www.ijecs.in
International Journal Of Engineering And Computer Science
Volume 14 Issue 01 January 2025, Page No. 26744-26750
ISSN: 2319-7242 DOI: 10.18535/ijecs/v14i01.4958

Microservices Architecture: Accelerating Feature Development and Scalability Through Monolith Decomposition

Kuciuk Artiom [1]

[1] Tech Lead at Greentube Vienna, Austria

Abstract

The article analyzes approaches to the transformation of software solutions based on microservice architecture. The issues of structuring systems are considered, such as the division of functionality, the allocation of separate domain areas, the creation of interfaces, and the management of components and their interaction. The practical aspects of the use of containerization and orchestration technologies, as well as mechanisms for ensuring data reliability and integrity in distributed environments, are highlighted. Examples are given demonstrating the reduction of component interdependence, simplification of update processes, and adaptation of systems to changing conditions. When writing the work, an analytical methodology based on a systematic approach to the collection, study, and synthesis of information was used. Scientific articles published by the author in the public domain, as well as materials that are on the Internet, were used as sources, which made it possible to comprehensively consider the topic. The author focuses on the need for a preliminary analysis of the current system, the development of a clear decomposition strategy, and the use of modern tools for a successful transition to a modular architecture. Recommendations concerning the implementation of container solutions, orchestration systems, continuous integration, and delivery processes are presented. The content is intended for specialists involved in development, architectural design, and infrastructure tasks. These findings demonstrate the potential of the microservice approach as a tool for creating scalable, sustainable applications.

Keywords: *microservices, architecture, monolith decomposition, scalability, containerization, API, automation.*

1. Introduction

The architecture of software systems is evolving in response to increasing demands for flexibility, scalability, and optimization of development processes. Monolithic applications, characterized by a unified structure, encounter significant challenges as load increases, business logic becomes more complex, and market conditions shift. These limitations are particularly pronounced in systems with high operational intensity, where functional updates and adaptation are essential. The microservices approach involves dividing a system into independent modules with clearly defined responsibilities. Each module operates autonomously, simplifying development, testing, and deployment of updates. This approach facilitates system adaptation to external changes, enhances fault tolerance, enables automation of processes such as testing and orchestration, and supports continuous delivery. Transitioning from a monolithic structure to a microservices architecture requires a thorough risk analysis, the development of a strategy for decomposing the system into separate components, and the selection of tools tailored to specific tasks. The

study of such transformations is increasingly relevant in the context of data growth, process complexity, and the need for rapid adaptation of software solutions to new operational conditions. This article explores methods of application decomposition, evaluates their impact on system performance and adaptability, and provides recommendations for transitioning to a microservices architecture.

2. Materials and Methods

Literature on the migration of monolithic applications to microservices architecture addresses a wide range of issues, including the analysis of architectural solutions, methodologies for software structuring, and assessments of their impact on system performance. Scholarly works explore both theoretical and practical aspects of architectural transformation.

The comparison of various architectures has been extensively studied in works by Blinowski G., Ojdowska A., Przybyłek A. [1], Jaskot K., Przyłucki S. [11], where operational characteristics of monolithic and microservices systems under real-world scenarios are analyzed, and differences in their adaptation to workloads are highlighted. Tapia F. et al. [2] examine factors influencing the successful implementation of microservices, including challenges during the transition phase and the advantages of changing architectural solutions.

Decomposition methodologies are discussed in articles by Wei Y. et al. [3], Cao L., Zhang C. [5], and Rand Ochimah S. et al. [12]. Wei Y. et al. propose an approach based on identifying functional characteristics of business logic. Cao L., Zhang C. applies interaction analysis of software elements for structuring microservices. Rochimah S. et al. utilize class clustering methods derived from source code analysis. Hao J., Zhao J., and Li Y. [7] investigate the transformation of relational databases to ensure data preservation. Camilli M. et al. [8] developed a multi-level evaluation system aimed at improving architectural flexibility. Nitin V. et al. [9] employ artificial intelligence to enhance the accuracy of component relationship analysis. Contemporary application modernization methodologies are presented in studies by Joselyne M. I., Bajpai G., Nzanywayingoma F. [4], Mishra R. et al. [10], Rahmatulloh A. et al. [6]. Joselyne M. I., Bajpai G., and Nzanywayingoma F. describe architectural transformation using Domain-Driven Design approaches. Mishra R. et al. investigate available tools that facilitate successful system transformation. Rahmatulloh A. et al. study event-driven methods and their impact on operational performance.

For the practical section describing the impact of microservices architecture on operations, sources [13,14] hosted on highscalability.com, korusconsulting.ru were utilized. Although microservices architecture demonstrates efficiency, questions remain regarding the selection of structuring methods in highly interconnected component environments. Aspects related to the long-term operation of systems, automation of application transformation, and compatibility remain underexplored. These areas require further study and the development of new solutions. The study employed an analytical methodology based on a systematic approach to collecting, analyzing, and synthesizing information.

3. Results and Discussion

The microservices approach involves structuring a system into autonomous modules, each performing specific functions. Each component operates independently, simplifying management, deployment, and adaptation to changes. Data exchange between modules utilizes standards such as REST, gRPC, and GraphQL [1]. Below, Figure 1 presents the elements of a microservices architecture.

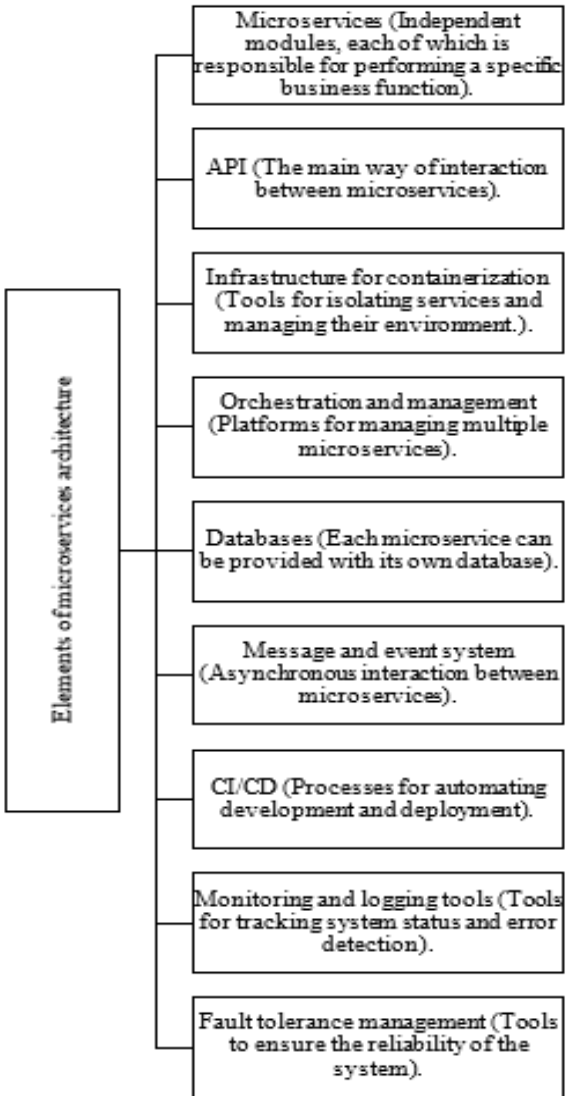

Fig.1. Elements of microservices architecture (compiled by the author).

The isolation of system components simplifies modifications and enhances fault tolerance. To achieve this, it is crucial to thoroughly define module boundaries using domain-driven design methodologies [1,8]. Below, Figure 2 illustrates the principles of microservices architecture.

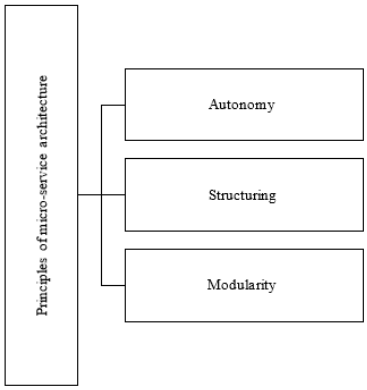

Fig.2. Principles of microservices architecture (compiled by the author).

Traditional architectures consolidate functional parts into a unified structure. Any modification to one element necessitates changes across all others, complicating updates and requiring a comprehensive system test, even for minor adjustments. This approach becomes challenging as the number of users increases. Decomposition involves splitting an application into independent services that interact through interfaces. This format eliminates component interdependencies, accelerating the implementation of changes and reducing the likelihood of errors.

The microservices approach makes the development process more structured. Teams can work on different tasks simultaneously without hindering one another, accelerating the deployment of new solutions and eliminating the need for system-wide testing after changes to a single part. System segmentation facilitates adaptation to changing conditions. Increased load demands resource redistribution only for segments experiencing strain. For instance, during peak sales, additional resources can be allocated specifically to order processing, leaving other modules unaffected. This optimizes resource allocation and minimizes infrastructure costs.

System stability is enhanced by isolating its elements. In the event of a component failure, the remaining modules continue to function. For example, if the notification system fails, it does not disrupt the performance of other tasks [7-9]. For successful decomposition, thorough preparation is required. An analysis of the existing system identifies areas that need segmentation. Priority is given to segments subject to frequent changes or high loads. Following this, the boundaries of each service are defined, and interaction rules are developed.

The transformation process begins with standalone modules. Gradual segmentation reduces risks and simplifies the transition. Service management is carried out using modern containerization technologies and orchestration systems, automating deployment and ensuring control. The transition from monolithic to modular

architecture starts with analyzing existing internal connections, identifying domain areas, and dividing them into isolated segments. After segmentation, API interfaces are designed to account for scalability.

The transformation of systems such as ERP demonstrates the potential of the microservices approach. Dividing functions like accounting, logistics management, and order processing into independent modules improves manageability. Tools like Kubernetes and Docker Compose are used to simplify deployment. Developing individual modules enables specialists to focus on specific tasks. Automation of testing and implementation of changes via CI/CD eliminates the risk of system disruption. Tools for API testing, such as Postman, and interaction simulation through WireMock streamline integration.

This architecture facilitates component adaptation to load. Resource allocation for operation processing can occur without redesigning the entire system. For instance, during periods of high transaction activity, resources are allocated to the component responsible for their processing. Scaling is achieved either by increasing the number of service instances or by enhancing hardware capabilities. Challenges arise during the transition to this architecture. Increasing the number of components complicates the management of interconnections. Interface changes must account for compatibility with other elements. Special mechanisms are developed to synchronize data in distributed systems. Tools like Prometheus and Jaeger are used to monitor system health.

Risk minimization is achieved through orchestration tools, distributed tracing, and automated monitoring. These measures enhance system resilience and simplify maintenance [6,10]. As examples, consider the experiences of PayPal, Amazon, and Netflix. The transition to a microservice architecture marked a significant milestone in the development of PayPal's systems, improving performance and increasing resilience under high loads. The integration of its proprietary NoSQL-based JunoDB database enables the processing of over 350 billion queries daily. The system maintains high availability with less than three seconds of downtime per month [12].

Amazon redefined its platform design by transitioning from a two-tier monolithic solution to a fully distributed, decentralized service platform that supports a wide range of applications. The implementation of a microservice model enabled developers to concentrate on building individual components, accelerating the release of updates. This new architecture significantly improved the platform's adaptability to changing conditions [13]. Netflix employs approximately 700 microservices to manage various elements that comprise the entire service. One microservice processes payments, another stores information about all the shows watched, and a third determines suitable content recommendations [14].

The following section examines an author-developed practical example. In this case, the specialist possesses experience in developing software solutions for online platforms supporting payment processing. Within the professional scope, methods were applied to flexibly adapt architectural elements to market changes. A key task involved transitioning from a monolithic structure to a microservices format, eliminating numerous limitations of the traditional model. The analysis of the existing system revealed significant drawbacks. The outdated approach complicated the process of function modification, increased the likelihood of errors when expanding functionality, and restricted the ability to distribute workloads. Integration challenges with new services diminished the platform’s adaptability to evolving requirements.

The proposed solution was based on dividing the structure into separate modules, each assigned a clearly defined set of tasks. The new model ensured the independence of functional components, simplified testing, and enhanced flexibility for implementing changes. Functional

localization eliminated the impact of faults on the entire system, thereby improving its resilience. The implementation was carried out in stages. Initially, functional elements were analyzed, and their boundaries and tasks were defined. Subsequently, a sequential transformation was performed, ensuring smooth updates. This phased approach minimized risks associated with system stability disruptions.

To enhance performance, automation tools were introduced, and technologies enabling the isolation of functional modules were employed. Automated processes for testing and updates reduced modification time and increased verification accuracy. The transition to a modular structure improved platform performance, accelerated the deployment of new services, and stabilized transaction processing. The new model demonstrated its effectiveness in a competitive market environment. This project highlighted the specialist's ability to address complex challenges, create robust architectural solutions, and align them with business objectives. Below, Table 1 outlines the advantages and disadvantages of microservices architecture.

Table 1. Advantages and Disadvantages of Microservices Architecture [2]

Category	Advantages	Disadvantages
Development Speed	- Independence of teams: each team works on its service, minimizing conflicts.	- Coordination between teams becomes more complex due to segmentation.
	- Less time required to implement changes and release new versions.	- Complexity in managing API versions and contracts between services.
Scalability	- Easier to scale individual parts of the system instead of the entire monolith.	- Requires infrastructure to manage scalability.
	- Ensures availability through isolated services.	- Increased overhead for monitoring and network management.
Monolith Decomposition	- Simplifies code maintenance: each service addresses its task independently.	- Complexity in refactoring existing monoliths to transition to microservices.
	- Improves testability with smaller codebases for each service.	- Debugging becomes more challenging due to issues arising at the service interaction level.
Technology Flexibility	- Enables the use of different technologies and programming languages for different services.	- Increases DevOps complexity and requires knowledge of diverse technology stacks.
Deployment	- Facilitates partial deployment (updating only the necessary service).	- Risks of incompatibility between services after deployment arise.
Data Management	- Allows isolated data management (each service having its database).	- Complexity in ensuring data consistency across services.
Overall Complexity	- Simplifies long-term system management by isolating each service's context.	- The overall architecture becomes more complex, requiring orchestration tools (e.g., Kubernetes).

The application of machine learning in data analysis not only automates processes but also enhances accuracy.

4. Conclusion

An analysis of the literature demonstrates that employing a microservices architecture enables the creation of adaptive systems with advantages over the monolithic approach. Dividing the system

into autonomous modules accelerates the implementation of new features, minimizes failure risks, and improves adaptability. Decomposition methods, including dependency mapping, domain area identification, and interface design, achieve component isolation. Containerization technologies simplify deployment processes and the management of distributed systems. However, transitioning to a microservices architecture involves challenges related to ensuring data consistency, organizing monitoring, and managing application programming interfaces.

The findings suggest that architectural design accelerates software development and provides a foundation for its sustainable functionality.

References

1. Blinowski G., Ojdowska A., Przybyłek A. Monolithic vs. microservice architecture: A performance and scalability evaluation //IEEE Access. – 2022. – Vol. 10. – pp. 20357-20374.
2. Tapia F. et al. From monolithic systems to microservices: A comparative study of performance //Applied sciences. – 2020. – vol. 10. – no. 17. – p. 5797.
3. Wei Y. et al. A feature table approach to decomposing monolithic applications into microservices //Proceedings of the 12th Asia-Pacific Symposium on Internetware. – 2020. – pp. 21-30.
4. Joselyne M. I., Bajpai G., Nzanywayingoma F. A systematic framework of application modernization to microservice-based architecture //2021 International Conference on Engineering and Emerging Technologies (ICEET). – IEEE, 2021. – pp. 1-6.
5. Cao L., Zhang C. Implementation of domain-oriented microservices decomposition based on node-attributed network //Proceedings of the 2022 11th International Conference on Software and Computer Applications. – 2022. – pp. 136-142.
6. Rahmatulloh A. et al. Event-Driven Architecture to Improve Performance and Scalability in Microservices-Based Systems //2022 International Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS). – IEEE, 2022. – pp. 01-06.
7. Hao J., Zhao J., Li Y. Research on Decomposition Method of Relational Database Oriented to Microservice Refactoring //2023 24th Asia-Pacific Network Operations and Management Symposium (APNOMS). – IEEE, 2023. – pp. 282-285.
8. Camilli M. et al. Actor-driven decomposition of microservices through multi-level scalability assessment //ACM Transactions on Software Engineering and Methodology. – 2023. – vol. 32. – No. 5. – pp. 1-46.
9. Nitin V. et al. Cargo: Ai-guided dependency analysis for migrating monolithic applications to microservices architecture //Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. - 2022. – pp. 1-12.
10. Mishra R. et al. Transition from Monolithic to Microservices Architecture: Need and proposed pipeline //2022 International Conference on Futuristic Technologies (INCOFT). – IEEE, 2022. – pp. 1-6.
11. Jaskot K., Przyłucki S. Analiza wybranych cech aplikacji opartych na architekturze monolitycznej i mikrousługowej //Journal of Computer Sciences Institute. – 2022. – Vol. 25.
12. Rochimah S. et al. Decomposing monolithic to microservices: Keyword extraction and BFS combination method to cluster monolithic's classes //Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi). – 2023. – Vol. 7. – No. 2. – pp. 263-270.
13. Amazon Architecture. [Electronic resource] Access mode:

https://highscalability.com/amazon-architecture/ (date of application: 5.12.2024).
14. Microservice architecture in simple terms: pros and cons of the approach. [Electronic resource] Access mode: https://korusconsulting.ru/infohub/mikroservisnaya-arkhitektura-prostymi-slovami-plyusy-i-minusy/ (date of application: 5.12.2024).