published in 2018. However, Fig. 7 shows that the trend is increasing each year. The first mentioned study [S38] is a chapter from the book Cloud Native Architecture and Design. It discusses observability and scarcely its domains. In our opinion, it does not propose, as the title of the book suggests, architecture and design concepts. Rather, it concentrates on the implementation and demonstrates useful tools such as Prometheus for metrics, Spring Cloud Sleuth, Zipkin, and Jaeger for tracing, Fluentd for logging, and Grafana for visualization. However, these tools have already been recommended by CNCF. More significant studies are expected to be published in the coming years. Fig. 8 presents the distribution of the studies in the observability domains while pointing to the most cited studies.

Extending the Cloud-native system of the observability capabilities requires deep reengineering of the system [S2], [S8]. The management of CNApp must address the entire

TABLE 8. Management category.

Requirements	include a range of factors such as end-user needs, compliance, and governance requirements. The entry is divided into two groups: • Functional – these are major concerns that have an impact on the final features of concepts of similar classes: -- Architecture, e.g., topology, construction, middleware -- Execution, e.g., provisioning, scheduling -- Management, e.g., SLA management (resource availability), policy-driven, end-to-end performance, root cause analysis • Nonfunctional – distributed, heterogenous, (self-)adaptable, autonomic, interoperability, self-service, integration, profiling
Development	according to DevOps
Tests	compliance, governance, efficiency, usability, security, performance, and other benchmarks.
Deployment	although Cloud-native requires that this step is completely automatized, this does not have to be described in the study. Hence we distinguished: automatic, not mentioned.
Maintenance	faults correction, adaptation, migration.

TABLE 9. Supporters category.

Property	of the study that helps term used the supporters
Free	a boolean indicating whether all supporters are freely available.
Used by	informs which communities use these supporters. It can take the following values: CNCF, Business, Other Research, and Own Usage.
Containerization and orchestration techniques	in Cloud-native these are mandatory features. These parameters indicate the deployment environment.
Names	the names of elaborated components.

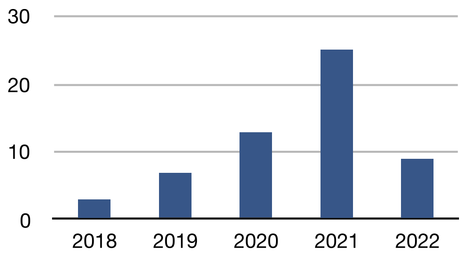


FIGURE 7. Distribution among years.

TABLE 10. Selected form responses.

	Ecosystem entries								Management category										
	μ-services				Infrastructure				Requirements						Maintenance				
Ref	Construction	Communication	Management	Execution	Construction	Communication	Management	Execution	Architecture	Execution	Management	Other ((non)+functional)	Dev Ops	Other (Testing)	Deployment	Faults Correction	Adaptation	Migration	Other
S1	x	x	x	x		x	x		x		x	low overhead, accuracy	energy efficiency	cost of data collection					
S2				x							x			fault injection					
S3	x	x	x	x	x	x	x	x	x	x	x	devops, reliability	kubernetes operators deployment in cloud	time constraints	x				state persistence
S4		x	x	x					x	x		security policies	CI/CD policies integration				x		
S5		x	x	x	x			x	x	x	x			chaos engineering		x			anomalies detection
S6	x	x	x	x	x	x	x	x	x	x	x	privacy, multi-tenancy	devops integration	resource utilization			x	x	slice management
S7	x	x	x				x		x		x	security and quality of observability, maintainability	ci/cd pipeline	code coverage			x	x	
S8									x		x								
S9	x	x	x	x					x			tracing is available	CI/CD pipeline for tracing	root cause analysis	x		x		monitoring quality of the trace data
S10	x	x	x	x	x				x	x	x		observability effectiveness	evaluation of algorithms	x		x		policies violations
S11	x	x	x		x	x		x	x	x	x	security	CI/CD pipeline		x	x			software evolution
S12		x	x	x		x	x		x		x	quality of monitoring data, fault tolerance	performance analysis	optimal resource allocation	x	x	x		eventual consistency in system
S13				x										penetration testing					
S14			x	x				x			x	anomalies detection in traces		trace data quality analysis	x	x			tracing standard format specification
S15	x	x	x	x			x	x	x	x	x			availability			x	x	
S16				x			x		x				tracing tools						
S17	x	x		x		x	x	x	x	x	x	centralized observability	CI/CD pipeline		x		x		metrics management
S18	x	x	x	x		x	x		x		x	different design patterns	CI/CD pipeline	survey study		x			survey study
S19				x							x			fault injection		x			
S20	x				x	x			x		x	anomaly detection and feedback loop	CI/CD pipeline, quality of observability	IoT system used in production	x	x			monitoring on demand
S21		x	x	x			x	x	x				trace sampling					x	
S22	x				x	x	x		x		x		deployment of architecture		x			x	
S23	x	x				x	x	x	x	x	x		virtualization				x	x	
S24		x	x	x					x	x	x	quality of reviews	trace and log dependency analysis			x		x	trace and root cause analysis system
S25		x		x							x								
S26	x			x					x		x								
S27		x	x	x							x	overhead minimization	devops integration	stress test sampling		x			invocation context
S28		x	x	x		x			x	x	x			resources overhead			x		
S29		x	x	x			x		x	x	x		analyzing logs			x			
S30			x	x					x	x	x	error prediction	data collection			x		x	
S31				x		x			x		x	devops integration	detecting faults	fault injection	x	x			
S32	x	x	x	x		x	x		x	x	x		observability quality		x		x		long-run monitoring
S33			x				x	x	x		x			load tests					load management
S34		x	x	x	x	x	x	x	x	x	x	overhead, e2e tracing efficiency	devops integration, microbenchmark	chaos engineering and fault localization		x	x	x	holistic monitoring, root cause analysis
S35		x							x	x	x	ethical consideration	integrating microservices				x		
S36						x			x	x	x	low overhead	network monitoring						
S37	x		x	x			x		x	x	x	security	CI/CD security pipeline	continuus testing	x				system monitoring
S38	x	x	x	x					x	x		observability stack	containerization of apps						
S39		x	x				x				x	usability and visibility	anomalies detection for different stakeholders				x		
S40	x	x	x	x			x	x	x	x	x	adherence to standards	focus on abstractions				x		benchmarking
S41			x						x	x	x	usability, governance and extensilibity	recovery and fault detection			x			root cause analysis
S42						x		x	x										
S43			x		x	x		x	x	x	x	scalability and resiliency	model training and optimalization	latency deterioration and faults injection	x	x	x		
S44							x		x	x			virtualization					x	
S45		x	x	x	x	x	x	x			x	resources management	devops integration	networking test	x			x	devops in IoT
S46							x	x				PCM metrics	performance prediction	interference scenarios	x				
S47	x	x	x	x			x		x	x		computation privacy							redundancy, replication
S48	x	x							x		x	reliability		extensibility of system			x	x	
S49		x	x	x					x		x	service level error detection	devops integration	fault injection		x			visualisation and verification tracing
S50		x	x	x			x		x		x	authentication	devops integration	storage	x		x		devops in maintenance
S51					x	x	x	x	x	x	x	real time processing	CI/CD pipeline		x		x		network heterogeneity
S52		x	x	x		x		x	x	x	x	low overhead, accuracy	CI/CD pipeline	fault injections		x			root cause analysis
S53	x		x				x		x		x	usability	cluster management		x		x		logs aggregation
S54	x	x							x		x	test coverage,		detection of cyclic dependencies					
S55	x	x	x	x			x	x	x	x	x	configurability	deployment patterns	dynamic metric selection			x		offline and online metrics analysis
S56	x	x	x	x			x	x	x	x	x	no instrumentation, reliability	devops integrations	latency, throughput			x		energy managemnet
Sum	24	35	36	37	12	20	28	20	47	28	46				18	17	22	12	

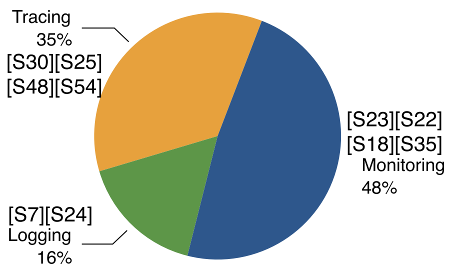


FIGURE 8. Distribution between observability domains.

TABLE 11. Distribution between popular publication forums.

	Name	Count	Studies
Journals	Journal of Systems and Software	3	[S22] [S26] [S18]
	IEEE Transactions on Network and Service Management	3	[S6] [S51] [S56]
	Journal of Grid Computing	2	[S14] [S15]
	IEEE Access	2	[S45] [S8]
	Journal of Cloud Computing	2	[S23] [S47]
Conferences	European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)	2	[S30] [S25]
	IEEE International Symposium on Network Computing and Applications (NCA)	1	[S2]
	IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)	1	[S52]
	International Conference on Service-Oriented Computing (ICSOC)	1	[S41]
	International World Wide Web Conference (WWW)	1	[S34]

CNApp execution environment. The observations must be performed across all levels in the Cloud-native application stack [S15]. Paper [S18] proposes to observe the changes in the code while the system migrates from a legacy and monolithic system to an ecosystem of microservices. This analysis helps to understand the technical debt. Such a holistic view aims to gain a deep understanding of how to design microservices systems, monitor, and test them [S18].

Among the detailed parameters of the studies, its publication title allows one to determine the most popular journals and conferences that researchers find appealing to present their work. Regardless of the quality of the publication forum, the study counts are small. However, it should be noted that the following study is the most cited and has 113 citations [S30] –

> X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, C. He (2019). Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.

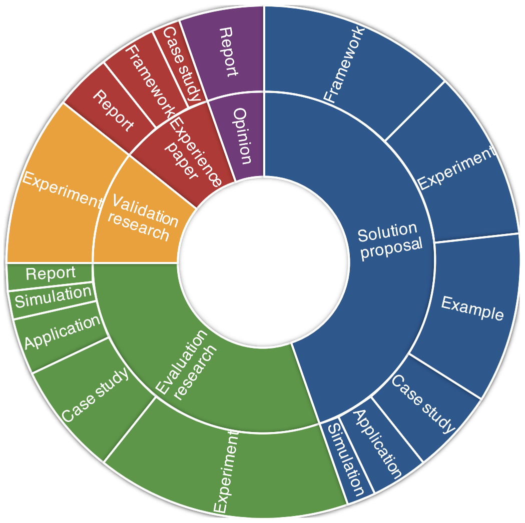


FIGURE 9. Distribution between type of contribution and research method.

1) RESEARCH CONTRIBUTION AND THE METHODOLOGY

Fig. 9 shows the distribution of primary studies by their type of contribution (shown in the inner ring) and the research method (shown in the outer ring). The two largest groups of contribution types are solution proposals (25) and evaluation research studies (17). Both groups are diverse in terms of the adopted methods. There are then six validation research papers that exclusively rely on the experiment method [S6], [S10], [S13], [S19], [S23], [S25]. Showing an experiment is the most frequently used research method (21 papers), followed by the framework (9), case study (8), report (6), example (5), and simulation (2). Within the report methodology, several studies have used user surveys for various purposes. These are experience papers [S18], [S24], or opinions [S41].

B. OBSERVABILITY-RELATED FINDINGS

During the keywording process of the observability SMS procedure (Fig. 4), we created a cloud of keywords. Fig. 10 shows the results. In this step, we can already draw the following conclusions. Cloud-native is most often described as composed of microservices (e.g. [S2], [S45], [S52], [S55]) running as Docker containers that manage the Kubernetes orchestrator (e.g. [S4], [S10], [S29], [S45]). The second-top keyword is monitoring (e.g. [S1], [S3], [S44], [S18]). From the observability ecosystem, this is the most widely researched telemetry but is also tracing (e.g. [S27], [S34], [S54]). Following the figure, the collected data help in anomaly detection [S13], [S32], fault localization [S30], root cause analysis [S34].

There are many directions of development (e.g., machine learning [S5], 5G networks [S6], etc.) worth looking.

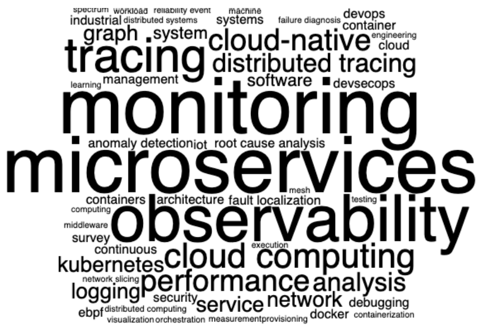


FIGURE 10. The cloud of keywords.

TABLE 12. Distribution between main research focus.

Computing field	Count	Studies
Cloud Computing	48	
Distributed Computing	48	
Internet of Things	11	[S3] [S8] [S9] [S20] [S21] [S35] [S41] [S44] [S45] [S47] [S49]
Edge Computing	11	[S6] [S8] [S9] [S11] [S20] [S31] [S36] [S50] [S51] [S56]
Autonomic Computing	9	[S7] [S10] [S12] [S15] [S23] [S28] [S30] [S36] [S53]
Fog Computing	3	[S3] [S20] [S56]
Peer-to-Peer Computing	3	[S23] [S34] [S36]

However, the efforts taken have a common goal, namely improving the performance [S1], [S42], [S51] in various directions (e.g., network [S36], middleware [S21], kernel [S28], etc.) of a CNApp.

1) THE FOUNDATIONS

The first observability-related category is the Concepts that concentrate on the basics of the Cloud-native with emphasizing observability. Cloud-native computing assumes the use of cloud infrastructure which is central to the majority (48 out of 56) of the reviewed studies. The rest do not directly mention Cloud Computing (CC), but, e.g. [S13] must operate in its context.

The selected research also addresses Computing fields such as the Internet of Things, Edge Computing, or Fog Computing. However, all of them come from the needs of CC

The tight integration of enterprise systems with a Cloud-native approach that is undoubtedly attractive from the financial perspective cannot complicate resource management and utilization. This requirement is particularly critical if the management cost increases significantly. It explains the appearance of Autonomous Computing in Table 12. AC provides self-management of resources by ensuring their capabilities, such as self-configuration, self-optimization, self-healing, and self-protection [91]. They emerge as a result of observing the whole execution environment.

The selected studies touch upon diverse subject areas, among which we can find: Mobile systems (19), Big data (11), Security (8), Machine Learning / Artificial Intelligence (ML/AI) (7), Networking (6), Telecommunication (6), Visualization (6), High-Performance Computing (HPC) (2). We found a few papers focusing on scientific computing, e.g., HPC systems [S42]. However, we can conclude that in HPC/scientific computing monitoring in the traditional sense is still employed.

a: THE ECOSYSTEM

The core of the observability ecosystem (Fig. 3) makes up its telemetry data. They are the subject of research in all primary studies. Apart from the data, the studies also focus on the Execution Environment (bottom part of the ecosystem figure). Our form surveys some of its components. The responses group into four concerns: construction, communication, management, and execution. Table 10 contains the responses of (i) the microservices and (ii) the entries of the computing infrastructure. At the end of the table, we also present some statistics. These concerns among orchestration systems and containers are as follows: construction – 14, communication – 13, management – 19, and execution – 22. The results are less than those achieved for microservices. It is because microservices run as containers managed by orchestrators.

In some cases, the observability ecosystem (Fig. 3) or its elements are themselves the subject of research. In [S37], this is motivated by a lack of a solution for integrated monitoring of infrastructure and applications. The novel monitoring systems propose [S6] for 5G networks, [S21] for microservice-based IoT middleware, or [S28] for microservice systems in general. Subsection IV-B3 includes an in-depth analysis in this regard.

2) MANAGEMENT TECHNIQUES

The management concerns address a wide range of topics, as described in Section III-D. For example, some studies address the observability as an aspect of application development (e.g., [S28]). In contrast, others focus entirely on observing a given application or system in a specific use case (e.g., [S45]).

Table 13 and partially Table 10 summarize our findings in the context of the functional requirements of the selected studies. We organized them into the following three groups: (i) Architecture, (ii) Execution, and (iii) Requirements. The architecture group focuses on the topology and construction of the observability layer and its dependencies, especially on issues related to microservices architecture and patterns such as Application Programming Interface (API) composition. In addition, middleware often addresses patterns used in application processing (e.g., Saga or Command and Query Responsibility Segregation (CQRS)). Studies also often address the observability of the network, its efficiency, and overhead.

The Execution group, on the other hand, is dominated by the provisioning concern (compare the number of studies in that group, which list Table 13). It is mainly due to the complexity of the execution environment. Hence, as part of the observability process, also because of the need to analyze all environmental traces. The scheduling concern addresses the issues related to scaling and optimizing execution

TABLE 13. Distribution between functional requirements.

Req	Concern	Terms	Count	Studies
Architecture	topology	architecture design, microservices dependencies, scalability	30	[S1] [S6] [S8] [S12] [S15] [S17] [S18] [S20] [S23] [S24] [S28] [S29] [S30] [S31] [S34] [S35] [S40] [S41] [S44] [S50] [S51] [S52] [S53] [S55]
	construction	system components, service mesh, software stack selection	29	[S3] [S4] [S5] [S7] [S9] [S10] [S11] [S15] [S18] [S23] [S28] [S32] [S40] [S43] [S48] [S49] [S50] [S52] [S54] [S55] [S56]
	middleware	middleware API	22	[S4] [S5] [S6] [S10] [S16] [S18] [S21] [S22] [S26] [S31] [S33] [S34] [S35] [S38] [S40] [S44] [S47] [S51] [S52] [S53] [S56]
	networking	processing network flows in system, data plane topology	10	[S3] [S36] [S6] [S42] [S18] [S51] [S56]
	efficiency	resource overhead, bottleneck	3	[S9] [S18] [S12]
Execution	provisioning	run-time tracing, system overhead, slice isolation, creating, faults injection	32	[S3] [S4] [S5] [S6] [S10] [S11] [S15] [S17] [S23] [S28] [S30] [S32] [S34] [S35] [S36] [S38] [S40] [S41] [S43] [S47] [S51] [S52]
	scheduling	scalability, autoscaling, queuing, optimizing, load balancing	16	[S6] [S10] [S44] [S47] [S15] [S17] [S55] [S28] [S29] [S30]
	teamwork	team organization and communication	2	[S37]
Management	root cause	anomalies detection and analyses, failure prediction, faults injection and management, dependency analysis	36	[S2] [S5] [S7] [S11] [S14] [S19] [S24] [S25] [S26] [S27] [S30] [S31] [S32] [S34] [S35] [S39] [S41] [S43] [S49] [S52] [S53] [S54]
	SLA	load management, observability overhead, resource availability, quality of service, SLA violations and management	30	[S6] [S7] [S8] [S10] [S11] [S12] [S15] [S17] [S18] [S20] [S22] [S23] [S27] [S28] [S29] [S33] [S36] [S39] [S41] [S45] [S48] [S50] [S51] [S52]
	end-to-end (e2e)	microservice performance, response time, latency analysis, e2e tracing	28	[S1] [S3] [S6] [S8] [S10] [S12] [S18] [S27] [S29] [S30] [S31] [S32] [S34] [S36] [S39] [S40] [S41] [S49] [S50] [S51] [S52] [S55]
	policy-driven	policy-based configuration and management	10	[S32] [S33] [S35] [S6] [S41] [S11] [S15] [S48] [S28]

overhead. Finally, only one study focuses on the execution of observability aspects in many functional teams [S37]. It is worth noting because it reflects the multi-aspect implications of observability.

In the last group, i.e., the Management group, an important role plays root cause analysis. It includes topics related to fault injection and management, anomaly detection, and service dependency analysis. Observability of the system often requires specific SLA agreements, especially in resource management, resource utilization, overhead, and high availability. These three top characteristics equally distribute in our analysis. Finally, an important role plays a policy-driven approach.

Summing up our findings, an application's complete life cycle management relies on observability, including security and automation.

The distributed nature of modern cloud applications and the increasingly complex integration of the execution environment are the most important non-functional requirements (Table 14). Surprisingly, scalability is not the primary concern. It is mainly because the execution environment offers some mechanisms for scalability on which the observability components may rely (such as CC and Virtualization). In addition, the characteristics of the Development phase, where DevOps integration and CI/CD pipeline dominate (DevOps column in Table 10), support this claim.

Another vital aspect that is visible in our survey is the increasing importance of the quality aspects of the observability (column Other (non)+functional in Table 10). Among the responses, the stress is on the quality and compatibility of the data ([S7], [S12], [S14], [S40]), accuracy and time constraints ([S1], [S51]) and the usability of the observability ([S39], [S41], [S53], [S46]).

TABLE 14. Distribution between non-functional requirements.

Almost every study is distributed. Hence, we do not list them.

Concern	Count	Studies
distributed	48	
integration	38	[S1] [S3] [S4] [S5] [S6] [S7] [S9] [S10] [S11] [S12] [S14] [S15] [S17] [S18] [S20] [S21] [S22] [S23] [S24] [S28] [S29] [S30] [S34] [S35] [S38] [S39] [S40] [S41] [S43] [S44] [S47] [S48] [S49] [S50] [S51] [S52] [S55] [S56]
autonomic	18	[S3] [S4] [S6] [S9] [S10] [S11] [S12] [S14] [S15] [S16] [S23] [S27] [S43] [S44] [S46] [S50] [S52] [S54]
heterogenous	17	[S5] [S8] [S9] [S11] [S15] [S18] [S21] [S23] [S28] [S34] [S36] [S38] [S40] [S41] [S45] [S51] [S54]
interoper-ability	18	[S3] [S5] [S6] [S15] [S18] [S21] [S22] [S32] [S35] [S39] [S40] [S47] [S48] [S51] [S53] [S54] [S56]
profiling	17	[S1] [S5] [S6] [S12] [S14] [S15] [S16] [S23] [S27] [S32] [S33] [S41] [S45] [S51] [S52] [S55] [S56]
self-adaptable	12	[S1] [S4] [S7] [S10] [S12] [S15] [S17] [S23] [S28] [S39] [S52] [S56]
self-service	7	[S4] [S15] [S22] [S23] [S41] [S47] [S56]
scalability	7	[S43] [S18] [S49] [S54] [S3] [S6] [S17]

Similarly, from a testing point of view, the focus on performance and efficiency of resource usage dominates in the analyses (Table 15).

The adaptation and faults correction concerns dominate from the application maintenance point of view (Maintenance columns in Table 10). It aligns with the requirements con-

TABLE 15. Testing concerns.

Almost every study touches performance aspects, hence we do not list them.

Concern	Count	Studies
performance	43	
efficiency	34	[S1][S3][S5] [S6] [S7] [S9] [S10] [S12][S17] [S18] [S20][S22] [S23] [S24] [S27] [S28] [S29] [S31] [S33] [S34][S36][S40] [S41] [S42] [S43] [S45] [S46] [S47] [S49] [S51] [S52] [S53] [S55] [S56]
security	17	[S4] [S6] [S7] [S8] [S11] [S12] [S15] [S20] [S29] [S35] [S37] [S39] [S41] [S47] [S48] [S50] [S51]
compliance	15	[S6] [S7] [S8] [S9] [S14] [S15] [S16] [S30] [S32] [S39] [S40] [S44] [S48] [S51] [S52]
governance	7	[S6] [S10] [S11] [S15] [S35] [S41] [S50]

TABLE 16. Distribution of containerization techniques.

Container type	Count	Studies
Docker	20	[S1] [S3] [S4] [S5] [S13] [S15] [S17] [S18] [S19] [S21] [S22] [S30] [S37] [S42] [S47] [S51] [S52] [S54] [S55] [S56]
Singularity	1	[S42]
Apache Mesos	1	[S15]

TABLE 17. Distribution between the orchestrators.

Orchestrator	Count	Studies
Kubernetes	31	[S1] [S2] [S4] [S5] [S6] [S9] [S10] [S11] [S15] [S17] [S19] [S21] [S22] [S28] [S29] [S30] [S31] [S34] [S38] [S39] [S40] [S43] [S44] [S45] [S47] [S49] [S50] [S51] [S52] [S53] [S55]
OpenShift	6	[S3] [S4] [S6] [S12] [S17] [S44]
Docker Swarm	2	[S15] [S37]

cerns (where root cause and SLA topics are the most popular) and the testing concern (where topics related to fault injection are widespread [S2], [S5], [S9], [S19], [S31], [S34], [S43], [S49], [S52]).

3) TECHNOLOGY, TOOLS, AND PLATFORMS

Cloud-native applications are built from microservices hosted in containers. The prevalent (Table 16) containerization technique in our primary studies is Docker. Singularity and Apache Mesos are also mentioned, but the popularity of those environments is low. Note that the popularity of Docker is greater because most papers are focused on higher-level considerations, such as orchestration and observability.

The services were typically orchestrated with Kubernetes or OpenShift. Some researchers organize their services in meshes, based mainly on Istio. The counts are summarized in Table 17.

Regarding the runtime cloud environment, it seems that there are no clear preferences concerning the IaaS provider (Table 18). Most often the OpenStack is used for private cloud, but the numbers do not show a significant domination. The papers rarely inform about the methods used for infrastructure creation; presumably, they are created using environment-specific tools (it is noteworthy that Terraform was mentioned only once [S11]).

Table 19 summarizes the software used in our primary studies. The main interest is in software similar to that proposed by CNCF for observability and analysis on its trail map [15]. Software of CNCF graduated level (i.e., stable and widely used in production) we highlighted in bold font. Concerning monitoring, logging, and tracing, it is safe to assume that the researchers follow the CNCF graduated projects. The suggested technologies fit their needs.

Elastic offers a complete observability stack. It was formerly known as Elasticsearch and commonly referred to as

TABLE 18. Distribution between IaaS providers.

IaaS cloud provider	Count	Studies
OpenStack on private cloud	3	[S11] [S14] [S47]
MS Azure	1	[S22]
Alibaba	1	[S5]
Huawei	1	[S14]
Google	1	[S28]
Amazon	1	[S50]
Kubernetes playground	1	[S51]

TABLE 19. Distribution of observability software usage.

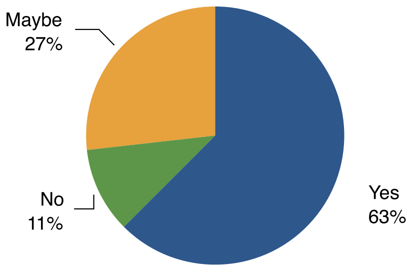
Domain	Name	Count	Studies
Observability Stack	Elastic	4	[S2] [S4] [S27] [S43]
Deployment and control	asperathos	1	[S47]
	CloudRanger	1	[S52]
Monitoring	**Prometheus**	16	Table 21
	Thanos	1	Table 21
	cAdvisor	1	[S11]
	nProbe	1	[S36]
	Zeek	1	[S36]
	Telegraf	1	[S51]
	OpenStack Monasca	1	[S47]
Logging	**Fluentd**	3	[S29] [S38] [S53]
	log4j	2	[S7] [S24]
	SLF4J	1	[S7]
	Splunk	1	[S7]
Tracing	**Jaeger**	14	[S2] [S13] [S14] [S17] [S19] [S24] [S26] [S31] [S32] [S34] [S38] [S40] [S53] [S55]
	Zipkin	8	[S14] [S21] [S24] [S26] [S38] [S40] [S49] [S55]
	OpenTelemetry	6	[S14] [S24] [S34] [S40] [S49] [S52]
Data aggregation	Apache Kafka	5	[S6] [S11] [S31] [S45] [S51]
	Esper	1	[S53]
Visualization	Grafana	13	[S1] [S4] [S6] [S14] [S15] [S17] [S29] [S37] [S38] [S43] [S47] [S50] [S51]
Root cause detection	MS-Rank	1	[S52]
	Monitor Rank	1	[S52]

TABLE 20. Distribution between infrastructure-specific tools used for observations.

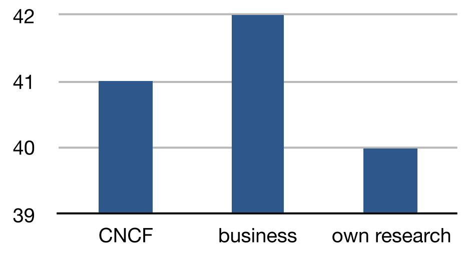
Tool name	Function	Count	Studies
eBPF	network traffic analysis	6	[S5] [S28] [S36] [S51] [S55] [S56]
bcc	BPF toolkit	2	[S28] [S55]
linkerd	network proxy	2	[S49] [S55]
XDP-tools	bypassing system networking stack	1	[S56]
Turbostat	CPU monitoring	1	[S56]

TABLE 21. Distribution of data storage methods.

Tool name	Type	Count	Studies
Prometheus	monitoring system	16	[S1] [S4] [S6] [S11] [S15] [S17] [S29] [S33] [S34] [S38] [S39] [S43] [S44] [S45] [S50] [S53]
Thanos	monitoring system	1	[S50]
InfluxDB	time series database	3	[S6] [S37] [S51]
Neo4j	graph database	2	[S9] [S54]
ArangoDB	graph database	1	[S51]
CascaDB	key-value database	1	[S47]
PostgreSQL	relational database	1	[S17]



(a) Percentage distribution of responses to question: Are all supporting technologies freely available?



(b) Responses to question: Which communities use the supporting technologies?

FIGURE 11. Additional features of the supporting technologies.

the ELK Stack (from Elasticsearch, Logstash, and Kibana). Elasticsearch is the core component of the Elastic Stack. Its unified interface for logs, traces, and metrics, which simplifies insight into the cloud system, is a significant virtue of this stack.

Observing the underlying systems sometimes forces us to use infrastructure-specific tools (Table 20). Noticeably, Extended Berkeley Packet Filters (eBPF) is of significant focus. It provides a low-level interface for network traffic analysis and CNCF projects that is, Cilium, Falco, or Hubble use it.

The results of the observations are kept by the monitoring systems or stored directly in databases (Table 21).

Research must be reproducible. To do this, all supporting technologies should be freely available. Fig. 11a shows the distribution of this characteristic among our selected studies. Thirty-five of them use free software, and only six studies use software with paid licenses.

Fig. 11 summarizes our findings. It is noteworthy that reviewers of two-thirds of the studies noted that all supporting technologies are freely available. Furthermore, more than 70% of the used technologies CNCF recommends.

V. CONCLUSION

Building and executing applications as Cloud-native is an architectural style that has gained importance in the research community. Its crucial aspect is observability, which consists of monitoring, logging, and tracing. Although Cloud-native observability is not a new research topic, there have been no systematic review studies so far that motivated our work. In this paper, we conducted a systematic mapping study (SMS) in which we reviewed 56 recent publications, which we selected according to the defined search criteria.

The main contribution of our work is a systematic map, which we presented in Section IV. We summarize the main conclusions in the following grouping. Each group represents our research questions.

- RQ1: What provides the motivations for equipping CNApps with observability capabilities? – The main motivations come from the execution stage of Cloud-native applications. They relate to (i) provisioning focusing on run-time tracing, (ii) measuring the system overhead, and (iii) scheduling, including scalability, autoscaling, and load balancing. From the management perspective, the most important motivations are those that relate to root cause detection (anomaly detection, failure prediction, fault management, dependency analysis), SLA management (including load, availability, QoS, and violations), as well as end-to-end management (covering performance, response time, latency analysis, and end-to-end tracing). Finally, observability is motivated by the distributed nature of the application, integration with many technologies, autonomic features, heterogeneity, and interoperability, as well as testing concerns such as performance and efficiency.
- RQ2: Which research areas are addressed? – In this respect, we observed that the main research areas are traditionally cloud and distributed computing, but new areas, including IoT, Edge, and Fog Computing, have also appeared. An important observation is the relation of Cloud-native observability studies to big data, security, and ML/AI topics.

- RQ3: How are observation approaches implemented? – Looking at the implementation landscape, we see that some solutions are most popular in research. Most notable are Prometheus as a monitoring system, Fluentd as a logging system, Jaeger as a tracing system, and Grafana as a visualization system. As a general-purpose observability stack, Elastic is the most popular. These often couple with Docker, Kubernetes, and OpenStack as an alternative to standard public clouds.
- RQ4: What are the recommended future trends in CNApps observability research? – Based on our research, we identified the following gaps in requirements and research areas covered. First and foremost, in the category of CNApp execution, scheduling has not been widely addressed. Second, in management category, policy-based approaches haven't been covered much. Third, from non-functional requirements concerns, we observed that the self-* aspects of applications and systems are not widely explored. Fourth, testing, security, compliance, and governance have not received much attention. Finally, we also expect that observability would cover ML/AI, visualization, and security.

One of the significant notions not addressed in this paper is Serverless Computing, which can be closely related to cloud and microservice architectures. We can expect that with increasing interest in serverless approaches and in the light of converging technological solutions (e.g., Knative serverless frameworks based on Kubernetes), the observability of serverless computing and applications will become a compelling research area. Another general trend we observe in computing is the expansive growth of data-driven methods for data analysis based on the progress in ML/AI techniques. This trend will influence the observability domain, as we expect the observability data to be input into ML models, which can drive autonomous decisions of computing systems and applications. It will require more quality data and careful development of models. Also, we can expect that the observability methods and frameworks for Cloud-native applications will enable new research areas in performance evaluation, benchmarking, and modeling. To enable this, the information gathered during observations must be exploited by the systematic experimental studies of real applications and systems in a Cloud-native style.

ACKNOWLEDGMENT

The authors are grateful to the people who conducted the reviews of the publications that formed the basis for the research. The reviewers were affiliated with AGH University or Sano Centre.

REFERENCES

[1] B. Scholl, T. Swanson, and P. Jausovec, *Cloud Native: Using Containers, Functions, and Data to Build Next-generation Applications* (System Administration). Sebastopol, CA, USA: O'Reilly Media, 2019.

[2] C. Davis, *Cloud Native Patterns: Designing Change-Tolerant Software*. Shelter Island, NY, USA: Manning Publications, 2019.

[3] N. Kratzke and P.-C. Quint, "Understanding cloud-native applications after 10 years of cloud computing—A systematic mapping study," *J. Syst. Softw.*, vol. 126, pp. 1–16, Apr. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121217300018

[4] (Mar. 2021). *The Twelve Factor App*. [Online]. Available: https://12factor.net

[5] M. Senapathi, J. Buchan, and H. Osman, "DevOps capabilities, practices, and challenges: Insights from a case study," in *Proc. 22nd Int. Conf. Eval. Assessment Softw. Eng.* New York, NY, USA: Association for Computing Machinery, 2018, pp. 57–67, doi: 10.1145/3210459.3210465.

[6] J. Wettinger, V. Andrikopoulos, F. Leymann, and S. Strauch, "Middleware-oriented deployment automation for cloud applications," *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 1054–1066, Oct. 2018.

[7] P. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*, 1st ed. Reading, MA, USA: Addison-Wesley Professional, 2007.

[8] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Reading, MA, USA: Addison-Wesley Professional, 2010.

[9] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

[10] S. Jain, *Linux Containers and Virtualization: A Kernel Perspective*. New York, NY, USA: Apress, 2020.

[11] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: A state-of-the-art review," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 677–692, Jul. 2019.

[12] S. Daya, N. Van Duy, K. Eati, C. Ferreira, D. Glozic, V. Gucer, M. Gupta, S. Joshi, V. Lampkin, and M. Martins, *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. Armonk, NY, USA: IBM Redbooks, 2016.

[13] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May 2016.

[14] (Jan. 2021). *Cloud Native Computing Foundation*. [Online]. Available: https://www.cncf.io

[15] (Mar. 2021). *Cloud Native LandScape*. [Online]. Available: https://github.com/cncf/landscape

[16] R. E. Kalman, "On the general theory of control systems," *IRE Trans. Autom. Control*, vol. 4, no. 3, p. 110, Dec. 1959.

[17] J. Kosińska and K. Zieliński, "Autonomic management framework for cloud-native applications," *J. Grid Comput.*, vol. 18, no. 4, pp. 779–796, Dec. 2020.

[18] N. Kratzke, "Cloud-native observability: The many-faceted benefits of structured and unified logging—A multi-case study," *Future Internet*, vol. 14, no. 10, p. 274, 2022. [Online]. Available: https://www.mdpi.com/1999-5903/14/10/274

[19] N. Marie-Magdelaine, "Observability and resources managements in cloud-native environnements," Ph.D. dissertation, Laboratoire Bordelais de Recherche en Informatique, Université de Bordeaux, Bordeaux, France, Nov. 2021. [Online]. Available: https://theses.hal.science/tel-03486157

[20] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Inf. Softw. Technol.*, vol. 64, pp. 1–18, Aug. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584915000646

[21] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Softw. Eng. Group School Comput. Sci. Math., Keele Univ., Dept. Comput. Sci., Univ. Durham, Durham, U.K., Tech. Rep. EBSE 2007-001, 2007. [Online]. Available: https://ebse.webspace.durham.ac.uk/wp-content/uploads/sites/49/2022/08/Systematic-reviews-5-8.pdf

[22] (Jan. 2023). *OpenTelemetry—High-Quality, Ubiquitous, and Portable Telemetry to Enable Effective Observability*. [Online]. Available: https://opentelemetry.io

[23] C. Majors, L. Fong-Jones, and G. Miranda, *Observability Engineering: Achieving Production Excellence*. Sebastopol, CA, USA: O'Reilly Media, 2022.

[24] A. D. Malony, "Performance observability," Ph.D. dissertation, Dept. Comput. Sci., Univ. Illinois at Urbana-Champaign, Chicago, IL, USA, 1990.

[25] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003, doi: 10.1109/MC.2003.1160055.

[26] I. Odun-Ayo, R. Goddy-Worlu, L. Ajayi, B. Edosomwan, and F. Okezie, "A systematic mapping study of cloud-native application design and engineering," *J. Phys., Conf. Ser.*, vol. 1378, no. 3, Dec. 2019, Art. no. 032092, doi: 10.1088/1742-6596/1378/3/032092.

[27] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: A systematic mapping study," in *Closer*. Setúbal: SciTePress, 2018.

[28] V. Yussupov, U. Breitenbücher, F. Leymann, and M. Wurster, "A systematic mapping study on engineering function-as-a-service platforms and tools," in *Proc. 12th IEEE/ACM Int. Conf. Utility Cloud Comput.* New York, NY, USA: Association for Computing Machinery, Dec. 2019, pp. 229–240, doi: 10.1145/3344341.3368803.

[29] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *J. Syst. Softw.*, vol. 150, pp. 77–97, Apr. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219300019

[30] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: A survey," *Comput. Netw.*, vol. 57, no. 9, pp. 2093–2115, Jun. 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/1389128613001084

[31] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A survey of cloud monitoring tools: Taxonomy, capabilities and objectives," *J. Parallel Distrib. Comput.*, vol. 74, no. 10, pp. 2918–2933, Oct. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731514001099

[32] J. S. Ward and A. Barker, "Observing the clouds: A survey and taxonomy of cloud monitoring," *J. Cloud Comput.*, vol. 3, no. 1, p. 24, Dec. 2014, doi: 10.1186/s13677-014-0024-2.

[33] C. B. Hauser and S. Wesner, "Reviewing cloud monitoring: Towards cloud resource profiling," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 678–685.

[34] R. Brondolin and M. D. Santambrogio, "A black-box monitoring approach to measure microservices runtime performance," *ACM Trans. Archit. Code Optim.*, vol. 17, no. 4, pp. 1–26, Dec. 2020.

[35] A. Bento, J. Correia, J. Duraes, J. Soares, L. Ribeiro, A. Ferreira, R. Carreira, F. Araujo, and R. Barbosa, "A layered framework for root cause diagnosis of microservices," in *Proc. IEEE 20th Int. Symp. Netw. Comput. Appl. (NCA)*, Jan. 2021, pp. 1–8.

[36] A. De Iasio, A. Furno, L. Goglia, and E. Zimeo, "A microservices platform for monitoring and analysis of IoT traffic data in smart cities," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2019, pp. 5223–5232.

[37] J. Bjørgeengen, "A multitenant container platform with OKD, harbor registry and ELK," in *Proc. High Perform. Comput., ISC High Perform. Int. Workshops*. Frankfurt, Germany: Springer, 2019, pp. 69–79.

[38] C. Liu, Z. Cai, B. Wang, Z. Tang, and J. Liu, "A protocol-independent container network observability analysis system based on eBPF," in *Proc. IEEE 26th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2020, pp. 697–702.

[39] M. Mekki, S. Arora, and A. Ksentini, "A scalable monitoring framework for network slicing in 5G and beyond mobile networks," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 1, pp. 413–423, Mar. 2022.

[40] B. Chen and Z. M. Jiang, "A survey of software log instrumentation," *ACM Comput. Surveys*, vol. 54, no. 4, pp. 1–34, May 2022.

[41] M. Usman, S. Ferlin, A. Brunstrom, and J. Taheri, "A survey on observability of distributed edge & container-based microservices," *IEEE Access*, vol. 10, pp. 86904–86919, 2022.

[42] C. Cassé, P. Berthou, P. Owezarski, and S. Josset, "A tracing based model to identify bottlenecks in physically distributed applications," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2022, pp. 226–231.

[43] H. Wu, Z. Cai, Y. Lei, J. Xu, and R. Buyya, "Adaptive processing rate based container provisioning for meshed micro-services in kubernetes clouds," *CCF Trans. High Perform. Comput.*, vol. 4, no. 2, pp. 165–181, Jun. 2022.

[44] C. Benzaïd, T. Taleb, and J. Song, "AI-based autonomic and scalable security management architecture for secure network slicing in B5G," *IEEE Netw.*, vol. 36, no. 6, pp. 165–174, Nov. 2022.

[45] A. Di Stefano, A. Di Stefano, and G. Morana, "Ananke: A framework for cloud-native applications smart orchestration," in *Proc. IEEE 29th Int. Conf. Enabling Technol., Infrastructure Collaborative Enterprises (WETICE)*, Sep. 2020, pp. 82–87.

[46] S. Jacob, Y. Qiao, Y. Ye, and B. Lee, "Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks," *Comput. Secur.*, vol. 118, Jul. 2022, Art. no. 102728.

[47] A. Bento, J. Correia, R. Filipe, F. Araujo, and J. Cardoso, "Automated analysis of distributed tracing: Challenges and research directions," *J. Grid Comput.*, vol. 19, no. 1, pp. 1–15, Mar. 2021.

[48] D. Zhang and M. Zheng, "Benchmarking for observability: The case of diagnosing storage failures," *BenchCouncil Trans. Benchmarks, Standards Evaluations*, vol. 1, no. 1, Oct. 2021, Art. no. 100006.

[49] J. S. Nunes, S. C. Sampaio, R. S. Malaquias, and I. de Morais Barroca Filho, "Deploying the observability of the SigSaude system using service mesh," in *Proc. 20th Int. Conf. Comput. Sci. Appl. (ICCSA)*, 2020, pp. 9–15.

[50] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," *J. Syst. Softw.*, vol. 182, Dec. 2021, Art. no. 111061.

[51] L. Meng, F. Ji, Y. Sun, and T. Wang, "Detecting anomalies in microservices with execution trace comparison," *Future Gener. Comput. Syst.*, vol. 116, pp. 291–301, Mar. 2021.

[52] M. A. López-Peña, J. Díaz, J. E. Pérez, and H. Humanes, "DevOps for IoT systems: Fast and continuous monitoring feedback of system availability," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10695–10707, Oct. 2020.

[53] R. Kang, Z. Zhou, J. Liu, Z. Zhou, and S. Xu, "Distributed monitoring system for microservices-based IoT middleware system," in *Proc. 4th Int. Conf. (ICCCS)*, Haikou, China: Springer, Jun. 2018, pp. 467–477.

[54] V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, "Does migrating a monolithic system to microservices decrease the technical debt?" *J. Syst. Softw.*, vol. 169, Nov. 2020, Art. no. 110710.

[55] A. Islam, A. Kumar, K. Mohiuddin, S. Yasmin, M. A. Khaleel, and M. R. Hussain, "Efficient resourceful mobile cloud architecture (mRARSA) for resource demanding applications," *J. Cloud Comput.*, vol. 9, no. 1, pp. 1–21, Dec. 2020.

[56] B. Li, X. Peng, Q. Xiang, H. Wang, T. Xie, J. Sun, and X. Liu, "Enjoy your observability: An industrial survey of microservice tracing and analysis," *Empirical Softw. Eng.*, vol. 27, no. 1, pp. 1–28, Jan. 2022.

[57] X. Guo, X. Peng, H. Wang, W. Li, H. Jiang, D. Ding, T. Xie, and L. Su, "Graph-based trace analysis for microservice architecture understanding and problem diagnosis," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2020, pp. 1387–1397.

[58] S. Lima, J. Correia, F. Araujo, and J. Cardoso, "Improving observability in event sourcing systems," *J. Syst. Softw.*, vol. 181, Nov. 2021, Art. no. 111015.

[59] H. Liu, J. Zhang, H. Shan, M. Li, Y. Chen, X. He, and X. Li, "JCallGraph: Tracing microservices in very large scale container cloud platforms," in *Proc. 12th Int. Conf., Held Part Services Conf. Fed. (SCF)*. San Diego, CA, USA: Springer, Jun. 2019, pp. 287–302.

[60] T. Weng, W. Yang, G. Yu, P. Chen, J. Cui, and C. Zhang, "Kmon: An in-kernel transparent monitoring system for microservice systems with eBPF," in *Proc. IEEE/ACM International Workshop Cloud Intelligence (CloudIntelligence)*, May 2021, pp. 25–30.

[61] B. Creane and A. Gupta, *Observability*. Sebastopol, CA, USA: O'Reilly Media, 2021.

[62] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proc. 2019 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 683–694.

[63] J. Rios, S. Jha, and L. Shwartz, "Localizing and explaining faults in microservices using distributed tracing," in *Proc. IEEE 15th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2022, pp. 489–499.

[64] R. Li, M. Du, Z. Wang, H. Chang, S. Mukherjee, and E. Eide, "LongTale: Toward automatic performance anomaly explanation in microservices," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.*, Apr. 2022, pp. 5–16.

[65] F. Lomio, S. Jurvansuu, and D. Taibi, "Metrics selection for load monitoring of service-oriented system," in *Proc. 5th Int. Workshop Mach. Learn. Techn. Softw. Quality Evol.*, Aug. 2021, pp. 31–36.

[66] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, and X. Li, "MicroRank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *Proc. Web Conf.*, Apr. 2021, pp. 3087–3098.

[67] M. Driss, D. Hasan, W. Boulila, and J. Ahmad, "Microservices in IoT security: Current solutions, research challenges, and future directions," *Proc. Comput. Sci.*, vol. 192, pp. 2385–2395, Jan. 2021.

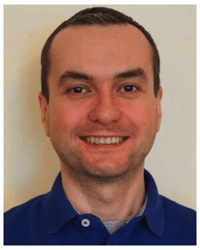
[68] M. Repetto and A. Carrega, "Monitoring network flows in containerized environments," in *Cybersecurity of Digital Service Chains: Challenges, Methodologies, and Tools*. Cham, Switzerland: Springer, 2022, pp. 32–55.
[69] A. Sojan, R. Rajan, and P. Kuvaja, "Monitoring solution for cloud-native DevSecOps," in *Proc. IEEE 6th Int. Conf. Smart Cloud (SmartCloud)*, Nov. 2021, pp. 125–131.
[70] K. Indrasiri and P. Siriwardena, *Microservices for the Enterprise*. Berkeley, CA, USA: Apress, 2018.
[71] S. R. Goniwada, *Observability*. Berkeley, CA, USA: Apress, 2022, pp. 661–676, doi: 10.1007/978-1-4842-7226-8_19.
[72] D. Ernst and S. Tai, "Offline trace generation for microservice observability," in *Proc. IEEE 25th Int. Enterprise Distrib. Object Comput. Workshop (EDOCW)*, Oct. 2021, pp. 308–317.
[73] S. Niedermaier, F. Koetter, A. Freymann, and S. Wagner, "On observability and monitoring of distributed systems—An industry interview study," in *Proc. 17th Int. Conf. (ICSOC)*. Toulouse, France: Springer, Oct. 2019, pp. 36–52.
[74] P. Liu and J. Guitart, "Performance characterization of containerization for HPC workloads on infiniband clusters: An empirical study," *Cluster Comput.*, vol. 25, pp. 847–868, Nov. 2022.
[75] L. Toka, G. Dobreff, D. Haja, and M. Szalay, "Predicting cloud-native application failures based on monitoring data of cloud infrastructure," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2021, pp. 842–847.
[76] A. Di Stefano, A. Di Stefano, G. Morana, and D. Zito, "Prometheus and AIOps for the orchestration of cloud-native applications in ananke," in *Proc. IEEE 30th Int. Conf. Enabling Technol., Infrastructure Collaborative Enterprises (WETICE)*, Oct. 2021, pp. 27–32.
[77] J. Han, Y. Hong, and J. Kim, "Refining microservices placement employing workload profiling over multiple kubernetes clusters," *IEEE Access*, vol. 8, pp. 192543–192556, 2020.
[78] D. Masouros, S. Xydis, and D. Soudris, "Rusty: Runtime interference-aware predictive monitoring for modern multi-tenant systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 184–198, Jan. 2021.
[79] A. Brito, C. Fetzer, S. Köpsell, P. Pietzuch, M. Pasin, P. Felber, K. Fonseca, M. Rosa, L. Gomes, R. Riella, C. Prado, L. F. Rust, D. E. Lucani, M. Sipos, L. Nagy, and M. Fehér, "Secure end-to-end processing of smart metering data," *J. Cloud Comput.*, vol. 8, no. 1, pp. 1–13, Dec. 2019.
[80] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service candidate identification from monolithic systems based on execution traces," *IEEE Trans. Softw. Eng.*, vol. 47, no. 5, pp. 987–1007, May 2021.
[81] D. Cha and Y. Kim, "Service mesh based distributed tracing system," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2021, pp. 1464–1466.
[82] G. Carcassi, J. Breen, L. Bryant, R. W. Gardner, S. McKee, and C. Weaver, "SLATE: Monitoring distributed kubernetes clusters," in *Proc. Pract. Exper. Adv. Res. Comput.*, 2020, pp. 19–25.
[83] P. Loreti, A. Mayer, P. Lungaroni, F. Lombardo, C. Scarpitta, G. Sidoretti, L. Bracciale, M. Ferrari, S. Salsano, A. Abdelsalam, R. Gandhi, and C. Filsfils, "SRv6-PM: A cloud-native architecture for performance monitoring of SRv6 networks," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 611–626, Mar. 2021.
[84] Z. Ye, P. Chen, and G. Yu, "T-rank: A lightweight spectrum based fault localization approach for microservice systems," in *Proc. IEEE/ACM 21st Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, May 2021, pp. 416–425.
[85] M. Scrocca, R. Tommasini, A. Margara, E. D. Valle, and S. Sakr, "The kaiju project: Enabling event-driven observability," in *Proc. 14th ACM Int. Conf. Distrib. Event-based Syst.*, Jul. 2020, pp. 85–96.
[86] S.-P. Ma, C.-Y. Fan, Y. Chuang, W.-T. Lee, S.-J. Lee, and N.-L. Hsueh, "Using service dependency graph to analyze and test microservices," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 2, Jun. 2018, pp. 81–86.
[87] J. Levin and T. A. Benson, "ViperProbe: Rethinking microservice observability with eBPF," in *Proc. IEEE 9th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2020, pp. 1–8.
[88] Z. Xiang, M. Höweler, D. You, M. Reisslein, and F. H. P. Fitzek, "X-MAN: A non-intrusive power manager for energy-adaptive cloud-native network functions," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 2, pp. 1017–1035, Jun. 2022.
[89] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: A proposal and a discussion," *Requirements Eng.*, vol. 11, no. 1, pp. 102–107, Mar. 2006.
[90] C. Ebert, "Improving engineering efficiency with PLM/ALM," *Softw. Syst. Model.*, vol. 12, no. 3, pp. 443–449, Jul. 2013.
[91] *A Practical Guide to the IBM Autonomic Computing Toolkit*, IBM, Armonk, NY, USA, 2004.

JOANNA KOSIŃSKA (Member, IEEE) received the Ph.D. degree in information and communication technology discipline.

Since 2020, she has been the CEO of Try IT Foundation. The foundation aims to promote the discipline of IT among girls and women. Since 2020, she has been the Director of the "Girls Go IT" Grant with the Institute of Computer Science, AGH University of Krakow, Poland, and Try IT Foundation. In 2022, she was a Visiting Researcher with the Sano Centre for Computational Medicine. The purpose of the internship was to find a new direction of research for computational medicine in the area of cloud-native computing and its observability. She is currently an Assistant Professor with the Institute of Computer Science, AGH University of Krakow. Her research interests include distributed computing, specifically cloud-native computing and resource management.

BARTOSZ BALIŚ received the Ph.D. and D.Sc. degrees in computer science. He is currently an Associate Professor with the Institute of Computer Science, AGH University of Krakow, and a Senior Researcher with the Sano Centre for Computational Medicine. He is also involved in the ALICE experiment with CERN. He has developed HyperFlow, a cloud-native scientific workflow management system. He has participated in national and EU-funded research projects CrossGrid, CoreGRID, K-Wf Grid, ViroLab, Gredia, PL-Grid, UrbanFlood, ISMOP, PaaSage, and WATERLINE. His research interests include scientific workflow management, cloud computing, cloud-native scientific computing, large-scale computing, and data engineering.

MAREK KONIECZNY (Member, IEEE) received the Master of Science degree from the AGH-UST, in 2002. He was a Software Engineer with Comarch and Nokia, from 2002 to 2011. Since 2011, he has been a Researcher. He is currently a Researcher with the Institute of Computer Science, AGH University of Krakow, Poland, in the field of distributed and networked systems. He is researching computer networks, especially software-defined networking (SDN), virtualization, and modern cloud-native architectures.



MACIEJ MALAWSKI received the M.Sc. degree in computer science and physics and the Ph.D. degree in computer science. From 2011 to 2012, he was a Postdoctoral Researcher with the University of Notre Dame, USA. He is currently the Director of the Sano Centre for Computational Medicine and an Associate Professor with the Institute of Computer Science, AGH University of Krakow. His research interests include parallel and distributed computing, large-scale data analysis, cloud technologies, resource management, and scientific applications, with a special focus on computational medicine.

SŁAWOMIR ZIELIŃSKI received the Ph.D. degree. He is currently an Assistant Professor with the Institute of Computer Science, AGH University of Krakow, Poland. His research interests include computer networking (with a focus on SDN and device programmability) as well as the design and development of distributed systems (with a focus on security). He is a CISSP.