UNIVERSIDAD DE COSTA RICA
SISTEMA DE ESTUDIOS DE POSGRADO

EVALUACIÓN DE UNA HERRAMIENTA DE PRUEBAS *END-TO-END* PARA MICROSERVICIOS IMPLEMENTADOS EN JAVA Y NODE.JS

Trabajo final de investigación aplicada sometido a la consideración de la Comisión del Programa de Estudios de Posgrado en Computación e Informática para optar al grado de Maestría Profesional en Computación e Informática

CRISTIAN MARTÍNEZ HERNÁNDEZ

Ciudad Universitaria Rodrigo Facio
2020

Dedicatoria

Este trabajo está dedicado a mi esposa, quien ha estado a mi lado en todo este tiempo apoyándome continuamente para lograr mis objetivos y metas. A mis padres por haberme educado como la persona que soy; muchos de mis logros se los debo a ellos.

Agradecimientos

En primera instancia deseo expresar mi más profundo agradecimiento a la profesora Alexandra Martínez Porras, por ser una excelente guía y pilar para poder concluir con este trabajo. Agradezco también a los profesores Christian Quesada López y Marcelo Jenkins Coronas por el apoyo brindado durante el desarrollo de la investigación.

Además, agradezco a la Universidad de Costa Rica dejarme formar parte de esta prestigiosa institución.

Hoja de aprobación

"Este Trabajo Final de Investigación Aplicada fue aceptado por la comisión del programa de Estudios de Posgrado en Computación e Informática de la Universidad de Costa Rica, como requisito parcial para optar al grado y título de Maestría Profesional en **Computación e Informática**".

Dr. Vladimir Lara Villagrán
Representante del Decano
Sistema de Estudios de Posgrado

Dra. Alexandra Martínez Porras
Profesora Guía

Dr. Christian Quesada López
Asesor

Dr. Marcelo Jenkins Coronas
Asesor

Dra. Gabriela Marín Raventós
Directora del Programa de Posgrado en Computación e Informática

Cristian Martínez Hernández
Sustentante

IV

Tabla de contenidos

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. Por lo tanto, nuestra meta es contribuir a esta área mediante la evaluación de dos herramientas que apoyen las pruebas *end-to-end* de microservicios. Las pruebas *end-to-end* permiten verificar si el sistema funciona correctamente como un todo, lo cual es particularmente relevante en sistemas implementados con microservicios.

En este trabajo, primero identificamos un conjunto de 40 herramientas de pruebas E2E reportadas tanto en la literatura académica como por expertos de la industria. Luego aplicamos el estándar IEEE 14102-2010 para realizar una evaluación inicial de las herramientas identificadas, con base en características que necesitan tener herramientas de pruebas *end-to-end* para microservicios. Las dos herramientas mejor evaluadas, Jaeger y Zipkin, fueron seleccionadas para una evaluación más a fondo de su efectividad, mediante un caso de estudio. La efectividad contempló las dimensiones de eficiencia y eficacia. Las medidas de eficiencia usadas fueron el tiempo de ejecución y el tiempo de detección de fallas. Por su parte, la eficacia fue medida en términos de la capacidad de la herramienta para detectar fallas, proveer información para inferir la severidad de las fallas, y brindar cobertura de flujo.

A partir de los resultados obtenidos en nuestra primera etapa de identificación de herramientas utilizadas para pruebas *end-to-end* de microservicios, encontramos que muchas de las herramientas reportadas realmente no son herramientas especializadas para pruebas de microservicios, sino que son herramientas genéricas de pruebas utilizadas en el contexto de microservicios. Por este motivo, muchas de ellas carecen de las características necesarias para realizar de forma apropiada pruebas *end-to-end* de microservicios, que realmente brinden información valiosa a los desarrolladores que las utilicen.

Por otro lado, los resultados de nuestro caso de estudio indican que la herramienta Jaeger es más eficiente y eficaz que Zipkin, puesto que en la mayoría de los escenarios de prueba exhibió menor tiempo de ejecución en las pruebas y menor tiempo en detección de fallas. También Jaeger ofrece más información que Zipkin sobre el trazado de las pruebas y dependencias entre los microservicios, lo cual ayuda a detectar fallas más fácilmente, aunque ambas proveen cobertura de flujo y detectan las fallas.

UNIVERSIDAD DE COSTA RICA

SEP Sistema de Estudios de Posgrado

Autorización para digitalización y comunicación pública de Trabajos Finales de Graduación del Sistema de Estudios de Posgrado en el Repositorio Institucional de la Universidad de Costa Rica.

Yo, Cristian Fernando Martínez Hernández, con cédula de identidad 304530347, en mi condición de autor del TFG titulado Evaluación de una herramienta de pruebas end-to-end para microservicios implementados en Java y Node.js

Autorizo a la Universidad de Costa Rica para digitalizar y hacer divulgación pública de forma gratuita de dicho TFG a través del Repositorio Institucional u otro medio electrónico, para ser puesto a disposición del público según lo que establezca el Sistema de Estudios de Posgrado. SI [X] NO *[ ]

***En caso de la negativa favor indicar el tiempo de restricción: _____ año (s).**

Este Trabajo Final de Graduación será publicado en formato PDF, o en el formato que en el momento se establezca, de tal forma que el acceso al mismo sea libre, con el fin de permitir la consulta e impresión, pero no su modificación.

Manifiesto que mi Trabajo Final de Graduación fue debidamente subido al sistema digital Kerwá y su contenido corresponde al documento original que sirvió para la obtención de mi título, y que su información no infringe ni violenta ningún derecho a terceros. El TFG además cuenta con el visto bueno de mi Director (a) de Tesis o Tutor (a) y cumplió con lo establecido en la revisión del Formato por parte del Sistema de Estudios de Posgrado.

INFORMACIÓN DEL ESTUDIANTE:

Nombre Completo: Cristian Fernando Martínez Hernández

Número de Carné: B03714 Número de cédula: 304530347

Correo Electrónico: crismh2@hotmail.com / crismh12@gmail.com

Fecha: 15-Enero-2021 Número de teléfono: 84442033

Nombre del Director (a) de Tesis o Tutor (a): Dra Alexandra Martínez Porras

CRISTIAN FERNANDO MARTINEZ HERNANDEZ (FIRMA)

Firmado digitalmente por CRISTIAN FERNANDO MARTINEZ HERNANDEZ (FIRMA)
Fecha: 2021.01.15 11:09:39 -06'00'

FIRMA ESTUDIANTE

Nota: El presente documento constituye una declaracion jurada, cuyos alcances aseguran a la Universidad, que su contenido sea tomado como cierto. Su importancia radica en que permite abreviar procedimientos administrativos, y al mismo tiempo genera una responsabilidad legal para que quien declare contrario a la verdad de lo que manifiesta, puede como consecuencia, enfrentar un proceso penal por delito de perjurio, tipificado en el articulo 318 de nuestro Codigo Penal. Lo anterior implica que el estudiante se vea forzado a realizar su mayor esfuerzo para que no sólo incluya información veraz en la Licencia de Publicacion, sino que también realice diligentemente la gestión de subir el documento correcto en la plataforma digital Kerwa.

Capítulo 1

Introducción

Los microservicios son un estilo arquitectónico utilizado para el diseño e implementación de sistemas distribuidos (Quenum & Aknine, 2018), que ha tenido auge durante la última década. Evidencia de ello es la adopción que ha tenido por parte de compañías como Netflix, Facebook, Twitter, Amazon, Spotify, eBay, Deutsche Telekom, LinkedIn, SoundCloud, Uber y Verizon (Quenum & Aknine, 2018)(Liu et al., 2016)(Larrucea, Santamaria, Colomo-Palacios, & Ebert, 2018)(Bogner, Fritzsch, Wagner, & Zimmermann, 2019)(Heorhiadi, Rajagopalan, Jamjoom, Reiter, & Sekar, 2016). Además, se proyecta que para el 2021, un 80% de las aplicaciones desarrolladas en plataformas de la nube serán construidas con la arquitectura de microservicios (Larrucea et al., 2018).

Se considera que la base (o predecesor) de la arquitectura de microservicios es la arquitectura orientada a servicios (SOA, por sus siglas en inglés: *service-oriented arquitecture*) (Fowler, M., Lewis, 2018)(Lewis, James; Fowler, 2014). Las arquitecturas monolíticas estilo SOA se utilizaban tradicionalmente para desarrollar el *backend* del software, y se caracterizaban por ser arquitecturas centralizadas, con un coordinador denominado "bus de servicios empresariales" (ESB, por sus siglas en inglés: *Enterprise Service Bus*), que dirigía la *orquestación* (Cerny, Donahoo, & Pechanec, 2017). Por su parte, las arquitecturas de microservicios se enfocan más bien en desarrollar una aplicación como un conjunto de servicios pequeños, independientes y con mecanismos ligeros de comunicación (a menudo usando APIs http), donde la necesidad de gestión centralizada es mínima, por lo que esta arquitectura se conoce como *coreografía* (en contraposición a la *orquestación* de SOA). La arquitectura de microservicios permite que los servicios puedan ser escritos en diferentes lenguajes de programación y con diferentes tecnologías de almacenamiento de datos (al ser independientes) (Fowler, M., Lewis, 2018)(Lewis, James; Fowler, 2014). Esto da 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).

A pesar de ser una arquitectura con amplia aceptación por parte de la industria, los microservicios aún presentan una serie de retos, entre los que destacan la diversidad de tecnologías, la complejidad de las aplicaciones, la coreografía, y las pruebas de software. La diversidad de tecnologías se refiere a que una aplicación puede tener componentes de software creados en diferentes lenguajes de programación,

característica que permite aprovechar los beneficios de las diferentes tecnologías, pero a la vez se puede convertir en un reto de gobernabilidad de las tecnologías utilizadas (Chen, 2018). Por su parte, la complejidad de las aplicaciones constituye un reto debido a que potencialmente miles de servicios podrían ejecutarse de forma asincrónica en una red distribuida, lo cual a su vez requiere de un sistema elástico que se adapte a la carga de trabajo para aumentar el rendimiento de las aplicaciones (los microservicios pueden duplicar y distribuir servicios a un clúster de servidores, pero la elección de dónde duplicar y distribuir puede tener un gran impacto en el rendimiento) (Liu et al., 2016). Otro de los retos, la coreografía, se refiere a la capacidad de los componentes de microservicios de invocar la funcionalidad de otros componentes. La gestión de estas invocaciones puede representar un desafío, especialmente cuando se dan invocaciones complejas (S.-P. Ma et al., 2018), que son secuencias de invocaciones de diferentes componentes de los microservicios. Finalmente, las pruebas en el contexto de microservicios representan un desafío debido a que las aplicaciones pueden contener cientos o miles de componentes que operan en conjunto (de Camargo, Salvadori, Mello, & Siqueira, 2016) y cada uno de ellos debe ser probado a medida que cambian, considerando características como el lenguaje de programación y la infraestructura. Los microservicios demandan entonces un enfoque diferente para las pruebas de sus componentes. La automatización es el camino para enfrentar muchos de los desafíos asociados a las pruebas de microservicios (Heinrich et al., 2017)(de Camargo et al., 2016)(Gazzola, Goldstein, Mariani, Segall, & Ussi, 2020) y para ello es necesario utilizar herramientas que apoyen y faciliten la creación y ejecución de dichas pruebas automatizadas.

El modelo de la pirámide de automatización de las pruebas introducido por Mike Cohn (Contan, Dehelean, & Miclea, 2018)(Cohn, 2009)(Vocke, 2018), propone que un sistema debe probarse en tres capas: pruebas unitarias en la base de la pirámide, pruebas de servicio (o integración) en el medio, y pruebas de interfaz de usuario (o aceptación) en la cima de la pirámide. Las pruebas unitarias automatizadas se enfocan en probar piezas independientes de código de forma aislada. Las pruebas de servicio consisten en la combinación de pruebas de API e integración. Por último, las pruebas de interfaz de usuario corresponden, en el caso de los microservicios, a la verificación del flujo del sistema de extremo a extremo (E2E, por sus siglas en inglés: *end-to-end*) (Quenum & Aknine, 2018)(de Camargo et al., 2016). En el contexto de los microservicios, se han estudiado especialmente las pruebas unitarias (Quenum & Aknine, 2018)(S.-P. Ma et al., 2018)(Shang-Pin Ma, Fan, Chuang, Liu, & Lan, 2019)(Stefano Munari, Sebastiano Valle, 2018) y de integración (Jindal, Podolskiy, & Gerndt, 2019)(Arcuri, 2019)(Du et al., 2018)(Nagarajan & Vaddadi, 2016)(Kargar & Hanifizade, 2018). Sin embargo, las pruebas *end-to-end* no han sido tan

estudiadas, a pesar de su relevancia en el contexto de los microservicios [1]. Las pruebas E2E son importantes porque permiten verificar el funcionamiento del sistema comparándolo contra el comportamiento esperado de los casos de uso (Shang-Pin Ma et al., 2019). De esta manera, verifican si el sistema cumple con los requisitos y logra sus objetivos, probando el sistema como un todo.

[1] Estrategia de prueba en una arquitectura de microservicios https://martinfowler.com/articles/microservice-testing/

1.1Justificación

Realizar pruebas de microservicios plantea retos adicionales comparado con la realización de pruebas de software tradicional (Eismann, Bezemer, Shang, Okanović, & van Hoorn, 2020), una forma de abordar los retos es por medio la automatización de pruebas, las cuales permiten entregar componentes en períodos cortos. 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.

Las pruebas *end-to-end* para microservicios han sido poco estudiadas, a pesar de que los microservicios resultan bastante difíciles de probar mediante métodos tradicionales (manuales) (Meinke & Nycander, 2015). De ahí que vemos la necesidad de investigar sobre herramientas que soporten pruebas de tipo *end-to-end*, para que nuestra investigación pueda beneficiar a profesionales de la industria que quieran realizar este tipo de pruebas en el contexto de sus organizaciones. Adicionalmente, el hecho de enfocar nuestra investigación hacia las tecnologías predominantes de microservicios (Java y Node.js) (Bogner et al., 2019)(de Camargo et al., 2016) nos permitirá tener un mayor impacto en la industria, promoviendo la adopción de herramientas de pruebas *end-to-end* para microservicios.

Nuestra investigación consistirá entonces en hacer primeramente una identificación de herramientas existentes para *end-to-end* de microservicios, delimitadas a aquellas que soporten microservicios implementados en Java o Node.js. Una vez identificadas las herramientas de interés, se aplicará el estándar IEEE 14102-2010 para elegir una herramienta a evaluar. La herramienta seleccionada será evaluada en términos de su efectividad, usando microservicios desarrollados en Java y Node.js (Ueda, Nakaike, & Ohara, 2016) y hospedados en Docker.

Nuestras principales contribuciones serán (1) una lista de herramientas existentes para realizar pruebas *end-to-end* de microservicios, haciendo una caracterización de las mismas (mediante los criterios del estándar), y (2) una evaluación de la efectividad de una herramienta para pruebas *end-to-end* de microservicios, mediante un caso de estudio. Para la academia, el estudio es valioso por la compilación de herramientas existentes para pruebas de microservicios, que podría ser de utilidad a otros investigadores

que deseen evaluar o analizar alguna de estas herramientas. Similarmente, esta compilación de herramientas ayudaría a profesionales de la industria u organizaciones de desarrollo de software que estén considerando la adopción de la arquitectura de microservicios, o la automatización de sus pruebas. Esperamos que este estudio también sirva como punto de partida para evaluar la efectividad de más herramientas a futuro.

1.2 Problema

Dado que las pruebas de software tienen un papel tan importante para la calidad del software, y que estas presentan retos importantes en el contexto de sistemas distribuidos que utilizan microservicios, esta investigación aborda el problema de la efectividad de una herramienta de pruebas *end-to-end* para microservicios. El alcance de la investigación se delimita a microservicios implementados en Java o Node.js debido a que son las tecnologías predominantes en la industria para crear servicios bajo la arquitectura de microservicios (Bogner et al., 2019)(de Camargo et al., 2016).

1.3 Objetivos

1.3.1 Objetivo general

Evaluar la efectividad de una herramienta de pruebas *end-to-end* para microservicios implementados en Java y Node.js.

1.3.2 Objetivos específicos

1. Seleccionar, mediante el estándar IEEE 14102-2010, una herramienta de pruebas *end-to-end* para microservicios implementados en Java y Node.js, a ser evaluada.

2. Evaluar la efectividad de la herramienta seleccionada mediante un caso de estudio.

El primer objetivo específico busca seleccionar una herramienta utilizada por la academia o la industria para pruebas e*nd-to-end* de microservicios implementados en Java y Node.js. El segundo objetivo específico busca evaluar la efectividad de la herramienta seleccionada para pruebas *end-to-end* en un sistema distribuido que utilice microservicios, mediante un caso de estudio.

El presente documento se divide de la siguiente manera: El capítulo 2 presenta un marco teórico que aborda conceptos y fundamentos asociados a la investigación. El capítulo 3 muestra los trabajos relacionados a la presente investigación, desde un enfoque asociado a herramientas utilizadas para probar microservicios. El capítulo 4 detalla la metodología de trabajo utilizada para llevar a cabo la realización del estudio. El capítulo 5 describe el diseño del caso de estudio, profundizando sobre el proceso e instrumentos utilizados en la identificación, la selección y la evaluación de la herramienta. El capítulo 6, detalla y analiza los resultados obtenidos de evaluar la herramienta en un contexto de microservicios. Por último, el capítulo 7 presenta las conclusiones de la investigación y trabajos futuros que pueden derivarse de nuestro estudio.

Capítulo 2

Marco teórico

En esta sección se describen los conceptos y fundamentos sobre los que se basa esta investigación.

2.1 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.

2.1.1 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 utilizadas 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.

2.1.2 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). Esto permite dividir sistemas complejos en múltiples componentes operacionales pequeños e independientes (Liu et al., 2016). El nivel de independencia de la arquitectura de microservicios permite que los servicios puedan ser implementados con diferentes lenguajes de programación y delega la gestión de los datos a cada servicio.

Los microservicios brindan mayores ventajas sobre aplicaciones desarrolladas con arquitecturas monolíticas, ya que al ejecutarse cada proceso de forma independiente en contenedores únicos y aislados, es posible asignar más recursos computacionales a los servicios que más lo necesitan, a diferencia de las aplicaciones monolíticas, que asignan recursos a toda la aplicación, por lo cual muchas veces los recursos no pueden aprovecharse plenamente (Vera-Rivera, 2018).

Diferentes sectores de la industria han comenzado a utilizar microservicios, incluso los más tradicionales y conservadores, como el sector financiero (Gil & D\'\iaz-Heredero, 2018). El éxito de este estilo arquitectónico se debe a sus características, donde destacan la implementación de microservicios como piezas pequeñas e independientes con un solo propósito de negocio (Yuan, 2019) (generalmente desarrollados como RESTful APIs), y la posibilidad de implementarse en diferentes lenguajes de programación (Fowler, M., Lewis, 2018).

2.1.3 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).

2.2 Implementación de microservicios

Cada microservicio es desplegado mediante una tecnología de virtualización ligera, conocida como “contenedores”. Esta tecnología se caracteriza por compartir el *kernel* del sistema operativo y muchos recursos del hardware entre los componentes ubicados conjuntamente, lo que reduce la sobrecarga resultante [32][33]. Además, cada contenedor cuenta con un entorno de ejecución completo, de manera que un microservicio o aplicación creada en un contenedor tiene todas las dependencias, bibliotecas y cualquier archivo binario requerido para su ejecución (Singh, Gaba, Kaur, & Kaur, 2019), propiciando las

propiedades de aislamiento, aprovisionamiento de recursos, y configuraciones específicas de servicios o aplicaciones (Rastogi, Davidson, De Carli, Jha, & McDaniel, 2017).

Para administrar la tecnología de contenedores se desarrollaron herramientas como Docker, Kubernetes, Mesos, Amazon EC2 y OpenWhisk (Barna, Khazaei, Fokaefs, & Litoiu, 2017)(Li et al., 2020)(Antichi & Rétvári, 2020). Cabe mencionar que las herramientas Amazon EC2 y OpenWhisk fueron desarrollados por Amazon, utilizando tecnología de Docker (Shahrad, Balkind, & Wentzlaff, 2019). Adicionalmente, debe considerarse que estos entornos pueden contar con cientos o miles de microservicios o aplicaciones, lo que implica el uso de herramientas complementarias para gestión de *clusters*, como por ejemplo Docker Swarm, Kubernetes, Cattle o Mesos (Aderaldo, Mendonça, Pahl, & Jamshidi, 2017)(Jawarneh et al., 2019). Estas herramientas apoyan la orquestación de contenedores, controlan el balanceo de cargas y permiten las actualizaciones graduales (Aderaldo et al., 2017). En la industria la herramienta para gestión de *clusters* de contenedores más destacada es Kubernetes (desarrollada por Google) (Jawarneh et al., 2019). Asimismo, Docker es considerada el referente en virtualización de contenedores desde que se popularizó en el 2013 (Singh et al., 2019)(Ibrahim, Bozhinoski, & Pretschner, 2019), ya que mejora la utilización de recursos de sus contenedores en relación con las máquinas virtuales, haciendo que los contenedores sean más fáciles y rápidos de crear, probar e implementar (Bin, Shulin, Xuelei, & Guyang, 2017)(Soenen et al., 2018). Considerando su relevancia en la industria y las ventajas que ofrece, en el presente estudio utilizamos Docker para gestionar nuestros contenedores.

Otra característica de los microservicios es su capacidad de utilizar diferentes lenguajes de programación, permitiendo aprovechar las ventajas de cada lenguaje. Por ejemplo, se puede implementar un microservicio para gestión de imágenes con Java y otro para gestión de información en C#, suponiendo que estos lenguajes cuenten con características para optimizar estas funciones. Algunos de los lenguajes de programación que se pueden utilizar para implementar microservicios son: C#, Java, Node.js, Go, Python, Scala, Swift, PHP, y Ruby (Yuan, 2019)(Shahrad et al., 2019). No obstante, los lenguajes más populares para la implementación de esta arquitectura son Java (con el framework Spring Boot) (de Camargo et al., 2016)(Yuan, 2019) y Node.js (Bogner et al., 2019). Por esta razón, nuestra investigación se enfoca en microservicios implementados en estos dos últimos lenguajes de programación.

2.3 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). 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 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).

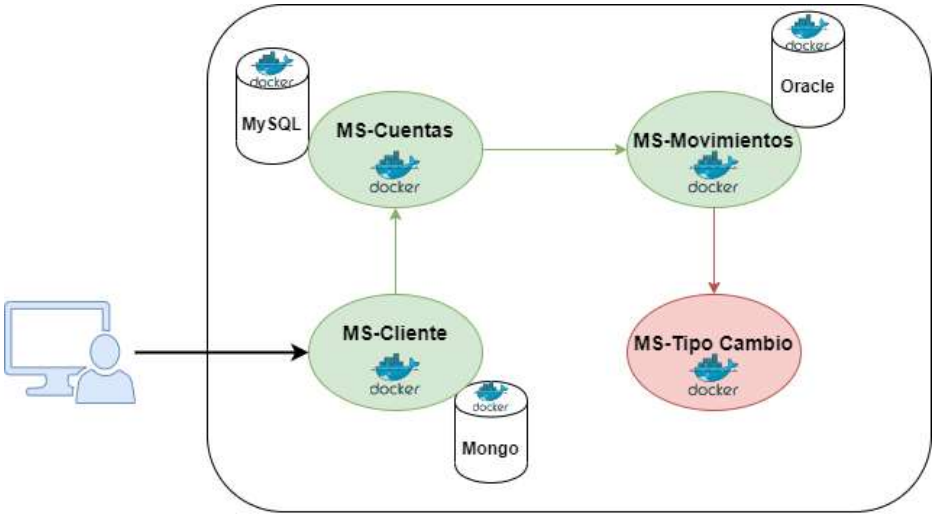


Figura 1: Ejemplo de arquitectura de microservicios para consultar el estado de cuenta.

La figura 1 muestra un ejemplo de una arquitectura de microservicios para consultar el estado de cuenta de un cliente. La arquitectura cuenta con 4 microservicios (MS-Cliente, MS-Cuentas, MS-

Movimientos, MS-Tipo Cambio) y 7 contenedores Docker (MS-Cliente, MS-Cuentas, MS-Movimientos, MS-Tipo Cambio, Mongo, MySQL, Oracle). En este contexto, un escenario de prueba podría ser obtener los saldos de las cuentas de un cliente, y para ello se invocarían los microservicios de MS-Cliente, MS-Cuenta, MS-Movimientos, MS-Tipo Cambio. Ahora bien, suponiendo que el MS-Tipo Cambio no se encuentra disponible, al ser todos los microservicios independientes, no hay forma de que los otros microservicios conozcan su estado. Por lo tanto, en caso de no contar con una herramienta para pruebas *end-to-end*, la respuesta del *back-end* sería un error, pero no se contaría con información para saber cuál microservicio falló ni cómo fue que falló.

2.5 Estándar IEEE 14102-2010

La organización internacional IEEE creó el estándar ISO/IEC 14102:2010, denominado *IEEE Standard for Adoption of ISO/IEC 14102:2010 Information Technology – Guideline for the Evaluation and Selection of CASE Tools*. Para su creación, este estándar tomó como base el ISO/IEC 14102:2008, que define un conjunto de procesos y características para herramientas CASE, utilizadas en la evaluación y selección de herramientas. Además, incorpora el modelo de evaluación de productos de software ISO/IEC 14598-5:1998 (Board, 2010).

El estándar ISO/IEC 14102:2010 está compuesto de cuatro procesos principales: el proceso de preparación, el proceso estructuración, proceso de evaluación y proceso de selección. El objetivo del estándar consiste en crear un estándar que permita ser una guía para obtener la herramienta apropiada. De esta manera se logrará reducir costos, mejorar la calidad y velocidad del desarrollo de software (Raulamo-Jurvanen, Hosio, & Mäntylä, 2019).

El estándar IEEE 14102:2010 es el mecanismo que utilizamos en este estudio para la evaluación inicial y selección de las herramientas de pruebas de microservicios.

Capítulo 3

Trabajo relacionado

En esta sección presentamos trabajos previos relacionados con nuestra temática de estudio, agrupados en, investigaciones sobre los tipos de pruebas más adecuados en el contexto de microservicios, investigaciones que presentan herramientas para probar microservicios, investigaciones que indican características deseables en estas herramientas, y finalmente, investigaciones que señalan los retos existentes asociados a las pruebas de microservicios.

3.1 Tipos de pruebas para microservicios

La literatura sugiere que los tipos de pruebas para microservicios son las unitarias (Lei et al., 2019), *end-to-end* (Lei et al., 2019), resiliencia (Heorhiadi et al., 2016) y regresión (Kargar & Hanifizade, 2018). Las pruebas unitarias tienen como objetivo probar de forma independiente cada microservicio, sin considerar otras partes de la aplicación (de Camargo et al., 2016). En el caso de las pruebas *end-to-end*, se consideran importantes debido a que es la mejor manera de comprobar si todo el sistema funciona bien como un todo (Lei et al., 2019). Por otra parte, las pruebas de regresión permiten comparar una nueva versión contra la versión previa, usando métricas como el uso de recursos, los fallos del sistema o el rendimiento del sistema (Kargar & Hanifizade, 2018). La resiliencia, por su parte, nos permite verificar la capacidad de tolerancia a fallos de los componentes (Heorhiadi et al., 2016).

Quenum et al. (Quenum & Aknine, 2018) presentan una propuesta inspirada en el modelo piramidal para la automatización de pruebas. El modelo indica que la base piramidal son las pruebas unitarias. La siguiente capa son las pruebas de servicio, las cuales incluyen pruebas de integración. Finalmente, en la cima de la pirámide se encuentran las pruebas *end-to-end*. Por su parte, Arcuri (Arcuri, 2019) destaca la necesidad de involucrar a las partes interesadas (desarrolladores, proveedores, integradores o certificadores externos), en la definición de pautas para los tipos de pruebas que se deben ejecutar.

3.2 Herramientas para pruebas de microservicios

Con respecto a las herramientas existentes para probar microservicios, encontramos una gran variedad, entre ellas: Hystrix (Vera-Rivera, 2018), Proxygen (Vera-Rivera, 2018), Resilience4J (Vera-Rivera, 2018), RSpec (Quenum & Aknine, 2018), Cucumber (Quenum & Aknine, 2018), Diffy (Ueda et al., 2016), EvoMaster (Nagarajan & Vaddadi, 2016), Chaos Monkey (Stefano Munari, Sebastiano Valle, 2018), GameDay (Nagarajan & Vaddadi, 2016), Screwdriver (Nagarajan & Vaddadi, 2016), Junit (de Camargo et al., 2016), Mockito (S.-P. Ma et al., 2018), Selenium (S.-P. Ma et al., 2018) y xUnit (Contan et al., 2018). Las herramientas RSpec, EvoMaster, Junit y xUnit se utilizan para la automatización de pruebas unitarias. Para las pruebas punto a punto se utiliza la herramienta de Selenium. Las herramientas Hystrix, Proxygen, Chaos Monkey, GameDay, Scewdriver y Resilience4J se utilizan para pruebas de resiliencia. Diffy es una herramienta para pruebas de regresión de microservicios, desarrollada por Twitter (Ueda et al., 2016).

Vera-Rivera (Vera-Rivera, 2018) presenta el proceso del desarrollo de una aplicación empresarial con microservicios, describiendo las fases de construcción (incluyendo la automatización de pruebas), prácticas, fundamentos, métodos y herramientas. El autor destaca el uso de las herramientas Hystix, Proxygen y Resilience4j para realizar pruebas de tolerancia a fallas, sobre una aplicación llamada Sinplafut, que ayuda en la planificación de entrenamiento de fútbol.

Quenum et al. (Quenum & Aknine, 2018) proponen la automatización de pruebas de microservicios por medio de agentes inteligentes, los cuales son componentes de software que capturan el comportamiento de los microservicios. En este estudio el enfoque es realizar pruebas de aceptación, para lo cual utilizan las herramientas RSpec para pruebas unitarias y Cucumber para las pruebas de aceptación.

Arcuri (Arcuri, 2019) crea una herramienta llamada EVOMASTER, la cual genera casos de prueba de forma automática, permitiendo obtener métricas de fallas del sistema o cobertura de código, para aplicaciones con la arquitectura de microservicios.

Sotomayor et al. (Sotomayor et al., 2019) proveen una clasificación de herramientas para pruebas de microservicios. Adicionalmente, evalúan las herramientas de código libre Gremlin, Hoverfly, Minikube y Telepresence con una aplicación llamada Rideshare, que está construida con diferentes lenguajes de

programación. Estas son utilizadas para realizar pruebas de componentes e integración basados en casos de uso. Una vez ejecutada las pruebas 10 veces con cada una de las herramientas, el estudio concluye que Telepresence es la aplicación que tarda más tiempo para ejecutar las pruebas. Sin embargo, Minikube también obtuvo altos tiempos de ejecución, por lo que se deduce que está relacionado con la configuración de comunicación con los servicios. Además, se indica que la herramienta con mejor tiempo de ejecución es Hoverfly. Otro aporte relevante de la investigación son los retos, el cual señala como desafíos la configuración de las herramientas para que funcionarán con la aplicación y los recursos necesarios para ejecutar tanto las herramientas como las aplicaciones.

3.3 Características deseables en herramientas que prueban microservicios

En cuanto a las características deseables en las herramientas que prueban microservicios, la literatura propone que sean independientes del lenguaje de programación, puesto que una aplicación con arquitectura de microservicios puede tener componentes desarrollados en diferentes lenguajes de programación (Heorhiadi et al., 2016). Nagarajan et al. (Nagarajan & Vaddadi, 2016) señalan que una herramienta para pruebas de tolerancia a fallas debe tener las siguientes características: capacidad para inyectar fallas de manera controlada, representación de la topología de sistemas, identificación de fallas apropiadas para inyectar, recuperación desde un estado de falla y métricas que representen el estado del sistema. En este estudio se introduce la herramienta Screwdriver como alternativa de GameDay o Chaos Monkey (Nagarajan & Vaddadi, 2016) .

Quenum et al. (Quenum & Aknine, 2018) señala que parte de la comunidad de ingenieros y arquitectos de software recomiendan usar técnicas de automatización de pruebas similares a las que se usan en SOA para microservicos. Sin embargo, la forma como se automatizan las pruebas en SOA no es adecuada para microservicios, ya que los componentes de los microservicios son independientes y cada uno puede tener su propia base de datos, mientras que los servicios SOA cuentan con nivel de centralización gestionado por el bus de servicios empresariales (Quenum & Aknine, 2018). Por lo tanto, nosotros creemos las herramientas para pruebas de microservicios deben adecuarse a estas necesidades.

En cuanto a las pruebas *end-to-end* de microservicios, encontramos diferentes propuestas de herramientas para su automatización. Sotomayor et al. (Sotomayor et al., 2019) en su clasificación de

herramientas utilizadas para apoyar las pruebas de microservicios menciona que Gremlin, Hoverfly, Spring Cloud Contract, Gatling, Selenium IDE y Docker Compose Rule pueden ser utilizadas para realizar estas pruebas. Schreiber (Schreiber, 2020) presenta una herramienta llamada PREvant, utilizada para proporcionar un API RESTful para implementar y componer microservicios en contenedores. Esta herramienta tiene la capacidad de automatizar pruebas E2E para comprobar el rendimiento de microservicios. Harsh et al. (Harsh et al., 2019) indicó que los sistemas de software complejos y con gran cantidad de componentes están proliferando debido al *cloud computing* y la necesidad de contar con aplicaciones elásticas, empujando a los desarrolladores hacia arquitecturas de software resistentes como los microservicios. Por lo tanto, las prácticas y herramientas deben evolucionar para apoyar las pruebas de estos nuevos enfoques. Para ello proponen un nuevo modelo de servicio de pruebas llamado *Testing as a Service* (TasS), y una herramienta llamada ElasTests, la cual permite realizar pruebas E2E. Meinke et al. (Meinke & Nycander, 2015) estudió como *learning-based testing* (paradigma emergente para la automatización total de pruebas de caja negra) puede ser utilizado para evaluar la exactitud funcional y la robustez a los fallos en un sistema distribuido. En su experimento utilizaron la herramienta LBTest, logrando concluir que el rendimiento de esta herramienta es mejor en relación a la utilización de técnicas manuales para pruebas E2E.

3.4 Retos asociados a las pruebas de microservicios

Finalmente, la literatura sugiere que existen muchos retos aún por resolver en esta área (Chen, 2018)(Contan et al., 2018). Por ejemplo, Camargo et al. (Contan et al., 2018) indican que las pruebas en el contexto de microservicios representan un desafío puesto que las aplicaciones pueden contener cientos o miles de componentes que operan en conjunto y cada uno de ellos debe ser probado a medida que cambian. El desacoplamiento de los servicios aumenta la complejidad de las actividades para realizar pruebas. Sin embargo, los autores proponen un modelo que permite automatizar la ejecución de pruebas de rendimiento mediante la herramienta de JMeter (Contan et al., 2018).

Heinrich et al. (Chen, 2018)(Heinrich et al., 2017) destaca la necesidad de sustituir y compensar las extensas pruebas de integración mediante la supervisión de entornos de producción. Además, recomienda alinear las pruebas de rendimiento y pruebas de regresión con prácticas de entrega continua, es decir, debe acelerarse las etapas de pruebas correspondientes (Chen, 2018). También, es necesario considerar características como el lenguaje de programación y la infraestructura (de Camargo et al., 2016).

La automatización es el camino para enfrentar muchos de los desafíos asociados a las pruebas de microservicios (de Camargo et al., 2016) (Heinrich et al., 2017) (Hasselbring & Steinacker, 2017), y para ello es necesario utilizar herramientas que apoyen y faciliten la creación y ejecución de dichas pruebas automatizadas.

Capítulo 4

Metodología

Esta sección presenta la metodología utilizada para desarrollar la investigación. La figura 2 muestra la metodología (junto con sus fases y actividades) asociada a cada objetivo específico de investigación. Seguidamente se detalla cada fase y sus actividades.

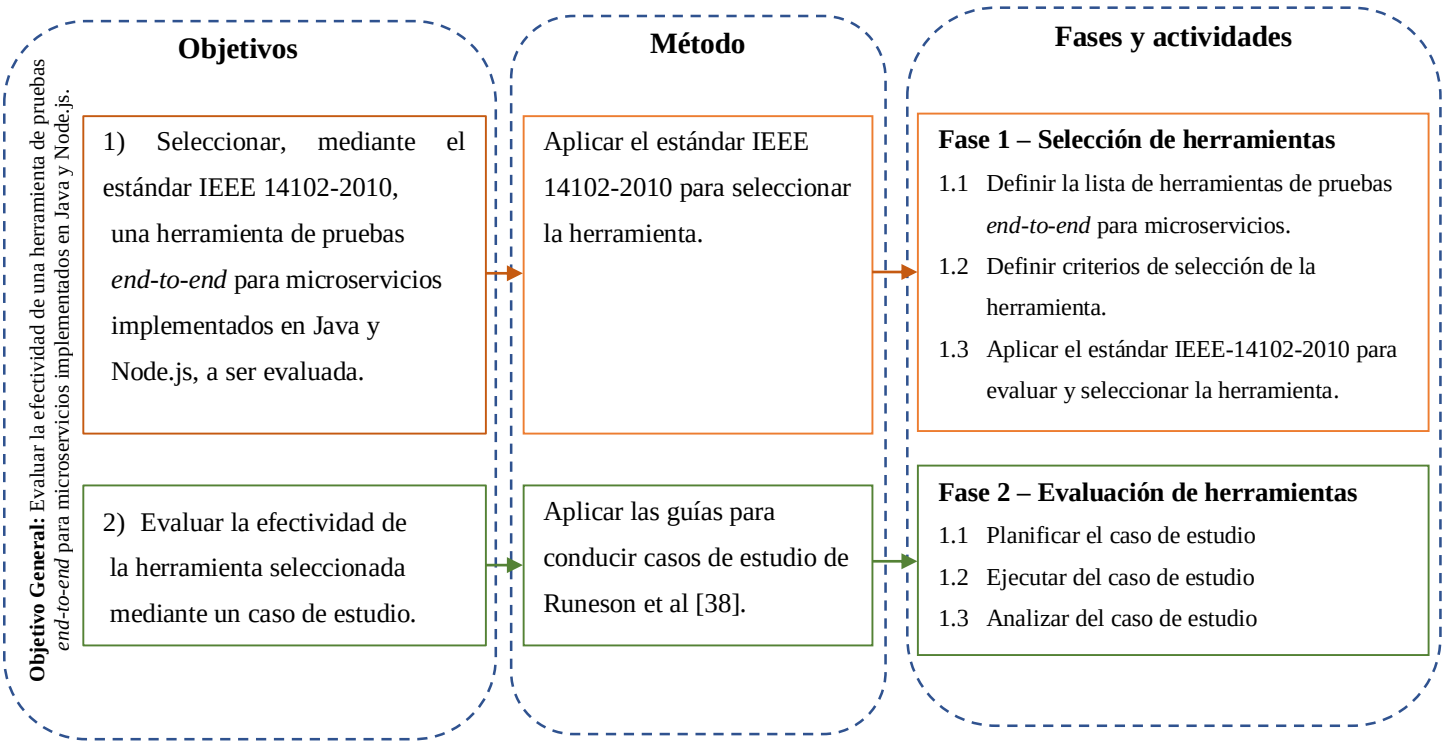


Figura 2: Metodología utilizada para la investigación (Runeson & Höst, 2009).

4.1 Fase I: 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.

4.1.1 Definición de la lista de herramientas

Con el fin de identificar las herramientas de pruebas E2E para microservicios a evaluar mediante el estándar IEEE-14102-2010, se realizó una revisión de literatura y adicionalmente se aplicaron dos encuestas a profesionales de la industria nacional e internacional.

La revisión de literatura se enfocó en herramientas reportadas en publicaciones académicas, utilizando algunos de los principios propuestos por Kitchenham y Petersen (BA & Charters, 2007). La búsqueda de artículos fue realizada en las bases de datos IEEE, ACM, Scopus y Springer, utilizando en la cadena de búsqueda los siguientes términos: *microservices*, *software*, *end-to-end*, *e2e*, *tool*, *test*. Los artículos fueron filtrados aplicando el siguiente criterio de inclusión: menciona o describe herramientas para pruebas *end-to-end*. Además, se usaron los siguientes dos criterios de exclusión: escrito en idioma diferente al inglés y/o no disponible en texto completo. La revisión de literatura se desarrolló en febrero de 2020. Como resultado, obtuvimos una lista inicial de herramientas de pruebas *end-to-end* reportadas en la literatura.

Para complementar la lista inicial de herramientas reportadas en la literatura académica, se procedió a identificar herramientas utilizadas por la industria. Para ello, se aplicó un cuestionario a los asistentes del evento *Dockerizing Microservices and Deploy them in Kubernetes*, realizado el 22 de enero de 2020 en Costa Rica por la comunidad de Docker. El cuestionario se implementó como un formulario de Google y está disponible en el siguiente enlace: https://forms.gle/i5SsmoRuMZoWL2RY8. Obtuvimos 6 respuestas, pero los resultados obtenidos no fueron los esperados, ya que los participantes no contaban con experiencia suficiente en la tecnología de microservicios, en sus pruebas.

Ante este escenario, recurrimos a expertos internacionales en microservicios, específicamente, a los *Docker Captains*[2], quienes son los especialistas más reconocidos en la comunidad de microservicios. Una vez identificados estos expertos (la lista completa de *capitanes* se encuentra en el Anexo 1), se les contactó vía correo electrónico, solicitándoles apoyo para responder un cuestionario. Este segundo

[2] https://www.docker.com/community/captains

cuestionario se tuvo que traducir a idioma inglés, y se modificó para eliminar algunas preguntas que ya no era tan relevantes (al ser ellos expertos en el tema) e incluir otras preguntas que nos interesaban. El mismo estuvo disponible entre los meses de mayo y junio de 2020 en el siguiente enlace: https://forms.gle/zZm7LCCFidPBoZqw9. En esta ocasión obtuvimos 7 respuestas, y la experiencia que reportaron los participantes en el uso de microservicios fue mucho mayor, por lo que los resultados de esta encuesta sí permitieron complementar la lista de herramientas de pruebas *end-to-end* para microservicios.

En síntesis, los resultados de la revisión de literatura y de la encuesta a expertos internacionales en microservicios, permitieron definir la lista final de herramientas a evaluar mediante el estándar IEEE-14102-2010.

4.1.2 Definición de los criterios de selección de herramientas

Para definir los criterios de selección a utilizar, se combinaron las características deseables en una herramienta de pruebas E2E para microservicios, con aspectos específicos de herramientas de software tipificados en un estudio previo (Powell, Vickers, Williams, & Cooke, 1996).

Se determinaron las siguientes características deseables en una herramienta de pruebas *end-to-end* para microservicios:

a. Se evaluaron las características de los microservicios como: independencia de componentes (microservicios), soporte de diferentes lenguajes de programación, soporte para *cloud computer*.
b. Característica deseable en el contexto de pruebas *end-to-end*: enfoque en métricas que dan valor a las pruebas.

Con base en la propuesta de Powell et al. (Powell et al., 1996) para una estrategia para evaluación de herramientas de software, se eligieron aspectos específicos que representaban idoneidad de la herramienta para su propósito, en el contexto de herramientas de pruebas *end-to-end* para microservicios.

Finalmente, se llegó a un conjunto de seis criterios con los cuales se evaluarían las herramientas: métricas, documentación, soporte, integrabilidad, costo y compatibilidad de plataformas.

4.1.3 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.

a. Preparación: establecimos el objetivo de la evaluación y definimos los criterios de selección (punto 4.1.2).
b. 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.
c. Evaluación: evaluamos la lista de candidatos, valorando las herramientas según los criterios y pesos establecidos.
d. 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.

4.2 Fase II: Evaluación de herramientas de pruebas E2E para microservicios

Esta fase corresponde al segundo objetivo específico de la investigación, y esencialmente consistió en planificar y ejecutar el caso de estudio y analizar sus resultados, siguiendo los lineamientos de Runeson et al. (Runeson & Höst, 2009). Seguidamente se describen las actividades realizadas.

4.2.1 Planificación del caso de estudio

Las tareas de planificación del caso de estudio fueron las siguientes:

1. Definición del objetivo del caso de estudio: el objetivo planteado fue evaluar la efectividad, en términos de eficiencia y eficacia, de dos[3] herramientas de pruebas E2E para microservicios.

[3] Originalmente se había propuesto evaluar solo una herramienta, pero dichosamente se lograron evaluar dos, por lo que ambos resultados se incluyen en este trabajo.

2. Selección de las herramientas a evaluar: se utilizaron las dos herramientas que obtuvieron mejor calificación en la evaluación mediante el estándar IEEE 14102-2010. De hecho, ambas herramientas tuvieron una calificación de 95 puntos.

3. Identificación de requisitos para realizar la evaluación de la herramienta:
    a. Identificar posibles aplicaciones de microservicios a probar (lo que comúnmente se conoce como SUT: *software under test*), preferiblemente con microservicios desarrollados en Java y Node.js. La aplicación debe ser funcional y tener documentación técnica disponible.
    b. Definir las características que requiere la máquina virtual donde se llevará a cabo la evaluación.
    c. Identificar las herramientas utilitarias necesarias para llevar a cabo las pruebas, tales como Docker y Maven.

4. Definición de los casos de prueba. Es relevante mencionar que fueron utilizados los elementos definidos por Eldh et al. (Eldh, Hansson, Punnekkat, Pettersson, & Sundmark, 2006) para evaluar la efectividad (efectividad = eficiencia + eficacia).
    a. Creación del caso de prueba para evaluar la eficiencia, medición de tiempos de ejecución y detección de fallas).
    b. Creación del caso de prueba para evaluar la eficacia, mediante la identificación de fallas, cobertura de flujo, e información relacionada a la severidad de las fallas.

Durante el desarrollo de la investigación se tomó la decisión de no utilizar el estándar ISO/IEC 25022:2016 (propuesto originalmente) para evaluar la efectividad de las herramientas. Dicho estándar provee un conjunto de características de calidad que pueden ser medidas para software o sistemas en uso, entre las cuales están la eficiencia y la eficacia. No obstante, tiene un enfoque hacia características desde la perspectiva del usuario, específicamente evaluando variables en tareas ejecutadas por los usuarios para lograr objetivos dados. Por ejemplo, se miden variables como el número de errores cometidos por el usuario durante una tarea, el lapso requerido por el usuario para realizar acciones productivas, la disminución del rendimiento humano después del uso continuo, entre otras (Standardization & Normalisation, 1987). Dado que nuestro estudio se enfocaba en evaluar la efectividad de las herramientas para apoyar las pruebas *end-to-end* de microservicios, consideramos que era mejor utilizar otro modelo de evaluación más orientado a

la funcionalidad técnica de la herramienta, ya que las características del estándar 25022:2016 no se adaptaban bien a nuestro objetivo de investigación.

4.2.2 Ejecución del caso de estudio

Las tareas desarrolladas para ejecutar el caso de estudio fueron las siguientes:

1. Selección y ejecución de la aplicación a probar:
    a. Se selecciona la aplicación a probar
    b. Se estudia la documentación técnica disponible
    c. Se ejecuta la aplicación y se verifica que funcione

2. Preparación del ambiente para evaluar las herramientas de pruebas E2E de microservicios:
    a. Creación de la máquina virtual
    b. Instalación de herramientas utilitarias
    c. Carga de la aplicación a probar y actualización de referencias
    d. Configuración de las herramientas de pruebas *end-to-end* sobre la aplicación

3. Ejecución de las pruebas:
    a. Pruebas para medir la eficiencia, se ejecutan dos casos de pruebas. Se ejecutan las pruebas con un escenario exitoso y uno fallido
    b. Se registran los tiempos generados por cada una de las herramientas
    c. Pruebas para medir la eficacia, se obtiene información de las métricas generadas por la herramienta en casos de fallas
    d. Se registran la respuesta de las herramientas cuando existe una o varias fallas

4.2.3 Análisis de los resultados del caso de estudio

Las tareas realizadas durante el análisis del caso de estudio fueron las siguientes:

1. Análisis de eficiencia: los resultados generados permitieron identificar en diferentes escenarios el comportamiento de las herramientas al realizar pruebas *end-to-end* con una arquitectura de microservicios.

2. Análisis de efectividad: la información generada por las herramientas permitió un análisis comparativo de sus características. Asimismo, permitieron determinar si estas características apoyaban a los desarrolladores con la identificación de fallas en un ambiente con microservicios.

Capítulo 5

Resultados

La primera fase de esta investigación permitió identificar herramientas utilizadas por la academia y la industria para pruebas *end-to-end* de microservicios. Esto se realizó mediante una revisión de literatura que permitió obtener 28 de herramientas, así como la aplicación de cuestionarios a expertos de la industria, que aportó 13 herramientas adicionales. Una de las herramientas fue encontrada por ambos métodos, por lo que el resultado final fue un conjunto de 40 herramientas diferentes (Ver Apéndice G). Estas herramientas fueron evaluadas siguiendo los procesos del estándar IEEE 141020-2010, para lo cual fue necesario definir características deseables en una herramienta de pruebas *end-to-end* para microservicios. Como resultado de dicha evaluación, se obtuvo que las dos herramientas mejor calificadas (con igual puntaje) fueron Jaeger y Zipkin. Adicionalmente, los cuestionarios aplicados a los expertos de la industria (Ver Apéndices B a E) generaron aportes relevantes para la investigación. Primero, todos los participantes consideraron necesaria la automatización de las pruebas de microservicios. Segundo, la mayoría de los participantes indicaron que el principal desafío de las pruebas de microservicios radica en la complejidad de las interacciones entre los múltiples servicios (evidenciando la importancia de las pruebas *end-to-end*). Tercero, los participantes indicaron que existen diferencias entre las pruebas de software tradicional y las pruebas de microservicios.

La segunda fase consistió en el desarrollo de un caso de estudio, donde se evaluó la efectividad de Jaeger y Zipkin como herramientas para pruebas *end-to-end* de microservicios, tomando en cuenta su eficiencia y eficacia. Nuestra investigación sugiere que Jaeger es más eficiente que Zipkin debido a que los tiempos promedios de ejecución y detección de fallas fueron menores en casi todos los escenarios de pruebas. Por otra parte, los resultados indican que ambas herramientas cuentan con características que permiten detectar fallas, representar la cobertura de flujo de datos, y ofrecer información a los desarrolladores que los guíe en la identificación de la severidad de las fallas. No obstante, la herramienta Jaeger muestra más información que la herramienta Zipkin, generando vistas alternativas para representar el trazado y creando gráficos de dependencias entre los servicios. El detalle del caso de estudio y sus resultados se encuentra debidamente reportado en el artículo adjunto en el Apéndice, el cual fue aceptado para publicación en la conferencia ICITS’21.

Capítulo 6

Conclusiones

En este trabajo hemos identificado un conjunto de 40 herramientas para pruebas *end-to-end* de microservicios, como resultado de una revisión de literatura y un cuestionario aplicado a expertos de la industria. Posteriormente, estas herramientas fueron evaluadas siguiendo el estándar IEEE 14102-2010, basados en criterios que evalúan la idoneidad de las herramientas para apoyar las pruebas E2E de microservicios. Además, realizamos una evaluación de la eficiencia y eficacia de Jaeger y Zipkin, ya que estas fueron las herramientas que obtuvieron mejor calificación en la evaluación del estándar. El tiempo de ejecución y el tiempo de detección de fallas se utilizaron como medidas de eficiencia. Evaluamos la eficacia con respecto a la capacidad para detectar fallas y la provisión de información que faciliten inferir la gravedad de las fallas, y la cobertura del flujo de datos.

A partir de nuestra evaluación inicial, descubrimos que actualmente se utilizan una gran variedad de herramientas para realizar pruebas E2E de microservicios, aunque muchas de ellas no fueron creadas para este fin. Incluso, ninguna de las herramientas identificadas permitió invocar a los microservicios y generar la métrica y/o trazado. Esta es una característica deseable, que recomendamos incluir en las próximas herramientas para las pruebas de microservicios E2E. Creemos que realizar E2E en un entorno de microservicios requiere herramientas especializadas con las características adecuadas.

Los resultados de nuestro caso de estudio sugieren que Jaeger es más eficiente que Zipkin debido a que obtuvo menores tiempos de ejecución y detección de fallas en la mayoría de los escenarios de prueba, haciéndola más eficiente. Con respecto a la eficacia, ambas herramientas pudieron detectar las fallas y proporcionar información que ayuda a los desarrolladores a inferir la gravedad de la falla, así como la cobertura de flujo de datos. Sin embargo, Jaeger ofrece más información que Zipkin, genera el trazado con vistas alternativas y crea gráficos de dependencia que suelen resultar útiles.

Creemos que estudios como este ayudarán a la industria del software a seleccionar herramientas apropiadas que permitan la automatización de pruebas *end-to-end en* sistemas distribuidos que utilicen microservicios. Este aporte es relevante ya que los expertos de la industria indican que el principal desafío

para realizar para pruebas de microservicios radica en la complejidad de las interacciones entre los servicios, señalando que es necesaria la automatización de las pruebas de microservicios.

Como trabajo futuro, sería interesante realizar un estudio similar sobre herramientas de prueba de microservicios para pruebas unitarias y pruebas de integración, dado que las pruebas de sistemas se abordaron en este estudio a través de pruebas E2E.

Bibliografía

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

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

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

BA, K., & Charters, S. (2007). *Guidelines for performing Systematic Literature Reviews in Software Engineering*. *2*.

Barna, C., Khazaei, H., Fokaefs, M., & Litoiu, M. (2017). Delivering Elastic Containerized Cloud Applications to Enable DevOps. *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 65–75. https://doi.org/10.1109/SEAMS.2017.12

Bin, W., Shulin, Y., Xuelei, R., & Guyang, W. (2017). Research on Digital Publishing Application System Based on Micro-Service Architecture. *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, 140–144. https://doi.org/10.1145/3171592.3171613

Board, I. S. (2010). IEEE 14102-2010. *{IEEE} Std 14102-2010*.

Bogner, J., Fritzsch, J., Wagner, S., & Zimmermann, A. (2019). Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality. *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, 187–195. https://doi.org/10.1109/ICSA-C.2019.00041

Carneiro, C., & Schmelmer, T. (2016). Microservices From Day One. In *Microservices From Day One*. https://doi.org/10.1007/978-1-4842-1937-9

Cerny, T., Donahoo, M. J., & Pechanec, J. (2017). Disambiguation and comparison of SOA, microservices and self-contained systems. *Proceedings of the 2017 Research in Adaptive and Convergent Systems, RACS 2017*. https://doi.org/10.1145/3129676.3129682

Chen, L. (2018). Microservices: Architecting for Continuous Delivery and DevOps. *Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018*, 39–46. https://doi.org/10.1109/ICSA.2018.00013

Cohn, M. (2009). The forgotten layer of the test automation pyramid. Retrieved December 17, 2019, from https://www.mountaingoatsoftware.com/%0Ablog/the-forgotten-layer-of-the-test-automation-pyramid.

Contan, A., Dehelean, C., & Miclea, L. (2018). Test automation pyramid from theory to practice. *2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, 1–5. https://doi.org/10.1109/AQTR.2018.8402699

de Camargo, A., Salvadori, I., Mello, R. dos S., & Siqueira, F. (2016). An Architecture to Automate Performance Tests on Microservices. *Proceedings of the 18th International Conference on*

Information Integration and Web-Based Applications and Services, 422–429. https://doi.org/10.1145/3011141.3011179

Du, Q., Ni, Z., Zhu, R., Xu, M., Guo, K., You, W., … Zheng, Q. (2018). A Service-Based Testing Framework for NFV Platform Performance Evaluation. *2018 12th International Conference on Reliability, Maintainability, and Safety (ICRMS)*, 254–261. https://doi.org/10.1109/ICRMS.2018.00055

Eismann, S., Bezemer, C.-P., Shang, W., Okanović, D., & van Hoorn, A. (2020). Microservices: A Performance Tester's Dream or Nightmare? *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, 138–149. https://doi.org/10.1145/3358960.3379124

Eldh, S., Hansson, H., Punnekkat, S., Pettersson, A., & Sundmark, D. (2006). A framework for comparing efficiency, effectiveness and applicability of software testing techniques. *Proceedings - Testing: Academic and Industrial Conference - Practice and Research Techniques, TAIC PART 2006*. https://doi.org/10.1109/TAIC-PART.2006.1

Fowler, M., Lewis, J. (2018). *Microservices*. 1–15.

García, B., Gallego, M., Gortazar, F., & López, L. (2017). ElasTest, an Open-source Platform to Ease End-to-End Testing. *Challenges and Opportunities in ICT Research Projects*, 3–21. https://doi.org/10.5220/0007904700030021

Gazzola, L., Goldstein, M., Mariani, L., Segall, I., & Ussi, L. (2020). Automatic Ex-Vivo Regression Testing of Microservices. *Proceedings of the IEEE/ACM 1st International Conference on Automation of Software Test*, 11–20. https://doi.org/10.1145/3387903.3389309

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

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

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

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

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

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

Computing Systems, *2016-Augus*, 57–66. https://doi.org/10.1109/ICDCS.2016.11

Ibrahim, A., Bozhinoski, S., & Pretschner, A. (2019). Attack Graph Generation for Microservice Architecture. *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 1235–1242. https://doi.org/10.1145/3297280.3297401

Jawarneh, I. M. A., Bellavista, P., Bosi, F., Foschini, L., Martuscelli, G., Montanari, R., & Palopoli, A. (2019). Container Orchestration Engines: A Thorough Functional and Performance Comparison. *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 1–6. https://doi.org/10.1109/ICC.2019.8762053

Jindal, A., Podolskiy, V., & Gerndt, M. (2019). Performance Modeling for Cloud Microservice Applications. *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 25–32. https://doi.org/10.1145/3297663.3310309

Kargar, M. J., & Hanifizade, A. (2018). Automation of regression test in microservice architecture. *2018 4th International Conference on Web Research, ICWR 2018*, 133–137. https://doi.org/10.1109/ICWR.2018.8387249

Larrucea, X., Santamaria, I., Colomo-Palacios, R., & Ebert, C. (2018). Microservices. *IEEE Software*, *35*(3), 96–100. https://doi.org/10.1109/MS.2018.2141030

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

Lei, Q., Liao, W., Jiang, Y., Yang, M., & Li, H. (2019). Performance and Scalability Testing Strategy Based on Kubemark. *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 511–516. https://doi.org/10.1109/ICCCBDA.2019.8725658

Lewis, James; Fowler, M. (2014). Microservices - A definition of this new architectural term.

Li, S., Xu, Q., Hou, P., Chen, X., Wang, Y., Zhang, H., & Rong, G. (2020). Exploring the Challenges of Developing and Operating Consortium Blockchains: A Case Study. *Proceedings of the Evaluation and Assessment in Software Engineering*, 398–404. https://doi.org/10.1145/3383219.3383276

Liu, D., Zhu, H., Xu, C., Bayley, I., Lightfoot, D., Green, M., & Marshall, P. (2016). CIDE: An integrated development environment for microservices. *Proceedings - 2016 IEEE International Conference on Services Computing, SCC 2016*, 808–812. https://doi.org/10.1109/SCC.2016.112

Ma, S.-P., Fan, C.-Y., Chuang, Y., Lee, W.-T., Lee, S.-J., & Hsueh, N.-L. (2018). Using Service Dependency Graph to Analyze and Test Microservices. *Proceedings - International Computer Software and Applications Conference*, *2*, 81–86. https://doi.org/10.1109/COMPSAC.2018.10207

Ma, Shang-Pin, Fan, C.-Y., Chuang, Y., Liu, I.-H., & Lan, C.-W. (2019). Graph-based and scenario-driven microservice analysis, retrieval, and testing. *Future Generation Computer Systems*, *100*, 724–735. https://doi.org/https://doi.org/10.1016/j.future.2019.05.048

Meinke, K., & Nycander, P. (2015). Learning-Based Testing of Distributed Microservice Architectures: Correctness and Fault Injection. In D. Bianculli, R. Calinescu, & B. Rumpe (Eds.), *Software Engineering and Formal Methods* (pp. 3–10). Berlin, Heidelberg: Springer Berlin Heidelberg.

Nagarajan, A., & Vaddadi, A. (2016). Automated Fault-Tolerance Testing. *Proceedings - 2016 IEEE*

International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2016, 275–276. https://doi.org/10.1109/ICSTW.2016.34

Neves, F., Vilaça, R., & Pereira, J. (2020). Black-Box Inter-Application Traffic Monitoring for Adaptive Container Placement. *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 259–266. https://doi.org/10.1145/3341105.3374007

Powell, A., Vickers, A., Williams, E., & Cooke, B. (1996). A practical strategy for the evaluation of software tools. In *Method Engineering*. https://doi.org/10.1007/978-0-387-35080-6_11

Quenum, J. G., & Aknine, S. (2018). Towards executable specifications for microservices. *Proceedings - 2018 IEEE International Conference on Services Computing, SCC 2018 - Part of the 2018 IEEE World Congress on Services*. https://doi.org/10.1109/SCC.2018.00013

Rastogi, V., Davidson, D., De Carli, L., Jha, S., & McDaniel, P. (2017). Cimplifier: Automatically Debloating Containers. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 476–486. https://doi.org/10.1145/3106237.3106271

Raulamo-Jurvanen, P., Hosio, S., & Mäntylä, M. V. (2019). Practitioner evaluations on software testing tools. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3319008.3319018

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*. https://doi.org/10.1007/s10664-008-9102-8

Schreiber, M. (2020). Prevant (Preview servant): Composing microservices into reviewable and testable applications. *OpenAccess Series in Informatics*, *78*. https://doi.org/10.4230/OASIcs.Microservices.2017-2019.5

Shahin, M., Babar, M. A., Zahedi, M., & Zhu, L. (2017). Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges. *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 111–120. https://doi.org/10.1109/ESEM.2017.18

Shahrad, M., Balkind, J., & Wentzlaff, D. (2019). Architectural Implications of Function-as-a-Service Computing. *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 1063–1075. https://doi.org/10.1145/3352460.3358296

Singh, C., Gaba, N. S., Kaur, M., & Kaur, B. (2019). Comparison of Different CI/CD Tools Integrated with Cloud Platform. *2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, 7–12. https://doi.org/10.1109/CONFLUENCE.2019.8776985

Soenen, T., Van Rossem, S., Tavernier, W., Vicens, F., Valocchi, D., Trakadas, P., … Lopez, D. (2018). Insights from SONATA: Implementing and integrating a microservice-based NFV service platform with a DevOps methodology. *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 1–6. https://doi.org/10.1109/NOMS.2018.8406139

Sotomayor, J. P., Allala, S. C., Alt, P., Phillips, J., King, T. M., & Clarke, P. J. (2019). Comparison of Runtime Testing Tools for Microservices. *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, *2*, 356–361. https://doi.org/10.1109/COMPSAC.2019.10232

Standardization, F. O. R., & Normalisation, D. E. (1987). *ISO/IEC 25022:2016. 1987*.

Stefano Munari, Sebastiano Valle, and T. V. (2018). *Microservice-Based Agile Architectures: An*

Opportunity for Specialized Niche Technologies (Vol. 10873). https://doi.org/10.1007/978-3-319-92432-8

Ueda, T., Nakaike, T., & Ohara, M. (2016). Workload characterization for microservices. *Proceedings of the 2016 IEEE International Symposium on Workload Characterization, IISWC 2016*, 85–94. https://doi.org/10.1109/IISWC.2016.7581269

Vera-Rivera, F. H. (2018). A development process of enterprise applications with microservices. *Journal of Physics: Conference Series*, *1126*(1). https://doi.org/10.1088/1742-6596/1126/1/012017

Vocke, H. (2018). The Practical Test Pyramid. Retrieved January 21, 2020, from https://martinfowler.com/articles/practical-test-pyramid.html

Yuan, E. (2019). Architecture Interoperability and Repeatability with Microservices: An Industry Perspective. *Proceedings of the 2nd International Workshop on Establishing a Community-Wide Infrastructure for Architecture-Based Software Engineering*, 26–33. https://doi.org/10.1109/ECASE.2019.00013

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

Apéndice A

Artículo de investigación

Como resultado del presente trabajo de investigación, se elaboró un artículo que fue aceptado para publicación en la conferencia *International Conference on Information Technology & Systems* ICITS'21. Esta conferencia se realizará en Península de Santa Elena, Ecuador, del 10 al 12 de febrero del 2021. El artículo será publicado en un libro de la serie *Advances in Intelligent Systems and Computing* de *Springer*, e indexado por ISI, EI-Compendex, SCOPUS y DBLP.
Referencia futura (algunos datos como el volumen y los editores no se conocen aún, pero los demás datos los inferimos de publicaciones anteriores realizadas en esta conferencia):

Martínez, C., Martínez, A., Quesada-López, C., Jenkins, M. (2021) "Comparison of end-to-end testing tools for microservices: A case study". In: (eds) Information Technology and Systems. ICITS 2021. Advances in Intelligent Systems and Computing, vol . Springer, Cham.

El artículo completo se incluye a continuación, así como la carta de aceptación del artículo e invitación a la conferencia.

Artículo científico aceptado

Comparison of end-to-end testing tools for microservices: A case study

Cristian Martínez Hernández, Alexandra Martínez, Christian Quesada-López, and Marcelo Jenkins

Universidad de Costa Rica, San José, Costa Rica
{cristian.martinezhernandez, alexandra.martinez, cristian.quesadalopez, marcelo.jenkins}@ucr.ac.cr

Abstract. Microservices has emerged as a architectural style that provides several benefits but also poses some challenges. One such challenge is testability, since an application may have hundreds or thousands of services operating together, and each of them needs to be tested as they evolve. To overcome this challenge, test automation is key, and together with it, the use of effective and efficient testing tools. Hence, we aim to contribute to this area by evaluating two tools that support end-to-end (E2E) testing of microservices. E2E tests allow to verify if the system works well as a whole (particularly relevant for systems made up of microservices). In this work, we first surveyed E2E testing tools reported in academic literature and by industry practitioners. Then, we applied the IEEE 14102-2010 standard to evaluate those tools. The two top-rated tools, Jaeger and Zipkin, were selected for further evaluation of their effectiveness and efficiency. Results from our case study reveal that Jaeger is more efficient and effective than Zipkin in terms of execution and failure detection times, as well as information provided to detect faults, severity and coverage.

Keywords: End-to-end testing, microservices, tools, automation, case study

1 Introduction

Microservices is a popular architectural style for designing and implementing distributed systems, which has had a huge impact on the software industry [11] [18]. When compared to monolithic architectures like SOA, microservices exhibits many benefits such as enabling a clearer application structure, simplifying the development, deployment and upgrade of large-scale applications, as well as increasing deployability and modifiability [5] [14]. However, adopting microservices still has challenges associated to an increased number of services, evolving contracts among services, technology diversity, and testing [5].

In particular, testing microservices is difficult due to the potentially large number of services involved, their heterogeneity (developed with different programming languages and technologies), as well as their asynchronous and independent nature. The path to meet this challenge is automation [16], and for

this we need tools that support microservices testing. Despite being a little investigated area [8], a recent study by Sotomayor et al. [22]compares several microservices testing tools, and concludes that the increasing use of microservices in software applications has evidenced the need for more suitable testing tools and techniques.

In this paper, we address end-to-end (E2E) testing of microservices because it helps diagnosing performance problems in distributed systems [13] as well as correctness of the system as a whole [14]. The increase in microservice complexity makes reliance on automated tests and tools critical for E2E testing [2]. Nevertheless, test automation is not trivial, and often generates monetary and time losses [9]. To help companies that want to automate their microservices testing, this study provides an evaluation of several E2E testing tools drawn from the literature and industry professionals. Using the IEEE 14102-2010 standard, two of these tools (Jaeger[1] and Zipkin[2]) are selected for further evaluation of their efficiency and effectiveness in the context of microservices. We believe that studies like this will aid software organizations to select the appropriate tool for their context, and thus adopt end-to-end testing automation.

The rest of the paper is structured as follows: Section 2 offers a background on microservices and E2E testing, Section 3 summarizes relevant previous works. Section 4 offers the first evaluation of tools based on IEEE 14102-2010 standard, Section 5 presents the second evaluation (case study) of Jaeger and Zipkin, and Section 6 outlines the conclusions.

2 Background

Microservices has emerged as an architecture alternative for the design and implementation of distributed systems, resulting in systems with low-coupled components that exhibit properties like flexibility, scalability, adaptability, and fault tolerance [11]. Martin Fowler, pioneer in the microservices architecture, defined it as an architectural style to develop a single application as a suite of small services, each running on its own process and communicating by lightweight mechanisms, often an HTTP resource API [7]. During the last decade, companies like Netflix, Facebook, Twitter, Amazon, Spotify, eBay, LinkedIn, Uber and SoundCloud have embraced microservices [3, 12]. Yet, there are still challenges for its adoption, including testability [4].

End-to-end tracing has emerged in the past decade as a valuable technique to diagnose correctness and performance problems in distributed systems, increase coverage, model workloads, resource usage and timings [13, 22]. Obtaining these metrics enables developers and operators to understand how their system works and why it fails. Meanwhile, the primary objective of the E2E tests is to ensure a consistent and reliable behavior, and to catch hard-to-test bugs before users do, when unit and integration tests are insufficient [14]. However, executing E2E tests for microservices by traditional (i.e., manual) methods is quite difficult [16],

[1] https://www.jaegertracing.io/
[2] https://zipkin.io/

hence automation seems to be the way to go. There is a need for more studies that expand the empirical evidence in this area [8].

3 Related Work

Here we present relevant related work, dealing especially with tools for microservice testing. In particular, Sotomayor et al. [22] compared the execution time of four testing tools for microservices (two E2E test tools and two test harness tools) on an open-source testbed application composed of multi-language microservices. They also characterized a set of 16 test tools for microservices, along several dimensions including test objective, test level, platform, interface, and supported languages.

Schreiber [20] introduced PREvant, a tool that provides a simple RESTful API for deploying and composing containerized microservices. This tool can automate E2E tests to verify the performance of microservices.

Harsh et al. [10] pointed out that complex and large software systems are proliferating due to the commodity of the cloud and the need for elastic applications, which push developers towards resilient software architectures like microservices. Consequently, practices and tools need to evolve to support testing in these new scenarios. They proposed a new testing service model called Testing as a Service, and a new tool named ElasTest, which enables E2E testing.

Meinke et al. [16] studied how learning-based testing (an emerging paradigm for fully automated black-box testing) can be used to evaluate the functional correctness and fault-robustness of a distributed system. In their experiment they used LBTest, a learning-based testing tool. They concluded that the performance of LBTest compared favorably to manual techniques for E2E testing.

Previous works mainly proposed new tools for E2E testing of microservices, or evaluated existing ones. Our study contributes to the field by first compiling a list of 40 test tools for microservices specifically geared towards end-to-end testing, which were extracted from the literature and industry experts. It also provides an evaluation of those tools, based on necessary features of E2E microservices test tools. Finally, it reports a case study where two E2E microservices test tools are evaluated for efficiency and effectiveness.

4 Tools that support E2E testing of microservices

Here we explain how the E2E testing tools were identified and evaluated.

4.1 Tools identification

In order to identify existing E2E testing tools, we first reviewed the academic literature, where 28 tools were found. Then we surveyed international experts in search of tools used in the industry. Specifically, during May and June of

2020 we sent a survey to 25 Docker's captains[3] and got response from 7 of them. This gave us 13 additional tools reported by industry practitioners. One of them, Selenium, had already been found in the literature survey. Hence, a total of 40 $(28 + 13 - 1)$ tools were identified. The complete list can be found at https://tinyurl.com/y4sq58mj.

Some interesting findings from our survey follow. First, all participants deemed necessary to automate microservices tests. Second, most participants indicated that a major challenge in microservices testing lies in the interaction complexity among multiple services. Third, all but one of the participants typically perform E2E testing of microservices. Fourth, participants argued that there are differences between traditional software testing and microservices testing. On one hand, testing individual microservices is easier due to their small size, but new problems arise due to the complex interactions between microservices. On the other hand, it is easier to launch a monolithic system by calling one of its functions than to launch tens of interdependent microservices in order to test one of the functional flows they provide.

4.2 Tools evaluation

In order to evaluate the previously identified tools, we followed the IEEE 14102-2010 standard and associated recommendations [15]. This is an international standard that provides guidelines for the evaluation and selection of CASE tools. Our evaluation criteria are based on the *tool specific* aspects typified by Powell et al. [17], which represent fitness for purpose of the tool. We selected a subset of these aspects that were considered relevant and even necessary in the context of E2E testing tools for microservices. Particularly, we used the metrics, documentation, support, integratibility, cost, and compatibility criteria. Table 1 shows the 5 tools with highest scores. The complete evaluation is available at https://tinyurl.com/y4sq58mj.

The *metrics* criterion evaluates the ability of the tool to generate useful information from E2E tests. A value of 30 is assigned if the tool generates tracing and timing information. A value of 15 is given to tools that generate only one useful metric; otherwise a 0 is assigned.

The *documentation* criterion gauges the amount of tool documentation available (videos, tutorials, forums, or formal documentation). A value of 20 is assigned when documentation is extensive, including that for E2E tests. A 10 is given when documentation is extensive but does not include E2E tests documentation. A 5 is given when documentation is scarce, and 0 when is non-existent.

The *support* criterion refers to the type of *technical support* available: provider (formal), community (informal), or none. We assigned a value of 15 to provider support, a value of 10 to community support, and 0 for no support.

The *integratibility* criterion assesses the tool's level of compatibility with existing languages or browsers (for Software as a Service). We assigned a value

[3] https://www.docker.com/community/captains

Table 1. Top 5 testing tools to support testing end-to-end of microservices.

Tool name	Metrics	Documentation	Support	Integratibility	Cost	Platform compat.	Total score
Jaeger	Tracing, Timing, Fails System, Resources use	High	Community support	High (Go, Java, Python, Node, C++, C#)	Free	Both(Local and Cloud)	**95\100**
	30\30	**20\20**	**10\15**	**10\10**	**20\20**	**5\5**	
Zipkin	Tracing, Timing, Fails System, Resources use	High	Community support	High (C#, Go, Java, Javascript, Ruby, Scala, PHP)	Free	Both(Local and Cloud)	**95\100**
	30\30	**20\20**	**10\15**	**10\10**	**20\20**	**5\5**	
Test Project	Fails System	High	Provider support	High(Browser Support)	Free	Both(Local and Cloud)	**85\100**
	15\30	**20\20**	**15\15**	**10\10**	**20\20**	**5\5**	
Spring Cloud	Timing, Resources Use	High	Provider support	Low (Java)	Free	Both(Local and Cloud)	**77\100**
	15\30	**20\20**	**15\15**	**2\10**	**20\20**	**5\5**	
Browser Stack	Fails System	High	Provider support	High(Browser Support)	Trial	Both(Local and Cloud)	**75\100**
	15\30	**20\20**	**15\15**	**10\10**	**10\20**	**5\5**	

of 10 if the tool supports 4 or more languages, a value of 5 if it supports 2 or 3 languages, and a value of 2 if only one language is supported.

The *cost* criterion considers only the licensing aspect of the tool. A value of 20 was assigned to free tools, a value of 10 was assigned if a trial version was available, and 0 if payment was required.

The *platform compatibility* criterion refers to platform dependence. If the tool is compatible with both local and cloud, a value of 5 is assigned. If it is only compatible with one platform, a 3 is assigned.

5 Case study

We provide here details about the case study conducted, following the guidelines proposed by Runeson et al. [19]. The **objective** of our case study was to evaluate the efficiency and effectiveness of Jaeger and Zipkin as tools for E2E testing of microservices. These two tools were chosen for having the best scores from our previous evaluation.

5.1 Experimental objects

Our experimental object was an open-source application called Socks Shop[4], built to aid research and testing of microservices technologies. This application encompasses the user-facing part of an e-commerce website that sells socks [1], and uses microservices implemented in Java, .Net Core, Go, and NodeJs as

[4]https://github.com/microservices-demo/microservices-demo

front-end. Docker was configured to orchestrate these applications since it is a reference technology in containers virtualization [21].

5.2 Data collection procedure

We first prepared a virtual machine with 8 MB RAM, 2 cores, Ubuntu OS v18.04, Docker v19.03.5, Docker-Compose v1.23.2, Apache Maven v3.6.0, Jaeger-client v3.18.0 and Zipkin v0.22.0.

The architecture and ecosystem of the Socks Shop microservice application used in the case study are depicted in Figure 1. Jaeger and Zipkin were configured on the following microservices: MS1-Order and MS5-Cart. The repositories were cloned and modified to include Zipkin and Jaeger, updating all their dependencies. Our MS1-Order and MS5-Cart containers were created, and the file docker-compose.yml was modified to use our containers.

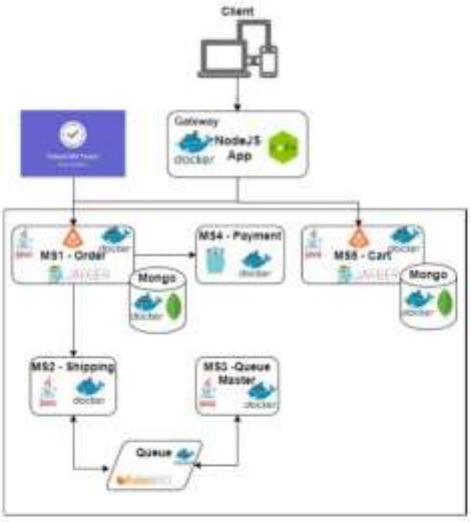


Fig. 1. Architecture of the evaluated microservices application.

Then, two E2E tests were designed and run (invoked by API Tester): the first one invokes the MS1-Order's microservice (POST), which in turns calls MS2-Shipping, MS4-Payment and MS3-Queue Master; the second one invokes only the MS5-Cart's microservice (GET). Both tests collected the following metrics: execution time, failure detection time, fault detection, error type identification,

fault severity, and dataflow coverage. We forced the occurrence of failures by stopping the Mongo container used by MS5-Cart, as well as the MS2-Shipping container used by MS1-Order.

To assess and compare the efficiency and effectiveness of the tools, we partly relied on the recommendations by Eldh et al. [6]. Efficiency is measured by the execution and failure detection times, while effectiveness is measured by the ability to detect faults and the provision of information for inferring fault severity and dataflow coverage.

5.3 Results

Next we present the results of our case study. Table 2 shows the results for efficiency: it contains the average execution time and average failure detection time (in ms) for Jaeger and Zipkin, when executing the E2E tests MS1-Order and MS5-Cart. Each test was run 5 times using the API Tester. These results suggest that Jaeger is more efficient than Zipkin since both execution and failure detection times are better in all but one case: the second test case (MS5-Cart). In this case, the failure detection time for Jaeger is slightly higher than that of Zipkin, but this difference is not significant as it is triggered by timeout. From Table 2, we see that for the first test case (MS1-Order), Jaeger's failure detection time is about a third of Zipkin's, and its execution time is close to half of Zipkin's. For the second test case, Jaeger's execution time is 12 times less than Zipkin's.

Table 3 shows the results for effectiveness in terms of three features that help developers and operators to identify and characterize bugs in microservices: ability to detect faults, and provision of information that allows to infer both fault severity and data coverage. Our results found that both Jaeger and Zipkin exhibit these features, with some differences among them. In the case of *fault detection capability*, the tools have a tag with the http status code to present the error type or response from the microservice. Jaeger, however, has additional tags such as component, error indicator, http method, http url, hostname, IP address, among others. With respect to the information offered by the tools to infer the *severity of the fault*, both provide the error stack, from which severity can be inferred, but Jaeger offers more information: message, event, handler method, and handler class name.

Finally, the information provided by the tools to infer the *dataflow coverage* are traces. These are data/execution paths through the system, which form a graph of spans (logical units of work), allowing for test tracing. Jaeger gives further alternative views like the trace graph or trace JSON, and can generate a graph of service dependencies. In summary, although both tools satisfy the three effectiveness properties, we consider Jaeger to be more effective than Zipkin because it provides more information in all three features.

Table 2. Efficiency measurements for Jaeger and Zipkin.

	Test 1 (MS1-Order)		Test 2 (MS5-Cart)	
	Jaeger	Zipkin	Jaeger	Zipkin
Execution time	47.81 ms	80.26 ms	4.30 ms	53.149 ms
Failure detection time	46.25 ms	167.42 ms	10.64 s	10.03 s

Table 3. Effectiveness results for Jaeger and Zipkin.

Feature	Jaeger	Zipkin
Fault detection	Shows entire trace and identifies point of failure. Provides depth, duration, services, trace ID, total spans, trace start, http status, component, error indicator, http method, http url, hostname, IP address, among other data.	Shows entire trace and identifies point of failure. Provides depth, duration, services, trace ID, total spans, and http status.
Fault severity	Provides error message, event, handler method, handler class name, and stack.	Provides the error stack.
Dataflow coverage	Offers traces, which can represent dataflow coverage. Allows alternative views like trace graph, trace JSON, and service dependencies graph.	Offers traces, which can represent dataflow coverage.

6 Conclusions

In this paper, we have identified a set of 40 tools for end-to-end testing of microservices, after surveying the literature and industry experts. These tools were then evaluated following the IEEE 14102-2010 standard, based on criteria that assess their suitability as E2E microservices test tools. Furthermore, we performed an evaluation of the efficiency and effectiveness of Jaeger and Zipkin, the two top-rated tools. Execution time and failure detection time were used as measures of efficiency. For effectiveness, we used the ability to detect faults, and the provision of information to infer fault severity and dataflow coverage.

We found that a variety of tools are used to perform end-to-end testing of microservices, despite not being meant for it. Even more, none of the identified tools allowed both test execution and report generation (a highly desirable feature). We believe that performing end-to-end testing in a microservice environment thus requires specialized tools with appropriate features.

Results from our case study suggest that Jaeger is more efficient than Zipkin because of its shorter execution time and failure detection time in most test scenarios. With respect to effectiveness, both are able to detect faults and provide information that can help developers infer fault severity as well as dataflow coverage. However, Jaeger offers more information than Zipkin, generates tracing with alternate views, and creates useful dependency graphs.

As future work, we suggest to complement this work by studing available tools for unit and integration testing of microservices (E2E are system tests).

Acknowledgments

This work was partially supported by University of Costa Rica's projects No. 834-B8-A27 and 834-C0-726, financed by the Research Center on ICT (CITIC) and the Department of Computer Science (ECCI). We thank the Empirical Software Engineering Group (ESEG) for its valuable feedback and help.

References

[1] C. M. Aderaldo et al. "Benchmark Requirements for Microservices Architecture Research". In: *2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*. 2017, pp. 8–13.

[2] Peter Alvaro et al. "Automating Failure Testing Research at Internet Scale". In: *Proceedings of the Seventh ACM Symposium on Cloud Computing*. SoCC '16. Santa Clara, CA, USA: Association for Computing Machinery, 2016, pp. 17–28. ISBN: 9781450345255.

[3] J Bogner et al. "Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality". In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. Mar. 2019, pp. 187–195. DOI: 10.1109/ICSA-C.2019.00041.

[4] André de Camargo et al. "An Architecture to Automate Performance Tests on Microservices". In: *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*. New York, NY, USA: ACM, 2016, pp. 422–429. ISBN: 978-1-4503-4807-2.

[5] L Chen. "Microservices: Architecting for Continuous Delivery and DevOps". In: *Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018*. 2018, pp. 39–46.

[6] S. Eldh et al. "A Framework for Comparing Efficiency, Effectiveness and Applicability of Software Testing Techniques". In: *Testing: Academic Industrial Conference - Practice And Research Techniques*. 2006, pp. 159–170.

[7] M Fowler and J Lewis. "Microservices". In: (2018), pp. 1–15.

[8] I Ghani et al. "Microservice testing approaches: A systematic literature review". In: *International Journal of Integrated Engineering* 11.8 (2019), pp. 65–80. DOI: 10.30880/ijie.2019.11.08.008.

[9] P Gkikopoulos. "Data Distribution and Exploitation in a Global Microservice Artefact Observatory". In: *2019 IEEE World Congress on Services*. Vol. 2642-939X. 2019, pp. 319–322. DOI: 10.1109/SERVICES.2019.00089.

[10] Piyush Harsh et al. "Cloud Enablers For Testing Large-Scale Distributed Applications", In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*. New York, NY, USA: ACM, 2019, pp. 35–42. DOI: 10.1145/3368235.3368838.

[11] Robert Heinrich et al. "Performance Engineering for Microservices: Research Challenges and Directions". In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. New York, NY, USA: ACM, 2017, pp. 223–226. ISBN: 978-1-4503-4899-7.

[12] V Heorhiadi et al. "Gremlin: Systematic Resilience Testing of Microservices". In: *Proceedings - International Conference on Distributed Computing Systems*. Vol. 2016-Augus. 2016, pp. 57–66.

[13] Pedro Las-Casas et al. "Weighted Sampling of Execution Traces: Capturing More Needles and Less Hay". In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 326–332. ISBN: 9781450360111. DOI: 10.1145/3267809.3267841.

[14] Q Lei et al. "Performance and Scalability Testing Strategy Based on Kubemark". In: *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. Apr. 2019, pp. 511–516.

[15] B. Lundell and B. Lings. "Comments on ISO 14102: The standard for CASE-tool evaluation". In: *Computer Standards and Interfaces* 24.5 (2002). cited By 7, pp. 381–388. DOI: 10.1016/S0920-5489(02)00064-8.

[16] Karl Meinke and Peter Nycander. "Learning-Based Testing of Distributed Microservice Architectures: Correctness and Fault Injection". In: *Software Engineering and Formal Methods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 3–10. ISBN: 978-3-662-49224-6.

[17] Antony Powell et al. "A Practical Strategy for the Evaluation of Software Tools". In: July 1996. ISBN: 1475758243. DOI: 10.1007/978-0-387-35080-6_11.

[18] S. Quenum, J.G. and Aknine. "Towards Executable Specifications for Microservices". In: *Proceedings - 2018 IEEE International Conference on Services Computing, SCC 2018 - Part of the 2018 IEEE World Congress on Services* (2018), pp. 41–48.

[19] Per Runeson and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical software engineering* 14.2 (2009), p. 131.

[20] M Schreiber. "Prevant (Preview servant): Composing microservices into reviewable and testable applications". In: *OpenAccess Series in Informatics*. Vol. 78. 2020. DOI: 10.4230/OASIcs.Microservices.2017-2019.5.

[21] C Singh et al. "Comparison of Different CI/CD Tools Integrated with Cloud Platform". In: *2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence)*. Jan. 2019, pp. 7–12.

[22] J P Sotomayor et al. "Comparison of Runtime Testing Tools for Microservices". In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 2. July 2019, pp. 356–361.

Carta aceptación del artículo e invitación a la conferencia

ACCEPTANCE & INVITATION LETTER

Dear **CRISTIAN FERNANDO MARTINEZ HERNANDEZ**
Universidad de Costa Rica
Costa Rica

On behalf of the ICITS'21 - The 2021 International Conference on Information Technology & Systems, I am pleased to inform you that your submission "***Comparison of end-to-end testing tools for microservices: A case study***" has been accepted as a Full Paper for publication and online oral presentation in this conference.

So, you are cordially invited to participate and present the paper in the ICITS'21 (http://www.icits.me/) to be held in Peninsula de Santa Elena, Ecuador, between the 10th and the 12th of February of 2021, an international scientific event sponsored and organized by Universidad Estatal Península de Santa Elena, IEEE SMC and ITMA.

We sincerely hope that you will join us in making ICITS'21 a success. We look forward to seeing you next February.

Sincerely,

Álvaro Manuel Reis da Rocha
ICITS'21, Chair

Apéndice B

Cuestionario para profesionales del evento *Dockerizing Microservices and Deploy them in Kubernetes* en Costa Rica

El cuestionario correspondiente a la primera encuesta realizada con profesionales que asistieron al evento de *Docker* en Costa Rica se puede consultar en el enlace: https://forms.gle/jFPxj5mzP5QqpMiA9

Microservices

Este cuestionario se aplicará a profesionales de la industria de software que asistieron al evento "Dockerizing Microservices and deploy them in Kubernetes". El cuestionario es anónimo y toma alrededor de 5 minutos contestarlo. Los resultados del mismo serán utilizados como parte de un trabajo final de investigación aplicada para optar por el grado de Maestría Profesional en Computación e Informática de la Universidad de Costa Rica. Agradecemos su participación.

* Required

1. 1. ¿Cuál es su puesto actual? *

   Mark only one oval.

   - ◯ Tester
   - ◯ Analista y Programador
   - ◯ Lider Técnico
   - ◯ Infraestructura
   - ◯ Other: _____

2. 2. ¿Cuántos empleados tienen la empresa donde trabaja? *

   _____

3. 3. ¿Cuántos años de experiencia posee en pruebas de software? *

   Mark only one oval.

   - ◯ No tengo experiencia en pruebas
   - ◯ Menos de 2 años
   - ◯ De 2 a 5 años
   - ◯ De 5 a 8 años
   - ◯ Más de 8 años
   - ◯ Other: _____

4. 4. ¿Ha realizado pruebas para microservicios ? *

Mark only one oval.

- ◯ Sí
- ◯ No
- ◯ Other: _____

5. 5. ¿Cuántos años de experiencia tiene trabajando con microservicios ? *

Mark only one oval.

- ◯ No tengo experiencia *Skip to question 6*
- ◯ Menos de 1 año *Skip to question 8*
- ◯ De 1 a 3 años *Skip to question 8*
- ◯ De 3 a 5 años *Skip to question 8*
- ◯ Más de 5 años *Skip to question 8*
- ◯ Other: _____

Skip to question 8

Microservicios

6. 6. ¿Durante los próximos 12 meses su empresa tiene pensado utilizar microservicios ? *

Mark only one oval.

- ◯ Sí
- ◯ No
- ◯ Other: _____

7. 7. ¿Cuáles herramientas ha utilizado para pruebas End-to-End?

Tecnología de microservicios

8. 6. ¿Cuáles lenguajes de programación ha utilizado para trabajar con microservicios? *

9. 7. ¿Cuál gestor de contenedores utiliza? *

Mark only one oval.

- ◯ Docker
- ◯ No uso gestor de contenedores
- ◯ Other: _____

10. 8. ¿Cuáles son los principales retos al realizar pruebas de microservicios?

11. 9. ¿Considera que es necesaria la automatización para probar los microservicios?

12. 10. ¿Qué tipo de pruebas realiza normalmente sobre microservicios? *

Check all that apply.

- [ ] Unitarias
- [ ] Integración
- [ ] End-to-End
- [ ] Other:

13. 11. ¿Cuáles herramientas utiliza para pruebas End-to-End? *

Apéndice C

Resultados de la encuesta pasada en el evento *Docker* Costa Rica

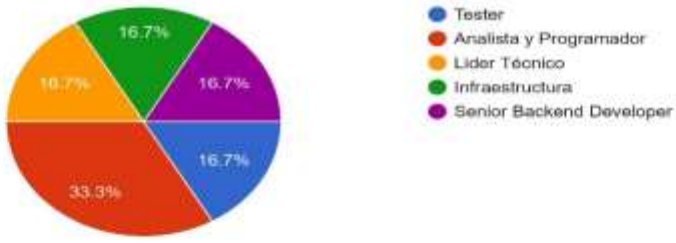
Microservices

6 responses

Publish analytics

1. ¿Cuál es su puesto actual?

6 responses



2. ¿Cuántos empleados tienen la empresa donde trabaja?

6 responses

300

1500

200

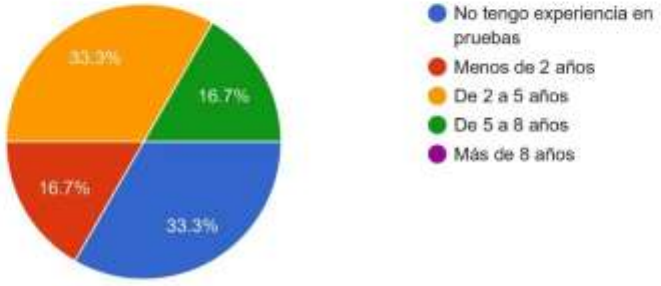
aprox 150

250

80

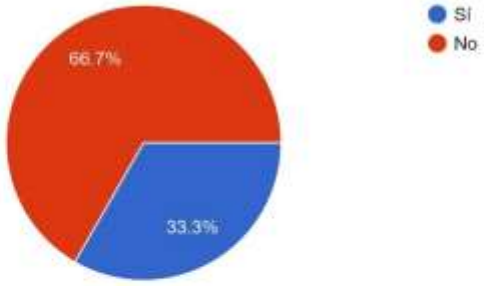
3. ¿Cuántos años de experiencia posee en pruebas de software?

6 responses



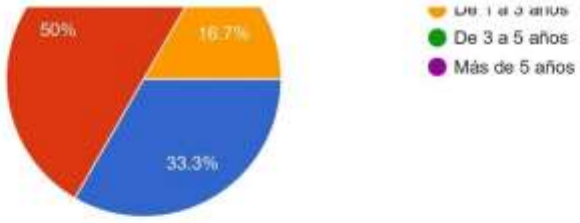
4. ¿Ha realizado pruebas para microservicios ?

6 responses



5. ¿Cuántos años de experiencia tiene trabajando con microservicios ?

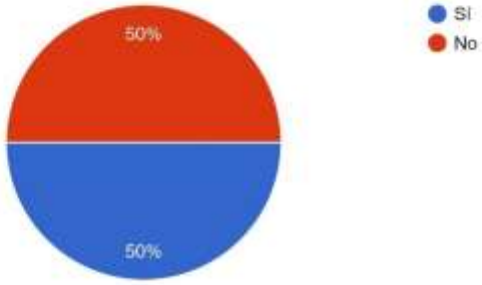
6 responses



Microservicios

6. ¿Durante los próximos 12 meses su empresa tiene pensado utilizar microservicios ?

2 responses



7. ¿Cuáles herramientas ha utilizado para pruebas End-to-End?

0 responses

No responses yet for this question.

Tecnología de microservicios

6. ¿Cuáles lenguajes de programación ha utilizado para trabajar con microservicios?

4 responses

JavaScript

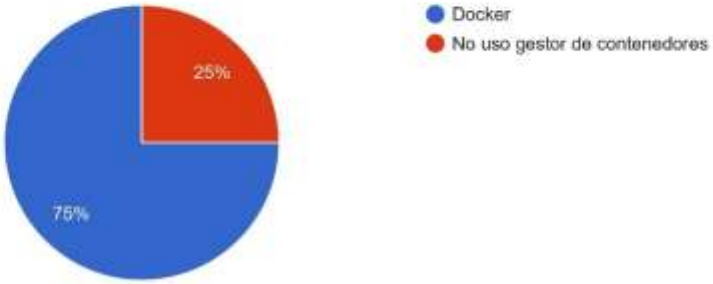
Microsoft Dot Net

Java

NodeJS and Python

7. ¿Cuál gestor de contenedores utiliza?

4 responses



8. ¿Cuáles son los principales retos al realizar pruebas de microservicios?

3 responses

La comunicación entre los mismos

Cambio de paradigma

La escabilidad y manejo de errores.

9. ¿Considera que es necesaria la automatización para probar los microservicios?

3 responses

Si

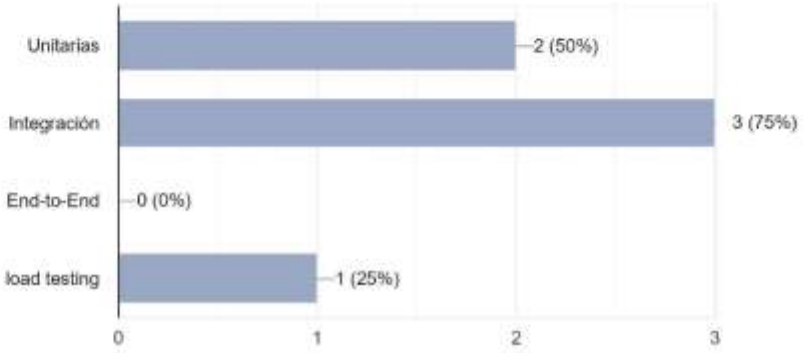
no se

Sumamente necesaria debido a que una aplicacion empresarial contendra muchos

microservicios y a la hora de realizar un cambio es necesario probar todos de los microservicios.

10. ¿Qué tipo de pruebas realiza normalmente sobre microservicios?

4 responses



11. ¿Cuáles herramientas utiliza para pruebas End-to-End?

4 responses

No aplica

Postman

moleculer

Unit Test

Apéndice D

Lista de *Docker captains*

Este anexo contiene la lista de capitanes *Docker* que fueron contactados para efectos de la segunda encuesta realizada en nuestro estudio.

La siguiente tabla contiene el nombre, la ubicación, la posición y el estado de cada capitán. La columna "estado" indica "enviado" para aquellos expertos a quienes se envió el cuestionario pero no se obtuvo respuesta; y "exitoso" para aquellos expertos a quienes se envió el cuestionario y completaron el cuestionario.

Id	Nombre	Ubicación	Posición	Estado
1	Andrian Mouat	Edinburgh, United Kingdom	Chief Scientist, Container Solutions	Exitoso
2	Adrien Blind	Rueil-Malmaison, France	DevOps Coach	Enviado
3	Ajeet Singh Raina	Bengaluru, Karnataka India	Senior Systems Development Engineer	Enviado
4	Andrey Sibiryov	New York, NY United States of America	Dropbox	Exitoso
5	Antonis Kalipetis	Athens, Greece	CTO	Enviado
6	Arun Gupta	San Jose, CA United States of America	Principal Open Source Technologist at Amazon Web Services	Enviado
7	Brandon Mitchell	Virginia United States of America	Solutions Architect, BoxBoat	Enviado
8	Bret Fisher	Virginia United States of America	DevOps Trainer and Consultant	Enviado
9	Brian Christner	Zürich, Switzerland	Cloud Architect & Advocate	Exitoso
10	Chanwit Kaewkasi	Thailand Thailand	Assistant Professor	Enviado
11	Dieter Reuter	Bamberg, Germany	Senior Consultant	Enviado
13	Gianluca Arbezzano	Italy Italy	Site Reliability Engineer	Enviado
14	Javier Ramírez	Madrid, Spain	IT Architect	Enviado
16	John Lees-Miller	London, United Kingdom	Cofounder and CTO at Overlea	Exitoso
19	Laura Frank Tacho	Berlin Germany	Director of Engineering	Enviado
20	Lee Calcote	TX United States of America	Head of Technology Strategy	Enviado
21	Lorenzo Fontana	Verbano Cusio Ossola Italy	DevOps Expert	Enviado

23	Manuel Morejón	Madrid, Spain	DevOps Engineer	Enviado
24	Marcos Lilljedahl	Buenos Aires, Argentina	Head of R&D	Enviado
25	Michael Irwin	Blacksburg, United States of America	Application Architect	Exitoso
27	Nicolas De Loof	New York United States of America	Full stack developer / Teacher	Enviado
28	Nigel Poulton	Livre sur Changon, France	-	Enviado
31	Scott Coulton	Lyon, France	Founder	Enviado
32	Thomas Shaw	Sydney, NSW Australia	Cloud developer	Exitoso
33	Viktor Farcic	Dublin, Ireland	Build Engineer	Exitoso

Apéndice E

Cuestionario para los *Docker Captains* internacionales

El cuestionario correspondiente a la segunda encuesta realizada con expertos internacionales, se puede consultar en el enlace: https://forms.gle/SZqvu9pyqwwLsxwv5

Microservices

This survey will be applied to experts in microservices of Docker's community. The survey is anonymous and takes around 5 minutes to answer it. The results of it will be used as part of a final work of applied research to opt for the grade of Professional Master's Degree in Computer Science at the University of Costa Rica.
We appreciate your participation.
* Required

1. 1) How long time have you been using microservices? *

   Mark only one oval.

   - ◯ Less than 2 years
   - ◯ Between 2 and 5 years
   - ◯ Between 5 and 8 years
   - ◯ More than 8 years

2. 2) How many years of experience do you have in software testing? *

   Mark only one oval.

   - ◯ Less than 2 years
   - ◯ Between 2 and 5 years
   - ◯ Between 5 and 8 years
   - ◯ More than 8 years

3. 3) Have you done microservices testing ? *

   Mark only one oval.

   - ◯ Yes
   - ◯ No
   - ◯ Other: _____

4. 4) Which programming language have you used to work with microservices? *

5. 5) What are the main challenges in microservices testing? *

6. 6) Do you consider that it is necessary the automatization in microservices testing? *

7. 7) How does the testing of microservices differ from the testing of "more traditional" software? *

8. 8) What kind of testing do you normally do in microservices? *

Check all that apply.

- [ ] Unitaries
- [ ] Integration
- [ ] End-to-End
- [ ] Other: _____

9. 9) Which tools do you use to make End-to-End testing? *

_____

_____

_____

_____

_____

Apéndice F

Resultados de la encuesta pasada a los *Docker Captains*

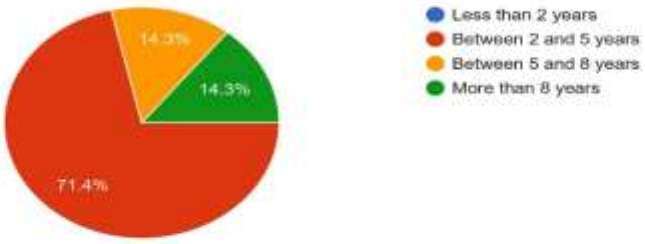
Microservices

7 responses

Publish analytics

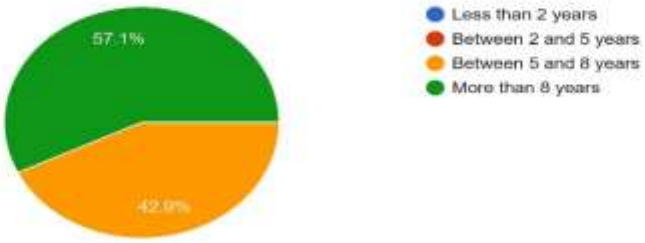
1) How long time have you been using microservices?

7 responses



2) How many years of experience do you have in software testing?

7 responses



3) Have you done microservices testing ?

7 responses

4) Which programming language have you used to work with microservices?

7 responses

Node mostly

Rust, Go, Python

Go, Python, C++

Bash, Python, Go, Java

Python

node.js

Go

5) What are the main challenges in microservices testing?

7 responses

Reliably mocking other dependent microservices

Interactions with other services.

Setting up the scaffolding & request routing fabric for all microservice dependencies that the service under testing requires and expects to function.

Testing Managed Services

Consistency of quality across services, communication between teams, standardization.

Setting up all the services in the test environment is slow and complicated, so people mock things out, and then the mocks diverge from reality, leading to tests that run fast but take a lot of dev time and deliver no actual value.

Coordination and dependency management

6) Do you consider that it is necessary the automatization in microservices testing?

7 responses

Yes

Not really sure what is being asked here. If you're thinking of a particular tool or technique, mention it.

Yes.

Depends how it is implemtned. For example, with Auth0 we don't test it

Yes. Otherwise you are entering a world of pain.

Yes. A bit of manual 'smoke testing' is fine, but complicated test scenarios should be automated.

yes

7) How does the testing of microservices differ from the testing of "more traditional" software?

4 responses

In general it's easier, as the individual microservices are small and easy to test. But new problems can arise due to complicated interactions between multiple microservices.

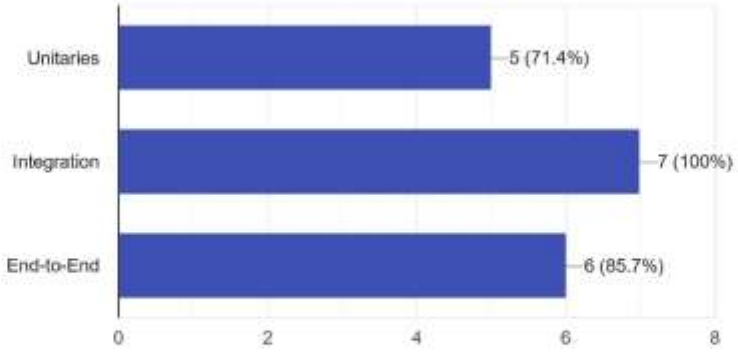
It's much easier to spin up a monolith and call one of its functions than to spin up tens of interdependent microservices in order to test one of the request flows they provide.

There is much more incidental complexity, leaving less resource to test (or deliver) actual business logic.

it does not

8) What kind of testing do you normally do in microservices?

7 responses



9) Which tools do you use to make End-to-End testing?

7 responses

Selenium-based tools. Arquillian. Cucumber

curl :)

Bazel, Jenkins, proprietary tools.

GitLab, Jaeger

Home grown tools, usually driven through CI/CD systems and may use tooling such as docker, compose, ansible, terraform to setup and teardown infrastructure matching prod, deploying services across the infrastructure and running real world type scenarios, user testing, load testing.

Mostly jsdom. Some things we now test with puppeteer.

Go and testcontainers

Zipkin

Apéndice G

Tabla completa con la evaluación de las 40 herramientas

ID	Tool's Name	Metrics	Documentation	Q&C of techical support	Integrability	Cost	Platforms Compatibility	Total
1	Jaeger	30	20	10	10	20	5	**95**
2	Zipkin	30	20	10	10	20	5	**95**
3	TestProject.io	15	20	15	10	20	5	**85**
4	Spring Cloud Contract	15	20	15	2	20	5	**77**
5	BrowserStack	15	20	15	10	10	5	**75**
6	Sealights	15	20	15	10	10	5	**75**
7	Gatling	15	10	15	5	20	5	**70**
8	ElasTest	15	10	10	10	20	5	**70**
9	Nightwatch.js	15	10	10	5	20	5	**65**
10	Saucelabs	15	10	15	10	10	5	**65**
11	Prevant	15	5	10	10	20	5	**65**

ID	Tool's Name	Metrics	Documentation	Q&C of techical support	Integrability	Cost	Platforms Compatibility	Total
12	Endtest	15	10	15	10	10	5	**65**
13	Selenium IDE	0	20	15	10	10	5	**60**
14	Cucumber	0	20	15	10	10	5	**60**
15	Jenkins	0	20	15	10	10	5	**60**
16	Appium	0	10	15	10	20	5	**60**
17	Docker Compose Rule	0	10	15	10	20	5	**60**
18	Applitools	0	20	15	10	10	5	**60**
19	Testim	0	20	15	10	10	5	**60**
20	Bazel	0	10	15	10	20	5	**60**
21	Go	0	10	15	10	20	5	**60**
22	Gremlin	0	20	15	10	10	3	**58**
23	Benchflow	15	5	0	10	20	5	**55**
24	WebDriverIO	0	10	15	2	20	5	**52**

ID	Tool's Name	Metrics	Documentation	Q&C of techical support	Integrability	Cost	Platforms Compatibility	Total
25	GitOps	0	10	15	10	10	5	**50**
26	GitLab Review Apps	0	10	15	10	10	5	**50**
27	MicroProfile	0	10	10	5	20	5	**50**
28	Mabl	0	10	15	10	10	5	**50**
29	Curl	0	0	15	10	20	5	**50**
30	JMeter	0	5	15	2	20	5	**47**
31	Protractor	0	10	10	2	20	5	**47**
32	Testcontainers	0	5	15	2	20	5	**47**
33	Browserling	0	5	15	10	10	5	**45**
34	Hoverfly	0	10	15	2	10	5	**42**
35	Arquillian	0	5	10	2	20	5	**42**
36	Nightcloud	0	5	0	10	20	5	**40**
37	Jsdom	0	10	0	2	20	5	**37**
38	Puppeteer	0	10	0	2	20	5	**37**
39	Test.AI	0	5	0	10	10	5	**30**

ID	Tool's Name	Metrics	Documentation	Q&C of techical support	Integrability	Cost	Platforms Compatibility	Total
40	MIDAS	0	0	0	0	0	0	**0**