UNIVERSIDAD CÉSAR VALLEJO

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Estudio comparativo cuantitativo de las tecnologías Microservicios y REST

AUTORES:

Macarlupu Paredes, Anderson Joel (ORCID: 0000-0002-3782-8114)

Marin Inga, Eduardo (ORCID: 0000-0001-9996-4755)

ASESORES:

Dr. Alfaro Paredes, Emigdio Antonio (ORCID: 0000-0002-0309-9195)

Mgtr. Liendo Arévalo, Milner David (ORCID: 0000-0002-7665-361X)

LÍNEA DE INVESTIGACIÓN:

Sistema de Información y Comunicaciones

LIMA – PERÚ

2020

Dedicatoria

A nuestra familia, universidad, profesores y principalmente a Dios por permitirnos llegar a este punto y darnos salud para lograr nuestras metas, además de su infinita bondad y amor.

Agradecimiento

Queremos agradecer a las autoridades y profesores de la Universidad César Vallejo y a nuestros familiares, quienes apoyaron el desarrollo de nuestra tesis. También estamos felices de completar esta etapa universitaria y convertirnos en profesionales.

Índice de contenidos

Índice de tablas

Índice de figuras

Resumen

La presente investigación fue desarrollada con el objetivo de comparar las arquitecturas microservicios y REST para determinar cuál presenta un mejor rendimiento de acuerdo al tiempo de respuesta, uso de recursos y nivel de seguridad utilizando la metodología METSA (Methodology for Evaluating Technologies of Service Architecture o Metodología para la evaluación de tecnologías de arquitecturas de servicios), la cual fue desarrollada como parte de esta investigación. Además, se desarrollaron cuatro aplicaciones con ambas arquitecturas para realizar las pruebas respectivas. Como resultado se mostró que las tecnologías de microservicios son ligeramente superiores que las tecnologías REST, por tener menor tiempo de respuesta de carga de imágenes y menor cantidad de vulnerabilidades; sin embargo, la tecnología REST fue superior en cuanto a la cantidad de ataques bloqueados.

Se recopiló estudios de distintas revistas y libros con un enfoque de evaluación de arquitecturas de servicios, técnicas y métodos de evaluación de software. Con ello se formó la metodología METSA que está compuesta por cuatro procesos: (a) preparar el entorno de pruebas, (b) ejecutar la prueba estrés (en este proceso se evaluó el indicador tiempo de carga de usuarios), (c) ejecutar la prueba de penetración (se evaluó los indicadores “cantidad de vulnerabilidades” y “ataques bloqueados”) y (d) ejecutar la prueba de carga (en este último proceso se evalúo los indicadores: uso de CPU, uso de memoria RAM y tiempo de respuesta de carga de usuarios). Asimismo, la metodología METSA utilizó herramientas gratuitas, tales como: JMeter Apache, OWASP Zap, Monitor de red de Firefox, Docker y Netdata. Finalmente, se propuso recomendaciones para ampliar el nivel de comparación utilizando nuevos indicadores y herramientas.

Palabras clave: Metodología para la evaluación, arquitectura microservicios, REST, tecnologías de microservicios, tecnologías de REST, arquitectura de servicios.

Abstract

This research was developed with the objective of comparing microservices and REST architectures to determine which one presents better performance according to response time, resource use and security level using the METSA methodology (Methodology for Evaluating Technologies of Service Architecture or Methodology for evaluation service architecture technologies), which was developed as part of this research. In addition, four applications were developed with both architectures to perform the respective tests. As a result, microservices technologies were shown to be slightly superior to REST technologies, as they have a shorter response time for loading images and fewer vulnerabilities; however, REST technology was higher in terms of the number of blocked attacks.

Studies were compiled from different journals and books with a focus on evaluating service architectures, techniques and software evaluation methods. With this, the METSA methodology was formed, which is composed of four processes: (a) to prepare the test environment, (b) to execute the stress test (in this process the user load time indicator was evaluated), (c) to execute the penetration test (the indicators "number of vulnerabilities" and "attacks blocked" were evaluated), and (d) to execute the load test (in this last process, the following indicators were evaluated: CPU usage, RAM memory usage, and response time user load). Likewise, the METSA methodology used free tools such as: JMeter Apache, OWASP Zap, Firefox Network Monitor, Docker, and Netdata. Finally, recommendations were proposed to broaden the level of comparison using new indicators and tools.

Keywords: Assessment methodology, microservices architecture, REST, microservices technologies, REST technologies, service architecture.

I. INTRODUCCIÓN

En esta investigación se realizó una comparación cuantitativa entre las arquitecturas de microservicios y REST. Debido a que no se encontraron estudios relacionados que permitan comparar estas dos arquitecturas, se elaboró una metodología que permitió evaluar el desempeño de los indicadores asociados a: tiempo de respuesta, nivel de seguridad y uso de recursos. Las justificaciones fueron: teórica, metodológica y tecnológica. Se planteó como hipótesis que la arquitectura de microservicios tuvo mejor rendimiento que la arquitectura REST en los indicadores planteados para la investigación.

Para esta investigación se recopiló artículos relacionados a estudios comparativos entre tecnologías similares a microservicios y REST. Tihomirovs y Grabis (2016) realizaron una comparación entre servicios SOAP y REST de acuerdo a un conjunto de métricas, concluyendo que no se puede afirmar que una tecnología es mejor que otra, porque dependerá de la necesidad del sistema. También, Shrestha (2019) comparó las diferencias entre paradigmas arquitectónicos como monolítico, nativo de la nube y microservicios para encontrar el paradigma apropiado que satisfaga a la empresa para continuar su negocio digital y concluyó que la arquitectura de microservicios presenta mayores ventajas y ha resuelto la mayoría de problemas que las empresas han identificado.

Por otro lado, no se encontraron estudios que comparen la arquitectura de microservicios y REST. Los estudios encontrados comparan estas arquitecturas con otras similares utilizando diferentes indicadores. Al respecto, Tihomirovs y Grabis (2016) utilizaron los indicadores de costo, velocidad de ejecución, línea de código, memoria y funcionalidad para comparar los servicios de SOAP y REST. En la investigación de Tihomirovs y Grabis (2016) no se realizó pruebas referentes al nivel de seguridad.

Debido a la falta de estudios similares, se elaboró la metodología METSA para evaluar las tecnologías de ambas arquitecturas de acuerdo a los indicadores tiempo de respuesta, uso de recursos y nivel de seguridad, a través de cuatro procedimientos que fueron recopilados de técnicas y métodos utilizados en la evaluación de software. Al no haber estudios similares, no se podía asegurar que la arquitectura de microservicios es mejor que REST y se

desconocía las ventajas de una tecnología respecto a la otra en los indicadores de uso de recursos, tiempo de respuesta y nivel de seguridad. Estos aspectos han dificultado la adecuada elección de estas arquitecturas para el desarrollo y el uso del software.

Las justificaciones que se tomaron para la investigación fueron: teórica, metodológica y tecnológica. Se tuvo una justificación teórica porque se realizó un estudio comparativo entre dos tecnologías, lo que permitió incrementar el conocimiento de estas tecnologías en base a los indicadores planteados. Se tuvo una justificación metodológica porque se elaboró y probó una metodología para la evaluación de tecnologías basadas en las arquitecturas microservicios y REST. Finalmente, se tuvo una justificación tecnológica porque se comparó tecnologías con una alta demanda, lo que ayudará la toma de decisiones sobre el uso de estas tecnologías para los gerentes de tecnologías de información y comunicaciones.

La justificación teórica de esta investigación se basó en el aporte de conocimiento generado a través de la comparación de dos tecnologías: microservicios y REST, para evaluar los criterios de tiempo de respuesta, uso de recursos y nivel de seguridad y determinar cuál de estas arquitecturas tiene mejor rendimiento bajo los indicadores planteados. Athar, Liaqat y Azam (2016) mencionaron que existen métodos para mejorar la calidad del sistema respecto a indicadores como: rendimiento, confiabilidad, mantenibilidad y usabilidad. Además, Jha y Popli (2017) indicaron que los principales indicadores para evaluar las arquitecturas de software son: (a) tiempo de respuesta, (b) latencia, (c) tasa de transferencia efectiva, (d) carga de trabajo, (e) uso de recursos y (f) desviación estándar; además, mencionaron que la evaluación de los indicadores mencionados sirve para garantizar que las aplicaciones funcionen correctamente con una carga de trabajo esperado.

Como justificación metodológica cabe resaltar que el uso de un marco o método es importante, ya que puede respaldar la escalabilidad, el rendimiento y la funcionalidad. Asimismo, se recopilaron métodos y técnicas para la investigación con el objetivo de generar una nueva metodología para la evaluación de rendimiento. Ashraf y Aljedaibi (2017) mencionaron que la

evaluación de la arquitectura se ha vuelto más importante debido a las complejidades cada vez mayores en el desarrollo de software. También, Ashraf y Aljedaibi (2017) afirmaron que el análisis de la arquitectura del sistema en las primeras etapas detecta y elimina los mayores defectos con un esfuerzo y costo mínimos. Además, Guardiola (2020) mencionó que a lo largo del tiempo se ha desarrollado y documentado diferentes guías para realizar pruebas de penetración y que mediante estas metodologías se puede descubrir y evaluar vulnerabilidades de una plataforma informática como Kubernetes (p. 18).

Esta investigación fue justificada tecnológicamente debido a que se hizo una contribución al conocimiento tecnológico a través de comparación de las arquitecturas microservicios y REST utilizando la metodología propuesta. Esto permitió conocer las ventajas y desventajas de cada arquitectura; además, se utilizó la tecnología microservicios, la que está en crecimiento. Al respecto, Richards (2016) explicó el crecimiento en la implementación de microservicios en las industrias de tecnologías de información debido a las ventajas que tiene frente a otras arquitecturas al desarrollar aplicaciones modulares y por ser altamente escalable. Además, Tapia et al. (2020) mencionaron que la arquitectura de microservicios está ganando espacio, que será parte del proceso de toma de decisiones en el ámbito técnico y financiero y que los microservicios están reemplazando los sistemas de arquitecturas monolíticas tradicionales.

El problema general de la investigación fue que no se han encontrado estudios que permitan comparar las tecnologías microservicios y REST según el rendimiento en el uso de recursos, el tiempo de respuesta y el nivel de seguridad. Por otro lado, los problemas específicos de la investigación fueron los siguientes:

- **PE1:** No se ha encontrado estudios que permitan comparar las tecnologías microservicios y REST según el uso de recursos.
- **PE2:** No se ha encontrado estudios que permitan comparar las tecnologías microservicios y REST según el tiempo de respuesta.
- **PE3:** No se ha encontrado estudios que permitan comparar las tecnologías microservicios y REST según el nivel de seguridad.

El objetivo general fue comparar las tecnologías de microservicios y REST según su rendimiento en base al uso de recursos, tiempo de respuesta y nivel de seguridad. Los objetivos específicos fueron los siguientes:

- **OE1:** Comparar las tecnologías de microservicios y REST según el uso de recursos.
- **OE2:** Comparar las tecnologías de microservicios y REST según el tiempo de respuesta.
- **OE3:** Comparar las tecnologías de microservicios y REST según el nivel de seguridad.

La hipótesis general de la investigación fue: “Las tecnologías de microservicios tuvieron un mejor rendimiento que las tecnologías REST”. Con la propuesta de las arquitecturas de microservicios viene el uso de la tecnología de contenedores como Docker. Al respecto, Scott (2017) indicó que los contenedores disponen cada carga de trabajo para obtener acceso a diversos recursos: (a) procesador, (b) memoria, (c) cuenta de servicios y (d) librerías, lo cual es primordial para un mejor rendimiento del proceso de desarrollo. Además, Scott (2017) mencionó que los contenedores corren como un grupo de técnicas y/o procesos aislados dentro de un sistema operativo, lo cual hace más rápido su rendimiento al iniciar y dar mantenimiento (p. 25). Las hipótesis especificas fueron los siguientes:

HE1: Las tecnologías de microservicios tuvieron menor uso de CPU que las tecnologías REST.

Belinchón (2018) mencionó que Docker presenta un menor uso de recursos y mayor rendimiento porque el sistema operativo es compartido y los contenedores comparten procesos del núcleo del sistema operativo, por lo que existe una mejora en el rendimiento respecto a las aplicaciones que arrancan un sistema operativo completo (p. 19). Además, Steinholt (2015) concluyó que los servicios web modernos deberían disponer de nuevas tecnologías, con la finalidad de consumir una menor cantidad de recursos posibles cuando el tráfico sea menor.

HE2: Las tecnologías microservicios tuvieron menor uso de memoria RAM que las tecnologías REST.

Tapia (2020) mencionó que microservicios presenta menor consumo de memoria RAM que las aplicaciones monolíticas, esto es gracias al uso de contenedores que encapsula cada servicio utilizado. Además, Flygare y Holmqvist (2017) mencionaron que la arquitectura microservicios tiene un uso menor de memoria RAM que una arquitectura monolítica.

HE3: Las tecnologías de microservicios tuvieron menor tiempo de respuesta de carga de imágenes que las tecnologías REST.
Gómez (2017) indicó que tener servicios escalables, adaptables, modulares y rápidamente accesibles hace que los microservicios favorezcan la evolución del software para dar una mejor respuesta a las demandas de los clientes en entornos cambiantes (p. 122). Tihomirovs y Grabis (2016) mencionaron que REST obtuvo tiempo de respuesta más cortos y un mejor rendimiento que SOAP. También mencionaron que REST tiene mejor desempeño en redes que presentan mayor carga. Además, Halili (2018) mencionó que REST presenta un mejor rendimiento al consumir el servicio web, esto debido a que utiliza un formato de mensaje más pequeño llamado JSON que no requiere un procesamiento intensivo (p. 179).

HE4: Las tecnologías de microservicios tuvieron menor tiempo de respuesta de carga de usuarios que las tecnologías REST.
Mikuła y Dzieńkowski (2020) mencionaron que para las pruebas de carga se utiliza un número diferente de usuarios que generan solicitudes como: “1, 5, 50, 100, 500 y 1000” simultáneamente; además, utilizaron métricas para comparar el rendimiento de las aplicaciones que utilizan diferentes estilos de intercambio de datos (p. 312). Además, Al-Debagy y Martinek (2018) compararon las arquitecturas microservicios y monolítica y concluyeron que los rendimientos de las dos arquitecturas presentan un promedio similar al intentar cargar información por medio un servidor.

HE5: Las tecnologías microservicios tuvieron menor cantidad de vulnerabilidades que las tecnologías REST.

Guardiola (2017) mencionó que solo se encontraron tres vulnerabilidades al momento al momento de analizar la seguridad en microservicios: a nivel de aplicación, a nivel de permisos y a nivel de contenedores. Además, Mateus, Cruz y Gonzaga (2020) mencionaron que las aplicaciones que utilizan microservicios también pueden ser víctimas de ataques críticos mencionados por la fundación OWASP. (p. 7). Sin embargo, Mateus, Cruz y Gonzaga (2020) también mencionaron que las solicitudes entre servidores suelen ser más seguras si se aplican medidas como firewall o segregación de red en microservicios (p. 8).

HE6: Las tecnologías REST tuvieron mayor porcentaje de ataques bloqueados que las tecnologías microservicios.

Mateus, Cruz y Gonzaga (2020) indicaron que no se puede afirmar que los microservicios presentan ventajas en términos de seguridad debido a su complejidad. Sin embargo, Bozhinoski y Pretschner (2019) indicaron que la arquitectura de microservicios y el uso de contenedores presenta serios riesgos de seguridad principalmente por sus requisitos de conectividad.

II. MARCO TEÓRICO

En este capítulo se muestra los antecedentes a través de estudios comparativos similares al tema de investigación. Luego, se describe las teorías relacionadas, que están comprendidas por las tecnologías, herramientas e indicadores utilizados en el desarrollo de esta investigación.

La siguiente sección de antecedentes cuenta con quince estudios comparativos similares al presente trabajo. Entre ellas se tiene a Arellano (2020), quien aplicó la metodología RUP para mejorar el rendimiento de un sistema informático. Por otro lado, Tihomirovs y Grabis (2016) compararon las tecnologías de REST y SOAP utilizando un conjunto de métricas para su evaluación. También, Shrestha (2019) estudió las diferencias entre paradigmas en las arquitecturas de software como monolíticas, microservicios y en la nube demostrando que son eficaces para el desarrollo aplicaciones web.

Arellano (2020) estudió la influencia del desarrollo de un sistema informático con la metodología RUP para la gestión académica de los estudiantes de un instituto, realizando una investigación pre-experimental con la participación de 367 estudiantes. Arellano (2020) concluyó que se llegó mejorar el rendimiento en el tiempo de promedio de registro y el tiempo de promedio de generación de notas. Finalmente, Arellano (2020) recomendó que para futuras investigaciones similares se usen los indicadores de tiempo de registro y tiempo de generación de notas para un buen proceso de gestión académica.

Tihomirovs y Grabis (2016) compararon los servicios SOAP y REST considerando distintas métricas de evaluación de software. Para ello se investigó sistemáticamente la literatura donde se defina un método de comparación conceptual para evaluar las arquitecturas SOAP y REST. Tihomirovs y Grabis (2016) concluyeron que cada arquitectura tiene un enfoque distinto; por ejemplo, si un proyecto se integra con dos sistemas de información simples, la opción correcta será REST; sin embargo, si sus sistemas son complejos y deben tener un buen nivel de seguridad, entonces la elección correcta será SOAP.

Pinchao (2020) estudió la forma de implementar un curso de MOOC aplicando la metodología MIA mediante la plataforma Edject para el instituto MIA Ibarra Ecuador. Pinchao (2020) presentó tres metodologías: la metodología XP que fue para el proceso desarrollo del sistema, la metodología DICREVOA 2.0 que se utilizó para la virtualización y la metodología MIA. Pinchao (2020) concluyó que la colección de información realizado ayudó a comprender los diferentes tipos de atención plena, generando buenas practicas días diarias educativas y estudios científicos hacia los usuarios.

López Hinojosa (2017) estudió la arquitectura de microservicios para el desarrollo de una aplicación web en paralelo con la Asamblea General Nacional de Ecuador y tecnologías de la información. López Hinojosa (2017) mencionó que luego de la información recopilada se obtuvo nuevas tecnologías y metodología existentes para la ejecución de los microservicios, enfocándose en un diseño de investigación cualitativo – descriptivo. Finalmente, López Hinojosa (2017) concluyó que los microservicios propuestos cumplen con las necesidades de evaluación planteados a través del método ATAM con las arquitecturas que componen junto a las tecnologías seleccionadas.

Shrestha (2019) estudió las diferencias entre paradigmas arquitectónicos como monolítico, nativo en la nube y microservicios para encontrar el paradigma apropiado que satisfaga a la empresa para continuar su negocio digital. Shrestha (2019) usó diferentes plataformas y entornos para una posible mejora en el rendimiento de desarrollo de software que permita ejecutar, desarrollar y enviar aplicaciones fácilmente en cualquier lugar. Finalmente, Shrestha (2019) concluyó que el modelo de microservicios es uno de los modelos de despliegue de servicios adecuados para sistemas comerciales para grandes empresas.

Ruelas (2019) investigó la manera de reforzar una aplicación web de comercio electrónico implementando un modelo de composición enfocado en microservicios utilizando la tecnología Kubernetes. Ruelas (2019) utilizó un modelo de proceso de negocio de comercio electrónico para la composición de microservicios para el análisis de requerimientos arquitectónicos. Finalmente, Ruelas (2019) obtuvo como resultado que la implementación de composición

de microservicios funciona efectivamente en relación a un modelo monolítico en un 104% en relación a su rendimiento y tiempo de respuesta.

Sánchez (2018) estudió la automatización de un clúster de microservicios basados en SOA. Además, Sánchez (2018) realizó una síntesis de herramientas nuevas para que permitan llevar a cabo una configuración propia de las máquinas y desplegar sin necesidad de hacer gran esfuerzo. Sánchez (2018) concluyó que el diseño del despliegue automatizado de un clúster de microservicios en el orquestador Kubernetes fue realizado de una manera rentable usando herramientas adicionales como Vagrant o Ansible que fueron parte del proyecto.

Zirkelbach, Krause y Hasselbring (2019) estudiaron el proceso de modularización y la arquitectura del proyecto de investigación de código abierto de ExploViz hacia una arquitectura de Microservicios, con el objetivo de facilitar el desarrollo colaborativo de todos los estudiantes impulsando la extensión de un mejor desarrollo a futuro, ofreciendo un mecanismo de extensión de complementos para un proyecto por lado del front end y del back end. Zirkelbach et al. (2019) concluyeron que implementar la arquitectura de microservicios es garantizada y recomendada para el desarrollo de aplicaciones web o distintos proyectos.

Steinholt (2015) investigó la capacidad que tiene un contenedor de Linux para permitir el escalado rápido de los servicios web. Steinholt (2015) utilizó una tecnología relativa que son los contenedores permitiendo la virtualización y la ejecución de aplicación. Finalmente, Steinholt (2015) concluyó que los servicios web modernos deberían poder utilizar Docker y Kubernetes con la finalidad de consumir menos cantidad de recursos posibles cuando el tráfico sea menor.

Simbaña (2016) estudió las herramientas Docker y Bitnami para comparar su funcionalidad y despliegue de aplicaciones web. Simbaña (2016) tuvo como finalidad darles una propuesta de implementación a la plataforma Cloud FICA contenedores, denominados como máquinas virtuales que son

menos exigentes al uso de recursos para el despliegue ligero y eficaz de las plataformas.

Mahjani (2015) estudió los problemas de seguridad de la virtualización en los entornos de computación en la nube y también analizó las técnicas de mitigación para mejorar la confiabilidad de la seguridad de los sistemas en la nube, analizando distintos artículos para identificar los problemas de seguridad más relevantes de la virtualización. Mahjani (2015) concluyó que se aplicará cuatros métodos diferentes para prevenir algunos problemas más concurrentes que se presenten en las aplicaciones web para obtener la seguridad de las aplicaciones al ser ejecutadas.

Cebeci (2019) estudió distintos diseños de software donde aplicó un diseñó de sistema de software fácilmente escalable, mantenible, altamente disponible, confiable y observable, mediante la comparación de arquitecturas variantes, métodos de comunicación y modelos de datos que ayudarían a elegir la arquitectura o modelo más apropiado para el propósito correcto. Simultáneamente se realizó una investigación de los beneficios de los microservicios y mejores prácticas para desarrollar con microservicios y contenedores (Cebeci, 2019).

Koskinen (2019) evaluó las mejores prácticas y beneficios de la arquitectura microservicios y contenedores para el desarrollo de la API del proyecto OneCloud de Ericsson. Koskinen (2019) concluyó que la información obtenida facilita el escalamiento independiente, ciclo de vida independiente y heterogeneidad tecnológica para que el equipo de desarrollo pueda construir los microservicios con las tecnologías elegidas ahora y en el futuro.

En la siguiente sección se describe las teorías relacionadas. Salvadori (2017) definió a los microservicios como pequeños servicios independientes. Además, Tarkowska (2018) indicó que la arquitectura REST es un método de comunicación entre un cliente y un servidor a través de un protocolo de transferencia de hipertexto (HTTP). También, se definieron los indicadores tiempo de respuesta, uso de recursos y nivel de seguridad y se describieron las

herramientas NetData, JMeter, Owas Zap y Monitor de red de Firefox (Tarkowska, 2018).

En los últimos años se ha hablado mucho de la arquitectura de microservicios, ya que son un conjunto de componentes distribuidos en un sistema donde cada componente expondrá la funcionalidad al resto del sistema, por lo que de esta forma se modularizan los sistemas a través de estos servicios independientes. Salvadori (2017) indicó: “Los microservicios son servicios pequeños e independientes. Sin embargo, no está definido en la literatura cuan pequeño e independiente debe ser un microservicio” (p. 4).

Salvadori (2017) mencionó: “Los microservicios se centran en satisfacer la calidad de los requisitos de software presentes en un dominio bien definido, que puede dividirse en múltiples contextos delimitados” (p. 4). Al tratarse de servicios independientes, las tecnologías utilizadas pueden ser distintas; al respecto, Salvadori (2017) mencionó: “Una vez que los microservicios son relativamente independientes entre sí, cada uno puede desarrollarse utilizando la tecnología más adecuada para su propósito” (p. 4). A continuación, se presenta la figura 1 donde se muestra la arquitectura microservicios y la distribución de sus servicios.

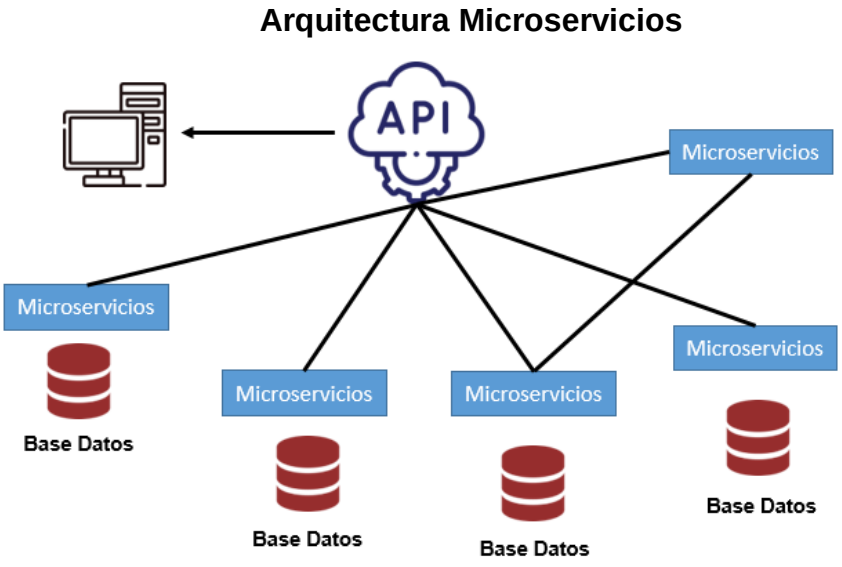


Figura 1 Arquitectura de microservicios

Adaptado de Salvadori (2017)

Tapia (2020) mencionó: “Los microservicios tienen la necesidad de ejecutar y realizar cambios en el software de forma rápida y sencilla” (p. 15). Actualmente, los programadores y desarrolladores están dejando de lado arquitecturas monolíticas en beneficio de los sistemas de microservicios. Tapia (2020) indicó: “Las organizaciones comienzan a darle importancia al proceso de productos de piezas intercambiables, actualizables y escalables, de manera que paulatinamente se van acomodando en un pensamiento centrado en lo humanitario, donde se requiere un flujo fluido de información que pueda apreciar el incremento por microservicios” (p. 26).

La arquitectura REST está basada en servicios web. Verborg (2015) mencionó: “Más que una tecnología o estándar específico, REST es un estilo arquitectónico que se ha utilizado para orientar el diseño y desarrollo de la arquitectura para la Web moderna” (p. 239). También, Tarkowska (2018) explicó: “La transferencia de estado representacional (REST) es un método popular para proporcionar interoperabilidad entre un cliente y un servidor utilizando el protocolo de transferencia de hipertexto (HTTP)” (p. 1). Además, utiliza el mismo bloque de construcción como la red mundial y un formato de intercambio común, por ejemplo, JSON” (Tarkowska, 2018). A continuación, en la figura 2 donde se muestra el funcionamiento de la arquitectura REST.

Arquitectura REST

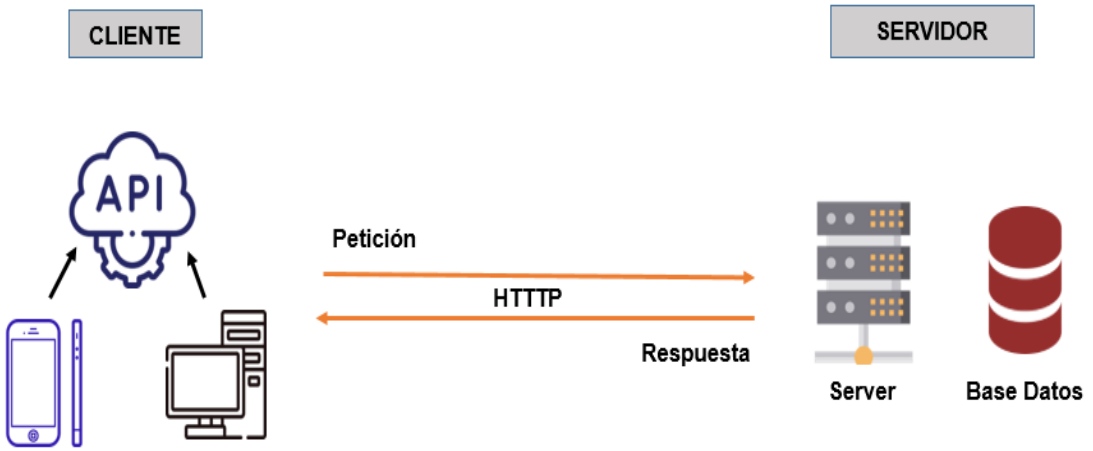


Figura 2 Arquitectura de REST.
Adaptado de Vergord (2015).

Para establecer los servicios web derivados de la arquitectura, se implementa un RESTful. Al respecto, Pielli (2015) explicó: “Para que un servicio sea identificado como RESTful, se deben respetar las siguientes cinco restricciones: (a) cliente-servidor, (b) sin estado, (c) sistema en capas, (d) almacenable en caché y (e) interfaz uniforme” (p. 4).

Entre las restricciones de esta arquitectura se encuentra el cliente-servidor que permite la comunicación con el servidor enviando y recibiendo información. Al respecto, Halili (2018) indicó que REST representa una arquitectura cliente-servidor donde el cliente envía las solicitudes, mientras que el servidor las procesa y devuelve las respuestas, que fue introducido en el año 2000 por Roy Fielding y que a diferencia de SOAP, los servicios REST no se limitan a XML, sino que también admiten JSON (p. 175).

Respecto a los indicadores, el tiempo de respuesta trata de la interacción del cliente y el servidor, a la hora de abstraer información o datos, a través del tiempo de respuesta y la precisión de las respuestas. Tihomirovs y Grabis (2016) mencionaron que la velocidad de ejecución del servicio web se puede medir objetivamente por el tiempo de respuesta cuando un servicio web devuelve una respuesta al cliente.

El indicador de uso de recursos evalúa la automatización del desempeño a través de pruebas que serán realizadas por herramientas que analizarán su tiempo de respuesta. Verona, Pérez, Torres, Delgado y Yáñez (2016) indicaron:

> Las pruebas de rendimiento sobre cualquier producto software muestran un comportamiento transversal a la ejecución de las funcionalidades. Cualquier herramienta que realice este tipo de pruebas, aunque no utilice el paradigma orientado a aspectos, actúa de forma transversal al sistema. El detalle es que para evaluar los elementos de desempeño es necesario ejecutar las funcionalidades del sistema. Pero, en la ejecución de una funcionalidad dentro de un sistema existen muchas otras funcionalidades que se activan consecutivamente después de la llamada a un determinado método, por lo que la herramienta estaría evaluando muchas funcionalidades que se encuentran en diferentes partes del sistema. (p. 19)

Corredor (2017) mencionó que el indicador nivel de seguridad es un tema muy relevante debido a las funciones de los sistemas integrados en muchos sistemas críticos para la seguridad (p. 51). Sin embargo, Corredor (2017) explicó la diferencia de ataques y la cantidad de vulnerabilidad que puedan ser causados por los errores en un programa, sistema de aplicación o por funciones intencionales que vienen en formas legítimas y la documentación en la que permiten que las aplicaciones accedan al sistema no siempre estará protegida contra ataques de programas maliciosos (p. 91).

Chow (2018) explicó que los ataques informáticos han evolucionado desde los primeros virus que aparecieron hace muchos años, hasta el malware avanzado que ataca con diferentes vectores. Hoy en día existen múltiples problemas de cómo y cuándo mitigarlos en tiempo real. Esto convierte a una aplicación de microservicios y REST vulnerables. Además, Chow (2018) mencionó que los equipos están completamente desprotegidos contra este tipo de ataques y nuevo malware.

Netdata es un software gratuito que recopila datos de rendimiento en tiempo real de sistemas Linux, aplicaciones y dispositivos SNMP y los procesa en una interfaz basada en web (Duffield, 2020). Los usuarios pueden monitorear cualquier cosa con el complemento API y también integrar fácilmente los gráficos a cualquier página web externa. Tiene su propio servidor web para mostrar el informe final en formato gráfico (Duffield, 2020).

Khan (2016) mencionó que Apache JMeter se puede utilizar para probar el rendimiento en recursos estáticos y dinámicos (servicios web SOAP / REST), lenguajes web dinámicos (PHP, Java, ASP.NET, archivos, etc.), objetos Java, bases de datos y consultas, servidor FTP, etc.). Se puede usar para simular cargas pesadas en un servidor, un grupo de servidores, redes u objetos para probar la fuerza o para analizar el rendimiento general bajo diferentes tipos de carga. (Khan, 2016, p. 136)

Simbaña (2016) explicó que la herramienta Docker también permite portabilidad en las aplicaciones, además de crear contenedores. También, Simbaña (2016) mencionó que Docker brinda la facilidad de crear repositorios

en la nube en los que eventualmente serán alojados los contenedores y que también se puede descargar estos contenedores desde otro host o plataforma Cloud que tenga instalado Docker y desplegarlo sin ningún problema (p. 23). Además, Dagar y Gupta (2020) mencionaron que OWASP Zed Attack Proxy (ZAP) es una herramienta automatizada que se utiliza para explotar en base a los protocolos web como HTTP, TCP/IP, SSH, etc.; además, el resultado obtenido de las pruebas de vulnerabilidad es rápido y confiable en comparación con las pruebas manuales en las que el tiempo estimado es mayor (p. 46).

El monitor de red muestra todas las solicitudes de red que Firefox realiza (por ejemplo, cuando carga una página o debido a XMLHttpRequests), cuanto tiempo toma cada petición y los detalles de cada petición (Mozilla Foundation, 2019). Para abrir el monitor se selecciona o se abre Herramientas de desarrollador web y se seleccione la pestaña red (Mozilla Foundation, 2019).

Al-debagy y Martinek (2018) mencionaron que la arquitectura monolítica es una aplicación con una sola base de código que incluye múltiples servicios. Esos servicios se comunican con sistemas externos a través de diferentes interfaces como servicios web y API REST. También, Al-debagy y Martinek (2018) indicaron que a menudo se compara la arquitectura de microservicios con la arquitectura orientada a servicios (SOA) en términos de servicio y características de la arquitectura. Además, Wagh y Thool (2012) mencionaron que la mayoría de marcos de servicios web están basados en SOAP y REST y que estos servicios web a su vez están basados en conceptos de SOA, el que fue la evolución de la computación distribuida porque habilita los componentes de software para ser expuestos como servicios.

Además, Flygare y Holmqvist (2017) afirmaron que la arquitectura monolítica funciona como una sola unidad, donde comúnmente las aplicaciones se construyen en tres partes: una interfaz de usuario del cliente, una aplicación del lado del servidor y una base de datos. (p. 4). También, Corredor (2017) mencionó que REST se basa en dos agentes básicos: cliente y servidor que intercambian información y el servidor maneja la lógica del negocio; de esta manera, el cliente mantiene solo la lógica de diseño (p. 19). Además, Corredor (2017) afirmó que REST hace posible la comunicación entre el cliente y el

servidor mediante una API REST y que para ello el cliente solicita los recursos al servidor utilizando los métodos GET, POST, PUT y DELETE. (p. 18).

También, Tihomirovs y Grabis (2016) mencionaron que para la ejecución de un servicio web, según el estilo de arquitectura REST se realza una consulta a su base de datos y que estas consultas presentan menor tiempo que otros enfoques como SOAP (p. 94). Asimismo, Halili y Ramadani (2018) indicaron que REST se basa en una arquitectura orientada a servicios y se representa en una arquitectura cliente-servidor, donde el cliente envía las solicitudes, mientras el servidor las procesa y devuelve una respuesta. (p. 175). Finalmente, Mikuła y Dzieńkowski (2020) mencionaron que aplicación REST presenta la conexión a la base de datos en la capa del servidor. (p. 311)

Como se puede observar, las arquitecturas monolítica y REST tienen en común lo siguiente: cliente, servidor y acceso a bases de datos a través del servidor, por lo que se puede concluir que estas arquitecturas tienen varias características semejantes. Por ello, ante la ausencia de estudios comparativos entre las arquitecturas microservicios y REST se ha realizado las comparaciones de los resultados de las hipótesis de algunos de los indicadores de este estudio entre las arquitecturas monolítica y REST.

III. METODOLOGÍA

En esta sección se detalla el tipo, diseño y enfoque de investigación utilizados, las variables e indicadores utilizados y las fichas de recolección de datos utilizadas como instrumentos de investigación. También se trata acerca de los métodos de análisis de datos utilizados y finalmente, se detalla cómo se está cumpliendo con el código de ética de investigación de la Universidad César Vallejo y con el código de ética del Colegio de Ingenieros del Perú.

3.1 Tipo y diseño de investigación

La investigación del estudio fue de tipo aplicada, ya que tuvo como objetivo mostrar el nivel de rendimiento de ambas tecnologías y por ende se realizó el enriquecimiento de desarrollo cultural y científico para llegar a dicho resultado. Al respecto, Ñaupas, Palacios, Romero y Valdivia (2018) indicaron: “Son aplicadas porque se basan en los resultados de la investigación básica, pura o fundamental, de las ciencias naturales y sociales que hemos visto, se formulan problemas e hipótesis de trabajo para resolver los problemas de la vida social de la comunidad regional o del país” (p. 135).

La presente investigación tuvo un enfoque cuantitativo, ya que se realizó una investigación para explicar las tecnologías microservicos y REST mediante un estudio comparativo. Previamente, se realizó pruebas estadísticas para determinar un resultado concreto bajo los criterios de tiempo de respuesta, uso de recursos y nivel de seguridad. Ñaupas et al. (2018) mencionaron: “En este enfoque cuantitativo se utiliza la recolección y análisis de datos, sin preocuparse demasiado de su cuantificación; la observación y la descripción de los fenómenos se realizan, pero sin dar mucho énfasis a la medición” (p. 141).

El diseño del estudio fue no experimental, ya que se realizó un análisis por cada tecnología: microservicios y REST, con una comparación basada en la aplicación de una metodología que permitió medir su tiempo de respuesta, uso de recursos y nivel de seguridad. Al respecto, Radhakrishnan (2016) indicó que el diseño de investigación no experimental es una de las categorías de diseños de investigación, en la que el investigador observa los fenómenos a medida que ocurren de forma natural y no se introducen variables externas y

que es un diseño de investigación en el que las variables no se manipulan deliberadamente ni se controla el ajuste (p. 25).

Asimismo, su tipo de diseño fue transversal y descriptivo, ya que se realizó un análisis de ambas tecnologías: microservicios y REST, mediante una serie de pruebas y determinando un resultado de acuerdo a sus indicadores: tiempo de respuesta, uso de recursos, nivel de seguridad en un tiempo determinado. Al respecto, Carmona (2015) indicó: “La mayoría de los estudios descriptivos se clasifican como transversales ya que la medición del evento en estudio (respuesta, resultado o variable dependiente) es simultánea con la medición de las variables independientes o de exposición” (p. 40).

3.2 Variables y operacionalización

Las variables utilizadas fueron las siguientes: (a) uso de recurso de las tecnologías microservicios y REST, (b) tiempo de respuesta de las tecnologías microservicios y REST y (c) nivel de seguridad de las tecnologías microservicios y REST. Las dimensiones utilizadas fueron las siguientes: (a) uso de recursos, (b) tiempo de respuesta y (c) nivel de seguridad.

Los indicadores utilizados fueron los siguientes: (a) uso de memoria RAM, (b) uso de CPU, (c) tiempo de respuesta ante la carga de imágenes, (d) tiempo de respuesta ante la carga de usuarios, (e) ataques bloqueados y (f) cantidad de vulnerabilidades. La matriz de operacionalización de variables contendrá la transformación de las variables teóricas en dimensiones e indicadores y se encuentra en el “Anexo 2 Matriz de Operacionalización”.

3.3 Población, muestra y muestreo

La población estuvo compuesta por dos tecnologías: “Microservicios y REST”, se evaluaron los siguientes aspectos: tiempo de respuesta, uso de recursos y nivel de seguridad con el objetivo de analizar quien tiene mejor un mejor rendimiento. Ñaupas et al. (2018) mencionaron: “la población es la totalidad de los individuos o elementos en los que está presente una determinada característica a estudiar” (p. 35).

Sobre la cantidad de registros, Mikuła y Dzieńkowski (2020) indicaron que para las pruebas de carga se tiene un número diferente de usuarios que

generan solicitudes simultáneamente: 1, 5, 50, 100, 500 y 1000 solicitudes. Además, Nyman (2018) utilizó 8000 solicitudes para un ataque DoS (Denial of Service). En el caso de las pruebas de carga, estrés y penetración se está tomando como población las solicitudes durante una semana.

La muestra para la presente investigación fue constituida por las aplicaciones de pruebas de microservicios y REST: Aplicación MS1 y Aplicación MS2 y REST: Aplicación R1 y Aplicación R2. Estas aplicaciones fueron evaluadas en diferentes aspectos en base a su tiempo de respuesta, uso de recursos y nivel de seguridad para analizar su rendimiento. Ñaupas et al. (2018) explicaron que la muestra es una porción de la población que permite la generalización de los resultados ya que tiene las mismas características que toda la muestra (p. 46).

El muestreo de esta investigación fue no probabilístico por conveniencia. Esta técnica comúnmente usada consiste en seleccionar una muestra de la población por el hecho de que sea accesible; es decir, los individuos empleados en la investigación se seleccionan porque están fácilmente disponibles y porque sabemos que pertenecen a la población de interés, no porque hayan sido seleccionados mediante un criterio estadístico (Ñaupas et al., 2018). Esta conveniencia se suele traducir en una gran facilidad operativa y en bajos costos de muestreo y tiene como consecuencia la imposibilidad de hacer afirmaciones generales con rigor estadístico sobre la población (Ñaupas et al., 2018).

Además, la muestra tuvo 100 imágenes para el indicador tiempo de respuesta ante la carga de imágenes. La muestra tuvo 1,656 solicitudes por aplicación para el indicador tiempo de carga de usuarios. Luego, en el indicador cantidad de ataques bloqueados se contó con 8000 peticiones. Finalmente, para la cantidad de vulnerabilidades, la muestra tuvo 149 alertas de vulnerabilidades de la herramienta OWASP ZAP.

3.4 Técnicas e instrumentos de recolección de datos

En la presente investigación se utilizó la técnica de observación y la ficha de recolección de datos como instrumento. Ñaupas et al. (2018) mencionaron que las técnicas de investigación: “Son un conjunto de reglas y procedimientos para regular un determinado proceso y lograr un determinado objetivo” (p. 13).

Fue fundamental contar con un instrumento que registre la información obtenida de las cinco herramientas utilizadas: Netdata, DOCKER, OWASP Zap, Apache JMeter y Monitor de red. Las fichas utilizadas fueron: FORM01 Tiempo de respuesta carga de usuarios (ver anexo 6), FORM02 Cantidad de vulnerabilidades encontradas (ver anexo 6), FORM03 Porcentaje de ataques bloqueados (ver anexo 6), FORM04 Tiempo de respuesta carga de imágenes (ver anexo 6), FORM04 Uso de CPU (ver anexo 6) y FORM04 Uso de memoria RAM (ver anexo 6).

Para esta investigación se aplicó la validez de contenido bajo los criterios de tiempo de respuesta, uso de recursos y nivel de seguridad. Sobre la validez de contenido, Hernández, Fernández y Baptista (2016) mencionaron: “se refiere al grado en que un instrumento refleja un dominio específico de contenido de lo que se mide” (p. 201). Según Ñaupas et al. (2018): “La validez es la pertinencia de un instrumento de medición, para medir lo que se quiere medir; se refiere a la exactitud con que el instrumento mide lo que se propone medir” (p. 276).

En esta investigación no se aplicó la medición de confiabilidad, ya que el instrumento de recolección de datos que se utilizó fue la ficha de recolección de datos. Se tomó el 95 % del nivel de confianza para las pruebas estadísticas aplicadas. Ñaupas et al. (2018) mencionaron: “Un instrumento es confiable cuando las mediciones realizadas no varían significativamente, ni por el tiempo, ni por la aplicación de diferentes muestras” (p. 216).

3.5 Procedimientos

Para el procedimiento se utilizó la metodología METSA, la que consta de cuatro procesos: a) preparar el entorno de prueba, b) ejecutar la prueba de estrés, c) ejecutar la prueba de penetración y d) ejecutar la prueba de carga, con el

propósito de evaluar el rendimiento de las arquitecturas microservicios y REST según los indicadores: tiempo de respuesta, uso de recursos y nivel de seguridad. Asimismo, en la sección anexos se encuentra la metodología con los procesos a detalle (anexo 4). Los procedimientos fueron los siguientes:

a) Preparar el entorno de prueba (ver el proceso 01 de la metodología METSA).
b) Ejecutar la prueba de estrés (ver el proceso 02 de la metodología METSA).
c) Ejecutar la prueba de penetración (ver el proceso 03 de la metodología METSA).
d) Ejecutar la prueba de carga (ver el proceso 04 de la metodología METSA).

3.6 Método de análisis de datos

Se utilizó la prueba de Kolmogorov-Smirnov para determinar si las distribuciones de las muestras fueron normales o no normales y también la prueba de Wilcoxon para realizar las pruebas de hipótesis. En la estadística descriptiva se utilizó tablas descriptivas y gráficos de barras para comparar el pre y post test**.** Además, Cruz, Olivares y González (2014) explicó que la prueba de Wilcoxon de rangos con signo tiene en cuenta la información sobre el signo de las diferencias y la magnitud de las diferencias entre los pares (p. 202).

Romero (2016) indicó que la prueba de Kolmogorov-Smirnov se usa cuando el tamaño de la muestra es mayor a 50. Asimismo, Romero (2016) mencionó que la prueba de Shapiro-Wilk se utiliza en otros casos en los que la muestra el tamaño es menor o igual a 50.

3.7 Aspectos éticos

De acuerdo a la información y datos obtenidos de libros, artículos científicos y otros documentos que han servido como fuente para esta investigación, se otorgó crédito a los autores a través de referencias bibliográficas, paráfrasis y citas textuales en base a la norma internacional ISO 690:2010.

Esta investigación presenta originalidad y brinda mediante citas las referencias utilizadas, lo que cumple con lo estipulado en el artículo 9 del Código de Ética de Investigación 2020 de la Universidad César Vallejo, el que hace referencia a la política anti plagio de las investigaciones. Además, se presenta con transparencia los resultados, pruebas y datos obtenidos de acuerdo al artículo 3 que menciona entre los principios de ética: probidad, transparencia, no maleficencia, respeto de la propiedad privada y responsabilidad.

Además, los procedimientos realizados para la metodología, pruebas y resultados se ha respetado las normas o disposiciones de la universidad, en cumplimiento del artículo 18 del Código de Ética del Colegio de Ingenieros del Perú que menciona respetar las leyes, ordenanzas y disposiciones vigentes y actuar con principios de honradez y moralidad. Asimismo, con esta investigación se contribuye al campo profesional a través de los resultados obtenidos al comparar las dos tecnologías, de acuerdo a lo indicado en el artículo 15 del Código de Ética del CIP que precisa los criterios y conceptos de la conducta que debe seguir el profesional.

IV. RESULTADOS

En este capítulo se muestra los resultados obtenidos de la investigación sobre la base de los indicadores de rendimiento de las tecnologías microservicios y REST, tales como: uso de memoria RAM, uso de CPU, tiempo de respuesta de carga de imágenes (10 KB, 100 KB, 1 MB, 10 MB y 100 MB), tiempo de respuesta de carga de usuarios (1, 5, 10, 50, 100, 500 y 1000), ataques bloqueados y cantidad de vulnerabilidades. Asimismo, se muestra el procesamiento de los datos obtenidos al aplicar la metodología METSA. Los datos fueron procesados a través del software estadístico SPSS.

4.1 Datos descriptivos

4.1.1 Tiempo de respuesta de carga de imágenes

El resultado obtenido para el indicador tiempo de respuesta de carga de imágenes figura en la tabla 1. Se realizó la misma prueba para microservicios y REST.

Tabla 1 Estadísticos descriptivos - Tiempo de respuesta de carga de imágenes

Descriptivo		
		Estadístico
Tiempo Carga-Microservicios	Media	43.649224999999994
	Desviación estándar	65.952614781119240
Tiempo Carga-REST	Media	48,983995000000010
	Desviación estándar	79.058872559675490

La distribución de datos del indicador de tiempo de respuesta de carga de imágenes (10KB, 100KB, 1MB, 10MB y 100MB) de las tecnologías Microservicios y REST muestran una dispersión donde se refleja el valor promedio (media) y la forma en que se acercan o alejan del conjunto de datos (desviación estándar). En el caso de las tecnologías REST, su promedio fue mayor que el promedio de las tecnologías de microservicios.

Prueba de normalidad

Para el caso del indicador tiempo de respuesta de carga de imágenes de las tecnologías microservicios y REST se aplicó la prueba de Kolmogorov-Smirnov, ya que la cantidad de datos ingresados fue mayor a 50. Los resultados de la prueba están en la tabla 2.

Tabla 2 Prueba normalidad - Tiempo de respuesta de carga de imágenes

Pruebas de normalidad						
	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Estadístico	gl	Sig.	Estadístico	gl	Sig.
Tiempo Carga-Microservicios	0.318105	200.000000	2.5856E-57	0,682609	200.000000	3.4747E-19
Tiempo Carga-REST	0.323824	200.000000	1.786E-59	0.660235	200.000000	8.3947E-20

De acuerdo a los resultados mostrados, los valores de significancia son menor 0.05; es decir que los datos no siguen una distribución normal.

4.1.2 Tiempo de respuesta de carga de usuarios

El resultado obtenido para el indicador tiempo de respuesta de carga usuarios se muestra en la tabla 3. Para esta prueba se realizó la simulación de carga de usuarios para microservicios y REST.

Tabla 3 Estadísticos descriptivos - Tiempo de respuesta de carga de usuarios

Descriptivo		
		Estadístico
Carga Usuarios-Microservicios	Media	7979.275000
	Desviación estándar	12426.4697011
Carga Usuarios-REST	Media	10362.514792
	Desviación estándar	14638.8337460

La distribución de datos del indicador de tiempo de respuesta de carga de usuarios (1, 5, 50, 100, 500 y 1000) de las tecnologías microservicios y REST muestran una dispersión. En el caso de la tecnología REST, su promedio fue mayor al promedio de la tecnología microservicios.

Prueba de normalidad

Para el caso del tiempo de respuesta de carga de imágenes de las tecnologías Microservicios y REST se aplicó la prueba de Shapiro-Wilk, ya que la cantidad de datos ingresados fue menor a 50. Los resultados de la prueba de normalidad están en la tabla 4.

Tabla 4 Prueba normalidad - Tiempo de respuesta de carga de usuarios

Pruebas de normalidad						
	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Estadístico	gl	Sig.	Estadístico	gl	Sig.
Carga Usuarios-Microservicios	0.314470	48.000000	4.4175E-13	0.605898	48.000000	4.0429E-10
Carga Usuarios-REST	0.325175	48.000000	4.8472E-14	0.716631	48.000000	2.6423E-8

4.1.3 Uso de CPU

El resultado obtenido para el indicador Uso de CPU se muestra en la tabla 5. Para esta prueba se midió el uso de CPU de cada prueba realizada tanto en microservicios como REST.

Tabla 5 Estadístico descriptivo - Uso de CPU

Descriptivo		
		Estadístico
Uso CPU-REST	Media	0.407333333333333
	Desviación estándar	0.218864449051973
Uso CPU-Microservicios	Media	0.441222222222222
	Desviación estándar	0.199252590375237

Las distribuciones de los datos del indicador de Uso de CPU de las tecnologías Microservicios y REST muestran una dispersión donde reflejan el valor promedio (media). En el caso la tecnología REST su promedio es menor al promedio de la tecnología Microservicios.

Prueba de normalidad

Para el caso de tiempo de respuesta de carga de imágenes de las tecnologías Microservicios y REST se aplicó la prueba de Shapiro-Wilk, ya que la cantidad de datos ingresados fue menor a 50. Los resultados de la prueba de normalidad están en la tabla 6.

Tabla 6 Prueba normalidad - Uso de CPU

Pruebas de normalidad						
	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Estadístico	gl	Sig.	Estadístico	gl	Sig.
Uso CPU-REST	0.150637	18.000000	0.200000	0.930414	18,000000	0.197474
Uso CPU-Microservicios	0.157041	18.000000	0.200000	0.950101	18,000000	0.426599

De acuerdo a los resultados mostrados, los valores de significancia que corresponden que se muestran 0.427 y 0.197 son mayores a 0.05; es decir, los datos siguen una distribución normal.

4.1.4 Uso de memoria RAM

El resultado obtenido para el indicador Uso de memoria RAM se muestra en la tabla 7. Para esta prueba se midió el uso memoria RAM de cada prueba realizada tanto en microservicios como REST.

Tabla 7 Estadísticos descriptivos – Uso de memoria RAM

Descriptivo		
		Estadístico
Uso RAM-Microservicios	Media	0.366111111111111
	Desviación estándar	0.474752429463026
Uso RAM-REST	Media	0.506666666666667
	Desviación estándar	0.750325419597469

Las distribuciones de los datos del indicador de Uso de Memoria RAM de las tecnologías microservicios y REST muestran una dispersión donde se refleja el valor promedio (media). En el caso la tecnología REST su promedio es mayor al promedio de la tecnología microservicios.

Prueba de normalidad

Para el caso del tiempo de respuesta de carga de imágenes de las tecnologías microservicios y REST se aplicó la prueba de Shapiro-Wilk, ya que la cantidad de datos ingresados fue menor a 50. Los resultados de la prueba de normalidad están en la tabla 8.

Tabla 8 Prueba normalidad - Uso de memoria RAM

Pruebas de normalidad						
	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Estadístico	gl	Sig.	Estadístico	gl	Sig.
Uso RAM-Microservicios	0.266108	18.000000	0.001528	0.707589	18.000000	0.000098
Uso RAM-REST	0.350043	18.000000	0.000003	0.670942	18.000000	0.000037

De acuerdo a los resultados mostrados, los valores de significancia que corresponden es 0.000 menor a 0.05; es decir los datos no siguen una distribución normal.

4.1.5 Cantidad de Vulnerabilidad

El resultado obtenido para el indicador cantidad de vulnerabilidades se muestra en la tabla 9. La prueba fue realizada con el programa OWASP ZAP para microservicios y REST.

Tabla 9 Estadísticos descriptivos - Cantidad de vulnerabilidades

Descriptivo		
		Estadístico
Cantidad Vulnerabilidad-Microservicios	Media	0.75
	Desviación estándar	0.847
Cantidad Vulnerabilidad-REST	Media	1.00
	Desviación estándar	1.022

Las distribuciones de datos del indicador de cantidad de vulnerabilidades de las tecnologías Microservicios y REST muestran una dispersión donde se refleja el valor promedio (media). En el caso la tecnología Microservicios su promedio fue menor al promedio de la tecnología REST.

Prueba de Normalidad

Para el caso de tiempo de respuesta de carga de imágenes de las tecnologías microservicios y REST se aplicó la prueba de Shapiro-Wilk, ya que la cantidad de datos ingresados fue menor a 50. Los resultados de la prueba están en la tabla 10.

Tabla 10 Prueba normalidad - Cantidad de vulnerabilidades

Pruebas de normalidad						
	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Estadístico	gl	Sig.	Estadístico	gl	Sig.
Cantidad Vulnerabilidad-Microservicios	0.312054	24.000000	0.000002	0.751143	24.000000	0.000052
Cantidad Vulnerabilidad-REST	0.336196	24.000000	1.285E-7	0.639538	24.000000	0.000002

De acuerdo a los resultados mostrados, los valores de significancia son menor a 0.05; es decir los datos no siguen una distribución normal.

4.1.5 Ataques bloqueados

El resultado obtenido para el indicador ataques bloqueados se muestra en la tabla 11. La prueba fue realizada con el programa JMeter para microservicios y REST.

Tabla 11 Estadísticos descriptivos - Ataques bloqueados

Descriptivo		
		Estadístico
AtaquesBloqueados-Microservicios	Media	26.508333
	Desviación estándar	0.8312140
AtaquesBloqueados-REST	Media	37.393750
	Desviación estándar	6.0086279

Las distribuciones de datos del indicador de ataques bloqueados de las tecnologías Microservicios y REST muestran una dispersión donde se refleja el valor promedio (media). En el caso la tecnología Microservicios su promedio fue menor que el promedio de la tecnología REST.

Prueba de normalidad

Para el caso de tiempo de respuesta carga de imágenes de las tecnologías Microservicios y REST se aplicó la prueba de la prueba de Shapiro-Wilk, ya ue la cantidad de datos ingresados es menor a 50 datos. Los resultados de la prueba de normalidad se muestran en la tabla 12.

Tabla 12 Prueba normalidad - Ataques bloqueados

Pruebas de normalidad						
	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Estadístico	gl	Sig.	Estadístico	gl	Sig.
AtaquesBloqueados-Microservicios	0.305711	6.000000	0,083340	0,867283	6.000000	0,215588
AtaquesBloqueados-REST	0.259992	6.000000	0,200000	0,833089	6.000000	0,114144

De acuerdo a los resultados mostrados, los valores de significancia que corresponden es 0.144 y 0.216 son mayor a 0.05; es decir los datos siguen una distribución normal.

4.2 Pruebas de hipótesis

La siguiente sección contiene las pruebas realizadas para cada indicador y el resultado de las hipótesis.

4.2.1 Hipótesis específica 1

HE1o: La tecnología Microservicios no tiene menor uso de CPU que la tecnología REST

HE1a: La tecnología Microservicios tiene menor uso de CPU que la tecnología REST.

Prueba T

Se utilizó la prueba T porque los registros eran menores que 30. Los resultados obtenidos para la hipótesis específica 1 se muestran en la tabla 13.

Tabla 13 Prueba T - Uso de CPU

Prueba de muestras emparejadas								
	Diferencias emparejadas					t	gl	Sig. (bilateral)
	Media	Desviación estándar	Media de error estándar	95% de intervalo de confianza de la diferencia				
				Inferior	Superior			
UsoCPU-REST- UsoCPU- Microservicios	-0.033889	0.095511	0.022512	-0.081385	0.013608	-1.505360	17	0.150585

Con los datos obtenidos, el nivel de significancia fue de 0.1520161 mayor a 0.05. Por lo cual se acepta la hipótesis nula y se rechaza la hipótesis alternativa. No se puede afirmar que la tecnología microservicios tiene menor uso de CPU que la tecnología REST.

4.2.2 Hipótesis específica 2

HE2o: La tecnología microservicios no tiene un menor uso de memoria RAM que tecnología REST.

HE2a: La tecnología microservicios tiene un menor uso de memoria RAM que la tecnología REST.

Prueba de Wilcoxon

Debido a que contiene más de 30 registros, se realizó la prueba de wilcoxon, los resultados obtenidos para la hipótesis específica 2 se muestran en la tabla 14.

Tabla 14 Prueba de Wilcoxon - Uso de memoria RAM

Rangos				
		N	Rango promedio	Suma de rangos
UsoRAM-REST UsoRAM-Microservicios	Rangos negativos	6[a]	7.333333	44.00000
	Rangos positivos	11[b]	9.909091	109.000000
	Empates	1[c]		
	Total	18		

En la Tabla 15 se muestran los estadísticos de prueba para la hipótesis específica 2. En esta tabla se encuentra valor de significancia.

Tabla 15 Prueba de hipótesis - Uso de memoria RAM

Estadísticos de prueba[a]	
	UsoRAM-REST - UsoRAM-Microservicios
Z	-1.543686
Sig. asin. (bilateral)	0.122664

Con los datos obtenidos de la prueba de wilcoxon, el nivel de significancia fue de 0.122664 mayor a 0.05. Por la cual se acepta la hipótesis nula y se rechaza la hipótesis alternativa. No se puede afirmar que la tecnología microservicios tiene menor uso de memoria RAM que la tecnología REST.

4.2.3 Hipótesis específica 3

HE3o: La tecnología de microservicios no tiene menor tiempo de respuesta de carga de imágenes que la tecnología REST.

HE3a: La tecnología de microservicios tiene menor tiempo de respuesta de carga de imágenes que la tecnología REST.

Prueba de Wilcoxon

Los resultados obtenidos en la prueba de wilcoxon para la hipótesis específica 3 se muestran en la tabla 16.

Tabla 16 Prueba de Wilcoxon - Tiempo de respuesta de carga de imágenes

Rangos				
		N	Rango promedio	Suma de rangos
TiempoCargaREST – TiempoCargaMicroservicios	Rangos negativos	113[a]	96.880531	10947.500000
	Rangos positivos	80[b]	97.168750	7773.500000
	Empates	7[c]		
	Total	200		

En la Tabla 17 se muestran los estadísticos de prueba para la hipótesis específica 3. En esta tabla se encuentra valor de significancia.

Tabla 17 Prueba de hipótesis - Tiempo de respuesta de carga de imágenes

Estadísticos de prueba[a]	
	TiempoCargaREST - TiempoCargaMicroservicios
Z	-2.042551
Sig. asin. (bilateral)	0.041097

Con los datos obtenidos de la prueba de wilcoxon, el nivel de significancia fue de 0.041097 menor a 0.05. Por lo cual se rechaza la hipótesis nula aceptando la hipótesis alterna con un 95% de nivel de confianza, donde se indica que La tecnología de microservicios tiene menor tiempo de respuesta de carga de imágenes que la tecnología REST.

4.2.4 Hipótesis específica 4

Ho: La tecnología de microservicios no tiene menor tiempo de respuesta de carga de usuarios que la tecnología REST.

Ha: La tecnología de microservicios tiene menor tiempo de respuesta de carga de usuarios que la tecnología REST.

Prueba de Wilcoxon

Debido a que contiene más de 30 registros, se realizó la prueba de wilcoxon, los resultados obtenidos en para la hipótesis específica 4 se muestran en la tabla 18.

Tabla 18 Prueba Wilxocon - Tiempo de respuesta de carga de usuarios

Rangos				
		N	Rango promedio	Suma de rangos
CargaUsuarios-REST - CargaUsuarios-Microservicios	Rangos negativos	19[a]	17.315789	329.000000
	Rangos positivos	25[b]	26.440000	661.000000
	Empates	4[c]		
	Total	48		

En la Tabla 19 se muestran los estadísticos de prueba para la hipótesis específica 4. En esta tabla se encuentra valor de significancia.

Tabla 19 Prueba de hipótesis - Tiempo de respuesta de carga de usuarios

Estadísticos de prueba[a]	
	CargaUsuarios-REST - CargaUsuarios-Microservicios
Z	-1.937252
Sig. asin. (bilateral)	0.052715

Con los datos obtenidos de la prueba de wilcoxon, el nivel de significancia fue de 0.052715 mayor a 0.05. Se rechaza la hipótesis alternativa. No se puede afirmar que la tecnología de microservicios tiene menor tiempo de respuesta de carga usuarios que la tecnología REST.

4.2.5 Hipótesis específicas 5

Ho: La tecnología microservicios no tiene menor cantidad de vulnerabilidades que la tecnología REST.

Ha: La tecnología microservicios tiene menor cantidad de vulnerabilidades que la tecnología REST.

Prueba de Wilcoxon

Debido a que contiene más de 30 registros, se realizó la prueba de Wilcoxon, los resultados obtenidos en para la hipótesis específica 5 se muestran en la tabla 20.

Tabla 20 Prueba de Wilcoxon - Cantidad de vulnerabilidades

Rangos				
		N	Rango promedio	Suma de rangos
CantidadVulnerabilidad-REST - CantidadVulnerabilidad-Microservicios	Rangos negativos	0[a]	0.0E0	0.0E0
	Rangos positivos	6[b]	3.500000	21.000000
	Empates	18[c]		
	Total	24		

En la Tabla 21 se muestran los estadísticos de prueba para la hipótesis específica 5. En esta tabla se encuentra valor de significancia.

Tabla 21 Prueba de hipótesis - Cantidad de vulnerabilidades

Estadísticos de prueba[a]	
	CantidadVulnerabilidad-REST - CantidadVulnerabilidad-Microservicios
Z	-2.449490
Sig. asin. (bilateral)	0.014306

Con los datos obtenidos de la prueba de wilcoxon, el nivel de significancia fue de 0.014306 menor a 0.05. Por rechaza la hipótesis nula y se acepta la hipótesis alternativa que señala que La tecnología microservicios tiene menor cantidad de vulnerabilidades que la tecnología REST.

4.2.6 Hipótesis específicas 6

Ho: La tecnología REST no tiene mayor porcentaje de ataques bloqueados que la tecnología microservicios.

Ha: La tecnología REST tiene mayor porcentaje de ataques bloqueados que la tecnología microservicios.

Prueba T

Se utilizó la prueba T porque los registros eran menores que 30. Los resultados obtenidos para la hipótesis específica 6 se muestran en la tabla 22.

Tabla 22 Prueba T - Ataques bloqueados

Prueba de muestras emparejadas								
	Diferencias emparejadas					t	gl	Sig. (bilateral)
	Media	Desviación estándar	Media de error estándar	95% de intervalo de confianza de la diferencia				
				Inferior	Superior			
AtaquesBloqueados-REST AtaquesBloqueados-Microservicios	10.8854167	5.7304745	2.3394564	4.8716524	16.899180	4.652968	5	0.005567

Con los datos obtenidos, el nivel de significancia fue 0.005567 menor a 0.05 Por lo tanto, se rechaza la hipótesis nula y se acepta la hipótesis alternativa que indica que la tecnología REST tiene mayor porcentaje de ataques bloqueados que la tecnología microservicios.

4.3 Resumen

Se plantearon seis hipótesis específicas, una por cada indicador. El resumen de las aceptaciones o rechazos de cada hipótesis está en la tabla 23.

Tabla 23 Resumen de resultados de las pruebas de hipótesis

Hipótesis específicas	Resultados	Condición
HE1o: La tecnología Microservicios no tiene menor uso de CPU que la tecnología REST. HE1a: La tecnología Microservicios tiene menor uso de CPU que la tecnología REST.	Con los datos obtenidos, el nivel de significancia fue de 0.1520161 mayor a 0.05. Por lo cual se acepta la hipótesis nula y se rechaza la hipótesis alternativa. No se puede afirmar que la tecnología microservicios tiene menor uso de CPU que la tecnología REST	Rechazada
HE2o: La tecnología Microservicios no tiene un menor uso de memoria RAM que tecnología REST. HE2a: La tecnología Microservicios tiene un menor uso de memoria RAM que la tecnología REST.	Con los datos obtenidos de la prueba de wilcoxon, el nivel de significancia fue de 0.122664 mayor a 0.05. Por la cual se acepta la hipótesis nula y se rechaza la hipótesis alternativa. No se puede afirmar que la tecnología microservicios tiene menor uso de memoria RAM que la tecnología REST a pesar que la diferencia de promedios es de 0.1406 MB, lo que significa que el uso de memoria RAM de microservicios es 27.74% menor que REST.	Rechazada

Hipótesis específicas	Resultados	Condición
HE3o: La tecnología de Microservicios no tiene menor tiempo de respuesta de carga de imágenes que la tecnología REST. HE3a: La tecnología de Microservicios tiene menor tiempo de respuesta de carga de imágenes que la tecnología REST.	Con los datos obtenidos de la prueba de wilcoxon, el nivel de significancia fue de 0.041097 menor a 0.05. Por lo cual se rechaza la hipótesis nula aceptando la hipótesis alterna con un 95% de nivel de confianza, donde se indica que La tecnología de microservicios tiene menor tiempo de respuesta de carga de imágenes que la tecnología REST. Con una diferencia de 1066.9540 segundos, se afirma que microservicios presenta un tiempo de respuesta menor de 10.89% respecto a REST.	Aceptada
HE4o: La tecnología de Microservicios no tiene menor tiempo de respuesta de carga de usuarios que la tecnología REST. HE4a: La tecnología de Microservicios tiene menor tiempo de respuesta de carga de usuarios que la tecnología REST.	Con los datos obtenidos de la prueba de wilcoxon, el nivel de significancia fue de 0.052715 mayor a 0.05. Se rechaza la hipótesis alternativa. No se puede afirmar que la tecnología de microservicios tiene menor tiempo de respuesta de carga usuarios que la tecnología REST.	Rechazada
HE5o: La tecnología Microservicios no tiene menor cantidad de vulnerabilidades que la tecnología REST. HE5a: La tecnología Microservicios tiene menor cantidad de vulnerabilidades que la tecnología REST.	Con los datos obtenidos de la prueba de wilcoxon, el nivel de significancia fue de 0.014306 menor a 0.05. Por rechaza la hipótesis nula y se acepta la hipótesis alternativa que señala que La tecnología microservicios tiene menor cantidad de vulnerabilidades que la tecnología REST. Se afirma que microservicios presenta un 25% menos cantidad de vulnerabilidades respecto a REST.	Aceptada
HE6o: La tecnología REST no tiene mayor porcentaje de ataques bloqueados que la tecnología microservicios. HE6a: La tecnología REST tiene mayor porcentaje de ataques bloqueados que la tecnología microservicios.	Con los datos obtenidos, el nivel de significancia fue 0.005567 menor a 0.05 Por lo tanto, se rechaza la hipótesis nula y se acepta la hipótesis alternativa que indica que la tecnología REST tiene mayor porcentaje de ataques bloqueados que la tecnología microservicios. Se afirma que REST presenta una diferencia de 10.8854% más ataques bloqueados que microservicios	Aceptada

V. DISCUSIÓN

En esta sección se discute acerca de los resultados obtenidos en la presente investigación referida a la elaboración del estudio comparativo cuantitativo de las tecnologías microservicios y REST para determinar cuál tiene el mejor rendimiento en los aspectos de tiempo de respuesta, uso de recursos y nivel de seguridad. Además, debido a que no se encontraron comparaciones entre las arquitecturas microservicios y REST en algunos indicadores asociados a las comprobaciones de las hipótesis, se está haciendo las comparaciones de semejanzas y diferencias entre la arquitectura microservicios y la arquitectura monolítica, ya que conforme se indicó en el marco teórico existen varias características semejantes entre las arquitecturas monolítica y REST. A continuación, se presenta las discusiones sobre los indicadores comparándolos con los resultados de estudios previos.

El indicador “tiempo de respuesta de carga de imágenes” fue evaluado por distintas imágenes de diferentes tamaños (10 KB, 100 KB, 1 MB, 10 MB y 100 MB). El resultado estadístico demostró que las tecnologías de microservicios tuvieron menor tiempo de respuesta de carga de imágenes que las tecnologías REST. Estos resultados fueron semejantes a los presentados por Richards (2016), quien indicó que los sistemas construidos utilizando la Arquitectura Orientada a Servicios (Service Oriented Architecture: SOA) son más lentos que los realizados utilizando la arquitectura de microservicios debido a que SOA generalmente se basa en varios servicios (y tipos de servicios) para completar una sola solicitud.

También, los resultados de la investigación con respecto al indicador tiempo de respuesta fueron semejantes a los presentados por Villamizar, Garcés, Castro, Verano, Salamanca y Casallas (2015), quienes indicaron que el tiempo de respuesta de microservicios es ligeramente menor a una arquitectura monolítica; pero, no determinante, debido a que se utilizó una sola instancia para el despliegue de la aplicación monolítica y tres instancias para microservicios (una para cada microservicio y una puerta de enlace).

El indicador “tiempo de respuesta de carga de usuarios” fue evaluado con las cantidades siguientes: 5, 10, 100, 500 y 1000 usuarios. Como resultado de la evaluación se concluyó que no se puede afirmar que la tecnología

microservicios tiene menor tiempo de respuesta de carga de usuarios que la tecnología REST. Este resultado es similar al presentado por Al-Debagy y Martinek (2018), quienes indicaron que los microservicios y la aplicación monolítica pueden tener un rendimiento similar bajo carga normal en la aplicación. En una carga pequeña con menos de 100 usuarios el tiempo de respuesta es menor en una aplicación monolítica; sin embargo, el promedio es similar al aumentar la cantidad de solicitudes. Esto ocurre porque el aumento del número de usuarios incrementa el tiempo de respuesta de las solicitudes, lo que lleva a la disminución del rendimiento general.

Con respecto al indicador uso de memoria RAM, con los resultados de esta investigación no se puede afirmar que la tecnología microservicios tiene menor uso de memoria RAM que la tecnología REST. Estos resultados son semejantes a los presentados por Flygare y Holmqvist (2017), quienes mencionaron que la arquitectura microservicios tiene un uso menor de memoria RAM que una arquitectura monolítica en parte porque las especificaciones y el sistema operativo utilizado fueron distintos a los presentados en esta investigación. Además, Flygare y Holmqvist (2017) concluyeron que el nivel de uso no se ve afectado por el uso de contenedores como Docker.

Con respecto al indicador uso de memoria RAM, los resultados de esta investigación fueron semejantes a los resultados de Tapia, Mora, Fuentes, Aules, Flores y Toulkeridis (2020), quienes indicaron que el uso promedio de memoria RAM es menor en 1.63 MB con los microservicios en comparación con la propuesta monolítica y se concluyó que la opción monolítica consume más memoria RAM porque la tecnología de microservicios realiza la transmisión o recepción de paquetes de manera más eficiente que la arquitectura monolítica. Cabe resaltar que en esta investigación se utilizó una aplicación funcionalmente idéntica para comparar las tecnologías microservicios y REST.

Con respecto al indicador uso de CPU se concluyó que no se puede afirmar que la tecnología microservicios tiene menor uso de CPU que la tecnología REST. Este resultado es distinto a los presentados por Flygare y Holmqvist (2017), quienes mencionaron que la tecnología microservicios usa

menos CPU que una tecnología diferente como la monolítica. Esta diferencia se debe a que utilizaron especificaciones distintas (procesador Intel Core i7-5557U, 16GB RAM y sistema operativo Ubuntu Server) a las utilizadas en la presente investigación. Además, Flygare et al. (2017) añadieron que el uso de contenedores no disminuye el rendimiento de estos indicadores.

Sin embargo, los resultados de la evaluación del indicador uso de memoria RAM de esta investigación fueron semejantes a los resultados de Tapia et al. (2020), quienes indicaron que el uso de CPU (%) en microservicios es 5.4 unidades más alto que en el escenario monolítico. Asimismo, Tapia et al. (2020) mencionaron que existe una diferencia insignificante a favor del escenario monolítico, el que genera un menor uso de este recurso. Este resultado se debe a que la aplicación de microservicios procesa un número más significativo de solicitudes por segundo y al tener cada servicio su propia base de datos y complejidad para administrarlos se afecta el uso de CPU.

Asimismo, los resultados de la evaluación del indicador uso de memoria RAM fueron semejantes a los resultados de Saransig (2018), quien mencionó que una aplicación de microservicios en promedio consume un 67.73% frente al 55.35% de una aplicación con estructura monolítica, dando una diferencia de 12.38% a favor de la segunda tecnología. También, Saransig (2018) mencionó que el mayor uso de CPU en microservicios se debe a un menor tiempo de respuesta, como en el caso específico de los microservicios con contenedores donde generalmente se manejan sistemas informáticos de gran escala, alto flujo y alto rendimiento de información.

Con respecto al indicador cantidad de vulnerabilidades, los resultados evidenciaron que la tecnología microservicios tiene menor cantidad de vulnerabilidades que la tecnología REST. Estos resultados fueron diferentes a los resultados de Guardiola (2020), quien mencionó que se encuentra un número mayor de vulnerabilidades al momento de analizar la seguridad en microservicios: (a) Cross-Site Scripting (XSS), (b) denegación de Servicios, (c) políticas de red escasa de Kubernetes, (d) servicios internos sin proteger con la autenticación de Ingress en Kubernetes, (e) acceso sin autorización al *etcd* de Kubernetes, (f) Service accounts con demasiados privilegios, (g) contenedores con privilegios root, (h) configuración local insegura de Docker, (i) control de

acceso débil de Docker y (j) distribución de imágenes de Docker (p. 23). Asimismo, estos resultados difieren con los resultados de Corredor (2017), quien encontró en una aplicación REST una cantidad menor de vulnerabilidades en lo referente a: (a) pérdida de autenticación, (b) administración de sesión, (c) configuración errónea de seguridad, (d) exposición de datos sensibles, (e) falta de control de acceso de nivel de funciones, (f) redireccionamiento y (g) reenvío no validado.

Con respecto al indicador porcentaje de ataques bloqueados, los resultados de esta investigación revelaron que la tecnología REST tiene mayor porcentaje de ataques bloqueados que la tecnología microservicios. Estos resultados fueron similares a los resultados de Ibrahim, Bozhinoski y Pretschner (2019), quienes indicaron que los contenedores y las arquitecturas de microservicios han demostrado serios riesgos de seguridad, principalmente debido a sus requisitos de conectividad y menor grado de encapsulación; pero, pueden lograr una mejor escalabilidad e implementación.

Finalmente, con respecto al indicador porcentaje de ataques bloqueados, los resultados de esta investigación fueron semejantes a los resultados de Mateus, Cruz y Gonzaga (2020), quienes indicaron que la seguridad de los microservicios presenta dificultades para su implementación por parte de los equipos de desarrollo debido a su estructura modular. Además, Mateus, Cruz y Gonzaga (2020) señalaron que los servicios están expuestos a posibles ataques en la arquitectura microservicios, debido a que los servicios deben comunicarse entre sí de manera implícita.

VI. CONCLUSIONES

En el contexto de la muestra de tecnologías de arquitecturas utilizadas, las conclusiones de la investigación fueron las siguientes:

1. Las tecnologías de microservicios tuvieron menor tiempo de respuesta de carga de imágenes con distintos tamaños (10 KB, 100 KB, 1 MB, 10 MB y 100MB) que las tecnologías REST con una diferencia de 1066.9540 segundos. Esto indicó que las tecnologías de microservicios presentaron un tiempo de respuesta menor en 10.89% respecto a REST.
2. Con respecto a la medición del indicador tiempo de respuesta de carga de usuarios para las cantidades de 1, 5, 50, 100, 500 y 1000, no se pudo afirmar que las tecnologías de microservicios tuvieron menor tiempo de respuesta que las tecnologías REST.
3. No se pudo afirmar que las tecnologías de microservicios tuvieron menor uso de memoria RAM que las tecnologías REST, a pesar que la diferencia de promedios fue 0.1406 MB, lo que significó que el uso de memoria RAM de microservicios fue 27.74% menor que REST.
4. Respecto al indicador uso de CPU no se pudo afirmar que las tecnologías de microservicios tuvieron menor uso de CPU que las tecnologías REST, aunque la diferencia promedio entre los usos de CPU de ambas tecnologías fue 3.38% a favor de REST.
5. Se obtuvo un promedio favorable para las tecnologías de microservicios frente a las tecnologías REST al realizar la medición del indicador cantidad de vulnerabilidades con una diferencia 25% menor en la cantidad de vulnerabilidades en las tecnologías microservicios respecto a las tecnologías REST.
6. Con respecto al indicador de ataques bloqueados, las tecnologías REST bloquearon 10.8854% más ataques que las tecnologías de microservicios.
7. Las tecnologías de microservicios resultaron ligeramente superiores que las tecnologías REST, en cuanto a los indicadores: tiempo de respuesta de carga de imágenes y cantidad de vulnerabilidades, a diferencia de las tecnologías REST que presentaron un mejor desempeño solo en el indicador ataques bloqueados.

8. La aplicación MS1 de microservicios tuvo un promedio de tiempo de respuesta menor en 13.5771% que R1 de REST y MS2 de microservicios presentó un tiempo de respuesta promedio menor en 34.6916% que REST.
9. El uso de uso de memora RAM fueron favorables para las tecnologías de microservicios, logrando que la aplicación MS1 obtuviera un promedio de 31.8878% menos uso de memoria RAM que R1 y MS2 mostró un promedio de 31.0680% menor uso de memoria RAM que R2 de REST.
10. De acuerdo con los resultados obtenidos en las pruebas de cantidad de vulnerabilidades, se encontró diferencias entre las aplicaciones REST R1 y R2 y las aplicaciones de microservicios MS1 y MS2, donde las aplicaciones MS2 y MS1 tuvieron una 25% menos cantidad de vulnerabilidades (9 cada una) respecto a REST (12 cada una).

VII. RECOMENDACIONES

Las recomendaciones para futuras investigaciones fueron las siguientes:

1. Ampliar la cantidad de indicadores utilizados para las comparaciones entre microservicios y REST, tales como: balance de carga y uso de red, utilizando la tecnología Eureka para el balanceo de carga en microservicios y NetData para evaluar el uso de red. Estos indicadores permitirán tener un mayor alcance en la evaluación de las arquitecturas microservicios y REST (Tapia et al., 2020).

2. Realizar las pruebas de la metodología METSA a las aplicaciones MS1, MS2, R1 y R2 alojadas en servicios en la nube como Amazon Web Services y Microsoft Azure, para evaluar si afectan a los resultados de los indicadores utilizados para comparar las arquitecturas microservicios y REST (Saransig y Tapia, 2018).

3. Ampliar la medición del rendimiento utilizando el orquestador de contenedores Kubernetes para microservicios, el que administra y automatiza los procesos en contenedores, para evaluar cómo afecta en el rendimiento de la aplicación de microservicios (Flygare y Holmqvist, 2017).

4. Realizar las pruebas de la metodología en sistemas operativos MS Windows debido a que presenta mayor presencia en el mercado de equipos y servidores (Hasnain y Rafi, 2019), así como en otras distribuciones de Linux como Ubuntu y Arch Linux debido a que presentan mayor cantidad de usuarios en el mercado y cuentan con licencia GPL de software libre y debido a que consumen menos recursos de CPU y memoria RAM que los demás sistemas operativos (Hasnain y Rafi, 2019). El uso de estos sistemas operativos en el entorno de prueba permitirá ampliar las comparaciones entre microservicios y REST (Hasnain y Rafi, 2019).

5. Ejecutar distintos procesos con las tecnologías Docker y Kubernetes orientadas a servicios (Netto et al., 2018) y después realizar una investigación longitudinal analizando estudios previos para evaluar su impacto en la calidad de servicio en diversas organizaciones.

6. Considerar los criterios siguientes: tiempo de respuesta, uso de recursos y nivel de seguridad, para la evaluación de las tecnologías microservicios y REST de la presente investigación y así poder realizar un estudio comparativo entre los sistemas operativos Linux y Windows, con la finalidad de ampliar la investigación y determinar cuáles de las tecnologías tiene mejor rendimiento en estos sistemas operativos.
7. Ampliar la muestra utilizada para la medición de los indicadores, utilizando una base de datos con una cantidad mayor de información de imágenes y solicitudes. Se sugiere utilizar 8000 solicitudes para las pruebas de carga de usuarios, uso de memoria RAM y uso de CPU (Nyman, 2018) y 50 imágenes por tamaño (10 KB, 100 KB, 1 MB, 10 MB y 100 MB), de manera que permita obtener conclusiones más exactas en los indicadores siguientes: carga de usuarios, uso de memoria RAM y uso de CPU.

REFERENCIAS

ALBERTOS, E. *Arquitecturas software para microservicios: una revisión sistemática de la literatura* [en línea]. Trabajo Fin de Máster. Universidad Politécnica de Madrid, 2018 [consultado: 23 de mayo de 2021]. Disponible en: http://oa.upm.es/51460/

AL-DEBAGY, O. y MARTINEK, P. A Comparative Review of Microservices and Monolithic Architectures. *IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. Budapest: IEEE, 2018, 1(1), pp. 49-154

ARELLANO, A. *Sistema informático para la gestión académica del instituto de educación superior tecnológico público "Adolfo Vienrich"*. Tesis de Pregrado. Universidad Peruana Los Andes. Tarma, 2020.

ASHRAF, M. y ALJEDAIBI, W. ATAM-based architecture evaluation using LOTOS formal method. *International Journal of Information Technology and Computer Science*. Jeddah, Arabia Saudita: King Abdulaziz University, 2017, 9(3), pp. 10-18.

ATHAR, A., LIAQAT, R. y AZAM, F. A Comparative Analysis of Software Architecture Evaluation Methods. *Journal of Software*. National University of Sciences and Technology, 2016, 11(9), pp. 934-942.

MORENO, D. *Despliegue de Microservicios mediante técnicas de virtualización ligeras basadas en contenedores* [en línea]. Trabajo de Fin de Grado. Universidad Politécnica de Madrid, 2018 [consultado: 23 de mayo de 2021]. Disponible en: http://oa.upm.es/cgi/export/53075/

CEBECI, K. *Design of a queue-based microservices architecture and performance comparison with monolith architecture*. Tesis de Máster. Marmara University. Estambul, 2019.

CHOW, W. *Prevención de ataques de Ransomware conocidos en redes informáticas, utilizando la tecnología Check Point Sandblast en el perímetro y en usuarios finales comprendido en el periodo de septiembre del 2017 a abril del 2018*. Tesis Doctoral. Universidad Nacional Autónoma de Nicaragua. León, 2018.

CORREDOR, N. *Estado del arte revisión sistemática de la seguridad orientada a Rest.* Trabajo de Fin de Grado. Universidad Católica de Colombia. Bogotá, Colombia, 2017.

CRUZ, C., OLIVARES, S. y GONZÁLEZ, M. *Metodología de la Investigación*. México: Grupo Editorial Patria, 2014. ISBN: 978-607438-876-3.

DAGAR, D. y GUPTA, A. A Comparison of Vulnerability Assessment Tools OWASP 2.7. 0 & Pentest on Demo Web Application. *CPJ Global Review A National Journal of Chanderprabhu Jain College of Higher Studies.* Delhi, India: CPJ College. 2020, 2(1), pp. 46-50.

VÉLEZ, P. y ESPINOZA, M. Propuesta de arquitectura de microservicios, metodología Scrum para una aplicación móvil de control académico: Caso Escuela Profesional de Obstetricia de la Universidad Nacional Mayor de San Marcos. Lima: *Hamut´ay*. 2019, 6(2), pp. 141-158.

DUFFIELD. R. Netdata Agent v1.24: Prometheus/OpenMetrics collector and multi-host database mode. En: Netdata [en línea]. 2020 [consulta: 23 de mayo de 2021]. Disponible en: https://www.netdata.cloud/blog/release-1-24/

FELDERER, M., BUCHLER, M., JOHNS, A., BRUCKER, A., BREU, R. y PRETSCHNER, A. Security Testing: A Survey*. Advances in Computers*. Elsevier. 2016, 101(1), pp. 1-51. ISSN 0065-2458.

FERNÁNDEZ, X. *Seguridad en DOCKER*. Trabajo Fin de Máster. Universitat Oberta de Catalunya, 2018.

FLYGARE, R. y HOLMQVIST, A. *Performance characteristics between monolithic and microservice-based systems*. Tesis de Pregrado. Faculty of Computing Blekinge Institute of Technology. Suecia, 2017.

GÓMEZ, K., ANAYA, R. y CANO, A. Un acercamiento a los microservicios. *Una Ciencia. Revista de Estudios e Investigaciones.* Corporación Universitario Adventista. 2017, 1(1), pp. 116-126.

GÓMEZ, O. Evaluando Arquitecturas de Software, Métodos de Evaluación. *Software Guru*. México. 2007, 1(1), pp. 36-39.

GUARDIOLA, J. *Metodología de pentesting para plataforma de microservicios KUBERNETES*. Tesis de pregrado. Universidad Politécnica de Madrid, 2020.

HALILI, F. y RAMADANI, E. Web services: a comparison of soap and rest services. *Modern Applied Science.* Canadian Center of Science and Education. 2018, 12 (3), pp. 175-183.

HASNAIN, G. y RAFI, F. Windows, Linux, Mac Operating System and Decision Making [Windows, Linux, sistema operativo Mac y toma de decisiones]. *International Journal of Computer Applications*. 2019, 177 (27), pp 11-15.

HERNÁNDEZ, R., FERNÁNDEZ, C. y BAPTISTA, P. *Metodología de la investigación*. 6. México: McGraw-Hill, 2016. ISBN: 978-1-4562-2396-0.

HUSNI, H. y SAIFAN, A. Cloud testing: Steps, tools, challenges. *Proceedings of the New Trends in Information Technology (NTIT'17)*. Amman: Universidad de Jordania. 2017, 1(1), pp. 25-27.

IBRAHIM, A., BOZHINOSKI, S. y PRETSCHNER, A. Attack graph generation for microservice architecture. *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. Cyprus: Association for Computing Machinery. 2019, 1(1), pp. 1235-1242

JHA, N. y POPLI, R. Comparative analysis of web applications using JMeter. *International Journal of Advanced Research in Computer Science*. India: IJARCS. 2017, 8(3), pp. 774-777.

KHAN, R. *Comparative Study of Performance Testing Tools: Apache JMeter and HP LoadRunner*. Tesis de Máster. Faculty of Computing Blekinge Institute of Technology 2016. Suecia, 2016.

KOSKINEN, M. *Microservices and containers: benefits and best practices*. Tesis de Pregrado. Turku University of Applied Sciences. Finlandia, 2016.

LEWIS, W. *Software Testing and Continuous Quality Improvement*. 3. New York: Auerbach Publications, 2009. ISBN: 978-1-4200-8073-5.

LÓPEZ DE JIMÉNEZ, R. Pruebas de penetración en aplicaciones web usando hackeo ético. *Revista Tecnológica*. El Salvador: Escuela Especializada en Ingeniería ITCA-FEPADE. 2017, 1(10), pp. 13-19.

LÓPEZ HINOJOSA, J. *Arquitectura de software basada en microservicios para desarrollo de aplicaciones web de la Asamblea Nacional.* Tesis de Maestría. Universidad Técnica del Norte. Ibarra, Ecuador, 2017.

MAHENDRA, N. y KHAN, S. A Categorized Review on Software Security Testing. *International Journal of Computer Applications*. India: Department of Computer Application Integral University. 2016, 154(1), pp. 21-25.

MAHJANI, A. Security issues of virtualization in cloud computing environments [Problemas de seguridad de la virtualización en entornos de computación en la nube]. Tesis de Maestría. Luleá University of Technology. Suecia, 2015.

MATEUS, N., CRUZ, M. y GONZAGA, L. Security in Microservices Architectures. *CENTERIS - International Conference on Enterprise Information Systems*. Portugal: Elsevier. 2020, 1(1), pp. 1-12.

Mozilla Foundation. Monitor de Red. *MDN web docs* [en línea]. 2019. [Consulta: 23 de mayo de 2021]. Disponible en: https://developer.mozilla.org/es/docs/Tools/Monitor_de_Red.

MIKUŁA, M. y DZIEŃKOWSKI, M. Comparison of REST and GraphQL web technology performance. *Journal of Computer Sciences Institute*. Polonia: Department of Computer Science, Lublin University of Technology. 2020, 16 (1), pp. 309-316.

MAS'AD, R. *Migración de un sistema monolítico a una arquitectura de microservicios* [en línea]. Tesis de Pregrado. Universidad Técnica Federico Santa María. Santiago, Chile, 2016. [Consulta: 23 de mayo de 2021]. Disponible en: http://hdl.handle.net/11673/19938

NETTO, H., LUIZ, A., DE OLIVEIRA, L. y OLIVEIRA, C. Koordinator: A service approach for replicating DOCKER containers in KUBERNETES [Koordinator: un enfoque de servicio para la replicación Contenedores

DOCKER en KUBERNETES]. *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2018, 1(1), pp. 58-63.

NYMAN, J. *Evaluating the mitigating effect on HTTP flood attacks using an application layer Challenge-response approach*. Tesis de maestría. Umeå University. Suecia, 2018.

ÑAUPAS, H., PALACIOS, J., ROMERO, H. y VALDIVIA, M*. Metodología y diseños en investigación científica. Cuantitativa–Cualitativa y Redacción de la Tesis*. 5. Bogotá: Ediciones de la U, 2018. ISBN: 978-958-762-876-0.

PIELLI, C., ZUCCHETTO, D., ZANELLA, A., VANGELISTA, L. y ZORZI, M. Platforms and Protocols for the Internet of Things. *EAI Endorsed Transactions on Internet of Things*. Padova, Italia: Department of Information Engineering, University of Padova. 2015, 15 (1), pp. 1-15.

PINCHAO, E. *Implementación de un curso MOOC aplicando la metodología MIA® (Mindfulness Into Action), mediante plataforma EDJET para el Instituto Mia®, Ibarra, Ecuador*. Tesis de Licenciatura. Universidad Técnica del Norte. Ibarra, 2020.

PUNDHIR, Y. Cloud computing applications and their testing methodology. *Bookman International Journal of Software Engineering*, 2013, 2(1), pp. 1-4.

RADHAKRISHNAN, G. Non-experimental research designs: Amenable to nursing contexts. *Asian Journal of Nursing Education and Research*. India: Bharatesh College of Nursing, 2016, 3(1), pp. 25-28.

RAMOS, J. Pruebas de penetración o Pen Test. *Revista de Información, Tecnología y Sociedad*. La Paz: Facultad de Ciencias Puras y Naturales, Universidad Mayor de San Andrés, 2013, 1(8), pp. 31-33. ISSN 1997-4044.

REZA, S., HASAN, W y CHAKRABORTY, S. A Comparative Overview on Penetration Testing. *Proc. of The Fourth Intl. Conf. On Advances in Computing, Electronics and Electrical Technology – CEET*. Estados

Unidos: Institute of Research Engineers and Doctors, 2015, 1(1), pp. 25-28.

RICHARDS, M*. Microservices vs. Service-Oriented Architecture*. California: O'Reilly Media, 2016. ISBN: 9781491975657.

RUELAS, D. *Modelo de composición de microservicios para la implementación de una aplicación web de comercio electrónico utilizando KUBERNETES*. Tesis de Pregrado. Universidad Nacional del Altiplano, 2017.

ROMERO, M. Pruebas de bondad de ajuste a una distribución normal. *Revista enfermería del trabajo*. 2016, 6 (3), p. 105-114.

SALVADORI, I., HUF, A., OLIVEIRA, B., DOS SANTOS, R. y SIQUEIRA, F. Improving entity linking with ontology alignment for semantic microservices composition. *International Journal of Web Information Systems*, Florianópolis, Brasil: Federal University of Santa Catarina, 2017. 13 (3), pp. 302-323. ISSN: 1744-0084.

SÁNCHEZ, M. *Despliegue automatizado de infraestructura de microservicios y prueba de concepto de funcionamiento*. Trabajo fin de Máster. Universidad de Granada, España. 2018.

SARANSIG, A. y TAPIA, F. Performance analysis of monolithic and micro service architectures–containers technology [Análisis de desempeño de arquitecturas monolíticas y de microservicio - tecnología de contenedores]. *International Conference on Software Process Improvement*. Springer, Cham, 2018, 1(1), pp. 270-279.

SAROJADEVI, H. Performance testing: methodologies and tools. *Journal of Information Engineering and Applications*, 2011, 1 (5), pp. 5-13.

SCOTT, J. *A practical Guide to Microservices and Containers: Mastering the Cloud, Data and Digital Transformation* [Una guía práctica de microservicios y contenedores: dominar la nube, los datos y la transformación digital]. MAPR Data Technologies.

SHRESTHA, H. A *Design Science Research Methodology for Microservice Architecture and System Research*. Tesis de Maestría. Aalto University. Finlandia, 2019.

SIMBAÑA, M. *Estudio del contenedor Cloud DOCKER y propuesta de implementación para la plataforma Cloud FICA*. Tesis de Pregrado. Universidad Técnica del Norte. Ibarra, 2016.

STEINHOLT, R. *A study of Linux Containers and their ability to quickly offer scalability for web services*. [Un estudio de los contenedores de Linux y su capacidad para ofrecer rápidamente escalabilidad para servicios web.]. 2015. Tesis de Maestría. Universidad de Oslo. Noruega, 2015.

SYAIKHUDDIN, M., ANAM, C., RINALDI, A. y CONORAS, M. Conventional Software Testing Using White Box Method. *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*. Indonesia: Universitas AMIKOM Yogyakarta, 2018, 3(1), pp. 65-72.

TAPIA, F., MORA, M., FUERTES, W., AULES, H., FLORES, E. y TOULKERIDIS, T. From Monolithic Systems to Microservices: A Comparative Study of Performance. Applied Sciences, 2020, 10(17), pp. 1-35.

TARKOWSKA, A. Eleven quick tips to build a usable REST API for life sciences. *PLOS Computational Biology*, 2018, 14 (12), pp. 1-8.

TIHOMIROVS, J. y GRABIS, J. Comparison of soap and rest based web services using software evaluation metrics. *Information Technology and Management Science*. Letonia: Riga Technical University, 2016, 19 (1), pp. 92-97.

VERBORGH, R. et al. The fallacy of the multi-API culture. *Journal of Documentation*, 2015, 9(2), pp. 233-252.

VERONA, S., PÉREZ, Y., TORRES, L., DELGADO, M. y YÁÑEZ, C. Pruebas de rendimiento a componentes de software utilizando programación orientada a aspectos. *Ingeniería Industrial*, 2016, 37 (3), pp. 278-285.

VILLAMIZAR, Mario, et al. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. *2015 10th Computing Colombian Conference (10CCC)*. Bogotá: IEEE, 2015, 1(1), pp. 583-590.

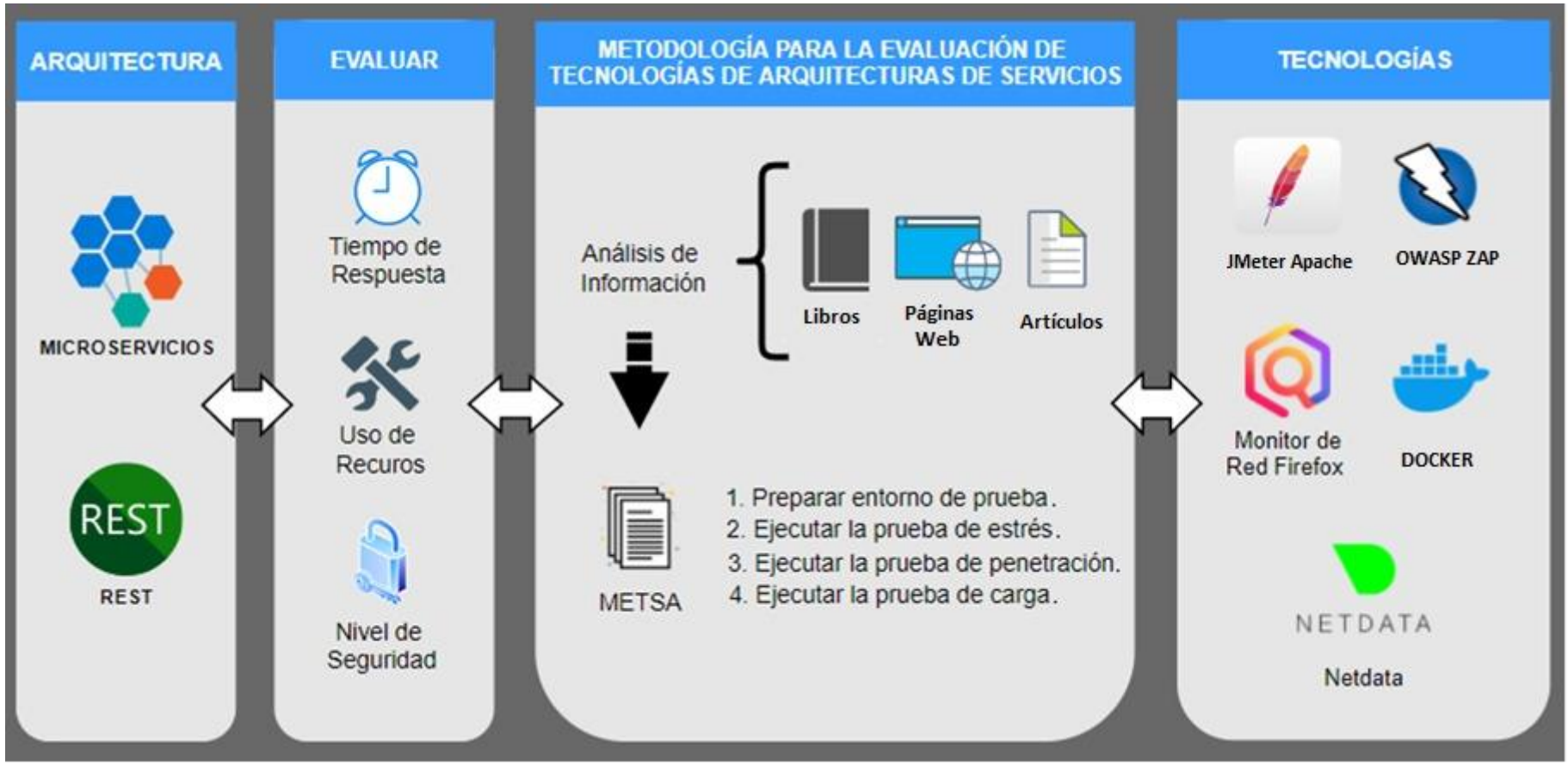
WAGH, K. y THOOL, R. A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host. Journal of Information Engineering and Applications. India: Institute of Technology Pune. 2012, 2 (5), pp. 12-16.

ZIRKELBACH, C., KRAUSE, A. y HASSELBRING, W. Modularization of research software for collaborative open source development [Modularización de software de investigación para el desarrollo colaborativo de código abierto]. *arXiv preprint arXiv:1907.05663*, 2019.

ANEXOS

Anexo 1: Arquitectura utilizada para la investigación

Figura 3 Arquitectura utilizada para la investigación



Anexo 2: Matriz de operacionalización de variables

Tabla 24 Matriz de operacionalización de variables

Variable	**Definición conceptual**	**Definición operacional**	**Dimensión**	**Indicador**	**Instrumento**	**Escala de medición**
Uso de recursos de las tecnologías de Microservicios y REST.	Para cargas de trabajo de alto rendimiento que requieren capacidades de hardware particulares para lograr su rendimiento objetivo (Los especialistas de Intel, 2018, p. 2)	El análisis de rendimiento debe realizarse en todos los niveles de prueba, con una construcción cuidadosa de los datos de prueba adecuados para los escenarios de prueba de rendimiento (Felderer, Buchler, Johns, Brucker, Breu y Pretschner, 2016, p. 9)	Uso Recursos (Verona, Pérez, Torres, Delgado y Yáñez, 2016, p. 281. Sarojadevi, 2012, p.5)	Uso de memoria RAM (Verona, Pérez, Torres, Delgado y Yáñez, 2016, p. 281).	Ficha de recolección de datos (Posada, 2016, p. 14).	De la razón
				Uso de CPU (Verona, Pérez, Torres, Delgado y Yáñez, 2016, p. 281).	Ficha de recolección de datos (Posada, 2016, p. 14).	De la razón
Tiempo de respuesta de las tecnologías de Microservicios y REST.	Recibe la solicitud del cliente (Ashraf y Aljedaibi, 2017, p. 13).	El tiempo de respuesta es una medida para evaluar los tiempos de respuesta de un producto de software cuando se realiza una solicitud. (Verona, Pérez, Torres, Delgado y Yáñez, 2016, p. 281).	Tiempo de respuesta (Ashraf y Aljedaibi, 2017, p. 13. Jha y Popli, 2017, p. 774)	Tiempo de respuesta de carga de imágenes. (Ashraf y Aljedaibi, 2017, p. 13)	Ficha de recolección de datos (Posada, 2016, p. 14).	De la razón
				Tiempo de respuesta de carga de usuarios. (Mikuła y Dzieńkowski, 2020, p. 13)	Ficha de recolección de datos (Posada, 2016, p. 14).	De la razón
Nivel de seguridad de las tecnologías Microservicios y REST.	Verifique las debilidades del mecanismo de seguridad implementado. Se hace para encontrar las vulnerabilidades de un sistema y para determinar si el sistema está protegido de intrusos o no (Mahendra y Khan, 2016, p.21)	Usar datos de entrada maliciosos y no esperados que probablemente aprovechen las vulnerabilidades consideradas. (Felderer, B¨uchler, Johns, Brucker, Breu y Pretschner, 2016, p. 8)	Nivel de seguridad (Felderer, Buchler, Johns, Brucker, Breu y Pretschner, 2016, p. 8)	Ataques bloqueados (Felderer, Buchler, Johns, Brucker, Breu y Pretschner, 2016, p. 8)	Ficha de recolección de datos (Posada, 2016, p. 14).	De la razón
				Cantidad de vulnerabilidades (Felderer, Buchler, Johns, Brucker, Breu y Pretschner, 2016, p. 8)	Ficha de recolección de datos (Posada, 2016, p. 14).	De la razón

Anexo 3: Matriz de consistencia

Tabla 25 Matriz de consistencia

PROBLEMA	OBJETIVO	HIPÓTESIS	VARIABLE	DIMENSIÓN	INDICADORES
General	General	General			
No se han encontrado estudio relacionados que permitan comparar las tecnologías Microservicios y REST según su uso de recursos, tiempo de respuesta y el nivel de seguridad.	Comparar las tecnologías de Microservicios y REST según su uso de recursos, tiempo de respuesta, nivel de seguridad.	La tecnología Microservicios tiene un mejor rendimiento que la tecnología REST (Scott, 2017, p.15)	-	-	-
Específica	Específica	Específica			Indicadores
No se han encontrado estudios relacionados que permitan comparar las tecnologías Microservicios y REST según su rendimiento de uso de recursos.	Comparar las tecnologías de Microservicios y REST según su rendimiento, uso de recursos.	La tecnología Microservicios tiene menor uso de CPU que la tecnología REST. (Belinchón, 2018, p. 19)	Uso de recursos de las tecnologías de Microservicios y REST.	Uso de recursos (Verona, Pérez, Torres, Delgado y Yáñez, 2016, p. 281. Sarojadevi, 2015, p.5)	Uso de CPU (Verona, Pérez, Torres, Delgado y Yáñez, 2016, p. 281).
		La tecnología Microservicios tiene un menor uso de memoria RAM que la tecnología REST. (Mas'ad, 2016, p. 54)			Uso de memoria RAM (Verona, Pérez, Torres, Delgado y Yáñez, 2016, p. 281).
No se han encontrado estudios relacionados que permitan comparar las tecnologías Microservicios y REST según su rendimiento de tiempo de respuesta.	Comparar las tecnologías de Microservicios y REST según su rendimiento de tiempo de respuesta.	La tecnología de Microservicios tiene menor tiempo de respuesta de carga de imágenes que la tecnología REST. (Gómez, 2017, p. 122)	Tiempo de respuesta de las tecnologías de Microservicios y REST.	Tiempo de respuesta (Ashraf y Aljedaibi, 2017, p. 13. Jha and Popli, 2017, p. 774)	Tiempo de respuesta carga de imágenes. (Ashraf y Aljedaibi, 2017, p. 13, Jha y Popli, 2017, p. 774)
		La tecnología de Microservicios tiene menor tiempo de respuesta de carga de usuarios que la tecnología REST. (Mikuła y Dzieńkowski, 2020, p. 13), Al-Debagy y Martinek (2018).			Tiempo de respuesta carga de usuarios. (Mikuła y Dzieńkowski, 2020, p. 13).

PROBLEMA	OBJETIVO	HIPÓTESIS	VARIABLE	DIMENSIÓN	INDICADORES
No se han encontrado estudios relacionados que permitan comparar las tecnologías Microservicios y REST según su rendimiento en nivel de seguridad	Comparar las tecnologías de Microservicios y REST según su rendimiento en nivel de seguridad.	La tecnología REST tiene mayor porcentaje de ataques bloqueados que la tecnología Microservicios. (Bozhinoski y Pretschner, 2019)	Nivel de seguridad de tecnologías de Microservicios y Rest.	Seguridad (Felderer, Buchler, Johns, Brucker, Breu y Pretschner, 2016, p. 8)	Ataques bloqueados (Felderer, Buchler, Johns, Brucker, Breu y Pretschner, 2016, p. 8)
		La tecnología REST tiene menor cantidad de vulnerabilidades que la tecnología Microsercicios. (Fernández, 2018, p. 47), Guardiola (2017)			Cantidad de vulnerabilidades (Felderer, Buchler, Johns, Brucker, Breu y Pretschner, 2016, p. 8)

Anexo 4: Uso de metodologías para la evaluación de METSA

La siguiente sección presenta las técnicas y métodos recopilados para el desarrollo de la metodología METSA.

1. Performance Assessment of Software Architecture (PASA)

La metodología PASA está orientada a evaluar el desempeño de una arquitectura en diferentes escenarios. Sobre la metodología PASA, Athar, Liaqat y Azam (2016) explicaron: “este método contiene anti-patrones y estilos de SA sensibles al rendimiento como herramientas de análisis y valida la actividad de análisis de SA del proceso de ingeniería del rendimiento informado” (p. 938). Entre sus objetivos está medir el tiempo de respuesta a eventos aleatorios. Athar, Liaqat y Azam (2016) explicaron:

> El principal objetivo de PASA es medir la capacidad del candidato SA con respecto al objetivo de desempeño. PASA ayuda a la actividad de análisis de SA utilizando escenarios relacionados con el desempeño como fuente de razonamiento. Además, este análisis también verifica otros atributos de calidad como la mantenibilidad. PASA también se utilizó para encontrar las diferencias entre diferentes SA. (p. 938)

Para la implementación de la metodología, se recomienda tener documentado el sistema o alternativamente, extraer los recursos faltantes del personal del equipo. Al respecto, Athar, Liaqat y Azam (2016) explicaron:

> PASA se puede aplicar al principio del proceso de desarrollo, lo que se denomina posterior a la implementación o durante la actualización de un sistema heredado. Este método se ha aplicado a sistemas integrados, sistemas en tiempo real y sistemas basados en la Web y en los sistemas financieros. Al utilizar varias vistas, PASA requiere un documento de descripciones de SA. Si la documentación de SA no está bien definida, entonces puede surgir el problema, que es la información arquitectónica que se extrae del código del software, los desarrolladores y algunos otros artefactos. (p. 938)

Este método tiene nueve pasos a realizar según Gómez (2007): “(a) presentar el método de evaluación, (b) presentar la arquitectura, (c) identificar los casos de uso críticos, (d) seleccionar los principales movimientos de desempeño, (e) identificar los objetivos de desempeño, (f) aclarar la

arquitectura y discutirla, (g) analizar la arquitectura, (h) identificar alternativas y (i) presentar resultados" (p. 37). Finalmente, sobre los resultados del uso del método, Athar, Liaqat y Azam (2016) indicaron:

> Este método incluye técnicas tanto cualitativas como cuantitativas para probar los riesgos potenciales que pueden ser intrínsecos en una SA. Además, este método también aclara cómo los diferentes escenarios pueden ser útiles para simbolizar atributos de calidad en tiempo de ejecución como el rendimiento. Mediante el uso de diferentes estudios de caso, PASA y sus diversas técnicas han sido validadas. (p. 939)

2. Cloud Testing

La virtualización de las aplicaciones ha permitido que las pruebas sean posibles en este entorno. De acuerdo con Pundhir (2013), las pruebas en la nube son una forma de prueba de software, con la diferencia que estas pruebas se realizan bajo la infraestructura de la nube (p. 1). Según Pundhir (2013), Cloud Testing se divide en:

1. Prueba de estrés
    a. Reconocer los problemas de aplicación que se hacen evidentes en condiciones de riesgo.
    b. Determinar la solidez, conveniencia y consistencia de una aplicación en condiciones extremas.
    c. No se espera que el sistema procese la sobrecarga sin los recursos adecuados, sino que se comporte (p. ej., falla) de una manera aceptable (p. ej., no corrompa o pierda datos o pérdida).
2. Prueba de carga
    a. Identificar los estados de desempeño serio.
    b. Identificar el estado de la carga de trabajo para distribuir toda la carga entre los escenarios clave.
    c. Identificar las métricas para verificarlas con sus objetivos de desempeño.
    d. Diseñar pruebas para simular la carga.
    e. Utilizar las herramientas disponibles para implementar la carga.
3. Prueba funcional
    a. Probar el comportamiento esperado de una aplicación.

b. Un requisito funcional puede ser nominal, comercial o basado en procesos.

4. Prueba de compatibilidad
    a. Probar la compatibilidad entre el producto o software con la plataforma.
    b. Incluir varias configuraciones de hardware, diferentes sistemas operativos o plataformas y entorno de red.
5. Prueba de rendimiento del navegador
    a. Verificar el soporte de la aplicación en diferentes navegadores.
    b. Mejorar el rendimiento en cada navegador web.
6. Prueba de punto a punto
    a. Probar la conectividad del producto con la red.
    b. Medir el rendimiento y el contraste de la red con el servicio adecuado.

3. Prueba de ciclo de vida

La prueba de ciclo de vida está presente en el ciclo de vida de la metodología en cascada en el desarrollo de software, que se lleva a cabo en paralelo al desarrollo y se aplica como un proceso continuo. Según Lewis (2009), esta prueba se produce en paralelo con el ciclo de vida del desarrollo y es un proceso continuo; además, el proceso de mejora continua de Deming se aplica a las pruebas de software utilizando el círculo de calidad, los principios y las técnicas estadísticas (p. 89).

Además, sobre las características de las pruebas en el ciclo de vida, Lewis (2009) mencionó que “fomenta que las pruebas se realicen fuera de la organización de desarrollo” (p. 89) y que “La motivación para esto es que existen requisitos claramente definidos y es más eficiente que un tercero verifique estos requisitos” (p. 89). Esto se refiere en parte al hecho de que los programadores no pueden probar su código porque se cuestiona la veracidad de los mismos, por lo que los responsables de las pruebas y el programador no pueden pertenecer al mismo departamento dentro de la organización. Esta prueba consta de dos enfoques según Lewis (2009):

> Los dos enfoques principales de verificación de garantía de calidad para cada fase del ciclo de vida son las revisiones técnicas y las pruebas de software. Las revisiones técnicas son más preventivas; es decir, su

objetivo es eliminar los defectos lo antes posible. Las pruebas de software verifican el código real que se ha producido. (p. 89)

Finalmente, se desarrolla un software de plan de prueba recomendado al comienzo del ciclo de vida para facilitar su adopción con las otras fases; además, debes tener los pasos a seguir para realizar las pruebas. Al respecto Lewis (2009) explicó:

> Un plan de prueba define los objetivos de la prueba, el alcance, la estrategia y el enfoque, los procedimientos de prueba, el entorno de prueba, los criterios de finalización de la prueba, los casos de prueba, los elementos a probar, las pruebas a realizar, los programas de prueba, los requisitos de personal, los procedimientos de informes, los supuestos riesgos y la planificación de contingencias. (p. 92)

El documento tiene una serie de pasos que según Lewis (2009) son los siguientes:

1. Definir los objetivos de las pruebas.
    a. Establecer los resultados esperados con las pruebas.
    b. Determinar las pruebas específicas a realizar.
    c. Determinar los factores críticos de éxito de las pruebas.
    d. Definir el alcance de las pruebas a realizar.
2. Desarrollar el enfoque de prueba.
    a. Definir cómo se realiza cada prueba.
    b. Establecer los criterios de entrada, salida y técnicas a utilizar y los procedimientos de prueba.
    c. Elaborar informes de estado, recursos y habilidades de prueba, riesgos y una definición de la base de prueba.
3. Definir el entorno de prueba.
    a. Definir el hardware, software y redes que se utilizarán en las pruebas.
    b. Definir las herramientas y si son de tipo automático o manual.
4. Desarrollar las especificaciones de prueba.
    a. Las especificaciones de prueba están documentadas.
    b. Identificar las interdependencias y el flujo de trabajo de las especificaciones de prueba. Programar la prueba.

c. Elaboración de cronograma de pruebas y desarrollo de acuerdo a la disponibilidad de recursos.
d. Definir los principales puntos de control y desarrolla planes de contingencia.

5. Revisar y aprobar el plan de prueba
    a. Se obtiene la aprobación y posterior implementación.

Además, Lewis (2009) agrupó algunas técnicas para evaluar software según las fases de desarrollo del proyecto. Son las siguientes:

1. Prueba unitaria:
    a. Se ejecuta en partes específicas del código muchas veces realizado por el mismo programador automatizándolos.
    b. Se encarga de validar funciones específicas del código.
2. Prueba de integración:
    a. Se realizan uniendo partes o módulos del sistema para validar su funcionamiento en conjunto.
    b. Es importante para el modelo cliente-servidor debido a las constantes llamadas del cliente.
3. Prueba del sistema:
    a. Centrarse en las especificaciones de los requisitos generales.
    b. Cubre todos los componentes del sistema.
4. Prueba de aceptación:
    a. Está dirigido a la respuesta del usuario final al software.
    b. Se evalúa de acuerdo con las especificaciones del usuario.

4. Pruebas de penetración

Es una técnica de seguridad para evaluar el comportamiento del programa frente a un posible ataque, según Felderer et al. (2016) “una aplicación o sistema se prueba desde el exterior utilizando una configuración que es comparable a un ataque real de un tercero malintencionado” (p. 19).

Las solicitudes enviadas deben estar conectadas a una red y tener un flujo de desarrollo establecido, Felderer et al. (2016) explicaron que “en la mayoría de los entornos, la entidad que realiza las pruebas tiene potencialmente solo información limitada sobre el sistema bajo prueba y solo

puede interactuar con las interfaces públicas del sistema" (p. 19). Según Felderer et al. (2016) la técnica tiene 4 fases:

1. Planificación
    a. Se documentan las condiciones y límites importantes para la prueba.
    b. Identificar los componentes relevantes de las aplicaciones a probar.
2. Descubrimiento
    a. Las interfaces externas del sistema que son accesibles se identifican y enumeran.
    b. Las interfaces identificadas son la base del ataque inicial de la prueba.
    c. Análisis de vulnerabilidad, identificar el tipo de vulnerabilidad adecuado para la interfaz.
3. Ataque
    a. Las interfaces identificadas se prueban con una serie de intentos de ataque.
    b. Busca encontrar vulnerabilidades de seguridad y recopilar información del sistema y componentes reactivos al ataque.
4. Informes
    a. Se desarrolló en paralelo a las otras fases, se documentan los resultados encontrados en cada fase.

Además, según López De Jiménez (2017), las pruebas de penetración se pueden dividir según sus objetivos. Estos objetivos son:

1. Tipos de pruebas de penetración:
    a. Penetración de caja negra: realiza pruebas de flujo de datos, sin tener en cuenta la estructura interna del programa, ignora los detalles internos.
    b. Penetración de caja blanca: pruebas más completas donde el responsable de la prueba conoce la composición del software y conexiones a la red.
    c. prueba Gray de penetración: pruebas intermedias donde se conoce parcialmente la estructura del proyecto. (p. 15).

Por otro lado, Ramos (2013) estableció otros criterios para delimitar los tipos de pruebas de penetración (p. 31). Estos criterios son los siguientes:

1. Pruebas de penetración con objetivo: se buscan vulnerabilidades en partes específicas de los sistemas informáticos críticos de la organización.
2. Pruebas de penetración no objetivas: consisten en examinar todos los componentes de los sistemas informáticos pertenecientes a la organización.
3. Ciegos pruebas de penetración: en estas pruebas solo se utiliza la información pública disponible sobre la organización.
4. Pruebas de penetración informadas: aquí se utiliza la información privada otorgada por la organización sobre sus sistemas informáticos.
5. Pruebas de penetración externas: se realizan desde lugares externos a las instalaciones de la organización.
6. Pruebas de penetración interna: se realizan dentro de las instalaciones de la organización con el objetivo de evaluar las políticas y mecanismos de seguridad interna de la organización.

Anexo 5: Metodología para la evaluación de tecnologías de arquitectura de servicios

La siguiente sección detalla los pasos seguidos para el desarrollo de la metodología METSA. También, se describen los cuatros procesos comprendidos detallando entradas, procedimiento y salidas de cada proceso.

Descripción y soporte de la solución

El desarrollo de pruebas de evaluación para tecnologías utilizando las arquitecturas de Microservicios y Rest no está determinada dentro de un marco metodológico específico, por lo que la nueva metodología se basa en los indicadores comúnmente tomados para las pruebas de evaluación, como detallan Syaikhuddin, Anam, Rinaldi y Conoras (2018) explicaron:

Los principios para una correcta aplicación de las pruebas de evaluación: a) las pruebas deben enfocarse en las necesidades del usuario, (b) el tiempo y los recursos disponibles son limitados, (c) optimizar el uso de recursos, (d) las pruebas deben ser escalables , (e) las pruebas deben ser realizadas por un equipo externo a la entidad responsable de la aplicación, (f) las pruebas deben basarse en las necesidades del cliente, (g) tener el personal adecuado para realizar las pruebas, (h) uso de informes de prueba, casos de prueba para un resumen de los resultados de las pruebas, (i) las pruebas deben realizarse lo antes posible en el proceso de desarrollo del software, (j) la planificación de las pruebas Es necesario para su puesta en servicio y (k) El plan de pruebas debe actualizarse de acuerdo con los eventos.

También es necesario establecer los objetivos marcados al realizar las pruebas. Según Jha y Popli (2017), los objetivos se pueden reducir a tres principales:

(a) Estabilidad: determina el tiempo de actividad de la aplicación durante un tiempo determinado y la respuesta del servidor a las consultas constantes. Las aplicaciones están diseñadas para durar mucho tiempo bajo una configuración con poca variación a eso.

(b) Tiempo de respuesta: establezca el tiempo entre el envío de la información del cliente y la respuesta del servidor. Esto dependerá del tamaño, volumen de la información, así como del rendimiento del servidor, el tiempo se minimiza.

(c) Escalabilidad: Estará determinada por el momento de la transacción, el número de usuarios, clientes soportados por el programa en un tiempo establecido con una mínima tolerancia de retardo.

De esta manera, la nueva metodología METSA (Metodología para la Evaluación de Tecnologías de Arquitectura de Servicio) comienza delimitando los objetivos principales, reconocer la información requerida establecidas en Performance Assessment of software Architecture (PASA) (Athar, Liaqat y Azam, 2016). En segundo lugar, se determinó el plan de pruebas y el entorno de pruebas utilizando la técnica de prueba de ciclo de vida (Lewis, 2009). Luego, mediante el conjunto de técnicas de Cloud Testing fijar los procedimientos que tendrá la nueva metodología (Pundhir, 2013). Finalmente, se utilizó los procedimientos de vulnerabilidad de las pruebas de penetración (Felderer et al., 2016).

Plan de proyecto

Las tareas a realizar fueron las siguientes:

1. Recopilación de la bibliografía referente a las metodologías presentadas.
2. Desarrollo de la nueva metodología para la evaluación de tecnologías de las arquitecturas Microservicios y REST.
3. Aplicación de la nueva metodología para evaluar las tecnologías de las arquitecturas Microservicios y REST.
4. Reporte de indicadores.

Proceso de desarrollo de la metodología

El proceso para llevar a cabo la nueva metodología fue el siguiente.

1. Definir los objetivos y los requerimientos previos establecido en la metodología PASA (Athar, Liaqat y Azam, 2016).
2. Preparar el entorno de pruebas con la referencia de las pruebas de ciclo de vida (Lewis, 2009).
3. Ejecutar los dos primeros procedimientos dentro de la metodología utilizando las pruebas de estrés y carga establecidas en Cloud Testing (Pundhir, 2013).

4. Ejecutar el tercer procedimiento dentro de la nueva metodología utilizando la búsqueda de vulnerabilidades y explotación de vulnerabilidades presentes en las pruebas de penetración (Felderer et al., 2016).

A continuación, se presenta la figura 4 que contiene las distintas metodologías o/u métodos planteados en la nueva arquitectura de metodológica. Los recuadros en vertical son los métodos, técnicas utilizados como referencia.

Arquitectura de la Metodología

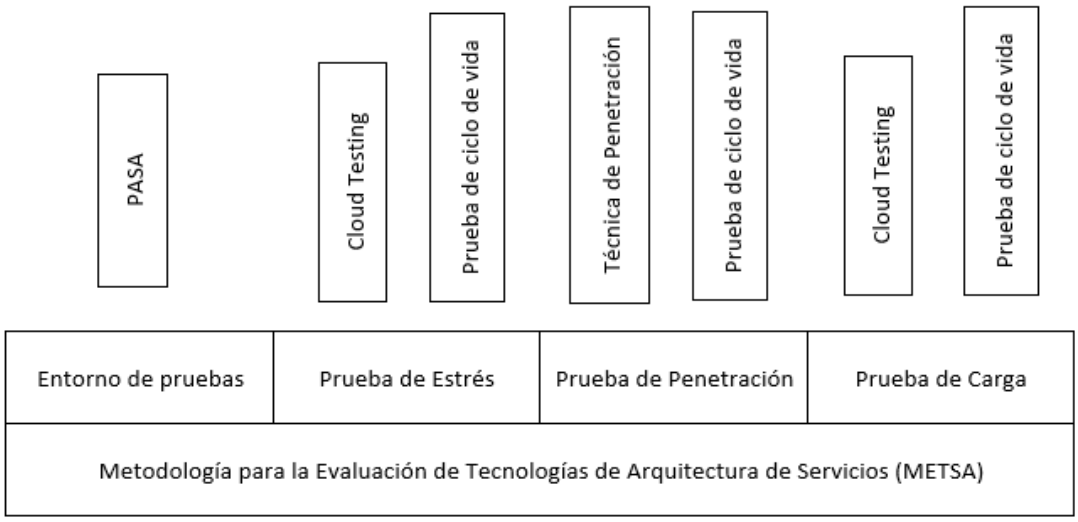


Figura 4 METSA - Arquitectura de la metodología

Descripción de la Metodología

En este apartado se describirá el contenido de los procesos de la metodología METSA. Finalmente se realizará una comparación con las metodologías existentes similar a la desarrollada.

Enfoque metodológico

Para la elaboración de la metodología METSA se utilizaron 5 metodologías y / o técnicas ampliamente utilizadas en las pruebas de software. Estas se detallan a continuación:

1. Performance Assessment of Software Architecture (PASA): Metodología para evaluar el desempeño de arquitecturas de software. Esta metodología perfilará sus procesos para desarrollar los posibles

escenarios de evaluación, enfocándose en los objetivos del proyecto, así como incorporando aspectos de calidad como la mantenibilidad de la arquitectura de software evaluada (Athar, Liaqat y Azam, 2016).

2. Cloud Testing: Consiste en una serie de pasos para evaluar tecnologías en la nube, aunque también puede tomarse como referencia para otra infraestructura. Esta técnica tomará los procesos de pruebas de estrés y pruebas de carga como parte de la metodología METSA (Pundhir, 2013).
3. Prueba de ciclo de vida: las pruebas presentes en la metodología en cascada presentan un ciclo asociado al ciclo de calidad de Deming. Esta técnica eliminará los procesos específicos de definir los objetivos de la prueba, desarrollar el enfoque de la prueba y definir el entorno de la prueba (Lewis, 2009).
4. Pruebas de penetración: Utilizada para medir el rendimiento de la tecnología frente a posibles ataques como software comercial. Se tomarán los procesos de planificación de pruebas, descubrimiento de los puntos afectados, ejecución del ataque sobre la aplicación y finalmente un informe del resultado del ataque (Felderer et al., 2016).

Objetivo

El objetivo de la metodología es evaluar el desempeño de las aplicaciones construidas con las arquitecturas de microservicios y REST con el fin de establecer las diferencias entre cada una según los indicadores de uso de recursos, respuesta y seguridad. Para llevar a cabo la metodología, se tomaron como base técnicas y metodologías como PASA, cloud testing, prueba de ciclo de vida y pruebas de penetración.

Alcance

1. Ejecución de procedimiento Prueba de estrés (Pundhir, 2013).
2. Ejecución de procedimiento Prueba de carga (Pundhir, 2013).
3. Ejecución de procedimiento Prueba de penetración (Felderer et al., 2016).

Entradas

Las entradas son los documentos previamente necesarios para realizar la prueba de esfuerzo, prueba de carga y prueba de penetración de la metodología METSA. Son los siguientes:

1. Establecer los objetivos de cada procedimiento (Athar, Liaqat y Azam, 2016).
2. Determinar las herramientas necesarias para las pruebas en cada procedimiento (Pundhir, 2013).
3. Definir el procedimiento de ejecución de cada prueba (Lewis, 2009).

A continuación, se muestra la figura 5 que contiene los procesos que contiene la metodología METSA. También se muestran los cuadros con las entradas y salidas esperadas.

Proceso METSA

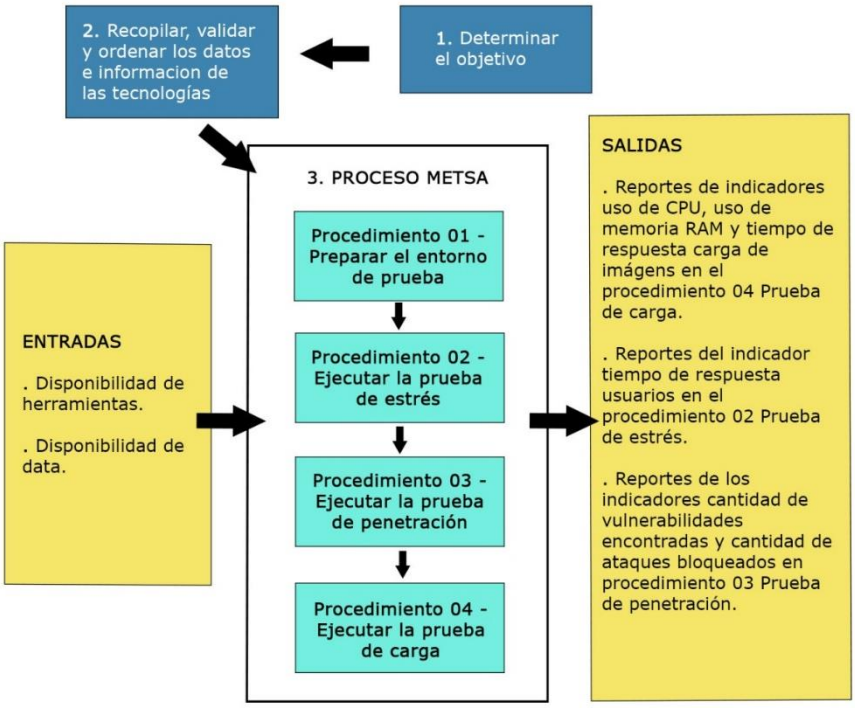


Figura 5 METSA - Proceso de la Metodología

Salidas

Los resultados de la metodología METSA son los siguientes:

1. Reportes de los indicadores uso de CPU, uso de memoria RAM y tiempo de respuesta carga de imágenes en el procedimiento prueba de carga.
2. Reportes del indicador tiempo de respuesta usuarios en procedimiento prueba de estrés.
3. Reportes de los indicadores cantidad de vulnerabilidades encontradas y cantidad de ataques bloqueados en procedimiento las pruebas.

Procesos específicos

P01: Preparar el entorno de prueba

Objetivo

El objetivo del procedimiento es realizar las configuraciones iniciales necesarias para realizar las pruebas (Husni y Saifan, 2017).

Alcance

El alcance del procedimiento contiene:

1. Identificación las especificaciones del hardware, software del sistema operativo donde se realizarán las pruebas (Felderer et al., 2016).
2. Especificación las herramientas necesarias para la realización de las pruebas (Felderer et al., 2016).

Entrada

Para realizar el entorno de prueba son necesarios algunos requisitos que se detallan a continuación:

1. Disponibilidad del sistema operativo para la instalación de las herramientas y tecnologías necesarias para las pruebas (Athar, Liaqat y Azam, 2016).
2. Disponibilidad de data en las aplicaciones de microservicios y REST (Husni y Saifan, 2017).

Proceso

Las actividades incluidas en el proceso de entorno de prueba:

1. Identificar las especificaciones del hardware del equipo donde se realizarán las pruebas. El equipo es una PC de escritorio con 4GB de memoria RAM, disco duro de 500GB, un procesador Intel Core i3-4150 y frecuencia de 3.5 GHz (Husni y Saifan, 2017).

2. Instalar el sistema operativo Linux Mint 20 en PC de escritorio (Husni y Saifan, 2017).
3. Dividir el sistema en 04 particiones (Husni y Saifan, 2017).
4. Instalar el entorno de ejecución node.js y los paquetes de angular 10 para el despliegue de las aplicaciones de microservicios y REST utilizadas (Husni y Saifan, 2017).
5. Instalar el contenedor Docker y la herramienta Docker Compose. para el despliegue de las aplicaciones microservicios. (Reza et al., 2015)
    a. Sudo apt install docker.io
    b. Sudo systemctl start docker
    c. Sudo systemctl enable docker
    d. Sudo apt install docker -compose
    e. Sudo groupadd docker
    f. Sudo gpasswd -a ${USER} docker
6. Instalar un editor de código para la configuración de Docker, en el presente trabajo se utilizó Visual Studio Code (Reza et al., 2015).
7. Instalar la herramienta Netdata utilizando los siguientes comandos (Reza et al., 2015):
    a. bash <(curl -Ss https://my-netdata.io/kickstart.sh)
    b. Abrir la interfaz en localhost:99999
8. Descargar la herramienta JMeter en comprimido desde su página oficial y ejecutar el archivo JMeter ubicado en 'path/apache-jmeter-5.3/bin'.
9. Instalar la herramienta OWASP ZAP.
10. Levantar el servicio backend con el comando "node app" y el frontend con el comando "ng serve" (Reza et al., 2015).
11. Ejecutar la aplicación de microservicios utilizando docker-compose para iniciar el contenedor:
    a. Sudo docker -compose build
    b. Sudo docker -compose up

Salidas

Los resultados del proceso de prueba de carga son los siguientes:

1. Configuraciones iniciales del software de prueba.

2. Herramientas instaladas en el equipo de prueba.

P02: Ejecución de la prueba de estrés

Objetivo

El objetivo del procedimiento es evaluar la capacidad de las aplicaciones para funcionar eficientemente ante situaciones críticas (Husni y Saifan, 2017).

Alcance

El alcance del procedimiento contiene:

1. Identificación de los objetivos de la prueba (Athar, Liaqat y Azam, 2016)
2. Especificación de la estructura y configuraciones necesarias (Lewis, 2009) (Husni y Saifan, 2017).
3. Descripción de herramientas a utilizar (Husni y Saifan, 2017).
4. Realización de pruebas de estrés de microservicios y arquitecturas Rest. (Husni y Saifan, 2017)
5. Documentación de las pruebas realizadas (Athar, Liaqat y Azam, 2016).

Entrada

Para realizar las pruebas de estrés son necesarios algunos requisitos que detallan las condiciones:

1. Disponibilidad de herramienta JMeter (Athar, Liaqat y Azam, 2016).
2. Disponibilidad de data en las aplicaciones de microservicios y REST (Husni y Saifan, 2017).

Proceso

Las actividades incluidas en el proceso de Prueba de estrés:

1. Identificar los objetivos de las pruebas según los resultados buscados (Athar, Liaqat y Azam, 2016).
2. Preparar el entorno de prueba, el sistema operativo será Linux en su distribución Mint 20 (Husni y Saifan, 2017).
3. Preparar formularios de prueba para indicadores y sus unidades de medida. (Husni y Saifan, 2017).
4. Utilizar herramienta JMeter para simular las peticiones y la consola de comandos del sistema para calcular el indicador Tiempo de

respuesta usuarios y registrar la información en el formulario FORM01 (Tabla4).

Salidas

Los resultados del proceso de prueba de carga son los siguientes:

1. Reporte del Indicador Tiempo de respuesta usuarios.

P03: Ejecución de la prueba de penetración

Objetivo

El objetivo de este procedimiento es evaluar el comportamiento de la tecnología rest y microservicios frente a un posible ataque. (Felderer et al., 2016). Para las pruebas de la presente metodología se tomarán las fases de rastreo de vulnerabilidades y ataques.

Alcance

El alcance del procedimiento contiene:

1. Realizar pruebas de penetración en las aplicaciones de las arquitecturas de microservicios y REST (Felderer et al., 2016).
2. Identificación de los objetivos de la prueba (Husni y Saifan, 2017)
3. Especificación del entorno de la plataforma para una prueba exitosa (Husni y Saifan, 2017).
4. Descripción de las herramientas utilizadas (Husni y Saifan, 2017).
5. Determinar las vulnerabilidades encontradas en cada arquitectura (Felderer et al., 2016).

Entrada

Las pruebas de penetración requieren requisitos que detallan las condiciones:

1. Disponibilidad de herramientas OWAS ZAP y JMeter. (Athar, Liaqat y Azam, 2016).
2. Disponibilidad de data en las aplicaciones de microservicios y REST (Husni y Saifan, 2017).

Proceso

Las actividades incluidas en el proceso de Prueba de Penetración:

1. Planificar los objetivos de la prueba (Husni y Saifan, 2017).

2. Preparar el entorno de prueba, el sistema operativo será Linux en su distribución Mint 20 (Felderer et al., 2016).
3. Preparar formularios de prueba para indicadores y sus unidades de medida (Husni y Saifan, 2017).
4. Utilizar la herramienta OWAS ZAP para rastrear las vulnerabilidades y registrar la información en el formulario FORM02 (Tabla 5).
5. Utilizar la herramienta JMeter para realizar un ataque DOS sobre las aplicaciones de microservicios y REST (Nyman, 2018). Luego registrar la información en el formulario FORM03 (Tabla 6).

Salidas

Los resultados del proceso de prueba de penetración son los siguientes:

1. Reporte Indicador Cantidad de vulnerabilidades encontradas.
2. Reporte Indicador Cantidad de ataques bloqueados.

P04: Ejecutar la prueba de carga

Objetivo

El objetivo del procedimiento es evaluar el rendimiento de las aplicaciones frente a un tráfico esperado de peticiones. (Husni y Saifan, 2017).

Alcance

El alcance del procedimiento contiene:

1. Identificación de los objetivos de la prueba (Athar, Liaqat y Azam, 2016)
2. Especificación del entorno de la plataforma para una prueba exitosa (Lewis, 2009) (Husni y Saifan, 2017).
3. Descripción de herramientas a utilizar (Husni y Saifan, 2017).
4. Realización de pruebas de carga en las aplicaciones de microservicios y REST (Husni y Saifan, 2017).
5. Documentación de las pruebas realizadas (Athar, Liaqat y Azam, 2016).

Entrada

Los requisitos que detallan las condiciones son necesarios para realizar las pruebas de carga:

1. Disponibilidad de herramientas JMeter y Firefox (Athar, Liaqat y Azam, 2016).
2. Disponibilidad de data en las aplicaciones de microservicios y REST (Husni y Saifan, 2017).

Proceso

Las actividades incluidas en el proceso de Prueba de carga:

1. Identificar los objetivos de las pruebas según los resultados buscados (Athar, Liaqat y Azam, 2016).
2. Preparar el entorno de prueba, el sistema operativo será Linux en su distribución Mint 20 (Husni y Saifan, 2017).
3. Preparar formularios de prueba para indicadores y sus unidades de medida (Husni y Saifan, 2017).
4. Utilizar herramienta Monitor de Red del navegador Firefox para calcular el indicador Tiempo de respuesta carga de imágenes y registrar la información en el formulario FORM03 (Tabla 6).
5. Utilizar herramienta JMeter para simular las peticiones y la herramienta Netdata para calcular el indicador Uso de CPU y registrar la información en el formulario FORM01 (Tabla 4).
6. Utilizar herramienta JMeter para simular las peticiones y la herramienta Netdata para calcular el indicador Uso de memoria RAM y registrar la información en el formulario FORM02 (Tabla 5).
7. Utilizar herramienta Monitor de Red del navegador Firefoz para calcular el indicador Tiempo de respuesta carga de imágenes y registrar la información en el formulario FORM03 (Tabla 6).

Salidas

Los resultados del proceso de prueba de carga son los siguientes

1. Reporte del indicador Uso de CPU.
2. Reporte del indicador Uso de memoria RAM.
3. Reporte del indicador Tiempo de respuesta carga de imágenes.

Anexo 6: Resultados de ficha de recopilación de datos

En la tabla 26 se muestran los resultados de las pruebas de estrés realizadas a las cuatro aplicaciones agrupados por sección de la página. Para esta prueba se utilizó la herramienta JMeter.

Tabla 26 Instrumento de recopilación de datos - FORM01 Tiempo de respuesta de carga de usuarios.

FORM01		**Procedimiento (02): Prueba de estrés**				
Indicador Tiempo de respuesta de carga de usuarios						
Herramientas: JMeter						
N.º	Sección	Cantidad de Solicitudes	Tiempo de respuesta microservicios MS1 (ms)	Tiempo de respuesta microservicios MS2 (ms)	Tiempo de respuesta alumno R1 (ms)	Tiempo de respuesta equipos R2 (ms)
1	/home	1	1175.0000	1495.0000	1106.0000	1025.0000
2	/home	5	790.8000	911.2000	869.4000	879.4000
3	/home	50	2249.8600	1691.4400	1769.1800	1798.7800
4	/home	100	2669.6400	2521.1600	2798.1000	3193.7300
5	/home	500	9979.9200	10377.9700	12127.9000	9996.5000
6	/home	1000	22256.3300	19635.3700	22256.3300	19423.5500
7	/talleres	1	1153.0000	-	418.0000	-
8	/talleres	5	1365.2000	-	512.0000	-
9	/talleres	50	2284.2200	-	2204.1200	-
10	/talleres	100	3264.7200	-	4384.1000	-
11	/talleres	500	10341.3900	-	22305.9000	-
12	/talleres	1000	52585.0800	-	52585.0800	-
13	/sedes	1	933.0000	-	199.0000	-
14	/sedes	5	1563.4000	-	199.6000	-
15	/sedes	50	1679.1600	-	2015.7800	-
16	/sedes	100	2876.7500	-	4180.7400	-
17	/sedes	500	8795.3500	-	22049.3400	-
18	/sedes	1000	52074.4800	-	52074.4800	-
19	/horarios	1	945.0000	-	191.0000	-
20	/horarios	5	863.2000	-	195.4000	-
21	/horarios	50	1660.7400	-	1954.4800	-
22	/horarios	100	2809.5700	-	4112.5000	-

FORM01		Procedimiento (02): Prueba de estrés				
Indicador Tiempo de respuesta de carga de usuarios						
Herramientas: JMeter						
23	/horarios	500	9939.3400		21145.2400	
24	/estadios	1	-	1148.0000	-	189.0000
25	/estadios	5	-	939.0000	-	200.8000
26	/estadios	50	-	1884.7600	-	1998.9000
27	/estadios	100	-	2394.7000	-	3983.5600
28	/estadios	500	-	9947.4100	-	18536.2700
29	/estadios	1000	-	18804.4300	-	37243.2500
30	/equipos	1	-	1106.0000	-	197.0000
31	/equipos	5	-	868.0000	-	289.6000
32	/equipos	50	--	1591.9600	-	2003.8600
33	/equipos	100	-	2820.8600	-	3969.3700
34	/equipos	500	-	11363.1200	-	18488.3300
35	/equipos	1000	-	20219.0000	-	36752.7700
36	/ligas	1	-	1453.0000	-	187.0000
37	/ligas	5	-	822.8000	-	196.6000
38	/ligas	50	-	1516.4600	-	1986.7600
39	/ligas	100	-	2857.6200	-	3966.5400
40	/ligas	500	-	9893.6200	-	18622.1200
41	/ligas	1000		18687.0400	-	36818.2200

En la tabla 27 se muestran los resultados de las pruebas de penetración realizadas a las cuatro aplicaciones agrupados por sección de la página. Para esta prueba se utilizó la herramienta OWASP ZAP.

Tabla 27 Instrumento de recopilación de datos - FORM02 Cantidad de vulnerabilidades.

FORM02	**Procedimiento (03): Prueba de Penetración**				
Indicador Cantidad de vulnerabilidades encontradas					
Herramientas: OWASP ZAP					
N.º	Sección	MS1 (unidades)	MS2 (unidades)	R1 (unidades)	R2 (unidades)
1	/talleres	3	-	4	-
2	/horario	3	-	4	-
3	/sede	3	-	4	-
4	/equipos	-	3	-	4
5	/estadio	-	3	-	4
6	/liga	-	3	-	4

En la Tabla 28 se muestran los resultados de las pruebas de penetración realizadas a las cuatro aplicaciones agrupados por sección de la página. Para esta prueba se utilizó la herramienta JMeter.

Tabla 28 Instrumento de recopilación de datos - FORM03 Porcentaje de ataques bloqueados

FORM03		**Procedimiento (03): Prueba de Penetración**				
Indicador Porcentaje de ataques bloqueados						
Herramientas: JMeter						
N.º	Cantidad Solicitudes	Sección	MS1 (%)	MS2 (%)	R1 (%)	R2 (%)
1	8000	/talleres	-	-	28.7500	31.2625
2	8000	/horario	-	-	37.9375	41.6375
3	8000	/sede	-	-	42.2625	42.5125
4	8000	/equipos	26.2125	25.5375	-	-
5	8000	/estadio	27.6500	26.1125	-	-
6	8000	/liga	27.4125	26.1250	-	-

En la Tabla 29 se muestran los resultados de las pruebas de penetración realizadas a las cuatro aplicaciones agrupados por sección de la página. Para esta prueba se utilizó la herramienta Monitor de Red del navegador Firefox.

Tabla 29 Instrumento de recopilación de datos - FORM04 Tiempo de respuesta de carga de imágenes

FORM04	**Procedimiento (04): Prueba de Carga**			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación R1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 KB-1	10 KB	1.4400	1.7400	0.3000

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación R1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 KB-2	10.1 KB	1.4600	2.2600	0.8000
10 KB-3	10.2 KB	1.5800	2.4500	0.8700
10 KB-4	10.7 KB	1.2500	2.0300	0.7800
10 KB-5	10.8 KB	1.2500	2.0300	0.7800
10 KB-6	10.5 KB	1.2500	2.0400	0.7900
10 KB-7	10.1 KB	1.4400	1.6400	0.2000
10 KB-8	10.9 KB	1.4400	1.6400	0.2000
10 KB-9	10.5 KB	1.4400	1.6500	0.2100
10 KB-10	10.7 KB	1.4400	1.7500	0.3100
10 KB-11	10.3 KB	1.4400	1.7600	0.3200
10 KB-12	10.6 KB	1.4400	1.7600	0.3200
10 KB-13	10.2 KB	1.4400	1.7800	0.3400
10 KB-14	10.6 KB	1.4400	1.8000	0.3600
10 KB-15	10.2 KB	1.4400	1.8100	0.3700
10 KB-16	10 KB	1.4400	1.8400	0.4000
10 KB-17	10.8 KB	1.4400	1.8500	0.4100
10 KB-18	10.3 KB	1.4400	1.8600	0.4200
10 KB-19	10.4 KB	1.4400	1.8800	0.4400
10 KB-20	10.3 KB	1.4400	1.8900	0.4500
1 MB-1	1.38 MB	1.4400	6.5400	5.1000
1 MB-2	1.39 MB	1.4400	6.5100	5.0700
1 MB-3	1.61 MB	1.4400	6.8700	5.4300
1 MB-4	1.67 MB	1.4400	6.9200	5.4800
1 MB-5	1.18 MB	1.4400	5.9600	4.5200

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación R1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
1 MB-6	1.13 MB	1.4400	5.9400	4.5000
1 MB-7	1.10 MB	1.4400	5.8600	4.4200
1 MB-8	1.19 MB	1.4400	6.1200	4.6800
1 MB-9	1.15 MB	1.4400	5.9200	4.4800
1 MB-10	1.53 MB	1.4400	6.8200	5.3800
1 MB-11	1.93 MB	1.4400	7.0300	5.5900
1 MB-12	1.83 MB	1.4400	7.0000	5.5600
1 MB-13	1.33 MB	1.4400	6.4800	5.0400
1 MB-14	1.60 MB	1.4400	6.9000	5.4600
1 MB-15	1.25 MB	1.4400	6.3000	4.8600
1 MB-16	1.43 MB	1.4400	6.6500	5.2100
1 MB-17	1.15 MB	1.4400	6.0000	4.5600
1 MB-18	1.9 MB	1.4400	5.7600	4.3200
1 MB-19	1.7 MB	1.4400	5.7900	4.3500
1 MB-20	1.58 MB	1.4400	6.8600	5.4200
10 MB-1	10.2 MB	1.5600	44.7800	43.2200
10 MB-2	10.8 MB	1.5700	45.8200	44.2500
10 MB-3	10.8 MB	1.5800	45.8400	44.2600
10 MB-4	10.5 MB	1.7100	45.6600	43.9500
10 MB-5	10 MB	1.3400	44.2000	42.8600
10 MB-6	10.9 MB	1.3400	45.6000	44.2600
10 MB-7	10.6 MB	1.5500	45.2300	43.6800
10 MB-8	10 MB	1.5500	43.6100	42.0600
10 MB-9	10.2 MB	1.5500	44.6000	43.0500

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación R1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 MB-10	10.2 MB	1.5500	44.7500	43.2000
10 MB-11	10.7 MB	1.5600	45.4600	43.9000
10 MB-12	10.8 MB	1.5500	45.5000	43.9500
10 MB-13	10 MB	1.5600	43.6900	42.1300
10 MB-14	10.2 MB	1.5600	44.8100	43.2500
10 MB-15	10.3 MB	1.5600	45.0000	43.4400
10 MB-16	10 MB	1.5600	44.0400	42.4800
10 MB-17	10.5 MB	1.5600	45.2600	43.7000
10 MB-18	10.3 MB	1.5600	44.8400	43.2800
10 MB-19	10.4 MB	1.5600	45.1700	43.6100
10 MB-20	10.3 MB	1.5600	45.0500	43.4900
100 KB-1	110 KB	1.5600	2.7000	1.1400
100 KB-2	101 KB	1.5600	2.7000	1.1400
100 KB-3	107 KB	1.5600	2.6900	1.1300
100 KB-4	106 KB	1.5600	2.7000	1.1400
100 KB-5	105 KB	1.5600	2.7000	1.1400
100 KB-6	119 KB	1.5600	2.7000	1.1400
100 KB-7	118 KB	1.5600	2.7000	1.1400
100 KB-8	122 KB	1.5600	2.7000	1.1400
100 KB-9	151 KB	1.5600	2.8000	1.2400
100 KB-10	152 KB	1.5600	2.8100	1.2500
100 KB-11	189 KB	1.5600	2.8600	1.3000
100 KB-12	154 KB	1.5600	2.7900	1.2300
100 KB-13	140 KB	1.5600	2.8100	1.2500

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación R1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
100 KB-14	111 KB	1.5600	2.7000	1.1400
100 KB-15	100 KB	1.5600	2.7000	1.1400
100 KB-16	157 KB	1.5600	2.8000	1.2400
100 KB-17	168 KB	1.5600	2.8700	1.3100
100 KB-18	127 KB	1.5600	2.7000	1.1400
100 KB-19	112 KB	1.5600	2.7000	1.1400
100 KB-20	147 KB	1.5600	2.8100	1.2500
100 MB-1	100 MB	2.0100	118.8000	116.7900
100 MB-2	102 MB	2.0100	171.0000	168.9900
100 MB-3	105 MB	0.9530	172.8000	171.8470
100 MB-4	111 MB	0.9590	219.0000	218.0410
100 MB-5	112 MB	0.9610	194.4000	193.4390
100 MB-6	114 MB	0.9630	133.2000	132.2370
100 MB-7	115 MB	118.2000	259.2000	141.0000
100 MB-8	116 MB	133.2000	302.4000	169.2000
100 MB-9	117 MB	107.4000	313.8000	206.4000
100 MB-10	118 MB	172.8000	327.6000	154.8000
100 MB-11	120 MB	194.4000	369.0000	174.6000
100 MB-12	122 MB	219.0000	390.0000	171.0000
100 MB-13	123 MB	259.2000	412.2000	153.0000
100 MB-14	127 MB	302.4000	540.0000	237.6000
100 MB-15	130 MB	313.2000	525.0000	211.8000
100 MB-16	146 MB	327.0000	536.4000	209.4000
100 MB-17	147 MB	369.0000	586.2000	217.2000

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación R1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
100 MB-18	164 MB	390.0000	571.8000	181.8000
100 MB-19	165 MB	412.2000	597.6000	185.4000
100 MB-20	179 MB	525.0000	619.2000	94.2000

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación MS1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 KB-1	10 KB	1.4700	1.7700	0.3000
10 KB-2	10.1 KB	1.4700	2.0900	0.6200
10 KB-3	10.2 KB	1.4800	2.0900	0.6100
10 KB-4	10.7 KB	1.4900	2.1100	0.6200
10 KB-5	10.8 KB	1.6100	2.3900	0.7800
10 KB-6	10.5 KB	1.3300	1.9200	0.5900
10 KB-7	10.1 KB	1.4700	1.6800	0.2100
10 KB-8	10.9 KB	1.4700	1.6800	0.2100
10 KB-9	10.5 KB	1.4700	1.6900	0.2200
10 KB-10	10.7 KB	1.4700	1.7700	0.3000
10 KB-11	10.3 KB	1.4700	1.7700	0.3000
10 KB-12	10.6 KB	1.4700	1.7700	0.3000
10 KB-13	10.2 KB	1.4700	1.8000	0.3300
10 KB-14	10.6 KB	1.4700	1.8100	0.3400
10 KB-15	10.2 KB	1.4700	1.8200	0.3500

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación MS1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 KB-16	10 KB	1.4700	1.8300	0.3600
10 KB-17	10.8 KB	1.4700	1.9000	0.4300
10 KB-18	10.3 KB	1.4700	1.9000	0.4300
10 KB-19	10.4 KB	1.4700	1.9100	0.4400
10 KB-20	10.3 KB	1.4700	1.9100	0.4400
1 MB-1	1.38 MB	1.4700	6.8200	5.3500
1 MB-2	1.39 MB	1.4700	6.8400	5.3700
1 MB-3	1.61 MB	1.4700	7.0100	5.5400
1 MB-4	1.67 MB	1.4700	7.0500	5.5800
1 MB-5	1.18 MB	1.4700	6.4900	5.0200
1 MB-6	1.13 MB	1.5100	6.4000	4.8900
1 MB-7	1.10 MB	1.5200	6.3200	4.8000
1 MB-8	1.19 MB	1.5400	6.5300	4.9900
1 MB-9	1.15 MB	1.5400	5.6400	4.1000
1 MB-10	1.53 MB	1.5400	6.6800	5.1400
1 MB-11	1.93 MB	1.5400	7.0300	5.4900
1 MB-12	1.83 MB	1.5500	6.9700	5.4200
1 MB-13	1.33 MB	1.5500	6.0700	4.5200
1 MB-14	1.60 MB	1.5500	6.7700	5.2200
1 MB-15	1.25 MB	1.5500	6.0300	4.4800
1 MB-16	1.43 MB	1.5500	6.5300	4.9800
1 MB-17	1.15 MB	1.5500	6.6300	5.0800
1 MB-18	1.9 MB	1.5500	5.3900	3.8400
1 MB-19	1.7 MB	1.5500	5.3700	3.8200
1 MB-20	1.58 MB	1.5500	6.7800	5.2300

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación MS1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 MB-1	10.2 MB	1.2300	47.3900	46.1600
10 MB-2	10.8 MB	1.3500	48.4100	47.0600
10 MB-3	10.8 MB	0.9990	48.0500	47.0510
10 MB-4	10.5 MB	1.0000	47.5100	46.5100
10 MB-5	10 M B	1.0100	46.8100	45.8000
10 MB-6	10. 9MB	1.0100	48.0700	47.0600
10 MB-7	10.6 MB	1.2100	47.7700	46.5600
10 MB-8	10 MB	1.2100	46.3700	45.1600
10 MB-9	10.2 MB	1.2100	46.5700	45.3600
10 MB-10	10.2 MB	1.2100	47.0800	45.8700
10 MB-11	10.7 MB	1.2100	47.9400	46.7300
10 MB-12	10.8 MB	1.2100	47.9800	46.7700
10 MB-13	10 MB	1.2100	46.3300	45.1200
10 MB-14	10.2 MB	1.2100	47.2300	46.0200
10 MB-15	10.3 MB	1.2100	47.4400	46.2300
10 MB-16	10 MB	1.2100	46.4600	45.2500
10 MB-17	10.5 MB	1.2100	47.7800	46.5700
10 MB-18	10.3 MB	1.2100	47.4700	46.2600
10 MB-19	10.4 MB	1.2100	47.5900	46.3800
10 MB-20	10.3 MB	1.2100	47.5700	46.3600
100 KB-1	110 KB	1.2100	2.3800	1.1700
100 KB-2	101 KB	1.2300	2.3900	1.1600
100 KB-3	107 KB	1.2300	2.3700	1.1400
100 KB-4	106 KB	1.2300	2.3700	1.1400
100 KB-5	105 KB	1.2300	2.3700	1.1400

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación MS1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
100 KB-6	119 KB	1.2300	2.3700	1.1400
100 KB-7	118 KB	1.2300	2.3700	1.1400
100 KB-8	122 KB	1.2300	2.3800	1.1500
100 KB-9	151 KB	1.2300	2.4700	1.2400
100 KB-10	152 KB	1.2300	2.4700	1.2400
100 KB-11	189 KB	1.2300	2.6300	1.4000
100 KB-12	154 KB	1.2300	2.4700	1.2400
100 KB-13	140 KB	1.2400	2.4700	1.2300
100 KB-14	111 KB	1.2400	2.3800	1.1400
100 KB-15	100 KB	1.2400	2.3800	1.1400
100 KB-16	157 KB	1.2400	2.4800	1.2400
100 KB-17	168 KB	1.2400	2.6400	1.4000
100 KB-18	127 KB	1.2800	2.3900	1.1100
100 KB-19	112 KB	1.2400	2.3900	1.1500
100 KB-20	147 KB	1.2500	2.4900	1.2400
100 MB-1	100 MB	1.9800	183.0000	181.0200
100 MB-2	102 MB	1.9700	120.0000	118.0300
100 MB-3	105 MB	1.0100	133.8000	132.7900
100 MB-4	111 MB	1.0200	159.0000	157.9800
100 MB-5	112 MB	1.0200	202.2000	201.1800
100 MB-6	114 MB	1.0300	198.0000	196.9700
100 MB-7	115 MB	119.4000	265.8000	146.4000
100 MB-8	116 MB	133.8000	283.8000	150.0000
100 MB-9	117 MB	159.0000	331.8000	172.8000
100 MB-10	118 MB	182.4000	343.2000	160.8000

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación MS1 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
100 MB-11	120 MB	198.0000	336.6000	138.6000
100 MB-12	122 MB	202.2000	414.6000	212.4000
100 MB-13	123 MB	265.2000	411.6000	146.4000
100 MB-14	127 MB	283.2000	542.4000	259.2000
100 MB-15	130 MB	331.8000	531.6000	199.8000
100 MB-16	146 MB	336.6000	519.6000	183.0000
100 MB-17	147 MB	343.2000	511.8000	168.6000
100 MB-18	164 MB	531.6000	585.6000	54.0000
100 MB-19	165 MB	414.6000	571.2000	156.6000
100 MB-20	179 MB	511.8000	601.2000	89.4000

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación R2 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 KB-1	10 KB	1.2600	1.6200	0.3600
10 KB-2	10.1 KB	0.9800	1.0800	0.1000
10 KB-3	10.2 KB	1.2900	1.9000	0.6100
10 KB-4	10.7 KB	1.3000	1.9100	0.6100
10 KB-5	10.8 KB	0.9840	1.0800	0.0960
10 KB-6	10.5 KB	1.3400	1.9400	0.6000
10 KB-7	10.1 KB	1.3400	1.9400	0.6000
10 KB-8	10.9 KB	1.3400	1.9500	0.6100
10 KB-9	10.5 KB	1.3000	1.9100	0.6100

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación R2 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 KB-10	10.7 KB	1.3400	1.9500	0.6100
10 KB-11	10.3 KB	0.9890	1.0900	0.1010
10 KB-12	10.6 KB	1.3500	1.9500	0.6000
10 KB-13	10.2 KB	1.3500	1.9500	0.6000
10 KB-14	10.6 KB	0.9920	1.1000	0.1080
10 KB-15	10.2 KB	1.3000	1.9100	0.6100
10 KB-16	10 KB	0.9940	1.1100	0.1160
10 KB-17	10.8 KB	1.3100	1.9200	0.6100
10 KB-18	10.3 KB	1.3500	2.2200	0.8700
10 KB-19	10.4 KB	1.3100	1.9200	0.6100
10 KB-20	10.3 KB	1.3100	1.9200	0.6100
1 MB-1	1.38 MB	0.9990	3.8800	2.8810
1 MB-2	1.39 MB	1.0000	4.6200	3.6200
1 MB-3	1.61 MB	1.3200	6.1300	4.8100
1 MB-4	1.67 MB	1.3500	6.2300	4.8800
1 MB-5	1.18 MB	1.3600	5.7800	4.4200
1 MB-6	1.13 MB	1.3600	5.7200	4.3600
1 MB-7	1.10 MB	1.3300	5.3200	3.9900
1 MB-8	1.19 MB	1.3200	5.6500	4.3300
1 MB-9	1.15 MB	1.0100	3.6000	2.5900
1 MB-10	1.53 MB	1.0100	5.2400	4.2300
1 MB-11	1.93 MB	1.0900	6.0500	4.9600
1 MB-12	1.83 MB	1.3600	6.2800	4.9200
1 MB-13	1.33 MB	1.3100	5.8100	4.5000
1 MB-14	1.60 MB	1.3300	6.1400	4.8100

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación R2 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
1 MB-15	1.25 MB	1.3700	5.9500	4.5800
1 MB-16	1.43 MB	1.3100	5.9400	4.6300
1 MB-17	1.15 MB	1.3100	5.3700	4.0600
1 MB-18	1.9 MB	1.1400	4.1800	3.0400
1 MB-19	1.7 MB	1.3800	5.6000	4.2200
1 MB-20	1.58 MB	1.2500	5.8000	4.5500
10 MB-1	10.2 MB	0.2670	41.3000	41.0330
10 MB-2	10.8 MB	0.2720	44.5200	44.2480
10 MB-3	10.8 MB	0.2740	44.5000	44.2260
10 MB-4	10.5 MB	0.2780	43.6900	43.4120
10 MB-5	10 MB	0.2870	42.4800	42.1930
10 MB-6	10.9 MB	0.3460	44.5800	44.2340
10 MB-7	10.6 MB	0.3500	43.9400	43.5900
10 MB-8	10 MB	0.3700	42.5300	42.1600
10 MB-9	10.2 MB	0.3730	43.1500	42.7770
10 MB-10	10.2 MB	0.3780	43.4100	43.0320
10 MB-11	10.7 MB	0.3980	44.4800	44.0820
10 MB-12	10.8 MB	0.4310	44.5400	44.1090
10 MB-13	10 MB	0.4330	42.5200	42.0870
10 MB-14	10.2 MB	0.4360	43.4600	43.0240
10 MB-15	10.3 MB	0.4390	43.6500	43.2110
10 MB-16	10 MB	0.4410	42.7100	42.2690
10 MB-17	10.5 MB	0.4430	43.8900	43.4470
10 MB-18	10.3 MB	0.4460	43.5600	43.1140
10 MB-19	10.4 MB	0.4480	43.8600	43.4120

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación R2 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 MB-20	10.3 MB	0.4490	43.6800	43.2310
100 KB-1	110 KB	0.4510	1.2600	0.8090
100 KB-2	101 KB	0.4530	1.2600	0.8070
100 KB-3	107 KB	0.4540	1.2700	0.8160
100 KB-4	106 KB	0.5540	1.3700	0.8160
100 KB-5	105 KB	0.5580	1.4100	0.8520
100 KB-6	119 KB	0.5660	1.3600	0.7940
100 KB-7	118 KB	0.5730	1.3700	0.7970
100 KB-8	122 KB	0.5800	1.3800	0.8000
100 KB-9	151 KB	0.5900	1.4200	0.8300
100 KB-10	152 KB	0.5960	1.4200	0.8240
100 KB-11	189 KB	0.5990	1.4400	0.8410
100 KB-12	154 KB	0.6640	1.5700	0.9060
100 KB-13	140 KB	0.6720	1.5600	0.8880
100 KB-14	111 KB	0.6750	1.4800	0.8050
100 KB-15	100 KB	0.6780	1.4800	0.8020
100 KB-16	157 KB	0.6810	1.5600	0.8790
100 KB-17	168 KB	0.8480	1.8700	1.0220
100 KB-18	127 KB	0.8540	1.7700	0.9160
100 KB-19	112 KB	0.8550	1.8700	1.0150
100 KB-20	147 KB	0.8570	1.8800	1.0230
100 MB-1	100 MB	1.2600	153.6000	152.3400
100 MB-2	102 MB	1.2600	153.6000	152.3400
100 MB-3	105 MB	1.2600	157.2000	155.9400
100 MB-4	111 MB	1.2600	179.4000	178.1400

FORM04	Procedimiento (04): Prueba de Carga
Indicador Tiempo de respuesta de carga de imágenes	
Herramientas: Monitor de Red del navegador Firefox	

Aplicación R2 – Nombre de imagen	Peso	Tiempo de Inicio (s)	Tiempo de Fin (s)	Tiempo Final de Carga (s)
100 MB-5	112 MB	1.2600	213.6000	212.3400
100 MB-6	114 MB	1.2600	244.2000	242.9400
100 MB-7	115 MB	127.2000	398.4000	271.2000
100 MB-8	116 MB	152.4000	454.2000	301.8000
100 MB-9	117 MB	156.6000	466.8000	310.2000
100 MB-10	118 MB	178.2000	493.8000	315.6000
100 MB-11	120 MB	213.0000	505.2000	292.2000
100 MB-12	122 MB	243.0000	514.8000	271.8000
100 MB-13	123 MB	392.4000	629.4000	237.0000
100 MB-14	127 MB	453.6000	697.2000	243.6000
100 MB-15	130 MB	466.8000	648.0000	181.2000
100 MB-16	146 MB	493.2000	660.6000	167.4000
100 MB-17	147 MB	505.2000	703.2000	198.0000
100 MB-18	164 MB	514.8000	694.2000	179.4000
100 MB-19	165 MB	629.4000	781.8000	152.4000
100 MB-20	179 MB	648.0000	745.8000	97.8000

FORM04	Procedimiento (04): Prueba de Carga
Indicador Tiempo de respuesta de carga de imágenes	
Herramientas: Monitor de Red del navegador Firefox	

Aplicación MS2 – Nombre de imagen	Peso	Tiempo de Inicio (s)	Tiempo de Fin (s)	Tiempo Final de Carga (s)
10 KB-1	10 KB	1.4800	1.9300	0.4500
10 KB-2	10.1 KB	1.4200	1.5200	0.1000
10 KB-3	10.2 KB	1.4800	1.9300	0.4500

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación MS2 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 KB-4	10.7 KB	1.4800	1.9300	0.4500
10 KB-5	10.8 KB	1.4200	1.5200	0.1000
10 KB-6	10.5 KB	1.5400	1.9600	0.4200
10 KB-7	10.1 KB	1.5600	2.0200	0.4600
10 KB-8	10.9 KB	1.5700	1.9200	0.3500
10 KB-9	10.5 KB	1.4900	1.9300	0.4400
10 KB-10	10.7 KB	1.5700	1.9200	0.3500
10 KB-11	10.3 KB	1.4300	1.5300	0.1000
10 KB-12	10.6 KB	1.5700	1.9300	0.3600
10 KB-13	10.2 KB	1.5900	1.9300	0.3400
10 KB-14	10. 6KB	1.4300	1.5400	0.1100
10 KB-15	10.2 KB	1.4900	1.9400	0.4500
10 KB-16	10 KB	1.4300	1.5500	0.1200
10 KB-17	10.8 KB	1.4900	1.9400	0.4500
10 KB-18	10.3 KB	1.6000	1.9400	0.3400
10 KB-19	10.4 KB	1.4900	1.9400	0.4500
10 KB-20	10.3 KB	1.5100	1.9400	0.4300
1 MB-1	1.38 MB	1.4400	5.4100	3.9700
1 MB-2	1.39 MB	1.4400	5.9100	4.4700
1 MB-3	1.61 MB	1.5100	1.9300	0.4200
1 MB-4	1.67 MB	1.4800	1.9300	0.4500
1 MB-5	1.18 MB	1.6600	5.4400	3.7800
1 MB-6	1.13 MB	1.6600	5.3200	3.6600
1 MB-7	1.10 MB	1.5400	5.7500	4.2100
1 MB-8	1.19 MB	1.5200	5.9600	4.4400

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación MS2 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
1 MB-9	1.15 MB	1.4400	5.1600	3.7200
1 MB-10	1.53 MB	1.4600	6.2000	4.7400
1 MB-11	1.93 MB	1.4600	6.5400	5.0800
1 MB-12	1.83 MB	1.6700	6.6100	4.9400
1 MB-13	1.33 MB	1.5000	6.1000	4.6000
1 MB-14	1.60 MB	1.5200	6.4800	4.9600
1 MB-15	1.25 MB	1.6700	5.7300	4.0600
1 MB-16	1.43 MB	1.5000	6.2900	4.7900
1 MB-17	1.15 MB	1.5000	5.8400	4.3400
1 MB-18	1.9 MB	1.4700	5.4200	3.9500
1 MB-19	1.7 MB	1.6800	5.0800	3.4000
1 MB-20	1.58 MB	1.4700	6.4400	4.9700
10 MB-1	10.2 MB	1.4200	45.2100	43.7900
10 MB-2	10.8 MB	1.4200	47.1300	45.7100
10 MB-3	10.8 MB	1.4300	47.1200	45.6900
10 MB-4	10.5 MB	1.4300	46.7400	45.3100
10 MB-5	10 MB	1.4300	45.7700	44.3400
10 MB-6	10.9 MB	1.4400	47.1600	45.7200
10 MB-7	10.6 MB	1.4400	46.8900	45.4500
10 MB-8	10 MB	1.4500	45.7100	44.2600
10 MB-9	10.2 MB	1.4500	46.0000	44.5500
10 MB-10	10.2 MB	1.4500	46.2400	44.7900
10 MB-11	10.7 MB	1.4600	47.1300	45.6700
10 MB-12	10.8 MB	1.4600	47.1400	45.6800
10 MB-13	10 MB	1.4600	45.7600	44.3000

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación MS2 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
10 MB-14	10.2 MB	1.4800	46.4000	44.9200
10 MB-15	10.3 MB	1.4800	46.6000	45.1200
10 MB-16	10 MB	1.4900	45.9000	44.4100
10 MB-17	10.5 MB	1.4900	46.8400	45.3500
10 MB-18	10.3 MB	1.4900	46.4700	44.9800
10 MB-19	10.4 MB	1.4900	46.7900	45.3000
10 MB-20	10.3 MB	1.5000	46.6900	45.1900
100 KB-1	110 KB	1.5000	2.5300	1.0300
100 KB-2	101 KB	1.5000	2.5300	1.0300
100 KB-3	107 KB	1.5000	2.5300	1.0300
100 KB-4	106 KB	1.5100	2.5300	1.0200
100 KB-5	105 KB	1.5100	2.5300	1.0200
100 KB-6	119 KB	1.5100	2.5300	1.0200
100 KB-7	118 KB	1.5100	2.6000	1.0900
100 KB-8	122 KB	1.5200	2.6000	1.0800
100 KB-9	151 KB	1.5200	2.6600	1.1400
100 KB-10	152 KB	1.5300	2.6600	1.1300
100 KB-11	189 KB	1.5300	2.7200	1.1900
100 KB-12	154 KB	1.5500	2.6700	1.1200
100 KB-13	140 KB	1.5600	2.5400	0.9800
100 KB-14	111 KB	1.5600	2.2700	0.7100
100 KB-15	100 KB	1.5700	2.2700	0.7000
100 KB-16	157 KB	1.5700	2.5300	0.9600
100 KB-17	168 KB	1.5700	2.6000	1.0300
100 KB-18	127 KB	1.5800	2.5300	0.9500

FORM04	Procedimiento (04): Prueba de Carga			
Indicador Tiempo de respuesta de carga de imágenes				
Herramientas: Monitor de Red del navegador Firefox				
Aplicación MS2 – Nombre de imagen	**Peso**	**Tiempo de Inicio (s)**	**Tiempo de Fin (s)**	**Tiempo Final de Carga (s)**
100 KB-19	112 KB	1.5800	2.5300	0.9500
100 KB-20	147 KB	1.5800	2.5300	0.9500
100 MB-1	100 MB	2.0600	154.8000	152.7400
100 MB-2	102 MB	2.0600	134.4000	132.3400
100 MB-3	105 MB	2.0700	157.8000	155.7300
100 MB-4	111 MB	2.0700	205.2000	203.1300
100 MB-5	112 MB	2.0700	162.6000	160.5300
100 MB-6	114 MB	0.9960	189.6000	188.6040
100 MB-7	115 MB	133.2000	262.8000	129.6000
100 MB-8	116 MB	153.6000	327.6000	174.0000
100 MB-9	117 MB	157.2000	338.4000	181.2000
100 MB-10	118 MB	161.4000	309.6000	148.2000
100 MB-11	120 MB	189.6000	333.0000	143.4000
100 MB-12	122 MB	204.0000	377.4000	173.4000
100 MB-13	123 MB	262.8000	415.8000	153.0000
100 MB-14	127 MB	309.6000	519.0000	209.4000
100 MB-15	130 MB	327.6000	527.4000	199.8000
100 MB-16	146 MB	333.0000	515.4000	182.4000
100 MB-17	147 MB	338.4000	534.6000	196.2000
100 MB-18	164 MB	377.4000	588.0000	210.6000
100 MB-19	165 MB	415.8000	592.8000	177.0000
100 MB-20	179 MB	515.4000	686.4000	171.0000

En la Tabla 30 se muestran los resultados de las pruebas de penetración realizadas a las cuatro aplicaciones agrupados por sección de la página. Para esta prueba se utilizó la herramienta JMeter, OWASP ZAP y Netdata.

Tabla 30 Instrumento de recopilación de datos - FORM05 Uso de CPU

FORM05		**Procedimiento (04): Prueba de carga**			
Indicador Uso de CPU					
Herramientas: JMeter, OWASP ZAP, Netdata					
N.º	Prueba realizada	MS1 (%)	MS2 (%)	RS1 (%)	RS2 (%)
1	Tiempo de respuesta usuario - 01	13.40%	6.00%	14.00%	7.10%
2	Tiempo de respuesta usuario - 05	17.80%	24.30%	18.60%	9.90%
3	Tiempo de respuesta usuario - 50	29.10%	30.30%	24.60%	18.30%
4	Tiempo de respuesta usuario - 100	47.40%	47.10%	34.60%	25.80%
5	Tiempo de respuesta usuario - 500	63.10%	60.50%	47.10%	58.40%
6	Tiempo de respuesta usuario - 1000	56.70%	56.50%	46.00%	57.80%
7	Tiempo de respuesta carga de imágenes	56.30%	55.30%	52.30%	61.90%
8	Cantidad de vulnerabilidades	73.50%	71.50%	72.60%	74.40%
9	Ataques bloqueados	43.10%	42.30%	58.70%	51.10%

En la Tabla 31 se muestran los resultados de las pruebas de penetración realizadas a las cuatro aplicaciones agrupados por sección de la página. Para esta prueba se utilizó la herramienta JMeter, OWASP ZAP, Netdata.

Tabla 31 Instrumento de recopilación de datos - FORM06 Uso de memoria RAM

FORM06		**Procedimiento (04): Prueba de carga**			
Indicador Uso de memoria RAM					
Herramientas: JMeter, OWASP ZAP, Netdata					
N.º	Prueba realizada	MS1 (MB)	MS2 (MB)	RS1 (MB)	RS2 (MB)
1	Tiempo de respuesta usuario - 01	0.04	0.02	0.02	0.02
2	Tiempo de respuesta usuario - 05	0.04	0.04	0.02	0.03
3	Tiempo de respuesta usuario - 50	0.04	0.03	0.05	0.04
4	Tiempo de respuesta usuario - 100	0.05	0.04	0.06	0.06
5	Tiempo de respuesta usuario - 500	0.11	0.18	0.16	0.14
6	Tiempo de respuesta usuario - 1000	0.14	0.35	0.29	0.19
7	Tiempo de respuesta carga de imágenes	1.26	1.31	1.35	1.81
8	Cantidad de vulnerabilidades	0.33	0.38	0.36	0.37
9	Ataques bloqueados	1.08	1.15	2.21	1.94

Anexo 7 Aplicaciones utilizadas para las pruebas

La figura 6 muestra la arquitectura de microservicios utilizada para la aplicación de prueba MS1.

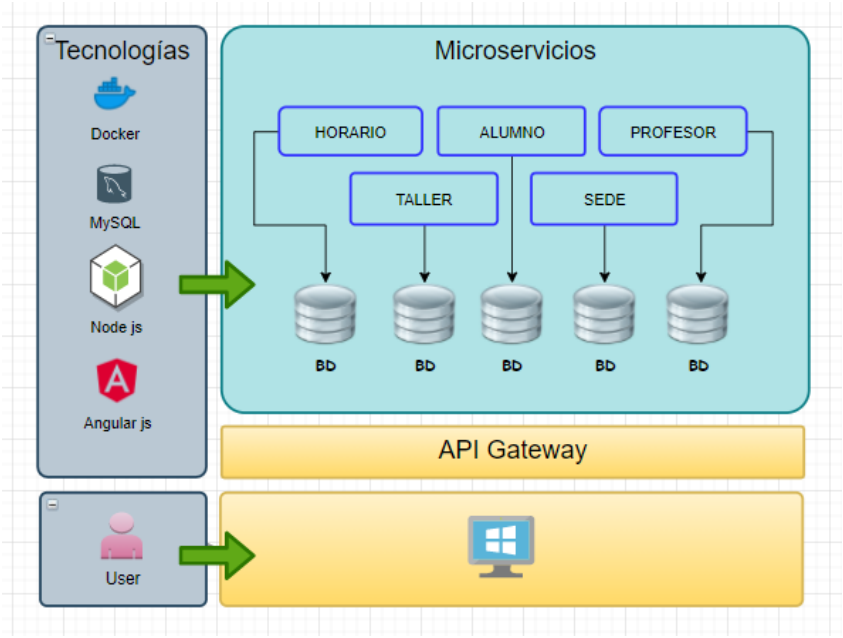


Figura 6 Arquitectura de aplicación MS1 - Microservicios

La figura 7 muestra la arquitectura de microservicios utilizada para la aplicación de prueba MS2.

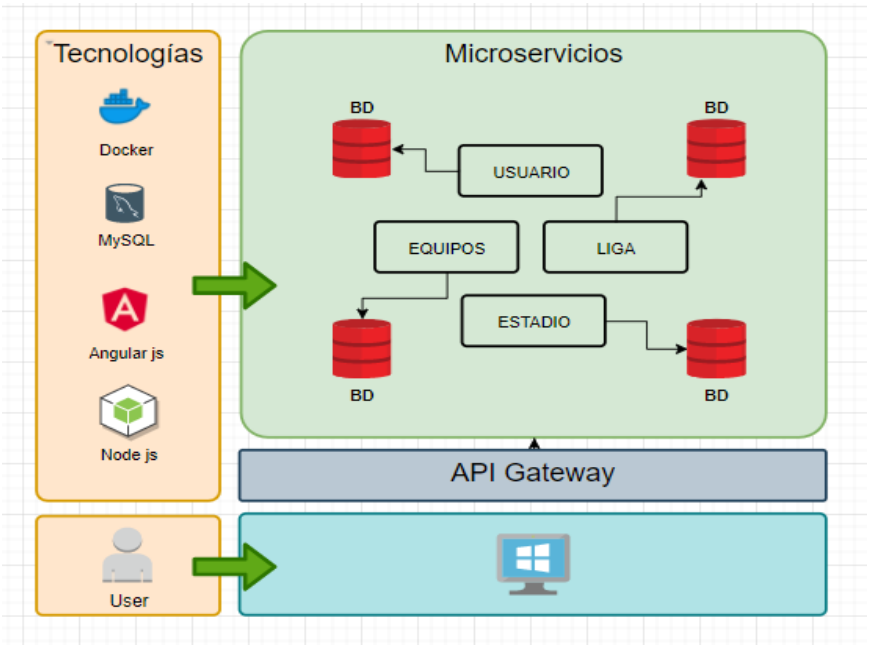


Figura 7 Arquitectura de aplicación MS2 - Microservicios

La figura 8 muestra la arquitectura REST utilizada para la aplicación de prueba R1.

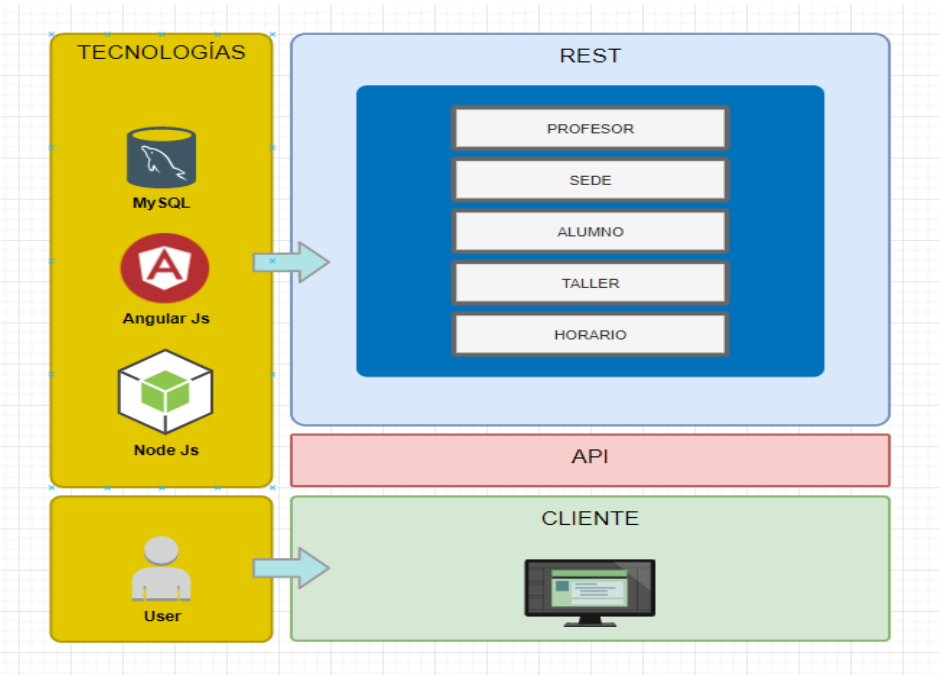


Figura 8 Arquitectura de aplicación R1 - REST.

La figura 9 muestra la arquitectura REST utilizada para la aplicación de prueba R2.

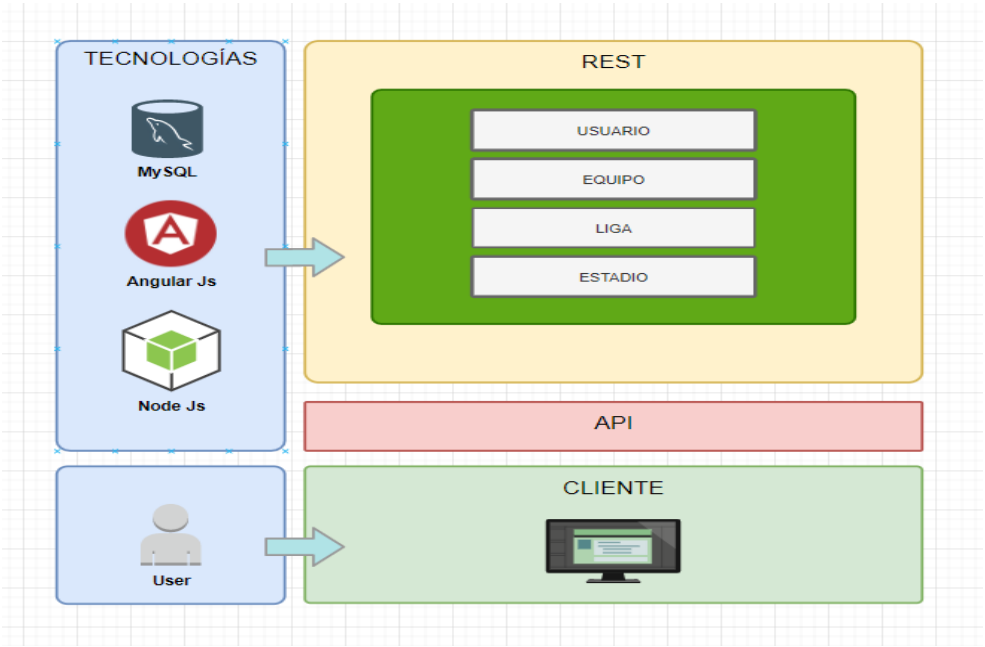


Figura 9 Arquitectura de aplicación R2 - REST.

La figura 10 muestra la pantalla de ingreso a la aplicación, la que es similar para las cuatro aplicaciones utilizadas.

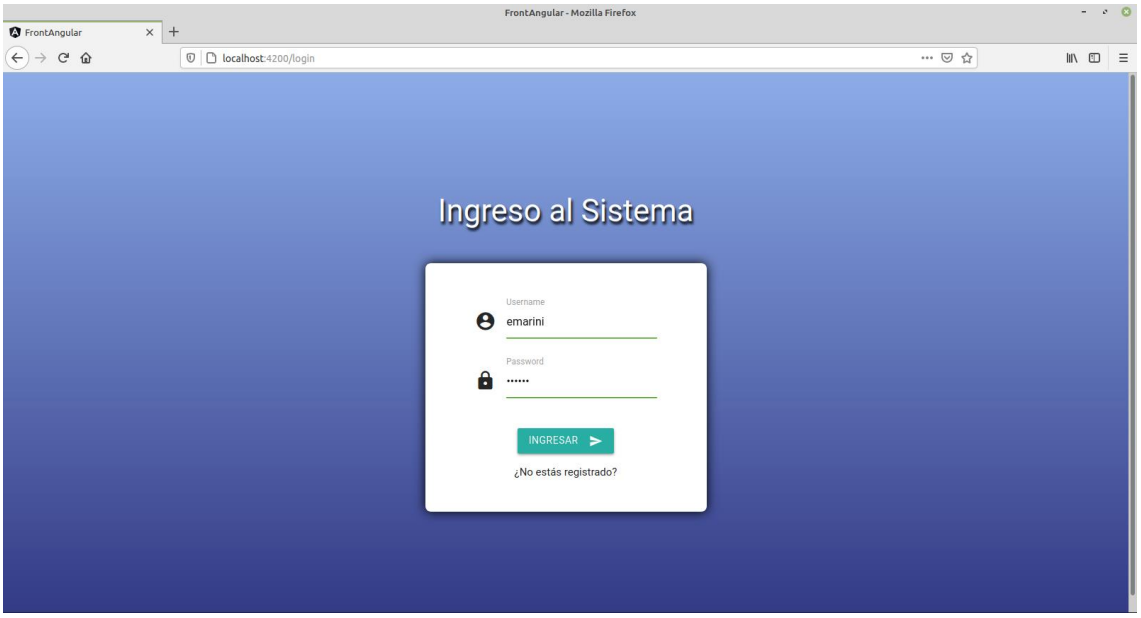


Figura 10 Pantalla de ingreso al sistema de aplicación MS1, MS2, R1 y R2.

La figura 11 muestra la página home presente en las aplicaciones de prueba R1 y MS1.

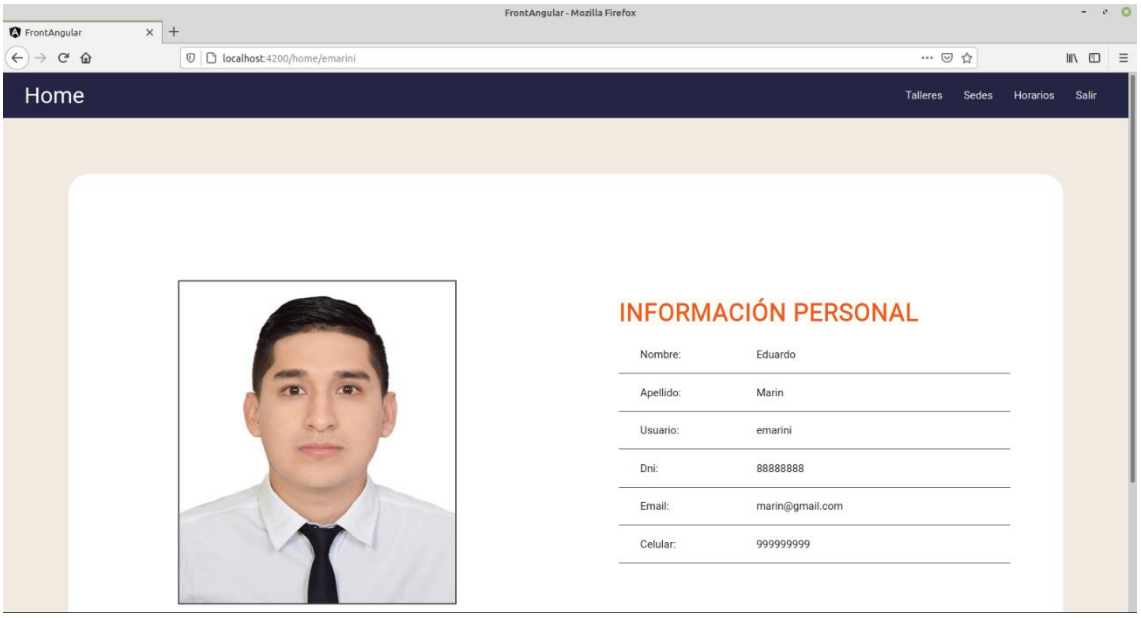


Figura 11 Sección Home de la aplicación R1 y MS1.

La figura 12 muestra la página talleres presente en las aplicaciones de prueba R1 y MS1.

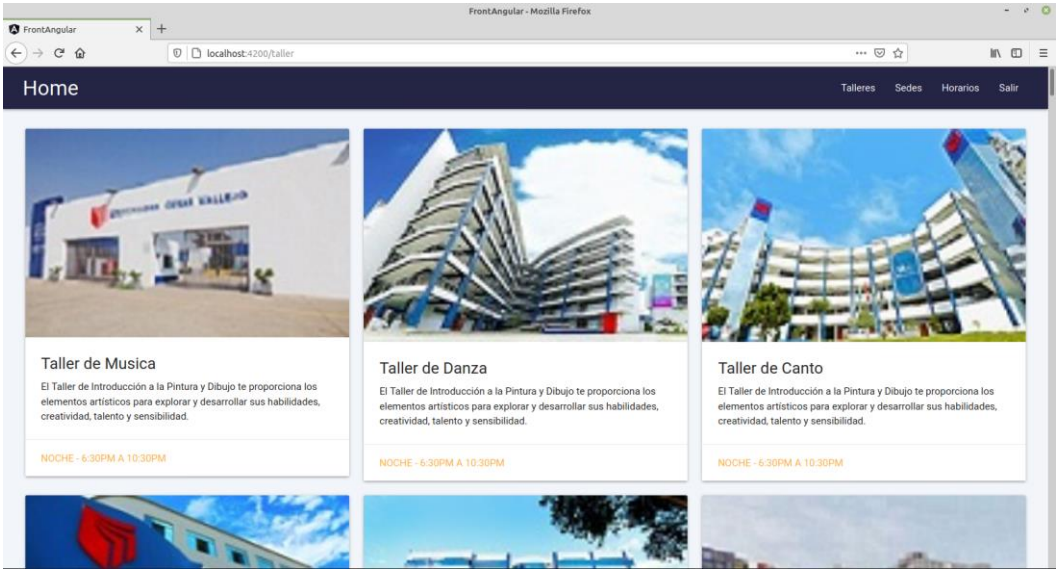


Figura 12 Sección Talleres de la aplicación R1 y MS1.

La figura 13 muestra la página Sedes, que está presente en las aplicaciones de prueba R1 y MS1.

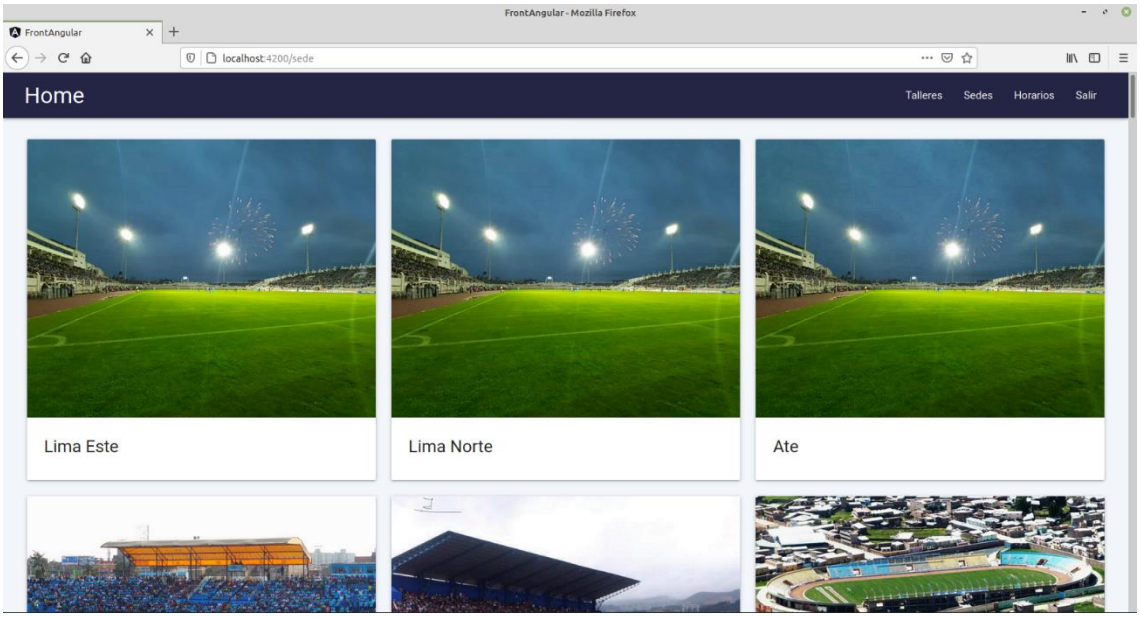


Figura 13 Sección Sedes de la aplicación R1 y MS1.

La figura 14 muestra la página de horarios presente en las aplicaciones de prueba R1 y MS1.

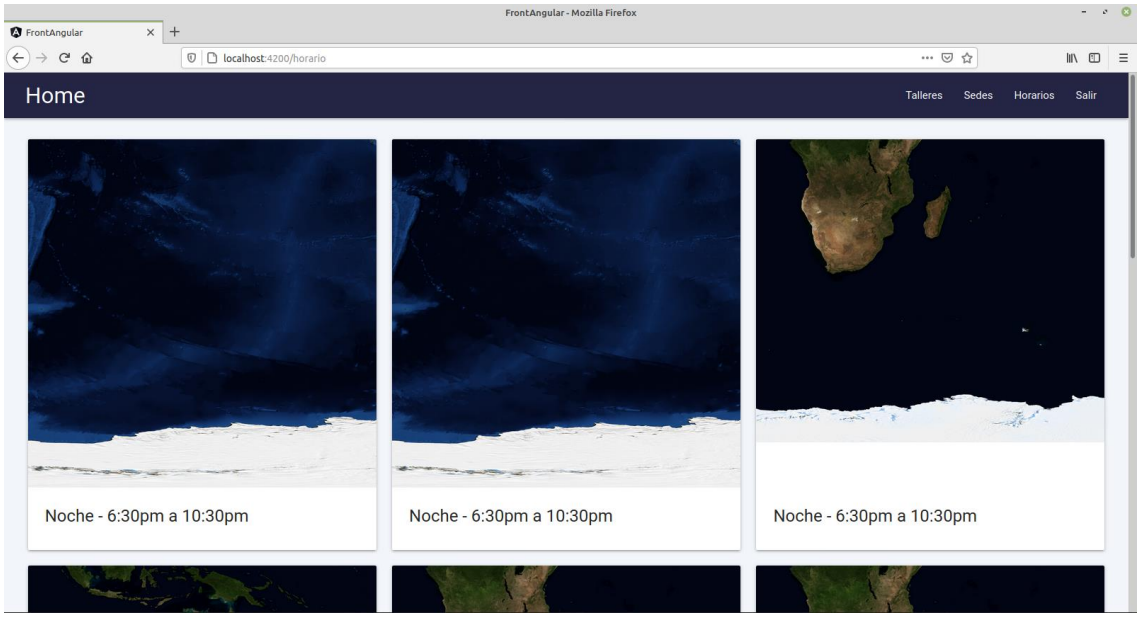


Figura 14 Sección Horarios de la aplicación R1 y MS1.

La figura 15 muestra la página home presente en las aplicaciones de prueba R2 y MS2.

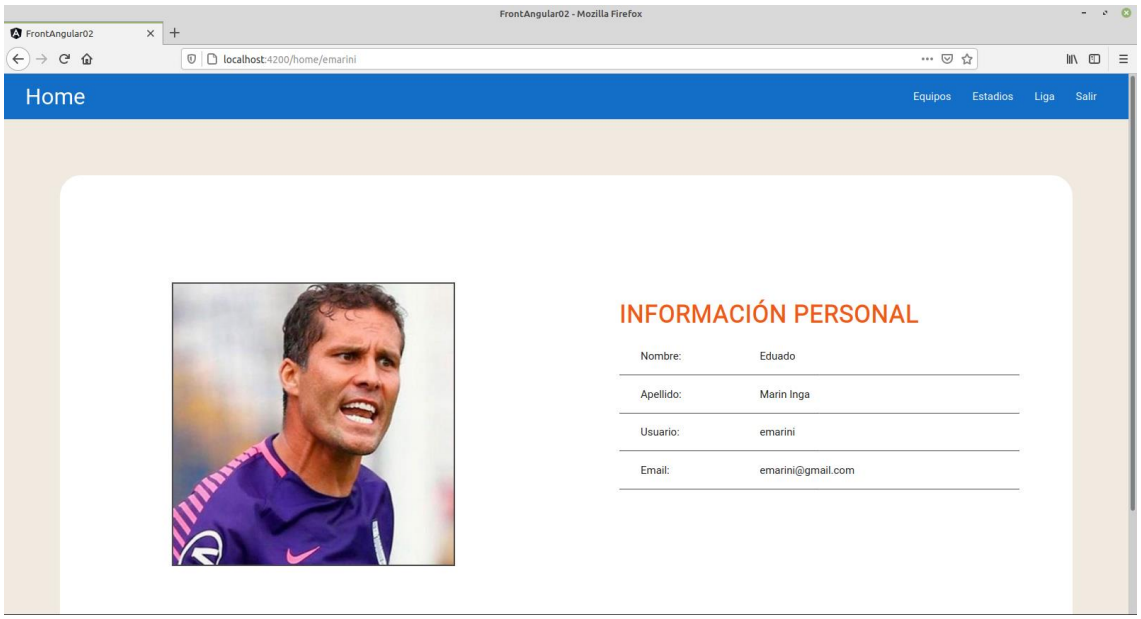


Figura 15 Sección Home de la aplicación R2 y MS2.

La figura 16 muestra la página de equipos presente en las aplicaciones de prueba R2 y MS2.

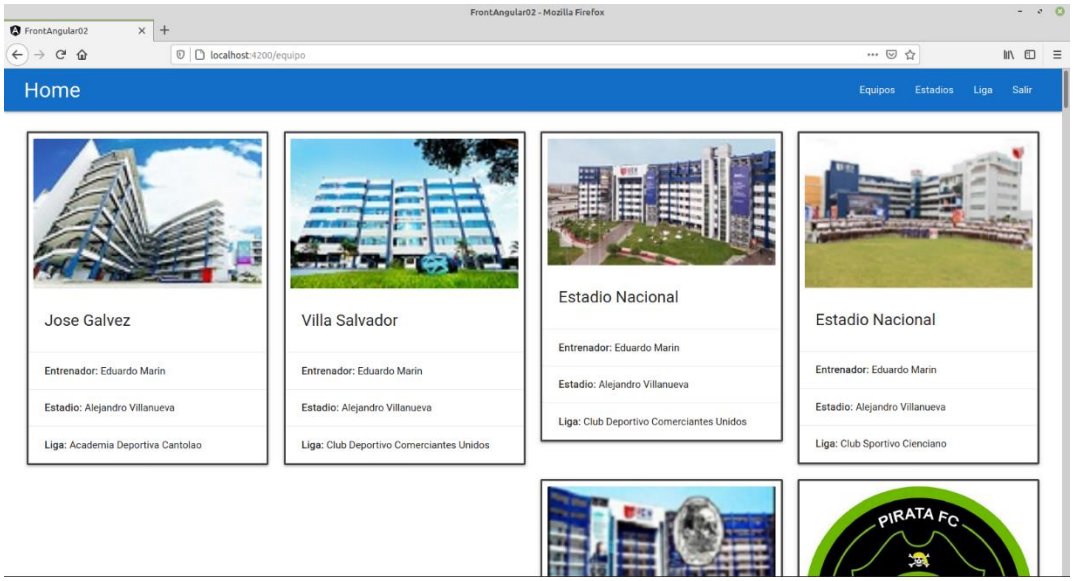


Figura 16 Sección Equipos de la aplicación R2 y MS2.

La figura 17 muestra la página de estadios presente en las aplicaciones de prueba R2 y MS2.

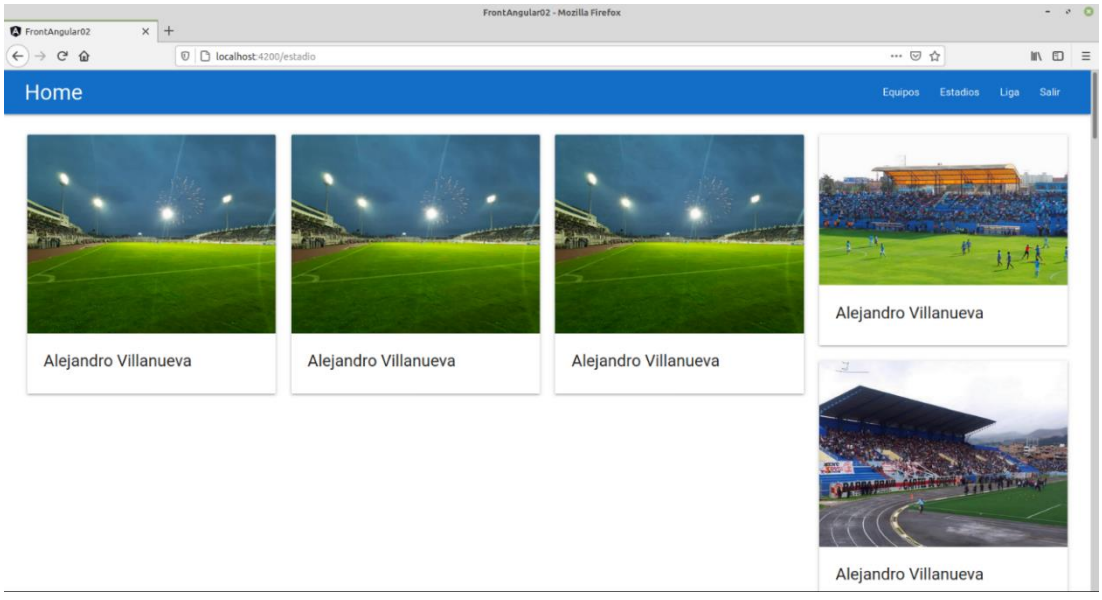


Figura 17 Sección Estadios de la aplicación R2 y MS2.

La figura 18 muestra la página de liga presente en las aplicaciones de prueba R2 y MS2.

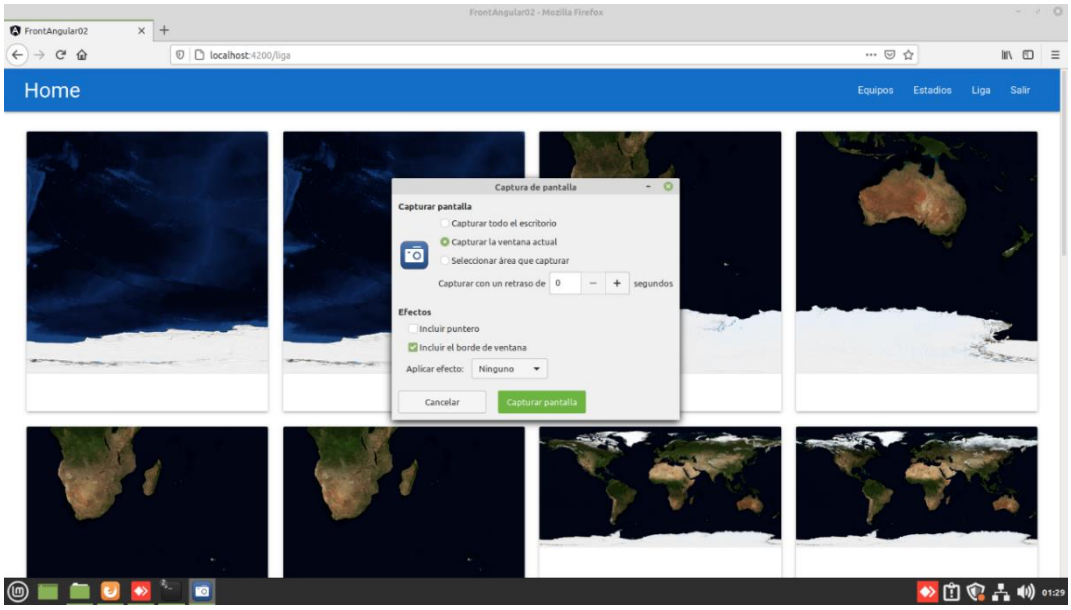


Figura 18 Sección Liga de la aplicación R2 y MS2.

La figura 19 muestra la interfaz de la herramienta JMeter utilizada para la prueba de carga, estrés y ataques bloqueados.

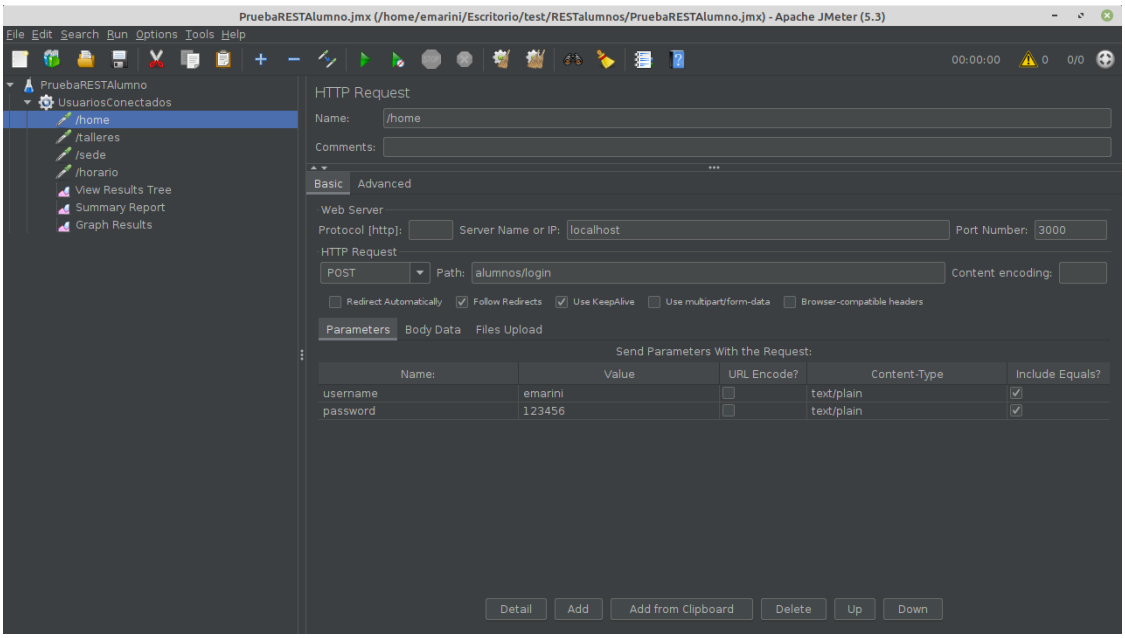


Figura 19 Interfaz de la herramienta JMeter.

La figura 20 muestra la interfaz de la herramienta Netdata utilizada para medir el uso de recursos.

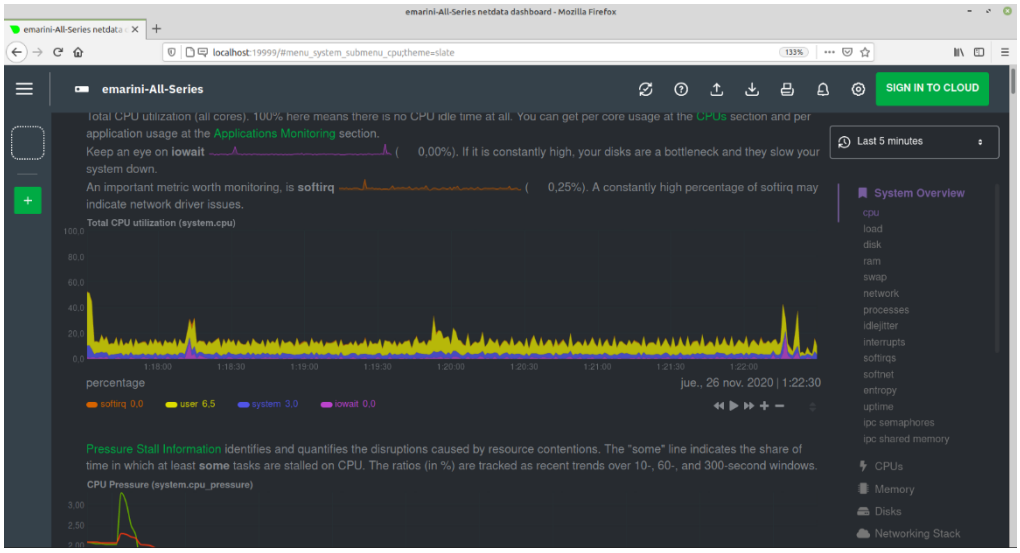


Figura 20 Interfaz de la herramienta Netdata.

La figura 21 muestra la interfaz de la herramienta OWASP ZAP utilizada para identificar la cantidad de vulnerabilidades

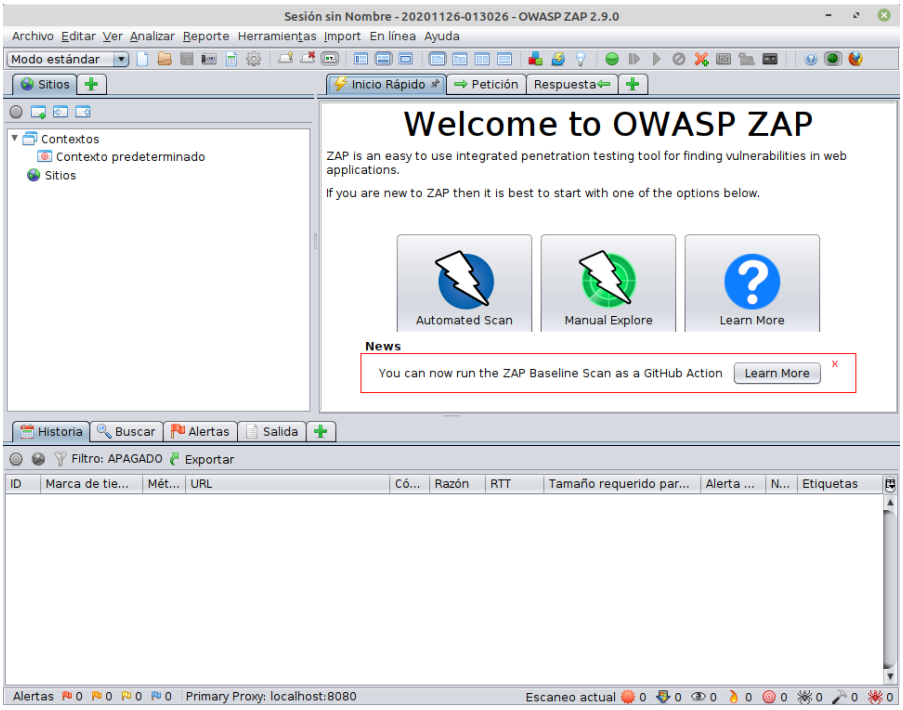


Figura 21 Interfaz de la herramienta OWASP ZAP.

La figura 22 muestra la interfaz de la herramienta monitor de red de Firefox, utilizada para medir el tiempo de respuesta de la carga de imágenes.

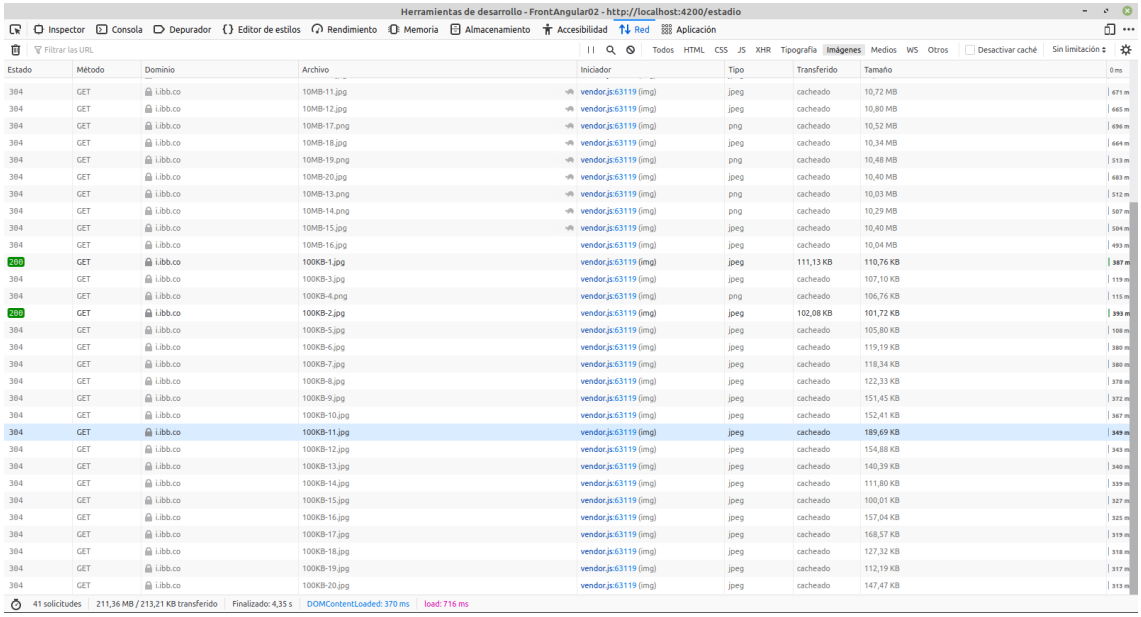


Figura 22 Interfaz de la herramienta Monitor de Red de Firefox.