2023 Global Conference on Information Technologies and Communications (GCITC)
Karnataka, India. Dec 1-3, 2023

2023 Global Conference on Information Technologies and Communications (GCITC) | 979-8-3503-0816-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/GCITC60406.2023.10426195

Circuit Breaker: A Resilience Mechanism for Cloud Native Architecture

1st Norton Stanley S A
Department of Computer Science
CHRIST (Deemed to be University)
Bangalore, India
0000-0002-5391-2631

2nd Shoney Sebastian
Department of Computer Science
CHRIST (Deemed to be University)
Bangalore, India
0000-0002-1296-884X

***Abstract*—Over the past decade, the utilization of cloud native applications has gained significant prominence, leading many organizations to swiftly transition towards developing software applications that leverage the powerful, accessible, and efficient cloud infrastructure. As these applications are deployed in distributed environments, there arises a need for reliable mechanisms to ensure their availability and dependability. Among these mechanisms, the circuit breaker pattern has emerged as a crucial element in constructing resilient and trustworthy cloud native applications in recent times. This research article presents a comprehensive review and analysis of circuit breaker patterns and their role within cloud-native applications. The study delves into various aspects of circuit breakers, encompassing their design, implementation, and recommended practices for their utilization in cloud native applications. Additionally, the article examines and compares different circuit breaker libraries available for employment in modern software development. The paper also presents a concept for improving the circuit breaker pattern, which will be pursued in our upcoming research.**

***Index Terms*—Cloud Native Applications, Microservice Patterns, Circuit Breaker, Fault Tolerance, Resilience**

I. Introduction

Business systems need to be complex, responsive, innovative, adaptable to changes, resilient, and scalable to facilitate strategic transformations and to accelerate an organization's growth. To achieve these organizational and business goals, companies have been executing application modernization programs, which often involve critical decisions such as whether to refactor or rebuild existing applications, in addition to considerations such as replatforming, replacing, or shifting. To design applications in newer cloud native environments, it is crucial to take into account the fundamental principles, patterns, and best practices of system architecture.

Legacy software have been in use for many decades. These may be outdated or no longer supported by the original developers. However, many organizations still continue to use legacy software since the applications may be crucial for the organization's day-to-day operations. Legacy systems may also be used since there is a lack of available resources to replace them with newer technologies. A system may inherit legacy governance if it replicates legacy functions, even if newer technologies were used but the underlying business processes were not improved [2].

Cloud Native Applications (CNAs) are software applications that have been designed and built to run natively on cloud-based platforms and systems. CNAs have been designed to take full advantage of the various attributes that are related to cloud-native technologies, including containerization, microservices, and serverless computing, which allow them to be highly agile, scalable, flexible, and resilient [1]. The benefits make CNAs the best viable option for modern, dynamic workloads for several processes including web and mobile applications, big data analytics, and machine learning.

A. Background

Compared to traditional legacy systems, CNAs are designed to accommodate modular, decentralized, and loosely-coupled microservices. These microservices may be deployed independently and scaled. Application Programming Interface (API) [23] can be used for these microservices to interact with each other. The microservices can be scaled up or scaled down depending on the needs of the software or to meet the system demands. In addition to microservices architecture, CNA also typically uses containerization technology, such as Docker, to package and deploy the application and its dependencies. These applications can be easily deployed and managed in different cloud environments, such as public or private clouds [3]. This allows CNAs to be highly adaptable to changing workloads, and to be able to handle large volumes of traffic and data.

Resiliency is a critical aspect of cloud-native applications [4], [5]. CNAs are designed to be highly resilient, with the ability to maintain availability and performance even in the face of failures or disruptions. The resiliency of a system is determined by its capacity to manage failures and bounce back from them. There are several key elements that contribute to the resiliency of cloud-native applications, including:

Microservices Architecture: CNAs are typically designed using a microservices architecture, where the application is broken down into smaller, independent services. This

architecture enables individual services to be developed, deployed, and managed independently, which reduces the risk of a single point of failure within the system.

Containerization: CNAs are often deployed using containerization technology, such as environment, which enhances portability and enables easy scaling up or down.

Distributed Systems: CNAs are often designed as distributed systems, where individual services communicate with each other using APIs. This approach can help minimize the impact of failures by isolating failures to individual services and enabling other services to continue functioning normally.

Overall, the resiliency of CNA is essential for ensuring high availability and performance in the cloud. By designing applications by considering these key elements, organizations can improve the reliability and resilience of their applications and minimize the impact of disruptions [6]. Resiliency pattern helps to prevent failures from cascading and ensures that a system remains functional in the event of dependency failures [7]. There are several resiliency patterns including timeout pattern, retry pattern, bulkhead pattern, rate limiter pattern, circuit breaker (CB) pattern and others [8].

Generally, in distributed systems, the probability of failure is quite large due to improper network connections or timeouts that preemptively halts a service. Although several resilience patterns, mentioned above, have been developed to address the issue of service failure with respect to CNAs, none of them have stood out as firmly as the CB pattern. In order to drive home this point, consider the case of resilience patterns such as the "Retry Pattern". The pattern attempts to reach out to the service by creating as many "retries" as possible, leading to a software kicking in "Denial-of-service" attacks within itself. In cases where the probability of successful access to the service is minimal, retry patterns do not fit the requirements. This is where CB could be the potential game-changer. When the system detects a failure and understands that any further attempts will only lead to more failure, the CB pattern can easily prevent the application from any further retry attempts.

CB has emerged as a critical component in building reliable and resilient CNAs in recent years [9]. CB is a design pattern used in modern software systems to handle faults and prevent cascading failures. It is emerging to be one of the most reliable, easy to access, and flexible methods of microservice architecture (MSA). The interesting thing about CB is that it is merely structural - meaning, the operations of the services that are offered are not altered in any way [10]. Since their adoption into MSA may prove to be complex, they are capable of influencing the way communication processes work within a system. The CB pattern is widely adopted in various fields of software engineering, such as microservices, cloud computing, edge computing, and Internet of Things (IoT). In microservices, the CB pattern is used to protect the system from failures and minimize downtime. Several studies have proposed adaptive CB mechanisms that can dynamically adjust the CBs' settings based on the system's performance. CB can basically detect failures and encapsulate the logic to prevent such failures from recurring all the time. Distributed CB systems that can detect and isolate faulty microservices to prevent cascading failures were also developed. Algorithm 1 vividly illustrates the operational mechanisms of circuit breaker patterns.

B. Outline and Our Contributions

In this paper CB patterns are reviewed and analyzed. The role of CB patterns in cloud-native applications are considered for the review. The various aspects of CB including their design, implementation, and best practices for use in CNAs are also discussed. Some of the recent research works which explore new CB design and implementation to improve resilience and fault tolerance of CNAs are discussed in detail. Further, various CB libraries that are available for use in modern software development are also compared. The paper is organized in the following structure. Section II outlines the research methodology employed. Section III covers the exploration of circuit breaker patterns, including their design, implementation, and various roles within different applications. Additionally, it encompasses a comprehensive comparison of the libraries available for the implementation of circuit breaker patterns. In section IV we outline the possibilities for enhancing the circuit breaker to bolster its resilience in the future. In conclusion, section V provides an overview of the paper's discoveries and hints at prospective directions for further investigation. It also presents a framework for enhancing the circuit breaker, a topic that we intend to explore in our upcoming research endeavors.

II. Systematic Literature Review

In [11], a systematic and meticulous method for performing literature review was introduced. A systematic literature review (SLR) based on this proposed methodology has been conducted in this research work to introduce the current state of research in the field of CNAs and circuit breaker pattern to develop potential methods that could improve the state-of-the-art.The article has been systematically categorized to help one in understanding the scope of research in the field. SLRs are capable of pointing directly at the problems that are faced by a particular area of study. They are designed to make evaluation and interpretation of the available literature easy. SLRs are generally categorized into three phases including:

- Planning
- Conduction
- Reporting (Analysis)

A. Planning

In the planning phase, the most relevant research questions are formulated based on the requirements of this research study. This study aims at providing a comprehensive review of methods that focus on the use of CB in CNSs. Hence, the research questions have been curated in such a way that they answer most of the commonly posed queries related to CB and microservices.

B. Research Questions

In order to perform an extensive literature review, several research questions were meticulously formulated in order to corroborate with the findings of this research work. These research questions are discussed below.

RQ1: How can CB be used to improve microservices architecture?
This RQ aims to identify how CB can be used to improve microservices architecture. This question allows the research work to identify the various methods that have been proposed using CB.

RQ2: How many places circuit breaker occurrences were present in academia and industry?
Provides an understanding of the familiarity and usage of circuit breaker patterns across industry and academia.

RQ3: What kind of applications can use circuit breaker pattern?
Provides an understanding of the variety of applications that circuit breaker patterns can be applied to.

RQ4: What are the various tools that are present to implement circuit breaker patterns?
Gives an overview to researchers and developers on the various tools that are available to implement circuit breaker and the differences among them.

C. Search Strategy

Multiple digital libraries were scoured to identify and compile relevant literature with respect to MSA and resiliency patterns in order to understand how CB patterns held their own. Utmost care was taken to ensure that the collected literature corresponded to CB patterns and provided answers to all of the research questions posed in the earlier section. All sources that were chosen for this extensive review were related to microservices, CB patterns, software deployment, and resiliency patterns in MSA. Table I details the various digital sources that were extensively used for the identification of relevant literature. To search for the most relevant literature, a list of search terms, that were chosen through mixing and matching, and trial and error, that could be used interchangeably for maximum efficiency were defined. The search terms were then split into several groups, which are listed in Table II. Boolean operators were used along with the search terms to enhance the search options. The search terms were used in combination with each other to improve the chances of obtaining potentially crucial research articles. The research papers that were collected were mostly published between 2000 to 2023.

TABLE I
THE LIST OF SELECTED SOURCES USED IN THIS REVIEW

Source	Type Of Material	Link to the libraries
Google Scholar	Digital Library	scholar.google.co.in/
IEEE Xplore	Digital Library	ieeexplore.ieee.org/
Science Direct	Digital Library	sciencedirect.com/
Wiley Online Library	Digital Library	https://www.wiley.com/en-in
Scopus	Digital Library	scopus.com/
Springer Link	Digital Library	springer.com/
IGI Global	Digital Library	IGI-global.com/
ACM Digital Library	Digital Library	https://dl.acm.org/

D. Inclusion and Exclusion Criteria

From the previous phases, a number of articles were procured. Since all of them were not important to the research work and did not fit the eligibility criteria, an inclusion and exclusion criteria was added to systematically scale-down the number of articles that are to be pushed to the next phases. These rules were devised in such a way that the selected articles provided solutions to the research questions that were formed earlier. The inclusion and exclusion criteria has been categorized into three sections including the abstract inclusion, full-text inclusion, and full-text quality.

Abstract inclusion: In this stage of inclusion, the articles were selected based on the contents of their abstracts.

Full-text inclusion: Once the abstract inclusion stage was completed, the articles were thoroughly reviewed. If the text discussed the topics relevant to this research work and contained more information on the keywords that were presented in Table II, the article was included for further scrutiny.

Full-Text Quality: In the final stage of the inclusion and exclusion criteria, the quality of the research articles that were shortlisted were verified. The quality of the research articles were calculated based on several quality checking measures including research questions such as,

QC1: Does the paper clearly discuss about the state of the art
QC2: Does the paper discuss CB patterns?
QC3: Does the paper propose the use of CB patterns to ensure resilience in MSA?

Exclusion criteria: Several exclusion criteria were devised to easily weed out sources that were not relevant to the research work undertaken. The exclusion criteria was defined as follows,

- Articles that are of non-English origin are excluded
- Articles with duplicated results are excluded

TABLE II
SEARCH TERMS USED FOR ONLINE SEARCHES

Source	Group 1	Group 2	Group 3	Group 4
Term 1	Microservices	Patterns	Resiliency	IoT
Term 2	Microservices-based systems	Design patterns	Reliability	Cloud
Term 3	Microservices Architecture	Pattern identification	Security	Edge
Term 4		Microservices pattern	Serverless	
Term 5		Circuit breaker patterns		

- Articles that only mention CB patterns in passing and not in detail are also excluded.
- Articles that discuss CB patterns but do not provide any sort of applications nor mention how it works are excluded.

E. Data collection and analysis

In this phase, the selected articles are reviewed in detail and a comparison is drawn between all of the methods chosen for introspection. The research questions that have been formed in the previous phases act the basis for this data collection process. In order to ensure the quality of the data collected and discussed, only the most relevant sources are selected so that peer-reviewed research articles can be obtained. The articles that were used for the collection of relevant data were published between the period of 2000-2023. The data extracted from the literature is discussed in Table III.

F. Conduction

This is the phase where the actual action is performed. The primary texts and other relevant sources of material are chosen for evaluation. These texts are based on the methodology developed through the previous phase. The search process is completed using the terms mentioned in Table II. The collected material is then organized into categories based on the type of research problems they address.

G. Reporting (Analysis)

Once the planning and conduction stages have been completed satisfactorily, it is important to form conclusions from the data that has been collected from the literature review conducted. The systematic review that has been undertaken allows the authors to choose, analyze and combine the results that discuss a chosen theme - in this case, understanding CB patterns in microservices and their applications. Through the systematic literature review, a non-biased system has been designed to gather data about several areas of interest so that any outcomes may be used in a variety of applications. The research questions that have been formed in the earlier phases have been answered methodically through the literature review that has been performed. The research questions can be understood by deriving data from the different articles that have been shortlisted. A list of sources that can be used for information on the relevant questions has already been discussed in Table I.

III. RESULTS AND FINDINGS

An extensive literature review that covers a vast majority of source material discussing resilience patterns in microservices and the role of CB in developing reliable, fault tolerant and scalable microservice architecture has been performed in this paper. The results obtained from the literature review that has been conducted is discussed in this section.

In [21], the authors present a cloud native architecture that is designed using microservice patterns. The reliability and scalability of this architecture, in the context of an automotive domain, was evaluated in a laboratory experiment. CB patterns were used in the proposed architecture and it took care of resiliency and reliability. Moreover, it was identified that CB could stop cascading failures to other services. This proof aligns with this article's proposal of using the CB mechanism for cloud native architecture.

In [13] the authors discuss the application of resiliency patterns to enhance the reliability of microservice-based distributed systems has garnered significant interest. The paper introduces a model-based analysis of two well-known resiliency patterns, namely Retry and Circuit Breaker, within a simple client-service interaction scenario using the PRISM model checker. The analysis reveals that when properly configured, both patterns effectively alleviate resource contention at the client-side compared to continuously retrying failed requests. The execution time only experiences a moderate increase. However, it is important to acknowledge the limitations of this work. The models solely capture the behavior of a single client service interacting with a single target service, omitting network failures and delays intentionally to maintain clarity and mitigate state explosion during model checking. Additionally, the analysis results have not been empirically validated. Furthermore, this review excludes the examination of other popular resiliency patterns based on asynchronous communication, such as Event Sourcing and Queue-Based Load Leveling.

In [24] the review paper introduces a formal approach for modeling microservices architectural patterns using Event-B. The primary objective of this approach is to address the challenges posed by natural language ambiguity, which can lead to misunderstandings or misapplications of these patterns. By providing formal models, the approach facilitates a deeper understanding of the runtime behavior of the patterns and

TABLE III
ITEMS AND DESCRIPTIONS OF THE DATA EXTRACTION FORM

Item	Description
Title	Selected article's title
Source of Publication	Journal, conference proceeding, book chapter or other source where the article was published
Year of Publication	The year in which the article was published
Focus of Research	The area of focus of the selected research article
Scope of Solution	How much of the problems mentioned in the study has been addressed and the global scope of the solution offered
Methodology used	The tools and techniques that were used for the implementation of the proposed method
Parameters considered	The parameters that were the focus of the selected research article for the proposed method's implementation
Advantages	The pros of the proposed research work
Disadvantages	The cons of the proposed research work
Result Validation	The way in which the results have been presented and discussed, especially with respect to other proposed methods.

enables the development of correct-by-construction systems.

In [19], a framework to support fault tolerance has been designed. The framework consisted of two microservices - real-time Fault tolerance and predictive fault tolerance, that made use of machine learning to identify patterns of failure and predict the probability of future failures in order to mitigate them. However, a RESTful architecture is employed to test the proposed method. This is a drawback since RESTful architecture provides strong coupling between the components. This meant that the failures could not be handled in isolation.

Cloud computing is another field where the circuit breaker pattern is widely used to enhance the reliability and fault tolerance of cloud services. Microservices have started to become the novel architectural standard when it comes to facilitating migrating on-premise architecture to cloud environments, even though the task is non-trivial. Although microservices have distribution complexities, if it is adopted into a context that requires high flexibility in terms of scalability and availability, microservices can get the job done [15].

In [20], the authors have tried to explore ways in which the CB pattern can be used as a communication intermediary between IoT nodes. A taxonomy that helps in the representation and validation of CB patterns that is customized for use in IoT applications has been proposed. An experimental study using a prototyped IoT application was also part of this research work.

In [12], the focus is on the examination of the circuit breaker pattern in the context of IoT. The authors aimed to address the increasing belief that certain practices of Microservices Architecture (MSA) hold promise for IoT, given the anticipated complexities of future IoT applications. The paper's primary contribution lies in the establishment of a taxonomy that encompasses various aspects related to the design and implementation of the circuit breaker pattern specifically tailored to IoT applications. Furthermore, the authors conducted an experimental validation, utilizing a prototype of a traffic light system, to assess the benefits of this pattern. The obtained results demonstrated numerous advantages for this specific application, particularly highlighting the circuit breaker's ability to significantly enhance performance, availability, and accuracy. These findings underscore the potential value of the circuit breaker pattern in improving the overall performance and reliability of IoT systems, thereby reinforcing its relevance in the IoT domain.

In [16], microservices patterns are combined with reactive systems in order to tackle problems of interoperability, scalability, and adaptability, among others, in large-scale IoT. The proposed method was implemented in a real-time smart agriculture domain and the results have shown that this method improves availability of IoT applications.

CB patterns have been proposed for use with IoT applications in [12]. The authors focuses on exploring the circuit breaker pattern in the context of IoT. The authors address the growing belief that Microservices Architecture (MSA) practices hold promise for IoT applications. The paper contributes by establishing a taxonomy specifically tailored to IoT applications, encompassing various design and implementation aspects of the circuit breaker pattern. Through experimental validation using a traffic light system prototype, the paper demonstrates significant benefits of the circuit breaker pattern, including enhanced performance, availability, and accuracy. These findings highlight the value of the circuit breaker pattern in improving IoT system performance and reliability, reaffirming its relevance in the IoT domain.

[14] also deals with IoT, patterns and best practices that are used in microservices. The paper provides an overview of emerging patterns and best practices in the microservice approach and their relevance to the internet of things (IoT). The study explores aspects such as self-containment, service versioning, monitoring, fault handling, container technology, and the choice between orchestration and choreography for service composition. The paper emphasizes that many microservice best practices and patterns such as circuit breaker are already present in IoT, albeit not widely adopted. By incorporating microservice considerations, IoT applications can leverage distributed services from various vendors,

enabling the creation of robust applications.

In [17], the authors discuss making use of microservices for a smart city project. The authors discussed a service-based platform to increase the energy efficiency of a city. Although resilience and scalability were discussed as advantages, an evaluation methodology to prove these benefits was not presented.

In [18], a method for reengineering an IoT middleware which was developed originally using a monolithic architecture has been proposed. Benefits such as failure isolation, elasticity, environmental reproducibility, reliability and availability of IoT applications were discussed. However, an evaluation methodology to prove these benefits was not presented.

In [25] the authors focus on the usage of resiliency patterns in the realm of serverless computing, several design patterns have emerged to address common challenges. They provide a concise overview of key concepts such as retries, circuit breakers, bulkheads, language agnosticism, and warming strategies within the context of the serverless framework. These patterns have gained significance based on their alignment with proposals derived from observing issues in complex system case studies. The paper shows that it is evident that these design patterns play a crucial role in ensuring the robustness and consistency of serverless architectures throughout the development process. The authors recommend that these patterns be implemented as common libraries or code from the early stages of project conception. This approach ensures their effective and widespread adoption across the entire system. The also bring in the notion that developers and architects should possess a solid understanding of the principles underlying these design patterns.

Some of the popular libraries for the implementation of circuit breakers have also been identified through the extensive literature review. The results have been mentioned in Table IV.

Algorithm 1: Circuit Breaker Implementation

```
Input  : Circuit breaker state, Failure count
Output: Response or circuit breaker state change
Function executeRequest:
if circuit breaker state is closed then
    Execute the request;
    if request is successful then
        Reset failure count to 0;
    else
        Increment failure count;
        if failure count exceeds threshold then
            tripCircuitBreaker();
    Exception Increment failure count;
    if failure count exceeds threshold then
        tripCircuitBreaker();
if circuit breaker state is open then
    if time elapsed since last failure exceeds timeout
     then
        Attempt to execute a test request;
        if test request is successful then
            Reset failure count to 0;
            Reset circuit breaker state to closed;
    else
        Continue in open state;
Function tripCircuitBreaker:
Set circuit breaker state to open;
Set timeout for the circuit breaker;
Function resetCircuitBreaker:
Set circuit breaker state to closed;
Reset failure count to 0;
```

IV. Future Directions

The future advancements in the field of circuit breakers encompass various areas of potential investigation. One particular research avenue involves the dynamic control and management of circuit breakers from external components, as discussed by the authors in [22]. Our paper proposes a novel method and approach to dynamically redirect the circuit through an alternative path in the event that the primary path is inaccessible or unavailable. This ensures that the functionality served by the primary path remains accessible through the alternative path. Our forthcoming work aims to enhance the default implementation of the circuit breaker by introducing an alternating state when the primary path is unavailable. Instead of immediately transitioning to the open state, the circuit breaker will enter an alternating state. The determination of the alternative path to fulfill the request will be computed dynamically using an alternating rule engine. Further architectural and implementation details of the proposed alternating circuit breaker will be explored in our future research endeavors.

V. Conclusion

In conclusion, this review paper explored the significance of the Circuit Breaker as a resilience mechanism in Cloud Native Architecture. this paper offers a comprehensive overview of the Circuit Breaker mechanism, catering to the needs of developers and researchers. By delving into the fundamental concepts, principles, and practical implementations, this paper equips readers with a solid understanding of Circuit Breaker's role in enhancing the resilience of distributed systems. By examining the existing literature and case studies, different libraries, this paper showcases the effectiveness of Circuit Breaker in enhancing the reliability, fault tolerance, and overall resilience of distributed systems. Furthermore, the paper shed light on the integration of Circuit Breaker with other resilience

TABLE IV
COMPARISON OF FAULT TOLERANCE LIBRARIES

Library	Language	Async Support	Metrics	Open Source	Retry	Bulkhead
Hystrix	Java	Yes	Yes	Yes	Yes	Yes
Resilience4j	Java	Yes	Yes	Yes	Yes	Yes
Polly	.NET	Yes	Yes	Yes	Yes	Yes
Akka	Scala/Java	Yes	No	Yes	Yes	Yes
Circuit	Ruby	Yes	No	Yes	Yes	No
Failsafe	Java	Yes	No	Yes	Yes	Yes
Pybreaker	Python	Yes	No	Yes	Yes	No
Turbine	Java	No	No	Yes	No	No
go-circuitbreaker	golang	No	No	Yes	No	No

patterns, such as retries, fallbacks, and load balancing. It emphasized the importance of adopting a holistic approach to resilience engineering by leveraging multiple mechanisms in conjunction with Circuit Breaker. Overall, this review paper contributes to the understanding of Circuit Breaker as a crucial component of Cloud Native Architecture. It underscores the significance of implementing resilient systems in modern cloud environments and provides valuable insights for practitioners and researchers in the field. As organizations increasingly embrace Cloud Native Architectures, the adoption of Circuit Breaker as a resilience mechanism becomes vital to ensure the stability and availability of applications and services. Moreover, the resolution of issues related to distributed circuit breakers and the implementation of effective management techniques offer promising avenues for future exploration. By focusing on these enhancements, researchers can contribute to the advancement of circuit breaker technology and its seamless integration into contemporary distributed systems. The insights provided in this paper serve as a valuable resource for those seeking to leverage Circuit Breaker in their development projects or contribute to the advancement of resilience engineering through further research.

REFERENCES

[1] D. Gannon, R. Barga, and N. Sundaresan. (2017). Cloud-Native Applications. IEEE Cloud Computing. 4. 16–21

[2] Z. Irani, R. M. Abril, V. Weerakkody, A. Omar, and U. Sivarajah (2023). "The impact of legacy systems on digital transformation in European public administration: Lesson learned from a multi case analysis, Government Information Quarterly, 40 (1).

[3] C. Pahl, and P. Jamshidi. (2016). "Microservices: A Systematic Mapping Study" in Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016), 1, 137-146.

[4] G. Toffetti, S. Brunner, M. Blöchlinger, J. Spillner, and T. M. Bohnert. (2017). "Self-managing cloud-native applications: Design, implementation, and experience," Future Generation Computer Systems, 72, 165–179.

[5] M. Szalay, P. Mátray, and L. Toka. (2021). "State Management for Cloud-Native Applications," Electronics, 10(4), 423.

[6] P. Štefanič et al., "SWITCH workbench: A novel approach for the development and deployment of time-critical microservice-based cloud-native applications," Future Generation Computer Systems, vol. 99, pp. 197–212, Oct. 2019

[7] S. R. Goniwada, "Microservices Architecture and Design," in Cloud Native Architecture and Design, Berkeley, CA: Apress, 2021, pp. 191–240. Accessed: Apr. 05, 2023. [Online]

[8] S. Frank, L. Wagner, A. Hakamian, M. Straesser and A. van Hoorn, "MiSim: A Simulator for Resilience Assessment of Microservice-Based Architectures," 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), Guangzhou, China, 2022, pp. 1014-1025.

[9] A. 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, May 2016.

[10] Molchanov, H., & Zhmaiev, A. (2018). CIRCUIT BREAKER IN SYSTEMS BASED ON MICROSERVICES ARCHITECTURE. Advanced Information Systems, 2(4), 74–77.

[11] B. Kitchenham, and S. Charters, "Guidelines for performing systematic literature reviews in software engineering version 2.3," Tech. Rep., Keele University and University of Durham, 2007.

[12] G. Aquino, R. Queiroz, G. Merrett, and B. Al-Hashimi. (2019) "The Circuit Breaker Pattern Targeted to Future IoT Applications" Service-Oriented Computing. Lecture Notes in Computer Science, ICSOC 2019, LNCS 11895, 390–396.

[13] N. C. Mendonca, C. M. Aderaldo, J. Camara and D. Garlan. (2020). "Model-Based Analysis of Microservice Resiliency Patterns", 2020 IEEE International Conference on Software Architecture (ICSA), Salvador, Brazil, 114-124.

[14] B. Butzin, F. Golatowski, and D. Timmermann. (2016). "Microservices Approach for the Internet of Things", IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA) - Microservices approach for the internet of things, 1–6.

[15] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Migrating to Cloud-Native Architectures Using Microservices: An Experience Report", In: Celesti, A., Leitner, P. (eds) Advances in Service-Oriented and Cloud Computing. ESOCC 2015., Communications in Computer and Information Science, vol 567. Springer, Cham., 2016

[16] C. Santana, L. Andrade, F.C Delicato, F.C. et al "Increasing the availability of IoT applications with reactive microservices", Service Oriented Computing and Applications 15, 109–126, 2021.

[17] A. Krylovskiy, M Jahn, E. Patti, "Designing a smart city inter- net of things platform with microservice architecture", In: 2015 3rd international conference on future internet of things and cloud, pp 25–30, 2015.

[18] LMC Martins, FLDC Filho, RTDS Júnior, WF Giozza, JAPC da Costa, "Increasing the dependability of IoT middleware with cloud computing and microservices", In: Companion proceedings of the 10th international conference on utility and cloud computing, UCC '17 companion, association for computing machinery, New York, pp 203–208, 2017.

[19] A Power, G Kotonya, "A microservices architecture for reactive and proactive fault tolerance in IoT systems", In: 2018 IEEE 19th international symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), pp 588–599, 2018

[20] G Aquino, R Queiroz, G Merrett, B Al-Hashimi, "The circuit breaker pattern targeted to future iot applications", In: Yangui S, Bouassida Rodriguez I, Drira K, Tari Z (eds) Service-oriented computing. Springer, Cham, pp 390–396, 2019.

[21] A. Banijamali, P. Jamshidi, P. Kuvaja, M. Oivo, "Kuksa: A Cloud-Native Architecture for Enabling Continuous Delivery in the Automotive Domain", 455-472, 2019, arXiv:1910.10190 [cs.SE].

[22] Falahah, K. Surendro, and W. D. Sunindyo, "Circuit breaker in microservices: State of the art and future prospects," IOP Conference Series: Materials Science and Engineering, vol. 1077, no. 1, p. 012065, 2021. doi:10.1088/1757-899x/1077/1/012065

[23] V. Kanvar, R. Jain, and S. Tamilselvam, "Handling communication via apis for microservices for ICSE 2023: IBM Research," IBM

Research Publications, https://research.ibm.com/publications/handling-communication-via-apis-for-microservices (accessed Jun. 23, 2023).
[24] S. Vergara, L. González and R. Ruggia, "Towards Formalizing Microservices Architectural Patterns with Event-B," 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), Salvador, Brazil, 2020, pp. 71-74, doi: 10.1109/ICSA-C50368.2020.00022.
[25] D. Bardsley, L. Ryan and J. Howard, "Serverless Performance and Optimization Strategies," 2018 IEEE International Conference on Smart Cloud (SmartCloud), New York, NY, USA, 2018, pp. 19-26, doi: 10.1109/SmartCloud.2018.00012.