




# Assessing the Impact of Microservices Architecture on Software Maintainability and Scalability

Vamsi Krishna Thatikonda   
Snoqualmie, Washington, USA

## Suggested Citation

Thatikonda, V.K. (2023). Assessing the Impact of Microservices Architecture on Software Maintainability and Scalability. *European Journal of Theoretical and Applied Sciences*, 1(4), 782-787.  
DOI: [10.59324/ejtas.2023.1\(4\).71](https://doi.org/10.59324/ejtas.2023.1(4).71)

## Abstract:

This evaluation delves into the influence of Microservices Architecture (MSA) on crucial factors like software maintainability and scalability, which are vital in today's software development. Exploring MSA's features and benefits reveals how it streamlines bug tracking, improves system comprehension, increases testability, enables independent scaling of services, and optimizes resource utilization. The assessment also identifies potential obstacles, including distribution challenges, communication overhead, network delays, and data consistency. Renowned organizations like Amazon

and Netflix have provided real-life scenarios with significant evidence of MSA's practicality and limitations. Though MSA's importance in building durable, scalable systems is underlined, the evaluation also stresses the need for solid design principles, practical management approaches, and constant refinement of procedures.

**Keywords:** *Microservices Architecture, Software Maintainability, Software Scalability, Monolithic Architecture, Independent Scalability, Resource Utilization, Network Latency, Data Consistency.*

## Introduction

Microservices Architecture (MSA) brings a unique approach to creating software systems. This approach organizes various systems into more minor, independent services that can perform specific tasks and communicate with each other efficiently to achieve business goals (Di Francesco, Lago, & Malavolta, 2019). This architectural style introduces a novel method of handling and deploying software applications, focusing on business capabilities and taking advantage of fully automated deployment machinery for independent deployment.

Efficient software development heavily relies on two critical components: maintainability and scalability. Maintainability is how a software system or component can be altered to fix errors, enhance performance, or adapt to

environmental changes (Dayanandan & Vivekanandan, 2016). Conversely, scalability refers to a system's ability to manage an increasing workload by integrating additional resources (Singh & Reddy, 2014). Both these characteristics significantly affect the software's cost-effectiveness, user satisfaction, and overall market competitiveness.

The concept of Microservices Architecture is primarily focused on software maintainability and scalability. It enables the separate components to be developed, deployed, upgraded, and scaled independently. This evaluation seeks to gauge the effect of Microservices Architecture on these essential software characteristics. It will determine whether this architectural style enhances or diminishes the important software properties.



Exploring the details of Microservices Architecture, its impact on maintainability and scalability, and the application of real-world case studies will all contribute to the discussion.

## Background of Microservices Architecture

Microservices Architecture (MSA) is not an entirely new idea but rather an integration of beneficial practices from various tried-and-tested methodologies, including Domain-Driven Design, Continuous Integration, and Service-Oriented Architecture, among others (Newman, 2021). The term "microservices" was first used around 2014 to categorize a style of software architecture that had been gradually forming in response to the evolving conditions of software development and deployment during the advent of cloud-based systems, distributed systems, and continuous delivery (Butzin, Golatowski & Timmermann, 2016).

Previously, software applications were commonly created with a Monolithic Architecture approach, which involved merging all application processes into a single self-

contained unit. This monolith integrated the user interface, business logic, and data layer within one application (Alpers et al., 2015). While this architecture offered simplicity in deployment and operation, it also presented certain limitations.

Monolithic architectures have significant issues, such as a lack of flexibility, instability, and inefficiency. The more giant the monolith, the more complicated it gets to comprehend and modify. A significant challenge with scalability is that it requires altering the entire application rather than just specific components.

Transitioning to Microservices Architecture arose from challenges experienced with monolithic systems. The goal is to dissect complex applications into more minor, more manageable services that are loosely connected. These services can be developed, deployed, and scaled independently (Megargel, Shankararaman, & Walker, 2020). This transition sought to enhance software systems' maintenance, scalability, and productivity. It also promotes swift and secure execution of alterations, guaranteeing that the system adapts to escalating workloads.

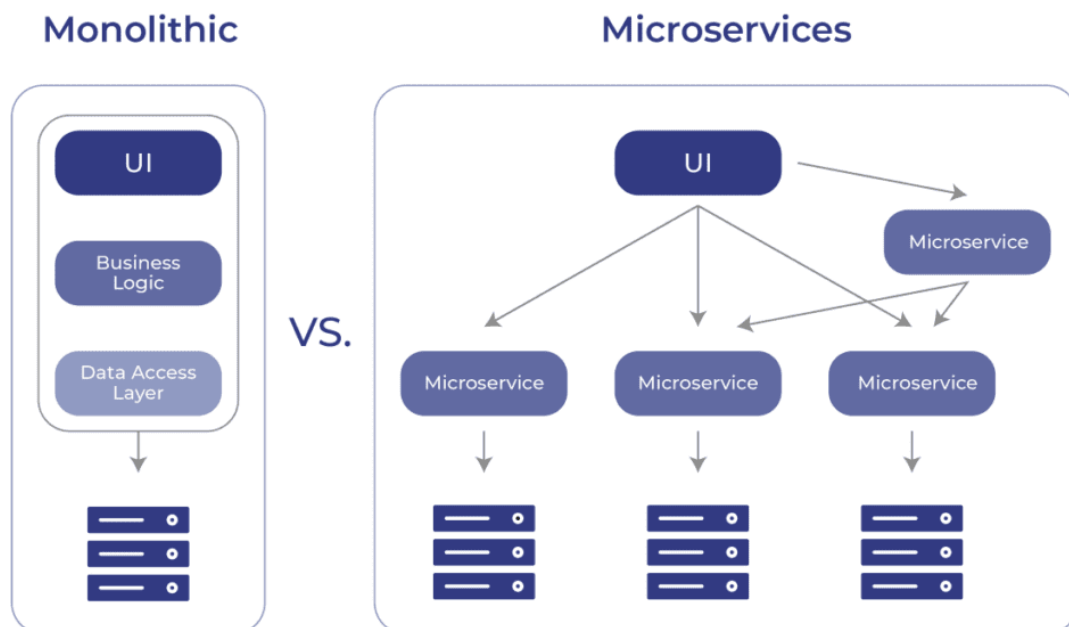


Figure 1. Monolithic Vs Microservices

Source: Kanjilal, J. (2021)

## Understanding Microservices Architecture

Microservices Architecture (MSA) offers a unique approach to software design. It involves creating an application that comprises multiple loosely connected services that can be deployed independently. These services are usually organized based on specific business capabilities. Each service can operate at its scale, use different programming languages, and employ various data storage technologies.

### Componentization through Services

The central principle of MSA involves breaking down complex systems into controllable, isolated services. Each service independently accomplishes a specific business function. Independent deployments of these services allow updates to a service without needing to redeploy the entire application, thus enhancing maintainability (Newman, 2021).

### Organizing around Business Capabilities

Every microservice is designed to perform a specific business function, ensuring that it comprises all essential elements such as user interface, databases, and backend services. This organized approach promotes flexibility, making adapting to evolving business needs easy.

### Products rather than Projects

The Microservices Architecture promotes a product-centric mindset where services are treated as products owned by a team from start to finish (Di Francesco, Lago, & Malavolta, 2019). This approach promotes accountability, user-centeredness, and long-term planning, resulting in higher-quality and easier-to-maintain software.

### Smart Endpoints and Dumb Pipes

Unlike SOA models, MSA employs microservices that communicate with simple, stateless protocols like HTTP REST or messaging. The intelligence is located in the endpoints rather than the communication middleware, resulting in a system design that is easier to understand, handle, and expand.

The main advantages of MSA stem from its flexibility and scalability. This architecture enables development teams to select the most suitable technology stack for each service, boosting productivity and ease of maintenance. Additionally, since each service operates independently, MSA allows for efficient scaling to meet demand, which can improve system resilience and minimize resource waste.

## Impact on Software Maintainability

Software Maintainability refers to the ease with which changes can be made to a software product, whether to correct defects, improve performance, or adapt to a new environment (Dayanandan & Vivekanandan, 2016). This is a crucial element of software quality, directly affecting the cost, effort, and time required for modifications or enhancements.

The Microservices Architecture, abbreviated as MSA, has a profound influence on the maintainability of software. Firstly, MSA promotes more effortless bug detection and rectification. The software, a blend of small, independent services, can isolate a bug to one specific service when it emerges. This isolation simplifies the task of identifying, debugging, and resolving the issue without causing any disturbance to the entire system (Alshuqayran, Ali, & Evans, 2016).

Secondly, the comprehension of the system becomes simpler with MSA due to the separation into modules. Each microservice has its independent business capability and can be comprehended separately (Da Silva, Justino, & de Adachi, 2011). This approach results in a less complex codebase, reduces the cognitive load on developers, and ultimately improves maintainability.

Thirdly, MSA improves software testability. Testing each microservice separately allows for more focused and efficient testing. Additionally, automated testing becomes easier to perform, leading to better maintainability.

## CHARACTERISTICS OF MICROSERVICE ARCHITECTURE

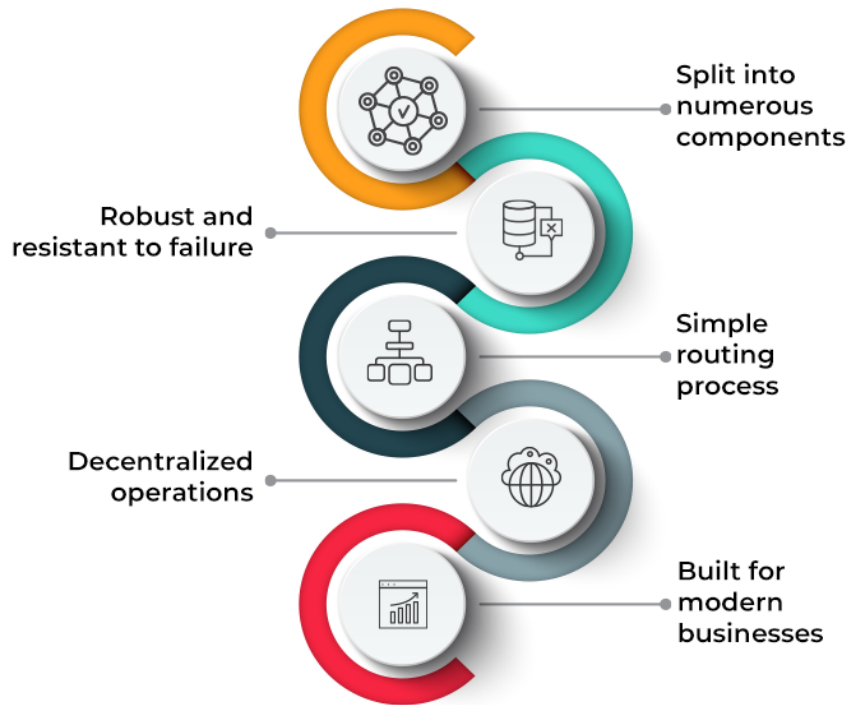


Figure 2. Characteristics of Microservice Architecture

Source: Ashtari, H. (2022)

Although MSA offers many benefits, it also presents several challenges regarding maintainability. The complexity of the system can increase due to the distribution of services. Maintaining dependencies, ensuring consistent data, and isolating faults among numerous services can be challenging. Additionally, effective communication between services is crucial. This requires well-structured APIs and messaging protocols to avoid negatively impacting system performance and complicating maintenance tasks. Proper management is essential to avoid communication overhead.

### Impact on Software Scalability

The scalability of software refers to its ability to handle increasing workloads by allocating resources proportionately. An ideal scalable system should be able to handle more requests

without sacrificing performance (Singh & Reddy, 2014). Scalability is crucial in today's constantly changing business environment, where applications must manage fluctuating traffic and large data volumes.

MSA has a significant influence on software scalability. Firstly, MSA allows each service to scale independently. This contrasts with monolithic architectures, where the entire system must scale. Each microservice in an MSA setup can be scaled based on its specific demand (Li et al., 2021). This level of precision affords a higher degree of control, ensuring more efficient resource allocation.

Secondly, MSA can help effectively allocate resources by minimizing underutilized resources and avoiding over-provisioning. By deploying each microservice in an environment that suits its unique needs, such as computational power,

memory, storage, or hardware type, performance can be improved while simultaneously reducing the cost of running the system.

Despite these benefits, MSA does present scalability-related challenges. The first is network latency and communication between services. As the number of microservices increases, inter-service communication also increases. This can result in network congestion and latency, which may negatively impact the system's overall performance.

Another challenge involves data consistency. In MSA, each service usually has its database, leading to consistency issues when services need to share data. Synchronizing data across services, particularly during scaling, can be complex and resource-intensive (Taibi, Lenarduzzi, & Pahl, 2017).

## Case Studies

Microservices Architecture has proven successful for many businesses by improving software maintainability and scalability. Here are two examples of its successful implementation:

According to a study conducted by Ren and their team in 2018, Amazon transitioned from a monolithic setup to the microservices approach to achieve superior scalability (Ren et al., 2018). This shift allowed Amazon to independently scale specific services during high-traffic periods like Black Friday or Christmas. As a result, Amazon was able to make significant cost savings and improve the performance of its system.

Another prime example is Netflix has successfully adopted the microservices model, as highlighted in a 2018 study by Ren and his team. With a subscriber base of over 200 million worldwide and a growing need for high-quality streaming services, Netflix turned to MSA to improve scalability and ensure excellent customer service. By transitioning to microservices, Netflix can now make thousands of modifications to its global infrastructure daily without causing any disruptions. This switch has

led to better maintainability and faster innovation.

## Conclusion

Microservices Architecture (MSA) dramatically impacts the maintainability and scalability of software positively. It provides several benefits, including easier tracking of bugs, a better understanding of the system due to the separation of modules, the ability to scale services independently, and more efficient use of resources. However, it also presents some challenges, such as increased complexity due to the distribution of services, inter-service communication overheads, network latency, and potential issues with data consistency.

As businesses strive for agile, resilient, and scalable systems, the significance of MSA is expected to grow. To fully leverage the potential of MSA, it is essential to adopt robust design principles and efficient management strategies and continuously refine practices to manage inherent complexities. Future studies may explore ways to simplify MSA implementation and its link with emerging trends such as serverless and edge computing.

## References

- Alpers, S., Becker, C., Oberweis, A. & Schuster, T. (2015). *Microservice based tool support for business process modelling*. 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop. <https://doi.org/10.1109/edocw.2015.32>
- Alshuqayran, N., Ali, N., & Evans, R. (2016). *A systematic mapping study in microservice architecture*. 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA). <https://doi.org/10.1109/soca.2016.15>
- Ashtari, H. (2022). Microservices definition, examples, architecture, and best practices. Retrieved from <https://www.spiceworks.com/tech/devops/articles/what-are-microservices/>

- Butzin, B., Golatowski, F. & Timmermann, D. (2016). *Microservices approach for the internet of things*. 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA). <https://doi.org/10.1109/etfa.2016.7733707>
- Da Silva, C.E., Justino, Y. & de Adachi, E. (2011). Spread: Service-oriented process for reengineering and DevOps. *Service Oriented Computing and Applications*, 16(1), 1–16. <https://doi.org/10.1007/s11761-021-00329-x>
- Dayanandan, U. & Vivekanandan, K. (2016). An empirical evaluation model for software architecture maintainability for Object Oriented Design. *Proceedings of the International Conference on Informatics and Analytics*, 1-4. <https://doi.org/10.1145/2980258.2980459>
- Di Francesco, P., Lago, P. & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*, 150, 77–97. <https://doi.org/10.1016/j.jss.2019.01.001>
- Kanjilal, J. (2021). Microservice scalability challenges and solutions. Retrieved from <https://www.developer.com/design/microservices-scalability-challenges/>
- Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., & Shan, Z. (2021). Understanding and addressing quality attributes of microservices architecture: A systematic literature review. *Information and Software Technology*, 131, 106449. <https://doi.org/10.1016/j.infsof.2020.106449>
- Megargel, A., Shankararaman, V., & Walker, D.K. (2020). Migrating from monoliths to cloud-based microservices: A banking industry example. *Computer Communications and Networks*, 85–108. [https://doi.org/10.1007/978-3-030-33624-0\\_4](https://doi.org/10.1007/978-3-030-33624-0_4)
- Newman, S. (2021). *Building microservices: Designing fine-grained systems*. 2nd ed. O'Reilly Media, Inc..
- Ren, Z., Wang, W., Wu, G., Gao, C., Chen, W., & Wei, J. (2018). *Migrating web applications from monolithic structure to microservices architecture*. Proceedings of the Tenth Asia-Pacific Symposium on Internetware. <https://doi.org/10.1145/3275219.3275230>
- Singh, D. & Reddy, C.K. (2014). A survey on platforms for Big Data Analytics. *Journal of Big Data*, 2(1). <https://doi.org/10.1186/s40537-014-0008-6>
- Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, 4(5), 22–32. <https://doi.org/10.1109/mcc.2017.4250931>