Proceedings of the Second International Conference on Edge Computing and Applications (ICECAA 2023)
IEEE Xplore Part Number: CFP23BV8-ART; ISBN: 979-8-3503-4757-9

SQL vs NoSQL Databases for the Microservices: A Comparative Survey

Madhavi Soni
Department of CE and IT
Veermata Jijabai Technological Institute
Matunga, Mumbai, India

Varshapriya Jyotinagar
Department of CE and IT
Veermata Jijabai Technological Institute
Matunga, Mumbai, India

2023 2nd International Conference on Edge Computing and Applications (ICECAA) | 979-8-3503-4757-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICECAA58104.2023.10212190

***Abstract*—Microservices is a software architecture technique that divides large and intricate applications into smaller, more manageable units. This method facilitates faster development, scalability, and enhanced fault tolerance as microservices can be developed, deployed, and scaled independently. Nevertheless, data management can be a significant challenge when using microservices. This survey explores the comparison between SQL and NoSQL databases in the context of microservices architecture. It investigates the advantages and challenges associated with each type of database and their suitability for supporting microservices-based applications. The survey analyzes factors such as data consistency, scalability, performance, and query capabilities. It highlights the strengths and weaknesses of SQL and NoSQL databases, considering their impact on maintaining data integrity and supporting distributed systems. By providing an overview of the characteristics and trade-offs of both database types, this survey aims to assist decision-making when choosing a suitable database solution for microservices-based applications.**

***Index Terms*—Software Design, Microservices Architecture, Monolithic System, Measuring Software, Evaluating Performance, Scalability, Cloud Computing.**

I. Introduction

The importance of flexibility in commercial software design cannot be overstated. The Microservice architecture offers a solution by designing and distributing software components on a smaller scale, enabling independent integration. By adopting the approach of separate deployment, specific services can be updated or replaced without causing disruptions to the entire system.

Although the concept of microservices offers various advantages in terms of flexibility, effectively handling databases in a microservices architecture presents notable difficulties. In monolithic applications, the MVC (Model-View-Controller) approach is commonly employed to separate the database from the application flow. However, in a microservices environment where services may not fully trust the data, ensuring data consistency becomes a complex task. It is essential to find a solution that does not compromise response time, especially for data-intensive web applications dealing with large datasets. Therefore, there is a need for a practical and comprehensive approach to database management within the context of microservices.

The drive to attain improved separation of concerns has been a fundamental catalyst in the evolution of software architecture. The notion of ”separation of concerns” pertains to the capability of breaking down and structuring systems into modules that exhibit logical coherence and loose coupling. This enables modules to conceal their internal complexities from one another while offering services through clearly defined interfaces [1], [2].

Microservices architecture has experienced a surge in popularity due to its ability to create scalable and distributed applications. Nonetheless, an important hurdle in microservices architecture revolves around ensuring effective data persistence across multiple services.

According to [3], in modern enterprise application development, two software engineering paradigms are prevalent: monolithic architecture and microservice-based architecture. The monolithic architecture follows a conventional approach in which the application is developed using a unified codebase that incorporates multiple services. These services do not possess the capability to function independently [4] and interact with external systems and end-users through various interfaces such as Web services, REST API, and HTTP(S)/HTML [5].

Large and complex enterprise applications typically employ a three-tier architecture. In the context of microservices, Martin Fowler explains in his article that microservices facilitate the establishment of individual databases for each service. Polyglot Persistence is a principle that enables the utilization of different database technologies or systems to handle diverse data types, either through separate instances of the same database technology or completely distinct database systems. To aid comprehension, a visual diagram is provided below, offering a clearer representation of the concept.

The research conducted in [6]-[8] characterizes the microservice architecture by breaking down a business domain into smaller and well-defined contexts. These contexts are implemented as separate, self-contained, and loosely coupled services that can be independently deployed. Notably, Netflix was an early adopter of microservices, having embraced this approach in 2009, even before it had a designated term. Software architects introduced the term ”microservices” in 2011, and it was formally presented at the 33rd Degree Conference in Krakow in 2012 [6].Following the publication of a blog by Lewis and Fowler

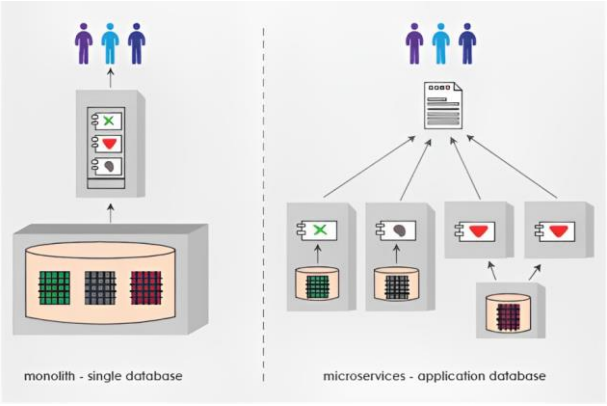

Fig. 1. Decentralized Data Management. Adapted from An Overview of Microservice Architecture." n.d. TO the NEW BLOG. https://www.tothenew.com/blog/an-overview-of-microservice-architecture-part-i/.

in 2014 [6] and Netflix's sharing of their successful transition expertise [9], microservices began to gain widespread recognition. These findings led to extensive research by both academic and industry experts, as evidenced by studies published in [4] and [10]-[18]. The increasing adoption of container technologies such as Kubernetes and Docker has played a significant role in the rising popularity of microservices, especially in cloud-native environments [23]-[25]. These advancements have contributed to the growing popularity of microservices in recent years.

Numerous global companies, including Coca-Cola, Zalando, Spotify, Uber, Airbnb, Amazon, Twitter, Groupon, LinkedIn and eBay, have successfully embraced microservices.

II. Microservices and Databases

Throughout human history, the preservation of events, experiences, and acquired knowledge has always held immense importance. Overcoming the challenge of data storage has been a significant obstacle faced by humanity. From ancient sculptures in caves to modern cloud storage, various technologies have been employed to document the history of our species. Engraving on stones, writing on leather and paper, and the evolution of technology have all played pivotal roles in the advancement of storage methods. These advancements have resulted in the rapid development of technology solutions for storing information.

Microservices performance can be measured by monitoring a number of metrics, including:

- Response time: The duration required for a microservice to provide a response to a given request.
- Throughput: The capacity of a microservice to process and handle a certain number of requests within a given time frame, typically measured in requests per second (RPS).
- Error rate: The percentage of requests that fail.
- Resource utilization: The utilization of computing resources such as CPU, memory, and network bandwidth by a microservice.

Typically, a database system serves as a centralized repository for storing and retrieving data when required. The adoption of databases as a substitute for file systems has emerged due to various limitations, including redundancy, consistency, concurrent access by multiple users, and more. Databases offer solutions to overcome these constraints and provide enhanced functionality for managing data efficiently.

Integrating databases into microservices architecture presents challenges when it comes to facilitating data sharing among independent services. To tackle this issue, multiple database alternatives have emerged, such as relational databases, NoSQL databases, and NewSQL databases, offering solutions for effective data management within the context of microservices architecture.

A. Types of Databases Used in Microservices

1) Relational Databases: Relational databases have been the traditional choice for microservices since they have been around for many years and offer several benefits such as consistency, durability, and reliability. However, they have limitations when it comes to scaling and performance.

Relational Database Management Systems (RDBMS) are designed with a predefined and fixed schema for organizing data in a tabular format, offering vertical scalability. These databases are particularly suitable for applications that demand complex querying capabilities. They can also address challenges related to Atomicity, Consistency, Isolation, and Durability (ACID) requirements. RDBMS supports a single data type, allowing for minor variations in the data.
Examples of RDBMS include MS-SQL, Postgres, Oracle, etc
Limitations of RDBMS :

- Scalability: SQL-oriented databases often lack built-in support for distributed data processing and storage. As a result, handling high volumes of data becomes challenging, requiring the use of powerful servers with significant computational capabilities. This can lead to expensive and undesirable solutions, as the need for specialized hardware increases to accommodate large data volumes.
- Flexibility and complexity: One of the primary reasons for adopting schema-less non-relational databases is the avoidance of a predefined data structure. This means that there is no need to explicitly define a schema for the stored data, allowing for flexible data storage without a predefined structure.

2) NoSQL Databases: NoSQL databases are capable of accommodating various data types, including structured, semi-structured, and unstructured data. They are specifically designed to address the limitations of relational database management systems, particularly when it comes to handling large volumes of data with high variability. NoSQL databases are known for their horizontal scalability, making them well-suited for applications that require hierarchical data storage.

NoSQL databases are specifically designed to handle unstructured data and offer greater scalability compared to relational databases. However, they may face challenges in maintaining transactional consistency, which is a key feature provided by relational databases. This limitation should be considered when evaluating the suitability of NoSQL databases for specific use cases.

Limitations of NoSQL :

- Data consistency: In general, NoSQL databases do not offer the same level of ACID guarantees (Atomicity, Consistency, Isolation, Durability) as SQL databases. As a result, ensuring consistent data across all nodes in the database can become challenging. This limitation may pose difficulties for applications that heavily rely on strong data consistency requirements.
- Schemaless design: NoSQL databases are typically schemaless, which means that the data structure is not defined in advance. This can make it difficult to enforce data integrity and to query the data efficiently.
- Heterogeneous data: NoSQL databases are designed to store heterogeneous data, which means that data of different types can be stored in the same database. This can make it difficult to manage the data and to query it efficiently.
- Security: NoSQL databases may not provide the same level of security as relational databases. This is because NoSQL databases are often designed to be open and scalable, which can make them more vulnerable to attack.
- Backup and recovery: NoSQL databases often have different backup and recovery requirements than relational databases. This can make it difficult to back up and recover data from a NoSQL database.
- Data modeling: NoSQL databases require a different data modeling approach than relational databases. Developers who lack familiarity with NoSQL data modeling techniques may find it challenging.
- Tool support: There is less tool support for NoSQL databases than for relational databases. This can make it difficult to develop and manage applications that use NoSQL databases.

3) NewSQL Databases: NewSQL databases are a newer type of database that combines the best of both worlds. They offer the scalability of NoSQL databases and the consistency and durability of relational databases. They are still relatively new, but they have gained a lot of traction in recent years.

NewSQL addresses the limitations of traditional SQL databases and can serve as an alternative to NoSQL in specific applications that require on-request analytics and decision-making with a strong emphasis on high consistency.

Examples of NewSQL include NuoDB, VoltDB, Clustrix, etc

Limitations of NewSQL :

- Applications that handle large volumes, surpassing a few terabytes, are not well-suited for this purpose.
- It lacks comprehensive compatibility with traditional SQL tools.

TABLE I
SQL VS NOSQL PERFORMANCE

Performance Aspect	SQL Database	NoSQL Database
Schema Flexibility	Rigid schema with predefined structure and strict data types	Flexible schema allowing dynamic schema changes
Querying Capabilities	Advanced SQL queries and complex joins	Varies depending on the NoSQL database type
Scalability	Vertical scalability (limited by hardware)	Horizontal scalability (ability to distribute data across multiple nodes)
Performance under High Volume Traffic	Generally optimized for transactional workloads and complex queries	Depends on the specific NoSQL database type and configuration
ACID Transactions	Full ACID compliance	Eventual consistency or relaxed consistency depending on the NoSQL database type
Data Integrity	Enforced by primary and foreign key constraints	Relaxed constraints and referential integrity
Join Operations	Supports complex join operations	Limited or no support for complex joins
Community & Ecosystem Support	Well-established community and extensive ecosystem of tools and libraries	Varied depending on the specific NoSQL database type

B. Benefits of Using Microservices

The implementation of microservices in the present time offers a multitude of IT and business advantages. Some of these benefits include:

1) Migrating legacy systems can be a time-consuming and expensive process. By avoiding this process altogether, organizations can save money and resources that can be used to improve their current systems or develop new ones.
2) Enabling independent releases, unlike traditional methods.
3) Enhancing modularity and reducing interdependencies, resulting in easier application development, understanding, and testing.
4) Allowing smaller teams to make their own decisions about how to develop and deploy new services.
5) Agilely adapting to shifting market conditions and developing new products that are accessible to a wide range of users, regardless of their preferred language or platform.

Moreover, microservices provide extensive support for development methodologies like DevOps, Agile, Continuous Delivery, and Deployment. Furthermore, they offer the advantage of independent deployment, which enhances resilience by preventing disruptions in other processes when an issue arises in a specific area.

III. RELATED WORK

X. J. Hong et al. [20] explores the performance aspects of RESTful API and RabbitMQ in the context of microservice web applications. The authors conducted experiments and

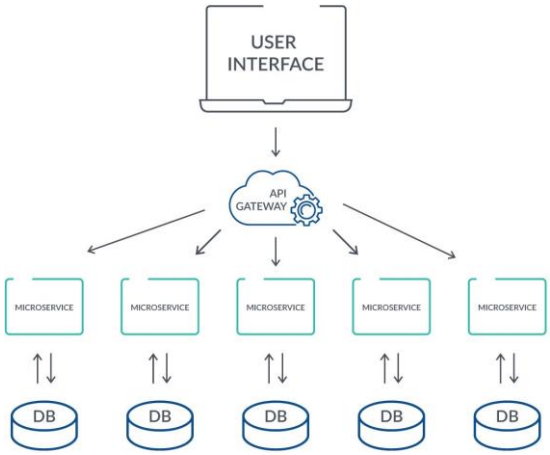

Fig. 2. A typical microservice ecosystem. Adapted from "Microservices and Databases: The Main Challenges." 2019. DB-maestro. July 10, 2019. https://www.dbmaestro.com/blog/database-automation/microservices-databases-challenges.

measurements to evaluate the response time and throughput of these two technologies. They compared the performance of RESTful API and RabbitMQ in scenarios involving various levels of message size and concurrency. The findings of the study provide insights into the strengths and limitations of these technologies in terms of handling communication and message exchange within microservice architecture. The research contributes to a better understanding of the performance characteristics of RESTful API and RabbitMQ, which can guide the selection and optimization of communication mechanisms in microservice-based web applications. The testing environment does not take into account many practical factors like throughput, response time that can make the test more accurate and reliable.

In a study conducted by G. Blinowski et al. [23], controlled experiments were carried out in three separate deployment environments, namely local, Azure Spring Cloud, and Azure App Service. The findings suggest that while microservice architecture offers various benefits, it may not be the most suitable choice for every context. In particular, for simple and small-sized systems that don't require extensive support for a high volume of concurrent users, a monolithic architecture appears to be a better alternative.

In their research, S.-P. Ma et al. [25] introduced MIADA (Microservices Identification using Analysis for Database Access), a scheme for identifying microservices that leverages historical data on database access requests. The MIADA system captures and analyzes this data to assist in selecting appropriate clustering methods and parameters. By identifying clusters of service endpoints, MIADA facilitates the creation of microservices. The effectiveness of MIADA was demonstrated through experiments conducted on two real-world scenarios, PlanApproval and CoCoME, where it exhibited its ability to accurately formulate microservices.

Khan et al. [27] presented PerfSim, a comprehensive performance simulator designed specifically for modeling and simulating the performance of microservice chains in cloud-native environments. The authors leverage performance tracing and monitoring tools to accurately model diverse microservices and their associated service chains within cloud-native orchestration platforms. PerfSim also enables the simulation of different resource management scenarios and placement policies, providing valuable insights into their effectiveness. The paper's findings contribute to the understanding and optimization of performance in cloud-native microservice architecture.

In their study, Maulidin et al. [28] introduced a novel approach for online integration of SQL and NoSQL databases using REST APIs. The effectiveness of this method was evaluated through its implementation on two furniture e-commerce sites, demonstrating favorable performance and scalability. It is worth noting that the method does require a certain level of expertise in REST API development, which can be considered as a limitation. Furthermore, the method has not been extensively tested on a large scale, indicating that its applicability may vary across different applications. Nonetheless, the authors express optimism regarding the potential value of this method as a useful tool for developers who require SQL and NoSQL database integration capabilities.

A. Hannousse and S. Yahiouche [30] presents a comprehensive literature survey on the topic of securing microservices and microservice architecture. The authors aim to identify and analyze the existing research in this area. Through a systematic mapping study, they gather and categorize relevant publications, highlighting the main security challenges and approaches in the context of microservices. The paper provides an overview of the different security aspects, such as authentication, authorization, communication security, and data protection. The findings of the study contribute to a better understanding of the current state of research in securing microservices and serve as a foundation for future research and development in this field.

In their comprehensive systematic review, Schmidt and Thiry [31] explored the existing literature on the identification and migration of microservices from monolithic systems, as well as the level of automation employed during the decomposition process. The review also highlighted the significance of non-functional factors that should be taken into account, such as design and modeling methods, performance considerations, and the availability of dynamic monitoring. The authors discovered that different studies utilized diverse approaches, with some collecting runtime data through tracking systems, while others derived microservices based on business models, including requirements models, business processes, and data flows.

Omar Al-Debagy et al. [32] presented a novel approach to decompose monolithic applications into microservices. The authors conduct a comprehensive literature survey to explore existing decomposition methods and identify their limitations. Based on their findings, they propose a new method that

combines domain-driven design principles and dependency analysis to ensure cohesive and loosely coupled microservices. The paper highlights the importance of considering both functional and non-functional requirements during the decomposition process. Through a case study, the authors demonstrate the effectiveness of their proposed method in designing scalable and maintainable microservices-based systems. The research contributes to the field by providing valuable insights and a practical approach for designing microservices architecture.

Amfiri [33] discusses a method for identifying microservices based on object-aware analysis. The author proposes an approach that considers the relationships between objects in a system to determine the boundaries of microservices. The paper highlights the importance of this approach in enabling finer-grained microservice design and improving system modularity. The methodology and results presented in the paper contribute to the field of microservices architecture and provide insights into effective microservice identification techniques.

To summarize, the microservice identification approaches discussed earlier have shown success in various aspects but lack consideration for database access characteristics. Given these considerations, the present study introduces a systematic approach and a complementary investigation aimed at identifying microservices through the analysis of database access records. This research recognizes the importance of the ”Database Per Service” pattern within the realm of microservices.

IV. Challenges Faced when using Databases in Microservices

Maintaining data consistency between microservices is a challenging task. In a monolithic architecture, all data is stored in a single database, and this makes it easy to maintain consistency. However, in a microservices architecture, each microservice has its own data store, and this can make it difficult to ensure that all data is consistent.

Within a microservices architecture, ensuring consistent data persistence across multiple services poses significant challenges. Each service may require access to shared data, making it difficult to maintain data consistency and integrity. Furthermore, managing data becomes more complex when services utilize different databases and data storage techniques.

When developing and deploying microservices, it is crucial to consider and address three primary parameters:

A. Read and Write Performance

When evaluating the read performance of microservices, commonly used metrics involve measuring the number of operations performed per second or considering a combination of query execution speed and result retrieval speed. The speed of retrieving results is dependent on the efficiency of data organization and indexing.

Regarding write performance, a simple metric is to assess the number of write operations carried out by the microservice within a second. Microservices that handle transient data collection and processing require databases that can efficiently handle thousands or even millions of write operations per second.

B. Efficiency

To provide seamless user experiences, it is essential to utilize a low-latency database, which can be achieved by deploying the microservice in close proximity to its corresponding database. When it comes to read and write operations, response times below 1 ms are generally regarded as low latency, whereas response times exceeding 10 ms are typically considered high latency.

In the context of microservices, it is crucial to minimize the database footprint while ensuring scalability. Furthermore, the development and deployment of microservice components should be efficient and rapid. This requires a database service that enables the on-demand creation of numerous instances per second.

C. Data Sharing

It is important to emphasize that the persistent data of each microservice should remain private and only accessible through its dedicated API. If multiple microservices were to share persistent data, careful coordination of schema changes would be required, leading to a notable slowdown in development. However, in many cases, there is a need for services to share data.

For instance, in scenarios where multiple services require access to user profile data, one approach could involve encapsulating the user profile data within a dedicated service that can be accessed by other services. Alternatively, an event-driven mechanism can be employed to replicate the data to each service that requires it, ensuring synchronized availability.

V. Conclusion

In conclusion, microservices architecture provides a strong approach to building scalable and resilient software systems. However, managing data within this architecture can present challenges, especially when dealing with large volumes of data. To ensure optimal performance and reliability in microservices architecture, it is important to employ suitable database types, effectively manage distributed transactions, utilize message queues, API gateways, and service meshes. Additionally, prioritizing the implementation of best practices for data security and failure management is crucial.

The choice between SQL and NoSQL databases should be based on the specific requirements of your application. If your application deals with structured data, requires complex querying, and strong consistency is crucial, an SQL database may be more appropriate. Conversely, if your application deals with unstructured or rapidly changing data, requires horizontal scalability, and can tolerate eventual consistency, a NoSQL database might be a better fit.

It’s important to note that performance can also be influenced by factors such as database design, indexing strategies, hardware resources, data volume, and specific use cases.

Benchmarking and performance testing against your application's workload and requirements are recommended to determine the most suitable database solution in terms of performance. In some cases, a hybrid approach, combining both types of databases in a polyglot persistence strategy, may offer the best of both worlds by leveraging the strengths of each database model for different microservices within your architecture. In summary, the future of database selection for microservices will involve a continuous focus on improvement and innovation to meet the evolving needs and requirements of microservices.

REFERENCES

[1] Przybyłek, Adam. 'Where the Truth Lies: AOP and Its Impact on Software Modularity'. In Fundamental Approaches to Software Engineering, edited by Dimitra Giannakopoulou and Fernando Orejas, 447–61. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[2] Przybyłek, Adam. 'An Empirical Study on the Impact of AspectJ on Software Evolvability'. Empirical Software Engineering 23, no. 4 (1 August 2018): 2018–50. https://doi.org/10.1007/s10664-017-9580-7.

[3] Lewis, James, and Martin Fowler. 2014. "Microservices." Martinfowler.com. 2014. https://martinfowler.com/articles/microservices.html.

[4] Bjørndal, Nicholas, Manuel Mazzara, Antonio Bucchiarone, Nicola Dragoni, and Schahram Dustdar. 'Migration from Monolith to Microservices: Benchmarking a Case Study'. The Journal of Object Technology 20 (01 2021): 3:1. https://doi.org/10.5381/jot.2021.20.2.a3.

[5] Terzic´, Branko, Vladimir Dimitrieski, Slavica Kordic´, Gordana Milosavljevic´, and Ivan Lukovic´. 'Development and evaluation of MicroBuilder: a Model-Driven tool for the specification of REST Microservice Software Architectures'. Enterprise Information Systems 12, no. 8–9 (October 2018): 1034–57. https://doi.org/10.1080/17517575.2018.1460766.

[6] J. Lewis and M. Fowler. (Mar. 2014). Microservices: A Definition of This New Architectural Term. https://martinfowler.com/articles/microservices.html

[7] C. Posta, Microservices for Java Developers: A Hands-on Introduction to Frameworks Containers. Newton, MA, USA: O'Reilly Media, 2016. https://www.oreilly.com/library/view/microservices-for-java/9781492042228/

[8] R. Rajesh, Spring Microservices. London, U.K.: Packt, 2016. https://hoclaptrinhdanang.com/downloads/pdf/spring/Spring Microservices.pdf

[9] A. Cockroft. (Aug. 2004). Migrating to Microservices. https://www.infoq.com/presentations/migration-cloud-native/

[10] Doroz˙yn´ski, Piotr, Adam Brzeski, Jan Cychnerski, and Tomasz Dziubich. 'Towards Healthcare Cloud Computing', 87–97, 01 2016. https://doi.org/10.1007/978-3-319-28564-1 _8.

[11] Soldani, Jacopo, Damian Tamburri, and Willem-Jan Heuvel. 'The Pains and Gains of Microservices: A Systematic Grey Literature Review'. Journal of Systems and Software 146 (09 2018). https://doi.org/10.1016/j.jss.2018.09.082.

[12] Vural, Hulya, Murat Koyuncu, and Sanjay Misra. 'A Case Study on Measuring the Size of Microservices', 454–63, 07 2018. https://doi.org/10.1007/978-3-319-95174-4 _36.

[13] Carvalho, Luiz, Alessandro Garcia, Wesley Assunc¸a˜o, Rafael de Mello, and Maria Lima. 'Analysis of the Criteria Adopted in Industry to Extract Microservices', 22–29, 05 2019. https://doi.org/10.1109/CESSER-IP.2019.00012.

[14] Carvalho, Luiz, Alessandro Garcia, Wesley Assunc¸a˜o, Rafael de Mello, and Maria Lima. 'Analysis of the Criteria Adopted in Industry to Extract Microservices', 22–29, 05 2019. https://doi.org/10.1109/CESSER-IP.2019.00012.

[15] Di Francesco, Paolo, Patricia Lago, and Ivano Malavolta. 'Architecting with Microservices: A Systematic Mapping Study'. Journal of Systems and Software 150 (2019): 77–97. https://doi.org/10.1016/j.jss.2019.01.001.

[16] Jaworski, Janusz, Waldemar Karwowski, and Marian Rusek. 'Microservice-Based Cloud Application Ported to Unikernels: Performance Comparison of Different Technologies', 255–64, 01 2020. https://doi.org/10.1007/978-3-030-30440-9 _24.

[17] Liu, Guozhi, Bi Huang, Zhihong Liang, Minmin Qin, Hua Zhou, and Zhang Li. 'Microservices: Architecture, Container, and Challenges', 629–35, 12 2020. https://doi.org/10.1109/QRS-C51114.2020.00107.

[18] Khazaei, Hamzeh, Nima Mahmoudi, Cornel Barna, and Marin Litoiu. 'Performance Modeling of Microservice Platforms'. IEEE Transactions on Cloud Computing 10, no. 4 (2022): 2848–62. https://doi.org/10.1109/TCC.2020.3029092.

[19] Vural, Hulya, Murat Koyuncu, and Sinem Guney. 'A Systematic Literature Review on Microservices', 203–17, 07 2017. https://doi.org/10.1007/978-3-319-62407-5 _14.

[20] Hong, Xian, Hyun Yang, and Young Kim. 'Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application', 257–59, 10 2018. https://doi.org/10.1109/ICTC.2018.8539409.

[21] Hasselbring, Wilhelm, and Guido Steinacker. 'Microservice Architectures for Scalability, Agility and Reliability in E-Commerce', 243–46, 04 2017. https://doi.org/10.1109/ICSAW.2017.11.

[22] Volynsky, Evgeny, Merlin Mehmed, and Stephan Krusche. 'Architect: A Framework for the Migration to Microservices'. In 2022 International Conference on Computing, Electronics & Communications Engineering (iCCECE), 71–76, 2022. https://doi.org/10.1109/iCCECE55162.2022.9875096.

[23] Blinowski, Grzegorz, Anna Ojdowska, and Adam Przybyłek. "Monolithic Vs. Microservice Architecture: A Performance and Scalability Evaluation". IEEE Access 10 (2022): 20357-74. https://doi.org/10.1109/access.2022.3152803.

[24] Li, Yishan, and Sathiamoorthy Manoharan. 'A Performance Comparison of SQL and NoSQL Databases'. In 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), 15–19, 2013. https://doi.org/10.1109/PACRIM.2013.6625441.

[25] Ma, Shang-Pin, Tsung-Wen Lu, and Chung-Chieh Li. 'Migrating Monoliths to Microservices Based on the Analysis of Database Access Requests'. In 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE), 11–18, 2022. https://doi.org/10.1109/SOSE55356.2022.00008.

[26] Gokan Khan, Michel, Javid Taheri, Auday Al-Dulaimy, and Andreas Kassler. 'PerfSim: A Performance Simulator for Cloud Native Computing'. IEEE Transactions on Cloud Computing, 2021. https://doi.org/10.1109/TCC.2021.3135757.

[27] Khasawneh, Tariq, Mahmoud Alsahlee, and Ali Safieh. 'SQL, NewSQL, and NOSQL Databases: A Comparative Survey', 013–021, 04 2020. https://doi.org/10.1109/ICICS49469.2020.239513.

[28] Hannousse, Abdelhakim, and Salima Yahiouche. 'Securing Microservices and Microservice Architectures: A Systematic Mapping Study'. Computer Science Review 41 (06 2021): 100415. https://doi.org/10.1016/j.cosrev.2021.100415.

[29] Schmidt, Roger Anderson, and Marcello Thiry. 'Microservices Identification Strategies: A Review Focused on Model-Driven Engineering and Domain Driven Design Approaches'. In 2020 15th Iberian Conference on Information Systems and Technologies (CISTI), 1–6, 2020. https://doi.org/10.23919/CISTI49556.2020.9141150.

[30] Al-Debagy, O., Martinek, P. "A New Decomposition Method for Designing Microservices", Periodica Polytechnica Electrical Engineering and Computer Science, 63(4), pp. 274–281, 2019. https://doi.org/10.3311/PPee.13925

[31] Amiri, Mohammad Javad. 'Object-Aware Identification of Microservices'. 2018 IEEE International Conference on Services Computing (SCC), 2018, 253–56.

[32] "An Overview of Microservice Architecture." n.d. TO the NEW BLOG. https://www.tothenew.com/blog/an-overview-of-microservice-architecture-part-i/.

[33] "Microservices and Databases: The Main Challenges." 2019. DBmaestro. July 10, 2019. https://www.dbmaestro.com/blog/database-automation/microservices-databases-challenges.