Monolitos vs. Microservicios en Arquitectura de Software: Perspectivas para un Desarrollo Eficiente

Valentín Torassa Colombero, Juan Pablo Estelles, Laureano Gallegos, Pedro Lopez

Facultad de Tecnología Informática, Universidad Abierta Interamericana (UAI), Rosario, Argentina
{Torassa Colombero}valentintorassa@hotmail.com
{Estelles}estellesjp@gmail.com
{Gallegos}laureano.gallegos@uai.edu.ar
{Lopez}Pedro.Lopez@uai.edu.ar

Resumen. Una de las decisiones más importantes a la hora de desarrollar un software es elegir el paradigma de arquitectura a utilizar; En la actualidad las empresas se debaten entre dos de los enfoques principales: los monolitos y los microservicios. En este estudio, nos adentramos en un análisis exhaustivo de ambas opciones desde una variedad de perspectivas y aspectos clave. Exploramos la estructura intrínseca de cada una, comprendiendo cómo se organizan y cómo interactúan sus componentes. Profundizaremos en la implementación práctica, evaluando los desafíos y las ventajas que cada una presenta en entornos de desarrollo reales. Analizáremos casos de estudio de empresas líderes en la industria que han adoptado uno u otro enfoque o incluso que han migrado entre ellos y examinaremos el despliegue y mantenimiento a largo plazo, considerando cómo cada arquitectura aborda las dificultades de escalar y mantener sistemas a lo largo del tiempo. En conjunto, este estudio proporciona una visión holística que ayuda a los profesionales del desarrollo y arquitectura de software para comprender mejor las implicaciones y consideraciones asociadas con la elección de arquitecturas monolíticas o de microservicios en proyectos de software.

Palabras clave. Desarrollo, Arquitectura, Monolitos, Microservicios, Diseño.

1 Introducción

La arquitectura de software es el fundamento sobre el cual se construyen todos los sistemas de software. Es como el esqueleto de un edificio, proporcionando la estructura y el soporte necesarios para que un sistema funcione de manera eficiente. Una arquitectura sólida puede facilitar el desarrollo, la prueba y el mantenimiento, mientras que una arquitectura deficiente puede dificultar todos estos aspectos y limitar la capacidad de una construcción para cumplir con los requisitos del usuario final.

En el software, la arquitectura puede definirse como la estructura organizativa del sistema, que incluye componentes, módulos, relaciones y principios que guían el diseño y la implementación del software. La elección de la arquitectura adecuada es crucial, ya que puede tener un impacto significativo en la calidad, el rendimiento, la escalabilidad y la mantenibilidad del software resultante. La arquitectura puede, a su vez, verse afectada por otros aspectos del desarrollo de software, como la distribución del trabajo entre los miembros del equipo, la elección de tecnologías y herramientas, y el proceso de desarrollo en sí mismo.[1]

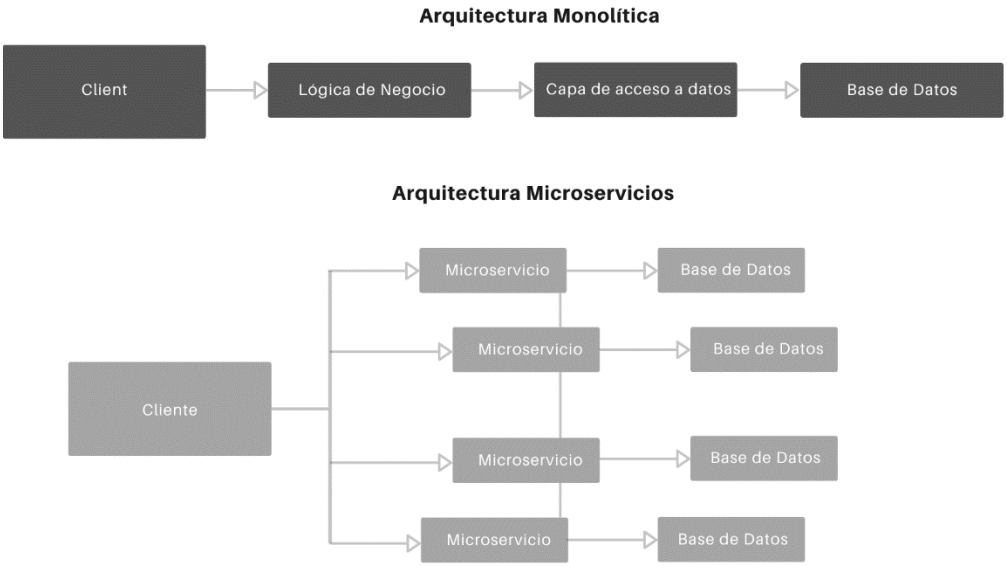

Fig. 1. Gráfico comparativo de arquitecturas

En un entorno tan dinámico como el del desarrollo de software, la selección de la arquitectura subyacente juega un papel crucial en el destino de un proyecto. Esta elección no solo determina la estructura del sistema, sino que también impacta en diferentes aspectos cuya influencia solo pueden verse en las etapas más tardías del

desarrollo o incluso una vez finalizado o en etapas posteriores. En este sentido, la dicotomía entre los monolitos y los microservicios ha surgido como un tema fundamental en el diseño y la implementación de sistemas de software.[2]

La arquitectura de software no es simplemente una cuestión técnica; sino que tiene implicaciones en la estrategia empresarial. La elección entre un enfoque monolítico y uno de microservicios puede influir en la agilidad de la empresa, su capacidad para adaptarse a las cambiantes demandas del mercado y su capacidad para innovar rápidamente.[1] La elección de la arquitectura, a su vez, puede tener ramificaciones en términos de costos de desarrollo y mantenimiento a largo plazo. Por lo tanto, es crucial comprender a fondo las características como también las ventajas y desventajas asociados con cada enfoque antes de tomar una decisión y dimensionar cuanto puede influir en el éxito o fracaso de un software.

2 Análisis

En el ámbito de la ingeniería de software, la elección de la arquitectura subyacente desempeña un papel crucial en la eficiencia de un sistema y si bien se realiza en una etapa temprana del desarrollo, es uno de los pilares para lograr construir un sistema funcional y eficaz. [1]

2.1 Descripción de las arquitecturas

Existen dos enfoques esenciales en el diseño: las arquitecturas monolíticas y de microservicios. Estos paradigmas, fundamentales en el desarrollo de sistemas, presentan soluciones únicas para los desafíos inherentes a la organización y construcción de software.[2]

Arquitecturas Monolíticas

Las arquitecturas monolíticas, caracterizadas por un enfoque donde se agrupan todas las funciones y servicios dentro de una base única y centralizada de código. Representa un modelo tradicional de desarrollo de software. En este diseño, todos los componentes de la aplicación, desde la interfaz de usuario hasta la lógica de negocio y el acceso a datos, están encapsulados en un solo paquete. Este diseño cohesivo simplifica la gestión y el despliegue, ya que toda la aplicación se implementa como una única entidad. La piedra angular de esta arquitectura es un servidor único y masivo que maneja todas las solicitudes y respuestas.[3]

La comunicación interna en una arquitectura monolítica se realiza a menudo a través de llamadas directas a funciones o mediante la manipulación de objetos compartidos. Esta estructura, si bien facilita la comprensión del flujo de control, puede resultar en un acoplamiento estrecho, contradiciendo los principios SOLID y las buenas prácticas en general. Estos principios sostienen que un desarrollo robusto y escalable debe basarse en un bajo acoplamiento y una alta cohesión, elementos cruciales para la sostenibilidad y adaptabilidad de cualquier sistema. El despliegue de una aplicación monolítica implica la implementación completa en un servidor, y aunque esto

simplifica la gestión en ciertos casos, puede generar desafíos en cuanto a la escalabilidad y la capacidad de adaptación a las demandas cambiantes del entorno.[5] Sin embargo, un servidor único que maneja todas las solicitudes y respuestas se convierte en el núcleo central de la estructura, brindando una eficiencia operativa que se traduce en una administración más sencilla durante las etapas iniciales del desarrollo.[3]

El acoplamiento estrecho inherente a las arquitecturas monolíticas, donde todas las funciones y servicios están integrados en un único paquete, puede generar problemas a medida que la aplicación evoluciona. La comunicación interna a través de llamadas directas a funciones o manipulación de objetos compartidos puede conducir a una estructura rígida y difícil de modificar.

Por un lado, el despliegue de una aplicación monolítica, al implementarse completamente en un solo servidor, simplifica la gestión en ciertos escenarios; Pero por el otro, enfrenta desafíos significativos en términos de escalabilidad y capacidad de adaptación a los cambios en el entorno. La necesidad de una implementación completa en cada actualización puede resultar ineficiente y propensa a errores, especialmente cuando se buscan ajustes específicos o en ciertas funcionalidades. La escalabilidad vertical, añadiendo más recursos al servidor único, puede tener límites prácticos, lo que resulta en un rendimiento subóptimo o incluso en la necesidad de realizar cambios arquitectónicos significativos.[5] La capacidad de escalar horizontalmente, es decir, agregar instancias adicionales de la aplicación para distribuir la carga, suele ser más limitada en una arquitectura monolítica.

La clave para superar estas limitaciones radica en encontrar un equilibrio entre la simplicidad inicial y la necesidad de escalabilidad y adaptabilidad a largo plazo. Las estructuras monolíticas pueden ser adecuadas para proyectos más pequeños o con requisitos estables.

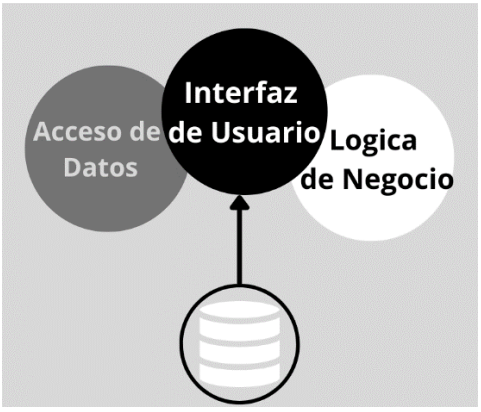

Fig. 2. Gráfico Arquitectura Monolítica.

Arquitecturas de Microservicios

La arquitectura de microservicios emerge como un enfoque innovador, proporcionando una alternativa prácticamente opuesta a la arquitectura monolítica tradicional. Este paradigma arquitectónico se basa en la idea de descomponer una aplicación en una serie de servicios independientes, cada uno enfocado en realizar una función específica.[5] A diferencia de las arquitecturas monolíticas, donde todas las funciones y servicios se integran en un único paquete, los microservicios permiten un modularidad que facilita la escalabilidad, la implementación y el mantenimiento.

En un entorno de microservicios, cada componente funcional de la aplicación se convierte en un servicio independiente. Cada servicio tiene su propia lógica de negocio, base de datos y se comunica con otros servicios a través de interfaces bien definidas. Esta separación proporciona una mayor independencia y autonomía a cada microservicio, permitiendo a los equipos de desarrollo trabajar de manera más eficiente y ágil.[4]

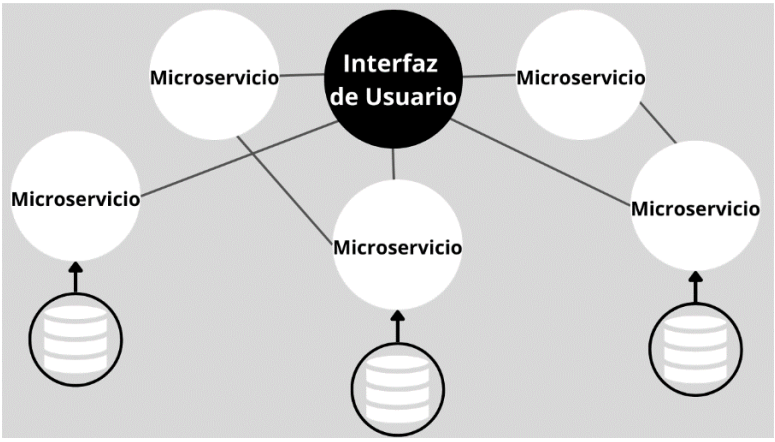

Fig. 3. Gráfico Arquitectura Microservicios.

La comunicación entre microservicios generalmente se realiza a través de interfaces de programación de aplicaciones (API) basadas en protocolos como HTTP o SOAP.[5] Esta elección de protocolos facilita la interoperabilidad y permite a los microservicios comunicarse de manera eficiente y sin acoplamiento excesivo. Se fomenta, de esta manera, la independencia entre los equipos de desarrollo, ya que cada microservicio puede ser implementado y actualizado de manera independiente sin afectar a otros componentes de la aplicación.

Uno de los beneficios clave de la arquitectura de microservicios es su capacidad para escalar horizontalmente. En lugar de tener que escalar toda la aplicación monolítica, los equipos pueden escalar servicios individuales según sea necesario para satisfacer la demanda específica de ese servicio. Esto proporciona una mayor modularidad en la gestión de recursos y optimiza el rendimiento de la aplicación en

función de las cargas de trabajo particulares de cada microservicio pudiendo destinar más recursos a aquellas que lo requieran, a diferencia de la arquitectura monolítica, que debía proporcionar de manera equitativa incluso las funciones o servicios menos utilizados del sistema.[4]

El modularidad inherente a los microservicios también facilita la implementación continua y la introducción de nuevas características. Los equipos pueden trabajar de forma independiente en la mejora de un servicio sin afectar el funcionamiento de otros.[5] Esto acelera el ciclo de desarrollo y permite la adaptación rápida a cambios en los requisitos del negocio o en el entorno operativo. Por esto mismo, si un microservicio falla, no afecta directamente a otros servicios, ya que operan de manera independiente. Los mecanismos de manejo de errores y recuperación pueden implementarse específicamente para cada microservicio, mejorando la robustez general de la aplicación.

No obstante, la arquitectura también presenta desafíos que no afectaban a la arquitectura monolítica. La fragmentación de la aplicación en múltiples servicios independientes puede aumentar la complejidad operativa y de desarrollo. La coordinación entre estos microservicios puede ser complicada y requerir una sólida planificación y diseño de interfaces. Por esto, el desarrollo de una arquitectura de microservicios suele ser más tardado en comparación con un enfoque monolítico. La necesidad de gestionar múltiples servicios, cada uno con su propia lógica de negocio y base de datos, implica un mayor esfuerzo en términos de diseño, implementación y pruebas. La complejidad inherente al despliegue y mantenimiento de numerosos servicios puede llevar a una curva de aprendizaje empinada y a una mayor propensión a errores durante el ciclo de desarrollo por lo que se requiere un equipo de desarrollo experimentado y mejor preparado para poder sobrellevar estas dificultades. [6]

2.2 Implementación y Desarrollo Práctico

La implementación y desarrollo práctico de las arquitecturas monolíticas y de microservicios presentan diferencias significativas que influyen en la elección de la estructura arquitectónica más adecuada para un proyecto específico. Si bien ambos enfoques tienen sus ventajas y desventajas, es crucial considerar diversos factores, como el tamaño y la complejidad del proyecto, las capacidades del equipo de desarrollo y los requisitos de escalabilidad y mantenimiento a largo plazo; Condiciones que, más allá de las bases teóricas de la arquitectura, impactan el desarrollo en aspectos prácticos y en escenarios reales. En estos contextos, los involucrados deben tomar decisiones informadas sobre la elección de la arquitectura más adecuada, evitando caer en errores concepción; evitando situaciones donde se sobredimensione el diseño o se subestime la complejidad del proyecto, lo que podría llevar a desviaciones en el desarrollo y a la necesidad de reajustes costosos en etapas posteriores.[1]

En el caso de las arquitecturas monolíticas, el proceso de implementación y desarrollo práctico suele ser más directo en comparación con las arquitecturas de microservicios.

Esto se debe a que todas las funciones y servicios están integrados en un único paquete, lo que simplifica la gestión y el despliegue.[4] Para proyectos pequeños o medianos con requisitos estables y bien definidos, la arquitectura monolítica puede ser la opción más práctica y eficiente.

El desarrollo de una aplicación monolítica implica trabajar en un entorno cohesivo donde todas las partes de la aplicación están interconectadas y comparten recursos comunes. Esto facilita la comprensión del código y la colaboración entre los miembros del equipo, ya que todos trabajan dentro del mismo contexto.[6] Por otro lado, el despliegue se realiza en un único servidor, lo que simplifica la gestión operativa y reduce la complejidad en comparación con los entornos distribuidos de las arquitecturas de microservicios.

Sin embargo, a medida que el proyecto crece en tamaño y complejidad, las limitaciones de las arquitecturas monolíticas pueden volverse más evidentes. La necesidad de implementar y desplegar la aplicación completa en cada actualización puede resultar en tiempos de desarrollo más largos y un ciclo de lanzamiento más lento. A su vez, la escalabilidad de una aplicación monolítica puede ser limitada, ya que agregar recursos adicionales al servidor único puede no ser suficiente para manejar un aumento en la carga de trabajo.[5]

Las arquitecturas de microservicios, por su parte, ofrecen una mayor flexibilidad y modularidad en el desarrollo práctico de aplicaciones, pero conllevan una mayor complejidad. La implementación de microservicios requiere un enfoque más profundo y una planificación metódica debido a la necesidad de gestionar múltiples servicios independientes.[4] Cada microservicio debe ser desarrollado, probado e implementado de forma individual, lo que aumenta la carga de trabajo y la complejidad del proceso de desarrollo, conllevando de esta forma que el desarrollo sea más costoso en términos de recursos y tiempo.[7]

La comunicación entre microservicios puede presentar ciertos inconvenientes, ya que los servicios deben interactuar entre sí a través de interfaces concretas. Esto requiere una sólida planificación y diseño para garantizar una comunicación fluida y eficiente entre los componentes de la aplicación.[7] En general, la gestión del ciclo de vida de los microservicios, incluyendo la escalabilidad, el monitoreo y el mantenimiento, es más compleja debido a la naturaleza distribuida y en constante comunicación de la arquitectura de microservicios.

A pesar de estas dificultades, las arquitecturas de microservicios ofrecen ventajas significativas como la capacidad de escalar horizontalmente servicios individuales según sea necesario permitiendo una adaptación más ágil a los cambios en los requisitos del negocio y las demandas del mercado.[4]

Tabla 1. Cuadro comparativo de la implementación práctica.

Implementación Practica	**Arquitectura Monolítica**	**Arquitectura de Microservicios**
Beneficios	Ventaja en proyectos pequeños o medianos debido a su simplicidad inicial.	Beneficios significativos en proyectos grandes y sofisticados donde la escalabilidad es clave.
Desventajas	Difícil de escalar	Desarrollo e implementación extensa en términos de tiempo y recursos
Características	Gestión de un único servidor, permitiendo foco en el desarrollo de funcionalidades.	Modularidad, independencia y capacidad de escalar horizontalmente

2.3 Casos de Estudio

Walmart, uno de los minoristas más grandes del mundo, ha mantenido una arquitectura monolítica durante muchos años para respaldar sus operaciones comerciales. La empresa optó por esta estructura centralizada debido a su simplicidad y facilidad de gestión en las etapas iniciales de su desarrollo. En los primeros días de Walmart, una arquitectura monolítica era suficiente para satisfacer sus necesidades tecnológicas, ya que les permitía consolidar todas sus funciones y servicios en una única base de código y les facilitaba el despliegue y la gestión de la aplicación en un servidor único allá por 1996.[8]

Sin embargo, a medida que Walmart creció y se expandió globalmente, comenzaron a surgir limitaciones con su arquitectura monolítica. No obstante, descubrió que la implementación de un monolito a largo plazo, por la naturaleza de su rubro y las costumbres de sus consumidores, fue útil y les ayudo a reducir los costos operativos, mejorar su rendimiento y la experiencia del cliente.[8]

Amazon, el gigante del comercio electrónico, es conocido por su enfoque pionero en la adopción de arquitecturas de microservicios en su shop. En el año 2001, el sitio web de Amazon se basaba en una arquitectura monolítica de varios niveles. Aunque los ingenieros reconocían que esta arquitectura no era escalable, la complejidad asociada con la actualización del sistema era considerable. Se encontraron con unidades de código que servían a un solo propósito, como una función que renderizaba el botón "Comprar" en las páginas de detalle de productos. Para abordar

esta complejidad, Amazon orientó a los desarrolladores a crear funciones que pudieran comunicarse externamente a través de sus propias APIs de servicios web.[9] Este enfoque permitió la creación de un sistema desacoplado, donde las funciones podían actualizarse de forma independiente.

El crecimiento acelerado de la base de código y la necesidad de implementar actualizaciones y nuevos proyectos de manera eficiente, sin la carga adicional de fusionar funciones, impulsaron a Amazon a buscar una mejora en su operatividad, llevándolos hacia la arquitectura de microservicios. Esta decisión se fundamentó en su enfoque centrado en el cliente y su mentalidad orientada al crecimiento rápido y la innovación continua. La capacidad de desplegar y escalar servicios individualmente permitió a Amazon introducir rápidamente nuevas características y adaptarse ágilmente a las cambiantes demandas del mercado. La modularidad de los microservicios también facilitó la colaboración entre equipos de desarrollo y estimuló la innovación en toda la organización,[9] lo que contribuyó al éxito sostenido de Amazon como uno de los principales destinos de compras en línea a nivel global.

eBay, el conocido mercado en línea, ha experimentado una transición significativa en su arquitectura de software en 2017. Inicialmente, eBay se basaba en una arquitectura monolítica para respaldar su plataforma de comercio electrónico.[10] La decisión de eBay de migrar hacia una arquitectura de microservicios surgió de la necesidad de abordar desafíos específicos que enfrentaban en términos de escalabilidad, flexibilidad y tiempo de comercialización.

eBay se dio cuenta de que su arquitectura monolítica no podía adaptarse fácilmente a la creciente complejidad de su plataforma y estaba perjudicando su eficiencia y experiencia de usuario. La migración a microservicios les permitió descomponer su aplicación en componentes más pequeños y manejables, lo que facilitó el desarrollo y la división de sus funciones en microservicios permitió dedicar servidores a las funciones de la página más utilizadas como la publicación y sobre todo la visualización de estas publicaciones,[10] la capacidad de escalar servicios individualmente les permitió mejorar la capacidad de respuesta de su plataforma y brindar una experiencia de usuario más fluida.

Amazon Prime Video, un servicio de suscripción de Amazon que ofrece una amplia gama de beneficios, también ha experimentado cambios en su arquitectura de software a lo largo del tiempo. Inicialmente, Prime Video adoptó una arquitectura basada en microservicios distribuidos y servicios serverless para su herramienta de revisión de calidad de audio y video de los streams. Esta arquitectura permitía una implementación más rápida y flexible, dividiendo los streams en unidades más pequeñas y distribuyendo el procesamiento en diferentes componentes.[11]

Conforme la aplicación crecía, surgieron problemas de costos, cuellos de botella y escalabilidad. Como respuesta, el equipo de calidad de Prime Video decidió migrar hacia un enfoque monolítico, trasladando todos los componentes a un único proceso. Esta transición permitió reducir costos hasta un 90% y mejorar la calidad global y la experiencia del usuario. Aunque esta decisión ha generado controversia en la comunidad de tecnología, algunos expertos argumentaron que para equipos de

desarrollo cohesionados y no demasiado grandes, el enfoque monolítico puede ser más efectivo que la arquitectura orientada a microservicios y aunque Amazon Prime originalmente optó por una arquitectura de microservicios por su escalabilidad y flexibilidad, la empresa reconoció que la complejidad y los costos asociados con esta arquitectura superaban sus beneficios en ese momento.[11]

2.4 Despliegue y mantenimiento a largo plazo

El despliegue y mantenimiento a largo plazo son aspectos críticos en el ciclo de vida de cualquier aplicación. Ambas arquitecturas presentan desafíos y consideraciones únicas que deben abordarse para garantizar un funcionamiento eficiente y confiable, encontrándose ya en la etapa de producción.

En el caso de las arquitecturas monolíticas, el despliegue suele ser más sencillo debido a la naturaleza centralizada de la aplicación. La implementación se realiza en un único servidor, lo que simplifica la gestión operativa y reduce la complejidad del proceso de despliegue. Sin embargo, esta simplicidad puede convertirse en una limitación a largo plazo, especialmente para aplicaciones que experimentan un crecimiento significativo en términos de usuarios y funcionalidades.[3]

Uno de los principales desafíos del mantenimiento a largo plazo en arquitecturas monolíticas es la necesidad de implementar y desplegar la aplicación completa en cada actualización.[5] Esto puede resultar en tiempos de desarrollo más largos y un ciclo de lanzamiento más lento, lo que dificulta la entrega rápida de nuevas características y actualizaciones. La escalabilidad puede ser un problema también, ya que agregar recursos adicionales al servidor único puede no ser suficiente para manejar un aumento en la carga de trabajo, lo que puede afectar el rendimiento y la experiencia del usuario.

Otro aspecto a considerar en el mantenimiento a largo plazo de arquitecturas monolíticas es la modularidad y la flexibilidad del código. Con el tiempo, es posible que la aplicación se vuelva más compleja y difícil de mantener, especialmente si no se han seguido prácticas de desarrollo de software sólidas. Los cambios en una parte del código pueden tener efectos no deseados en otras áreas de la aplicación, lo que dificulta la identificación y corrección de errores.[12]

En contraste, las arquitecturas de microservicios ofrecen una mayor flexibilidad y escalabilidad en el despliegue y mantenimiento a largo plazo. La modularidad inherente a los microservicios permite que los equipos de desarrollo implementen, actualicen y desplieguen servicios individuales, lo que facilita la introducción de nuevas características y la corrección de errores sin afectar el funcionamiento de otros componentes del sistema.[4] Esta capacidad de despliegue independiente no solo agiliza el proceso de desarrollo, sino que también reduce el riesgo de interrupciones en el servicio, ya que los cambios pueden implementarse de manera incremental y sin afectar la totalidad del sistema.

La escalabilidad horizontal proporcionada por los microservicios permite una distribución más eficiente de la carga de trabajo, ya que los servicios pueden escalarse individualmente según sea necesario para satisfacer la demanda del usuario.[4] Esto se traduce en una mayor capacidad para gestionar picos de tráfico repentinos y una mejor adaptación a las fluctuaciones en la demanda, lo que resulta en una experiencia del usuario más consistente.

No obstante, la naturaleza distribuida de los microservicios significa que hay más componentes que deben ser monitoreados y mantenidos. Esto requiere una planificación y la implementación de herramientas adecuadas para el monitoreo del rendimiento, la detección de errores y la gestión de la configuración.[7] A su vez, es importante la gestión de la consistencia de los datos. Dado que cada microservicio tiene su propia base de datos, es necesario implementar estrategias de sincronización y replicación de datos para garantizar la coherencia entre los diferentes servicios.

3 Conclusiones

La investigación detallada sobre las arquitecturas monolíticas y de microservicios ha revelado una variedad de consideraciones cruciales para el diseño, desarrollo, implementación y mantenimiento de sistemas de software. A lo largo de este estudio, hemos explorado sus características, implementación práctica, casos de estudio y despliegue y mantenimiento a largo plazo.

Las arquitecturas monolíticas, con su enfoque tradicional de integrar todas las funciones y servicios en un único paquete, ofrecen simplicidad y cohesión inicial. Son ideales para proyectos más pequeños o con requisitos estables, donde la gestión y el despliegue simplificados son prioritarios. Sin embargo, a medida que el proyecto crece en tamaño y complejidad, las limitaciones de las arquitecturas monolíticas pueden volverse más evidentes, especialmente en términos de escalabilidad y mantenimiento a largo plazo.

Por otro lado, las arquitecturas de microservicios ofrecen una mayor modularidad y flexibilidad, lo que facilita el desarrollo, implementación y mantenimiento de aplicaciones complejas. Permiten a los equipos de desarrollo trabajar de manera más independiente y ágil, introduciendo nuevas características y correcciones de errores de forma incremental; La capacidad de escalar horizontalmente servicios individuales según sea necesario brinda, a su vez, una mayor capacidad de adaptación a las demandas del mercado y del negocio.

Para finalizar esta guía práctica para la toma de decisiones, enumeraremos algunas consideraciones importantes a tener en cuenta al elegir entre una arquitectura u otra, las cuales han surgido como conclusión de nuestra investigación:

Tabla 2. Cuadro comparativo de arquitecturas monolíticas y de microservicios.

Criterio	**Arquitectura Monolítica**	**Arquitectura Microservicios**
Tamaño y Complejidad del Proyecto	Adecuada para proyectos más pequeños o con requisitos estables y bien definidos debido a su simplicidad y cohesión.	Mejor para proyectos grandes y complejos, ofrece mayor modularidad y escalabilidad
Escalabilidad	Puede enfrentar desafíos al escalar, especialmente en funciones específicas del sistema.	Adecuada para escalar horizontalmente servicios individuales según sea necesario, permite adaptación ágil.
Adaptabilidad y Flexibilidad	Puede enfrentar dificultades para adaptarse a cambios debido a su acoplamiento estrecho y estructura centralizada.	Ofrece mayor independencia y facilita la introducción de nuevas características, aunque aumenta la complejidad operativa.
Experiencia y Recursos del Equipo	Más adecuada para equipos menos experimentados o con recursos limitados, implementación y mantenimiento más directos.	Requiere equipos de desarrollo experimentados y bien preparados, debido a la mayor complejidad de planificación y gestión.

En última instancia, la elección entre arquitecturas monolíticas y de microservicios depende de las necesidades específicas de cada proyecto y de la capacidad del equipo de desarrollo para abordar cada enfoque. Es fundamental tomar decisiones informadas que aprovechen al máximo las fortalezas de cada arquitectura, considerando aspectos prácticos como el tamaño del proyecto, la complejidad de los requisitos, las capacidades del equipo de desarrollo y las expectativas de escalabilidad y mantenimiento a largo plazo.

A medida que el desarrollo de software continúa evolucionando en un entorno empresarial cada vez más dinámico, comprender las implicaciones de estas decisiones arquitectónicas es esencial para garantizar el éxito de los proyectos de software en el futuro.

Referencias

1. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. 3rd edn. Addison-Wesley Professional, United States of America (2012).
2. K. Gos and W. Zabierowski, "The Comparison of Microservice and Monolithic Architecture," 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH), Lviv, Ukraine, 2020, pp. 150-153
3. Ford, N., Richards, M., Sadalage, P., Dehghani, Z.: Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures. O'Reilly Media, United States of America (2021)
4. Newman, S.: Monolith to Microservices. O'Reilly Media, Inc., United States of America (2020).
5. Richardson, C.: Microservices Patterns: With examples in Java. Manning Publications, United States of America (2018).
6. M. Mosleh, K. Dalili and B. Heydari, "Distributed or Monolithic? A Computational Architecture Decision Framework," in IEEE Systems Journal, vol. 12, no. 1, pp. 125-136, 2018
7. Taibi, D., Lenarduzzi, V., Pahl, C., & Janes, A. (2017, May). Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In Proceedings of the XP2017 Scientific Workshops (pp. 1-5).
8. Martha, V. S., & Lenglart, M. Webservices engineering. Webservices: Theory and Practice, (2019)., pp. 173-196.
9. P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead," in IEEE Software, vol. 35, no. 3, pp. 24-35, 2018.
10. J. Jung, "Randy Shoup on Evolving Architecture and Organization at eBay," in IEEE Software, vol. 40, no. 1, pp. 98-100, 2023
11. Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%. Prime Video Tech, 2023 https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90.
12. Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. IEEE Access, 10, 20357-20374.