Universidad CENFOTEC

Maestría en Ingeniera del software con énfasis en arquitectura y diseño de software

Documento final de Proyecto de Investigación Aplicada 1

Tema:

Marco de migración de aplicaciones monolíticas hacia arquitecturas *serverless* en la nube

Elaborado por:

Orozco Montenegro Roberto Antonio

Diciembre, 2025

Declaratoria de derechos de autor

Dedicatoria

Tabla de Contenidos

Contenido

Resumen Ejecutivo

Las organizaciones que mantienen sistemas monolíticos enfrentan hoy un reto común: la dificultad para escalar, adaptarse y operar con eficiencia en entornos digitales que cambian rápido. Este proyecto estudia cómo esos sistemas pueden modernizarse mediante una migración directa hacia arquitecturas serverless, analizando la evidencia disponible sobre rendimiento, costos, mantenibilidad y riesgos. A partir de una revisión sistemática de estudios académicos y casos reales, se identifican las ventajas y limitaciones de ambos enfoques y se describen los principales factores que influyen en una transición exitosa.

La investigación se desarrolla con un enfoque cualitativo y se centra en comprender cómo serverless responde a problemas típicos del monolito, como el acoplamiento extremo, los despliegues lentos y la gestión constante de infraestructura. También se analizan patrones como Strangler Fig y conceptos de Domain-Driven Design, que permiten extraer funcionalidades de forma gradual y con bajo riesgo. Los hallazgos muestran que serverless sobresale en escenarios con demanda variable y que su adopción reduce significativamente la deuda técnica y la carga operativa.

El resultado del estudio es un marco conceptual de migración que integra criterios técnicos, económicos y arquitectónicos para evaluar la viabilidad de transformar un monolito en una solución serverless.

Frases clave: Arquitectura monolítica, Arquitectura *serverless*, Migración de software, *Strangler Fig, Domain-Driven Design* (*DDD*), Funciones como Servicio (FaaS), Modernización de aplicaciones legadas, Atributos de calidad del software, Escalabilidad, Deuda técnica, NoOps.

Capítulo 1. Introducción

1.1 Generalidades

En los últimos años, la computación en la nube ha dado lugar a nuevos modelos arquitectónicos que prometen mayor escalabilidad, agilidad y eficiencia en el desarrollo de software. Uno de estos modelos emergentes es la arquitectura *serverless*, en la cual los desarrolladores despliegan funciones en la nube sin preocuparse por la administración de servidores, pagando únicamente por el consumo real de recursos (Gaurav, s.f.). Este paradigma ha cobrado auge por abstraer la infraestructura subyacente, permitiendo enfocarse en la lógica de negocio. En contraste, muchas organizaciones tradicionales todavía operan con arquitecturas monolíticas, es decir, aplicaciones construidas como un solo bloque de software donde todos los módulos y funcionalidades se integran en un único código desplegable.

Las aplicaciones monolíticas fueron durante mucho tiempo la norma debido a su simplicidad inicial de desarrollo y despliegue; sin embargo, a medida que crecen en tamaño y complejidad, suelen presentar problemas de escalabilidad y mantenibilidad que dificultan responder ágilmente a los cambios del negocio (Gaurav, s.f.).

En este contexto general, se percibe la necesidad de contar con marcos de migración que orienten la transformación de aplicaciones legadas en infraestructuras monolíticas hacia arquitecturas más modernas, como microservicios o *serverless*, aprovechando las ventajas de la nube.

En particular, la migración a *serverless* se plantea como una estrategia prometedora para sistemas con cargas variables, necesidad de rápida escalabilidad y optimización de costos de infraestructura (Gaurav, s.f.). No obstante, dicha migración conlleva retos significativos en términos de rediseño arquitectónico, manejo de estado, orquestación de servicios y cambio de la cultura organizacional.

1.2 Antecedentes del problema

La dependencia de arquitecturas monolíticas se ha convertido en un desafío crítico para las organizaciones que buscan modernizarse. Estos sistemas, aunque funcionales, presentan una serie de debilidades significativas a medida que el negocio crece y las demandas cambian. Una de las limitaciones más evidentes es la dificultad para escalar de manera selectiva. Para aumentar la capacidad de un componente específico, como un módulo de procesamiento de pedidos, es necesario escalar la aplicación completa, lo que resulta en un uso ineficiente de los recursos y un aumento innecesario de los costos.

Además, la estrecha interdependencia entre los componentes del monolito crea serios cuellos de botella en el proceso de desarrollo y despliegue de cualquier software. Cualquier cambio, por más pequeño que sea, requiere volver a desplegar toda la aplicación. Esto prolonga los ciclos de despliegue y aumenta el riesgo de fallos, ya que un error en una parte del código puede tener un efecto dominó y derribar todo el sistema (Bustos, n.d.). La complejidad inherente de los monolitos grandes también impone una pesada carga cognitiva a los equipos, dificultando la incorporación de nuevos desarrolladores y la adopción de prácticas modernas de desarrollo como DevOps.

El dilema de la modernización es una encrucijada estratégica que muchas empresas enfrentan hoy en día. Continuar invirtiendo en el mantenimiento de un monolito frágil y costoso, o asumir los riesgos y la complejidad de una migración hacia una arquitectura más ágil y escalable. Para ilustrar esta problemática, se pueden citar casos como el de Netflix, que en 2009 enfrentó problemas de crecimiento con su infraestructura monolítica, lo que lo llevó a migrar a microservicios para soportar el ritmo acelerado de la demanda. (Bustos, n.d.). Otro estudio es el caso de la migración de un sistema *FinTech* de procesamiento de documentos (Alireza Goli O. H., 2020), estos casos demuestran que este problema es un desafío generalizado en la industria, no limitado a los gigantes tecnológicos. Estos ejemplos recalcan la necesidad de una metodología clara y segura para abordar la re-arquitectura, una tarea que va más allá de un simple cambio de código y que implica una transformación completa de la cultura, la gestión y la tecnología de una organización.

1.3 Definición y descripción del problema

El problema que promueve resolver esta investigación se debe a que no existe un marco sistemático que oriente la migración de aplicaciones monolíticas hacia arquitecturas *serverless* en la nube, abordando de manera integral los desafíos técnicos y organizacionales que conlleva este proceso.

En la práctica, muchas organizaciones aún dependen de aplicaciones monolíticas que presentan limitaciones importantes:

- Escalabilidad restringida: para soportar picos de demanda (por ejemplo, compras en línea en fechas especiales, o pagos de impuestos en cierto periodo del año) se debe escalar la aplicación completa, incluso en módulos que no lo requieren, generando un uso ineficiente de recursos.
- Baja resiliencia: un error en un módulo puede comprometer el sistema entero.
- Despliegue lento y riesgoso: incluso una actualización menor obliga a reconstruir y desplegar todo el sistema, aumentando la posibilidad de fallas y tiempos de inactividad.
- Altos costos operativos y mala experiencia de usuario: la rigidez de la arquitectura impide adaptarse rápidamente a cambios en el negocio.

Frente a estas limitaciones, la arquitectura *serverless* aparece como una alternativa viable: permite dividir la lógica en funciones pequeñas, independientes, que se ejecutan bajo demanda, escalan automáticamente y se facturan solo por uso.

No obstante, el proceso de migración está lejos de ser trivial y plantea retos específicos:

- Identificación de límites funcionales: definir *bounded contexts* y fragmentar la lógica del monolito en unidades coherentes sin dependencias circulares.
- Gestión del estado y la base de datos: externalizar sesiones y datos compartidos, ya que las funciones son *stateless* y efímeras.
- Comunicación y orquestación: coordinar flujos de negocio mediante eventos, colas o servicios como *Step Functions*, adaptándose al modelo *event-driven*.

- Rendimiento: mitigar problemas de latencia por *cold starts* o sobrecarga en funciones altamente interactivas.
- Seguridad y control: redefinir roles, autenticación y autorizaciones, además de considerar el riesgo de dependencia de un proveedor específico.
- Aspectos organizacionales: el equipo debe adquirir nuevas competencias (infraestructura como código, monitoreo distribuido, prácticas DevOps adaptadas).

En síntesis, el problema central no es únicamente cómo migrar técnicamente un monolito a *serverless*, sino la falta de un marco integral que abarque:

- Evaluación de factibilidad.
- Planificación de la migración (módulos prioritarios, estrategia incremental).
- Aplicación de patrones adecuados (estrangulador, refactorizaciones).
- Verificación post-migración (comparación de rendimiento, costos y cumplimiento de requisitos no funcionales frente al monolito original).

La ausencia de un marco formal provoca que muchas migraciones se realicen de forma artesanal/empírica, con riesgos de errores, sobrecostos y resultados insatisfactorios. Este proyecto busca cerrar esa brecha mediante la definición y validación de un "Marco de migración monolito a *serverless*" que sirva de guía práctica para arquitectos y desarrolladores en procesos de modernización.

1.4 Justificación

El presente estudio adopta un enfoque innovador al proponer una migración directa desde arquitecturas monolíticas hacia entornos serverless, omitiendo la fase intermedia basada en microservicios. Esta decisión se sustenta en evidencia empírica reciente que demuestra que, en muchos contextos, que el uso de microservicios introduce complejidad operativa innecesaria, mayores costos de orquestación y una deuda técnica adicional difícil de gestionar. En contraste, las arquitecturas serverless ofrecen una adopción acelerada del modelo NoOps, al eliminar la administración de

infraestructura, escalar automáticamente por demanda y pagar únicamente por lo que se usa.

Migrar directamente a serverless permite reducir la dependencia de equipos especializados en DevOps o administración de contenedores, lo que se traduce en ahorros significativos en costos operativos y tiempo de despliegue. Además, este enfoque disminuye la deuda técnica heredada del monolito, ya que las funciones serverless son más fáciles de aislar, refactorizar y probar de forma incremental. Este tipo de transición resulta particularmente ventajoso para pequeñas y medianas empresas, que pueden acceder a tecnologías *cloud* sin necesidad de contratar personal adicional ni realizar grandes inversiones en capacitación de Ops.

Además, este trabajo pretende proponer un marco de migración que aborde los principales retos técnicos y organizacionales asociados al cambio hacia *serverless*. Se espera que dicho marco sirva como referencia para:

- Optimizar el uso de recursos mediante escalabilidad automática.
- Incrementar la disponibilidad y resiliencia de los sistemas.
- Reducir costos al adoptar un modelo de pago por ejecución.
- Acelerar los ciclos de despliegue de nuevas funcionalidades.
- Determinar en qué escenarios la migración no es viable o puede generar más problemas que beneficios, evitando decisiones costosas o contraproducentes.

La elección de una transición directa del monolito hacia serverless, omitiendo la fase intermedia de microservicios, responde a principios arquitectónicos documentados en estudios empíricos recientes. Serverless habilita un modelo *NoOps*, en el cual la organización no gestiona servidores ni orquestación compleja, reduciendo significativamente las tareas de DevOps y la carga operativa. Este enfoque disminuye la deuda técnica heredada del monolito al promover componentes más pequeños, aislados y fáciles de probar. Además, la adopción de funciones serverless permite que pymes y equipos con recursos limitados adopten tecnologías *cloud* avanzadas sin requerir personal especializado en contenedores, *Kubernetes* o mallas de servicio. Estos beneficios estratégicos hacen que la ruta directa monolito a serverless sea una alternativa más rápida, económica y sostenible, especialmente en organizaciones con menor madurez operativa

1.5 Viabilidad: Técnica, Operativa y Económica

La viabilidad del proyecto ha sido analizada desde tres aristas fundamentales:

1.5.1 Viabilidad Técnica

El entorno tecnológico actual proporciona todas las herramientas necesarias, así como documentación disponible, para efectuar la migración propuesta. Los principales proveedores de nube pública como: *AWS*, *Microsoft Azure* y *Google Cloud* cuentan con plataformas maduras de *serverless* como *AWS Lambda*, *Azure Functions*, *Google Cloud Functions* respectivamente, y una amplia gama de servicios complementarios como pasarelas de *API*, orquestadores de funciones, bases de datos *serverless*, almacenamiento de estados, monitoreo distribuido, etc. Esto asegura que es técnicamente posible implementar una arquitectura *serverless* robusta que reemplace gradualmente las funcionalidades del sistema monolítico.

1.5.2 Viabilidad Operativa

El enfoque metodológico propuesto, se centra en la creación de un "artefacto" conceptual (un marco de migración) en lugar de una implementación completa en un entorno de producción real. Esto permite un progreso significativo de la investigación sin requerir la disrupción de sistemas críticos de negocio. La aplicación práctica y la validación empírica en un entorno de producción se abordan como la propuesta de esta investigación, a manera de prueba de concepto.

1.5.3 Viabilidad económica

Los costos directos del proyecto se consideran mínimos. El principal costo es el tiempo y el esfuerzo del investigador para la recopilación y el análisis de información. Los recursos de infraestructura en la nube, si fueran necesarios para alguna prueba conceptual, se pueden obtener a través de las "capas gratuitas" (*free tiers*) que ofrecen

proveedores como AWS. (AWS, s.f.) Por lo tanto, el proyecto es económicamente viable sin necesidad de una inversión externa significativa.

1.6 Objetivos

Para el desarrollo de los objetivos se utiliza la taxonomía de Bloom revisada de 2001.

1.6.1 Objetivo general

Proponer un marco de migración de aplicaciones monolíticas hacia arquitecturas *serverless* en la nube, que permita a los arquitectos de software tomar decisiones fundamentadas y evaluar los beneficios en términos de atributos de calidad del software.

1.6.2 Objetivos específicos

- Analizar y comparar las características, ventajas y desventajas de las arquitecturas monolíticas y *serverless*, identificando sus debilidades y fortalezas clave a través de una revisión sistemática de la literatura.
- Evaluar las estrategias de migración existentes y los patrones de diseño como el Patrón *Strangler Fig*, para determinar su aplicabilidad en la transición de monolitos a *serverless*.
- Definir un conjunto de métricas e instrumentos que permitan medir objetivamente el impacto de la migración en atributos de calidad del software como el rendimiento, la escalabilidad, la mantenibilidad y el costo.
- Diseñar un marco de decisión integral que combine los patrones de migración y las métricas de evaluación para guiar el proceso de re-arquitectura, sentando las bases para su validación práctica.

1.7 Alcances y Limitaciones

1.7.1 **Alcance**

El alcance de este proyecto se centra en el diseño formal y documentado de un Marco de Migración de arquitecturas monolíticas a *Serverless* (FaaS), proporcionando una guía para líderes técnicos y arquitectos. La principal característica de este marco es el enfoque en la transición directa de un Monolítico al paradigma *Serverless*, omitiendo la arquitectura de Microservicios basada en contenedores como fase obligatoria, con el objetivo de capitalizar los beneficios de *NoOps* y la eficiencia en costos de manera acelerada, logrando un ahorro considerable en operaciones (*DevOps*) y reduciendo la deuda técnica inherente a la gestión de infraestructura. Esta simplificación está particularmente diseñada para potenciar a las pequeñas y medianas empresa sin excluir las transnacionales a adoptar la nube moderna sin necesidad agregar personal de infraestructura.

El marco se distingue por estar anclado en la literatura empírica, utilizando criterios de decisión en torno al Costo-Eficiencia, la mitigación de Latencia (*Cold Start*) y los factores Socio-Técnicos de la organización.

Su validación será conceptual y empírica, contrastando sus fases y criterios con casos de estudio reales identificados en la investigación académica, garantizando su pertinencia práctica.

Además, aunque el estudio contempla aplicaciones monolíticas, quedan excluidos sistemas que presentan restricciones incompatibles con el modelo FaaS, como aplicaciones embebidas, sistemas de tiempo real estricto, procesamiento científico de larga duración o cargas que requieren GPU persistente. Estos escenarios dependen de ciclos de ejecución continuos, estados prolongados en memoria o dispositivos especializados que no pueden mapearse eficazmente a funciones efímeras y sin estado propias de serverless.

1.7.2 **Limitaciones**

El estudio presenta ciertas limitaciones derivadas de su alcance y propósito. En primer lugar, la migración analizada es exclusivamente de monolitos a serverless, excluyendo

arquitecturas intermedias como microservicios, contenedores gestionados o modelos híbridos. Asimismo, se considera la viabilidad técnica de aplicaciones monolíticas de escritorio, pero se excluyen otros tipos de sistemas cuyos modelos de despliegue y dependencias difieren del paradigma serverless.

El análisis se basa en evidencia documental y estudios empíricos previamente publicados, por lo que no se realizarán pruebas experimentales ni mediciones de rendimiento en entornos de producción. Además, no se abordarán en profundidad aspectos de gestión organizacional, cambio cultural o adopción de DevOps, aunque se reconocen como factores complementarios de éxito en procesos de modernización tecnológica.

1.8 Marco de referencia organizacional y socioeconómico

La modernización de aplicaciones monolíticas hacia arquitecturas *serverless* representa un reto estratégico para un amplio conjunto de organizaciones que dependen de software crítico para su operación.

En primer lugar, el marco metodológico propuesto es especialmente útil para empresas de tecnología de la información (TI), proveedores de servicios en la nube, entidades financieras, comercios electrónicos, instituciones de salud y administraciones públicas entre muchas otras que aún operan con aplicaciones monolíticas. Estas organizaciones suelen enfrentar altos costos de mantenimiento, baja flexibilidad ante nuevas demandas del mercado y dificultades para escalar aplicaciones en momentos críticos de uso. La posibilidad de contar con lineamientos claros para evaluar y ejecutar una migración a *serverless* significa reducir riesgos, optimizar costos y aumentar la capacidad y el poder de innovación.

Desde la perspectiva socioeconómica, la adopción de un marco de migración tiene efectos más amplios:

- Permite a pequeñas y medianas empresas (pymes) acceder a tecnologías modernas sin necesidad de mantener infraestructura propia, pagando únicamente por uso real de recursos.

- Contribuye a la competitividad de las empresas locales, ya que posibilita procesos de digitalización más ágiles y sostenibles, permitiendo a las pymes tener acceso a tecnologías modernas.

En este contexto, el marco metodológico propuesto no solo busca servir como guía técnica, sino también como herramienta de toma de decisiones estratégicas que ayude a las organizaciones a determinar en qué escenarios la migración es conveniente y en cuáles no lo es, evitando inversiones riesgosas o desalineadas con los objetivos de negocio.

1.9 Revisión de la literatura

1.9.1 **Revisión sistemática**

Este proyecto realiza una revisión sistemática de la literatura enfocada en la migración de aplicaciones monolíticas hacia arquitecturas *serverless* en la nube, con el fin de identificar enfoques, patrones y métricas documentadas en la última década. Para ello, se consultaron bases académicas como *IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect* y *Google Scholar*, aplicando términos clave relacionados con modernización de software y patrones de migración como *monolithic applications, serverless migration, legacy modernization* y *cloud-native patterns*.
El proceso de selección, basado en criterios claros de inclusión y exclusión, permitió reunir estudios empíricos, revisiones y casos prácticos que aportan una visión integral sobre los beneficios, retos y limitaciones de este tipo de transformaciones arquitectónicas.

1.9.1.1. Problema

El auge de la computación en la nube ha impulsado la adopción de arquitecturas modernas como *serverless*, estas nuevas arquitecturas prometen escalabilidad automática, reducción de costos y mayor agilidad en el despliegue de aplicaciones. Sin embargo, muchas organizaciones aún dependen de sistemas monolíticos que, si bien son funcionales, presentan limitaciones de escalabilidad, resiliencia y mantenibilidad. Actualmente no existe un marco metodológico consolidado que guíe de manera integral la migración de aplicaciones monolíticas hacia arquitecturas *serverless*, considerando tanto los beneficios potenciales como los escenarios en los que dicha migración puede resultar poco viable o generar más problemas que ventajas.

1.9.1.2. Formulación de la pregunta

La formulación de la pregunta de investigación ayuda a delimitar el alcance de la búsqueda y a enfocar la información que será analizada para lograr los objetivos del estudio. Para esta investigación, la pregunta central que se busca responder es:

¿Cuáles son las estrategias, desafíos, patrones de diseño y métricas de evaluación para la migración de aplicaciones monolíticas a arquitecturas serverless, y cómo pueden integrarse en un marco metodológico coherente?

1.9.1.3. Palabras claves

A continuación, se listan las palabras que se utilizan para conducir la búsqueda e identificación de literatura relacionada con este trabajo. Dado que la mayoría de los trabajos al respecto están en el idioma inglés se incluyen todas en este idioma con traducciones en español donde sea necesario.

Ilustración 1: Palabras clave.

Inglés	Español
Monolithic applications	Aplicaciones monolíticas
Legacy modernization	Modernización de sistemas heredados
Cloud migration	Migración a la nube
Serverless	Sin servidor
Serverless architecture	Arquitectura sin servidor
Serverless frameworks	
Strangler pattern	
Cloud transformation	Transformación a la nube
Function as a service (FaaS)	Función como servicio

Fuente: Elaboración propia.

1.9.1.4. Intervención

La intervención en esta investigación es analizar los antecedentes existentes sobre migración de sistemas monolíticos hacia arquitecturas *serverless*, buscando patrones de diseño, métricas de evaluación y trabajos previos documentados. A partir de estos documentos, se pretende compilar las técnicas, métodos y lineamientos que permitan crear un marco de migración de aplicaciones monolíticas, considerando tanto los beneficios potenciales como los riesgos de migraciones no viables.

1.9.1.5. Control

Al iniciar la investigación no se cuenta con un marco de migración definido ni se ha encontrado alguno similar como base. El proceso comienza con una búsqueda sistemática de información a partir de las palabras clave seleccionadas y la aplicación

de criterios de inclusión y exclusión. Esto asegura que el material recopilado sea pertinente, actualizado y de calidad académica, constituyendo la base para el análisis y la propuesta de este trabajo.

1.9.1.6. Efectos

Se espera recopilar una lista de antecedentes y estudios de caso relacionados con migraciones monolíticas a *serverless*. Esperando que estos hallazgos proporcionen contexto, evidencias empíricas y buenas prácticas que sirvan como guía para el marco de migración propuesto, además de contribuir a identificar escenarios donde la migración puede ser contraproducente o poco eficiente para las industrias.

1.9.1.7. Medida de salida

Se establece en función de la calidad y relevancia de los estudios seleccionados. Cada documento ya sean revistas indexadas, conferencias académicas, libros, papers entre otras son evaluados de acuerdo con su fuente. De esta manera se busca garantizar que las conclusiones derivadas se basen en información confiable y aplicable al objeto de investigación.

1.9.1.8. Aplicación

Se busca que este estudio resulte de utilidad para arquitectos de software, ingenieros de sistemas y profesionales de TI que participan en procesos de modernización tecnológica. La propuesta generada busca orientar la toma de decisiones sobre migraciones monolíticas hacia *serverless*, ofreciendo criterios claros no solo para implementar la transición, sino también para determinar cuándo no conviene llevarla a cabo.

1.9.1.9. Diseño experimental

Se realiza una clasificación y análisis de los estudios recopilados, priorizando aquellos con mayor relevancia y rigurosidad metodológica además de prestigio de donde se publicó. Esto permite establecer un conjunto de parámetros que fundamenten la construcción del marco de migración. El proceso garantiza un balance adecuado: suficiente documentación para obtener resultados representativos, pero sin dispersarse en un rango excesivamente amplio que reste claridad a la investigación.

1.9.1.10. **Selección** de fuentes

En esta sección se identifican las fuentes de estudios primarios que se utilizan para esta investigación

1.9.1.11. Criterio de selección de fuentes

Se toman en cuenta publicaciones y papers indexados en el motor de búsqueda Google Scholar, utilizando como métrica la calificación de las revistas (donde aplique) en el ranking de Scimagojr, así como la posición de la Universidad en el ranking global de Universidades Top Universities.

1.9.1.12. Lenguaje de estudio

Se utilizará principalmente el idioma inglés para las búsquedas de literatura dado que en este se encuentra la mayoría de las publicaciones al respecto, el idioma español para búsquedas y para la formulación de este estudio.

1.9.1.13. Cadena de búsqueda

Se optó por crear cadenas de búsqueda utilizando el operador lógico AND en algunos escenarios, con el objetivo de abarcar la mayor cantidad posible de documentación

relacionada con el tema. A partir de esta combinación de términos se generaron las siguientes cadenas de búsqueda*: “intitle: "monolithic" AND intitle: "migration" AND intitle: "framework" AND intitle: "serverless"”, “intitle: "framework" AND intitle: "serverless"”, “migrating monolithic to microservices”, “Serverless + Architectural”, “Strangler Fig Pattern”, “Serverless Architectures”, “Modernization of Legacy Applications”, “Modernization of Legacy Applications”, “Monoliths to Serverless”*.

1.9.1.14. Selección de estudios

Para verificar si los artículos encontrados en las búsquedas eran útiles para los objetivos de este trabajo, se aplicó la técnica de *skimming*, que permite obtener de manera rápida una idea general del contenido. Asimismo, se revisó el resumen de cada artículo con el fin de comprender mejor su alcance.

1.9.1.14.1. Definición del criterio de inclusión y exclusión

Se utilizan los criterios o palabras claves detallados en la tabla 1, para incluir o excluir un documento. Los artículos que cumplan los requisitos son candidatos a ser incluidos.

Ilustración 2: Criterios de inclusión y exclusión.

Pregunta de investigación	**Término principal para criterio de inclusión**	**Criterio de exclusión**
¿Cuáles son las estrategias, desafíos, patrones de diseño y métricas de evaluación para la migración de aplicaciones monolíticas a arquitecturas	*"Monolithic applications", "Legacy modernization", "Cloud migration", "Serverless", "Serverless architecture", "Serverless frameworks ", "Strangler pattern", "Cloud*	• La publicación o revista tiene una calificación menor a 15 en el scimagojr ranking, o la Universidad a la que corresponden los autores del *paper* se ubique dentro del Top 1000 de

serverless, y cómo pueden integrarse en un marco metodológico coherente?	*transformation", "Function as a service (FaaS)"*	universidades en Top Univerisites.

Fuente: Elaboración propia.

1.9.1.14.2. Procedimiento para la selección de los estudios

Este es el proceso iterativo para la selección de los estudios:

1. Realizar la búsqueda en Google Scholar de las cadenas de búsqueda mencionadas anteriormente, filtrando por estudios publicados “desde 2020”.
2. Evaluar rápidamente los resultados de la primera página por su título, descartando por los criterios de Tabla 2.
3. Si no existen suficientes resultados satisfactorios en el paso anterior, ampliar la búsqueda de paso 1 a “desde 2016”.
4. Evaluar los resultados obtenidos después de este descarte inicial en el paso 2, y aplicar los criterios de exclusión basados en el *Abstract* y *keywords* del artículo.
5. Seleccionar los resultados considerados relevantes para la fuente consultada y repetir el proceso con las demás fuentes disponibles.

1.9.1.14.3. Identificación de fuentes

Se lleva a cabo una selección de las fuentes más relevantes empleadas para la consulta de información vinculada a este proyecto. En dicha elección se considera la popularidad de la fuente, su credibilidad y el reconocimiento que posee dentro de la comunidad académica, así como la diversidad de documentación disponible y la facilidad de acceso, o bien, la posibilidad de contar con las credenciales necesarias para consultar el contenido requerido. La búsqueda se efectúa mediante el uso de palabras clave y temas afines, tanto en español como en inglés. Entre las fuentes

consultadas destacan las siguientes por su importancia en relación con el tema de este proyecto:

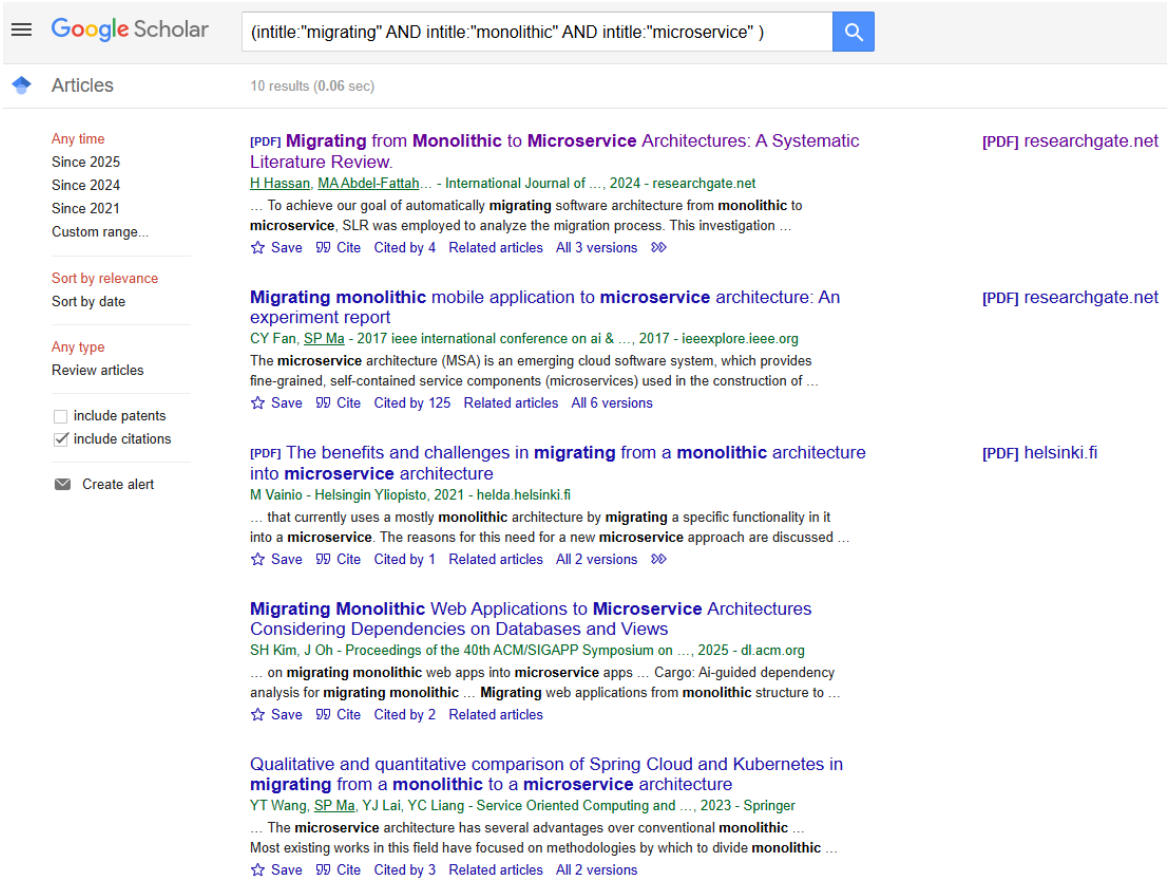
1.9.2 Ejecución de la revisión

Se presenta a continuación el proceso de selección llevado a cabo sobre la fuente seleccionada.

1.9.2.1 Ejecución de la selección de la fuente Google Scholar

Se procedió a realizar la consulta de la fuente basado en las cadenas de búsqueda planteadas en los puntos anteriores:

Ilustración 3: Ejecución de la selección de fuente.



Fuente: Captura tomada del sitio: https://scholar.google.com/

Se documenta el proceso de búsqueda en el sitio de Google Scholar, sin embargo, se realizaron búsquedas utilizando todas las cadenas de búsqueda.

Ilustración 4: Publicaciones seleccionadas de Google Scholar.

#	Título	Autores	URL	Año
1	*Migrating from Monolithic to Serverless: A FinTech Case Study*	Alireza Goli, Omid Hajihassani, Hamzeh Khazae, Omid Ardakanian, Moe Rashidi, Tyler Dauphinee	https://research.spec.org/icpe_proceedings/2020/companion/p20.pdf	2020
2	*The Journey to Serverless Migration: An Empirical Analysis of Intentions, Strategies, and Challenges*	Muhammad Hamza, Muhammad Azeem Akbar, Kari Smolander	https://arxiv.org/pdf/2311.13249	2023
3	*Navigating Decision-Making in Serverless Migration: A Socio-Technical Grounded Theory Approach*	Muhammad Hamza, Muhammad Azeem Akbar, Kari Smolander	https://papers.ssrn.com/sol3/Delivery.cfm/5161731.pdf?abstractid=5161731&mirid=1	2025

4	*Microservices vs Serverless: A Performance Comparison on a Cloud-native Web Application*	Chen-Fu Fan, Anshul Jindal, Michael Gerndt	https://www.scitepress.org/Papers/2020/97927/97927.pdf	2020
5	*Analysis of cost-efficiency of serverless approaches*	Nakhat Syeda, Harsh Shah, Rajvinder Singh, Suraj Jaju, Sumedha Kumar, Gourav Chhabra, Maria Spichkova	https://www.researchgate.net/publication/392514773_Analysis_of_cost-efficiency_of_serverless_approaches	2025
6	*Understanding Cost Dynamics of Serverless Computing*	Muhammad Hamza, Muhammad Azeem Akbar, Rafael Capilla	https://arxiv.org/pdf/2311.13242	2023
7	*Improving Application Migration to Serverless Computing Platforms: Latency Mitigation with Keep-Alive Workloads*	Minh Vu, Baojia Zhang, Olaf David, George Leavesley, Wes Lloyd	https://faculty.washington.edu/wlloyd/papers/wosc_camera_ready.pdf	2018
8	*Investigating differences in performance*	Jonathan Manousian	https://www.diva-portal.org/smash/record	2022

	between monolithic and serverless based architectures		.jsf?pid=diva2%3A1692921&dswid=4207	
9	*Cost optimization strategies in fintech using microservices and serverless architectures*	Sumit Bhatnagar	https://www.researchgate.net/profile/Sumit-Bhatnagar-7/publication/389818634_COST_OPTIMIZATION_STRATEGIES_IN_FINTECH_USING_MICROSERVICES_AND_SERVERLESS_ARCHITECTURES/links/67d32457be849d39d676e1b8/COST-OPTIMIZATION-STRATEGIES-IN-FINTECH-USING-MICROSERVICES-AND-SERVERLESS-ARCHITECTURES.pdf	2025

Fuente: Elaboración propia.

1.9.2.1.1 Evaluación estudio 1

Ilustración 5: Evaluación de fuentes encontradas en Google Scholar.

Fuente: Captura tomada del sitio web: https://www.topuniversities.com/

1.9.2.1.2 Evaluación estudio 2

Ilustración 6: Evaluación de fuentes encontradas en Google Scholar.

Fuente: Captura tomada del sitio web: https://www.topuniversities.com/

1.9.2.1.3 Evaluación estudio 3

Ilustración 7: Evaluación de fuentes encontradas en Google Scholar.

Fuente: Captura tomada del sitio web: https://www.topuniversities.com/

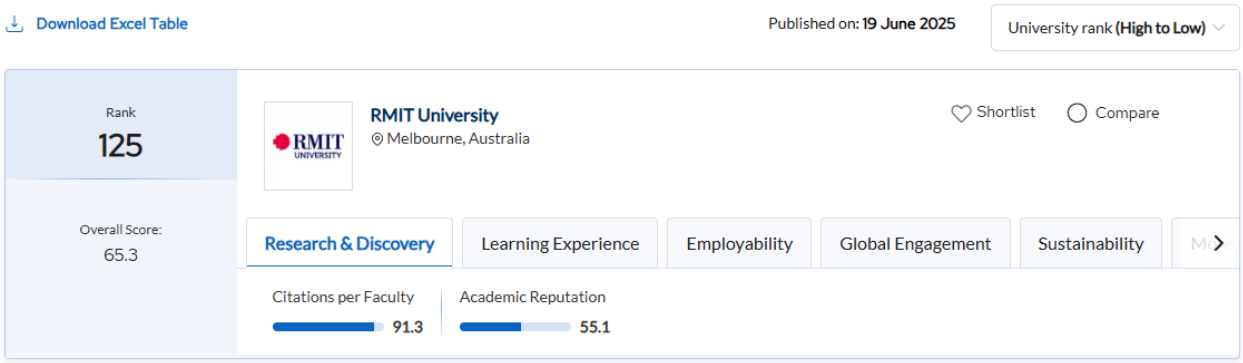
1.9.2.1.4 Evaluación estudio 4

Ilustración 8: Evaluación de fuentes encontradas en Google Scholar.

Fuente: Captura tomada del sitio web: https://www.scimagojr.com/

1.9.2.1.5 Evaluación estudio 5

Ilustración 9: Evaluación de fuentes encontradas en Google Scholar.



Fuente: Captura tomada del sitio web: https://www.topuniversities.com/

1.9.2.1.6 Evaluación estudio 6

Ilustración 10: Evaluación de fuentes encontradas en Google Scholar.

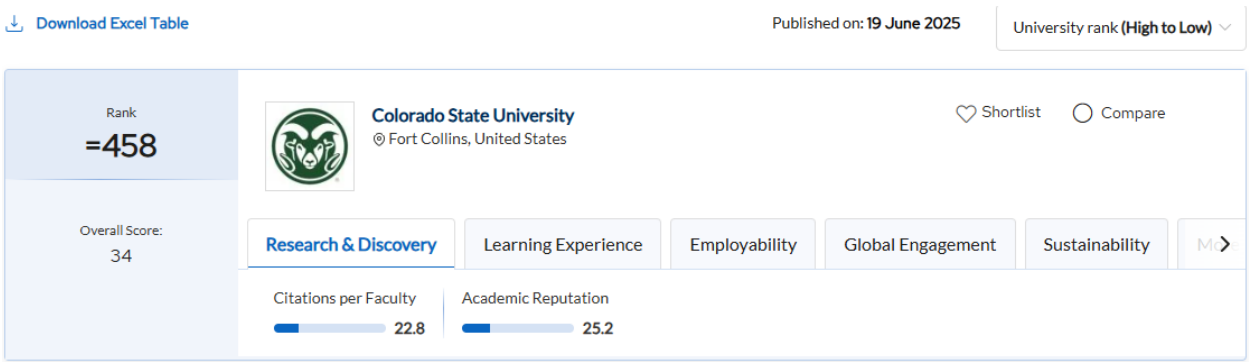
Fuente: Captura tomada del sitio web: https://www.topuniversities.com/

1.9.2.1.7 Evaluación estudio 7

Ilustración 11: Evaluación de fuentes encontradas en Google Scholar.

Fuente: Captura tomada del sitio web: https://www.topuniversities.com/

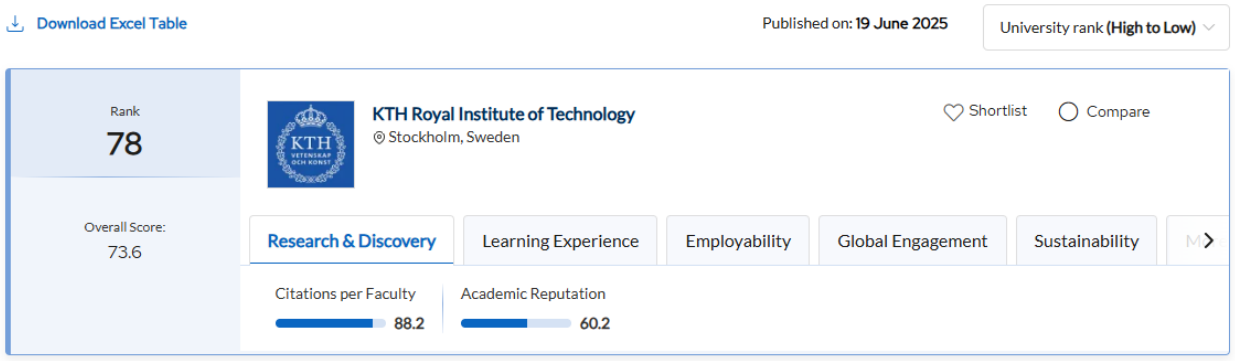
Ilustración 12: Evaluación de fuentes encontradas en Google Scholar.



Fuente: Captura tomada del sitio web: https://www.topuniversities.com/

1.9.2.1.8 Evaluación estudio 8

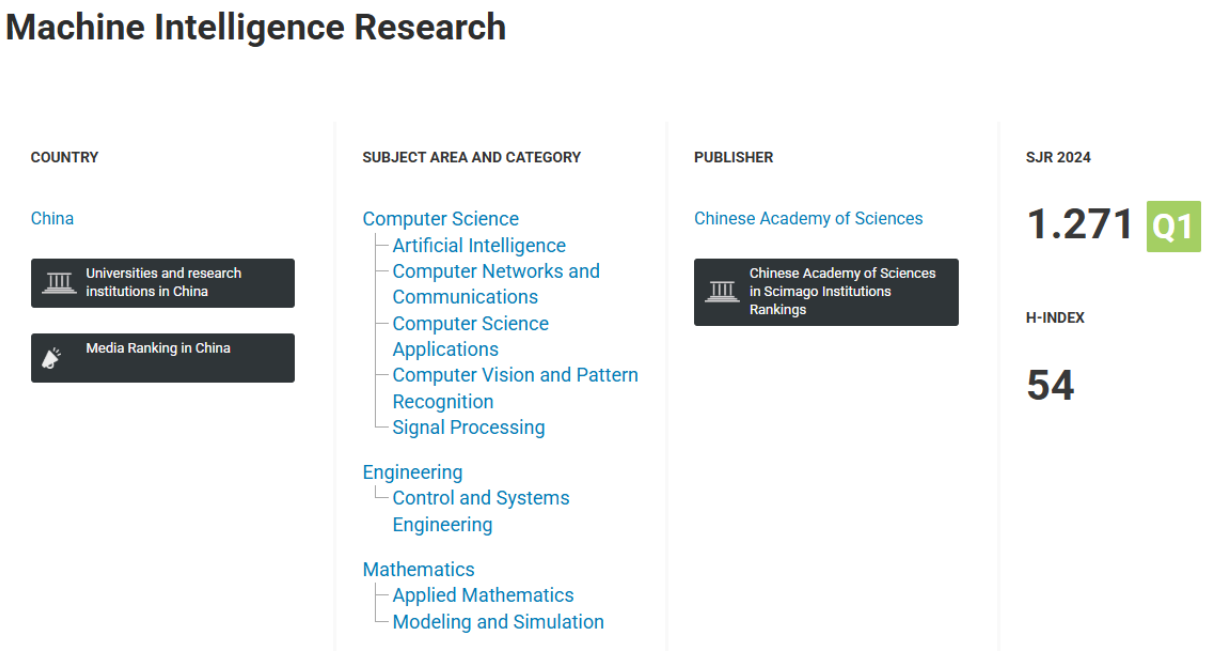
Ilustración 13: Evaluación de fuentes encontradas en Google Scholar.



Fuente: Captura tomada del sitio web: https://www.topuniversities.com/

1.9.2.1.9 Evaluación estudio 9

Ilustración 14: Evaluación de fuentes encontradas en Google Scholar.



Fuente: Captura tomada del sitio web: https://www.scimagojr.com/

1.9.2.2 Revisión de la selección

Se realizó una revisión inicial de los estudios seleccionados mediante el resumen o abstracto de cada documento y adicionalmente es procedió a realizar la técnica de skimming para completar la revisión de cada documento.

1.9.2.2.1 Extracción de la información

La información obtenida de los documentos debe incluir características y detalles que apoyen la toma de decisiones para asegurar que se cumplan con los objetivos establecidos.

Ilustración 15: Extracción de fuente 1.

Sección	Descripción
Titulo	Migrating from Monolithic to Serverless: A FinTech Case Study
Autor(es)	Alireza Goli, Omid Hajihassani, Hamzeh Khazae, Omid Ardakanian, Moe Rashidi, Tyler Dauphinee
Tema	Migración de un sistema *FinTech* monolítico a *serverless* y evaluación de métricas de rendimiento y costo.
Tesis	La migración a tecnologías *serverless* mejora el rendimiento y la elasticidad con incrementos mínimos de costo, justificando su adopción en dominios críticos como FinTech.
Propósito	El propósito de esta tesis es documentar el proceso de migración y cuantificar beneficios y limitaciones del cambio de arquitectura.
Ideas centrales	1) *Serverless* mejoró la elasticidad y el throughput de la aplicación comparado con la versión monolítica. 2) Costo ligeramente mayor pero controlado gracias a

	facturación bajo demanda (dependiendo del nivel de procesamiento del sistema). 3) Uso práctico del patrón Strangler Fig para transición segura de los componentes. 4) Desafíos en persistencia de estado y coordinación de bases de datos. 5) FinTech como dominio crítico confirma aplicabilidad ya que es un sistema demandante y necesita elasticidad en ciertos periodos del tiempo.
Conceptos claves	FaaS, cold start, elasticidad, Strangler Fig Pattern, observabilidad. cost
Conclusiones	El caso valida que *serverless* ofrece ventajas claras en entornos de alta demanda variable, pero también muestra sus límites. Aporta datos concretos para métricas y confirma la necesidad de un marco que incluya cuándo no migrar.

Fuente: Elaboración propia

Ilustración 16: Extracción de fuente 2.

Sección	Descripción
Titulo	The Journey to *Serverless* Migration: An Empirical Analysis of Intentions, Strategies, and Challenges
Autor(es)	Muhammad Hamza, Muhammad Azeem Akbar, Kari Smolander
Tema	Análisis empírico de motivaciones, estrategias y retos en migraciones *serverless*.
Tesis	Las migraciones se impulsan por escalabilidad y costos, pero requieren ajustes técnicos y culturales ya que implementan un nuevo paradigma de programación.
Propósito	Identificar lecciones aprendidas y patrones comunes de migración.

Ideas centrales	1) Domain Driven Design (DDD) y event-driven como enfoques frecuentes para realizar el proceso de migración, acompañado del patrón Strangler fig. 2) Obstáculos en pruebas y estandarización ya que al cambiar el paradigma se debe cambiar el enfoque en cómo se trabaja. 3) Cambios organizacionales y de mentalidad necesarios para completar el objetivo de las migraciones 4) La migración se debe manejar como un proceso incremental.
Conceptos claves	DDD, event sourcing, Strangler Fig, testing distribuido
Conclusiones	La migración *serverless* no es solo técnica, también organizacional. Requiere procesos iterativos y gobernanza de decisiones. Este tema es importante ya que aclara que el marco de migración debe ser integral y contemplar la parte organizacional ya que únicamente con la parte técnica no es suficiente.

Fuente: Elaboración propia

Ilustración 17: Extracción de fuente 3.

Sección	Descripción
Titulo	Navigating Decision-Making in *Serverless* Migration: A Socio-Technical Grounded Theory Approach
Autor(es)	Muhammad Hamza, Muhammad Azeem Akbar, Kari Smolander
Tema	Teoría fundamentada sobre puntos de decisión en migraciones *serverless*
Tesis	Las decisiones en migraciones son interdependientes y requieren un marco explícito para evitar errores estratégicos.
Propósito	Identificar y clasificar las decisiones críticas en migraciones *serverless*.

Ideas centrales	1) Migrar es un proceso de múltiples decisiones encadenadas en las cuales se deben documentar e identificar en el análisis inicial. 2) Factores técnicos y sociales se entrelazan y ambos sin críticos para llegar a buen puerto en la migración. 3) La falta de un marco aumenta riesgos de sobrecosto y retrabajo. 4) La iteración reduce riesgos en el proceso de migración. 5) Se necesitan guías claras de decisión para la migración.
Conceptos claves	STGT, decision points, iterative governance, stakeholder alignment.
Conclusiones	Proporciona un marco para mapear decisiones técnicas y sociales. Apoyando la tesis de la necesidad de crear un marco de migración robusto e integral.

Fuente: Elaboración propia

Ilustración 18: Extracción de fuente 4.

Sección	Descripción
Titulo	Microservices vs Serverless: A Performance Comparison on a Cloud-native Web Application
Autor(es)	Chen-Fu Fan, Anshul Jindal, Michael Gerndt
Tema	Se basa en la comparación empírica de microservicios y *serverless* en temas de latencia, confiabilidad, costo y escalabilidad.
Tesis	Cada paradigma tiene sus fortalezas en distintos contextos: microservicios en cargas prolongadas y *serverless* en picos variables.
Propósito	Evaluar métricas clave para guiar la elección de la arquitectura e identificar las fortalezas de cada paradigma, si bien no hace

	mención de la arquitectura monolítica si ayuda a visualizar las fortaleza y métricas que podemos tomar en cuenta para nuestro marco de migración.
Ideas centrales	1) Paradigma *serverless* supera a al paradigma microservicios en elasticidad ante picos. 2) Los microservicios más eficientes en cargas largas y estables. 3) Latencia de *cold start* del *serverless* es un reto clave que se debe tener presente en cualquier migración. 4) Recomendación de arquitecturas híbridas.
Conceptos claves	Cold start, autoscaling granular, throughput
Conclusiones	El estudio refuerza la importancia de medir métricas específicas antes de migrar. No hay modelo universal, y confirma la necesidad de un marco de decisión integral.

Fuente: Elaboración propia

Ilustración 19: Extracción de fuente 5.

Sección	Descripción
Titulo	Analysis of Cost-Efficiency of Serverless Approaches
Autor(es)	Nakhat Syeda, Harsh Shah, Rajvinder Singh, Suraj Jaju, Sumedha Kumar, Gourav Chhabra, Maria Spichkova
Tema	Análisis de costo-eficiencia del modelo *serverless*.
Tesis	*Serverless* puede ser costo-eficiente en cargas variables, pero su conveniencia depende del patrón de uso y la duración de las funciones.
Propósito	Establecer modelos comparativos de costo.
Ideas centrales	1) Facturación por invocación frente a capacidad reservada. 2) Sensibilidad al tamaño del payload. 3) Optimización mediante direct-to-service.

	4) Identificación de umbrales de conveniencia. 5) Relevancia de la modelación de costos.
Conceptos claves	Unit cost, cost modeling, break-even, pricing heterogéneo.
Conclusiones	Aporta criterios claros para identificar cuándo el paradigma *serverless* deja de ser rentable. Refuerza el objetivo de incluir métricas de costo en el marco metodológico.

Fuente: Elaboración propia

Ilustración 20: Extracción de fuente 6.

Sección	Descripción
Titulo	Understanding Cost Dynamics of serverless Computing: An Empirical Study
Autor(es)	Muhammad Hamza, Muhammad Azeem Akbar, Rafael Capilla
Tema	Estudio empírico de dinámicas de costo en arquitecturas *serverless*.
Tesis	El costo depende fuertemente de la naturaleza de la carga; *serverless* es rentable en cargas variables, pero puede no serlo en uso intensivo.
Propósito	Desarrollar una taxonomía de costos y estrategias de optimización.
Ideas centrales	1) *Serverless* ideal para cargas impredecibles. 2) No siempre rentable en uso sostenido. 3) Necesidad de workload profiling. 4) Estrategias de optimización claras. 5) Propuesta de taxonomía de costos.
Conceptos claves	Workload suitability, unit economics, *serverless-first*, cost taxonomy.

Conclusiones	Este artículo ofrece herramientas para determinar escenarios donde la migración no conviene, reforzando el objetivo de definir criterios de éxito basados en costos y escalabilidad.

Fuente: Elaboración propia

Ilustración 21: Extracción de fuente 7.

Sección	Descripción
Titulo	Improving Application Migration to Serverless Computing Platforms: Latency Mitigation with Keep-Alive Workloads
Autor(es)	Minh Vu, Baojia Zhang, Olaf David, George Leavesley, Wes Lloyd
Tema	Mitigación de latencia en migraciones a *serverless* mediante keep-alive workloads.
Tesis	Pre-calentar funciones mediante cargas controladas reduce la latencia de cold start y mejora la experiencia del usuario.
Propósito	Evaluar el impacto de mantener funciones activas en rendimiento y costo.
Ideas centrales	1) El cold start afecta UX y métricas clave. 2) Estrategias de pre-warm reducen colas, pero aumenta costos. 3) El costo adicional puede ser menor que la pérdida por latencias. 4) Lenguaje de programación y el SDK pueden afectar los resultados. 5) El tuning es esencial.
Conceptos claves	Cold start, keep-alive, latency p95, SLOs, performance tuning.
Conclusiones	Esta tesis muestra una solución práctica a uno de los retos principales de serverless. Aporta evidencia útil para nuestro marco metodológico, en especial en métricas de rendimiento.

Fuente: Elaboración propia

Ilustración 21: Extracción de fuente 8.

Sección	Descripción
Titulo	Investigating Differences in Performance between Monolithic and Serverless based Architectures
Autor(es)	Jonathan Manousian
Tema	Compara el rendimiento entre aplicaciones monolíticas y *serverless*.
Tesis	*Serverless* ofrece ventajas en elasticidad y respuesta a picos, pero el monolito sigue siendo más estable en cargas regulares.
Propósito	Medir rendimiento comparado en escenarios equivalentes.
Ideas centrales	1) Monolitos muestran menor latencia base. 2) *Serverless* responde mejor a picos inesperados. 3) *Cold starts* afectan consistencia. 4) Importancia de entender la carga antes de migrar. 5) No existe arquitectura universal.
Conceptos claves	Baseline monolítica, tail latency, autoscaling, elasticidad.
Conclusiones	Este artículo valida que la decisión de migrar depende de la naturaleza de la carga. Confirma la necesidad de definir un marco que considere escenarios donde no migrar puede ser al largo plazo la mejor opción.

Fuente: Elaboración propia

Ilustración 23: Extracción de fuente 9.

Sección	Descripción

Titulo	Cost Optimization Strategies in FinTech using Microservices and Serverless Architectures
Autor(es)	Sumit Bhatnagar
Tema	Estrategias de optimización de costos en *FinTech* con arquitecturas híbridas de microservicios y *serverless*.
Tesis	La combinación de microservicios y *serverless* permite alcanzar un balance costo-rendimiento ideal en sectores con alta variabilidad como *FinTech*.
Propósito	Presentar *playbooks* de optimización de costos aplicados a cargas financieras.
Ideas centrales	1) Estrategias híbridas permiten aprovechar lo mejor de ambos mundos. 2) Uso de function fusion para mejorar latencia y costo. 3) Importancia de *FinOps* (metodologia) y monitoreo constante. 4) Escenarios financieros requieren cumplimiento normativo, lo que condiciona las decisiones si no hay un cambio cultural 5) No hay estrategia única, requiere iteración.
Conceptos claves	Function fusion, event-driven finance, FinOps, observabilidad de costos.
Conclusiones	El artículo demuestra que no siempre lo óptimo es migrar todo a *serverless* y que la combinación con microservicios puede ser más adecuada dependiendo el contexto de la aplicación.

Fuente: Elaboración propia

Capítulo 2. Marco Conceptual

Se genera la nube de palabras con el objetivo de recalcar los conceptos más relevantes de la información extraída de los artículos seleccionados en la revisión de la literatura del presente documento.

Ilustración 24: Nube de palabras.

Fuente: Elaboración propia. Generado con https://www.wordclouds.com/

El Marco Conceptual establece los cimientos teóricos y terminológicos esenciales para comprender la necesidad de modernizar arquitecturas monolíticas hacia el paradigma *serverless*. Esta sección aborda las características de la arquitectura de software y contrasta los modelos de diseño tradicionales contra los modelos de FaaS (*Function as*

a Service), y formaliza los atributos de calidad que impulsan y miden el éxito de la modernización para la apoyar una toma de decisiones orientada e informada.

2.1 El paradigma de la arquitectura de software en la ingeniería moderna

Al proponer un marco de migración de una arquitectura monolítica a un paradigma *serverless* se requiere una base de comprensión profunda de cómo se estructura un sistema de información. La arquitectura es la columna vertebral que define la viabilidad, el rendimiento y la capacidad de evolución de un producto a largo plazo.

2.1.1 Definición y alcance de la Arquitectura de software

La arquitectura de software se define como la estructura fundamental de un sistema, que describe y organiza los componentes, las relaciones entre ellos, sus propiedades y los principios que rigen su diseño. "Describe la organización y el diseño de un sistema complejo, abarcando tanto los aspectos físicos como los lógicos. Esto incluye cómo los componentes de hardware, software y redes se interrelacionan y se comunican entre sí. A su vez, proporciona una vista de alto nivel sobre cómo se organizan y funcionan los diferentes elementos dentro de un sistema.” (https://www.initiumsoft.com/, 2024).

La migración a un modelo *serverless* implica una re arquitectura que transforma las estructuras actuales de los sistemas, pasando de módulos monolíticos a funciones *serverless*, redefiniendo las interacciones mediante *API Gateways* y cambiando la asignación a un entorno de nube totalmente gestionado.

2.1.2 Arquitectura orientada a servicios

La evolución de sistemas monolíticos hacia arquitecturas distribuidas, como microservicios y *serverless*, es impulsada por la necesidad de agilidad y eficiencia. En

estos modelos, los servicios se diseñan, despliegan y escalan de manera independiente. El enfoque *serverless* lleva esta tendencia al máximo nivel de granularidad, basándose en la ejecución de funciones como lo respalda (Juan Mera Menéndez, 2024).

Para una migración exitosa a arquitecturas distribuidas, la descomposición del sistema monolito debe estar alineada con la lógica del negocio y su nivel de criticidad. La disciplina de *Domain-Driven Design* (DDD) enfatiza la identificación de Contextos Delimitados (Bounded Contexts) para establecer fronteras lógicas claras entre los nuevos servicios. Adicionalmente se tiene el patrón de migración *Strangler fig*, que consta en ir migrando pequeñas secciones de la aplicación por funcionalidad, a fin de evitar colapsar completamente algún modulo funcional de la aplicación monolítica. Si la modernización no se apoya en una correcta identificación de estos contextos de negocio, la nueva arquitectura distribuida corre el riesgo de heredar el alto acoplamiento y la rigidez del sistema monolítico original.

2.2 Arquitectura monolítica

La arquitectura monolítica es el estilo tradicional donde la aplicación se construye como una unidad única de software, integrando todas las funcionalidades en un bloque indivisible de código desplegable.

2.2.1 Estructura y acoplamiento

En un sistema monolítico, aunque se utilicen patrones de diseño internos como la arquitectura de capas que consiste en la separación de lógica de negocio, presentación y acceso a datos o el Modelo-Vista-Controlador (*MVC*), la deficiencia clave reside en el acoplamiento de despliegue ya que la unidad de compilación y despliegue es única,

imponiendo limitaciones severas a la capacidad de respuesta de las organizaciones ante cualquier eventualidad en el sistema.

2.2.2 Desafíos operacionales del sistema Monolítico

El sistema monolítico de gran tamaño no solo es técnicamente rígido, sino que además genera altos costos operativos y una considerable carga adicional para los equipos de desarrollo al momento de trabajar ya que hay una interdependencia estricta entre los componentes y esto crea cuellos de botella que ralentizan el ciclo de desarrollo y despliegue.

2.2.3 Limitaciones críticas

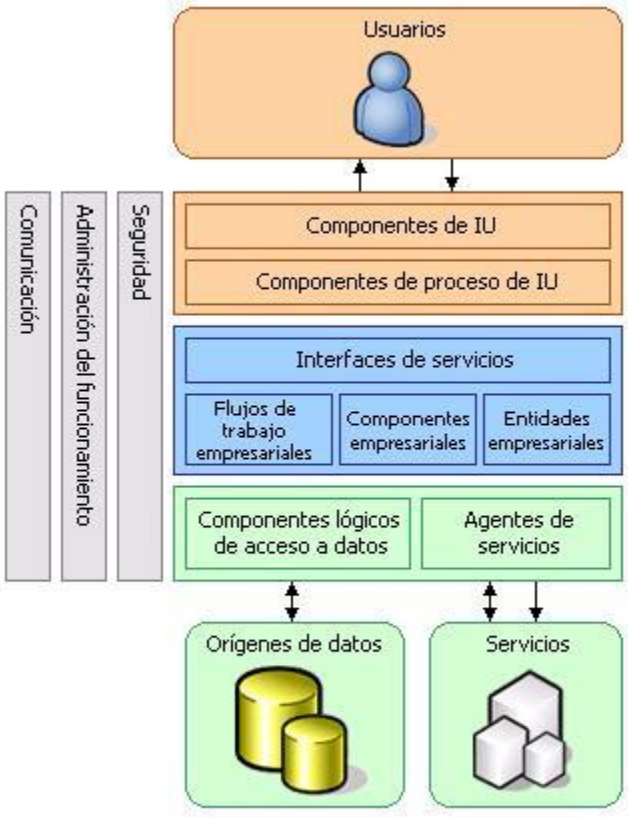
Entre las limitaciones criticas identificadas a partir de las lecturas (Alireza Goli O. H., 2020), (Nakhat Syeda, 2020) y (Manousian, Investigating differences in performance between monolithic and serverless based architectures, 2022), justifican que en el proceso de migración de un sistema monolítico a *serverless* están:

- Escalabilidad Restringida: El sistema obliga a escalar la aplicación completa para manejar picos de demanda en un módulo específico.
- Baja Resiliencia: La alta interdependencia entre componentes implica que una falla en un solo módulo puede generar un efecto dominó que comprometa todo el sistema.
- Despliegue Lento y Riesgoso: Cualquier modificación, por más menor que sea, requiere la reconstrucción y el despliegue de toda la aplicación monolítica.
- Acoplamiento Excesivo: Las dependencias rígidas entre módulos dificultan la incorporación de nuevas funcionalidades o la sustitución de componentes individuales.
- Mantenibilidad Limitada: La complejidad del código y la falta de modularidad incrementan los costos de mantenimiento y el tiempo de resolución de errores.

Se requiere más tiempo de los desarrolladores en entender completamente todo el sistema.

- Rigidez Tecnológica: La integración con nuevas herramientas, librerías o servicios en la nube se vuelve difícil debido a dependencias obsoletas.
- Costos Operativos Elevados: Los servidores deben permanecer activos de forma permanente, incluso cuando no hay una carga pesada de procesamiento.
- Baja Elasticidad: El sistema no puede ajustar dinámicamente su capacidad según las variaciones del nivel de transaccionalidad.
- Riesgos de Seguridad Ampliados: La centralización de módulos y configuraciones facilita que una vulnerabilidad en un punto afecte a todo el sistema.
- Entrega de Valor Lenta: Los largos ciclos de desarrollo y la dependencia entre equipos impiden una respuesta ágil a las necesidades del negocio.

Ilustración 25: Arquitecturas monolíticas.



Fuente: Imagen obtenida de: https://laurmolina7821.wordpress.com/1-1-1-aplicaciones-monoliticas/

2.2.4 Fortalezas del modelo monolítico

A pesar de sus limitaciones, las arquitecturas monolíticas poseen ventajas relevantes en escenarios específicos. Su simplicidad de despliegue, la comunicación interna directa entre módulos y la facilidad para mantener coherencia transaccional las hacen apropiadas para sistemas de escritorio o aplicaciones con cargas estables y bajos requerimientos de escalabilidad. En entornos donde la infraestructura es limitada o el costo de orquestación distribuida resulta alto, el monolito ofrece una solución eficiente y de menor complejidad operacional (Hamza et al., 2023; Manousian, 2022).

2.3 Arquitectura *serverless* (*FAAS*)

La arquitectura *serverless* se presenta como la solución a la lista de limitaciones planteadas referentes a los sistemas monolíticos

2.3.1 Principios y ventajas de FaaS

El modelo *serverless* busca la abstracción completa de la infraestructura, permitiendo a los desarrolladores desplegar código sin gestionar servidores, por medio de la implementación de Función como Servicio (*FaaS*). *FaaS* divide la lógica del sistema en funciones pequeñas, independientes, que se ejecutan bajo demanda en respuesta a eventos.

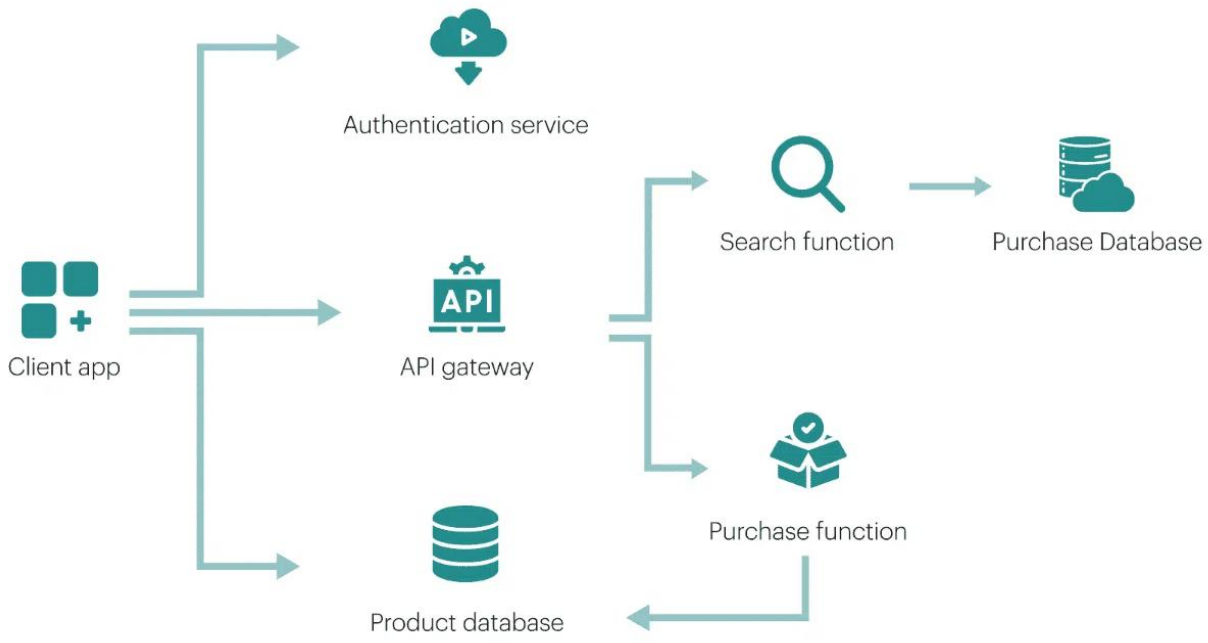
Entre las ventajas de FaaS que se detectaron durante las lecturas de (Alireza Goli O. H., 2020), (Blinowski, 2022), (Nakhat Syeda, 2020) y (Team, s.f.) se encuentran:

- Escalabilidad Automática: Las funciones se ejecutan bajo demanda y el proveedor *cloud* ajusta los recursos de forma automática según la carga de trabajo.
- Optimización de Costos: En el modelo *pay-per-use*, solo se paga por el tiempo de ejecución y los recursos efectivamente consumidos.
- Mantenimiento Reducido: El proveedor se encarga de gestionar la infraestructura.
- Alta Disponibilidad y Resiliencia: Los proveedores garantizan alto nivel de disponibilidad.
- Ciclos de Desarrollo Más Rápidos: Favorece el desarrollo modular mediante funciones independientes y reutilizables.
- Elasticidad ante Picos de Carga: Las funciones escalan dinámicamente ante eventos de alta demanda y regresan a cero cuando no hay actividad.
- Menor Tiempo de Implementación: Al eliminar la necesidad de configurar servidores y entornos complejos.

2.3.2 Proveedores y servicio complementarios

La viabilidad técnica de la migración está respaldada por plataformas maduras como *AWS Lambda, Azure Functions* y *Google Cloud Functions*. Sin embargo, una arquitectura *serverless* funcional es un ecosistema de servicios que deben integrarse, incluyendo *API Gateways*, bases de datos *serverless* y herramientas de monitoreo distribuido que es importante mencionar en esta investigación.

Ilustración 26: Arquitectura *Serverless.*



Fuente: Imagen obtenida de: https://markovate.com/blog/serverless-application-architecture/

2.3.3 Comparativa de atributos críticos

A continuación de detallaran la lista de atributos críticos que justifican una migración de un sistema monolítico a *serverless*, en esta lista como se menciona se enumeran los atributos críticos, mas no es la totalidad de beneficios que se pueden encontrar. Sin embargo, como se verá más adelante no quiere decir que al ser un beneficio aplique para todas las organizaciones ya que se debe analizar los sistemas y migraciones individualmente.

Ilustración 27: Comparativa Arquitectónica: Monolito vs. *Serverless* (FaaS).

Atributo de Calidad	**Arquitectura Monolítica**	**Arquitectura *Serverless* (FaaS)**	**Implicaciones Clave para la Migración**

Escalabilidad	Limitada, escalamiento forzado de la unidad completa	Automática, escalamiento por función bajo demanda.	Optimización de recursos.
Costo Operativo	Alto, está basado en el aprovisionamiento constante 24/7.	Optimizado, modelo *pay-per-use* (se paga lo que se usa).	Justificación económica.
Resiliencia	Baja, si falla un componente puede comprometer el sistema entero.	Alta, aislamiento de fallas por función/servicio	Mayor estabilidad y tolerancia a fallas.
Velocidad de Despliegue	Lento y riesgoso, requiere *rebuild* total.	Ágil y de bajo riesgo, despliegue independiente de funciones.	Permite ciclos de entrega más cortos y la adopción de DevOps.
Mantenibilidad	Alta, por el acoplamiento de todos sus módulos.	Modular y flexible, actualizaciones de código específicas.	Menor carga cognitiva y mayor facilidad para el mantenimiento.

Fuente: Elaboración propia

2.4 Atributos de calidad críticos en la transición arquitectónica

El marco de migración planteado debe definirse y tener métricas medibles y objetivas.

La definición de los atributos de calidad y las métricas asociadas son el núcleo del marco de migración propuesto, dentro de las principales métricas obtenidas de los artículos seleccionados para esta investigación se encuentran:

2.4.1 Rendimiento y latencia

Para arquitecturas *serverless*, las métricas deben reflejar la eficiencia operativa bajo demanda por lo que se tomaron de (Saquicela, Diseño de una Arquitectura, 2025) las siguientes métricas:

- Tasa de transferencia (*Throughput*): Se encarga de medir el número de solicitudes que el sistema es capaz de procesar por unidad de tiempo (Saquicela, Diseño de una Arquitectura en Microservicios con Indexación Dinámica y Distribuida para Optimizar la Búsqueda en Tiempo Real de la Disponibilidad en un Sistema de Reservas Empresarial Coexistente con una Arquitectura Monolítica, 2025). Una migración exitosa debe demostrar un *throughput* eficiente con un uso optimizado de recursos.
- Tiempo de Respuesta: Medición del tiempo medio que tarda el sistema en responder a una solicitud.
- Percentiles de Latencia: a (90.º, 95.º, 99.º): Valores que permiten identificar el comportamiento del sistema ante casos extremos.
- Tasa de error: Porcentaje de solicitudes fallidas.

Así mismo (Saquicela, Diseño de una Arquitectura, 2025) establece un umbral de aceptación por cada métrica tomando como referencia una serie estudios recientes y elaboró la siguiente tabla:

Ilustración 28: Umbrales de aceptación para la validación técnica.

MÉTRICA	UMBRAL DE ACEPTACIÓN	FUENTE /JUSTIFICACIÓN
Tiempo promedio de respuesta	≤ 1000 ms	(Tapia, 2020) , diferencia crítica entre escenarios monolítico y microservicios

Percentil 95 de latencia	≤ 1500 ms	(Blinowski, 2022), resultados de stress test con JMeter
Tasa de error	≤ 0.5 %	(Tapia, 2020), máximo permitido en microservicios antes de afectar estabilidad
Throughput mínimo esperado	≥ 1.4 solicitudes por segundo bajo carga	(Tapia, 2020) , caso optimizado con microservicios

Fuente: (Saquicela, Diseño de una Arquitectura, 2025)

2.4.2 Escalabilidad y Uso Eficiente de Recursos

La escalabilidad automática de *serverless* es su principal ventaja económica (Team, s.f.). El marco debe demostrar que la organización pasa de pagar por capacidad ociosa como es el paradigma monolito a pagar solo por el consumo elástico (FaaS), lo cual se mide en términos de tasa de utilización de recursos versus costo operativo.

2.4.3 Mantenibilidad y Agilidad de Despliegue

La Mantenibilidad se relaciona con la facilidad de modificar y reparar el sistema (ISO/IEC 25010). En *serverless*, la modularidad de las funciones (*FaaS*) facilita la mantenibilidad. La facilidad para diagnosticar problemas es un desafío en los sistemas distribuidos, lo que subraya la necesidad de incorporar observabilidad en el marco metodológico.

2.4.4 Costo Operativo

El Costo es un atributo de calidad que mide el gasto financiero. El modelo *pay-per-use* es la justificación económica muy importante y determinara la viabilidad de la migración y la evaluación debe cuantificar la mejora económica como el ahorro en infraestructura y los costos adicionales asociados a la observabilidad distribuida.

2.5 Estrategias y patrones de migración a *serverless*

La transición de arquitecturas monolíticas hacia paradigmas *serverless* debe fundamentarse en un diagnóstico claro de sus limitaciones estructurales y operativas. Los desafíos de escalabilidad, resiliencia y acoplamiento identificados en el capítulo anterior determinan las estrategias de migración más adecuadas. Por ello, los patrones como *Strangler Fig* y *Domain-Driven Design (DDD)* se convierten en ejes centrales del marco, al permitir una descomposición controlada y orientada al dominio, minimizando el riesgo de disrupción y preservando la lógica de negocio durante la transformación arquitectónica

2.5.1 Strangler Fig (Higuera Estranguladora)

El patrón *Strangler Fig* (Higuera Estranguladora) es el predilecto para la estrategia de re-arquitectura de monolitos complejos. Este patrón está diseñado para mitigar el riesgo de una migración y permitir una transformación incremental con mínima interrupción del negocio.

El patrón se aplica en un ciclo de tres fases, como se desprende del articulo (AWS, Strangler fig pattern, s.f.):

1. Transformar: Se identifica una funcionalidad específica del monolito (idealmente un Contexto Delimitado) y se reescribe como una función *serverless* separada que se implementa en paralelo.

2. Coexistir: Se introduce un *proxy* (generalmente un *API Gateway*) entre las aplicaciones cliente y el sistema legado. Esto para redirigir el tráfico de la funcionalidad reescrita hacia la nueva función *FaaS*, mientras que el tráfico no migrado continúa dirigiéndose al monolito. Esta fase de coexistencia permite la validación en producción con bajo riesgo.
3. Eliminar: Una vez que toda la funcionalidad ha sido extraída y no existen dependencias del sistema legado, el monolito puede ser "eliminado".

La eficacia del patrón *Strangler Fig* radica en su naturaleza incremental, ideal para gestionar el riesgo en sistemas grandes y complejos.

Ventajas:

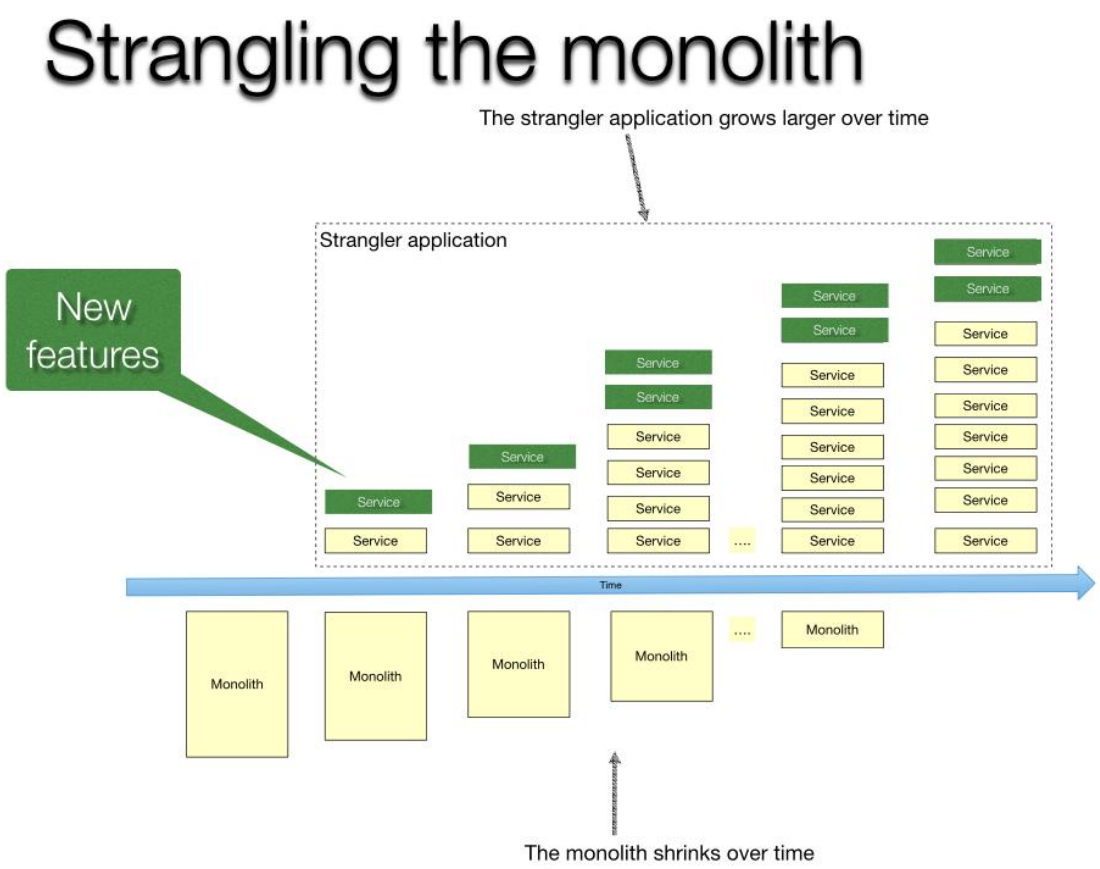
- Migración incremental y controlada: Permite transformar el sistema paso a paso, reduciendo el riesgo de fallos críticos.
- Coexistencia segura: El nuevo sistema puede operar en paralelo con el monolito, validando funcionalidades sin afectar la operación del negocio.
- Despliegue flexible: Facilita la adopción de estrategias *canary release* o *blue-green deployment* al redirigir gradualmente el tráfico mediante un *API Gateway*.
- Validación continua: Cada módulo migrado puede evaluarse individualmente mediante métricas de rendimiento, latencia y estabilidad antes de continuar con la siguiente fase.
- Reversibilidad: En caso de error o degradación, el tráfico puede revertirse fácilmente al módulo original del monolito.
- Alineación con arquitectura serverless: Encaja naturalmente con entornos basados en eventos, funciones desacopladas y servicios gestionados, favoreciendo la escalabilidad automática.

Desventajas

- Complejidad operativa: La coexistencia temporal de dos sistemas (monolito y serverless) exige monitoreo, sincronización de datos y gestión de rutas más compleja.

- Duplicidad temporal de componentes: Durante la fase de coexistencia, es posible mantener versiones duplicadas de lógica o datos, lo que incrementa el esfuerzo de mantenimiento.
- Dependencias ocultas: Si el monolito no está bien documentado, pueden surgir dependencias entre módulos que dificulten la extracción o generen errores no previstos.
- Gestión de estado y base de datos compartida: Migrar módulos que dependen de una base de datos centralizada puede ser difícil sin una estrategia de desacoplamiento progresivo.
- Costos iniciales elevados: Aunque reduce el riesgo técnico, la creación de *proxies*, *gateways*, herramientas de observabilidad y pipelines incrementa el costo inicial del proceso.
- Duración prolongada del proyecto: Al ser incremental, la migración completa puede requerir más tiempo que un enfoque de reemplazo total.

Ilustración 29: Strangler fig.



Fuente: Imagen obtenida de: https://microservices.io/patterns/refactoring/strangler-application.html

2.5.2 Bounded Context Extraction (DDD) - Extracción por Contextos Delimitados

Domain-Driven Design (*DDD*) identificar, delimitar y extraer funcionalidades del monolito según contextos de negocio coherentes (*bounded contexts*) (Fowler, 2014). El objetivo es minimizar el acoplamiento entre lo que se extrae y lo que permanece, y maximizar la cohesión funcional, preparando módulos que puedan convertirse en funciones *serverless* (FaaS) o en microservicios.

Ciclo de aplicación (fases):

1. Descubrir y delimitar: Mapear dominios y subdominios, detectar *bounded contexts* y dependencias críticas (datos compartidos, integraciones, sesiones).
2. Encapsular y aislar: Introducir interfaces claras (puertos/adaptadores) para cortar dependencias implícitas del monolito; definir contratos (*APIs*, eventos, esquemas).
3. Extraer y publicar: Reescribir el contexto elegido como funciones serverless (o microservicios mínimos), publicando eventos/colas o *APIs* detrás de *API Gateway* y mantener coexistencia con el monolito mientras se valida.
4. Optimizar datos: Migrar gradualmente el modelo de datos del contexto (evitar DB compartida), aplicar patrones de eventos, CDC o proyecciones para sincronizar.

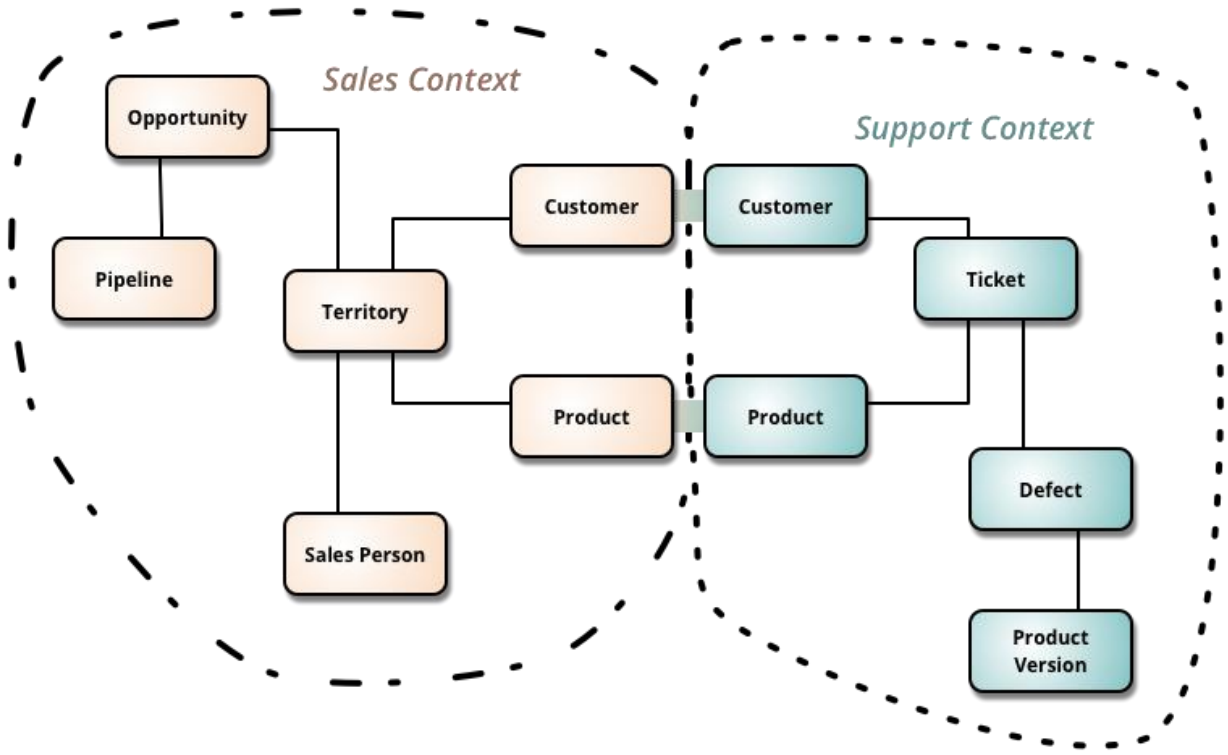
Ventajas

- Enfoque dirigido por negocio: las extracciones siguen fronteras naturales del dominio.
- Reduce acoplamiento y mejora cohesión, facilitando escalabilidad y mantenibilidad.
- Prepara el terreno para *serverless* event-driven (funciones con disparadores por colas, tópicos, webhooks).

Desventajas

- Requiere descubrimiento de dominio (talleres, mapas contextuales); demanda tiempo y experticia.
- Riesgo de fronteras mal definidas.
- La migración de datos (de DB compartida a modelos propios) es compleja y sensible.

Ilustración 30: Bounded Context.



Fuente: Imagen obtenida de: https://martinfowler.com/bliki/BoundedContext.html

Aunque *Domain-Driven Design* (*DDD*) no es un patrón de migración, constituye una disciplina fundamental para identificar límites funcionales (*bounded contexts*) dentro del monolito. Estos límites permiten definir unidades coherentes que pueden migrarse mediante patrones.

2.5.3 Branch by Abstraction - Rama por Abstracción

Este patrón sirve para cambiar implementación “por debajo” de una interfaz estable sin interrumpir el servicio. Se introduce una abstracción (interfaz), se implementa la versión nueva (*serverless*, servicios o adaptadores) y se conmuta de la implementación antigua a la nueva de manera controlada (*feature flag*, *routing* por API Gateway, canary) (Hammant, 2020).

Ciclo de aplicación:

1. Crear la abstracción: Definir una interfaz estable (puerto) sobre la funcionalidad a migrar.
2. Doblar el monolito: Hacer que el monolito consuma esa interfaz, la implementación “A” es la actual (legado).
3. Implementar la rama nueva: Desarrollar la implementación “B” en *serverless* (funciones + colas/eventos + almacenamiento externo).
4. Conmutar gradualmente: Redirigir tráfico a “B” con feature flags/canary; medir SLOs, errores, latencia; roll-back si falla.
5. Retirar la vieja: Al cumplir criterios de éxito, eliminar implementación “A” y consolidar.

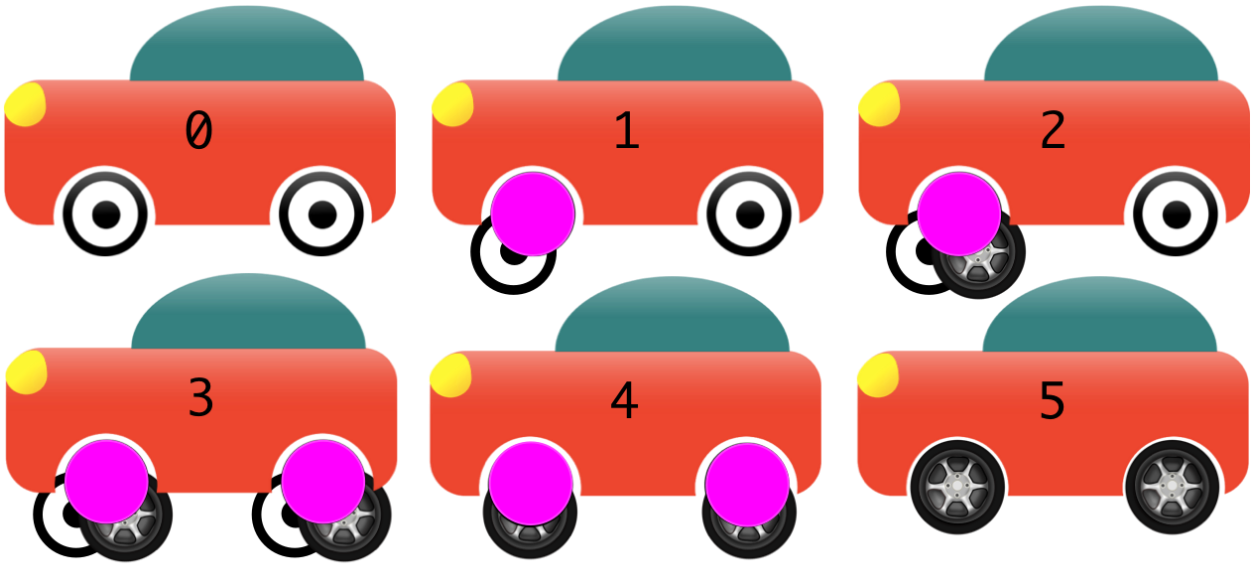
Ventajas

- Riesgo controlado: cambio detrás de una interfaz y conmutación gradual.
- Reversibilidad (*roll-back* rápido) si la nueva ruta no cumple métricas.
- Facilita pruebas A/B y validación en producción con bajo impacto.

Desventajas

- Trabajo adicional para crear y mantener la abstracción.
- Puede duplicar lógica temporalmente hasta la desactivación final.
- Requiere observabilidad fina (trazas, métricas, *logging*) y *feature-flagging* disciplinado.

Ilustración 31: Branch by Abstraction.



Fuente: Imagen obtenida de: https://trunkbaseddevelopment.com/branch-by-abstraction/

2.5.4 **Marco de decisión integral: Patrones y Métricas**

El diseño del marco de migración debe ser integral, combinando de manera coherente los patrones de migración y las métricas de evaluación definidas previamente. Entre los distintos enfoques analizados, se privilegia el uso del patrón *Strangler Fig*, ampliamente recomendado por AWS en sus lineamientos de modernización (AWS, Strangler fig pattern, n.d.), debido a su capacidad para gestionar el riesgo y facilitar una transformación incremental del sistema.

Este patrón permite tomar decisiones basadas en evidencia, al evaluar de forma objetiva el impacto de cada componente migrado sobre los atributos de calidad del software, tales como rendimiento, escalabilidad, mantenibilidad y costo operativo.

En el marco propuesto, la eliminación definitiva del componente monolítico solo se aprueba cuando las métricas obtenidas demuestran una mejora verificable en los atributos críticos definidos, garantizando así que la migración aporte valor tangible al sistema y no represente una regresión arquitectónica.

2.6 Desafíos arquitectónicos y operacionales de *serverless*

Este marco conceptual pretende mantener un enfoque riguroso, tomando en cuenta no solo los beneficios del modelo *serverless*, sino también sus riesgos y posibles desventajas. La idea es tener una visión completa que ayude a definir cómo mitigar esos riesgos y entender mejor las limitaciones antes de proponer una migración desde un sistema monolítico.

2.6.1 Cold Start - Latencia en inicio en frio

Como indica (Muhammad Hamza M. A., 2023) en su artículo original “El Inicio en Frío (*Cold Start*) es un desafío de rendimiento en *FaaS*. Ocurre cuando una función debe inicializarse después de un período de inactividad, lo que resulta en una latencia inicial más alta.” Este efecto se refleja directamente en los percentiles más altos de latencia. El marco debe incluir una clasificación de las funcionalidades a migrar, recomendando estrategias activas de mitigación como *provisioned concurrency* o *keep-alive* pings para funciones críticas que requieren baja latencia como por ejemplo *APIs* de cara al cliente.

Como indica (Manousian, Investigating differences in performance between monolithic and serverless based architectures, 2022) existen diferentes metodologías en cómo mantener las funciones activas para evitar el *cold start*, sin embargo, esto conlleva una serie de desventajas como el incremento de los costos en *cloud*.

2.6.2 Bloqueo del proveedor (vendor Lock-in)

El Bloqueo del Proveedor (Vendor Lock-in) es una barrera importante en la adopción de la nube, muy importante en el paradigma *serverless* debido a la fuerte dependencia de *APIs* y servicios propietarios de los proveedores y la falta de estandarización

compromete la portabilidad de las funciones como lo respalda (Justice Opara-Martins, 2016) en su artículo.

El marco de migración debe abordar la mitigación del Vendor Lock-in promoviendo la conciencia sobre las dependencias propietarias y recomendando la selección de proveedores que soporten formatos y protocolos estandarizados para la transferencia de datos.

2.6.3 Desafíos de monitoreo distribuido

La naturaleza distribuida y efímera de la arquitectura *serverless* complica la operación y el monitoreo, ya que los logs y el flujo de ejecución están descentralizados.

El marco de migración debe estipular la implementación de una estrategia de observabilidad, incluyendo la trazabilidad distribuida y el uso de plataformas de Gestión de *Logs*. Esto asegura la monitorización de los atributos de calidad recomendado (Tamushi, 2022) y por ende la salud del sistema garantizando que la agilidad del despliegue no se vea comprometida por la dificultad para diagnosticar problemas rápidamente.

2.6.4 Seguridad y cumplimiento

La migración debe garantizar que se cumplan los principios de seguridad y *compliance*, ya que las funciones *serverless* se ejecutan con roles específicos y deben seguir el principio de mínimo privilegio.

2.6.5 Decisiones socio-técnicas y marcos de gobierno

Al pasar a una arquitectura *serverless* no basta con únicamente conectarse a otra tecnología, ya que se debe replantear la forma de trabajar. En un artículo se estudiaron 22 migraciones y 31 blogs desde una perspectiva socio-técnica y documentaron 29

hitos decisorios que atraviesa cualquier organización en ese camino. (Muhammad Hamza M. A., 2023). Algunos de estos hitos están al inicio como son clarificar por qué se migra en términos de escalabilidad, ahorro e innovación, y evaluar si el equipo está listo y elegir qué módulos irán primero. Otros aparecen al final como seleccionar la arquitectura de destino, decidir entre comunicación síncrona o asíncrona, elegir proveedor de nube y definir reglas de gobierno y permisos. Según los autores, lo crucial es que todas esas decisiones estén en sintonía con la cultura y la estructura de la organización. De ahí que resalten la adopción de prácticas DevOps, el aprendizaje continuo y una actitud iterativa como factores tan determinantes como la tecnología.

2.7 Síntesis comparativa basada en la evidencia empírica

En la literatura mencionada previamente se han realizado estudios empíricos que comparan arquitecturas monolíticas y *serverless* (y en algunos casos microservicios) en términos de rendimiento, escalabilidad y costo. En esta sección se sintetizan los hallazgos más relevantes con el fin de ofrecer una visión integrada que sirva de base al problema de investigación y a la posterior construcción de un marco de migración. (Manousian, Investigating differences in performance between monolithic and serverless based architectures, 2022) realiza una comparación controlada entre una aplicación web implementada como monolito desplegado en *PaaS* y una versión equivalente basada en funciones *serverless* sobre *AWS Lambda*. Sus resultados muestran que, bajo cargas estables y moderadas, ambas arquitecturas presentan tiempos de respuesta y tasas de error similares, siempre que se configuren adecuadamente los recursos. Sin embargo, cuando la carga aumenta de forma súbita la arquitectura *serverless* mantiene mejor la latencia y la confiabilidad gracias al autoescalado fino por función, mientras que el monolito requiere sobreaprovisionar instancias completas para sostener el mismo nivel de servicio. En términos de costos, el estudio concluye que *serverless* optimiza el gasto en escenarios de baja a mediana demanda, pero a partir de un umbral de solicitudes mensuales el modelo *pay-per-use* puede superar el costo fijo de un despliegue monolítico sobredimensionado.

Un estudio complementario de (Alireza Goli O. H., 2020) compara una aplicación *cloud-native* de gestión de hojas de tiempo en dos variantes: microservicios desplegados en contenedores (*ECS*) y una versión *serverless* basada en Lambda y API Gateway. Los autores analizan cuatro dimensiones: escalabilidad, confiabilidad, costo y latencia. Sus resultados indican que la arquitectura de microservicios o monolítica tiene ventaja de costo cuando se trata de servicios de larga duración o cargas muy sostenidas, ya que el costo por segundo de ejecución en funciones *serverless,* se vuelve menos competitivo en ejecuciones prolongadas. En cambio, para solicitudes con respuestas de mayor tamaño o picos de carga irregulares, la arquitectura *serverless* se comporta mejor, ya que escala con mayor rapidez sin requerir gestión explícita de instancias.
Por su parte, los estudios de (Muhammad Hamza M. A., 2023) se centran en comprender la dinámica de costos y la adecuación de *workloads* a arquitecturas *serverless* a partir de entrevistas con profesionales de ocho organizaciones, determinaron que *serverless* resulta particularmente atractivo cuando las cargas son impredecibles, altamente estacionales o con picos marcados, mientras que para aplicaciones con tráfico muy estable y alto volumen puede ser más económico mantener arquitecturas tradicionales o basadas en contenedores. En el mismo sentido, investigaciones sobre estrategias de optimización de costos refuerzan que la combinación de *serverless* y microservicios permite reducir costos en componentes altamente variables, manteniendo servicios de larga duración en plataformas más tradicionales (Nakhat Syeda, 2020).

En conjunto, la evidencia empírica sugiere que:

- No existe una arquitectura “dominante” en todos los escenarios:
    - Las arquitecturas monolíticas siguen siendo competitivas en contextos de baja complejidad y cargas estables.
    - Los monolitos tienes ventajas cuando son procesamientos de larga duración, además si son cargas constantes durante el mes sin experimentar picos de trabajo.

    - Las arquitecturas *serverless* destacan en elasticidad, reducción de la gestión infraestructural y eficiencia de costos ante cargas variables o difíciles de predecir.
- El comportamiento en costo y rendimiento está fuertemente mediado por el patrón de carga (estable vs. estacional vs. altamente impredecible) y por características del dominio (criticidad, requisitos de latencia, volumen de datos, regulaciones).

Esta síntesis refuerza la necesidad de contar con un marco de decisión que ayude a determinar, para un sistema monolítico específico, cuándo una migración hacia *serverless* es recomendable, qué componentes priorizar y cómo equilibrar los atributos de calidad involucrados (rendimiento, costo, mantenibilidad, escalabilidad, entre otros).

2.8 Manejo del Estado en la Migración a Serverless

El tratamiento del estado es uno de los factores más relevantes al evaluar la viabilidad de una migración desde un monolito hacia *serverless*. En un sistema monolítico tradicional, el estado suele mantenerse en memoria compartida o gestionarse en sesiones persistentes, lo cual dificulta su separación en unidades funcionales independientes. Según (Manousian, Investigating differences in performance between monolithic and serverless based architectures, 2022), la transición a *serverless* exige externalizar el estado, por ejemplo, hacia bases de datos, colas de eventos o servicios de cache, esto para evitar dependencias internas que impidan el escalado automático y la ejecución paralela de funciones. Esta externalización también reduce el riesgo de fallos en cascada y facilita el aislamiento por invocación, uno de los atributos más citados en la literatura como ventaja fundamental de FaaS.

2.9 Consideraciones de Logging, Monitoreo y Observabilidad en Arquitecturas Serverless

La migración desde un sistema monolítico hacia una arquitectura *serverless* modifica de forma sustancial la manera en que se gestionan el *logging*, el monitoreo y la

observabilidad. Mientras que en el monolito el registro de eventos se concentra en un único proceso, en *serverless* las funciones se ejecutan de forma distribuida y efímera, dispersando los *logs* entre múltiples puntos de ejecución. Esto incrementa la complejidad para correlacionar solicitudes y reconstruir flujos *end-to-end*. (Chen-Fu Fan, 2020) evidencian que esta fragmentación exige mecanismos avanzados de recopilación centralizada y trazabilidad distribuida. (Muhammad Hamza M. A., The Journey to Serverless Migration: An Empirical Analysis of Intentions, Strategies, and Challenges, 2023) confirman que *serverless* reduce la carga operativa tradicional, pero introduce nuevos desafíos en la supervisión granular de funciones.

En la práctica, los entornos *serverless* dependen de servicios especializados de observabilidad proporcionados por los proveedores *cloud*. En AWS, estos incluyen: *Amazon CloudWatch*, *AWS X-Ray*, *AWS CloudTrail*, *AWS Lambda Insights*, *Amazon OpenSearch Service* y *Amazon EventBridge.* En Google Cloud, los servicios comúnmente utilizados son: *Cloud Logging*, *Cloud Monitoring*, *Cloud Trace*, *Cloud Error Reporting*, *Cloud Profiler* y *Operations Suite* (*Stackdriver*). La existencia y uso de estos servicios no forma parte del marco metodológico de migración, pero sí constituyen un fundamento conceptual necesario para comprender las diferencias operativas entre un monolito y un sistema basado en funciones, y para anticipar los desafíos que se evaluarán en etapas posteriores del proyecto.

2.10 Conclusiones

El marco conceptual establece que la migración de aplicaciones monolíticas a *serverless* debe verse como un proceso gradual y estratégico ya que no existe una respuesta universal debido a que el éxito depende de las características del dominio, el patrón de carga, la tolerancia a la latencia y los objetivos de negocio. Las decisiones deben basarse en métricas cuantitativas y consideraciones cualitativas, teniendo en cuenta la estructura socio-técnica de la organización.

Los estudios revisados demuestran que el paradigma *serverless* aporta beneficios significativos en términos de escalabilidad y reducción de costos para cargas impredecibles, pero también expone limitaciones como los *cold starts* y la dependencia del proveedor *cloud* elegido.

El marco de migración efectivo debe proporcionar guías para seleccionar componentes, definir patrones de migración, monitorear métricas de calidad y adaptar la cultura organizacional.

Capítulo 3 Marco Metodológico

3.1 Tipo de Investigación

El presente trabajo de investigación tiene como objetivo analizar las características de las arquitecturas monolíticas y *serverless*, con el fin de proponer un marco de migración que sirva como guía para realizar transiciones efectivas entre ambos enfoques y para lograrlo, se realizará una revisión de literatura académica y técnica que permita comprender las estrategias, patrones y métricas más utilizadas en procesos de modernización de software, así como los retos asociados a la adopción del paradigma *serverless*.

Además, se examinan estudios y casos documentados que describen experiencias de migración de aplicaciones monolíticas hacia entornos en la nube, con el propósito de identificar patrones comunes, buenas prácticas y factores que determinen cuándo una migración resulta conveniente y cuándo puede generar más desventajas que beneficios.

Con base en esta evaluación, se desarrollará una propuesta metodológica que integre las principales estrategias y criterios de decisión arquitectónica, considerando aspectos como el rendimiento, la escalabilidad, la mantenibilidad, los costos y la seguridad. El objetivo final es ofrecer una guía clara y práctica que ayude a la comunidad de

profesionales y equipos técnicos a planificar y ejecutar procesos de migración de manera estructurada, minimizando riesgos y maximizando beneficios.

El presente estudio se clasifica como una investigación aplicada, ya que busca generar una solución práctica a un problema real en el campo de la ingeniería del software como es la modernización de sistemas monolíticos hacia arquitecturas *cloud* más flexibles y escalables en el análisis y comparación de fuentes existentes sin manipular variables.

El propósito central es desarrollar un marco metodológico que contribuya al avance en la toma de decisiones arquitectónicas dentro de procesos de migración hacia *serverless*, sirviendo como referencia para profesionales de software y organizaciones que busquen modernizar sus aplicaciones de forma segura, eficiente y fundamentada.

3.2 Alcance Investigativo

Para la presente investigación se consideran dos tipos principales de investigación, de acuerdo con la clasificación propuesta: exploratoria y descriptiva. Asimismo, el estado de la cuestión en la investigación y la perspectiva del investigador son factores que influyen en la definición del alcance de la investigación (Roberto Hernández Sampieri, 2014).

3.2.1 Exploratoria

Según este mismo autor, los estudios exploratorios tienen como propósito principal proporcionar una visión general y aproximada del fenómeno estudiado, cuando este ha sido poco abordado o no existen suficientes marcos teóricos consolidados. Este trabajo se enmarca dentro de esta categoría, ya que el campo de la migración de aplicaciones monolíticas hacia arquitecturas *serverless* es relativamente reciente en la práctica profesional y aún carece de lineamientos metodológicos estandarizados.

El estudio busca familiarizar al investigador con las distintas estrategias de modernización, identificar patrones arquitectónicos utilizados en entornos reales como

el Strangler Fig Pattern o DDD y analizar los retos técnicos y organizacionales implicados.

3.2.2 Descriptiva

De acuerdo con la metodología de investigación propuesta, los estudios descriptivos se orientan a especificar propiedades, características y perfiles de fenómenos que se analizan y de los cuales ya hay cierta base teórica (Roberto Hernández Sampieri, 2014). En esta investigación, el enfoque descriptivo permite caracterizar las arquitecturas monolíticas y *serverless*, documentando sus ventajas, limitaciones, patrones asociados y atributos de calidad como el rendimiento, la escalabilidad, la mantenibilidad y el costo operativo.

La finalidad es representar con claridad cómo se manifiestan las prácticas y estrategias de migración en distintos contextos, permitiendo a los arquitectos de software comprender de manera estructurada los elementos que influyen en la decisión de adoptar una arquitectura *serverless*. Además, el enfoque descriptivo posibilita la elaboración de un marco comparativo que servirá como base para definir métricas y criterios de evaluación en etapas posteriores del proyecto.

3.3 Enfoque

Se adopta un enfoque de investigación cualitativo. Esta elección metodológica se justifica dada la naturaleza del problema de la arquitectura de *software*, que requiere la interpretación y síntesis de información no estandarizada, como las experiencias documentadas y los casos de estudio. El enfoque cualitativo permite construir un marco de migración a partir de las lecciones aprendidas por profesionales en la transición de arquitecturas monolíticas a *serverless*.

La investigación se basa en la recolección y análisis de datos cualitativos derivados de la literatura. Esto incluye: definiciones teóricas, caracterización de desafíos arquitectónicos (*Cold Start*, *Vendor Lock-in*) y la identificación de métricas de decisión

que definen la viabilidad técnica y operativa. Este análisis interpretativo es esencial para generar las pautas y los criterios de exclusión que estructurarán el marco propuesto.

3.4 Diseño

Para este proyecto, el diseño de investigación utilizado es de naturaleza evaluativa y propositiva con un enfoque descriptivo-transversal.

- Diseño Propositivo: La fase central del proyecto consiste en la creación de un Marco de Migración de Aplicaciones Monolíticas hacia Arquitecturas *Serverless*. Este marco representa una solución estructurada que busca orientar la toma de decisiones arquitectónicas y metodológicas en las organizaciones. Un diseño propositivo se enfoca en generar un modelo o una solución aplicable para resolver una problemática identificada previamente.
- Diseño Evaluativo: Esta característica se aplica a la fase inicial y final. Primero, se evalúan las limitaciones técnicas (escalabilidad, latencia) y operativas (costo, mantenibilidad) de la arquitectura monolítica. Posteriormente, el marco propuesto está diseñado para ser evaluado mediante criterios de viabilidad y métricas de decisión que permiten a la organización justificar la migración o descartarla en componentes específicos.
- Diseño Transversal: La recolección de información se realiza en un momento único del tiempo a lo largo del periodo de investigación. Este enfoque se centra en la revisión documental sistemática y la síntesis de hallazgos para describir las mejores prácticas, los patrones de migración (*Strangler Fig y DDD*) y los criterios de exclusión como son: *cold start*, requisitos de latencia crítica entre otros que definen el estado del arte y sirven de insumo para la propuesta.

El diseño opera en una secuencia lógica de diagnóstico de limitaciones, síntesis de soluciones y criterios, y la creación del marco operativo.

3.5 Instrumento de Recolección de Datos

El instrumento fundamental para la recolección de datos en un diseño documental es la Documentación (Roberto Hernández Sampieri, 2014). Para garantizar el rigor y la objetividad en la extracción de la información de la muestra, se utiliza una Ficha de Contenido y Codificación.

Este instrumento permite la extracción estructurada de los elementos clave, asegurando que la información recopilada se alinee directamente con las variables de la investigación y los componentes del Marco de Migración propuesto:

- Identificación: Título, Autor(es), Año, Fuente (ej. IEEE, ACM).
- Contexto: Tipo de Arquitectura (Monolito, *Serverless*) y Patrón de Migración analizado.
- Datos Cualitativos: Identificación de desafíos (ej. *Cold Start, Vendor Lock-in*) y las estrategias arquitectónicas de mitigación propuestas.
- Conclusiones y Aportes: Resumen de las conclusiones del estudio primario y su contribución al Marco de Migración.

3.6 Técnicas de Análisis de la Información

Las técnicas de análisis de información tienen como objetivo analizar, procesar e interpretar los resultados de la revisión documental para dar a conocer el contenido esperado de la investigación. La información recolectada es analizada mediante un enfoque cualitativo y descriptivo. En el análisis de documentos, se realiza un proceso de sistematización por categorías temáticas, como, por ejemplo, patrones de migración, desafíos arquitectónicos y métricas de viabilidad, con el fin de organizar conceptos clave a los objetivos del proyecto.

El análisis cualitativo permitirá identificar patrones de migración, desafíos frecuentes y decisiones arquitectónicas mediante la revisión documental. La validez de los resultados, que indica el grado en que el contenido refleja lo que se requiere medir.

Este rigor analítico es esencial para plantear la definición de una serie de métricas que determinarán la factibilidad y establecerán los criterios de exclusión o inviabilidad de la migración.

Capítulo 4. Análisis del Diagnóstico

4.1. Propósito y Fuentes del Diagnóstico

Este capítulo presenta el análisis del estado actual de la problemática descrita y desarrolla, de manera estructurada, la síntesis de los hallazgos derivados de la revisión técnica, arquitectónica y literaria realizada hasta el momento, a partir de los documentos seleccionados aplicables a migraciones hacia arquitecturas *serverless*. El objetivo es caracterizar las limitaciones inherentes de las arquitecturas monolíticas tradicionales, identificar los atributos de calidad que resultan más afectados en dichos entornos y examinar el potencial de una transición controlada hacia arquitecturas *serverless* basadas en funciones como servicio (*Function as a Service*, *FaaS*) cuando aplique.

La información que se analiza en este diagnóstico proviene de dos fuentes principales dentro de la propia investigación aplicada:

a) La caracterización del problema y la justificación del proyecto (Capítulo 1), donde se establecen las restricciones técnicas y operativas asociadas al modelo monolítico.

b) El marco conceptual y la revisión de la literatura (Capítulo 2 y 3), en el cual se definen y contrastan las arquitecturas, se caracterizan los atributos de calidad críticos como son escalabilidad, latencia, mantenibilidad, costo operativo y se introducen las estrategias de migración documentadas.

Siguiendo el enfoque metodológico de la investigación aplicada con alcance descriptivo, este diagnóstico se centra en la interpretación y síntesis de esas evidencias para alinearlas con las dimensiones fundamentales de la toma de decisión

arquitectónica, sirviendo como la base empírica para el diseño del marco de migración efectivo.

4.2. Limitaciones Operativas y Técnicas de la Arquitectura Monolítica

El punto de partida del diagnóstico es el análisis del estado actual de los sistemas basados en arquitecturas monolíticas, caracterizadas por una alta interdependencia interna. Los componentes de negocio, las capas de presentación, la lógica de negocio y la capa de datos se encuentran fuertemente acoplados en una única unidad de despliegue y ejecución.

Este acoplamiento introduce severas limitaciones operativas y técnicas, que han sido descritas en la literatura como los principales desafíos del modelo:

1. Despliegue indivisible: El despliegue de cualquier cambio, incluso si afecta únicamente un módulo específico, exige la reconstrucción y el despliegue de todo el sistema. Esto ralentiza los ciclos de liberación, incrementa la complejidad del mantenimiento y eleva el riesgo sistémico, ya que un error localizado tiene el potencial de degradar o interrumpir el servicio completo.
2. Escalabilidad ineficiente: La escalabilidad en el modelo monolítico es típicamente de tipo vertical (aumentar recursos al bloque completo) o horizontal forzada (replicar todo el bloque). Para atender picos de carga en una sola funcionalidad crítica, la organización se ve obligada a sobredimensionar la infraestructura de toda la aplicación, lo que resulta en un uso ineficiente de los recursos y un consecuente aumento del costo operativo.
3. Aumento de la Deuda Técnica: La concentración de múltiples responsabilidades en una única base de código dificulta la evolución tecnológica, el control de versiones y el *onboarding* de nuevos desarrolladores. Esta rigidez se traduce en lentitud para la respuesta a incidentes y una acumulación progresiva de deuda técnica.

En síntesis, la arquitectura monolítica presenta un patrón de rigidez que impacta negativamente en la Escalabilidad, la Resiliencia Operativa y la Agilidad de Despliegue, elementos que constituyen el problema base que justifica la necesidad de un marco de migración hacia modelos más elásticos y desacoplados.

4.3. Atributos de Calidad Críticos en la Transición hacia *Serverless*

El marco conceptual de esta investigación identifica una serie de atributos de calidad que resultan determinantes cuando se evalúa una migración arquitectónica: rendimiento y latencia, escalabilidad y uso eficiente de recursos, mantenibilidad y agilidad de despliegue, y costo operativo. El diagnóstico se enfoca en contrastar estos atributos para evidenciar el potencial de mejora al adoptar un enfoque *serverless* (FaaS).

A continuación, la Tabla 4.1 resume el impacto que la transición de una arquitectura a otra tiene sobre los atributos de calidad críticos analizados:

Ilustración 32: Analizar fortalezas y debilidades del monolito y cómo migrarlo efectivamente hacia tecnología moderna.

Debilidad del Monolito	Riesgo Arquitectónico Asociado	Cómo lo Mitiga Serverless (FaaS)	Implicaciones para la Migración
Escalabilidad rígida (escalado de toda la aplicación)	Sobreaprovisionamiento, costos elevados y fallos bajo picos de carga.	Escalado automático por función; adaptación granular a la demanda.	La migración prioriza módulos con alta variabilidad para maximizar el beneficio inicial.

Acoplamiento extremo entre módulos	Fallos en cascada; dificultad para aislar problemas o evolucionar partes del sistema.	Aislamiento por función/evento; desacoplamiento implícito basado en triggers.	Migrar primero funciones con menor dependencia reduce riesgo y permite validación incremental.
Despliegues lentos y de alto riesgo	Ventanas de downtime; errores pequeños afectan todo el sistema.	Despliegue independiente por función; rollback inmediato.	Permite estrategia incremental tipo Strangler Fig con validación continua.
Dependencia de infraestructura fija (servidores 24/7)	Costos permanentes, baja eficiencia de recursos.	Modelo pay-per-use; eliminación de mantenimiento infraestructural (NoOps).	Justifica omitir microservicios y migrar directamente hacia serverless para reducir costos operativos.
Baja resiliencia y efectos dominó	Un fallo individual puede comprometer el sistema completo.	Aislamiento transaccional por invocación; contención de fallas por función.	La migración debe priorizar módulos críticos donde la resiliencia es más valiosa.
Dificultad de mantener el sistema (código grande, deuda técnica)	Altos tiempos de diagnóstico, menor velocidad de desarrollo.	Funciones pequeñas, acotadas y más fáciles de probar/observar.	Migración reduce deuda técnica de forma progresiva al extraer bounded contexts.
Rigidez tecnológica (depende de versiones antiguas)	Obstáculos para adoptar frameworks modernos, mayor costo de actualización.	Componentes nuevos se construyen con tecnologías actuales sin afectar el monolito.	Serverless actúa como vector de modernización tecnológica incremental.

Elasticidad limitada	No se adapta a cambios repentinos en carga, comprometiendo UX y disponibilidad.	Autoscaling instantáneo sin intervención humana.	Migración mejora directamente los atributos de disponibilidad y experiencia del usuario.
Costos operativos altos	Pago por capacidad ociosa + mantenimiento de servidores.	Facturación por invocación, almacenamiento y ejecución real.	Impacto económico inmediato, especialmente útil para pymes o sistemas con picos marcados.

Fuente: Elaboración propia.

4.4. Estrategias y Patrones de Migración como Componentes Críticos

Dentro del marco conceptual se describieron estrategias y patrones de migración orientados a descomponer gradualmente una arquitectura monolítica. Este diagnóstico reconoce que la migración no es un evento monolítico ni puramente técnico, sino un proceso estructurado e incremental.

En la literatura analizada (Alireza Goli O. H., 2020) (Muhammad Hamza M. A., 2025), se observa que no existe un patrón único ni lineal, sino que las organizaciones tienden a combinar enfoques según el contexto y la madurez de su sistema.

Entre los patrones de migración más utilizados destacan el *Strangler Fig Pattern*, que permite reemplazar progresivamente componentes del monolito sin interrumpir la operación, y *Domain-Driven Design* (*DDD*), que sirve para identificar límites funcionales (*bounded contexts*) y definir unidades lógicas independientes antes de la migración.

Es importante indicar que ambos patrones se complementan:

- *DDD* aporta la base conceptual y estructural, ayudando a dividir el dominio en módulos coherentes.

- *Strangler Fig* define la estrategia práctica y secuencial para reemplazar o envolver partes del monolito con funciones o servicios *serverless*.

En migraciones directas hacia *serverless*, esta combinación permite mantener continuidad operativa, reducir riesgos y facilitar la evolución hacia un modelo *event-driven* y escalable.
Los estudios empíricos revisados confirman que esta estrategia híbrida es la más frecuente en organizaciones que modernizan sistemas legados complejos sin reescribirlos por completo.

4.5. Factores Organizacionales y de Viabilidad

La viabilidad de la propuesta exige que el diagnóstico analice la migración como una decisión que trasciende lo tecnológico para convertirse en un cambio estratégico y organizacional.

- Viabilidad Técnica: Requiere que la organización desarrolle las competencias necesarias para operar servicios en la nube, comprender el modelo *FaaS* y administrar entornos distribuidos. Esto incluye el dominio de conceptos como observabilidad a nivel de funciones, orquestación de eventos y la definición de límites funcionales claros (desacoplamiento).
- Viabilidad Operativa: La transición implica una adaptación de los procesos de desarrollo, despliegue, monitoreo y soporte. El paso a un modelo altamente distribuido exige prácticas de DevOps y de gobierno riguroso que sustituyen la lógica centralizada del monolito.
- Viabilidad Económica: Aunque la arquitectura *serverless* apunta a la optimización de costos por cómputo (pago por uso), el análisis económico debe ser riguroso. El diagnóstico reconoce que el valor no se limita al ahorro inmediato, sino que incluye el retorno de inversión generado por la reducción de la deuda técnica y la mayor agilidad de entrega de valor en el mediano y largo plazo.

El diagnóstico confirma que la decisión de migrar es multidimensional, y el marco debe contemplar tanto el impacto en los atributos técnicos de calidad como las restricciones reales de la organización (recursos humanos, madurez operativa y presupuesto) y de no alcanzar una madurez mínima en estos tres pilares la migración de los componentes o del monolítico completo puede resultar inviable.

4.6. Síntesis del Diagnóstico

El análisis realizado en este capítulo permite extraer y sintetizar cinco hallazgos principales, que constituyen el fundamento de la propuesta:

1. Concentración y Acoplamiento: La arquitectura monolítica actual concentra responsabilidades, genera un acoplamiento fuerte y obliga a escalar el sistema completo, lo que incrementa el riesgo operativo, el costo de infraestructura y los tiempos de liberación.
2. Mejora de Atributos de Calidad: Los atributos de calidad críticos para la organización (escalabilidad, rendimiento, mantenibilidad y costo operativo) se ven afectados negativamente bajo el modelo monolítico y muestran un potencial de mejora significativo al adoptar principios *serverless*/*FaaS*, gracias al desacoplamiento funcional, el escalamiento granular y la facturación por uso.
3. Proceso Estructurado e Incremental: La migración hacia *serverless* debe concebirse como un proceso estructurado, basado en patrones de modernización como el propuesto *Strangler Fig* en combinación con *DDD* que permitan extraer capacidades funcionales del monolito sin interrumpir la operación. Este enfoque actúa como el elemento metodológico clave para viabilizar la transformación.
4. Criterios de Viabilidad y Exclusión: La migración debe ser regida por un conjunto de métricas de decisión que no solo validen la aptitud técnica del componente (por ejemplo, tolerancia a latencia por *Cold Start* o bajo consumo constante),

sino que también descarten aquellos componentes donde la migración es contraproducente o inviable económica y funcionalmente.

5. Decisión Estratégica Multidimensional: El diagnóstico confirma que la decisión de migrar tiene dimensiones técnicas, operativas y económicas. El marco de migración debe guiar la evaluación conjunta de estas dimensiones para asegurar no solo la adopción de *serverless*, sino su sostenibilidad en el tiempo.

Estos hallazgos proporcionan la base empírica y analítica para la elaboración del siguiente paso de la investigación el cual es la formulación de un marco de migración detallado que permita guiar, priorizar y justificar la decisión de migrar o no migrar las etapas de transformación de una aplicación monolítica.

4.7 Síntesis Conceptual sobre el Manejo del Estado y su Relevancia en la Migración.

El manejo del estado surge como uno de los elementos interpretativos más críticos en la transición desde un monolito hacia *serverless* por lo que se interpreta como un indicador esencial para la clasificación de componentes migrables. La evidencia analizada en la literatura muestra que el éxito de una migración depende de la capacidad de separar estado y lógica, transformando flujos transaccionales largos en funciones más pequeñas e independientes. Según (Manousian, Investigating differences in performance between monolithic and serverless based architectures, 2022), la externalización del estado no solo mejora la escalabilidad, sino que permite que cada función opere bajo los principios de aislamiento y ejecución efímera propios del modelo *FaaS*.

4.8 Integración de Criterios de No-Viabilidad en la Evaluación Arquitectónica.

El análisis documental evidencia que existen escenarios donde la migración directa hacia *serverless* no es recomendable. Los estudios sobre costos (Muhammad Hamza

M. A., 2023) y casos reales en FinTech (Alireza Goli O. H., 2020) coinciden en que cargas constantes, procesos de ejecución prolongada o componentes con dependencias fuertes de infraestructura limitan la eficiencia del modelo FaaS. La interpretación de estos hallazgos permite a esta investigación identificar categorías que deberán incorporarse como filtros dentro del marco de decisión futuro.

4.9 Consideraciones sobre Observabilidad y Operación en Entornos Serverless

Los estudios comparativos señalan que, aunque *serverless* reduce de manera significativa la carga de operación tradicional, introduce nuevos desafíos relacionados con observabilidad, trazabilidad y monitoreo distribuido. (Chen-Fu Fan, 2020) destaca que el análisis de funciones individuales requiere mecanismos centralizados de *logging* y trazas, debido a que cada función constituye una unidad autónoma de ejecución. Esto contrasta con la observabilidad unificada del monolito, donde los flujos se ejecutan dentro de un único proceso.

La interpretación para este estudio sugiere que la observabilidad debe considerarse un atributo de calidad crítico para anticipar la complejidad operativa de la migración.

Glosario

Acoplamiento: Dependencia fuerte entre módulos dentro de un sistema, que dificulta la evolución y el mantenimiento en arquitecturas monolíticas.

Atributos de calidad: Características no funcionales como escalabilidad, rendimiento y mantenibilidad, utilizadas para evaluar una arquitectura.

Autoscaling: Mecanismo que permite ajustar recursos automáticamente según la demanda, es nativo en plataformas serverless.

Bounded Context: Límite lógico dentro de un dominio definido mediante Domain-Driven Design, útil para extraer funcionalidades de un monolito.

Cold Start: Retraso inicial que ocurre cuando una función serverless debe arrancar tras un periodo de inactividad.

DDD (Domain-Driven Design): Enfoque que divide un dominio complejo en contextos delimitados para facilitar el diseño y la modernización del sistema.

Deuda técnica: Consecuencia de decisiones de desarrollo que generan complejidad y dificultan cambios futuros.

Desacoplamiento: Reducción de dependencias entre módulos, permitiendo que cada uno evolucione o falle sin impactar a toda la aplicación.

Elasticidad: Capacidad de un sistema para adaptarse rápidamente a variaciones en la carga de trabajo.

Escalabilidad: Capacidad del sistema para aumentar o reducir su potencia de procesamiento según la demanda.

FaaS (Function as a Service): Modelo serverless donde el proveedor ejecuta funciones bajo demanda sin administrar servidores.

Función serverless: Unidad mínima de ejecución en una arquitectura serverless, activada por eventos y facturada por uso.

Latencia: Tiempo de respuesta de un sistema ante una solicitud.

Microservicios: Arquitectura basada en servicios pequeños e independientes, no la estamos utilizando como fase intermedia en este marco.

Migración incremental: Proceso gradual de modernización que permite la coexistencia temporal de un monolito y sus componentes migrados.

Modelo pay-per-use: Esquema de facturación donde el usuario solo paga por el tiempo real de ejecución y consumo de recursos.

Monolito: Arquitectura donde toda la aplicación está empaquetada en un único artefacto de despliegue.

NoOps: Enfoque operativo donde la infraestructura es completamente gestionada por el proveedor cloud, eliminando tareas DevOps.

Observabilidad distribuida: Conjunto de técnicas de monitoreo, métricas y trazabilidad para sistemas basados en componentes independientes.

Patrón Strangler Fig: Patrón de migración que reemplaza partes del monolito de forma progresiva, redirigiendo tráfico al nuevo componente.

Refactorización: Mejora del código sin alterar su comportamiento externo, para facilitar cambios futuros.

Resiliencia: Capacidad del sistema para recuperarse y operar a pesar de fallos.

Serverless: Paradigma cloud en el que la infraestructura es invisible para el desarrollador y el sistema escala automáticamente bajo demanda.

Sistema legado: Aplicación existente, usualmente monolítica, cuya evolución se ve limitada por su antigüedad y complejidad.

Vendor Lock-in: Dependencia hacia servicios propietarios de un proveedor cloud que dificultan la portabilidad.

Referencias

Alireza Goli, O. H. (2020). *Migrating from Monolithic to Serverless: A FinTech Case Study.* Toronto: https://research.spec.org/. Recuperado el 17 de 09 de 2025, de https://research.spec.org/icpe_proceedings/2020/companion/p20.pdf

Alireza Goli, O. H. (2020). *Migrating from Monolithic to Serverless: A FinTech Case Study.*

AWS. (s.f.). *AWS Free Tier*. Recuperado el 18 de 09 de 2025, de https://aws.amazon.com/free/: https://aws.amazon.com/free/

AWS. (s.f.). *Strangler fig pattern*. Recuperado el 07 de 10 de 2025, de https://docs.aws.amazon.com/: https://docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/strangler-fig.html

Blinowski, G. O. (2022). *Monolithic vs. microservice architecture: A performance and scalability evaluation.* IEEE Access, 10, 20357–20373.

Bustos, S. (s.f.). *Lo que debes saber de una aplicación monolítica Vs. microservicios*. Recuperado el 17 de 09 de 2025, de https://codster.io/blog: https://codster.io/blog/aws-lambda/aplicacion-monolitica-vs-microservicios/

Fowler, M. (2014). *Bounded Context*. Recuperado el 15 de 10 de 2025, de https://martinfowler.com/: https://martinfowler.com/bliki/BoundedContext.html

Gaurav, S. (s.f.). *Serverless vs. Monolithic: Which is Right for You?* Obtenido de https://dev.to/: https://dev.to/imsushant12/serverless-vs-monolithic-which-is-right-for-you-3mg7#:~:text=%2A%20Function,high%20availability%2C%20enhancing%20application%20resilience

Guido Tebes, D. P. (2020). *Proceso para Revisión Sistemática de Literatura y Mapeo.* Recuperado el 17 de 10 de 2025, de https://sedici.unlp.edu.ar/bitstream/handle/10915/135071/Documento.pdf-PDFA.pdf?sequence=1&isAllowed=y

Hammant, P. (2020). *Branch by Abstraction*. Recuperado el 15 de 10 de 2025, de https://trunkbaseddevelopment.com/: https://trunkbaseddevelopment.com/branch-by-abstraction/

Hevner, A. R. (2004). *Design science in information system research.* Recuperado el 17 de 10 de 2025, de https://doi.org/10.2307/25148625

https://www.initiumsoft.com/blog_initium/arquitectura-de-sistema/. (11 de 123 de 2024). *Arquitectura de sistema: estructura y componentes*. Recuperado el 10 de 05 de 2025, de www.initiumsoft.com: https://www.initiumsoft.com/

Juan Mera Menéndez, J. E. (2024). *A comparison between traditional and Serverless.* University of Oviedo. Recuperado el 05 de 10 de 2025, de https://labra.weso.es/pdf/2023_Serverless.pdf

Justice Opara-Martins, R. S. (2016). *Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective.* Journal of Cloud Computing. Recuperado el 10 de 10 de 2025, de https://www.semanticscholar.org/paper/Critical-analysis-of-vendor-lock-in-and-its-impact-Opara-Martins-Sahandi/f2c0babcc12d93f88f4d813297fe03549140ad79

Manousian, J. (2022). *Investigating differences in performance between monolithic and serverless based architectures.* Recuperado el 08 de 10 de 2025, de https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1692921&dswid=4207

Manousian, J. (2022). *Investigating differences in performance between monolithic and serverless based architectures.* Recuperado el 09 de 10 de 2025, de https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1692921&dswid=4207

Muhammad Hamza, M. A. (2023). *Understanding Cost Dynamics of Serverless Computing.* arvix. Recuperado el 09 de 10 de 2025, de https://arxiv.org/pdf/2311.13242

Muhammad Hamza, M. A. (2025). *Navigating Decision-Making in Serverless Migration: A Socio-Technical Grounded Theory Approach.* Recuperado el 10 de 10 de 2025, de

https://papers.ssrn.com/sol3/Delivery.cfm/5161731.pdf?abstractid=5161731&mirid=1

Nakhat Syeda, H. S. (2020). *Analysis of cost-efficiency of serverless approaches.* Recuperado el 08 de 10 de 2025, de https://www.researchgate.net/publication/392514773_Analysis_of_cost-efficiency_of_serverless_approaches

Reyes-Ruiz, L. &. (2020). *Consideraciones para la.* Recuperado el 17 de 10 de 2025, de https://bonga.unisimon.edu.co/server/api/core/bitstreams/da669314-e7cd-4a39-b759-a846187efc5a/content

Reyes-Ruiz, L. &. (2020). *La investigación documental para la.* Recuperado el 17 de 10 de 2025, de https://bonga.unisimon.edu.co/server/api/core/bitstreams/2af35a4b-2abf-4f78-a550-0a4e4764e674/content

Roberto Hernández Sampieri, C. F. (2014). *Metodología de la investigación.* MC Graw Hill.

Saquicela, L. F. (2025). *Diseño de una Arquitectura.* Recuperado el 08 de 10 de 2025, de https://dspace.ups.edu.ec/bitstream/123456789/30657/1/MSQ1040.pdf

Saquicela, L. F. (2025). *Diseño de una Arquitectura en Microservicios con Indexación Dinámica y Distribuida para Optimizar la Búsqueda en Tiempo Real de la Disponibilidad en un Sistema de Reservas Empresarial Coexistente con una Arquitectura Monolítica.* Recuperado el 05 de 10 de 2025, de https://dspace.ups.edu.ec/bitstream/123456789/30657/1/MSQ1040.pdf

Tamushi. (2022). *Atributos de calidad de software: todo lo que necesitas saber.* Recuperado el 10 de 10 de 2025, de https://www.testingit.com.mx/blog/atributos-de-calidad-de-software

Tapia, F. M. (2020). *From monolithic systems to microservices: A comparative study of performance. Applied Sciences.*

Team, E. (s.f.). *Serverless Architecture vs. Monolithic Architecture: What's the Difference?* Recuperado el 08 de 10 de 2025, de https://www.antstack.com/: https://www.antstack.com/blog/serverless-architecture-vs-monolithic-architecture-what-s-the-difference/