Arquitectura para Sistemas de Información con una Alta Demanda en Bases de Datos en AWS

Daniel Esteban Perez Bohorquez, Juan Francisco Teran Roman

Diciembre 2023

1 Abstract

Efficiently managing vast volumes of data is crucial in the modern era, where the demand for information is growing exponentially. This article introduces two architectures based on Amazon Web Services (AWS) to tackle specific challenges associated with handling transactional databases during periods of exceptional demand. The aim is to illustrate the practical differences between them.

2 Introducción

En la actualidad, la eficiente administración de vastos conjuntos de datos es esencial, ya que la demanda de información experimenta un crecimiento exponencial, ya sea en un momento determinado donde pocos usuarios la demandan o donde cada día más y más usuarios piden información.

Este artículo introduce una arquitectura diseñada en Amazon Web Services (AWS) para enfrentar desafíos particulares relacionados con la gestión de bases de datos transaccionales, especialmente durante momentos de demanda excepcional. Destacando principios clave como escalabilidad, elasticidad y alta disponibilidad.

Se presentan dos prototipos, cada uno con su arquitectura correspondiente, que contrastan con la rigidez percibida en entornos tradicionales, como el de algunas startups, frente a empresas que han adoptado estructuras más flexibles.

La intención principal es evidenciar de manera clara las notables diferencias entre estos dos enfoques, subrayando la creciente necesidad de soluciones dinámicas en la gestión de datos en comparación con enfoques estáticos arraigados en la tradición empresarial.

3 Descripción y Caracterización del Problema

Muchos sistemas transaccionales tienden a crecer a la par que las empresas lo hacen. Inicialmente, los sistemas de información tradicionales suelen ser suficientes para las organizaciones emprendedoras, pero a medida que estas empresas crecen, y con ellas la información que manejan, los sistemas transaccionales tradicionales ponen límites a este crecimiento principalmente por la gestión ineficaz de una alta demanda de transacciones simultáneas. Por ello, las organizaciones crecientes, bajo un sistema de información tradicional, enfrentan desafíos relacionados con la gestión de bases de datos transaccionales bajo una alta demanda. Problemas como congestión temporal, respuestas lentas de la base de datos y fluctuaciones en los requisitos de recursos representan obstáculos significativos que cualquier empresa busca evitar.

Por ejemplo, en uno de nuestros proyectos, denominado *tu catalogo.digital*, una startup que ha experimentado un crecimiento constante de usuarios, nos hemos enfrentado al desafío de adquirir servicios de almacenamiento que se adapten de manera precisa a nuestras necesidades. La decisión de compra se ha vuelto crucial, ya que optar por un servicio con exceso de recursos podría resultar en gastos innecesarios, mientras que elegir uno demasiado pequeño podría ocasionar problemas de almacenamiento e incluso la pérdida de usuarios debido a la ralentización del sistema.

La arquitectura transaccional tradicional comienza con la creación de la base de datos, donde se suelen implementar estrategias como la inclusión de disparadores o el uso de transacciones para evitar conflictos cuando dos usuarios intentan editar la misma fila simultáneamente. Este enfoque es esencial para mantener la integridad de los datos, especialmente cuando estas transacciones son solicitadas desde varios dispositivos. La presencia de transacciones en una única base de datos puede, sin embargo, ralentizar el sistema, incluso cuando un usuario simplemente busca realizar una lectura, expresada en el programa mediante una instrucción tipo SELECT.

Estos desafíos subrayan la necesidad de una arquitectura robusta y escalable que pueda adaptarse dinámicamente a cargas de trabajo variables, como sucede cuando Mercado Libre está próximo a una época de compras (como el "Black Friday"), inscribimos materias en la plataforma de horarios de la universidad para un periodo académico o cualquier momento de alta demanda predecible en el tiempo.

En cuanto a los sistemas de información tradicionales y según Zhiguo, Zhiqiang y Hao [4], las bases de datos SQL han mejorado la disponibilidad y la capacidad de escalabilidad horizontal más que los modelos de datos NoSQL. Estos autores afirman que estas mejoras dependen del modelo maestro-esclavo, el middleware y la fragmentación.

4 Marco Teórico

La fundamentación teórica de esta propuesta se cimenta en los principios esenciales de la computación en la nube, concentrándose en la escalabilidad y la elasticidad como pilares fundamentales para enfrentar los retos actuales de la gestión de datos.

- Escalabilidad: La capacidad de un sistema de sostener las cargas de trabajo creciente mediante el uso de recursos adicionales.

  Por ejemplo, si una aplicación web tiene cada vez más usuarios y transacciones, el sistema puede crecer para soportar ese volumen de datos y procesamiento.

- Escalamiento Vertical (Scale UP & DOWN): Incrementar las capacidades del recurso actual (Memoria, Almacenamiento, Cómputo, red, etc.), tipo de escalado más fácil, soluciones a corto plazo.

  Por ejemplo, si una base de datos necesita más espacio o velocidad, se puede ampliar el disco duro o el procesador del servidor donde está alojada. Esto permite mejorar el rendimiento y la capacidad del sistema, pero tiene un límite físico y un mayor coste.

- Escalamiento Horizontal: Agregar más instancias del recurso utilizado, para en conjunto hacer frente a la nueva carga de trabajo, es un escalado más robusto, soluciones a largo plazo.

  Por ejemplo, si una base de datos es muy grande y compleja, se puede dividir en partes más pequeñas y repartirlas entre varios servidores o máquinas para que las almacenen, lo que se conoce como particionamiento o sharding. De este manera, se reduce la carga de trabajo de cada servidor y se aprovecha el poder de procesamiento y el espacio de almacenamiento de varios servidores. Esto permite mejorar el rendimiento, la capacidad, la disponibilidad y la tolerancia a fallos del sistema.

- Maestro-esclavo: Forma de organizar los recursos informáticos de manera que haya un sistema principal (maestro) que controla y distribuye los recursos a otros sistemas secundarios (esclavos).

  Por ejemplo, la replicación de bases de datos consiste en una base de datos maestra que recibe las consultas de escritura y las envía a las bases de datos esclavas, que solo reciben las consultas de lectura. Así se logra balancear la carga y evitar que la base de datos maestra se sature. Además, si la base de datos maestra se cae, se puede elegir una de las esclavas como nueva maestra y seguir ofreciendo el servicio.

- Elasticidad: La capacidad de adquirir recursos cuando los necesite y liberar recursos cuando ya no los necesite. Habilidad para adaptarse al cambio.

Por ejemplo, si una aplicación web tiene muchos usuarios en un momento determinado, el sistema puede asignar más recursos para atender las peticiones, y si luego hay menos usuarios, el sistema puede liberar esos recursos para ahorrar costes."

5 Estado del Arte

Se han llevado a cabo diversas investigaciones en torno a este tema, centrándose en los servicios de Amazon diseñados para lograr escalabilidad y elasticidad, incluso cuando la configuración es realizada manualmente. Uno de los autores que exploramos en nuestra investigación es Williams [1], quien proporciona una breve descripción de distintas fases para explicar la escalabilidad y las herramientas útiles que AWS ofrece para su implementación y evaluación. En su análisis, señala que cada caso puede presentar variaciones significativas.

Desde esta perspectiva, nos fundamentamos en dos prototipos que, al final, se traducen en dos fases distintas. La primera fase se inspira en la fase inicial presentada por Williams [1], sirviendo como un punto de partida común observado en startups o emprendimientos. Posteriormente, si estas organizaciones se enfrentan a un período crítico donde experimentan un crecimiento importante de usuarios, se realiza la transición a la segunda fase. Este trabajo no solo aborda la teoría, sino que también explora experimentalmente cómo se logra el aumento y asignación de recursos sin perder de vista la búsqueda de menores costos.

Por otro lado, según los autores Zhiguo, Zhiqiang y Hao [4], en el paper titulado "Research on high availability architecture of SQL and NoSQL.", ellos describen arquitecturas de alta disponibilidad NoSQL y SQL, comparando sus distintas características. Concluyen que ambas arquitecturas de bases de datos escalan horizontalmente mediante fragmentación y tienen alta disponibilidad y seguridad de los datos mediante maestro-esclavo o replicación, el modelo de datos SQL es más escalable que las bases de datos NoSQL y menos flexible, y finalmente proponen una arquitectura híbrida para manejar datos de forma universal y eficiente compuesta tanto de manejo de datos SQL como NoSQL.

6 Propuesta de Solución

6.1 Arquitectura General de la Solución

La solución propuesta aboga por la utilización de servicios de AWS para abordar los desafíos de bases de datos transaccionales bajo alta demanda.

6.1.1 Arquitectura General del primer prototipo

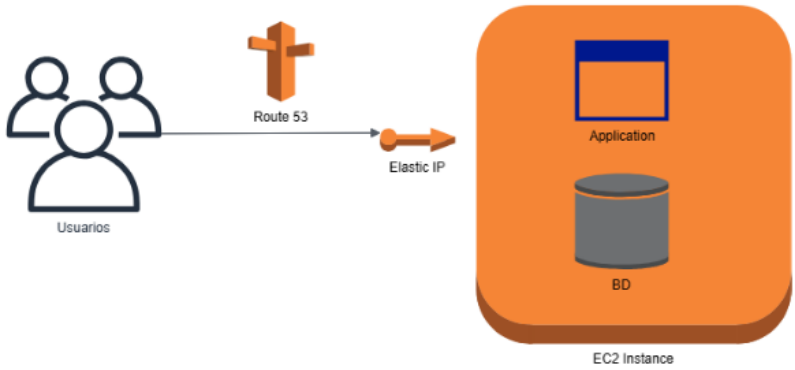

Figure 1: Primer prototipo. Creado en "Visual Paradigm" - Suite de Productividad Online (`visual-paradigm.com`)

El sistema permite acceso remoto a través de instancias EC2 en AWS, configuradas como máquinas virtuales en la nube. Estas instancias, denominadas "computadoras de alguien más", se configuran con SSH y permiten personalizar hardware, sistema operativo y software. Se utiliza Elastic IP para direcciones fijas y S3 para almacenamiento en la nube. Route 53 facilita la adquisición de dominios y su configuración.

6.1.2 Arquitectura General del segundo prototipo

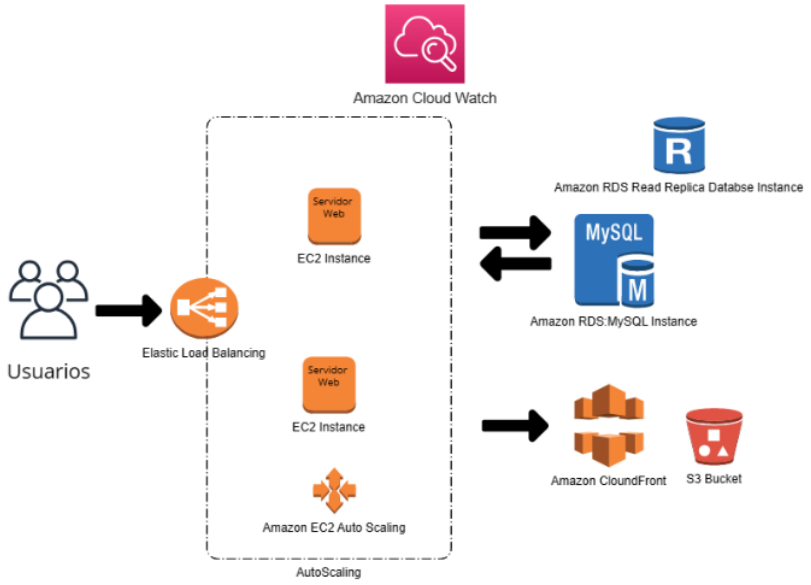

Figure 2: Segundo prototipo. Creado en "Visual Paradigm" - Suite de Productividad Online (`visual-paradigm.com`)

Grupo de usuarios: En este caso, son los estudiantes.

Elastic Load Balancing: Desde el punto de vista del cliente que consume los datos, no queremos tener una dirección para cada instancia, por lo que se construye un único punto de acceso a través de un balanceador de carga. Esta funcionalidad, provista por AWS Load Balancer, se encarga de distribuir el tráfico a través de las instancias EC2 configuradas, determinando el estado de las mismas y enviando solo pedidos a las instancias que se presenten saludables.

Amazon EC2 Auto Scaling: Esta característica permite escalar horizontalmente los servidores web según la demanda. Significa que puedes tener más servidores funcionando durante los períodos pico y menos durante los períodos de menor actividad, lo que puede ayudar a manejar la carga y ahorrar costos.

Instancia EC2: Hablemos de la instancia EC2 que configuraremos. Si usamos una instancia pequeña y hay mucha actividad, el sistema podría colapsar. Podríamos mejorarla dándole más potencia, pero si algo sale mal o se reinicia, todo el sistema queda fuera de juego. Necesitamos algo a prueba de errores. La idea es agregar más instancias que sean copias exactas. Así, cada una puede manejar peticiones y comparten la misma base de datos. Esto no solo mejora la capacidad, sino que también protege el sistema si algo sale mal, manteniendo el

servicio disponible para los usuarios. Podría funcionar con solo 2 perfectamente.

Amazon Cloud Watch: Proporciona monitoreo en tiempo real de los recursos de AWS y las aplicaciones de los clientes que se ejecutan en la infraestructura de Amazon.

Amazon RDS: Utilizaremos una base de datos relacional como MySQL, para manejar eficientemente los datos académicos y de inscripción. Estas bases de datos son robustas y ofrecen una gran cantidad de características para el manejo de datos.

Instancias de lectura: Crear instancias de lectura puede ayudar a distribuir la carga de consultas durante el período crítico. Esto puede mejorar el rendimiento del sistema al permitir que varias instancias manejan las solicitudes de lectura.

Amazon CloudFront: Este es un servicio de red de entrega de contenido (CDN) que acelera la entrega de sitios web, APIs, contenido de video u otros recursos web. Al distribuir contenido estático como imágenes y archivos CSS a los usuarios de manera eficiente, puedes mejorar la velocidad y la experiencia del usuario.

Amazon S3 Bucket: Utilizando S3 logramos otro nivel de desacoplamiento al encapsular la información estática en un ambiente separado siendo el path hacia los archivos lo que debemos conservar como punto de acceso desde otros componentes. Además suele utilizarse en conjunto con AWS Cloudfront de modo que los archivos en cuestión sean entregados con las políticas de entrega más veloces.

6.2 Arquitectura del Prototipo para el Experimento en AWS

6.2.1 Descripción del Primer prototipo, limitación a la Arquitectura General

En este contexto, se propone la implementación de una instancia configurada en RDS con un motor de base de datos de MYSQL, con 2 unidades de procesamiento central (CPU) virtuales, 1 gigabytes (GB) de memoria RAM y una clase de instancia tipo db.t3.micro . En situaciones críticas, caracterizadas por la demanda concurrente de múltiples usuarios solicitando el servicio, la estrategia adoptada implica la obtención de un promedio de dicha demanda o, alternativamente, la expectativa de que no se sobrepase, mitigando así posibles inconvenientes, y basados en dicho promedio hacer la compra o presupuesto para lo que necesitamos por el momento.

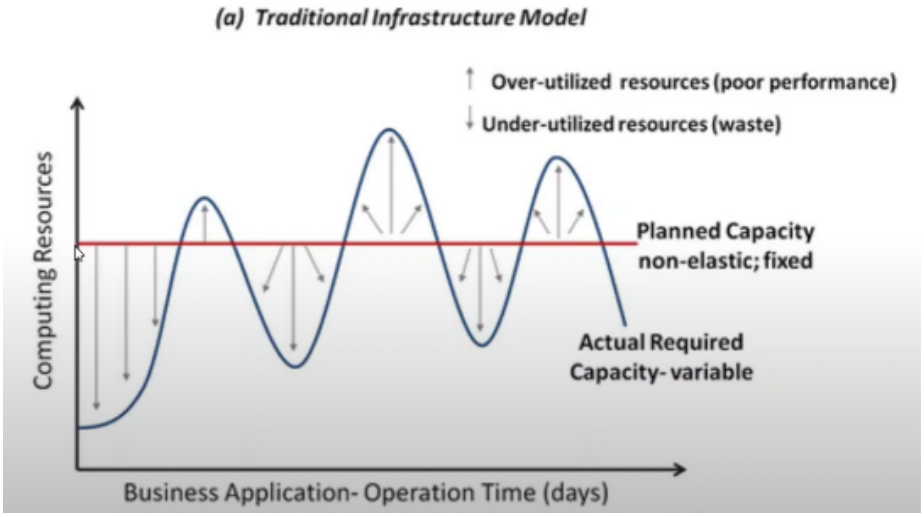

Figure 3: Rendimiento de un modelo sin elasticidad. Tomado de: GdeE Cloud Practitioner Sesion 9 AWS EC2 costos Elasticidad Escalabilidad. (s/f). (`https://www.youtube.com/watch?v=DJsArV-oqAc&t=2882s`)

6.2.2 Descripción del Segundo prototipo, limitación a la Arquitectura General

Nos proponemos adoptar una estrategia distinta, orientada a la implementación de soluciones proporcionadas por Amazon Web Services (AWS) con el propósito de fragmentar los recursos empleados, a pesar de fragmentar vamos a utlizar los mismos recursos que usamos en el primer prototipo, practicamente eliminando el escalamiento vertical, para evitar ventaja entre prototipos y enfocandonos en el escalamiento horizontal y elastico (de manera manual). Se busca evitar la consolidación de todos los elementos en una única instancia de RDS, optando en su lugar por una distribución descentralizada, donde tendremos por un lado una primera instacia encargada de la escritura de base de datos y otra en la lectura de la base de datos. Se persigue alcanzar una configuración que se asemeje a la representación visual adjunta en la siguiente imagen. [**?**].

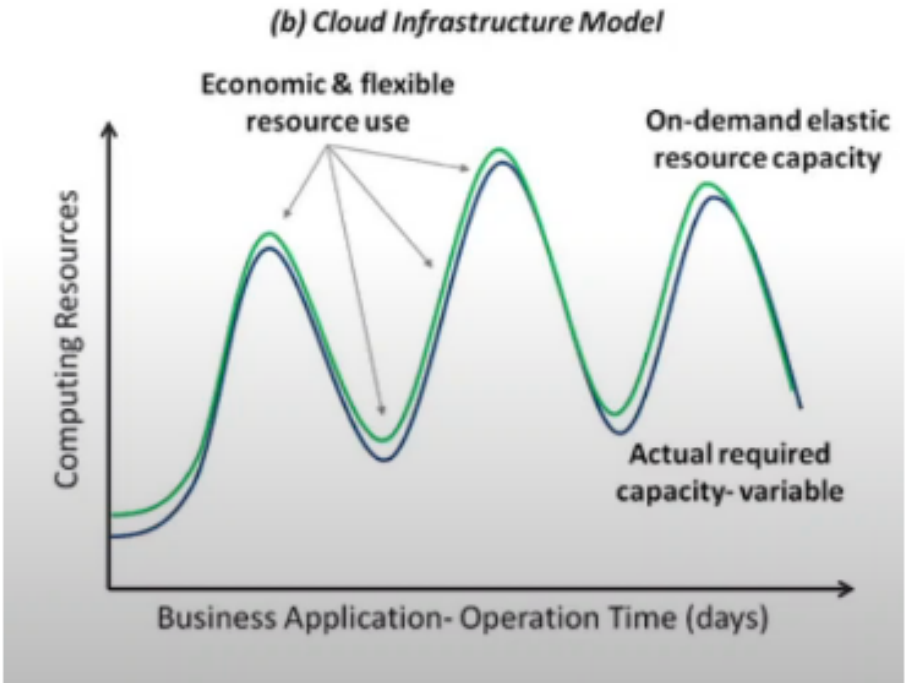

Figure 4: Rendimiento de un modelo con elasticidad ideal. Tomado de: GdeE Cloud Practitioner Sesion 9 AWS EC2 costos Elasticidad Escalabilidad. (s/f). (`https://www.youtube.com/watch?v=DJsArV-oqAc&t=2882s`)

La imagen que observamos anteriormente es una representación prácticamente perfecta. Es bastante difícil lograr tanta precisión, pero esto es precisamente lo que buscamos emular en nuestro segundo prototipo.

Especificando el funcionamiento de la base de datos y por qué se realiza de esta manera.

En ambientes de replicación de lectura en bases de datos, las operaciones de escritura, tales como eliminación, actualización y borrado, son comúnmente ejecutadas en la instancia principal para asegurar la coherencia de los datos. En sistemas de alta disponibilidad, la instancia principal se encarga de todas las operaciones de escritura, posteriormente sincronizando estos cambios con las réplicas de lectura, las cuales se dedican principalmente a operaciones de lectura para distribuir la carga. Es de vital importancia evitar operaciones de escritura directa en réplicas de lectura para prevenir conflictos de datos y asegurar una visualización coherente de los datos en todas las instancias.

El riguroso modelo de consistencia de bases de datos relacionales, como MySQL y PostgreSQL, refuerza la importancia de llevar a cabo todas las opera-

ciones de escritura en una única instancia principal para preservar la integridad y coherencia de los datos.

7 Evaluación del Experimento

El experimento se inició el 5 de Diciembre de 2023, a las 10:30 a. m (importante tener en cuenta esta hora para observar las graficas de los resultados). Creamos dos instancias de base de datos y una instancia adicional para que sirva como réplica. Posteriormente, en cada instancia, creamos la base de datos, las tablas básicas y la información mínima necesaria para iniciar. A continuación, sometimos estas bases de datos a una carga de trabajo mediante la ejecución de 100 consultas por segundo. Esto se realizó a través de una página web implementada en PHP, donde se invoca el script para ejecutar las consultas, especificando la conexión mediante el puerto de enlace proporcionado por Amazon en su servicio RDS. Por ejemplo, la dirección del primer prototipo sería "databaseprototipo1.co2c3dt7yhhx.us-east-1.rds.amazonaws.com".

Figure 5: Creación de 3 instancias de Base de Datos con el servicio RDS de Amazon.

Se evaluarán 3 indicadores críticos de rendimiento: Uso de la CPU, la latencia de escritura y la latencia de lectura, especialmente en lo que respecta a la velocidad de recuperación de datos, sistemas en tiempo real y aplicaciones que dependen en gran medida del manejo de información en grandes cantidades.

7.1 Latencia de Escritura

La latencia de escritura se define como el tiempo que transcurre desde el inicio de una operación de escritura hasta que la operación se completa con éxito. En otras palabras, es el período necesario para que los datos se registren de manera efectiva en la base de datos.

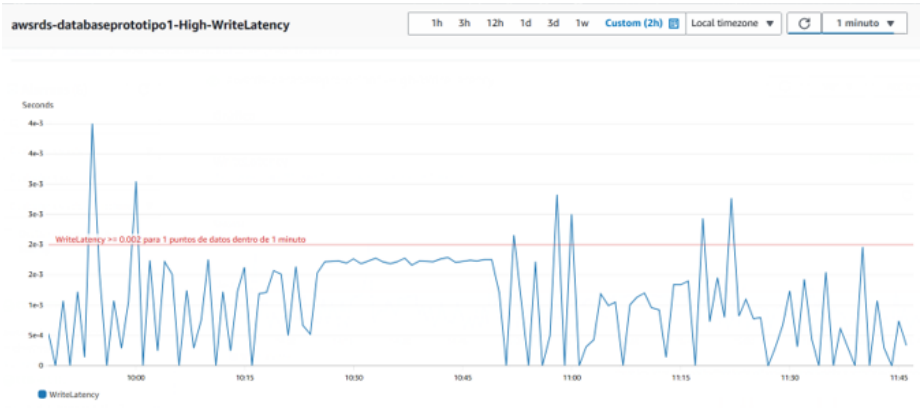

Figure 6: Estadísticas de Latencia de Escritura para el Prototipo 1 obtenidas a través del servicio CloudWatch

Figure 7: Estadísticas de Latencia de Escritura para el Prototipo 2 obtenidas a través del servicio CloudWatch

7.2 Latencia de Lectura

La latencia de lectura se refiere al tiempo que transcurre desde que se solicita una operación de lectura hasta que la operación se completa y los datos solicitados están disponibles.

Figure 8: Estadísticas de Latencia de Lectura para el Prototipo 1 obtenidas a través del servicio CloudWatch

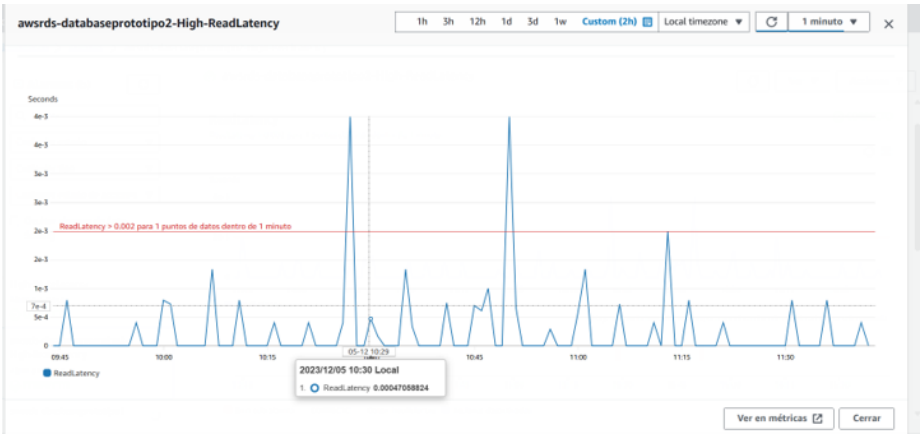

Figure 9: Estadísticas de Latencia de Lectura para el Prototipo 2 obtenidas a través del servicio CloudWatch

7.3 Utilización de la CPU

La utilización de la CPU se mide típicamente en porcentaje y representa la fracción de tiempo que la CPU está ocupada ejecutando procesos o tareas en relación con el tiempo total.

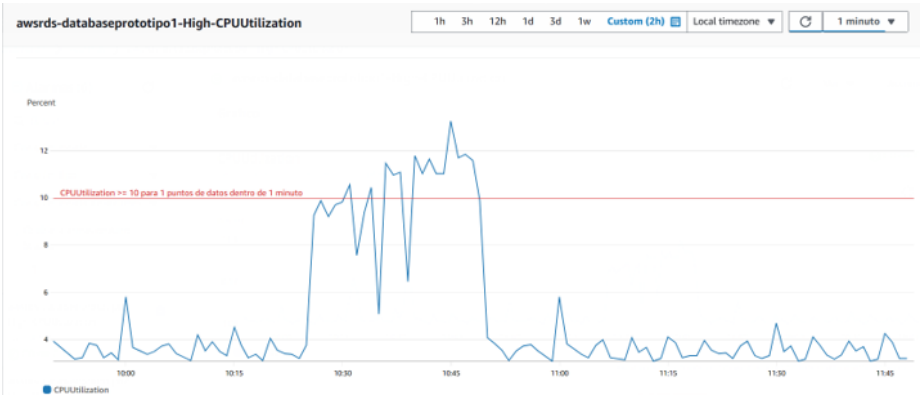

Figure 10: Estadísticas de Utlizacion de la CPU para el Prototipo 1 obtenidas a través del servicio CloudWatch

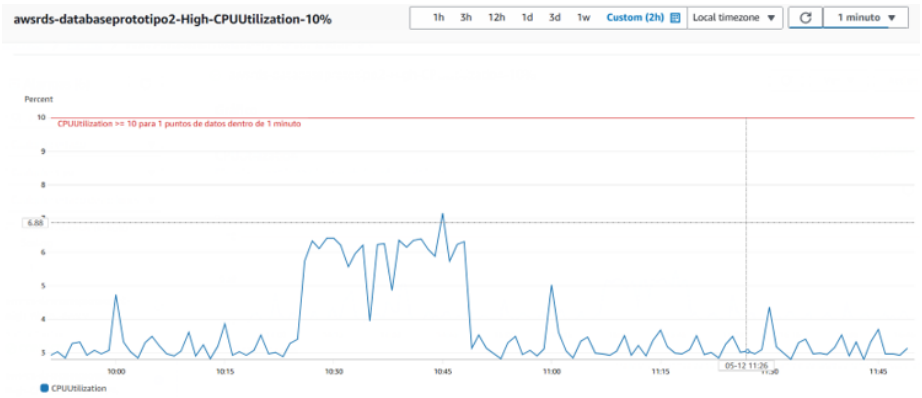

Figure 11: Estadísticas de Utlizacion de la CPU para el Prototipo 2 obtenidas a través del servicio CloudWatch

7.4 Resultados

Los resultados revelan un rendimiento notablemente superior en el Prototipo 2 en comparación con el Prototipo 1, tanto en la utilización de la CPU como en la latencia de escritura. La eficiencia del Prototipo 2 se evidencia claramente, ya que la latencia de escritura apenas roza el margen de los 0.003 segundos. Además, la utilización de la CPU se mantiene consistentemente por debajo del 10%, demostrando su capacidad para manejar procesos de manera eficiente y con una escritura rápida.

En cuanto a la latencia de lectura, es importante destacar que el Prototipo 2 cuenta con una réplica dedicada para estas operaciones. Este enfoque se refleja

en un único pico de latencia, en contraste con el Prototipo 1, que experimenta más de seis picos consecutivos. Estos resultados subrayan la eficacia del diseño del Prototipo 2, que logra gestionar las operaciones de lectura de manera más eficiente y robusta.

8 Conclusiones

- La arquitectura propuesta en AWS ofrece una solución integral para sistemas que enfrentan una alta demanda en bases de datos transaccionales, ofreciendo mayor escalabilidad, disponibilidad y rendimiento que el sistema transaccional tradicional.
- La combinación de escalabilidad, elasticidad y alta disponibilidad asegura un rendimiento óptimo y la capacidad de adaptarse a cambios en la carga de trabajo, estableciendo una base sólida para aplicaciones críticas.
- El experimento realizado sirve como una herramienta de evaluación continua para refinar el rendimiento del sistema bajo escenarios de alta demanda y hacer pruebas en diferentes contextos para probar su adaptabilidad, tanto en escenarios académicos como de emprendimiento.

References

[1] Williams, O. (2016, noviembre 24). Scalability in Amazon Web Services - Williams O. Medium. https://medium.com/@liams_o/scalability-inamazon-web-services-4464e415b616

[2] Difference between scalability, elasticity, high availability, and agility in the context of Amazon Cloud. (n.d.). Molecularsciences.org. Recuperado el 20 de octubre de 2023, de https://molecularsciences.org/content/difference-between-scalability-elasticity-high-availability-and-agility-in-the-context-

[3] Horizontal vs. Vertical scaling: Which is right for your app? (s/f). Missioncloud.com. Recuperado el 20 de octubre de 2023, de https://www.missioncloud.com/blog/horizontal-vs-vertical-scaling-which-is-right-for-your-app

[4] Zhiguo Wang, Zhiqiang Wei, Hao Liu; Research on high availability architecture of SQL and NoSQL. AIP Conf. Proc. 13 March 2017; 1820 (1): 090021. Recuperado el 3 de diciembre de 2023, de https://doi.org/10.1063/1.4977405

[5] @thesisbranes2019, author = Brañes Vilche, Rossemery Elizabeth, title = Back End Architecture With Amazon Web Services (AWS) For School Systems, year = 2019, school = Universidad de Montemorelos, Facultad de Ingeniería y Tecnología, howpublished = https:

//www.proquest.com/openview/385bfdbfa3af30a678eb9d62d46c7fdb/1?pq-origsite=gscholar&cbl=18750&diss=y, note = ProQuest. (2019).,

[6] @thesiswubu2020, author = Wubu, Tesfaye, title = Migration of Traditional IT System to Cloud Computing with Amazon Web Services, year = 2020, school = Metropolia University of Applied Sciences, type = Bachelor's Thesis, numberofpages = 41, date = 18 April 2020, degree = Bachelor of Engineering, degreeprogram = Information Technology, professionalmajor = Cloud Computing and IoT, instructors = Erik Pätynen, Senior Lecturer, howpublished = https://www.theseus.fi/bitstream/handle/10024/342872/Wubu_Tesfaye.pdf?sequence=2,