EthAIca. 2024; 3:147
doi: 10.56294/ai2024147

REVIEW

Simplification of microservice development through low-code tools

Simplificación del desarrollo de microservicios mediante herramientas Low-code

Tomás Darquier[1], Pablo Alejandro Virgolini[1]

[1]Universidad Siglo 21, Licenciatura en Informática, S.C de Bariloche. Argentina.

Cite as: Darquier T, Virgolini PA. Simplification of microservice development through low-code tools. EthAIca. 2024; 3:147. https://doi.org/10.56294/ai2024147

Submitted: 03-09-2023 **Revised:** 21-01-2024 **Accepted:** 05-06-2024 **Published:** 06-06-2024

Editor: PhD. Rubén González Vallejo

ABSTRACT

Introduction: enterprise application development has undergone a significant transformation in recent years, driven by the need for scalable and flexible solutions. This evolution favored the adoption of microservice architectures, which, while offering benefits such as resilience and modular scalability, also introduced technical complexities in integration and maintenance. Against this backdrop, low-code tools emerged in response to the demand for more accessible and faster-to-implement solutions.
Development: the study addressed the technical fundamentals of microservice architectures, characterized by their ability to divide applications into independent services. However, these architectures posed challenges in terms of interoperability, the use of multiple languages, and increased testing and deployment efforts. Various commercial and academic platforms proposed low-code-based solutions aimed at simplifying microservice design through visual interfaces and reusable components. The technological proposal of the work integrated modern tools such as Java, Spring, Docker, and Kubernetes, along with visual development approaches, with the aim of facilitating the creation of efficient distributed systems.
Conclusions: the research concluded that low-code tools offered an effective way to mitigate the complexity inherent in microservices. These solutions reduced the technical burden without compromising software quality, promoting more inclusive and sustainable development environments. Likewise, the comparative analysis of platforms highlighted the need to continue refining these tools to achieve greater flexibility and standardization.

Keywords: Microservices; Low-code; Interoperability; Distributed architecture; Automation.

RESUMEN

Introducción: el desarrollo de aplicaciones empresariales experimentó una transformación significativa en los últimos años, motivado por la necesidad de soluciones escalables y flexibles. Esta evolución favoreció la adopción de arquitecturas de microservicios, las cuales, aunque ofrecieron beneficios como resiliencia y escalabilidad modular, también introdujeron complejidades técnicas en la integración y mantenimiento. Ante este panorama, surgieron herramientas Low-code como respuesta a la demanda de soluciones más accesibles y rápidas de implementar.
Desarrollo: el estudio abordó los fundamentos técnicos de las arquitecturas de microservicios, caracterizadas por su capacidad de dividir aplicaciones en servicios independientes. No obstante, estas arquitecturas implicaron desafíos en cuanto a interoperabilidad, uso de múltiples lenguajes y mayor esfuerzo en pruebas y despliegue. Diversas plataformas comerciales y académicas propusieron soluciones basadas en Low-code, orientadas a simplificar el diseño de microservicios a través de interfaces visuales y componentes reutilizables. La propuesta tecnológica del trabajo integró herramientas modernas como Java, Spring, Docker y Kubernetes, junto con enfoques visuales de desarrollo, con el objetivo de facilitar la creación de sistemas distribuidos eficientes.
Conclusiones: la investigación permitió concluir que las herramientas Low-code ofrecieron una vía efectiva

para mitigar la complejidad inherente a los microservicios. Estas soluciones permitieron reducir la carga técnica sin comprometer la calidad del software, promoviendo entornos de desarrollo más inclusivos y sostenibles. Asimismo, el análisis comparativo de plataformas destacó la necesidad de seguir perfeccionando estas herramientas para alcanzar mayor flexibilidad y estandarización.

Palabras clave: Microservicios; Low-code; Interoperabilidad; Arquitectura Distribuida; Automatización.

INTRODUCTION

In recent years, enterprise application development has evolved significantly, driven by the need for scalability, flexibility, and faster delivery times. This context has favored the adoption of microservice-based architectures, an approach that allows an application to be divided into small, independent, and easily deployable components. While this methodology offers benefits such as system resilience and individualized component scalability, it also introduces significant technical challenges. These include integration complexity, the management of multiple services developed in different languages, and the need for specialized tools to ensure interoperability between components.

Authors such as Newman(1) and Fowler have highlighted these difficulties, pointing out how the implementation of microservices requires advanced technical resources, as well as additional effort in the development, testing, and maintenance stages. Despite these barriers, the demand for more agile solutions has led to the emergence of platforms that seek to abstract this complexity. In this scenario, low-code tools have emerged, which allow the development of complex systems through graphical interfaces and preconfigured elements, thus reducing the need for manual programming.

Companies such as Microsoft (with Power Apps) and BettyBlocks have introduced platforms aimed at simplifying the design of distributed architectures, democratizing access to software development even for users with limited technical knowledge. This approach has also been addressed by academic research proposing domain-specific languages (DSLs) and visual environments based on techniques such as drag and drop or graphical blocks, similar to those used in educational platforms such as Scratch.(2)

This paper presents a technological proposal that seeks to optimize the development of microservices through low-code tools, integrating modern technologies such as Java, Spring, PostgreSQL, Docker, and Kubernetes, together with visual interfaces that facilitate the creation and integration of services. The research is based on a theoretical framework that analyzes both the technical fundamentals of microservices and the impact of low-code solutions on development productivity. Through this initiative, we aim to contribute to the design of more accessible, scalable, and sustainable systems, reducing the technical burden without sacrificing the quality and robustness of the resulting software.

DEVELOPMENT

THEORETICAL FRAMEWORK

Problem Domain

First, we must define what we mean when we talk about a distributed microservices architecture. As Newman(1) points out, "distributed microservices architectures divide an application into small, autonomous, independently deployable services, allowing organizations to scale specific components as needed and improve overall system resilience".

This type of architecture allows interoperability between microservices developed in different programming languages, due to its communication standards such as REST or gRPC and its service contracts.

Interoperability in microservice architectures is crucial, given that services can be built in different programming languages or platforms. Maintaining common standards in interfaces and service contracts is essential to ensure that these services can communicate effectively, although this also introduces additional complexities into the development and maintenance process.

The aforementioned complexities inherent in microservice architectures can delay development, testing, and integration times, affecting the production capabilities of a company or developer.

These systems are typically designed to be scalable and resilient, but that does not come without a cost. It can add complexity to the development process, as it is anothe y system that you may need to run to develop and test your services. Additional machines and expertise may also be required to keep this infrastructure running.(1)

Due to the above, many companies decided to provide the market with tools to try to mitigate the conflicts exposed. This is why web tools such as Microsoft's Power Apps or BettyBlocks had their genesis in providing developers with abstractions from the excessive complexity presented by distributed architectures, allowing the development of applications of this nature in a simpler way through low-code initiatives.

In turn, the proposed solution for simplifying the development of microservices using low-code tools has been and continues to be investigated by various authors in the academic field, highlighting the presence of the problem addressed. Examples include Chaudhary and Margaria[2] with their solution based on DSL and drag and drop, and Dhoke and Lokulwar with their proposal supported by an interface similar to that found on the Scratch platform, which allows backend services to be programmed using pre-built visual blocks and templates.

Tics

The technologies used in the development of the project are listed below:

Programming

Java: “Java is a cross-platform, object-oriented, network-centric language that can be used as a platform in itself”.

JavaScript: “JavaScript is a programming or scripting language that allows you to implement complex functions on web pages”.[3]

Frameworks and Libraries

Spring: “Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications on any deployment platform”.[4]

Apache Velocity: “Velocity is a Java-based template engine. It allows anyone to use a simple but powerful template language to reference objects defined in Java code”.[5]

Apache Jena: “Jena is a Java framework for creating Semantic Web applications. It provides extensive Java libraries to help developers develop code that handles RDF, RDFS, RDFa, OWL, and SPARQL”.[5]

Thymeleaf: “The main goal of Thymeleaf is to bring natural and elegant templates to your HTML development workflow, which can be correctly rendered in browsers and also work as static prototypes.”

Bootstrap: “Bootstrap is a front-end framework used to develop web applications and mobile-first sites, i.e., with a layout that adapts to the screen of the device used by the user”.[6]

Database Engine

PostgreSQL: “PostgreSQL, commonly pronounced ‘Post-GRES’, is an open source database with a solid reputation for reliability, flexibility, and support for open technical standards”.[7]

Other Technologies

GitHub: “GitHub is a platform where you can store, share, and work together with other users to write code”.[8]

RDF: “RDF is a standard model for exchanging data on the Web. RDF has features that make it easy to combine data even if the underlying schemas differ”.

SPARQL: “SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions via RDF”.

HTML5: “The latest stable version of HTML, HTML5 converts HTML from a simple markup format for structuring documents into a complete application development platform”.[3]

CSS: “CSS, short for Cascading Style Sheets, is a declarative language that controls the appearance of web pages in the browser”.[3]

Docker: “Docker packages software into standardized units called containers that include everything needed for the software to run, including libraries, system tools, code, and runtime”.

Kubernetes: “Kubernetes is a portable and extensible open source platform for managing workloads and services. Kubernetes makes automation and declarative configuration easy”.[9]

MinIO: “Minio is a high-performance distributed object storage server designed for large-scale private cloud infrastructure”.[7,10,11]

Apache Kafka: “Apache Kafka is an open source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications”.[5,12]

IntelliJ IDEA: “IntelliJ IDEA features one of the most powerful code editors in the industry. Understand the ins and outs of your code thanks to initial indexing.”[13,14]

Postman: “Postman is a platform for creating and using APIs. Postman simplifies every step of the lifecycle and streamlines collaboration so you can build better APIs faster”.[10,15,16]

Trello: “Trello is a visual tool that allows teams to manage any type of project and workflow, as well as monitor tasks”.[17]

Competition

The following table shows and compares the main features of the different solutions available on the market that are similar to the one proposed in this project.

Table 1. Comparison of companies with low-code solutions

Pages	Automatic Microservice Generation	REST support	gRPC support	Integration with external systems	Container Support
outsystems.com	X	X		X	X
mendix.com	X	X		X	X
appian.com	X	X		X	
microsoft.com/ **/power-apps		X		X	
bettyblocks.com	X	X		X	X

CONCLUSIONS

Throughout this paper, we have seen how the evolution of business application development has generated new technical demands, particularly with regard to the adoption of microservice-based architectures. While this approach offers substantial benefits such as individualized scalability, system resilience, and greater flexibility in component implementation, it also introduces a number of complex challenges. Interoperability between services, the management of different programming languages, the need for robust infrastructure, and the sophistication of distributed system maintenance represent significant obstacles for development teams.

Faced with this technical complexity, low-code tools are emerging as a strategic solution that seeks to reduce the operational burden without compromising software quality. By enabling application development through visual interfaces, predefined graphical elements, and intuitive environments, these platforms democratize access to distributed system development, even for users with limited technical knowledge. Solutions offered by companies such as Microsoft, BettyBlocks, or academic platforms based on DSL and drag-and-drop techniques demonstrate the viability of this trend and its potential impact on the productivity and efficiency of work teams.

The proposal presented in this project is part of this innovative trend, integrating established technologies such as Java, Spring, PostgreSQL, Docker, and Kubernetes with low-code tools that make it easier to build, deploy, and integrate microservices. This combination not only simplifies the technical process but also promotes the creation of scalable and sustainable systems aligned with current market demands.

The comparative analysis carried out between different low-code platforms highlights the strengths and limitations of existing solutions, underscoring the need to continue developing more comprehensive, flexible, and microservice-oriented tools. Despite the progress made, challenges remain related to the integration of standards, performance optimization, and user training in the effective use of these technologies.

Low-code tools represent a valid response to the complexity of modern architectures, offering a bridge between technical sophistication and the need for agility. The solution proposed in this paper provides a balanced approach that can contribute significantly to the development of more efficient software that is accessible and adaptable to the constant changes in the technological environment.

BIBLIOGRAPHICAL REFERENCES

1. Newman S. Building microservices: Designing fine-grained systems. Newton: O'Reilly Media; 2015.

2. Chaudhary HAA, Ahmed T. Integration of micro-services as components in modeling environments for low code development. ISP RAS. 2021;33(4):19-30. doi:10.15514/ISPRAS-2021-33(4)-2

3. Mozilla Developer Network. CSS. 2023. Available from: https://developer.mozilla.org/es/docs/Glossary/CSS

4. Spring. Spring Framework. 2024. Available from: https://spring.io/projects/spring-framework

5. Apache Software Foundation. Apache Kafka. 2024. Available from: https://kafka.apache.org/

6. Rock Content. What is Bootstrap? 2020. Available from: https://rockcontent.com/es/blog/bootstrap/

7. IBM. Minio. 2021. Available from: https://www.ibm.com/docs/es/cloud-private/3.2.x?topic=private-minio

8. GitHub. About GitHub and Git. 2024. Available from: https://docs.github.com/es/get-started/start-your-

journey/about-github-and-git

9. DigitalOcean. DigitalOcean Managed Kubernetes. 2024. Available from: https://www.digitalocean.com/products/kubernetes

10. Postman. What is Postman? 2024. Available from: https://www.postman.com/product/what-is-postman/

11. Banco Central de la República Argentina. Cotizaciones por fecha. 2024. Available from: http://www.bcra.gob.ar/PublicacionesEstadisticas/Cotizaciones_por_fecha_2.asp

12. Brown T, Smith L. The impact of Low-code platforms and intuitive interfaces on software development efficiency. Journal of Software Innovation. 2023;18:75-89.

13. Consejo Profesional de Ciencias Informáticas de la Provincia de Córdoba. Honorarios Recomendados. 2024. Available from: https://cpcipc.org.ar/honorarios-recomendados/

14. Elgheriani NS, Ahmed NA. Microservices vs. monolithic architectures. International Journal of Applied Science and Technology. 2022;4:501-14. doi:10.47832/2717-8234.12.47

15. Misic B, Novkovic M, Ramac R, Mandic V. Do the microservices improve the agility of software development teams? In: International Scientific Conference on Industrial Systems; 2017. p. 170-5.

16. Schwaber K, Sutherland J. Scrum: The art of doing twice the work in half the time. Houston: Crown Business; 2010.

17. Trello. Trello Tour. 2023. Available from: https://trello.com/es/tour

FINANCING
None.

CONFLICT OF INTEREST
The authors declare that there is no conflict of interest.

AUTHOR CONTRIBUTION
Conceptualization: Tomás Darquier, Pablo Alejandro Virgolini.
Data curation: Tomás Darquier, Pablo Alejandro Virgolini.
Formal analysis: Tomás Darquier, Pablo Alejandro Virgolini.
Research: Tomás Darquier, Pablo Alejandro Virgolini.
Methodology: Tomás Darquier, Pablo Alejandro Virgolini.
Project management: Tomás Darquier, Pablo Alejandro Virgolini.
Resources: Tomás Darquier, Pablo Alejandro Virgolini.
Software: Tomás Darquier, Pablo Alejandro Virgolini.
Supervision: Tomás Darquier, Pablo Alejandro Virgolini.
Validation: Tomás Darquier, Pablo Alejandro Virgolini.
Visualization: Tomás Darquier, Pablo Alejandro Virgolini.
Writing - original draft: Tomás Darquier, Pablo Alejandro Virgolini.
Writing - review and editing: Tomás Darquier, Pablo Alejandro Virgolini.