IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 15 June 2022, accepted 15 July 2022, date of publication 21 July 2022, date of current version 24 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3193102

SURVEY

A Survey on Observability of Distributed Edge & Container-Based Microservices

MUHAMMAD USMAN[1], (Member, IEEE), SIMONE FERLIN[1,2], ANNA BRUNSTROM[1], (Member, IEEE), AND JAVID TAHERI[1], (Senior Member, IEEE)
[1]Department of Computer Science, Karlstad University, 65188 Karlstad, Sweden
[2]Red Hat, 16451 Stockholm, Sweden

Corresponding author: Anna Brunstrom (anna.brunstrom@kau.se)

This work was supported by the Knowledge Foundation of Sweden (KKS) through the Synergy Project AIDA—A Holistic AI-Driven Networking and Processing Framework for Industrial IoT under Grant Rek:20200067.

ABSTRACT Edge computing is proposed as a technical enabler for meeting emerging network technologies (such as 5G and Industrial Internet of Things), stringent application requirements and key performance indicators (KPIs). It aims to alleviate the problems associated with centralized cloud computing systems by placing computational resources to the network's edge, closer to the users. However, the complexity of distributed edge infrastructures grows when hosting containerized workloads as microservices, resulting in hard to detect and troubleshoot outages on critical use cases such as industrial automation processes. Observability aims to support operators in managing and operating complex distributed infrastructures and microservices architectures by instrumenting end-to-end runtime performance. To the best of our knowledge, no survey article has been recently proposed for distributed edge and containerized microservices observability. Thus, this article surveys and classifies state-of-the-art solutions from various communities. Besides surveying state-of-the-art, this article also discusses the observability concept, requirements, and design considerations. Finally, we discuss open research issues as well as future research directions that will inspire additional research in this area.

INDEX TERMS Automation, cloud-native, containers, DevOps, edge, Industrial Internet of Things (IIoT), microservices, monitoring, observability, operation, software-defined infrastructure, visibility.

I. INTRODUCTION

Cloud-native technologies [1] empower organizations to build and run applications in modern, scalable and dynamic distributed Information Technology (IT) infrastructures. This approach exemplifies containers, microservices, and immutable infrastructures. These techniques ultimately aim at leading to more robust, maintainable, and loosely coupled ecosystems. Combined with automation, these techniques also enable developers to make changes more frequently and predictably with minimal effort. The Cloud Native Computing Foundation (CNCF) [1] is ambitious to accelerate the adoption of this paradigm by maintaining an ecosystem of open-source, vendor-neutral projects.

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

The emergence of microservices architectures [2] together with the Development and Operations (DevOps) design philosophy [3] have made considerable changes to the way user applications are developed, deployed, and managed. Compared to a typical monolithic application, where the entire application is built as a single unified system, a microservices architecture decomposes the application into several independent executable pieces that coherently interoperate to provide specific application functionalities. Techniques like lightweight RESTful Application Programming Interfaces (APIs) [4] are extensively adopted to facilitate runtime communication among microservices.

As IT infrastructures continue to grow and become more distributed [5], enticed by a prospect of faster innovation, cost reduction, and boost agility, many businesses are migrating their traditional monolithic applications to microservices.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

For smooth execution, the requirement to monitor[1] these infrastructures at different levels of the software stack appears to be seamlessly crucial with the introduction of distributed environments and the transition to microservices.

Monitoring is by no means a new problem or new requirement [7]. Traditional monitoring tools, however, do fall short of meeting the challenges posed by operation monitoring demands of containerized microservices deployed over distributed IT infrastructures [8]. With the emergence of Industrial Internet of Things (IIoT), edge, and cloud combinations, where mission critical IIoT applications are hosted, introduce significant complexity for operators. In these setups, multi-site resource islands are connected via heterogeneous networks requiring several operation monitoring tools to manage aspects such as users, application data, tenants, nodes, or networks. Naive monitoring of bare-metal servers, virtual machine (VM) instances, or application container metrics without understanding the topological relationships between these various entities is incomplete for diverse operations (Ops) teams when troubleshooting performance issues [9].

Even though monitoring has served as a core function of IT for the last few decades, it started to prove itself limited due to various factors, including agile development methodologies, cloud-native deployments, and new DevOps practices. These factors have altered how the entire IT ecosystem (e.g., infrastructures, systems, and applications) should be monitored to respond promptly to incidents. For example, most edge infrastructures are orchestrated through *Kubernetes* [10], which is intended to abstract a lot of the underlying complexity. However, the platform itself consists of multiple independent projects as well as interconnected components. As such, simply lifting and applying traditional monitoring tools does not work out of the box for several reasons, e.g., several influencing components are hosted outside the containerized application, the underlying environment is more dynamic, and applications have more rapid deployment cycles. To this end, observability is an emerging set of practices combined with tools that goes beyond monitoring to provide insight into the internal state of a system by analyzing its external outputs [11]. The core of the observability is to quickly learn what is happening within the IT infrastructure to avoid extended outages, and during an outage, quickly enable so-called *root cause analysis* of the problem. For example, the Mean Time to Resolution (MTTR) metric measures outages, and the observability objective is to drive MTTR value as close to zero as possible.

Despite the systematic literature review on monitoring technologies, only a few studies such as [7], [12], [13] survey this field. In addition, no recent studies have surveyed latest distributed IT systems and microservices observability-related ramifications. Furthermore, considerable developments have occurred since then [2], [14], [15], and several aspects of distributed IT systems and application patterns have revamped. Thus, to better understand and tackle the observability problem, we first survey relevant works for modern distributed IT infrastructures and microservices observability. Next, we detail various observability principles to highlight fundamental requirements and challenges in this domain. Finally, we present open issues and challenges in the observability research domain. In summary, the key contributions of this paper are as follows:

- We provide a detailed review of state-of-the-art monitoring and observability contributions available in recent literature from both academic and industrial communities for delivering monitoring in distributed IT systems that are mostly realized via Linux-based systems and hosts microservices.
- Based on the literature survey, we extract key requirements and define fundamental characteristics that an operable observability system should exhibit.
- Highlight open issues and future research directions to promote further research in this exciting area of observability.

The rest of this paper is structured in line with these contributions. We first provide the background knowledge on distributed systems and application architecture patterns, highlight differences between monitoring and observability, and investigate related work (Section II). Next, we review state-of-the-art in observability landscape (Section III). After that, we outline requirements for realizing an observability system (Section IV) followed by details about open issues and challenges in this area (Section V). Finally, we present our final thoughts on the findings (Section VI).

II. BACKGROUND AND RELATED WORK

Establishing end-to-end runtime performance insights for today's dynamic, complex distributed IT environments is critical for guaranteeing containerized microservices agreed-upon Service Level Agreements (SLAs) [16]. However, the terms monitoring and observability are often mixed up, even though they are complementary concepts. In short, the term observability got significant traction in software engineering around 2018, as a natural evolution of monitoring practices. This section details how distributed IT systems have evolved (Section II-A), how application patterns have changed driven by the possibilities offered in the underlying distributed infrastructure (Section II-B), the fundamental differences between monitoring and observability (Section II-C), the complexities associated with traditional monitoring approaches (Section II-D), and the related surveys that are available in the literature (Section II-E).

A. DISTRIBUTED IT SYSTEMS EVOLUTION

A distributed infrastructure consists of numerous components installed on disparate networked computers and communicate and coordinate by passing messages. The need to achieve excellent Quality of Service (QoS) to facilitate good Quality of Experience (QoE) is one of the notable factors

[1]Monitoring is a method of gaining visibility into different parts of an end-to-end system to diagnose different types of operational issues [6].

that has brought substantial evolution in the distributed infrastructures [14]. An overall trend to illustrate how computing systems have evolved over time is depicted in Fig. 1.

Cloud computing [17], [18] is one such big paradigm that gained traction in the mid-2000s with the introduction of the Amazon Web Services (AWS). It has since picked the interest of both service providers and consumers, owing to its pay-per-use service model, which eliminates the upfront cost of purchasing physical hardware while also masking network connectivity complexity among other factors that make applications hosted on these systems more resilient and easily managed. It offers on-demand IT resources by dynamically scaling its provisioning power. The unprecedented development of diverse applications and increasing smart mobile devices for supporting IoT has lately posed substantial constraints on the centralized paradigm of cloud computing in terms of latency, bandwidth, and connectivity. To address centralized cloud computing constraints, research interests are shifting towards distributed paradigms [19].

Edge computing [15] is a good example of a decentralized paradigm. It aims to move data processing to the network's edge, near where data is generated and also consumed. This approach leverages an edge device computing power to process data without sending it to a private datacenter or a cloud service provider's network. Conceptually, edge computing focuses on running several services at the network edge to overcome the associated limitations of centralized datacentres where cloud computing services are hosted. Therefore, it is more than just a cloud extension and also functions as an integration platform for cloud and a multitude of services to facilitate and ensure successful system interaction [20].

The cloud computing era introduced application developers to the concept of cloud-native [21], a model in which developers are more concerned with how applications are created, deployed, and maintained at hyper-scale rather than where they are physically deployed. As a result, application developers are able to fully concentrate on continuous, agile software delivery with a ground assumption that infrastructure is always present. At the edge, cloud-native extends this basic assumption to enable agile development regardless of network topology, geography, or hardware diversity. It also offers a similar developer experience as a cloud datacenter, by supporting an incredibly diverse, highly dispersed edge environment and automatically accounting for new conditions without changing how developers work.

These paradigms nevertheless require further research due to the required resource management and control that demands the massive traffic to be supported by the network. Edge nodes, for example, are often equipped with limited computational and storage capacity, which makes them unsuitable for supporting large-scale user requests. On the other hand, cloud resources are often located far from consumers, making cloud servers unable to offer services requiring low latency. A rapid development of these technological paradigms are also leading to an explosion of new applications and services, putting a strain on what Ops teams are able to monitor and control.

B. APPLICATION ARCHITECTURE PATTERNS EVOLUTION

In recent years, the software architecture landscape has also significantly changed. Software application architecture refers to the fundamental structures of a software system and the discipline of creating such structures and systems [22]. A principal shift is the breakdown of large monolithic and coarse-grained applications into fine-grained deployment units named microservices, communicating predominantly by way of synchronous Representational State Transfer (REST) and asynchronous events [23].

1) STATIC MONOLITHIC ARCHITECTURE

Monolithic architecture is regarded as a traditional method of developing applications. A monolithic application tightly integrates three disparate functional components into a single deployable unit:

- A *database* is comprised of many tables typically in a relational database management system.
- A *client-side User Interface (UI)* typically consisting of HyperText Markup Language (HTML) pages and JavaScript running in a web browser.
- A *server-side application* mostly handling Hypertext Transfer Protocol (HTTP) requests, executing domain-specific logic, retrieving data from the database, and populating, e.g., the HTML views.

In summary, ''a single logical executable'' is what makes a monolith architecture. To make any changes to such a system, a software developer must build and deploy an updated version of the server-side application.

2) ELASTIC MICROSERVICES ARCHITECTURE

Microservices is an architectural approach in which an application is structured as a collection of highly maintainable and testable services that are loosely coupled, independently deployable, organized around business capabilities, and managed by a small Ops team.

The advantages of this architecture are numerous, but drawbacks are equally apparent: Aspects of formerly software development that were routine, including debugging, profiling, and performance management, are now orders of magnitude more complex to realise. For example, software built as a monolith makes the task of generating and collecting observability data such as logs, or traces part of the application developer's responsibility. In a microservice architecture however the application and infrastructure domains are commonly responsibility of different teams, often with limited access and understanding of other areas. For instance, the Ops team does not have access to how the application generates its performance-related data. The application can still generate data, but its collection in a certain backend as well as collection of data from the infrastructure running

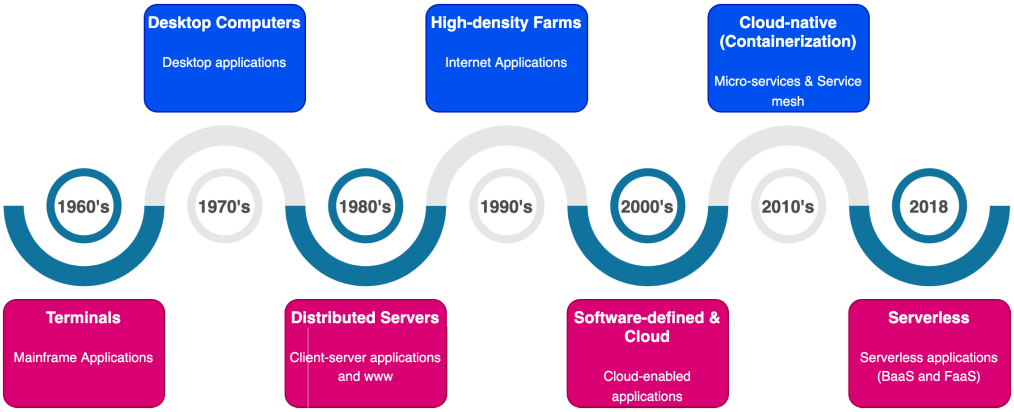

FIGURE 1. IT technologies have experienced a rapid evolution as well as significant form-factor changes.

the application, are responsibilities of a different, e.g., Ops team. Furthermore, in an extensive system, components will suffer minor outages at any given time, potentially affecting a subset of users, typically without the operator's knowledge.

Thus, as developers split applications into smaller units called microservices, ship them in containers across distributed cloud providers, and redeploy them continuously under the watchful eye of the DevOps team, the demand for fine-grained observability becomes more critical. The methodologies for designing and maintaining distributed systems continuously improve, making observability into our services and infrastructure more essential than ever for smooth operations.

C. DIFFERENCE BETWEEN MONITORING AND OBSERVABILITY

Due to this rapid evolution in distributed IT systems and application patterns, several new terminologies and approaches have surfaced in the operations management domain. One critical new concept is *observability* that is defined as *"the ability to measure the internal state of a system only by its external outputs"*. These external outputs are primarily known as telemetry data such as logs, traces and metrics, for a distributed system like microservices. It contains information such as the resource consumption of a machine, different log-level data generated by the applications running on a machine, and several others. Observability provides high-level overviews of the system's health and granular insights into the system's implicit failure modes. Furthermore, an observable system provides ample context about its inner workings, allowing deeper, systemic faults to be discovered.

In DevOps, it is frequent to hear about monitoring with observability. They both aim to maintain the system's reliability, whereas they have a subtle difference and, in fact, an association between them. In short, observability is not a replacement for monitoring, nor does it eliminate the need for monitoring; they work together.

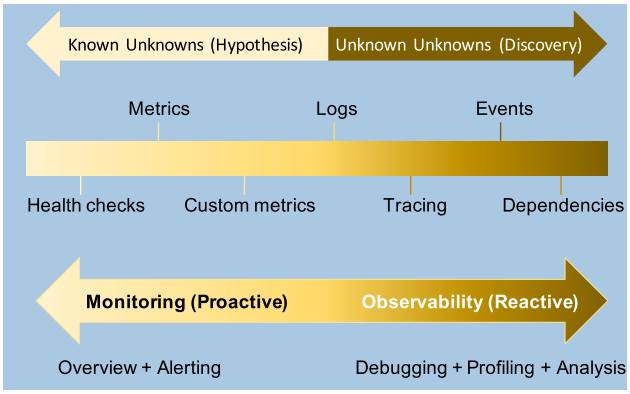

FIGURE 2. Observability vs monitoring. Monitoring is about testing hypotheses and observability is exploring new discoveries.

While monitoring is a process of keeping track of a system's health, using a predefined set of metrics and logs, it implies that we are looking for a specific collection of failures. However, in a distributed system, many changes are dynamic, also happening in parallel, and can occur regularly. As a result, we end up with issues that do not fall into the category of the issues that we were looking for. Thus, the monitoring system may miss them. To implement observability, monitoring is a prerequisite. Monitoring alerts operators about operational failures, whereas observability assists in determining where and why the failure occurred and what triggered the issue. Fig. 2 depicts the difference between the scope of both terms.

In other words, one can say that observability also permits so-called *Whitebox* monitoring [24], i.e., monitoring in which the design, code internals, and application structure are all known, the application reports about its inside. On the other hand, simple monitoring solutions allow for so-called *Blackbox* monitoring [24] and metrics collection. The tester does not know the application's internal details nor structure, it observes it from the outside. This type of monitoring is useful identifying symptoms of a problem but it does not help finding about the original triggers, e.g., error rate is up or the server cannot be reached.

D. COMPLEXITY OF TRADITIONAL MONITORING APPROACHES

Modern IT systems have become increasingly complex in four distinct ways (i.e., modular, distributed, dynamic, and ephemeral). This composite complexity gradient presents a significant challenge to conventional monitoring technology.

These four dimensions of complexity mean that IT systems are being composed of a growing number of ever more differentiated, autonomous components. In the past, monitoring technology has been tailored according to the nature of the monitored components. Since networks, servers, storage, and applications were purchased and implemented independently, the monitoring technologies also fragmented into tools for network monitoring, infrastructure monitoring, storage monitoring, and application monitoring. This specialization was motivated by two factors. First, each component type produced unique data types in distinct situations. As a result, it was logical to tailor data ingestion technologies to the context in which data was collected. Second, the components interacted with one another infrequently and in predictable ways. Therefore, integrating observations from one component with observations from another was not required. Meaning root cause analysis rarely required looking anywhere else for troubleshooting.

Now imagine the hyper-scale and complexity of today's IT systems. The modern IT infrastructure concept has weakened the original motivation for specialization. The quantity of diverse components has just outgrown the ability to monitor them individually. Architectural layers have been multiplied, but the function and behavior of components within each architectural layer (e.g., containers or not) have also changed radically. In particular, the principles for interpreting the self-describing data generated by one component cannot be applied to another.

Monitoring is a relatively deterministic problem in static, monolithic applications. In a monolithic architecture, an application is frequently in one of two states (up or down) that directly impact customer experience. For instance, Ops teams managing monolithic applications, often, set up Nagios [25] checks to acquire insights into the application performance. They can access logs from a single log file from the application hosting server to troubleshoot. Furthermore, applications are typically deployed over single runtime such as the Java Virtual Machine (JVM), enabling rich telemetry down to the application code level (e.g., active threads).

Nevertheless, in microservices, the interconnections of diverse components have become significantly more complex because of multiplication and differentiation. The interdependence between components have surprisingly grown stronger. This fact means the root cause of a performance glitch in one component often originates from a state alteration of one or more other components. A specialized, component-specific monitoring context would produce only local information. This data would not be sufficient for an Ops team to pinpoint the cause of a problem, let alone anticipate its reappearance in the future.

E. RELATED WORK

To the best of our knowledge, there is no other survey covering the same scope as ours, however, there are a number of monitoring surveys available [7], [26]–[29]. The remainder of this section discusses the selected surveys that are closely associated with our study.

In [7], Aceto *et al.* reviewed the essential characteristics of cloud monitoring systems and indicated open issues. Authors assessed the need for robustness, scalability, and flexibility. Despite the fact they cover important monitoring issues, it is outdated considering recent advances in distributed systems. Rodrigo *et al.* discussed enterprise system resource and application monitoring from performance and architectural perspectives [30]. The authors concentrate on topics such as monitoring metrics, resource scheduling, and new trends such as cloud elasticity and load prediction algorithms. In comparison to these studies, we survey high-level and low-level telemetry solutions from various communities to enable an operable observability system design for futuristic distributed systems.

Hassan *et al.* [13] presented a brief overview of the various tools and automated solutions used for monitoring cloud applications and discussed a thematic taxonomy for categorizing existing cloud monitoring solutions according to predefined criteria. Bula *et al.* [31] characterize various phases of cloud monitoring activities and present a comparative analysis of the state-of-the-art works of these phases. Both of these studies focus on cloud monitoring, whereas our study includes recent academic and industrial literature and covers a broader scope, including edge and microservices.

Niedermaier *et al.* [32] research the challenges, requirements, and current best practices in terms of monitoring and observability of distributed systems and possible solutions. They discovered that monitoring and observability of distributed systems is no longer only a technical issue instead it evolved into a more cross-cutting and strategic one, vital to the sustainability of service. Tamburi *et al.* [12] investigated three main aspects (a) monitoring practices and tools adopted in the industry, (b) the magnitude and complexity of industrial monitoring challenges, and (c) the role of system design in monitoring strategies to throw light on the state of monitoring practices in the industry. Both of these studies concentrated on the general and cultural aspects of monitoring in industry. In contrast, our research focuses on what solutions are available for a specific problem domain from both industry and academics.

III. STATE-OF-THE-ART ON OBSERVABILITY

Our research investigates the challenges, requirements, and current best practices and solutions in monitoring[2] and observability landscape. Because of the rapid evolution of distributed IT systems and shifts in application design patterns, as explained through Section II, we discern that the

[2]We use the term monitoring because it is still used in most academic studies, whereas the term observability is more commonly used in industry.

majority of related work in the monitoring and observability landscape becomes quickly obsolete. Therefore, we focus on surveying papers primarily published after the year 2015.

We identified relevant literature by querying well-known scholarly databases for the key terms *monitoring*, *observability*, *distributed systems (cloud and edge)*, and *microservices* with different combinations. The most relevant returned articles were then carefully read and critically analyzed. The scholarly databases that we queried involved: Google Scholar, IEEE Xplore, ScienceDirect, and ACM Digital Library. In addition, we investigated online materials from various open-source and commercial monitoring/observability vendors to develop a better understanding of their product offerings and capabilities.

This section subsequently provides a brief review of state-of-the-art from both academia (Section III-A) and industry (Section III-B), highlighting particular features of proposed solutions. Also, a detailed comparison is provided in Table 1, where the Table 1 columns are described as follows. The *scope* of work determines the type of distributed environment and application architecture for which the monitoring work is proposed. *Multi-tenancy* refers to support for independently monitoring different tenants' resources and applications. *Multi-layer* monitoring covers at least two out of: application performance, platform performance, system performance, and network performance. *Instruments* indicate the type of data taken into account for a particular monitoring system. The bare minimum for *open-source* is project hosting on GitHub or a similar platform. The rest of this section is organized based on contributing community and then on the scope column from Table 1.

A. CONTRIBUTIONS FROM ACADEMIA

Recent years have witnessed many efforts from academia in developing novel edge monitoring techniques. Edge Monitoring Framework for Multi-Cloud Environments (EMMCS) [33] monitors edge computing infrastructure and employs RESTful microservices. This framework uses Simple Network Management Protocol (SNMP) agents to gather metrics and handles all monitoring jobs at the edge of each cloud to boost network transmission and data processing at the central monitoring server. ZerOps4E [34] is an Artificial Intelligence for IT Operations (AIOps) platform applicable in heterogeneous, distributed environments. The overhead of ZerOps4E is assessed on edge devices, and performance trials on the applicability of platform showed encouraging results. Flexible monitoring solution at the edge (FMonE) [35] is proposed to allow monitoring workflows that comply with the demands of edge computing for gathering system-level performance metrics.

Lately, researchers have invested efforts in building novel cloud monitoring techniques. For private clouds, SmartX Multi-View Visibility Framework (MVF) [77]–[81] proposed to unify the multi-layer visibility of physical and virtual resources for timely detection and mitigation of various operational issues. In [36], an extension of SmartX MVF with flow-centric visibility is presented for the simultaneous monitoring of physical-virtual resources and flows classification for secure operation of private clouds. In addition, Ahmad *et al.* [82] presented a setup of distributed resources with cloud-native edge capabilities and a solution for persistently maintaining monitoring data collections in a variety of environments. Hababeh *et al.* proposed a practical multivariate control framework for monitoring cloud systems performance in [38]. It capitalizes cloud system performance metrics, gathers and renders them as graphs, stores the graphical data, and makes the data available on-demand without third-party software. Syed *et al.* [41] focused on collecting data from the host Operating System (OS) non-intrusively and linked data with the cloud controller via the proc filesystem (procfs) of the host OS. Then, this data was linked to information with the monitoring dashboard on the cloud controller node.

A few academic studies proposed monitoring solutions for multi-cloud systems. A multi-cloud strategy is one in which an organization uses two or more cloud computing platforms. Cross-Layer Multi-Cloud Application Monitoring and Benchmarking as-a-Service (CLAMBS) [37] introduced a solution for QoS monitoring and benchmarking of cloud applications deployed in multi-cloud environments. It can monitor and benchmark individual application components that are distributed across cloud layers and cloud providers. Multi-virtualization and Multi-cloud monitoring in cloud-based Cyber-PhysicAl systems (M2CPA) [39] keeps track of the performance of application components that are operating over diverse virtualization platforms and deployed across multiple clouds. Chhetri *et al.* [40] proposed CL-SLAM, a Cross-Layer SLA Monitoring Framework for Cloud Service-based Applications (CSBAs). It offers visibility into CSBA performance, visual analytics to establish correlations and interdependencies among cross-layer performance metrics, temporal characterization of CSBA performance, proactive monitoring detection, root-cause analysis of SLA breach, and support for reactive and proactive adaptation in support of quality assured CSBA provision. A couple of frameworks also focused on self-adaptive monitoring [42] and network-aware monitoring [43] for cloud environments.

Cloud datacenter network faults are hard to debug due to their scale as well as complexity, and some studies attempt to address such challenges. Kannan *et al.* [83] proposed a system, SyNDB, for transient faults that provide packet-level visibility, retrospection, and correlation. It uses data-plane time synchronization and data-plane storage (SRAM) to temporarily store packet records to aid in network debugging. SyNDB offers ability to examine the sequence of events preceding the occurrence of a network fault. Abranches *et al.* [84] developed a software-based network monitoring framework that significantly reduces network analytics resource consumption by consolidating tasks relevant to all applications and activating only when necessary. They identify Extended Berkeley Packet Filter (eBPF) and Express Data Path (XDP) as a natural fit for such and

TABLE 1. A detailed breakdown of the various monitoring works available in the literature.

Reference	Research Work	Scope	Multi-		Instruments			Open-source
			Tenant	Layer	Metrics	Logs	Tracing	
[33]	EMMCS	Edge	✗	✓	✓	✗	✗	✗
[34]	ZerOps4E	Edge	✗	✗	✓	✗	✗	✗
[35]	FMonE	Edge	✗	✓	✓	✗	✗	✗
[36]	SmartX MVF	Cloud	✓	✓	✓	✗	✗	✓
[37]	CLAMBS	Cloud	✗	✓	✓	✗	✗	✗
[38]	Monitoring cloud	Cloud	✗	✗	✓	✗	✗	✗
[39]	M2CPA	Cloud	✗	✓	✓	✗	✗	✗
[40]	CL-SLAM	Cloud	✗	✓	✓	✗	✗	✗
[41]	CloudProcMon	Cloud	✗	✓	✓	✗	✗	✗
[42]	Self-adaptive monitoring	Cloud	✗	✓	✓	✗	✗	✗
[43]	Network-aware monitoring	Cloud	✓	✗	✓	✗	✗	✗
[44]	Monasca	Cloud	✓	✓	✓	✗	✗	✓
[45]	Instana	Cloud	✳	✓	✓	✓	✓	✗
[46]	Splunk	Cloud	✳	✓	✓	✓	✓	✗
[47]	Scalyr	Cloud & Security	✳	✓	✗	✓	✗	✗
[48]	NetBeez	Cloud & Network	✳	✓	✓	✓	✗	✗
[49]	AdaM	IoT	✗	✓	✓	✗	✗	✗
[50]	IoT monitoring	IoT	✓	✗	✓	✗	✗	✗
[51]	CHARISMA	5G	✓	✓	✓	✗	✗	✗
[52]	Multi-site monitoring	5G	✗	✓	✓	✗	✗	✗
[53]	Grafana	Visualization	✓	✓	✓	✓	✓	✓
[54]	Kibana	Visualization	✓	✓	✓	✓	✗	✓
[55]	Black-box Monitoring	Microservices	✗	✓	✓	✗	✗	✓
[56]	Non-intrusive techniques	Microservices	✗	✓	✓	✓	✗	✗
[57]	M3	Microservices	✗	✓	✓	✗	✗	✗
[58]	milliScope	Microservices	✗	✓	✓	✓	✗	✗
[59]	QoE framework	Microservices	✓	✓	✓	✗	✗	✗
[60]	Cilium (Hubble)	Microservices	✓	✓	✓	✗	✗	✓
[61]	ViperProbe	Microservices	✗	✓	✓	✗	✗	✗
[62]	The rise of eBPF	Microservices	✗	✓	✓	✗	✗	✗
[63]	System visibility and security	Microservices	✗	✓	✓	✗	✗	✓
[64]	Container network observability	Microservices	✗	✓	✓	✗	✗	✗
[65]	Apache SkyWalking	Microservices	✓	✓	✓	✓	✓	✓
[66]	OpenTelemetry	Microservices	✓	✓	✓	✓	✓	✓
[67]	Consul	Microservices	✓	✓	✓	✓	✗	✓
[68]	Prometheus	Microservices	✓	✓	✓	✗	✗	✓
[69]	Jaeger	Microservices	✓	✓	✗	✗	✓	✓
[70]	Dynatrace	Microservices	✳	✓	✓	✓	✓	✗
[71]	Datadog	Microservices	✳	✓	✓	✓	✓	✗
[72]	New Relic	Microservices	✳	✓	✓	✓	✓	✗
[73]	Zebrium	Microservices	✳	✓	✓	✓	✗	✗
[74]	Sumo Logic	Microservices	✓	✓	✓	✓	✓	✗
[75]	SolarWinds	Microservices	✳	✓	✓	✓	✓	✗
[76]	Honeycomb	Microservices	✓	✓	✓	✓	✓	✗

(✓) Feature is supported. (✗) Feature is not supported. (✳) Limited or no information available.

build a system prototype on top of these technologies. Their findings show that combining conditional execution of analytics tasks with modern packet I/O reduces resource footprint of continuous network analytics. Zhang. *et al.* [85] describe a system named MimicNet for accurate performance estimates for large datacenters. Authors claim that it gives users the familiar abstraction of a packet-level simulation for a portion of the network, while leveraging redundancy and recent advances in machine learning to approximate slices of the network that are not instantly observable.

Academicians put forward a number of investigations focusing on IoT and 5G monitoring. Trihinas *et al.* introduced a lightweight, adaptive monitoring framework for smart IoT devices with limited processing capabilities [49]. This framework dynamically adjusts the monitoring intensity, based on a low-cost adaptive and probabilistic learning model capturing the progression and variability of the data stream at runtime. A framework for IoT system monitoring and management that combines AllJoyn open-source project (interconnecting IoT devices), MongoDB (Big Data storage), and Storm (real-time data analytics) is proposed in [50]. This work also highlighted how the proposed system helps addressing the limitations of AllJoyn in terms of large-scale smart environment monitoring and Big data storage and analytics. CHARISMA [51] discusses the key features of monitoring as well as the main requirements to resource monitoring systems for future 5G deployments and services. Furthermore, the major components of a generic architecture for monitoring, both physical and virtual resources, whether software-defined or legacy, are described and explored. Perez *et al.* [52] presented a monitoring architecture for the distribution and ingestion of metrics and KPIs for

5G multi-site platforms, where disparate verticals from different stakeholders are applied over a shared infrastructure. The authors also evaluated the performance of the publish-subscribe model to verify that it suited the provisions of these scenarios.

There are numerous research initiatives that attempted to realize microservices monitoring. A black-box monitoring approach to track microservices at scale, concentrating on architectural metrics, power consumption, application, and network performance is discussed in [55]. The solution claims to be transparent to applications and generates less overhead than state-of-the-art black-box systems. Cinque *et al.* [56] presented their proposal for a unique monitoring framework with non-intrusive techniques based on passive tracing and log analysis to address challenges in current application performance monitoring. A generic monitoring framework Multi-microservices Multi-virtualization Multi-cloud (M3) [57] is introduced to monitor the performance of microservices deployed over heterogeneous virtualization platforms in a multi-cloud environment. milliScope (mScope) [58] is a millisecond granularity software-based resource and event monitoring framework offering both low overhead at high frequency and high accuracy matching other available monitoring tools. Laghari *et al.* [59] conducted a study on the QoE framework for cloud computing to monitor end-users' video streaming services.

Discussions regarding performance-oriented solutions leveraging eBPF for microservices monitoring dominated research in recent years. ViperProbe [61] proposed a scalable eBPF-based dynamic and adaptive microservices metrics collection framework. Authors claim that ViperProbe can effectively decrease the set of gathered metrics by assessing the performance profile of microservice patterns before deployment, enhancing those metrics' efficiency and effectiveness. Cassagnes *et al.* [62] discussed lessons learned after using eBPF to monitor and profile performance. Authors in [86] and [63] showed how eBPF enabled the development of a new generation of runtime security monitoring tools that outperform legacy tools in terms of performance, context, and overall signal to noise ratio. A protocol-independent network monitoring at the kernel level for the Alibaba Kubernetes cluster is proposed in [64]. By non-intrusive collection of user application L7/L4 layers based on eBPF, authors claim a significant throughput per second can be attained without modifying the kernel nor the application.

B. CONTRIBUTIONS FROM INDUSTRY

Several vendors have emerged with acquisitions and targeted solutions developed in the observability space. For example, IBM acquisition includes Instana [45], and Splunk [46] acquisitions include cloud performance monitoring specialist Flowmill. The rest of this section categorizes monitoring and observability solutions from various vendors into two broad categories (Open Source Software and Closed Source Software) and briefly highlights key features.

1) OPEN SOURCE SOFTWARE (OSS) FROM INDUSTRY

The industry made significant investments in developing novel cloud monitoring solutions in recent years. Monasca [44] is an open-source monitoring-as-a-service multi-tenant, scalable, and fault-tolerant solution that integrates with OpenStack. It offers a RESTful API for high-speed metrics processing and querying as well as streaming alarm and notification engines. Another open-source project with broad traction is OpenTelemetry for gathering telemetry data from cloud-native applications and their infrastructure to monitor overall health and performance [66]. OpenTelemetry originated from a merger of OpenCensus and OpenTracing. This project aims at standardizing how telemetry data is produced and collected in distributed systems. It is a member of the CNCF and gaining widespread acceptance as an emerging industry standard for handling of observability data.

The industry has two cutting-edge solutions for monitoring data visualization that are open-source. First, Grafana Labs [53] has developed an open-source monitoring and observability platform. No matter where metrics are stored, Grafana enables query, view, alert, and understands their meaning quickly. Second, the ELK Stack [54] is a collection of three open-source products, namely Elasticsearch, Logstash, and Kibana. The ELK stack provides centralized logging to aid server or application problems discovery from one place.

There are also several open-source software focusing on microservices monitoring. Apache SkyWalking [65] is an open-source observability tool designed to facilitate operators to identify issues, receive critical alerts, and monitor system health. Consul [67] is a service mesh solution providing a full-featured control plane with service discovery, configuration, and segmentation functionality. Cilium [60] is the eBPF-based open-source solution to secure network connectivity among services that are deployed with Linux container management platforms such as Docker and Kubernetes [10]. Hubble is the low-level observability layer of Cilium for obtaining cluster-wide visibility into the communication and behavior of services and the networking infrastructure.

There are a number of open-source tools available looking into how to build your own observability solution from the ground up. In addition to the solutions discussed following this section, we acknowledge that there are numerous alternatives out in the market. Because it is unfeasible to cover all of them in this article, we share a link to the rest of the CNCF observability projects [87] that are at various stages of maturity (e.g., graduated, incubating, and sandbox).

Prometheus [88] is a CNCF project that offers a multi-dimensional time-series system for resources and services monitoring. It collects metrics from configured targets at predefined intervals, evaluates rule expressions, displays the results, and can send alerts. Lately, the ELK stack also extended its capabilities to gather metrics by providing Beats [89], lightweight data shippers and installed on servers to capture various types of operational data, e.g., Metricbeat.

Loki [90] is a Prometheus-inspired horizontally scalable, multi-tenant log aggregation system. It indexes a set of labels for each log stream rather than the contents of the logs. Fluentd [91] is an another open source data collector that unifies data collection and consumption. Logstash [54] is also a server-side pipeline that ingests data from multiple sources, transforms it, and sends it to the preferred stash.

Distributed tracing first proposed via Microsoft's Magpie [92] and X-Trace [93], which introduced some of the concepts known today as distributed tracing. Later, Google's Dapper [94] introduced the ability to sample events to reduce overhead and improve application-level transparency. Jaeger [69] is a popular distributed tracing implementation following OpenTracing. It provides a framework for distributed transaction monitoring and root-cause analysis.

2) CLOSED SOURCE SOFTWARE (CSS) FROM INDUSTRY

Several vendors begun to offer observability solutions with varying capabilities for cloud-based infrastructures. Instana [45] offers an observability platform with automated application monitoring. For Instana, it does not matter where applications are deployed, whether on-premises or in public or private clouds, including mobile devices. Splunk [46] is an infrastructure monitoring, troubleshooting, and incident response solution. All sample architecture of Splunk uses full-fidelity data for better results, and the real-time streaming feature analyzes all data as it arrives to reduce MTTR.

Public cloud providers have created their own extendable (via APIs, Software Development Kits (SDKs), and backends) monitoring tools supporting both platforms and applications: Microsoft Azure leverages its own Azure Monitor [95] system. It can collect, analyze, and act on telemetry data from Azure and on-premises environments. Azure Monitor can monitor a web application's availability, performance, and usage as well as analyze and optimize the performance of its underlying infrastructure. Amazon CloudWatch [96] is used to monitor Amazon Web Services (AWS) resources and applications deployed on AWS. It maintains a metrics repository for visibility into resource utilization, application performance, and operational health, i.e., it also covers the whole stack from application to the supporting infrastructure. Stackdriver was acquired by Google in 2014 and later renamed to Google Cloud Operations Suite [97]. It offers several components, covering several of the observability dimensions, such as Cloud Monitoring, Cloud Trace or Cloud Logging. Each of them offers a comprehensive view of system metrics and application logs or traces. For example, Cloud Monitoring collects system-level metrics whereas Cloud Logging collects log data from applications, and Cloud Trace collects distributed tracing data.

There are also several other solutions towards microservices monitoring. Dynatrace [70] established itself as a dependable Application Performance Monitoring (APM) solution provider. Dynatrace solution comes in two flavors: First full-SaaS (Software-as-a-Service) and second distributed-SaaS. Datadog [71] is a data observability service for cloud-based applications leveraging SaaS-based data analytics platform to monitor servers and applications using an agent and API calls. This agent runs inside the user's Kubernetes cluster to collect metrics, traces, and logs in real-time. New Relic One [72] is a cloud-based observability solution combining infrastructure, mobile, and native client monitoring, along with APM. Users can correlate cloud-based servers' performance to application logs and configuration changes with infrastructure monitoring.

A couple of other vendors focus on detecting and diagnosing microservices' operational troubles. Sumo Logic [74] is a multi-tenant, SaaS-based, cloud-native observability platform. Initially, it started as log management and big data analytics solution, which later incorporated metrics and tracing to expand the offering into a complete observability platform. Zebrium [73] is an AIOps observability platform using unsupervised machine learning to detect and diagnose software issues. There is no requirement to set up the system manually. It is ready to detect incidents after training on corporate topology and baseline the system. The system does not only assist end-users in determining the root cause of problems, but it also proactively discovers faults and delivers root cause reports.

Finally, there is a growing list of vendors, such as SolarWinds [75], Scalyr [47], NetBeez [48], and newcomer Honeycomb [76] attempting to provide off-the-shelf instrumentation and observability. However, negligible amounts of internal implementation details are available for these closed source solutions and platforms. This limitation leaves significant room for academia community to cover.

IV. REQUIREMENTS TO REALIZE OBSERVABILITY

It is critical to comprehend several challenges to realize an effective end-to-end runtime performance observability system. This section explains the major features of an observability system, based on the findings and guidelines from the literature survey. First of all, Section IV-A focuses on the core telemetry data types that must be measured, collected, and processed by an observability system about distributed IT infrastructures and microservices runtime performance. Most of the proposals in Section III are directed towards one or more of these data types. Our literature review shows that this data is also crucial for creating various signals to alert operators about potential failures or aid operational diagnostic processes. This is referred as golden signals. In addition, we ascertain some basic functionalities and characteristics that are essential for an observability system to process collected data and accelerate troubleshooting. These functions and characteristics are detailed in Sections IV-B and IV-C.

A. DATA REQUIRED FOR OBSERVABILITY SYSTEM

There are three main data types, often called three pillars of observability (logs, metrics, and traces), and tackled at various levels by most vendors delivering observability systems. It is important to highlight that these are not the only

types of telemetry data available, but they are certainly the most widespread when it comes to observability. Different APM vendors often add their differentiation to these types of telemetry data. For example, some vendors add user experience and others add events, while others list dependencies, arguing that knowing how each application depends on other components is essential for troubleshooting. Fig. 3 shows the potential data types that we can capture for an end-to-end runtime performance observability and remediation of the entire distributed system. Also, we point out that telemetry data from a single application is insufficient; the entire IT environment, including their connectivity points, should contribute telemetry data. When each application provides the correct information, operators can promptly determine where, what, and when an operational glitch occurred or it is about to start occurring. Following in this section, we discuss various types of data and measurements derived from them.

Logs are structured and unstructured lines of text that a system generates when a specific part of code executes. In general, a log is a record of an event within an application. Logs help uncover unpredictable and emergent behaviors displayed by components of microservices. They are easy to generate, they indicate different severity levels [98], e.g., warning or error logs and instrument because most application frameworks, languages, and libraries have built-in support for logging. Virtually every component of a distributed system produces logs and events at any point to offer detailed information about a system such as a fault and the precise time it occurred. By analyzing the logs, one can troubleshoot the code and figure out where and why an error occurred. Logs-based monitoring is focused in [56].

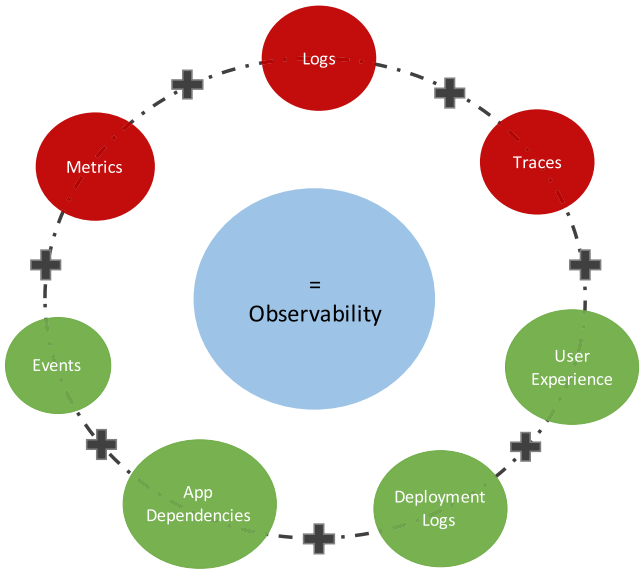

FIGURE 3. Three fundamental pillars of observability (red) plus other data considered vital for observability by different vendors (green).

Metrics are numerical representations of data that Ops teams use to determine the overall behavior of a system, service, or network component over time. A metric comprises a set of attributes (e.g., timestamp, name, label, and value) that convey information about SLAs, they can also be used to describe applications or system performance, e.g., in form of application-specific defined metrics. Unlike an event log, metrics are measured values derived from system runtime performance. They are real time-savers, because operators can correlate them across the entire infrastructure, from application to network components, to get a holistic view of system health and performance. Also, metrics data are easier to query and store. Application-level performance metrics may include the number of application instances, average response time, requests processed per second, etc. System-level performance metrics examples are system uptime, CPU usage, and memory utilization. Network-level performance metrics may involve latency, number of successful or terminated TCP connections, and HTTP requests, etc. Metrics-based monitoring is focused in [35]. Typically, these metrics are used to trigger alerts whenever a system value exceeds a certain threshold.

Traces depicts the entire journey of a request or action as it moves through various components of a distributed system. While it is agreed that logs and metrics effectively determine an individual system's behavior and performance, they are rarely helpful in understanding the lifecycle of a request in a distributed IT system, e.g., understand causality in a number of microservice calls. Operators apply the tracing technique to view and understand the entire lifecycle of a request across multiple systems. Traces make it possible to profile systems, primarily containerized applications and microservices. Operators can measure the overall system health by analyzing trace data, pinpointing bottlenecks, resolving performance issues faster, and prioritizing high-value areas for optimization. Trace-based monitoring is focused in [93].

1) MEASUREMENTS FROM TELEMETRY DATA

Information from the three types of telemetry data (logs, metrics, and traces) is combined to provide four Site Reliability Engineering (SRE) golden signals of observability (latency, traffic, errors, and saturation) [99]. These metrics focus on the end-to-end runtime performance of user requests and applications. In addition, these metrics also serve as a foundation for *Alerting* to inform the Ops teams when something is wrong, *troubleshooting* to help isolating and fixing the problem, and *Tuning and Capacity Planning* to assist in improving the setup over time.

Latency (request service time) tracks the time it takes for an application to process a particular request successfully. It is a measure used to identify performance issues in an application, e.g., a bottleneck microservice. When measuring latency, operators need to specify a threshold for a successful request, then monitor the results against latency failing requests. This way, they can quickly identify which services are underperforming, discover incidents more quickly, and timely respond to incidents.

Traffic (user demand) tracks the number of requests flowing through an application per unit of time. Depending on the application operators are tracking, traffic can have a variety of values. For example, they can utilize HTTP requests per second or bandwidth consumption to measure the traffic for a particular web application. One technique to monitor traffic in a distributed system is viewing the number of network conversations. By monitoring the traffic in an application, operators can observe how the application acts throughout a rise and plan for a spike in demand.

Errors (rate of failed requests) tracks the rate of failed requests in a system. The most common errors are explicit error (such as HTTP 500), implicit error (such as HTTP 200 success response that does not deliver the right content), and policy violation when the request takes longer than the agreed timeout interval. Because no system is 100% error-free, one must always track errors to determine if something is wrong. It is critical to report server and client errors separately when tracking those errors. That way, the Ops team can properly understand different error types and get to the root cause of specific problems much faster.

Saturation (overall system capacity) measures how much a server or network resource is being loaded. It is a metric that indicates how active a service is and is often used as an early indicator of system slowdowns and failures. System metrics such as CPU, disk space, and memory use are common signs of production system saturation. When measuring saturation, try to select metrics that limit performance of a system. For example, operators can use CPU load for processor-intensive applications and memory for memory-intensive applications. They set a utilization benchmark, as every service has a limit after which performance decreases.

To understand these concepts better, consider a simple example of the three pillars, how they interact to create the four golden signals, and how they differentiate observability from monitoring. Consider a server with a full disk. A basic monitoring tool will only show us a *Blackbox* view of an application, e.g., the error rate is up. It will only alert reactively one day that a particular disk is 95% full. Nevertheless, it provides no relevant meta-information: why did it fill up, how quickly it reached 95% and is this merely a temporary increase in disk utilization that will or will not reduce later?

Long before it reaches the alarming 95% threshold, an observability system can use its *Whitebox* Point of View (PoV) to proactively gather and present metrics showing a dashboard suggesting a significant week-over-week increase in disk utilization. With access to application and system logs, operators can pinpoint the exact program driving the spike. Finally, fine-grained tracing will reveal that the applications *disk write* activity abruptly spiked precisely five weeks ago when one of the changes during the upgrade was *verbose flag* set for all of the apps logging.

B. BASIC FUNCTIONALITIES FROM AN OBSERVABILITY SYSTEM

An observability system is more than just data exposure. Other functionalities that are considered crucial in delivering observability are broken down into three dimensions (i.e., referred as F*) and discussed below in details.

F1) Correlation: The key for observability tools is to identify a significant anomaly from a potential number of other anomalies and link that issue to other pertinent data from log files, traces, or metrics [42], [73], [79]. For example, a state-of-the-art observability system should gather, process, analyze, and provide findings in a second interval via the user interface. Then, by showing correlated information in the context of the anomaly observed, the user should quickly determine the potential root cause of a particular problem.

F2) Topology: As stated in the Section IV-A, some vendors also include dependency graphs in their list of required data. We agree that topology information is vital to understand the relationships between components in dynamic, distributed cloud-native environments [81], [100]. This process should be complemented by continual automatic detection of an application's components and continuous baselining of all data. For instance, proprietary tools deploy specialized agents to detect what is running inside the container or the JVM and automatically inject the correct instrumentation.

F3) Incident response: Finally, the observability system should handle issues without manual intervention through automatic remediation using machine learning [34], [101]. It needs to be generally more granular than in a traditional monitoring system. We list here three categories of events:

Change: An event signalling a change in the environment such as a configuration update, Pod start/stop, or deployment.

Issue: An event indicates a service or infrastructure component is in an unhealthy state, such as an insufficient number of Kubernetes replica sets or a high system CPU load.

Incident: A breach of a KPI on an edge service or an infrastructure issue is referred to as an incident. An incident occurs when user experience or service deteriorates. To offer context and facilitate root cause analysis, we link relevant issues and changes to the occurrence.

Based on these events, one can define an alert to notify changes, issues, or incidents to an external app, which can inform an Ops team or initiate an automated response [102].

C. KEY CHARACTERISTICS OF AN OBSERVABILITY SYSTEM

Observability system should be the Ops team's single source of truth as they troubleshoot, debug, and optimize the distributed IT environment. For example, a single unified view allowing connected context and performing accurate analytics can help operators detect and resolve operational issues quickly, based on infrastructure and application logs, distributed tracing, metrics, all the way to the end-user experience, without needing new tools or switching between them. Thus, defining the key system characteristics (i.e., referred as C*) is paramount before entering the design phase to deliver a maintainable observability system.

C1) Connected context: When operators access metrics about the health of one of their microservices, they should be able to observe how that service affects other services or components of the entire distributed system, as well as how those workloads are affected by the Kubernetes cluster that hosts them, and vice versa.

C2) Easier and faster exploration: Consider having all telemetry data in one view in near real-time from everywhere. That view should offer a pretty innovative design. Because for the Ops team to efficiently traverse large, complex, distributed systems and instantly comprehend and prioritize any performance issues, they will need intuitive visualizations that require zero-touch configuration.

C3) A single source of truth: It includes storing, alerting, and analyzing telemetry data using unified APIs, irrespective of data placement, which could be distributed across multiple nodes in an edge cluster. Usually, operators search for a platform capable of ingesting metrics, events, logs, and traces from diverse sources, including proprietary and open-source agents, APIs, and built-in instrumentation. However, data access via such unified APIs should be scalable enough to handle the ingest load on the busiest days.

C4) Allow capturing arbitrarily wide events: Complex distributed environments and microservices that must be up and operating at all times, or that must be fixed as rapidly as possible will require a large number of events that answer questions beyond ''is it broken?''. In such a case, metrics and logs should set up the core. Besides, adding traces should save effort when debugging various operational problems if the production environment consists of several intricate intertwined components.

C5) Decouple data sources from sinks: By decoupling sources and sinks, it should become easy to introduce or change tools and reroute data without impacting production systems. For instance, a messaging bus introduction into the observability system should completely decouple source and sink and no longer interact with each other directly.

V. DISCUSSION AND OPEN ISSUES

Research on observability is still in its early stages, primarily led by industry, and requires further development and experimental evaluation. The rest of this section discusses the compelling open issues in this domain.

A. INFRASTRUCTURE HETEROGENEITY

The most challenging aspect of edge performance observability is the emergence of new, more complex architectures ranging from containers, virtual machines, and clusters [103]. The heterogeneity in these areas results in lack of understanding of the overall system, its individual components, and the processed requests. More devices send data, which necessitates more data processing, resulting in more layers of computing at the edge. These layers are created and maintained by various development teams and operational teams. For instance, sensors on a factory floor may send data to a single gateway, which may then send it to the private or public cloud. Similarly, a single, smart garage may collect data from all the structure and process it at a single gateway before sending it to a single cloud application. Furthermore, systems usually contain both legacy and modern applications running in parallel, necessitating additional tooling to integrate legacy applications. To ensure that each layer is operative, an end-to-end view of computing service requests is required. However, the heterogeneity of the infrastructure and the rate of innovation puts observability systems to the test. Developers, for instance, can select the best technology for distributed systems, but technological heterogeneity then makes consistent use of observability systems challenging.

B. NEW DEMANDING USE CASES

The ability to track critical business transactions at the edge and correlate those requests with their impact on an organization's profit and loss will be the ultimate key to success in an IoT-dependent environment. In the case of consumer devices, a user's level of satisfaction with their new car, home entertainment system, and others may influence future purchasing decisions or lead to negative reviews. Factories will be able to predict and schedule maintenance to maximize production. As IoT-enabling microservices grow, so will the applications at the edge and the need to monitor the business logic that resides there. For better or worse, actions at the edge will increasingly influence business outcomes, determining whether SLAs are met or violated, user relationships are strengthened or strained, and new business is excelling or declining. These are all diverse application scenarios in which an observability system must timely support dynamic configurations and decision-making based on collected data [104], [105].

C. VAST AMOUNT OF DATA

An observability platform is expected to handle massive amounts of data. The management and timely processing of such massive amounts is a critical issue that necessitates additional research. The identified challenge could be prioritizing user alerts and drawing meaningful conclusions from collected metrics, logs, and traces data. Furthermore, identifying the location of faults and the responsibilities to fix them is extremely difficult for operational problems in which one request is being handled by multiple application components developed by different teams. Also, several complexities are encountered by operators when correlating metrics and timestamped logs from multiple services, which is frequently accompanied by insufficient metadata. Similarly, the combined use of system and distributed tracing to inspect what happens inside and across microservices remains an open research problem. Furthermore, defining appropriate measurement intervals/points and an ample data amount that is efficiently processable and analyzable by resource-constrained edge systems are also intriguing research topics.

D. ALL-IN-ONE OBSERVABILITY PLATFORM

Platforms that offer all observability functions as a single product are pivotal. However, survey in Section III reveals that such open solutions are mostly nonexistent. In the best-case scenario, the market offers solutions, providing basic monitoring functionalities (e.g., performance measurement or logging). Often one has to expand the existing tools by integrating other solutions. This can include new standards, technologies, and pre-existing solutions. Consequently, comprehensive solutions in this context represent various independent tools and technologies integrated together. To achieve a true all-encompassing observability solution, isolated solutions must be avoided or substituted with standard observability platforms, open standards, and modern technologies in the future. Open standards for instrumentation are the primary focus of projects such as OpenTelemetry [66]. However, more efforts on open standards and integrated technologies for data processing are worth further investigations.

E. SECURITY AND COMPLIANCE CONCERNS

One of the major benefits of hosting microservices over distributed systems is their accessibility from anywhere, allowing development teams and application users to connect regardless of their locality. Unfortunately, if observability system is not properly configured, many of the technologies with which users interact, such as APIs, are vulnerable to security attacks. Because of vulnerabilities, it is critical to use web application firewalls to ensure that all requests originate from legitimate traffic, ensuring that web applications and operations that rely on APIs are always protected. Furthermore, there is considerable concern among application providers and platform/infrastructure providers about what data can be shared and how to ensure application users' privacy is not compromised, the so-called data in-flight and data at-rest scenarios. In this aspect, new measures for secured data access and measures addressing data usage compliance, such as access control [106], must be investigated further considering the new stream of use cases.

F. ORGANIZATIONAL CULTURE AND INDIVIDUALS MINDSET CHANGE

An overly ambitious plan to replace all current tools with a new observability platform may not be the best strategy. Depending on the nature of the organization's current estate, many legacy applications are often not designed for the observability approach. An observability solution, on the other hand, should be aimed at the increasingly rapid and distributed cloud-native application deployments. These changing environments are more likely to employ DevOps practices, making automated data collection for observability a far advanced solution. It is also more likely that the developers will be able to incorporate the required instrumentation into their software, allowing it to be fully observable. To summarize, the three pillars do not automatically combine to achieve observability; people, processes, and system design primitives must also be aligned around a set of shared goals.

VI. CONCLUSION

Observability in cloud-native space (i.e., distributed containerized edge infrastructures and microservices) is an area that is yet to be fully realized. It plays a vital role in efficiently managing various edge operational areas, including SLA validation, resource provisioning, and optimization.

This paper provides a detailed survey of state of the art in distributed IT environments and microservices observability. Our research goal was to investigate the challenges, requirements, and current best practices and solutions in the observability of distributed systems. Therefore, we surveyed the most relevant research and recently published articles. We highlighted diverse types of telemetry data that should aid in resolving operational issues. This telemetry data provide valuable signals (e.g., latency, traffic, errors, and saturation) to serve as a foundation for alerting, troubleshooting, and capacity planning. We determined and discussed the fundamental functionalities and characteristics that we considered crucial in delivering observability. Finally, we discussed open research issues or challenges in the domain of observability. These issues are primarily related to heterogeneous infrastructures and microservices architectures realization in various use case scenarios. There is also an interesting organizational culture and individual mindset aspect that can affect the adoption of new observability tools and practices.

In the future work, we aim to work on practical aspects, such as designing and evaluating an observability framework based on the presented considerations in this survey. Specifically, we will target a Kubernetes-based edge infrastructure that hosts containerized microservices. This includes integrating telemetry data collection solutions and various plugins for flexible processing of telemetry data.

REFERENCES

[1] M. Wurster, U. Breitenbücher, A. Brogi, F. Leymann, and J. Soldani, "Cloud-native deploy-ability: An analysis of required features of deployment technologies to deploy arbitrary cloud-native applications," in *Proc. 10th Int. Conf. Cloud Comput. Services Sci.*, 2020, pp. 171–180. [Online]. Available: https://www.scitepress.org/PublicationsDetail.aspx?ID=cs0+4jJjCM4=&t=1

[2] S. Baškarada, V. Nguyen, and A. Koronios, "Architecting microservices: Practical opportunities and challenges," *J. Comput. Inf. Syst.*, vol. 60, no. 5, pp. 428–436, Sep. 2020, doi: 10.1080/08874417.2018.1520056.

[3] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, May 2016.

[4] M. Biehl, *RESTful API Design: APIs Your Consumers Will Love*. Rotkreuz, Switzerland: API-Univ. Press, 2016.

[5] A. Alkhatib, A. Al Sabbagh, and R. Maraqa, "Pubic cloud computing: Big three vendors," in *Proc. Int. Conf. Inf. Technol. (ICIT)*, Jul. 2021, pp. 230–237.

[6] R. Ewaschuk and B. Beyer. (2016). *Monitoring Distributed Systems*. [Online]. Available: https://www.safaribooksonline.com/library/view//9781492029670/?ar

[7] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, "Cloud monitoring: A survey," *Comput. Netw.*, vol. 57, no. 9, pp. 2093–2115, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128613001084

[8] I. Kohyarnejadfard, D. Aloise, M. R. Dagenais, and M. Shakeri, "A framework for detecting system performance anomalies using tracing data analysis," *Entropy*, vol. 23, no. 8, p. 1011, Aug. 2021. [Online]. Available: https://www.mdpi.com/1099-4300/23/8/1011

[9] J. Arkko, S. Farrell, M. Kühlewind, and C. Perkins, "Report from the IAB COVID-19 network impacts workshop 2020," *Internet RFC*, Jul. 2021, p. 20. [Online]. Available: https://datatracker.ietf.org/doc/rfc9075

[10] J. Shah and D. Dubaria, "Building modern clouds: Using docker, kubernetes and Google cloud platform," in *Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2019, pp. 0184–0189.

[11] A. Thurai and S. D. Linthicum, "GigaOm radar for cloud observability," GigaOm, Santa Barbara, CA, USA, Tech. Rep., 2021, p. 37.

[12] D. A. Tamburri, M. Miglierina, and E. D. Nitto, "Cloud applications monitoring: An industrial study," *Inf. Softw. Technol.*, vol. 127, Nov. 2020, Art. no. 106376. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920301452

[13] H. J. Syed, A. Gani, R. W. Ahmad, M. K. Khan, and A. I. A. Ahmed, "Cloud monitoring: A review, taxonomy, and open research issues," *J. Netw. Comput. Appl.*, vol. 98, pp. 11–26, Nov. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804517302783

[14] I. A. Alimi, R. K. Patel, A. Zaouga, N. J. Muga, Q. Xin, A. N. Pinto, and P. P. Monteiro, *Trends Cloud Computing Paradigms: Fundamental Issues, Recent Advances, and Research Directions Toward 6G Fog Networks*. Rijeka, Croatia: IntechOpen, Jun. 2021.

[15] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[16] L. J. de Melo de Azevedo, J. C. Estrella, C. F. M. Toledo, and S. Reiff-Marganiec, "A multi-objective optimized service level agreement approach applied on a cloud computing ecosystem," *IEEE Access*, vol. 8, pp. 122469–122479, 2020.

[17] J. S. Hurwitz, *Cloud Computing for Dummies*, 2nd ed. Indianapolis, IN, USA: Wiley, 2020.

[18] A. C. Risdianto, M. Usman, and J. Kim, "SmartX Box: Virtualized hyper-converged resources for building an affordable playground," *Electronics*, vol. 8, no. 11, p. 1242, Oct. 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/11/1242

[19] M. De Donno, K. Tange, and N. Dragoni, "Foundations and evolution of modern computing paradigms: Cloud, IoT, edge, and fog," *IEEE Access*, vol. 7, pp. 150936–150948, 2019.

[20] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2017.

[21] D. Gannon, R. Barga, and N. Sundaresan, "Cloud-native applications," *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 16–21, Sep. 2017.

[22] J. T. Taylor and W. T. Taylor, "Software architecture," in *Patterns Machine: A Software Engineering Guide to Embedded Development*, J. T. Taylor and W. T. Taylor, Eds. Berkeley, CA, USA: Apress, 2021, pp. 63–82, doi: 10.1007/978-1-4842-6440-9_5.

[23] L. De Lauretis, "From monolithic architecture to microservices architecture," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2019, pp. 93–96.

[24] R. Kumar. (Mar. 2020). *White Box Monitoring and Black Box Monitoring explained*. [Online]. Available: https://www.devopsschool.com/blog/white-box-monitoring-and-black-box-monitoring-explained/

[25] J. Renita and N. E. Elizabeth, "Network's server monitoring and analysis using nagios," in *Proc. Int. Conf. Wireless Commun., Signal Process. Netw. (WiSPNET)*, Mar. 2017, pp. 1904–1909.

[26] M. Jelassi, C. Ghazel, and L. A. Saïdane, "A survey on quality of service in cloud computing," in *Proc. 3rd Int. Conf. Frontiers Signal Process. (ICFSP)*, Sep. 2017, pp. 63–67.

[27] D. Deepshikha and S. Prakash, "A survey on QoS in cloud computing environment," in *Proc. 3rd Int. Conf. Comput. Methodol. Commun. (ICCMC)*, Mar. 2019, pp. 574–578.

[28] A. A. G. Abushagur, T. S. Chin, R. Kaspin, N. Omar, and A. T. Samsudin, "Hybrid software-defined network monitoring," in *Internet Distributed Computing Systems* (Lecture Notes in Computer Science), R. Montella, A. Ciaramella, G. Fortino, A. Guerrieri, and A. Liotta, Eds. Cham, Switzerland: Springer, 2019, pp. 234–247.

[29] M. Abderrahim, M. Ouzzif, K. Guillouard, J. Francois, and A. Lebre, "A holistic monitoring service for fog/edge infrastructures: A foresight study," in *Proc. IEEE 5th Int. Conf. Future Internet Things Cloud (FiCloud)*, Aug. 2017, pp. 337–344.

[30] R. da Rosa Righi, M. Lehmann, M. M. Gomes, J. C. Nobre, C. A. da Costa, S. J. Rigo, M. Lena, R. F. Mohr, and L. R. B. de Oliveira, "A survey on global management view: Toward combining system monitoring, resource management, and load prediction," *J. Grid Comput.*, vol. 17, no. 3, pp. 473–502, Sep. 2019. [Online]. Available: http://link.springer.com/10.1007/s10723-018-09471-x

[31] M. Birje and C. Bulla, "Cloud monitoring system: Basics, phases and challenges," *International J. Recent Technol. Eng.*, vol. 8, no. 3, pp. 4732–4746, 2019.

[32] S. Niedermaier, F. Koetter, A. Freymann, and S. Wagner, "On observability and monitoring of distributed systems—An industry interview study," in *Service-Oriented Computing* (Lecture Notes in Computer Science), S. Yangui, I. Bouassida Rodriguez, K. Drira, and Z. Tari, Eds. Cham, Switzerland: Springer, 2019, pp. 36–52.

[33] S. Khoudali, K. Benzidane, and A. Sekkaki, "EMMCS: An edge monitoring framework for multi-cloud environments using SNMP," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 1, pp. 1–11, 2019. [Online]. Available: https://thesai.org/Publications/ViewPaper?Volume=10&Issue=1&Code=IJACSA&SerialNo=78

[34] S. Becker, F. Schmidt, A. Gulenko, A. Acker, and O. Kao, "Towards AIOps in edge computing environments," Feb. 2021, *arXiv:2102.09001*.

[35] Á. Brandón, M. S. Pérez, J. Montes, and A. Sanchez, "FMonE: A flexible monitoring solution at the edge," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–15, Nov. 2018. [Online]. Available: https://www.hindawi.com/journals/wcmc/2018/2068278/

[36] M. Usman and J. Kim, "Smartx multi-view visibility framework for unified monitoring of SDN-enabled multisite clouds," *Trans. Emerg. Telecommun. Technol.*, Dec. 2019, Art. no. e3819. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/ett.3819

[37] K. Alhamazani, R. Ranjan, P. Prakash Jayaraman, K. Mitra, C. Liu, F. Rabhi, D. Georgakopoulos, and L. Wang, "Cross-layer multi-cloud real-time application QoS monitoring and benchmarking as-a-service framework," *IEEE Trans. Cloud Comput.*, vol. 7, no. 1, pp. 48–61, Jan. 2019.

[38] I. Hababeh, A. Thabain, and S. Alouneh, "An effective multivariate control framework for monitoring cloud systems performance," *KSII Trans. Internet Inf. Syst.*, vol. 13, no. 1, p. 86, Jan. 2019. [Online]. Available: https://link.gale.com/apps/doc/A582098127/AONE?sid=bookmark-AONE&xid=40d1dc77

[39] A. Noor, K. Mitra, E. Solaiman, A. Souza, D. N. Jha, U. Demirbaga, P. P. Jayaraman, N. Cacho, and R. Ranjan, "Cyber-physical application monitoring across multiple clouds," *Comput. Electr. Eng.*, vol. 77, pp. 314–324, Jul. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790618330830

[40] M. B. Chhetri, Q. B. Vo, and R. Kowalczyk, "CL-SLAM: Cross-layer SLA monitoring framework for cloud service-based applications," in *Proc. 9th Int. Conf. Utility Cloud Comput.*, Dec. 2016, pp. 30–36.

[41] H. J. Syed, A. Gani, F. H. Nasaruddin, A. Naveed, A. I. A. Ahmed, and K. Khan, "CloudProcMon: A non-intrusive cloud monitoring framework," *IEEE Access*, vol. 6, pp. 44591–44606, 2018.

[42] T. Wang, J. Xu, W. Zhang, Z. Gu, and H. Zhong, "Self-adaptive cloud monitoring with online anomaly detection," *Future Gener. Comput. Syst.*, vol. 80, pp. 89–101, Mar. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X1730376X

[43] M. Jabbarifar, A. Shameli-Sendi, and B. Kemme, "A scalable network-aware framework for cloud monitoring orchestration," *J. Netw. Comput. Appl.*, vol. 133, pp. 1–14, May 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804519300505

[44] *Monasca—OpenStack*. Accessed: Aug. 17, 2022. [Online]. Available: https://wiki.openstack.org/wiki/Monasca

[45] *Instana—Enterprise Observability and APM for Cloud-Native Applications*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.instana.com/

[46] *What is Splunk?* Accessed: Aug. 17, 2022. [Online]. Available: https://www.splunk.com/en_us/about-splunk.html

[47] *Blazing-Fast Log Management and Observability for Modern Apps*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.scalyr.com

[48] *NetBeez Network Monitoring From the User Perspective*. Accessed: Aug. 17, 2022. [Online]. Available: https://netbeez.net/

[49] D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Low-cost adaptive monitoring techniques for the Internet of Things," *IEEE Trans. Services Comput.*, vol. 14, no. 2, pp. 487–501, Mar. 2021.

[50] A. Celesti and M. Fazio, "A framework for real time end to end monitoring and big data oriented management of smart environments," *J. Parallel Distrib. Comput.*, vol. 132, pp. 262–273, Oct. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731518308256

[51] I. Angelopoulos, E. Trouva, and G. Xilouris, "A monitoring framework for 5G service deployments," in *Proc. IEEE 22nd Int. Workshop Comput. Aided Modeling Design Commun. Links Netw. (CAMAD)*, Jun. 2017, pp. 1–6.
[52] R. Perez, J. Garcia-Reinoso, A. Zabala, P. Serrano, and A. Banchs, "A monitoring framework for multi-site 5G platforms," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2020, pp. 52–56.
[53] *Grafana Loki*. Accessed: Aug. 17, 2022. [Online]. Available: https://grafana.com/oss/loki/
[54] *Logstash: Collect, Parse, Transform Logs*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.elastic.co/logstash
[55] 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. 34:1–34:26, Nov. 2020, doi: 10.1145/3418899.
[56] M. Cinque, R. Della Corte, and A. Pecchia, "Advancing monitoring in microservices systems," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2019, pp. 122–123.
[57] A. Noor, D. N. Jha, K. Mitra, P. P. Jayaraman, A. Souza, R. Ranjan, and S. Dustdar, "A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments," in *Proc. IEEE 12th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2019, pp. 156–163.
[58] C.-A. Lai, J. Kimball, T. Zhu, Q. Wang, and C. Pu, "MilliScope: A fine-grained monitoring framework for performance debugging of n-tier web services," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 92–102.
[59] A. A. Laghari, H. He, A. Khan, N. Kumar, and R. Kharel, "Quality of experience framework for cloud computing (QoC)," *IEEE Access*, vol. 6, pp. 64876–64890, 2018.
[60] *Introduction to Cilium & Hubble—Cilium 1.10.3 Documentation*. Accessed: Aug. 17, 2022. [Online]. Available: https://docs.cilium.io/en/v1.10/intro/#what-is-cilium
[61] 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.
[62] C. Cassagnes, L. Trestioreanu, C. Joly, and R. State, "The rise of eBPF for non-intrusive performance monitoring," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2020, pp. 1–7.
[63] L. Deri, S. Sabella, and S. Mainardi, "Combining system visibility and security using eBPF," in *Proc. ITASEC*, 2019.
[64] 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.
[65] *Apache SkyWalking*. Accessed: Aug. 17, 2022. [Online]. Available: https://skywalking.apache.org/
[66] *OpenTelemetry*. Accessed: Aug. 17, 2022. [Online]. Available: https://opentelemetry.io/
[67] *Consul by HashiCorp*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.consul.io/
[68] Prometheus. *Overview Bar Prometheus*. Accessed: Aug. 17, 2022. [Online]. Available: https://prometheus.io/docs/introduction/overview/
[69] *Jaeger: Open Source, End-to-End Distributed Tracing*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.jaegertracing.io/
[70] *The Leader in Cloud Monitoring*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.dynatrace.com/
[71] Datadog. *Cloud Monitoring as a Service Bar Datadog 0400*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.datadoghq.com/
[72] *New Relic Deliver More Perfect Software*. Accessed: Aug. 17, 2022. [Online]. Available: https://newrelic.com/
[73] Zebrium. *Automatic Root Cause Analysis (RCA) for Logs and Metrics Zebrium*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.zebrium.com
[74] *Cloud Log Management, Monitoring, SIEM Tools*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.sumologic.com/
[75] *IT Management Software & Remote Monitoring Tools SolarWinds*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.solarwinds.com
[76] *Homepage*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.honeycomb.io/
[77] M. Usman, A. C. Risdianto, J. Han, M. Kang, and J. Kim, "SmartX multiview visibility framework leveraging open-source software for SDN-cloud playground," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jul. 2017, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/8004242/
[78] M. Usman, A. C. Risdianto, and J. Kim, "Resource monitoring and visualization for OFTEIN SDN-enabled multi-site cloud," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*. Kota Kinabalu, Malaysia, Jan. 2016, pp. 427–429. [Online]. Available: http://ieeexplore.ieee.org/document/7427150/
[79] M. Usman, A. C. Risdianto, J. Han, and J. Kim, "Inter-correlation of resource-/flow-level visibility for apm over OF@TEIN SDN-enabled multi-site cloud," in *Quality, Reliability, Security and Robustness in Heterogeneous Networks* (Lecture Notes of the Institute for Computer Sciences), vol. 199, J.-H. Lee and S. Pack, Eds. Cham, Switzerland: Springer, 2017, pp. 478–484. [Online]. Available: http://link.springer.com/10.1007/978-3-319-60717-7_48
[80] M. A. Rathore, M. Usman, and J. Kim, "Integrating active monitoring for restricted topology-awareness of SmartX multi-view playground visibility," in *Proc. 13th Int. Conf. Future Internet Technol.*, Seoul, South Korea, Jun. 2018, pp. 1–4.
[81] J.-S. Shin, M. Usman, and J. Kim, "Conceptual verification of multi-level visibility points for SmartX MultiView security," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2018, pp. 1449–1451.
[82] M. A. Rathore, M. Usman, and J. Kim, "Maintaining SmartX multi-view visibility for OF@TEIN+ distributed cloud-native edge boxes," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 6, pp. 1–24, Jun. 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/ett.4101
[83] P. G. Kannan, N. Budhdev, R. Joshi, and M. C. Chan, "Debugging transient faults in data center networks using synchronized network-wide packet histories," in *Proc. Symp. Netw. Syst. Design Implement.*, 2021, p. 17.
[84] M. Abranches, O. Michel, E. Keller, and S. Schmid, "Efficient network monitoring applications in the kernel with eBPF and XDP," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2021, pp. 28–34.
[85] Q. Zhang, K. K. W. Ng, C. Kazer, S. Yan, J. Sedoc, and V. Liu, "MimicNet: Fast performance estimates for data center networks with machine learning," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 287–304, doi: 10.1145/3452296.3472926.
[86] G. Fournier, S. Afchain, and S. Baubeau, "Runtime security monitoring with ebpf," in *Proc. 17th Symp. la Sécurité des Technologies de l'Information et de la Communication (SSTIC)*, 2021, pp. 1–23.
[87] CNCF. (Feb. 2022). *CNCF Cloud Native Interactive Landscape*. [Online]. Available: https://landscape.cncf.io/card-mode?category=monitoring,logging,tracing&grouping=category&license=open-source&project=hosted,member,no
[88] Prometheus. *Prometheus—Monitoring System & Time Series Database*. Accessed: Aug. 17, 2022. [Online]. Available: https://prometheus.io/
[89] (Mar. 2022). *Beats—The Lightweight Shippers of the Elastic Stack*. [Online]. Available: https://github.com/elastic/beats
[90] (Mar. 2022). *Loki: Like Prometheus, but for Logs*. [Online]. Available: https://github.com/grafana/loki
[91] F. Project. *What is Fluentd? Fluentd*. Accessed: Aug. 17, 2022. [Online]. Available: https://www.fluentd.org/architecture
[92] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using Magpie for request extraction and workload modelling," in *Proc. OSDI*, 2004, p. 14.
[93] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "X-Trace: A pervasive network tracing framework," in *Proc. 4th USENIX Symp. Netw. Syst. Design Implement.*, 2004, p. 14.
[94] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," Google, Menlo Park, CA, USA, Tech. Rep., 2010. [Online]. Available: https://research.google.com/archive/papers/dapper-2010-1.pdf
[95] R. Sahay, "Azure monitoring," in *Microsoft Azure Architect Technologies Study Companion: Hands-on Preparation and Practice for Exam AZ-300 and AZ-303*, R. Sahay, Ed. Berkeley, CA, USA: Apress, 2020, pp. 139–167. [Online]. Available: https://doi.org/10.1007/978-1-4842-6200-9_5
[96] *Amazon CloudWatch*. Accessed: Aug. 17, 2022. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html
[97] *Operations: Cloud Monitoring & Logging*. Accessed: Aug. 17, 2022. [Online]. Available: https://cloud.google.com/products/operations
[98] R. Gerhards, *The Syslog Protocol*, document Internet Engineering Task Force, Request for Comments RFC 5424, Mar. 2009, p. 38. [Online]. Available: https://datatracker.ietf.org/doc/rfc5424
[99] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA, USA: O'Reilly Media, Mar. 2016.

[100] M. Usman, A. C. Risdianto, J. Han, and J. Kim, "Interactive visualization of SDN-enabled multisite cloud playgrounds leveraging SmartX MultiView visibility framework," *Comput. J.*, vol. 62, no. 6, pp. 838–854, Jun. 2019, doi: 10.1093/comjnl/bxy103.

[101] B. Mohammed, M. Kiran, and B. Enders, "NetGraf: An end-to-end learning network monitoring service," in *Proc. IEEE Workshop Innovating Netw. Data-Intensive Sci. (INDIS)*, St. Louis, MO, USA, Nov. 2021, pp. 12–22.

[102] Y. I. Binev. (Jan. 2021). *Centralised Monitoring and Alerting Solution for Complex Information Management Infrastructure*. [Online]. Available: https://run.unl.pt/handle/10362/112992?mode=full

[103] Y. Mansouri and M. A. Babar, "A review of edge computing: Features and resource virtualization," *J. Parallel Distrib. Comput.*, vol. 150, pp. 155–183, Apr. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0743731520304317

[104] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, "Serverless edge computing: Vision and challenges," in *Proc. Australas. Comput. Sci. Week Multiconference*, Feb. 2021, pp. 1–10, doi: 10.1145/3437378.3444367.

[105] T. Zhang, Y. Li, and C. L. P. Chen, "Edge computing and its role in industrial internet: Methodologies, applications, and future directions," *Inf. Sci.*, vol. 557, pp. 34–65, May 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0020025520311865

[106] B. Ali, M. A. Gregory, and S. Li, "Multi-access edge computing architecture, data security and privacy: A review," *IEEE Access*, vol. 9, pp. 18706–18721, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9330515/

MUHAMMAD USMAN (Member, IEEE) received the B.S. degree in information technology from the School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), Pakistan, in 2011, and the integrated M.S. and Ph.D. degrees in computer science from the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology (GIST), Republic of Korea, in 2019. Currently, he is working as a Postdoctoral Researcher with the Department of Mathematics and Computer Science, Karlstad University, Sweden. His research interests include cloud and edge computing, distributed systems measurements and performance observability, DevOps automation, and big data processing and visualizations.

SIMONE FERLIN received the Dipl.-Ing. degree in information technology with major in telecommunications from Friedrich-Alexander Erlangen-Nuernberg University (FAU), Germany, in 2010, and the Ph.D. degree in computer science from the University of Oslo (UiO), Norway, in 2017. She is currently a Senior Performance Engineer at Red Hat. Her research interests include intersection of cellular networks and the internet, with a focus on computer networking, QoS and cross-layer design, transport protocols, congestion control, network performance, security, and measurements. Her dissertation focused on improving robustness in multipath transport for heterogeneous networks with MPTCP. She actively serves on technical boards of major conferences and journals in these areas.

ANNA BRUNSTROM (Member, IEEE) received the B.Sc. degree in computer science and mathematics from Pepperdine University, CA, USA, in 1991, and the M.Sc. and Ph.D. degrees in computer science from the College of William & Mary, VA, USA, in 1993 and 1996, respectively. She joined the Department of Computer Science, Karlstad University (KAU), Sweden, in 1996, where she is currently a Full Professor and the Research Manager of the Distributed Systems and Communications Research Group. Her research interests include internet architectures and protocols, techniques for low latency internet communication, multi-path communication, and performance evaluation of mobile broadband systems, including 5G. She is currently the KAU Principal Investigator within the EU H2020 Project 5GENESIS. She has authored/coauthored over 200 international journals and conference papers. She is the Co-Chair of the RTP Media Congestion Avoidance Techniques (RMCAT) Working Group within the IETF.

JAVID TAHERI (Senior Member, IEEE) received the Ph.D. degree in mobile computing from the School of Information Technologies, The University of Sydney, Australia. He is currently a Professor with the Department of Computer Science, Karlstad University, Sweden. His research interests include profiling, modeling, optimization techniques for private and public cloud infrastructures, profiling, modeling, optimization techniques for software-defined networks, and network-aware scheduling algorithms for cloud computing.