Evaluación de la Migración de FaaS a IaaS: Costos, Rendimiento y Desafíos en AWS

Julián Casaburi[1][0009-0005-1660-5930] and Mario Matías Urbieta[2][0000-0002-4508-1209]

[1] Facultad de Informática - Universidad Nacional de La Plata, La Plata BA 1900, ARG
[2] LIFIA, Facultad de Informática - Universidad Nacional de La Plata, La Plata BA 1900, ARG
juliancasaburi@gmail.com, matias.urbieta@lifia.info.unlp.edu.ar

Resumen. En entornos cloud-native, la elección del modelo de servicio constituye un factor determinante para la optimización tanto de los resultados operativos como de los costos asociados. El presente estudio examina el proceso de migración desde un modelo serverless basado en Function-as-a-Service (FaaS), concretamente AWS Lambda, hacia una arquitectura monolítica desplegada sobre Infrastructure-as-a-Service (IaaS), concretamente Amazon EC2.

A través de una serie de pruebas que contemplan diversos tipos de aplicaciones y patrones de carga, se evalúa el comportamiento de ambas arquitecturas en términos de rendimiento y costos. Los resultados obtenidos muestran que, si bien el modelo FaaS proporciona simplicidad y escalado automático, puede implicar un costo significativamente elevado en escenarios de alta demanda. En contraposición, el modelo IaaS ofrece un mayor grado de control y una mayor capacidad de optimización económica, aunque introduce retos adicionales en cuanto a la gestión y el mantenimiento de la infraestructura.

Asimismo, se caracterizan los principales desafíos técnicos y operativos asociados al proceso de migración, identificando patrones comunes y buenas prácticas que pueden orientar futuras transiciones entre modelos de servicio en la nube.

Palabras clave: Computación en la Nube, Función como Servicio (FaaS), Infraestructura como Servicio (IaaS), AWS Lambda, Evaluación de Rendimiento.

Assessing the Migration from FaaS to IaaS: Cost, Performance, and Challenges in AWS

Abstract. In cloud-native environments, the choice of service model is a critical factor in optimizing both operational outcomes and associated costs. This study examines the migration from a serverless Function-as-a-Service (FaaS) model—specifically AWS Lambda—to a monolithic solution deployed on Infrastructure-as-a-Service (IaaS), more specifically Amazon EC2.

Through a series of tests covering different types of applications and workload patterns, the performance and operational costs of both architectures are

evaluated. The results indicate that while the FaaS model offers simplicity and automatic scaling, it may incur significantly higher costs in high-demand scenarios. In contrast, the IaaS model provides greater control and cost optimization potential, though it introduces additional challenges related to infrastructure management and maintenance.

Moreover, the study identifies and characterizes the main technical and operational challenges associated with the migration process, outlining common patterns and best practices to guide future transitions between cloud service models.

Keywords: Cloud Computing, Function as a Service (FaaS), Infrastructure as a Service (IaaS), AWS Lambda, Monolithic Architecture, Performance Benchmarking, Cost Optimization.

1 Introducción

La computación en la nube ha evolucionado a través de distintos modelos de servicio, cada uno con un nivel de abstracción diferente. Infrastructure-as-a-Service (IaaS) se consolidó como uno de los modelos fundamentales, otorgando a las organizaciones un control granular sobre recursos de infraestructura virtualizados, como servidores y almacenamiento. El principal desafío en IaaS ha sido la gestión eficiente de dichos recursos para equilibrar costos y rendimiento, un área de estudio ampliamente documentada (Manvi, Shyam, 2014). Como respuesta a la complejidad operativa de IaaS, surgieron modelos de mayor abstracción. Function-as-a-Service (FaaS), un modelo de computación serverless, representa un cambio de paradigma que permite a los desarrolladores ejecutar código en respuesta a eventos sin gestionar la infraestructura subyacente (Rajan, 2020). AWS Lambda (AWS Lambda – Developer Guide, 2024), es un ejemplo destacado de FaaS, ofreciendo escalado automático, alta disponibilidad y un modelo de precios basado en el uso. Sin embargo, a medida que las aplicaciones crecen en complejidad o escala, FaaS puede enfrentar limitaciones, haciéndolo menos adecuado o incluso inadecuado para ciertos casos de uso.

Los casos de migración en la industria refuerzan las limitaciones de FaaS para aplicaciones de alta demanda. La transición de Amazon Prime Video de AWS Lambda a una solución monolítica en 2023 (Kolny, 2023) es destacable. Amazon Prime Video desarrolló una solución serverless para su servicio de monitoreo de calidad de video, orquestado por AWS Step Functions (What is AWS Step Functions? - Amazon Web Services, 2024) y potenciado por funciones Lambda. Al consolidar las operaciones en una solución monolítica en Amazon EC2, Prime Video logró una reducción de costos de aproximadamente el 90%.

Aunque múltiples proveedores ofrecen servicios FaaS e IaaS, la elección de Amazon Web Services (AWS) para este estudio responde a su posición como proveedor dominante en el mercado de infraestructura en la nube (Richter, 2025).

Por lo expuesto anteriormente, este artículo explora la migración de AWS Lambda a una solución monolítica, a través del análisis de las consideraciones relevantes y la evaluación empírica, motivada por los siguientes factores clave:

- Eficiencia de costos: FaaS puede generar costos más altos en escenarios de alta demanda.
- Utilización de recursos: El modelo FaaS puede resultar en recursos infrautilizados para tareas intensivas en I/O o con alta inactividad, como consultas a bases de datos o llamadas a API. Cada instancia de función atiende una única solicitud, independientemente de sus necesidades de recursos. La migración a IaaS permite el manejo multihilo de múltiples solicitudes dentro de una sola instancia, mejorando la eficiencia de los recursos.
- Limitaciones del modelo de servicio: AWS Lambda impone un tiempo máximo de ejecución de 15 minutos por instancia de función. En algunos casos, las tareas de larga duración no pueden dividirse en múltiples instancias de función, lo que hace que Lambda sea inadecuado. Incluso cuando la división es factible, surgen costos adicionales debido a la orquestación mediante AWS Step Functions, que se cobra según el número de transiciones de estado (What is AWS Step Functions? - Amazon Web Services, 2024).

Este artículo se organiza de la siguiente manera: la sección 2 presenta los trabajos relacionados. La sección 3 introduce los modelos de precios. La sección 4 caracteriza los desafíos asociados a la migración, mientras que la sección 5 expone los resultados obtenidos a partir de los casos de estudio. Finalmente, la sección 6 formula las principales conclusiones y señala posibles líneas de trabajo futuro.

2 Trabajos Relacionados

La migración de arquitecturas monolíticas a microservicios (Newman, 2019; Abgaz et al., 2023), así como a FaaS (Würz et al., 2023; Pedratscher et al., 2022) (un caso particular de microservicios), ha sido un área ampliamente investigada en la literatura, detallando los desafíos y las mejores prácticas. Sin embargo, la dirección opuesta, es decir, la migración de soluciones FaaS a otros tipos, ha sido menos explorada.

Un estudio destacable en este sentido es el realizado en el año 2020 por BBVA, una empresa multinacional de servicios financieros, comparando AWS Lambda con Amazon EC2 (Rodríguez et al., 2018). Este estudio desafió la suposición predominante de que FaaS es universalmente más rentable al examinar patrones de tráfico y distribuciones de carga de trabajo realistas. Los hallazgos indicaron que las ventajas de costos de Lambda disminuyen en escenarios de alto tráfico caracterizados por procesos constantes o de larga duración. Si bien Lambda demostró ser rentable y de bajo mantenimiento para usos esporádicos o de bajo tráfico, EC2 resultó más económico para tráfico estable y de alto volumen. El modelo de precios de Lambda (AWS Lambda Pricing, 2024) rápidamente condujo a costos que superaban los de una configuración altamente disponible en EC2.

Es importante señalar que, tras el estudio de costos de BBVA y la migración de Prime Video, AWS introdujo un modelo de precios escalonado para Lambda en agosto de 2022 (AWS: Introducing tiered pricing for AWS Lambda, 2022). Esta nueva estructura de precios ofrece tarifas diferenciadas según el volumen de GB-segundos consumidos. Por ejemplo, una función Lambda ejecutada en arquitectura x86 con 2048 MB de memoria,

una duración promedio de 60 segundos y 75 millones de invocaciones mensuales tendría los siguientes cambios de costos:

- Sin precios escalonados: USD 150,015.30
- Con precios escalonados: USD 145,015.29

Esto resulta en un ahorro mensual de USD 5,000.01. Por lo tanto, tras esta actualización, los costos de Lambda pueden volverse más competitivos para cargas de trabajo grandes.

Estas modificaciones pueden afectar las conclusiones de análisis de costos anteriores, justificando una reevaluación de los escenarios. Por lo tanto, en este trabajo se realiza un análisis de costos y rendimiento en distintos tipos de escenarios y se evalúan las problemáticas involucradas, con el objetivo de proporcionar una mejor comprensión de cuándo y por qué este tipo de migración puede ser ventajosa.

3 Modelos de Precios

El modelo de precios de AWS Lambda (AWS Lambda Pricing, 2024) se basa en solicitudes y duración (en "GB-segundos"). Por ejemplo, una función que utiliza 1 GB de memoria durante 1 segundo consume 1 GB-segundo. La asignación de memoria varía entre 128 MB y 10,240 MB, con recursos de CPU proporcionales a la memoria. Una mayor asignación de memoria puede reducir el tiempo de ejecución y disminuir los costos.

AWS Lambda cuenta con una estructura de precios escalonados (AWS: Introducing tiered pricing for AWS Lambda, 2022) que ofrece descuentos en función del número total de GB-segundos consumidos, con ahorros de hasta el 20% para el nivel más alto de uso. Este modelo de precios está diferenciado por arquitectura (x86 y ARM64) y región.

Por otro lado, los usuarios de Amazon EC2 son facturados por la capacidad total de sus instancias, independientemente de la utilización real. El precio depende de varios factores de configuración, incluyendo el tipo de instancia, CPU, RAM, capacidades de red y opciones de almacenamiento. AWS ofrece varios planes de pago para adaptarse a diferentes necesidades (How AWS Pricing Works, 2024).

AWS Lambda es particularmente ventajoso para cargas de trabajo con acceso poco frecuente. Sin embargo, puede generar costos más altos que IaaS para cargas de trabajo caracterizadas por volúmenes constantes y altos o demandas significativas de procesamiento.

4 Desafíos de la Migración

La migración no está exenta de desafíos. Existe una serie de problemáticas operativas y organizativas que pueden surgir durante la transición y en la operación posterior. Para cada problema identificado, también proponemos soluciones preliminares para mitigar los posibles impactos negativos.

Implementación de las Características Ofrecidas por Lambda
La migración requiere evaluar diversos aspectos clave, tanto funcionales como no funcionales. Aunque es posible replicar las capacidades de Lambda en EC2, se requieren ajustes significativos en arquitectura y gestión de recursos.

Desde el punto de vista funcional, el código puede provenir de un "Lambdalith" (una sola función que gestiona múltiples endpoints) o de múltiples funciones especializadas, siendo el segundo caso más complejo de migrar. Además, mientras Lambda es inherentemente stateless, una aplicación monolítica puede ser stateful o stateless. Asimismo, la programación basada en eventos en Lambda puede replicarse en EC2 mediante colas de mensajes (Amazon SQS) o webhooks.

En términos de control de tráfico, el rate limiting en Lambda se gestiona con API Gateway, mientras que en EC2 puede implementarse con Redis o a nivel de proxy inverso. Las transacciones en bases de datos requieren adaptar patrones como SAGA (Saga pattern, 2024), y la autenticación puede mantenerse con servicios externos (Auth0, Cognito) o integrarse en la aplicación monolítica mediante JWT o soluciones SSO self-managed.

Desde la perspectiva no funcional, el monitoreo y logging en Lambda se centraliza con CloudWatch, la herramienta de observabilidad nativa de AWS, mientras que en EC2 se puede replicar mediante herramientas como Prometheus y Grafana. Los permisos IAM en Lambda se otorgan a nivel de función, mientras que en EC2 se gestionan mediante roles asignados a instancias. Finalmente, la alta disponibilidad, nativa en Lambda, en EC2 se logra con instancias distribuidas en múltiples Zonas de Disponibilidad, escaladas automáticamente mediante Auto Scaling Groups.

Tiempo de Inactividad Durante las Actualizaciones del Sistema
En una solución monolítica, una falla en un componente puede comprometer la estabilidad de toda la aplicación, provocando posibles interrupciones del servicio (Kaloudis et al., 2024). Para minimizar este riesgo, se recomienda una estrategia de despliegue blue/green, que permite redirigir el tráfico a la versión estable en caso de problemas con la nueva versión.

Incremento en los Costos Operativos Asociados con la Gestión y el Mantenimiento de Infraestructura
La migración a IaaS incrementa los costos operativos debido a la necesidad de gestionar y mantener la infraestructura subyacente. Para mitigar estos costos, se proponen las siguientes soluciones preliminares:

- Aprovisionamiento Automatizado de Infraestructura: Utilizar Infraestructura como Código (p.ej., Terraform) para automatizar el aprovisionamiento y la configuración de los recursos de infraestructura, garantizando consistencia y reduciendo los potenciales errores manuales.
- Escalabilidad Dinámica: Configurar Auto Scaling groups para ajustar automáticamente el número de instancias EC2 según la demanda, o utilizar mecanismos similares en otros entornos de nube para optimizar la asignación de recursos.

- Monitoreo Automatizado y Gestión Proactiva: Configurar alertas automáticas y scripts correctivos para abordar problemas como fallas de servidores, minimizando la necesidad de intervención manual.

Infrautilización de Recursos Debido al Escalado Horizontal
El escalado horizontal puede conducir a una subutilización de recursos. En consecuencia, los costos aumentan sin aprovechar completamente la capacidad de las instancias. Como solución preliminar, se recomienda configurar Grupos de Auto Scaling con un mayor número de instancias pequeñas en lugar de menos instancias más grandes, para lograr una utilización más eficiente.

Incremento en el Tiempo de Aprovisionamiento para el Escalado Horizontal
A diferencia de AWS Lambda, que escala casi instantáneamente, el despliegue en IaaS requiere el inicio de nuevas instancias a medida que aumenta la demanda. Los retrasos en el aprovisionamiento pueden afectar la capacidad de respuesta de la aplicación ante picos de tráfico, impactando la disponibilidad y el rendimiento. Las soluciones propuestas incluyen:

- Escalado Predictivo Combinado con Escalado Dinámico: Implementar escalado predictivo basado en patrones históricos de carga o eventos anticipados, permitiendo el pre-escalado para manejar picos de demanda junto con el escalado dinámico.
- Optimización de AMI: Minimizar el tiempo de inicialización integrando dependencias y configuraciones dentro de la Amazon Machine Image (AMI), una plantilla preconfigurada para lanzar instancias virtuales, en lugar de depender de scripts de inicio.

Incremento en los Riesgos de Seguridad
La migración desde FaaS a IaaS representa un cambio fundamental en el modelo de responsabilidad compartida de la seguridad. Mientras que en Lambda el proveedor se encarga de la seguridad del sistema operativo y la red subyacente, en IaaS estas tareas críticas recaen directamente en el usuario, ampliando la superficie de ataque. Esta transición expone la aplicación a amenazas bien documentadas en la literatura, entre las que destacan la gestión de parches de seguridad, las configuraciones incorrectas de los firewalls y el control de acceso a la infraestructura (Alouffi et al., 2021).

Para mitigar estos riesgos, se proponen las siguientes soluciones:

- Abstracción y segmentación a través de un proxy inverso: Implementar un proxy inverso (por ejemplo, NGINX o AWS Elastic Load Balancer) como intermediario para ocultar las direcciones IP de los servidores de origen, haciéndolos más resistentes a ataques como DDoS. Los atacantes solo atacarían el proxy, que puede ser reforzado y escalado para soportar ataques.

- Actualizaciones automatizadas del sistema operativo: Utilizar herramientas como AWS Systems Manager Patch Manager para programar y automatizar los parches y actualizaciones del sistema operativo.
- Actualizaciones regulares de lenguajes, frameworks y dependencias: Las actualizaciones frecuentes reducen la vulnerabilidad al incorporar las últimas mitigaciones de seguridad.
- Análisis automático de seguridad del código: Integrar el análisis estático de código y la verificación de dependencias en el pipeline CI/CD para identificar vulnerabilidades antes del despliegue.
- Seguridad del acceso SSH: Desactivar el acceso SSH basado en contraseñas, permitiendo solo la autenticación mediante claves públicas o eliminar el acceso SSH por completo si no es necesario.

5 Resultados

Esta sección evalúa la viabilidad de la migración mediante la medición de las diferencias de rendimiento y costo en aplicaciones con distintos requerimientos funcionales y patrones de tráfico. Para cada escenario, el código de la aplicación, las plantillas de Infraestructura como Código y los scripts de pruebas de carga se encuentran disponibles en el repositorio (GitHub Repository – FaaS – IaaS - Applications, Load Testing Scripts, and Infrastructure as Code, 2025) para garantizar la reproducibilidad de las pruebas.

A continuación se describe la metodología para diseñar y llevar a cabo las pruebas de carga, así como las características clave del entorno de despliegue de las aplicaciones evaluadas.

Pruebas de Carga
Las pruebas de carga se realizaron utilizando K6 (Grafana K6, 2024), una herramienta de benchmarking de código abierto, para recopilar métricas detalladas, incluyendo los tiempos de respuesta (por ejemplo, promedio y P99) y las tasas de fallos. K6 se desplegó en una instancia de EC2 m5.2xlarge en una VPC separada dentro de la misma cuenta y región de AWS que las aplicaciones. Cada prueba se repitió tres veces y los resultados se promediaron para asegurar la consistencia.

Lenguajes y Entornos de Ejecución
Todas las aplicaciones fueron desarrolladas en JavaScript utilizando el entorno de ejecución Node.js (versión 20) (Node v20.9.0 (LTS), 2024).

Aplicación de los Conceptos de Migración
Para cada escenario, se utilizaron los conceptos de migración aplicables, mencionados en la sección 4, para obtener aplicaciones análogas y mitigar las problemáticas asociadas. Por ejemplo, todos los escenarios incorporaron los enfoques para la migración de código y la alta disponibilidad.

Aplicaciones Monolíticas
Para las aplicaciones monolíticas, se utilizó el gestor de procesos PM2 en modo clúster (PM2 Documentation - Cluster Mode, 2024) para ejecutar múltiples instancias de la aplicación en paralelo. Esta configuración aprovecha los recursos de hardware disponibles distribuyendo la carga entre varios núcleos de CPU.

Selección de Instancias
Los tipos de instancia se seleccionaron según su capacidad para manejar el número requerido de solicitudes por segundo para cada escenario, manteniendo la rentabilidad. Para los escenarios que utilizan tipos de instancias con créditos escalonados, se configuró el modo de créditos estándar. Las pruebas confirmaron que los créditos obtenidos superaron consistentemente a los créditos consumidos, asegurando un rendimiento sostenido de la CPU durante las pruebas.

Costos

- Se identificaron las configuraciones de recursos óptimas para cada escenario utilizando la aplicación AWS Lambda Power Tuning (Profiling functions with AWS Lambda Power Tuning, 2024) como se recomienda en la documentación de Amazon Web Services.
- El costo de la solución Lambda se calculó utilizando la duración promedio de ejecución de la función según lo informado por Amazon CloudWatch. El costo excluye la capa “Always free” ya que esta es compartida entre múltiples funciones.
- Los costos de API Gateway no se incluyen en la comparación, ya que son idénticos para ambas soluciones. En cuanto a la transferencia de datos, se factura de manera similar para AWS Lambda y Amazon EC2, se excluye de la comparación.
- Para los escenarios que implican instancias EC2 escaladas horizontalmente, los costos incluyen tanto las tarifas fijas como las variables asociadas al Application Load Balancer (ALB).

5.1 Escenarios de Migración Impulsados por Costos

En esta sección se presentan los resultados de tres aplicaciones, donde cada una representa patrones de carga distintos.

Aplicación 1: Patrón de Tráfico Alto y Constante
En este escenario, fue probada una aplicación intensiva en CPU con una tasa de tráfico sostenida. Se utilizaron las siguientes configuraciones: **Tipo de instancia de EC2** t3.small, **Configuración de Lambda** 128 MB de RAM, **Tasa de solicitudes** 100 RPS, mantenida utilizando el ejecutor de tasa constante de k6 (*constant-arrival-rate-executor*).

La Tabla 1 lista los resultados de rendimiento y costos para ambos tipos de solución. Bajo una carga constante de 100 RPS, la solución monolítica superó a AWS Lambda en términos de tiempos de respuesta promedio. En esta configuración, sin considerar

alta disponibilidad, las instancias on-demand de EC2 resultaron en un ahorro de costos del 73.76%, mientras que las instancias reservadas ofrecieron una reducción del 83.48% en comparación con Lambda.

La Figura 1 representa el punto de equilibrio para los dos tipos de soluciones, con y sin alta disponibilidad.

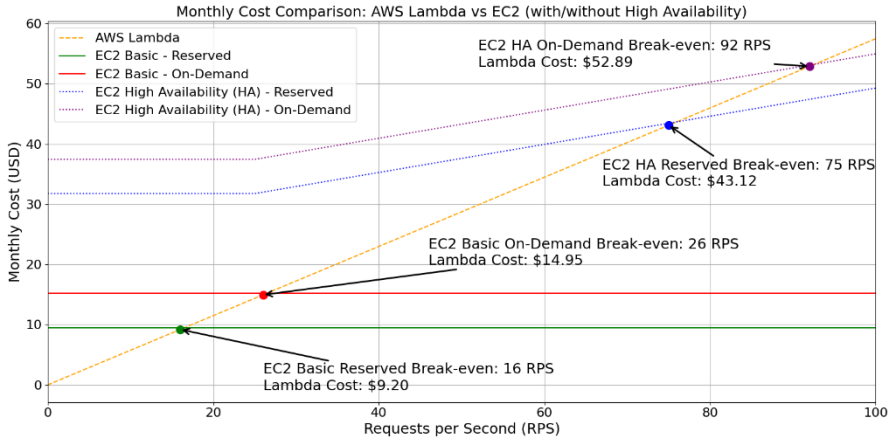

Fig. 1. Punto de equilibrio de diferentes tipos de soluciones para el escenario 1: Patrón de tráfico constante alto (RPS).

En caso de que la alta disponibilidad sea un requerimiento imprescindible, con una configuración de escalado horizontal de al menos dos instancias (por ejemplo, t3.micro) y un Application Load Balancer (ALB) activo, el ahorro mensual frente a AWS Lambda se reduce a 4.38% con instancias on-demand y 14.25% con instancias reservadas. El ALB tiene un costo base fijo de USD 16.43, que se vuelve menos relevante a medida que aumenta el tráfico, mejorando la eficiencia de costos.

También es importante destacar que se pueden lograr ahorros potenciales si el servicio de API Gateway es prescindible en la solución monolítica, dependiendo de las características necesarias como el rate limiting, autenticación, validación y transformación de payloads. Por ejemplo, en el escenario probado, el costo mensual de usar API Gateway es de USD 262.80 para una API HTTP o USD 919.80 para una API REST.

Aplicación 2: Tráfico Impredecible

Este escenario evaluó el comportamiento de una aplicación ante picos repentinos de tráfico para analizar las capacidades de escalabilidad de las arquitecturas monolíticas. Se probaron configuraciones específicas para comparar el rendimiento y los costos entre AWS Lambda y EC2. Se utilizaron las siguientes configuraciones: **Tipo de instancia de EC2** t3.small, **Configuración de Lambda** 1536 MB de RAM, **Tasa de solicitudes** de 10 RPS y 20 RPS.

Los resultados mostraron que la solución monolítica logró tiempos de respuesta más bajos consistentemente en ambas tasas de solicitud. Sin embargo, la capacidad de

adaptación rápida de Lambda permitió manejar los picos sin necesidad de aprovisionamiento manual o preescalado.

La Figura 2 ilustra el punto de equilibrio entre ambas soluciones, considerando los costos operativos y el impacto del tráfico en la escalabilidad de cada enfoque.

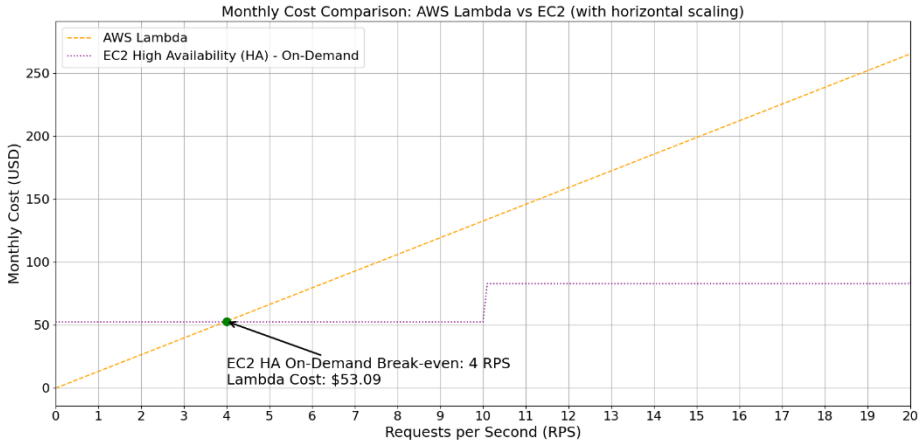

Fig. 2. Punto de equilibrio de diferentes tipos de soluciones para el escenario 2: Tráfico Impredecible.

A pesar de mantener cuatro instancias activas en todo momento, la solución basada en AWS Lambda continúa siendo más costosa en comparación con la alternativa en EC2. Al implementar instancias EC2 con escalado automático y un balanceador de carga de aplicaciones, se logró una reducción de costos del 43.82%.

Aplicación 3: Limitada por I/O

En este escenario, se evaluó una aplicación limitada por I/O. Como se mencionó en la introducción, el modelo de ejecución de AWS Lambda incurre en costos incluso durante los períodos de inactividad, como cuando la aplicación espera respuestas de API de terceros, consultas a bases de datos u otras operaciones de I/O. Este tiempo de espera puede reducir significativamente la eficiencia de costos de Lambda en estos casos de uso.

Este escenario fue identificado como una posible área para pruebas futuras en un estudio comparativo previo (Villamizar et al., 2017), lo cual refuerza la relevancia de analizar el impacto de la latencia de I/O en la viabilidad económica de Lambda frente a soluciones basadas en EC2.

Se utilizaron las siguientes configuraciones: **Tipo de instancia de EC2** m5.large (bajo demanda), **Configuración de Lambda** 128 MB de RAM, **Tasa de solicitudes** 4000 RPS.

La Tabla 1 presenta los resultados de rendimiento y costos para ambas soluciones. La Figura 3 ilustra el punto de equilibrio entre ambas soluciones, considerando los costos operativos y el impacto del tráfico en la escalabilidad de cada enfoque.

La aplicación monolítica mostró tiempos de respuesta menores en comparación con Lambda, lo que evidencia una mayor eficiencia en la gestión de solicitudes de alta concurrencia. Además, la solución basada en Lambda incurrió en costos significativamente más altos, con ahorros mensuales del 70.69% para EC2 bajo demanda y del 72.10% para instancias reservadas al utilizar el enfoque monolítico.

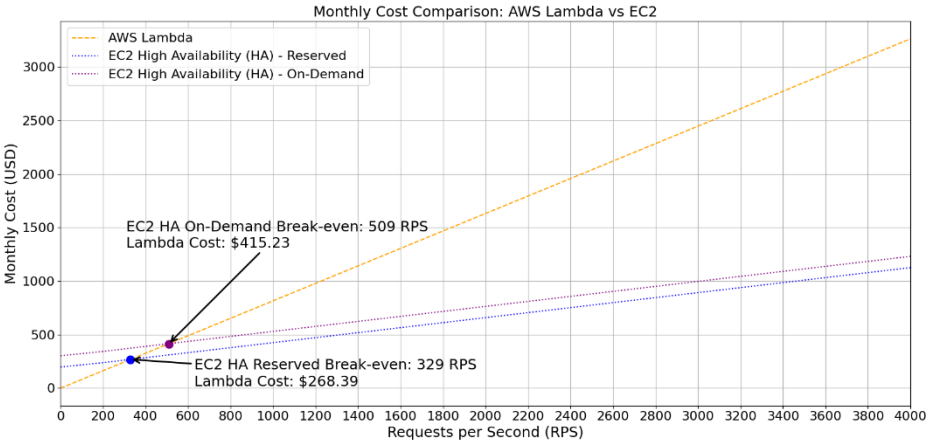

Fig. 3. Punto de equilibrio de diferentes tipos de soluciones para el escenario 3: Limitado por I/O.

5.2 Escenarios de Migración Impulsados por Limitaciones de FaaS

Aplicación 4: Aplicación de Larga Duración

El ejemplo propuesto para este escenario involucra una función activada por la carga de archivos de video a un bucket de Amazon S3. La función descarga el archivo al almacenamiento efímero de la instancia, lo transcodifica utilizando FFmpeg para reducir su bitrate y tamaño, y luego almacena el archivo procesado en otro bucket de S3.

Al subir al bucket de entrada un video de 43 minutos y 32 segundos de duración, con dimensiones 1280x720, bitrate de 584 kbit/s y una tasa de cuadros de 30, el procesamiento excede el límite de ejecución de una instancia de función Lambda, es decir, 15 minutos.

En este escenario, ante el cambio en los requerimientos, es necesario migrar a una solución que permita el procesamiento de larga duración. Se analizaron dos alternativas: una solución Lambda orquestada por una Step Function implica dividir el video, procesar cada parte y luego fusionarlas. Sin embargo, incurre en costos adicionales por las transiciones de estado y las invocaciones de Lambda. En contraste, una solución monolítica que implica el manejo de mensajes a través de SQS y el procesamiento mediante instancias de EC2. El escalado horizontal es gestionado por una target tracking policy (Target tracking scaling policies for Amazon EC2 Auto Scaling - Amazon Web Services, 2024), que ajusta la cantidad de instancias según la longitud de la cola (medida mediante la métrica ApproximateNumberOfMessagesVisible, reportada por SQS).

La prueba fue realizada utilizando las siguientes configuraciones: **Tipo de instancia** c6a.4xlarge (bajo demanda), **Configuración de Lambda** 3008 MB de RAM (para todas las funciones, tanto en la solución de Lambda única como en las funciones de la solución orquestada por una Step Function).

Los resultados, listados en la Tabla 1, muestran que la solución monolítica es más rápida y menos costosa.

5.3 Resumen de Resultados

Esta sección presenta los resultados obtenidos tras la comparación de costos y tiempos de respuesta, para todos los escenarios.

Table 1. Comparación de Costos y Tiempos de Respuesta.

Escenario	Lambda (Costo)	EC2 On-demand (Costo)	Lambda (Tiempo)	EC2 (Tiempo)	Diferencia
1: Tráfico alto y constante	57,49 USD	15,18 USD (73,60% ahorro)	25,79 ms	7,04 ms	-72,69%
2: Tráfico impredecible	147,90 USD	83,08 USD (43,82% ahorro)	220,36 ms	178,92 ms	-18,81%
3: Limitado por I/O	3.723 USD	1.090,8 USD (70,69% ahorro)	73,01 ms	59,20 ms	-18,91%
4: Tarea de larga duración	0,06483 USD	0,02856 USD (55,93% ahorro)	3 min 34 s	2 min 48 s	-21,50%

6 Conclusiones y Trabajos Futuros

Este artículo analizó la migración de una solución FaaS, desplegada en AWS Lambda, a una solución monolítica sobre Infrastructure-as-a-Service (IaaS), destacando los desafíos, beneficios y consideraciones clave.

En los escenarios probados se han logrado ahorros de hasta 80% en costos de infraestructura, especialmente si no se requieren características como alta disponibilidad, tal como se observó en el escenario 1. Aunque los ahorros son inferiores en otros casos, aún son significativos. Por lo tanto, se recomienda utilizar AWS Lambda para tareas internas o de baja demanda, mientras que, para otros casos, los arquitectos deben evaluar cuidadosamente los requerimientos y costos antes de decidir.

Además, dado que la migración no está exenta de desafíos, se plantearon soluciones preliminares a problemáticas comunes.

Se destaca que las pruebas fueron ejecutadas en un entorno académico, por lo que se sugiere realizar una validación en un contexto industrial para obtener una evaluación más completa de los costos.

Finalmente, se plantean líneas de investigación futura, que podrían complementar y ampliar los hallazgos obtenidos:

- Evaluación de frameworks FaaS de código abierto: Analizar la viabilidad de migrar a soluciones FaaS autogestionadas, como OpenFaaS (OpenFaaS, 2024) o Knative (Knative, 2024), planteando un escenario de migración intermedio, para determinar si ofrecen una mejor relación costo-rendimiento en comparación con AWS Lambda.
- Validación en otras plataformas cloud: Extender el estudio a otros proveedores cloud, como Microsoft Azure y Google Cloud, para comprobar si los hallazgos sobre costos y rendimiento son generalizables fuera del ecosistema de AWS.
- Automatización de la migración: Para concluir, es necesario analizar tanto los enfoques manuales como los automatizados, lo que puede implicar la creación de herramientas específicas para reducir la intervención manual y minimizar los errores asociados.

References

Manvi, S. S., & Shyam, G. K. (2014). Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. Journal of network and computer applications, 41, 424-440.

Rajan, A. P. (2020). A review on serverless architectures-function as a service (FaaS) in cloud computing. TELKOMNIKA (Telecommunication Computing Electronics and Control), 18(1), 530-537.

AWS Lambda - Developer Guide (2024). https://docs.aws.amazon.com/lambda/latest/dg/welcome.html, last accessed 2024/02/19.

Kolny, M. Scaling up the Prime Video audio/video monitoring service and reducing costs by 90% (2023). https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90, last accessed 2024/02/19.

Richter, F. Amazon and Microsoft Stay Ahead in Global Cloud Market (2025). https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/, last accessed 2025/06/09.

What is AWS Step Functions? - Amazon Web Services (2024). https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html, last accessed 2024/03/19.

Newman, S. (2019). Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media.

Abgaz, Y., et al. (2023). Decomposition of monolith applications into microservices architectures: A systematic review. IEEE Trans. Softw. Eng. 49(8), 4213–4242. https://doi.org/10.1109/TSE.2023.3287297.

Würz, H.M., Krämer, M., Kaster, M., Kuijper, A. (2023). Migrating monolithic applications to function as a service. Softw. Pract. Exp. 53(12), 2353–2373. https://doi.org/10.1002/spe.3263.
Pedratscher, S., Ristov, S., Fahringer, T. (2022). M2FaaS: Transparent and fault-tolerant FaaSification of Node.js monolith code blocks. Future Gener. Comput. Syst. 135, 57–71. https://doi.org/10.1016/j.future.2022.04.021.
Alda Rodríguez, Á., Álvarez, F., Díaz López, G., Evgeniev, M., Horrillo, P. Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. https://www.bbva.com/en/innovation/economics-of-serverless/, last accessed 2024/03/21.
AWS (2024). Introducing tiered pricing for AWS Lambda. https://aws.amazon.com/blogs/compute/introducing-tiered-pricing-for-aws-lambda/, last accessed 2024/02/19.
AWS Lambda Pricing (2024). https://aws.amazon.com/lambda/pricing/, last accessed 2024/02/19.
How AWS Pricing Works (2024). https://docs.aws.amazon.com/whitepapers/latest/how-aws-pricing-works/amazon-ec2.html, last accessed 2024/04/01.
Saga pattern (2024). https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/saga-pattern.html, last accessed 2024/12/04.
Kaloudis, M. (2024). Evolving software architectures from monolithic systems to resilient microservices: Best practices, challenges, and future trends. Int. J. Adv. Comput. Sci. Appl. 15(9). https://doi.org/10.14569/IJACSA.2024.0150901.
Alouffi, B., Hasnain, M., Alharbi, A., Alosaimi, W., Alyami, H., & Ayaz, M. (2021). A systematic literature review on cloud computing security: threats and mitigation strategies. Ieee Access, 9, 57792-5780
GitHub Repository – FaaS – IaaS - Applications, Load Testing Scripts, and Infrastructure as Code. https://github.com/juliancasaburi/jaiio-faas, last accessed 2025/06/09.
Grafana K6. https://k6.io/, last accessed 2024/07/16.
Node v20.9.0 (LTS) (2024). https://nodejs.org/en/blog/release/v20.9.0, last accessed 2024/02/19.
PM2 Documentation - Cluster Mode (2024). https://pm2.keymetrics.io/docs/usage/cluster-mode/, last accessed 2024/07/16.
Profiling functions with AWS Lambda Power Tuning (2024). https://docs.aws.amazon.com/lambda/latest/operatorguide/profile-functions.html, last accessed 2024/08/08.
Villamizar, M., Garcés, O., Ochoa, L., et al. (2017). Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. Service Oriented Comput. Appl. 11, 233–247. https://doi.org/10.1007/s11761-017-0208-y.
Target tracking scaling policies for Amazon EC2 Auto Scaling - Amazon Web Services (2024). https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-target-tracking.html, last accessed 2024/10/19.
OpenFaaS - Serverless Functions, Made Simple (2024). https://www.openfaas.com/, last accessed 2024/12/14.
Knative (2024). https://knative.dev/docs/, last accessed 2024/12/14.