Architectural Evolution from Monolithic to Microservices in Scalable Systems: A Case Study of Netflix

Carlos Henríquez [1], Jarol Derley Ramón Valencia[2], German Sánchez [3]

[1] *Phd., Universidad Autónoma del Caribe. Barranquilla, Colombia, https://orcid.org/0000-0002-0439-4954*

[2] *Phd. Programa de Ingeniería Ambiental. Universidad de Pamplona. Pamplona, Colombia*

[3] *PhD, Universidad del Magdalena, Santa Marta, Colombia, https://orcid.org/0000-0002-9069-0732*

Autor para correspondencia:

carlos.henriquez@uac.edu.co

Cite this article as: C. Henríquez, J.D. Ramon, G. Sanchez-Torres
"Architectural Evolution from Monolithic to Microservices in Scalable Systems: A Case Study of Netflix",
Prospectiva, Vol. 23 N° 1 2025

Recibido: 23/10/2024 / Aceptado: 09/03/2025

http://doi.org/ 10.15665/rp.v23i1.3683

ABSTRACT

This article examines how Netflix has evolved its technological infrastructure through the adoption of microservices architecture, which has enhanced its scalability, fault tolerance, and overall performance. Various architectural styles implemented during different phases of Netflix's growth are reviewed, highlighting how these styles influenced the company's ability to adapt to a significant increase in users and content demand. Key benefits and challenges faced during this transition are discussed, supported by analyses of academic literature and industry studies. Finally, the article evaluates the lessons learned and offers best practices for other organizations looking to adopt microservices. This exploration not only sheds light on Netflix's journey but also provides valuable insights into the broader implications of microservices architecture in modern software development.

Key Words: microservices; Netflix; architectural evolution; scalability; best practices

RESUMEN

Este artículo examina cómo Netflix ha evolucionado su infraestructura tecnológica mediante la adopción de una arquitectura de microservicios, lo que ha mejorado su escalabilidad, resiliencia y rendimiento general. Se analizan los distintos estilos arquitectónicos implementados en las diversas etapas de crecimiento de la empresa, destacando cómo estos han influido en su capacidad para adaptarse al aumento significativo de usuarios y demanda de contenido. Además, se discuten los principales beneficios y desafíos enfrentados durante esta transición, respaldados por análisis de literatura académica y estudios de la industria. Finalmente, el artículo evalúa las lecciones aprendidas y ofrece mejores prácticas para otras organizaciones que buscan adoptar microservicios. Esta exploración no solo arroja luz sobre la evolución de Netflix, sino que también proporciona información valiosa sobre las implicaciones más amplias de la arquitectura de microservicios en el desarrollo de software moderno.

Palabras clave: microservicios; Netflix; evolución arquitectónica; escalabilidad; mejores prácticas

1. INTRODUCTION

Architectural styles serve as conceptual frameworks that delineate the interactions among system components to achieve shared objectives. In software design, selecting an appropriate architectural style is critical for ensuring scalability, flexibility, maintainability, and efficiency [1]. Prominent architectural styles include monolithic architecture, Service-Oriented Architecture (SOA), event-driven architecture, and microservices architecture, each presenting unique advantages and challenges [2].

Monolithic systems integrate all software components into a single codebase, facilitating initial development but often resulting in complexities as the system scales, making dependency management increasingly challenging [3]. Conversely, SOA decomposes applications into independent services; however, it typically necessitates sophisticated middleware management for inter-service communication [4].

Microservices architecture, an evolution of SOA, has gained interest in large-scale systems due to its capability to decompose applications into smaller, independently deployable components that have horizontal scalability autonomously [5]. Nonetheless, this architectural approach introduces complexities in managing a network of services and can lead to communication latency [6] . Event-driven architecture proves advantageous for highly distributed systems; however, its implementation is often complicated by the necessity for asynchronous message management [7].

The transition from monoliths to microservices also represents a paradigm shift in software engineering [8]. Microservices architecture (MSA) relies on designing applications in a modular approach using small services that can be deployed independently and communicate through well-defined application programming interfaces (APIs) [9]. This approach to modularity improves operational efficiency and enables rapid adaptation in high-demand environments. The need for continuous integration and continuous deployment drives the move to MSA, given that microservices facilitate independent deployments and enhance fault tolerance [10]. Some significant experiences, such as Netflix, demonstrate the success of microservices in achieving scalability and performance. However, migrating from monoliths, often characterized by tightly coupled components, presents challenges such as operational overhead and complex communication mechanisms [8].

Many highly successful companies have faced significant challenges as they scaled due to the limitations of poorly designed software architecture. Rapid growth often exposes inefficiencies in monolithic systems, leading to performance bottlenecks, system failures, and difficulties in maintaining and deploying new features[11]. Industry giants such as Amazon, Uber, and Twitter encountered these obstacles, prompting them to adopt more scalable and modular architectures. In this context, Netflix stands out as a company that successfully navigated these challenges by transitioning to a microservices-based architecture, enabling it to handle millions of users seamlessly[12].

Recently, the implications of transitioning from monoliths to microservices have been the subject of study mainly because, while microservices can improve scalability and agility, migration also presents difficulties [13]. Monoliths sometimes outperform microservices in specific scenarios, particularly when deployed on limited computing infrastructure, raising questions about when and under what conditions organizations should consider migrating, especially those with fewer concurrent users [10]. The literature emphasizes the need for automated methodologies to refactor monolithic applications into microservices, as manual processes are slow and error-prone [13], [14]. In this context, decision-support models such as the Multi-Layer Fuzzy Cognitive Map (MLFCM) have recently provided frameworks for assessing the suitability of microservices adoption [15]. However, gaps remain regarding migration challenges, especially in high-demand solutions [16], and existing studies do not yet offer a comprehensive view of the advantages and disadvantages of microservices in terms of performance and resource allocation [17]. Additionally, many works focus on individual case studies without addressing broader implications [18], and there is little consensus on optimal strategies for identifying and extracting microservices, which are crucial for an effective migration path [19].

Netflix, a leading provider of online content, faced significant challenges in scaling its monolithic infrastructure to accommodate growing demand. To overcome these limitations, the company adopted a microservices architecture, improving system fault tolerance, scalability, and the agility to deploy new features [20]. Today, Netflix's global streaming service exemplifies the power of microservices in handling millions of users seamlessly. But what led to this transformation? By moving away from a monolithic system, Netflix built a more adaptable and efficient architecture. This article examines the evolution of Netflix's

software infrastructure, detailing the key components that drive its performance, the factors behind its success, and the lessons it offers for modern software development. We explore how adopting microservices has fueled Netflix's digital transformation and optimized its operations.

The remainder of this article is structured as follows. Section 2 outlines the methodology used to analyze Netflix's architectural evolution, detailing the sources and criteria considered. Section 3 presents the key findings, highlighting the impact of microservices on scalability, fault tolerance, and service delivery. Finally, Section 4 discusses the broader implications of Netflix's approach, comparing it with other industry practices.

2. METHODOLOGY

In Figure 1, the case study methodology on Netflix's architectural transformation is structured into three comprehensive phases following a qualitative approach.

Figure 1. Methodology

2.1 **Source Review**

An extensive review of scholarly articles, case studies, industry reports, and relevant news articles will be conducted to gather insights on Netflix's transition to microservices. This phase will focus on understanding the historical context of Netflix's architecture, highlighting the motivations for transitioning from a monolithic system to microservices. Sources will be critically evaluated for their relevance and contributions to the understanding of scalability and fault tolerance improvements

.

2.2. **Comparative Analysis**

This phase will involve a detailed comparison of the architectural styles adopted by Netflix throughout its growth journey. Key milestones, such as the initial monolithic architecture, the transition to microservices, and the integration of technologies like containers and continuous integration/continuous deployment (CI/CD), will be examined. The analysis will focus on identifying the benefits and drawbacks associated with each architectural choice and how these decisions influenced Netflix's operational efficiency and service delivery

.

2.3. Lessons Learned and Impact Assessment

The final phase will reflect on the lessons learned from Netflix's architectural transformation. This will include an examination of: Benefits: Identifying how the architectural changes have

improved performance, scalability, and user experience. Challenges: Discussing the obstacles encountered during the transition, such as team coordination, technological hurdles, and initial performance issues. Problems Identified: Analyzing specific problems that arose during implementation and how they were resolved, utilizing interviews and statements from Netflix engineers, extracted from technical blogs, conferences, and industry reports
.
This comprehensive evaluation will offer insights into the practical implications of adopting microservices in a large-scale technology company and contribute to the broader academic discourse on architectural strategies in software engineering.

3. RESULTS

a. The Netflix case

A search on Google Scholar using the terms "Netflix microservices" reveals numerous academic articles and case studies analyzing Netflix's transition from a monolithic architecture to a microservices-based approach. Although an exact number cannot be provided without an updated search, a previous bibliometric analysis identified 210 articles indexed in Web of Science related to Netflix [21] . It is reasonable to assume that a significant fraction of these articles addresses technical aspects, including microservices architecture. In addition to academic sources, numerous industry reports and specialized articles discuss this topic. For instance, Netflix's technical blog provides direct insights into its technological infrastructure and architectural decisions. Other resources, such as technology analysis articles, offer detailed perspectives on how Netflix has implemented and optimized its microservices architecture to enhance the platform's scalability and fault tolerance. Below we present the most relevant documents:

In [22] provides a detailed look at the microservices architecture that underpins Netflix's streaming platform. It highlights how Netflix has implemented microservices architecture to achieve high availability and efficient data processing, delivering a seamless streaming experience to over 180 million subscribers worldwide.

As illustrated in [23], Netflix under the leadership of former cloud architect Adrian Cockcroft, transitioned from a traditional development model to a microservices architecture. This shift enabled small, independent teams to develop and manage specific services, enhancing scalability and allowing for independent upgrades of each component. A key practice in this transformation was the assignment of dedicated data stores for each microservice, eliminating dependencies and granting teams the flexibility to choose the most suitable database for their needs. This approach facilitated the seamless integration of specialized services, collectively ensuring an efficient streaming experience for millions of users.

[24] details Netflix's transition from a monolithic to a microservices-based architecture. To address scalability and availability issues, Netflix adopted a microservices architecture that

allowed small, independent teams to develop and manage specific services. This transformation improved scalability and the ability to update each component independently. A key practice was allocating separate data stores for each microservice, avoiding dependencies and allowing each team to select the database best suited for its service. This strategy facilitated the deployment of specialized services that operate together to deliver efficient streaming experience to millions of users.

In [25] we take a detailed look at how Netflix migrated its IT infrastructure to the AWS public cloud and adopted a microservices architecture to improve the availability and scalability of its video streaming services. This transition allowed Netflix to focus on its core business and deliver a high-quality user experience globally.

This article [26] explores Netflix's transition from a monolithic to a microservices-based architecture. Initially, Netflix operated with a monolithic architecture that facilitated rapid development and end-to-end testing due to its unified code. However, as the application grew in complexity, limitations arose such as difficulties in scaling specific modules, technological constraints, and complications in debugging and development. To overcome these challenges, Netflix adopted a microservices architecture, allowing independent components to operate autonomously, which improved scalability, agility in feature development, and technological flexibility. This transformation facilitated more efficient resource allocation and easier onboarding of new developers, contributing to a more robust streaming experience for millions of users.

The paper [27] analyzes and compares two architectural patterns, monolithic and microservices, in the context of deploying web applications in cloud computing environments. The study focuses on how large Internet companies, such as Amazon, Netflix, and LinkedIn, have used microservices architecture to deploy large-scale applications in the cloud, allowing for dynamic allocation of computing resources based on demand.

In [28], NDBench is introduced as a performance evaluation tool developed by Netflix to assess and test its microservices and cloud data storage systems. NDBench enables dynamic tuning of evaluation parameters at runtime, allowing for rapid iteration across various tests and failure scenarios. Designed for continuous operation, the tool provides configurable workload patterns, support for customizable client APIs, and capabilities for detecting long-term issues such as memory leaks and heap pressure. This tool has played a crucial role in ensuring the high availability and efficiency of Netflix's services within a large-scale microservices environment.

The article [29] explores how Netflix Conductor facilitates workflow orchestration in microservices architectures. Conductor enables developers to design, automate, and manage complex workflows using a centralized platform, improving application efficiency and scalability. The tool offers configurable patterns and support for customizable client APIs, allowing teams to focus on business logic without worrying about managing the underlying infrastructure.

In [30], the transition from monolithic to microservices architectures is examined through case studies of companies such as Netflix. The discussion covers the challenges encountered, the strategies employed, and the lessons learned throughout these transformations. Additionally, practical code examples illustrate key concepts relevant to microservices implementation.

The article in [31] discusses Netflix's transition from a monolithic to a microservices-based architecture, highlighting how this transformation has enabled unprecedented scalability and efficient management of its IT infrastructure. Adopting microservices has made it easier for Netflix to manage its vast content catalog and deliver a seamless user experience to millions of subscribers around the world.

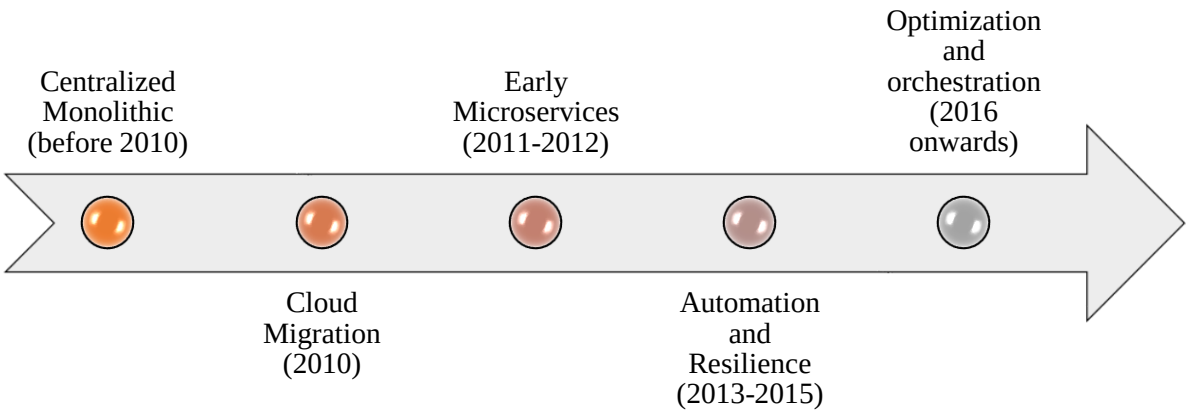
The following table presents a summary of the reviewed papers, providing an overview of the research on Netflix's transition from monolithic architecture to a microservices-based approach. These studies highlight the challenges faced, strategies implemented, and lessons learned in the process of modernizing its infrastructure.

Table 1. Summary of articles for the Netflix's case

Title	Source
System Design Netflix | A Complete Architecture	[22]
Adopting Microservices at Netflix: Lessons for Architectural Design	[23]
The Story of Netflix and Microservices	[24]
A design analysis of cloud-based microservices architecture at Netflix	[25]
Architectural Battle: Monolith vs. Microservices - A Netflix Story	[26]
Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud"	[27]
NDBench: Benchmarking Microservices at Scale	[28]
Workflow orchestration using Netflix Conductor	[29]
From Monolith to Microservices: Real-World Case Studies and Lessons Learned.	[30]
How Does Netflix Work? Microservices Architecture Explained	[31]

Throughout this transition (Figure 2), Netflix followed a progressive line of key changes to its architecture:

Figure 2. Netflix's technological transition.

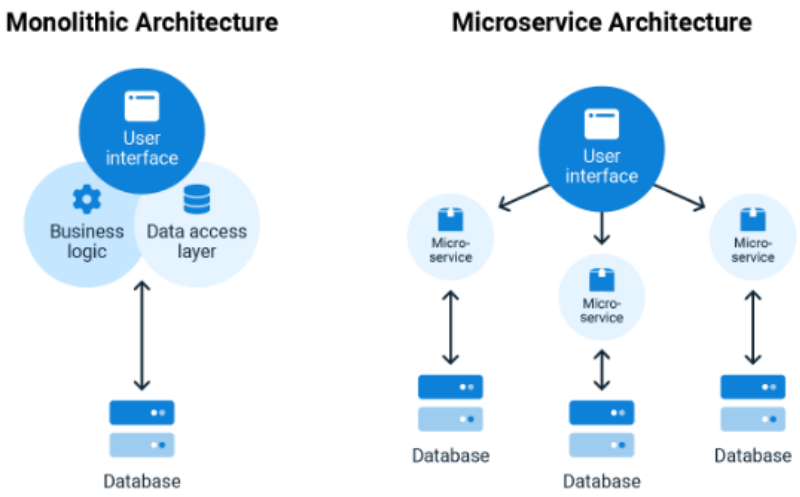


- Centralized Monolithic: Initially, Netflix operated under a monolithic model, where all services were tightly coupled into a single application, making scalability and fault tolerance difficult.
- Cloud Migration (2010): To improve availability and performance, Netflix began its migration to Amazon Web Services (AWS), gradually separating critical functions.
- Early Microservices (2011-2012): Independent microservices were introduced, each managed by autonomous teams, reducing reliance on a single centralized system.
- Automation and fault tolerance (2013-2015): With the expansion of microservices, tools such as Chaos Monkey and Spinnaker were implemented for fault tolerance testing and continuous deployment.
- Optimization and orchestration (2016 onwards): Netflix refined its microservices ecosystem with solutions such as Conductor for workflow orchestration and improved performance through advanced monitoring and load balancing techniques.

This evolution enabled Netflix to improve its scalability, availability and capacity for innovation, consolidating its position as a benchmark in the adoption of microservices at scale.

From the analysis of the articles reviewed on Netflix's transition from a monolithic architecture to microservices (Figure 3), several key lessons learned and impact assessments are identified:

Figure 3. Monolithic vs Microservices Architecture.

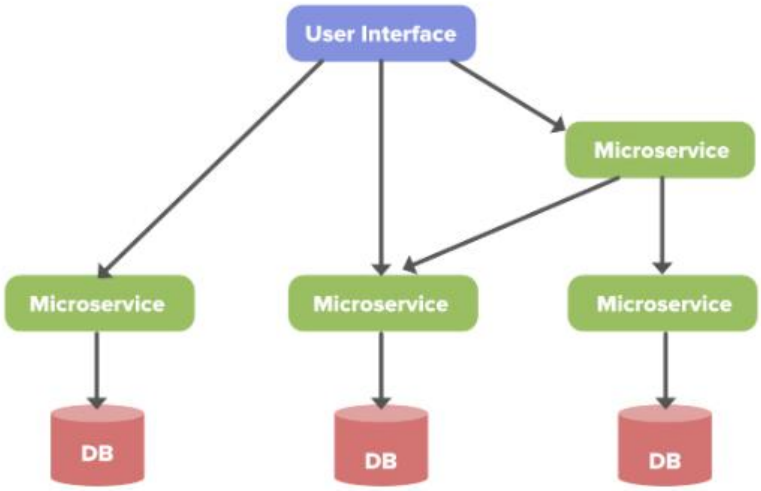


Source (CoderCo)

- Key Benefits

Improved scalability and availability: The microservices architecture enabled Netflix to horizontally scale its services and improve the overall availability of its platform.Agile deployment and resiliency: Tools such as Spinnaker and Chaos Monkey facilitated deployment automation and resiliency testing, reducing the impact of failures.Technology flexibility: Each microservice was able to use the most appropriate technology and database, optimizing performance and data management. Team autonomy: Small, specialized teams managed independent services, accelerating the development and innovation of new functionality (Figure 4).

Figure 4. Microservices architecture.



- Challenges encountered

Complexity in managing microservices: As the number of microservices increased, orchestration and monitoring became more complex, requiring specialized tools such as Conductor and Atlas.Communication between services: Dependency on API calls introduced latency and new points of failure, mitigated by load balancing and patterns such as circuit breakers. Progressive migration and associated risks: The transition from a monolithic system to microservices required well-planned strategies to avoid disruption of the system.

In the following table we present the problems found and their solution:

Table 2. Problem vs Solution

Problem	Solution
Uncontrolled failures in production	Implementing Chaos Engineering for fault tolerance testing
Latency in communication between microservices	Use of caches, load balancing and circuit breakers
Difficulty in coordinating services	Adoption of orchestration tools such as Conductor
Increase in operational complexity	Advanced monitoring with Atlas and centralization of logs with tools such as ELK Stack
Risks in data migration	Gradual migration strategies and use of specialized databases per service

The transition to a microservices architecture has had a profound impact on Netflix's platform, significantly enhancing its reliability and fault tolerance, ensuring a stable streaming experience for millions of users. By leveraging cloud infrastructure, Netflix has optimized scalability costs through efficient resource utilization. Additionally, the modularity of microservices has accelerated innovation, enabling faster development and deployment cycles. Furthermore, Netflix's successful implementation has set a benchmark in the industry, influencing other technology companies to adopt similar architectural approaches. This analysis confirms that, despite the challenges associated with microservices adoption, the long-term benefits in scalability, fault tolerance, and innovation justify the investment.

While Netflix's transition to microservices demonstrates the benefits of scalability and resilience, the process of migration from monolithic systems can be complex and challenging. To mitigate these difficulties, organizations have explored various automated migration techniques that facilitate a smoother transition. The next section examines these techniques and their role in streamlining the microservices adoption process.

b. Automated Migration Techniques

The automation of migration plays a key role in reducing the complexity of refactoring monolithic applications to MSAs. Recent advancements describe the effectiveness of automated approaches for large-scale systems:

- Systematic Refactoring: [13] presents a methodology to refactor ORM-based monolithic web applications into microservices, automating the extraction of APIs for inter-service communication. Evaluations across 120 applications indicate a ~72% successful refactoring rate, showing that automation can accelerate migration while preserving functional integrity (as confirmed by unit test consistency). MAT-Go: [32] introduces a tool specialized in transforming Go-based monolithic applications into microservices by leveraging the Abstract Syntax Tree (AST). MAT-Go identifies independent components, reorganizes them into modular microservices, and produces standalone executables. This approach improves maintainability and scalability, aligning with modern continuous integration and continuous delivery demands. Both [13] and [32] demonstrate how automation mitigates manual effort and human error, while still facing potential limitations. Nevertheless, Netflix's success exemplifies how combining automation with strategic architectural decisions can address inherent complexities when scaling microservices.

- Performance Trade-offs: When transitioning to MSAs, performance trade-offs can arise from increased network traffic, potential communication latencies, and the overhead of orchestrating multiple services. Studies such as [10] note that monolithic architectures may outperform microservices in simpler systems or those requiring minimal concurrency, especially in single-machine setups. This underscores the importance of thorough assessments before migrating, ensuring that the organization's scale and demands justify the added complexity. While many works demonstrate performance gains in high-load scenarios, such as reduced deployment times [9] and enhanced scalability [33], it is critical to consider the potential for performance degradation early in the modularization phase [8]. Additionally, some studies caution against increased operational costs and CPU usage [17]. Thus, a structured approach to microservices decomposition and robust testing frameworks are paramount to achieving the promised benefits.

  Beyond performance considerations and automation, there are broader organizational and cultural obstacles. In [18] it describes methodologies and tools for microservices identification, emphasizing standardized evaluation benchmarks and consideration of human factors. Studies agree that effective microservices identification is pivotal; poorly chosen boundaries can lead to service duplication, data inconsistencies, or communication bottlenecks. These issues reflect the broader complexities of transitioning to MSAs, such as inter-service data management, service granularity, and distributed transaction handling. Companies like Netflix showcase how autonomous teams, continuous delivery pipelines, and robust observability (logging, monitoring, tracing) underpin successful microservices adoption. Yet, not all organizations have the same technical maturity or cultural readiness, underscoring the need for context-specific strategies. Future research, as suggested by [18] and others, should work toward

integrating technical, organizational, and human dimensions into cohesive migration frameworks.

4. CONCLUSION

Netflix's transition from a monolithic architecture to a microservices-based system underscores the critical role of scalability and fault tolerance in modern application design. The company recognized that its monolithic approach could not sustain its rapid growth, resulting in system failures and downtime. By adopting microservices, Netflix achieved greater scalability, enabling independent service management, which significantly enhanced system reliability and fault tolerance.

A key takeaway from this transformation was the importance of automation in ensuring system fault tolerance. The implementation of tools like Chaos Monkey allowed Netflix to proactively identify and mitigate vulnerabilities, reinforcing the principle that designing for failure is essential in distributed systems. This proactive approach ensured operational continuity even when individual components encountered failures. Additionally, the decentralization of system architecture and the autonomy granted to development teams fostered innovation and responsiveness to market demands, further contributing to Netflix's success.

Furthermore, Netflix's strategic adoption of cloud infrastructure, particularly through Amazon Web Services (AWS), played a crucial role in maximizing scalability and operational efficiency. By leveraging cloud-based disaster recovery and storage capabilities, the company was able to focus on its core business—content delivery—while benefiting from the robustness of cloud services. Prioritizing user experience throughout this transformation enabled Netflix to rapidly deploy new features and personalized recommendations, solidifying its competitive advantage in the streaming industry. These insights serve as valuable lessons for organizations aiming to modernize their systems and adapt to the evolving technological landscape.

Another key aspect of microservices adoption is the automation of migration processes. Systematic refactoring techniques and automated tools, such as MAT-Go and ORM-based refactoring methodologies, play a role in reducing human effort and minimizing errors during migration. While these tools improve efficiency and accelerate the transition, they also introduce new challenges, such as increased network overhead and complex service orchestration. Organizations considering a shift to microservices must weigh these trade-offs carefully, ensuring that the benefits of modularization outweigh the potential drawbacks.

REFERENCES

[1] L. Bass, *Software architecture in practice*. Pearson Education India, 2012.

[2] C. Pahl y P. Jamshidi, «Microservices: A Systematic Mapping Study.», *CLOSER (1)*, pp. 137-146, 2016.

[3] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley, 2012.

[4] N. Niknejad, W. Ismail, I. Ghani, B. Nazari, M. Bahari, y others, «Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation», *Information Systems*, vol. 91, p. 101491, 2020.

[5] S. Newman, *Building microservices: designing fine-grained systems*. « O'Reilly Media, Inc.», 2021.

[6] S. Sarstedt y others, «Serverless: The Good, the Bad and the Ugly», Hochschule für Angewandte Wissenschaften Hamburg, 2024.

[7] G. Blair, «Complex distributed systems: The need for fresh perspectives», en *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1410-1421.

[8] D. Faustino, N. Gonçalves, M. Portela, y A. Rito Silva, «Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation», *Performance Evaluation*, vol. 164. 2024. doi: 10.1016/j.peva.2024.102411.

[9] A. Aggarwal y V. Singh, «Migration aspects from monolith to distributed systems using software code build and deployment time and latency perspective», *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 22, n.º 4. pp. 854-860, 2024. doi: 10.12928/TELKOMNIKA.v22i4.25655.

[10] G. Blinowski, A. Ojdowska, y A. Przybylek, «Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation», *IEEE Access*, vol. 10. pp. 20357-20374, 2022. doi: 10.1109/ACCESS.2022.3152803.

[11] H. Cervantes y R. Kazman, *Designing software architectures: a practical approach*. Addison-Wesley Professional, 2024.

[12] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, y Y. Al-Hammadi, «The evolution of distributed systems towards microservices architecture», en *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2016, pp. 318-325.

[13] F. Freitas, A. Ferreira, y J. Cunha, «A methodology for refactoring ORM-based monolithic web applications into microservices», *Journal of Computer Languages*, vol. 75. 2023. doi: 10.1016/j.cola.2023.101205.

[14] I. Trabelsi *et al.*, «From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis», *Journal of Software: Evolution and Process*, vol. 35, n.º 10. 2023. doi: 10.1002/smr.2503.

[15] A. Christoforou, A. S. Andreou, M. Garriga, y L. Baresi, «Adopting microservice architecture: A decision support model based on genetically evolved multi-layer FCM», *Applied Soft Computing*, vol. 114. 2022. doi: 10.1016/j.asoc.2021.108066.

[16] M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen, y S. Dustdar, «Microservices: Migration of a Mission Critical System», *IEEE Transactions on Services Computing*, vol. 14, n.º 5. pp. 1464-1477, 2021. doi: 10.1109/TSC.2018.2889087.

[17] F. Tapia, M. ángel Mora, W. Fuertes, H. Aules, E. Flores, y T. Toulkeridis, «From monolithic systems to microservices: A comparative study of performance», *Applied Sciences (Switzerland)*, vol. 10, n.º 17. 2020. doi: 10.3390/app10175797.
[18] I. Oumoussa y R. Saidi, «Evolution of Microservices Identification in Monolith Decomposition: A Systematic Review», *IEEE Access*, vol. 12. pp. 23389-23405, 2024. doi: 10.1109/ACCESS.2024.3365079.
[19] L. Qian, J. Li, X. He, R. Gu, J. Shao, y Y. Lu, «Microservice extraction using graph deep clustering based on dual view fusion», *Information and Software Technology*, vol. 158. 2023. doi: 10.1016/j.infsof.2023.107171.
[20] F. S. Miguel, N. Mareddy, A. K. Moorthy, y X. Liu, «Microservices for multimedia: video encoding», *Proceedings of the 1st Mile-High Video Conference*, 2022.
[21] A. Naranjo-Barnet y L. Fernández-Ram\'\irez, «Netflix en Web of Science: una aproximación bibliométrica», 2022.
[22] GeeksforGeeks, «System Design Netflix | A Complete Architecture», 2025.
[23] A. Cockcroft, «Adopting Microservices at Netflix: Lessons for Architectural Design», *F5 Networks Blog*, 2015.
[24] GeeksforGeeks, «The Story of Netflix and Microservices», 2025.
[25] «Un análisis de diseño de la arquitectura de microservicios basada en la nube en Netflix», *ICHI.PRO*, 2025.
[26] Y. Rai, «Architectural Battle: Monolith vs. Microservices - A Netflix Story», *DEV Community*, 2023.
[27] N. Dragoni *et al.*, «Microservices: Yesterday, Today, and Tomorrow», en *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 1-8. doi: 10.1109/ICSAW.2017.11.
[28] I. Papapanagiotou y V. Chella, «NDBench: Benchmarking Microservices at Scale», *arXiv preprint arXiv:1807.10792*, 2018.
[29] N. Sharma, «Workflow orchestration using Netflix Conductor», *Medium*, 2023.
[30] J. Wasike, «From Monolith to Microservices: Real-World Case Studies and Lessons Learned», *DEV Community*, 2024.
[31] TechAhead, «How Does Netflix Work? Microservices Architecture Explained», *TechAhead Blog*, 2020.
[32] K.-H. Hsu, Y.-Y. Chen, y A. W. Hou, «MAT-Go: A Study on Automated Transformation of Monolithic Go Application Systems Into Microservice Architecture», *Journal of Information Science and Engineering*, vol. 41, n.º 1. pp. 77-96, 2025. doi: 10.6688/JISE.202501_41(1).0005.
[33] H. W. Hutomo y A. S. Girsang, «Implementations of Microservice on Self-service Application Using Service Oriented Modelling and Architecture: A Case Study», *Journal of System and Management Sciences*, vol. 13, n.º 3. pp. 205-218, 2023. doi: 10.33168/JSMS.2023.0314.