UNIVERSIDAD DON BOSCO

DIRECCIÓN DE EDUCACIÓN A DISTANCIA

TRABAJO DE GRADUACIÓN:

COMPARATIVA DE ARQUITECTURAS SERVERLESS Y BASADAS EN CONTENEDORES EN ENTORNOS DE NUBE

PARA OPTAR AL GRADO DE:

MAESTRO EN ARQUITECTURA DE SOFTWARE

AUTORES:

YURI ERNESTO CALDERÓN RODRÍGUEZ

MOISÉS ERNESTO MORENO CARTAGENA

ASESOR:

Dr. MANUEL CARDONA

Antiguo Cuscatlán, La Libertad El Salvador C. A.

DICIEMBRE, 2025

Rector Universidad Don Bosco

Dr. Mario Rafael Olmos

Secretaria General

Inga. Yesenia Xiomara Martínez Oviedo

Director de Educación a Distancia

Dr. Eduardo Menjívar Valencia

Director de la Maestría

Mg. Vicent Ramon Palasí Lallana

Asesor del proyecto de graduación

Dr. Manuel Cardona

Lectora del proyecto de graduación

Mg. Karens Medrano

Agradecimientos

Yuri Ernesto Calderon Rodriguez

Agradezco a Dios por darme la vitalidad para continuar adelante aún cuando mis fuerzas flaquearon, su influencia guia el camino y se refleja en el contenido de este documento. Agradezco a mi esposa Lucrecia por darme el apoyo necesario y recordarme que no estoy solo en mi desarrollo profesional, la carga es liviana si es llevada por dos. Agradezco a mis 2 hijos Gloria e Isaac por ayudarme a encontrar la motivación para seguir adelante.

Moisés Ernesto Moreno Cartagena

Primero presento mis agradecimientos a Dios. El poder haber finalizado este proceso es parte no tan solo del trabajo duro realizado durante estos años, sino que también abarca un apoyo familiar permanente, por tal motivo agradezco a mi madre Sonia Cartagena, a mi hermano Mario y mi hermana Diana por motivarme e inculcarme el esfuerzo y perseverancia para mantenerme actualizado en los conocimientos de este campo profesional.

A nuestro asesor Dr. Manuel Cardona, por su eficacia en la orientación de alcanzar nuestro objetivo.

Resumen

Este documento presenta un análisis comparativo de las arquitecturas sin servidor y basadas en contenedores de entornos en la nube, centrándose en tres parámetros: escalabilidad, rentabilidad y rendimiento. Utilizamos infraestructuras en la nube, Amazon AWS, Google Clud y Microsoft Azure, todos como proveedores de prueba para el estudio, dentro de cada una de las infraestructuras empleamos tecnologías como AWS Lambda, Cloud Run (function) y Apps Functions; para contenedores utilizamos Elastic Container Service (ECS), Cloud Run (container) y Containers Instances.

Realizamos pruebas de carga por medio de la herramienta Locust simulando solicitudes en simultaneo de muchos usuarios, medimos los tiempos de respuesta, las tasas de fallo y la utilización de recursos. Los resultados mostrarán que arquitectura es adecuada y en que escenario deben emplearse, esto en la evaluación de escalabilidad, rentabilidad y rendimiento; las arquitecturas serverless muestran un mejor rendimiento cuando existen cargas variables o impredecibles, mientras las arquitecturas basadas en contenedores ofrecen un rendimiento y una consistencia superiores para cargas de trabajo sostenidas y predecibles.

Proporcionamos resultados con información útil para que los equipos de desarrollo seleccionen la arquitectura adecuada en función de los requisitos específicos de la aplicación a desplegar, optimizando así la asignación de recursos y minimizando los costos.

Palabras claves: Sin servidor, contenedores, computación en la nube, es-

calabilidad, rentabilidad, rendimiento

Índice

Índice de figuras

Índice de tablas

Lista de abreviaturas

TI	Tecnología de la Información
FaaS	Function as a Service
API	Application Programming Interface
RPS	Request Per Second
AWS	Amazon Web Services
ECS	Elastic Container Service
EKS	Elastic Kubernetes Service
LXC	Linux Containers
GCP	Google Cloud Plataform
GKE	Google Kubernetes Engine
AKS	Azure Kubernetes Service
AZR	Azure
IAAS	infraestructura como servicio

Introducción

En años recientes ha habido un auge importante en el uso de computación en la nube, arquitecturas emergentes como serverless y otras basadas en contenedores se han convertido en el estandard de despliegue para la mayoría de aplicaciones existentes. Estas tecnologías han tomado dos caminos principales, por un lado las arquitecturas Serverless permiten a los desarrolladores despreocuparse de la infraestructura sobre las que deben correr sus aplicaciones y en el otro se encuentran aquellas arquitecturas que con cierta flexibilidad permiten un control fino de las infraestructura para despliegue de sistemas.

Ante esta situación surge la necesidad de una comparativa objetiva de ambas tecnologías, esta investigación pretende dar respuesta a que casos de uso aplican según cada una de ellas; actualmente los temas investigados y relacionados han sido variados, algunos han puesto su enfoque las arquitecturas serverles Hassan, Barakat, y Sarhan (2021); Kodakandla (2021); y Menéndez, Gayo, Canal, y Fernández (2023). Las anteriores investigaciones han tratado de dar respuesta a preguntas como, ¿Quienes están investigando respecto al tema y su relevancia?, ¿Existe alguna diferencia entre serverless y la computación en la nube tradicional?, ¿Qué beneficios aporta esta tecnología emergente?, ¿Cuáles son las plataformas más utilizadas?, ¿Cuáles son los desafíos que afronta esta arquitectura?, ¿Existen migraciones en curso de la arquitectura de nube tradicional a serverless?.

También hemos investigado la literatura existente referente a la tecnología

de los contenedores, por ejemplo Velp, Rivière, y Sadre (2020);. Y nos hemos encontrado con estudios con metodologías rigurosas para la evaluación de diferentes configuraciones (72 configuraciones válidas). Con enfoques prácticos donde los resultados se aplican a casos de uso real y con una profundidad técnica donde se analizan interacciones complejas entre componentes y se comparan tecnologías emergentes (Firecracker, Kata Containers) con soluciones tradicionales (Docker, LXC).

Además han existido investigaciones que se aproximan a una comparativa entre las tecnologías investigadas en este estudio, pero aplicada a escenarios específicos, como es el caso del estudio aplicado a pipelines de datos Lekkala (2023). En esta comparativa, se tuvieron en cuenta varios enfoques, como el técnico, dónde se analizó las ventajas y desventajas de cada enfoque. Algunas de las variables medidas para el rendimiento fueron la latencia, escalabilidad y eficiencia. Dentro de los costos se midieron el pago por uso (para el caso de serverless) y los costos fijos (para los contenedores).

Sin embargo, el no poder distinguir cuál tecnología es mejor aplicable en qué escenario representa un problema ya que impide que un equipo de desarrollo no sea capaz de adaptar su tecnología a necesidades concretas, porque cada arquitectura tiene sus ventajas y desventajas en términos de escalabilidad, costes, rendimiento, etc.

Por eso la importancia de conocer acerca de estas arquitecturas, para evitar malas decisiones en la implementación de estas tecnologías que podrían derivar

en retrasos, sobrecostos, limitaciones en la evolución de los sistemas o problemas de integración. Lo que se quiere lograr entonces con esta investigación es ofrecer una comparativa de ambas arquitecturas en los términos anteriormente mencionados para entender mejor las fortalezas y debilidades de cada una de las arquitecturas.

Para hacer la comparativa escogimos Amazon Web Services (AWS), Google Cloud y Microsoft Azure como proveedores de la nube por ser de los más ampliamente usados y reconocidos. Además, utilizamos tecnologías de despliegue para poder correr un pequeño código de prueba: por mencionoar un escenario, corrimos un servicio completo de Serverless como lo es AWS Lambda, y uno intermedio que abarca ambas tecnologías, como lo es ECS. Aparte, para las pruebas de carga utilizamos Locust, una solución de código abierto.

Este trabajo está organizado de la siguiente forma: Formulación del problema, que presenta las variables consideradas en el estudio; Justificación, en la que se expone el motivo que condujo a la investigación y los beneficios esperados; Objetivo general, describe lo que se logrará, para qué y cómo se desarrollará; Objetivos específicos, que comprenden el desarrollo del marco teórico, la justificación del trabajo y la solución propuesta; Marco Teórico y Estado del Arte, reúne referencias a investigaciones previas; Hipótesis de la investigación, la afirmación anticipada; Metodología de investigación, donde se describe el diseño metodológico, las variables empleadas, la muestra considerada, los instrumentos utilizados y el procedimiento de recolección de datos; Presentación de

resultados, incluye las tablas y los gráficos; Discusión, núcleo interpretativo del estudio, que analiza los hallazgos y aborda las limitaciones y su impacto; y, finalmente, las Conclusiones.

Definición del tema de investigación

El presente estudio centra sus objetivos en realizar una comparación entre arquitecturas *serverless* y arquitecturas basadas en contenedores con el fin de poder aportar una visión cuantitativa sobre métricas de selección para toma de decisiones sobre cuando conviene una u otra tecnología. Esta temática resulta relevante por la gran cantidad de aplicaciones que se están desarrollando de las cuales deben ser desplegadas en contenedores o arquitecturas serverless, pretendemos que el lector pueda tener elementos de juicio sólidos para tomas la mejor decisión de despliegue.

La investigación se delimita al análisis de las características, ventajas, limitaciones y casos de uso de ambos enfoques, considerando dimensiones fundamentales como la gestión de recursos, el rendimiento, la elasticidad y los modelos de facturación; Se examinan sus comportamientos en plataformas ampliamente adoptadas en la industria tecnológica, tales como Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform (GCP).

Asimismo, el estudio se sustenta en una amplia gama de documentación técnica y académica que demuestra con mayor profundidad la necesidad de establecer con precisión criterios comparativos claros, especialmente ante la gran diversidad, complejidad y especialización de los servicios disponibles, así como frente a la rápida y constante evolución de los modelos computacionales que se desarrollan y ofrecen en la nube. En este sentido, la investigación no solo pretende organizar y clarificar dicho panorama, sino también proveer una base

conceptual, metodológica y práctica más completa, que facilite comprender con exactitud, detalle y fundamento en qué escenarios, contextos o tipos de proyecto resulta más apropiada, eficiente o ventajosa la adopción de cada arquitectura tecnológica.

Formulación del problema

A pesar de los avances en la literatura relacionada con la comparación entre arquitecturas *serverless* y arquitecturas basadas en contenedores, persiste un vacío de conocimiento debido a la falta de estudios que integren de forma actualizada y que evalúen con evidencia cuantitativa las tecnologías vigentes y sus implicaciones en entornos reales de computación en la nube. Este vacío dificulta la identificación de criterios sistemáticos para seleccionar la arquitectura más adecuada en función de necesidades específicas.

En este contexto, surge el problema central de investigación: ¿cómo influyen la eficiencia de costos, la elasticidad efectiva, el desempeño bajo cargas impredecibles y la complejidad operativa en la elección entre arquitecturas *serverless* y basadas en contenedores? Estas variables, fundamentales para la toma de decisiones arquitectónicas, requieren un análisis comparativo riguroso que considere tanto métricas técnicas como factores operacionales.

El abordaje de este problema es relevante desde una perspectiva científica y tecnológica, ya que permitirá establecer lineamientos que faciliten la selección óptima de arquitecturas en proyectos de nueva generación, particularmente en sectores que demandan servicios digitales dinámicos y de alta disponibilidad. Con ello, la investigación busca contribuir a una toma de decisiones más informada y basada en evidencia.

Justificación

La presente investigación se justifica debido a la necesidad real y creciente de contar con criterios rigurosos que orienten la selección entre arquitecturas *serverless* y arquitecturas basadas en contenedores en entornos de computación en la nube. La motivación principal radica en la ausencia de un análisis comparativo actualizado y sistemático que integre las tecnologías vigentes y sus implicaciones técnicas y operativas, lo cual genera incertidumbre en la toma de decisiones arquitectónicas dentro de organizaciones y proyectos tecnológicos.

Asimismo, el estudio resulta oportuno porque las empresas y desarrolladores se enfrentan de manera constante al desafío de escoger el modelo de despliegue más adecuado para aplicaciones escalables, decisión que repercute directamente en costos, desempeño, elasticidad y complejidad de mantenimiento. En un escenario donde la adopción de la nube continúa acelerándose, comprender las ventajas y limitaciones de cada enfoque no solo es relevante, sino imprescindible para la sostenibilidad tecnológica.

Finalmente, la investigación es significativa porque sus resultados permitirán a profesionales, equipos de desarrollo e instituciones fundamentar sus decisiones en evidencia empírica y no únicamente en tendencias o percepciones del mercado. Al ofrecer lineamientos claros y comparativos, este trabajo contribuye a la optimización del diseño arquitectónico y, en consecuencia, al incremento de la eficiencia operativa en diversos sectores que dependen de sistemas digitales dinámicos.

Objetivo general

Analizar y comparar las principales plataformas proveedoras de servicios en la nube, *serverless* y basadas en contenedores, con el objetivo de medir cuantitativamente la escalabilidad el costo y el desempeño, esto permitirá conocer las ventajas y limitaciones, mediante la revisión sistemática de documentación técnica, estudios previos y pruebas cuantitativas comprobables y replicables.

Objetivos específicos

- Desarrollar el marco teórico que sustente conceptualmente las arquitecturas serverless y basadas en contenedores, integrando definiciones, antecedentes y estudios relevantes.
- Justificar la pertinencia del estudio mediante el análisis de la problemática actual, la relevancia tecnológica y la necesidad de criterios comparativos para la toma de decisiones arquitectónicas.
- Proponer una solución metodológica que permita evaluar objetivamente y contrastar ambos enfoques de manera cuantitativa, estableciendo criterios precisos para su comparación.

Marco Teórico y Estado del Arte

Marco Teórico

Son diversas las razones que han llevado a los profesionales de TI, especialmente a los arquitectos de software, a optar por el uso de tecnologías serverless y contenerizadas. La primera razón que mencionaremos es la reducción de la complejidad en el manejo de infraestructura a la hora de la programación. Con el crecimiento de las plataformas Function-as-a-Service (FaaS), como AWS Lambda, Google Cloud Functions y Azure Functions, las arquitecturas serverless se han vuelto cada vez más populares en diversos sectores debido a su capacidad de escalar fácilmente y a su bajo costo operativo (Li, Lin, Wang, Ye, y Xu, 2023).

El principal atractivo de la computación serverless radica en su capacidad para asignar recursos de manera dinámica, permitiendo que las aplicaciones se escalen automáticamente según la demanda de trabajo (Yao, Chen, Yuan, y Ou, 2023). A pesar de sus beneficios, esta arquitectura también presenta desafíos, como la latencia en los arranques en frío, la variabilidad del rendimiento y los riesgos de seguridad (Sharma, 2023). Estas limitaciones hacen necesario desarrollar estrategias de optimización innovadoras que mejoren su desempeño y fiabilidad (Ghorbian y Ghobaei-Arani, 2024).

Podemos mencionar otra razón, la popularidad e incremento de la utilización de arquitecturas de microservicio, resaltando el uso de Docker (Merkel, 2014). Los contenedores representan una forma de virtualización a nivel del sistema operativo que permite crear entornos de ejecución independientes con sis-

temas de archivos aislados. Las imágenes de contenedor están compuestas por capas incrementales que solo contienen los archivos y bibliotecas que no están presentes en el sistema operativo base, lo cual permite generar instancias livianas y con poca sobrecarga en comparación con otras formas de virtualización. Gracias a su capacidad para encapsular, aislar y escalar microservicios, los contenedores se consideran una de las tecnologías clave que habilitan esta arquitectura.

La veloz evolución de la tecnología de la información (TI) ha revolucionado tanto las industrias como la vida diaria. Sin embargo, este avance también ha generado desafíos ambientales significativos, particularmente relacionados con el consumo energético y las emisiones de gases de efecto invernadero provenientes de los centros de datos y la infraestructura en la nube (Katal, Dahiya, y Choudhury, 2023; Yaqub y Alsabban, 2023). Ante el crecimiento continuo de la demanda global de servicios en la nube, mejorar su eficiencia, capacidad de expansión y sostenibilidad se ha convertido en una prioridad esencial para investigadores y profesionales del sector (Chiang, 2024).

Arquitectura Serverless

Empezaremos la descripción de la arquitectura serverless mencionando algunos de sus principales conceptos que la definen, las plataformas más comunes donde se puede implementar esta arquitectura y mostrando algunos casos de uso de la misma.

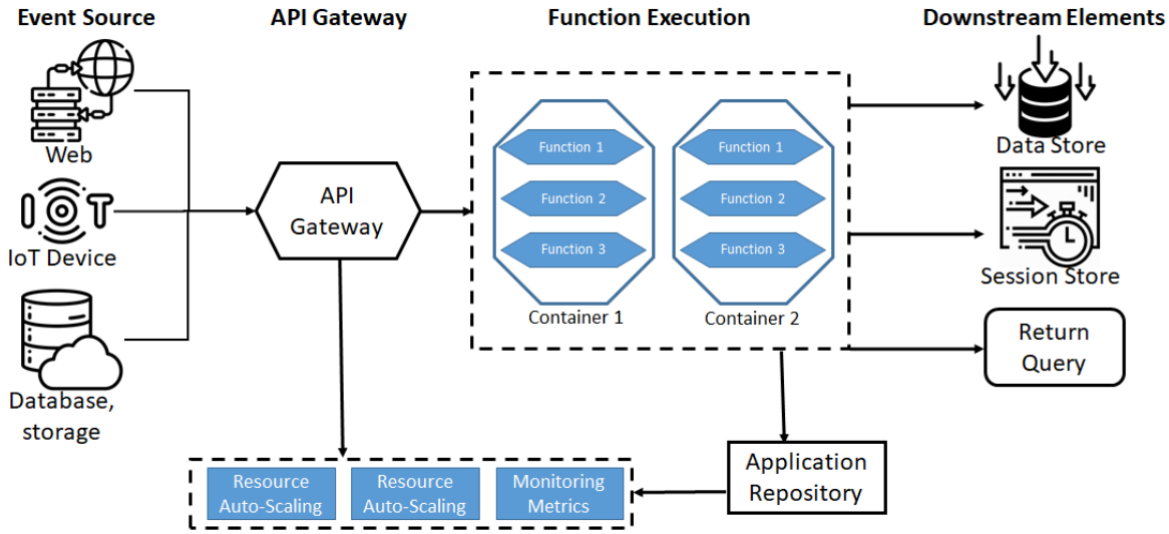
Definición y conceptos básicos. El concepto se remonta al año 2014 cuando Amazon introdujo su producto llamado Amazon Lambda. Serverless busca agre-

gar una capa de abstracción a los existentes paradigmas de computación en la nube con el objetivo de simplificar o eliminar las tareas relacionadas con la administración de servidores para que los programadores se puedan centrar en tareas más relacionadas a su área (Hassan y cols., 2021). Desde entonces, esta tecnología se ha convertido en un paradigma emergente de la computación en la nube adoptado por muchas aplicaciones (Wen, Chen, Jin, y Liu, 2023). El desarrollador se encarga de implementar la lógica de cada función, la cual genera una URL específica. Dado que la arquitectura serverless opera de manera orientada a eventos, los usuarios pueden activar la función correspondiente mediante el acceso a dicha URL. Para ello, se configura un entorno de ejecución particular, empaquetado con Docker y almacenado en un repositorio, con el fin de satisfacer los requisitos de la función. Cuando los usuarios acceden a las URLs, las solicitudes se canalizan a través de la puerta de enlace de la API. Posteriormente, el programador gestiona la carga de trabajo entrante, obtiene la información necesaria de la función y carga el entorno de ejecución para llevar a cabo su ejecución.

Para entender más a fondo esta arquitectura es necesario comprender algunos conceptos primero, como el concepto de ’cold start’ (Rad y Ghobaei-Arani, 2025). En el contexto de la computación en la nube, un cold start o ’inicio en frio’, se refiere al tiempo de espera adicional que ocurre cuando un servicio serverless debe inicializar un entorno de ejecución desde cero para manejar una solicitud entrante. Esto sucede cuando no hay instancias activas o ’calientes’, disponibles

Figura 1

Arquitectura Serverless



Nota. La figura muestra el funcionamiento de la Arquitectura Sereverless. Tomada de Akour y Alenezi (2025).

para procesar dicha solicitud inmediatamente. Durante este tiempo de espera la plataforma asigna recursos (como CPU y memoria), carga el entorno y el código e inicializa las dependencias necesarias. Este proceso agrega latencia, por eso forma parte de las variables en los estudios de comparación.

Otro concepto muy relacionado a los entornos de nube es la seguridad (Vaño, Lacalle, Sowiński, S-Julián, y Palau, 2023), y con ello nos referimos al conjunto de tecnologías y controles para proteger los datos, las aplicaciones, servicios y la infraestructura en sí. Su objetivo principal es garantizar la con-fiabilidad, integridad, (Mondal, Pan, y Kabir, 2022) disponibilidad y privacidad de la información frente a amenazas tanto internas como externas. Se incluyen

aspectos como control de acceso a recursos, cifrado de datos, y monitoreo de actividades en la nube.

Plataformas más comunes. Las plataformas más comunes utilizadas para este tipo de arquitectura son: AWS Lambda, Azure Functions y Cloud Function (Chippagiri, 2025). Estas plataformas están gestionadas por 3 empresas de reconocimiento mundial que ofrecen plataformas para servicios en la nube: Amazon, Google y Microsoft respectivamente.

Casos de uso. El estudio de Menéndez y cols. (2023) se encuentra en una fase de investigación aplicada, con prototipos funcionales y resultados validados experimentalmente. Sin embargo, los autores señalan que: Las pruebas se realizaron en una aplicación simple (e-commerce básico), por lo que los resultados podrían variar en sistemas complejos.

Se identifican limitaciones, como el tiempo de aprovisionamiento en Serverless (ej.: cold start de AWS Lambda) y la necesidad de reescribir código para adaptarse a FaaS. El trabajo futuro incluye extender la comparación a otros proveedores de nube (Azure, Google Cloud) y aplicar las lecciones a proyectos reales.

Además del comercio electrónico, la arquitectura sin servidor se puede aplicar en una amplia gama de sectores industriales, como la sanidad, los medios de comunicación y el entretenimiento, los servicios financieros, el transporte y la logística, y la educación (Ghorbian y Ghobaei-Arani, 2025).

Arquitectura basada en contenedores

La arquitectura basada en contenedores es un enfoque de desarrollo y despliegue de software en el que las aplicaciones y sus dependencias se agrupan en unidades ligeras (Akour y Alenezi, 2025) y portátiles llamadas contenedores. Estos contenedores se ejecutan de una forma aislada sobre un mismo sistema operativo, lo que permite una mayor eficiencia en el uso de recursos comparado con las máquinas virtuales tradicionales.

¿Es esta tecnología solo una tendencia pasajera o realmente aporta un valor significativo? Más allá del entusiasmo inicial, esta tecnología se presenta como una fuerza de cambio profunda que ha ganado popularidad por su escalabilidad y portabilidad (Brogi y cols., 2016). Aprovecha las bases de tecnologías previas mientras introduce ideas innovadoras para crear sistemas más eficientes, superando el enfoque tradicional de los mainframes (Felter, Ferreira, Rajamony, y Rubio, 2015). Los contenedores representan una nueva etapa en la evolución de la computación distribuida, cuyo propósito es optimizar el uso de recursos distribuidos y combinarlos para mejorar el rendimiento y afrontar desafíos computacionales complejos, especialmente a nivel de microservicios (Manu, Patel, Akhtar, Agrawal, y Murthy, 2016). Es importante señalar que la tecnología de contenedores no es completamente nueva dentro del desarrollo y la operación de aplicaciones web de gran escala basadas en arquitecturas orientadas a servicios (SOA) (Kecskemeti, Marosi, y Kertész, 2016). Además, facilita la creación de portales web escalables y rentables sobre infraestructuras sólidas y tolerantes

a fallos.

La contenedorización, una forma ligera de virtualización, ha transformado la gestión de aplicaciones en entornos de nube. Esta tecnología permite encapsular las aplicaciones junto con sus dependencias en entornos aislados, asegurando una implementación uniforme en distintas plataformas. A medida que su uso se ha extendido, ha surgido el desafío de gestionar y coordinar eficazmente la creación y el despliegue de contenedores, tanto de manera individual como en clústeres. La adopción de esta tecnología —con Docker como principal referente— ha revolucionado el desarrollo y la implementación de aplicaciones, ofreciendo beneficios como portabilidad, eficiencia en el uso de recursos y escalabilidad rápida (Muzumdar, Basyal, y Vyas, 2020). No obstante, la orquestación de múltiples contenedores en entornos complejos requiere herramientas y soluciones avanzadas que permitan una gestión más precisa y coordinada.

Las plataformas de orquestación de contenedores, como Kubernetes, han adquirido gran relevancia al ofrecer soluciones a este desafío (Mell y Grance, 2011). Estas herramientas proporcionan mecanismos sólidos para automatizar la implementación, el escalado y la administración de aplicaciones basadas en contenedores. En particular, Kubernetes se ha consolidado como el estándar de facto en este ámbito, gracias a sus funcionalidades de descubrimiento de servicios, balanceo de carga y escalado automático según el uso de recursos (Muzumdar, 2022). En definitiva, la contenedorización ha inaugurado una nueva etapa en la forma de desplegar aplicaciones; sin embargo, la gestión eficiente de los con-

tenedores —tanto de manera individual como en clústeres— se ha convertido en una prioridad clave para las operaciones de TI actuales. Esto ha impulsado el surgimiento de plataformas de orquestación avanzadas como Kubernetes, que hoy desempeñan un papel esencial en la administración de aplicaciones en contenedores a gran escala.

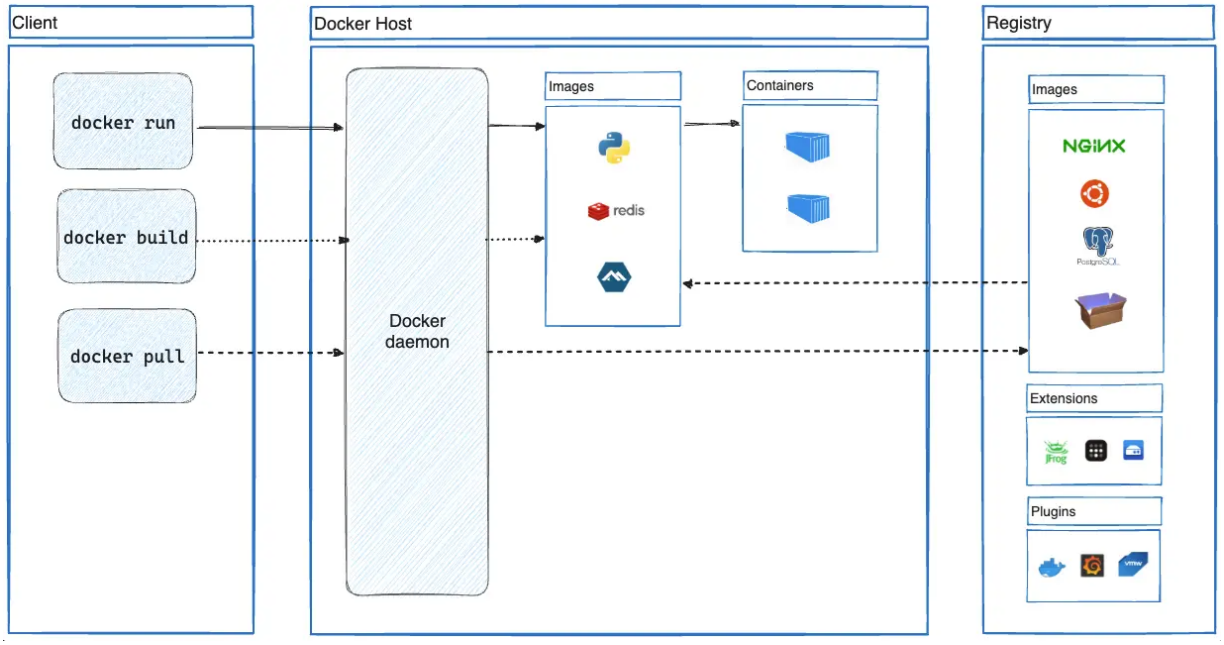
Definición y conceptos básicos. La virtualización de recursos consiste en incorporar una capa adicional de software sobre el sistema operativo anfitrión con el fin de administrar múltiples recursos. Las máquinas virtuales (VM) actúan como entornos de ejecución independientes, y existen diversos métodos para lograr esta virtualización, siendo la basada en hipervisor una de las más utilizadas. Entre las soluciones más reconocidas en esta categoría destacan KVM y VMware. La implementación de esta tecnología requiere un monitor de máquinas virtuales que opere sobre el sistema físico subyacente, mientras que cada VM ofrece soporte a sistemas operativos invitados aislados (Liu, Loo, y Mao, 2011). De este modo, un único sistema operativo host puede albergar varios sistemas operativos invitados dentro de este entorno virtualizado.

En cambio, la virtualización mediante contenedores propone un enfoque distinto. En este modelo, los recursos de hardware se dividen para generar múltiples instancias con un nivel de aislamiento seguro (Kratzke, 2015). La principal diferencia radica en la forma en que los procesos invitados interactúan con la tecnología. En los sistemas basados en contenedores, los procesos acceden directamente a las abstracciones del sistema a través de una capa de virtualización

ubicada en el nivel del sistema operativo (SO). Por el contrario, los métodos basados en hipervisores suelen asignar una máquina virtual a cada sistema operativo invitado. Las soluciones con contenedores, por lo general, comparten un mismo núcleo del sistema operativo entre las instancias virtuales, lo que puede implicar un nivel de seguridad menor frente a los hipervisores. No obstante, desde la perspectiva del usuario, los contenedores se comportan como sistemas operativos independientes, aparentemente capaces de operar de forma autónoma respecto al hardware y al software subyacente.

Figura 2

Arquitectura contenerizada



Nota. La figura muestra como funciona la Arquitectura contenerizada. Tomada de DockerInc (2025).

Plataformas más comunes. De igual manera que las plataformas más comunes utilizadas para serverless, existe su contraparte para contenedores: Amazon ECS/EKS, Azure Kubernetes Service (AKS) y Google Kubernetes Engine (GKE). Además de herramientas como Docker, creación y ejecución de contenedores (Muzumdar, Bhosale, Basyal, y Kurian, 2024) y Kubernetes, para la orquestación de múltiples contenedores.

Casos de uso. El estudio de Pérez, Moltó, Caballer, y Calatrava (2018) proporciona una descripción general de soluciones comerciales, analizando la orquestación de contenedores por medio de Kubernetes.

Este otro estudio de Carlos Guerrero (2018) presenta un enfoque para optimizar la implementación de aplicaciones basadas en microservicios utilizando contenedores en arquitecturas multinube. Los objetivos de optimización incluyen el coste del servicio en la nube, la latencia de la red entre microservicios y el tiempo necesario para iniciar un nuevo microservicio cuando un proveedor no está disponible.

Estado del Arte

Menéndez y cols. (2023) han abordado la comparación entre arquitecturas tradicionales (basadas en Kubernetes) y Serverless (FaaS) en un entorno de microservicios, con un enfoque en:

- Migración: Análisis del proceso de migrar una aplicación de comercio electrónico desde Kubernetes a tecnologías Serverless (AWS Lambda, ECS Fargate) y semiserverless (DocumentDB, DynamoDB).

- Métricas clave: Evaluación de costos (mensuales) y rendimiento (latencia, escalabilidad) de 9 prototipos diferentes.
- Enfoque práctico: Utilización AWS como proveedor de nube y pruebas reales con herramientas como JMeter y AWS CloudWatch.

El tratamiento es técnico y empírico, con datos cuantitativos (tablas de costos y latencias) y cualitativos (discusión de desafíos como el Cold Start en Lambda).

En Lekkala (2023) también se han comparado arquitecturas contenerizadas (Docker/Kubernetes) y Serverless (AWS Lambda, Azure Functions) para pipelines de datos, centrándose en:

1. Enfoque técnico: Análisis de ventajas y desventajas de cada enfoque en términos de:
    - Rendimiento: Latencia, escalabilidad y eficiencia.
    - Costos: Pago por uso (Serverless) vs. costos fijos (contenedores).
    - Mantenibilidad: Complejidad operativa y gestión de dependencias.
2. Escenarios prácticos: Ejemplos como procesamiento por lotes (mejor con contenedores) y streaming en tiempo real (óptimo para Serverless).
3. Metodología: Revisión teórica con soporte en literatura académica y casos de uso, sin experimentación propia.

El tratamiento es comparativo y descriptivo, con énfasis en aplicaciones para big data y computación en la nube.

Por otra parte, Vaño y cols. (2023) realiza una orquestación de cargas de trabajo nativas en Nube en el Borde (Edge Computing) donde:

1. Lleva a cabo un análisis del contexto.
2. Revisa la virtualización de servicios en la nube.
3. Proporciona una descripción general de soluciones comerciales existentes a la fecha.
4. Enumera las opciones de implementación del tipo más común de cargas de trabajo.
5. Analiza la orquestación por medio de Kubernetes.
6. Describe las tendencias más comunes que actualmente se están dando

Balalaie, Heydarnoori, y Jamshidi (2016) informaron sobre las lecciones aprendidas de la migración de un back-end móvil comercial a una arquitectura de microservicios implementada con contenedores.

Hipótesis de la investigación

Las arquitecturas basadas en contenedores proporcionan mayor flexibilidad, portabilidad y capacidad de configuración que las arquitecturas serverless, por lo que se espera que ofrezcan un mejor desempeño y costos más predecibles en escenarios de carga sostenida de peticiones.

Metodología

Para el diseño experimental de este estudio, utilizaremos las plataformas de computación en nube Amazon AWS, Microsoft Azure y Google Cloud, todas gozan de una sólida reputación entre los administradores de arquitecturas en la nube y cuentan con senda documentación para su uso e implementación.

App objeto de la prueba

Para este estudio, implementaremos una pequeña app de prueba escrita en Python, pueden encontrar el código de la misma en los anexos, se tomó el código y se desplegó en las tres plataformas descritas en el párrafo anterior.

Sobre despliegue de contenedores

Los tecnología de contenedor requiere de un componente esencial para su despliegue, dado que requiere de una imagen de servidor publicada previamente para su despliegue; lo anterior nos obligo (solo para el contenedor) el uso de tecnologías para el registro de las imágenes de contenedor

Sobre despliegue de Serverless

Las funciones son la manera en que se utiliza la arquitectura serverless, básicamente son código directamente inyectado dentro del servidor, sin tener que preocuparme por aprovisionar recursos para ello.

En resumen estas fueran las tecnologías empleadas:

Empezaremos con la tecnología AWS ECS para la aplicación contenerizada y con la aplicación migrada a la arquitectura serverless, utilizaremos AWS Lambda. Continuaremos con la tecnología Azure equivalente, para aplicación

Tabla 1

Tecnologías y servicios equivalentes por proveedor de nube

Arquitectura	Amazon AWS	Microsoft Azure	Google Cloud Platform
Imagen del Contenedor	ECR (Elastic Container Registry)	Container Registries	Google Container Registry
Contenedores	ECS (Elastic Container Service)	Container Instances	Google Container Instances
Serverless	Lambda (Functions)	App Functions	Google Functions

Nota. Comparación de los nombres comerciales de los servicios utilizados en la investigación para cada proveedor.

contenerizada y con la aplicación migrada a la arquitectura serverless, utilizaremos Azure equivalente. Finalizaremos con la tecnología G Cloud equivalente, para aplicación contenerizada y con la aplicación migrada a la arquitectura serverless, utilizaremos G Cloud equivalente.

Pruebas de estrés

Para evaluar la escalabilidad, se realizan pruebas de estrés con ayuda de la herramienta Locust, el cual esta desarrollado en Python y permite desde un equipo cliente simular multiples peticiones a los diferentes servidores (Containers y Serverless); La pruebas consistieron en simular 25 y 100 usuarios simultáneos sobre las diferentes tecnologías; con el tiempo predefinido por la herramienta (4 minutos y 28 segundos); los tiempos de respuesta fueron medidos en milisegun-

dos (ms).

El análisis del coste financiero de las arquitecturas basadas en contenedores y serverless se realizará asumiendo la provisión de la infraestructura necesaria para ejecutar las pruebas proyectadas durante un periodo de un mes.

Las métricas de rendimiento se obtendrán de las herramientas de monitorización que centraliza todas las estadísticas de rendimiento (por ejemplo, CloudWatch en el caso de AWS). Es importante señalar que para las funciones serverless de prueba (por ejemplo, Lambda en el caso de AWS), las métricas de uso de CPU y RAM no estarán disponibles; dichas métricas se proporcionan solo para los contenedores (por ejemplo, ECS en el caso de AWS).

Resultados

Los siguientes test fueron ejecutados para evaluar cuantitativamente los contenedores y las plataformas serverless, con resultados presentados en gráficas y tablas comparativas. ver 2

Tabla 2

Criterios comparativos entre arquitecturas

Criterio	Serverless	Contenedores
Escalabilidad	Automática y granular	Más configuraciones activas
Costo	Aplicaciones con cargas variables	Implica una mayor estabilidad pero potenciales costos
Desempeño	Variaciones en desempeño	Control previo de parámetros

Nota. Resumen de las diferencias clave consideradas para la selección de la arquitectura.

Resultados para Amazon AWS

Escalabilidad

Para evaluar la escalabilidad, se ejecutaron pruebas de estrés, se usó la herramienta para pruebas llamada Locust. El test consistió en simular 100 usuarios concurrentes durante un periodo de tiempo de 4 minutos con 28 segundos, se empleó el mismo tiempo para simular 25 usuarios, se muestran los tiempos de respuesta, se midieron en milisegundos. Los resultados se presentan en la tabla 3, 4 y tabla 5.

Tabla 3

Resultados de la prueba de rendimiento para AWS Lambda Functions

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
54738	48224	77.76	62	699	80.74	800.18	704.95

Nota. La prueba se ejecutó con 100 usuarios concurrentes. URL del punto final: `https://avruhjuyl6gcudftupm2vcoahu0dzcgr.lambda-url.us-east-1.on.aws`. Duración: 11/5/2025, 4:10:19 p. m. – 4:11:27 p. m. (1 minuto y 8 segundos). Abreviaturas: #Req = Peticiones totales; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo.

Tabla 4

Resultados de rendimiento para AWS Lambda Functions con 25 usuarios concurrentes

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
87419	61864	74.26	62	717	68.79	326.19	230.84

Nota. La prueba se ejecutó en la URL `https://avruhjuyl6gcudftupm2vcoahu0dzcgr.lambda-url.us-east-1.on.aws`. Duración: 11/5/2025, 4:10:19 p. m. – 4:11:27 p. m. (1 minuto y 8 segundos). Abreviaturas: #Req = Peticiones totales; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo.

Tabla 5

Resultados de la prueba de escalabilidad en contenedores

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
201468	0	114.06	58	704	44	751.69	0

Nota. La prueba se realizó con 100 usuarios concurrentes en la URL `http://54.221.143.244:8000`. La duración fue del 11/5/2025, de 3:59:32 p.m. a 4:04:00 p.m. (4 minutos y 28 segundos). Abreviaturas: #Req = Número de peticiones; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo.

Tabla 6

Resultados de la prueba de escalabilidad en contenedores con 25 usuarios concurrentes

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
89994	0	71.82	58	427	44	336.93	0

Nota. La prueba se realizó con 25 usuarios concurrentes en la URL `http://54.221.143.244:8000`. La duración fue del 11/5/2025, de 3:59:32 p.m. a 4:04:00 p.m. (4 minutos y 28 segundos). Abreviaturas: #Req = Número de peticiones; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo.

Análisis de Costos

Se aprovisionó los recursos necesarios para la arquitectura de contenedores y la arquitectura serverless durante el plazo de un mes, los resultados se presentan en las tablas 8 y **??**.

Tabla 7

Estimación de costo por transacciones en Lambda

Descripción	Costos (USD)
Costos proyectados a un mes	$109

Nota. Estimación basada en el volumen de transacciones de la prueba y el esquema de precios vigente de AWS.

Tabla 8

Estimación de costo por transacciones en Contenedores

Descripción	Costos (USD)
Costos proyectados a un mes	$36.04

Nota. Estimación basada en el volumen de transacciones de la prueba y el esquema de precios vigente de AWS.

Desempeño

Para AWS las métricas de soporte se obtuvieron de AWS cloudWatch, la herramienta de monitoreo centraliza el registro de todas las estadísticas. Para las Lambda Function las metricas de uso de CPU y memoria RAM no se encuentran disponibles, por tanto la tabla 9 solo incluye las cifras del contenedor.

Los tiempos de respuesta se midieron en milisegundos, los resultados se presentan en tabla 10, Fig. 3, Fig. 12 y Fig. 13.

Tabla 9

Comparativa de uso de recursos de hardware por tipo de arquitectura

Tipo de Arquitectura	Usuarios	CPU	RAM
Lambda Functions	25	No disponible	No disponible
Lambda Functions	100	No disponible	No disponible
Contenedores	25	20 %	1.16 GB
Contenedores	100	48 %	1.24 GB

Nota. Los valores de consumo de infraestructura (CPU/RAM) en AWS Lambda son abstraídos por el proveedor y no están disponibles para medición directa. Abreviaturas: GB = Gigabytes.

Tabla 10

Percentiles de tiempos de respuesta por tipo de arquitectura

Tipo	Usuarios	50 %	60 %	70 %	80 %	90 %	95 %	99 %	100 %
L	25	68	70	72	74	81	100	200	720
L	100	70	72	75	78	84	100	230	700
C	25	64	65	67	70	82	110	200	430
C	100	96	100	110	120	170	230	390	700

Nota. Valores expresados en milisegundos (ms). Abreviaturas: L = Lambda Function; C = Contenedores ECS. Para la representación gráfica de estos datos, véase la Figura 3.

Figura 3

Tiempo de respuesta y distribución

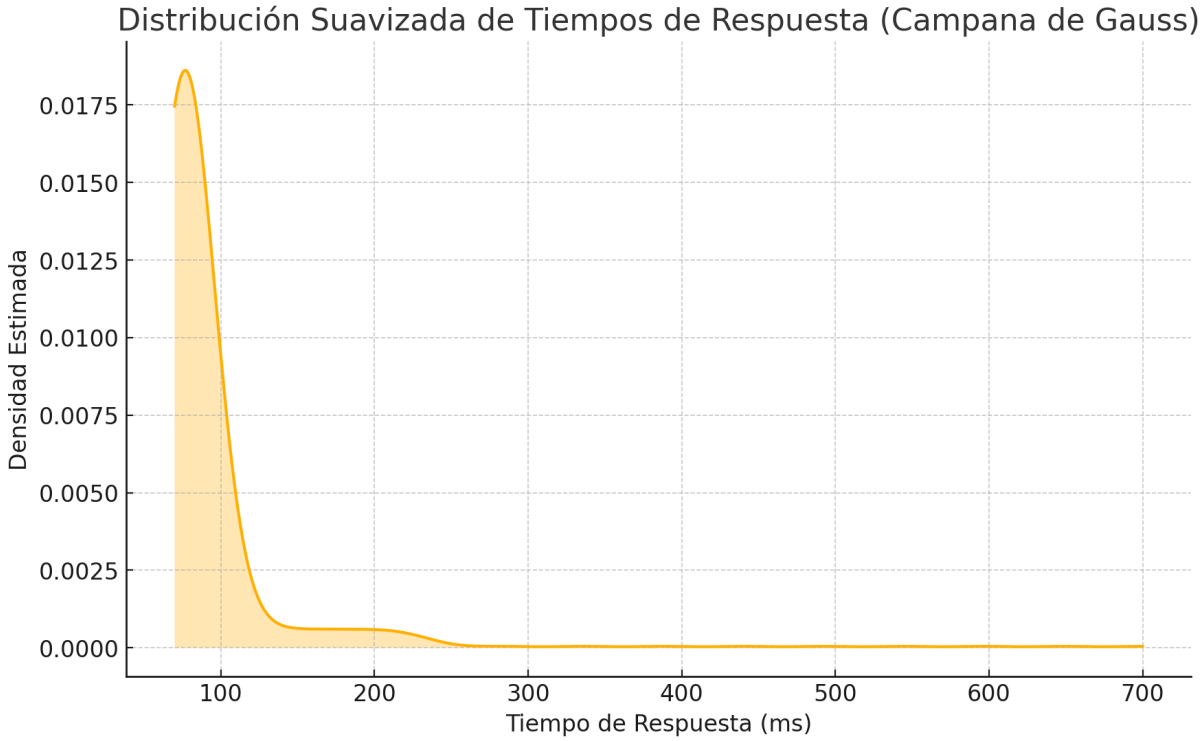


Figura 4

Pruebas a contenedores

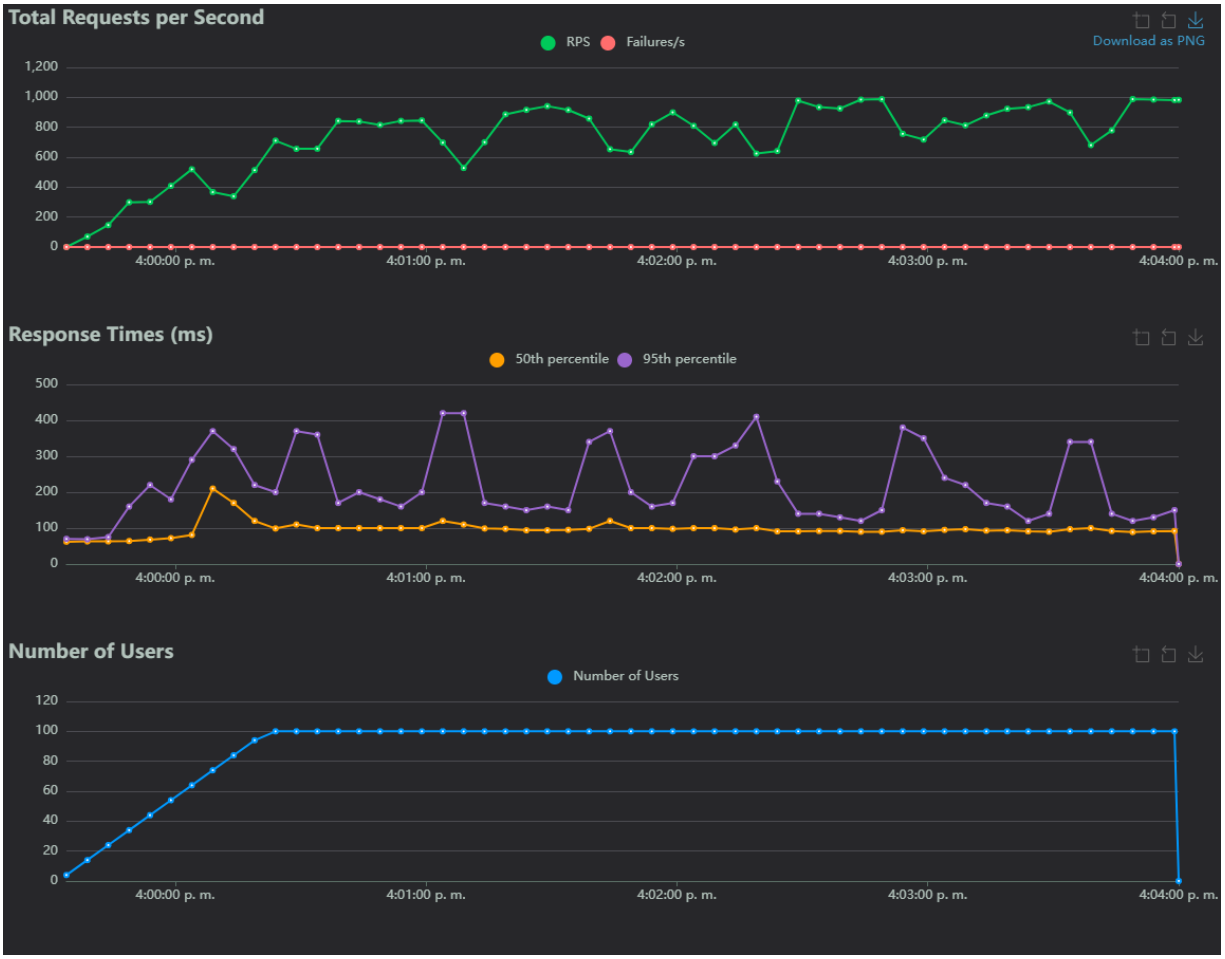
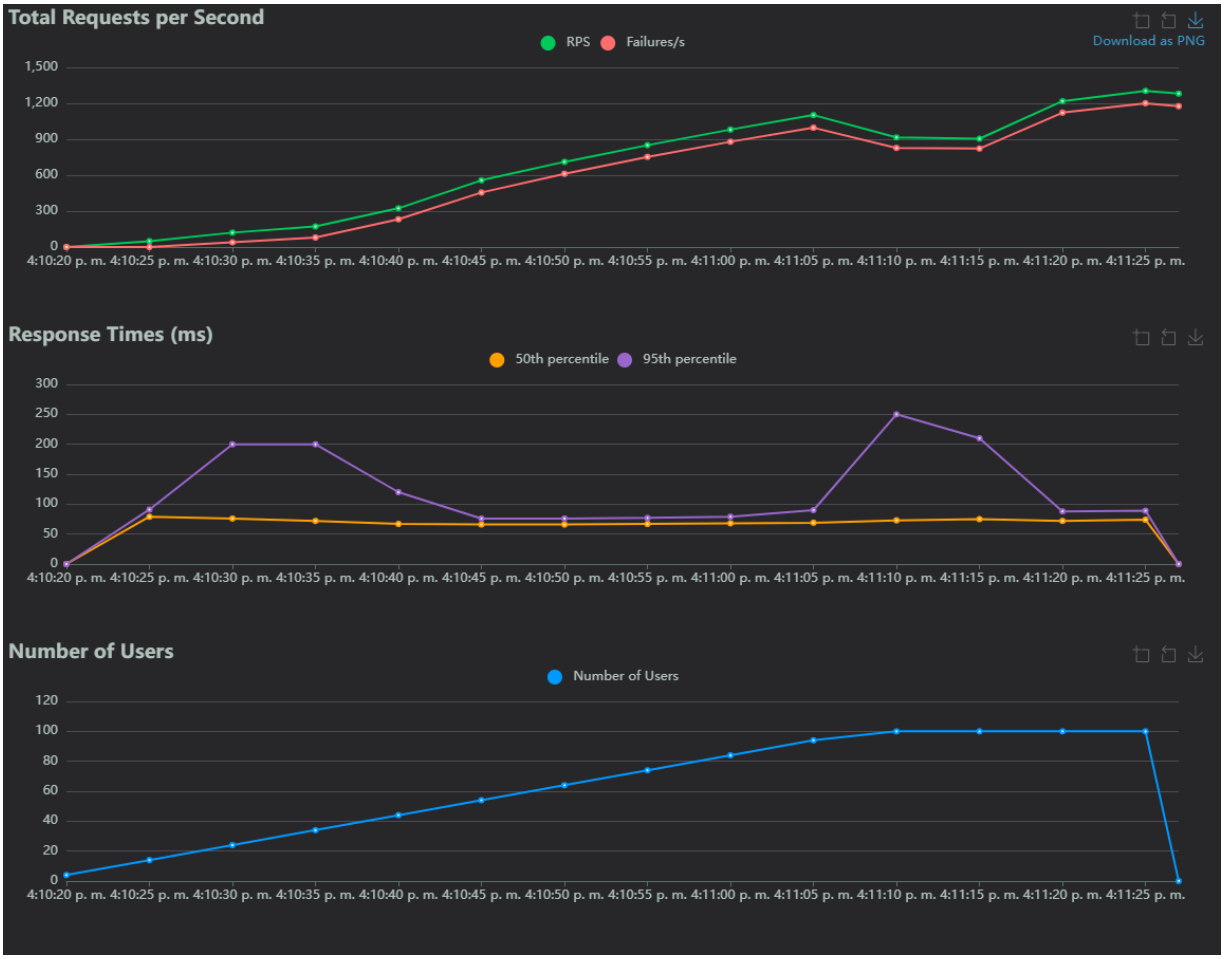


Figura 5

Pruebas a Lambda Function



Resultados para Google Cloud Platform

Escalabilidad

Los mismos test fueron realizados en la plataforma de Google Cloud obteniendo los resultados para la métrica de escalabilidad, modalidad función, en las tablas 11 y 12; modalidad contenedor, en las tablas 13 y 14.

Análisis de Costos

Los análisis de costos están indicados en las tablas 15 y 16.

Desempeño

El desempeño de las pruebas de 25 y 100 usuarios para ambas tecnologías se comparan en la tabla 17 y los resultados son mostrados en la tabla 18 y desde la Fig. 6 a 10.

Tabla 11

Resultados de la prueba de rendimiento para Cloud Run (Function) con 25 usuarios concurrentes

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
1201	0	92.5	160	457	n/a	18.8	0

Nota. La prueba se realizó con 25 usuarios concurrentes en la URL `https://function-1071301274116.europe-west1.run.app`. La duración fue del 24/9/2025, de 9:13:34 p.m. a 9:18:02 p.m. (4 minutos y 28 segundos). Abreviaturas: #Req = Número de peticiones; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo; n/a = no disponible.

Tabla 12

Resultados de la prueba de rendimiento para Cloud Run (Function) con 100 usuarios concurrentes

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
23214	0	91.4	120.4	406.4	n/a	76.6	0

Nota. La prueba se realizó en la URL `https://function-1071301274116.europe-west1.run.app`. Duración: 24/9/2025, 9:40:12 p. m. – 9:44:12 p. m. (4 minutos y 28 segundos). Abreviaturas: #Req = Número de peticiones; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo; n/a = no disponible.

Tabla 13

Resultados de la prueba de rendimiento para Cloud Run (Container) con 25 usuarios concurrentes

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
1119	0	91.9	165.5	204.9	n/a	18.6	0

Nota. La prueba se realizó con 25 usuarios concurrentes en la URL `http://54.221.143.244:8000`. Duración: 26/9/2025, de 10:26:06 p. m. a 10:30:34 p. m. (4 minutos y 28 segundos). Abreviaturas: #Req = Número de peticiones; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo; n/a = no disponible.

Tabla 14

Resultados de la prueba de rendimiento para Cloud Run (Container) con 100 usuarios concurrentes

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
23439	0	90.5	123.7	319.7	n/a	76.8	0

Nota. La prueba se realizó con 100 usuarios concurrentes en la URL `http://54.221.143.244:8000`. La duración fue del 26/9/2025, de 10:42:21 p.m. a 10:46:49 p.m. (4 minutos y 28 segundos). Abreviaturas: #Req = Número de peticiones; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo; n/a = no disponible.

Tabla 15

Estimación de costo por transacciones en Cloud Run (Function)

Descripción	Costos (USD)
Costos proyectados a un mes	$89

Nota. Estimación basada en el volumen de transacciones de la prueba y el esquema de precios vigente de Google Cloud Platform (GCP).

Tabla 16

Estimación de costo por transacciones en Cloud Run (Container)

Descripción	Costos (USD)
Costos proyectados a un mes	$45

Nota. Estimación basada en el volumen de transacciones de la prueba y el esquema de precios vigente de Google Cloud Platform (GCP).

Tabla 17

Comparativa de uso de recursos en Google Cloud Platform (GCP)

Tipo de Arquitectura	Usuarios	CPU	RAM
Cloud Run (Function)	25	No disponible	No disponible
Cloud Run (Function)	100	No disponible	No disponible
Cloud Run (Container)	25	1 %	0.03 GB
Cloud Run (Container)	100	12 %	0.04 GB

Nota. Los valores de consumo de infraestructura en el modo totalmente gestionado (Function) son abstraídos por el proveedor. Abreviaturas: GB = Gigabytes.

Tabla 18

Percentiles de tiempos de respuesta por tipo de arquitectura en GCP

Tipo	Usuarios	50 %	60 %	70 %	80 %	90 %	95 %	99 %	100 %
F	25	90	95	96	97	97	98	98	99
F	100	90	90	94	95	95	95	95	96
C	25	92	94	94	95	98	99	99	101
C	100	91	93	93	94	94	94	94	95

Nota. Valores expresados en milisegundos (ms). Abreviaturas: F = Cloud Run (Function); C = Cloud Run (Container). Para la representación gráfica de estos datos, véase la Figura 6.

Figura 6

Tiempo de respuesta y distribución

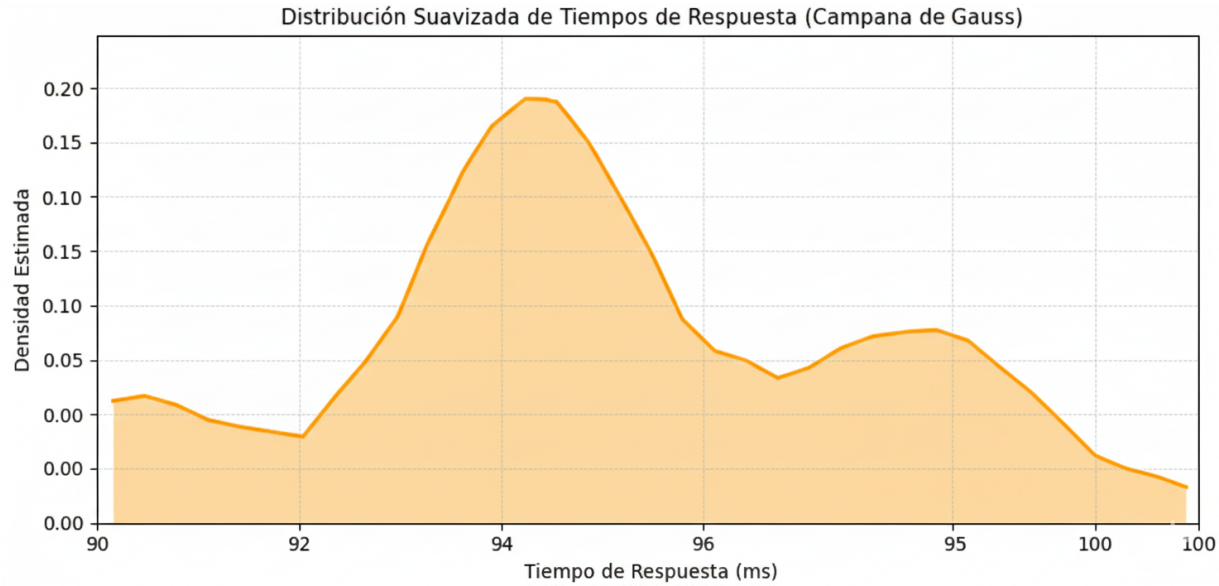


Figura 7

Pruebas a Cloud Run/function - 25 usuarios

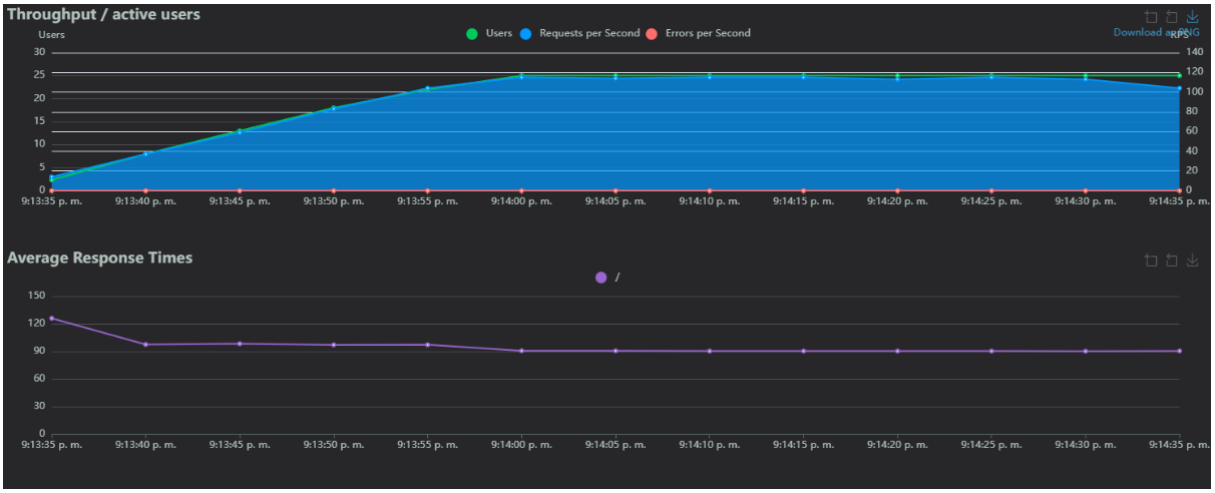


Figura 8

Pruebas a Cloud Run/function - 100 usuarios

Figura 9

Pruebas a Cloud Run/container - 25 usuarios

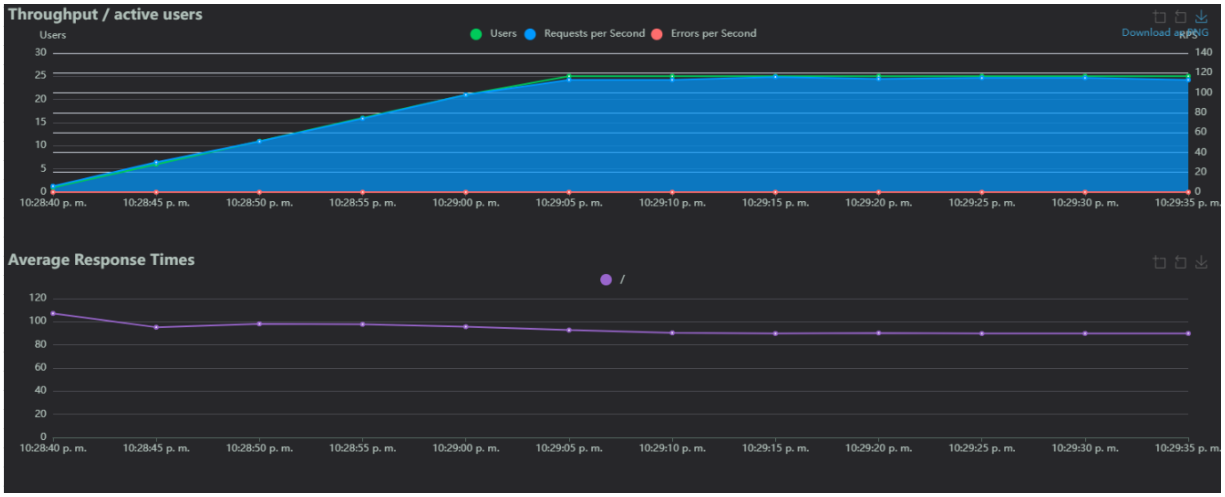
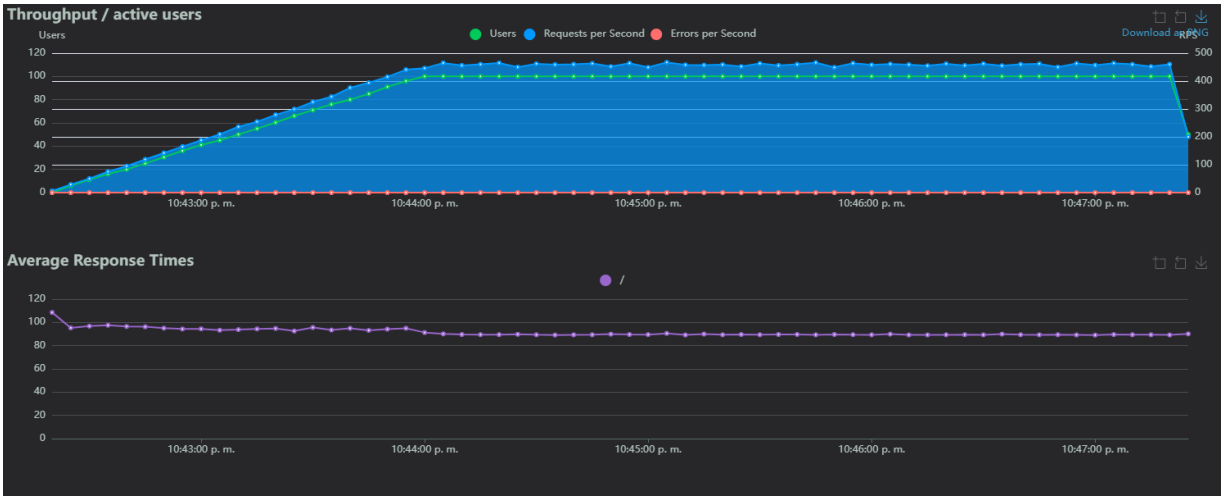


Figura 10

Pruebas a Cloud Run/container - 100 usuarios



Resultados para Microsoft Azure

Los mismos test fueron realizados en la plataforma Microsoft Azure empleando las tres métricas enunciadas, a continuación estos son los resultados.

Escalabilidad

El desempeño de las pruebas de 25 y 100 usuarios se comparan en la tabla 19, 20,22 y 21.

Tabla 19

Resultados de rendimiento para Azure Function App con 100 usuarios concurrentes

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
198304	0	108.48	54	524	69	739.23	0

Nota. La prueba se realizó con 100 usuarios concurrentes. URL del punto final: `https://apppython-e0h0gta7h3cyctcs.eastus2-01.azurewebsites.net/api/http_trigger1?`. Duración: 18/11/2025, 10:58:03 p. m. – 11:02:31 p. m. (4 minutos y 28 segundos). Abreviaturas: #Req = Número de peticiones; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo.

Tabla 20

Resultados de rendimiento para Azure Function App con 25 usuarios concurrentes

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
95052	0	67.09	53	2689	69	354.67	0

Nota. La prueba se realizó con 25 usuarios concurrentes. URL del punto final: `https://apppython-e0h0gta7h3cyctcs.eastus2-01.azurewebsites.net/api/http_trigger1?`. Duración: 18/11/2025, 10:58:03 p. m. – 11:02:31 p. m. (4 minutos y 28 segundos). Abreviaturas: #Req = Número de peticiones; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo.

Tabla 21

Resultados de rendimiento para Azure Container Instances con 100 usuarios concurrentes

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
307237	0	68.13	50	384	44	1146.39	0

Nota. La prueba se realizó con 100 usuarios concurrentes en la URL `http://20.161.225.37:8567/`. Duración: 18/11/2025, 11:28:56 p. m. – 11:33:24 p. m. (4 minutos y 28 segundos). Abreviaturas: #Req = Número de peticiones; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo.

Tabla 22

Resultados de rendimiento para Azure Container Instances con 25 usuarios concurrentes

#Req	#Fallos	Prom	Min	Max	PromSz	RPS	Fallos/s
114706	0	55.56	50	339	44	428.01	0

Nota. La prueba se realizó con 25 usuarios concurrentes en la URL `http://20.161.225.37:8567/`. Duración: 18/11/2025, 11:28:56 p. m. – 11:33:24 p. m. (4 minutos y 28 segundos). Abreviaturas: #Req = Número de peticiones; Prom = Promedio (ms); PromSz = Tamaño promedio (bytes); RPS = Peticiones por segundo.

Análisis de Costos

Se aprovisionó los recursos necesarios para la arquitectura de contenedores y la arquitectura serverless durante el plazo de un mes, los resultados se presentan en las tablas 23 y 24.

Tabla 23

Estimación de costo por transacciones en Azure App Function

Descripción	Costos (USD)
Costos proyectados a un mes	$54.75

Nota. Estimación basada en el volumen de transacciones de la prueba y el esquema de precios vigente de Microsoft Azure.

Tabla 24

Estimación de costo por transacciones en Azure Container Instances

Descripción	Costos (USD)
Costos proyectados a un mes	$19.50

Nota. Estimación basada en el volumen de transacciones de la prueba y el esquema de precios vigente de Microsoft Azure.

Desempeño

Para Microsoft Azure las métricas de soporte se obtuvieron de los registros en Azure, esta herramienta está asociada al contenedor. Al igual que con las App Functions las métricas de uso de CPU y memoria RAM no se encuentran disponibles, por tanto la tabla 9 solo incluye las cifras del contenedor.

Tabla 25

Comparativa de uso de recursos en Microsoft Azure

Tipo de Arquitectura	Usuarios	CPU	RAM
App Functions	25	No disponible	No disponible
App Functions	100	No disponible	No disponible
Contenedores	25	23 %	40 MB
Contenedores	100	65 %	50 MB

Nota. Los valores de consumo de infraestructura en App Functions son abstraídos por el proveedor. Abreviaturas: MB = Megabytes; CPU = Unidad Central de Procesamiento.

Tabla 26

Percentiles de tiempos de respuesta por tipo de arquitectura en Microsoft Azure

Tipo	Usuarios	50 %	60 %	70 %	80 %	90 %	95 %	99 %	100 %
A	25	63	65	68	72	79	88	110	2700
A	100	110	110	120	130	150	160	210	520
C	25	55	56	56	58	60	61	71	340
C	100	67	69	71	75	80	86	98	380

Nota. Valores expresados en milisegundos (ms). Abreviaturas: A = App Function; C = Container Instances. Para la representación gráfica de estos datos, véase la Figura 11.

Figura 11

Tiempo de respuesta y distribución

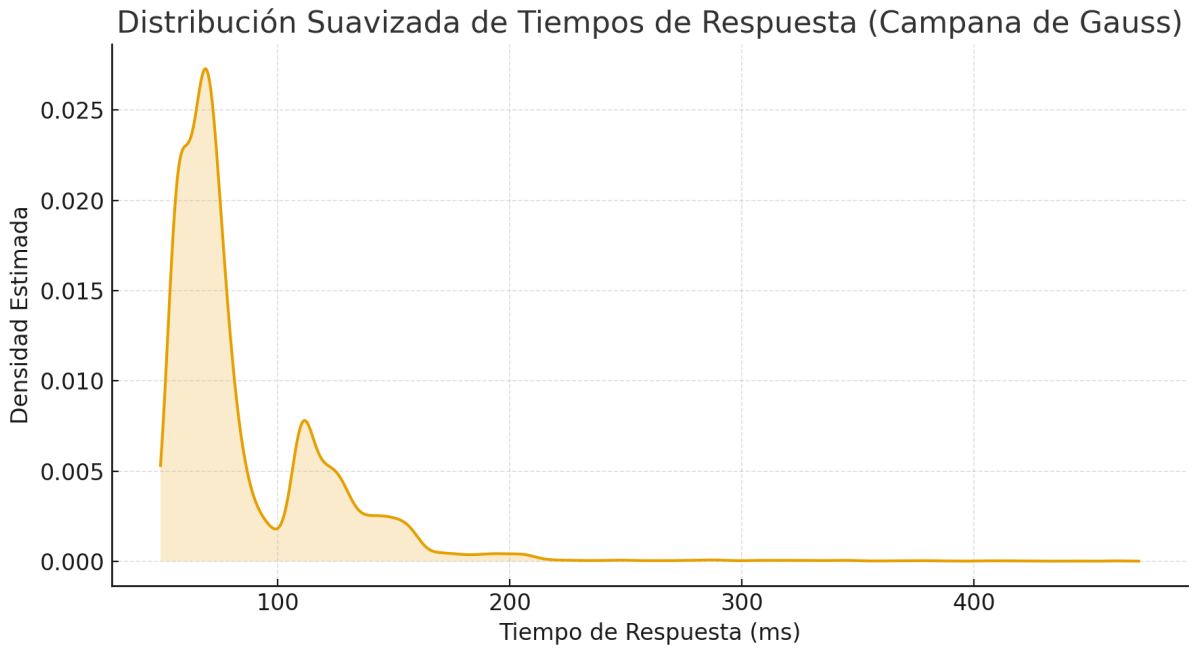


Figura 12

Pruebas en Container Instances

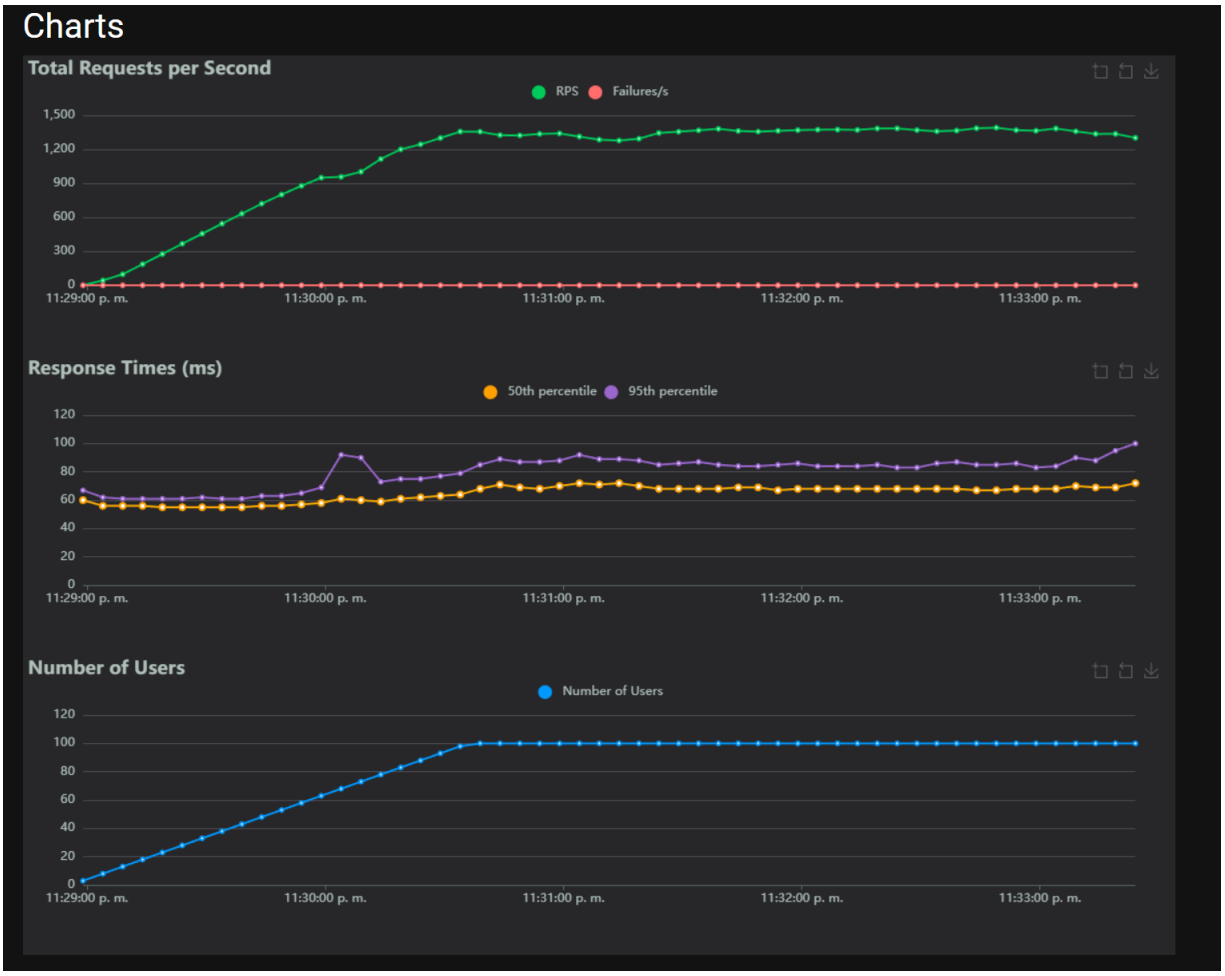
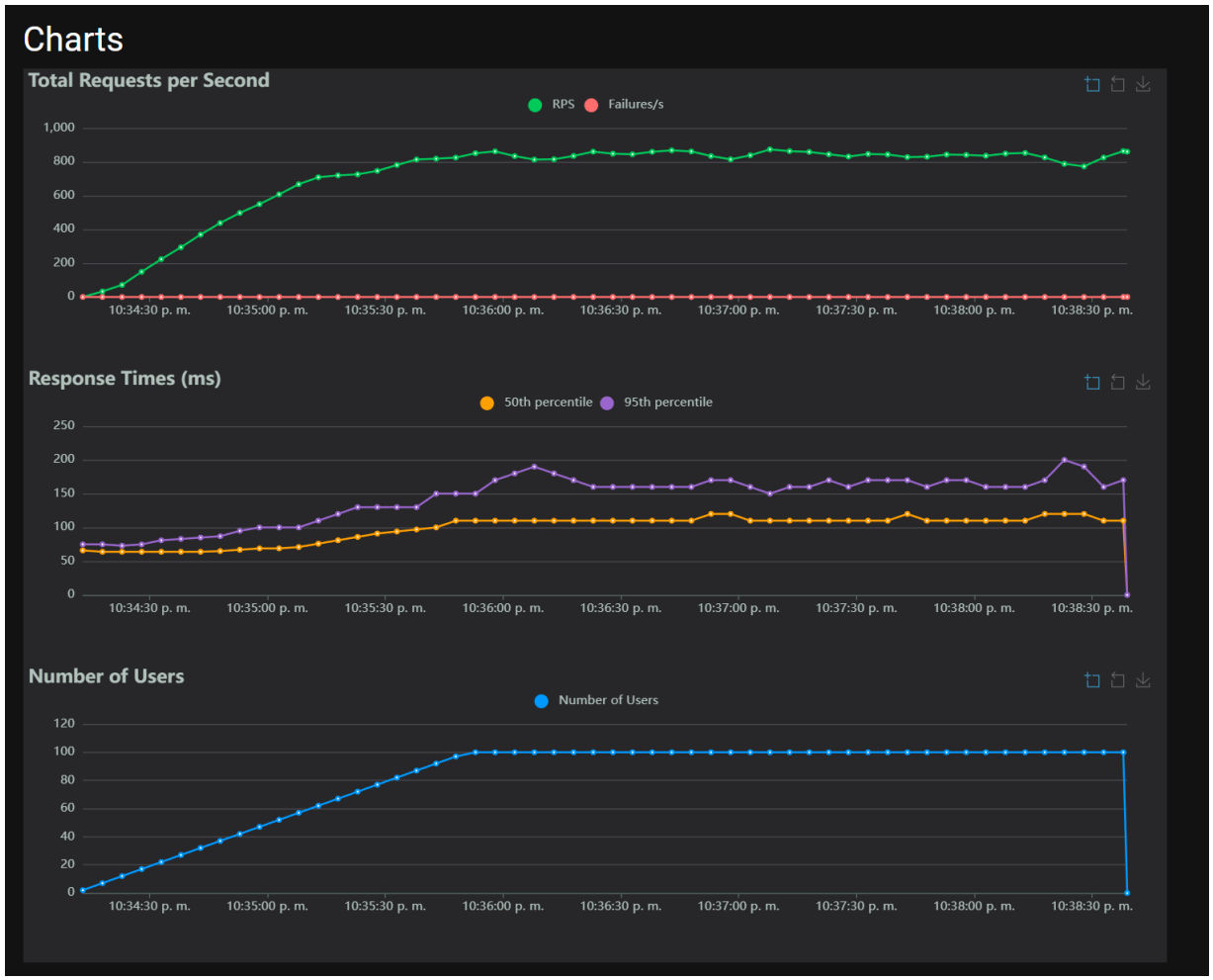


Figura 13

Pruebas en App Function Microsoft



Discusión

Tal como se anticipaba en la hipótesis los contenedores arrojan una mejor opción de acuerdo a las métricas definidas en la metodología de investigación (Escalabilidad, Coste y desempeño).

Se promediaron los resultados para obtener una visión general a la luz de las tres métricas propuestas en la metodología de investigación, los resultados los mostramos en las tablas 27, 28, 29

Escalabilidad

Los valores obtenidos en la métrica de escalabilidad se obtienen cuando los servidores están obteniendo un incremento de usuarios, cada uno de ellos generando múltiples peticiones por segundo, a mayor número en la medición de esta métrica mejor es el resultado; se observa que en las tecnologías serverles es menor debido al fenómeno del cold start.

Cold Start

Este fenómeno provoca que por cada petición a la url, una instancia de servidor tenga que ser levantada solo para procesar dicha petición, lo cual consume reduce en aproximadamente un 22 % el número total de peticiones que pueden procesar los servidores.

Costo

En cuanto al coste económico en resultado de las pruebas arroja un claro ganador; la tecnología de contenedores es mas asequible, lo cual se traduce en una reducción aproximada del 40 % en los costes si se decanta por la tecnología

Tabla 27

Promedios de RPS (Request Per Second)

Tecnología	AWS	Azure	GCP	Promedio
Serverless	800.18	739.23	76.6	538.67
Container	751.69	1145.39	76.80	658.29

del contenedor.

Hemos contrastado nuestros resultados con los presentados por Menéndez y cols. (2023) quienes determinaron que la arquitectura Serverless (Lambda + DynamoDB) resultaba ser la opción más económica con un costo mensual de €19.23, frente a los €55.8 de ECS Fargate.

Esta discrepancia se explica por el modelo de carga: mientras Menéndez asumió una carga constante moderada de 5 usuarios/segundo (13 millones de peticiones/mes), nuestro estudio sometió a las arquitecturas a pruebas de estrés con alta concurrencia (100 usuarios simultáneos).

Esto confirma que Serverless es eficiente en costos para cargas bajas o esporádicas, pero los Contenedores (ECS) son superiores financieramente bajo cargas de trabajo sostenidas e intensas, donde el modelo de pago por uso de Lambda se vuelve desventajoso

Desempeño

Para finalizar en las pruebas de desempeño tomamos el resultado del percentíl 99, esto en razón que representa la mayor cantidad de solicitudes a x ms, la forma de interpretar este resultado es entender que si el promedio de la tecnología

Tabla 28

Valores por costo en dólares americanos

Tecnología	AWS	Azure	GCP	Promedio
Serverless	109	19.50	89	72.50
Container	36.04	54.75	45	45.26

serverless es 176.67 esto significa que el 99 % de las peticiones se resolvieron en un valor igual o menor a 176.67 ms.

Si observamos el resultado promedio de todas las plataformas se observa un menor valor para las tecnologías serverless lo cual indica una leve mejoría ante los 194 ms de los contenedores pero que a nuestro juicio dado los resultados en las métricas de costes y escalabilidad no es significante para poder decir que serverless es una mejor opción.

Los Contenedores ofrecen enfoques distintos para la arquitectura serverless; esta destaca en casos de uso ligeros y basados en eventos, mientras que la flexibilidad y el modelo de concurrencia de contenedores lo hacen más adecuado para cargas de trabajo que requieren desempeño y escalabilidad a un menor coste.

La variable Regional

Otros estudios centraron su enfoque en una sola plataforma, los resultados producidos en esta investigación son el promedio de Microsoft Azure, Google Cloud y Amazon AWS los tres referentes de la computación en la nube; nuestro enfoque proporciona un visión holística que permite evaluar las arquitecturas

Tabla 29

Desempeño en ms (milisegundos)

Tecnología	AWS	Azure	GCP	Promedio
Serverless	230	210	90	176.67
Container	390	98	94	194

serverless y container en diferentes escenarios.

Si bien el enfoque integral de los resultados en esta investigación, es necesario ir un paso mas allá y generar resultados para distintas regiones; las regiones dentro de las plataformas IAAS se identifican como la selección regional del datacenter en que correrán los containers y functions.

En próximos estudios es recomendable realizar los mismo experimentos con pruebas de cargar que impliquen diferentes regiones en los datacenters seleccionados. Garantizaría un resultado que abarque el criterio geográfico como una variable mas para este tipo de estudios

Conclusiones

Según las mediciones obtenidas de las tres plataformas objeto del estudio (AZR,GCP,AWS), las tecnologías basadas en contenedores demostraron una clara ventaja, recordamos al lector que nuestra investigación se concentró en la respuesta a una carga incremental de usuarios simultáneos. En este sentido, los contenedores ofrecen una ventaja innegable, para las pruebas en contenedores se procesó con éxito el 100 % de las solicitudes, mientras que la misma prueba en serverless dio como resultado una tasa de fallo del 88 %.

Las figuras 3, 6 y 11 muestra que los tiempos de respuesta de las tres plataformas (AZR,GCP,AWS) son similares, ya que la mayoría de las solicitudes se completan en menos de 50 ms, un tiempo de respuesta aceptable. Sin embargo, las tecnologías basadas en contenedores muestran un ligero aumento en el número de solicitudes que superan los 90 ms cuando se gestionan 100 usuarios simultáneos.

Las tecnologías basadas en contenedores son tres veces más rentables que las funciones cuando se gestiona el mismo número de usuarios simultáneos. Esto hace que Serverless sea una solución rentable solo para tareas muy específicas y poco frecuentes, pero inadecuada para gestionar múltiples solicitudes simultáneas.

En términos de rendimiento bajo múltiples solicitudes, las tecnologías basada en contenedores (ECS, Cloud Run/container y Azure containers) demostraron ser significativamente más eficiente, gestionando el 100 % de las solicitudes

y utilizando solo el 50 % de los recursos de CPU y RAM disponibles. Nuestro estudio se limitó a realizar pruebas en servicios proporcionados por Amazon AWS, Google GCP y Microsoft Azure.

Recomendaciones

A partir del análisis comparativo realizado entre las arquitecturas *serverless* y basadas en contenedores, se formulan las siguientes recomendaciones, orientadas a guiar la selección y aplicación de cada paradigma en función de las características y requerimientos específicos de los sistemas a desarrollar.

En primer término, se recomienda la adopción de arquitecturas *serverless* en escenarios donde las cargas de trabajo presenten una naturaleza altamente variable y donde la prioridad sea reducir la gestión de la infraestructura subyacente. Este enfoque resulta particularmente apropiado para aplicaciones orientadas a eventos, procesos de corta duración o servicios que demandan una escalabilidad automática y una rápida respuesta ante fluctuaciones de la demanda. No obstante, es fundamental considerar las limitaciones inherentes a este modelo, como los tiempos de arranque en frío, las restricciones en la duración de las ejecuciones y la dependencia de un proveedor específico (*vendor lock-in*). Por ello, se sugiere realizar una evaluación exhaustiva de los requerimientos funcionales y no funcionales antes de proceder con su implementación.

En contraposición, se recomienda el uso de arquitecturas basadas en contenedores cuando se requiera un mayor control sobre el entorno de ejecución, la configuración del sistema y la persistencia de los servicios. Los contenedores ofrecen una mayor portabilidad entre diferentes plataformas y proveedores, así como un nivel superior de personalización.

Los contenedores soportan mucho mejor la demanda de peticiones concu-

rrentes, hasta el limite razonable de los recursos aprovisionados, si bien dentro de la escala de costos considerados en este estudio (precio/rendimiento), si no se posee la capacidad de prever la demanda serverless constituye una mejor solución tal y como lo demuestra Menéndez y cols. (2023).

Como se indico en la discusión, nuestro estudio se limita a cargas de prueba lanzadas desde una sola ubicación geográfica, recomendamos para futuras investigaciones evaluaciones que se ejecuten desde diferentes ubicaciones o zonas geolocalizadas.

Asimismo, se propone considerar arquitecturas híbridas que integren ambos enfoques, aprovechando las ventajas complementarias de cada uno. Por ejemplo, los contenedores pueden emplearse para servicios persistentes y de larga duración, mientras que el modelo *serverless* puede destinarse a funciones eventuales o a la gestión de picos de carga. Este tipo de integración puede contribuir a optimizar tanto la eficiencia operativa como la relación costo-beneficio del sistema.

Referencias

Akour, M., y Alenezi, M. (2025). Reducing environmental impact with sustainable serverless computing. *Sustainability*, *17*(7). Descargado de https://www.mdpi.com/2071-1050/17/7/2999 doi: 10.3390/su17072999

Balalaie, A., Heydarnoori, A., y Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, *33*(3), 42-52. doi: 10.1109/MS.2016.64

Brogi, A., Carrasco, J., Cubo, J., D'Andria, F., Nitto, E. D., Guerriero, M., … Soldani, J. (2016). *Seaclouds: An open reference architecture for multi-cloud governance*. doi: 10.1007/978-3-319-48992-6_25

Carlos Guerrero, C. J., Isaac Lera. (2018). *Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications.*

Chiang, C.-T. (2024). A systematic literature network analysis of green information technology for sustainability: Toward smart and sustainable livelihoods. *Technological Forecasting and Social Change*, *199*, 123053. Descargado de https://www.sciencedirect.com/science/article/pii/S0040162523007382 doi: https://doi.org/10.1016/j.techfore.2023.123053

Chippagiri, S. (2025, enero). *The rise of serverless computing: A systematic review of challenges and solutions with optimization strategies* (Vol. 5) (n.º 1). Salt Lake City, UT, USA. Descargado de https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5138747 doi: 10.55454/rcsas.5.01.2025.006

Felter, W., Ferreira, A., Rajamony, R., y Rubio, J. (2015). *An updated performance comparison of virtual machines and linux containers* (Vol. 00). doi: 10.1109/ISPASS.2015.7095802

Ghorbian, M., y Ghobaei-Arani, M. (2024). A survey on the cold start latency approaches in serverless computing: an optimization-based perspective. *Computing*, *106*, 3755-3809. Descargado de https://doi.org/10.1007/s00607-024-01335-5 doi: 10.1007/s00607-024-01335-5

Ghorbian, M., y Ghobaei-Arani, M. (2025). *Serverless computing: Architecture, concepts, and applications.* Descargado de https://arxiv.org/abs/2501.09831

Hassan, H. B., Barakat, S. A., y Sarhan, Q. I. (2021, 12). *Survey on serverless computing* (Vol. 10). Springer Science and Business Media Deutschland GmbH. doi: 10.1186/s13677-021-00253-7

Katal, A., Dahiya, S., y Choudhury, T. (2023). Energy efficiency in cloud computing data centers: a survey on software technologies. *Cluster Computing*, *26*, 1845-1875. Descargado de https://doi.org/10.1007/s10586-022-03713-0 doi: 10.1007/s10586-022-03713-0

Kecskemeti, G., Marosi, A., y Kertész, A. (2016). *The entice approach to decompose monolithic services into microservices*. doi: 10.1109/HPCSim.2016.7568389

Kodakandla, N. (2021). *Serverless architectures: A comparative study of performance, scalability, and cost in cloud-native applications.*

Kratzke, N. (2015). *About microservices, containers and their underestimated impact on network performance*. doi: 10.13140/RG.2.1.2039.3046

Lekkala, C. (2023, 2). Containerization vs. serverless architectures for data pipelines. *Journal of Engineering and Applied Sciences Technology*, 1-5. Descargado de https://www.onlinescientificresearch.com/articles/containerization-vs-serverless-architectures-for-data-pipelines.pdf doi: 10.47363/JEAST/2023(5)258

Li, Y., Lin, Y., Wang, Y., Ye, K., y Xu, C. (2023, 3). Serverless computing: State-of-the-art, challenges and opportunities. *IEEE Transactions on Services Computing*, *16*, 1522-1539. doi: 10.1109/TSC.2022.3166553

Liu, C., Loo, B. T., y Mao, Y. (2011). Declarative automated cloud resource orchestration. En *Proceedings of the 2nd acm symposium on cloud computing.* Association for Computing Machinery. Descargado de https://doi.org/10.1145/2038916.2038942 doi: 10.1145/2038916.2038942

Manu, A., Patel, J. K., Akhtar, S., Agrawal, V. K., y Murthy, K. N. B. S. (2016). Docker container security via heuristics-based multilateral security-conceptual and pragmatic study. *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, 1-14. Descargado de https://api.semanticscholar.org/CorpusID:15649133

Mell, P., y Grance, T. (2011). *The nist definition of cloud computing.* Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD. doi: https://doi.org/10.6028/NIST.SP.800-145

Menéndez, J. M., Gayo, J. E. L., Canal, E. R., y Fernández, A. E. (2023, 5). A comparison between traditional and serverless technologies in a microservices setting.

Merkel, D. (2014, marzo). Docker: lightweight linux containers for consistent development and deployment. *Linux J.*, *2014*(239).

Mondal, S., Pan, R., y Kabir, H. e. a. (2022). Kubernetes in it administration and serverless computing: An empirical study and research challenges. *J Supercomput 78*, *17*(7). Descargado de https://doi.org/10.1007/s11227-021-03982-3 doi: 10.1007/s11227-021-03982-3

Muzumdar, P. (2022, 3). The effect of review valence on purchase of time-constrained and discounted goods. *Journal of Information Systems Applied Research and Analytics*, *15*, 11-23.

Muzumdar, P., Basyal, G. P., y Vyas, P. (2020, 11). Antecedents of student retention: A predictive modelling approach. *International Journal of Contemporary Research and Review*, *11*, 21906-21913. doi: 10.15520/ijcrr.v11i11.860

Muzumdar, P., Bhosale, A., Basyal, G. P., y Kurian, G. (2024, enero). *Navigating the docker ecosystem: A comprehensive taxonomy and survey.* Descargado de https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4682785

Pérez, A., Moltó, G., Caballer, M., y Calatrava, A. (2018). Serverless computing for container-based architectures. *Future Generation Computer Systems*, *83*, 50-59. Descargado de https://www.sciencedirect.com/science/article/pii/S0167739X17316485

doi: https://doi.org/10.1016/j.future.2018.01.022

Rad, Z. S., y Ghobaei-Arani, M. (2025). Federated serverless cloud approaches: A comprehensive review. *Computers and Electrical Engineering*, *124*, 110372. Descargado de https://www.sciencedirect.com/science/article/pii/S0045790625003155 doi: https://doi.org/10.1016/j.compeleceng.2025.110372

Sharma, P. (2023, octubre). Challenges and opportunities in sustainable serverless computing. *SIGENERGY Energy Inform. Rev.*, *3*(3), 53–58. Descargado de https://doi.org/10.1145/3630614.3630624 doi: 10.1145/3630614.3630624

Vaño, R., Lacalle, I., Sowiński, P., S-Julián, R., y Palau, C. E. (2023). Cloud-native workload orchestration at the edge: A deployment review and future directions. *Sensors*, *23*(4). Descargado de https://www.mdpi.com/1424-8220/23/4/2215 doi: 10.3390/s23042215

Velp, G. E. D., Rivière, E., y Sadre, R. (2020, 12). Understanding the performance of container execution environments. En *Woc 2020 - proceedings of the 2020 6th international workshop on container technologies and container clouds, part of middleware 2020* (p. 37-42). Association for Computing Machinery, Inc. doi: 10.1145/3429885.3429967

Wen, J., Chen, Z., Jin, X., y Liu, X. (2023, julio). Rise of the planet of serverless computing: A systematic review. *ACM Trans. Softw. Eng. Methodol.*, *32*(5). Descargado de https://doi.org/10.1145/3579643 doi: 10.1145/3579643

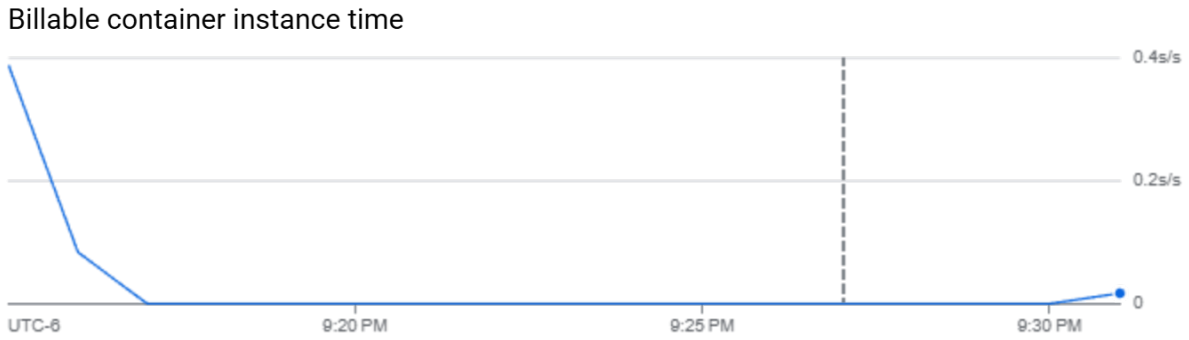
Yao, X., Chen, N., Yuan, X., y Ou, P. (2023, 2). Performance optimization of serverless edge computing function offloading based on deep reinforcement learning. *Future Generation Computer Systems*, *139*, 74-86. doi: 10.1016/j.future.2022.09.009

Yaqub, M. Z., y Alsabban, A. (2023). Industry-4.0-enabled digital transformation: Prospects, instruments, challenges, and implications for business strategies. *Sustainability*, *15*. Descargado de https://www.mdpi.com/2071-1050/15/11/8553 doi: 10.3390/su15118553

Apéndices

Figura A1

Resumen de costes en GCP en variación del tiempo



Apéndice A

Métricas Cloud Run (function - 25 usuarios)

Google GCP dentro de su consola proporciona una serie de resúmenes o métricas para poder medir el uso de la nube utilizados en un periodo de tiempo proporcionado, en este caso en el periodo de tiempo que duraron las pruebas para 25 usuarios, modalidad función y se muestran en las figuras A1 y A2.

Figura A2

Métrica de bytes recibidos en GCP

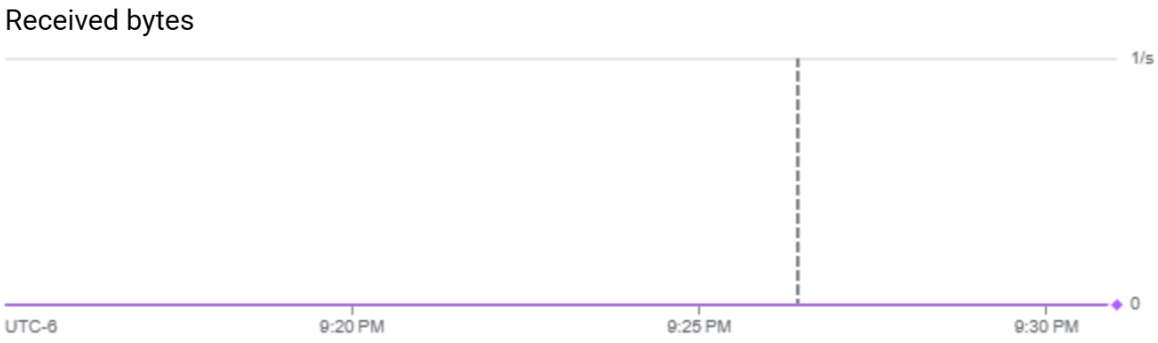
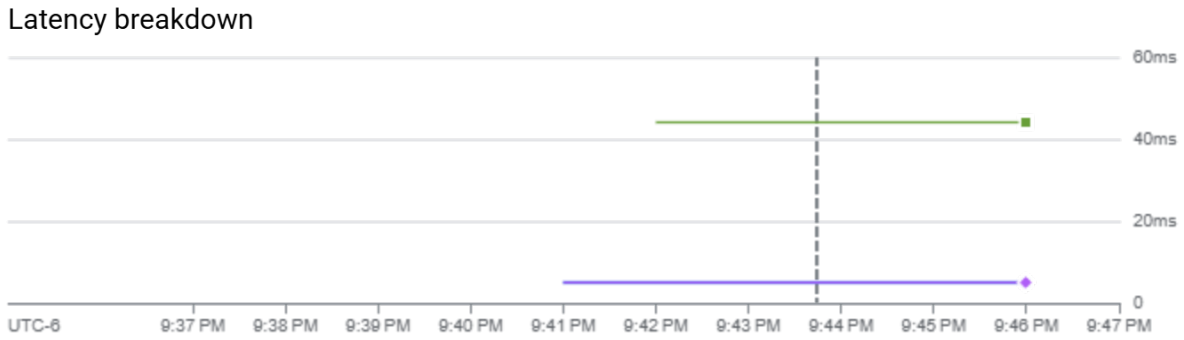


Figura B1

Análisis de la latencia en GCP



Apéndice B

Métricas Cloud Run (function - 100 usuarios)

Google GCP dentro de su consola proporciona una serie de resúmenes o métricas para poder medir el uso de la nube utilizados en un periodo de tiempo proporcionado, en este caso en el periodo de tiempo que duraron las pruebas para 100 usuarios, modalidad función y se muestran en las figuras B1, B2 y B3.

Figura B2

Resumen de costes en GCP en variación del tiempo

Figura B3

Métrica de bytes recibidos en GCP

Figura C1

Métrica de peticiones realizadas en GCP

Apéndice C

Métricas Cloud Run (container - 25 usuarios)

Google GCP dentro de su consola proporciona una serie de resúmenes o métricas para poder medir el uso de la nube utilizados en un periodo de tiempo proporcionado, en este caso en el periodo de tiempo que duraron las pruebas para 25 usuarios, modalidad contenedor y se muestran en las figuras C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11 y C12.

Figura C2

Resumen de latencia de las peticiones en GCP

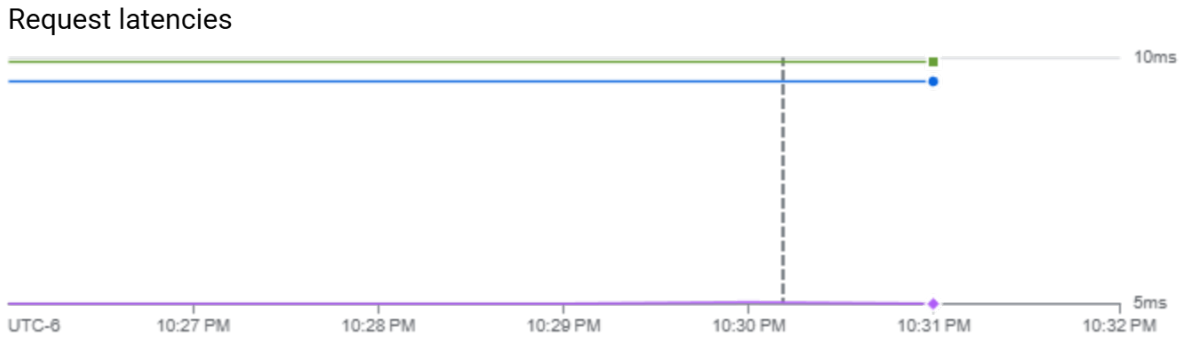


Figura C3

Métrica Latencia de la solicitud de extremo a extremo en GCP

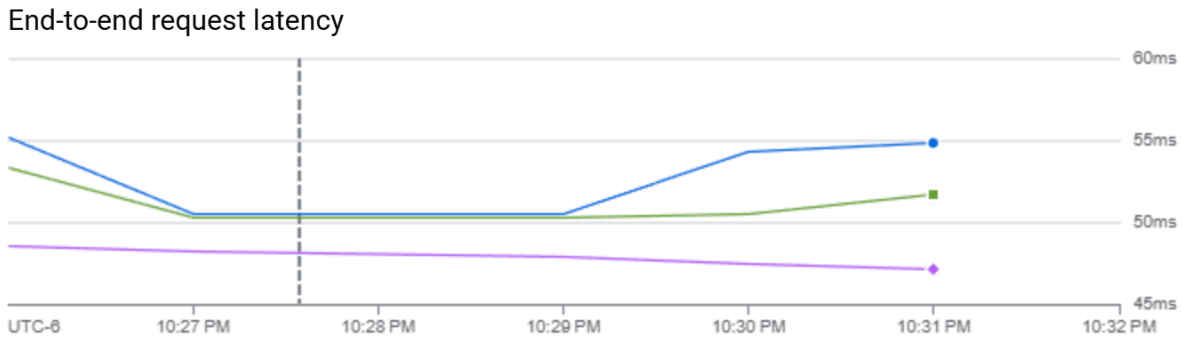


Figura C4

Análisis de la latencia en GCP

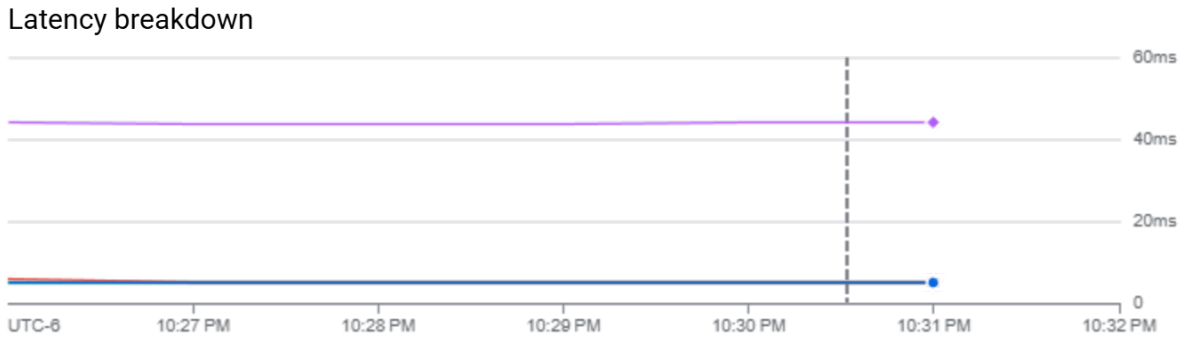


Figura C5

Resumen de instancias de contenedores utilizados en GCP

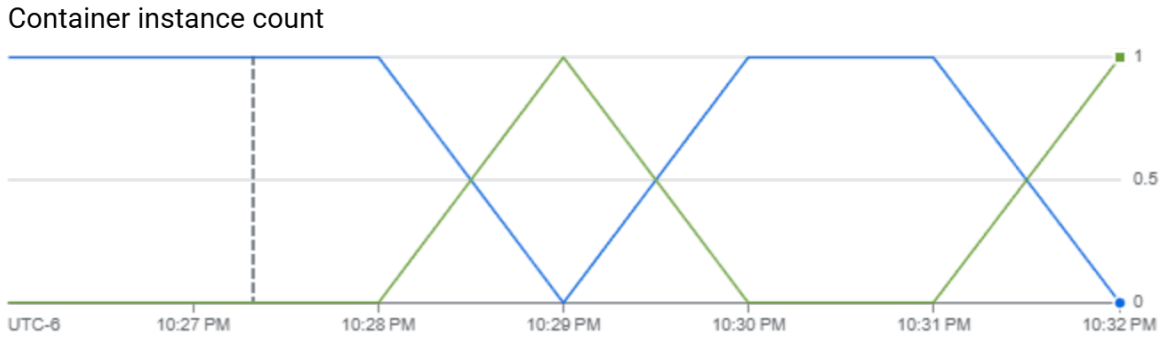


Figura C6

Resumen de costes en GCP en variación del tiempo

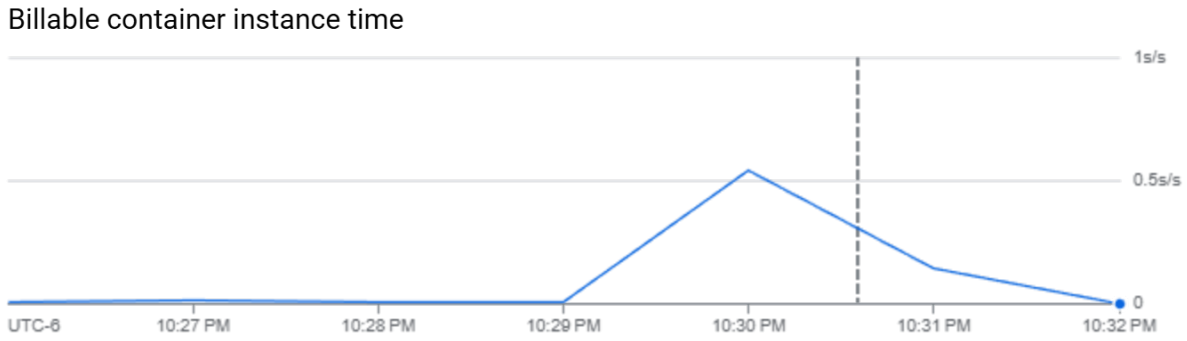


Figura C7

Utilización de CPU en GCP

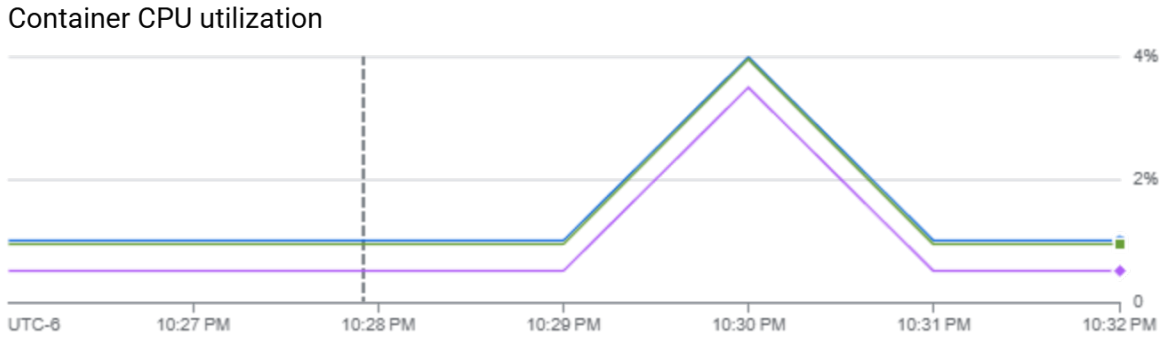


Figura C8

Utilización de memoria en GCP

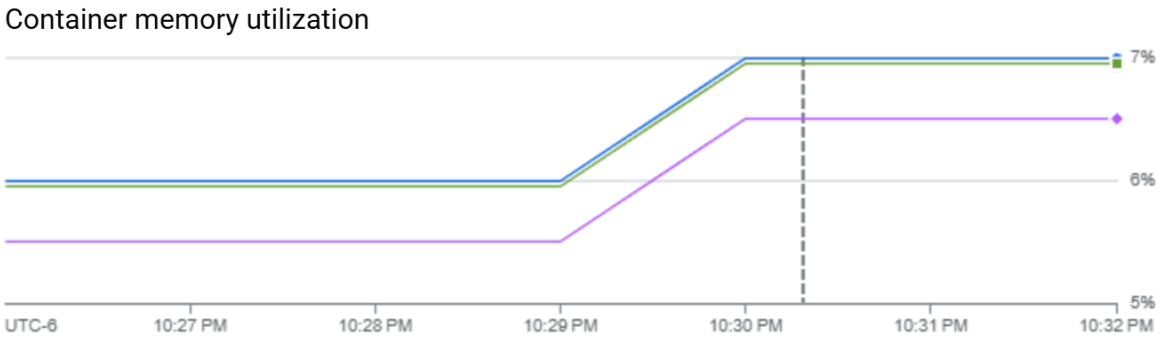


Figura C9

Métrica de bytes enviados en GCP

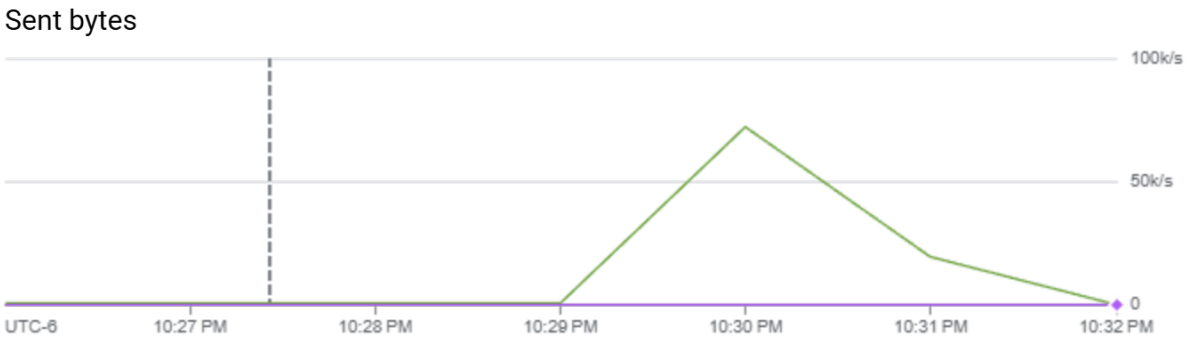


Figura C10

Métrica de bytes recibidos en GCP

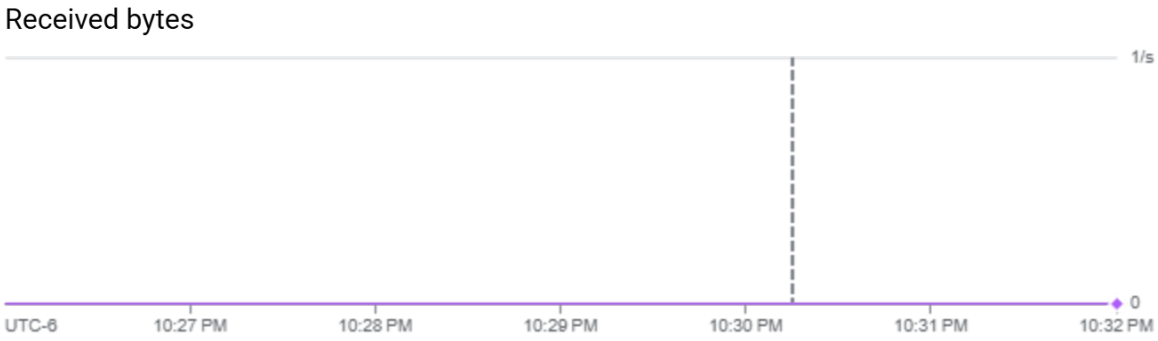


Figura C11

Máximo número de solicitudes simultáneas en GCP

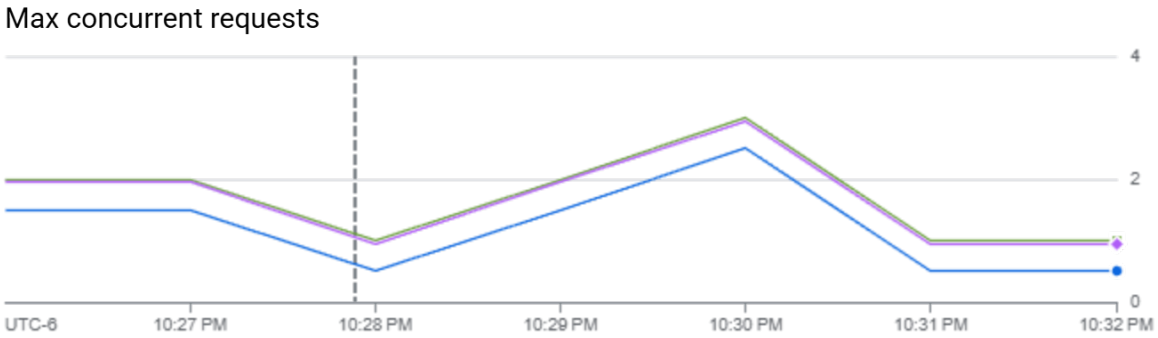
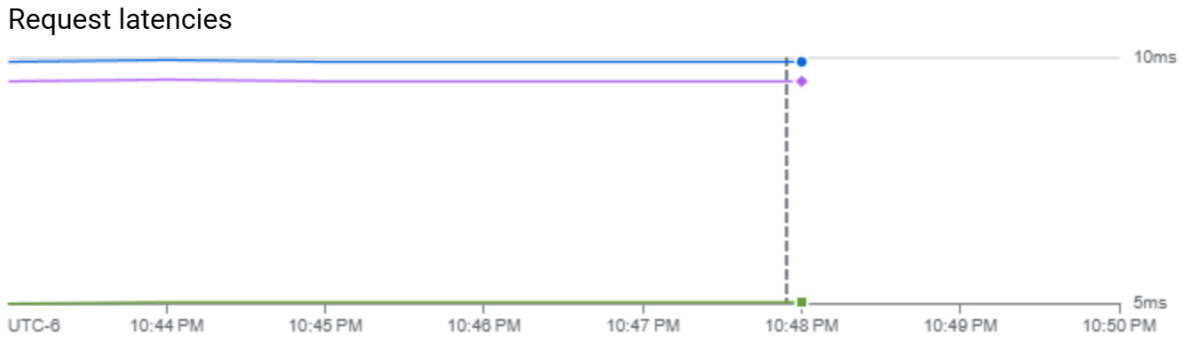


Figura C12

Latencia de arranque del contenedor en GCP

Figura D1

Resumen de latencia de las peticiones en GCP



Apéndice D

Métricas Cloud Run (container - 100 usuarios)

Google GCP dentro de su consola proporciona una serie de resúmenes o métricas para poder medir el uso de la nube utilizados en un periodo de tiempo proporcionado, en este caso en el periodo de tiempo que duraron las pruebas para 100 usuarios, modalidad contenedor y se muestran en las figuras D1, D2, D3, D4, D5, D6, D7 y D8

Figura D2

Métrica Latencia de la solicitud de extremo a extremo en GCP

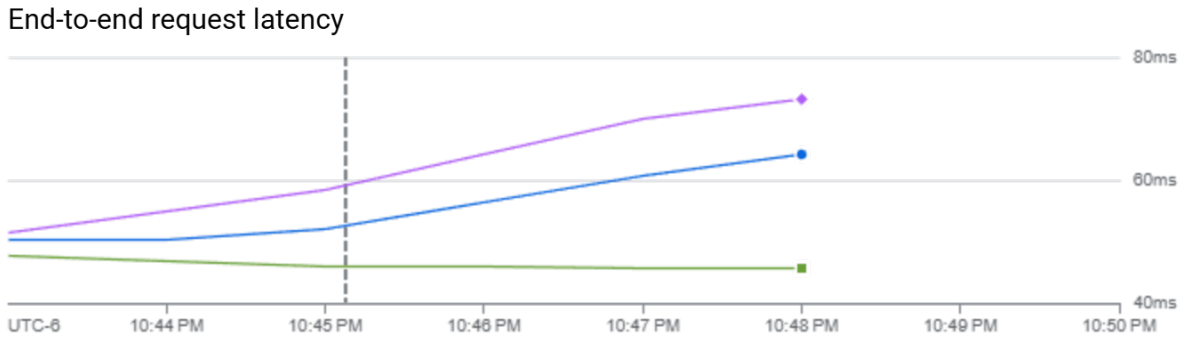


Figura D3

Análisis de la latencia en GCP

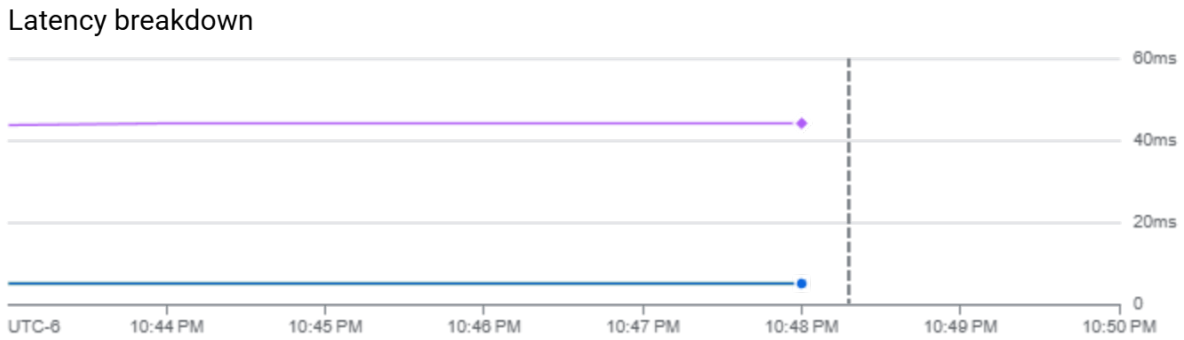


Figura D4

Resumen de instancias de contenedores utilizados en GCP

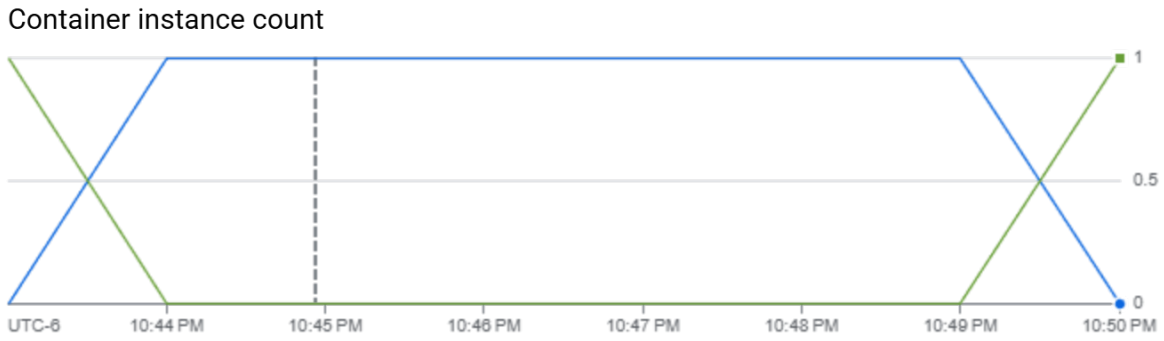


Figura D5

Resumen de costes en GCP en variación del tiempo

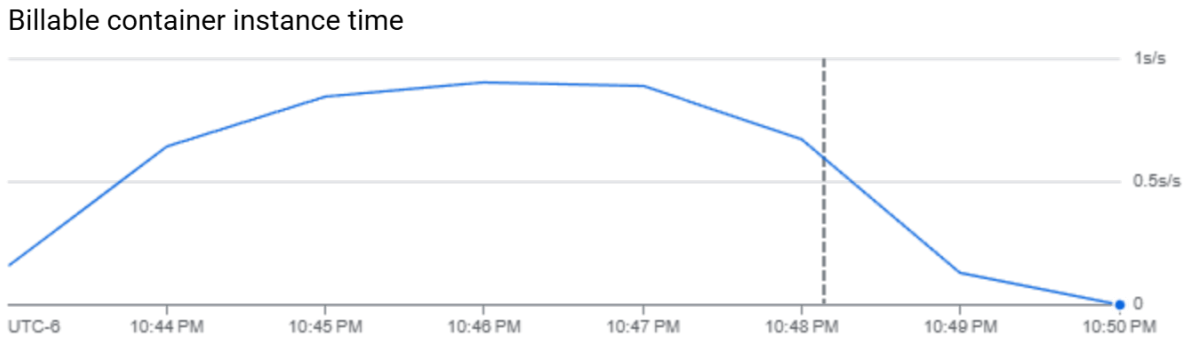


Figura D6

Utilización de CPU en GCP

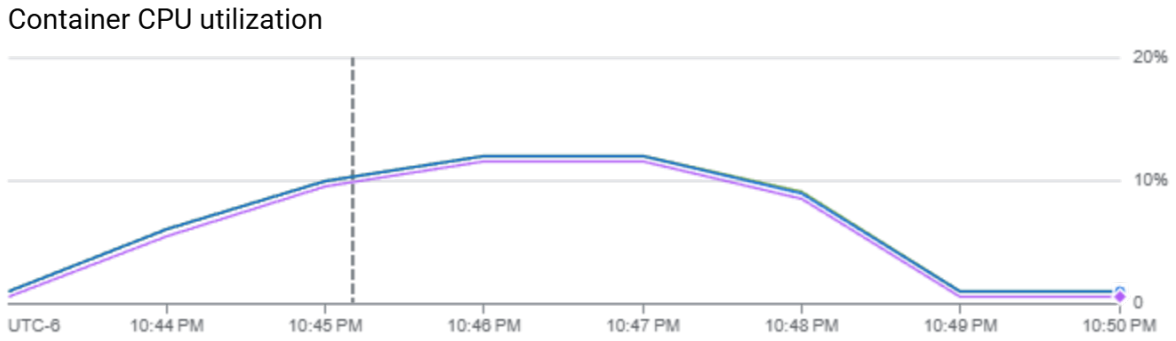


Figura D7

Utilización de memoria en GCP

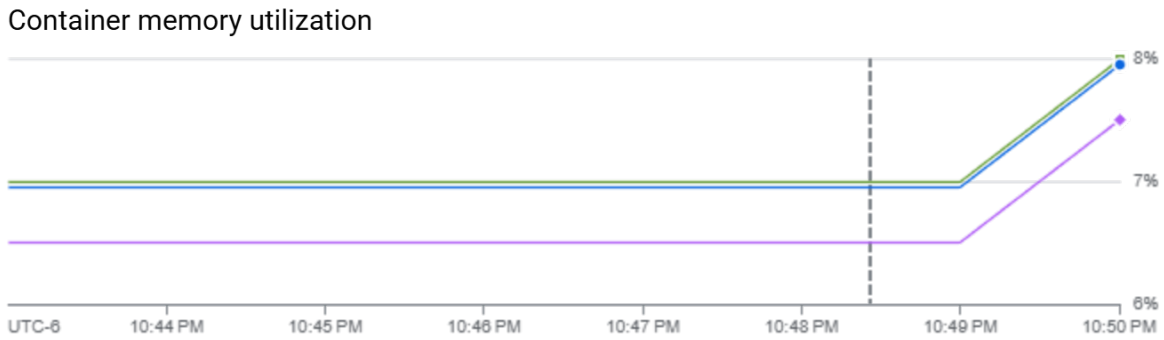


Figura D8

Métrica de bytes recibidos en GCP

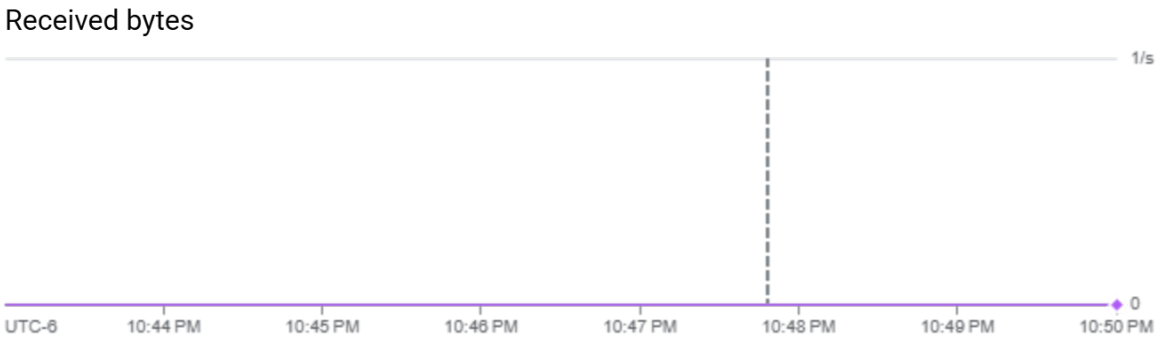
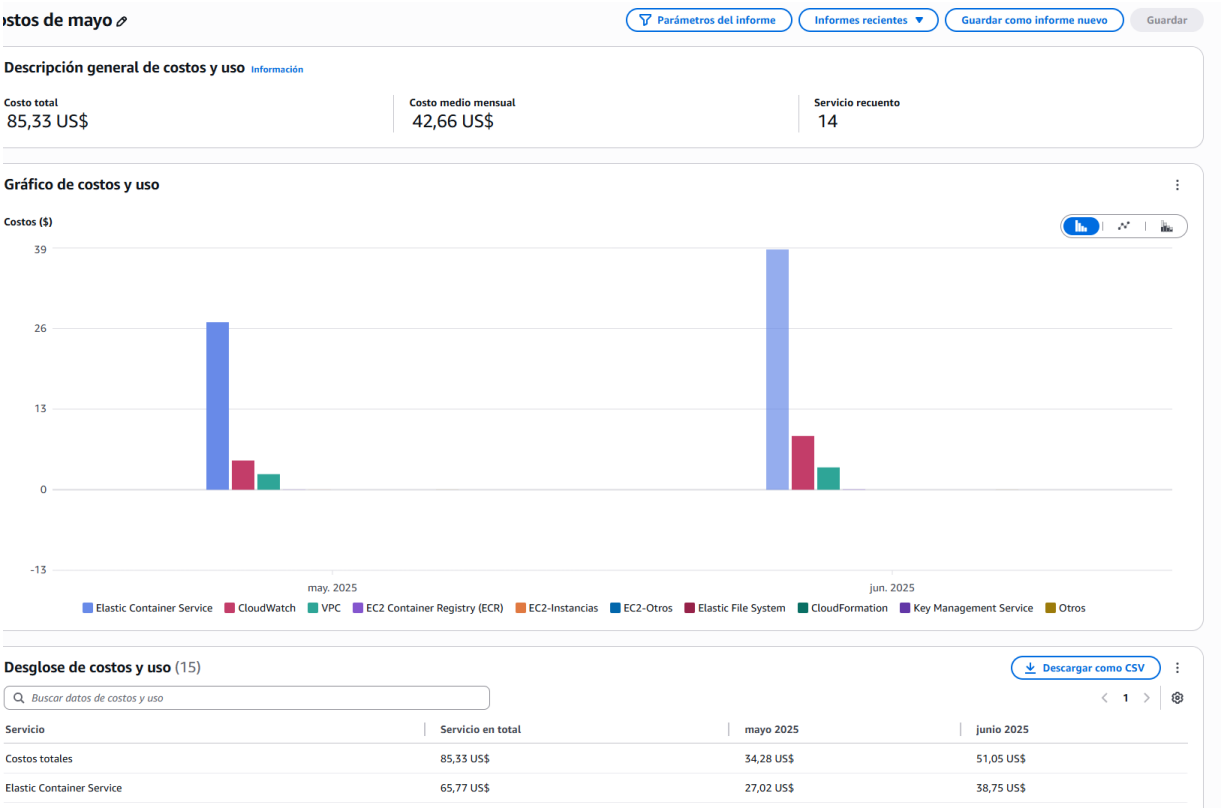


Figura E1

Resumen de costes en AWS utilizando Cost Explorer



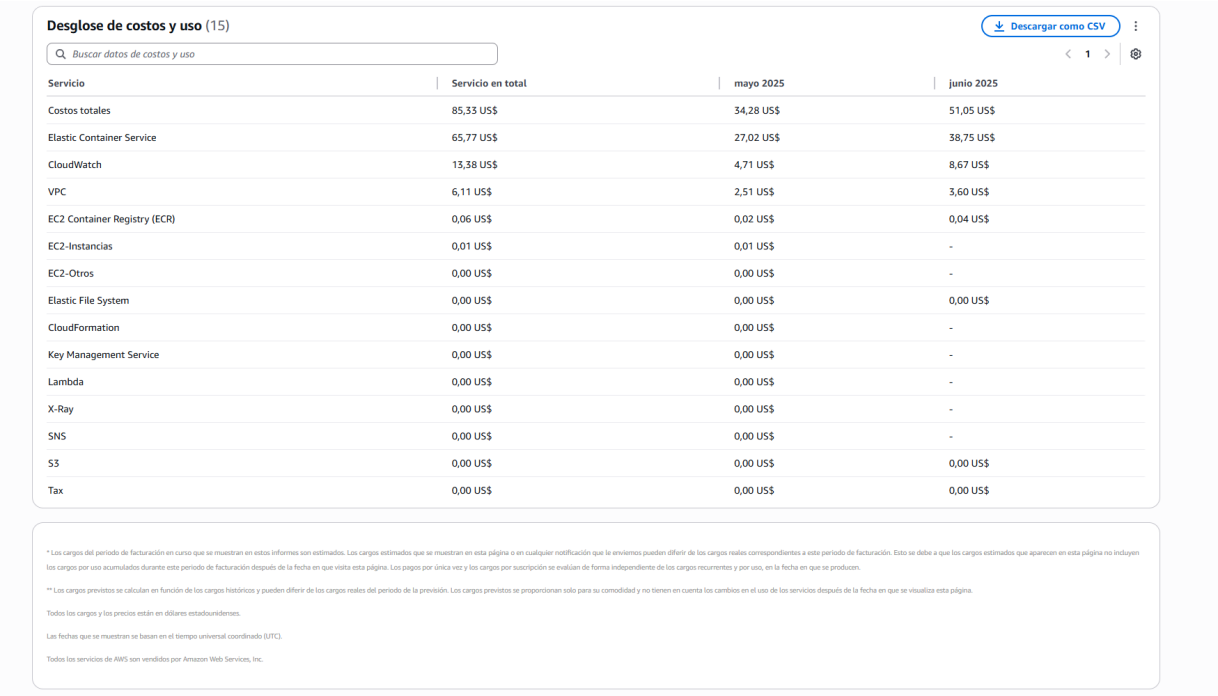
Apéndice E

Métricas Amazon AWS (container y lambda Function)

Amazon AWS proporciona resultados medibles a través de un componente de control y métricas llamado Cloud Watch, compartimos los resultados en las figuras E1,E2,E3, E4

Figura E2

Detalle de costes por servicio en AWS (Cost Explorer)



Desglose de costos y uso (15)

Descargar como CSV

Buscar datos de costos y uso

1

Servicio	Servicio en total	mayo 2025	junio 2025
Costos totales	85,33 US$	34,28 US$	51,05 US$
Elastic Container Service	65,77 US$	27,02 US$	38,75 US$
CloudWatch	13,38 US$	4,71 US$	8,67 US$
VPC	6,11 US$	2,51 US$	3,60 US$
EC2 Container Registry (ECR)	0,06 US$	0,02 US$	0,04 US$
EC2-Instancias	0,01 US$	0,01 US$	-
EC2-Otros	0,00 US$	0,00 US$	-
Elastic File System	0,00 US$	0,00 US$	0,00 US$
CloudFormation	0,00 US$	0,00 US$	-
Key Management Service	0,00 US$	0,00 US$	-
Lambda	0,00 US$	0,00 US$	-
X-Ray	0,00 US$	0,00 US$	-
SNS	0,00 US$	0,00 US$	-
S3	0,00 US$	0,00 US$	0,00 US$
Tax	0,00 US$	0,00 US$	0,00 US$

* Los cargos del periodo de facturación en curso que se muestran en estos informes son estimados. Los cargos estimados que se muestran en esta página o en cualquier notificación que le enviemos pueden diferir de los cargos reales correspondientes a este periodo de facturación. Esto se debe a que los cargos estimados que aparecen en esta página no incluyen los cargos por uso acumulados durante este periodo de facturación después de la fecha en que visita esta página. Los pagos por única vez y los cargos por suscripción se evalúan de forma independiente de los cargos recurrentes y por uso, en la fecha en que se producen.

** Los cargos previstos se calculan en función de los cargos históricos y pueden diferir de los cargos reales del periodo de la previsión. Los cargos previstos se proporcionan solo para su comodidad y no tienen en cuenta los cambios en el uso de los servicios después de la fecha en que se visualiza esta página.

Todos los cargos y los precios están en dólares estadounidenses.

Las fechas que se muestran se basan en el tiempo universal coordinado (UTC).

Todos los servicios de AWS son vendidos por Amazon Web Services, Inc.

Figura E3

Métricas de uso de recursos en AWS a partir de Cost Explorer

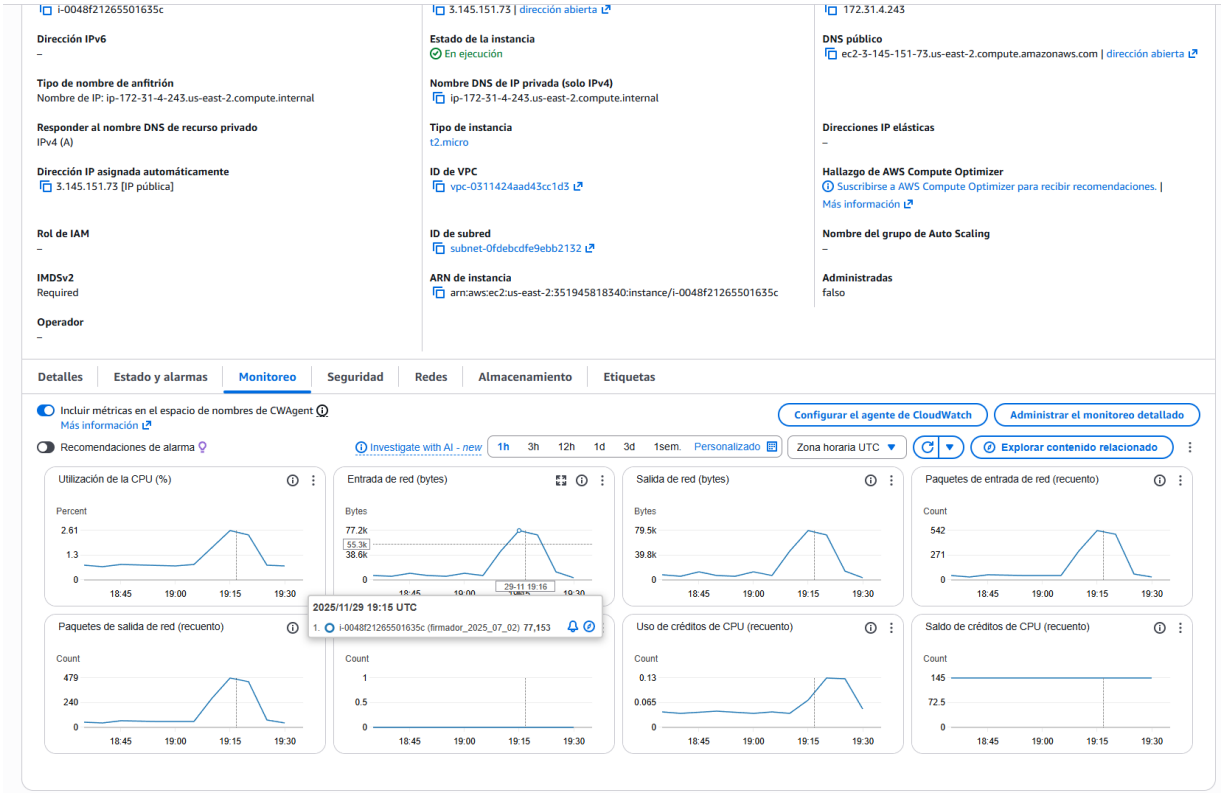
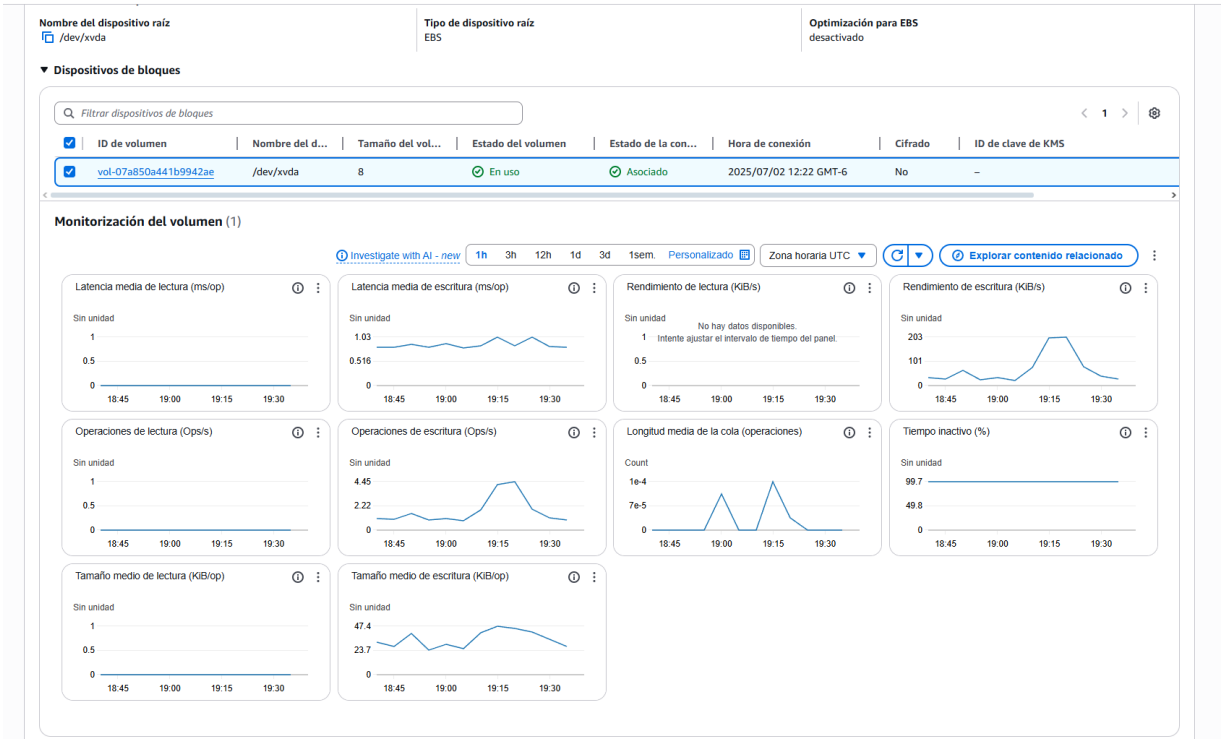


Figura E4

Métricas de almacenamiento en AWS para la arquitectura evaluada



Apéndice F

Resultados Locust para AWS (container y lambda Function)

Para Amazon AWS se corrieron similares pruebas de carga, consistieron en simular 100 y 25 usuarios simultáneos a un incremento de 1 usuario /segundo, presentamos los resultados en las figuras F1,F2,F3,F4

Figura F1

Prueba de carga en AWS Lambda (25 usuarios concurrentes)

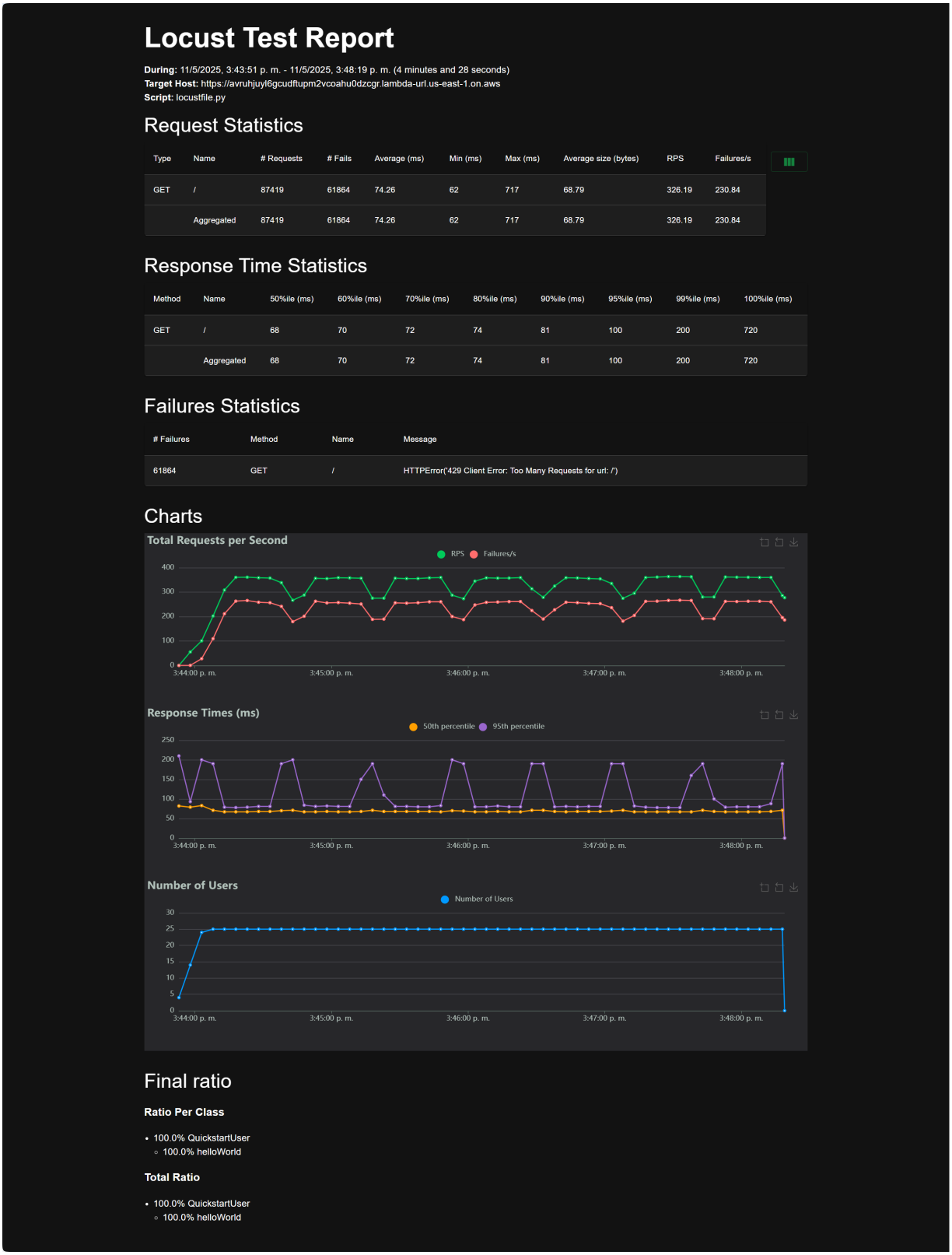
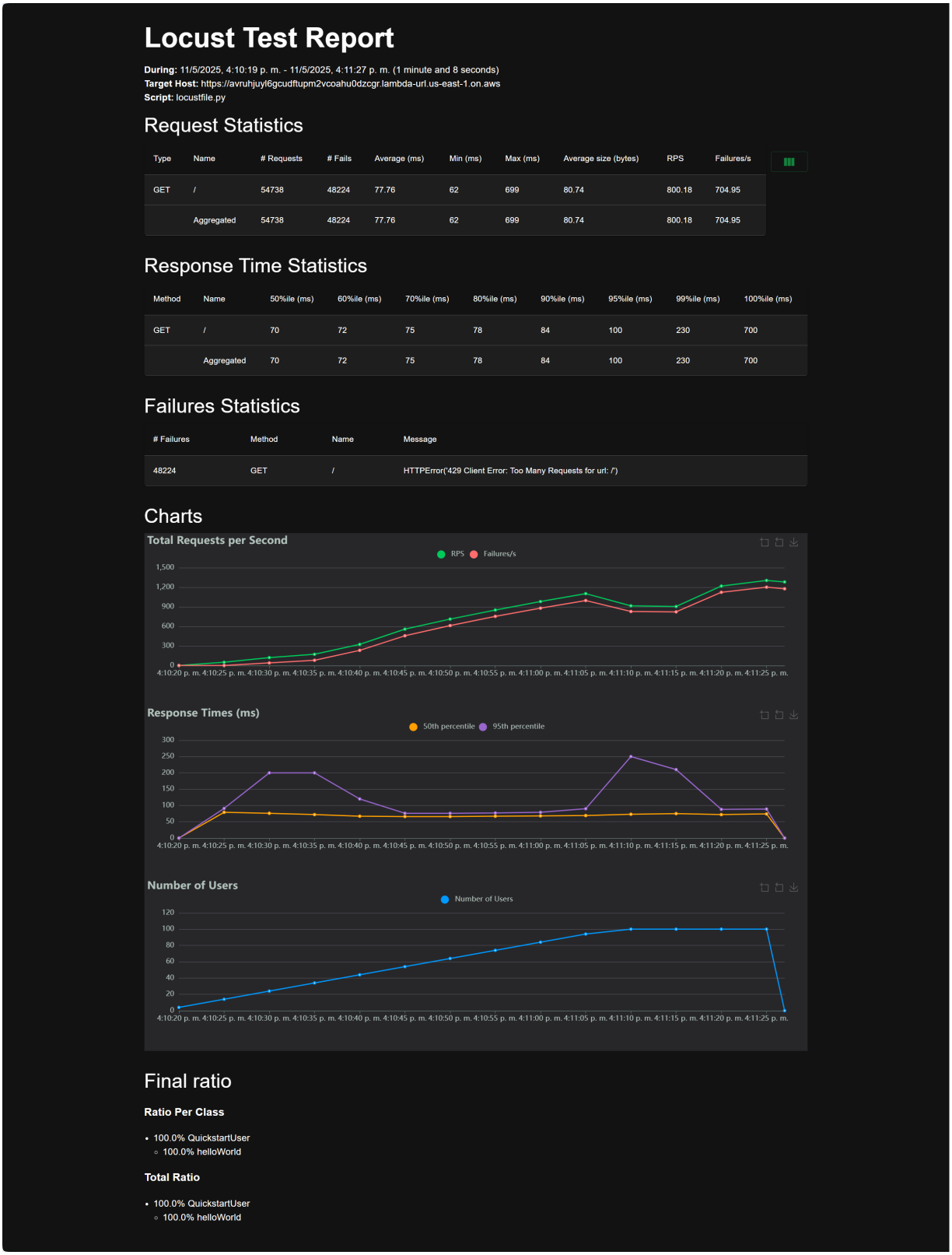


Figura F2

Prueba de carga en AWS Lambda (100 usuarios concurrentes)



Locust Test Report

During: 11/5/2025, 4:10:19 p. m. - 11/5/2025, 4:11:27 p. m. (1 minute and 8 seconds)
Target Host: https://avruhjuyl6gcudftupm2vcoahu0dzcgr.lambda-url.us-east-1.on.aws
Script: locustfile.py

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/	54738	48224	77.76	62	699	80.74	800.18	704.95
	Aggregated	54738	48224	77.76	62	699	80.74	800.18	704.95

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/	70	72	75	78	84	100	230	700
	Aggregated	70	72	75	78	84	100	230	700

Failures Statistics

# Failures	Method	Name	Message
48224	GET	/	HTTPError('429 Client Error: Too Many Requests for url: /')

Charts

Final ratio

Ratio Per Class

- 100.0% QuickstartUser
  - 100.0% helloWorld

Total Ratio

- 100.0% QuickstartUser
  - 100.0% helloWorld

Figura F3

Prueba de carga en AWS con contenedores (25 usuarios concurrentes)

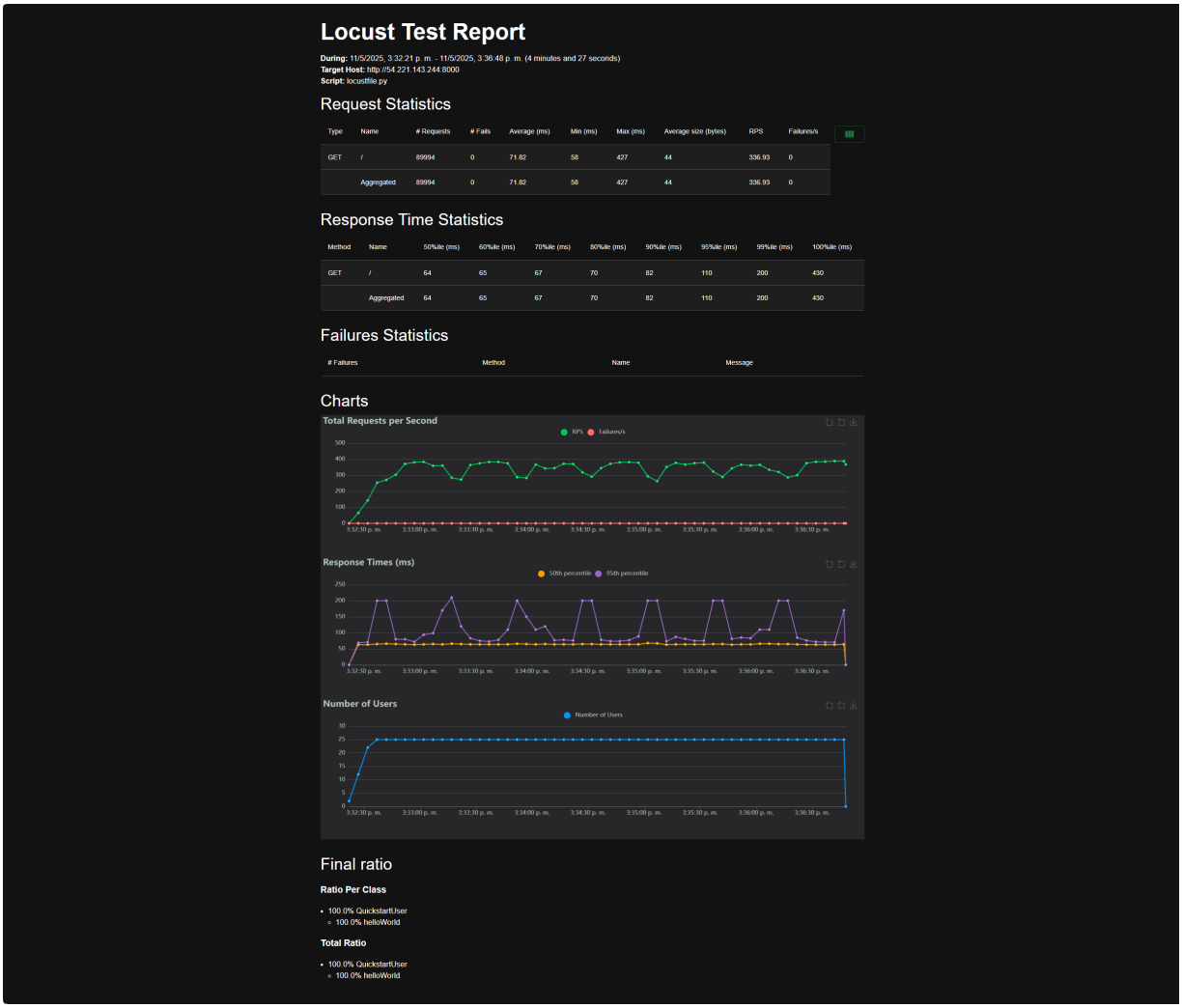


Figura F4

Prueba de carga en AWS con contenedores (100 usuarios concurrentes)

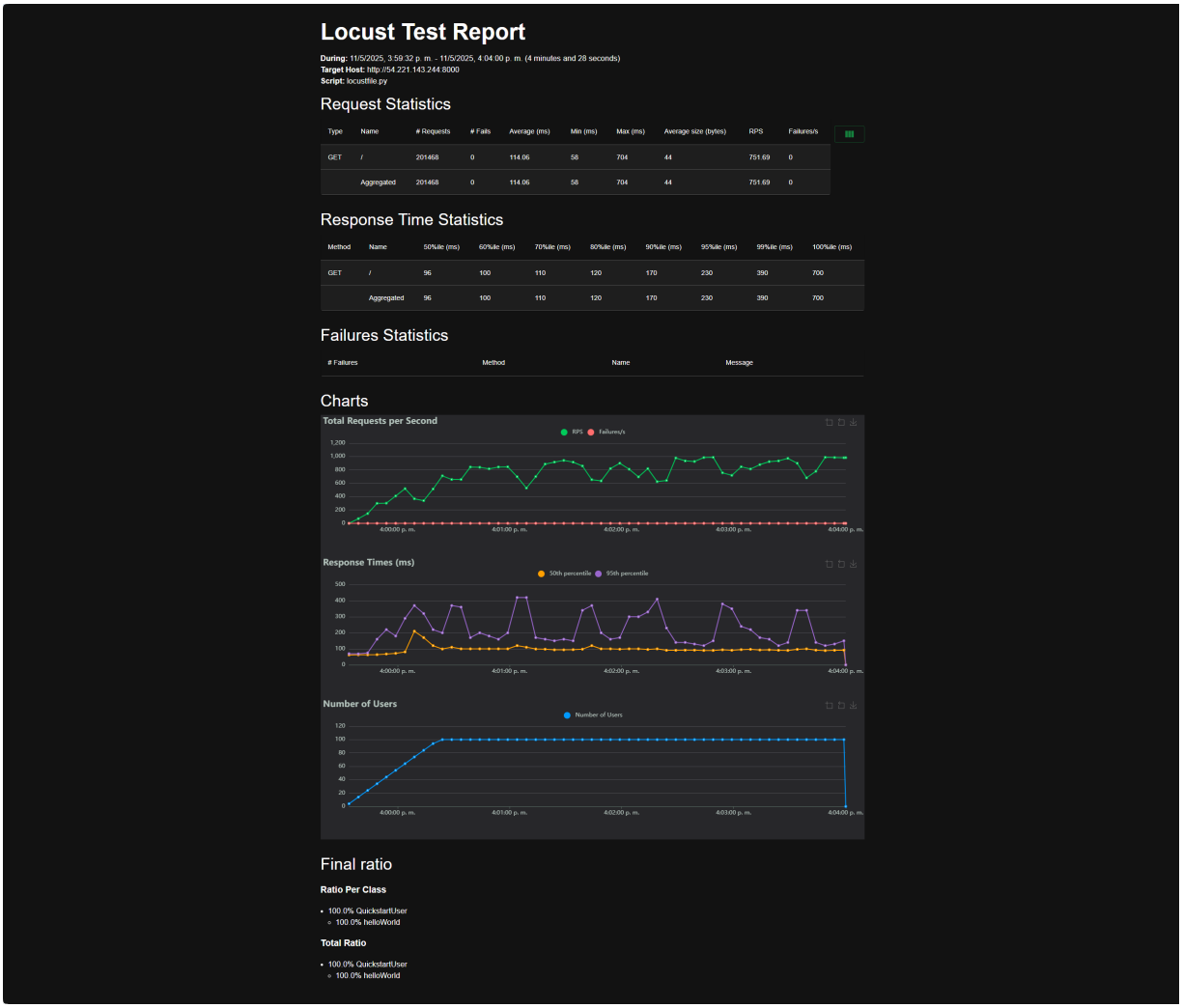
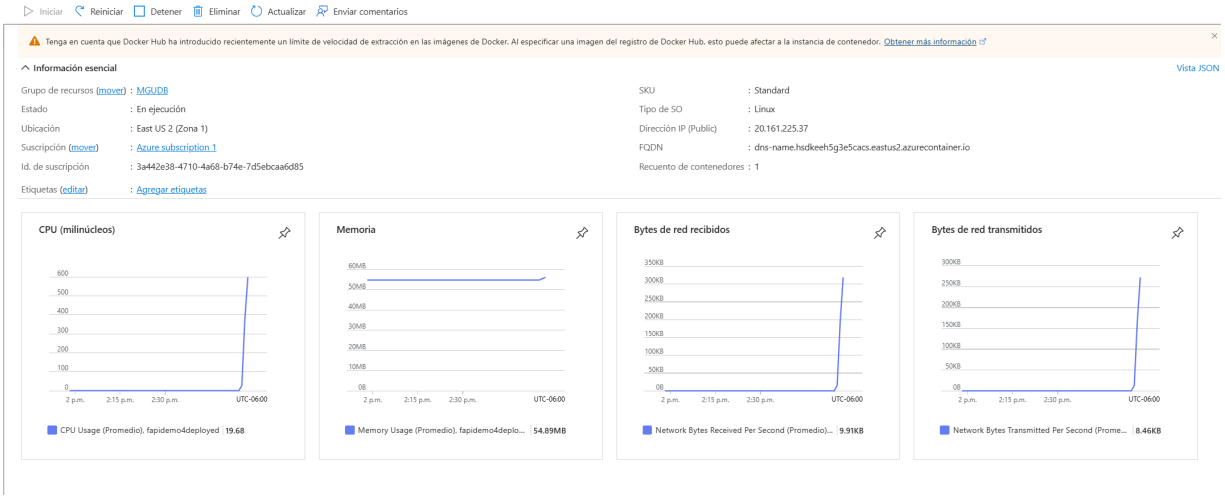


Figura G1

Métricas de uso de recursos en Azure para la arquitectura basada en contenedores



Apéndice G

Métricas Microsoft Azure (container y App Functions)

En el lado del servidor Microsoft Azure provee métricas concretas, los resultados se fundamentaron en ellas, a continuación presentamos pruebas de cada una de ellas en las figuras G1, G2, G3

Figura G2

Métricas de uso de recursos en Azure para la arquitectura serverless (funciones Lambda)

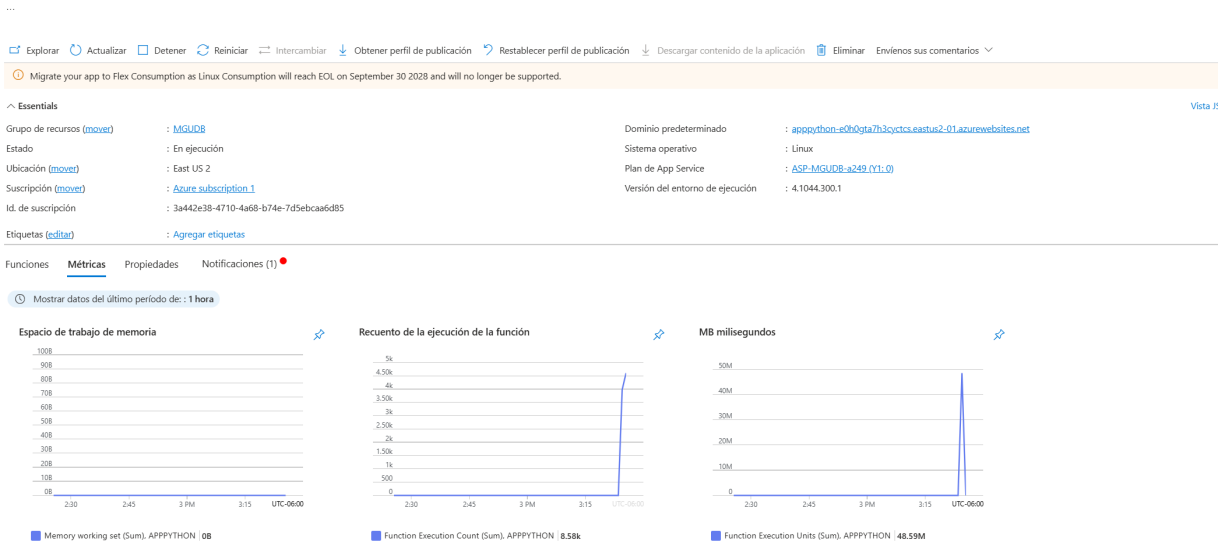
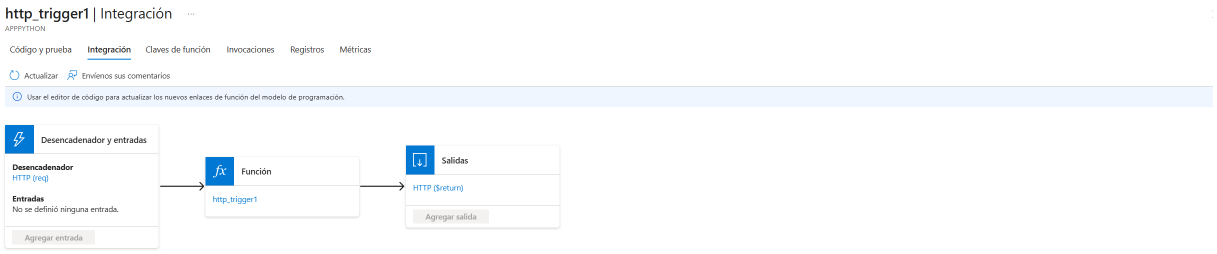


Figura G3

Configuración de desencadenadores (triggers) en Azure Functions para la aplicación evaluada



Apéndice H

Resultados Locust para Azure (container y lambda Function)

Las pruebas de estrés ejecutadas sobre Microsoft Azure arrojaron los resultados de las figuras H2,H1,H3,H4 estas pruebas se realizaron utilizando la herramientas Locust, ejecutando distintas test de carga para evaluar los resultados promedios de cada una de ellas:

Figura H1

Prueba de carga con contenedores en Azure (100 usuarios concurrentes)

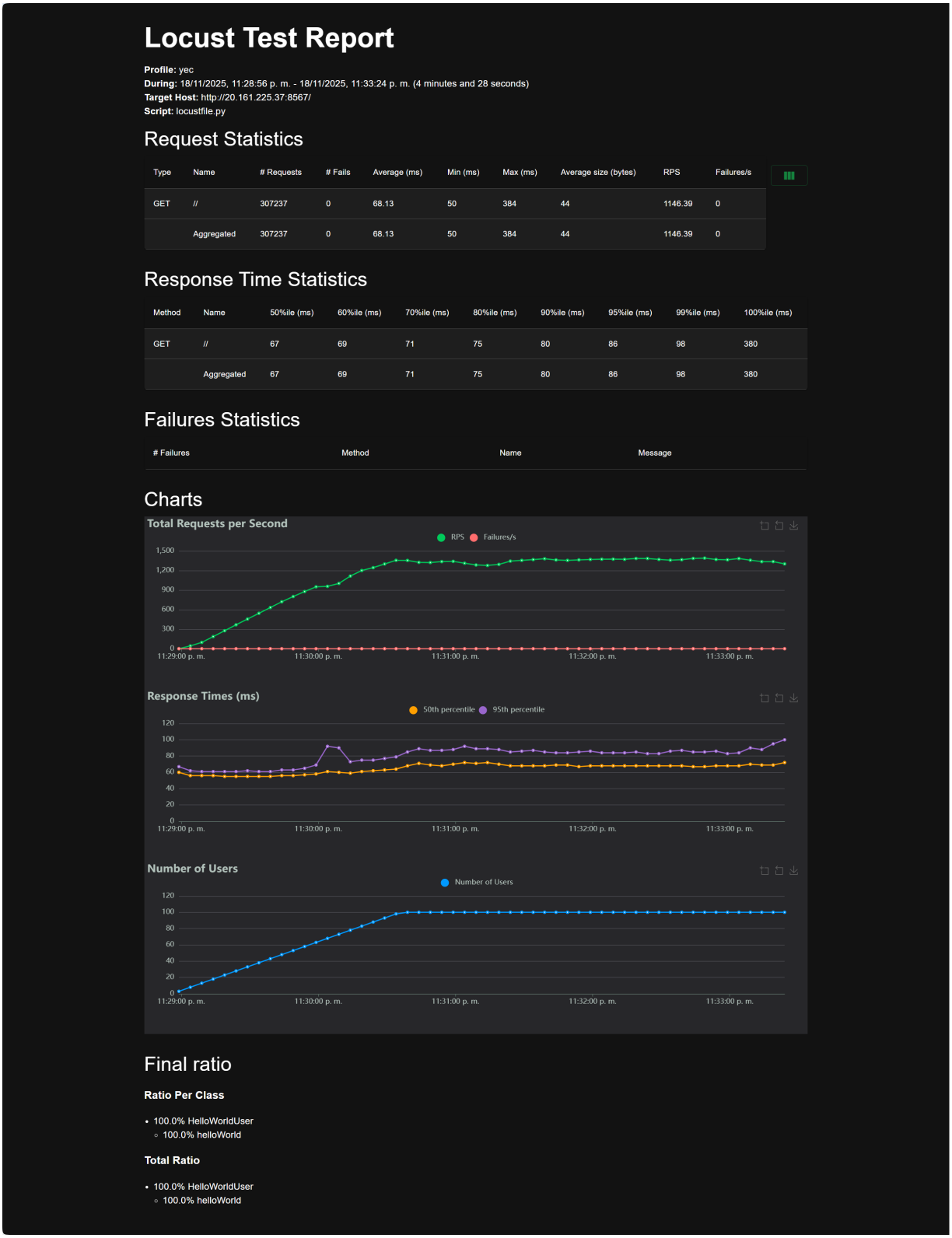


Figura H2

Prueba de carga con contenedores en Azure (25 usuarios concurrentes)

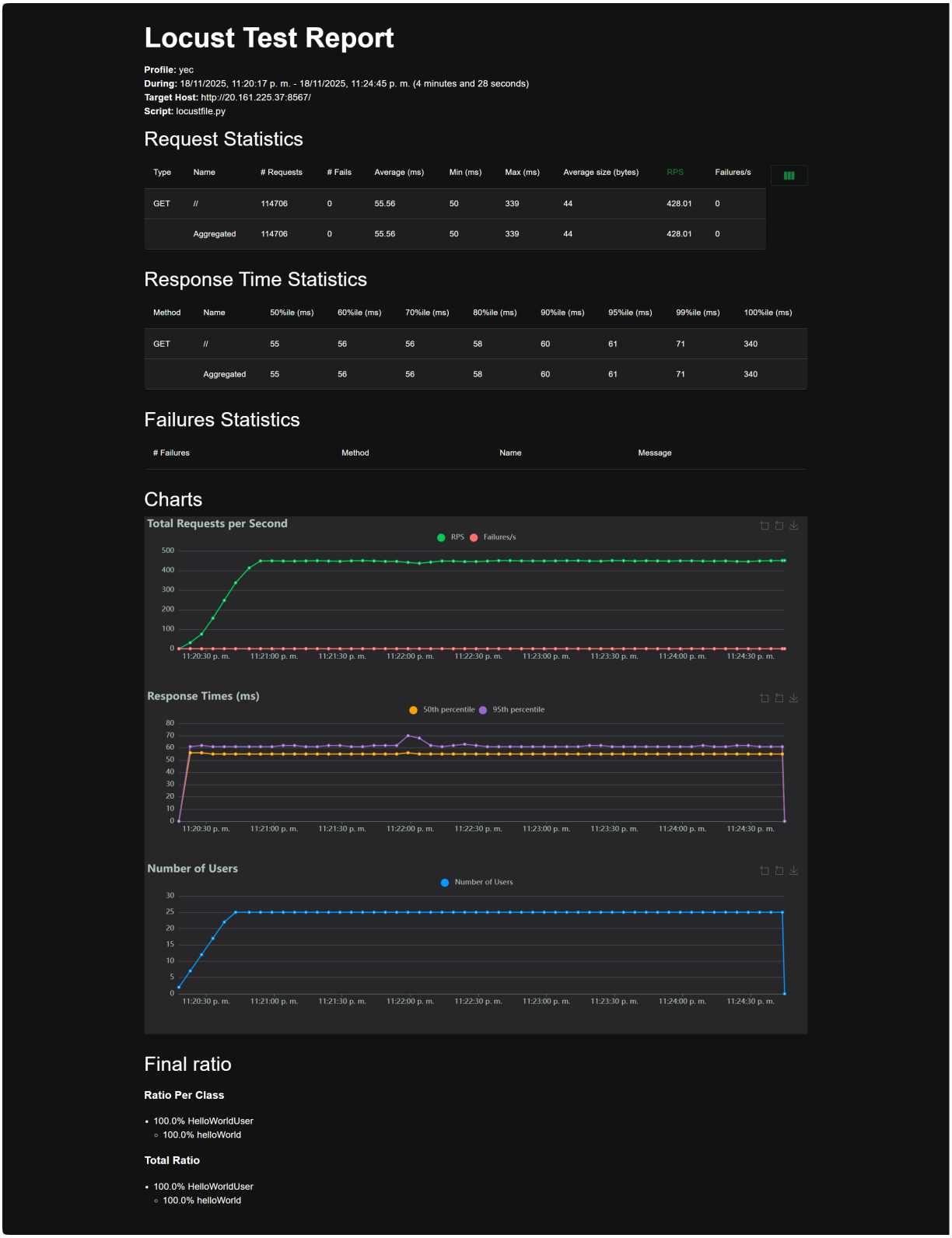


Figura H3

Prueba de carga con funciones Lambda en Azure (100 usuarios concurrentes)

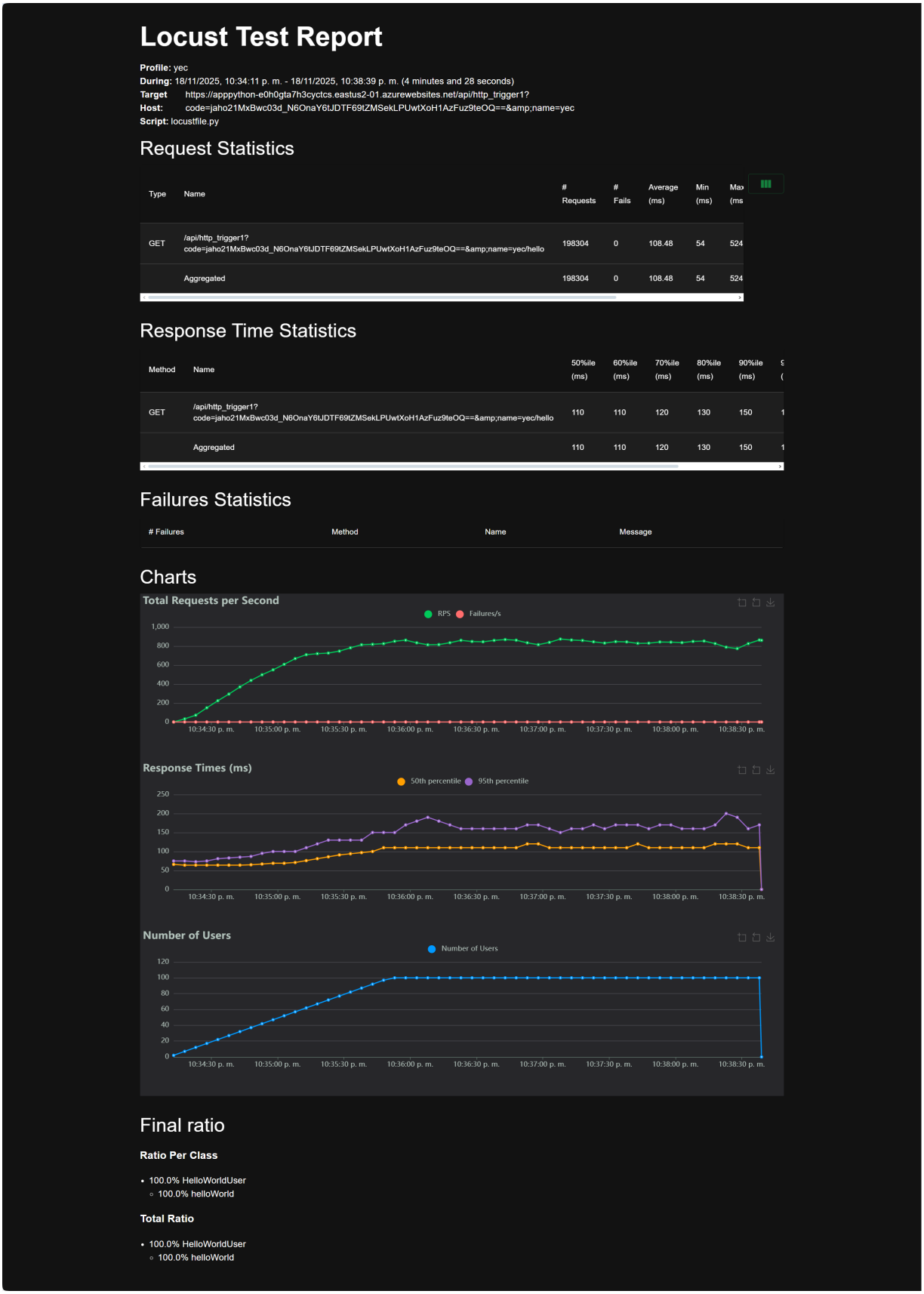


Figura H4

Prueba de carga con funciones Lambda en Azure (25 usuarios concurrentes)