Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

DISEÑO DE UNA ARQUITECTURA HÍBRIDA ADAPTANDO MICROSERVICIOS Y SERVERLESS PARA SUPERAR LAS LIMITACIONES DE LA ARQUITECTURA MONOLITICA DEL *SOFTWARE* MAGENTO PARA MEJORAR LA ESCALABILIDAD

José Marcos García Olmino

Asesorado por Mtro. Ing. Juan Pablo Ruiz Guerra

Guatemala, noviembre de 2024

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO DE INVESTIGACIÓN DE UNA ARQUITECTURA HÍBRIDA ADAPTANDO MICROSERVICIOS Y SERVERLESS PARA SUPERAR LAS LIMITACIONES DE LA ARQUITECTURA MONOLITICA DEL *SOFTWARE* MAGENTO PARA MEJORAR LA ESCALABILIDAD

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JOSÉ MARCOS GARCÍA OLMINO
ASESORADO POR MTRO. ING. JUAN PABLO RUIZ GUERRA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, NOVIEMBRE DE 2024

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. José Fernando Gómez Rivera (a.i.)
VOCAL II	Ing. Mario Renato Escobedo Martinez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Ing. Kevin Vladimir Cruz Lorente
VOCAL V	Ing. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. José Francisco Gómez Rivera (a.i.)
EXAMINADOR	Ing. Pedro Pablo Hernández Ramírez
EXAMINADOR	Ing. Oscar Alejandro Paz Campos
EXAMINADOR	Ing. Carlos Alfredo Azurdia Morales
SECRETARIO	Ing. Hugo Huberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO DE UNA ARQUITECTURA HÍBRIDA ADAPTANDO MICROSERVICIOS Y SERVERLESS PARA SUPERAR LAS LIMITACIONES DE LA ARQUITECTURA MONOLITICA DEL *SOFTWARE* MAGENTO PARA MEJORAR LA ESCALABILIDAD

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 21 de septiembre de 2024.

José Marcos García Olmino

EEPFI-PP-5170-2024
Guatemala, 28 de septiembre de 2024

Director
Carlos Gustavo Alonzo
Escuela De Ingenieria En Sistemas
Presente.

Estimado Carlos Gustavo Alonzo

Reciba un cordial saludo de la Escuela de Estudios de Postgrado de la Facultad de Ingeniería.

El propósito de la presente es para informarle que se ha revisado y aprobado el Diseño de Investigación titulado: **DISEÑO DE UNA ARQUITECTURA HÍBRIDA ADAPTANDO MICROSERVICIOS Y SERVERLESS PARA SUPERAR LAS LIMITACIONES DE LA ARQUITECTURA MONOLITICA DEL SOFTWARE MAGENTO PARA MEJORAR LA ESCALABILIDAD** , el cual se enmarca en la línea de investigación: **Área de Investigación - Sistemas para impulsar la integración de sistemas de información**, presentado por el estudiante **José Marcos García Olmino** carné número **201903895**, quien optó por la modalidad del "PROCESO DE GRADUACIÓN DE LOS ESTUDIANTES DE LA FACULTAD DE INGENIERÍA OPCIÓN ESTUDIOS DE POSTGRADO". Previo a culminar sus estudios en la Maestría en Artes en Tecnologias De La Inf. Y La Comunicacion.

Y habiendo cumplido y aprobado con los requisitos establecidos en el normativo de este Proceso de Graduación en el Punto 6.2, aprobado por la Junta Directiva de la Facultad de Ingeniería en el Punto Décimo, Inciso 10.2 del Acta 28-2011 de fecha 19 de septiembre de 2011, firmo y sello la presente para el trámite correspondiente de graduación de Pregrado.

Atentamente,

"Id y Enseñad a Todos"

Juan Pablo Ruiz Guerra
INGENIERO EN CIENCIAS Y SISTEMAS
COLEGIADO No. 18,545

Mtro. Juan Pablo Ruiz Guerra
Asesor(a)

UNIVERSIDAD DE SAN CARLOS
ESCUELA DE POSTGRADO
FACULTAD DE INGENIERIA
DE GUATEMALA

Mtro. Marion Antonio Perez Turk
Coordinador(a) de Maestría

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECTORA
POSTGRADO

Mtra. Aurelia Anabela Cordova Estrada
Directora
Escuela de Estudios de Postgrado
Facultad de Ingeniería

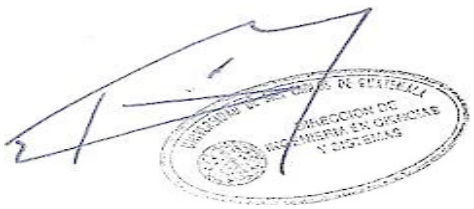
UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

EEP-EICS-5032-2024

El Director de la Escuela De Ingenieria En Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el visto bueno del Coordinador y Director de la Escuela de Estudios de Postgrado, del Diseño de Investigación en la modalidad Estudios de Pregrado y Postgrado titulado: **DISEÑO DE UNA ARQUITECTURA HÍBRIDA ADAPTANDO MICROSERVICIOS Y SERVERLESS PARA SUPERAR LAS LIMITACIONES DE LA ARQUITECTURA MONOLITICA DEL SOFTWARE MAGENTO PARA MEJORAR LA ESCALABILIDAD** , presentado por el estudiante universitario **José Marcos García Olmino**, procedo con el Aval del mismo, ya que cumple con los requisitos normados por la Facultad de Ingeniería en esta modalidad.

ID Y ENSEÑAD A TODOS



Mtro. Carlos Gustavo Alonzo
Director
Escuela De Ingenieria En Sistemas

Guatemala, septiembre de 2024

Decanato
Facultad e Ingeniería

24189101- 24189102

LNG.DECANATO.OIE.775.2024

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **DISEÑO DE UNA ARQUITECTURA HÍBRIDA ADAPTANDO MICROSERVICIOS Y SERVERLESS PARA SUPERAR LAS LIMITACIONES DE LA ARQUITECTURA MONOLITICA DEL SOFTWARE MAGENTO PARA MEJORAR LA ESCALABILIDAD** , presentado por: **José Marcos García Olmino** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Firmado electrónicamente por: José Francisco Gómez Rivera
Motivo: Informe final PREGRADOPOSTGRADO
Fecha: 25/11/2024 19:01:29
Lugar: Facultad de Ingeniería, USAC.

Ing. José Francisco Gómez Rivera
Decano a.i.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO a.i.
Facultad de Ingeniería

Guatemala, noviembre de 2024

ACTO QUE DEDICO A:

Dios	Porque siempre estoy agradecido por las oportunidades que me ha permitido tener
Mis padres	Por el apoyo incondicional que me han brindado, sin ellos este logro no lo hubiera podido alcanzar
Mi hermano	Porque estuvo presente cuando quería algún consejo y necesite apoyo
Mis amigos	Porque empezamos como compañeros de clase, pero terminamos con una amistad sincera apoyándonos a lo largo de la carrera

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser la casa de estudios que facilita el acceso a la educación superior
M.A Ing. Juan Pablo Ruiz Guerra	Por brindarme su tiempo y conocimiento durante esta investigación

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

1. INTRODUCCIÓN

En la era digital, la necesidad de que las aplicaciones sean altamente escalables, eficientes y adaptables ha impulsado la evolución de las arquitecturas de software. Las arquitecturas monolíticas, que alguna vez fueron la solución dominante en el desarrollo de aplicaciones empresariales, están mostrando sus limitaciones frente a la demanda de mayor flexibilidad y capacidad de respuesta en entornos cambiantes. Esto ha llevado a la adopción de enfoques más modernos, como los microservicios y las tecnologías *serverless*, que permiten una mayor escalabilidad, optimización de recursos y reducción de costos operacionales.

En este contexto, Magento 2.0, una plataforma líder en comercio electrónico sigue utilizando en gran medida una arquitectura monolítica que presenta desafíos en términos de escalabilidad y flexibilidad operativa. Este estudio tiene como objetivo desarrollar un prototipo que permita migrar la arquitectura monolítica de Magento hacia una arquitectura híbrida que combine microservicios y tecnologías *serverless*. La arquitectura híbrida se basa en descomponer aplicaciones complejas en componentes independientes que pueden escalarse de manera autónoma, y aprovechar tecnologías *serverless* para gestionar funciones que solo consumen recursos cuando es necesario.

Para lograr estos objetivos, se seleccionarán módulos clave de Magento que serán migrados a una arquitectura híbrida. Estos módulos se analizarán en función de su impacto en el rendimiento del sistema y su capacidad para manejar altos volúmenes de transacciones. Mediante pruebas exhaustivas de rendimiento y escalabilidad, se evaluarán aspectos como tiempos de respuesta, latencia y

costos operacionales, comparando la arquitectura monolítica original con el prototipo híbrido.

El estudio se estructurará en varios capítulos que cubrirán las diferentes etapas del estudio. El primer capítulo abordará los antecedentes de la arquitectura monolítica donde se habla acerca de migraciones de arquitecturas monolíticas hacia microservicios o *serverless* y sobre la plataforma Magento. El segundo capítulo se dedicará a la justificación del estudio, donde se argumentará la importancia de migrar a una arquitectura híbrida, destacando los beneficios esperados en términos de escalabilidad y eficiencia operativa.

En el tercer capítulo, se definirán los alcances del estudio, detallando los alcances investigativos, técnicos y los resultados que se esperan obtener. Este capítulo también establecerá los límites del estudio y las áreas que no serán cubiertas. El cuarto capítulo presentará el marco teórico, proporcionando una revisión exhaustiva de las arquitecturas monolíticas, microservicios y *serverless*, y apoyándose en estudios previos que respaldan la viabilidad de esta migración.

El quinto capítulo, titulado Implementación del prototipo, se centrará en la selección y codificación de los módulos de Magento que serán migrados, detallando el diseño de la arquitectura del prototipo, las herramientas tecnologías utilizadas y pruebas de integración del prototipo. En el sexto capítulo, se expondrán los resultados de las pruebas de integración, pruebas de rendimiento y la comparación entre la arquitectura monolítica y el prototipo. Finalmente, el séptimo capítulo estará dedicado a la discusión de los resultados, donde se interpretarán los datos obtenidos, se evaluarán los desafíos técnicos encontrados y se destacarán las mejoras logradas en comparación con la arquitectura monolítica original.

2. ANTECEDENTES

En el estudio realizado por Lin (2015) sobre Magento, se detalla cómo esta plataforma de comercio electrónico basada en PHP y MySQL se caracteriza por su arquitectura modular, la complejidad inherente de esta estructura sugiere desafíos típicos de las arquitecturas monolíticas, especialmente en términos de escalabilidad y mantenimiento. Adicionalmente, el documento identifica áreas susceptibles de mejora, como la alta demanda de recursos del servidor. Estos factores subrayan la necesidad de optimizaciones que puedan mejorar el rendimiento y la usabilidad para mantener la competitividad de Magento en el dinámico mercado de e-commerce.

El estudio llevado a cabo por Velepucha *et al.* (2018) examina exhaustivamente las deficiencias de las arquitecturas monolíticas, particularmente en contextos en los que la escalabilidad y la flexibilidad son esenciales. Este estudio presenta el modelo MOMMIV, que se basa en el principio de ocultación de información para facilitar la descomposición de arquitecturas monolíticas en arquitecturas de microservicios. Este modelo aborda los desafíos comunes de las arquitecturas monolíticas, como la dificultad en extraer la lógica de negocio por su naturaleza integrada y los problemas asociados con bases de datos relacionales que no soportan la escalabilidad horizontal.

Además, Velepucha *et al.* (2018) destacan cómo la transición a microservicios puede aumentar la complejidad en la integración y gestión de servicios, pero con el adecuado diseño de descomposición, estas dificultades pueden minimizarse significativamente. La metodología propuesta ofrece una

estructura que no solo supera las restricciones físicas y lógicas de las arquitecturas monolíticas, sino que también aumenta la capacidad de las empresas para adaptarse rápidamente a los cambios del mercado, proporcionando actualizaciones más ágiles y menos disruptivas.

En el estudio realizado por Goli *et al.* (2020) se destaca cómo la adopción de soluciones *serverless* puede mejorar significativamente el rendimiento y optimizar los costos operativos en sectores como el *FinTech*. Este enfoque elimina la necesidad de gestionar la infraestructura, permitiendo que los desarrolladores se concentren en el código de la aplicación. Al hacerlo, *serverless* no solo simplifica la gestión de recursos, sino que también acelera el desarrollo y despliegue de nuevas funcionalidades, reduciendo el tiempo y el costo asociados con las operaciones tradicionales.

Además de los beneficios ya mencionados de la adopción de soluciones *serverless* por Goli *et al.* (2020), también resaltan cómo la implementación de arquitecturas *serverless* en el sector *FinTech* no solo reduce los costos operativos sino que también mejora significativamente la velocidad de procesamiento de solicitudes. Usaron casos prácticos, como el de Yubl, para demostrar cómo la adopción de un despliegue *serverless* puede acortar los plazos de entrega de funciones y el tiempo de llegada al mercado. Adzic y colaboradores (2017) quienes se pusieron en contacto con Yubl mencionan que de hacer 4 a 6 despliegues productivos por mes pasaron a 80 o más por mes contando siempre con el mismo equipo de ingenieros (6 ingenieros), lo cual demuestra lo fundamental que es esta solución para mantener la competitividad en el dinámico mercado tecnológico

Por otro lado, Allen (2023) aborda las ventajas de las arquitecturas de microservicios sobre las arquitecturas monolíticas y menciona el uso de patrones

específicos como Event Sourcing y CQRS (Command Query Responsibility Segregation) para optimizar las operaciones en arquitecturas de microservicios, donde Event Sourcing asegura que todos los cambios en el estado de la aplicación se almacenen como una secuencia de eventos que, no solo pueden ser consultados, sino también usados para recrear estados pasados del sistema y CQRS, por otro lado, separa las operaciones de lectura y escritura en modelos distintos, lo que facilita la escalabilidad y la optimización del rendimiento al permitir que estas operaciones se escalen de manera independiente.

Además, Allen (2023) profundiza en cómo las arquitecturas de microservicios, a través de patrones como Event Sourcing y CQRS, no solo mejoran la separación de responsabilidades y la gestión de datos, sino que también permiten una implementación y escalamiento más dinámicos de los servicios. Allen subraya que, en contraste con las arquitecturas monolíticas, los microservicios permiten una reducción considerable en los tiempos de inactividad durante las actualizaciones y una gestión de errores más eficaz.

Lima (2019) menciona que, en la exploración de la migración de sistemas monolíticos a arquitecturas basadas en microservicios, tecnologías específicas como Docker y Kubernetes desempeñan un papel fundamental en facilitar este proceso. Docker permite encapsular aplicaciones en contenedores, proporcionando un entorno de ejecución consistente que elimina muchos problemas comunes de "funciona en mi máquina", mientras que Kubernetes ofrece una plataforma robusta para la orquestación de estos contenedores, asegurando que se gestionen eficientemente en producción.

El estudio de Lima (2019) también subraya que estas tecnologías no solo simplifican el despliegue y la gestión de microservicios, sino que también mejoran significativamente la capacidad de escalar y mantener sistemas complejos. Esta

transición resulta en una notable reducción en la complejidad operativa y los costos, ya que las organizaciones pueden escalar servicios de manera independiente y eficiente, adaptándose mejor a las demandas cambiantes sin la necesidad de gestionar una arquitectura monolítica grande y rígida.

Finalmente, en el estudio de Hasan *et al.* (2023) en el contexto de la transición de arquitecturas monolíticas a microservicios, destacan la importancia de utilizar métricas estructurales para asegurar la calidad y mantenibilidad de la arquitectura. El artículo propone métricas específicas como el acoplamiento, la complejidad, y la cohesión, que son vitales para evaluar la mantenibilidad durante la migración a entornos en la nube. Este enfoque metodológico no solo facilita la medición de la calidad arquitectónica, sino que también guía a las organizaciones para realizar transiciones más efectivas y sostenibles, minimizando la deuda técnica y maximizando los beneficios de los microservicios.

Estas investigaciones acumulan hallazgos importantes que demuestran claramente que la evolución hacia arquitecturas distribuidas no es solo una mejora técnica, sino una necesidad estratégica para mantener la competitividad en el panorama tecnológico en constante evolución. Al desglosar las aplicaciones en estructuras más flexibles y resilientes, las empresas no solo optimizan sus operaciones internas, sino que también mejoran su capacidad para adaptarse rápidamente a cambios en el mercado y a las exigencias de los consumidores.

3. PLANTEAMIENTO DEL PROBLEMA

3.1. Planteamiento del problema

En la era digital actual, la elección de la arquitectura de software es fundamental para el éxito empresarial a largo plazo. Las empresas se enfrentan al desafío de desarrollar aplicaciones que no solo cumplan con los requisitos funcionales actuales, sino que también sean ágiles y escalables para adaptarse a las necesidades futuras. En este escenario, muchas organizaciones han optado tradicionalmente por arquitecturas monolíticas debido a su aparente simplicidad y coherencia en las etapas iniciales de desarrollo.

Las arquitecturas monolíticas enfrentan problemas significativos de escalabilidad y costos operativos a medida que las aplicaciones crecen. Este tipo de arquitectura, que concentra todas las funcionalidades en un solo proceso o servicio, requiere que cada ajuste o mejora implique un despliegue completo de la aplicación, incrementando el riesgo de errores y llevando a tiempos de inactividad que afectan negativamente la experiencia del usuario y la competitividad de la empresa.

Además, las arquitecturas monolíticas pueden llevar a una pérdida de competitividad debido a su rigidez en la adaptación tecnológica. La inflexibilidad para adoptar nuevas tecnologías o *frameworks* sin reescribir aplicaciones completas limita la capacidad de las empresas para responder eficazmente a las demandas cambiantes del mercado y aprovechar nuevas oportunidades de negocio. Estos factores pueden desencadenar un estancamiento en el mercado, donde las empresas no logran explotar plenamente el potencial de la innovación

debido a las limitaciones impuestas por una infraestructura de TI inflexible y costosa.

Este enfoque tradicional subraya la necesidad crítica de evaluar alternativas que permitan actualizaciones y mejoras continuas sin comprometer la integridad del sistema en su totalidad, lo que es esencial para mantener y mejorar la eficiencia operativa y la capacidad de innovación en un entorno empresarial dinámico. En este sentido, es imperativo explorar arquitecturas de software que no solo faciliten la integración de nuevas tecnologías y metodologías, sino que también promuevan una mayor flexibilidad y modularidad.

3.1.1. Pregunta central

¿Cómo diseñar una arquitectura que combine *serverless* y microservicios para aumentar la escalabilidad en comparación con la arquitectura monolítica tradicional de Magento?

3.1.2. Preguntas Auxiliares

- ¿Cuáles son los indicadores clave que demuestran que las arquitecturas de microservicios y *serverless* pueden mejorar la escalabilidad en comparación con las arquitecturas monolíticas tradicionales?
- ¿Cuáles son los desafíos y limitaciones técnicas que enfrentan las empresas al intentar integrar arquitecturas *serverless* y de microservicios en sistemas existentes dominados por estructuras monolíticas?
- ¿De qué manera la inflexibilidad tecnológica de las arquitecturas monolíticas afecta la capacidad de innovación y adaptación de las empresas ante cambios en el mercado?

4. JUSTIFICACIÓN

En la búsqueda de métodos para impulsar la integración de sistemas de información efectiva y eficiente, esta investigación se inscribe en la línea de Sistemas para impulsar la integración de sistemas de información, un campo de mayor relevancia en el ámbito tecnológico empresarial. La integración de sistemas es fundamental para el aprovechamiento de datos y recursos tecnológicos, y la elección de una arquitectura de *software* adecuada es crucial para alcanzar este objetivo. En este contexto, se utilizará Magento, una plataforma de comercio electrónico conocida por su amplia adopción y estructura modular, como caso de estudio para explorar la transición de su arquitectura monolítica tradicional hacia una más desacoplada mediante microservicios y *serverless*.

En el sector tecnológico actual, las empresas enfrentan desafíos constantes para mantenerse competitivas y eficientes. En este contexto, la elección de la arquitectura de software juega un papel crucial en la operatividad y escalabilidad de las soluciones empresariales. Tradicionalmente, muchas empresas han adoptado arquitecturas monolíticas por su simplicidad inicial y coherencia operacional. Sin embargo, este tipo de arquitectura a menudo resulta en ineficiencias significativas en costos operativos y limitaciones en la escalabilidad, lo que puede obstaculizar la adaptabilidad y el crecimiento empresarial a largo plazo.

Esta investigación se propone explorar cómo las arquitecturas que combinan *serverless* y microservicios pueden ofrecer alternativas viables y superiores en términos de costos operacionales y escalabilidad en comparación

con las arquitecturas monolíticas tradicionales. Al hacerlo, el estudio buscará proporcionar una base sólida para la toma de decisiones en el desarrollo y gestión de software, subrayando cómo las arquitecturas más modernas y flexibles pueden contribuir significativamente a la eficiencia operativa y a la capacidad de adaptación al cambio.

Asimismo, este estudio permitirá entender mejor las implicaciones económicas y técnicas de las transiciones arquitectónicas en ambientes empresariales, facilitando a las organizaciones la evaluación de sus infraestructuras actuales y la planificación de mejoras futuras. Al proporcionar esta comprensión, la investigación no solo ayudará a resolver conflictos operativos y estratégicos actuales, sino que también establecerá un marco para el aprovechamiento de tecnologías emergentes y la innovación continua.

Además, la transición hacia arquitecturas más modernas como microservicios y *serverless* representa una evolución crítica en la forma en que las organizaciones desarrollan y despliegan sus aplicaciones. Este cambio no solo promete mejoras en la agilidad y la eficiencia, sino que también introduce nuevas formas de gestionar la seguridad y la continuidad del servicio, aspectos que son vitales en un entorno empresarial cada vez más dependiente de tecnologías digitales robustas y seguras.

Explorar estas arquitecturas en profundidad proporcionará *insights* valiosos sobre las mejores prácticas y estrategias para mitigar riesgos asociados con la implementación y el mantenimiento de sistemas complejos en la nube. Al entender estas dinámicas, las empresas podrán anticipar mejor los retos que conlleva la modernización de su infraestructura tecnológica y estarán mejor equipadas para gestionar el cambio de manera proactiva y efectiva.

5. OBJETIVOS

5.1. General

Diseñar una arquitectura que combine *serverless* y microservicios para aumentar la escalabilidad de Magento, en comparación con su arquitectura monolítica tradicional.

5.2. Específicos

- Identificar los indicadores clave que demuestran que las arquitecturas de microservicios y *serverless* pueden mejorar la escalabilidad.

- Diseñar y evaluar una arquitectura que combine elementos de *serverless* y microservicios, para identificar y superar los desafíos técnicos que enfrentan las empresas al integrar estas tecnologías en sistemas existentes.

- Implementar un prototipo de arquitectura combinando *serverless* y microservicios para analizar cómo la flexibilidad de esta integración puede mejorar la capacidad de innovación y adaptación en entornos empresariales frente a las limitaciones de las arquitecturas monolíticas.

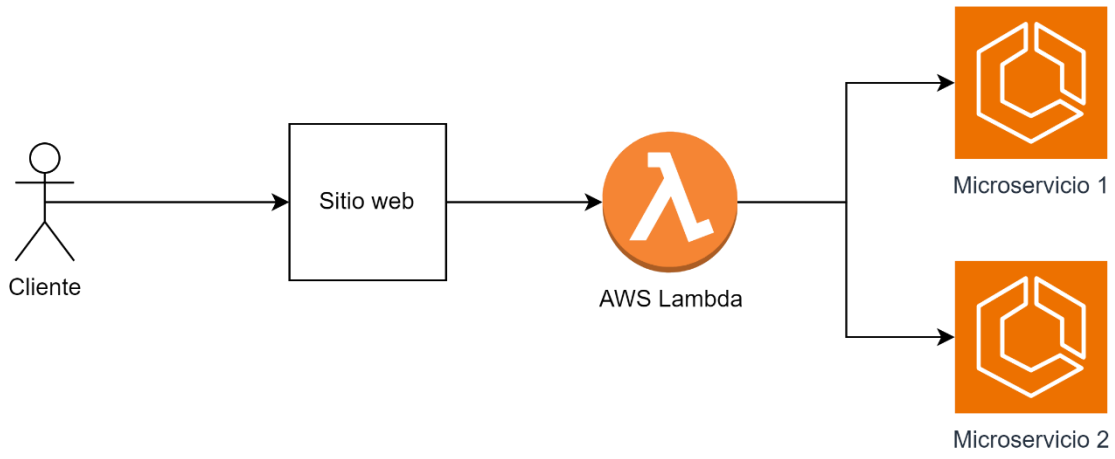
6. NECESIDADES A CUBRIR Y ESQUEMA DE SOLUCIÓN

Para enfrentar los desafíos asociados con la ineficiencia en costos y la falta de escalabilidad de las arquitecturas monolíticas en aplicaciones empresariales, se propone una arquitectura híbrida que integra *serverless* y microservicios. Este enfoque se centra en superar las limitaciones de las arquitecturas tradicionales, aprovechando la flexibilidad operacional y la reducción de costos.

El desarrollo de esta arquitectura híbrida se basará en la combinación de microservicios independientes y funciones *serverless*, lo cual permite una gestión más eficiente y escalable de los recursos. Los microservicios se encargarán de manejar las operaciones core de la aplicación, permitiendo una mejor distribución de carga y facilitando actualizaciones más ágiles y menos disruptivas. Por otro lado, las funciones *serverless* se utilizarán para tareas específicas que requieran escalabilidad instantánea y gestión de alta demanda sin la necesidad de mantener recursos constantemente activos.

Figura 1.

Arquitectura híbrida con serverless y microservicios



Nota. Este enfoque arquitectónico permite desplegar servicios tradicionales consumiendo recursos mínimos en el sistema operativo que se alojan. Elaboración propia, realizado con draw.io.

En la Figura 1, se presenta un diagrama con una representación visual que ayuda a conceptualizar cómo estructurar una solución de este tipo, y una representación inicial que puede adaptarse y refinarse según las necesidades del entorno empresarial. Sirve como guía para el diseño preliminar y como punto de partida para discusiones futuras y ajustes que podrían ser necesarios según evoluciona el entendimiento de las demandas operativas y tecnológicas.

En la Figura 1 se muestra cómo las solicitudes desde un sitio web son dirigidas inicialmente a AWS Lambda (servicio *serverless*), que actúa como un procesador ligero y eficiente, ideal para ejecutar operaciones que no requieren un estado persistente.

Lambda puede resolver la solicitud internamente o redirigirla a uno de los microservicios para su procesamiento. Este enfoque modular asegura que las operaciones intensivas en recursos sean manejadas por componentes dedicados, optimizando así el rendimiento y la escalabilidad.

Los microservicios manejan funciones específicas dentro de la aplicación de manera independiente, permitiendo que cada aspecto del sistema sea escalable y fácil de mantener. Este modelo híbrido aprovecha la eficiencia de AWS Lambda para reducir los costos operativos mientras utiliza la robustez y escalabilidad de los microservicios para garantizar una gestión eficiente y efectiva de las operaciones más críticas.

El diseño propuesto será evaluado a través de pruebas rigurosas para validar su eficiencia operativa y escalabilidad, asegurando que la arquitectura no solo es técnicamente viable, sino que también apoya eficazmente las operaciones empresariales en un entorno real. Adoptando esta arquitectura, las empresas no solo mantendrán su competitividad, sino que también liderarán en innovación y eficiencia operacional.

7. ALCANCES

7.1. Alcances de investigación

Este proyecto de investigación se centrará en la descomposición de la plataforma Magento en su versión 2.0, una herramienta líder en el ámbito del comercio electrónico, examinando específicamente cómo su arquitectura monolítica tradicional puede ser transformada utilizando enfoques basados en microservicios y tecnologías *serverless*. El estudio se concentrará en la descomposición de módulos clave, adaptándolos a arquitecturas de microservicios y *serverless* para mejorar su escalabilidad y eficiencia operativa. Este enfoque permitirá una evaluación detallada de cómo la implementación de estas tecnologías modernas puede ofrecer beneficios tangibles en términos de rendimiento, costos y capacidad de respuesta ante fluctuaciones del mercado.

Además, se llevará a cabo una comparación sistemática entre las configuraciones monolíticas originales y las nuevas implementaciones, proporcionando un análisis profundo de las mejoras en escalabilidad y eficiencia que estos cambios arquitectónicos representan para Magento en un entorno comercial dinámico.

7.2. Alcances técnicos

Este proyecto se centrará en desarrollar un prototipo funcional que transformará ciertos componentes de la plataforma Magento de una configuración monolítica a una arquitectura optimizada usando microservicios y *serverless*. La selección de módulos para esta conversión se basará en un

análisis detallado que identificará aquellos elementos de la plataforma que más se beneficiarían de esta modernización, teniendo en cuenta su impacto en la escalabilidad y eficiencia operativa.

Se utilizarán las tecnologías adecuadas para el desarrollo de microservicios y la implementación de soluciones *serverless*. Los microservicios serán encapsulados utilizando Docker y AWS ECS. Paralelamente, las funciones *serverless* se implementarán mediante AWS Lambda, lo que permitirá una gestión eficiente del procesamiento de demandas sin la necesidad de recursos de servidor permanentemente activos. Las tecnologías AWS Lambda y AWS ECS son servicios de nube ofrecidos por la plataforma *Amazon Web Services*

Este enfoque de desarrollo está diseñado para confirmar los beneficios potenciales de una arquitectura descompuesta, con un foco particular en la mejora de la gestión de recursos. La meta es demostrar cómo la integración de microservicios y *serverless* puede proporcionar una base más flexible y escalable para operaciones de comercio como en el caso de Magento.

Se delimitará el alcance del prototipo a funcionalidades específicas de los módulos seleccionados, y no se garantiza una cobertura completa de todas las funcionalidades de Magento.

7.3. Alcances de resultados

- Prototipo funcional que implemente módulos seleccionados de Magento en arquitecturas de microservicios y *serverless*.

- Reportes sobre las evaluaciones realizadas al prototipo y al *software* original, incluyendo tiempos de respuesta, capacidad de manejo de cargas y análisis de costos.

- Documentación completa y manuales de operación que describen en detalle las modificaciones implementadas y cómo gestionarlas.

- Análisis de la viabilidad de esta migración, con recomendaciones estratégicas basadas en los beneficios observados y los desafíos enfrentados durante la implementación.

8. MARCO TEÓRICO

8.1. Arquitecturas de *software*

La arquitectura de *software* se refiere al marco organizacional empleado para diseñar, desarrollar y desplegar aplicaciones de *software*. Estas se clasifican generalmente en diferentes tipos, basados en cómo se estructuran y se integran los componentes del sistema. Cada tipo de arquitectura ofrece distintos beneficios y enfrenta diferentes desafíos, influenciados por factores como los requisitos del negocio, la naturaleza del proyecto, la escalabilidad deseada, y las preferencias tecnológicas.

8.1.1. Arquitecturas monolíticas

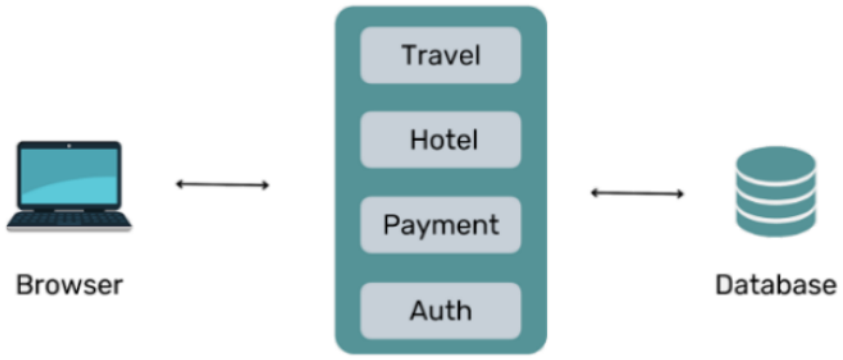
Las arquitecturas monolíticas representan uno de los enfoques de diseño de *software* más antiguos y tradicionales. En estas arquitecturas, una aplicación se construye como una entidad única e indivisible donde todos los componentes de *software* están interconectados y dependientes entre sí (Elgheriani & Ahmed 2022). Estas arquitecturas centralizadas agrupan todas las funcionalidades en un único proceso de ejecución, lo que simplifica las pruebas y el despliegue inicial, pero puede dificultar la escalabilidad y el mantenimiento a medida que la aplicación aumenta en tamaño y complejidad.

La figura 2 ilustra un ejemplo de arquitectura monolítica, donde múltiples módulos funcionales como la gestión de viajes, reservas de hoteles, procesamiento de pagos y autenticación de usuarios están integrados en una

única aplicación. En este tipo de arquitectura, los componentes están altamente acoplados, compartiendo tanto la base de datos como el entorno de ejecución.

Figura 2.

Arquitectura monolítica



Nota. Explica cómo funciona la arquitectura monolítica con bloque. Adaptado de S. Elgheriani & N. Ahmed. Microservices vs. Monolithic Architectures. *International Journal of Applied Sciences and Technology, 3*(6), p. 508. https://www.minarjournal.com/dergi/microservices-vs-monolithic-architectures-the-differential-structure-between-two-architectures20221202031410.pdf

8.1.1.1. Características de las arquitecturas monolíticas

- Todos los componentes de la aplicación, incluyendo la lógica de negocio, la interfaz de usuario, y la gestión de datos, están fuertemente acoplados dentro de un único proceso ejecutable.
- Debido a que la aplicación es un único bloque construido, el despliegue suele ser más directo, generalmente requiriendo la gestión de un solo archivo o directorio.

- Las aplicaciones monolíticas típicamente escalan mediante el escalado vertical, lo que significa aumentar la capacidad del servidor donde se aloja la aplicación.
- Las arquitecturas monolíticas suelen estar limitadas por la tecnología en la que se desarrollan inicialmente. Cualquier actualización o cambio requiere una revisión de todo el sistema, lo que puede ser tanto arriesgado como laborioso.
- Conforme la aplicación aumenta en tamaño y complejidad, gestionar, actualizar y escalarla puede volverse progresivamente más complicado, afectando potencialmente el funcionamiento de toda la plataforma.

8.1.1.2. Ventajas y desventajas en el contexto empresarial

En el contexto empresarial, las arquitecturas monolíticas ofrecen tanto beneficios claros como desafíos significativos (Elgheriani & Ahmed, 2022). A continuación, se exploran las ventajas y desventajas más relevantes de adoptar una arquitectura monolítica para aplicaciones empresariales.

- Ventajas
    - La arquitectura monolítica permite una mayor simplicidad en el desarrollo y el despliegue debido a su estructura integrada. Esto facilita la gestión ya que todas las operaciones se manejan dentro de un único proceso, lo que puede ser particularmente beneficioso para las empresas con infraestructuras menos complejas o con menos recursos para gestionar sistemas distribuidos.

    - Dado que todos los componentes de una aplicación monolítica se ejecutan dentro de un mismo proceso, no se requieren

comunicaciones entre procesos, lo que puede resultar en menor latencia y un rendimiento más rápido en comparación con arquitecturas distribuidas, como los microservicios.

- Estas aplicaciones pueden facilitar las pruebas y la depuración, dado que no requieren la configuración de múltiples servicios ni la administración de su comunicación. Esta ventaja resulta fundamental en las etapas de desarrollo y mantenimiento, ya que disminuye los recursos requeridos y el tiempo para dichas actividades.

- Desventajas

  - La escalabilidad representa una limitación importante en las arquitecturas monolíticas, ya que aumentar la capacidad de la aplicación implica hacerlo a todo el sistema, lo cual puede ser menos eficiente y más costoso que escalar componentes individuales en una arquitectura de microservicios.

  - Con el crecimiento de la aplicación, su mantenimiento y actualización pueden tornarse más complicados y arriesgados. Cualquier cambio en el sistema puede demandar un despliegue completo, elevando el riesgo de fallos y de interrupciones en el servicio.

  - Las aplicaciones monolíticas suelen estar vinculadas a un único *stack* tecnológico, lo que puede dificultar y encarecer la adopción de nuevos *frameworks* o tecnologías. Esta rigidez puede impedir la innovación y la adaptabilidad a nuevas tendencias del mercado.

Aunque existen ventajas, el modelo monolítico pierde por las limitaciones de metodologías modernas de entrega de *software*. En un entorno empresarial que exige agilidad y flexibilidad, las limitaciones en escalabilidad y dificultad para adoptar nuevas tecnologías pueden representar desventajas significativas (Elgheriani & Ahmed, 2022).

8.1.2. Arquitecturas basadas en microservicios

El enfoque de microservicios es una estrategia de diseño de aplicaciones que divide el sistema en componentes independientes, con cada servicio asignado a una función específica. Estos servicios operan en entornos distribuidos, facilitando la escalabilidad y la adaptabilidad. Esta modularidad permite actualizaciones individuales sin afectar al sistema global y apoya la utilización de tecnologías diversas para cada servicio, optimizando así la eficiencia y flexibilidad del desarrollo.

8.1.2.1. Fundamentos teóricos de los microservicios

Los microservicios se fundamentan en el principio de descomposición de aplicaciones grandes en unidades más pequeñas, independientes y manejables, que se comunican entre sí mediante interfaces ligeras, comúnmente APIs RESTful. Esta descomposición facilita la modularidad y mejora la mantenibilidad del *software*. Esta característica permite desarrollar, probar, desplegar y escalar cada servicio de forma independiente, lo cual es un cambio significativo respecto a las monolíticas y SOA tradicionales.

En la figura 3 se puede observar cómo se puede descomponer un sistema grande en varias unidades más pequeñas con funcionalidades diferentes.

Figura 3.

Ejemplo de una descomposición

Nota. Descomposición orientada a objetos. Adaptado de V. Velepucha y Flores (2023). A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. *IEEE Access, 11.* p. 88341. (https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10220070).

Los microservicios operan bajo la teoría de que cada servicio debe ser autónomo, gestionando su propio ciclo de vida y dependencias. Esto minimiza las dependencias complejas entre componentes y facilita la escalabilidad horizontal de los sistemas. Esta autonomía es esencial para aplicaciones empresariales que requieren alta disponibilidad y flexibilidad para adaptarse a cambios rápidos en el mercado o en la tecnología (Garlan & Shaw 2020).

Los microservicios en lugar de seguir un modelo de gobernanza centralizada típico de arquitecturas monolíticas, los microservicios adoptan un enfoque descentralizado. Esto ofrece a los equipos de desarrollo la flexibilidad de seleccionar las herramientas y tecnologías que mejor se ajusten a sus necesidades, promoviendo la innovación y la capacidad de adaptación. Este concepto es esencial para preservar la agilidad y eficacia en el desarrollo y despliegue de nuevos servicios.

8.1.2.2. Patrones de diseño en microservicios

Los patrones de diseño en microservicios son esenciales para abordar los retos comunes en el desarrollo y la gestión de aplicaciones distribuidas. Estos patrones proporcionan soluciones probadas para problemas específicos que surgen al descomponer una aplicación en servicios más pequeños, gestionar su interacción, y asegurar su despliegue y operación eficientes (Taibi *et al.*, 2018).

8.1.2.2.1. Patrones de orquestación y coordinación

Estos patrones juegan un papel clave en la interacción de los componentes y el manejo de datos dentro de los sistemas basados en microservicios Incluyen mecanismos para la coordinación y la orquestación de servicios, esenciales para mantener la funcionalidad y eficiencia del sistema (Taibi *et al.*, 2018).

- Api gateway:

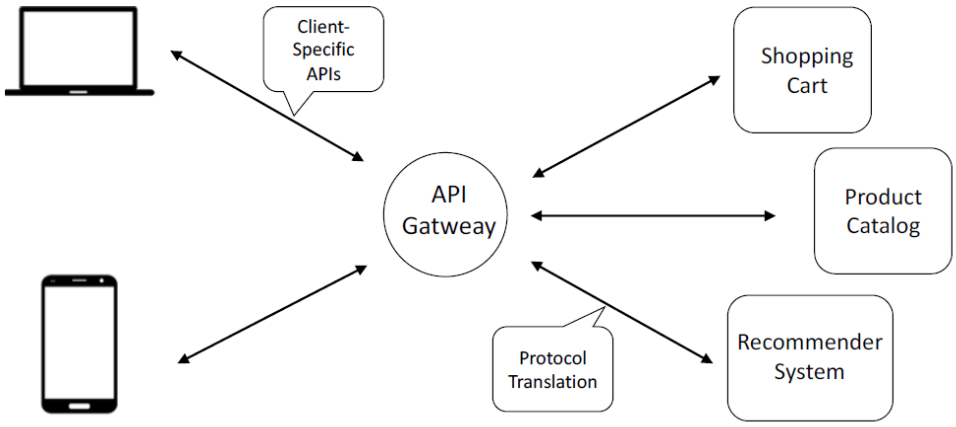
En este patrón, el api gateway actúa como un intermediario que maneja las solicitudes de los clientes, dirigiéndose a los microservicios apropiados. Este patrón es esencial para manejar la seguridad, la tasa de límites y las transformaciones de protocolos necesarias en un entorno distribuido.

Puede encargarse de varias tareas como autenticación, monitoreo y manejo de respuestas estáticas, simplificando así las interacciones y aumentando el rendimiento del sistema al reducir el número de solicitudes que cada cliente necesita realizar.

En la figura 4 se muestra un ejemplo típico de una arquitectura de microservicios, donde un *API Gateway* actúa como un punto central de acceso para los clientes que interactúan con varios servicios especializados. En este esquema, el *API Gateway* maneja las solicitudes de los clientes y las enruta a los microservicios correspondientes.

Figura 4.

Patrón de arquitectura Api Gateway



Nota. Catálogo de patrones. Obtenido de D. Taibi, Lenarduzzi, V. & C. Pahl. Architectural Patterns for Microservices: A Systematic Mapping Study. *8th International Conference on Cloud Computing and Services Science,* p. 3. (10.5220/0006798302210232).

8.1.2.2.2. Patrones de Descubrimiento de Servicios

Estos patrones son clave para manejar múltiples instancias de microservicios que pueden correr en diferentes contenedores virtualizados o VMs. La comunicación entre ellos necesita definirse dinámicamente, y los

clientes deben poder comunicarse eficientemente con la instancia apropiada del microservicio que cambia dinámicamente. Estos patrones apoyan dinámicamente la resolución de direcciones DNS en direcciones IP (Taibi *et al*, 2018).

- Patrón de Descubrimiento del lado del cliente

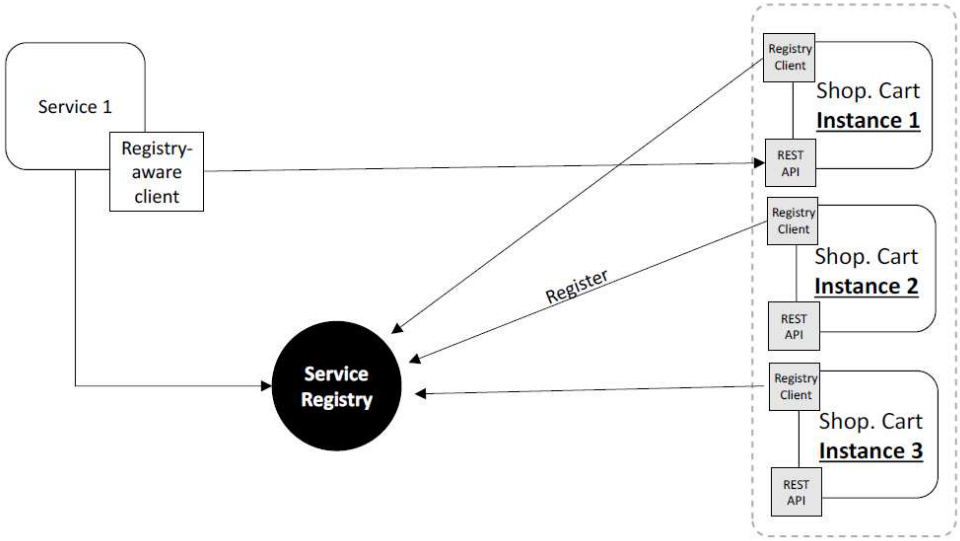
Los clientes consultan el Registro de Servicios, seleccionan una instancia disponible y realizan una solicitud directamente. Este patrón implica que el cliente es responsable de elegir una de las instancias de servicios disponibles y de sus ubicaciones de red.

La principal ventaja de este patrón está conectada a la facilidad de desarrollo, ya que los clientes son conscientes de las ubicaciones de las instancias de servicio y, por lo tanto, pueden conectarse directamente a ellas sin añadir la complejidad de desarrollo del descubrimiento del lado del servidor.

En la figura 5 se observa como los clientes consultan un Registro de Servicios para obtener la dirección de las instancias disponibles de un servicio específico, en este caso, un carrito de compras. Cada instancia de servicio se registra en el Registro de Servicios al iniciarse, proporcionando su ubicación actual. El cliente, que está "consciente del registro", consulta este registro para determinar a cuál instancia conectarse, utilizando la API REST del servicio correspondiente.

Figura 5.

Patrón de descubrimiento del lado del cliente



Nota. Varios artículos también identificaron desventajas de patrones. Adaptado de D. Taibi, Lenarduzzi, V. & C. Pahl. Architectural Patterns for Microservices: A Systematic Mapping Study. *8th International Conference on Cloud Computing and Services Science,* p. 5. Doi: (10.5220/0006798302210232).

8.1.2.2.3. Patrones de almacenamiento de datos en microservicios

En arquitecturas de microservicios, el almacenamiento de datos es un aspecto crítico que debe ser gestionado cuidadosamente para mantener la independencia y escalabilidad de los servicios. Estos patrones permiten diseñar sistemas que pueden crecer y adaptarse eficientemente a los desafíos asociados con la distribución y la gestión de datos (Taibi *et al*, 2018).

- Patrón de base de datos por servicio

En este patrón, cada microservicio dispone de una base de datos separada e independiente. Este método es uno de los más simples y se adopta frecuentemente al migrar sistemas monolíticos hacia una arquitectura de microservicios. Al mantener bases de datos independientes para cada microservicio, se logra una cohesión interna fuerte y se minimiza el acoplamiento entre los servicios, lo que a su vez facilita la escalabilidad y el mantenimiento del sistema. Este patrón es compatible tanto con bases de datos relacionales como NoSQL.

- Patrón de Clúster de Bases de Datos

Este patrón propone almacenar los datos en un clúster de bases de datos, lo que aumenta la escalabilidad y permite trasladar las bases de datos a hardware especializado. En este enfoque, cada microservicio puede tener acceso a un conjunto específico de tablas o a un esquema privado dentro de la base de datos. Desde la perspectiva del microservicio, este patrón es similar al Servidor de Base de Datos Compartido, dado que la interacción con la base de datos se efectúa de la misma manera.

8.1.3. Arquitecturas *serverless*

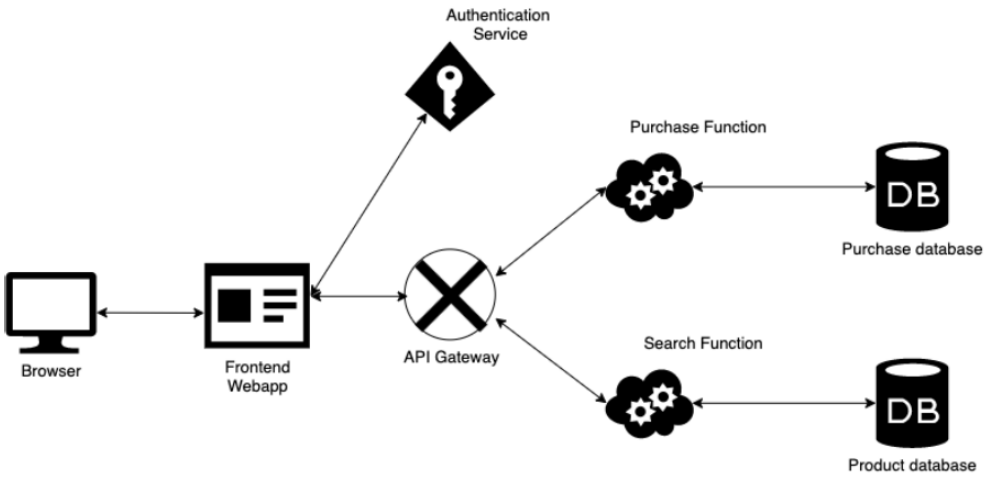
Las arquitecturas *serverless* representan un enfoque moderno en la construcción de aplicaciones, caracterizadas por la descomposición de sistemas en múltiples funciones independientes que se ejecutan en la nube. En contraste con las arquitecturas monolíticas o de microservicios, donde los componentes están interconectados, en una arquitectura serverless cada función es independiente, se dispara en respuesta a eventos específicos y ajusta su escala

automáticamente según las necesidades. Con esta metodología, se minimiza la gestión de la infraestructura, dando a los desarrolladores la libertad de enfocarse por completo en los procesos clave del negocio.

En la figura 6 se ejemplifica como una tienda en línea implementa una arquitectura *serverless*, en donde se observa una función para buscar y otra para realizar una compra.

Figura 6.

Arquitectura Serverless



Nota. Ejemplo de arquitectura sin servidor. Adaptado de O. Andel (2020). *Architectural Implications of Serverless and Function-as-a-Service.* (https://www.diva-portal.org/smash/get/diva2:1442437/FULLTEXT01.pdf), consultado el 15 de agosto de 2024. Reservado derechos de autor.

8.1.3.1. Fundamentos teóricos de *serverless*

Serverless o *Function-as-a-Service* (FaaS) ha aparecido como una innovación en la computación en la nube, situándose entre los modelos Platform-

as-a-Service (PaaS) y *Software*-as-a-Service (SaaS) en términos del control que los desarrolladores tienen sobre la plataforma. A pesar del nombre, las funciones *serverless* siguen ejecutándose en servidores; sin embargo, la administración tanto de la infraestructura como de los servidores está totalmente bajo la responsabilidad de un proveedor de la nube. Este modelo facilita que los desarrolladores se concentren exclusivamente en los procesos del negocio, mientras el proveedor se ocupa del aprovisionamiento, escalado y gestión de los recursos. (Andell, 2020).

La arquitectura *serverless* se caracteriza por la descomposición de aplicaciones en funciones autónomas y sin estado, que se activan en respuesta a eventos específicos, como solicitudes HTTP o cambios en la base de datos. Esto ofrece ventajas significativas, como la capacidad de escalado automático, la reducción de costos de infraestructura al pagar solo por el uso real, y la eliminación de la necesidad de gestionar servidores.

No obstante, este enfoque también conlleva desafíos, como la dependencia del proveedor de la nube, lo que puede llevar a un problema de *vendor lock-in*, y la complejidad añadida por los *cold starts*, que pueden aumentar la latencia cuando una función no se ha ejecutado recientemente. A pesar de sus limitaciones, la arquitectura *serverless* sigue siendo una opción viable y atractiva para desarrollar aplicaciones escalables y rentables, especialmente en escenarios con demandas de recursos impredecibles.

8.1.3.2. Comparación entre arquitecturas *serverless* y tradicionales

Las arquitecturas *serverless* se diferencian significativamente de las arquitecturas tradicionales, como las monolíticas y las basadas en microservicios,

en varios aspectos clave. En una arquitectura monolítica convencional, toda la lógica de la aplicación está integrada en un solo y amplio bloque de código, lo que facilita las fases iniciales de desarrollo e implementación, pero complica la escalabilidad y el mantenimiento a medida que el sistema crece. En contraste, las arquitecturas basadas en microservicios separan la aplicación en servicios autónomos más pequeños, que se pueden escalar y desplegar de forma independiente. Esto soluciona varios problemas de las arquitecturas monolíticas, pero introduce una mayor complejidad en la administración de múltiples servicios y sus interrelaciones. (Andell, 2020).

Las arquitecturas *serverless* van un paso más allá en esta descomposición, dividiendo la aplicación en funciones aún más pequeñas y autónomas que se activan en respuesta a eventos concretos. No solo se reduce el costo y la dificultad en la gestión de la infraestructura, que es completamente administrada por el proveedor del servicio, sino que también se habilita un escalado automático más preciso y eficiente.

En cuanto al rendimiento, las arquitecturas tradicionales tienden a ser más predecibles y estables en términos de latencia, especialmente en sistemas donde la latencia baja y consistente es crítica. En contraste, las arquitecturas *serverless* pueden sufrir de latencias más altas e impredecibles debido a los *cold starts*, lo que las hace menos adecuadas para aplicaciones que requieren respuestas instantáneas. Además, aunque las arquitecturas *serverless* prometen reducir los costes operacionales al facturar solo por el uso real, la gestión y optimización de costes puede ser más compleja en comparación con los modelos de precios más predecibles de las arquitecturas tradicionales.

8.1.3.3. Beneficios de *serverless* para aplicaciones dinámicas y de alto tráfico

Las arquitecturas *serverless* son particularmente beneficiosas para aplicaciones dinámicas y de alto tráfico debido a su capacidad inherente de escalar automáticamente en respuesta a la demanda. A diferencia de las arquitecturas tradicionales, donde es necesario prever el número de servidores necesarios y gestionarlos manualmente, *serverless* permite que los recursos se ajusten de manera automática y granular en función de las necesidades en tiempo real, lo que garantiza que la aplicación pueda manejar picos de tráfico sin intervención manual y sin incurrir en costos adicionales significativos cuando el tráfico es bajo (Andell, 2020).

Este enfoque también es ventajoso en términos de costos operacionales. Las aplicaciones *serverless* solo generan costos cuando están activas, ya que la infraestructura es gestionada por los proveedores de nube, quienes desactivan automáticamente las funciones cuando no están en uso. Esto contrasta con las arquitecturas tradicionales, donde se pagan costos fijos por servidores que pueden estar infrautilizados durante periodos de baja demanda.

Además, la naturaleza sin estado de las funciones *serverless* permite que múltiples instancias de una función se ejecuten en paralelo sin conflictos, optimizando aún más la capacidad de la aplicación para manejar altos volúmenes de tráfico. Esto es crucial para aplicaciones que experimentan patrones de uso altamente variables, ya que la arquitectura puede adaptarse de manera eficiente a estos cambios sin necesidad de una planificación extensa o intervención administrativa.

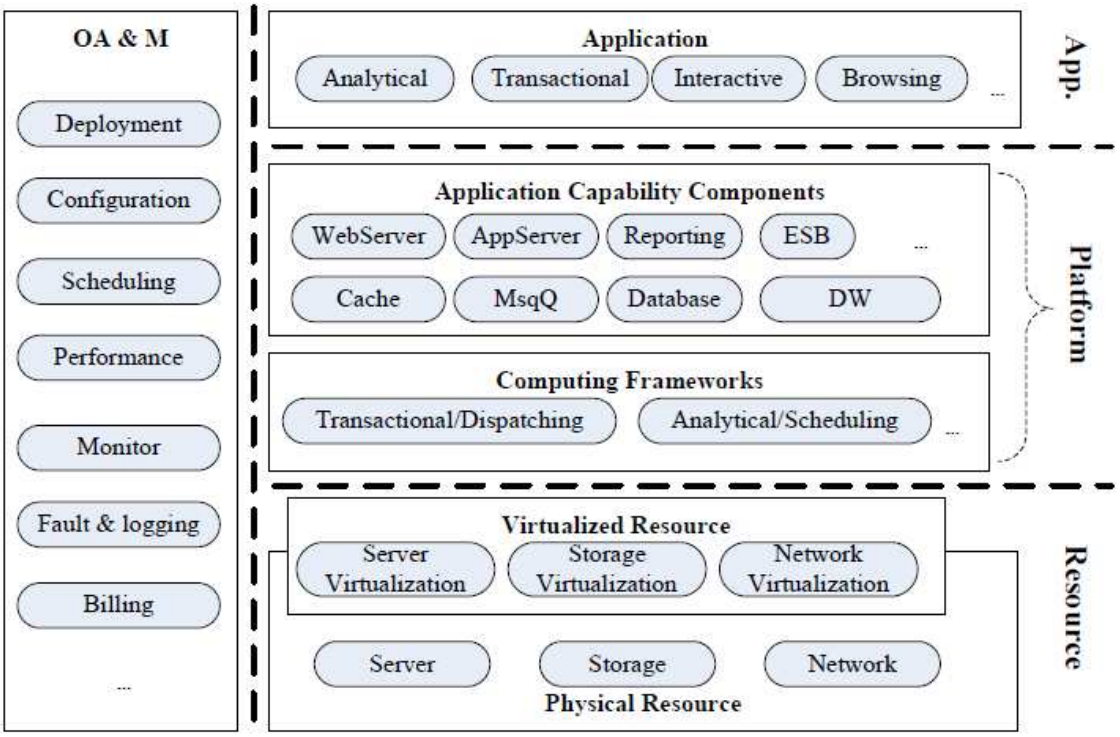
8.2. Computación en la nube

La computación en la nube se ha establecido como una de las tecnologías más innovadoras y revolucionarias dentro del campo de la tecnología de la información. Con esta tecnología, las organizaciones pueden acceder y usar recursos de TI, tales como almacenamiento, bases de datos y capacidad de procesamiento, por medio de Internet, evitando así la necesidad de adquirir y mantener una infraestructura física que resulta costosa y compleja. Ha pasado de ser un concepto básico de almacenamiento remoto a convertirse en un sistema sofisticado que respalda aplicaciones empresariales críticas y procesos clave de negocio (Qian *et al*., 2009).

En la figura 7 se proporciona una visión clara de cómo las diferentes capas y componentes interactúan en una arquitectura de computación en la nube. Cada capa tiene un rol específico, desde la gestión de los recursos físicos hasta la entrega de aplicaciones finales a los usuarios. Para que un entorno en la nube sea exitoso, es fundamental la integración y administración eficiente de cada una de sus capas y elementos. Esta arquitectura escalonada facilita la flexibilidad, escalabilidad y eficiencia operativa, permitiendo que las empresas ajusten sus recursos y capacidades según las demandas cambiantes.

Figura 7.

Arquitectura en la nube



Nota. Arquitectura de computación en la nube. Adaptado de L. Qian, Z. Luo, Y. Du & L. Guo. Cloud Computing: An Overview. *Cloud Computing, First International Conference, CloudCom 2009, Beijing, China,* p. 628. https://www.researchgate.net/publication/221276709_Cloud_Computing_An_Overview

- Recursos físicos: esta es la capa más básica, que incluye los componentes físicos como servidores, almacenamiento y redes. Estos son los elementos tangibles que proporcionan la base para todos los servicios en la nube.

- Recursos virtualizados: por encima de los recursos físicos, la virtualización permite que estos recursos físicos se dividan en múltiples instancias virtuales. La virtualización del servidor, almacenamiento y red permite que

estos recursos se utilicen de manera más eficiente y sean escalables. Los recursos virtualizados son fundamentales para proporcionar servicios en la nube de manera flexible y dinámica.

- Marcos de trabajo computacionales: esta capa incluye los frameworks de procesamiento que soportan las operaciones de la nube. Ejemplos incluyen el procesamiento transaccional y el análisis y programación de tareas. Estos marcos permiten la ejecución eficiente de tareas que pueden ser de naturaleza transaccional o analítica.

- Componentes de capacidad de aplicación: en esta subcapa, se encuentran componentes como servidores web, servidores de aplicaciones, bases de datos, sistemas de colas de mensajes (MsqQ), y otras herramientas esenciales para la construcción de aplicaciones. Estos componentes son cruciales para brindar servicios específicos a las aplicaciones que operan en la nube.

- Aplicaciones: en esta capa se encuentran las aplicaciones que realizan funciones específicas como análisis, transacciones, interacción y navegación. Estas aplicaciones son las que los usuarios finales utilizan y se benefician de los servicios y capacidades proporcionados por las capas inferiores. Cada tipo de aplicación puede requerir diferentes capacidades de la infraestructura subyacente.

- Operación, administración y mantenimiento: en el lateral izquierdo de la figura, se detalla un conjunto de funciones relacionadas con la operación, administración y mantenimiento de la infraestructura de la nube. Esto incluye la implementación, configuración, programación, monitoreo, gestión del rendimiento, y facturación, entre otros. Estas funciones son

esenciales para asegurar que todos los recursos y aplicaciones en la nube operen de manera eficiente, segura y dentro de los costos previstos.

8.2.1. Conceptos básicos de la computación en la nube

La nube se refiere a la provisión de diferentes servicios informáticos, como servidores, almacenamiento, bases de datos, redes y *software*, así como herramientas de análisis, todo a través de Internet. Este modelo de servicio permite a las organizaciones alquilar recursos informáticos en lugar de adquirir y mantener su propia infraestructura física, lo que se traduce en una mayor flexibilidad y eficiencia en costos.

Existen diferentes enfoques para implementar soluciones en la nube. El más elemental es la Infraestructura como Servicio (IaaS), que ofrece a los usuarios acceso a recursos informáticos virtualizados. Las organizaciones pueden alquilar estos recursos según sus necesidades, escalando vertical u horizontalmente de acuerdo con la demanda. Ejemplos populares de este enfoque incluyen Amazon Web Services (AWS) y Microsoft Azure (Qian *et al*., 2009).

El modelo Plataforma como Servicio (PaaS) proporciona una plataforma sobre la cual los desarrolladores pueden crear, desplegar y administrar aplicaciones sin preocuparse por la infraestructura que está detrás. Este modelo abarca tanto el hardware requerido como las herramientas de desarrollo y el middleware, que simplifican el proceso de creación de *software*. Google App Engine y Microsoft Azure son ejemplos de servicios PaaS (Rashid & Chaturvedi, 2019).

El modelo *Software* como Servicio (SaaS) ofrece aplicaciones completas a los usuarios finales mediante Internet. Al estar alojadas en la nube, estas aplicaciones pueden ser utilizadas desde distintos dispositivos con conexión a la red. SaaS elimina la necesidad de instalar y ejecutar aplicaciones en los ordenadores de los usuarios, lo que simplifica el mantenimiento y el soporte técnico. Ejemplos de SaaS incluyen Google Workspace y Salesforce.

La computación en la nube también se caracteriza por cinco atributos esenciales: autoservicio bajo demanda, acceso amplio a la red, agrupación de recursos, elasticidad rápida y servicio medido. Estos atributos garantizan que las organizaciones puedan gestionar y utilizar los recursos de manera eficiente y adaptarse rápidamente a los cambios en las demandas del mercado.

8.2.2. Ventajas de la nube para el escalado y manejo de aplicaciones empresariales

La computación en la nube proporciona múltiples beneficios para el escalado y la gestión de aplicaciones empresariales, cambiando la forma en que las empresas operan y administran sus recursos tecnológicos.

- La habilidad de escalar recursos automáticamente según las necesidades. Las aplicaciones empresariales que enfrentan variaciones en la carga de trabajo, como aumentos de tráfico durante promociones o eventos especiales, pueden aprovechar considerablemente la habilidad de la nube para ajustar los recursos de manera dinámica. (Qian *et al.*, 2009).

- En lugar de invertir en infraestructura costosa y subutilizada, las organizaciones solo pagan por los recursos que realmente utilizan. Esto es particularmente beneficioso para las aplicaciones que tienen patrones

de uso variables, como las que se ejecutan de manera intensiva durante ciertas horas del día o en temporadas específicas (Rashid & Chaturvedi, 2019).

- Proporciona alta disponibilidad y redundancia incorporada, lo que es importante para aplicaciones empresariales críticas. Los proveedores de estos servicios implementan infraestructuras robustas con múltiples centros de datos distribuidos geográficamente, permitiendo que las aplicaciones permanezcan operativas incluso en caso de fallos en uno o más servidores.

- La nube ofrece una flexibilidad sin precedentes para el manejo de aplicaciones empresariales. Las empresas pueden implementar nuevas aplicaciones o servicios en la nube de manera rápida y sencilla, lo que les permite adaptarse a las necesidades cambiantes del mercado y responder a las oportunidades con mayor agilidad.

8.2.3. Amazon Web Services

Es una de las plataformas de servicios en la nube más adoptadas del mundo, ofreciendo servicios para desarrollar, implementar y gestionar aplicaciones con muchas herramientas. AWS proporciona infraestructura bajo demanda, lo que quiere decir que las empresas pueden acceder a diferentes recursos en función de sus necesidades, pagando únicamente lo necesario, sin realizar inversiones iniciales significativas en hardware (Kewate *et al.*, 2022).

Soporta una amplia gama de servicios que incluyen cómputo, almacenamiento, bases de datos, análisis, redes, movilidad, herramientas para desarrolladores, administración, IoT, seguridad, y aplicaciones empresariales.

8.2.3.1. Servicios enfocados en Microservicios

AWS ofrece una amplia gama de servicios diseñados específicamente para apoyar arquitecturas basadas en microservicios. Estos servicios permiten a las organizaciones desarrollar, desplegar y escalar aplicaciones compuestas por microservicios de manera eficiente y flexible. Por ejemplo, Amazon Elastic Container Service (ECS) es un servicio altamente escalable que permite ejecutar y gestionar contenedores Docker en un clúster.

8.2.3.1.1. Amazon Elastic Container Service

Se trata de un servicio completamente administrado por AWS que permite desplegar y gestionar contenedores *Docker* a gran escala. ECS facilita la implementación de aplicaciones que siguen una arquitectura de microservicios, donde cada microservicio se ejecuta en un contenedor independiente, permitiendo una mayor modularidad, escalabilidad y resiliencia.

ECS funciona al crear un clúster de recursos que pueden distribuirse entre diferentes centros de datos. Estos clústeres pueden llenarse con instancias de EC2 que actúan como la infraestructura subyacente para los contenedores. Cada instancia de EC2 en el clúster ejecuta un agente ECS, que registra la instancia en el clúster y gestiona la comunicación entre los contenedores y la infraestructura subyacente (Salah *et al*., 2022).

Es un buen servicio a considerar para la gestión de aplicaciones basadas en microservicios, gracias a su integración profunda con la infraestructura de AWS y sus capacidades avanzadas de gestión y escalado automatizado.

8.2.3.2. Servicios *Serverless*

Los servicios *serverless* han transformado la forma en que se desarrollan y despliegan las aplicaciones, al eliminar la necesidad de que el usuario final gestione la infraestructura. Con este enfoque, los desarrolladores pueden concentrarse únicamente en los procesos operativos del negocio, mientras que el proveedor de soluciones en la nube se encarga del aprovisionamiento, escalado y gestión de los servidores. En el contexto de AWS, uno de los servicios *serverless* más destacados es AWS Lambda, que fue lanzado en 2015 como la primera plataforma comercial de Function-as-a-Service (FaaS) (Andell, 2020).

8.2.3.2.1. Amazon Lambda

Lambda ofrece a los desarrolladores la posibilidad de ejecutar código en reacción a eventos, eliminando la necesidad de gestionar servidores. Esto se logra mediante la división de aplicaciones en pequeñas funciones independientes, que se activan automáticamente en respuesta a eventos como solicitudes HTTP, cambios en bases de datos, o mensajes de servicios de colas. Este enfoque elimina la carga de aprovisionar y mantener infraestructura, permitiendo que las aplicaciones escalen automáticamente según la demanda (Andell, 2020).

Entre las características más destacadas de AWS Lambda se encuentra su capacidad para escalar automáticamente, ajustando el número de instancias de la función según la carga de trabajo. Esto es particularmente útil en aplicaciones con patrones de tráfico impredecibles, ya que permite que las aplicaciones manejen picos de tráfico sin intervención manual. Además, Lambda ofrece un modelo de precios basado en el uso real, cobrando solo por el tiempo de ejecución de las funciones, lo que reduce significativamente los costos

operacionales en comparación con arquitecturas tradicionales que requieren servidores siempre encendidos.

Este servicio es compatible con una amplia variedad de lenguajes de programación, lo que ofrece a los desarrolladores una gran flexibilidad para utilizar el lenguaje que mejor se adapte a sus necesidades. Entre los lenguajes soportados se incluyen Node.js, Python, Java, Ruby, C# (.NET Core), Go, y PowerShell. Además, AWS Lambda permite ejecutar funciones en otros lenguajes a través del uso de contenedores personalizados, lo que amplía la flexibilidad de la plataforma para soportar casi cualquier entorno de ejecución. Esta capacidad multilenguaje hace que Lambda sea una opción ideal para equipos de desarrollo que trabajan con diferentes tecnologías y necesitan un entorno que soporte sus herramientas y flujos de trabajo preferidos.

8.3. Integración de Microservicios y *Serverless*

La integración de microservicios con arquitecturas *serverless* es un enfoque que combina lo mejor de ambos mundos: la modularidad y la independencia de los microservicios con la eficiencia y la escalabilidad automática de las plataformas *serverless*. Estas dos arquitecturas están diseñadas para abordar los desafíos de las aplicaciones modernas, donde la demanda de agilidad, escalabilidad y reducción de costos operacionales es crucial.

La arquitectura *serverless* permite ejecutar microservicios de manera que solo se utilizan los recursos computacionales cuando es necesario, eliminando la necesidad de mantener servidores activos constantemente. Esto se logra mediante el uso de servicios como AWS Lambda, que ejecuta funciones en respuesta a eventos y se escala automáticamente según la demanda. La

combinación de microservicios y *serverless* también facilita la implementación de aplicaciones altamente escalables y resilientes, ya que cada microservicio puede desplegarse, gestionarse y escalarse de manera independiente (Sadek y colaboradores, 2022) (Andell, 2018).

8.3.1. Integración y Gestión

Integrar microservicios en una arquitectura *serverless* requiere estrategias de gestión específicas para abordar la complejidad y garantizar un rendimiento óptimo. Una estrategia crucial es la implementación de un API Gateway para centralizar la gestión de las solicitudes que se envían a los microservicios. API Gateway actúa como un punto de entrada que enruta las solicitudes a los microservicios correspondientes, aplica políticas de seguridad, y maneja el tráfico de manera eficiente. Además de facilitar la comunicación entre microservicios, también mejora la seguridad y la capacidad de escalar la aplicación.

Otra estrategia efectiva es utilizar patrones de orquestación mediante servicios como *AWS Step Functions*, que permiten coordinar múltiples microservicios y funciones *serverless* en flujos de trabajo automatizados. Este enfoque es esencial para manejar dependencias complejas entre microservicios y garantizar que las tareas se ejecuten en el orden correcto. Además, la implementación de servicios de colas y mensajerías que permiten una comunicación asíncrona y desacoplada entre los microservicios, lo que mejora la resiliencia y la escalabilidad del sistema (Heikkinen, 2023).

Para manejar la complejidad de estas arquitecturas, es crucial disponer de herramientas de monitoreo y registro que ofrezcan visibilidad. En AWS existen servicios que apoyan a esta causa, tales como Cloudwatch y Cloudtrail.

8.3.2. Estrategias de Migración

Pasar de un sistema monolítico a una combinación de microservicios y *serverless* puede presentar desafíos, pero con la estrategia correcta, se pueden minimizar los riesgos y aprovechar al máximo sus ventajas. Un enfoque común es la migración gradual, donde partes del sistema monolítico se van desacoplando y migrando a microservicios, que luego pueden integrarse con funciones *serverless*. Este enfoque ofrece a los equipos de desarrollo la posibilidad de probar la nueva arquitectura en pequeñas partes del sistema antes de comprometerse por completo con la migración. (Heikkinen, 2023).

Otro enfoque efectivo es el uso de contenedores para encapsular partes del sistema que aún no se pueden convertir completamente en microservicios o funciones *serverless*. Al utilizar plataformas como AWS ECS, es posible ejecutar estos contenedores de manera *serverless*, lo que facilita la integración con los componentes ya migrados y reduce la complejidad de la infraestructura (Sadek *et al*., 2022).

Además, es crucial realizar un análisis profundo de las dependencias del sistema antes de comenzar la migración. Identificar los componentes que tienen muchas interdependencias con otras partes del sistema puede ayudar a planificar mejor la secuencia de la migración, priorizando el desacoplamiento de estas partes para reducir el riesgo de fallos durante la transición (Heikkinen, 2023).

Identificar qué servicios son los más adecuados para extraer en una arquitectura monolítica es una parte muy importante, una estrategia es extraer aquellos que se encuentran en las hojas del árbol de dependencias de servicios, obtenidos al analizar la base de código. Estos servicios no dependen de otros y

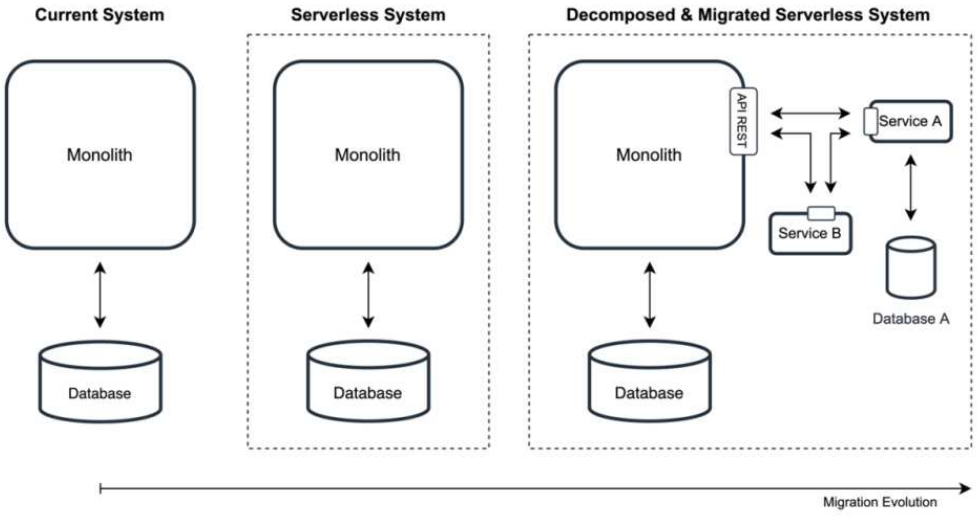
generalmente tienen contextos delimitados de manera más clara, lo que evita complicaciones comunes (Brula, 2023).

Aunque algunos servicios son más fáciles de extraer que otros, como regla general, se recomienda extraer primero un servicio que tenga una necesidad urgente de funcionar de manera independiente del monolito y que cuente con contextos bien definidos. Si se selecciona un servicio de esta manera, el esfuerzo de refactorización, mejora y migración eventualmente dará frutos. Las razones pueden ser variadas, desde la perspectiva organizacional, la mantenibilidad, el rendimiento, la disponibilidad, entre otras. Migrar y mantener microservicios sin un propósito claro puede resultar costoso tanto en tiempo como en dinero.

La figura 8 ilustra la evolución de un prototipo de sistema monolítico hacia una arquitectura *serverless* y finalmente hacia un sistema completamente descompuesto y migrado que incorpora microservicios junto con la arquitectura *serverless*.

Figura 8.

Descripción general de los pasos de migración del prototipo



Nota. Implementación del prototipo. Adaptado de M. Brula (2023). *From Monolith to Serverless Microservices Migration.* (https://epub.technikum-wien.at/obvftwhsmmig/download/pdf/9746870), consultado el 16 de septiembre de 2024. Derechos reservados del autor.

8.4. Introducción a Magento

Magento, una plataforma de comercio electrónico que ofrece una versión de paga y otra de código abierto, destaca por su notable flexibilidad y opciones avanzadas de personalización. Desarrollada originalmente por Varien Inc. y lanzada en 2008, Magento ha crecido hasta convertirse en una de las plataformas más reconocidas del mercado, especialmente entre pequeñas y medianas empresas. Magento permite a los desarrolladores crear tiendas en línea altamente personalizables, ofreciendo una amplia gama de funcionalidades, desde la gestión de productos y categorías hasta la integración de múltiples métodos de pago y la optimización para motores de búsqueda (Lin, 2015).

La versión 2.0 (lanzada en noviembre de 2015), ha experimentado cambios en términos de estructura y patrones de desarrollo, y actualmente se encuentra en su segunda versión principal, Magento 2. (Morizur, 2019).

8.4.1. Descripción general de la plataforma Magento

Magento se distingue por su capacidad de personalización. Los usuarios pueden adaptar casi todos los aspectos de su tienda, desde el diseño del *frontend* hasta las funciones de *backend*, mediante la instalación de módulos y temas personalizados. Esto facilita que las empresas ofrezcan experiencias de compra exclusivas y adaptadas a las necesidades de su audiencia. Sin embargo, esta flexibilidad viene con una complejidad inherente, lo que puede requerir un conocimiento técnico considerable para su implementación y mantenimiento.

Magento se basa en el lenguaje de programación PHP, lo que le permite aprovechar la robustez y la amplia comunidad de desarrolladores de PHP. Esta elección de lenguaje también facilita la integración con diversas herramientas y servicios web, lo que es crucial para crear experiencias de compra en línea personalizadas y eficientes.

Las principales características de Magento incluyen la gestión avanzada de catálogos de productos, diversas opciones de pago y envío, y la capacidad de gestionar múltiples sitios web desde una sola instalación. Además, la plataforma cuenta con una robusta interfaz de administración que facilita la gestión eficiente de las tiendas, soporte para motores de búsqueda, múltiples monedas e idiomas, y una amplia variedad de extensiones de terceros para ampliar la funcionalidad de la tienda.

8.4.2. Arquitectura modular

Uno de los principales puntos fuertes de Magento es su arquitectura modular, que ofrece a los desarrolladores la capacidad de crear y gestionar funcionalidades particulares de forma independiente dentro de la misma aplicación. Cada módulo en Magento encapsula una funcionalidad particular, como la gestión de productos, el sistema de pagos, o el manejo de usuarios. Estos módulos están diseñados para ser independientes entre sí, lo que significa que se pueden añadir, modificar o eliminar sin afectar la estabilidad del resto del sistema.

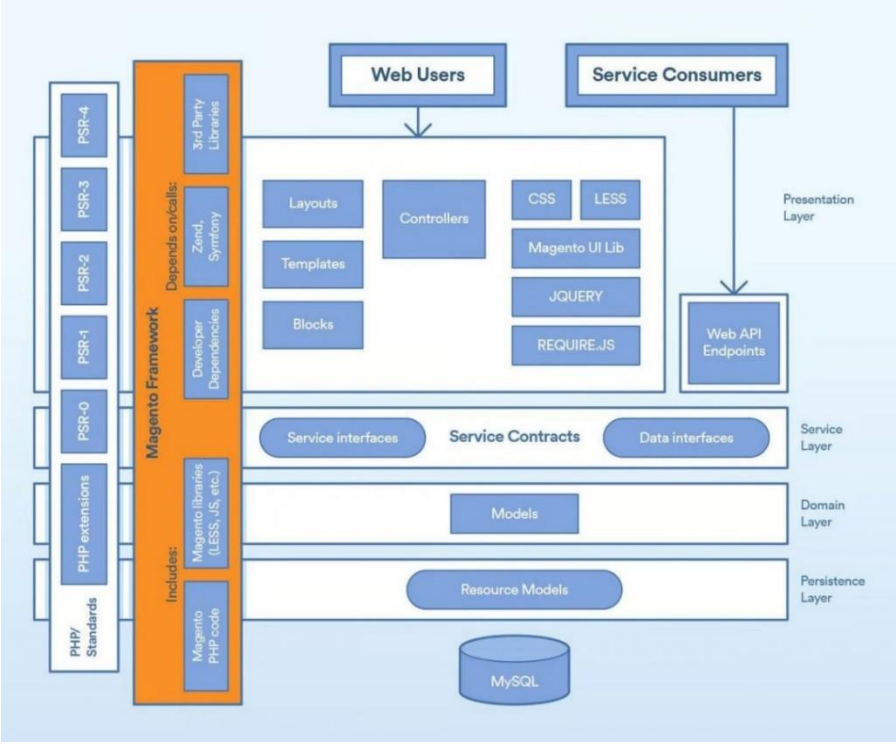
Los módulos en Magento pueden interactuar entre sí mediante relaciones específicas que son cuidadosamente gestionadas para asegurar la coherencia y estabilidad del sistema. Por ejemplo, un módulo puede utilizar otro módulo, reaccionar a eventos generados por otro módulo, o incluso personalizar y extender la funcionalidad de otro módulo sin necesidad de modificar el núcleo del sistema. Esto se consigue mediante una mezcla de dependencias estrictas y flexibles, las cuales están claramente especificadas en la configuración de cada módulo.

Además, Magento permite la instalación y gestión de módulos de forma sencilla utilizando Composer, un gestor de dependencias para PHP. Esto facilita la adición de nuevas funcionalidades a una instalación existente de Magento, así como la actualización y mantenimiento de módulos sin interferir con otros componentes del sistema. La capacidad de Magento para gestionar módulos de forma efectiva es una de las principales razones de su flexibilidad y adaptabilidad, permitiendo a las empresas ajustar su experiencia de comercio electrónico sin comprometer la estabilidad del sistema.

La estructura de Magento se divide en dos partes clave: el back-end, que abarca la base de datos, MySQL, y las interfaces de modelo, datos y servicio. Estas partes están directamente conectadas y se utilizan en los Bloques, Diseños y Plantillas de Magento, que constituyen el front-end de la aplicación. La figura 9 muestra la arquitectura general de Magento, que está estructurada en varias capas, cada una con funciones específicas que permiten la flexibilidad y extensibilidad de la plataforma (Morizur, 2019).

Figura 9.

Representación de la arquitectura de Magento en su versión 2.0



Nota. Capas de Magento. Adaptado de D. Morizur, (2019). *Enhancing Magento Frontend Performance With Reactjs and Comparing It To Knockout*. (https://www.theseus.fi/bitstream/handle/10024/161745/Thesis.pdf?sequence=2), consultado el 16 de septiembre de 2024. Derechos reservados del Autor.

9. PROPUESTA DE ÍNDICE DE CONTENIDOS

10. METODOLOGÍA

10.1. Tipo de estudio

El presente estudio se enmarca en un enfoque mixto, combinando métodos cuantitativos y cualitativos. El componente cuantitativo del estudio se enfocará en la medición y análisis de indicadores clave relacionados con la escalabilidad y el rendimiento, proporcionando datos objetivos que permitan comparar de manera precisa las arquitecturas híbridas con las tradicionales.

Simultáneamente, el enfoque cualitativo se utilizará para explorar en profundidad los desafíos técnicos y las experiencias durante la implementación de la arquitectura híbrida. Al combinar ambos enfoques, el estudio proporcionará una visión integral apoyada en datos cuantitativos para evaluar la escalabilidad y también en cuenta las perspectivas prácticas y los desafíos técnicos de las empresas al migrar de arquitecturas monolíticas a arquitecturas más modernas y escalables.

10.2. Diseño de la investigación

El estudio adopta un diseño de investigación experimental. Este diseño es adecuado para abordar el objetivo de diseñar una arquitectura híbrida que combine microservicios y tecnologías *serverless* para mejorar la escalabilidad en comparación con las arquitecturas monolíticas tradicionales, específicamente en la plataforma Magento 2.0. Además, este enfoque permite la identificación de los módulos específicos de Magento que son viables para la migración y que se beneficiarían de ella, realizando una selección informada y estratégica de

aquellos componentes que optimizarán la eficiencia operativa y la capacidad de respuesta del sistema.

10.3. Alcance de la investigación

El presente estudio tiene un alcance explicativo, centrado en analizar y comprender las causas y efectos de la transformación de la arquitectura monolítica de Magento 2.0 en una arquitectura híbrida basada en microservicios y *serverless*. Este enfoque permite explorar en profundidad cómo la migración de módulos clave impacta en la escalabilidad y eficiencia operativa del sistema, estableciendo una relación causal entre el tipo de arquitectura implementada y los beneficios en términos de rendimiento y costos operacionales.

A través del desarrollo y evaluación de un prototipo experimental, la investigación busca no solo describir las mejoras observadas, sino también explicar los mecanismos que conducen a estas mejoras. Este alcance explicativo también permite la generación de conocimientos predictivos, orientados a anticipar cómo otras plataformas monolíticas podrían beneficiarse de una transición similar.

- Crear un prototipo o modelo que permita observar y medir el impacto de la migración a una arquitectura híbrida.

- Comparar los resultados obtenidos de diferentes configuraciones o estados, como comparar el rendimiento de la arquitectura monolítica original con la nueva arquitectura híbrida.

- Realizar pruebas para determinar la escalabilidad y rendimiento operacional.

- Ofrecer recomendaciones para la implementación de la arquitectura híbrida en otros contextos o empresas.

10.4. Variables e indicadores

En la tabla 1 se muestran las variables y los indicadores que se tomarán en cuenta para la investigación.

Tabla 1.

Definición de las variables

Variable	Definición técnica	Definición operativa
Escalabilidad	Capacidad del sistema para manejar un aumento en la carga de trabajo sin comprometer el rendimiento.	Tiempo de respuesta en segundos o milisegundos ante ajuste de recursos adicionales ante un pico de carga
Eficiencia Operativa	Capacidad del sistema para operar utilizando los menores recursos posibles manteniendo un rendimiento óptimo.	Porcentaje de utilización de recursos (CPU, Memoria) y costos totales
Módulos Migrados	Componentes específicos de Magento seleccionados para la migración a microservicios y *serverless*.	Número total de módulos migrados
Número de Desafíos Técnicos Identificados	Cantidad de problemas técnicos surgidos durante la migración de la arquitectura.	Recuento y clasificación de los problemas técnicos encontrados durante el proceso de migración.
Tiempos de respuesta	Tiempo promedio de procesamiento en milisegundos o segundos.	Tiempo promedio de funciones clave que hayan sido afectadas por las modificaciones en la arquitectura

Nota. Variables independientes. Elaboración propia, realizado con Excel.

10.5. Fases del estudio

A continuación, se describen las fases que realizaran para el estudio.

10.5.1. Fase 1: revisión de literatura y estructuración del proyecto

En esta fase inicial, se llevará a cabo una investigación exhaustiva de la literatura existente, incluyendo artículos académicos, estudios de caso, y otras fuentes relevantes que traten sobre la migración de arquitecturas monolíticas a microservicios y *serverless*. El objetivo es fundamentar la investigación en el conocimiento existente, identificar lagunas en la literatura que tu estudio pueda abordar, y establecer un contexto claro sobre la importancia del problema a investigar. También se desarrollará la planificación y estructuración del proyecto, que abarcará la selección del tipo de estudio, el alcance del proyecto, la metodología a emplear en las siguientes fases, y la definición de los pasos clave a seguir.

10.5.2. Fase 2: revisión y análisis de la arquitectura actual

Se realizará una revisión exhaustiva de la arquitectura monolítica actual de Magento 2.0, incluyendo la identificación de los módulos clave, la evaluación de su interdependencia, y la documentación del estado actual del sistema en términos de rendimiento, escalabilidad y costos operativos. Esta fase tiene como objetivo establecer una línea base que servirá para comparar los resultados después de la implementación de la nueva arquitectura.

10.5.3. Fase 3: selección y diseño de la arquitectura híbrida

En esta fase, se seleccionarán los módulos de Magento que son candidatos para la migración a microservicios y tecnologías *serverless*, y se diseñará la arquitectura híbrida, definiendo cómo se dividirán los módulos y qué servicios se transformarán en microservicios y *serverless*. El objetivo es crear un

plan detallado de la nueva arquitectura, asegurando que cada componente esté optimizado para la escalabilidad y eficiencia.

10.5.4. Fase 4: desarrollo del prototipo

Se implementará la nueva arquitectura híbrida basada en el diseño desarrollado en la fase anterior, utilizando tecnologías como *Docker*, AWS ECS y AWS Lambda para crear un prototipo funcional. Esta fase tiene como objetivo desarrollar un prototipo operativo que permita realizar pruebas de rendimiento y escalabilidad.

10.5.5. Fase 5: pruebas y evaluación

Se llevarán a cabo pruebas exhaustivas del prototipo para medir su rendimiento en comparación con la arquitectura monolítica original, incluyendo la evaluación de la escalabilidad, latencia, tiempos de respuesta, y costos operacionales. El objetivo de esta fase es validar la efectividad de la arquitectura híbrida, identificar cualquier desafío técnico adicional, y comparar los resultados con la línea base establecida en la fase 2.

10.5.6. Fase 6: análisis de resultados y optimización

Se realizará un análisis detallado de los datos recopilados durante las pruebas y se efectuarán ajustes y optimizaciones en el prototipo basado en los resultados obtenidos. Esta fase también incluye la evaluación de la viabilidad económica y técnica de una implementación a gran escala, con el objetivo de refinar la arquitectura híbrida y proporcionar recomendaciones basadas en datos para futuras implementaciones.

10.5.7. Fase 7: documentación y presentación de resultados

Se elaborará una documentación completa del proceso de migración, incluyendo manuales de operación, reportes de evaluación, análisis de costos y recomendaciones estratégicas. Finalmente, se presentarán los resultados de la investigación, destacando los beneficios y desafíos de la nueva arquitectura, con el objetivo de entregar un informe final que detalle el proceso, resultados, y conclusiones del estudio.

10.6. Técnicas de recolección de datos

- Análisis de textos: se revisará la documentación y los estudios de casos sobre migraciones similares de arquitecturas monolíticas a microservicios y *serverless*. Esta revisión permitirá identificar mejores prácticas, posibles desafíos y estrategias de mitigación que pueden ser aplicables al proyecto.

- Observación directa: se llevará a cabo una observación directa del proceso de migración y de la operación del sistema en el entorno de prueba, documentando cualquier incidente, ajuste o decisión técnica relevante. El objetivo es capturar información en tiempo real sobre la implementación y funcionamiento del prototipo, identificando problemas y soluciones emergentes.

- Pruebas de rendimiento y escalabilidad: se realizarán pruebas controladas en el prototipo de la arquitectura híbrida para medir su rendimiento y escalabilidad, incluyendo pruebas de carga y estrés que simulen condiciones de alto tráfico y demanda. El propósito es obtener datos cuantitativos sobre la capacidad del sistema para manejar diferentes volúmenes de carga y su tiempo de respuesta bajo condiciones variables.

11. TÉCNICAS DE ANÁLISIS DE LA INFORMACIÓN

El análisis descriptivo tiene como objetivo resumir y organizar los datos obtenidos de las pruebas de rendimiento, escalabilidad, costos operacionales, latencia, y tiempo de respuesta. Esto permite presentar de manera clara y concisa las características principales de los datos recolectados antes y después de la implementación de la nueva arquitectura híbrida. A través de esta técnica, se podrán identificar patrones generales y tendencias en el comportamiento del sistema, lo que facilitará una visión preliminar del impacto de la migración.

Para este análisis se utilizarán medidas de tendencia central, como la media y la mediana, así como medidas de dispersión como la desviación estándar. Estas herramientas estadísticas ayudarán a describir de manera más precisa la variabilidad de los datos.

El análisis inferencial se centrará en determinar si las diferencias observadas entre la arquitectura monolítica y la arquitectura híbrida son estadísticamente significativas. Esta técnica es esencial para evaluar de manera precisa la efectividad de la nueva arquitectura en términos de mejora de escalabilidad y reducción de costos operacionales. Al aplicar pruebas estadísticas de inferencia, se podrá verificar si los cambios observados se deben realmente a la migración o si podrían ser producto del azar.

El análisis de contenido se empleará para evaluar los desafíos técnicos identificados durante la implementación de la nueva arquitectura híbrida. Esta técnica cualitativa permitirá identificar patrones y temas recurrentes en los problemas y observaciones encontradas durante la fase de migración. El análisis

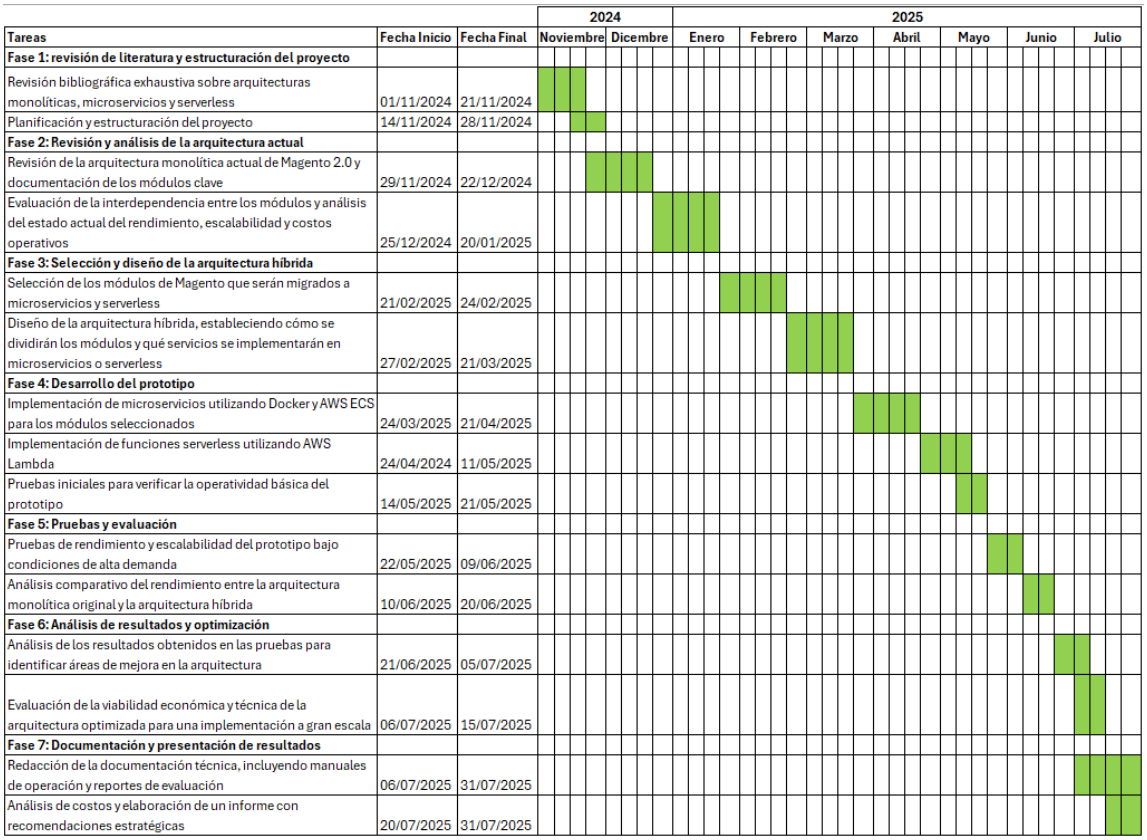
de contenido ayudará a categorizar y organizar estos desafíos, proporcionando una mejor comprensión de las dificultades que podrían surgir y las posibles soluciones que se podrían aplicar.

La visualización de datos tras las pruebas realizadas facilitará la interpretación y comunicación de los resultados del estudio. Al presentar los datos mediante gráficos y diagramas, se podrán observar de manera clara las comparaciones entre el rendimiento de la arquitectura monolítica y la híbrida, así como las relaciones entre variables clave como costos, latencia y tiempo de respuesta.

12. CRONOGRAMA

Tabla 2.

Cronograma de actividades para cada fase de la investigación



Tareas	Fecha Inicio	Fecha Final
Fase 1: revisión de literatura y estructuración del proyecto		
Revisión bibliográfica exhaustiva sobre arquitecturas monolíticas, microservicios y serverless	01/11/2024	21/11/2024
Planificación y estructuración del proyecto	14/11/2024	28/11/2024
Fase 2: Revisión y análisis de la arquitectura actual		
Revisión de la arquitectura monolítica actual de Magento 2.0 y documentación de los módulos clave	29/11/2024	22/12/2024
Evaluación de la interdependencia entre los módulos y análisis del estado actual del rendimiento, escalabilidad y costos operativos	25/12/2024	20/01/2025
Fase 3: Selección y diseño de la arquitectura híbrida		
Selección de los módulos de Magento que serán migrados a microservicios y serverless	21/02/2025	24/02/2025
Diseño de la arquitectura híbrida, estableciendo cómo se dividirán los módulos y qué servicios se implementarán en microservicios o serverless	27/02/2025	21/03/2025
Fase 4: Desarrollo del prototipo		
Implementación de microservicios utilizando Docker y AWS ECS para los módulos seleccionados	24/03/2025	21/04/2025
Implementación de funciones serverless utilizando AWS Lambda	24/04/2024	11/05/2025
Pruebas iniciales para verificar la operatividad básica del prototipo	14/05/2025	21/05/2025
Fase 5: Pruebas y evaluación		
Pruebas de rendimiento y escalabilidad del prototipo bajo condiciones de alta demanda	22/05/2025	09/06/2025
Análisis comparativo del rendimiento entre la arquitectura monolítica original y la arquitectura híbrida	10/06/2025	20/06/2025
Fase 6: Análisis de resultados y optimización		
Análisis de los resultados obtenidos en las pruebas para identificar áreas de mejora en la arquitectura	21/06/2025	05/07/2025
Evaluación de la viabilidad económica y técnica de la arquitectura optimizada para una implementación a gran escala	06/07/2025	15/07/2025
Fase 7: Documentación y presentación de resultados		
Redacción de la documentación técnica, incluyendo manuales de operación y reportes de evaluación	06/07/2025	31/07/2025
Análisis de costos y elaboración de un informe con recomendaciones estratégicas	20/07/2025	31/07/2025

Nota. Diagrama de Gantt. Elaboración propia, realizado con Microsoft Excel.

13. FACTIBILIDAD DEL ESTUDIO

13.1. Factibilidad Técnica

El presente estudio es técnicamente factible, ya que se cuenta con los recursos necesarios para su desarrollo siendo estos los siguientes:

- Recursos tecnológicos: se tiene acceso a plataformas y servicios clave como AWS (*Amazon Web Services*) para la implementación de los microservicios y *serverless*, incluyendo tecnologías como AWS Lambda y AWS ECS. Además, se utilizarán Docker para los microservicios y Magento 2.0 como el sistema a modificar. También se contempla el uso de Amazon EC2 para simular la arquitectura monolítica de Magento 2.0.

- Acceso a información: Existe acceso completo a la documentación técnica de Magento 2.0 y a estudios previos sobre migraciones similares.

- *Software* de código abierto: se trabajará con la versión 2.0 de Magento en su modalidad de código abierto, que será la base sobre la cual se desarrollará el prototipo de la nueva arquitectura híbrida. Magento 2.0 es una plataforma de comercio electrónico altamente flexible, que ofrece un entorno robusto y personalizable.

- Infraestructura y equipos: el estudio se realizará con recursos en la nube para simular diferentes escenarios de carga y realizar pruebas de escalabilidad. Se trabajará con equipo personal para la parte de codificación y microservicios de los módulos.

13.2. Factibilidad Financiera

En la tabla 2 se muestra a detalle el costo total estimado del proyecto que es de Q45,904.00.

Tabla 3.

Costos del estudio

Recurso	Descripción	Costo mensual	Costo total
AWS Lambda	Servicio *serverless* para ejecutar funciones de código	Q0.00 (capa gratuita de 1 millón de solicitudes mensuales)	Q 0.00
AWS ECS	Servicio para alojar los microservicios con base en los módulos seleccionados	Q584.00 (por 4 contenedores por 8h/día)	Q 3,504.00
AWS EC2	Servicio de cómputo para levantar servidor donde ira alojada Magento en su versión monolítica	Q400.00 (8h/día)	Q 2,400.00
Equipo de cómputo	Equipo utilizado durante el desarrollo de la investigación	Q5,000.00 (pago único)	Q 5,000.00
Salario	Salario de la persona que desarrollara el estudio	Q5,000.00	Q 35,000.00
Total			**Q 45,904.00**

Nota. Detalle de los costos del estudio. Elaboración propia, realizado con Excel.

REFERENCIAS

Allen, C. (2023). *Microservices vs Serverless Functions: A Comparison of Performance and Price*. [Tesis de maestría, Tampere University]. Archivo digital. https://trepo.tuni.fi/bitstream/handle/10024/151886/AllenChristopher.pdf;jsessionid=C1BF2CC8654AB55269352F072B632939?sequence=2

Andell, O. (2020). *Architectural Implications of Serverless and Function-as-a-Service.* [Tesis de maestría, Linköping University]. Archivo digital. https://www.diva-portal.org/smash/get/diva2:1442437/FULLTEXT01.pdf

Brula, M. (2023). *From Monolith to Serverless Microservices Migration.* [Tesis de maestría, University of Applied Sciences Technikum Wien]. Archivo digital. https://epub.technikum-wien.at/obvftwhsmmig/download/pdf/9746870

ElGheriani, N. & Ahmed, N. (2022). Microservices vs. Monolithic Architectures: The Differential Structure Between Two Architectures. *International Journal of Applied Sciences and Technology, 3*(6), 485-498. https://doi.org/10.47832/2717-8234.12.47.

Garlan, D. & Shaw, M. (2020). *Software Architecture.* Phi.

Goli, A., Hajihassani, O., Khazaei, H., Ardakanian, O., Rashidi, M., & Dauphinee, T. (2020). Migrating from Monolithic to Serverless: A FinTech Case Study. *Migrating from Monolithic to Serverless: A FinTech Case Study*. In *ACM/SPEC International Conference on Performance Engineering Companion (ICPE '20 Companion)*, 20-24. https://doi.org/10.1145/3375555.3384380

Hasan, M., Osman, M., Admodisastro, N. & Muhammad, M. (2023). From monolith to microservice: Measuring architecture maintainability. *International Journal of Advanced Computer Science and Applications*, 14(5), 857-866. https://www.researchgate.net/publication/371174225_From_Monolith_to_Microservice_Measuring_Architecture_Maintainability

Heikkinen, J. (2023). *Serverless and Microservice Architecture in Modern Software Development*. [Tesis de maestría, Jyväskylä University of Applied Sciences]. Archivo digital. https://www.theseus.fi/bitstream/handle/10024/795181/masters_thesis_jussi_heikkinen.pdf;jsessionid=EF5E48691477EE20149A22C107BD7F1E?sequence=2

Kewate, N., Raut, A., Dubekar, M., Raut, Y. & Patil, A. (2022). A Review on AWS - Cloud Computing Technology. *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, *10*(1), 258-263. https://doi.org/10.22214/ijraset.2022.39802.

Lima, P. (2019). *Migración de aplicaciones monolíticas a arquitecturas basadas en microservicios*. [Trabajo de Fin de Grado, Universidad de La Laguna]. Archivo digital. https://riull.ull.es/xmlui/handle/915/15475

Lin, Y. (2015). *Online store based on Magento E-Commerce*. [Tesis de licenciatura, University of Applied Sciences]. Archivo digital. https://www.theseus.fi/bitstream/handle/10024/90976/ThesisYuLin.pdf?sequence=1

Morizur, D. (2019). *Enhancing Magento Frontend Performance with ReactJS and Comparing It to Knockout.* [Tesis de licenciatura, University of Applied Sciences]. Archivo digital. https://www.theseus.fi/bitstream/handle/10024/161745/Thesis.pdf?sequence=2

Qian, L., Luo, Z., Du, Y. & Guo, L. Cloud Computing: An Overview. *Cloud Computing, First International Conference, CloudCom 2009, Beijing, China,* https://www.researchgate.net/publication/221276709_Cloud_Computing_An_Overview

Rashid, A., & Chaturvedi, A. (2019). Cloud Computing Characteristics and Services: A Brief Review. *International Journal of Computer Sciences and Engineering*, *7*(2), 421-426. https://doi.org/10.26438/ijcse/v7i2.421426.

Sadek, J., Craig, D., & Trenell, M. (2022). Design and Implementation of Medical Searching System Based on Microservices and Serverless Architectures. *Procedia Computer Science, 196*, 615–622. https://doi.org/10.1016/j.procs.2021.12.056.

Salah, T., Zemerly, M. J., Yeun, C. Y., Al-Qutayri, M., & Al-Hammadi, Y. (2017). *Performance Comparison Between Container-based and VM-based Services*. *IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud),* 185-190. https://doi.org/10.1109/FiCloud.2017.33.

Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). *Architectural Patterns for Microservices: A Systematic Mapping Study*. *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018)*, 221-232. https://doi.org/10.5220/0006798302210232.

Velepucha, V. y Flores (2023). A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. *IEEE Access, 11.* 88341. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10220070