International Journal Of Core Engineering & Management
Volume-7, Issue-04, 2022 SSN No: 2348-9510

OBSERVABILITY IN CLOUD-NATIVE ENVIRONMENTS CHALLENGES AND SOLUTIONS

Premkumar Ganesan
Technology Leader in Digital Transformation for Government and Public Sector, Baltimore, USA

Abstract

The need for resilient and scalable software has grown significantly in the modern software development landscape, especially due to the exponential expansion of web-based services and applications. With their new architectural methods, cloud-native technologies provide dynamic scalability and robust resilience, making them a transformational force in tackling these difficulties. This article provides an in-depth analysis of how software development might benefit from cloud-native technologies in terms of scalability and resilience. Starting with cloud-native architecture's guiding principles—including containerization, microservices, and declarative APIs—the paper moves on to examine its underlying notions. Following these guidelines, programmers can create and release apps with fault tolerance, high availability, and the capacity to scale dynamically according to user demand. Additionally, the evaluation delves into the essential elements of ecosystems that are native to the cloud, such as Kubernetes and other container orchestration tools that automate the scaling and maintenance of containerized applications. Also covered is how service meshes help make systems more resilient by allowing microservices to communicate with each other in a safe and dependable manner. In addition, exploring themes like distributed tracing, circuit breaking, and chaotic engineering, the article dives into patterns and best practices for building resilient and scalable cloud-native apps. To make their systems more resilient, developers can use these approaches to find any weak spots and fix them before they happen. The importance of cloud-native technologies in facilitating the development of resilient and scalable applications is highlighted in this review. Organizations may successfully adapt to the ever-changing requirements of software development in today's competitive market by utilizing cloud-native ideas and the right tools and processes.

Keywords—Cloud-Native; Technologies; Software; Development; Resilience.

I. INTRODUCTION

Observability is an essential part of managing and maintaining complex cloud-native systems. As more and more companies embrace cloud-native designs—which depend on distributed and highly dynamic infrastructures—the need for comprehensive observability becomes vital. In this post, we will look at what observability is, how it works, and why it's important for systems

that are built on the cloud. In addition to discussing the benefits of observability for performance optimization and troubleshooting, we will analyze its core components—metrics, logs, and traces [1]. What we mean by "observability" in the context of cloud-native systems is that we can figure out how the system works by looking at its outputs, rather than trying to figure out the system's underlying structure [2]. It surpasses the scope of traditional monitoring methods, which primarily focus on basic health and component availability. To facilitate effective management and rapid issue resolution, observability endeavors to furnish a thorough comprehension of system behavior, performance, and interdependencies. the third Because applications in the cloud often consist of numerous microservices and containers operating on dynamic infrastructure, observability is of the utmost importance in these systems. It was easier to understand and locate systemic issues with traditional monolithic designs. The breadth and depth of cloud-native settings, however, make it challenging to quickly detect and resolve problems without observability processes in place [4]. There are three primary components to observability: metrics, logs, and traces. Numbers pertaining to many aspects of the system, such as memory and CPU utilization, request latency, and error rates, are recorded by metrics [5]. They provide system operators with a bird's-eye perspective of the system's performance and health, letting them see patterns, generate alerts, and make informed decisions.

Contrarily, logs document all system occurrences and actions in detail. These log entries include alarms, errors, cautions, and anything else that is relevant. By collecting and analyzing logs, operators can gain a greater understanding of the system's internal condition, identify abnormalities, and address problems more effectively [6]. Understanding the flow of requests among distributed application components is the primary objective of traces. Tracking individual requests, including their path through different microservices, durations, and any potential bottlenecks, allows operators to see the overall performance and behavior of the system [7]. There are many benefits to having observability in cloud-native systems. As a first benefit, it helps with problem-solving by illuminating the origins of issues. In the event of an issue, the operator can rapidly determine its source by cataloguing the affected components, checking relevant data, inspecting logs, and following the request flow. This shortens the resolution procedure and decreases system downtime [8]. It is also easier to optimize performance when it is observable. By monitoring and studying metrics, operators can identify inefficiencies, bottlenecks, or anomalous behavior. They can use this information to determine the optimal distribution of resources, adjust the size of individual components as needed, and optimize the system's setup for maximum performance [9]. The ability to actively monitor and plan for capacity is another benefit of observability. Operators can identify patterns and trends in system activity and metrics over time, which could indicate issues or resource constraints. This allows for the optimization of workloads or the scalability of resources before problems affect the availability or performance of the system [10].

International Journal Of Core Engineering & Management
Volume-7, Issue-04, 2022 SSN No: 2348-9510

II. LITERATURE REVIEW

The growth of cloud computing and microservice architectures has caused a sea change in the app development, deployment, and control processes. Reliability, scalability, and optimal application performance in today's dynamic and dispersed environment are prerequisites for cloud-native operations and observability. If the system is observable, you can learn a lot about its health and how it is functioning. What makes cloud-native apps easy to manage and fix are the procedures and tools that are "native" to the cloud [11]. The desire for faster, more flexible, and scalable software development and deployment has pushed many towards cloud-native architectures. It is now common practice to build cloud-native apps using microservices, which are independent, loosely linked components. Over sixty-five percent of companies have used microservices, with sixty-one percent of those companies running fifty or more microservices, according to a poll by the Cloud Native Computing Foundation (CNCF) [12]. But it's more difficult to manage, monitor, and repair system-wide issues when microservices are dispersed. Traditional monitoring methods like siloed tools with restricted access just don't cut it in today's cloud-native systems. When it comes to understanding the inner workings and behavior of the system, cloud native observability is invaluable [13]. The three primary components of observability in the cloud are metrics, distributed tracing, and logs. The function of logging is to record system events and operations in a thorough manner. This enables operators to track the operations' flow and identify issues. Quantitative metrics reveal the efficiency of a system, the utilization of its resources, and the actions of its users. This facilitates proactive monitoring and capacity planning. You may improve service communication and identify the source of latency issues with the help of distributed tracing, which shows the complete request lifecycle as it moves through microservices [14]. A study conducted by the CNCF demonstrated the significance of observability. The results showed that unlike organizations with immature procedures, those with advanced observability processes saw 43% fewer production difficulties and a 64% shorter mean time to resolution (MTTR) [15]. For cloud-native programs to be robust and dependable, observability is crucial.

Not only are cloud-native activities observable, but they are also crucial to the proper operation of these complex systems. When it comes to cloud-native operations, automation is essential for launching, scaling, and recovery because it streamlines the whole process. With Infrastructure as Code (IaC) solutions, such as Ansible and Terraform, you may declaratively define and provision cloud resources. Doing so guarantees uniformity and adaptability across settings. By automating software delivery, Continuous Integration/Continuous Deployment (CI/CD) pipelines allow for rapid and reliable app updates [16]. The use of container orchestration tools, such as Kubernetes, to manage workloads in cloud-native settings has become the de facto standard. An excellent platform for scheduling, managing, and scaling containers is Kubernetes. The infrastructure behind is hidden, allowing developers to focus on the apps' logic. Most businesses (91%) utilize Kubernetes in production, with 78 percent of those businesses running more than 50 clusters, according to a CNCF poll [17].

International Journal Of Core Engineering & Management
Volume-7, Issue-04, 2022 SSN No: 2348-9510

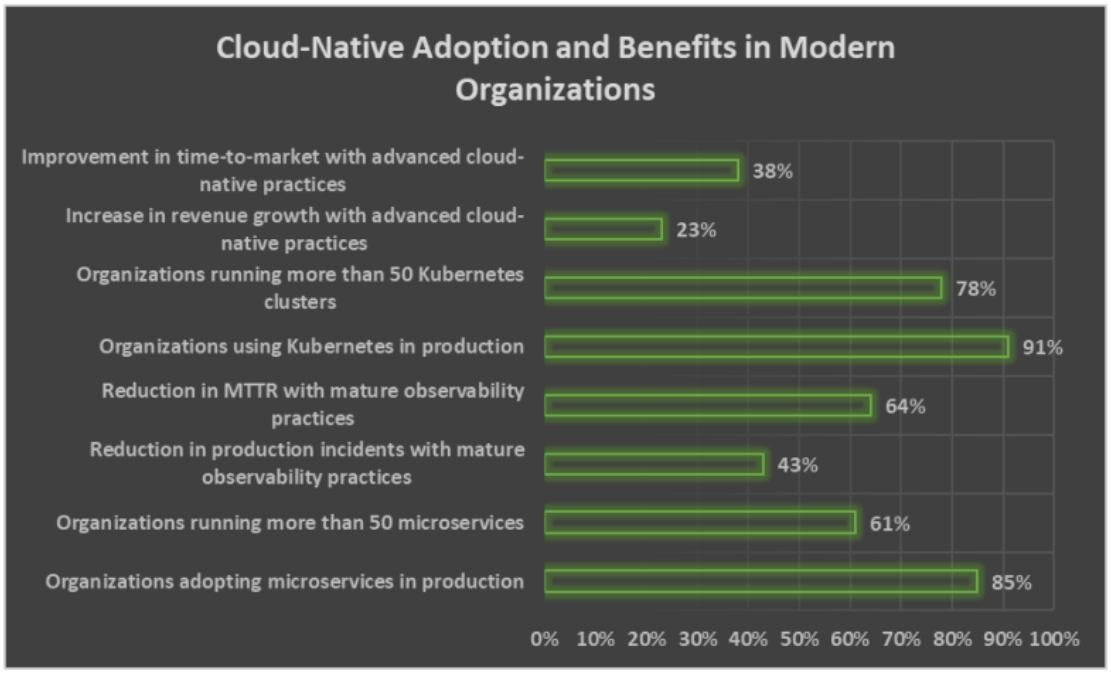

Fig. 1: The Impact of Observability and Kubernetes on Cloud-Native Success

Businesses have benefited greatly from cloud-native observability and operational approaches. Businesses who adopted cutting-edge cloud-native strategies outperformed their rivals by 23% in revenue growth and 38% in time to market, according to research from Google Cloud and Harvard Business Review [18]. These findings highlight the significance of utilizing cloud-native observability and operations in generating value and innovation for businesses. When it comes to building and maintaining scalable and robust apps in the cloud, cloud-native operations and observability are crucial components [19]. With the use of analytics, automation, distributed tracing, logging, and container orchestration platforms, organizations may simplify deployment, learn more about application behavior, and increase speed. Companies who wish to thrive in the modern digital economy will need to adopt these practices and technologies as the cloud-native environment evolves [20].

III. OBSERVATIONS ON CLOUD-NATIVE OBSERVABILITY

Developers' daily lives aren't complete without cloud native and observability. When developers are aware of their roles within observability at scale, they are better able to address the problems they encounter every day. Observability goes beyond simple data collection and storage, and developers are crucial for overcoming these obstacles.

Observability Foundations

International Journal Of Core Engineering & Management
Volume-7, Issue-04, 2022 SSN No: 2348-9510

We no longer need to wait for fresh resources to deploy our code, debug services within our development toolchain, and monitor a known application environment. Thanks to auto-scaling infrastructure, this is now available in the final production deployment environments fast, flexible, and dynamically. Modern developers frequently aim to own their code for its entire lifecycle, from development to production, and they want to be able to observe everything they create along the way. Cloud settings with thousands of microservices are dynamic and unpredictable, making it impossible for legacy technologies like Nagios and HP OpenView to keep up. Being able to dynamically scale infrastructure is a must for cloud-native deployments. Observability platforms help reduce data noise and identify trends that could cause downtime, allowing for proactive mitigation.

A. Splintering of Responsibilities in Observability

Team	Focus	Maturity Goals
DevOps	Software development lifecycle automation and optimization encompassing post-launch repairs and updates	Initial phases: developer efficiency
Platform engineering	Creating and designing procedures and toolchains that allow developers to self-service	Achieving developer maturity and increasing productivity in the early stages
CloudOps	Helps businesses streamline their operations by managing their resources in the cloud according to DevOps and IT operations best practices.	Cloud resource management, expenses, and company agility in subsequent stages
SRE	Regardless of whether an app or its underlying infrastructure is cloud native or not, this all-encompassing role's primary objective is to maintain reliability in any given context.	Stages ranging from early to late: on-call engineers aiming to minimise downtime
Central observability team	In charge of maintaining tooling and observability data storage, as well as delivering critical data to engineering teams and creating standards and procedures for observability.	Later on, when you own The first step is to establish guidelines for monitoring. transmit data gathered from monitoring to the engineering teams Third, make sure the monitoring systems are stable and reliable. 4. Oversee the measurement data storage and tooling processes

Table 1. Observability Team

International Journal Of Core Engineering & Management
Volume-7, Issue-04, 2022 SSN No: 2348-9510

Not only did cloud-native complexity alter the landscape of developers, it also altered the organizational structure of many companies. Multiple newly formed organizational teams are now responsible for developing, implementing, and overseeing cloud-native infrastructure. With the rise of hybrid jobs, developers are expected to take on more responsibilities beyond code generation and work more closely with other team members. Observability teams have been formed to offer a particular service to organizations within the cloud infrastructure by concentrating on a particular part of the cloud-native ecosystem. Table 1 shows how these specialized teams have broken down traditional organizational responsibilities. To grasp the interdependencies among these groups, picture a big, established, cloud-native company with all the groups listed in Table 1:

- When it comes to standardizing the processes of code creation, management, testing, updates, and deployment, the DevOps team is at the forefront. The platform engineering group supplies them with toolchains and workflow, which they use.
- Continuous improvement is the goal of DevOps, which is why it provides advice on new tools and workflows.
- Managing cloud resources and making the most of other teams' cloud budgets are the main focuses of a CloudOps team.
- To ensure that the organization's supporting infrastructure never goes down, a dedicated SRE team is available 24/7 to handle reliability management. To help all the teams enhance their platforms, methods, and tools, they offer input.

The observability standards are created by the central observability team, which also manages tooling and data storage, ensures that the proper observability data is sent to the right teams, and oversees the overall observability effort.

IV. WHY OBSERVABILITY IS IMPORTANT TO CLOUD NATIVE

Theproliferation of cloud native apps has left developers with an excessive amount of work to do beyond writing code alone. Observability is becoming crucial for developers to solve many of the problems they are encountering, due to the complexity that cloud-native systems provide.

A. Challenges

Developers are delivering more code at a faster rate and passing more stringent tests to guarantee that their apps function at cloud native scale, which is causing the complexity of cloud native applications to increase. Because of these difficulties, the requirement for transparency in the environment where developers typically coded has grown. Instrumenting their code to monitor business metrics is an additional requirement on top of providing code and testing infrastructure for their applications. Over time, developers realized that it was unnecessary to fully automate measurements, and that a lot of the data was superfluous. Because of this, developers started using manual instrumentation to hone their techniques of

measurement and capture only the metrics they needed. Integrating new observability techniques into an organization's current application ecosystems is another difficult task. Many developers fail to appreciate the effort required to manually instrument their applications to supply an observability platform with the data it needs.

There are already enough of problems for developers before they must deal with new observability technologies that aid with metrics, logs, and traces. organizations pay a premium for advanced observability technologies, but only a select few really know how to use them. This leads to knowledge silos and wasteful spending on the tools.

Finally, examining the consumed data from our cloud infrastructure makes it quite clear that not all of it must be retained. The capacity to manage telemetry data and identify unnecessary information by observability teams is essential. Some enquiries concerning the ways in which we can accomplish this require answers:

- Find consumed data that our observability teams haven't applied to dashboards, alerting rules, or ad hoc searches.
- Aggregate and apply rules to control telemetry data before storing it for the long term, which can be costly
- Limit telemetry data usage to what is necessary for application landscape monitoring.

To make this data useful for the organization, it is vital to deal with the deluge of cloud data in a way that filters out the unnecessary telemetry data and keeps just that which is applied for our observability needs.

B. Cloud Native at Scale

Although cloud-native infrastructure offers a great deal of flexibility, even little complications might become major obstacles when implemented on a large scale. This is because, according to cloud native principles, we outline the ideal configuration for our infrastructure, the best way to deploy our applications and microservices, and, lastly, how it should handle automated scaling. A company's production infrastructure is less likely to be able to withstand spikes in consumer usage when this method is used.

C. Empowering Developers

Platform engineering teams that priorities developer experiences are the foundation for empowering developers. Our company prioritizes observability in the developer experiences we design, and we invest resources into developing a telemetry strategy from the very beginning. By fostering an environment that values observability in tandem with testing, continuous integration, and continuous deployment, we are preparing development teams to

succeed in the cloud. In addition to taking responsibility of the code they produce, developers are also encouraged and given the authority to generate, test, and own the telemetry data generated by their microservices and applications. They are fostering agility and consensus across the several teams developing cloud solutions in this exciting new environment where they are the bosses of their own job. Any business that wants to succeed in today's cloud-native environment must be able to meet the problems of observability head-on. Developers should keep observability in the forefront of their minds, integrate it into their everyday workflows, and rely on it to help them overcome obstacles.

D. Artificial Intelligence and Observability

Both developer tooling and the observability domain have seen an uptick in the use of artificial intelligence (AI). One of two scenarios describes the use of AI to observability:

1. Keeping an eye on systems that use machine learning (ML) or large language models (LLM).
2. Using AI as a helpful helper in observability tooling.

In the first scenario, you wish to keep tabs on AI workloads, such ML or LLMs. You may further categories them into the training platform and the production platform, each of which you may choose to keep an eye on. It is possible to treat training infrastructure, and the process involved like any other workload: with simple, easily attained monitoring using instruments and current approaches, like watching traces through a solution. While this is just a part of the monitoring process, pre-built observability solutions can easily assist with monitoring these workloads' infrastructure and applications.

In the second scenario, developers are exposed to observability tools that incorporate AI assistants, including chatbots. It usually takes the kind of a code assistant, like the kind that lets us tweak dashboard settings or run ad hoc queries on our time series data. While these are convenient, companies are careful about how developers use them when entering searches involving confidential or private information. To better train the agents for future query assistance, it is crucial to comprehend that training these tools may involve utilizing confidential data in their training sets or even data inputted by developers. Because organizations will always guard their data from being used in ways they can't control, it will be difficult to foresee how AI-assisted observability will evolve in the future. Therefore, having agents trained solely on in-house data would be a step in the right direction towards increased adoption, but it would result in a smaller training data set compared to agents that are publicly available.

E. Cloud-Native Observability: The Developer Survival Pattern

As developers, we know that tooling isn't the magic bullet for all of our complicated challenges, even if we spend a lot of time on it. Just like any other problem, observability can be difficult to

International Journal Of Core Engineering & Management
Volume-7, Issue-04, 2022 SSN No: 2348-9510

solve without taking the broader view into account. However, developers are frequently led astray by the mantra of metrics, logs, and traces. No amount of data collection can ever keep up with the data produced by cloud-native systems, particularly when operating at scale. The performance of development teams is negatively impacted by the deluge of data, the difficulties that come with it, and the incapacity to filter through it to identify the sources of problems. Supporting developers with the correct quantity of data in the right formats at the right time would be more helpful in solving challenges. If issues can be resolved swiftly, circumstances can be remedied efficiently, and developers are happy with the outcomes, then observability is not a concern. All we need is a single log line, two trace spans, and three metric labels to do this. To achieve this goal, it is ideal if developers could be notified in advance when problems with their apps or services are about to occur. As a first step in troubleshooting, they use data that their instrumented applications have already identified to pinpoint problematic regions inside the application. Developers using these tools can view dashboards with visual information that pinpoints the source of the problem and when it may have begun. Developers must have the ability to fix the issue, possibly by reverting a code change or deployment, for the program to keep supporting interactions with customers. Figure 2 shows the approach that cloud native developers used to resolve issues with observability. The final thing a developer should do is think about ways to avoid such problems in the future.

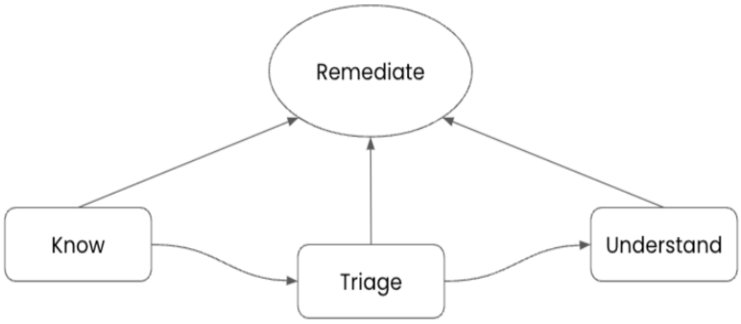

Figure 2. Observability pattern.

V. RESEARCH METHODOLOGIES

A. Methodology Overview

Our methodology is grounded in systematic mapping and systematic literature review preparation, both adapted to fit our research focus. We propose merging these approaches (as shown in Fig. 4) to synthesize evidence and provide a comprehensive secondary study. The observability SMS we suggest includes a sequence of results, each detailed in the following sections.

International Journal Of Core Engineering & Management
Volume-7, Issue-04, 2022 SSN No: 2348-9510

B. Research Objectives

The observability SMS process begins with defining Research Questions (RQs)(Table 2), which guide subsequent steps and outcomes. These RQs cover multiple domains of the studied topic, providing a broad yet unbiased perspective on cloud-native observability.

C. Search Strategy

Our search strategy combines automated and manual searches to identify relevant studies. Automated searches addressed RQs 2, 3, and 4, while RQ1 was addressed through manual citation searches. We formulated search terms using the PICO method to ensure comprehensive coverage of the research questions.

D. Study Selection

We applied a systematic selection process based on predefined inclusion and exclusion criteria. This process prioritized studies on cloud-native application observability published within the last five years. The studies were then classified based on their relevance and contribution to the field.

E. Classification Scheme

The classification scheme was developed by keywording the abstracts and applying an entry form designed by independent reviewers. This form, refined through iterative reviews, ensured a structured categorization of the studies. The resulting analysis offers a clear overview of contributions to cloud-native application observability research.

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

Table 2: Research questions

VI. CONCLUSIONS

It is necessary for organizations to have observability to be successful in a world that is cloud native. It is impossible to overlook the fact that observability obligations are broken up into multiple parts, as well as the difficulties that cloud-native systems present when applied at a large scale. To achieve observability bliss, it is essential to have a solid understanding of the issues that developers encountered in cloud-native organizations. To effectively manage observability in contemporary cloud environments, it is essential to both empower developers and provide solutions to observability difficulties. Additionally, it is essential to have an idea of how the future of observability may look.

REFERENCES

1. B. Scholl, T. Swanson, and P. Jausovec, Cloud Native: Using Containers, Functions, and Data to Build Next-generation Applications, ser. System administration. O'Reilly Media, 2019.
2. C. Davis, Cloud Native Patterns: Designing change-tolerant software. Manning Publications, 2019.
3. N. Kratzke and P.-C. Quint, "Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study," Journal of Systems and Software, vol. 126, pp. 1–16, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/ S0164121217300018
4. "The twelve factor app," https://12factor.net, last seen on March, 2021.
5. M. Senapathi, J. Buchan, and H. Osman, "Devops capabilities, practices, and challenges: Insights from a case study," in Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, ser. EASE'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 57–67. [Online]. Available: https://doi.org/10.1145/3210459.3210465
6. J. Wettinger, V. Andrikopoulos, F. Leymann, and S. Strauch, "Middlewareoriented deployment automation for cloud applications," IEEE Transactions on Cloud Computing, vol. 6, no. 04, pp. 1054–1066, oct 2018.
7. P. Duvall, S. Matyas, and A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk, 1st ed. Addison-Wesley Professional, 2007.
8. J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 1st ed. Addison-Wesley Professional, 2010.
9. M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," IEEE Access, vol. 5, pp. 3909–3943, 2017.
10. S. Jain, Linux Containers and Virtualization: A Kernel Perspective. Apress, 2020.

11. C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: A state-of-the-art review," IEEE Transactions on Cloud Computing, vol. 7, no. 03, pp. 677–692, jul 2019.
12. S. Daya, N. Van Duy, K. Eati, C. Ferreira, D. Glozic, V. Gucer, M. Gupta, S. Joshi, V. Lampkin, M. Martins et al., Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach. IBM Redbooks, 2016.
13. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," IEEE Software, vol. 33, no. 3, pp. 42–52, 2016.
14. "Cloud Native Computing Foundation," https://www.cncf.io, last seen on January 2021.
15. "Cloud Native LandScape," https://github.com/cncf/landscape, last seen on March, 2021.
16. R. Kalman, "On the General Theory of Control Systems," IRE Transactions on Automatic Control, vol. 4, pp. 110– 110, 01 1960.
17. J. Kosinska and K. Zieli ´ nski, "Autonomic management framework for ´ cloud-native applications," Journal of Grid Computing, vol. 18, no. 4, pp. 779–796, 2020.
18. N. Kratzke, "Cloud-native observability: The many-faceted benefits of structured and unified logging—a multi-case study," Future Internet, vol. 14, no. 10, 2022. [Online]. Available: https://www.mdpi.com/1999-5903/14/10/274
19. N. Marie-Magdelaine, "Observability and resources managements in cloud-native environnements," Theses, Université de Bordeaux, Nov. 2021. [Online]. Available: https://theses.hal.science/tel-03486157
20. K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," Information and Software Technology, vol. 64, pp. 1–18, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/ S0950584915000646.