UNIVERSIDAD PERUANA UNIÓN

ESCUELA DE POSGRADO

Unidad de Posgrado de Ingeniería y Arquitectura

Método de referencia para balanceo de carga en servicios web con topología REST usando herramientas Edge Route

Trabajo de Investigación para optar el Grado Académico de Maestro en Ingeniería de Sistemas con mención en Ingeniería de Software

Autor:

Edwin René Calsin Quinto

Asesor:

Abel Angel Sullom Macalapu

Lima, marzo del 2021

DECLARACIÓN JURADA DE AUTORÍA DE TRABAJO DE INVESTIGACIÓN

Abel Angel Sullom Macalapu, de la Escuela de Posgrado, Unidad de Posgrado de Ingeniería y Arquitectura, de la Universidad Peruana Unión.

DECLARO:

Que la presente investigación titulada: **"Método de referencia para balanceo de carga en servicios web con topología REST usando herramientas Edge Route"** constituye la memoria que presenta el bachiller Edwin René Calsin Quinto para aspirar al Grado Académico de Maestro en Ingeniería de Sistemas con mención en Ingeniería de software, cuyo trabajo de investigación ha sido realizada en la Universidad Peruana Unión bajo mi dirección.

Las opiniones y declaraciones en este informe son de entera responsabilidad del autor, sin comprometer a la institución.

Y estando de acuerdo, firmo la presente declaración en la ciudad de Juliaca, a los 15 días del mes de marzo del año 2021.

Abel Angel Sullom Macalapu

ACTA DE SUSTENTACIÓN DEL TRABAJO DE INVESTIGACIÓN

En Lima, Ñaña, Villa Unión, a 22 días del mes de febrero del año 2021, siendo las 9:00 a.m, se reunieron en la modalidad online sincrónica, bajo la dirección del Señor Presidente del Jurado: Dr. Josué Edison Turpo Chaparro, el secretario: Mg. Nemias Saboya Rios, los demás miembros: Mg. Immer Elias Cuellar Rodríguez y el Mg. Omar Leonel Loaiza Jara y el asesor: Mg. Abel Angel Sullon Macalupu, con el propósito de administrar el acto académico de sustentación de Tesis de Maestro(a) titulada: Método de referencia para el balanceo de carga en servicios web con topología REST usando herramientas Edge Route. del Bachiller/Licenciado(a) Edwin René Calsin Quinto

.....Conducente a la obtención del Grado Académico de Maestro(a) en: Ingeniería de Sistemas

(Nomenclatura del Grado Académico)

.....con Mención en Ingeniería de Software. El Presidente inició el acto académico de sustentación invitando al candidato hacer uso del tiempo determinado para su exposición. Concluida la exposición, el Presidente invitó a los demás miembros del Jurado a efectuar las preguntas, cuestionamientos y aclaraciones pertinentes, los cuales fueron absueltos por el candidato. Luego se produjo un receso para las deliberaciones y la emisión del dictamen del Jurado.

Posteriormente, el Jurado procedió a dejar constancia escrita sobre la evaluación en la presente acta, con el dictamen siguiente:

Bachiller/Licenciado (a): Edwin René Calsin Quinto

CALIFICACIÓN	ESCALAS			Mérito
	Vigesimal	Literal	Cualitativa	
Aprobado	18.3	A-	Con nominación muy bueno	Sobresaliente

(*) Ver parte posterior

Finalmente, el Presidente del Jurado invitó al candidato a ponerse de pie, para recibir la evaluación final. Además, el Presidente del Jurado concluyó el acto académico de sustentación, procediéndose a registrar las firmas respectivas.

Presidente

Secretario

Asesor

Miembro

Miembro

Bachiller/Licenciado(a)

Método de referencia para el balanceo de carga en servicios web con topología REST usando herramientas Edge Route

Edwin R. Calsin Quinto[1] (0000-0002-2010-7273), A. Angel Sullon[2] (0000-0003-4142-7230), and Fredy Abel Huanca Torres[1] (0000-0001-7645-714)

[1] Escuela de Posgrado, Unidad de posgrado de Ingeniería y Arquitectura. Universidad Peruana Unión, Lima Perú,
[2] Universidad Peruana Unión, Juliaca Perú
edwin_calsin,angeli,abel.huanca@upeu.edu.pe

Abstract. En la actualidad, debido a la gran demanda en las tecnologías de la información, toda organización ofrece productos y servicios mediante servicios web, por eso resulta incuestionable que deben funcionar de forma ininterrumpida y estable. La fuerte implementación de servicios web con topología REST conlleva a desarrollar sistemas distribuidos facilitando la escalabilidad y portabilidad de los servicios web, mas no en estabilidad y rendimiento. El objetivo de esta investigación es desarrollar un método de referencia para el balanceo de carga en servicios web con topología REST usando herramientas Edge Route. Se desarrolló un método de referencia basado en los paradigmas de servicios web y se probó 3 API REST *Laravel* con la información de 3 usuarios escogidos aleatoriariamente por solicitud de un módulo de matrícula universitaria, balanceando la carga con *Treafik* y escalando en contenedores *Docker*; se simuló una carga de 7,575 solicitudes cliente usando *Apache JMeter* y se evaluó los indicadores de tiempo de respuesta máximo y rendimiento de las API REST. En cuanto a los resultados de la aplicación del método propuesto, se mejoró la estabilidad en tiempo de respuesta máximo a 1.1 segundos desde 2.2 segundos en promedio; asimismo, se tuvo un mejor rendimiento en las API REST con resultados positivos al evaluar los indicadores propuestos, a mayor estabilidad mayor escalamiento en contenedores. Tener servicios web implica planificar una arquitectura amplia basada en los paradigmas de servicios web y balanceo de carga para tener una mejor estabilidad y rendimiento.

Keywords: Servicios web, API REST, Edge Route, Estabilidad, Rendimiento, Docker, Traefik

1 Introducción

Hoy en día las organizaciones orientan su oferta utilizando tecnologías de la información, toda organización ofrece productos y servicios mediante aplicaciones, esto debido a una creciente demanda del uso de servicios web; organizaciones como Yahoo, Google, Facebook y Twitter, usan servicios web por el alto

desacoplamiento y escalabilidad [15]. La implementación de servicios web con topología REST conlleva a desarrollar sistemas distribuidos donde clientes web realizan múltiples solicitudes HTTP. En esta perspectiva las aplicaciones web están orientadas a tener un nivel funcional y un nivel reactivo [2], donde el nivel funcional genera el 83% de todo el tráfico web [24]. En ese sentido, empresas que adoptan soluciones como Kubernetes o balanceadores de tráfico en la nube generan alto costo de implementación [8].

Esta investigación se justifica, porque más del 90% de las organizaciones esperan una latencia de menos de 50 milisegundos, mientras que el 60% esperan una latencia de 20 milisegundos o menos [12], lo que conlleva a implementar servicios web de alto rendimiento y estabilidad. El propósito del diseño arquitectónico REST es la portabilidad de la interfaz de usuario en múltiples sistemas distribuidos y su escalabilidad [6], mas no en el rendimiento y estabilidad de los servicios web. De esta manera, la implementación de servicios web con topología REST es solo backend, mas cuando se implementa junto a otros paradigmas de tecnologías disminuye su rendimiento.

Stelios Sotiriadis, et al. [19] diseñaron un método de reconfiguración dinámica de cómo administrar un microservicio escalado basado en proveedores entre nubes, su objetivo era mejorar la elasticidad a nivel de infraestructura, mas no en el rendimiento y estabilidad de servicios web con topología REST. Existen tecnologías como *Elastic load balancing* de *AWS*, *Amazon EKS*, *DO k8s* y *GCP k8s* que solucionaron el problema del tráfico web, mas para ser implementadas se necesita un personal capacitado; ocasionando dependencia de servicios para garantizar un servicio óptimo.

El método de referencia surge ante la necesidad de reducir el alto tiempo de respuesta en servicios web con topología REST, mejorando su rendimiento y estabilidad; asimismo, un equipo encargado del desarrollo de software puede mejorar los niveles mínimos en la métrica del comportamiento en el tiempo de la subcaracterística eficiencia en la evaluación de la calidad interna de software para la ISO/IEC 25000 [5, 18, 7].

Tomando como antecedentes, en el Perú se implementó una plataforma en EC2 de Amazon, para centralizar múltiples canales de servicios web de pagos Visanet, transformando su arquitectura y mejorando la concurrencia y estabilidad de sus servicios distribuidos. Por otro lado, se desarrollaron dos arquitecturas de software, la primera en SOA y la segunda en .NET desplegados en *Azure Cloud Services*, mejorando el intercambio de información en lugares remotos del Perú. Ambas investigaciones implementaron servicios web mejorando la escalabilidad y distribución de sus aplicaciones bajo un costo monetario en la nube [3, 16].

Otra investigación realizó un estudio de rendimiento en tiempo de respuesta, comparando las arquitecturas REST y GraphQL, como resultado GraphQL respondió en menor tiempo frente a REST. Así también, en otro estudio se mejoró un 47% en rendimiento de descarga de archivos de sus servicios web usando multiplexación con HTTP/2. [21, 14].

Además, una investigación determinó que HAProxy resultó ser más rápido al atender solicitudes frente a Nginx, siendo que HAProxy utilizó un 82% de

CPU mayor que Nginx con un 40% de consumo de CPU. Por otro lado, en otra investigación el uso de algoritmos de distribución de carga fue vital para el funcionamiento óptimo en sus servicios web, donde el algoritmo *Leastconn* superó en velocidad de conexión, frente al algoritmo *Round robin* el cual es generalmente usado [20, 17].

El diseño arquitectónico REST en servicios web, está enfocado en la portabilidad y la escalabilidad, en esta perspectiva una página web puede realizar múltiples solicitudes a servicios web distribuidos [6, 10], generando alto tráfico en la infraestructura del *backend*; es el caso de algunos estudios, donde mejoraron la portabilidad de sus servicios web con topología REST mas no su rendimiento en tiempo de respuesta [22, 23].

En ese sentido, el objetivo de esta investigación es desarrollar un método de referencia para el balanceo de carga en servicios web con topología REST basado en herramientas Edge Route.

Este artículo está organizado de la siguiente manera: la sec. 2 describe la metodología; la sec. 3 presenta la prueba del método planteado y el análisis de resultados y finalmente la sec. 4 las conclusiones.

1.1 Servicios web

En el mercado actual una de las plataformas de tecnología más usadas de la Arquitectura Orientada a Servicios (SOA) son los servicios web. SOA es un modelo arquitectónico independiente de cualquier plataforma tecnológica [4]. Existen diversos estándares de servicios web como XML, HTTP, SOAP, REST, WSDL y UDDI; sin embargo, SOA no necesariamente implementa los mencionados estándares. Fielding [6] presenta un estilo arquitectónico híbrido llamado Representational State Transfer (REST) enfocado en sistemas distribuidos, también usa un único identificador HTTP de recurso para la interacción entre componentes.

1.2 Balanceo de carga

Un equilibrador de carga del servidor (SLB) es un proceso y una tecnología que distribuye el tráfico de un sitio web entre varios servidores utilizando un dispositivo basado en red. Asimismo, un equilibrador de carga es transparente para el usuario final. En relación a este tema un balanceador de carga comparte el trabajo entre varios procesos y recursos, siendo de gran ayuda cuando los servicios web tienen una alta demanda de consumo [1].

1.3 Herramientas Edge Route

La aplicación de herramientas *Edge Route* es una de las soluciones para el equilibro de carga; satisfacer la carga de trabajo del tráfico de la red depende de la cantidad de procesadores [13, 11, 9]. Un estudio realizado sobre herramientas *Edge Route* evaluó dos herramientas de balanceo de carga que implementa una

API REST, midiendo solicitudes completadas y ejecutando una prueba de 100 solicitudes simultáneas con *Apache Benchmark*; como servidor y balanceador de carga usó *HAproxy* y *Traefik*; resultando que *Traefik* resuelve mayor cantidad de solicitudes en menor tiempo frente a HAproxy [25]. *Traefik* es un enrutador de borde de código abierto, es compatible con *Docker* y para esta investigación se usó la aplicación proxy de *Traefik* como equilibrador de carga; lo que hace que *Traefik* sea diferente de Nginx y HAProxy es la capacidad de configuración automática, dinámica y en descubrimiento de servicios.

2 Metodología

La construcción del método de referencia para el balanceo de carga, desde la perspectiva más general, integra el paradigma orientado a servicios en la infraestructura de aplicaciones de servicios web; esta integración ayuda a definir *clusters* de balanceo de carga para tener mayor estabilidad en los servicios web; dentro de este orden de ideas, se plantea cuatro pasos con actividades para el desarrollo del método de referencia como muestra la figura 1.

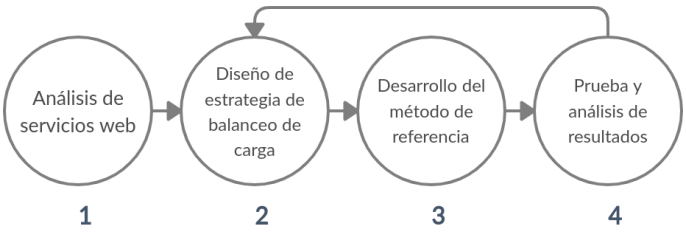

Fig. 1. Metodología para el desarrollo del método de referencia

2.1 Análisis de servicios web

Este paso analiza los servicios web, identificando los modelos de servicio: entidad, tarea y utilidad; estos modelos dan a conocer una clasificación de recursos en las APIs. Los modelos de servicio son descritos por el paradigma orientado a servicios, llamados también capas de abstracción de servicios; estas capas forman una jerarquía que ayuda a indentificar los servicios más reutilizados, para luego determinar que existen servicios que reciben mayor ó menor cantidad de solicitudes del tráfico web. El modelo de servicio de entidad se centra en las entidades de negocio empresarial, razón por la cual son altamente reutilizables. Por otro lado, el modelo de servicio de tarea está enfocado a procesos empresariales específicos; existen algunos que son necesariamente orquestados con la finalidad

de tener un controlador que ejecute múltiples funciones específicas. Finalmente, el modelo de servicio de utilidad no tiende a ser altamente utilizado porque no está asociado con la lógica de negocio, sino con la tecnología [4].

2.2 Diseño de estrategia de balanceo de carga

En el paso anterior se clasificaron los servicios web, esta clasificación ayuda a determinar qué servicios tienden a tener mayor cantidad de solicitudes y aumentar el tráfico web; en ese sentido, el tráfico web también varía según la clasificación de modelos de servicio. La estrategia de balanceo de carga es clasificar recursos según el modelo de servicio y tener un servidor *cluster* por cada uno; esta clasificación se definió según la utilización de las capas de abstracción de servicios web [4].

El servidor web es calibrado por cada modelo de servicio, asignando recursos según el escenario de la aplicación, es el caso del modelo de servicio de entidad. Según el paradigma orientado a servicios web se determina que este último es el más reutilizado, es decir, recibirá mayor cantidad de solicitudes cliente. Existen servicios que no utilizan muchos recursos; así, los tipos de una tabla no tienen muchos registros y simplemente son una selección de datos; en cambio, existen servicios de entidad o tarea con métodos que utilizan bastantes recursos al ejecutar hilos de procesos consumiendo CPU.

2.3 Desarrollo del método de referencia

La clasificación de servicios web en modelos de servicio como estrategia de balanceo comprende el uso y asignación de recursos en los servicios web. Visto de esta forma, se identifican recursos web para diseñar un balanceo de carga; asimismo, se realiza una composición explícita del balanceo de carga para despliegues escalables de forma dinámica; por otro lado, la identificación de recursos se evalúa para medir su rendimiento y estabilidad en tiempos de respuesta máximo, dentro de este orden de ideas se propone los pasos mostrados en la siguiente figura 2.

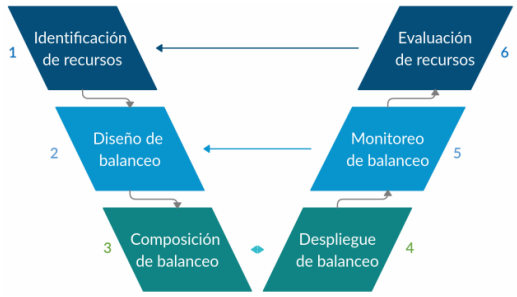

Fig. 2. Método de referencia para el balanceo de carga

Identificación de recursos Este paso del método identifica y clasifica las API REST según el modelo de servicio web: entidad, tarea y utilidad [4]; asimismo, determina los objetivos que se desea alcanzar.

Diseño de balanceo Seguidamente, este paso del método comprende dos actividades: la primera es diseñar la distribución de los servicios web por modelo de servicio, teniendo en cuenta que los servicios de entidad y tarea son los más reutilizados y requieren más recursos para ser representados en *clusters* de servicios web, como también en un *cluster* de base de datos para equilibrar la carga; si se identifica una gran cantidad de microservicios, es necesario usar un orquestador de contenedores para un mejor control de *clusters*. La segunda actividad es realizar escalamientos que sirven como *pivot* para mejorar la estabilidad en los servicios web.

Composición de balanceo Este paso del método establece una configuración explícita del diseño de balanceo usando herramientas alternativas *Edge Route* como Traefik, HAProxy o Nginx; esta composición debe ser escrita según la herramienta de balanceo de carga, se recomienda la composición de *Docker* para el escalamiento dinámico de contenedores.

Despliegue de balanceo Para el despliegue es necesario tener un servidor de *clusters* con contenedores descartables para su balanceo. Según el escenario de la infraestructura se puede realizar un escalamiento dinámico de servicios o uno explícito en un grupo de servidores con contenedores *Docker*, teniendo en consideración que al escalar se necesitan más recursos en el servidor.

Monitoreo de balanceo En este tramo del método se verifican los servicios y recursos asignados, como alternativa para ejecutar el comando *docker stats* que proporciona información de recursos por contenedor; es recomendable que la herramienta de balanceo proporcione la información de control de puertos de entrada, protocolos de transporte, routes y servicios configurados.

Evaluación de recursos Evaluar el tiempo de respuesta promedio, depende de muchos factores que pueden ser: latencia, alto procesamiento de información y consultas SQL no indexadas; el propósito del método de referencia es lograr la estabilidad en las API REST, es decir, reducir los picos altos en tiempo de respuesta. Es necesario tener una herramienta de prueba de carga como *Apache JMeter* o *Locust* que proporcionan información para su evaluación. Se consideran los indicadores de tiempo de respuesta máximo y rendimiento de servicio, en consecuencia, si el tiempo de respuesta máximo sigue siendo alto y los servicios web dejan de funcionar por la alta concurrencia, entonces es necesario regresar al paso de diseño de balanceo; es por ello se plantea los siguientes indicadores en la tabla 1.

Table 1. Indicadores de evaluación

Indicador	Unidad de medida
Tiempo de respuesta máximo	milisegundos
Rendimiento de servicio	Numérico

Para la evaluación de los indicadores descritos en la tabla 1 se plantea las siguentes variables que son calculadas por *Apache JMeter*:

t1 = Tiempo de respuesta máximo sin balanceo
t2 = Tiempo de respuesta máximo con balanceo
r1 = Rendimiento sin balanceo
r2 = Rendimiento con balanceo

Indicador tiempo de respuesta máximo Indica el promedio del tiempo máximo en milisegundos, desde que solicita un cliente web hasta su entrega, considerando las siguientes métricas en la tabla 2.

Table 2. Métrica tiempo de respuesta máximo

Estado	Resultado
t2 menor t1	Bueno
t2 mayor t1	Por mejorar

Indicador rendimiento de servicio Este indicador ayuda a evaluar la capacidad del servidor en términos de cuanta carga puede soportar, el rendimiento es igual al número de solicitudes entre el tiempo total de prueba descrito por la herramienta *Apache JMeter* representado en la siguiente tabla 3.

Table 3. Metrica de rendimiento de servicio

Estado	Resultado
r2 mayor r1	Bueno
r2 menor r1	Por mejorar

3 Prueba y análisis de resultados

La prueba del método planteado, consiste en evaluar los indicadores de tiempo de respuesta máximo y rendimiento de servicios web con topología REST, de un módulo de matrícula universitaria, balanceando una carga de tráfico de 7,575 solicitudes cliente que es un promedio total de usuarios que acceden al mencionado módulo; de esta manera cada API REST consultó de forma aleatoria la información de 3 usuarios distintos. El estado actual de las API REST sin balanceo están descritas en la evaluación de recursos de esta sección, donde se muestra que existe inestabilidad a un alto tráfico web. Se utilizó la herramienta *Apache JMeter*, para simular las solicitudes cliente durante 20 minutos por *cluster*, para obtener los resultados de la prueba.

3.1 Identificación de recursos

Se identificó tres recursos API REST de un módulo de matrícula universitaria que representan los modelos de servicio y se establecieron los siguientes objetivos:

1. *user-info*, el recurso de utilidad debe tener un tiempo de respuesta máximo menor a 1.5 segundos para 7,575 solicitudes cliente.
2. *enrollment-courses*, el recurso de tarea debe tener alto rendimiento y un tiempo de respuesta máximo menor a 1.5 segundos, porque inicia procesos internos de matrícula.
3. *enrollment-list*, el recurso de entidad debe tener un tiempo de respuesta máximo menor a 1.5 segundos para 7,575 solicitudes cliente.

3.2 Diseño de balanceo

Como primera actividad, se diseñó una distribución en contenedores *Docker* según modelo de servicio como: utilidad, entidad y tarea para su balanceo; no solo en los servicios web, sino también en la base de datos si en caso existe picos altos en tiempo de respuesta.

La figura 3 muestra la infraestructura de servicios web. Como balanceador carga de los servicios web y de base de datos se usó *Traefik*, seguidamente para el *cluster* de contenedores se usó *Docker*, distribuidos en dos grupos: el primero para servicios de utilidad y el segundo para servicios de entidad y tarea.

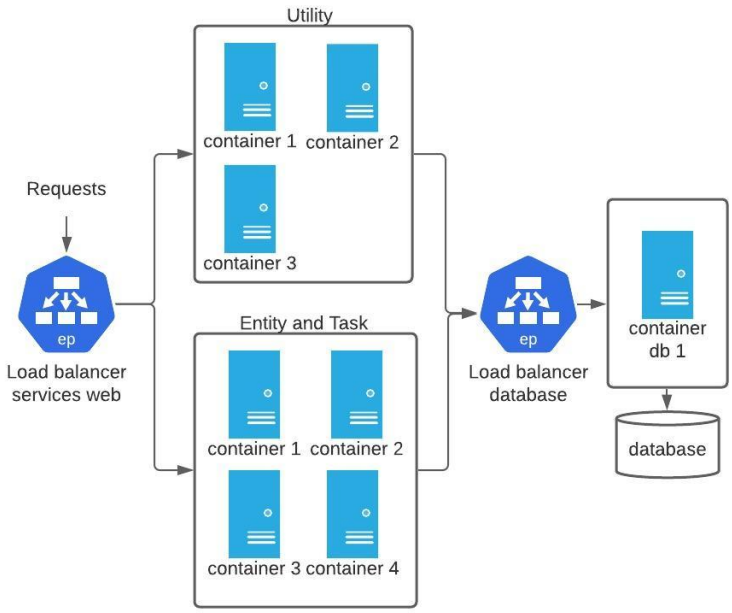

Fig. 3. Diseño de balanceo de carga en servicios web

La segunda actividad consistió en asignar réplicas; para el *cluster* de servicios de utilidad se realizó 7 replicas y para el *cluster* de servicios de entidad y tarea se realizó 10 réplicas; este número de réplicas fue lo considerable según los recursos del servidor para esta prueba de investigación.

3.3 Composición de balanceo

Para la composición de balanceo se usó *Docker* con la imagen *Traefik proxy* en su segunda versión, inicia definiendo puntos de entrada que son los puertos 80 y 443, seguidamente se crea una red *bridge* con *docker network*, a esta red se conectaron los *clusters* descritos en el diseño de balanceo, para esta prueba no se validó la *introspection oauth2* porque el objetivo principal es desarrollar el método de referencia mas no una infraestrutura de servicios web, finalmente se vinculó el archivo de la configuración de *traefik* en el contenedor de balanceo.

3.4 Despliegue de balanceo

El despliegue de balanceo para esta investigación creó tres instancias Ubuntu Server 20.04 LTS (Focal Fossa) en una plataforma de alojamiento en la nube *UpCloud* descritos en la siguiente tabla 4.

Table 4. Recursos de servidores

Name	CPU	Storage	Memory
services balancer	6 cores	15GB	8GB
services basic	2 cores	15GB	4GB
server oracle db	4 cores	20GB	8GB

La tabla 4 muestra las instancias *services balancer* y *services basic* para los servicios API REST implementado con el framework *laravel* y la imagen *php:7.4.14-apache-buster* como servidor de aplicación. Por otro lado la instancia *server oracle db* usa la imagen *oracle/database:18.4.0-xe* como servidor de base de datos. En esta perspectiva se levantó un contenedor *auth* para el modelo de servicio de utilidad, un contenedor *enrollment* para los modelos de servicio entidad y tarea; finalmente se levantó un contenedor oracle como servidor de base de datos.

3.5 Monitoreo de balanceo

Para el monitoreo del balanceo de carga se ejecutó el comando *docker stats* para observar el consumo de recursos en CPU y RAM de los contenedores definidos en el paso de diseño de balanceo y desplegados en el paso anterior. Como resultado se llegó a usar hasta un 47% de CPU en enrollment, un 32% de CPU en auth y un 68% de CPU en base de datos, con un promedio de 3GB de memória RAM en consumo. Por otro lado como alternativa se verificó el *dashboard* de *Traefik proxy* con información de los servicios, proveedores y réplicas de los contenedores *Docker*.

3.6 Evaluación de recursos

Se evaluó los indicadores de tiempo de respuesta máximo y rendimiento de servicios según las métricas descritas en el método; asimismo, se configuró los recursos identificados en un grupo de subprocesos usando dos *Sampler HTTP Request* con balanceo y sin balanceo en *Apache JMeter* de la siguiente forma:

Número de procesos = 7575
Periodo de aceleración = 1200
Contador bucle = 1

El número de procesos 7575 indica las solucitudes que se realizaron, el periodo de aceleración es el tiempo de prueba en segundos que en minutos equivalen a 20 y el contador bucle es el número de veces que ejecuta el proceso; finalmente, se configuró los oyentes *View Result Tree*, *Summary Report*, *Aggregate Report*, *View Result in Table* y *jp@gc - Response Times Over Time*.

Cluster de Utilidad Esta prueba consistió en ejecutar solicitudes cliente a la API REST *user-info* distribuidas de forma tradicional y balanceada como se muestra en la siguente figura 4.

Fig. 4. Tiempo de respuesta de *user-info*

La figura 4 muestra que la API REST *user-info* balanceada tiene mayor tiempo de respuesta promedio cuando no es balanceada, pero es más estable; en la API REST no balanceada se ven picos altos en tiempo de respuesta, provocando inestabilidad, el promedio en tiempo de respuesta incrementó al ser balanceada la carga, esto sucede porque al solicitar se realiza una redirección al servidor replicado, el reporte devuelto por *Apache JMeter* muestra la siguiente tabla 5.

Table 5. Resultados de *user-info*

Label	Samples	Average	Min	Max	Throughput
user-info	7575	439	416	1170	6.32769
user-info (balanced)	7575	446	419	675	6.32771

La tabla 5 muestra que la API REST *user-info* tiene un tiempo de respuesta máximo de 1170 milisegundos frente a 675 milisegundos de la API REST *user-info (balanced)*. Visto de esta forma se tiene resultados positivos al balancear la carga de 7,575 solicitudes cliente y en un tiempo de respuesta máximo menor a 1.5 segundos, como objetivo logrado se consiguió una estabilidad al ser balanceada.

Cluster de entidad y tarea Esta prueba consistió en ejecutar solicitudes cliente a las API REST *enrollment-list* y *enrollment-courses* distribuidas de forma tradicional y balanceada; la API REST *enrollment-courses* procesa mas

información frente a la API REST *enrollment-list* como se muestra en la siguente figura 5.

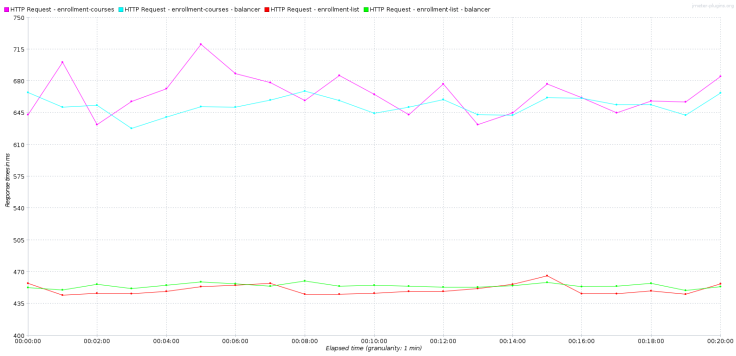

Fig. 5. Tiempo de respuesta de *enrollment-list* y *enrollment-courses*

La figura 5 muestra que la API REST *enrollment-list balanced* tiene mayor estabilidad, como también menor tiempo de respuesta, esto sucede porque el servicio no resuelve una lógica compleja y no consume muchos recursos. En cambio, la API REST *enrollment-courses balanced* resuelve una lógica compleja, procesando bastante información, siendo que es mas notorio el tiempo de respuesta y la estabilidad al ser balanceada, el reporte devuelto por JMeter muestra la siguente tabla 6.

Table 6. Resultados de *enrollment-list* y *enrollment-courses*

Label	Samples	Average	Min	Max	Throughput
enrollment-list	7575	450	419	1668	6.3276
enrollment-courses	7575	666	422	4020	6.32583
enrollment-list (balanced)	7575	454	421	1492	6.32682
enrollment-courses (balanced)	7575	652	424	1296	6.32586

La tabla 6 muestra la API REST *enrollment-list* con un tiempo de respuesta máximo de 1668 milisegundos frente a 1492 milisegundos al ser balanceada siendo menor a 1.5 segundos y logrando el objetivo propuesto. la API REST *enrollment-courses* tiene un tiempo de respuesta máximo de 4020 milisegundos frente a 1296 milisegundos al ser balanceada siendo menor a 1.5 segundos y cumpliendo el objetivo, estos resultados positivos nos ayudan a determinar que los servicios web son mas estables y se redujo los picos altos en tiempo de respuesta. Por

consiguente según los indicadores de evaluación se tiene los resultados en la siguente tabla 7.

Table 7. Resultados según indicador

API REST	Indicador	Resultado
user-info	tiempo de respuesta máximo	Bueno
enrollment-list	tiempo de respuesta máximo	Bueno
enrollment-courses	tiempo de respuesta máximo	Bueno
enrollment-courses	rendimiento	Bueno

La tabla 7 muestra resultados positivos según las métricas de evaluación por indicador en relación con los objetivos propuestos; al aplicar el método de referencia se obtiene una mejora en la estabilidad y rendimiento de los servicios web con topología REST.

4 Conclusión

Se desarrolló un método de referencia de seis pasos para el balanceo de carga en servicios web con topologia REST basados en herramientas Edge Route, logrando reducir el tiempo de respuesta máximo y mejorando la estabilidad de tres servicios web de un módulo de matrícula universitaria, tomando en consideración que realizar un balanceo de carga aumenta en pequeñas cifras el tiempo de respuesta promedio, se logró tener mayor estabilidad en los servicios realizando replicas de los servicios web y balanceando la carga de tráfico en *clusters* según modelo de servicio; asimismo, a una mayor estabilidad de los servicios web, mayor es el escalamiento.

Tener servicios web implica planificar una arquitectura amplia, el crecimiento de servicios web implica implementar un core base con principios teóricos de la arquitectura orientada a servicios SOA, no solo es tener servicios distribuidos sino organizados para tener un mejor rendimiento y estabilidad al ser balanceadas. Por lo tanto, a mayor escalamiento, mayor estabilidad y menor tiempo de respuesta máximo.

References

1. Bourke, T.: Server Load Balancing, vol. 1, first edition edn. O Reilly Associates, Inc., 101 Morris Street, Sebastopol, CA 95472 (2001)
2. Caparrini, F.S.: Programación funcional reactiva (2016). URL http://www.cs.us.es/~fsancho/?e=111
3. Díaz Ayasta Krysthiam Luciano, O.E.J.E.: "sbusiness core": Núcleo transaccional de negocios electrónicos que integra distintos canales y productos con diferentes proveedores. Thesis (2014)

4. Erl, T.: Service-Oriented Architecture: Analysis and Design for Services and Microservices, vol. 1, second edition edn. Mark Taub, Printed in the United States of America (2017)
5. Esaki, K., Azuma, M., Komiyama, T.: Introduction of quality requirement and evaluation based on iso/iec square series of standard. pp. 94–101. Springer Berlin Heidelberg. DOI 10.1007/978-3-642-35795-4_12
6. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Thesis (2000). URL https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
7. Figueroa, M.A.A.: Calidad en la industria del software. la norma iso-9126 (2012). URL https://www.repositoriodigital.ipn.mx/handle/123456789/5321
8. Google: Prácticas recomendadas para ejecutar aplicaciones de kubernetes con optimización de costos en gke (2020). URL https://cloud.google.com/solutions/best-practices-for-running-cost-effective-kubernetes-applications-on-gke
9. Haas, R., Kencl, L., Kind, A., Metzler, B., Pletka, R., Waldvogel, M., Frelechoux, L., Droz, P., Jeffries, C.: Creating advanced functions on network processors: experience and perspectives. IEEE Network **17**, 46–54 (2003). DOI 10.1109/MNET.2003.1220696. URL https://ieeexplore.ieee.org/abstract/document/1220696
10. Haupt, F., Leymann, F., Vukojevic-Haupt, K.: Api governance support through the structural analysis of rest apis. Computer Science-Research and Development **33**(3-4), 291–303 (2018). DOI 10.1007/s00450-017-0384-1. URL ⟨GotoISI⟩://WOS:000438657800003
11. Hofmeyr, S., Costin, I., Blagojevi: Load balancing on speed. SIGPLAN Not. **45**(5), 147–158 (2010). DOI 10.1145/1837853.1693475. URL https://doi.org/10.1145/1837853.1693475
12. Jennifer Thomson, G.M.: Apis the determining agents between success or failure of digital business. International Data Corporation (IDC) (2019)
13. Kachris, C., Stamatis, V.: A reconfigurable platform for multi-service edge routers. Sbcci '07 p. 165–170 (2007). DOI 10.1145/1284480.1284529. URL https://doi.org/10.1145/1284480.1284529
14. Nikravesh, A., Yihua, G., Xiao, Z., Feng, Q., Morley, M.Z.: Mp-h2: A client-only multipath solution for http/2. MobiCom '19 (2019). DOI 10.1145/3300061.3300131. URL https://doi.org/10.1145/3300061.3300131
15. Palacios Aguilar, P., Ynga Palacios, C.: Propuesta de implementaciÓn de un marco de trabajo para el desarrollo de aplicaciones android. Thesis (2015)
16. Posadas, J.V.: Application of mixed distributed software architectures for social productive projects management in perú pp. 1–4 (2017). DOI 10.1109/INTERCON.2017.8079698
17. Prasetijo, A.B., Widianto, E.D., Hidayatullah, E.T.: Performance comparisons of web server load balancing algorithms on haproxy and heartbeat. 2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE) pp. 393–396 (2016). DOI 10.1109/ICITACEE.2016.7892478. URL https://ieeexplore.ieee.org/document/7892478
18. Reina, E., Susana, P.: Evaluación de la calidad en uso de un sistema web móvil de control de asistencia a clases de docentes y estudiantes aplicando la norma iso iec 25000 square. RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao **E19**, 108 (2019)
19. S. Sotiriadis N. Bessis, C.A.R.B.: Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling **12**, 319–334 (2019). DOI 10.1109/TSC.2016.2634024

20. Sahand Kh, S., Yahiya, T.A.: Load balancing evaluation tools for a privatecloud: A comparative study. ARO-The Scientific Journal of Koya University **VI, Nro2**, 7 (2018). DOI 10.14500/aro.10438
21. Seabra, M., Nazário, M.F., Pinto, G.: Rest or graphql? a performance comparative study. Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse (2019). DOI 10.1145/3357141.3357149. URL https://doi.org/10.1145/3357141.3357149
22. Takahashi, K., Aida, K., Tanjo, T., Sun, J.: A portable load balancer for kubernetes cluster. Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (2018). DOI 10.1145/3149457.3149473. URL https://doi.org/10.1145/3149457.3149473
23. Thaler, D.G., Ravishankar, C.V.: Using name-based mappings to increase hit rates. IEEE/ACM Trans. Netw. **6**(1), 1–14 (1998). DOI 10.1109/90.663936. URL https://doi.org/10.1109/90.663936
24. Tony Lauro, M.F., Steiner, M., Schomp, K., Dalky, R.A.: State of the internet security, retail attacks and api traffic. Akamai intelligent security starts at the edge **5**, 23 (2019). URL https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/state-of-the-internet-security-retail-attacks-and-api-traffic-report-2019.pdf
25. Voronenko: Traefik as an alternative reverse proxy to nginx for self hosted dockerized applications. URL https://www.codementor.io/@slavko/traefik-as-an-alternative-reverse-proxy-to-nginx-for-self-hosted-dockerized-applications-bm5tpcsmj