Revista Innovación y Software
Vol. 4, No. 1, Mes Marzo-Agosto, 2023
ISSN: 2708-0935
Pág. 151-160
https://revistas.ulasalle.edu.pe/innosoft

Tipo de artículo: Artículos originales
Temática: Calidad de software
Recibido: 21/01/2023 | Aceptado: 27/02/2023 | Publicado: 30/03/2023

Identificadores persistentes:
ARK: ark:/42411/s11/a86
PURL: 42411/s11/a86

Pruebas de Software para Microservicios

Software Testing for Microservices

Cesar Adolfo Laura Mamani [1]

[1] Universidad La Salle. Arequipa, Perú. clauram@ulasalle.edu.pe

[*] Autor para correspondencia: clauram@ulasalle.edu.pe

Resumen

Los microservicios han surgido como un estilo arquitectónico que ofrece muchas ventajas, pero también plantea desafíos. Uno de estos desafíos gira alrededor de las pruebas, puesto que una aplicación puede tener cientos o miles de servicios que funcionan juntos, y cada uno de ellos requiere ser probado a medida que evolucionan. Para superar este desafío, la automatización adquiere un papel clave, y junto con ella, el uso de herramientas de pruebas eficientes y eficaces.

Palabras clave: microservicios, pruebas de software, seguridad, rendimiento.

Abstract

Microservices have emerged as an architectural style that offers many advantages, but also poses challenges. One of these challenges revolves around testing, as an application may have hundreds or thousands of services running together, each requiring testing as they evolve. To overcome this challenge, automation takes on a key role, and along with it, the use of efficient and effective testing tools.

Keywords: *microservices, software testing, security, performance.*

Introducción

Actualmente existe un gran cambio y desafío para el desarrollo de las aplicaciones de software, ya que atravesamos una época de transiciones a lo digital, ya que desde el 2020 la pandemia provocó apresuradamente un gran cambio en el manejo de la información. Entonces al tener una gran demanda de

usuarios por consumo de X aplicación, se requiere una mayor eficiencia y mantenibilidad de los servicios de las aplicaciones, por ello la transición de los servicios monolíticos a microservicios fue el mayor acierto para afrontar esta demanda. Se pretende tratar todo lo referente a microservicios, ya que es el estándar actual en el desarrollo de software. Pero se necesita mantener una delgada línea entre la calidad y rapidez de entrega para estos servicios, entonces una manera confiable para respaldar que los microservicios desarrollados hagan lo que debían hacer, para esto es necesario realizar diferentes pruebas y estrategias para garantizar la calidad de los microservicios. Para ello se necesitan herramientas eficientes y eficaces que acompañen el proceso (Heinrich et al., 2017) y faciliten la automatización de las pruebas, ayudando a la vez a enfrentar los retos asociados a las pruebas en el contexto de microservicios (Heinrich et al., 2017)(Stefano Munari, Sebastiano Valle, 2018). De ahí que el estudio de herramientas que soporten la automatización de pruebas para microservicios sea un terreno fértil de investigación.

Métodos y Metodología computacional

El objetivo es entender la importancia de las pruebas de software aplicadas a microservicios, así de esta manera encontrar la mejor manera para aumentar la calidad de los microservicios.

Arquitecturas de servicios

Las dos arquitecturas principales usadas para descomponer un sistema en servicios son: la arquitectura orientada a servicios (SOA, por sus siglas en inglés: service-oriented architecture) y la arquitectura de microservicios. A continuación se describe cada una de ellas.

Arquitectura SOA

Los servicios SOA consisten en un diseño de descomposición de servicios integrados en un proyecto por mecanismos de enrutamiento inteligente, que proporciona una gobernanza global (o administración centralizada) (Cerny et al., 2017). De ahí que es frecuente la connotación de “arquitectura orquestación” o monolítica. Por ello, los procesos de los servicios se encuentran vinculados a un único contexto general. Además, SOA se encarga de encapsular la funcionalidad de negocio en una única interfaz (servicios SOA como web service o restfull), por medio de la cual el productor proporciona la funcionalidad y el consumidor

la puede solicitar. El Enterprise Service Bus es el medio de comunicación entre productores y consumidores, permitiendo tener comunicación punto a punto (Quenum & Aknine, 2018). SOA continúa siendo una arquitectura utilizada por las organizaciones. Sin embargo, SOA no se adapta bien a las necesidades de las metodologías ágiles (Chen, 2018), ni de movimientos como DevOps, Integración y Entrega Continua. Estas corrientes imponen la necesidad de implementar piezas pequeñas de software y colocarlas lo más rápido posible en los ambientes de producción. Por su parte, los proyectos creados con la arquitectura SOA adquieren grandes dimensiones y para colocar cambios en producción se requiere enviar todo el proyecto, por lo que no se alinean a las corrientes ágiles.

Arquitectura de microservicios

A inicios del año 2000 comienza la conceptualización de la arquitectura de los microservicios, de la mano del surgimiento de metodologías ágiles. Empresas como Amazon expresan la necesidad de una arquitectura con mayor capacidad de escalabilidad, en la cual sus componentes tengan mayor aislamiento e independencia (Carneiro & Schmelmer, 2016). En el año 2011 aparece por primera vez el término “microservicios” (Ueda et al., 2016)(Carneiro & Schmelmer, 2016). Los microservicios surgen como una alternativa de arquitectura para el diseño e implementación de sistemas distribuidos, dando como resultado sistemas con bajo acoplamiento de componentes que exhiben propiedades como flexibilidad, escalabilidad, adaptabilidad, y tolerancia a fallas, entre otros (Heinrich et al., 2017). La definición con mayor aceptación por parte de la comunidad de software es la de Martín Fowler, pionero en la arquitectura de microservicios. Fowler define los microservicios como un enfoque para el desarrollo de aplicaciones compuesto por un conjunto de servicios pequeños, donde cada servicio se ejecuta en su propio proceso y se comunica a través de mecanismos ligeros, a menudo usando APIs Http (Fowler, M., Lewis, 2018)(Lewis, James; Fowler, 2014)[12]. Esto permite dividir sistemas complejos en múltiples componentes operacionales pequeños e independientes (Liu et al., 2016).

Relación entre SOA y microservicios

Los microservicios están relacionados con la arquitectura SOA al punto de que existe un debate sobre si los microservicios son una arquitectura nueva o son más bien una subcategoría (caso especial) de la arquitectura SOA (Quenum & Aknine, 2018)(Vera-Rivera, 2018). Ambas arquitecturas están estrechamente relacionadas,

puesto que los microservicios están basados en SOA, razón por la que a menudo se considera a los microservicios como una versión de SOA para sistemas distribuidos (Vera- Rivera, 2018) o simplemente una versión extendida de SOA (Quenum & Aknine, 2018). No obstante, sí existen elementos que las diferencian. En el caso de los servicios SOA, los componentes se encapsulan en una única interfaz. Además, en la arquitectura SOA es necesaria una orquestación, que es facilitada por el Enterprise Service Bus. Este tiene la responsabilidad de ser el punto de integración para las comunicaciones con los servicios (Quenum & Aknine, 2018). La arquitectura de microservicios, por el contrario, no necesita del Enterprise Service Bus, dada la capacidad de independencia de los servicios que la componen. Los microservicios se diferencian por el nivel de granularidad de los servicios y su eliminación de la gobernanza central (de Camargo et al., 2016). Quizás la principal diferencia entre las arquitecturas SOA y microservicios es el propósito con el que fueron creadas (Heinrich et al., 2017): los microservicios surgen como arquitectura emergente ante la necesidad de aplicaciones de servicios con mayor capacidad de escalabilidad e independencia, mientras que la arquitectura SOA tiene características de centralización de los servicios controlado por el bus de servicios empresariales (Cerny et al., 2017) (Zúñiga-Prieto, Insfran, Abrahão, & Cano-Genoves, 2017)[13].

Pruebas end-to-end para microservicios

Las pruebas end-to-end surgieron en la última década como una herramienta valiosa para diagnosticar problemas de corrección y rendimiento en sistemas distribuidos (Las-Casas, Mace, Guedes, & Fonseca, 2018)[14]. Este tipo de pruebas se consideran pruebas funcionales de caja negra (Sotomayor et al., 2019), puesto que emulan el funcionamiento real del sistema y verifican su comportamiento (García, Gallego, Gortazar, & López, 2017). En el contexto de microservicios, las pruebas E2E son relevantes por ser la mejor manera de evaluar si el sistema como un todo funciona apropiadamente (Lei, Liao, Jiang, Yang, & Li, 2019).

Un aspecto clave de las pruebas E2E radica en las métricas y el “trazado” que pueden generar, los cuales permiten a los desarrolladores tomar decisiones sobre el rendimiento del sistema o la identificación de fallas. El “trazado” de las pruebas se refiere a la representación visual del flujo (Shahin, Babar, Zahedi, & Zhu, 2017), donde se muestra información como el orden de las información como el orden de las invocaciones, los eventos ejecutados, y la relación entre componentes y errores (Las-Casas et al., 2018). Otras métricas

Revista Innovación y Software
Vol. 4, No. 1, Mes Marzo-Agosto, 2023
ISSN: 2708-0935
Pág. 151-160
https://revistas.ulasalle.edu.pe/innosoft

importantes son las cargas de trabajo, y el uso de recursos y tiempo (Shahin et al., 2017). Estas métricas ayudan a comprender cómo funciona el sistema, a detectar anomalías en tiempo de ejecución, y a analizar por qué está fallando (Las-Casas et al., 2018).

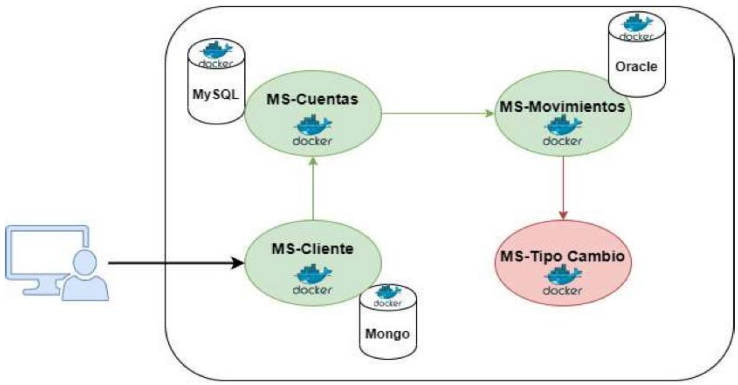

Figura 1: Ejemplo de arquitectura de microservicios para consultar el estado de cuenta

Prueba de la unidad

Una práctica que a menudo se pasa por alto cuando se prueban microservicios es la prueba unitaria. ¿Qué son las pruebas unitarias? Estas pruebas verifican que los métodos y clases que los desarrolladores escriben funcionan como se esperaba. Si bien las pruebas unitarias son una tarea muy técnica para los desarrolladores, un conjunto sólido de pruebas unitarias proporciona una red de seguridad crítica para detectar consecuencias no deseadas cuando los desarrolladores cambian el código. Y paga dividendos al alertar a los desarrolladores exactamente en qué parte del código han roto la funcionalidad existente. Esta es una práctica valiosa para escribir software de alta calidad. Sin embargo, las pruebas unitarias por sí solas no son suficientes. Como analogía, el hecho de que todas las partes de un motor estén mecanizadas con una especificación perfecta no significa que el motor funcionará y funcionará como se espera.

Prueba de componentes

Esta prueba de microservicios no se concentra en cómo el desarrollador escribió el código de microservicios, sino que se enfoca en ejecutar el microservicio como una caja negra y probar el tráfico que se mueve a través de la interfaz. Desde la perspectiva de un único microservicio, ahora está probando el motor para asegurarse

de que cumple con sus requisitos. En la mayoría de los casos, está probando un servicio REST. Por lo que desea pruebas automatizadas que actúen como clientes del servicio, enviando varias solicitudes positivas y negativas al servicio y verificando las respuestas que devuelve el servicio. Un desafío es que puede ser complejo y difícil probar microservicios de forma aislada porque a menudo llaman a muchos otros microservicios para responder a la solicitud de su cliente de prueba. Para probar un microservicio, es posible que necesite docenas más implementadas y disponibles para que su microservicio hable y lo pruebe correctamente.

Pruebas de integración

Al usar la virtualización de servicios para simplificar y estabilizar las pruebas del microservicio como un componente individual, también desea probar que el microservicio funciona con los otros microservicios REALES involucrados. Los desarrolladores a menudo hacen esto en una etapa de "QA" o "integración", donde muchos de los sistemas requeridos en el ecosistema general se implementan e integran juntos. Con esta práctica de prueba, está comenzando a ensamblar el automóvil para asegurarse de que todas las piezas encajen y funcionen juntas, pero aún no lo está probando en la carretera.

Pruebas de extremo a extremo

También se llama prueba del sistema. En algún momento, una gran red de microservicios tiene puntos de entrada donde interactúan los usuarios finales de la aplicación. Por ejemplo, una aplicación de Netflix en su Apple TV habla con microservicios dentro del centro de datos de Netflix. Pero representan solo una pequeña parte de su funcionalidad principal, que son componentes pequeños e individuales responsables de cosas específicas como un servicio de recomendaciones, un servicio de transmisión de video, un servicio de detalles de cuenta, etc. Así que esta también es una oportunidad para probar microservicios. A menudo es dolorosamente lento y de alto mantenimiento probar estas interacciones automáticamente, ya sea web o móvil. Para las rutas críticas y los recorridos del usuario, es imprescindible, pero poder representar estas transacciones completas o de un extremo a otro desde la perspectiva del usuario final como una secuencia de llamadas de microservicio tiene muchos beneficios. Básicamente, elimina la interfaz de usuario y simula todas las llamadas a la API que la interfaz de usuario hace a su arquitectura de microservicios para que pueda

verificar que todos los microservicios funcionan juntos correctamente para el contexto de requisitos comerciales / de usuario final más amplio.

Pruebas de seguridad

Los piratas informáticos pueden explotar áreas bajo el paraguas de los microservicios. Por lo tanto, los desarrolladores deben probar los microservicios a fondo para que estén protegidos contra las vulnerabilidades de seguridad. Parasoft Jtest tiene tecnología de análisis de código estático que escanea el código fuente subyacente e identifica las debilidades de codificación segura para que los desarrolladores puedan corregirlas. Parasoft también permite a los evaluadores reutilizar los casos de prueba funcionales que han escrito en SOAtest para realizar pruebas de penetración del microservicio.

Pruebas de carga y rendimiento

Para asegurarse de que el microservicio pueda mantener los SLA (acuerdos de nivel de servicio), los desarrolladores deben comprender cómo funcionan los SLA bajo carga y también determinar los puntos de ruptura.

Resultados y discusión

Cinco consejos para probar microservicios

1. Considere cada servicio como un módulo de software. Realice pruebas unitarias en un servicio como lo haría con cualquier código nuevo. En la arquitectura de microservicios, cada servicio se considera una caja negra. Por lo tanto, pruebe cada uno de manera similar.
2. Determine los vínculos fundamentales en la arquitectura y pruébalos. Por ejemplo, si existe un vínculo sólido entre el servicio de inicio de sesión, la interfaz que muestra los detalles del usuario y la base de datos para obtener los detalles, pruebe estos vínculos.
3. No se limite a probar escenarios de caminos felices. Los microservicios pueden fallar y es importante simular escenarios de fallas para generar resiliencia en su sistema.
4. Haz tu mejor esfuerzo para probar en todas las etapas. La experiencia ha demostrado que los probadores que utilizan una combinación diversa de prácticas de prueba, comenzando en el desarrollo

y progresando a través de alcances de prueba más grandes, no solo aumentan la posibilidad de que los errores se revelen, sino que lo hacen de manera eficiente. Esto es particularmente cierto en entornos virtuales complicados donde existen pequeñas diferencias entre varias bibliotecas y donde la arquitectura de hardware subyacente puede producir resultados imprevistos e indeseables a pesar de la capa de visualización.

5. Utilice “pruebas canarias” en código nuevo y pruebe en usuarios reales. Asegúrese de que todo el código esté bien instrumentado. Y también utilice toda la supervisión que ofrece su proveedor de plataforma. Esto cumple con las pruebas de "cambio a la izquierda" con las pruebas de "cambio a la derecha" porque también está probando "en la naturaleza".

Tres estrategias para maximizar el ROI de las pruebas de microservicios

1. Aumente la calidad de la cobertura de prueba de API funcional con IA para garantizar que los servicios implementados cumplan con los requisitos.
2. Automatice flujos de trabajo complejos basados en eventos para acelerar las pruebas.
3. Mejorar el entorno de prueba para mejorar la confiabilidad y estabilidad de las pruebas.

Selección de herramientas de pruebas E2E para microservicios

Esta primera fase, correspondiente al primer objetivo específico de la investigación, requirió en primera instancia identificar las herramientas de pruebas E2E para microservicios que se iban a evaluar mediante el estándar IEEE-14102-2010. En segundo lugar, se definieron los criterios mediante los cuales se evaluarían las herramientas. En tercer lugar, se aplicaron los procesos definidos en el estándar, con el fin de evaluar y posteriormente seleccionar las mejores herramientas. A continuación, se describen estas tres actividades.

Aplicación del estándar IEEE 14102-2010

Para evaluar y seleccionar herramientas El estándar IEEE 14102-2010 se divide en cuatro procesos: preparación, estructuración, evaluación y selección. A continuación, se detallan las actividades realizadas para cada proceso.

1. Preparación: establecimos el objetivo de la evaluación y definimos los criterios de selección.

2. Estructuración: definimos la lista de candidatos, y establecimos los pesos para cada criterio de selección, según su importancia relativa. Además, definimos las características a evaluar por cada criterio. Por ejemplo: el criterio de métrica tiene un peso de 30 sobre 100; si la herramienta genera métricas de trazados y tiempos obtiene un valor de 30, si genera únicamente una de las dos, se le asigna 15, y si no genera ninguna obtiene 0.
3. Evaluación: evaluamos la lista de candidatos, valorando las herramientas según los criterios y pesos establecidos.
4. Selección: seleccionamos las dos herramientas con mejor puntaje, ya que son las herramientas que cuentan con las características más apropiadas para realizar pruebas end- to-end para microservicios. Las herramientas seleccionadas obtuvieron una calificación similar.

Conclusiones

Los fundamentos de las pruebas de microservicios no son nuevos en comparación con los servicios web tradicionales o las pruebas SOA, pero la importancia de hacerlo solo se ha vuelto más crítica en los sistemas modernos. Al repasar por los diferentes conceptos para obtener una mayor calidad en los microservicios, conocimos lo suficiente como para tener en cuenta muchos métodos para realizar eficazmente pruebas de software para microservicios, por ende aumentar su calidad para enfrentar a desafíos actuales.

Referencias

[1] Aderaldo, C. M., Mendonça, N. C., Pahl, C., & Jamshidi, P. (2017). Benchmark Requirements for Microservices Architecture Research. 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE), 8–13. https://doi.org/10.1109/ECASE.2017.4

[2] Estructuración: definimos la lista de candidatos, y establecimos los pesos para cada criterio de selección, según su importancia relativa. Además, definimos las características a evaluar por cada criterio. Por ejemplo: el criterio de métrica tiene un peso de 30 sobre 100; si la herramienta genera métricas de trazados y tiempos obtiene un valor de 30, si genera únicamente una de las dos, se le asigna 15, y si no genera ninguna obtiene 0.

[3] Antichi, G., & Rétvári, G. (2020). Full-Stack SDN: The Next Big Challenge? Proceedings of the Symposium on SDN Research, 48–54. https://doi.org/10.1145/3373360.3380834

[4] Arcuri, A. (2019). RESTful API Automated Test Case Generation with EvoMaster. ACM Trans. Softw. Eng. Methodol., 28(1), 3:1--3:37. https://doi.org/10.1145/3293455
\item BA, K., & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. 2.

[5] Gil, D. G., & D\'\iaz-Heredero, R. A. (2018). A Microservices Experience in the Banking Industry. Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings. https://doi.org/10.1145/3241403.3241418

[6] Gu, G., Hu, H., Keller, E., Lin, Z., & Porter, D. E. (2017). Building a Security OS With Software Defined Infrastructure. Proceedings of the 8th Asia-Pacific Workshop on Systems. https://doi.org/10.1145/3124680.3124720

[7] Harsh, P., Ribera Laszkowski, J. F., Edmonds, A., Quang Thanh, T., Pauls, M., Vlaskovski, R., ... Gallego Carrillo, M. (2019). Cloud Enablers For Testing Large-Scale Distributed Applications. Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion, 35– 42. https://doi.org/10.1145/3368235.3368838

[8] Hasselbring, W., & Steinacker, G. (2017). Microservice architectures for scalability, agility and reliability in e-commerce. Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings, 243–246. https://doi.org/10.1109/ICSAW.2017.11

[9] Heinrich, R., van Hoorn, A., Knoche, H., Li, F., Lwakatare, L. E., Pahl, C., ... Wettinger, J. (2017). Performance Engineering for Microservices: Research Challenges and Directions. Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, 223–226. https://doi.org/10.1145/3053600.3053653

[10] Heorhiadi, V., Rajagopalan, S., Jamjoom, H., Reiter, M. K., & Sekar, V. (2016). Gremlin: Systematic Resilience Testing of Microservices. Proceedings - International Conference on Distributed

[11] MARTÍNEZ HERNÁNDEZ, Cristian Fernando. Evaluación de una herramienta de pruebas end-to-end para microservicios implementados en Java y Node. js.

[12] Fowler, M., Lewis, J. (2018). Microservices. 1–15.

[13] Zúñiga-Prieto, M., Insfran, E., Abrahão, S., & Cano-Genoves, C. (2017). Automation of the incremental integration of microservices architectures. In Lecture Notes in Information Systems and Organisation (Vol. 22, pp. 51–68). https://doi.org/10.1007/978-3-319-52593-8_4

[14] Las-Casas, P., Mace, J., Guedes, D., & Fonseca, R. (2018). Weighted Sampling of Execution Traces: Capturing More Needles and Less Hay. Proceedings of the ACM Symposium on Cloud Computing, 326–332. https://doi.org/10.1145/3267809.3267841

Roles de Autoría

Cesar Adolfo Laura Mamani: Conceptualización, Investigación, Metodología, Redacción - borrador original.