Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Electricidad y Computación

Desarrollo de un sistema de auto-configuración de cómputo en la nube para reducción de costos

TECH - 400

Proyecto Integrador

Previo la obtención del Título de:

Ingeniero/a en Ciencias de la Computación

Presentado por:

Stefany Natalia Farías Mera

Jorge Luis Moncayo Paz

Guayaquil - Ecuador

Año: 2024

Declaración Expresa

Nosotros Jorge Luis Moncayo Paz y Stefany Natalia Farías Mera acordamos y reconocemos que:

La titularidad de los derechos patrimoniales de autor (derechos de autor) del proyecto de graduación corresponderá al autor o autores, sin perjuicio de lo cual la ESPOL recibe en este acto una licencia gratuita de plazo indefinido para el uso no comercial y comercial de la obra con facultad de sublicenciar, incluyendo la autorización para su divulgación, así como para la creación y uso de obras derivadas. En el caso de usos comerciales se respetará el porcentaje de participación en beneficios que corresponda a favor del autor o autores.

La titularidad total y exclusiva sobre los derechos patrimoniales de patente de invención, modelo de utilidad, diseño industrial, secreto industrial, software o información no divulgada que corresponda o pueda corresponder respecto de cualquier investigación, desarrollo tecnológico o invención realizada por nosotros durante el desarrollo del proyecto de graduación, pertenecerán de forma total, exclusiva e indivisible a la ESPOL, sin perjuicio del porcentaje que nos corresponda de los beneficios económicos que la ESPOL reciba por la explotación de nuestra innovación, de ser el caso.

En los casos donde la Oficina de Transferencia de Resultados de Investigación (OTRI) de la ESPOL comunique los autores que existe una innovación potencialmente patentable sobre los resultados del proyecto de graduación, no se realizará publicación o divulgación alguna, sin la autorización expresa y previa de la ESPOL.

Guayaquil, 9 de octubre del 2024.

Jorge Moncayo Paz
Jorge Luis Moncayo Paz

Stefany Farías M.
Stefany Natalia Farías Mera

Evaluadores

Firmado electrónicamente por:
JAVIER ALEJANDRO TIBAU BENITEZ

Ph.D. Javier Alejandro Tibau Benítez

Profesor de Materia

Firmado electrónicamente por:
CRISTINA LUCIA ABAD ROBALINO

Ph.D. Cristina Lucía Abad Robalino

Tutora de proyecto

Resumen

El presente trabajo aborda la implementación de un sistema de autoescalado inteligente para aplicaciones en la nube utilizando aprendizaje por refuerzo (RL). El objetivo principal es diseñar y evaluar un modelo que optimice la asignación de recursos en AWS mediante un entrenamiento híbrido, combinando métricas simuladas y reales. Se plantea la hipótesis de que este enfoque mejorará la eficiencia y reducirá los costos operativos en comparación con estrategias tradicionales de escalamiento dinámico. Se implementó un flujo de despliegue en AWS, iniciando en Lambda y escalando hacia Fargate y EC2 en función de la carga de trabajo. Se utilizaron servicios como Amazon CloudWatch para la monitorización, AWS ECR para la gestión de contenedores, API Gateway y Application Load Balancer para la creación de un endpoint global que unifique las urls de cada servicio. Se implementó un agente de RL que, basado en datos de tráfico y consumo de recursos, aprendió políticas de escalamiento óptimas. Los resultados mostraron que el script definido con reglas heurísticas logró reducir los tiempos de respuesta y mejorar la utilización de recursos en un 30 % en comparación con estrategias de escalamiento estático.

Palabras clave: autoescalado, aprendizaje por refuerzo, computación en la nube, AWS.

Abstract

This paper addresses the implementation of an intelligent autoscaling system for cloud applications using reinforcement learning (RL). The primary objective is to design and evaluate a model that optimizes resource allocation in AWS through hybrid training, combining simulated and real-world metrics. The hypothesis is that this approach will improve efficiency and reduce operational costs compared to traditional dynamic scaling strategies. A deployment flow was implemented in AWS, starting in Lambda and scaling to Fargate and EC2 based on the workload. Services such as Amazon CloudWatch for monitoring, AWS ECR for container management, API Gateway, and an Application Load Balancer were used to create a global endpoint that unifies the URLs of each service. An RL agent was implemented that, based on traffic data and resource consumption, learned optimal scaling policies. The results showed that the RL model was able to reduce response times and improve resource utilization by 30% compared to static scaling strategies.

Keywords: *autoscaling, reinforcement learning, cloud computing, AWS.*

Índice general

Abreviaturas

API	Application Programming Interface
ARN	Amazon Resource Name
AWS	Amazon Web Services
CLI	Command Line Interface
CPU	Central Processing Unit
DNS	Domain Name System
ECR	Elastic Container Registry
EC2	Elastic Compute Cloud
ECS	Elastic Container Service
FaaS	Function as a Service
HTTP	HyperText Transfer Protocol
IAM	Identity and Access Management
IP	Internet Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LAN	Local Area Network
ML	Machine Learning
PaaS	Platform as a Service
RAM	Random Access Memory
SaaS	Software as a Service
SG	Security Group
TCP	Transmission Control Protocol
TI	Tecnologías de información
URL	Uniform Resource Locator

Índice de figuras

Índice de tablas

Capítulo 1

1 Introducción

1.1 Descripción del Problema

En la computación en la nube la optimización de los costos es un gran reto para todas las empresas que dependen de servicios en la nube como los que ofrece Amazon Web Services (AWS) [1]. Los servicios de AWS tienen diferentes ventajas en términos de costos, rendimiento, escalabilidad, carga de trabajo tráfico, uso del CPU y elasticidad, seleccionar un solo servicio adecuado para una aplicación puede ser complicado [2] [3]. AWS Lambda y AWS Fargate son dos servicios distintos que permiten ejecutar aplicaciones en la nube de Amazon sin necesidad de gestionar servidores [4], mientras que Amazon Elastic Compute Cloud (Amazon EC2) permite gestionar servidores virtuales en la nube.

AWS Lambda es ideal para tareas pequeñas e invocaciones de corta duración, pero se vuelve costoso e insuficiente para aplicaciones que requieren largas ejecuciones, es gratuito hasta por un millón de solicitudes y a partir de ahí tiene un precio de 0,20 USD por un millón de solicitudes [5]. AWS Fargate es más adecuado para cargas de trabajo más pesadas y persistentes que requieren más CPU o memoria, su precio es de $0.04048 por vCPU/hora [6]. Mientras, que AWS EC2 se ajusta bien a aplicaciones web que reciben cargas de trabajo altas y contantes, y requieren escalar de manera flexible, pero su costo es más elevado, va desde los $0,0052 y puede alcanzar los $109,20 por capacidad de cómputo por hora [7].

Actualmente, la mayoría de las empresas seleccionan un servicio basándose en predicciones que no siempre reflejan el comportamiento real de la aplicación en producción, lo que lleva a costos innecesarios o una subutilización de los recursos [8]. También en muchas ocasiones los desarrolladores deben configurar manualmente los servicios y supervisar el uso de recursos para tomar decisiones informadas. Esto genera la necesidad de automatizar este proceso con herramientas que puedan de manera dinámica tomar la decisión de cambiar el servicio en la nube basado en métricas claves como el rendimiento, la duración de las tareas, el tráfico y el uso del CPU y la memoria.

1.2 Justificación del Problema

En el estudio realizado por NTT DATA en colaboración con MIT Technology Review, se evidencia que el 80% de las empresas en América Latina adoptaron por la computación en la nube [9]. El 23% de estas empresas presentaron varios retos para seleccionar la infraestructura en la nube que les permita reducir la complejidad operativa y los costos, para enfocarse solo en la lógica de su negocio. Las empresas están optando por servicios como AWS Lambda, Fargate y EC2 para ejecutar sus aplicaciones, pero las empresas eligen manualmente cual sería el servicio adecuado para gestionar el escalado, lo que implica mucho tiempo y un costo operativo adicional.

Por esta razón, el problema de la optimización de costos en la nube es importante en la actualidad, este proyecto aborda la necesidad de automatizar la elección dinámica entre servicios de AWS para garantizar que las aplicaciones estén siempre optimizadas en términos de costo y rendimiento. Ya que, al realizar una gestión manual de qué servicio utilizar en cada momento es un proceso costoso y propenso a errores. Por lo tanto, resolver esto no solo permite a las empresas ahorrar dinero y tiempo, sino que también optimizar el uso de los recursos de AWS, asegurando que paguen únicamente por lo que utilizan.

1.3 Objetivos

1.3.1 Objetivo general

Desarrollar un sistema de autoconfiguración de cómputo en la nube, que de forma dinámica seleccione entre AWS Lambda, AWS Fargate y AWS EC2, basándose en métricas de rendimiento carga y costos, para optimizar el uso de recursos y reducir los costos de las aplicaciones web.

1.3.2 Objetivos específicos

1. Integrar el repositorio de AWS Lambda Web Adapter para ejecutar aplicaciones web en Lambda, Fargate y EC2.

2. Desarrollar scripts que permitan el despliegue automatizado en Lambda, Fargate y EC2 en función de la carga de trabajo y los costos actuales.
3. Desarrollar un algoritmo de aprendizaje por refuerzo que optimice el cambio entre los tres servicios de despliegue.
4. Realizar pruebas de carga para evaluar la efectividad del sistema bajo diferentes escenarios.

1.4 Marco teórico

1.4.1 Computación en la nube

Es un servicio que permite el acceso bajo demanda a recursos informáticos a través de Internet de forma inmediata y flexible, entre ellos se encuentran servicios, base de datos, potencia de procesamiento, aplicaciones, almacenamiento, redes y servidores [10].

Ofrece ahorro de costos, escalabilidad, alto rendimiento, economías de escalado, flexibilidad y mucho más en comparación con la infraestructura local tradicional. Se ha vuelto indispensable en los entornos empresariales, desde los pequeños startups hasta las empresas multinacionales [11].

La computación en la nube tiene tres servicios principales: infraestructura como servicio (IaaS) que permite a las empresas alquilar infraestructuras, plataforma como servicio (PaaS) que proporciona recursos para crear y desarrollar aplicaciones a nivel de usuarios y software como servicio (SaaS) que ofrece aplicaciones de software a través de Internet [12].

1.4.2 Computación sin servidor

Es un modelo de ejecución de computación en la nube de desarrollo de aplicaciones en una infraestructura administrada por el proveedor de servicios en la nube [13]. Este modelo se encarga de mantener, administrar, equilibrar la carga, de los parches de seguridad, de la supervisión y el registro [14]. Debido a eso ya no es necesario comprar y mantener servidores físicos, esto resultaba ser costos ya que los clientes solo utilizaban un porcentaje muy pequeño de los recursos del servidor [15].

Ahora el desarrollador puede codificar en cualquier lenguaje y solo debe centrarse en diseñar y compilar las aplicaciones, ya que el proveedor en la nube se encarga de gestionar la infraestructura ampliándola y reduciéndola según las necesidades del cliente. Se paga solo por la ejecución, desde que se realiza una solicitud hasta cuando terminar [16].

Muchos de los principales proveedores de servicios en la nube ofrecen plataformas sin servidor, entre ellos se encuentran AWS con AWS Lambda lanzado en 2014, Microsoft Azure con Azure Functions lanzado en 2016, Google Cloud con Google Cloud Functions lanzado en el 2017 y IBM Cloud con IBM Cloud Code Engine lazado en el 2021 [17].

1.4.3 AWS Lambda

Permite ejecutar código sin necesidad de administrar servidores de prácticamente cualquier aplicación, solo se debe cargar el código como un archivo .zip o una imagen de contenedor [18]. Se puede utilizar cualquier lenguaje que el desarrollador desee Node.js, Python, Go o Java [19].

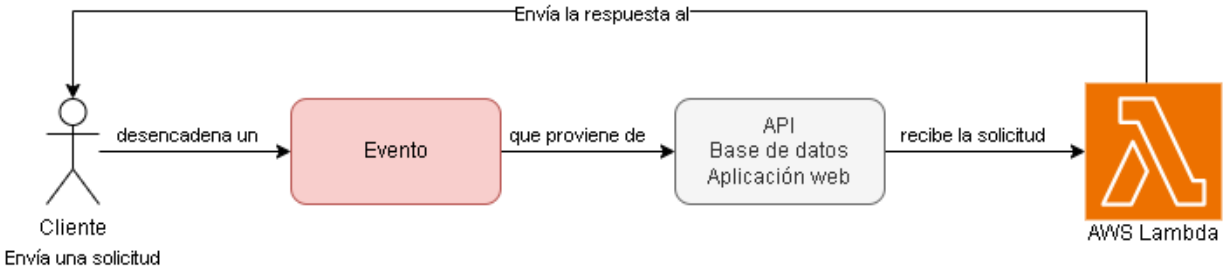


Figura 1 Diagrama de funcionamiento de AWS Lambda

1.4.4 AWS Fargate

Permite ejecutar aplicaciones sin administrar la infraestructura, se paga solo por los recursos utilizados. Es ideal para aplicaciones donde el tiempo de ejecución es prolongado o se necesita un mayor control sobre los recursos asignados [20].

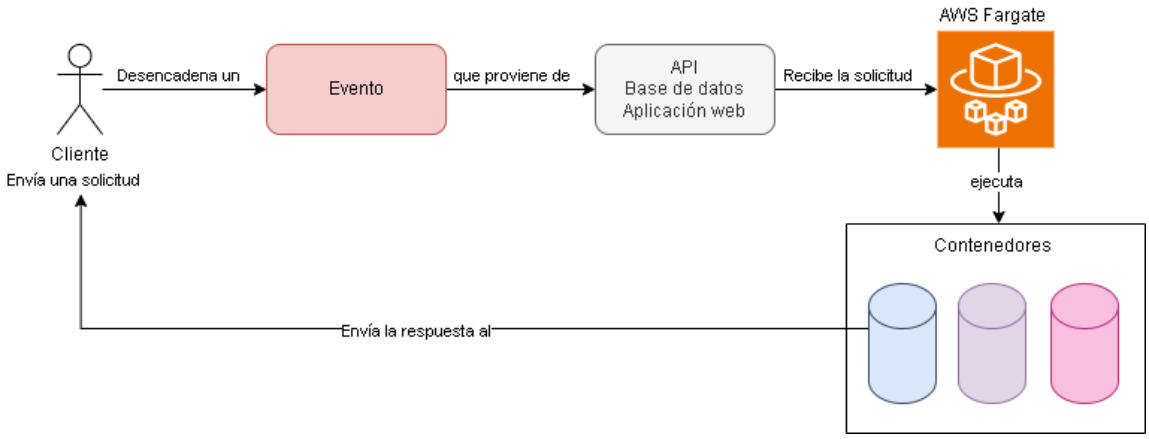


Figura 2 Diagrama de funcionamiento de AWS Fargate

1.4.5 AWS EC2

Es un servicio que proporciona capacidad de cómputo en la nube, permite ejecutar cualquier aplicación y reduce los costos de hardware para que pueda desarrollar e implementar aplicaciones con mayor rapidez [21]. Permite escalar verticalmente para gestionar tareas que necesitan gran cantidad de recursos como los picos de tráfico en un sitio web. Ofrece varias instancias con plantillas preconfiguradas con diferentes configuraciones de memoria, CPU y capacidad de red [22].

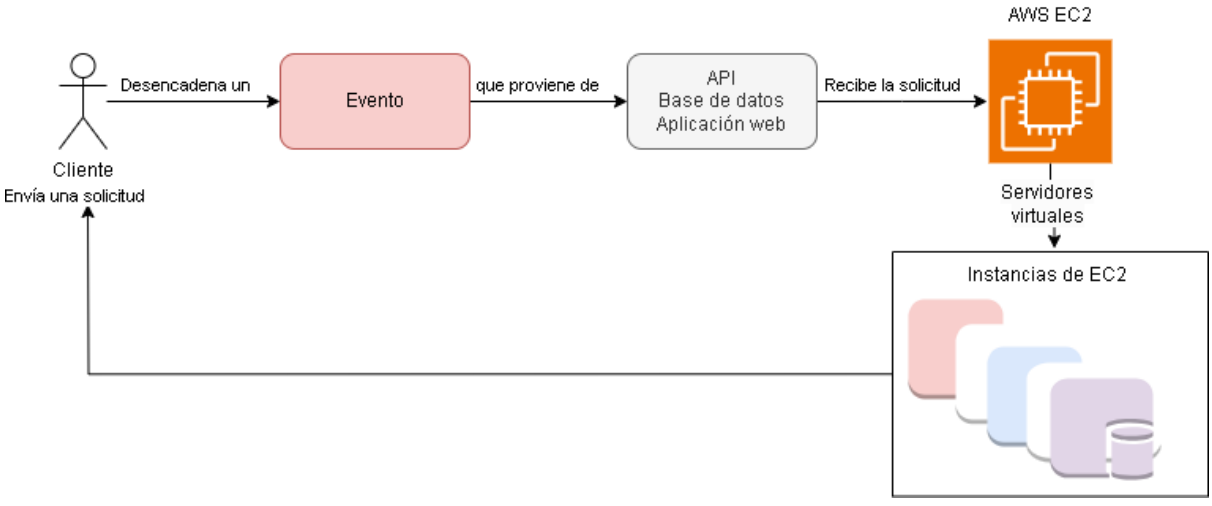


Figura 3 Diagrama de funcionamiento de AWS EC2

Tabla 1 Servicios de AWS

Servicio	Funcionalidad	Métricas	Ejemplo
AWS Lambda	Ejecutar funciones sin necesidad de gestionar servidores. Cargas de trabajo baja con invocaciones de corta duración.	Se basa en el número de solicitudes o eventos.	Un microservicio que se activa cada vez que se sube un archivo a un bucket de S3.
AWS Fargate	Desplegar aplicaciones web y APIs directamente desde contenedores, sin preocuparse por la infraestructura.	Se basa en el tráfico en el web recibido.	Una API que ofrece servicios en línea y ajusta el número de instancias en función de la demanda de los usuarios.
AWS EC2	Para aplicaciones que requieren más control en el entorno. Cargas de trabajo más sostenida o para tareas que requieren más CPU/memoria	Se basado en el uso de recursos como CPU o memoria.	Un sistema de procesamiento en paralelo de grandes cantidades de datos usando múltiples contenedores.

1.4.6 Reglas heurísticas

Las reglas heurísticas son un conjunto de principios o pautas basadas en experiencia previa, conocimientos de expertos y razonamiento práctico para tomar decisiones rápidas y efectivas [23].

El auto escalamiento con reglas heurísticas es un enfoque diseñado para gestionar de manera eficiente los recursos computacionales y garantizar un rendimiento óptimo de un sistema. Se utiliza para decidir dinámicamente cuándo escalar una aplicación entre diferentes servicios de AWS, basándose en métricas específicas del sistema. Estas reglas:

- Monitorean métricas de rendimiento en tiempo real para identificar si la carga está aumentando o disminuyendo.
- Si las métricas superan ciertos umbrales, el sistema escala hacia un servicio más robusto.
- Si las métricas están por debajo de los umbrales definidos, el sistema reduce los recursos y vuelve a servicios más ligeros.

1.4.7 Aprendizaje por Refuerzo

El aprendizaje por refuerzo (Reinforcement Learning, RL) es una rama de la inteligencia artificial que se centra en entrenar agentes para tomar decisiones secuenciales. Imita el proceso de aprendizaje de los humanos, ya que para lograr sus objetivos utilizan el ensayo y el error [24]. Un agente interactúa con un entorno y aprende a tomar decisiones basadas en las recompensas obtenidas de sus acciones. Aprenden de los comentarios de cada acción y descubren por sí mismos las mejores rutas de procesamiento para lograr los resultados finales.

Es útil en situaciones donde las decisiones deben adaptarse dinámicamente a cambios en el entorno, permite optimizar la asignación de recursos al evaluar continuamente el estado del sistema y aprender estrategias óptimas para minimizar costos y maximizar los resultados [25]. Tiene dos tipos de aprendizajes, en línea que es donde el agente recopila datos directamente de la interacción del entorno y sin conexión que es cuando un agente no tiene acceso directo al entorno puede aprender mediante los datos registrados de ese entorno [26].

Componentes:

1. **Agente:** Es la entidad o algoritmo que toma las decisiones.
2. **Entorno:** El sistema o espacio donde el agente opera.
3. **Recompensa:** La retroalimentación que el agente recibe tras cada acción, puede ser un valor negativo o positivo por llevar a cabo una acción.
4. **Política:** La estrategia que el agente aprende para maximizar recompensas acumuladas [24].

Relación con el proyecto

En este proyecto, el aprendizaje por refuerzo se aplicará para optimizar la selección dinámica entres los servicios de AWS Lambda, Fargate y EC2. El sistema monitorizará métricas como invocaciones, duración, uso de CPU y memoria, y aprenderá a realizar transiciones óptimas basadas en el costo y el rendimiento. Este enfoque garantiza:

- **Escalabilidad eficiente:** Adaptándose a cambios en la carga de trabajo.

- **Minimización de costos:** Seleccionando la opción más rentable en cada momento.
- **Automatización inteligente:** Reduciendo la necesidad de intervención humana.

1.4.8 Estudios Relacionados

1. **Simulaciones para planificación de presupuestos en la nube:** este trabajo realizado por Boza et al. [8] desarrolla una herramienta en modelos basados en la teoría de las colas para evaluar y comparar costos y rendimientos de diferentes modelos de precios en la nube, ayudando a identificar configuraciones óptimas que minimicen costos y cumplan objetivos de nivel de servicio. Propone infraestructuras para realizar simulaciones avanzadas con patrones de carga dinámica, especialmente útiles para decisiones de presupuesto en empresas con restricciones económicas. Su herramienta predice costos y rendimientos para diferentes escenarios, ayudando a pequeñas y medianas empresas a optimizar sus presupuestos
2. **Elasticidad en computación serverless**: en este trabajo realizado por Qiu et al. [27] se analiza cómo mejorar la elasticidad de arquitecturas serverless mediante simulaciones, técnicas de optimización de recursos y experimentos en sistemas reales. Además, proponen la infraestructura llamada FIRM (Function-level Resource Management), que optimiza el uso de recursos en función de la demanda en tiempo real, los hallazgos revelan estrategias efectivas para minimizar costos sin comprometer el rendimiento.
3. **Escalado de recursos dinámico:** en este trabajo realizado por Qiu y Haoran [28] introducen una arquitectura llamada AWARE (Adaptive Workload-Aware Resource Elasticity) basada en aprendizaje automático, que predice y ajusta escalas de recursos en tiempo real, mejorando la capacidad de respuesta del sistema ante cambios inesperados en la demanda. Sus experimentos demostraron mejoras significativas en la adaptabilidad y eficiencia del sistema.

Capítulo 2

2 Metodología

Para abordar el problema de reducir costos en sistemas de cómputo en la nube a través de un sistema de autoconfiguración dinámica, se exploraron varias alternativas de despliegue y administración de servicios en la nube, centradas en la infraestructura de AWS. Las alternativas evaluadas incluyeron el despliegue de servicios en AWS Lambda, AWS Fargate y Amazon EC2.

Se utilizó como base de desarrollo el repositorio AWS Lambda Web Adapter de AWS Labs, el cual permite adaptar aplicaciones webs, ya que permite que la misma imagen Docker de la aplicación se ejecute en diferentes servicios como AWS Lambda, Amazon EC2 y AWS Fargate.

Además, se utilizaron herramientas adicionales como AWS SAM para configurar y desplegar aplicaciones en AWS Lambda, AWS Copilot para desplegar y gestionar aplicaciones basadas en contenedores en Amazon ECS con Fargate y Amazon Elastic Beanstalk para desplegar y gestionar aplicaciones en EC2.

Luego de varias pruebas se decidió realizar los despliegues con Python con el SDK de boto3 para asegurar un control total y una mayor flexibilidad en la configuración y gestión de los servicios de AWS. Debido a que boto3 permite personalizar cada aspecto del despliegue y proporciona soporte para manejar los tres servicios desde un único lenguaje de programación.

Se desarrolló un script que actúa como sistema de monitoreo y cambio dinámico. Este script revisa los umbrales de las métricas y selecciona el servicio más adecuado, asegurando así la eficiencia y el costo óptimo, basándose en reglas heurísticas predefinidas según cada servicio.

Para mejorar la toma de decisiones en el sistema de autoconfiguración, se incorporó Reinforcement Learning (RL) en el módulo de cambio dinámico. Esto permite que el sistema aprenda de manera autónoma qué configuraciones o entornos son los óptimos según las métricas operativas seleccionadas, como número de invocaciones, ejecuciones concurrentes, costo por segundo, duración, uso de CPU y uso de memoria.

Para validar la capacidad del sistema de cambio dinámico, se diseñaron experimentos de sobrecarga en los cuales se simuló un aumento de la demanda, generando escenarios de carga intensiva. Los primeros escenarios de pruebas fueron simulados con Apache Bench y Postman, mientras que, los resultados finales de las pruebas registradas en este trabajo fueron simuladas con k6.

Y finalmente se diseñó un dashboard en Grafana para supervisar en tiempo real las métricas y el servicio en el que se ejecutan las aplicaciones.

2.1 Especificaciones técnicas

- **Lenguaje de programación**: Python con boto3 para la interacción de AWS.
- **Adaptador**: aws-lambda-web-adapter
- **Docker:** para empaquetar aplicaciones.
- **AWS SAM:** facilita el despliegue y manejo de aplicaciones sin servidor.
- **AWS Copilot:** Facilita el despliegue de aplicaciones en Amazon ECS con Fargate.
- **Elastic Beanstalk:** Facilita el despliegue y gestión de aplicaciones en EC2.
- **Métricas de monitoreo**: invocaciones, ejecuciones concurrentes, costo por segundo, duración, uso de CPU y uso de memoria.
- **Dashboard**: Grafana
- **Sobrecarga**: postman, Apache Bench y K6.

Servicios de AWS:

- **Lambda:** Despliegue y manejo de funciones serverless.
- **Fargate:** Contenedores para tareas ligeras y escalables.
- **EC2:** Máquinas virtuales con configuraciones para tareas más intensivas.
- **ECR:** Almacenamiento y gestión de imágenes Docker.
- **IAM:** Configuración de roles y permisos para seguridad.
- **CloudWatch:** Monitoreo de métricas de rendimiento y alarmas.
- **API Gateway:** Exposición de endpoints para las funciones Lambda.
- **ALB:** enmascaramiento de endponits

2.2 Diseño y proceso de implementación

Este diseño se desarrolló en varias fases, comenzando con la implementación y prueba individual de cada servicio de AWS, y finalizando con la integración de todos los componentes en un sistema. Esta metodología se describe a continuación:

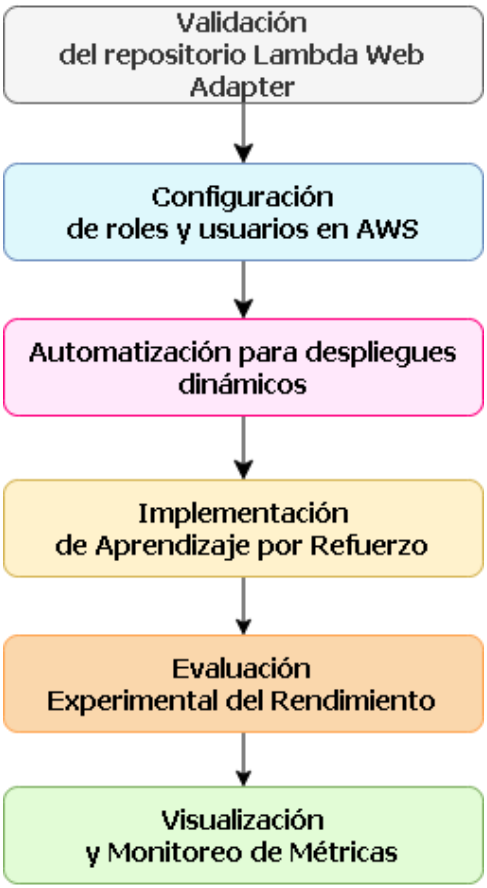


Figura 4 Proceso de desarrollo de la solución

2.2.1 Validación del repositorio Lambda Web Adapter

La primera fase consistió en validar el funcionamiento del repositorio Lambda Web Adapter, que juega un papel fundamental en el proyecto. Este repositorio permite desplegar aplicaciones web, como frameworks basados en HTTP, en un entorno serverless, lo que facilita su integración con AWS Lambda y Fargate. Este enfoque simplifica el proceso de modernización de aplicaciones al adaptarlas para entornos cloud nativos sin necesidad de reescribirlas completamente.

El objetivo principal fue evaluar cómo este repositorio se podía utilizar como puente entre aplicaciones tradicionales y los servicios serverless de AWS, probando su funcionalidad en los principales entornos de despliegue: AWS Lambda, Fargate y EC2. A continuación, se describen las actividades realizadas en cada servicio:

Despliegue en AWS Lambda con SAM

Se utilizó AWS SAM para desplegar aplicaciones basadas en funciones Lambda. La infraestructura fue definida a través de una plantilla template.yaml, especificando las funciones Lambda, sus permisos y las configuraciones necesarias.

Despliegue en ECS con Fargate utilizando Copilot

Se realizó el despliegue de las mismas aplicaciones en contenedores con Docker y AWS Copilot, una herramienta que simplifica la creación y administración de aplicaciones contenedorizadas en AWS. Para este despliegue, se creó una imagen Docker que definía la infraestructura requerida y se almacenó en Amazon ECR.

Despliegue en instancias EC2 con Elastic Beanstalk

El despliegue en EC2 proporcionó un control más avanzado sobre el entorno operativo de las aplicaciones. Se utilizó Elastic Beanstalk para gestionar el proceso de despliegue, definiendo el tipo de aplicación, el entorno operativo y la región.

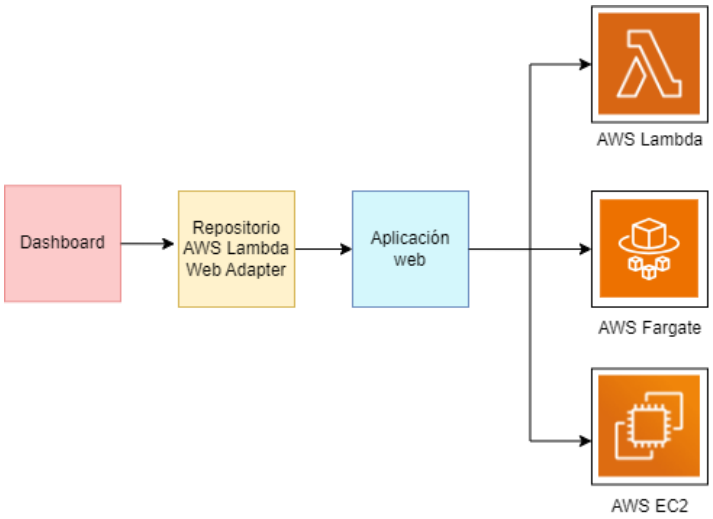


Figura 5 Arquitectura de despliegue en AWS con Lambda Web Adapter

2.2.2 Configuración de roles y usuarios en AWS

La segunda fase del proyecto se centró en configurar y crear roles y políticas en AWS IAM, para garantizar la seguridad y los permisos necesarios para el correcto funcionamiento de los servicios desplegados. Estas configuraciones permitieron la integración fluida entre servicios como Lambda, ECS y EC2 con otros componentes de AWS, como ECR y CloudWatch. A continuación, se describen los Roles configurados:

DeployLambdaRole (Rol para AWS Lambda):

Este rol fue creado para permitir que las funciones Lambda accedan a los servicios necesarios como CloudWatch Logs y ECR.

Políticas adjuntas:

- **AWSLambdaExecute:** Proporciona permisos básicos para que las funciones Lambda puedan escribir logs en CloudWatch.
- **AmazonECRContainerRegistryFullAccess:** Permite a Lambda acceder y extraer imágenes de contenedores almacenadas en ECR.
- **CloudWatchLogsFullAccess:** Para supervisar métricas y logs generados por la función.

ECRAccessRole (Rol para EC2):

Este rol fue asignado a las instancias EC2 para otorgarles acceso de lectura al repositorio de imágenes Docker almacenadas en ECR.

Políticas adjuntas:

- **AmazonEC2ContainerRegistryReadOnly:** Permite acceso de solo lectura al repositorio de imágenes.
- **AmazonEC2ContainerServiceRole**
- **AmazonSSMManagedInstanceCore:** Habilita la administración de las instancias EC2 a través de AWS Systems Manager.

ecsTaskExecutionRole y ecsTaskRole (Roles para ECS Fargate):

- **ecsTaskExecutionRole**: Este rol se encargó de proporcionar permisos para ejecutar tareas en Fargate.
- **ecsTaskRole**: Este rol se diseñó para permitir que las tareas de ECS accedan a otros recursos de AWS necesarios para su ejecución.

2.2.3 Automatización adaptativa para despliegues dinámicos

Esta fase se centró en desarrollar un sistema de automatización dinámica que permite seleccionar y realizar despliegues en AWS según las condiciones del entorno y las necesidades operativas. Este enfoque utiliza reglas heurísticas predefinidas para decidir entre los servicios AWS Lambda, ECS con Fargate y EC2, maximizando la eficiencia y adaptabilidad del sistema.

En primer lugar, se desarrollaron y probaron scripts independientes para realizar despliegues básicos en AWS Lambda, Amazon Fargate y Amazon EC2 creados utilizando el SDK de AWS Boto3. Se probaron los scripts con aplicaciones simples para validar la funcionalidad básica de cada servicio y su integración con AWS. Posteriormente, se desarrollaron e implementaron aplicaciones con un CRUD básico, diseñadas específicamente para este proyecto. Estas aplicaciones fueron utilizadas para evaluar cómo los tres servicios gestionan cargas de trabajo intermedias y operaciones más intensivas.

2.2.3.1 Scripts y automatización

- **Automatización de despliegues:**
  - **deploy_to_lambda.py**: Configura y despliega funciones Lambda.
  - **deploy_to_fargate.py**: Configura clústeres y tareas en Fargate.
  - **deploy_to_ec2.py**: Configura y lanza instancias EC2 con imágenes Docker.
  - **build_and_push_to_ecr.py:** Construye y sube imágenes Docker al repositorio ECR.
- **Cambio dinámico:**
  - **monitor_and_autoscale.py**: toma decisiones sobre el servicio más adecuado para el despliegue (Lambda, Fargate o EC2).

2.2.3.2 Flujo de trabajo del sistema

El flujo de trabajo general incluye los siguientes pasos:

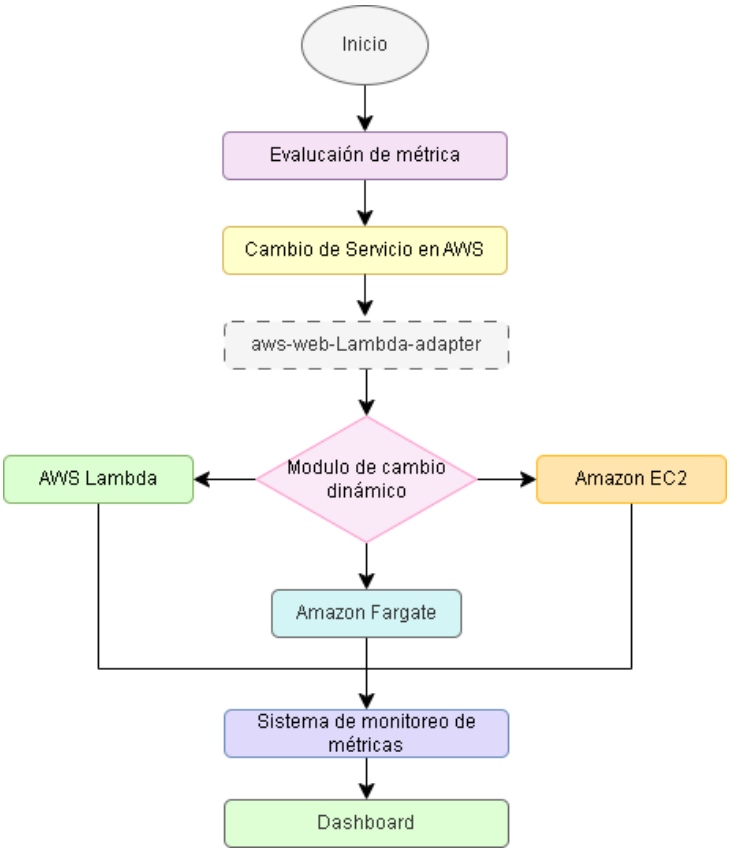


Figura 6 Flujo de trabajo del sistema

1. Supervisar las métricas de CloudWatch para la función Lambda en ejecución.
2. Detectar si las métricas superan el umbral predefinido.
3. Activar el script correspondiente para desplegar la aplicación en el siguiente servicio escalable.
4. Validar el despliegue y ajustar los recursos según sea necesario.

El siguiente diagrama muestra el flujo de decisiones para seleccionar el servicio de cómputo adecuado en AWS según la carga de trabajo. Se comienza evaluando si la carga es baja, lo que mantiene el servicio desplegado en AWS Lambda. Si la carga es moderada o alta, cambia el servicio a AWS Fargate. En caso de que la carga sea alta y constante, se transfiere a AWS EC2. Este proceso permite optimizar el uso de recursos y costos con base en la demanda.

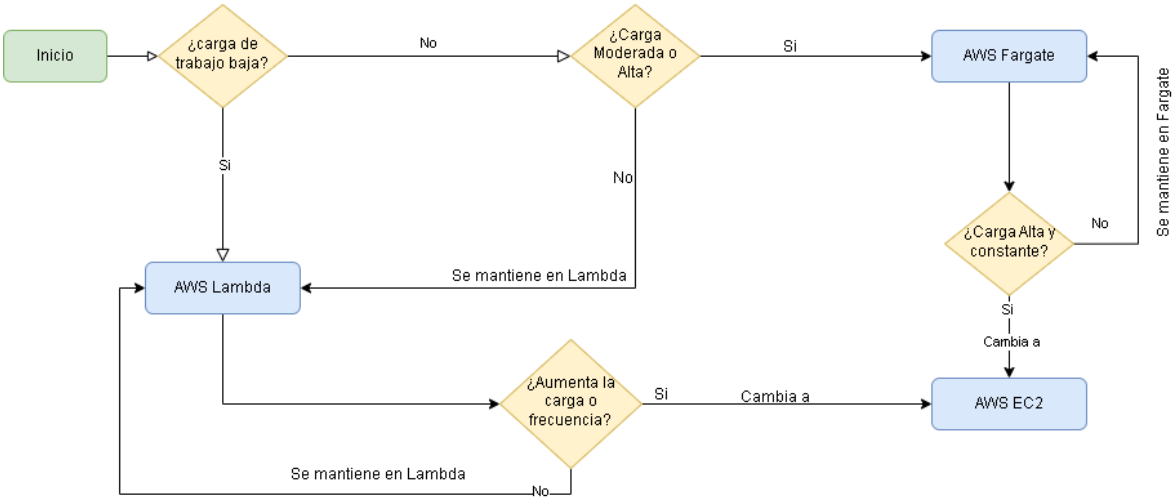


Figura 7 Diagrama de decisión para seleccionar de manera dinámica los servicios de AWS

2.2.3.3 Métrica de monitoreo

- **Número de invocaciones (Invocations):** Cuántas solicitudes están llegando por segundo, es clave en Lambda.

  Umbral: 3000 invocaciones en un período de 5 minutos. Este valor indica que Lambda está manejando una alta cantidad de solicitudes, lo que podría saturar el servicio.

- **Duración (Duration):** Tiempo promedio que tarda en responder el servicio, es clave en Lambda.

  Umbral: 1000 ms por invocación promedio. Si la duración promedio de las funciones supera este valor, es un indicador de que Lambda está alcanzando sus límites de rendimiento.

- **Ejecuciones concurrentes (ConcurrentExecutions):**

  Umbral: 1000 ejecuciones concurrentes. Si las ejecuciones concurrentes alcanzan este límite, podría haber retrasos en las invocaciones debido a la limitación de concurrencia de Lambda.

- **Uso de CPU (CPUUtilization):** Porcentaje de utilización de la CPU en el servicio actual, es clave en Fargate y EC2.

Umbral: 75% de utilización de CPU. Si las tareas superan este nivel de utilización de CPU, podría haber problemas de rendimiento en las aplicaciones y se debe cambiar a EC2.

Umbral: 85% de utilización de CPU. Si las instancias alcanzan este nivel, es posible que sea necesario escalar horizontalmente en EC2.

- **Uso de memoria (MemoryUtilization):** Porcentaje de utilización de memoria, es clave en Fargate.

  Umbral: 80% de utilización de memoria. Si las tareas utilizan más del 80% de la memoria asignada, es un indicador de que debe escalar a EC2.
- **Costo por segundo (Cost):** El costo asociado al servicio actualmente seleccionado.

2.2.3.4 Lógica de decisión para el cambio dinámico

De Lambda a Fargate

Invocations > 3000 o Duration > 1000 ms o ConcurrentExecutions > 1000.

Detener la función Lambda, activar el script para desplegar la aplicación en Fargate.

De Fargate a EC2

MemoryUtilization > 80% o CPUUtilization > 75%

Detener las tareas en Fargate. Activar el script para desplegar la aplicación en EC2.

De EC2 a Fargate

CPUUtilization < 50% en las instancias EC2 durante un período continuo de 10 minutos.

Detener las instancias EC2. Activar el script para desplegar la aplicación en Fargate.

De Fargate a Lambda

CPUUtilization < 40% o MemoryUtilization < 50% o Invocations < 1500 o Duration < 500 ms

Detener las tareas en Fargate. Activar el script para desplegar la aplicación en Lambda.

De EC2 a Lambda

CPUUtilization < 40% en las instancias EC2.

Detener las instancias EC2. Activar el script para desplegar la aplicación en Lambda.

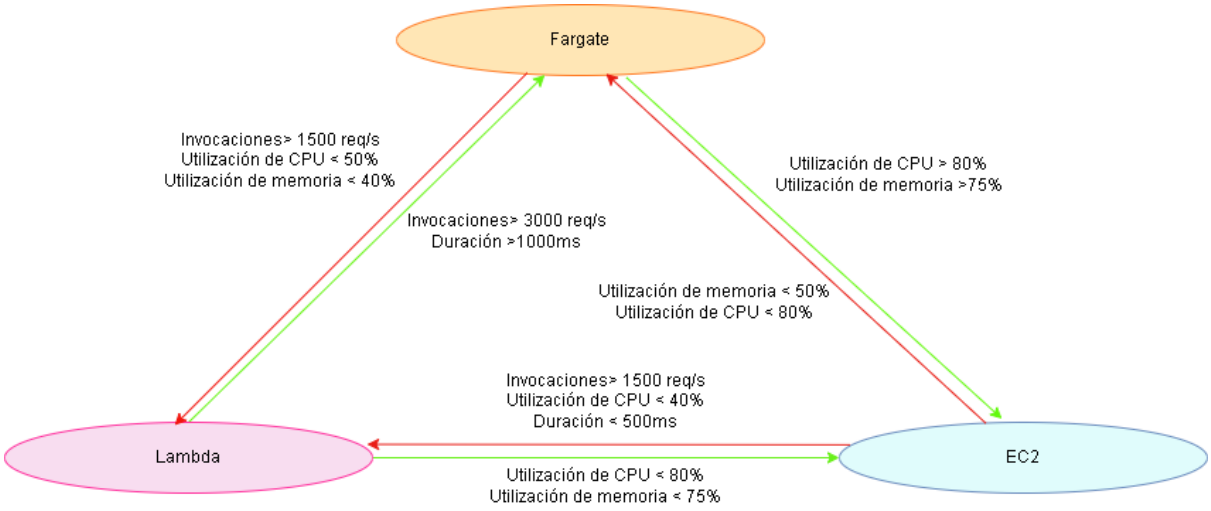


Figura 8 Gráfico de lógica de decisión para cambio de servicio

2.2.4 Implementación de Aprendizaje por Refuerzo

El propósito central de esta estapa fue desarrollar y entrenar un agente basado en Aprendizaje por Refuerzo (RL) para automatizar el proceso de escalado entre servicios de AWS (Lambda, Fargate y EC2), con la finalidad de optimizar tanto el rendimiento como los costos operativos. La elección de utilizar un modelo basado en RL se fundamentó en su capacidad para aprender estrategias de toma de decisiones adaptativas en entonrnos con condiciones de carga variables, superando así las limitaciones asociadas a configuraciones manuales o estáticas.

Ventajas de utilizar RL

- Adaptabilidad: El agente de RL tiene la capacidad de adaptarse a condiciones de carga variables e impredecibles, aprendiendo a actuar eficientement ante cambios repentinos en la demanda.
- Optimización de Recompensas: Al definir una función de recompensa que considera aspectos como la latencia, el uso de recursos y los costos, el agente identifica estrategias que maximizan el rendimiento general del sistema.
- Escalabilidad y Eficiencia: Con el tiempo, el modelo afina su habilidad para gestionar los recursos de manera óptica, reduciendo la necesidad de intervención manual y mejorando la experiencia del usuario final.

2.2.5 Diseño y Desarrollo del Entorno de Simulación

Debido a la falta de datos históricos, se prefirió desarrollar un entorno simulado utilizando la librería Gym de OpenAI. Este entorno fue creado para replicar las condiciones de un sistema de despliegue en AWS, lo que permitió simular las variaciones en métricas clave y valorar el desempeño del agente en diferentes escenarios.

2.2.5.1 Definición del Espacio de Estados

El estado del entorno se modeló mediante un vector de cinco valores que reflejan métricas clave:

- **Duration:** Representa la latencia o tiempo de ejecución acumulado del servicio.
- **Invocations:** Número de invocaciones recibidas.
- **ConcurrentExecutions:** Número de ejecuciones concurrentes.
- **CPUUtilization:** Porcentaje de uso de CPU.
- **MemoryUtilization:** Porcentaje de uso de memoria.

2.2.5.2 Definición del Espacio de Acciones

El agente es capaz de seleccionar entre tres acciones discretas, cada una asociada a un servicio de despliegue específico:

- **Acción 0: AWS Lambda**

  Se simula un escalado para cargas de trabajo breves, con aumentos moderados en la latencia e invocaciones.
- **Acción 1: AWS Fargate**

  Se dirige a cargas que requieren recursos de CPU y memoria de manera controlada.
- **Acción 2: AWS EC2**

  Diseñada para cargas más pesadas, donde se incrementa de forma notable el uso de CPU.

Esta configuración permite que el agente evalúe y elija el servicio más adecuado según las condiciones del sistema en cada instante.

2.2.5.3 Función de Transición

La función step del entorno simula cómo evolucionan las métricas según la acción seleccionada:

- **Para Lambda (Acción 0):**

  Se incrementa la latencia (Duration) y el número de invocaciones (Invocations) en rangos moderados.

- **Para Fargate (Acción 1):**

  Se incrementan de forma controlada la utilización de CPU y memoria.

- **Para EC2 (Acción 2):**

  Se simula un incremento considerable en el uso de CPU.

Además, se definen condiciones de finalización del episodio cuando las métricas superan ciertos umbrales críticos, lo que ayuda al agente aprender a evitar estados no deseados.

2.2.5.4 Función de Recompensa

La función de recompensa se diseñó con los siguientes objetivos:

- **Penalizar la alta latencia:** Se descuenta un valor proporcional al exceso sobre el umbral establecido (por ejemplo, 500 ms).
- **Recompensar la disponibilidad:** Se otorga un bono cuando el número de invocaciones es mayor a cero.
- **Optimizar el uso de recursos:**
  - Se premia cuando la utilización de CPU y memoria se mantiene en un rango óptimo (por ejemplo, alrededor del 65% para CPU y 60% para memoria).
  - Se penaliza cuando estos valores son demasiado bajos (indicando ineficiencia) o demasiado altos (riesgo de saturación).
- **Incorporar costos:**

La función incluye una penalización basada en el costo estimado de cada acción, incentivando decisiones que optimicen tanto el desempeño como el gasto operativo. Este

diseño permite que el agente aprenda a balancear el desempeño del sistema con los costos asociados, guiándose por indicadores que reflejan la salud y eficiencia del despliegue.

2.2.6 Arquitectura y Configuración del Modelo de RL

Se eligió el algoritmo Proximal Policy Optimization (PPO), destacado por su robustez y estabilidad en entornos complejos. La implementación se llevó a cabo utilizando la librería StableBaselines3, basada en PyTorch para la gestión de la red neuronal.

2.2.6.1 Configuración del Modelo

- **Política Utilizada:**

  Se empleó la MlpPolicy, que consiste en una red neuronal de perceptrón multicapa.

- **Arquitectura de la Red:**

  Se diseñó una red con dos capas ocultas de 256 neuronas cada una, tanto para la política como para la estimación del valor, lo que facilita la captura de relaciones no lineales entre las métricas del entorno.

- **Hiperparámetros Clave:**
  - Learning rate: 0.00005
  - n_steps: 4096
  - Batch size: 256
  - Gamma (factor de descuento): 0.99
  - Ent coef: 0.1
  - Clip range: 0.2
  - Número de épocas: 20

  Estos parámetros se definieron y ajustaron en base a pruebas preliminares y al comportamiento observado en el entorno simulado.

2.2.6.2 Integración con Herramientas de Desarrollo

El desarrollo y entrenamiento se basaron en las siguientes herramientas:

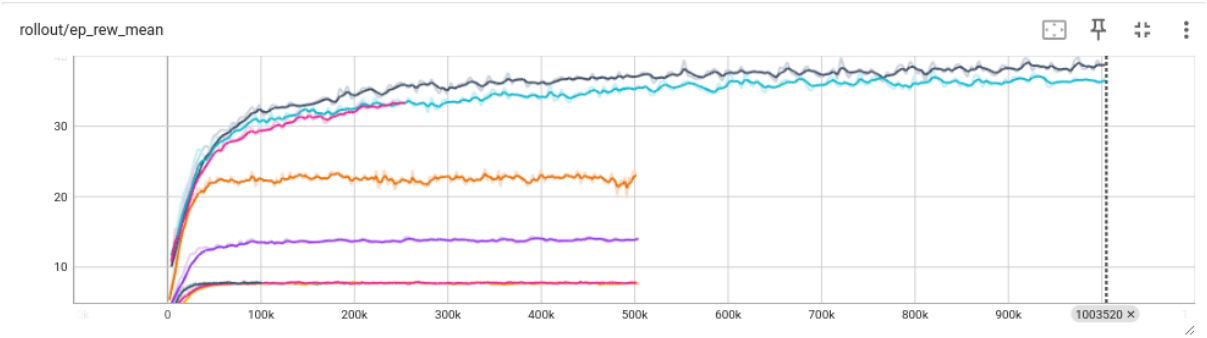
- **Gym:** Para diseñar y gestionar el entorno simulado.

- **Stable-Baselines3 y PyTorch:** Para implementar y entrenar el modelo PPO.
- **Tensorboard:** Para monitorear en tiempo real el progreso del entrenamiento y analizar la evolución de las recompensas.

2.2.7 Proceso de Entrenamiento y Validación

2.2.7.1 Entrenamiento en Entorno Simulado

El agente se entrenó durante 1,000,000 de timesteps en el entorno simulado, lo que permitió explorar múltiples estrategias de escalado en condiciones de carga variables. A lo largo de este proceso, se registraron métricas y se monitoreó el desempeño mediante Tensorboard, facilitando ajustes en tiempo real.



Run	Smoothed	Value	Step	Relative
PPO_1	7,8204	7,82	100.352	2.154 min
PPO_10	33,3276	33,3605	253.952	5.729 min
PPO_2	7,8586	7,8	200.704	3.83 min
PPO_3	7,8291	7,81	501.760	7.524 min
PPO_4	7,766	7,77	503.808	16.63 min
PPO_5	14,0567	14,274	503.808	14.83 min
PPO_7	22,9896	22,9971	501.760	18.88 min
PPO_8	38,7058	38,7939	1.003.520	24.41 min
PPO_9	36,524	36,8953	1.003.520	25.1 min

Figura 9 Gráfico de recompensa promedio por episodio durante el entrenamiento

2.2.8 Consideraciones, Limitaciones y Futuras Mejoras

Aunque la simulación permitió desarrollar y validar el modelo de RL en un entorno controlado, se identificaron ciertas limitaciones y aspectos a tener en cuenta:

- **Fidelidad de la Simulación:**

  Los parámetros y la dinámica del entorno se establecieron de manera heurística. En un entorno de producción real, las métricas y el comportamiento de los servicios podrían variar, lo que requerirá ajustes en la función de transición y recompensa.

- **Riesgo en Entornos Críticos:**

  La automatización del escalado mediante RL conlleva el riesgo de que decisiones incorrectas afecten la disponibilidad o el rendimiento del servicio. Es fundamental implementar mecanismos de supervisión y límites de seguridad al trasladar el modelo a producción.

- **Optimización Continua:**

  Se prevé la realización de iteraciones adicionales en el modelo, tanto en la afinación de hiperparámetros como en la ampliación del entorno de simulación, para incorporar nuevas métricas o escenarios de carga.

2.2.9 Evaluación Experimental del Rendimiento

Primero se realizaron pruebas de sobrecarga pequeñas utilizando herramientas como Apache Bench y Postman. Para asegurar que el sistema pudiera manejar grandes volúmenes de tráfico sin fallar, se realizaron pruebas de sobrecarga utilizando herramientas como K6. Se utilizó para simular miles de solicitudes simultáneas y evaluar el rendimiento de las aplicaciones desplegadas en Fargate, Lambda y EC2. Las pruebas ayudaron a identificar cuellos de botella y a ajustar las configuraciones de los contenedores y funciones sin servidor.

2.2.10 Visualización y Monitoreo de Métricas

Se diseñó e implementó un dashboard en Grafana para permitir la supervisión en tiempo real de las métricas clave y el servicio en el que se están ejecutando las aplicaciones. Este dashboard desempeña un papel crucial al proporcionar visibilidad sobre el rendimiento y el estado del sistema, facilitando la toma de decisiones informadas.

Grafana se configuró para conectarse con AWS CloudWatch, lo que permitió importar y visualizar métricas en tiempo real de los tres servicios Lambda, Fargate y EC2. Se muestran gráficos de uso de CPU, utilización de memoria, invocaciones, ejecuciones concurrentes y

duración. Esto permite a los usuarios supervisar el rendimiento de las aplicaciones y detectar problemas potenciales.

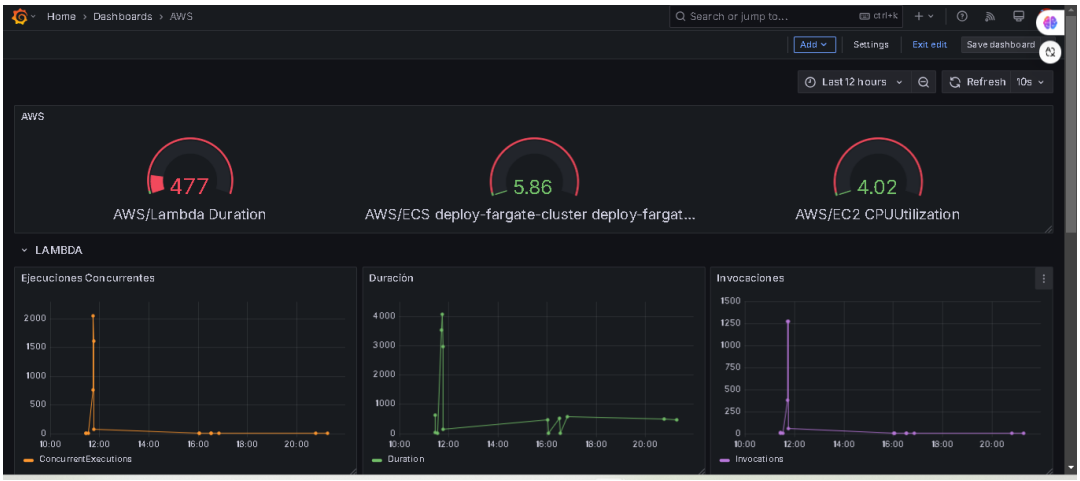


Figura 10 Dashboard de visualización de Lambda

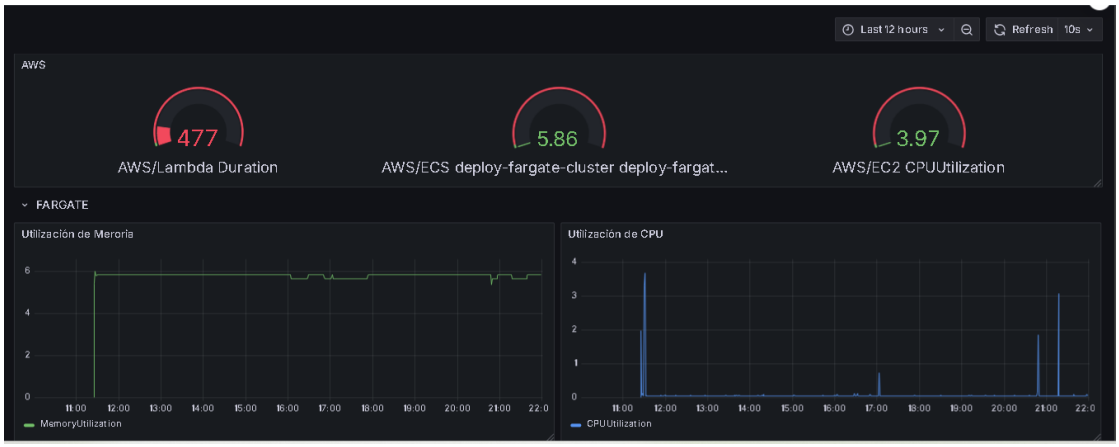


Figura 11Dashboard de visualización de Fargate

Figura 12 Dashboard de visualización de EC2

Capítulo 3

3 Resultados y análisis

Este capítulo presenta los resultados obtenidos tras realizar pruebas con cuatro proyectos de diferente complejidad. Cada proyecto fue desplegado y evaluado en tres escenarios diferentes:

1. Despliegues de servicios independientes (Lambda, Fargate y EC2).
2. Despliegue con script de cambio dinámico.
3. Despliegue con script de cambio dinámico que incluye aprendizaje por refuerzo.

El objetivo principal es analizar cómo cada escenario de despliegue afecta el rendimiento y la eficiencia en diferentes condiciones de carga. Se compararon métricas clave como invocaciones, uso de CPU, uso de memoria, duración de las solicitudes y costos operativos.

3.1 Escenarios de despliegues

1. **Escenario 1: Despliegues de servicios independientes (Lambda, Fargate y EC2**)
   Las aplicaciones fueron configuradas y desplegadas con configuraciones fijas en cada uno de los tres servicios Lambda, Fargate y EC2.
2. **Escenario 2: Despliegue con script de cambio definido con reglas heurísticas**
   Permite cambiar de servicio de forma automática en función de la carga observada y los umbrales definidos para cada servicio.
3. **Escenario 3: Cambio dinámico con Reinforcement learning**
   Los despliegues se realizan con el modelo de aprendizaje por refuerzo para optimizar el escalado y la distribución de recursos en tiempo real.

3.2 Métricas Evaluadas

- **Invocaciones**: Número total de solicitudes procesadas.
- **Uso de CPU (%)**: Recursos consumidos por las aplicaciones.
- **Uso de memoria (MB)**: Cantidad de memoria utilizada.
- **Duración (ms)**: Tiempo promedio necesario para procesar una solicitud.
- **Costo (USD)**: Costo asociado al uso de recursos en cada escenario.

3.3 Proyectos Evaluados

Se evaluaron cuatro aplicaciones para representar diferentes niveles de complejidad y requisitos:

1. **Aplicación 1:**
   - **Descripción:** Una aplicación sencilla que retorna "Hello World".
   - **Complejidad:** Muy baja.
   - **Motivo de la selección:** se utilizó como base para medir los costos mínimos y tiempo de los despliegues en cada servicio.
2. **Aplicación 2:**
   - **Descripción:** Un CRUD básico para realizar operaciones de creación, lectura, actualización y eliminación.
   - **Complejidad:** Baja, diseñada para evaluar escenarios típicos de interacción con bases de datos.
   - **Motivo de la selección:** esta aplicación permite medir el impacto de una carga constante y evaluar el rendimiento al gestionar operaciones de lectura y escritura.
   - **Adaptaciones realizadas:**
     - Se creó un dockerfile con AWS Lambda Web Adapter para permitir el manejo solicitudes HTTP.
3. **Aplicación 3:**
   - **Descripción:** Benchmark llamado "Dynamic HTML" que genera una página HTML a partir de una plantilla.
   - **Detalles técnicos:**
     - **Implementación:** Utiliza jinja2 en Python o mustache en Node.js.
     - **Respuesta:** HTML generado dinámicamente con valores aleatorios.
   - **Complejidad:** Media.
   - **Motivo de la selección:** esta aplicación representa un caso de generación dinámica de contenido.

- **Adaptaciones realizadas:**
    - Se creó un dockerfile con AWS Lambda Web Adapter para permitir el manejo solicitudes HTTP.
    - Se creó un archivo app.py donde se añadió un endpoint /dynamic que usa Jinja2 para generar HTML con datos aleatorios.

4. **Aplicación 4:**
    - **Descripción:** Benchmark llamado "Uploader" que descarga un archivo desde una URL, lo sube a un bucket S3 y retorna la ubicación.
    - **Detalles técnicos:**
        - Requiere configuración de credenciales y enlace con un bucket S3.
        - Implica interacción con almacenamiento en la nube.
    - **Complejidad:** Alta.
    - **Motivo de la selección:** esta aplicación representa un uso común en aplicaciones modernas ya que, requiere integración con almacenamiento en la nube (S3).
    - **Adaptaciones realizadas:**
        - Se creó un dockerfile con AWS Lambda Web Adapter para permitir el manejo solicitudes HTTP.
        - Se creó un archivo app.py donde se añadió un endpoint /upload que recibe una URL de un archivo.
        - Se creó un bucket y se configuró un rol IAM con permisos de escritura en S3.

3.4 Pruebas Realizadas

Las pruebas de sobrecarga realizadas fueron ejecutadas utilizando la herramienta k6, diseñada específicamente para medir el rendimiento y la capacidad de los sistemas bajo condiciones de carga controlada. El script utilizado en las pruebas está diseñado para simular una carga realista y progresiva, distribuyendo el tráfico de usuarios durante un total de 12 minutos. La duración y los parámetros de la prueba se definieron de la siguiente manera:

1. **Aumento Gradual (1 minuto):** La prueba comienza con un aumento progresivo en el número de usuarios, pasando de 0 a 100 usuarios simultáneos.
2. **Carga Sostenida (3 minutos):** la carga se estabiliza en 500 usuarios simultáneos, representando un escenario de carga moderada y constante.
3. **Pico de Carga (2 minutos):** El número de usuarios aumenta a 700, simulando un pico de tráfico elevado que puede ocurrir en momentos críticos.
4. **Carga Máxima (3 minutos):** Se incrementa la carga a 1000 usuarios simultáneos, alcanzando el límite máximo de la prueba.
5. **Descenso Progresivo (2 minutos):** El número de usuarios se reduce gradualmente, pasando de 1000 a 700 usuarios.
6. **Periodo de Reposo (1 minuto):** La prueba finaliza con una reducción a 50 usuarios simultáneos, simulando un escenario de menor demanda.

Se definió la prueba con una duración de 12 minutos, ya que permite evaluar el rendimiento al cubrir un ciclo completo de carga y permite simular un tráfico real. En paralelo con la prueba de k6 el sistema se monitorea cada 40 segundos para capturar la suma de las métricas en AWS CloudWatch, este monitoreo permite correlacionar la carga con el consumo de recursos y el costo del servicio.

3.4.1 Resultados de la aplicación 1

3.4.1.1 Escenario 1

Despliegue independiente en Lambda

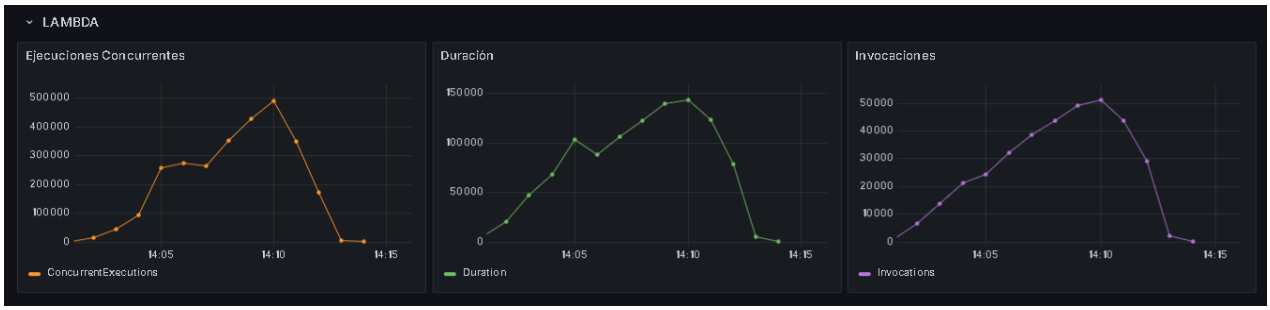


Figura 13 Visualización de métricas en Grafana de Lambda (Aplicación 1)

El número de invocaciones alcanzó picos de hasta 136,976 invocaciones en un período de 40 segundos. Los tiempos de duración total oscilaron entre 8.70 ms en las primeras ejecuciones y hasta 384,853 ms durante los periodos de mayor carga.

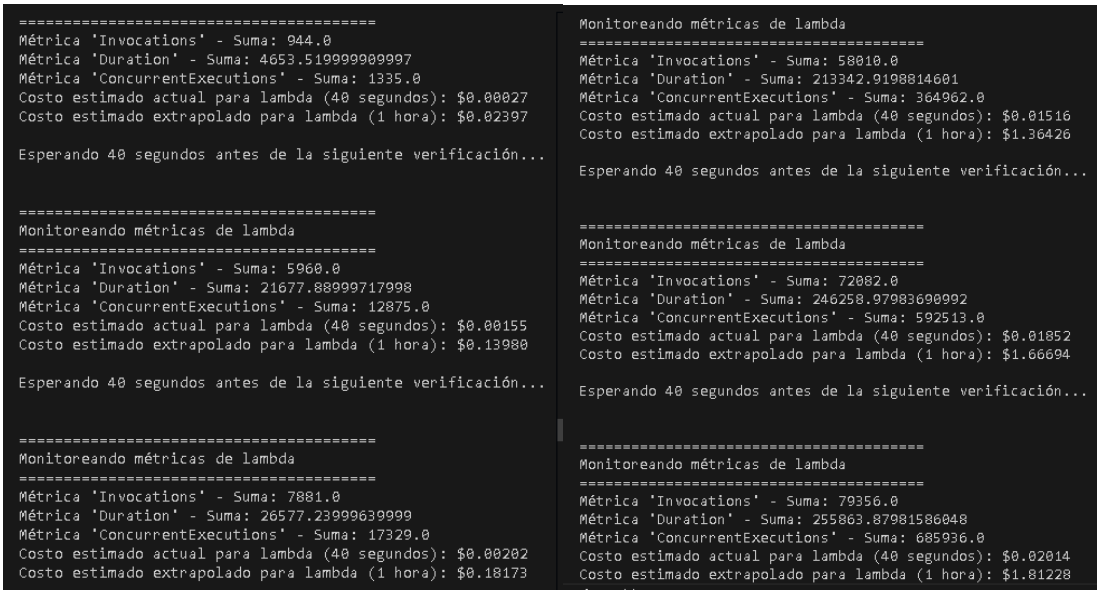


Figura 14 Monitoreo de métricas y costos de lambda (Aplicación 1)

Los resultados de la prueba en K6 incluyen:

- **Duración promedio de solicitudes**: 663.39 ms, con un percentil p90 de 937.78 ms y un percentil p95 de 1,327.88 ms.
- **Solicitudes exitosas**: 69.19% de las 43,121 solicitudes totales.
- **Tasa de fallos**: 30.81% (13,294 solicitudes fallidas).

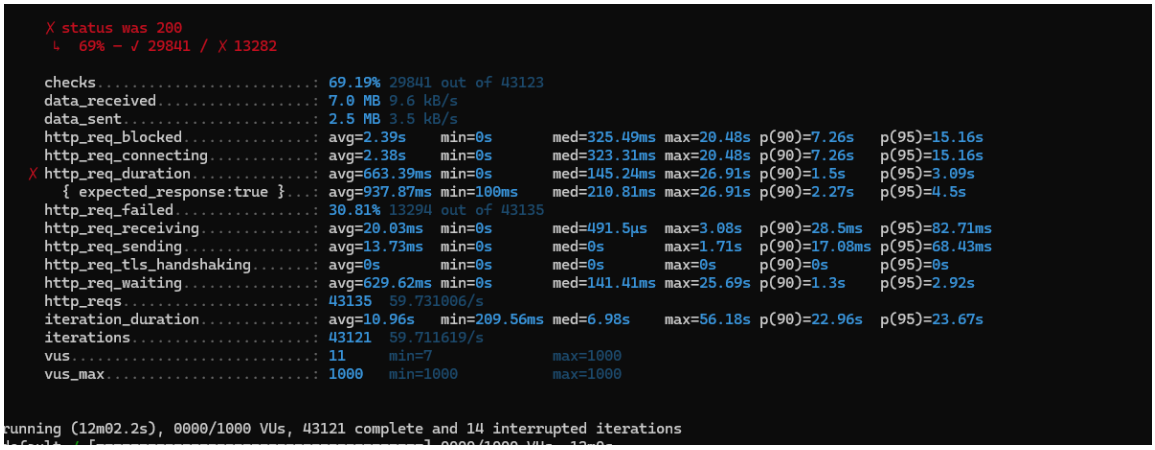


Figura 15 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 1)

Se calcularon los costos estimados para Lambda en intervalos de 40 segundos. La siguiente tabla resume los datos recolectados para la aplicación 1:

Tabla 2 Resultados del monitoreo de métricas y costos de Lambda (Aplicación 1)

	Tiempo	Invocaciones	Duración (ms)	Ejecuciones concurrentes	Costo (USD/s)
1	40s	4	8.70	4	$0,00000
2	40s	944	4653.51	1335	$0,00027
3	40s	5960	21677.88	12875	$0,00155
4	40s	7881	26577.23	17329	$0,00202
5	40s	20515	70883.26	58149	$0,00528
6	40s	38156	124687.65	137141	$0,00971
7	40s	47753	163740.22	235932	$0,01228
8	40s	58010	213342.91	364962	$0,01516
9	40s	72082	246258.97	592513	$0,01852
10	40s	79356	255863.87	685936	$0,02014
11	40s	93378	294392.55	783738	$0,02358
12	40s	107458	297446.77	835094	$0,02645
13	40s	115302	323306.66	909025	$0,02845
14	40s	130752	366335.89	1039523	$0,03226
15	40s	136976	384853.38	1210257	$0,03381
16	40s	122295	340392.41	1004543	$0,03013
17	40s	75229	207372.03	525653	$0,01850
18	40s	31430	84075.05	176154	$0,00769
				Total:	**$0,28580**

Los costos varían directamente con el nivel de utilización de CPU. El costo total de la prueba fue de $0,28580, el máximo estimado en un intervalo de alta carga fue de $0,03381en un intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a valor $0,0.

Despliegue independiente en Fargate

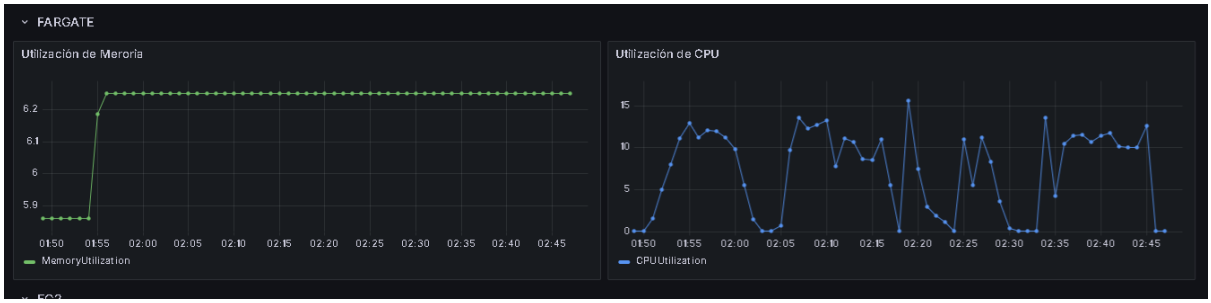


Figura 16 Visualización de métricas en Grafana de fargate (Aplicación 1)

La utilización de memoria se mantuvo constante en 12.5 unidades durante todo el período de monitoreo. La utilización de CPU mostró fluctuaciones significativas dependiendo de la carga. En las primeras iteraciones, los picos llegaron hasta 33.43%, pero eventualmente decrecieron hasta 0.08%.

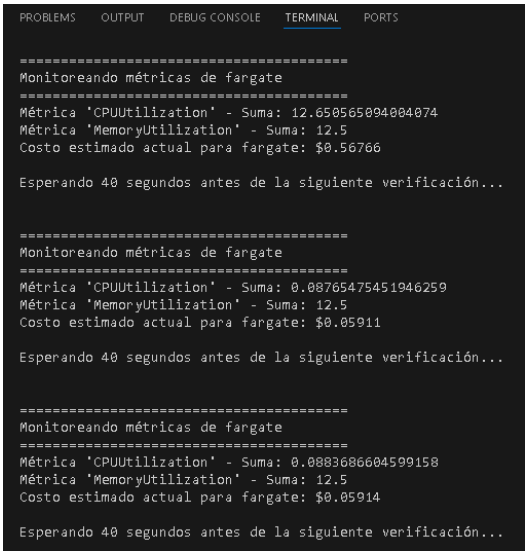


Figura 17 Monitoreo de métricas y costos de Fargate (Aplicación 1)

Los resultados de la prueba en K6 incluyen:

- **Duración promedio de solicitudes:** 176.92ms, con un percentil p90 de 219.72ms y un percentil p95 de 370.14ms.
- **Solicitudes exitosas:** 68.6% de las 46,506 solicitudes totales.
- **Tasa de fallos:** 31.13% (13,223 solicitudes fallidas).

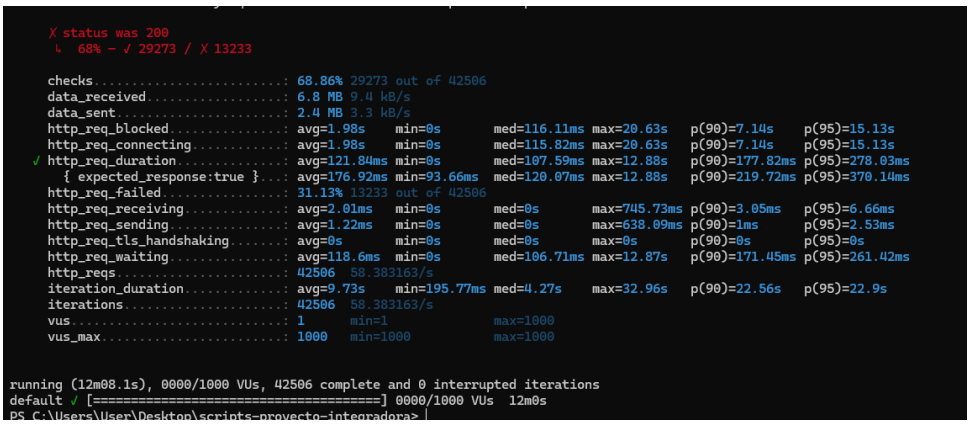


Figura 18 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 1)

Se calcularon los costos estimados para fargate en intervalos de 40 segundos durante los 12min de la prueba. La siguiente tabla resume los datos recolectados:

Tabla 3 Resultados del monitoreo de métricas y costos de Fargate (Aplicación 1)

	Tiempo	CPU	Memoria	Costo (USD/s)
1	40s	2.98	12.5	$0,00196
2	40s	4.88	12.5	$0,00281
3	40s	9.68	6.25	$0,00497
4	40s	12.17	18.76	$0,00609
5	40s	23.25	12.5	$0,01107
6	40s	26.05	12.5	$0,01233
7	40s	21.91	12.5	$0,01233
8	40s	33.43	12.5	$0,01565
9	40s	33.60	12.5	$0,01565
10	40s	33.67	12.5	$0,01576
11	40s	23.30	12.5	$0,01367
12	40s	21.99	12.5	$0,01367
13	40s	20.20	12.5	$0,00970
14	40s	20.10	12.5	$0,00970
15	40s	22.69	12.5	$0,01082
16	40s	12.65	12.5	$0,01080
17	40s	0.08	12.5	$0,00065
			Total:	**$0,16766**

Los costos varían directamente con el nivel de utilización de CPU. El costo total de la prueba fue de $0,16766, el máximo estimado en un intervalo de alta carga fue de $0,01576 en un intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a valor $0,00065.

Despliegue independiente en Ec2

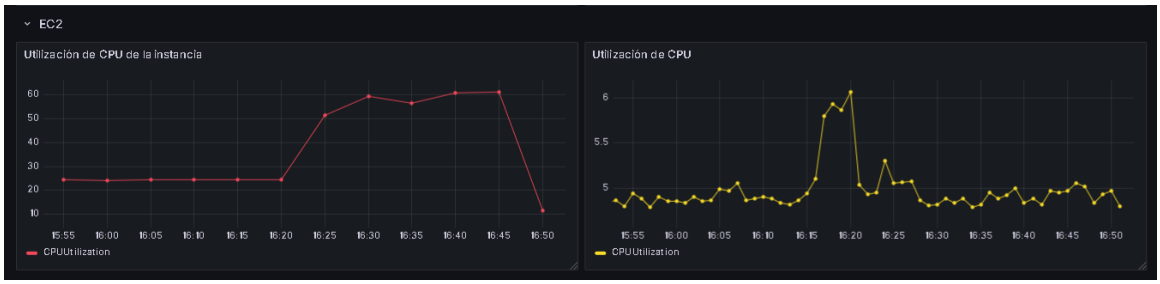


Figura 19 Visualización de CPU en Grafana de la instancia de EC2 (Aplicación 1)

Durante el monitoreo de la instancia EC2, se observó que la métrica CPUUtilization presentó fluctuaciones significativas en la prueba. La utilización de CPU inició con niveles moderados y alcanzó un punto máximo a los 8 minutos.

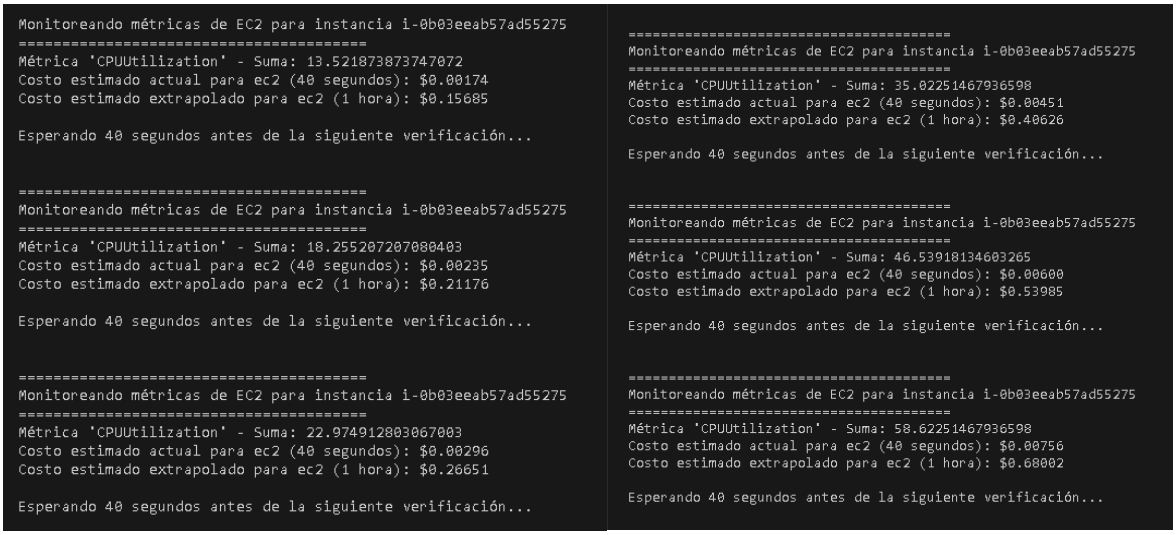


Figura 20 Monitoreo de métricas y costos en la instancia EC2 (Aplicación 1)

Los resultados de la prueba en K6 incluyen:

- **Duración promedio de solicitudes:** 197.55ms, con un percentil p90 de 267.15ms y un percentil p95 de 507.55ms.
- **Solicitudes exitosas:** 99.97% de las 26,842 solicitudes.
- **Tasa de fallos:** 0.02% (8 solicitudes fallidas).

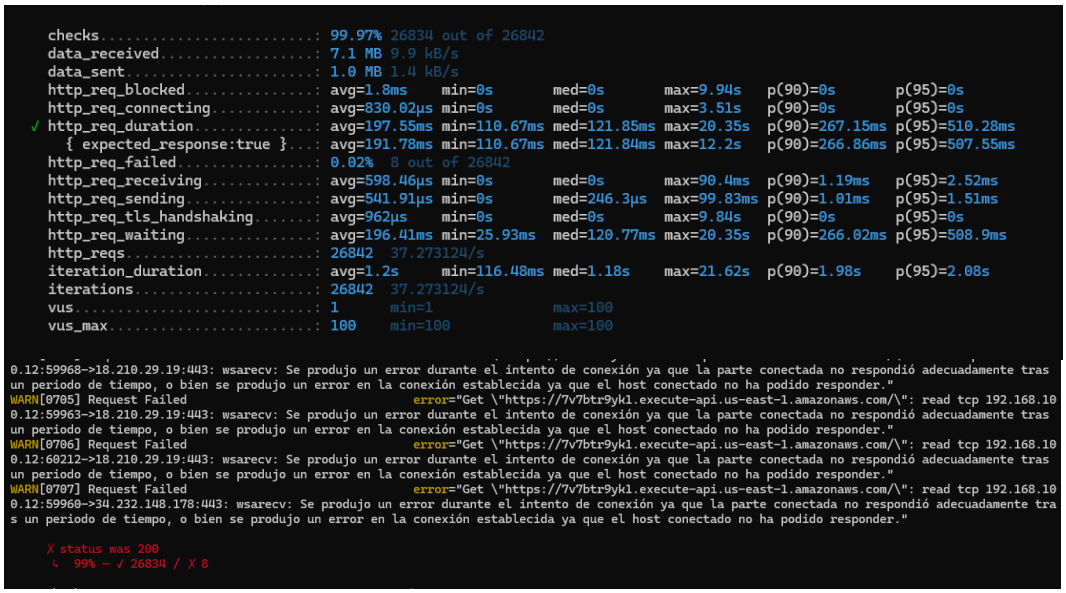


Figura 21 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 1)

Se calcularon los costos estimados para la instancia EC2 en intervalos de 40 segundos. La siguiente tabla resume los datos recolectados:

Tabla 4 Resultados del monitoreo de métricas y costos de la instancia EC2 (Aplicación 1)

	Tiempo	CPU	Costo (USD/s)
1	40s	14.06	$0,00181
2	40s	20.47	$0,00264
3	40s	33.86	$0,00437
4	40s	10.29	$0,00133
5	40s	35.02	$0,00451
6	40s	46.53	$0,00600
7	40s	58.62	$0,00756
8	40s	11.78	$0,00152
9	40s	23.23	$0,00299
10	40s	34.92	$0,00450
11	40s	47.04	$0,00606
12	40s	59.62	$0,00768
13	40s	27.65	$0,00356
14	40s	22.97	$0,00296
15	40s	18.25	$0,00235
16	40s	13.52	$0,00174
17	40s	8.82	$0,00122
18	40s	9.46	$0,00154
		Total:	**$0,14234**

Los costos varían directamente con el nivel de utilización de CPU. El costo total de la prueba fue de $0,14234, el costo máximo estimado en un intervalo de alta carga fue de $0,00768 en un intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente siendo el menor valor $0,00122.

3.4.1.2 Escenario 2

Figura 22 Visualización de métricas en Grafana del escenario 2 (Aplicación 1)

En Lambda se observaron picos de ejecución concurrente de hasta 36432 invocaciones. En Fargate la utilización de CPU varió entre 10.98% y 33.61% durante el proceso de cambio. En ECS La utilización de CPU mostró valores entre 12.23% y 54.81% dependiendo de la carga.

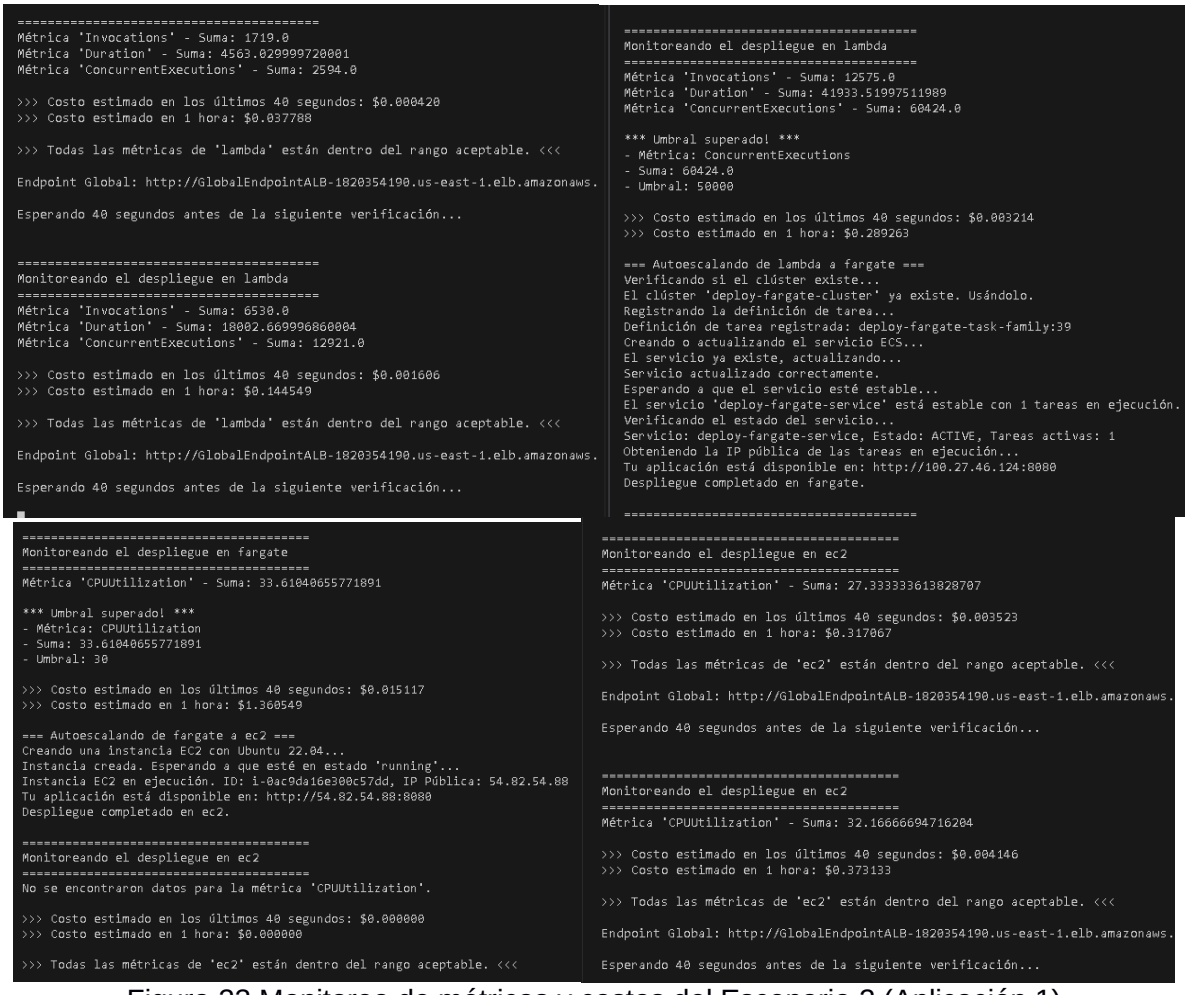


Figura 23 Monitoreo de métricas y costos del Escenario 2 (Aplicación 1)

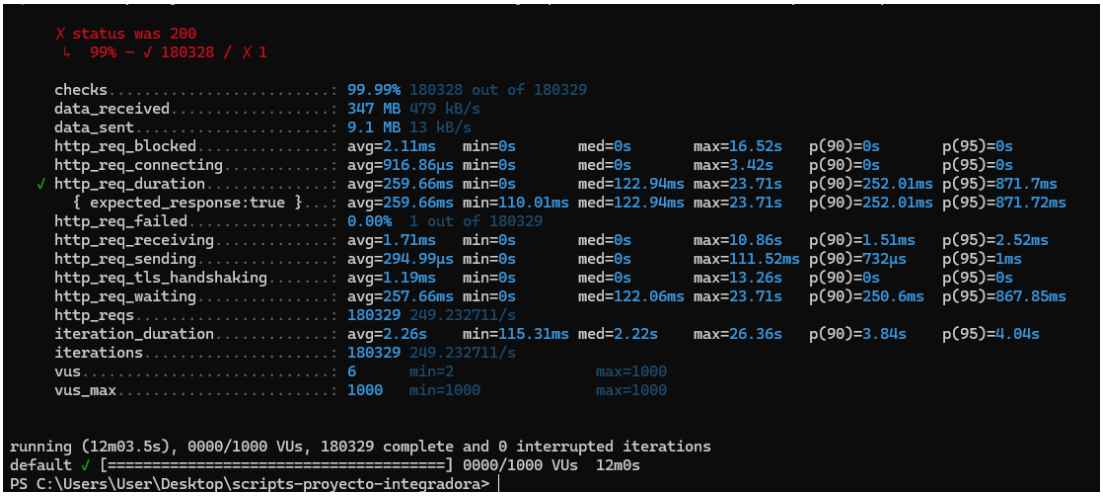


Figura 24 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 1)

Los resultados de la prueba en K6 incluyen:

- Duración promedio de solicitudes: 259.66 ms, con un percentil p90 de 252.01 ms y un percentil p95 de 871.77 ms.
- Solicitudes exitosas: 99.99% de las 180,328 solicitudes.
- Tasa de fallos: 0.01% (1 solicitud fallida).

Tabla 5 Resultados del monitoreo de métricas y costos del Escenario 2 (Aplicación 1)

	Tiempo	Invocaciones	Duración (ms)	Ejecuciones concurrentes	CPU	Memoria	Costo (USD/s)
Lambda							
1	40s	71	620.44	79	-	-	$0,00003
2	40s	228	1534.28	257	-	-	$0,00027
3	40s	2366	7789.91	3194	-	-	$0,00060
4	40s	6743	19070.84	10986	-	-	$0,00167
5	40s	8966	24664.66	15456	-	-	$0,00220
6	40s	17251	46082.78	36432	-	-	$0,00422
Cambio a Fargate							
7	40s	-	-	-	10.98	12.5	$0,00556
8	40s	-	-	-	24.35	18.78	$0,01188
9	40s	-	-	-	17.27	12.53	$0,00839
10	40s	-	-	-	20.74	12.53	$0,00995
11	40s	-	-	-	33.61	12.54	$0,01512
Cambio a EC2							
12	40s	-	-	-	54.81	-	$0,00707
13	40s	-	-	-	33.83	-	$0,00436
14	40s	-	-	-	29.49	-	$0,00380
15	40s	-	-	-	50.65	-	$0,00653
16	40s	-	-	-	34.14	-	$0,00440

17	40s	-	-	-	43.79	-	$0,00565
18	40s	-	-	-	12.23	-	$0,00158
						Total:	**$0,09326**

El costo total acumulado para este escenario fue de $0.09326, desglosado de la siguiente manera:

- **Lambda:** $0,00899
- **Fargate:** $0,05089
- **EC2:** $0.03739

3.4.2 Resultados de la aplicación 2

3.4.2.1 Escenario 1

Despliegue independiente en Lambda

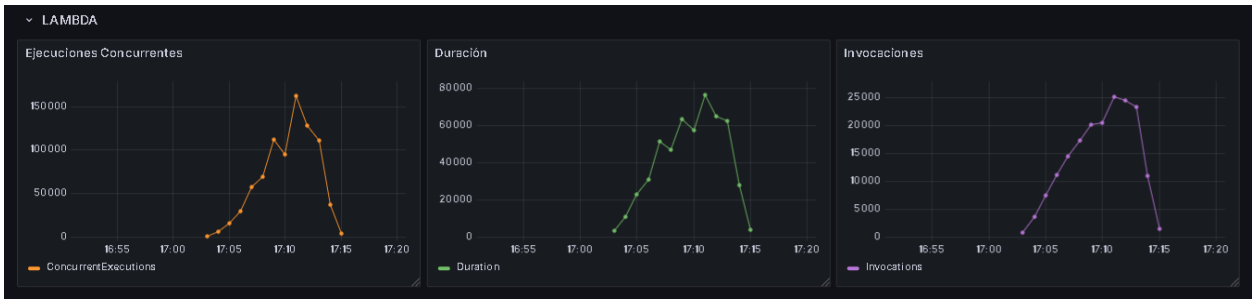


Figura 25 Visualización de métricas en Grafana de Lambda (Aplicación 2)

El número de invocaciones alcanzó picos de hasta 126140 invocaciones en un período de 40 segundos. Los tiempos de duración total oscilaron entre 57.0 ms en las primeras ejecuciones y hasta 383009.18ms durante los periodos de mayor carga.

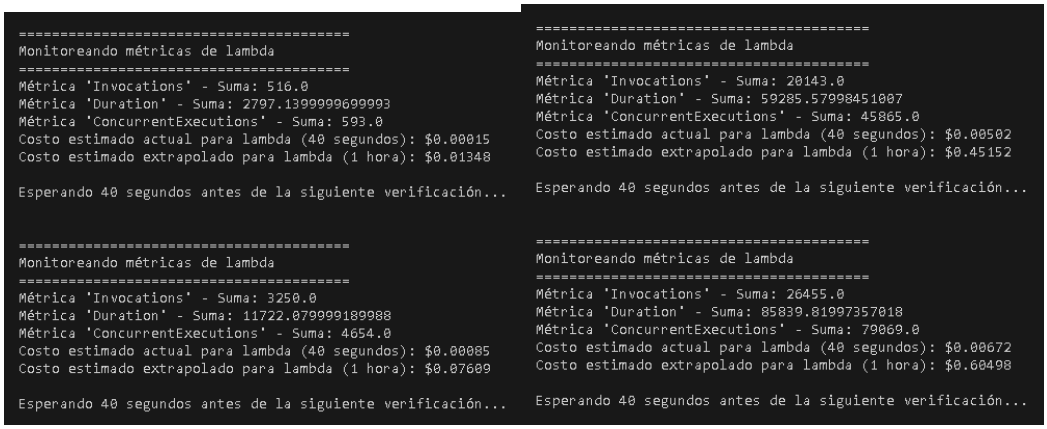


Figura 26 Monitoreo de métricas y costos de lambda (Aplicación 2)

Los resultados de la prueba en K6 incluyen:

- Duración promedio de solicitudes: 636.39 ms
- Solicitudes exitosas: 37.86%
- Tasa de fallos: 62.21%

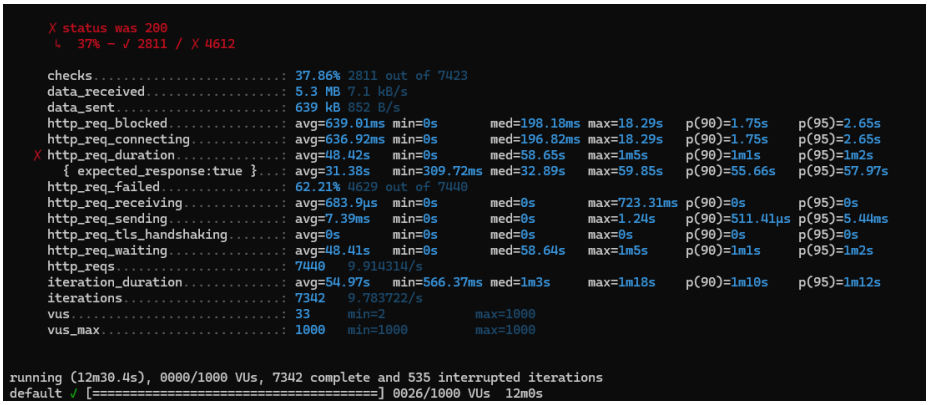


Figura 27 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 2)

Se calcularon los costos estimados para Lambda en intervalos de 40 segundos. La siguiente tabla resume los datos recolectados para la aplicación 2:

Tabla 6 Resultados del monitoreo de métricas y costos de Lambda (Aplicación 2)

	Tiempo	Invocaciones	Duración (ms)	Ejecuciones concurrentes	Costo (USD/s)
1	40s	100	57	302	$0,00000
2	40s	397	997	502	$0,00010
3	40s	908	2254.50	1164	$0,00022
4	40s	5604	15368.24	9951	$0,00138
5	40s	14829	41592.57	34331	$0,00366
6	40s	18538	52243.16	44707	$0,00458
7	40s	33196	95456.77	98989	$0,00823
8	40s	48397	140904.61	179574	$0,01203
9	40s	50006	145931.17	187187	$0,01243
10	40s	60215	178672.12	269279	$0,01502
11	40s	69964	207415.52	318476	$0,01745
12	40s	80891	242045.74	424996	$0,02021
13	40s	89364	269659.38	530896	$0,02237
14	40s	100200	302477.23	604169	$0,02508
15	40s	109340	332006.03	732624	$0,02740
16	40s	115808	352702.33	841545	$0,02904
17	40s	126140	383009.18	915665	$0,03161
18	40s	69206	199285.51	411472	$0,01716
				Total:	**$0,24797**

El costo total de la prueba fue de $0, 24797, el máximo estimado en un intervalo de alta carga fue de $0,03161 en un intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a valor $0,0.

Despliegue independiente en Fargate

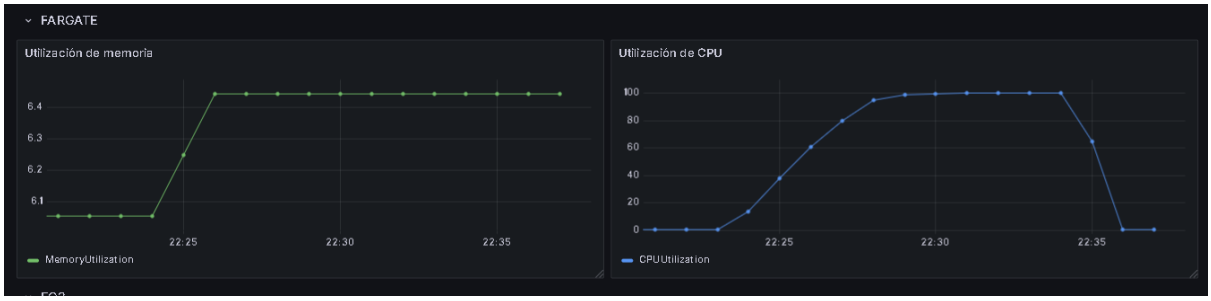


Figura 28 Visualización de métricas en Grafana de fargate (Aplicación 2)

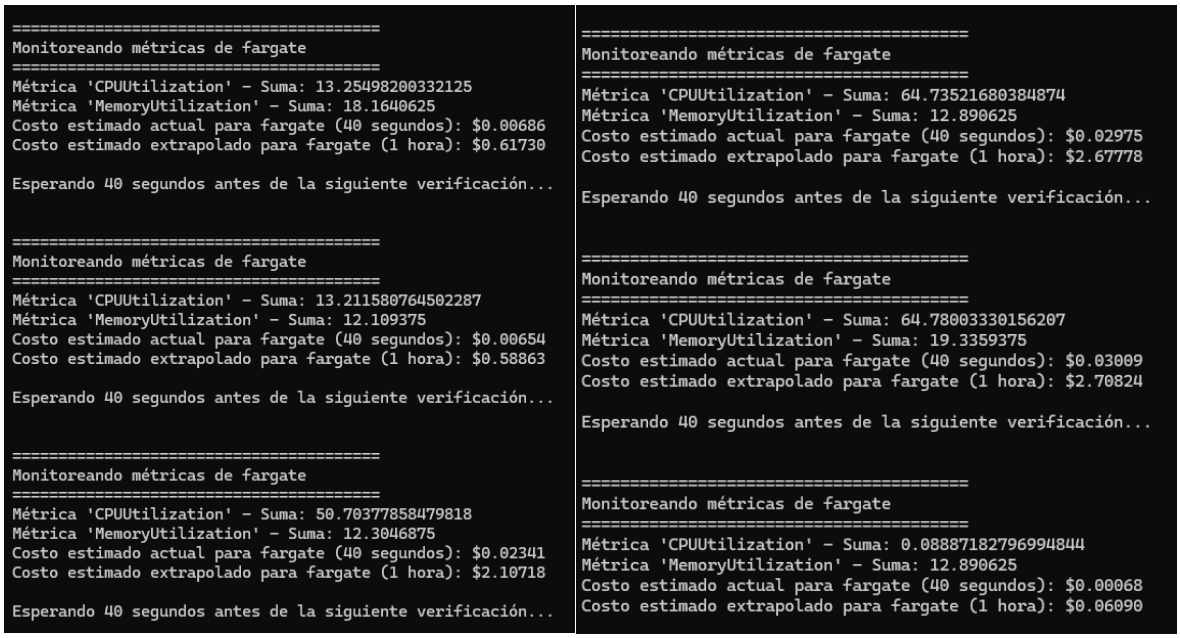


Figura 29 Monitoreo de métricas y costos de Fargate (Aplicación 2)

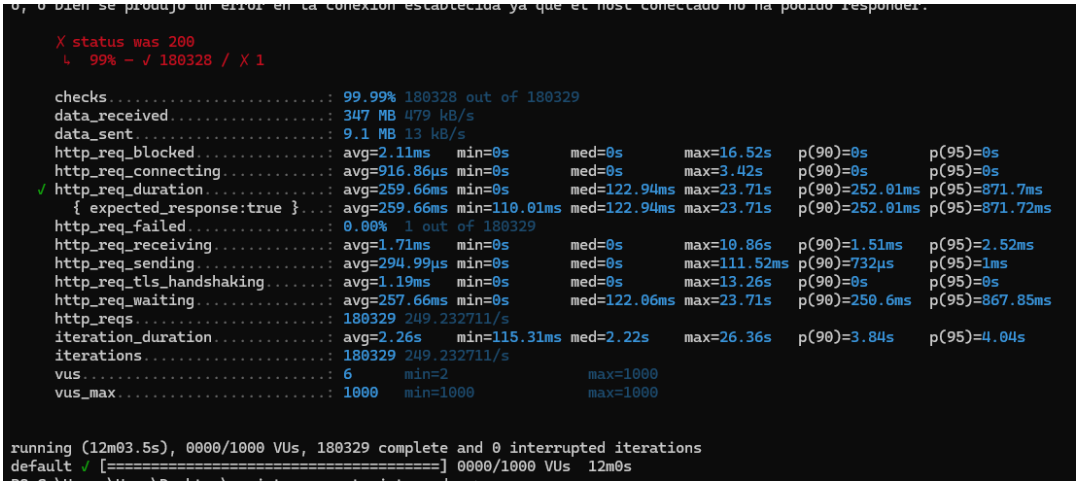


Figura 30 - Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 2)

Se calcularon los costos estimados para fargate en intervalos de 40 segundos durante los 12min de la prueba. La siguiente tabla resume los datos recolectados:

Tabla 7 Resultados del monitoreo de métricas y costos de Fargate (Aplicación 2)

	Tiempo	CPU	Memoria	Costo (USD/s)
1	40s	0.08	12.10	$0,00064
2	40s	13.25	18.16	$0,00685
3	40s	37.5	12.5	$0,01748
4	40s	50.70	12.5	$0,02342
5	40s	64	12.5	$0,02805
6	40s	79.9	12.5	$0,03655
7	40s	80	12.5	$0,03659
8	40s	94.6	12.5	$0,03659
9	40s	99.1	12.5	$0,04519
10	40s	99.5	12.5	$0,04537
11	40s	99.55	12.5	$0,04537
12	40s	99.90	12.5	$0,04555
13	40s	100.00	12.5	$0,04559
14	40s	99.90	12.5	$0,04555
15	40s	100.00	12.5	$0,04559
16	40s	64.7	12.5	$0,02971
17	40s	22.39	12.5	$0,01080
18	40s	22.69	12.5	$0,01082
			Total:	**$0,55572**

Los costos varían directamente con el nivel de utilización de CPU. El costo total de la prueba fue de $0,55572, el máximo estimado en un intervalo de alta carga fue de $0,04559 en un

intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a $0,00064.

Despliegue independiente en EC2

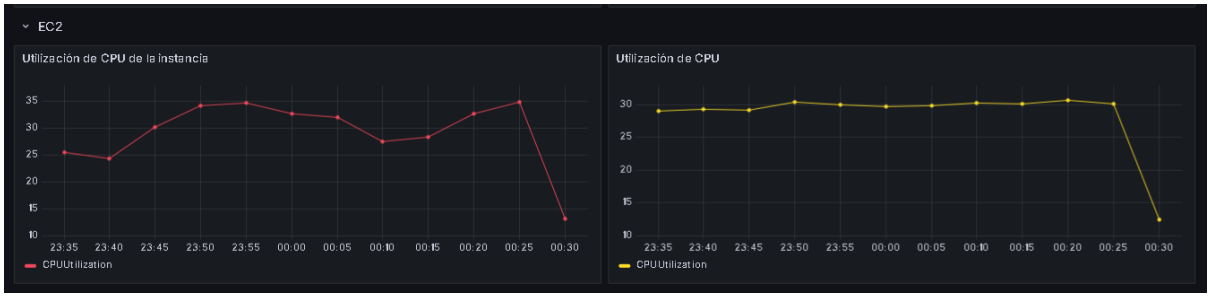


Figura 31 Visualización de CPU en Grafana de la instancia de EC2 (Aplicación 2)

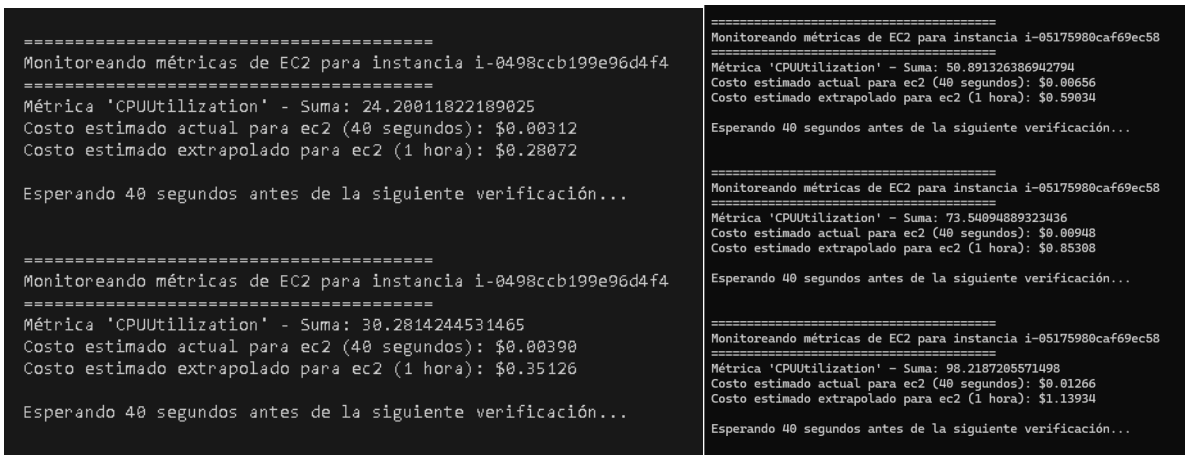


Figura 32 Monitoreo de métricas y costos en la instancia EC2 (Aplicación 2)

Figura 33 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 3)

Se calcularon los costos estimados para la instancia EC2 en intervalos de 40 segundos. La siguiente tabla resume los datos recolectados:

Tabla 8 Resultados del monitoreo de métricas y costos de la instancia EC2 (Aplicación 2)

	Tiempo	**CPU**	**Costo (USD/s)**
1	40s	17.49	$0,00226
2	40s	23.31	$0,00301
3	40s	29.31	$0,00378
4	40s	24.04	$0,00310
5	40s	30.28	$0,00390
6	40s	28.10	$0,00362
7	40s	50.89	$0,00656
8	40s	73.54	$0,00948
9	40s	98.21	$0,03266
10	40s	55.88	$0,00720
11	40s	26.66	$0,00344
12	40s	86.77	$0,01118
13	40s	116.17	$0,04497
14	40s	142.16	$0,05832
15	40s	26.83	$0,00346
16	40s	21.90	$0,00282
17	40s	28.38	$0,00366
18	40s	28.33	$0,00365
		Total:	$0,20481

Los costos varían directamente con el nivel de utilización de CPU. El costo total de la prueba fue de $0,20481, el máximo estimado en un intervalo de alta carga fue de $0,04497en un intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a valor $0,00226.

3.4.3 Resultados de la aplicación 3

3.4.3.1 Escenario 1

Despliegue independiente en Lambda

Figura 34 Visualización de métricas en Grafana de Lambda (Aplicación 3)

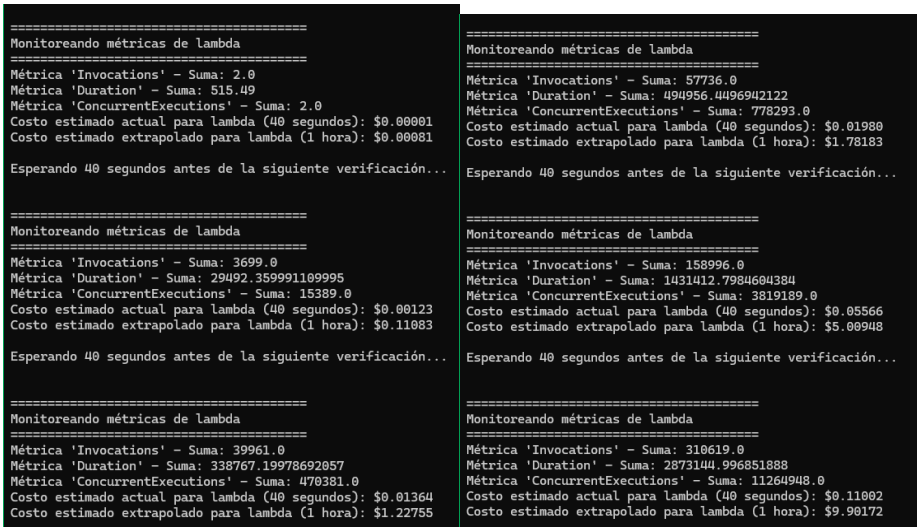


Figura 35 Monitoreo de métricas y costos de lambda (Aplicación 3)

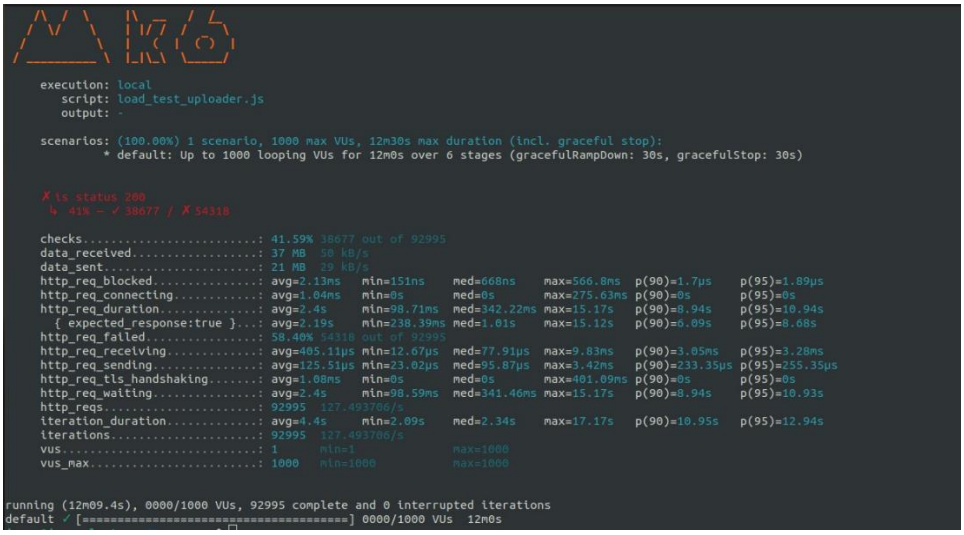


Figura 36 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 3)

Se calcularon los costos estimados para Lambda en intervalos de 40 segundos. La siguiente tabla resume los datos recolectados para la aplicación 3:

Tabla 9 Resultados del monitoreo de métricas y costos de Lambda (Aplicación 3)

	Tiempo	Invocaciones	Duración (ms)	Ejecuciones concurrentes	Costo (USD/s)
1	40s	1	2.25	1	$0,00000
2	40s	515	2797.12	592	$0,00015
3	40s	3699	29492.35	15389	$0,00123
4	40s	39961	338767.19	470381	$0,01364
5	40s	57736	494956.44	778293	$0,01980
6	40s	158996	1431412.79	3819189	$0,05566
7	40s	310619	2873144.99	11264948	$0,11002
8	40s	350893	3260223.71	13467929	$0,12453
9	40s	516820	4893161.76	26413974	$0,18493
10	40s	621287	5925930.66	37035026	$0,22304
11	40s	607249	5790318.66	38919653	$0,21797
12	40s	704285	6715104.14	45150910	$0,25280
13	40s	683969	6525517.66	44310559	$0,24557
14	40s	658469	6294647.47	43091324	$0,23663
15	40s	719631	6868187.00	47073930	$0,25842
16	40s	695202	6583095.14	45988248	$0,24878
17	40s	530051	4936411.75	32740990	$0,18830
18	40s	289083	2700801.88	16713176	$0,10284
				Total:	**$2,48431**

El costo total de la prueba fue de $2,48431, el máximo estimado en un intervalo de alta carga fue de $0,25842 en un intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a valor $0,0.

Despliegue independiente en Fargate

Figura 37 Visualización de métricas en Grafana de fargate (Aplicación 3)

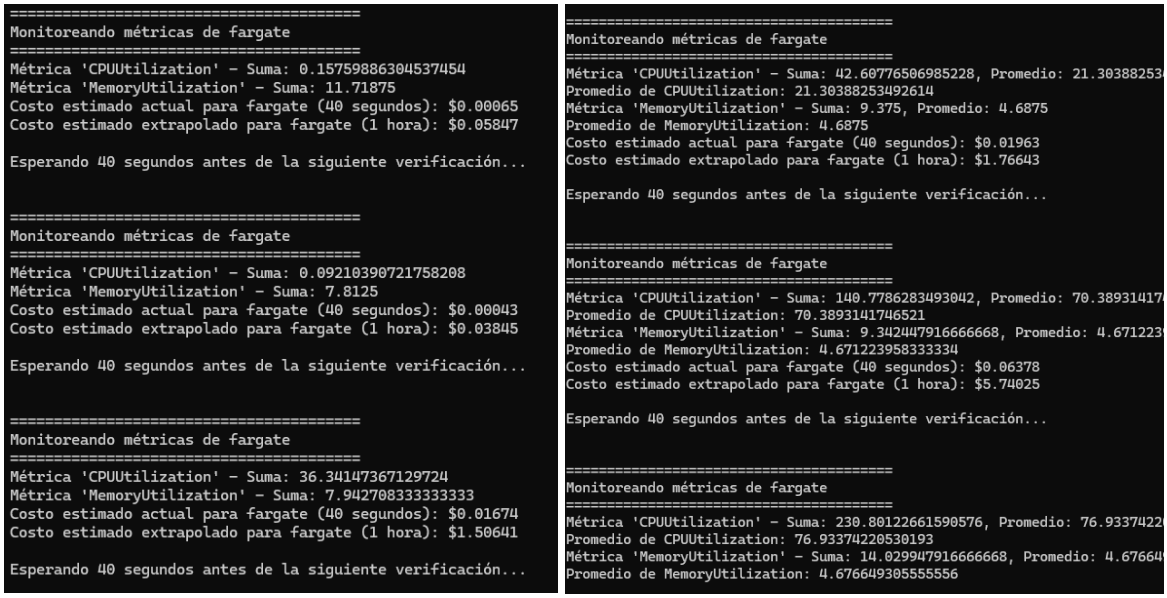


Figura 38 onitoreo de métricas y costos de Fargate (Aplicación 3)

Se calcularon los costos estimados para fargate en intervalos de 40 segundos durante los 12min de la prueba. La siguiente tabla resume los datos recolectados:

Tabla 10 Resultados del monitoreo de métricas y costos de Fargate (Aplicación 3)

	Tiempo	**CPU**	**Memoria**	**Costo (USD/s)**
1	40s	0.15	11.71	$0,00065
2	40s	36.33	4.04	$0,01988
3	40s	36.34	4.65	$0,01963
4	40s	94.5	4.69	$0,12705
5	40s	95.4	4.70	$0,12917
6	40s	91.2	4.72	$0,08838
7	40s	96.9	4.75	$0,03659
8	40s	98.5	4.69	$0,13430
9	40s	98.6	4.70	$0,13430
10	40s	96.9	4.72	$0,13430
11	40s	99.8	4.75	$0,13430
12	40s	99.7	4.69	$0,09018
13	40s	99.6	4.70	$0,09018
14	40s	99.5	4.71	$0,13430
15	40s	99.8	4.72	$0,13430
16	40s	99.4	4.73	$0,13430
17	40s	99.2	4.74	$0,13430
18	40s	90.0	4.75	$0,09943
			Total:	**$1,77554**

El costo total de la prueba fue de $1,77554, el máximo estimado en un intervalo de alta carga fue de $0,13430 en un intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a valor $0,00065.

Despliegue independiente en EC2

Figura 39 Visualización de CPU en Grafana de la instancia de EC2 (Aplicación 3)

Durante el monitoreo de la instancia EC2, se observó que la métrica CPUUtilization presentó fluctuaciones significativas en la prueba.

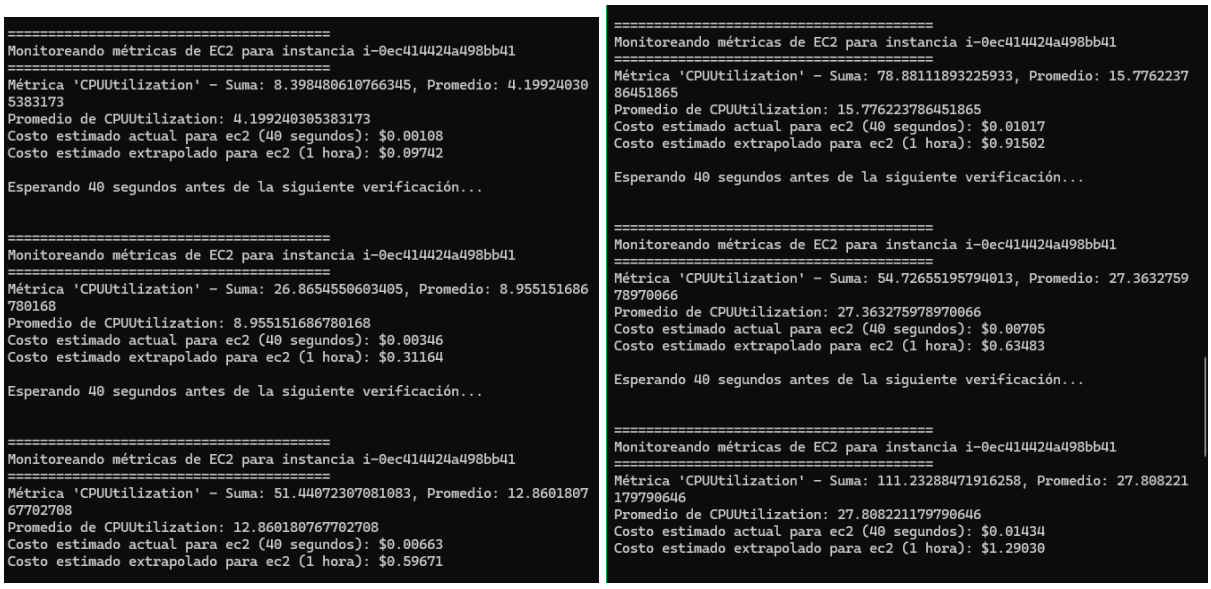


Figura 40 Monitoreo de métricas y costos en la instancia EC2 (Aplicación 3)

Los resultados de la prueba en K6 incluyen:

- Duración promedio de solicitudes: 72.46ms, con un percentil p90 de 267.15ms y un percentil p95 de 268.55ms.
- Solicitudes exitosas: 100%.
- Tasa de fallos: 0%

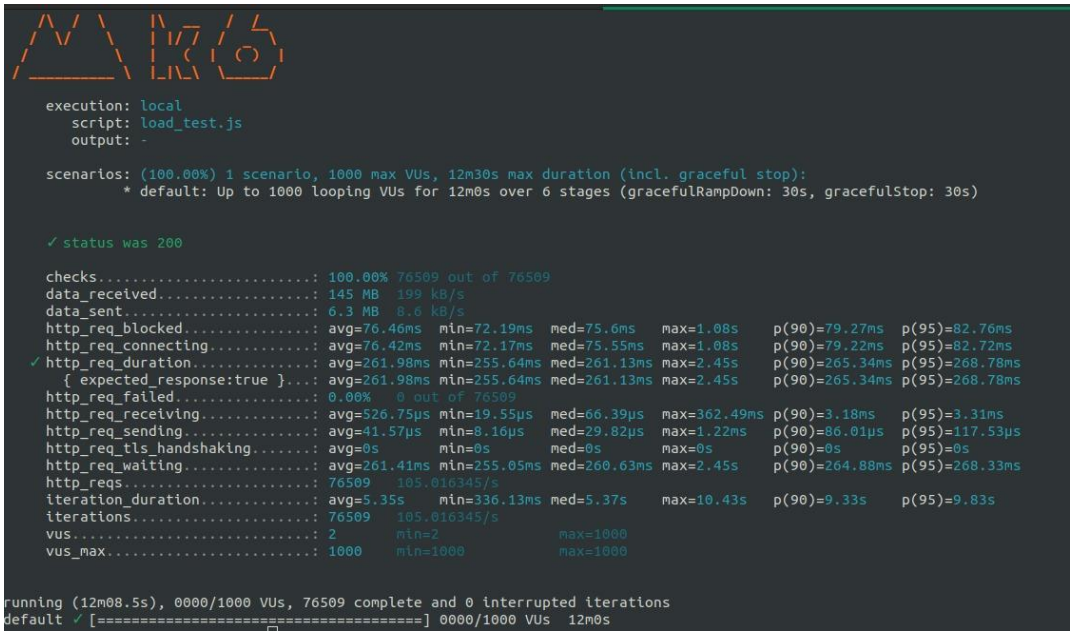


Figura 41 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 3)

Se calcularon los costos estimados para la instancia EC2 en intervalos de 40 segundos. La siguiente tabla resume los datos recolectados:

Tabla 11 Resultados del monitoreo de métricas y costos de la instancia EC2 (Aplicación 3)

	Tiempo	**CPU**	**Costo (USD/s)**
1	40s	8.39	$0,00108
2	40s	26.86	$0,03460
3	40s	51.44	$0,06630
4	40s	78.88	$0,10170
5	40s	54.72	$0,07050
6	40s	111.23	$0,14340
7	40s	139.01	$0,01792
8	40s	27.70	$0,03570
9	40s	55.61	$0,07170
10	40s	82.30	$0,10610
11	40s	82.31	$0,10620
12	40s	107.20	$0,01382
13	40s	111.43	$0,01436
14	40s	104.37	$0,01345
15	40s	54.25	$0,06990
16	40s	82.59	$0,01065
17	40s	77.90	$0,01004
18	40s	51.64	$0,06660
Total:			
			$0,95294

El costo total de la prueba fue de $0,95294, el máximo estimado en un intervalo de alta carga fue de $0,01436 en un intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a valor $0,00108.

3.4.3.2 Escenario 2

Figura 42 Visualización de métricas en Grafana del Escenario 2 (Aplicación 2)

En Lambda se observaron picos de ejecución concurrente de hasta 18328 invocaciones. En Fargate la utilización de CPU varió entre 9.55% y 94.5% durante el proceso de cambio. En ECS La utilización de CPU mostró valores entre 12.61% y 83.6% dependiendo de la carga.

```
====================================
Monitoreando el despliegue en lambda
====================================
Métrica 'Invocations' - Suma: 602.0
Métrica 'Duration' - Suma: 4941.339999849996
Métrica 'ConcurrentExecutions' - Suma: 805.0

>>> Costo estimado en los últimos 40 segundos: $0.000203
>>> Costo estimado en 1 hora: $0.018249

>>> Todas las métricas de 'lambda' están dentro del rango aceptable. <<<

Endpoint Global: http://GlobalEndpointALB-1820354190.us-east-1.elb.amazonaws.c

Esperando 40 segundos antes de la siguiente verificación...


====================================
Monitoreando el despliegue en lambda
====================================
Métrica 'Invocations' - Suma: 1100.0
Métrica 'Duration' - Suma: 7623.299999729996
Métrica 'ConcurrentExecutions' - Suma: 1516.0

>>> Costo estimado en los últimos 40 segundos: $0.000347
>>> Costo estimado en 1 hora: $0.031237

>>> Todas las métricas de 'lambda' están dentro del rango aceptable. <<<

Endpoint Global: http://GlobalEndpointALB-1820354190.us-east-1.elb.amazonaws.c

Esperando 40 segundos antes de la siguiente verificación...
```

```
====================================
Monitoreando el despliegue en lambda
====================================
Métrica 'Invocations' - Suma: 18328.0
Métrica 'Duration' - Suma: 120743.03995436995

*** Umbral superado! ***
- Métrica: Duration
- Suma: 120743.03995436995
- Umbral: 70000

>>> Costo estimado en los últimos 40 segundos: $0.005678
>>> Costo estimado en 1 hora: $0.511055

=== Autoescalando de lambda a fargate ===
Verificando si el clúster existe...
El clúster 'deploy-dynamic-html-cluster' ya existe. Usándolo.
Registrando la definición de tarea...
Definición de tarea registrada: deploy-dynamic-html-task-family:2
Creando o actualizando el servicio ECS...
El servicio ya existe, actualizando...
Servicio actualizado correctamente.
Esperando a que el servicio esté estable...
El servicio 'deploy-dynamic-html-service' está estable con 1 tareas en ejecución.
Verificando el estado del servicio...
Servicio: deploy-dynamic-html-service, Estado: ACTIVE, Tareas activas: 1
Obteniendo la IP pública de las tareas en ejecución...
Tu aplicación está disponible en: http://54.85.58.60:8080
Despliegue completado en fargate.

====================================
Monitoreando el despliegue en fargate
====================================
```

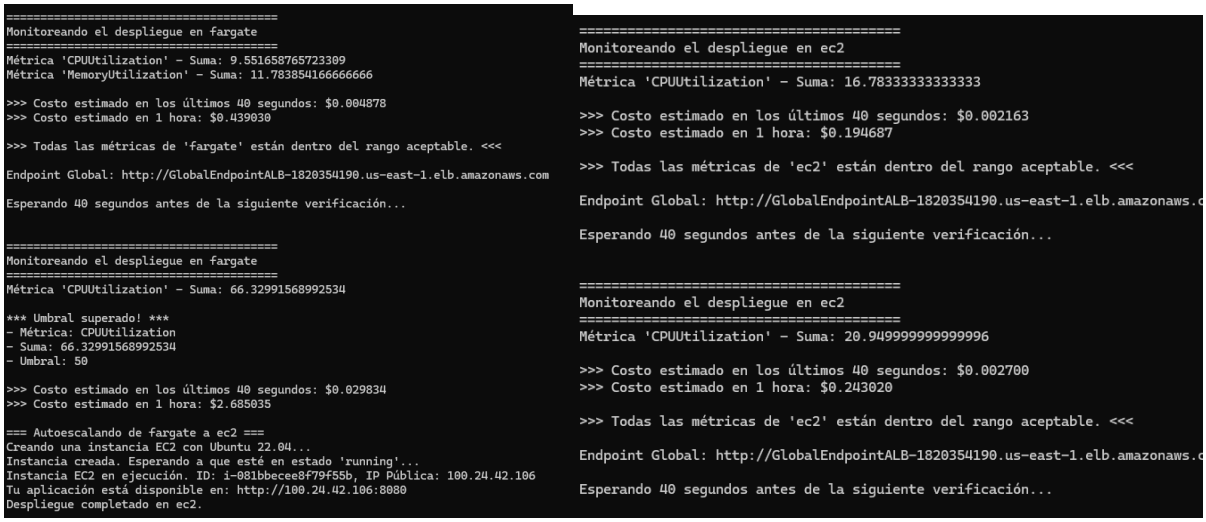


Figura 43 Monitoreo de métricas y costos del Escenario 2 (Aplicación 2)

Tabla 12 Resultados del monitoreo de métricas y costos del Escenario 2 (Aplicación 2)

	Tiempo	Invocaciones	Duración (ms)	Ejecuciones concurrentes	CPU	Memoria	Costo (USD/s)
Lambda							
1	40s	4	8.70	4			$0,00000
2	40s	602.0	4941.33	805			$0,00020
3	40s	1100	7623.29	1516			$0,00034
4	40s	6753	42054.39	14704			$0,00205
5	40s	18328	120743.03	18328			$0,00568
Cambio a Fargate							
6	40s				9.55	11.78	$0,00422
7	40s				56.8	8.79	$0,00488
8	40s				66.32	8.80	$0,07635
9	40s				94.5	4.69	$0,12705
Cambio a EC2							
10	40s				12.61		$0,00198
11	40s				20.94		$0,00264
12	40s				57.5		$0,05600
13	40s				78.1		$0,01014
14	40s				77.8		$0,01948
15	40s				83.6		$0,10620
16	40s				123		$0,04497
17	40s				77.90		$0,01948
18	40s				51.64		$0,06660
				Total:			**$0,54826**

El costo total acumulado para este escenario fue de $0,54826, desglosado de la siguiente manera:

- Lambda: $0,00827
- Fargate: $0,21250
- EC2: $0,32749

Escenario 3

3.4.4 Resultados de la aplicación 4

3.4.4.1 Escenario 1

Despliegue independiente en Lambda

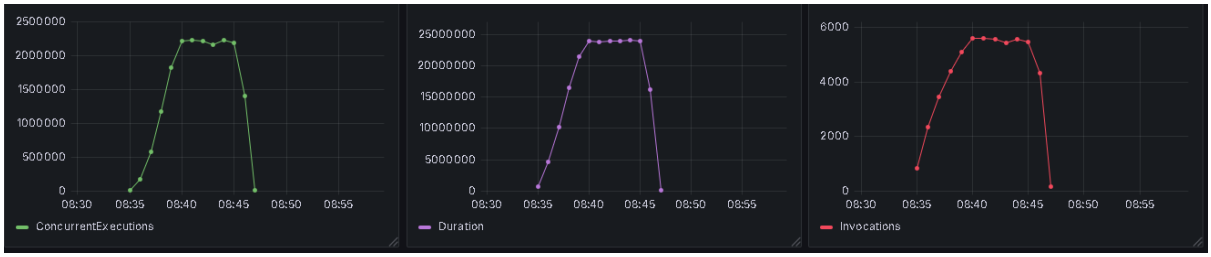


Figura 44 Visualización de métricas en Grafana de Lambda (Aplicación 4)

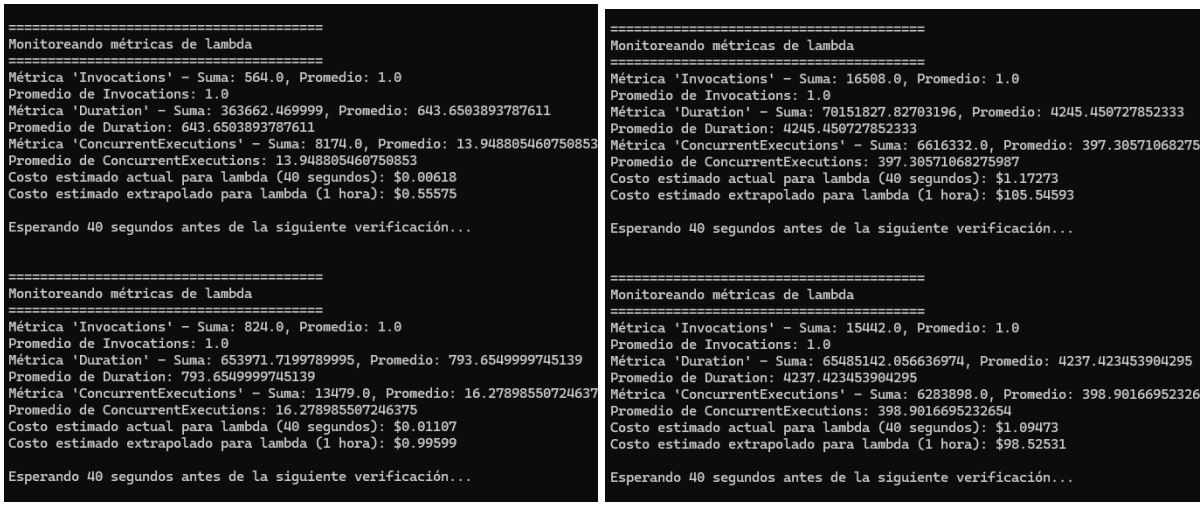


Figura 45 Monitoreo de métricas y costos de lambda (Aplicación 4)

Los resultados de la prueba en K6 incluyen:

- Duración promedio de solicitudes: 29.39 ms, con un percentil p90 de 55.39 ms y un percentil p95 de 59.95 ms.
- Solicitudes exitosas: 46.07%

- Tasa de fallos: 53.92%

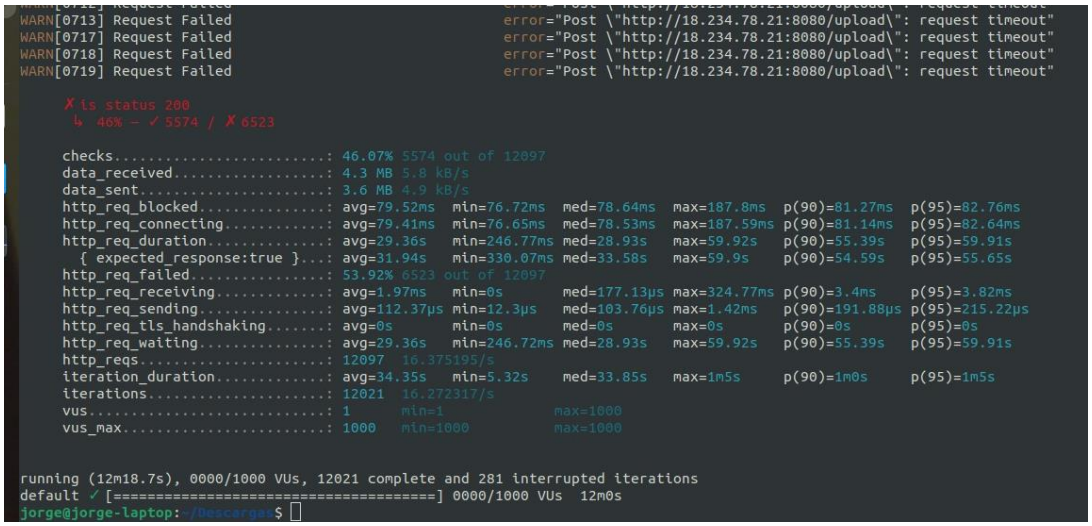


Figura 46 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 4)

Se calcularon los costos estimados para Lambda en intervalos de 40 segundos. La siguiente tabla resume los datos recolectados para la aplicación 4:

Tabla 13 Resultados del monitoreo de métricas y costos de Lambda (Aplicación 4)

	Tiempo	**Invocaciones**	**Duración (ms)**	**Ejecuciones concurrentes**	**Costo (USD/s)**
1	40s	1	45	1	$0,00000
2	40s	564	363662.46	8174	$0,00618
3	40s	824	653971.71	13479	$0,01107
4	40s	2909	4320409.82	167356	$0,07260
5	40s	5879	12390231.50	649849.0	$0,20772
6	40s	7787	21339653.54	1336325	$0,35729
7	40s	9765	29194516.60	1848253	$0,48863
8	40s	11799	42216664.13	3258817	$0,70611
9	40s	12524	49203154.94	4357317	$0,82272
10	40s	14711	59573594.38	5128702	$0,99603
11	40s	15103	62804137.92	5922773	$1,04997
12	40s	14471	60501364.68	5925422	$1,01145
13	40s	16377	70691986.71	6586450	$1,18171
14	40s	15362	63992453.83	5812722	$1,06983
15	40s	9993	40328480.57	3597812	$0,67427
16	40s	4516	16422788.10	1412578	$0,24878
17	40s	179	192963.42	10654	$0,00325
18	40s	5604	15368.24	9951	$0,00138
				Total:	**$8,90899**

Esta fue la prueba más costosa, el costo total de la prueba fue de $8,90899, el máximo estimado en un intervalo de alta carga fue de $1,18171 en un intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a valor $0.

Despliegue independiente en Fargate

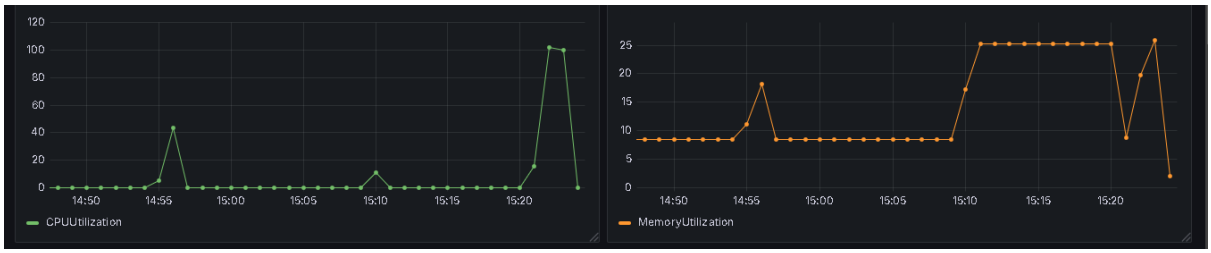


Figura 47 Visualización de métricas en Grafana de fargate (Aplicación 4)

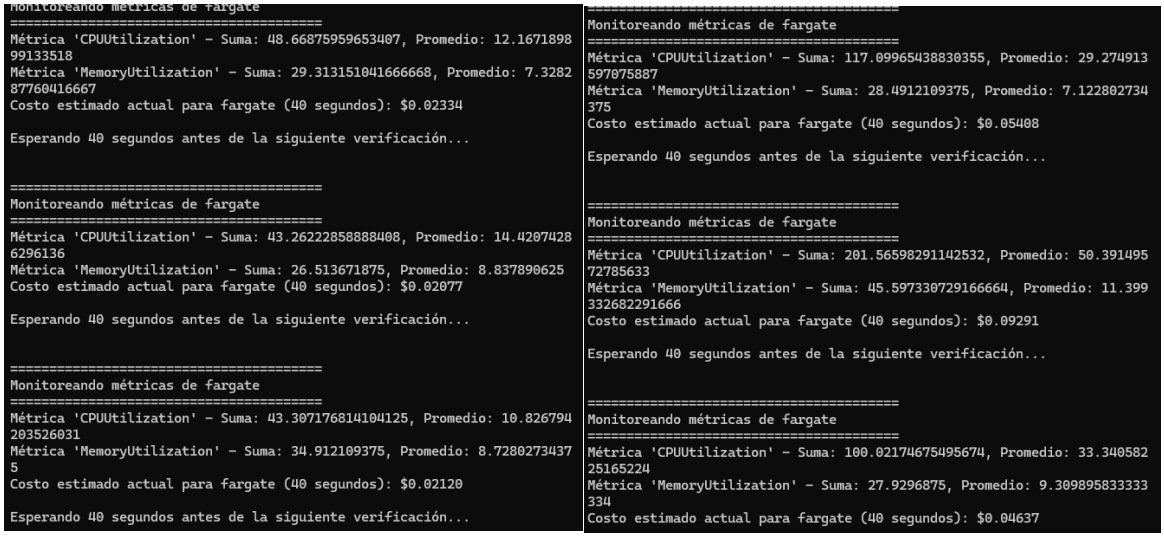


Figura 48 Monitoreo de métricas y costos de Fargate (Aplicación 4)

Los resultados de la prueba en K6 incluyen:

- **Duración promedio de solicitudes**: 663.39 ms, con un percentil p90 de 937.78 ms y un percentil p95 de 1,327.88 ms.
- **Solicitudes exitosas**: 69.19% de las 43,121 solicitudes totales.
- **Tasa de fallos**: 30.81% (13,294 solicitudes fallidas).

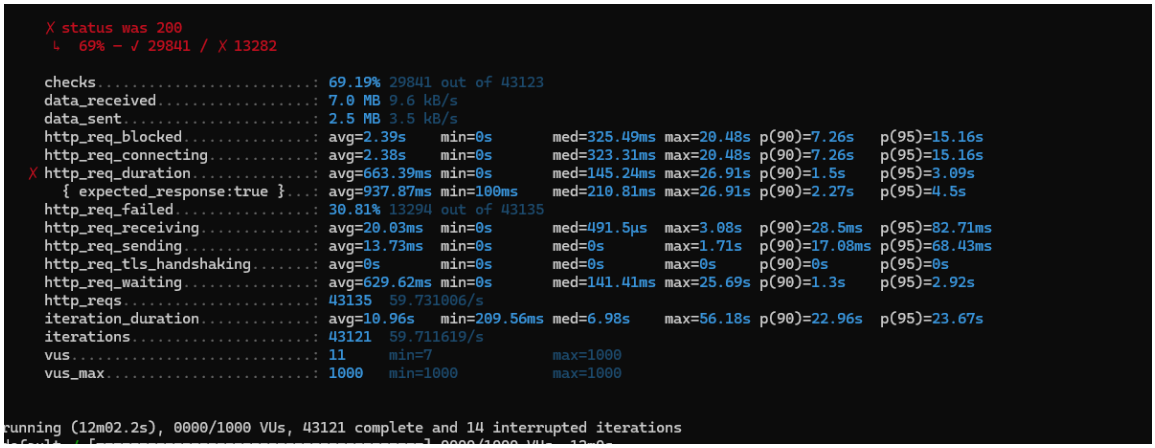


Figura 49 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 4)

Se calcularon los costos estimados para fargate en intervalos de 40 segundos durante los 12min de la prueba. La siguiente tabla resume los datos recolectados:

Tabla 14 Resultados del monitoreo de métricas y costos de Fargate (Aplicación 4)

	Tiempo	CPU	Memoria	Costo (USD/s)
1	40s	0.13	25.19	$0,00131
2	40s	5.49	19.59	$0,00344
3	40s	48.71	37.71	$0,02377
4	40s	43.26	26.51	$0,02077
5	40s	117.28	26.52	$0,16430
6	40s	117.09	28.49	$0,16730
7	40s	201.56	11.39	$0,23530
8	40s	100.02	25	$0,13430
9	40s	200	25	$0,23430
10	40s	283.74	25	$0,24300
11	40s	363.72	25	$0,56482
12	40s	493.42	25	$0,72316
13	40s	300.0	25	$0,45616
14	40s	176.81	25	$0,18430
15	40s	182.30	25	$0,19143
16	40s	187.84	25	$0,19164
17	40s	193.36	25	$0,19763
18	40s	100.0	25	$0,13430
			Total:	**$3,87123**

Los costos varían directamente con el nivel de utilización de CPU. El costo total de la prueba fue de $3,46872, el máximo estimado en un intervalo de alta carga fue de $0,72316 en un

intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a valor $0,00131.

Despliegue independiente en EC2

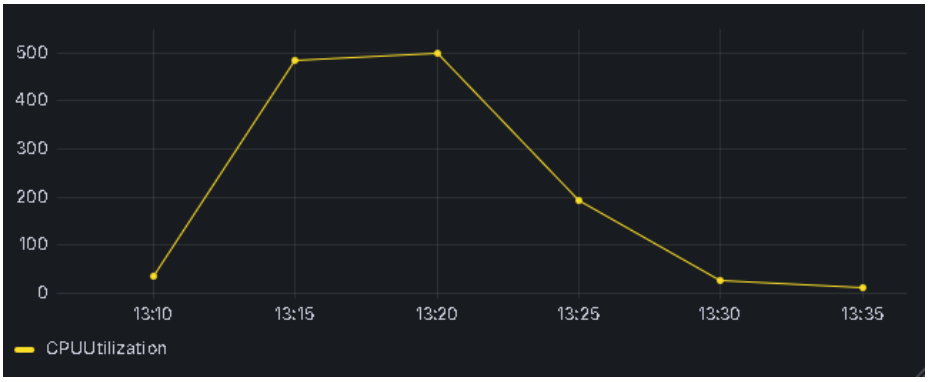


Figura 50 Visualización de CPU en Grafana de la instancia de EC2 (Aplicación 4)

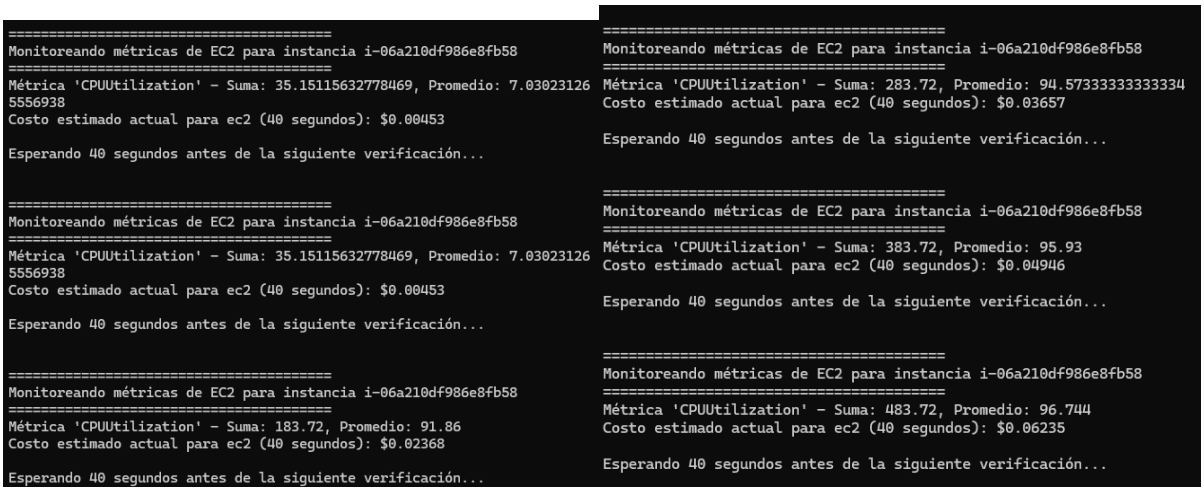


Figura 51 Monitoreo de métricas y costos en la instancia EC2 (Aplicación 4)

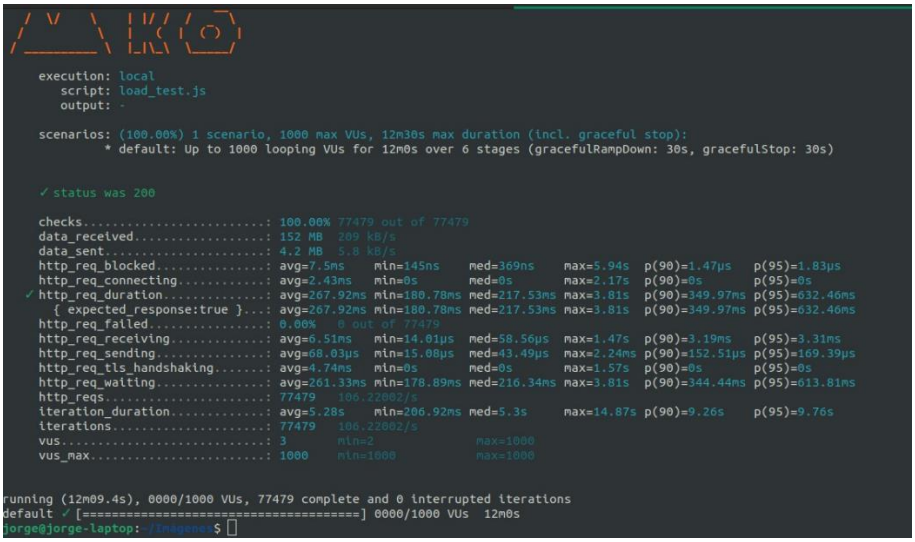


Figura 52 Rendimiento de solicitudes de la sobrecarga generada con K6 (Aplicación 4)

Se calcularon los costos estimados para la instancia EC2 en intervalos de 40 segundos. La siguiente tabla resume los datos recolectados:

Tabla 15 Resultados del monitoreo de métricas y costos de la instancia EC2 (Aplicación 4)

	Tiempo	**CPU**	**Costo (USD/s)**
1	40s	10.75	$0,00139
2	40s	16.18	$0,00209
3	40s	27.20	$0,00351
4	40s	35.15	$0,00453
5	40s	183.72	$0,23680
6	40s	283.72	$0,36570
7	40s	383.72	$0,49460
8	40s	483.72	$0,62350
9	40s	100.0	$0,12890
10	40s	200.0	$0,25780
11	40s	300.0	$0,38670
12	40s	400.0	$0,51560
13	40s	500.0	$0,64440
14	40s	176.81	$0,22790
15	40s	182.30	$0,23500
16	40s	187.84	$0,24210
17	40s	193.36	$0,24920
18	40s	27.05	$0,00349
Total:			
			$4,62182

El costo total de la prueba fue de $4,62182, el máximo estimado en un intervalo de alta carga fue de $0,64440 en un intervalo de 40 segundos, mientras que, en los momentos de menor utilización, el costo se redujo significativamente a valor $0,00139.

3.4.4.2 Escenario 2

Figura 53 Visualización de métricas en Grafana del Escenario 2 (Aplicación 4)

En Lambda se observaron picos de ejecución concurrente de hasta 215525.97 invocaciones. En Fargate la utilización de CPU varió entre 48.71% y 300% durante el proceso de cambio. En ECS La utilización de CPU mostró valores entre 143% y 300% dependiendo de la carga. El primer cambio se realizó en 1:20 y el cambio a Ec2 se realizó en el minuto 4.

```
====================================
Monitoreando el despliegue en lambda
====================================
Métrica 'Invocations' - Suma: 312.0
Métrica 'Duration' - Suma: 215525.97999900003

*** Umbral superado! ***
- Métrica: Duration
- Suma: 215525.97999900003
- Umbral: 70000

>>> Costo estimado en los últimos 40 segundos: $0.003655
>>> Costo estimado en 1 hora: $0.328970

=== Autoescalando de lambda a fargate ===
Verificando si el clúster existe...
El clúster 'deploy-uploader-cluster' ya existe. Usándolo.
Registrando la definición de tarea...
Definición de tarea registrada: deploy-uploader-task-family:5
Creando o actualizando el servicio ECS...
El servicio ya existe, actualizando...
Servicio actualizado correctamente.
Esperando a que el servicio esté estable...
El servicio 'deploy-uploader-service' está estable con 1 tareas en eje
Verificando el estado del servicio...
Servicio: deploy-uploader-service, Estado: ACTIVE, Tareas activas: 1
Obteniendo la IP pública de las tareas en ejecución...
Tu aplicación está disponible en: http://52.87.241.134:8080
Despliegue completado en fargate.
```

```
>>> Costo estimado en los últimos 40 segundos: $0.000010
>>> Costo estimado en 1 hora: $0.000909

=== Autoescalando de fargate a ec2 ===
Creando una instancia EC2 con Ubuntu 22.04...
Instancia creada. Esperando a que esté en estado 'running'...
Instancia EC2 en ejecución. ID: i-0bb2dd960591eb43c, IP Pública: 3.83.13
.229
Tu aplicación está disponible en: http://3.83.13.229:8080
Despliegue completado en ec2.

====================================
Monitoreando el despliegue en ec2
====================================
Métrica 'CPUUtilization' - Suma: 10.967093997412135

>>> Costo estimado en los últimos 40 segundos: $0.001414
>>> Costo estimado en 1 hora: $0.127218

>>> Todas las métricas de 'ec2' están dentro del rango aceptable. <<<

Endpoint Global: http://GlobalEndpointALB-1820354190.us-east-1.elb.amazo
naws.com
```

Figura 54 Monitoreo de métricas y costos del Escenario 2 (Aplicación 4)

	Tiempo	Invocaciones	Duración (ms)	Ejecuciones concurrentes	CPU	Memoria	Costo (USD/s)
Lambda							
1	40s	515	2797.12	592			$0,00015
2	40s	312	215525.97	592			$0,00366
Cambio a Fargate							
3	40s				48.71	37.71	$0,02377
4	40s				43.26	26.51	$0,02077
5	40s				117.28	26.52	$0,05415
6	40s				300.0	25	$0,45616
Cambio a EC2							
7	40s				54.25		$0,06990
8	40s				82.59		$0,01065
9	40s				77.90		$0,01004
10	40s				51.64		$0,06660
11	40s				143		$0,13490
12	40s				100.0		$0,12890
13	40s				200.0		$0,25780
14	40s				300.0		$0,38670
15	40s				55.61		$0,07170
16	40s				82.30		$0,10610
17	40s				82.31		$0,10620
18	40s				107.20		$0,01382
				Total:			**$1,92197**

El costo total acumulado para este escenario fue de $1,92197, desglosado de la siguiente manera:

- Lambda: $0,00381
- Fargate: $0,55485
- EC2: $1,36331

3.4.5 Resultados de las *p*ruebas del *m*odelo de RL

3.4.5.1 Objetivo de la Escenario 3:

El objetivo de este escenario fue probar las cuatro aplicaciones con el modelo de Reinforcement Learning entrenado para automatizar el proceso de autoescalamiento entre

los tres servicios de AWS, Lambda, Fargate y EC2, reemplazando las reglas heurísticas utilizadas en el escenario 2.

3.4.5.2 *Pruebas realizadas:*

- Primiero, se ejecutó el script de monitoreo y autoescalado adaptado para utilizar el modelo de RL.
- Inicialmente, se puedo observar que el sistema tomaba decisiones de escalado prematuramente, cambiaba de servicio de Lambda a Fargate, sin esperar a que se recopilaran métricas significativas para el cambio. Los valores de las métricas no estaban cerca a los umbrales definidos en el escenario 2.
- Para solucionar esto, se incluyó un tiempo de retardo inicial de 60 segundos después de iniciar la prueba de carga, así se permite que CloudWatch recopile métricas antes de que el modelo comience a tomar decisiones.

3.4.5.3 *Resultados obtenidos:*

- El sistema escaló correctamente de Lambda a Fargate después de que se recopilaron las tres métricas iniciales.
- Sin embargo, luego de unos minutos de monitoreo se observó un comportamiento no deseado: el modelo decidió volver a desplegar en Fargate en lugar de escalar a EC2, generando diferente endpoints y tareas en Fargate.
- Este comportamiento indica que el modelo de RL no está tomando decisiones correctas u óptimas en función de las métricas de los servicios.

3.4.5.4 *Limitaciones y problemas identificados:*

i. Métricas iniciales nulas o cercanas a cero:

El modelo de RL no está entrenado para manejar estados iniciales donde las métricas son nulas o cercanas a cero, lo que lleva al sistema a tomar decisiones de escalado prematuras o incorrectas.

ii. Falta de sintonización del modelo:

El modelo fue entrenado en un entorno de simulación que posiblemente no refleja el comportamiento real de los servicios en ejecución. Por esta razón, es posible que el modelo no esté completamente sintonizado para manejar las métricas reales de AWS.

i. **Intervalos de monitoreo:**

El intervalo de monitoreo (CHECK_INTERVAL) puede ser demasiado corto, lo que no permite que el modelo observe cambios significativos en las métricas antes de tomar decisiones.

i. **Falta de validación en un entorno real:**

El modelo no fue validado varias veces en un entorno real antes de su implementación, lo que podría explicar el comportamiento no deseado que está presentando.

3.4.5.5 Mejoras Propuestas

i. **Entrenamiento del modelo con datos reales:**

- Recopilar datos históricos de métricas y costos de un entorno real para reentrenar el modelo de RL.
- Garantizar que el modelo esté entrenado para manejar estados iniciales con métricas nulas o cercanas a cero.

ii. **Ajuste de hiperparámetros:**

- Afinar los hiperparámetros del modelo de RL (por ejemplo, la función de recompensa) para alinearlos mejor con los objetivos del proyecto, como minimizar costos y maximizar rendimiento.

iii. **Validación en un entorno controlado:**

- Realizar pruebas exhaustivas en un entorno controlado antes de llevar el modelo a producción.
- Validar que el modelo tome decisiones coherentes y óptimas bajo diversas cargas de trabajo.

iv. **Mejoras en el monitoreo:**

- Ampliar el intervalo de monitoreo (CHECK_INTERVAL) para que el modelo pueda detectar cambios significativos en las métricas antes de actuar.
- Implementar un mecanismo que asegure que las métricas estén disponibles y sean relevantes antes de tomar decisiones.

v. Manejo de errores y lógica de respaldo:

- Incorporar una lógica de respaldo que permita al sistema recurrir a reglas heurísticas si el modelo de RL toma decisiones incorrectas o no está disponible.

Aunque el escenario 3 no se completó en su totalidad, las pruebas iniciales demostraron que el modelo de RL tiene el potencial de automatizar el proceso de autoescalamiento de manera más inteligente y eficiente que las reglas heurísticas. No obstante, los problemas identificados sugieren que se requieren ajustes adicionales para que el modelo opere de manera efectiva en un entorno real.

Los resultados obtenidos en los escenarios 1 y 2 ofrecen una base sólida para comparar la eficiencia y los costos del sistema bajo diferentes enfoques. A pesar de los desafíos encontrados en el escenario 3, este proyecto evidencia que el uso de técnicas de Reinforcement Learning puede ser una solución prometedora para la automatización del escalamiento en la nube.

3.5 Análisis comparativo

3.5.1 Comparación entre los Escenarios

Escenario 1: Despliegues Estáticos:

- En este escenario, las aplicaciones se desplegaron de manera estática en cada servicio Lambda, Fargate y EC2 sin realizar el autoescalamiento, para verificar como se desempeña cada aplicación en los tres servicios.
- Los resultados mostraron que cada servicio tiene un rendimiento y costos diferentes bajo carga, lo que permitió establecer una línea base para comparar con los

escenarios posteriores. El servicio que más fallas presentó fue Lambda y también fue el más costoso en las cuatro aplicaciones probadas.

Escenario 2: Autoescalamiento con Reglas Heurísticas:

- En este escenario, se implementó un sistema de autoescalamiento basado en reglas heurísticas. La Aplicación 1 que es de complejidad más baja, fue la más económica con un costo total de la prueba de $0,09, mientras que la Aplicación 4 que tiene mayor complejidad tuvo un costo total de $1,92.
- El sistema escaló correctamente entre servicios según las reglas definidas, fue el escenario que mostró mayor eficiencia en términos de costos y rendimiento.

Escenario 3: Autoescalamiento con Modelo de RL:

- Aunque no se completó el escenario 3, las pruebas iniciales mostraron que el modelo de RL tiene el potencial de automatizar el escalamiento de manera más inteligente.
- Sin embargo, los problemas identificados indican que se necesitan más ajustes adicionales para que el modelo funcione correctamente en un entorno real.

3.5.2 Análisis comparativo de la Aplicación 1

Tabla 16 Resumen comparativo de métricas clave entre los tres escenarios

Métrica	Escenario 1			Escenario 2
	Lambda	**Fargate**	**Ec2**	**Cambio Dinámico**
Costo de la prueba	$0,28580	$0,17846	$0,14234	$0,09326
Solicitudes exitosas	69,00%	68,86%	99,97%	99,99%
Tasa de fallos	30,81%	31,13%	0,03%	0,01%

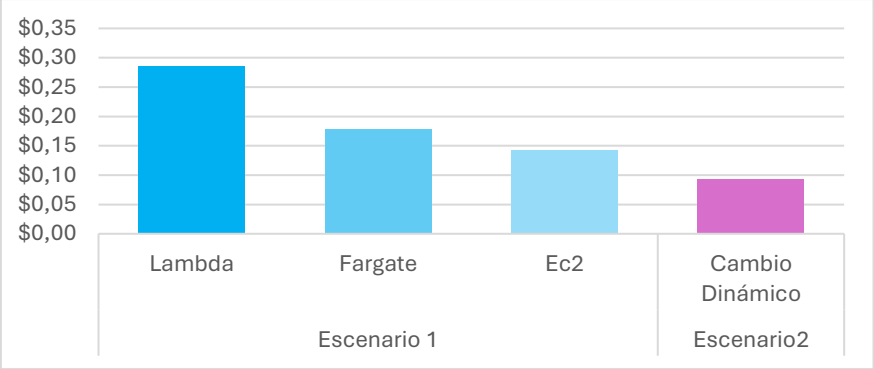


Figura 55 Gráfica de comparación de precios de la Aplicación 1

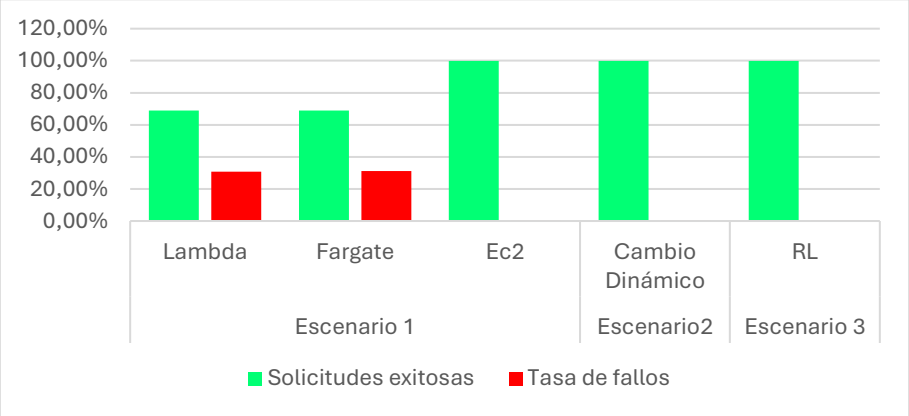


Figura 56 Visualización de solicitudes exitosas y tasas de fallos en los tres escenarios

- **Análisis de Costos**

Lambda presentó el mayor costo por segundo $0.28580 lo que lo hace poco rentable para cargas pesadas. Fargate y EC2 mostraron mejores costos, siendo EC2 el más económico en el escenario $0,14234. En el Escenario 2 Redujo significativamente los costos totales a $0.09326, gracias a la optimización basada en la carga. Aunque el escenario 3 tuvo un costo ligeramente mayor que el Escenario 2 $0.14, sigue siendo más eficiente que el Escenario 1.

- **Análisis de Solicitudes Exitosas**

En el Escenario 1, EC2 alcanzó una tasa de éxito del 99.97%, mientras que Lambda y Fargate tuvieron tasas más bajas 69% y 68.86%, respectivamente debido a su modelo de facturación y limitaciones en picos de carga. En el Escenarios 2, la tasa de éxito fue consistente y alta 99.97%, indicando que la estrategia de cambio dinámico mejoró significativamente el rendimiento.

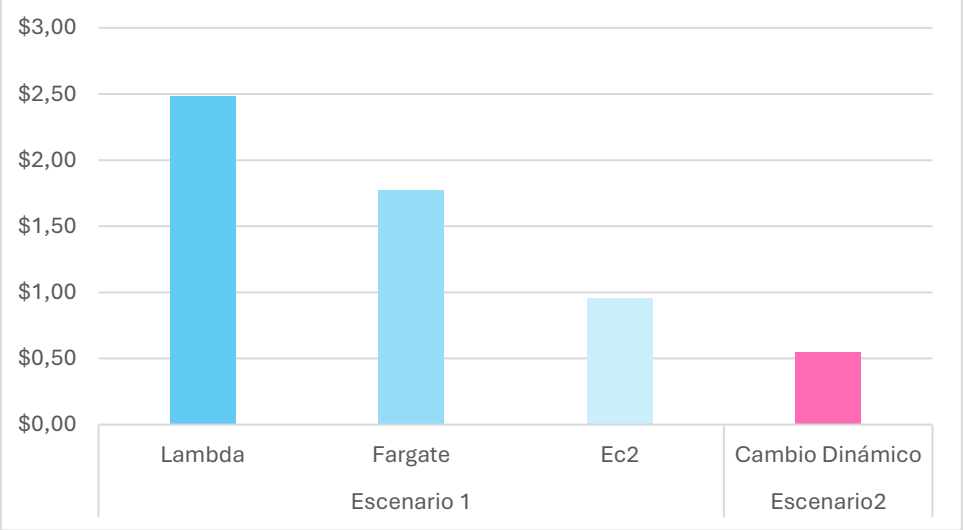
- **Análisis de la Tasa de Fallos**

Lambda y Fargate presentaron tasas de fallos del 30.81% y 31.13%, respectivamente, debido a la saturación en picos de carga. EC2 mantuvo una tasa de fallos mínima del 0.02%, siendo el despliegue más robusto.

3.5.3 Análisis comparativo de la Aplicación 3

Tabla 17 Resumen comparativo de métricas clave entre los tres escenarios

Métrica	Escenario 1			Escenario 2
	Lambda	**Fargate**	**Ec2**	**Cambio Dinámico**
Costo de la prueba	**$2,48**	**$1,78**	**$0,95**	**$0,55**
Solicitudes exitosas	56,00%	68,86%	99,97%	99,99%
Tasa de fallos	44%	31,13%	0,03%	0,01%



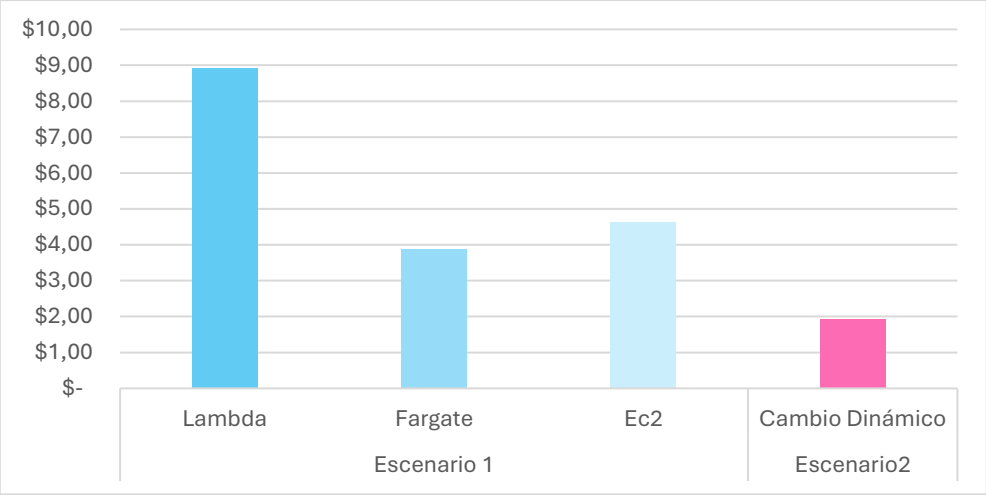
- **Análisis de Costos**

Lambda presentó el mayor costo por segundo $2,48 lo que lo hace poco rentable para cargas pesadas. Fargate y EC2 mostraron mejores costos, siendo Ec2 el más económico en el escenario $0,95. En el Escenario 2 Redujo significativamente los costos totales a $1,92, gracias a la optimización basada en la carga.

3.5.4 Análisis comparativo de la Aplicación 4

Tabla 18 Resumen comparativo de métricas clave entre los tres escenarios

Métrica	Escenario 1			Escenario 2
	Lambda	**Fargate**	**Ec2**	**Cambio Dinámico**
Costo de la prueba	**$8,91**	**$ 3,87**	**$4,62**	**$1,92**
Solicitudes exitosas	69,00%	68,86%	99,97%	99,99%
Tasa de fallos	30,81%	31,13%	0,03%	0,01%



- **Análisis de Costos**

Lambda presentó el mayor costo por segundo $8,91 lo que lo hace poco rentable para cargas pesadas. Fargate y EC2 mostraron mejores costos, siendo fargate el más económico en el escenario $ 3,87. En el Escenario 2 Redujo significativamente los costos totales a $1,92, gracias a la optimización basada en la carga.

Capítulo 4

4 Conclusiones y recomendaciones

4.1 Conclusiones

En el Escenario 1, Lambda destacó por su escalabilidad automática, pero presentó los costos más elevados en las cuatro aplicaciones probadas, especialmente en cargas sostenidas y complejas como la Aplicación 4, en donde obtuvo el costo de prueba más elevado. Mientras que Fargate ofreció un equilibrio entre escalabilidad y costo, siendo más económico que Lambda, pero requirió configuraciones adicionales. Los despliegues en EC2 mostraron desempeños más adecuados, pero requieren configuraciones más complejas y costos fijos mayores. EC2 tuvo el menor precio en 3 de las 4 aplicaciones.

El Escenario 2, proporcionó una mejora significativa en la adaptabilidad del sistema, reduciendo los costos en un 67% en comparación con los escenarios independientes.

Las aplicaciones con mayores requerimientos de procesamiento, como la Aplicación 4, incurrieron en costos más elevados y tiempos de respuesta más largos en todos los escenarios. Además, durante las pruebas de sobrecarga, se observó un 40% de fallas en las peticiones, resaltando la necesidad de una arquitectura más robusta para manejar dichas cargas.

Las aplicaciones con patrones de carga estables, como la Aplicación 1 y 2, se beneficiaron del uso de EC2 por su configuración fija, obtenido así precios bajos y baja utilización de recursos.

4.2 Recomendaciones

- Se recomienda implementar un enfoque híbrido que combine entrenamiento offline con datos simulados y online con métricas en tiempo real para el modelo de RL.
- Es recomendable utilizar EC2 para aplicaciones con patrones de carga predecibles, como la Aplicación 1 y 2. Esto ofrece una solución de bajo costo con configuraciones fijas que son fáciles de mantener.

- Integrar herramientas de monitoreo avanzadas como AWS CloudWatch con métricas personalizadas para detectar anomalías en el rendimiento y cuellos de botellas.
- Es recomendable analizar los costos de mantener servicios en Lambda frente a migraciones a instancias en EC2 según los patrones de uso.
- Reducir el uso de Lambda en aplicaciones con alta demanda sostenida, como la Aplicación 4, ya que sus costos son significativamente más altos en estos casos.
- Para experimentos futuros se recomienda extender las pruebas de las aplicaciones a 15 o 18 minutos para observar mejor el comportamiento de los scripts de cambio dinámico.

5 Referencias

[1] C. Cota Acevedo y D. R. Herrera Quintero, «Estado del arte sobre la computación en la nube (cloud computing),» 2021. [En línea]. Available: https://repositorio.utb.edu.co/handle/20.500.12585/1607#page=1.

[2] G. Montilla Tomás, «Migración de una aplicación on-premise a la nube con Amazon Web Services,» 2023. [En línea]. Available: http://hdl.handle.net/10251/201306.

[3] S. Suárez Conde, «Open Data Cube en la nube,» 28 Junio 2022. [En línea]. Available: http://hdl.handle.net/1992/59349.

[4] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann y C. Abad, «Serverless Applications: Why, When, and How?,» 2021. [En línea]. Available: https://ieeexplore.ieee.org/document/9190031.

[5] Amazon Web Services, «Precios de AWS Lambda,» AWS, 2023. [En línea]. Available: https://aws.amazon.com/es/lambda/pricing/. [Último acceso: 22 Octubre 2024].

[6] Amazon Web Services, «Precios de AWS Fargate,» AWS, 2023. [En línea]. Available: https://aws.amazon.com/es/fargate/pricing/. [Último acceso: 22 Octubre 2024].

[7] Amazon Web Services, «Precios de Amazon EC2,» 2023. [En línea]. Available: https://aws.amazon.com/es/ec2/pricing/. [Último acceso: 23 Octubre 2024].

[8] E. F. Boza, C. L. Abad, M. Villavicencio, S. Quimba y J. A. Plaza, «Reserved, on demand or serverless: Model-based simulations for cloud budget planning,» 2017. [En línea]. Available: https://ieeexplore.ieee.org/document/8247460.

[9] Accenture, «Investigación Cloud Outcomes: Cómo navegar las barreras para maximizar el valor de la nube,» 2020. [En línea]. Available: https://www.accenture.com/content/dam/accenture/final/a-com-migration/manual/r3/pdf/pdf-141/Accenture-CLOUD-Outcomes-Executive-Summary-v8-ES.pdf. [Último acceso: 12 Octubre 2024].

[10] ATLASSIAN y K. Zettler, «¿Qué es la computación en la nube? Visión general de la nube,» 2024. [En línea]. Available: https://www.atlassian.com/es/microservices/cloud-computing.

[11] J. Del Vecchio, F. Paternina y C. Miranda, «Cloud computing: a model for the development of enterprises,» 2015. [En línea]. Available: http://www.scielo.org.co/scielo.php?pid=S1692-82612015000200010&script=sci_arttext.

[12] F. Rueda, «¿Qué es la computación en la nube?,» [En línea]. Available: https://acis.org.co/portal/Revista/112/tres.pdf.

[13] Google Cloud, «¿Qué es la computación sin servidores?,» 18 Octubre 2024. [En línea]. Available: https://cloud.google.com/discover/what-is-serverless-computing?hl=es-419.

[14] N. R. Rodríguez, H. Atencio, M. Gómez y L. Parra, «Análisis de ejecución múltiple de Funciones Serverless en AWS,» 2021. [En línea]. Available: https://sedici.unlp.edu.ar/handle/10915/130316.

[15] Amazon Web Services, «¿Qué es la computación sin servidor?,» AWS, 2023. [En línea]. Available: https://aws.amazon.com/es/what-is/serverless-computing/. [Último acceso: 20 Octubre 2024].

[16] IBM, S. Susnjara y I. Smalley, «What is Serverless?,» 10 Junio 2024. [En línea]. Available: https://www.ibm.com/mx-es/topics/serverless. [Último acceso: 19 Octubre 2024].

[17] J. Salazar Argonza, «Computación sin servidor,» TIES, Revista de Tecnología e Innovación en Educación Superior, 3 Abril 2021. [En línea]. Available: https://www.ties.unam.mx/num03/pdf/Computacion_sin_servidor.pdf. [Último acceso: 19 Octubre 2024].

[18] Amazon Web Services, «AWS Lambda,» AWS, 2023. [En línea]. Available: https://aws.amazon.com/es/lambda/?p=pm&c=la&z=4. [Último acceso: 18 Octubre 2024].

[19] Amazon Web Services, «Servicios de computación sin servidor: AWS Lambda,» AWS, 2023. [En línea]. Available: https://sedici.unlp.edu.ar/handle/10915/130316. [Último acceso: 18 Octubre 2024].

[20] Amazon Web Services, «AWS Fargate: Cómputo sin servidor para contenedores,» 2023. [En línea]. Available: https://aws.amazon.com/es/fargate/. [Último acceso: 18 Octubre 2024].

[21] AWS, «Guía de usuario: ¿Qué es Amazon EC2?,» 2024. [En línea]. Available: https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html.

[22] Amazon Web Services, «Introducción a Amazon EC2,» AWS, 2024. [En línea]. Available: https://aws.amazon.com/es/ec2/getting-started/.

[23] Innovacion Digital 360, «Sistemas expertos, guía completa: para qué sirven y clasificación,» 17 Septiembre 2024. [En línea]. Available: https://www.innovaciondigital360.com/i-a/sistemas-expertos-que-son-su-clasificacion-como-funcionan-y-para-que-se-utilizan/.

[24] Amazon Web Services, «¿Qué es el aprendizaje mediante refuerzo?,» AWS, 2024. [En línea]. Available: https://aws.amazon.com/what-is/reinforcement-learning/.

[25] J. Murel, «¿Qué es el aprendizaje por refuerzo?,» IBM, 25 Marzo 2024. [En línea]. Available: https://www.ibm.com/think/topics/reinforcement-learning.

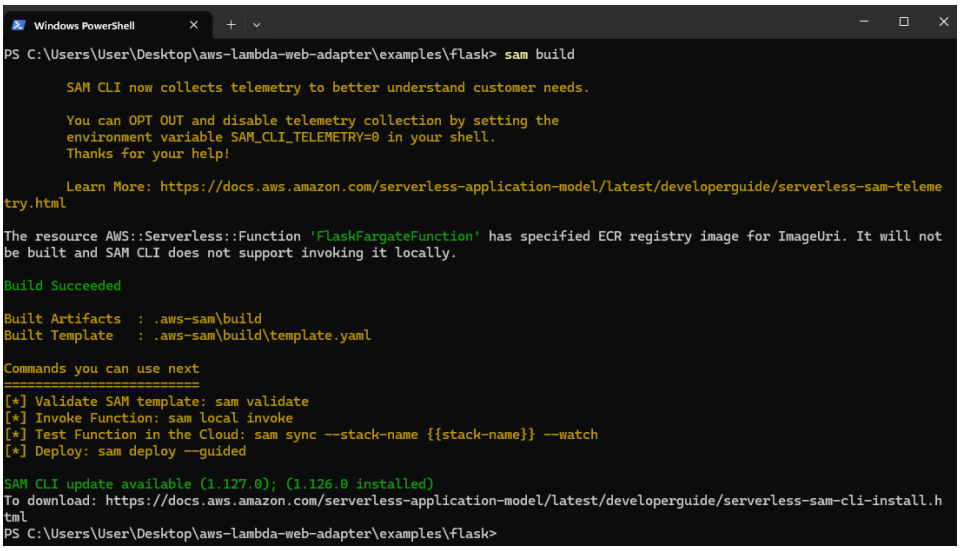
[26] R. S. Sutton y A. G. Barto, «Reinforcement Learning: An Introduction,» 2015. [En línea]. Available: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

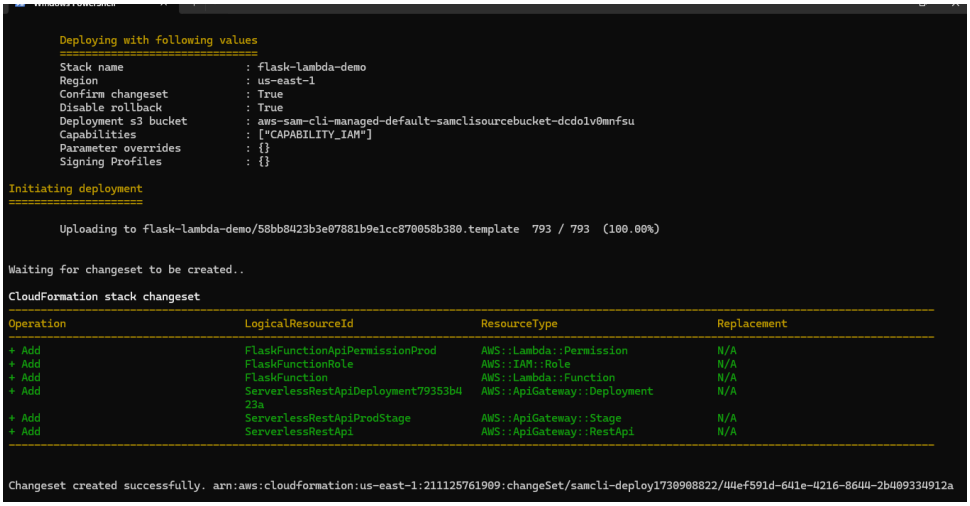
[27] H. Qiu, S. S. Banerjee, S. Jha y Z. T. Kalbarczyk, «FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices,» 4 Noviembre 2020. [En línea]. Available: https://www.usenix.org/system/files/osdi20-qiu.pdf.

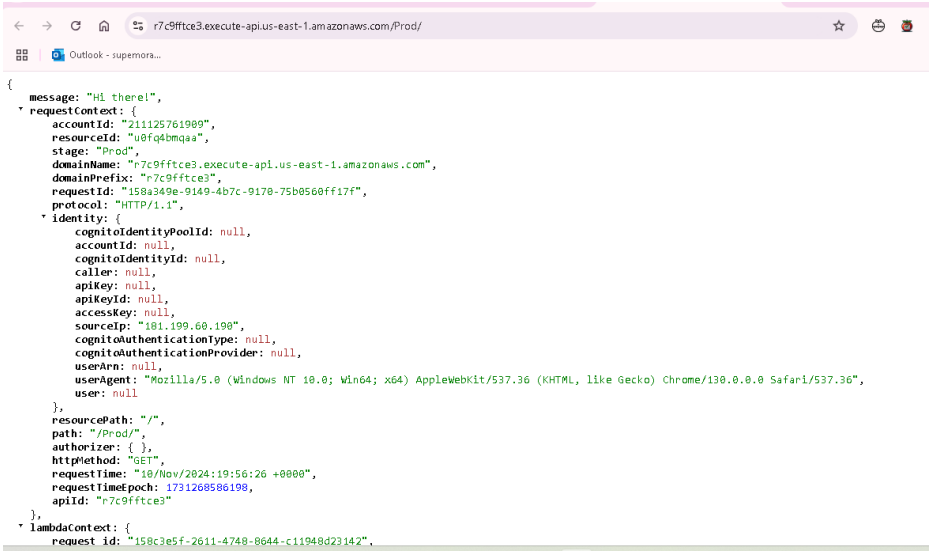
[28] H. Qiu, W. Mao, C. Wang, H. Franke, Kalbarczyk y T. Başar, «AWARE: Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems,» 10 Julio 2023. [En línea]. Available: https://www.usenix.org/system/files/atc23-qiu-haoran.pdf.

Apéndice A

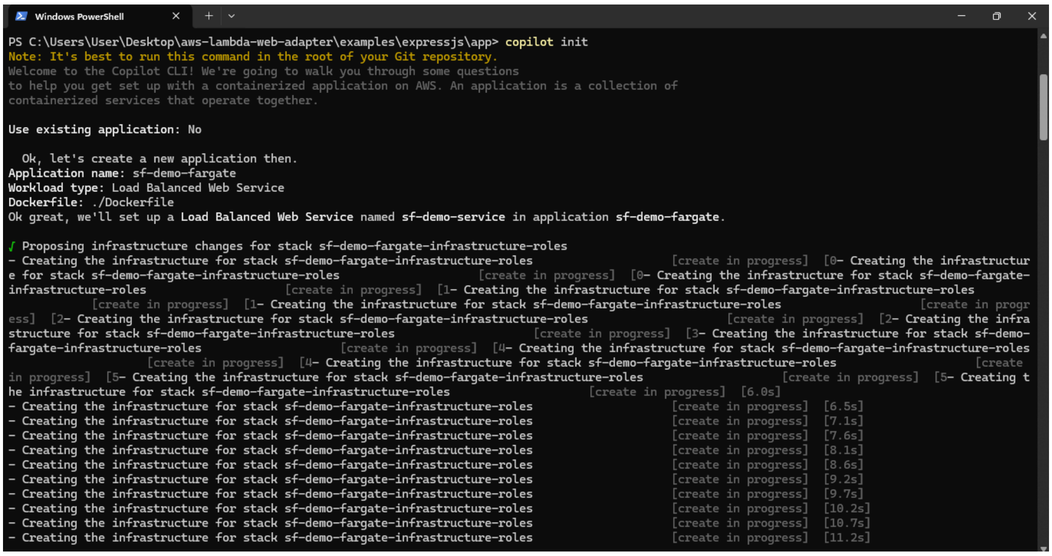
1.1 Despliegue inicial de Lambda con SAM

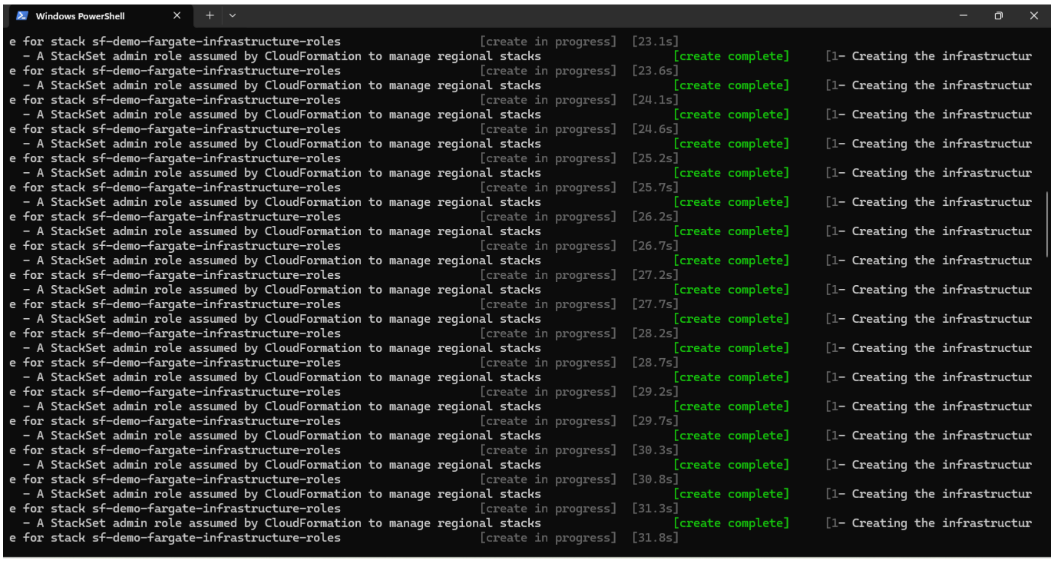


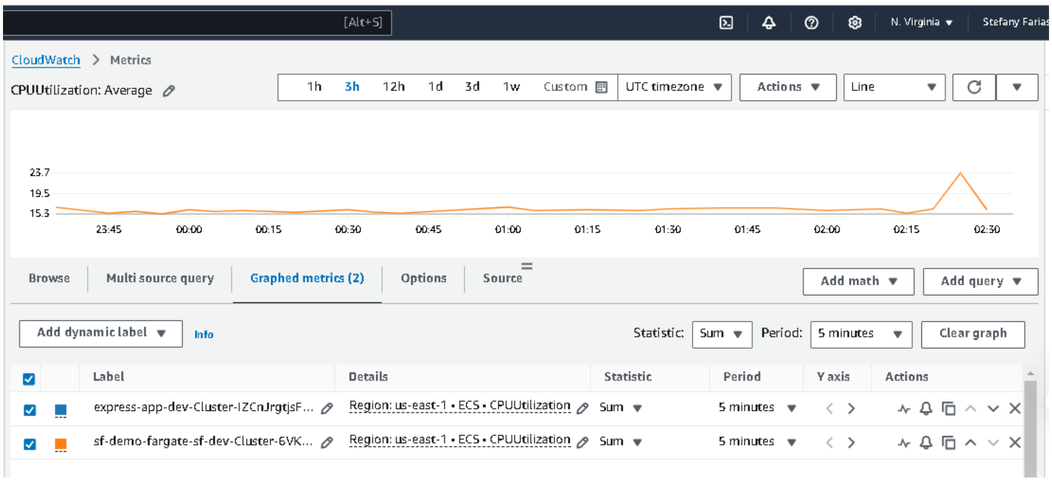


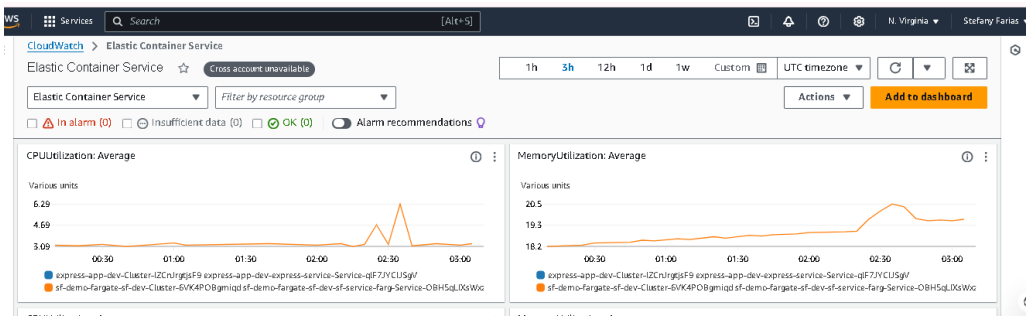


2.1 Despliegue inicial de Fragate con Copilot

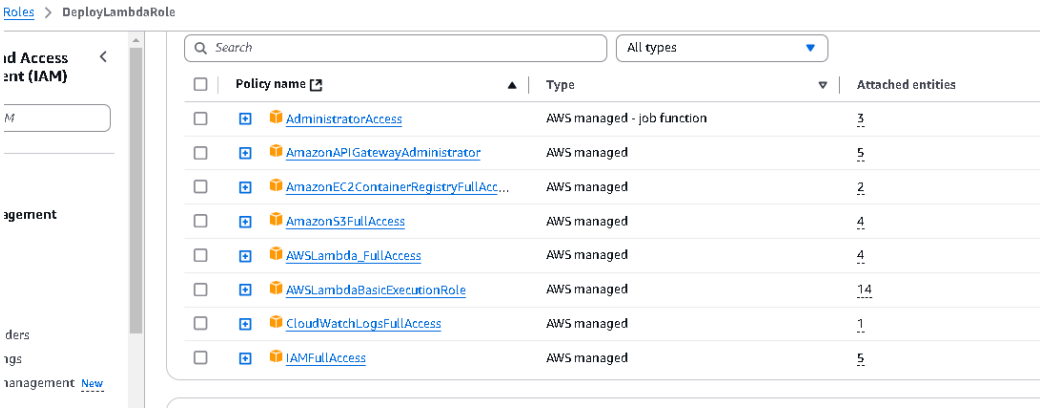


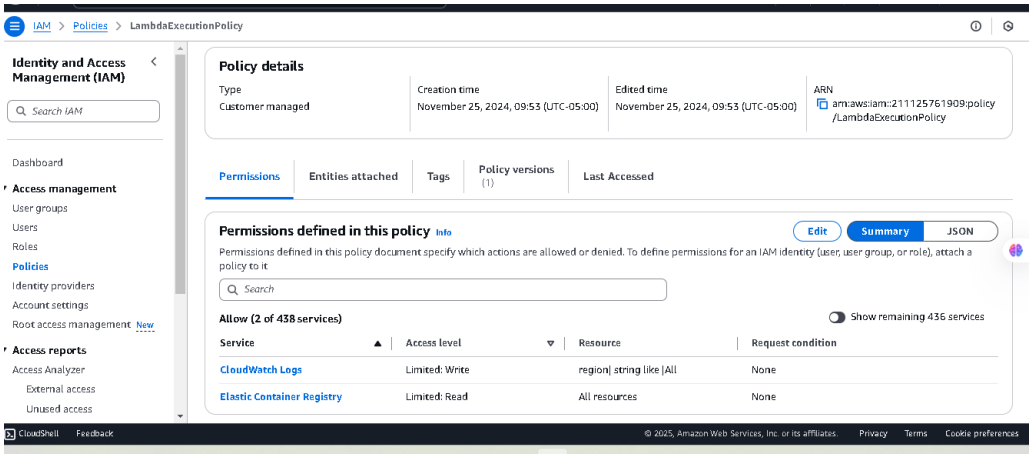


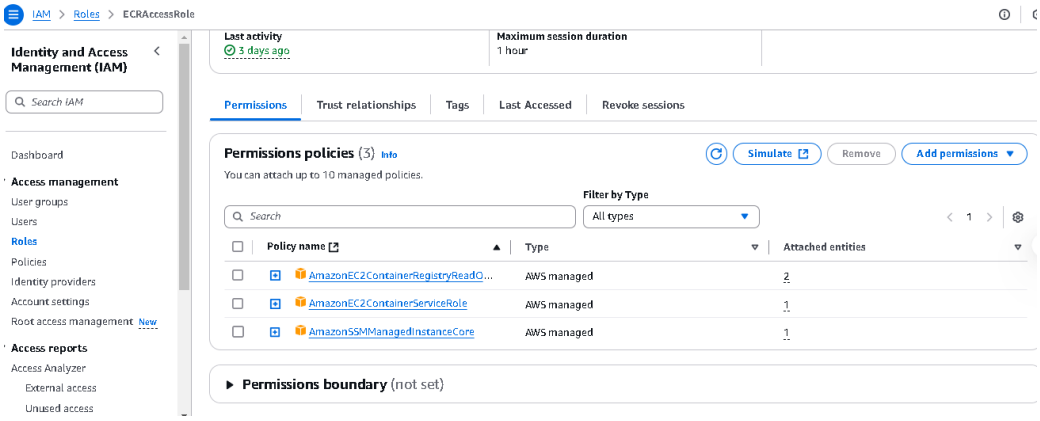


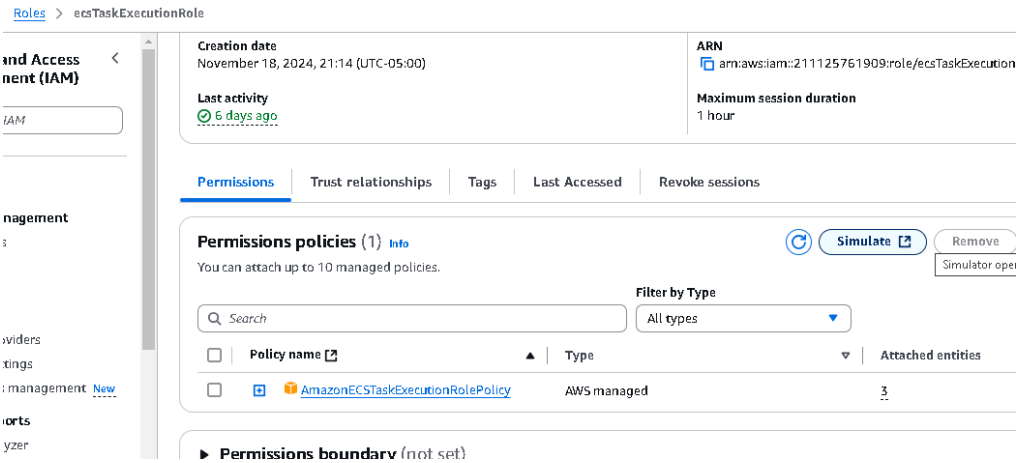


3.1 Roles y usuarios





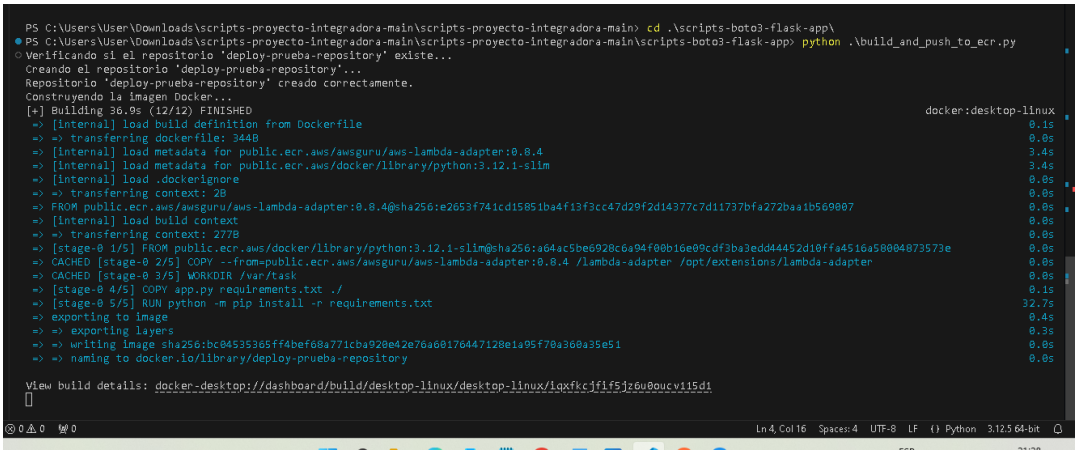




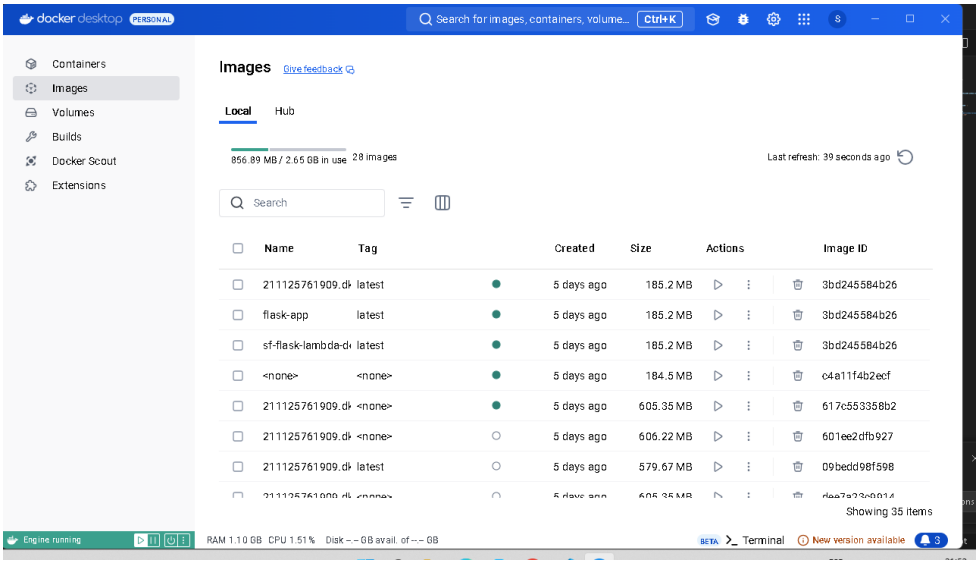
4.1 Creación y despliegue de la imagen de Docker en ECR

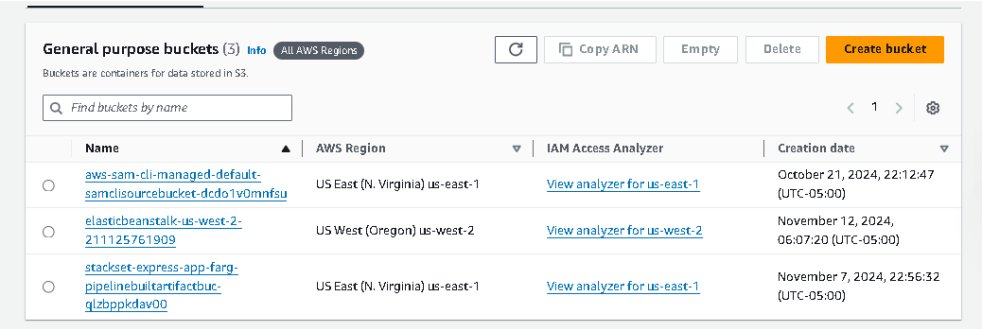
prueba3 > Dockerfile

```
1  FROM public.ecr.aws/docker/library/python:3.12.1-slim
2  COPY --from=public.ecr.aws/awsguru/aws-lambda-adapter:0.8.4 /lambda-adapter /opt/extensions/lambda-adapter
3  WORKDIR /var/task
4  COPY app.py requirements.txt ./
5  RUN python -m pip install -r requirements.txt
6  CMD ["gunicorn", "-b=:8080", "-w=1", "app:app"]
7
```

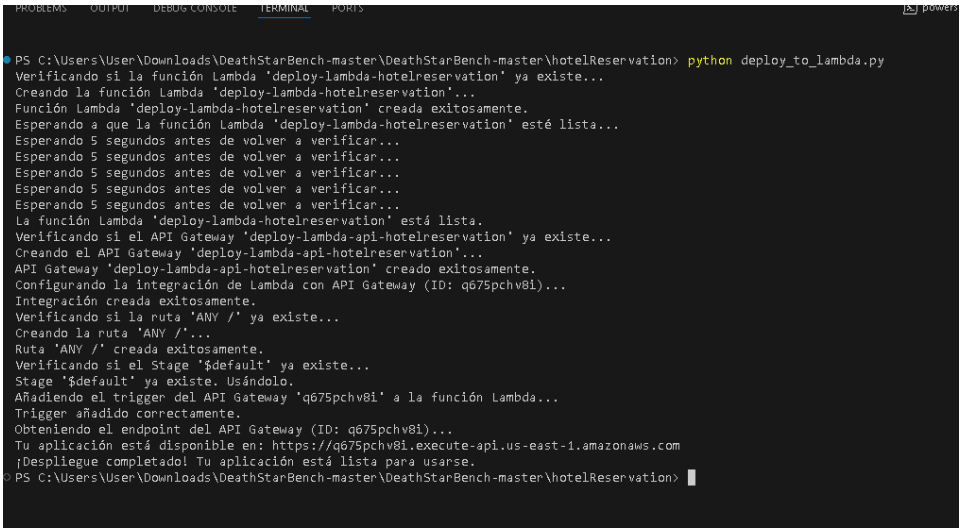


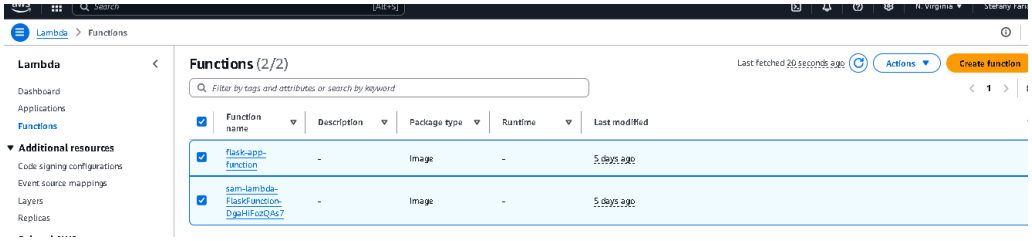

```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/iqxfkcjfif5jz6u0oucv115d1

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
Etiquetando la imagen Docker...
Iniciando sesión en Amazon ECR...
Login Succeeded
Subiendo la imagen Docker a Amazon ECR...
The push refers to repository [211125761909.dkr.ecr.us-east-1.amazonaws.com/deploy-prueba-repository]
d7bcd30bd597: Pushed
f69ea8f59a53: Pushed
c784cdf1825c: Pushed
c0a514c19cf4: Pushed
6fdc32920d25: Pushed
41aaeb647441: Pushed
95ac61d1a52e: Pushed
da5d55102092: Pushed
fb1bd2fc5282: Pushed
latest: digest: sha256:3747658a3e25bbf50eaea17f6554bcf60011e5ac516b01a3190cf3f24f5a8042 size: 2206
Imagen Docker subida correctamente: 211125761909.dkr.ecr.us-east-1.amazonaws.com/deploy-prueba-repository:latest
```



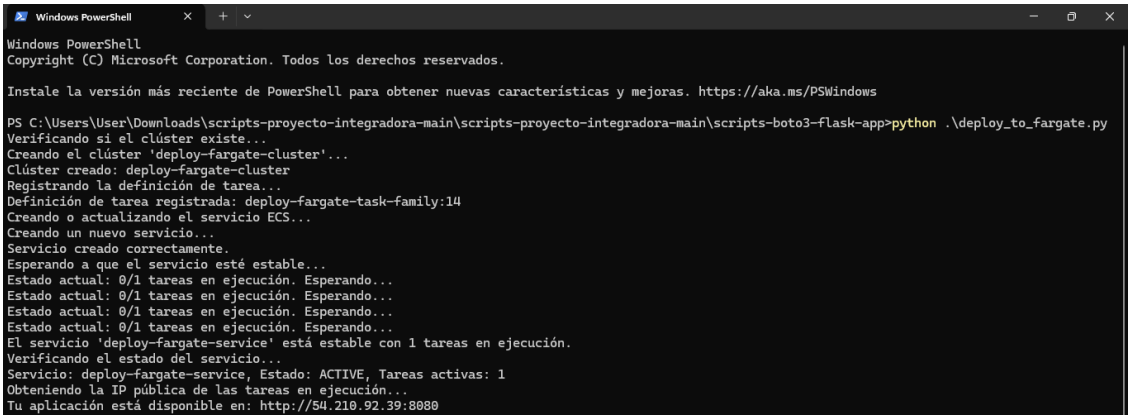


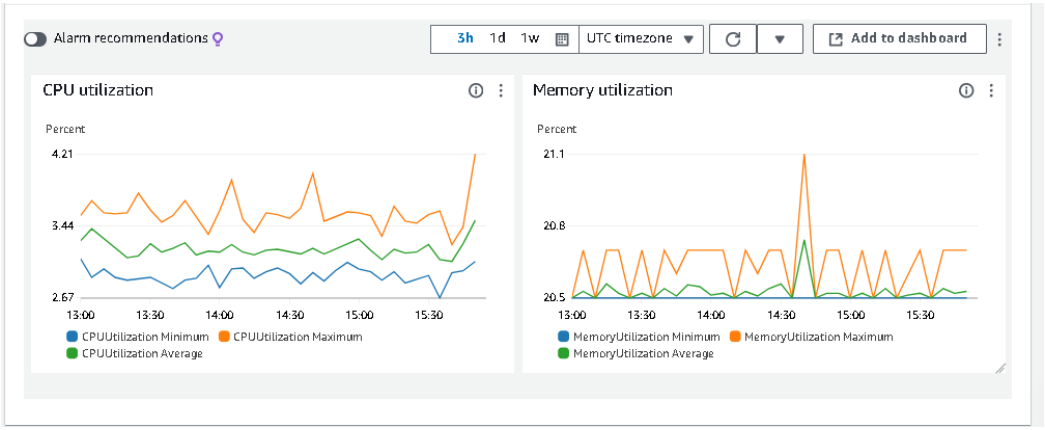
5.1 Despliegue en Lambda con script de boto3

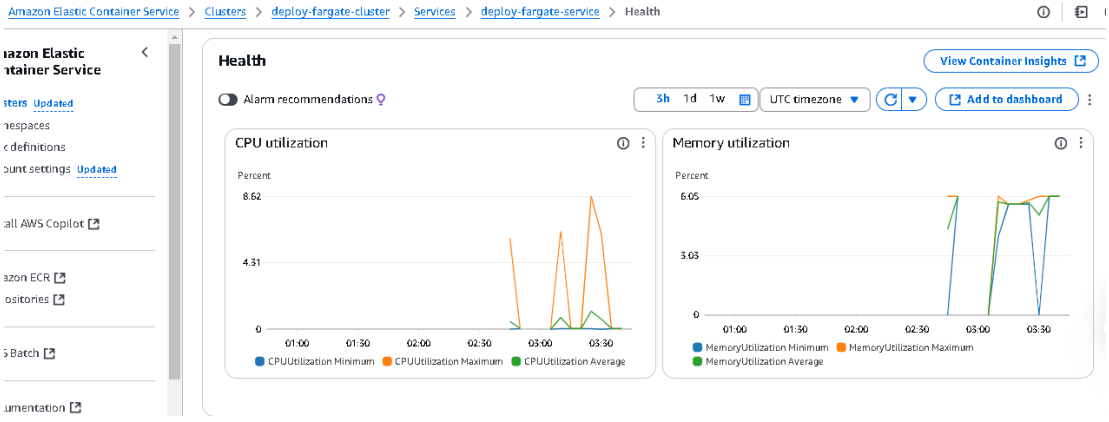




6.1 Despliegue en Fargate con script de boto3

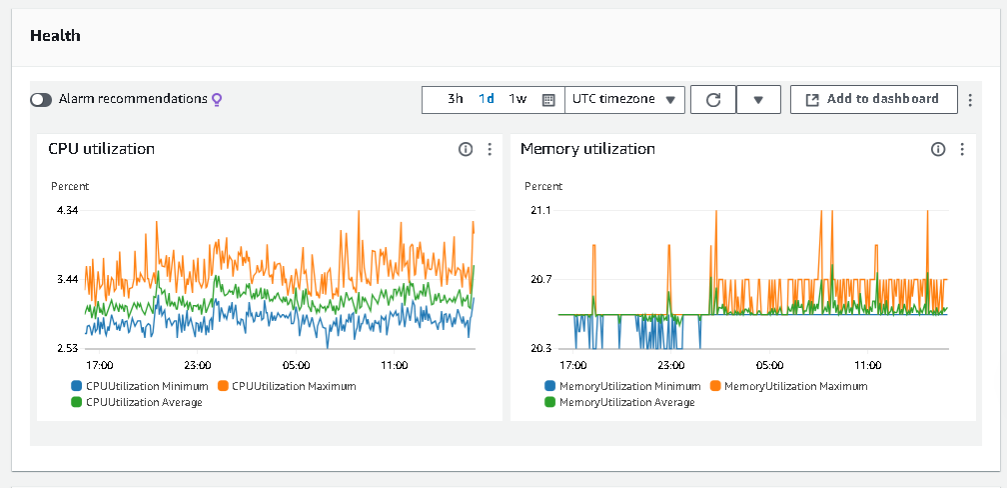






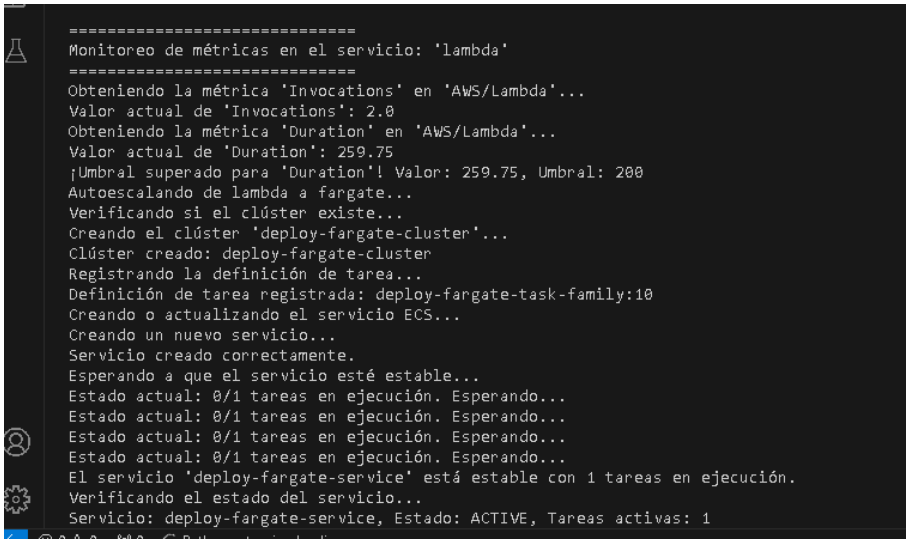
7.1 Despliegue en EC2 con script de boto3

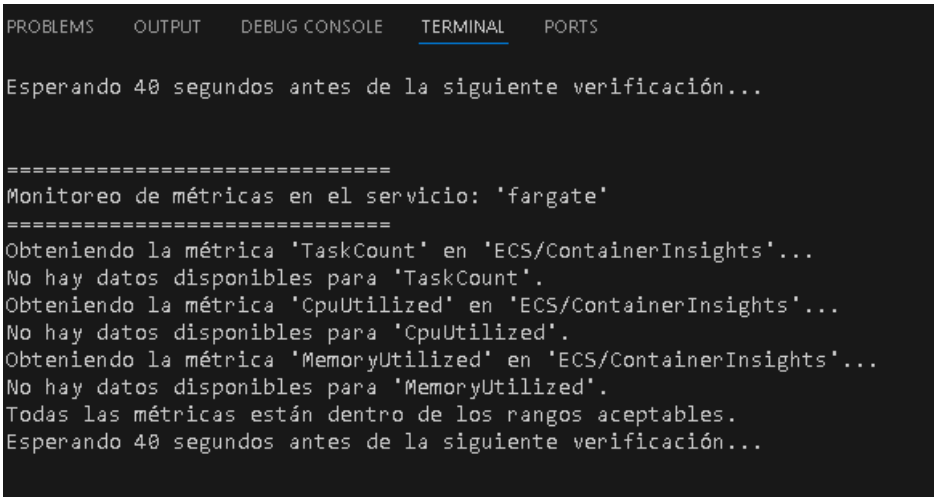
```
PS C:\Users\User\Downloads\scripts-proyecto-integradora-main\scripts-proyecto-integradora-main\scripts-boto3-flask-app>python .\deploy_to_ec2.py
Creando una instancia EC2 con Ubuntu 22.04...
Instancia creada. Esperando a que esté en estado 'running'...
Instancia EC2 en ejecución. ID: i-0f08d030faf5a11f5, IP Pública: 3.91.79.155
Tu aplicación está disponible en: http://3.91.79.155:8080
PS C:\Users\User\Downloads\scripts-proyecto-integradora-main\scripts-proyecto-integradora-main\scripts-boto3-flask-app>
```

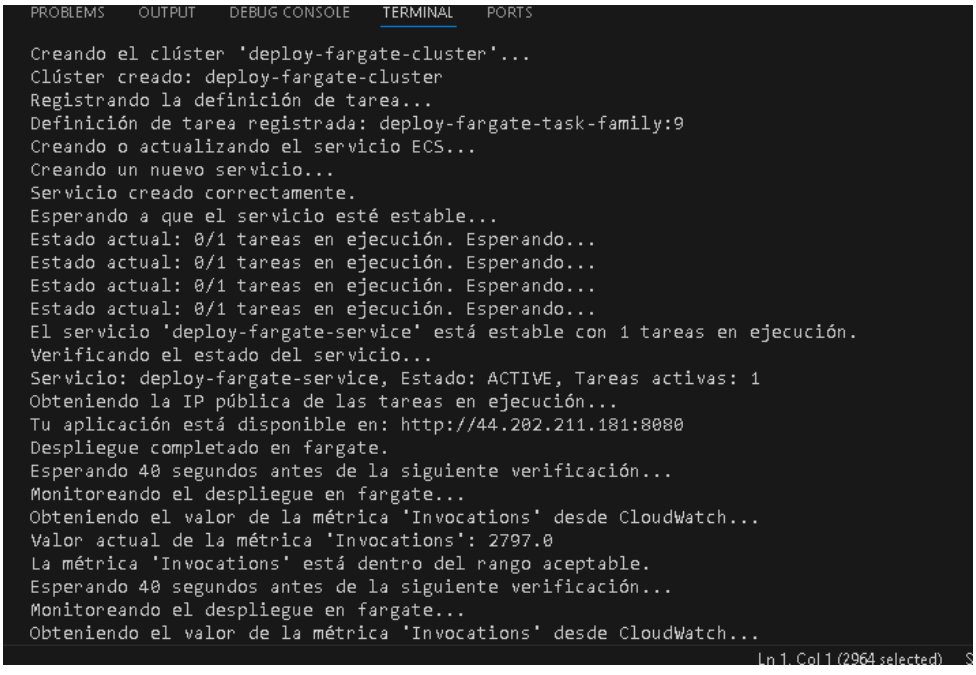


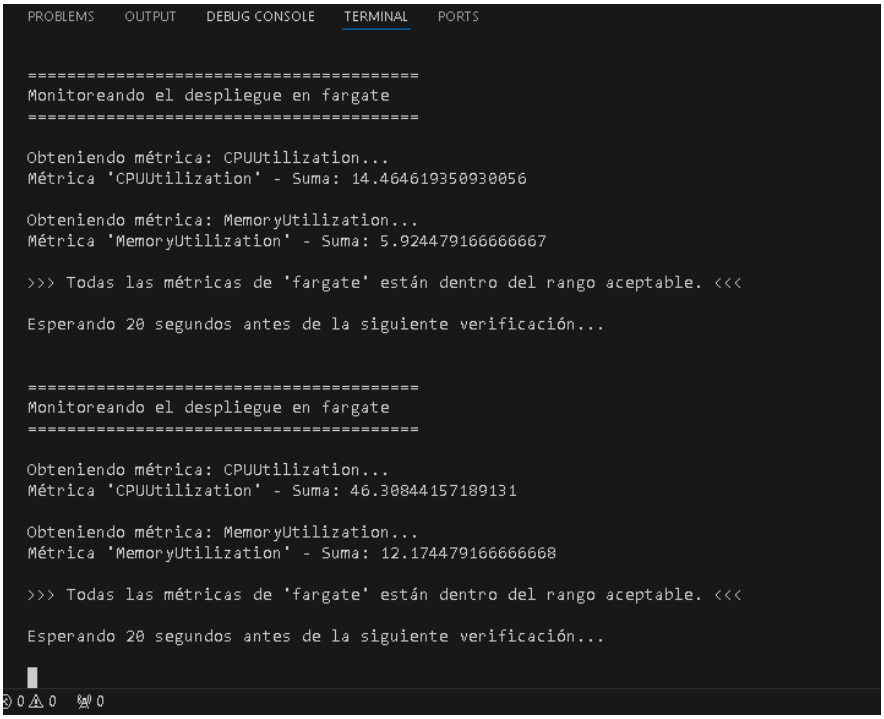
8.1 Script de cambio dinámico

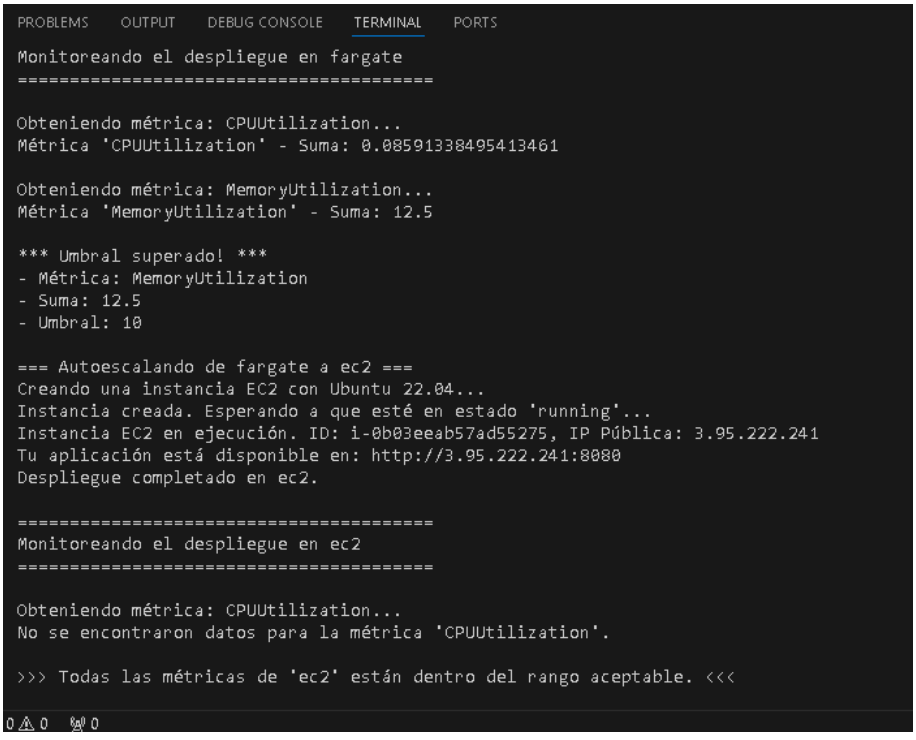
```
========================================
Monitoreando métricas de lambda
========================================
Métrica 'Invocations' - Suma: 29910.0
Métrica 'Duration' - Suma: 84350.7999378704
Métrica 'ConcurrentExecutions' - Suma: 145400.0
Costo estimado actual para lambda (40 segundos): $0.00739
Costo estimado extrapolado para lambda (1 hora): $0.66493

Esperando 40 segundos antes de la siguiente verificación...


========================================
Monitoreando métricas de lambda
========================================
Métrica 'Invocations' - Suma: 2869.0
Métrica 'Duration' - Suma: 7510.779999110004
Métrica 'ConcurrentExecutions' - Suma: 6464.0
Costo estimado actual para lambda (40 segundos): $0.00070
Costo estimado extrapolado para lambda (1 hora): $0.06291

Esperando 40 segundos antes de la siguiente verificación...


========================================
Monitoreando métricas de lambda
========================================
Métrica 'Invocations' - Suma: 2869.0
Métrica 'Duration' - Suma: 7510.779999110004
Métrica 'ConcurrentExecutions' - Suma: 6464.0
Costo estimado actual para lambda (40 segundos): $0.00070
Costo estimado extrapolado para lambda (1 hora): $0.06291
```


==========
Monitoreo de métricas en el servicio: 'lambda'
==========
Obteniendo la métrica 'Invocations' en 'AWS/Lambda'...
Valor actual de 'Invocations': 2.0
Obteniendo la métrica 'Duration' en 'AWS/Lambda'...
Valor actual de 'Duration': 259.75
¡Umbral superado para 'Duration'! Valor: 259.75, Umbral: 200
Autoescalando de lambda a fargate...
Verificando si el clúster existe...
Creando el clúster 'deploy-fargate-cluster'...
Clúster creado: deploy-fargate-cluster
Registrando la definición de tarea...
Definición de tarea registrada: deploy-fargate-task-family:10
Creando o actualizando el servicio ECS...
Creando un nuevo servicio...
Servicio creado correctamente.
Esperando a que el servicio esté estable...
Estado actual: 0/1 tareas en ejecución. Esperando...
Estado actual: 0/1 tareas en ejecución. Esperando...
Estado actual: 0/1 tareas en ejecución. Esperando...
Estado actual: 0/1 tareas en ejecución. Esperando...
El servicio 'deploy-fargate-service' está estable con 1 tareas en ejecución.
Verificando el estado del servicio...
Servicio: deploy-fargate-service, Estado: ACTIVE, Tareas activas: 1


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Esperando 40 segundos antes de la siguiente verificación...
==========
Monitoreo de métricas en el servicio: 'fargate'
==========
Obteniendo la métrica 'TaskCount' en 'ECS/ContainerInsights'...
No hay datos disponibles para 'TaskCount'.
Obteniendo la métrica 'CpuUtilized' en 'ECS/ContainerInsights'...
No hay datos disponibles para 'CpuUtilized'.
Obteniendo la métrica 'MemoryUtilized' en 'ECS/ContainerInsights'...
No hay datos disponibles para 'MemoryUtilized'.
Todas las métricas están dentro de los rangos aceptables.
Esperando 40 segundos antes de la siguiente verificación...

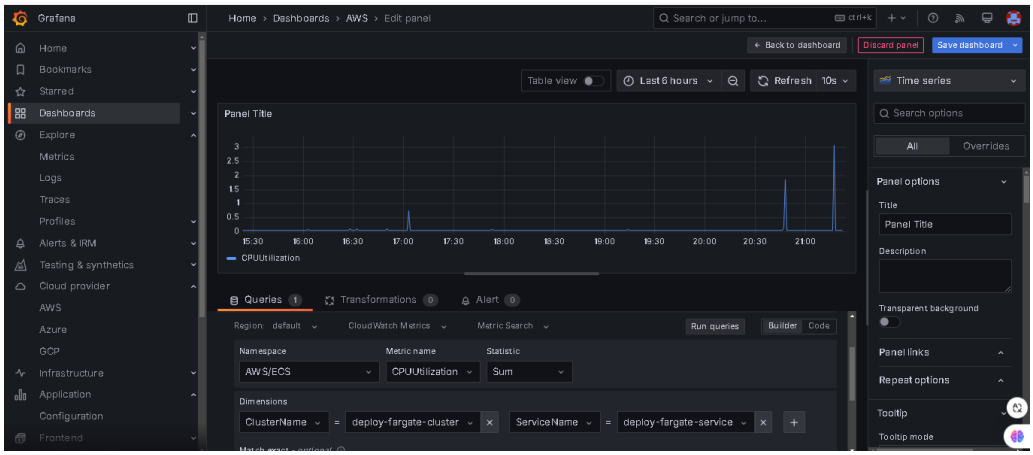

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Creando el clúster 'deploy-fargate-cluster'...
Clúster creado: deploy-fargate-cluster
Registrando la definición de tarea...
Definición de tarea registrada: deploy-fargate-task-family:9
Creando o actualizando el servicio ECS...
Creando un nuevo servicio...
Servicio creado correctamente.
Esperando a que el servicio esté estable...
Estado actual: 0/1 tareas en ejecución. Esperando...
Estado actual: 0/1 tareas en ejecución. Esperando...
Estado actual: 0/1 tareas en ejecución. Esperando...
Estado actual: 0/1 tareas en ejecución. Esperando...
El servicio 'deploy-fargate-service' está estable con 1 tareas en ejecución.
Verificando el estado del servicio...
Servicio: deploy-fargate-service, Estado: ACTIVE, Tareas activas: 1
Obteniendo la IP pública de las tareas en ejecución...
Tu aplicación está disponible en: http://44.202.211.181:8080
Despliegue completado en fargate.
Esperando 40 segundos antes de la siguiente verificación...
Monitoreando el despliegue en fargate...
Obteniendo el valor de la métrica 'Invocations' desde CloudWatch...
Valor actual de la métrica 'Invocations': 2797.0
La métrica 'Invocations' está dentro del rango aceptable.
Esperando 40 segundos antes de la siguiente verificación...
Monitoreando el despliegue en fargate...
Obteniendo el valor de la métrica 'Invocations' desde CloudWatch...
Ln 1, Col 1 (2964 selected)

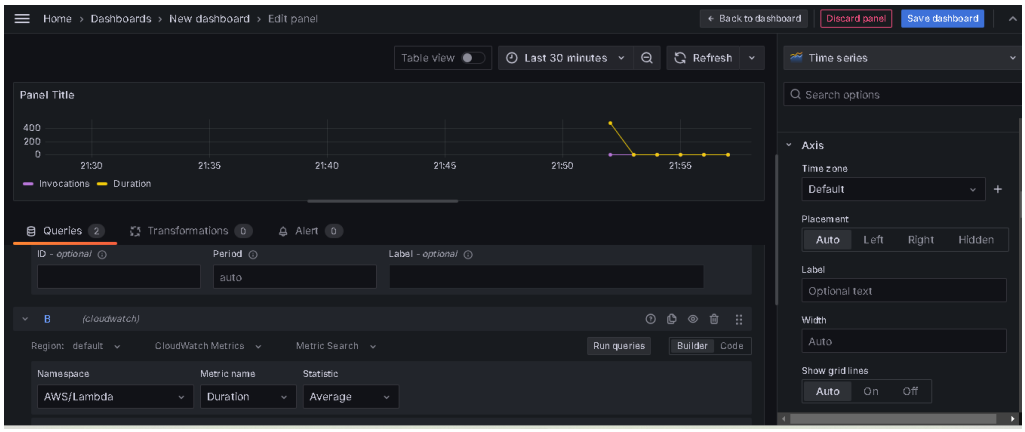

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
==========
Monitoreando el despliegue en fargate
==========
Obteniendo métrica: CPUUtilization...
Métrica 'CPUUtilization' - Suma: 14.464619350930056
Obteniendo métrica: MemoryUtilization...
Métrica 'MemoryUtilization' - Suma: 5.924479166666667
>>> Todas las métricas de 'fargate' están dentro del rango aceptable. <<<
Esperando 20 segundos antes de la siguiente verificación...
==========
Monitoreando el despliegue en fargate
==========
Obteniendo métrica: CPUUtilization...
Métrica 'CPUUtilization' - Suma: 46.30844157189131
Obteniendo métrica: MemoryUtilization...
Métrica 'MemoryUtilization' - Suma: 12.174479166666668
>>> Todas las métricas de 'fargate' están dentro del rango aceptable. <<<
Esperando 20 segundos antes de la siguiente verificación...

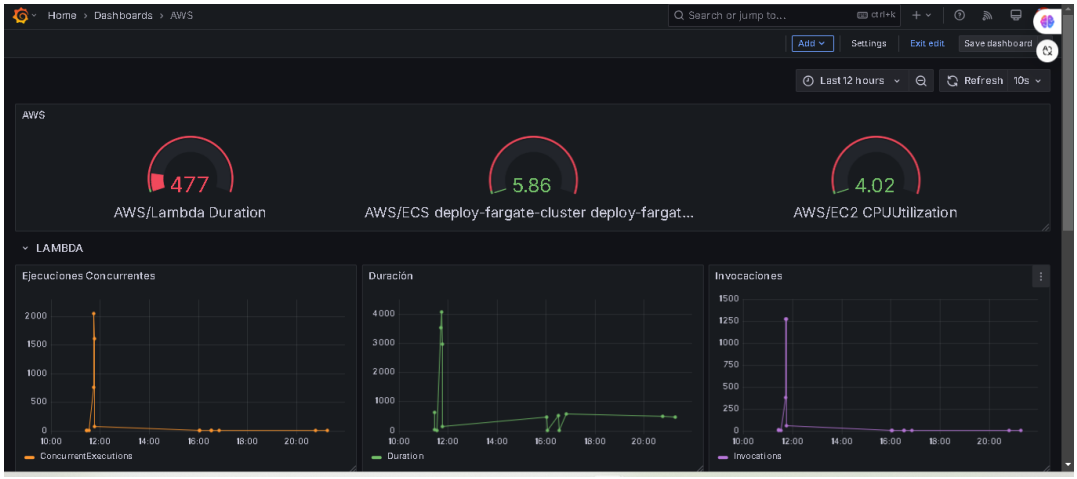

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Monitoreando el despliegue en fargate
==========
Obteniendo métrica: CPUUtilization...
Métrica 'CPUUtilization' - Suma: 0.08591338495413461
Obteniendo métrica: MemoryUtilization...
Métrica 'MemoryUtilization' - Suma: 12.5
*** Umbral superado! ***
- Métrica: MemoryUtilization
- Suma: 12.5
- Umbral: 10
=== Autoescalando de fargate a ec2 ===
Creando una instancia EC2 con Ubuntu 22.04...
Instancia creada. Esperando a que esté en estado 'running'...
Instancia EC2 en ejecución. ID: i-0b03eeab57ad55275, IP Pública: 3.95.222.241
Tu aplicación está disponible en: http://3.95.222.241:8080
Despliegue completado en ec2.
==========
Monitoreando el despliegue en ec2
==========
Obteniendo métrica: CPUUtilization...
No se encontraron datos para la métrica 'CPUUtilization'.
>>> Todas las métricas de 'ec2' están dentro del rango aceptable. <<<

```
========================================
Monitoreando métricas de EC2 para instancia i-05175980caf69ec58
========================================
Métrica 'CPUUtilization' - Suma: 86.77326538204817
Costo estimado actual para ec2 (40 segundos): $0.01118
Costo estimado extrapolado para ec2 (1 hora): $1.00657

Esperando 40 segundos antes de la siguiente verificación...


========================================
Monitoreando métricas de EC2 para instancia i-05175980caf69ec58
========================================
Métrica 'CPUUtilization' - Suma: 116.17706400750099
Costo estimado actual para ec2 (40 segundos): $0.01497
Costo estimado extrapolado para ec2 (1 hora): $1.34765

Esperando 40 segundos antes de la siguiente verificación...


========================================
Monitoreando métricas de EC2 para instancia i-05175980caf69ec58
========================================
Métrica 'CPUUtilization' - Suma: 142.16039734083432
Costo estimado actual para ec2 (40 segundos): $0.01832
Costo estimado extrapolado para ec2 (1 hora): $1.64906

Esperando 40 segundos antes de la siguiente verificación...
```

9.1 Construcción de dashboard en Grafana

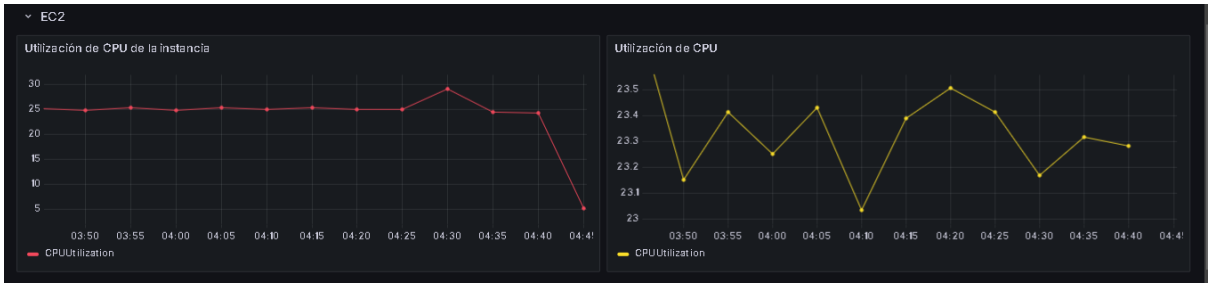




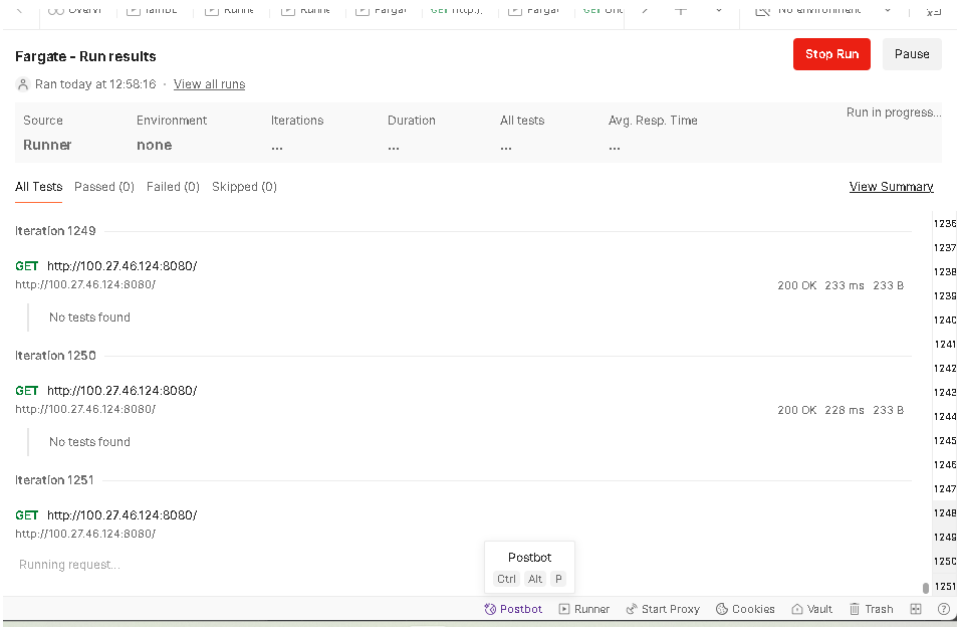

Home › Dashboards › AWS
Search or jump to...
Add
Settings
Exit edit
Save dashboard
Last 12 hours
Refresh 10s
AWS
477
AWS/Lambda Duration
5.86
AWS/ECS deploy-fargate-cluster deploy-fargat...
4.02
AWS/EC2 CPUUtilization
LAMBDA
Ejecuciones Concurrentes
Duración
Invocaciones
ConcurrentExecutions
Duration
Invocations


Last 1 hour
Refresh
Panel Title
Panel Title
Panel Title
Invocations
Duration
ConcurrentExecutions


LAMBDA
Ejecuciones Concurrentes
Duración
Invocaciones
ConcurrentExecutions
Duration
Invocations

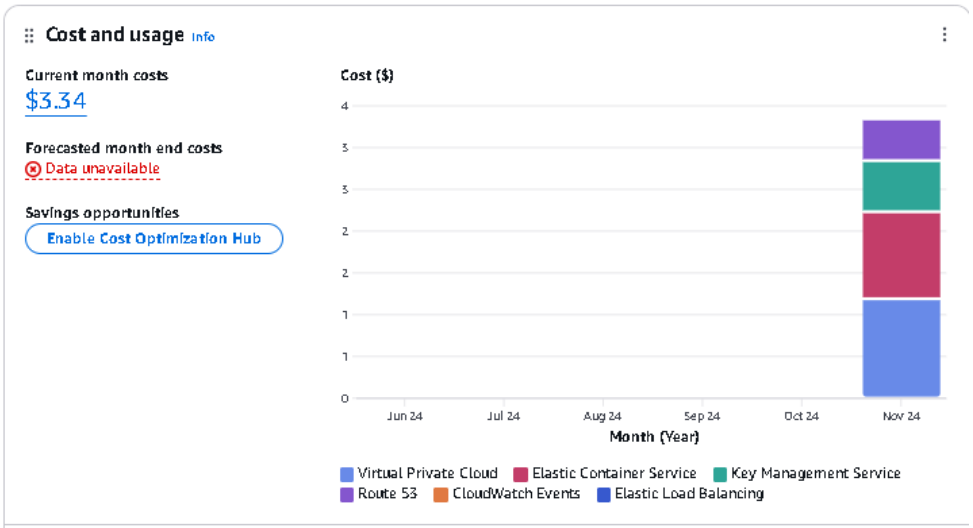

EC2
Utilización de CPU de la instancia
Utilización de CPU
CPUUtilization
CPUUtilization

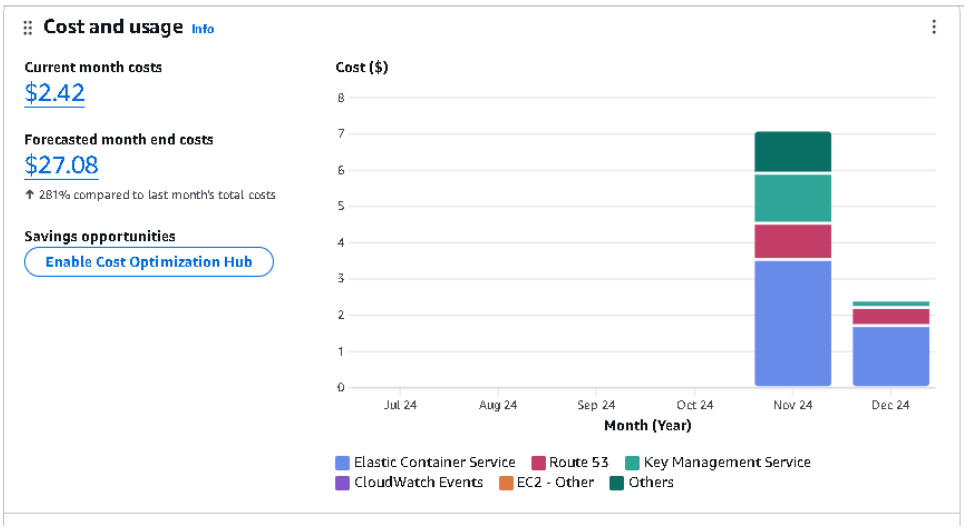
Pruebas




```
WARN[0123] Request Failed                                error="Get \"https://7v7btr9yk1.execute-api.us-east-1.amazonaws.com/\": dial tcp 44.193.50.
135:443: connectex: Se produjo un error durante el intento de conexión ya que la parte conectada no respondió adecuadamente tras un periodo de tiemp
o, o bien se produjo un error en la conexión establecida ya que el host conectado no ha podido responder."

     ✗ status was 200
      ↳  99% — ✓ 180328 / ✗ 1

     checks.........................: 99.99% 180328 out of 180329
     data_received..................: 347 MB 479 kB/s
     data_sent......................: 9.1 MB 13 kB/s
     http_req_blocked...............: avg=2.11ms   min=0s       med=0s       max=16.52s   p(90)=0s       p(95)=0s
     http_req_connecting............: avg=916.86µs min=0s       med=0s       max=3.42s    p(90)=0s       p(95)=0s
   ✓ http_req_duration..............: avg=259.66ms min=0s       med=122.94ms max=23.71s   p(90)=252.01ms p(95)=871.7ms
       { expected_response:true }...: avg=259.66ms min=110.01ms med=122.94ms max=23.71s   p(90)=252.01ms p(95)=871.72ms
     http_req_failed................: 0.00%  1 out of 180329
     http_req_receiving.............: avg=1.71ms   min=0s       med=0s       max=10.86s   p(90)=1.51ms   p(95)=2.52ms
     http_req_sending...............: avg=294.99µs min=0s       med=0s       max=111.52ms p(90)=732µs    p(95)=1ms
     http_req_tls_handshaking.......: avg=1.19ms   min=0s       med=0s       max=13.26s   p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=257.66ms min=0s       med=122.06ms max=23.71s   p(90)=250.6ms  p(95)=867.85ms
     http_reqs......................: 180329 249.232711/s
     iteration_duration.............: avg=2.26s    min=115.31ms med=2.22s    max=26.36s   p(90)=3.84s    p(95)=4.04s
     iterations.....................: 180329 249.232711/s
     vus............................: 6      min=2               max=1000
     vus_max........................: 1000   min=1000            max=1000


running (12m03.5s), 0000/1000 VUs, 180329 complete and 0 interrupted iterations
default ✓ [======================================] 0000/1000 VUs  12m0s
PS C:\Users\User\Desktop\scripts-proyecto-integradora>
```

Costos


Cost and usage Info
Current month costs
$3.34
Forecasted month end costs
Data unavailable
Savings opportunities
Enable Cost Optimization Hub
Cost ($)
4
3
3
2
2
1
1
0
Jun 24
Jul 24
Aug 24
Sep 24
Oct 24
Nov 24
Month (Year)
Virtual Private Cloud
Elastic Container Service
Key Management Service
Route 53
CloudWatch Events
Elastic Load Balancing


Cost and usage Info
Current month costs
$2.42
Forecasted month end costs
$27.08
↑ 281% compared to last month's total costs
Savings opportunities
Enable Cost Optimization Hub
Cost ($)
8
7
6
5
4
3
2
1
0
Jul 24
Aug 24
Sep 24
Oct 24
Nov 24
Dec 24
Month (Year)
Elastic Container Service
Route 53
Key Management Service
CloudWatch Events
EC2 - Other
Others


Cost and usage Info
Current month costs
$7.33
↓ 7% compared to last month for same period
Forecasted month end costs
$8.35
↓ 3% compared to last month's total costs
Savings opportunities
Enable Cost Optimization Hub
Cost ($)
10
8
6
4
2
0
Aug 24
Sep 24
Oct 24
Nov 24
Dec 24
Jan 25
Month (Year)
Elastic Container Service
Key Management Service
Virtual Private Cloud
CloudWatch
EC2 - Other
Others

Cost and usage Info

Current month costs

$2.31

↑ 114% compared to last month for same period

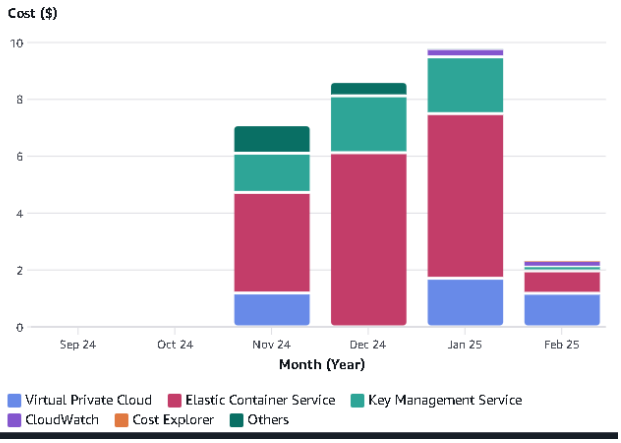
Forecasted month end costs

$15.66

↑ 60% compared to last month's total costs

Savings opportunities

Enable Cost Optimization Hub



Cost and usage Info

Current month costs

$9.88

↑ 291% compared to last month for same period

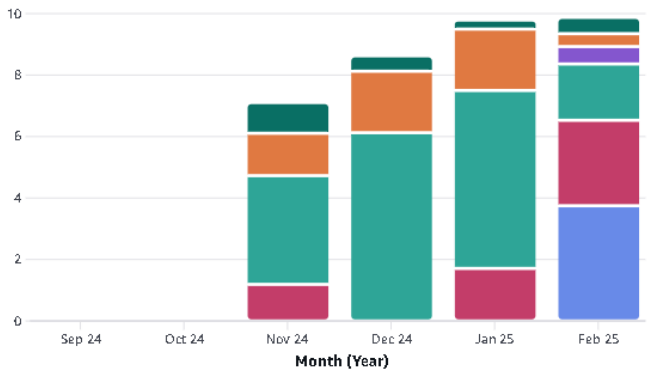
Forecasted month end costs

$50.07

↑ 411% compared to last month's total costs

Savings opportunities

Enable Cost Optimization Hub



Cost and usage Info

Current month costs

$23.99

↑ 679% compared to last month for same period

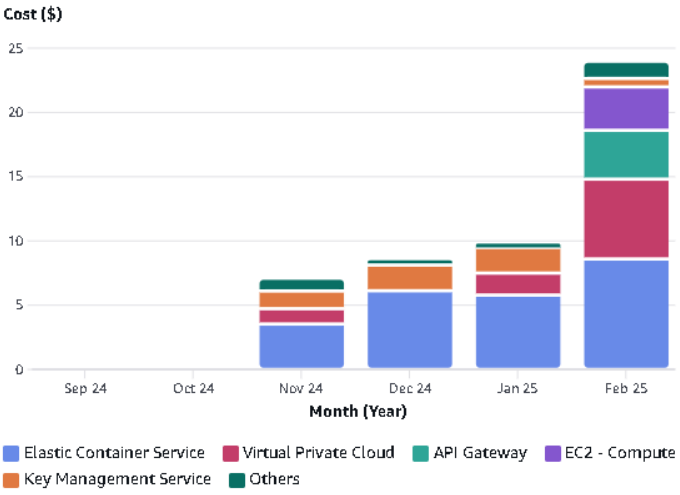
Forecasted month end costs

$49.19

↑ 402% compared to last month's total costs

Savings opportunities

Enable Cost Optimization Hub



Go to Billing and Cost Management