http://www.moderntechno.de/index.php/meit/article/view/meit29-01-038
DOI: 10.30890/2567-5273.2023-29-01-038

MODULAR MONOLITH AS A MICROSERVICES PRECURSOR

Shablii Taras
2nd-year master's degree student, Software Engineering
Tytenko Sergiy
Ph.D., Associate Professor
ORCID: 0000-0002-7548-9053
American University Kyiv,
Ukraine, Kyiv, Poshtova Pl. 3, 02000

Abstract. *The world of software architecture is in a constant state of evolution, and the development of distributed architectures continues to shape the industry's landscape. Monoliths, characterized by their unified codebase and singular deployment, have long been the traditional choice for software development. On the other hand, microservices, with their small, independently deployable services, have gained prominence due to their scalability and resilience. This article explores the dynamics of monoliths and microservices, the challenges associated with this migration, and proposes the idea of modular monoliths as a precursor to microservices to reduce migration efforts. It also highlights the lack of research in the domain of structuring monoliths in advance for future efficient migration to microservices.*

Key words: *software architecture, microservices, modular monolith*

Introduction

In the world of software architecture, two dominant styles have emerged, each with its set of advantages and disadvantages: monoliths and microservices. Monolith is a "*system in which all of the code is deployed as a single process*" [1]. In this architectural style all code resides in a single repository and forms a single deployment artifact, all data is most often located within a single database (Figure 1).

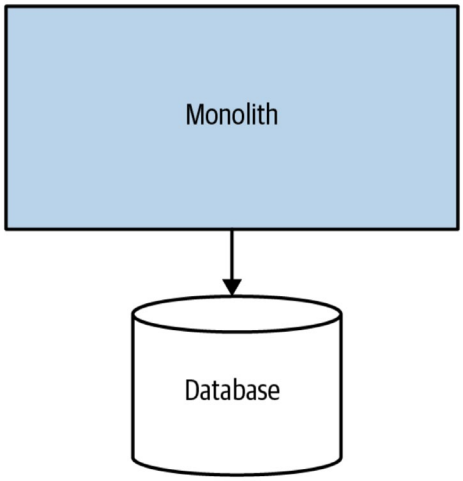

Figure 1 - Single process monolith

A source: [1]

In contrast, a microservice (Figure 2) is a "*single-purpose, separately deployed unit of software that does one thing really, really well*" [2].

While both approaches have their merits, the shift towards microservices has been on the rise in the recent decade (Figure 3).

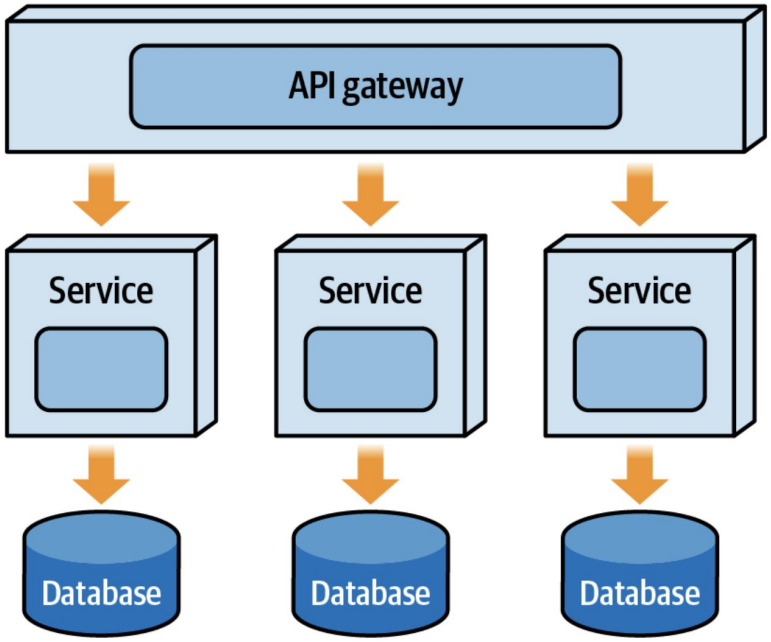

Figure 2 - Microservices topology

A source: [2]

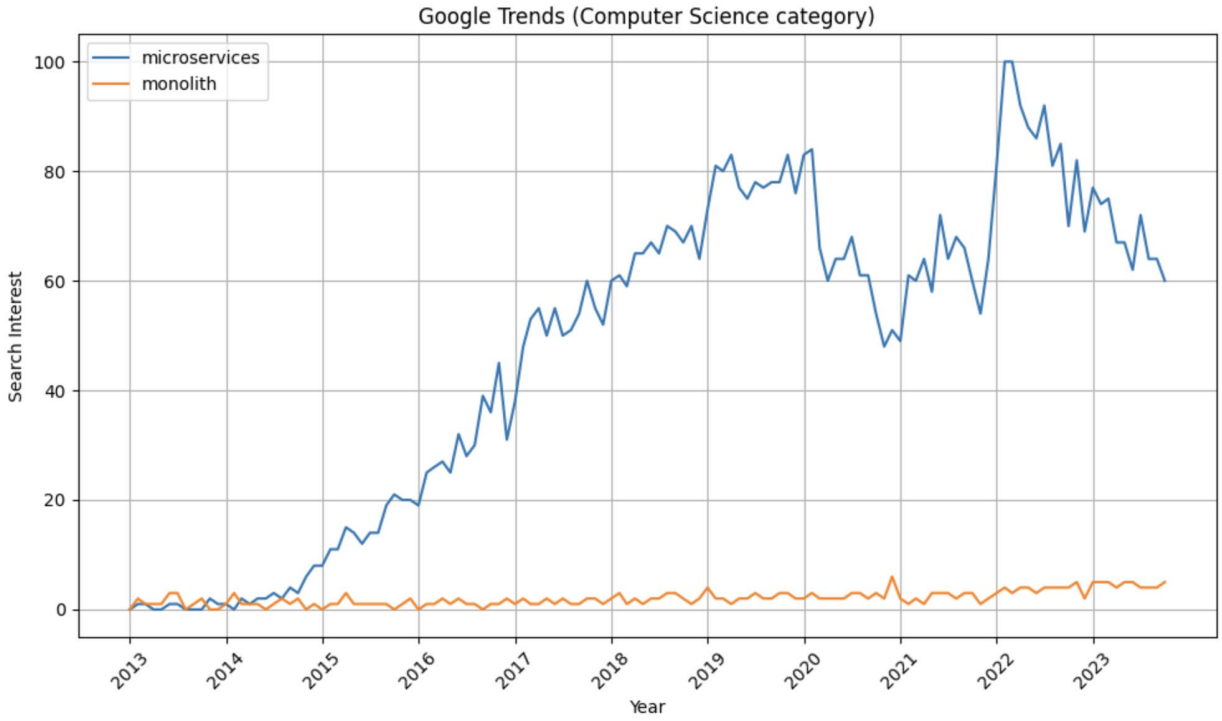

Figure 3 - Raising interest in microservices architecture

A source: [3]

Monoliths vs. Microservices

Monoliths have been the traditional choice for software development for decades. In a monolithic architecture, the entire application, from user interface to database access, is bundled together in a single codebase. This tightly coupled nature simplifies development but introduces challenges in terms of scalability and maintainability [4].

Monolithic architecture provides the following benefits [5]: simplicity of

development and deployment since all code is located within a single codebase and is deployed as a single artifact; optimal performance, since components communicate via direct function calls without any network overhead; faster time to market since developing and deploying a single codebase that does not require sophisticated infrastructure. At the same time, monolithic architecture is characterized by natural degradation of scalability (increasing capacity often involves scaling the entire application, which can be inefficient) and maintainability (monoliths tend to become increasingly complex and harder to maintain) [5].

Microservices, in contrast, advocate for breaking down an application into smaller, autonomous services, each with a specific singular responsibility. These services communicate with each other over the network, allowing for flexibility and independent scalability [5].

The heightened interest in microservices over the past decade can be attributed to several key factors. First, there has been a growing demand for applications with enhanced availability, fault tolerance, and resilience, especially as businesses rely more on digital services [6]. Microservices provide a framework for achieving these goals by allowing individual services to fail gracefully without compromising the entire system's functionality. Second, the rapidly evolving technological landscape has introduced greater complexity, making it challenging to develop and maintain monolithic applications efficiently. Microservices offer a solution by breaking down complex systems into smaller, manageable components that can be developed, deployed, and scaled independently [7]. Third, the surge in data volumes, driven by the advancements of data-driven applications and IoT devices, has necessitated scalable and distributed architectures. Microservices' ability to scale horizontally makes them well-suited for handling data-intensive workloads [6]. Lastly, advancements in IaaS cloud technology have made it easier to adopt microservices, with cloud providers offering robust infrastructure, platforms and services that facilitate the deployment and management of microservices. These factors have collectively fueled the rising interest in microservices as a modern software architecture paradigm [7].

Benefits of microservices include [5]: scalability since each microservice can be scaled independently, enabling cost-effective resource allocation; smaller codebases are more maintainable, reducing the risk of system-wide disruptions; microservices are fault-tolerant and can be easily made highly available, as the failure of one service doesn't necessarily affect the entire application.

At the same time, microservices architecture is complex, as developing and managing microservices can be complex due to the need for inter-service communication and infrastructure management; it brings added overhead in terms of communication, deployment, and monitoring; and introduces additional layers of communication between services which negatively affects overall system performance due to network latencies [4].

The Migration Trend: Monoliths to Microservices

In recent years, many organizations have embarked on the journey of migrating from monoliths to microservices. Several compelling reasons drive this migration [7][8]:

1. Scalability: Monoliths often struggle to scale efficiently, as any increase in load affects the entire application. Microservices allow organizations to scale only the parts of the application that require additional resources. The entire system can scale up or down in response to demand changes.

2. Technology diversity: Microservices support a wider range of technologies for different components of the application. This enables organizations to choose the best tool for each job, enhancing overall performance.

3. Distributed team autonomy: In a monolithic environment, changes to one part of the application can affect other components. Microservices promote team autonomy, as technologically diverse teams can work on separate services with minimal interference.

4. Faster deployment: Microservices enable independent deployment, allowing organizations to release updates and new features more rapidly.

5. Resilience: Microservices inherently offer better fault isolation. If one service fails, it doesn't necessarily lead to the entire application's collapse.

Challenges of Migrating Monoliths to Microservices

While the benefits of microservices are clear, the journey from monoliths to microservices is fraught with challenges [9]. These challenges encompass both domain and codebase refactoring, as well as infrastructure and data-related hurdles [4]. Moreover, the cost of migration, both in terms of time and resources, can be substantial [10].

Domain and Codebase Refactoring

Refactoring a monolith into microservices requires tearing the application into smaller bounded contexts that encompass specific business domains. This process involves identifying boundaries, separating concerns, and establishing clear APIs for inter-service communication.

These refactoring efforts often face the following common challenges:

● *Boundary identification*: Determining the right boundaries for microservices can be complex, and mistakes can lead to overly chatty communication between services or services that are too tightly coupled.

● *Data separation*: Monoliths often share a common database. Refactoring requires separating the often tightly coupled data into distinct databases for each microservice, which can be intricate for some systems [4].

● *API design*: Designing clear APIs for microservices is a critical task, as these APIs serve as a domain separation boundary and enable inter-service communication.

While there is plenty of research on the attempts to automate or semi-automate monolith decomposition by static code analysis [11] or by analyzing data flow [12], the prospect remains a daunting one [13].

Infrastructure and Data Challenges

In addition to domain and codebase refactoring, the migration from monolith to microservices entails addressing infrastructure and data-related challenges [14]:

● *Deployment and orchestration*: Microservices require robust CI/CD practices to manage the deployment of multiple services across a distributed environment.

● *Data consistency*: Maintaining data consistency and ensuring data availability in a microservices environment is a complex task, particularly when multiple services

access the same data or when there is a need to implement more complex data replication or CQRS patterns.

● *Monitoring and observability*: With the increase in the number of services, monitoring and troubleshooting become more challenging. Each service needs to be individually monitored and each service call needs to be traced as it propagates through a distributed system with varying availability.

● *Communication and networking*: Microservices rely heavily on network communication. Designing and managing network traffic and security is vital.

● *Infrastructure cost*: The distributed nature of microservices can lead to increased infrastructure costs, as additional resources are required to maintain the architecture.

The cost of migration should not be underestimated. It encompasses not only the time and effort required for refactoring but also the effect on organization communication patterns and potential business disruptions during the migration process [4].

Modular Monoliths: A Potential Solution

Despite their shortcomings, monoliths remain an attractive architectural choice for many projects due to their speed of development, shorter time to market, infrastructure relative simplicity, and performance benefits. Not every greenfield project should start from microservices. In fact, many falling prey to the trend find themselves not being able to deliver on time. So, the question to be asked is: what architecture is the best fit for a greenfield project that needs fast time to market at first and is very likely to scale exponentially later? Or rather: are there any options other than painful monolith to microservices migration or unnecessary and expensive microservices from the start?

One way to alleviate the challenges associated with migrating from monoliths to microservices is to consider the concept of modular monoliths. A modular monolith is a monolithic application that is structured in a way where each module already represents a separate domain or subdomain. This modular structure mimics the decomposition of a microservices architecture within a monolithic codebase. While modularizing a monolith is a challenging endeavor [15] it will certainly pay off when a need to scale arises and the organization decides on migration to microservices. In this respect, modularizing a monolith from the start can be viewed as earning architectural credit early on.

A modular monolith brings:

● *Fast time to market*: Inherent to monolithic architectures ease of development, simplicity of deployment, and lack of interservice communication overhead mean that greenfield projects can freely experiment and deliver functionality faster than with microservices.

● *Clear domain boundaries*: A well-structured modular monolith already exhibits distinct domain boundaries, making the transition to microservices more straightforward.

● *Loose coupling*: The modular structure reduces interdependencies between modules, making it easier to extract and deploy them as separate services.

● *Infrastructure simplicity*: Modular monolith inherits simplicity of deployment and required infrastructure from the monolith by the virtue of being a single deployable artifact.

● *Incremental migration*: Organizations can gradually migrate modules to microservices, reducing the disruption and risk associated with a complete migration.

● *Performance*: Direct method calls within a singular deployment artifact bring a natural performance optimization as opposed to a distributed system.

However, it's important to note that achieving a well-structured modular monolith is not a trivial task and requires careful planning and design. A clear understanding of the business domains and subdomains, along with effective API design, is crucial for success. Despite structural complexity required from the start, greenfield projects are likely to adopt a modular monolith approach as part of addressing the tech debt [16].

The Research Gap

Despite the increasing popularity of microservices and the challenges associated with migrating from monoliths to microservices, there is a noticeable scarcity of research in the domain of structuring a monolith to facilitate a smooth transition. Among the few research attempts in this area, a bachelor's thesis by Tsechelidis should be noted [17]. The paper proposes to standardize monolith module structure by leveraging hexagonal architecture as an easy way to control granularity of services and promote microservice-like domain-centric design of each module [17]. Author emphasizes on the benefits of infrastructure simplicity but does not extend an argument for further splitting of such modular monoliths into microservices.

Apart from the lack of scientific research in the area there are conflicting opinions from the industry voices. Fowler is convinced that "*you shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile*" [18]. Newman notes that it is more practical to start with a monolithic system, warns against the pitfall of introducing unnecessary complexity overhead, but does not rule out using microservices for greenfield projects altogether [19]. Tilkov, on the other hand, argues that carving the new system into pieces should be done as early as possible and going with microservices first is the right way to achieve that [20].

While industry best practices and case studies provide valuable insights, there is a need for systematic and scientifically grounded approaches. Research question that remains unexplored is: what are the best practices for designing modular monoliths that are more amenable to migration to microservices? What design patterns and architectural principles should be applied?

Summary and Conclusions

The architectural choices in software development are pivotal, and the transition from monoliths to microservices is a significant endeavor for many organizations. While the benefits of microservices are clear, the migration process is challenging, both in terms of domain and codebase refactoring, as well as infrastructure and data considerations. The concept of modular monoliths, where each module already represents a separate domain or subdomain, offers a potential solution to ease this transition.

However, it's important to note that the landscape of structuring monoliths for microservices migration is under-researched. More scientific inquiry and systematic studies are needed to develop best practices, tools, and frameworks that can guide organizations in this transition. By closing the research gap, we can make the path from monoliths to microservices more predictable and efficient. This will allow organizations to harness advantages of both architectural styles, alleviate the fear of being "stuck" with the monolith, and enable easier transitions from monoliths to microservices when the need to scale arises.

References:

1. Newman, S., 2021. *Building microservices*. O'Reilly Media, Inc.

2. Richards, M., 2022. *Software architecture patterns*, 2nd edition. O'Reilly Media, Inc.

3. Google Trends. (n.d.). Explore - Google Trends. Retrieved October 15, 2023. URL:https://trends.google.com/trends/explore?cat=1227&date=2013-01-01%202023-10-15&q=microservices,monolith&hl=en

4. Kalske, M., Mäkitalo, N. and Mikkonen, T., 2018. Challenges when moving from monolith to microservice architecture. In Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17 (pp. 32-47). Springer International Publishing.DOI: https://doi.org/10.1007/978-3-319-74433-9_3

5. Gos, K. and Zabierowski, W., 2020, April. The comparison of microservice and monolithic architecture. In 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 150-153). IEEE. DO: https://doi.org/10.1109/MEMSTECH49584.2020.9109514

6. Pahl, C. and Jamshidi, P., 2016. Microservices: A Systematic Mapping Study. CLOSER (1), pp.137-146. URL: https://www.scitepress.org/PublishedPapers/2016/57855/57855.pdf

7. Salah, T., Zemerly, M.J., Yeun, C.Y., Al-Qutayri, M. and Al-Hammadi, Y., 2016, December. The evolution of distributed systems towards microservices architecture. In 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST) (pp. 318-325). IEEE. DOI: https://doi.org/10.1109/ICITST.2016.7856721

8. Fritzsch, J., Bogner, J., Haug, M., Wagner, S. and Zimmermann, A., 2022. Towards an architecture-centric methodology for migrating to microservices. arXiv preprint arXiv:2207.00507. DOI: https://doi.org/10.48550/arXiv.2207.00507

9. Razzaq, A. and Ghayyur, S.A., 2023. A systematic mapping study: The new age of software architecture from monolithic to microservice architecture—awareness and challenges. Computer Applications in Engineering Education, 31(2), pp.421-451. DOI: https://doi.org/10.1002/cae.22586

10. Faustino, D., Gonçalves, N., Portela, M. and Silva, A.R., 2022. Stepwise migration of a monolith to a microservices architecture: Performance and migration effort evaluation. arXiv preprint arXiv:2201.07226. DOI: https://doi.org/10.48550/arXiv.2201.07226

11. Gouigoux, J.P. and Tamzalit, D., 2017, April. From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture. In 2017 IEEE international conference on software architecture workshops (ICSAW) (pp. 62-65). IEEE. DOI: https://doi.org/10.1109/ICSAW.2017.35

12. Chen, R., Li, S. and Li, Z., 2017, December. From monolith to microservices: A dataflow-driven approach. In 2017 24th Asia-Pacific Software Engineering Conference (APSEC) (pp. 466-475). IEEE. DOI: https://doi.org/10.1109/APSEC.2017.53

13. Seedat, M., Abbas, Q. and Ahmad, N., 2023. Systematic Mapping of Monolithic Applications to Microservices Architecture. arXiv preprint arXiv:2309.03796. DOI: https://doi.org/10.48550/arXiv.2309.03796

14. Velepucha, V. and Flores, P., 2023. A survey on microservices architecture: Principles, patterns and migration challenges. IEEE Access. DOI: https://doi.org/10.1109/ACCESS.2023.3305687

15. Gonçalves, N., Faustino, D., Silva, A.R. and Portela, M., 2021, March. Monolith modularization towards microservices: Refactoring and performance trade-offs. In 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C) (pp. 1-8). IEEE. DOI: https://doi.org/10.1109/ICSA-C52384.2021.00015

16. Cico, O., Souza, R., Jaccheri, L., Nguyen Duc, A. and Machado, I., 2021. Startups transitioning from early to growth phase-a pilot study of technical debt perception. In Software Business: 11th International Conference, ICSOB 2020, Karlskrona, Sweden, November 16–18, 2020, Proceedings 11 (pp. 102-117). Springer International Publishing. DOI: https://doi.org/10.1007/978-3-030-67292-8_8

17. Tsechelidis, M., 2023. Developing distributed systems with modular monoliths and microservices. URL: http://dspace.lib.uom.gr/handle/2159/29357

18. Fowler, M. (2015). MonolithFirst. Retrieved October 15, 2023. URL: https://www.martinfowler.com/bliki/MonolithFirst.html

19. Newman, S. (2015). Microservices for Greenfield? Retrieved October 15, 2023. URL: https://samnewman.io/blog/2015/04/07/microservices-for-greenfield/

20. Tilkov, S. (2015). Don't start with a monolith when your goal is microservices architecture. Retrieved October 15, 2023. URL: https://www.martinfowler.com/articles/dont-start-monolith.html

Article sent: 19.10.2023