Revista de +D TECNOLÓGICO
Portal de revistas: http://revistas.utp.ac.pa

Diseño e implementación de una arquitectura de microservicios orientada a trabajar con transacciones distribuidas

Design and implementation of a microservices architecture oriented to work with distributed transactions

Brayan Rodriguez[1], Denis Cedeño-Moreno[2*]
[1]*Factultade de Sistemas Computacionales, Universidad Tecnológica de Panamá, Panamá*
[2]*Grupo de Investigación en Salud Electrónica y Supercomputación, Universidad Tecnológica de Panamá, Panamá*
denis.cedeno@utp.ac.pa

***Autor de correspondencia:** denis.cedeno@utp.ac.pa

RESUMEN:
Al trabajar con microservicios existen diferentes caminos que se pueden tomar, un sinfín de tecnologías que se pueden utilizar y por ahora no hay un camino estandarizado que los arquitectos y desarrolladores de software deben tomar para implementar este patrón de arquitectura. Para transacciones distribuidas a nivel de diferentes microservicios, no existe un camino especifico a tomar, como en tecnologías anteriores que se apalancaban en servidor de aplicaciones y componentes especializados, desarrollados a medida para tecnologías de una casa de software que lograban que la confianza en las soluciones fuese alta. Es por ello por lo que en esta investigación se propone un sistema de procesamiento de transacciones para aplicar compensación y marcha atrás en transacciones distribuidas de microservicios, ajena a la tecnología implementada y sin que afecte el rendimiento de los microservicios. Se propone y se implementa una arquitectura basada en el patrón de eventos, apalancándonos en la computación en la nube para disponibilidad los microservicios, el procesamiento de los eventos y la persistencia de datos. Esto con el fin de brindar una herramienta a los arquitectos y desarrolladores de software para resolver el problema de orquestar transacciones distribuidas a nivel de diferentes microservicios.

Palabras clave. Microservicios, transacciones distribuidas, computación en la nube, arquitectura de software.

ABSTRACT. When working with microservices there are different paths that can be taken, endless technologies that can be used, and for now there is no standardized path that software architects and developers should take to implement this architecture pattern. For distributed transactions at the level of different microservices, there is no specific path to take, as in previous technologies that leveraged application servers and specialized components, custom-developed for software house technologies that made trust in solutions was high. That is why this research proposes a transaction processing system to apply compensation and reversal in distributed transactions of microservices, unrelated to the implemented technology and without affecting the performance of microservices. An architecture based on the event pattern is proposed and implemented, leveraging on cloud computing for availability of microservices, event processing and data persistence. In order to provide a tool for software architects and developers to solve the problem of orchestrating distributed transactions at the level of different microservices.

Keywords. Microservices, distributed transactions, cloud computing, software architecture.

1. Introducción

Al momento de implementar una arquitectura de microservicios que procese transacciones, nos encontramos con diferentes problemas de implementación de esta, ya que los microservicios al ser un componente de alta disponibilidad, y poca dependencia entre otros componentes y tecnologías, se dificulta al momento de poder procesar transacciones que involucren a más de un microservicio [1].

Por este motivo, esta investigación se centra en la generación de una arquitectura que pueda superar este inconveniente, proponiendo diferentes patrones de implementación como lo es el patrón SAGA el cual es una secuencia de transacciones, donde si alguna transacción viola una regla de negocio, se ejecuta una compensación mediante la invocación de una operación que deshaga lo realizado anteriormente [2]. El patrón Event Sourcing que nos permite evitar una arquitectura de tipo "Solicitud y respuesta", añadiendo latencia a los microservicios, donde de forma asíncrona se notifique a al bus de servicios de la transacción y luego en él se pueda procesar la misma en caso tal sea necesario [3]. Estos dos patrones se verán a detalle en la investigación y como en conjunto estos patrones habilitan a la arquitectura para procesar transacciones en diferentes circunstancias tales como: transacciones asíncronas, transacciones síncronas, cancelación de una transacción en proceso, y relanzamiento de transacción.

2. Materiales y Métodos/Metodología

2.1 Definición de la arquitectura

Para lograr la generación de esta arquitectura de microservicios, se ha implementado el framework de Java llamado Spring que es uno de los estándares de la industria para desarrollo de componentes con bajo acoplamiento, con baja dependencia de otros componentes y con su servidor para despliegue embebido listo para utilizar esto para simular el lenguaje de programación que llevará la parte de lógica del negocio [4]. Para persistir datos se utiliza una instancia de base de datos de MySQL que se desplegará directamente en el contenedor, en caso tal se necesite persistir datos del microservicio [5].

Para la parte de la gestión de las transacciones la idea fue enviar eventos al EventBridge de Amazon Web Services. Este componente es un servicio conductor de eventos sin servidor que permite utilizar para conectar aplicaciones con datos de varias fuentes. Este proporciona un flujo de datos en tiempo real en sus aplicaciones [6]. Esto para que se pueda persistir el estado de las transacciones ya sean síncronas o asíncronas, y se pueda gestionar el estado de estas desde el bus de eventos. Se implementará la lógica de la gestión de las transacciones desde el bus de eventos mediante el uso de "Content Filtering Event patterns" que permiten capturar y procesar información de los eventos que llegan y actualizar la información de las transacciones habilitado por el EventBridge [7].

Para visualizar las transacciones se utilizará un portal desarrollado en Angular para conectarnos a los servicios de Amazon Web Services, y leer los estados de las transacciones, a su vez que se pueda gestionar las mismas desde este portal [8].

2.2 Modelo de la arquitectura propuesta

La arquitectura desarrollada dentro de esta investigación es una arquitectura de microservicios, simulando transacciones entre cuentas de una entidad bancaria. En la figura 1 vemos la arquitectura propuesta.

Lo que se busca simular es que dentro de la transacción existen varios pasos y que cada uno de estos pasos tenga interacción con un microservicio desplegado.

Esta primera parte de la arquitectura recibe la petición desde el consumidor, lo procesa con el componente API Gateway utilizado como puerta de entrada entre el consumidor y los servicios expuestos, y posterior se envían directamente a los microservicios desarrollados en Spring para el procesamiento [9].

Estos microservicios están desplegados en una instancia de EC2 la cual es una máquina virtual proporcionada por Amazon Web Services, cada uno está aislado dentro de una instancia de Docker y frente a ellos hay una instancia de Load Balancer de Amazon Web Services, para encargarse del balanceo de cargas entre las diferentes instancias que pueda haber de cada microservicio [10].

Cada uno de estos micro servicios tiene una conexión hacia el EventBridge y coloca un evento de la transacción que proceso que contiene en su mensajería el resultado de esta.

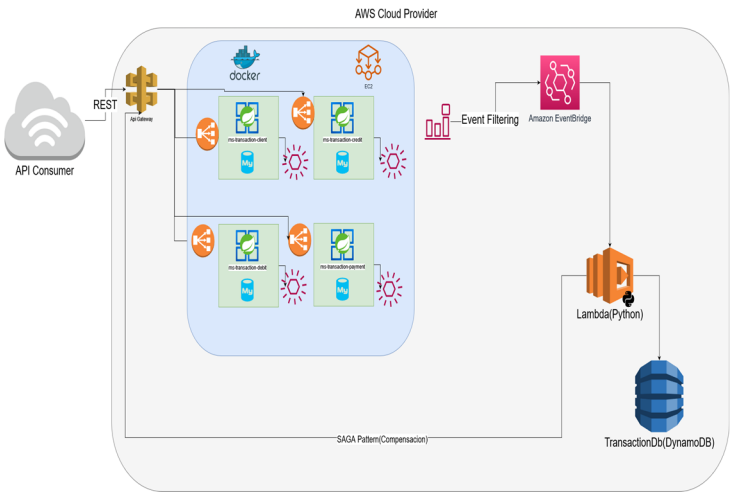

Figura 1. Microservicios y procesamiento de eventos.

Posteriormente este mensaje es capturado por un evento de tipo event filtering que está desplegado en el Amazon EventBridge y existe unas validaciones dependiendo de lo que llegue en la mensajería puede o no desencadenar un evento de compensación o marcha atras de la transacción. Una vez procesado este evento y sí se requiere la compensación o marcha atras de la transacción esto es colocado en una base de datos de tipo NoSQL llamada DynamoDb que permite persistir la información del evento desde la nube y poder en la segunda parte consultar la información y tratar esta información desde un tablero de mando [11]. En la figura 2 observamos como es la gestión de transacciones.

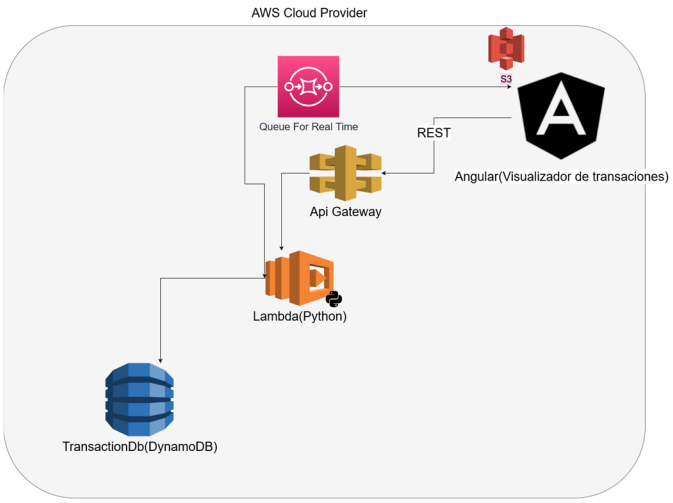

Figura 2. Gestión de transacciones y visualización.

En esta segunda parte se consulta a la base de datos donde se guarda la información de las transacciones que se deben compensar, y mediante otra instancia de API Gateway o cola se permite el acceso a la información de las diferentes transacciones guardadas para ser compensadas, y de esta forma se le disponibilidad hacia el tablero de mando la información de las transacciones procesadas, donde se podrá:

- Visualizar las transacciones realizadas en tiempo real
- Visualizar que transacciones se han compensado
- Compensar de forma manual transacciones
- Ver detalle de transacciones

De esta forma podemos cubrir de forma completa los escenarios de compensación o marcha atrás entre diferentes microservicios registrados.

2.3 Vista de componentes a nivel del microservicio

A nivel de componentes para desarrollar la lógica dentro de los microservicios se escoge la tecnología Java, específicamente con el Framework Spring y Springboot, dividiéndolo en 4 capas:

- Controlador
- Repositorio
- Servicio
- Modelo

La capa del controlador es la encargada de recibir la petición en el formato especifico, y transformarla hacia un objeto Java con el modelo definido. El Repositorio encapsula la lógica necesaria para acceder a la base de datos que contiene la información del cliente y aplica las transacciones. El servicio contiene la información necesaria para acceder ala base de datos especifica, con los datos de la conexión y el tratamiento de esta conexión. El modelo contiene los datos de la mensajería que viaja entre el consumidor y el servicio [12]. Una vez procesada la transacción en la base de datos se responde al consumidor y de forma asíncrona se arma la mensajería de la respuesta para enviarla al bus de eventos. A continuacion en la figura 3 vemos las capas de las aplicaciones.

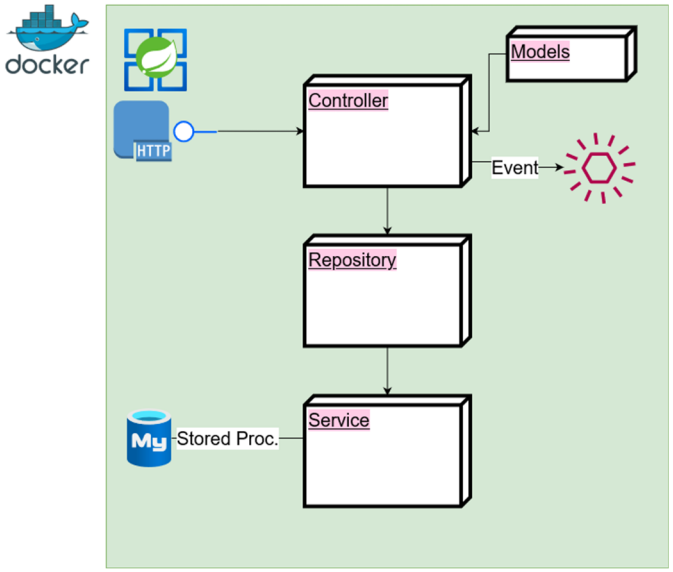

Figura 3. Arquitectura de las aplicaciones en Spring.

2.4 Procesamientos de eventos

Los eventos generados llevaran una estructura con la información de la transacción y esta información se le

dará tratamiento mediante la funcionalidad de "Content Filtering", desde la mensajería enviada provista por los microservicios, en tiempo real se va a detectar de que componente llegó, su estado y a que transacción pertenece como muestra la figura 4, de esta forma si la transición llega con un resultado tipo "OK", se considerara como correcta y si llega otro tipo de resultado se considerara como errónea y desencadenara una marcha atrás o compensación a ejecuciones previas de otros microservicios. Una vez detectada la acción a tomar, se persiste el estado en la base de datos DynamoDB para lograr visualizar estos datos en el tablero de mandos [13].

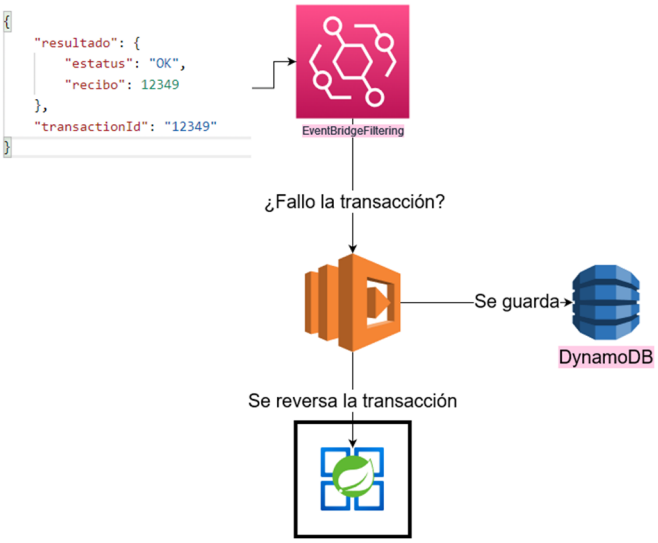

Figura 4. Diagrama de flujo para procesar las transacciones.

2.5 Muestra de los datos

Por último, tal y como muestra la figura 5, utilizando ya sea el API de consulta de transacciones o las colas asíncronas para visualizar en tiempo real, se desarrollaron dos pantallas aplacándonos en el Framework Angular versión 14, el consumo de la API expuesta de la información de las transacciones se puebla la información de las pantallas brindándonos detalles de las transacciones realizadas, transacciones en ejecución, transacciones fallidas y transacciones exitosas.

Al presionar en alguna de las transacciones, esta redirige a la pantalla de detalle, donde se logran visualizar los detalles de la transacción como los servicios de negocios involucrados, su estado, hora/fecha y detalle.

Figura 5. Tablero de mando para ver las transacciones

En la figura 6 podemos ver la transaccion en el tablero.

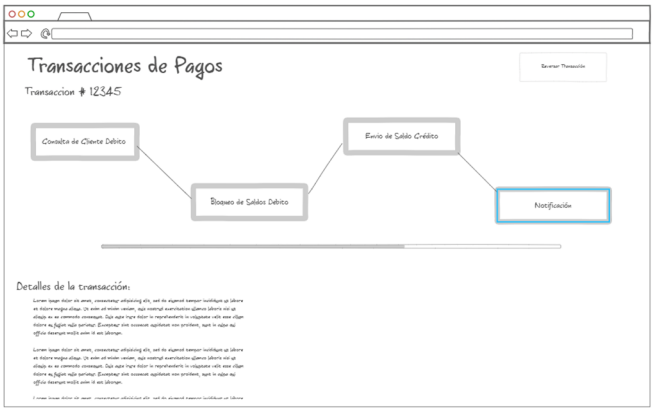

Figura 6. Visualizando una transacción en el tablero.

3. Resultados y discusión

Mediante el uso de la Herramienta Postman, que es un cliente para consumir API REST [14], le iniciamos las diferentes ejecuciones de pruebas que consisten en: Crear una colección con los diferentes casos de prueba:

- Consultar Información de Clientes y sus cuentas.
- Realizar transferencias entre cuentas.

De esta forma se ejecutan pruebas automatizadas con diferentes clientes, diferentes cuentas, donde los más importantes son aquellos candidatos donde se deben reversar las transferencias.
Una vez iniciada las transacciones notamos que en los casos donde el cliente que recibe el monto no está habilitado, lo cual el microservicio envía mediante los eventos al bus de eventos que esta transacción debe dar

marcha atrás, ya que se está violando un caso de negocio que es no permitir el envío de saldo a aquellos clientes que no están habilitados.

Posteriormente se muestra en el tablero de mando que esta transacción ha sido reversada y en el saldo del cliente que se le sustrajo el monto, éste se le aplicó un saldo positivo revirtiendo así la transacción que primeramente se había realizado. También de forma manual se deshabilitó el microservicio de crédito del monto al cliente esto para ver cómo se mostraba el resultado en el tablero de mando coma el cual se colocó como en espera de la última fase y de forma manual en el tablero de mando se detuvo la transacción y se reverso aplicando el saldo positivo al cliente que se le débito el monto. A su vez se nota también que las diferentes instancias de los microservicios que se van añadiendo no se ven afectadas por el procesamiento de las transacciones ya que estas siguen respondiendo de forma normal hacia el consumidor y como es de forma asíncrona el posteo del evento, no se ve reflejado en el rendimiento de la respuesta de estos microservicios.

4. Conclusiones

Luego de efectuar esta investigación y diseño e implementación de nuestra arquitectura de microservicios podemos concluir que los microservicios pueden ser implementados en distintos lenguajes de programación y tecnologías que se van a utilizar para resolver diferentes tipos de problema, sin embargo en estos casos transaccionales es necesario mantener una forma de lograr compensar o dar marcha atrás a transacciones fallidas, y en el caso de los microservicios al mantener una gran paleta de tecnologías disponibles, es necesario una herramienta que logre unir estos microservicios y puedan realizar esta función de compensación, como ya en tecnologías pasadas se lograba realizar, pero con la desventaja que estas tecnologías eran propietarias y regularmente solo se podían implementar con software de una misma compañía. Y este desarrollo logrado en este trabajo resuelve este complejo problema, pero también tiene una gran capacidad para crecer y lograr resolver otros inconvenientes mas detallados como registrar de forma autónoma estos microservicios, o precisar de más acciones a realizar en las transacciones.

El desarrollo de esta investigación utilizando las tecnologías de la nube permitió que este trabajo se realizará de forma mucho más rápida, ya que la mayoría de las tecnologías utilizadas ya estaban desarrolladas y proveían una interfaz para la implementación de las diferentes soluciones propuestas en esta arquitectura. Nos dedicamos solamente al desarrollo de las funcionalidades de negocio, el filtro de eventos la configuración y despliegue de microservicios y la configuración de las diferentes herramientas, como también la creación de las pantallas para el tablero de mandos. Pero esto fue de forma muy abreviada porque solo era cuestión de centrarse en la lógica de negocio, y Amazon Web services se encargaba de brindar las herramientas para acceder a estas funcionalidades.

Las tecnologías en la nube permiten que configuremos en tiempo real las reglas para las transacciones y esto lo tome al momento de salvar estas configuraciones, brindando 0% de tiempo de baja de servicio y logrando una mayor confianza en la solución. Es importante desarrollar un tablero de mando para que las personas técnicas o de negocio puedan visualizar los datos y transacciones, esto da una mayor flexibilidad para la herramienta y no queda encajonada en una solución netamente técnica, que fácilmente es remplazada por otra tecnología.

AGRADECIMIENTOS

Agradecemos a la Secretaría Nacional de Ciencia, Tecnología e Innovación (SENACYT, Panamá). Como también al Sistema Nacional de Investigación (SNI). Asimismo a GISES-CIDITIC-UTP.

REFERENCIAS

[1] G. Blinowski, A. Ojdowska, and A. Przybyłek, “Monolithic vs . Microservice Architecture : A Performance and Scalability Evaluation,” pp. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[2] M. Stefanko, O. Chaloupka, and B. Rossi, “The saga pattern in a reactive microservices environment,” *ICSOFT 2019 - Proc. 14th Int. Conf. Softw. Technol.*, no. Icsoft, pp. 483–490, 2019, doi: 10.5220/0007918704830490.

[3] F. Alongi, M. M. Bersani, N. Ghielmetti, R. Mirandola, and D. A. Tamburri, “Event-sourced, observable software architectures: An experience report,” *Softw. - Pract. Exp.*, vol. 52, no. 10, pp. 2127–2151, 2022, doi: 10.1002/spe.3116.

[4] A. Chatterjee and A. Prinz, “Applying Spring Security Framework with KeyCloak-Based OAuth2 to Protect Microservice Architecture APIs: A Case Study,” *Sensors*, vol. 22, no. 5, 2022, doi: 10.3390/s22051703.

[5] F. Dahri, A. M. El Hanafi, D. Handoko, and N. Wulan, “Implementation of Microservices Architecture in Learning Management System E-Course Using Web Service Method,” *Sinkron*, vol. 7, no. 1, pp. 76–82, 2022, doi: 10.33395/sinkron.v7i1.11229.

[6] Y. Jangir, R. Kumar, U. Nrupesh Surya, M. Mahajan, and V. Naik, “A Cloud-based Architecture using Micro-services for

the IoT-based Applications," *Proc. - 22nd IEEE/ACM Int. Symp. Clust. Cloud Internet Comput. CCGrid 2022*, no. May, pp. 893–898, 2022, doi: 10.1109/CCGrid54584.2022.00107.

[7] D. Kuryazov, D. Jabborov, and B. Khujamuratov, "Towards Decomposing Monolithic Applications into Microservices," *14th IEEE Int. Conf. Appl. Inf. Commun. Technol. AICT 2020 - Proc.*, pp. 21–24, 2020, doi: 10.1109/AICT50176.2020.9368571.

[8] N. Patrylo and M. Miłosz, "Comparison of AngularJS and VueJS frameworks efficiency," *J. Comput. Sci. Inst.*, vol. 5, pp. 204–207, 2017, doi: 10.35784/jcsi.622.

[9] R. Gadia, R. Shah, S. Varshney, and V. Sawant, "A System on Automated Database and API (Application Programming Interface) Management," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 10, no. 4, pp. 3226–3234, 2022, doi: 10.22214/ijraset.2022.41827.

[10] I. Karamitsos, S. Albarhami, and C. Apostolopoulos, "Applying devops practices of continuous automation for machine learning," *Inf.*, vol. 11, no. 7, pp. 1–15, 2020, doi: 10.3390/info11070363.

[11] H. B. S. Reddy, R. R. S. Reddy, R. Jonnalagadda, P. Singh, and A. Gogineni, "Analysis of the Unexplored Security Issues Common to All Types of NoSQL Databases," *Asian J. Res. Comput. Sci.*, vol. 14, no. 1, pp. 1–12, 2022, doi: 10.9734/ajrcos/2022/v14i130323.

[12] P. Manuel, "A Microservices e-Health System for Ecological Frailty Assessment Using Wearables," pp. 1–23, 2020.

[13] N. Navarro and D. Cedeno-Moreno, "Analysis of Emerging Technologies Proposed to Reduce the Impact of the COVID-19 Pandemic: A Review," *Congr. Int. en Intel. Ambient. Ing. Softw. y Salud Electrónica y Móvil*, vol. doi: 10.11, pp. 1–7, 2022.

[14] B. G. Wolde and A. S. Boltana, "REST API composition for effectively testing the Cloud," *J. Appl. Res. Technol.*, vol. 19, no. 6, pp. 676–693, 2021, doi: 10.22201/icat.24486736e.2021.19.6.924.