EVALUACIÓN DEL RENDIMIENTO DE DIFERENTES MODELOS DE PROGRAMACIÓN WEB UTILIZADOS EN DIVERSOS CONTEXTOS

VIVIANA ANDREA MALDONADO BELTRÁN
EDWIN ALFONSO VESGA ARIAS

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2019

EVALUACIÓN DEL RENDIMIENTO DE DIFERENTES MODELOS DE PROGRAMACIÓN WEB UTILIZADOS EN DIVERSOS CONTEXTOS

VIVIANA ANDREA MALDONADO BELTRÁN
EDWIN ALFONSO VESGA ARIAS

TRABAJO DE GRADO PARA OPTAR EL TITULO DE INGENIERO DE SISTEMAS

DIRECTOR
GABRIEL RODRIGO PEDRAZA FERREIRA
Ph.D. CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2019

Dedicatorias

A Dios por guiar mi camino y ayudarme a alcanzar este logro.
A mis padres Miriam y Pedro por su inmenso apoyo, por sus palabras de ánimo, por ese gran amor y esfuerzo para que pudiera culminar esta etapa. Por sus grandes consejos y su paciencia.
A mi hermano Juank por sus sabios consejos, por enseñarme a ser autodidacta, disciplinada y a dar lo mejor de mí siempre. Por demostrarme su cariño y apoyo incondicional.
A mi nonita Delia por sus enseñanzas, por su cuidado y su amor.
A mis amigos por su sinceridad, lealtad y por su buena energía siempre.

Viviana Andrea Maldonado Beltrán

Dedicado a Dios por permitirme cumplir con este objetivo y a mis padres por brindarme su apoyo incondicional.

Edwin Alfonso Vesga Arias

Agradecimientos

Agradecemos a Dios por permitirnos culminar el proyecto con éxito y por las nuevas oportunidades que aparecen en nuestro camino, al profesor Gabriel Pedraza por la dedicación brindada en su labor como docente y como guía, así como también por brindarnos su confianza al permitirnos realizar este proyecto.

A nuestras familias por brindarnos su apoyo y por motivarnos a ser cada día mejores personas, a todos nuestros compañeros y amigos de la universidad que nos acompañaron en el transcurso de la carrera.

CONTENIDO

pág.

LISTA DE TABLAS

pág.

LISTA DE FIGURAS

pág.

LISTA DE ANEXOS

pág.

RESUMEN

TÍTULO: EVALUACIÓN DEL RENDIMIENTO DE DIFERENTES MODELOS DE PROGRAMACIÓN WEB UTILIZADOS EN DIVERSOS CONTEXTOS[*]

AUTORES: VIVIANA ANDREA MALDONADO BELTRÁN, EDWIN ALFONSO VESGA ARIAS[**]

PALABRAS CLAVE: PROGRAMACIÓN SÍNCRONA, PROGRAMACIÓN ASÍNCRONA, ESCENARIOS DE PRUEBA, PRUEBAS DE RENDIMIENTO, INDICADOR CLAVE DE RENDIMIENTO, PRUEBAS DE CARGA, PRUEBAS DE PICOS, GRUPO DE HILOS, TIEMPO DE RESPUESTA, LATENCIA, RENDIMIENTO.

DESCRIPCIÓN:

Se evalúa el rendimiento de las aplicaciones implementadas en el modelo de programación síncrono y el modelo de programación asíncrono. Para su desarrollo se definen los requerimientos y funcionalidades de las aplicaciones que se implementan en las tecnologías seleccionadas: Vert.x, Java Servlets con Tomcat y Jetty, y Node.js. La metodología planteada busca definir el entorno de ejecución apropiado para las pruebas de rendimiento, donde se evalúan cada una de las aplicaciones tipo desarrolladas en las diferentes tecnologías simulando las solicitudes de los usuarios por medio de la herramienta de pruebas de carga JMeter. El entorno de ejecución de las aplicaciones está comprendido por servidores alojados en la nube de AWS, y el despliegue es automatizado por medio de contenedores Docker. Luego, se ejecutan las pruebas de rendimiento según las diferentes configuraciones planteadas por los componentes de la prueba (período, grupo de hilos, niveles de carga, entre otros), adicionalmente se implementan pruebas de carga con diferentes límites de concurrencia para cada una de las tecnologías y una variante de la prueba de estrés, conocida como prueba de picos donde la aplicación pasa por cambios drásticos en la concurrencia de usuarios. De los reportes obtenidos se analizan los indicadores clave de rendimiento, los más conocidos para las aplicaciones web son: el número de usuarios, el tiempo de respuesta, el rendimiento, el consumo de recursos, el número de muestras iniciadas por segundo, entre otros. Al realizar el análisis en función de estos indicadores, se puede observar la eficiencia que presenta cada modelo de programación en los escenarios de prueba, así como también se pueden tomar decisiones sobre el desarrollo y el mantenimiento de la aplicación.

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico- Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: PhD. Gabriel Rodrigo Pedraza Ferreira

ABSTRACT

TITLE: EVALUATION OF THE PERFORMANCE OF DIFFERENT WEB PROGRAMMING MODELS USED IN VARIOUS CONTEXTS[*]

AUTHORS: VIVIANA ANDREA MALDONADO BELTRÁN, EDWIN ALFONSO VESGA ARIAS[**]

KEYWORDS: SYNCHRONOUS PROGRAMMING, ASYNCHRONOUS PROGRAMMING, TEST SCENARIOS, PERFORMANCE TESTS, KEY PERFORMANCE INDICATOR, LOAD TESTS, SPIKE TESTS, THREAD GROUP, RESPONSE TIME, LATENCY, PERFORMANCE.

DESCRIPTION:

The performance of the applications implemented in the synchronous programming model and the asynchronous programming model is evaluated. For its development, the requirements and functionalities of the applications that are implemented in the selected technologies are defined: Vert.x, Java Servlets with Tomcat and Jetty, and Node.js. The proposed methodology seeks to define the appropriate execution environment for the performance tests, where each of the typical applications developed in the different technologies is evaluated by simulating the requests of the users by means of the JMeter load testing tool. The execution environment of the applications is comprised of servers hosted in the AWS cloud, and the deployment is automated through Docker containers. Then, the performance tests are executed according to the different configurations proposed by the test components (period, group of threads, load levels, among others), load tests are also implemented with different limits of concurrence for each of the technologies and a variant of the stress test, known as peak test where the application goes through drastic changes in the concurrency of users. From the reports obtained, the key performance indicators are analyzed, the best known for web applications are: the number of users, the response time, the performance, the consumption of resources, the number of hits per second, among others. By performing the analysis based on these indicators, you can see the efficiency of each programming model in the test scenarios, as well as decisions on the development and maintenance of the application.

[*] Undergraduate final Project

[**] School of Physical-Mechanical Engineering. Department of Systems Engineering and Informatics. Advisor: PhD. Gabriel Rodrigo Pedraza Ferreira

INTRODUCCIÓN

La evaluación del rendimiento es una de las actividades clave de los equipos de desarrollo software para determinar la capacidad de respuesta, el consumo de recursos y la eficiencia y eficacia de las aplicaciones Web, siendo parte de una gestión de control de calidad. Entonces, es necesario plantear una metodología que provea agilidad y eficiencia a la hora de implementar las pruebas en un entorno definido, para poder enmarcar aquellos factores de riesgo que limitan el rendimiento de las aplicaciones Web.

Estos factores de riesgo normalmente son detectados empleando pruebas de rendimiento, que brindan soporte a la parte encargada del desarrollo del software. Sin embargo, es importante entender que siempre existirá un factor de riesgo en una aplicación web, y que las pruebas de rendimiento solo son un mecanismo de control, empleado por el equipo de desarrollo para entender en un ámbito real, un software que ha sido diseñado desde una perspectiva ideal. Debido a esto, las pruebas de rendimiento son necesarias durante las etapas de desarrollo y mantenimiento de todo tipo de software, y en particular en las aplicaciones Web.

Con la aparición de nuevos modelos y plataformas de desarrollo, la arquitectura de las aplicaciones web se está volviendo cada vez más compleja, y con el fin de proveer mecanismos para aprovechar los recursos computacionales y ofrecer mejor rendimiento en las aplicaciones Web, se han propuesto diversos modelos de programación que buscan ofrecer nuevas alternativas tecnológicas a los desarrolladores, así como nuevos paradigmas de programación. Entre estos modelos se pueden mencionar el modelo de programación síncrono y el modelo de programación asíncrono; estos se diferencian en la forma en que los hilos manejan los procesos, siendo para el caso síncrono un manejo de procesos de

forma secuencial, y para el modelo asíncrono un manejo de procesos mucho más complejo donde entran conceptos como las promesas, los callbacks, entre otros.

En el presente trabajo se busca evidenciar y evaluar el comportamiento del modelo de programación según el tipo de aplicación Web implementada, teniendo en cuenta para este proyecto aquellas que manejan operaciones intensivas de I/O (lectura y escritura) o intensivas en cómputo, por medio de la ejecución de diferentes pruebas de rendimiento desarrolladas en un entorno controlable.

El desarrollo del proyecto inicia con el planteamiento de las aplicaciones a implementar, escogiendo diferentes tecnologías que implementan cada modelo de programación, y en cada una de estas se desarrollan las funcionalidades y requerimientos planteados. Luego, se planifica el entorno en el cual se ejecutan las pruebas, en este punto, se definen las variables que enmarcan los escenarios que se simulan durante las pruebas. Después de definir los escenarios, se plantea el tipo de pruebas a desarrollar, porque dentro de las pruebas de rendimiento existe una gran variedad de configuraciones y de estas se deben elegir las que permitan obtener los mejores indicadores de rendimiento. Finalmente, se ejecutan las pruebas planteadas y se analizan los resultados de cada prueba; según lo observado durante el análisis se pueden tomar muchas decisiones, las cuales están enfocadas al desarrollo y al mantenimiento de las aplicaciones.

1. DEFINICIÓN DEL PROBLEMA

La arquitectura de las aplicaciones web ha ido evolucionando constantemente según las necesidades de los usuarios. En un comienzo estas aplicaciones se diseñaron bajo un modelo monolítico, donde todo el procesamiento lógico y la interfaz de usuario se implementan en un único módulo. Estas aplicaciones estaban enfocadas en atender pocos usuarios dado que las redes apenas estaban surgiendo, y en medio de los crecientes cambios tecnológicos, las aplicaciones web fueron desarrollando nuevas arquitecturas que las impulsaron al enfoque multiusuario que se conoce hoy en día.

Los servicios web surgen aprovechando un conjunto de protocolos FTP, HTTP, SMTP1 [1], entre otros, para facilitar el intercambio de datos entre aplicaciones web implementadas en distintas plataformas. La mayoría de estos protocolos están enfocados en un modelo de comunicación cliente-servidor, siendo el protocolo HTTP uno de los principales dado su gran uso en los navegadores web, produciéndose un aumento en la cantidad de datos que navegan a través de la red y volviéndose tendencia en la mayoría de las organizaciones.

Actualmente, el desarrollo de aplicaciones web posee una arquitectura que está basada en microservicios. Este tipo de arquitectura promueve el desarrollo y despliegue de aplicaciones compuestas por unidades independientes, autónomas, modulares y auto-contenidas, algo muy diferente a la forma tradicional o monolítica. Además, los microservicios tienen un enfoque de desarrollo que consiste en construir una aplicación como un conjunto de servicios pequeños, ejecutándose cada uno en su propio proceso y realizando la comunicación entre

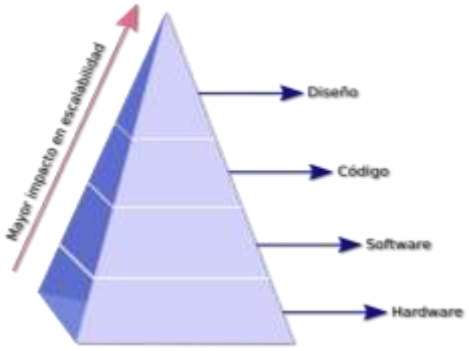
[1] ALCÓCER GARCÍA, Alejandro Carlos. Redes de computadoras: Servicios de aplicación: TELNET, FTP, TFTP, NFS, SMTP, POP3, IMAP4, WWW Y GOPHER. [En línea]. Perú: Infolink, 2000. p. 318 - 337. (Recuperado en 16 marzo 2018). Disponible en http://repositorio.pucp.edu.pe/index/bitstream/handle/123456789/28691/Redes_Cap22.pdf?sequence=22

ellos, con mecanismos ligeros como una API de recursos HTTP. Estos servicios implementan una funcionalidad completa y pueden usar diferentes tecnologías de almacenamiento de datos [2].

Por otra parte, la capacidad de las aplicaciones web de responder a distintas exigencias, determinará el éxito o fracaso que estas tendrán a lo largo de sus etapas de crecimiento. Ya que a veces al diseñar un software no se tiene en cuenta su capacidad de crecimiento, se pueden presentar factores de riesgo como inestabilidad en su funcionamiento, o incluso la interrupción de su ejecución dado que no es capaz de soportar la creciente carga. La propiedad que se relaciona directamente con la capacidad de crecimiento de un software es la escalabilidad y ha sido de gran importancia durante los últimos años, su estudio permite a los equipos de desarrollo diseñar software capaz de adaptarse a condiciones de operación exigentes, responder a tiempo y de forma satisfactoria a los requerimientos de los clientes.

La escalabilidad está determinada por las características del hardware, el software, el código y el diseño. En la Figura 1 se puede observar el impacto de cada uno de estos factores en la escalabilidad de un sitio web.

Figura 1. Factores que determinan la escalabilidad de un sitio web.



[2] ÁLVAREZ, Esaú. Qué son Microservicios y ejemplos reales de uso. [En línea]. En: OpenWebinars, 2016. (Recuperado en 16 marzo 2018). Disponible en https://openwebinars.net/blog/microservicios-que-son/

Se plantean entonces, varios métodos para poder asegurar la escalabilidad de un sistema, entre ellos se encuentran la escalabilidad horizontal, la escalabilidad vertical, los balanceadores de carga y el desarrollo adecuado de aplicaciones basándose en modelos. Por ejemplo, un sistema escala horizontalmente cuando se agregan más equipos de hardware y se despliegan las aplicaciones en cada uno de ellos. En la escalabilidad vertical se cambia la infraestructura hacia un hardware más potente de lo necesario. En cuanto a los balanceadores de carga, son usados para dividir el trabajo a compartir entre varios procesos, ordenadores, u otros recursos [3]. Los primeros tres métodos se enfocan principalmente en optimizar el hardware del sistema y como se ha dicho con anterioridad, estos métodos poseen distintos tipos de limitaciones, dentro de las cuales se pueden encontrar: las limitaciones físicas, que son aquellas donde se presentan límites en la potencia que puede proveer el hardware, entre otros; limitaciones en el diseño, donde influye la complejidad del tipo de escalabilidad a implementar; por último, limitaciones en los costos, ya sea costos en mantenimiento o costos de adquisición, siempre son factores decisivos a la hora de elegir el tipo de escalabilidad.

La selección de tecnologías y herramientas es una de las principales habilidades que debe tener un ingeniero al desarrollar un proyecto. En el diseño y desarrollo de aplicaciones web, se deben estudiar y elegir las tecnologías que faciliten la implementación y garanticen el mantenimiento. En la actualidad, las tecnologías computacionales buscan dar solución a las necesidades de las organizaciones y una de las más importantes es la tecnología de Cliente/Servidor. Además, se cuenta con variedad de herramientas para el desarrollo de aplicaciones web como entornos de desarrollo integrado, manejadores de bases de datos, entre otros.

[3] SIGCHO GONZÁLEZ, Fabiola Jacqueline. Análisis e implementación de soluciones con base de datos NoSQL para el manejo de Big Data, aplicando técnicas de escalabilidad y tolerancia a fallos. [En línea]. Trabajo de grado Ingeniero en sistemas informáticos y computación. Loja: Universidad Técnica Particular de Loja, 2017. p. 12 - 15. (Recuperado en 16 marzo 2018). Disponible en http://dspace.utpl.edu.ec/bitstream/20.500.11962/20658/1/Sigcho%20Gonz%C3%A1lez%2c%20Fabiola%20Jacqueline.pdf

El proyecto se enfoca en la evaluación de los modelos de programación Web existentes, con el fin de caracterizar el comportamiento de estos modelos desde el punto de vista del rendimiento de la aplicación. La evaluación busca determinar cómo se comportan estos modelos de programación bajo diversos contextos y cuando las aplicaciones a desarrollar tienen diversas propiedades.

2. OBJETIVOS

2.1 OBJETIVO GENERAL

Evaluar el rendimiento de diferentes modelos de programación web utilizados en diversos contextos.

2.2 OBJETIVOS ESPECÍFICOS

- Caracterizar el tipo de aplicaciones que se implementarán en los diferentes modelos web.
- Establecer un marco de referencia para comparar el rendimiento de los modelos web.
- Diseñar y ejecutar un conjunto de pruebas para establecer el rendimiento utilizando el marco de referencia.
- Analizar los resultados de las pruebas y evaluar el rendimiento de cada modelo web.

3. MARCO TEÓRICO

3.1 EVALUACIÓN TECNOLÓGICA

La evaluación tecnológica es un ejercicio analítico que consiste en valorar información y resultados experimentales en función del impacto que tienen sobre la tecnología de la organización, tanto la desarrollada internamente como la que pueda adquirirse externamente. Para poder realizar una evaluación tecnológica se requiere el establecimiento de ciertas pautas o estándares, los cuales, bajo un marco de trabajo adecuado, permitirán que el resultado obtenido en la evaluación no esté sesgado bajo las premisas u opiniones de quienes realizan dicha evaluación. La evaluación se da por medio de una metodología que una vez conformada, establecerá las métricas de interés que estarán interrelacionadas en los sistemas que serán objetivo de estudio y que bajo un mismo contexto de evaluación dará como resultado un análisis [4].

3.2 MEDICIÓN Y EVALUACIÓN DEL DESEMPEÑO

Los principales propósitos de la medición y evaluación del desempeño son ayudar en el diseño de hardware y software, ayudar en la selección de un sistema informático y mejorar el desempeño de un sistema existente. La medición y evaluación del rendimiento de un sistema informático es difícil debido a la complejidad de su estructura interna y a la dificultad de describir y predecir la carga de trabajo. Es necesario caracterizar la carga en un sistema para poder hacer declaraciones significativas sobre su desempeño. Un aspecto de este problema es determinar qué características de la carga definen con mayor

[4] BELLIDO, Félix y GÓMEZ FONTANILLS, David. Herramientas de Evaluación Tecnológica en Gestión de la tecnología. [En línea]. En: Wiki EOI, 2012. (Recuperado en 17 marzo 2018). Disponible en http://www.eoi.es/wiki/index.php/Herramientas_de_Evaluaci%C3%B3n_Tecnol%C3%B3gica_en_Gesti%C3%B3n_de_la_tecnolog%C3%ADa

precisión las medidas de rendimiento de interés. Otro, es determinar los valores de los parámetros del modelo de carga de trabajo para un estudio de rendimiento particular, y es particularmente difícil si el sistema aún no está operativo. Pero incluso con un sistema operacional, la carga de trabajo puede variar con el tiempo y las características de la carga de trabajo medidas dependen del período de medición elegido [5].

3.3 PRUEBAS DE RENDIMIENTO

Una aplicación con buen rendimiento permite al usuario final realizar una tarea determinada sin demora. La prueba de rendimiento se define como la investigación técnica realizada para determinar o validar las características de velocidad, escalabilidad o estabilidad del producto bajo prueba. Las actividades relacionadas con el rendimiento, como las pruebas y el ajuste, tienen que ver con lograr tiempos de respuesta, rendimiento y niveles de utilización de recursos que cumplan los objetivos de rendimiento para la aplicación bajo prueba [6].

3.3.1 Tipos de pruebas de rendimiento. El objetivo de las pruebas de rendimiento es determinar el rendimiento del sistema bajo una carga de trabajo definida utilizando diferentes tipos de pruebas de rendimiento. En la Tabla 1 se encuentran las pruebas de rendimiento más comunes para aplicaciones web.

[5] MUNTZ, Richard. Performance Measurement and evaluation. En: Encyclopedia of Computer Science. Enero, 2003, p. 1385-1390.

[6] BANSODE, Prashant, et al. Performance Testing Guidance for Web Applications. [En línea]: Chapter 2 – Types of Performance Testing. En: Microsoft, 2010. (Recuperado en 17 marzo 2018). Disponible en https://msdn.microsoft.com/en-us/library/bb924357.aspx

Tabla 1. Tipos de pruebas de rendimiento.

Término	Propósito
Prueba de rendimiento	Para determinar o validar velocidad, escalabilidad y/o estabilidad.
Prueba de carga	Para verificar el comportamiento de la aplicación en condiciones de carga máxima y normal.
Prueba de estrés	Para determinar o validar el comportamiento de una aplicación cuando se empuja más allá de las condiciones normales o de carga máxima.
Prueba de capacidad	Para determinar cuántos usuarios y/o transacciones soportará un sistema determinado y aun así cumplir los objetivos de rendimiento.

Fuente: BANSODE, Prashant, et al. Performance Testing Guidance for Web Applications. [En línea]: Chapter 2 – Types of Performance Testing. En: Microsoft, 2010. (Recuperado en 17 marzo 2018). Disponible en https://msdn.microsoft.com/en-us/library/bb924357.aspx

3.4 MODELOS DE PROGRAMACIÓN

Entre los diferentes modelos de programación existentes se encuentra el modelo de programación síncrono, siendo uno de los más tradicionales que consiste en que una orden solo se puede ejecutar luego que se ejecuta la anterior. El problema de este modelo es que los procesos ocupan más memoria, y son menos eficientes. Por su parte, el modelo de programación asíncrono se basa en llamadas que pueden ser cumplidas ahora o en un futuro. Es decir, las variables pueden ser asignadas en cualquier momento de la ejecución del programa. Adicional a esto, se encuentra el modelo de programación reactiva, que se enfoca en el manejo de streams de datos asíncronos y la propagación del cambio. A

continuación se muestra un explicación más detallada de cada uno de los modelos de programación: síncrono, asíncrono y reactivo.

3.4.1 Modelo de programación síncrono. La programación síncrona o también conocida como programación secuencial es aquella en la que un programa ejecuta de forma ordenada una secuencia de instrucciones y en la que no se da comienzo a la ejecución de un nuevo proceso hasta no haber terminado con el proceso anterior. Algunas de las ventajas de la programación síncrona son: la fácil implementación dado su uso extendido a lo largo de los años y su buen funcionamiento en sistemas de poca escalabilidad. Sin embargo, llega a ser poco eficiente al aumentar la escalabilidad del sistema, por lo que supone un mayor costo computacional que el necesario en la programación asíncrona [7].

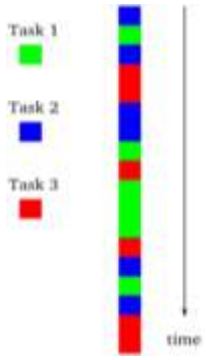
La diferencia entre la ejecución síncrona y asíncrona puede parecer un poco confusa al principio. En la ejecución de un programa síncrono se llama a una función y la ejecución del programa espera hasta que esa función regrese algún valor o termine de ejecutarse antes de continuar con la siguiente línea de código, es decir, el programa se ejecuta línea por línea. Esto puede tener ramificaciones indeseables. Con la ejecución síncrona, el programa se puede quedar "atascado", esperando que algún proceso finalice, sin salida. La ejecución asíncrona a evita este cuello de botella. Básicamente, se busca que en la llamada a una función que va a llevar mucho tiempo, el programa no tenga que esperarla mientras se ejecuta [8].

[7] VÉLEZ REYES, Javier. Programación asíncrona: paso de continuadores, eventos, promesas y generadores. [En línea]. En: TodoJS, 2015. (Recuperado en 19 marzo 2018). Disponible en https://www.todojs.com/programacion-asincrona-paso-de-continuadores-eventos-promesas-y-generadores/

[8] SYNCHRONOUS VS. Asynchronous Execution [Anónimo]. [En línea]. En: Software Bisque. (Recuperado en 01 abril 2018). Disponible en https://www.bisque.com/products/orchestrate/RASCOMHelp/RASCOM/Synchronous_vs_Asynchronous_Execution.htm

3.4.2 Modelo de programación asíncrono. Para ciertas aplicaciones, un modelo asíncrono puede producir beneficios de rendimiento sobre el multihilo tradicional. En este modelo, las tareas se entrelazan entre sí, pero en un solo hilo de control [9] (ver Figura 2).

Figura 2. Modelo asíncrono.



Fuente: FONSECA, Rodrigo. Introduction to Asynchronous Programming. [En línea]. En: Computer Science at Brown University. p. 1 - 6. (Recuperado en 19 marzo 2018). Disponible en http://cs.brown.edu/courses/cs168/s12/handouts/async.pdf

La programación asíncrona es una técnica para escribir aplicaciones que contienen tareas que se pueden ejecutar de forma asincrónica, lo que puede mejorar el rendimiento de la aplicación y capacidad de respuesta de la GUI en aplicaciones con tareas de larga ejecución o uso intensivo de cómputo [10]. Esta programación tiene demanda en la actualidad porque la capacidad de respuesta es cada vez más importante en todos los dispositivos modernos: computadores de escritorio, dispositivos móviles o aplicaciones web. Por lo tanto, los principales

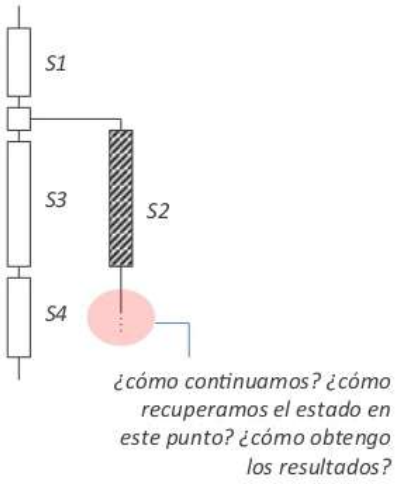
[9] FONSECA, Rodrigo. Introduction to Asynchronous Programming. [En línea]. En: Computer Science at Brown University. p. 1 - 6. (Recuperado en 19 marzo 2018). Disponible en http://cs.brown.edu/courses/cs168/s12/handouts/async.pdf

[10] DEITEL, Paul y DEITEL, Harvey. Visual C# How to Program: Chapter 28 Asynchronous Programming with async and await. 5 ed. Upper Saddle River, NJ: Pearson Education, 2014. p. 1 - 14.

lenguajes de programación tienen una API que admite operaciones asíncronas sin bloqueo (por ejemplo, para acceder a la web o para operaciones de archivos) [11].

A pesar de esto, en los lenguajes asíncronos la depuración no es tan sencilla. El principal problema de la programación asíncrona se refiere a cómo dar continuidad a las operaciones no bloqueantes del algoritmo una vez que éstas han terminado su ejecución (ver Figura 3) [12]. Un código puede que compile correctamente y puede que no funcione como lo pensamos. Puede que tengamos una función que depende de otra y se ejecuta cuando no debería hacerlo.

Figura 3. Desventajas de la programación asíncrona.



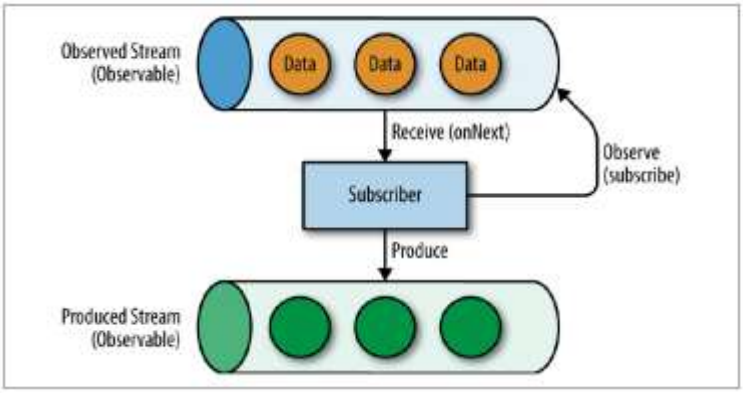
Fuente: VÉLEZ REYES, Javier. Programación Asíncrona en Node JS [diapositivas]. Slideshare. 29 de mayo de 2014, diapositivas 1-53. (Recuperado en 19 marzo 2018). Disponible en https://es.slideshare.net/jvelez77/presentacion-35264918

[11] OKUR, Semih, et al. A Study and Toolkit for Asynchronous Programming in C#. En: Proceedings of the 36th International Conference on Software Engineering. Mayo - junio, 2014. p. 1 - 12.

[12] VÉLEZ REYES, Javier. Programación Asíncrona en Node JS [diapositivas]. Slideshare. 29 de mayo de 2014, diapositivas 1-53. (Recuperado en 19 marzo 2018). Disponible en https://es.slideshare.net/jvelez77/presentacion-35264918

3.4.3 Modelo de programación reactiva. La programación reactiva es un paradigma enfocado en el trabajo con flujos de datos finitos o infinitos de manera asíncrona. En otras palabras, si un programa propaga todos los cambios que modifican sus datos a todas las partes interesadas (usuarios, otros programas, componentes y sub-partes), entonces este programa se puede llamar reactivo [13]. Reactivo se puede definir como "mostrar una respuesta a un estímulo" [14]. Esto quiere decir que el software reactivo reacciona y adapta su comportamiento en función de los estímulos que recibe. Por lo tanto, la programación reactiva se puede definir como un modelo de desarrollo centrado en la observación de flujos de datos, reaccionando a los cambios y propagándolos (ver Figura 4) [15].

Figura 4. La programación reactiva se trata sobre el flujo de datos y su reacción.



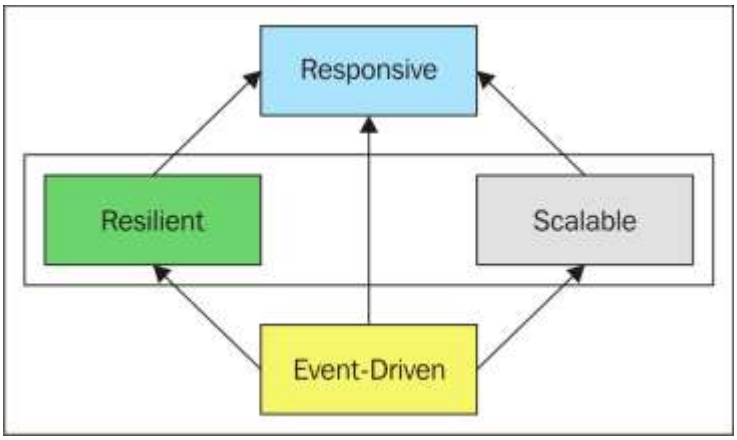
Fuente: ESCOFFIER, Clement. Building Reactive Microservices in Java Asynchronous and Event-Based Application Design. Sebastopol: O'Reilly Media, 2017. p. 4.

3.4.3.1 Sistemas reactivos. Un sistema reactivo se puede definir como un estilo de arquitectura utilizado para construir sistemas distribuidos receptivos y robustos

[13] TSVETINOV, Nickolay. Learning Reactive Programming With Java 8. Kindle ed. Packt Publishing, 2015, p. 2 - 4.
[14] OXFORD UNIVERSITY PRESS. Reactive. [En línea]. En: Oxford Dictionaries. (Recuperado en 18 de marzo 2018). Disponible en https://en.oxforddictionaries.com/definition/reactive
[15] ESCOFFIER, Clement. Building Reactive Microservices in Java Asynchronous and Event-Based Application Design. Sebastopol: O'Reilly Media, 2017. p. 3-4.

basados en el paso de mensajes asíncrono. Los sistemas que son construidos como sistemas reactivos son mucho más flexibles y escalables, lo que hace que sean más fáciles de desarrollar y sensibles al cambio. Además, son más tolerantes a fallas (ver Figura 5) [16].

Figura 5. Características de un sistema reactivo.



Fuente: BONÉR, Jonas, et al. The Reactive Manifesto. [En línea]. En: Reactive Manifesto. (Recuperado en 18 marzo 2018). Disponible en https://www.reactivemanifesto.org/

[16] BONÉR, Jonas, et al. The Reactive Manifesto. [En línea]. En: Reactive Manifesto. (Recuperado en 18 marzo 2018). Disponible en https://www.reactivemanifesto.org/

4. HERRAMIENTAS Y TECNOLOGÍAS

Durante el desarrollo del proyecto fue necesario emplear un conjunto de herramientas que permitieran cumplir con los objetivos planteados. Las herramientas necesarias están organizadas según las siguientes actividades:

- Ejecución de las aplicaciones: servidores de aplicaciones.
- Persistencia de las aplicaciones: MySQL.
- Automatización del desarrollo: Maven.
- Despliegue de las aplicaciones: Docker.
- Plataforma Cloud Computing: AWS.
- Ejecución de las pruebas: JMeter.
- Análisis de resultados: Notebooks.

A continuación se mencionan cada una de las herramientas y se mencionan las características que aportaron significativamente al desarrollo del proyecto.

4.1 SERVIDORES DE APLICACIONES WEB

Los servidores de aplicaciones Web gestionan solicitudes de contenido dinámico, tales como los servlets. Entre los más populares usando el modelo de programación síncrono se encuentran Tomcat y Jetty, servidores web y contenedores Servlet/JavaServer Pages que implementan la plataforma Java. Usando una plataforma multilenguaje sobre JVM se encuentra Vert.x, que utiliza el modelo de programación asíncrono y como estándar el uso de Verticles. Por otra parte, Node.js tiene un enfoque similar al de Vert.x, con el mismo modelo de programación pero con la implementación de la plataforma JavaScript. En la Tabla 2 se pueden ver los servidores de aplicaciones Web empleados en el desarrollo del proyecto y sus características.

Tabla 2. Características de los servidores de aplicaciones Web del proyecto.

Nombre	Modelo	Plataforma	Standard
Tomcat	Síncrono	Java	Servlet
Jetty	Síncrono	Java	Servlet
Vert.x	Asíncrono	Multilenguaje (JVM)	Verticle
Node.js	Asíncrono	Javascript	N.A.

4.1.1 Servlet. Es una clase de Java que amplía dinámicamente los servidores habilitados para Java. Proporciona un marco general para los servicios construidos con la implementación del paradigma de solicitud-respuesta [17]. El servlet se usa principalmente en las capas de controlador y sigue siendo el elemento real que se encarga de responder a las solicitudes en las aplicaciones web [18]. Los servlets se ejecutan en una máquina virtual dentro del proceso del servidor, gestionados por un motor de servlets. Cada petición HTTP recibida se procesa en un hilo, e invoca un método del servlet [19]. Por otro lado, el contenedor de servlets controla el ciclo de vida de un servlet, no siendo necesario usar el constructor o el método principal para crear instancias e invocar objetos servlet debido a que el contenedor servlet se responsabiliza por crear instancias y hacer que el objeto esté disponible para su uso, así como destruyéndolo cuando ya no sea necesario.

4.1.2 Apache Tomcat. Es un contenedor web con soporte de Servlets y JSPs que permite el despliegue de aplicaciones web. Catalina es el módulo de Tomcat

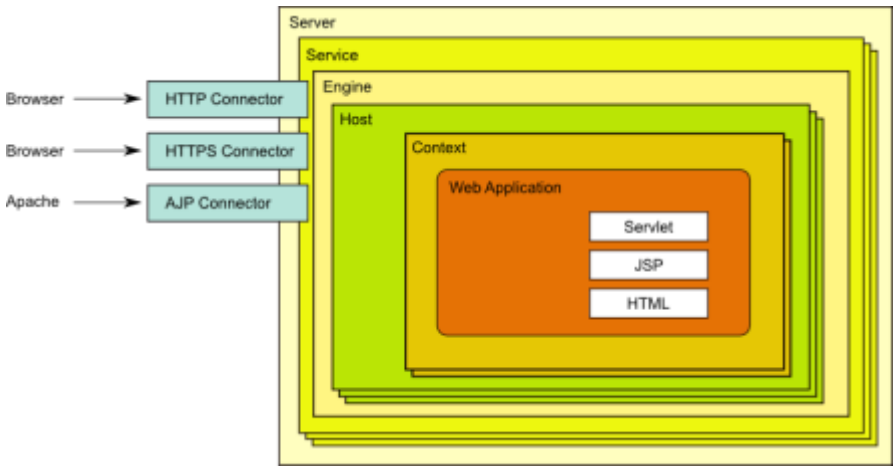
[17] ORACLE. The Java Servlet API White Paper. [En línea] En: Oracle. (Recuperado en 10 septiembre 2018). Disponible en https://www.oracle.com/technetwork/java/whitepaper-135196.html

[18] KULANDAI, Joseph. What is Servlet. [En línea]. En: Javapapers, 2014. (Recuperado en 10 septiembre 2018). Disponible en https://javapapers.com/servlet/what-is-servlet/

[19] ARIAS FISTEUS, Jesús. Servlets. [En línea]. En: Universidad Carlos III de Madrid - Ingeniería Telemática, 2008. (Recuperado en 5 septiembre 2018). Disponible en https://www.it.uc3m.es/labttlat/2007-08/material/servlets/

que proporciona soporte para Servlets y Jasper es el módulo que proporciona soporte para procesar e interpretar ficheros JSP y así generar los Servlets que proporcionan respaldo al respecto [20]. Con la configuración predeterminada, solamente una instancia de Tomcat es creada en el JVM y las solicitudes son manejadas en procesos o hilos independientes. La instancia de Tomcat consiste de un conjunto de contenedores que existen en una jerarquía bien definida. El principal componente en esta jerarquía es el motor de Servlets "Catalina" [21]. La arquitectura de Apache Tomcat se compone principalmente de cinco elementos (ver Figura 6).

Figura 6. Arquitectura de Apache Tomcat.



Fuente: TOMCAT HOSTING with JVM Host - your Java and Tomcat Host [Anónimo]. [En línea] En: JVMHost, 2013. (Recuperado en 13 septiembre 2018). Disponible en https://www.jvmhost.com/tomcat-hosting/

El *Server* representa el motor de Servlets "Catalina" y puede contener uno o más contenedores *Service*. El *Service* mantiene uno o más elementos Connector (normalmente son HTTP, SSL, AJP) y comparte un único elemento *Engine,* que

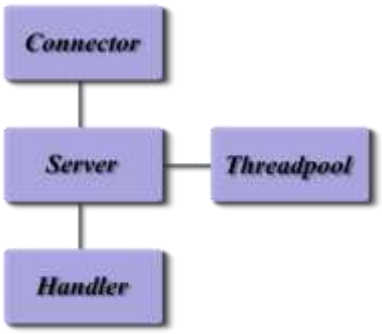
[20] APACHE SOFTWARE FOUNDATION. Apache Tomcat. [En línea]. En: Apache Tomcat, 2019. (Recuperado en 13 septiembre 2018). Disponible en http://tomcat.apache.org/

[21] TOMCAT HOSTING with JVM Host - your Java and Tomcat Host [Anónimo]. [En línea] En: JVMHost, 2013. (Recuperado en 13 septiembre 2018). Disponible en https://www.jvmhost.com/tomcat-hosting/

recibe y procesa todas las solicitudes de los elementos *Connector*, redirigiendo la respuesta al *Connector* indicado para transmitirla al cliente. El *Host* es una asociación del nombre de una red. Cada *Host* puede mantener múltiples aplicaciones web, cada una de ellas representadas por un elemento *Context*. El *Context* representa una aplicación web. No existe un valor límite del número de elementos Context que pueden existir dentro de un elemento Host [22].

4.1.3 Jetty. Es un servidor web de código abierto alojado por la fundación Eclipse desde 2009. Es un servidor HTTP y un contenedor de Servlets que se puede configurar para servir contenido estático y dinámico. Puede ser usado en las aplicaciones utilizando una configuración basada en Java o XML o ejecutándolo directamente desde Maven [23]. La versión actual recomendada para su uso es Jetty 9. La arquitectura principal de Jetty se compone de cuatro componentes (ver Figura 7).

Figura 7. Arquitectura de Jetty.



Fuente: DIRKSEN, Jos. Jetty. [En línea]: A Lightweight, Open-Source Web Server and Servlet Container. Cary: DZone Refcardz, 2012. (Recuperado en 5 septiembre 2018). Disponible en https://dzone.com/refcardz/jetty

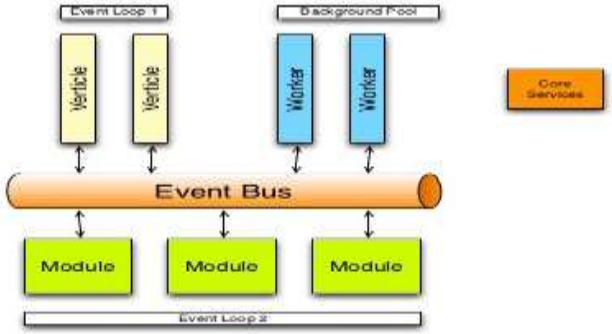
[22] SHAPIRA, Yoav; ARCAND, Jeanfrancois y HANIK, Filip. The Apache Tomcat 5.5 Servlet/JSP Container. [En línea]: Tomcat Architecture. En: Apache Tomcat, 2012. (Recuperado en 14 septiembre 2018). Disponible en https://tomcat.apache.org/tomcat-5.5-doc/architecture/overview.html

[23] ECLIPSE FOUNDATION. Jetty - Servlet Engine and Http Server. [En línea]. En: Eclipse, 2016. (Recuperado en 6 septiembre 2018). Disponible en http://www.eclipse.org/jetty/

El *Server* es la clase contenedor para todos los demás componentes de la arquitectura. Contiene una cantidad de *Connectors* que escuchan en un puerto específico y aceptan conexiones (Ej: HTTP, HTTPS o AJP13) de los clientes. Un *Server* también contiene un conjunto de *handlers* que procesan las solicitudes de los clientes desde los *Connectors* y producen respuestas que se devuelven al cliente. Los hilos recuperados del *threadpool* configurado hacen este proceso. El *threadpool* provee hilos a los handlers, el trabajo hecho por los *Connectors* y los *handlers*. Jetty tiene dos tipos de *threadpools*: "ExecutorThreadPool" y "QueuedThreadPool" [24].

4.1.4 Vert.x. El desarrollo de Vert.x está basado en el proyecto Netty, una librería de redes asíncronas de alto rendimiento para el JVM [25]. Posee una arquitectura dirigida por eventos y está definida principalmente por los *Verticles* y el *Event Bus*; el flujo de eventos que llegan a los Verticles son gestionados por los Event loops (ver Figura 8).

Figura 8. Arquitectura de Vert.x.



Fuente: BLONDEAU, Antoine. FinistDevs April Edition: About Microservices and Infrastructure Automation. [En línea]. En: A Medium Corporation, 2018. (Recuperado en 25 enero 2018). Disponible en https://medium.com/@ant.blondeauw/finistdevs-april-edition-about-microservices-and-infrastructure-automation-664b66d34f48

[24] DIRKSEN, Jos. Jetty. [En línea]: A Lightweight, Open-Source Web Server and Servlet Container. Cary: DZone Refcardz, 2012. (Recuperado en 5 septiembre 2018). Disponible en https://dzone.com/refcardz/jetty

[25] PONGE, Julien; SEGISMONT, Thomas y VIET, Julien. A gentle guide to asynchronous programming with Eclipse Vert.x for Java developers. [En línea]. En: Eclipse Vert.x, 2018. (Recuperado en 23 septiembre 2018). Disponible en https://vertx.io/docs/guide-for-java-devs/#_what_is_vert_x

El *Verticle* es la unidad de desarrollo de las aplicaciones en Vert.x y procesa los eventos entrantes sobre un *Event loop*; estos eventos pueden hacer referencia a cualquier cosa, tal como recibir paquetes de red, eventos periódicos o mensajes enviados por otros *Verticles*. En Vert.x cada *Event loop* es asignado a un hilo y por defecto se asignan dos *Event loop* por núcleo de la CPU de la máquina. Como consecuencia, un *Verticle* regular siempre procesa los eventos en el mismo hilo y no son necesarios mecanismos de coordinación para manipular el estado de un *Verticle* [26].

Por otra parte, el *Event Bus* es la principal herramienta usada por los distintos *Verticles* existentes en una aplicación para poder comunicarse por medio del envío de mensajes asíncronos [27]. Se permite el envío de cualquier tipo de datos, sin embargo, JSON es el formato preferencial dado que permite a los *Verticles* escritos en distintos lenguajes comunicarse entre ellos. Además permite la implementación de los patrones de comunicación como mensajería punto a punto, mensajería solicitud/respuesta y mensajería publicar/subscribir [28]. El código bloqueante en Vertx se gestiona con los *Workers* que funcionan de forma similar a un *Verticle* normal, pero no se ejecutan en un *Event loop*, ya que son asignados a un pool de hilos destinados al manejo de este tipo de procesos.

4.1.5 Node.js. Concebido como un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node.js está diseñado para construir aplicaciones en red escalables. Está basado en el motor V8 de Javascript de Google. Este motor está diseñado para correr en un navegador y ejecutar el código de Javascript de una

[26] PONGE, Julien; SEGISMONT, Thomas y VIET, Julien. A gentle guide to asynchronous programming with Eclipse Vert.x for Java developers. [En línea]. En: Eclipse Vert.x, 2018. (Recuperado en 23 septiembre 2018). Disponible en https://vertx.io/docs/guide-for-java-devs/#_core_vert_x_concepts

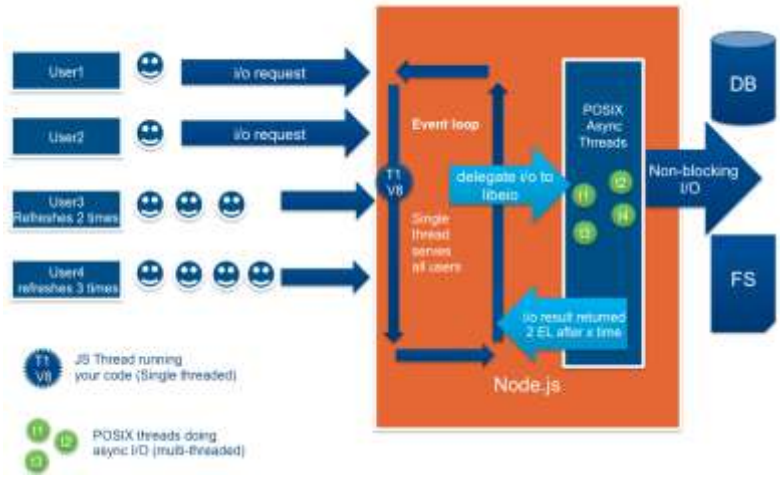
[27] BLONDEAU, Antoine. FinistDevs April Edition: About Microservices and Infrastructure Automation. [En línea]. En: A Medium Corporation, 2018. (Recuperado en 25 enero 2018). Disponible en https://medium.com/@ant.blondeauw/finistdevs-april-edition-about-microservices-and-infrastructure-automation-664b66d34f48

[28] PONGE. Loc. Cit.

forma extremadamente rápida [29]. Con la creación de Node.js, los desarrolladores ahora pueden usar el mismo lenguaje, JavaScript, tanto en el cliente como en el servidor. Es usado comúnmente con el Node Package Manager (NPM) y el framework Express.js.

La arquitectura de Node.js (ver Figura 9) se basa en manejar eventos utilizando el *Event loop* y ejecutar JavaScript del lado del servidor. Node.js sigue el modelo de *Single Thread* con *Event Loop* y con mecanismo de devolución de llamada (callback) de Javascript. Son muchas las ventajas, como la facilidad de manejar más y más solicitudes de clientes concurrentes sin necesidad de crear más threads, debido al *Event Loop*. Por lo tanto, al hacer uso de menos threads, Node.js utiliza menos recursos o memoria, y debido a su arquitectura de *Single Thread* con *Event loop*, los desarrolladores también pueden realizar operaciones intensivas de I/O sin bloqueo.

Figura 9. Arquitectura de Node.js.



Fuente: POSA, Rambabu. Node JS Architecture - Single Threaded Event Loop. [En línea]. En: JournalDev, 2015. (Recuperado en 15 septiembre 2018). Disponible en https://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop

[29] ÁLVAREZ, Cecilio. ¿Cómo funciona Node.js?. [En línea]. En: Genbeta, 2014. (Recuperado en 9 septiembre 2018). Disponible en https://www.genbeta.com/desarrollo/como-funciona-node-js

El *Event loop* además comprueba que cualquier solicitud del cliente se coloque en la cola de eventos (*Event Queue*). Si no, espera las solicitudes entrantes de forma indefinida siendo su propiedad asíncrona muy importante para el rendimiento en Node.JS, ya que no espera a que el código se complete antes de procesar otras solicitudes. Cada solicitud colocada en la parte superior del *Event loop* se maneja por separado de manera asíncrona. Aunque es *Single Thread* para procesar eventos, es más que suficiente porque la única responsabilidad de este *Thread* es delegar las solicitudes entrantes a los *threads asíncronos*. Esos *threads asíncronos* delegan las operaciones de I/O a operaciones sin bloqueo para que no bloqueen otras solicitudes. Aunque en teoría es una arquitectura sin bloqueo, hay casos en los que podría privar de recursos al entorno y eso se limita principalmente a la CPU. Debido a que hay un solo hilo, si la CPU no tiene ciclos disponibles, el *Event loop* no podrá procesar más eventos y tendrá que esperar hasta que la CPU esté disponible. Esto se puede resolver generando varios *Workers* que se pueden distribuir sobre las CPU para que pueda utilizar el entorno de varios núcleos [30].

4.2 MYSQL SERVER

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base datos de código abierto más popular del mundo, y una de las más populares junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web [31]. MySQL se puede ejecutar en plataformas de computación en la nube como Microsoft Azure, Amazon EC2 y Oracle Cloud Infrastructure. En aplicaciones web hay baja concurrencia en la modificación de

[30] POSA, Rambabu. Node JS Architecture - Single Threaded Event Loop. [En línea]. En: JournalDev, 2015. (Recuperado en 15 septiembre 2018). Disponible en https://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop

[31] DB-Engines Ranking. [En línea]. En: DB-Engines, 2018. (Recuperado en 18 septiembre 2018). Disponible en http://db-engines.com/en/ranking

datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones. Sea cual sea el entorno en el que se va a utilizar, es importante monitorizar de antemano el rendimiento para detectar y corregir errores tanto de SQL como de programación [32].

4.3 APACHE MAVEN

Apache Maven es una herramienta de software para la gestión y construcción de proyectos Java. Su funcionalidad es similar a la de Apache Ant, pero cuenta con un modelo de configuración de construcción más simple basado en un formato XML. Actualmente, es un proyecto de nivel superior de Apache Software Foundation [33]. Por medio de un Project Object Model (POM), Maven separa limpiamente el código del proyecto de los archivos de configuración, la documentación y las dependencias. También ofrece una plataforma flexible que por medio de plugins facilita la adquisición de complementos de aplicaciones sin importar el lenguaje de implementación [34].

4.4 DOCKER

Docker es un proyecto de código abierto que permite desplegar aplicaciones de forma automática por medio del uso de contenedores de software. Proporciona una capa adicional de abstracción de la virtualización de aplicaciones que es compatible en múltiples sistemas operativos [35]. Al ser muy sencillo gestionar

---

[32] MTOP: MONITOREO de carga en MySQL [Anónimo]. [En línea]. Gabriel Barrera. 28 de octubre de 2009. (Recuperado en 18 septiembre 2018). Disponible en http://tecnocacharrero.blogspot.com/2009/10/mtop-monitoreo-de-carga-en-mysql.html

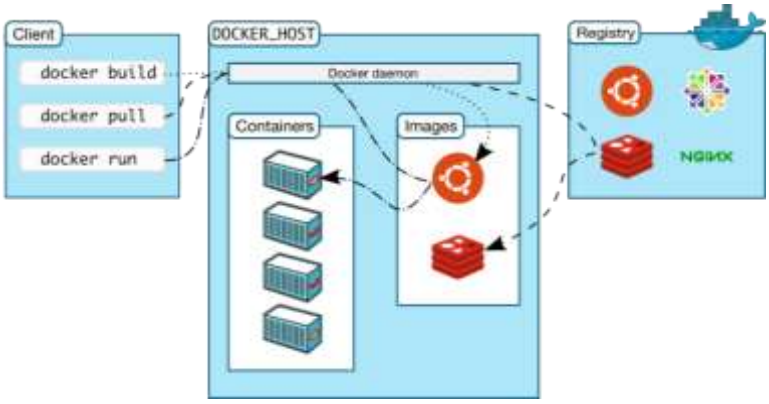
[33] APACHE SOFTWARE FOUNDATION. Maven – Introduction. [En línea]. En: Apache Maven Project, 2019. (Recuperado en 19 septiembre 2018). Disponible en https://maven.apache.org/what-is-maven.html

[34] REDMOND, Eric. The Maven 2 POM demystified. [En línea]. En: JavaWorld, 2006. (Recuperado en 19 septiembre 2018). Disponible en https://www.javaworld.com/article/2071772/java-app-dev/the-maven-2-pom-demystified.html

[35] WALSH, Daniel. ¿Qué es DOCKER?. [En línea]. En: Red Hat. (Recuperado en 20 septiembre 2018). Disponible en https://www.redhat.com/es/topics/containers/what-is-docker

contenedores con Docker siendo estos bastante ligeros, son muy adecuados para desplegar entorno de pruebas donde se pueda hacer el testing [36]. Docker usa una arquitectura cliente-servidor (ver Figura 10).

Figura 10. Arquitectura de Docker.



Fuente: MUÑOZ, José Domingo. Introducción a docker. [En línea]. En: PLEDIN 3.0, 2015. (Recuperado en 20 septiembre 2018). Disponible en https://www.josedomingo.org/pledin/2015/12/introduccion-a-docker/

El *cliente* de Docker habla con el *Daemon* de Docker que hace el trabajo de crear, correr y distribuir los contenedores. Tanto el *cliente* como el *Daemon* pueden ejecutarse en el mismo sistema, o se puede conectar un *cliente* de Docker remoto a un Daemon de Docker [37]. Docker está formado fundamentalmente por el *cliente*, el *daemon*, los *registros* y los *objetos*.

El *daemon* (dockerd) escucha las solicitudes de la API de Docker y administra los objetos de Docker, como imágenes, contenedores, redes y volúmenes. También puede comunicarse con otros *daemons* para administrar los servicios de Docker.

[36] MUÑOZ, José Domingo. Introducción a docker. [En línea]. En: PLEDIN 3.0, 2015. (Recuperado en 20 septiembre 2018). Disponible en https://www.josedomingo.org/pledin/2015/12/introduccion-a-docker/

[37] SOTO, Jason. Arquitectura de Docker. [En línea]. En: GitBook - Jsitech1, 2016. (Recuperado en 20 septiembre 2018). Disponible en https://jsitech1.gitbooks.io/meet-docker/content/arquitectura_de_docker.html

El *cliente* (docker) es la forma principal en que muchos usuarios interactúan con Docker. Cuando se usan comandos como 'docker run', el *cliente* envía estos comandos a 'dockerd' quien los ejecuta. Por otro lado, los *registros* de Docker guardan las imágenes. Son repositorios públicos o privados donde se pueden subir o descargar imágenes [38]. El *registro* público del proyecto se llama Docker Hub y es el componente de distribución de Docker. Finalmente, se encuentran los objetos de Docker como imágenes, contenedores, redes, volúmenes y/o complementos.

4.4.1 Docker Compose. Docker Compose es una herramienta para definir y correr aplicaciones multicontenedores. Se utiliza un archivo docker-compose.yml para configurar los servicios que necesita la aplicación. Luego con el uso de un comando, se crean los contenedores necesarios e inician todos los servicios especificados en la configuración. Compose es bueno para el desarrollo, pruebas y flujos de trabajo de integración continua. En el caso de los desarrolladores, tienen la habilidad de correr las aplicaciones en un ambiente aislado e interactuar con la aplicación. El archivo de compose provee una forma de documentar y configurar todas las dependencias de la aplicación y al final solo tiene que hacer uso de un comando para subir todo el ambiente [39].

4.5 APACHE JMETER

Apache Jmeter es un software de código abierto desarrollado en el lenguaje de programación Java, enfocado en el diseño de pruebas funcionales sobre el comportamiento de las aplicaciones para poder medir su rendimiento. Permite simular escenarios de carga sobre algún elemento de prueba, tal como un

[38] MUÑOZ, José Domingo. Primeros pasos con Docker. [En línea]. En: PLEDIN 3.0, 2016. (Recuperado en 21 septiembre 2018). Disponible en https://www.josedomingo.org/pledin/2016/02/primeros-pasos-con-docker/

[39] SOTO, Jason. Docker Compose. [En línea]. En: GitBook - Jsitech1, 2016. (Recuperado en 21 septiembre 2018). Disponible en https://jsitech1.gitbooks.io/meet-docker/content/docker_compose.html

servidor, una red o un grupo de servidores, entre otros [40]. Presenta la posibilidad de cargar y probar muchas aplicaciones diferentes, así como servidores y tipos de protocolos. También, permite cargar pruebas por línea de comandos en cualquier sistema operativo compatible con Java, reduciendo la cantidad de recursos que se consumen durante su ejecución e incrementando la confiabilidad de los resultados obtenidos, y así permitiendo el despliegue de las pruebas sin afectar los escenarios de experimentación. Ofrece un IDE con múltiples funcionalidades para el desarrollo de los distintos planes de prueba. Además, establece un marco de trabajo multihilo, que permite el muestreo concurrente de múltiples hilos y el muestreo simultáneo de distintas funciones en grupos de hilos separados [41].

4.6 JUPYTER NOTEBOOK

Es una aplicación web de código abierto, que permite crear, compartir y editar documentos en los que se puede ejecutar código Python, hacer anotaciones, insertar ecuaciones, visualizar resultados y documentar funcionalidades. Está diseñada generalmente para tener una compatibilidad avanzada con Python, Markdown e incluye la posibilidad de exportar documentos hechos con la herramienta a otros formatos. Generalmente esta herramienta es utilizada para el aprendizaje del lenguaje de programación Python, la limpieza y transformación de datos científicos, la simulación numérica, el modelado estadístico y puede abarcar muchas otras áreas [42].

---

[40] APACHE SOFTWARE FOUNDATION. Apache JMeter. [En línea]. En: Apache JMeter. (Recuperado en 22 septiembre 2018). Disponible en https://jmeter.apache.org/

[41] JUNTA DE ANDALUCÍA. Ejecución de Pruebas con JMeter. [En línea]. España: Marco de Desarrollo de la Junta de Andalucía. (Recuperado en 23 septiembre 2018). Disponible en http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/401

[42] TORO, Luigys. Jupyter notebook: documenta y ejecuta código desde el navegador. [En línea]. Blog DESDELINUX. 21 de septiembre de 2017. (Recuperado en 27 enero 2019). Disponible en https://blog.desdelinux.net/jupyter-notebook/

4.7 AMAZON EC2

Amazon Elastic Compute Cloud (Amazon EC2) proporciona capacidad de computación escalable en la nube de Amazon Web Services (AWS). El uso de Amazon EC2 elimina la necesidad de invertir inicialmente en hardware, de manera que se pueden desarrollar e implementar aplicaciones en menos tiempo. Se puede usar Amazon EC2 para lanzar tantos servidores virtuales como se necesiten, configurar la seguridad y las redes y administrar el almacenamiento. Amazon EC2 permite escalar hacia arriba o hacia abajo para controlar cambios en los requisitos o picos de popularidad, con lo que se reduce la necesidad de prever el tráfico [43]. Además, proporciona una amplia selección de tipos de instancias optimizadas para adaptarse a diferentes casos de uso. Los tipos de instancia abarcan varias combinaciones de capacidad de CPU, memoria, almacenamiento y redes. Proporcionan flexibilidad para elegir la combinación de recursos adecuada para las aplicaciones. Cada tipo de instancia incluye uno o varios tamaños de instancia, lo que permite escalar los recursos según los requisitos de la carga de trabajo de destino [44].

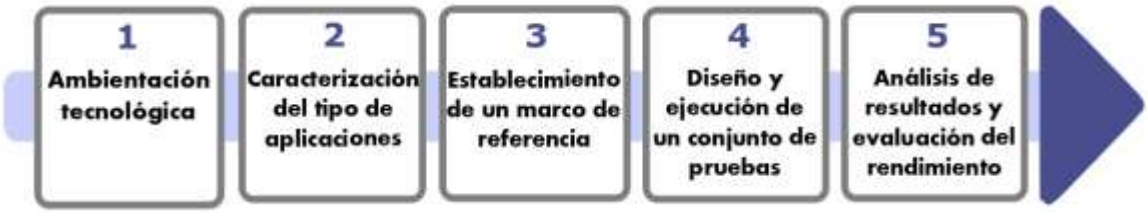
[43] AMAZON WEB SERVICES. ¿Qué es Amazon EC2? - Amazon Elastic Compute Cloud. [En línea]. En: Documentación de AWS. (Recuperado en 25 enero 2019). Disponible en https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html

[44] AMAZON WEB SERVICES. Tipos de instancias de Amazon EC2. [En línea]. En: Amazon Web Services. (Recuperado en 25 enero 2019). Disponible en https://aws.amazon.com/es/ec2/instance-types/

5. METODOLOGÍA

Se propone una metodología conformada por cinco etapas enfocadas a la planeación y desarrollo del proyecto, iniciando desde una ambientación tecnológica hasta el análisis de los resultados de las pruebas y la evaluación del rendimiento de cada modelo web. A continuación se puede observar el esquema del flujo de trabajo (ver Figura 11):

Figura 11. Metodología.



5.1 AMBIENTACIÓN TECNOLÓGICA

En esta etapa del proyecto se lleva a cabo la adquisición de los conocimientos necesarios para comprender y analizar a profundidad cada uno de los modelos web. Es de importancia conocer el estado del arte del desarrollo de aplicaciones web y profundizar en los conceptos relacionados con el diseño de pruebas de rendimiento.

Actividades

A1.1 Investigación de los fundamentos teóricos de los modelos web.

A1.2 Estudio de las posibles herramientas a utilizar.

A1.3 Estudio de los entornos de trabajo basados en los diferentes modelos web.

Productos

P1.1 Fundamentos teóricos y estado del arte.

5.2 CARACTERIZACIÓN DEL TIPO DE APLICACIONES

En esta etapa se define el conjunto de requerimientos que poseerán las aplicaciones implementadas en los diferentes modelos de desarrollo web. Con base en el funcionamiento de cada uno de estos modelos, se plantean los requerimientos funcionales y no funcionales de estas aplicaciones, para poder obtener información característica sobre el desempeño de cada modelo al realizar las pruebas de rendimiento.

Actividades

A2.1 Establecimiento de las características esenciales del funcionamiento de cada modelo web.

A2.2 Planteamiento de los requerimientos funcionales que en diferentes contextos, resaltan las características establecidas.

A2.3 Planteamiento de los requerimientos no funcionales que optimizan el desarrollo de las aplicaciones.

A2.4 Desarrollo de cada aplicación web en los diferentes modelos web.

Productos

P2.1 El código fuente y los binarios de las aplicaciones web implementadas.

P2.2 Conjunto de requerimientos de las aplicaciones web.

5.3 ESTABLECIMIENTO DE UN MARCO DE REFERENCIA

En esta etapa se busca establecer el conjunto de métricas que se utilizarán durante el diseño, ejecución y análisis de las pruebas de rendimiento. Cada una de estas métricas estará ligada a las características de las aplicaciones web

implementadas, y representarán las diferentes magnitudes del desempeño que provee la arquitectura de cada modelo web. Este conjunto de métricas definen el marco de referencia para poder comparar las propiedades de cada modelo web.

Actividades

A3.1 Agrupación de todas las posibles métricas que se pueden observar por medio de las diferentes herramientas de apoyo empleadas durante las pruebas.

A3.2 Investigación sobre cada una de las métricas.

A3.3 Selección del conjunto de métricas que mejor se relacionan con las propiedades de los modelos web a emplear.

Productos

P3.1 El conjunto de métricas utilizadas para comparar los modelos.

5.4 DISEÑO Y EJECUCIÓN DE UN CONJUNTO DE PRUEBAS

En esta etapa se diseñarán y ejecutarán un conjunto de pruebas en función de las aplicaciones web y del marco de referencia planteados en la segunda y tercera etapa. Cada prueba desarrollada tendrá como propósito medir, bajo un marco de referencia, el rendimiento al utilizar las aplicaciones en los distintos modelos web bajo un contexto definido.

Actividades

A4.1 Establecimiento del tipo de pruebas que se implementarán.

A4.2 Selección del conjunto de herramientas y tecnologías que se utilizarán en cada prueba.

A4.3 Desarrollo de cada prueba en función de las aplicaciones web implementadas.

A4.4 Ejecución del conjunto de pruebas un número determinado de veces.

Productos

P4.1 Protocolo de pruebas a ejecutar.

P4.2 Script de automatización de pruebas en la herramienta seleccionada.

P4.3 Conjunto de datos obtenidos luego de ejecutar las pruebas.

5.5 ANÁLISIS DE RESULTADOS Y EVALUACIÓN DEL RENDIMIENTO

En esta etapa se utilizarán los datos obtenidos al ejecutar las pruebas para realizar el debido análisis, recurriendo a un método que permita obtener conclusiones sobre el rendimiento, que cada modelo de desarrollo web presentó en los distintos contextos de prueba.

Actividades

A5.1 Selección del método a utilizar para analizar los datos.

A5.2 Clasificación de los datos obtenidos en las pruebas.

A5.3 Aplicación del método sobre cada conjunto de datos.

A5.4 Evaluación del rendimiento de cada modelo web según los resultados obtenidos al analizar los datos.

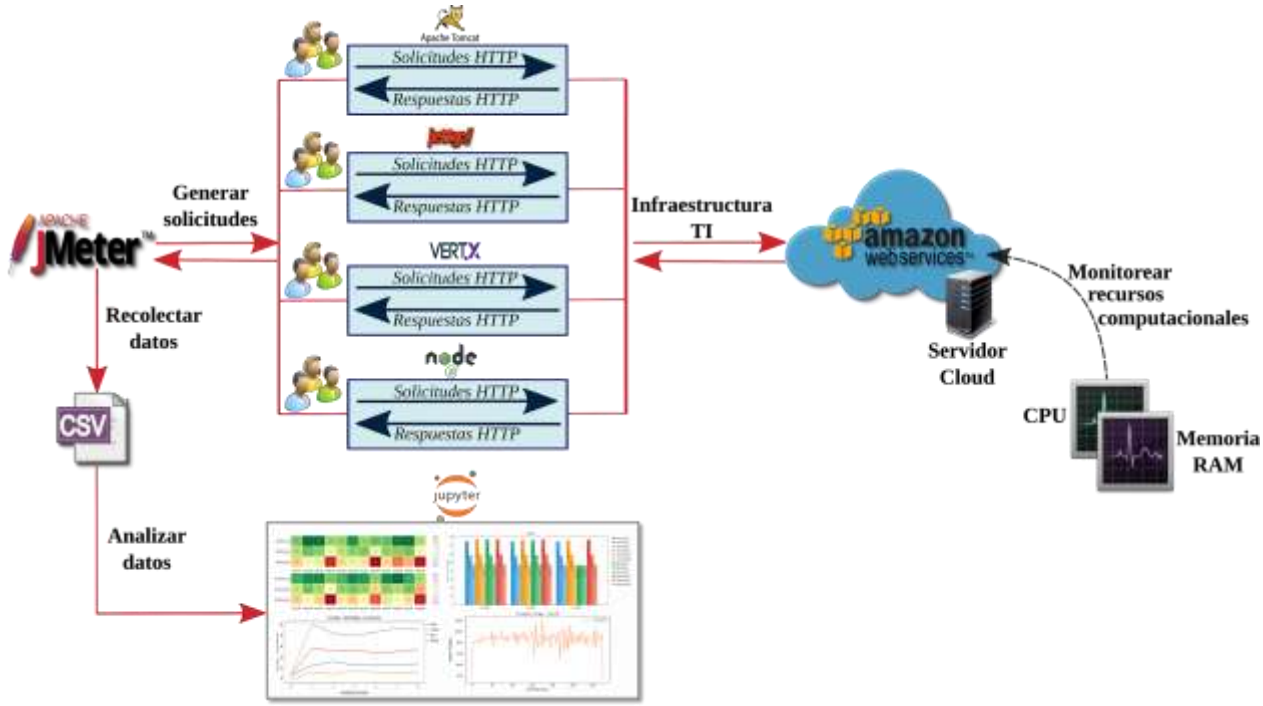
Productos

P5.1 Informe sobre el rendimiento de cada modelo web implementado.

6. DISEÑO DE LA EVALUACIÓN

En este capítulo se presenta el diseño de la evaluación que se va a realizar sobre los diferentes modelos de programación Web. Primero se deben caracterizar las aplicaciones tipo que servirán como base de la evaluación. Enseguida se ilustra cómo se desarrollaron las aplicaciones que se implementan en cada tecnología de la evaluación. Se presentan entonces el marco de referencia que se utilizará para evaluar el rendimiento de las diferentes aplicaciones implementadas en los modelos de programación evaluados. Finalmente, se presenta el diseño de las pruebas que van a utilizarse en el proceso de evaluación. A continuación se ilustra el diseño de la evaluación mostrando un panorama general de la implementación de cada una de las tecnologías y herramientas previamente seleccionadas para la ejecución de pruebas y análisis de resultados (ver Figura 12).

Figura 12. Enfoque del proyecto.

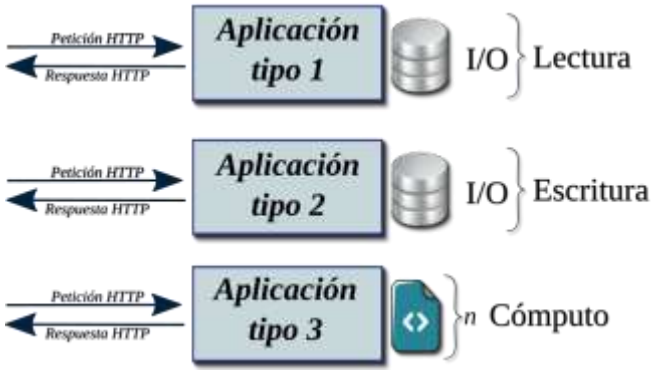


6.1 CARACTERIZACIÓN DEL TIPO DE APLICACIONES

Para definir el tipo de las aplicaciones se deben tener presente los objetivos planteados y lo que se desea evaluar de los modelos seleccionados. Se tiene que estas aplicaciones se desarrollarán basándose en aquellas operaciones pertenecientes al conjunto de operaciones bloqueantes que son comunes en las aplicaciones Web, dentro de estas se pueden encontrar operaciones de lectura y escritura en una base de datos, operaciones de cómputo, entre otras. Las operaciones bloqueantes se destacan porque se presentan normalmente en el modelo síncrono. Por otra parte, el modelo asíncrono posee una arquitectura que le permite enfocar el desarrollo del código con operaciones no bloqueantes.

Con base en las características de las tecnologías a implementar para los modelos de programación durante el proceso de evaluación del rendimiento, se definen tres aplicaciones tipo (ver Figura 13) con tres niveles de carga diferentes para cada una:

Figura 13. Aplicaciones tipo del proyecto.



- **Entrada/Salida – Lectura:** El proceso de lectura que busca ser representado por esta aplicación se desarrollará por medio de consultas a una base de datos. Las consultas se realizarán sobre tres cantidades diferentes de registros, las cuales han sido definidas en 1000, 5000 y 10000 registros.

- **Entrada/Salida – Escritura:** Se representará el proceso de escritura por medio de inserciones a una base de datos. Las inserciones definidas en el proceso de evaluación son más exigentes computacionalmente que las consultas, dado que la aplicación realizará tres cantidades diferentes de inserciones en cada una de las tres tablas definidas en la base de datos para esta aplicación. Las cantidades de inserciones definidas son 1, 3 y 6 registros.
- **Cómputo:** El proceso de cómputo que se define para esta aplicación es el de contar los números primos dentro de los siguientes límites: 1000, 2000 y 3000 números.

6.2 DISEÑO Y DESARROLLO DE LAS APLICACIONES

De acuerdo al enfoque planteado durante la caracterización, se procede al desarrollo de las aplicaciones usando las cuatro tecnologías seleccionadas para la implementación de los dos modelos de desarrollo web: Tomcat y Jetty en la programación síncrona, Vert.x y Node.js en la programación asíncrona (ver Tabla 3).

Tabla 3. Tecnologías correspondientes a cada modelo de programación.

Modelo de programación	Tecnología / Plataforma
Asíncrono	Vert.x (Multilenguaje sobre JVM)
	Node.js (Javascript)
Síncrono	Tomcat (Java)
	Jetty (Java)

6.2.1 Implementación de la base de datos. La base de datos de la aplicación será la misma para las diferentes tecnologías. Para su implementación, se ha

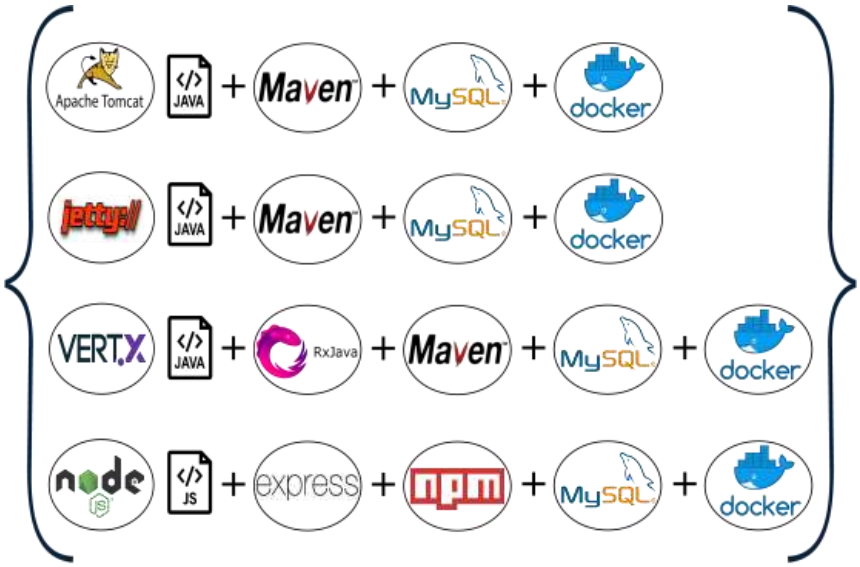
optado por trabajar con MySQL ya que es un gestor de base de datos relacionales común en el área académica y profesional. La base de datos MySQL que se emplea en las aplicaciones tipo 1 y 2 (ver Figura 13), simulará una institución académica, donde las entidades principales son los profesores, los estudiantes y las materias. Estas tablas se llenan con datos aleatorios haciendo uso de procedimientos almacenados que se encuentran integrados dentro las funcionalidades de MySQL. Los procedimientos almacenados contienen un conjunto de comandos SQL que son ejecutados de forma automatizada directamente en el motor de la base de datos MySQL, ofreciendo un rendimiento inmejorable ya que no es necesario transportar datos a ninguna parte, y permitiendo ejecutar operaciones complejas en pocos pasos.

6.2.2 Diseño de las aplicaciones. Para el diseño de las aplicaciones se deben tener en cuenta lo requerimientos planteados hasta el momento y la implementación de algunas tecnologías que faciliten el trabajo de desarrollo y mantenimiento. Se utiliza Maven para agilizar la gestión y el proceso de construcción de los proyectos en Java, en aplicaciones desarrolladas utilizando Jetty, Tomcat y Vert.x. En el caso de Node.js se utiliza el gestor de paquetes 'npm' que es especializado en los entornos de ejecución para Javascript y el framework Express.js. Por otro lado, se integra Vert.x con el framework RxJava permitiendo usar observables en cualquier lugar donde se puedan usar streams o resultados asíncronos para el JVM. Con respecto al despliegue de las aplicaciones, se utiliza Docker que facilita la creación, implementación y ejecución de las aplicaciones mediante el uso de contenedores. Esto asegura que las aplicaciones se ejecuten en cualquier otra máquina Linux, independientemente de las configuraciones personalizadas que la máquina pueda tener y que difieran de la máquina utilizada para escribir y probar el código.

Otro aspecto a tener en cuenta es la configuración de los threads, conexiones y tamaño del pool en las aplicaciones. Todas deben estar en las mismas

condiciones, es decir, llevarlas a su punto máximo para que no se vean limitadas por valores por defecto, arrojando los valores reales de rendimiento y consumo de recursos en la ejecución de pruebas. En la Figura 14 se ilustran cada una de las tecnologías y herramientas utilizadas para el diseño de cada aplicación web.

Figura 14. Herramientas y tecnologías implementadas en las aplicaciones del proyecto.

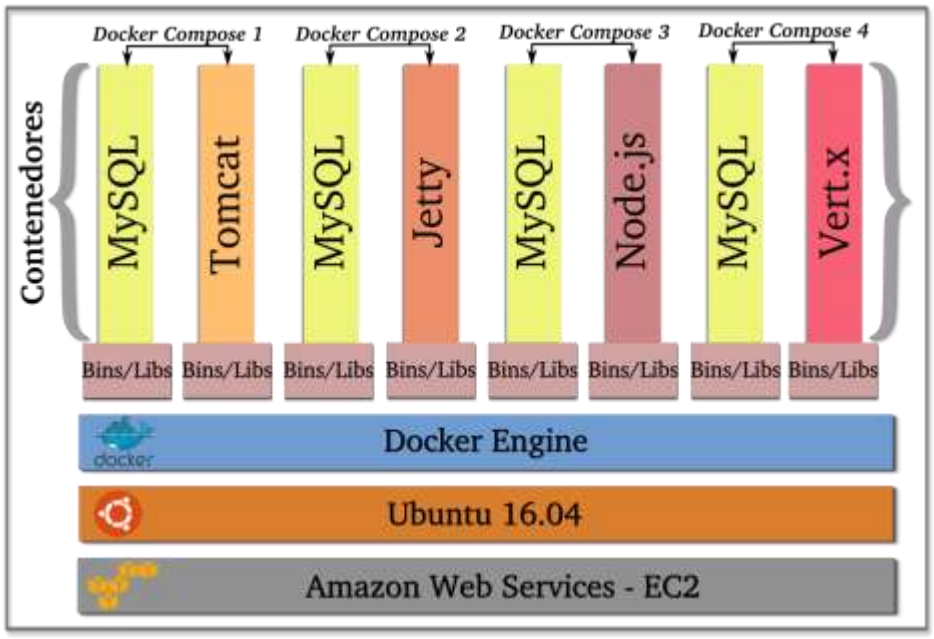


6.2.3 Despliegue de las aplicaciones. Con el fin de hacer más eficiente el despliegue de las aplicaciones, se optó por virtualizar por medio de Docker la base de datos y los servidores de las aplicaciones. Este proyecto consta de 5 imágenes Docker (ver Anexo A) configuradas de acuerdo a los requerimientos de las aplicaciones. Por otro lado, es necesario conectar los servicios, esto se logra a través de Docker Compose que es una herramienta para definir y ejecutar aplicaciones Docker de múltiples contenedores. A través de archivos YAML [45] se configuran los servicios de las aplicaciones y luego, con un solo comando, se crean e inician todos los servicios desde la configuración. Este proyecto consta de 4 archivos YAML y en cada uno se establecen dos servicios/contenedores para

[45] EVANS, Clark; BEN-KIKI, Oren y DÖT NET, Ingy. YAML 1.2. [En línea]. En: The Official YAML Web Site, 2009. (Recuperado en 23 noviembre 2018). Disponible en https://yaml.org/

que puedan ejecutarse juntos en un entorno aislado. Uno de los servicios es la base de datos MySQL y el otro es una de las 4 tecnologías (Node.js, Vert.x, Tomcat o Jetty). Se define también la ruta para construir la imagen del fichero Dockerfile, la exposición de puertos, las variables de entorno, comandos adicionales y las dependencias de un servicio con respecto a otro. En la Figura 15 se ilustra la estructura final de los contenedores para todas las aplicaciones usando como base el sistema operativo Ubuntu 16.04 y Amazon EC2 para la infraestructura.

Figura 15. Estructura de los contenedores Docker.



Es importante tener en cuenta que los contenedores de Docker deben ser configurados para poder acceder y utilizar de los servidores una buena cantidad de memoria RAM, que por defecto es pequeña, evitando así en la ejecución de las pruebas enfrentar limitaciones y datos erróneos sobre el rendimiento de las tecnologías a evaluar. Para evitar esto, se establece el “Heap Size” en 10 GB para todas las aplicaciones, por lo tanto, sólo podrán ser desplegadas en un servidor con más de 10 GB de memoria RAM.

6.3 MARCO DE REFERENCIA DE LA EVALUACIÓN

El marco de referencia para las pruebas permite tomar decisiones respecto a cómo se evaluará cada modelo de desarrollo web. Para esto se deben tener en cuenta los factores que normalmente definen el rendimiento de una aplicación Web. En la Tabla 4 se pueden observar el conjunto de métricas que normalmente se manejan para las peticiones HTTP y su descripción.

Tabla 4. Descripción de las métricas.

Métrica	Descripción
Tiempo de Respuesta	Denota el tiempo transcurrido en milisegundos desde justo antes de enviar la solicitud hasta después de recibir la última respuesta.
Media	Es el valor promedio del tiempo transcurrido en milisegundos de las muestras.
Mediana	Es un número que mide las muestras en dos mitades iguales. Es otro nombre para el percentil 50 o la línea del 50 por ciento.
Desviación estándar	Representa la dispersión de los datos alrededor del valor promedio del tiempo transcurrido.
% de error	Porcentaje de solicitudes fallidas por etiqueta.
Mínimo	Es el valor mínimo del tiempo transcurrido en milisegundos dentro del conjunto de muestras.
Máximo	Es el valor máximo del tiempo transcurrido en milisegundos dentro del conjunto de muestras.
Rendimiento (Throughput)	Representa la cantidad de solicitudes por unidad de tiempo que fluyen a través del sistema. Se calcula dividiendo el número de solicitudes por el tiempo total transcurrido desde que se envió la primera solicitud hasta que se recibió la respuesta de la última solicitud.
Número de Muestras Iniciadas por Segundo	Se calcula contando el número de solicitudes por segundo que llegan al servidor.

6.4 DISEÑO Y EJECUCIÓN DE LAS PRUEBAS DE EVALUACIÓN

Dos aspectos de calidad importantes para reducir riesgos en la producción de una aplicación son la funcionalidad y el rendimiento. El rendimiento de una aplicación se puede medir a través del uso de herramientas para simular cargas que generan múltiples usuarios conectados concurrentemente. Una de las generadoras de carga open source más populares es JMeter. Mientras se ejecuta la carga se analiza el desempeño en la búsqueda de cuellos de botella y oportunidades de mejora. De esta manera, se puede saber si una aplicación responde adecuadamente a una cantidad de carga específica, además de saber la cantidad de usuarios que puede soportar antes de dejar de responder o la manera en que se repone luego de un pico de carga.

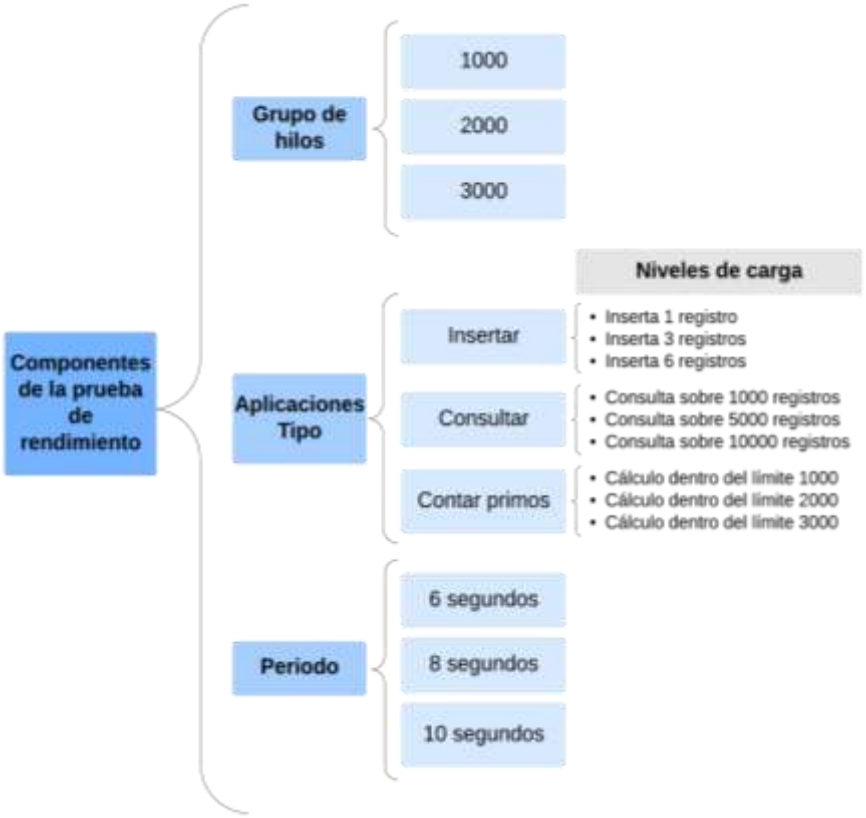
El proceso de diseñar y ejecutar las pruebas para la evaluación del rendimiento de los modelos de programación en las diversas tecnologías seleccionadas, se lleva a cabo después de hacer un testing exploratorio en diferentes escenarios. A partir del comportamiento de las aplicaciones en dichos escenarios se toman decisiones respecto a la complejidad de las funcionalidades a evaluar, la cantidad de conexiones concurrentes en un periodo de tiempo que se van a simular, el tipo de pruebas que se van a realizar y el entorno óptimo para ejecutar el conjunto de pruebas.

6.4.1 Pruebas de rendimiento. Una prueba de rendimiento determina o valida la velocidad, escalabilidad y/o características de estabilidad de un sistema bajo prueba. A partir de las pruebas exploratorias, se determina evaluar el rendimiento en las tres aplicaciones tipo (ver Figura 13) en todos los niveles de carga para las cuatro tecnologías (Vert.x, Node.js, Tomcat y Jetty). Se seleccionan 3 escenarios de carga (grupo de hilos), el primero con 1000 conexiones, el segundo con 2000 y el último con 3000 conexiones concurrentes. La conexiones se simulan dentro de un periodo de tiempo determinado, es decir, las n conexiones (1000, 2000 o 3000)

se pueden distribuir en 6 segundos, 8 segundos o 10 segundos. Por ejemplo, si se indica un grupo de 1000 hilos con un periodo de 6 segundos, la herramienta de distribución de carga JMeter, crea automáticamente cada 0.006 segundos un hilo nuevo. Cada prueba es ejecutada 5 veces.

Los componentes que forman parte de la prueba de rendimiento son: grupo de hilos, aplicaciones tipo y periodo (ver Figura 16). El grupo de hilos representa la cantidad de usuarios/solicitudes que se lanzan en un determinado periodo de tiempo, y el periodo de tiempo representa el número de segundos que transcurrirá desde el inicio de la prueba hasta que se alcance el número máximo de hilos seleccionado.

Figura 16. Componentes de la prueba de rendimiento.



6.4.2 Prueba de carga. Las pruebas de carga simulan la realidad a la cual estará sometido un sistema para analizar su desempeño ante una situación. El objetivo de esta prueba es determinar cuántos usuarios pueden manejar las aplicaciones, hallando el verdadero throughput de cada aplicación y analizando el consumo de recursos sobre el límite de concurrencia soportado.

Se selecciona la aplicación tipo *"Entrada/Salida - Lectura"* y el nivel de carga "*Consulta sobre 10000 registros*". A través de pruebas exploratorias en forma de búsqueda binaria, se encuentra el límite de concurrencia que permite cada tecnología (Vert.x, Node.js, Tomcat y Jetty) en los escenarios a evaluar. La prueba consiste en la simulación de un número de conexiones concurrentes que va aumentando cada 10 segundos hasta llegar al límite permitido de conexiones concurrentes. Todo el proceso de paso lleva 100 segundos. En cuanto alcanza el límite de subprocesos, todos ellos continúan ejecutándose manteniendo la concurrencia y conectando el servidor durante 30 segundos.

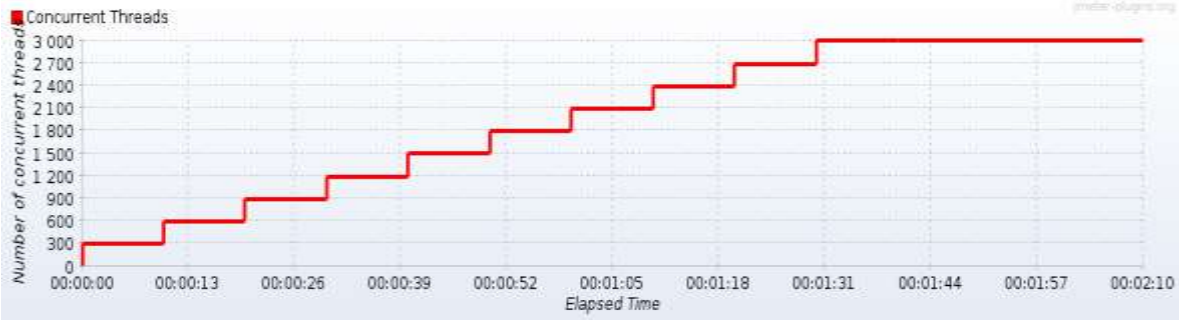
En la Tabla 5 se pueden ver las tecnologías y su respectivo límite de concurrencia, los escenarios a evaluar y el número de conexiones concurrentes que irán aumentando cada 10 segundos hasta llegar al límite.

Tabla 5. Diseño de la prueba de carga de la evaluación.

Tecnología	Aplicación Tipo	Nivel de carga	Incremento de concurrencia cada 10 segundos	Límite de concurrencia
Vert.x	Entrada/Salida - Lectura	Consulta sobre 10000 registros	140 conexiones	1400 conexiones
Node.js			129 conexiones	1290 conexiones
Tomcat			300 conexiones	3000 conexiones
Jetty			124 conexiones	1240 conexiones

La configuración de la prueba para Tomcat se ilustra en la Figura 17. Esta representación de la prueba de carga es similar para las tecnologías restantes.

Figura 17. Representación de la prueba de carga de Tomcat en el límite.



6.4.3 Prueba de estrés (Spike). Las pruebas de estrés someten el sistema a una carga por encima de los límites requeridos con el objetivo de encontrar el punto de ruptura. Si la prueba de estrés incluye un aumento repentino en el número de usuarios virtuales, se denomina prueba de pico (spike testing). Este es el tipo de prueba elegido para evaluar el comportamiento de las cuatro aplicaciones bajo cargas intensas y el modo en que estas se recuperan después de picos de carga, midiendo el consumo de los recursos. Se establece para la prueba de estrés (spike) la aplicación tipo *"Entrada/Salida - Lectura"* y el nivel de carga "*Consulta sobre 10000 registros*". Se escoge un pico de carga diferente para cada aplicación, en el que al realizar la prueba arroje un error entre el 1% y 3%. Este número de conexiones (pico de carga) se encuentra a través de pruebas exploratorias en forma de búsqueda binaria.

Se comienza la prueba sin agregar conexiones. Un minuto después del inicio de la prueba, se empiezan a agregar conexiones que van aumentando linealmente durante 30 segundos, hasta llegar al máximo de conexiones concurrentes (pico de carga) de la aplicación. Se mantiene en esa concurrencia durante 60 segundos para luego empezar a disminuir las conexiones linealmente durante 10 segundos, hasta que no se lance ninguna conexión. Se mantiene en ese estado durante 3

minutos y 10 segundos, y como en el primer pico de carga, se empiezan a agregar linealmente conexiones durante 30 segundos hasta llegar al número de conexiones del pico de carga, manteniéndose así durante 60 segundos y disminuyendo linealmente durante 10 segundos hasta que no se genere ninguna conexión. En la Figura 18 se ilustra una simulación de la prueba de estrés para Vert.x. La representación gráfica para Node.js, Tomcat y Jetty es similar a la de Vert.x.

Figura 18. Representación de la prueba de estrés (spike) de Vert.x.

En la Tabla 6 se muestra el diseño de la prueba de estrés para cada tecnología, los escenarios a evaluar y el número de conexiones concurrentes en el pico de carga.

Tabla 6. Diseño de la prueba de estrés de la evaluación.

Tecnología	Aplicación Tipo	Nivel de carga	Número de conexiones concurrentes en el pico de carga
Vert.x	Entrada/Salida - Lectura	Consulta sobre 10000 registros	1324
Node.js			1250
Tomcat			1500
Jetty			1210

6.4.4 Automatización de las pruebas. La automatización de pruebas permite controlar la ejecución de pruebas y la comparación entre los resultados obtenidos y los resultados esperados; además de ser necesaria cuando las pruebas se deben repetir muchas veces. Otro factor que indica la conveniencia de automatizar es si la prueba es muy larga, tediosa o compleja, lo que permite asegurar que la prueba automatizada se ejecuta consistentemente todas las veces y así evitar errores. Con el fin de disminuir la cantidad de tiempo que se emplea durante la preparación y ejecución de las pruebas, se ha optado por programar un script que utilice el shell de Linux para mejorar la ejecución de los comandos por consola.

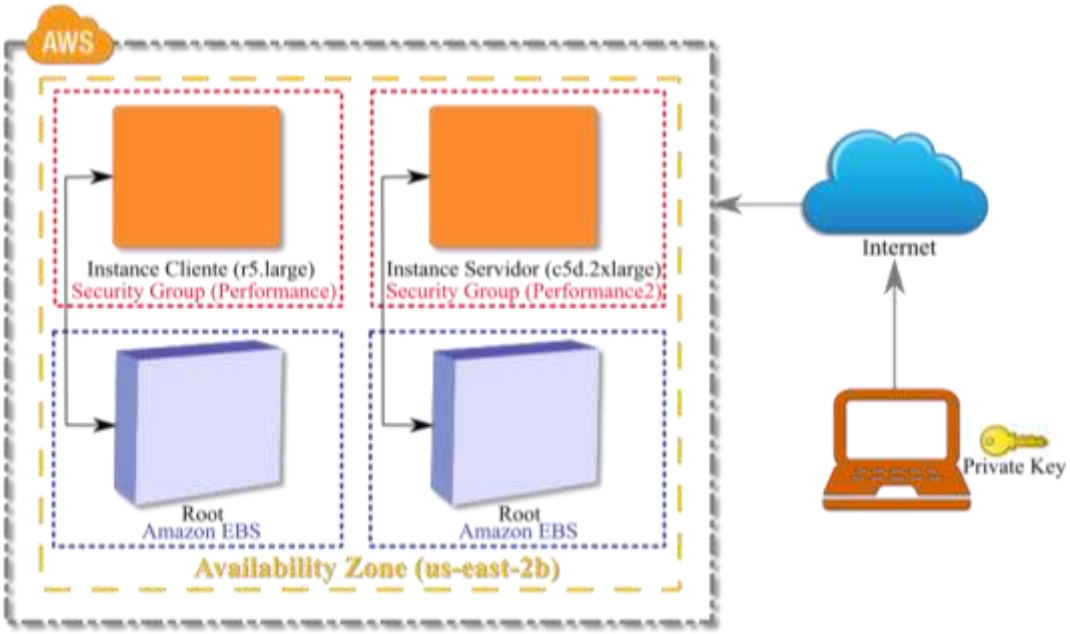
En general, se busca agilizar procesos tales como el despliegue de las aplicaciones, la ejecución de un conjunto de planes de pruebas, eligiendo el tipo de escenario de prueba y el tipo de tecnología, y también se busca agilizar el proceso de detener, limpiar y eliminar las imágenes y contenedores de Docker. Para poder ejercer el monitoreo del consumo de la CPU y de memoria RAM de cada una de las tecnologías, es necesario ejecutar la opción “Activar Plugins JMeter” en el cliente y la opción *“Activar PerfMon”* en el servidor, antes de realizar el despliegue de las aplicaciones. Una vez terminadas las pruebas, se ejecuta la opción *“Detener PerfMon”* en el cliente, ingresando la dirección IP pública del servidor, para detener el monitoreo del consumo de recursos de las aplicaciones.

Para realizar la ejecución de pruebas de rendimiento es necesario indicar la cantidad de pruebas que se van a realizar, el periodo de tiempo en que se lanzará el grupo de hilos y la dirección IP pública del servidor. Por otro lado, para las pruebas de carga y de estrés, solo es necesario ingresar el valor límite de concurrencia y la dirección IP pública del servidor. En el Anexo B se puede observar el diagrama de flujo del Script de automatización diseñado.

6.4.5 Preparación de la infraestructura. La infraestructura consiste básicamente de los dispositivos que cumplen el rol de cliente, el rol de servidor, y la red por la

cual se envían los datos. De estos tres elementos que definen el entorno, la red es la más susceptible a factores externos, por ello se ha optado por usar una red privada de AWS, con el fin de mitigar el ruido que generan estos factores en los resultados. A través de Amazon Elastic Compute Cloud (Amazon EC2) se realiza el lanzamiento, la conexión y el uso de las instancias de Linux. Una instancia es un servidor virtual en la nube de AWS. Para el desarrollo de las pruebas, son necesarias dos instancias EC2 que cumplan la función de Cliente y Servidor conectadas a la misma red privada. La instancias EC2 seleccionadas son instancias con respaldo Amazon EBS (lo que significa que el volumen raíz es un volumen de EBS). Se especifica la zona de disponibilidad '*US East (Ohio)*' para ejecutar cada instancia. Para lanzar las instancias, estas deben ser protegidas especificando un par de claves y grupos de seguridad. Para este caso, se utilizan dos grupos de seguridad llamados '*Performance*' para el cliente y '*Performance2*' para el servidor. En estos grupos de seguridad se especifican los puertos que escuchan las aplicaciones. En la Figura 19 se puede ver la estructura básica de Amazon EC2 empleada en el proyecto.

Figura 19. Estructura básica de Amazon EC2 implementada.



Por otra parte, para hacer la conexión a las instancias hay que estar ubicado en la carpeta donde se encuentra la clave privada del par de claves que se especificaron cuando se lanzaron las instancias, y una vez hecha la conexión se instalan las herramientas necesarias para el despliegue de las aplicaciones con Docker. Se deben instalar en cada instancia: Oracle Java 8, Apache JMeter 4.0, Unzip, Docker, Docker Compose y descargar el *repositorio en GitHub del proyecto* [46] para poder realizar el despliegue y la ejecución de las pruebas. En la Tabla 7 se pueden ver el tipo de instancias EC2 de AWS seleccionadas para el proyecto y sus especificaciones.

Tabla 7. Especificaciones de los dispositivos empleados para la ejecución de pruebas.

Rol	***Tipo de instancia EC2***	***Descripción***
Cliente	*r5.large*	● Intel(R) Xeon(R) Platinum 8175M CPU @ 2.50GHz ● 2 CPUs ● 16 GB de memoria RAM ● Sistema operativo Ubuntu 16.04
Servidor	*c5d.2xlarge*	● Intel(R) Xeon(R) Platinum 8124M CPU @ 3.00GHz ● 8 CPUs ● 16 GB de memoria RAM ● Sistema operativo Ubuntu 16.04

[46] MALDONADO, Viviana Andrea. Performance-MW. [En línea]. En: GitHub. (Recuperado en 29 enero 2018). Disponible en https://github.com/vivianamaldonado/Performance-MW

7. ANÁLISIS DE RESULTADOS Y EVALUACIÓN DEL RENDIMIENTO

La evaluación del rendimiento de las aplicaciones tipo se realizará teniendo en cuenta las métricas planteadas en la Sección 6.3 (tiempo medio de respuesta, rendimiento, consumo de CPU y memoria RAM, número de usuarios activos, entre otros) para cada una de las pruebas definidas en la Sección 6.4 (pruebas de rendimiento, pruebas de carga y pruebas estrés).

7.1 EVALUACIÓN A TRAVÉS DE LAS PRUEBAS DE RENDIMIENTO

Los indicadores clave del rendimiento seleccionados para evaluar este tipo de prueba son: el tiempo medio de respuesta, el rendimiento (throughput) y el consumo de recursos. Dentro del análisis de las pruebas de rendimiento solo se toman cuatro de las cinco pruebas realizadas para cada aplicación tipo en los distintos niveles de carga definidos, debido a que la primera prueba se realiza para preparar la base de datos dado que esta se optimiza con cada consulta e inserción realizada.

7.1.1 Tiempo medio de respuesta. El tiempo medio de respuesta es el tiempo medio empleado por cada una de las aplicaciones tipo para atender cada solicitud. En este análisis de resultados se desea observar el comportamiento de cada una de las aplicaciones tipo implementadas en cada modelo de programación en los escenarios de carga planteados, donde varían la cantidad de usuarios concurrentes y los periodos de tiempo. A continuación se mostrarán por medio de mapas de calor los tiempos medios de respuesta de todas las pruebas de rendimiento para cada una de las aplicaciones tipo definidas (Insertar, Consultar, Contar Primos). Los valores que se muestran en estos mapas de calor se obtuvieron por medio del siguiente procedimiento: primero se descartaron las pruebas que mostraron mayor desviación respecto a las demás dejando solo tres

pruebas y por último se promediaron los tiempos de respuesta de estas pruebas resultantes. Cada fila del mapa de calor representa un nivel de carga de la aplicación tipo y cada columna representa la concurrencia de usuarios para cada tecnología. Los mapas de calor de las pruebas de rendimiento referentes a la aplicación tipo “Insertar” se ilustran en las Figuras 20 - 22.

Figura 20. Tiempo medio de respuesta para “Insertar” con Periodo 6.

Figura 21. Tiempo medio de respuesta para “Insertar” con Periodo 8.

Figura 22. Tiempo medio de respuesta para “Insertar” con Periodo 10.

Se observa que en el modelo asíncrono (Vert.x, Node.js), el tiempo medio de respuesta tiende a ser menor que en el modelo síncrono (Tomcat, Jetty) para las operaciones de escritura en una base de datos, siendo Node.js el que se destaca por poseer los tiempos medios de respuesta más bajos y Jetty el que posee los tiempos medios de respuesta más altos de las cuatro tecnologías (ver Figuras 20 - 22). Los mapas de calor de las pruebas de rendimiento referentes a la aplicación tipo “Consultar” están representados en las Figuras 23 - 25.

Figura 23. Tiempo medio de respuesta para “Consultar” con Periodo 6.

Figura 24. Tiempo medio de respuesta “Consultar” con Periodo 8.

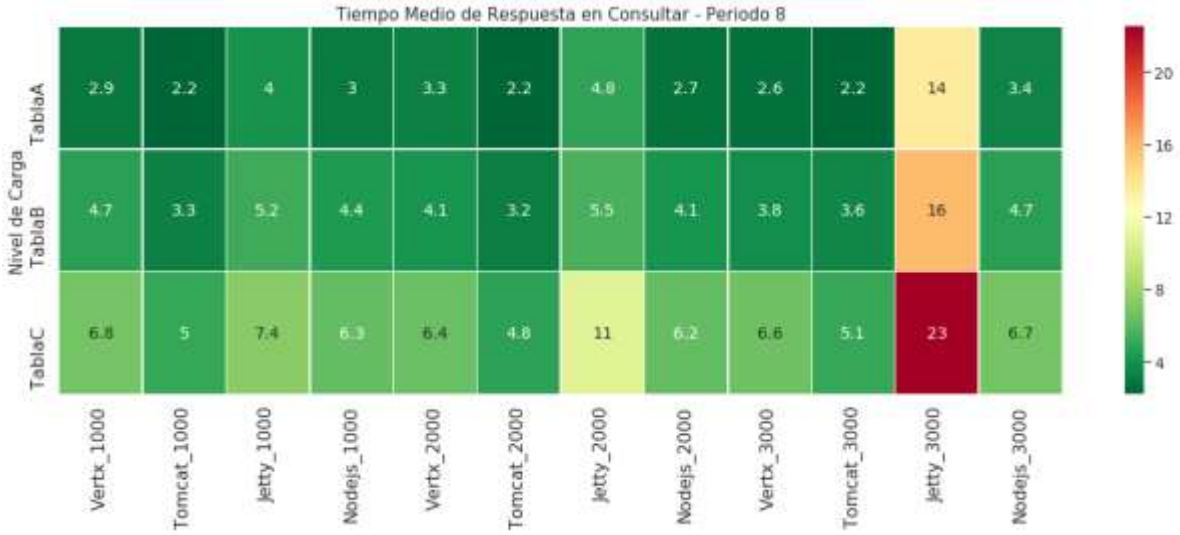
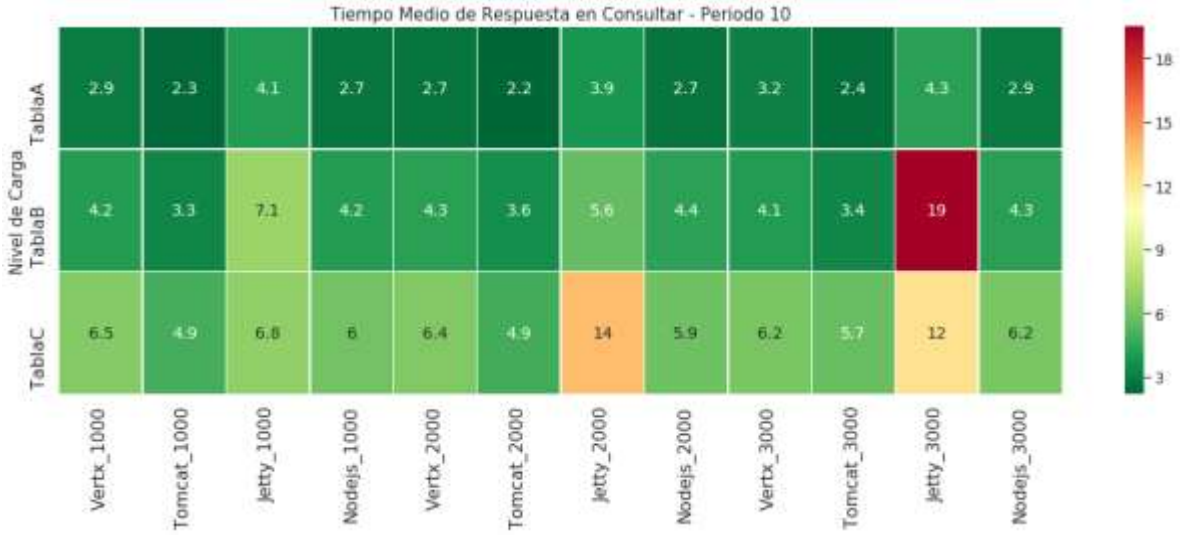


Figura 25. Tiempo medio de respuesta para “Consultar” con Periodo 10.



Para las operaciones de lectura en una base de datos como lo es realizar consultas en diferentes tablas (ver Figuras 23 - 25), se observa que Tomcat es el que en general obtuvo los menores tiempos medios de respuesta y Jetty el que obtuvo los mayores tiempos medios de respuesta.

Los mapas de calor de las pruebas de rendimiento referentes a la aplicación tipo “Contar Primos” se ilustran las Figuras 26 - 28.

Figura 26. Tiempo medio de respuesta para “Contar Primos” con Periodo 6.

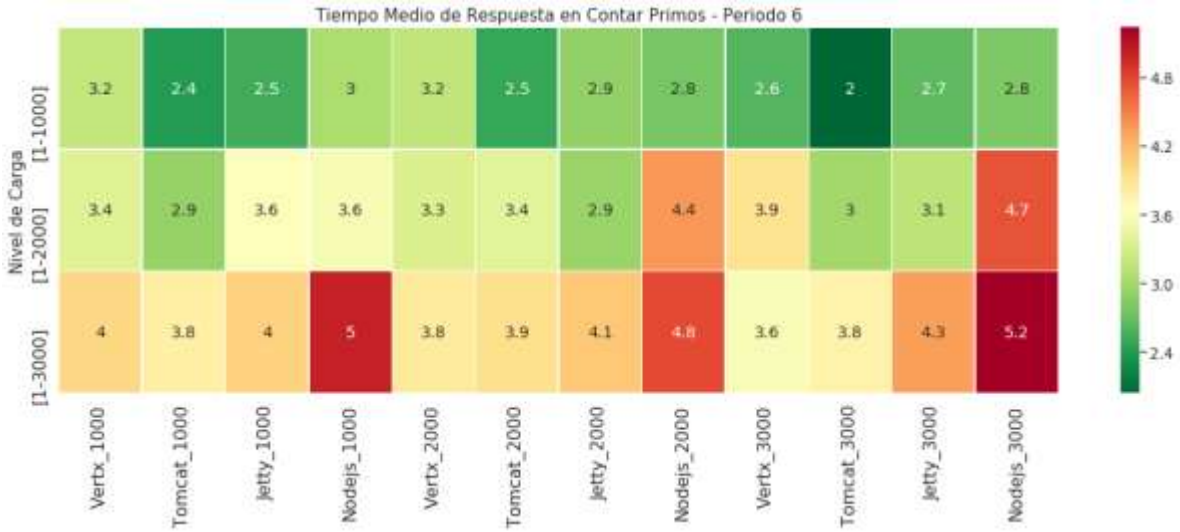


Figura 27. Tiempo medio de respuesta para “Contar Primos” con Periodo 8.

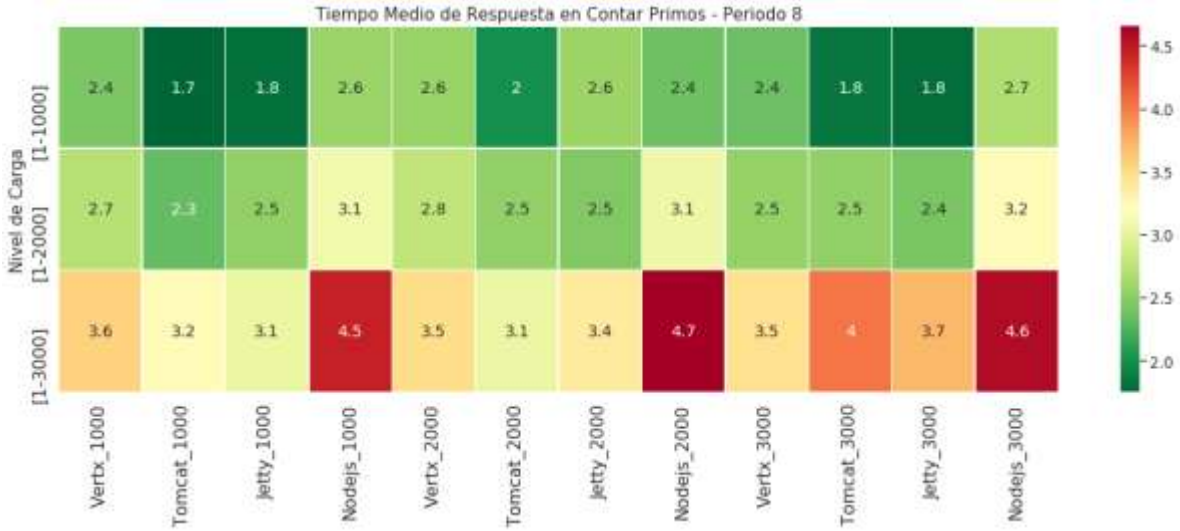
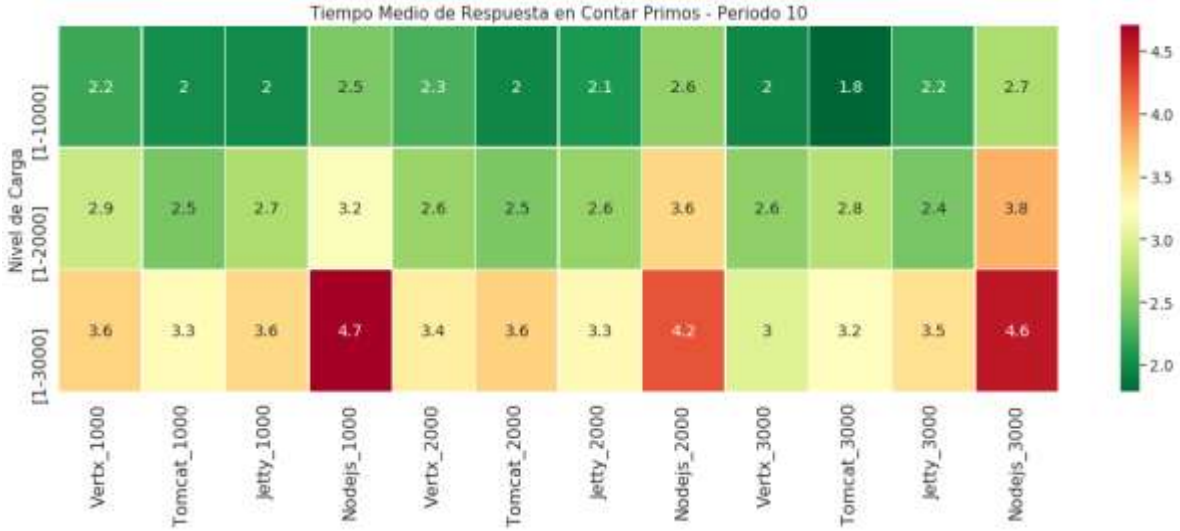


Figura 28. Tiempo medio de respuesta para “Contar Primos” con Periodo 10.



Los tiempos medios de respuesta obtenidos por la aplicación tipo “Contar Primos” son cercanos entre las cuatros tecnologías, siendo Node.js quien normalmente posee los valores más altos. De estos resultados (ver Figuras 26 - 28) no es posible definir claramente alguna característica destacable sobre cada modelo de programación implementado.

7.1.2 Rendimiento. El throughput o rendimiento indica la cantidad de transacciones por segundo que pueden manejar las aplicaciones, es decir, la cantidad de transacciones producidas a lo largo del tiempo durante una prueba. Este indicador junto con el tiempo medio de respuesta permite medir la capacidad de respuesta de las aplicaciones tipo implementadas en cada escenario de carga en los distintos modelos de programación. El throughput medido para las pruebas de rendimiento referentes a la aplicación tipo “Insertar” (ver Figuras 29 - 31), muestra la cantidad de solicitudes por segundo que se atienden al realizar operaciones de escritura en una base de datos.

Figura 29. Rendimiento de “Insertar” con 1000 Solicitudes.

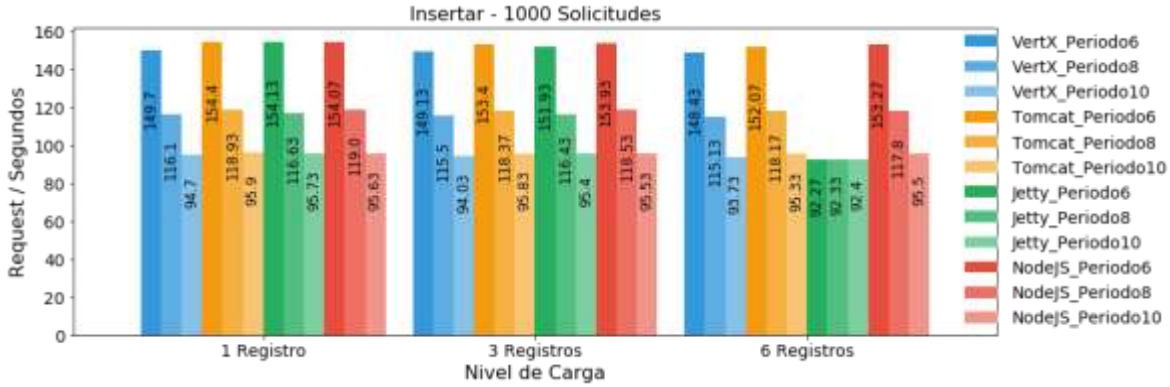


Figura 30. Rendimiento de “Insertar” con 2000 Solicitudes.

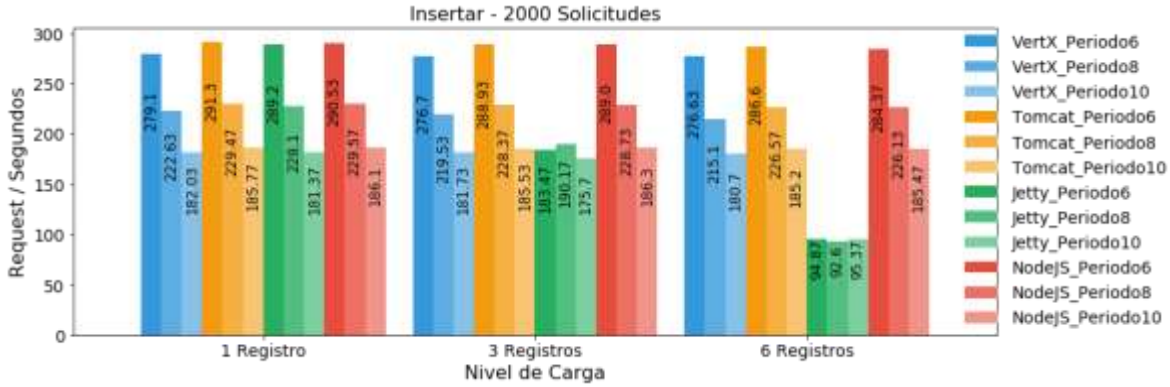
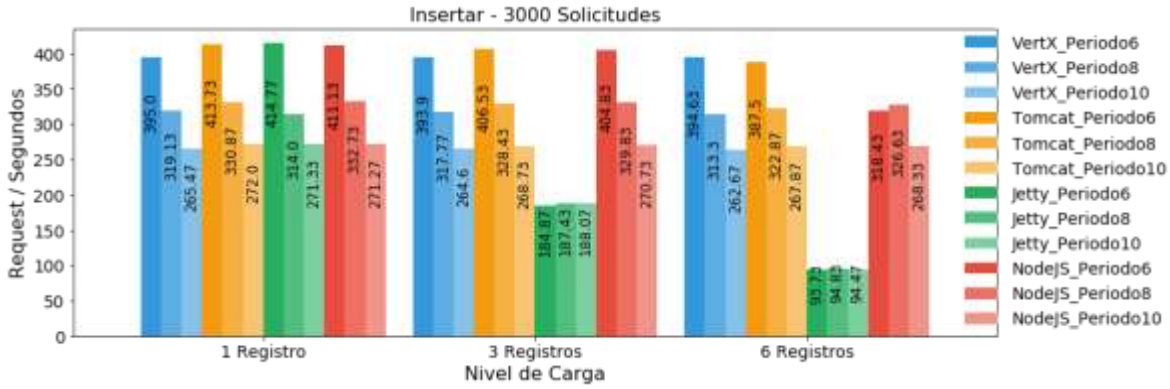


Figura 31. Rendimiento de “Insertar” con 3000 Solicitudes.



Observando el rendimiento obtenido se puede decir que Jetty se degrada al aumentar el nivel de carga y la concurrencia de usuarios. Node.js también

presenta una disminución en el rendimiento cuando inserta 6 registros con 3000 usuarios concurrentes. Las otras tecnologías (Tomcat y Vert.x) no muestran variación significativa al aumentar la concurrencia o el nivel de carga.

Un aspecto importante a mencionar es que el modelo síncrono posee tiempos medios de respuesta relativamente altos comparados con las tecnologías del modelo asíncrono para la aplicación tipo “Insertar”, sin embargo, al comparar el rendimiento obtenido por ambos modelos para esta aplicación tipo no se observan diferencias significativas. Este comportamiento puede ser causado por diversas razones, una de ellas es la forma en la que cada modelo procesa las solicitudes recibidas, que para el caso del modelo síncrono son procesadas en hilos independientes cada una, y a pesar de que el tiempo de procesamiento de cada solicitud es mayor a los obtenidos en el modelo asíncrono, el tiempo empleado para atender todas las solicitudes es relativamente igual dado el comportamiento concurrente de los hilos procesados.

Otro factor que pudo influir es la dispersión de los datos, sin embargo, al observar la desviación media del tiempo de respuesta (ver Anexo C), no se observan diferencias significativas con las desviaciones mostradas en ambos modelos, salvo para Jetty que mostró en la mayor parte de las pruebas valores de tiempos medios de respuesta relativamente altos.

El throughput medido para las pruebas de rendimiento referentes a la aplicación tipo “Consultar” (ver Figuras 32 - 34), muestra la cantidad de solicitudes por segundo que se atienden al realizar operaciones de lectura en una base de datos.

Figura 32. Rendimiento de “Consultar” con 1000 Solicitudes.

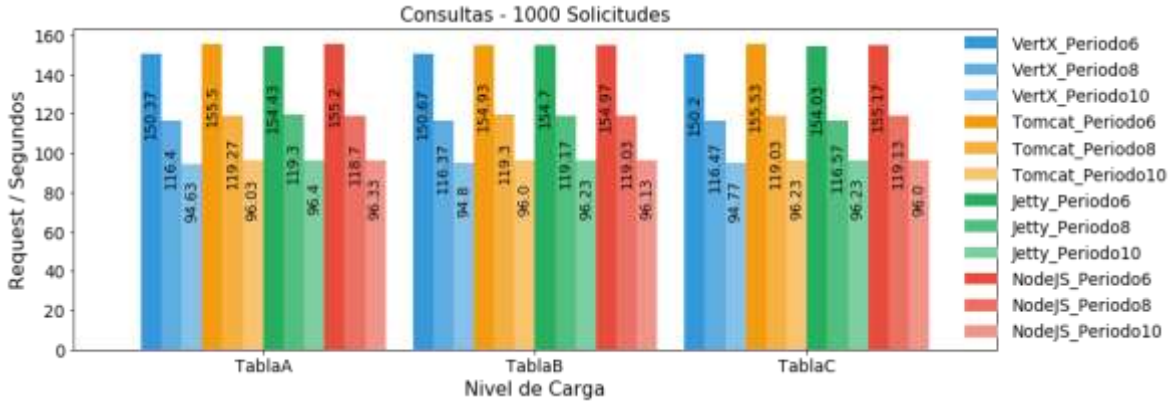


Figura 33. Rendimiento de “Consultar” con 2000 Solicitudes.

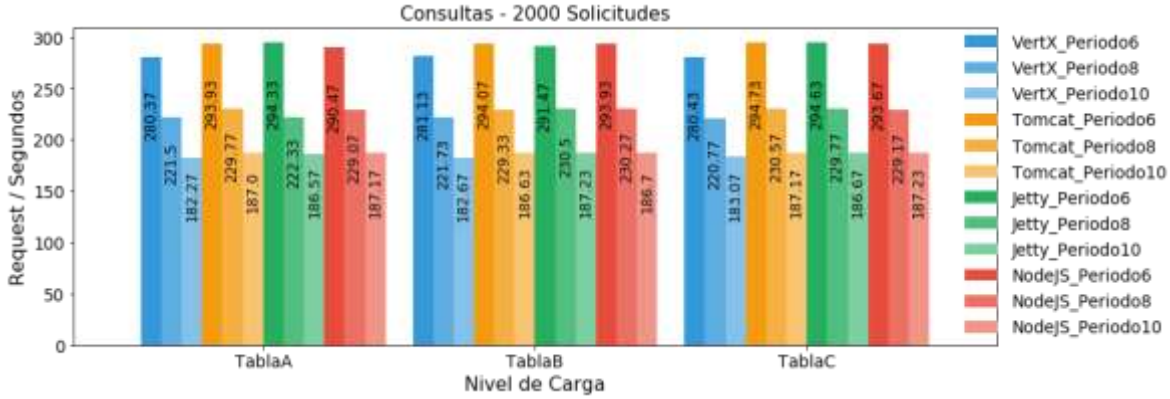
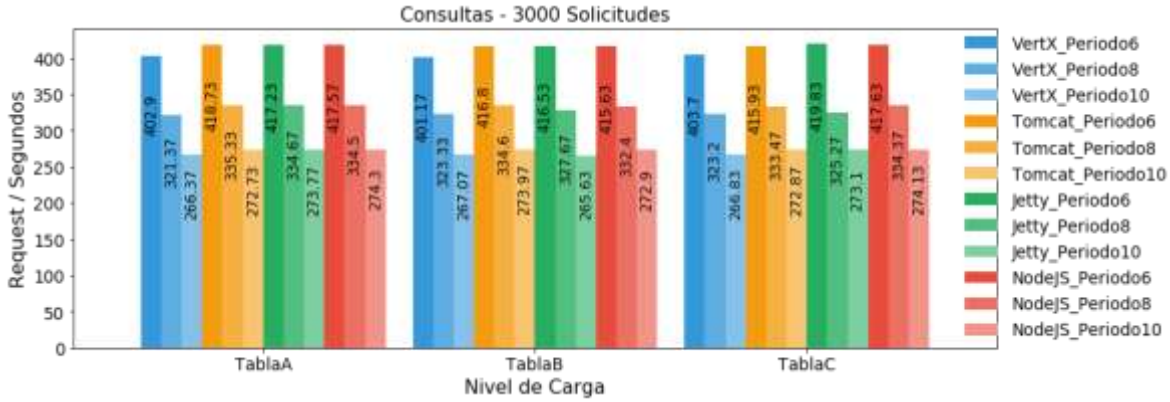


Figura 34. Rendimiento de “Consultar” con 3000 Solicitudes.



El rendimiento que se observa para la aplicación tipo “Consultar” es similar en todas las tecnologías para todos los casos de concurrencia de usuarios; otro aspecto a destacar es que no se observan cambios al aumentar el nivel de carga. El throughput medido para las pruebas de rendimiento referentes a la aplicación tipo “Contar Primos” (ver Figuras 35 - 37), muestra la cantidad de solicitudes por segundo que se atienden al realizar operaciones de cómputo.

Figura 35. Rendimiento de “Contar Primos” con 1000 Solicitudes.

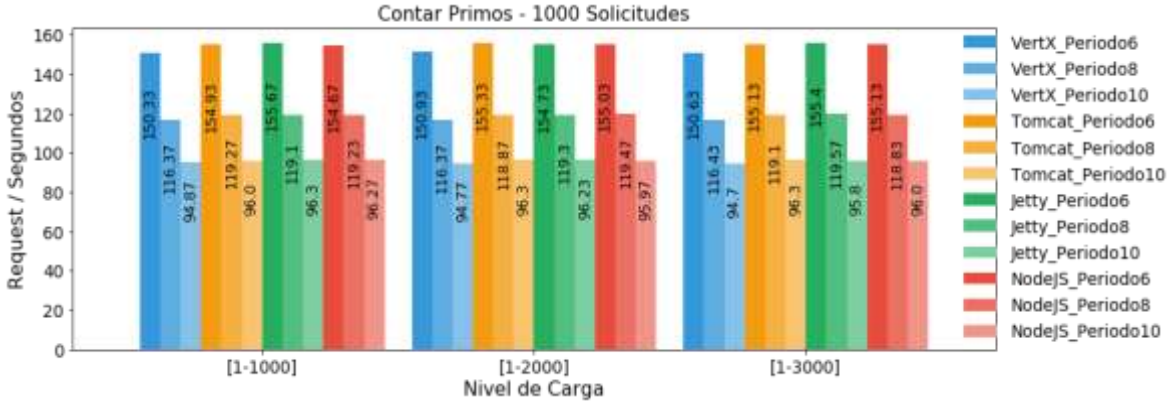


Figura 36. Rendimiento de “Contar Primos” con 2000 Solicitudes.

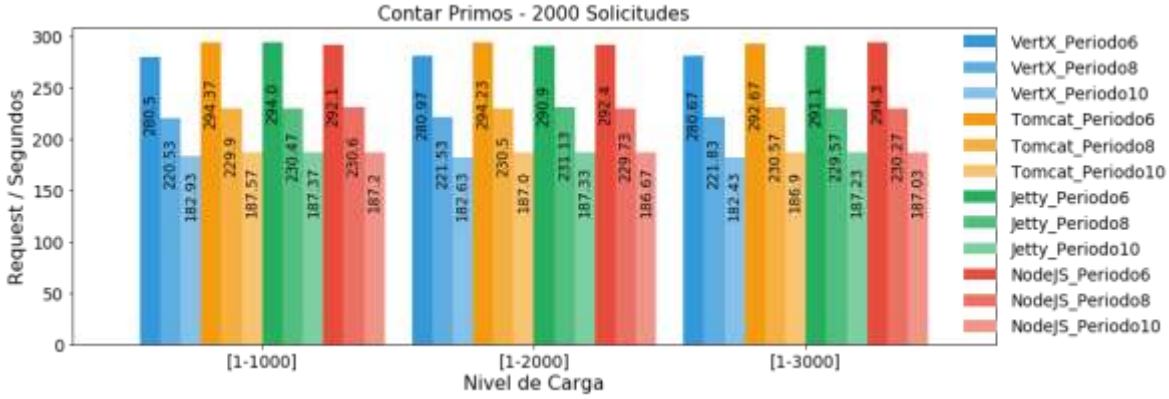
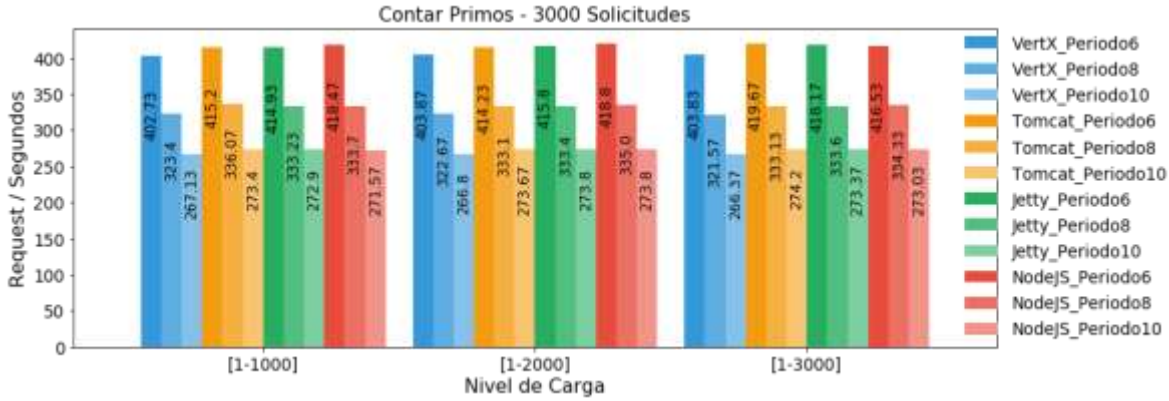


Figura 37. Rendimiento de “Contar Primos” con 3000 Solicitudes.



El rendimiento observado en la aplicación tipo “Contar Primos” muestra los mismos patrones descritos en la aplicación tipo “Consultar”. Si se compara el rendimiento con el tiempo de respuesta, lo ideal sería que las tecnologías que tienen los menores tiempos de respuesta, sean las que mayor rendimiento presentan. Sin embargo, existen otros factores que pueden afectar el tiempo total de las pruebas y, por lo tanto, el rendimiento. Dentro de estos factores es probable que se puedan encontrar: el tipo de transacción, el consumo de recursos, la concurrencia, entre otros.

7.1.3 Consumo de recursos. Se observará el consumo de recursos que presenta cada modelo implementado en las aplicaciones tipo dentro de los grupos de hilos definidos. Este análisis comprende el consumo de CPU y el consumo de la memoria RAM durante la ejecución de las pruebas de rendimiento, y solo se observan los niveles de carga y grupos de hilos más exigentes para cada aplicación tipo.

En las Figuras 38 - 43 se observan gráficas del porcentaje de CPU y memoria RAM que consumen las aplicaciones tipo de cada tecnología durante el tiempo transcurrido de las pruebas seleccionadas.

Figura 38. Consumo de CPU en la aplicación tipo “Insertar”.

Figura 39. Consumo de CPU en la aplicación tipo “Consultar”.

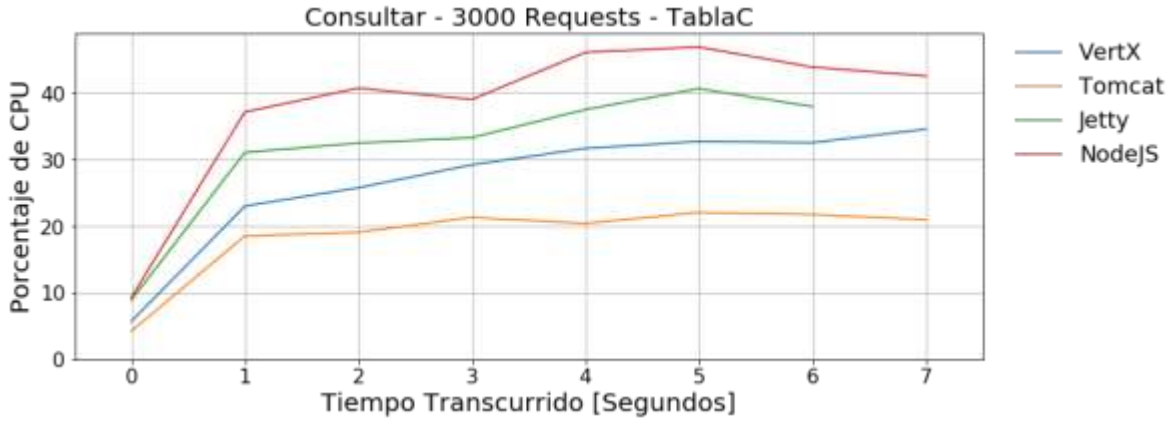


Figura 40. Consumo de CPU en la aplicación tipo “Contar Primos”.

Figura 41. Consumo de memoria RAM en la aplicación tipo “Insertar”.

Figura 42. Consumo de memoria RAM en la aplicación tipo “Consultar”.

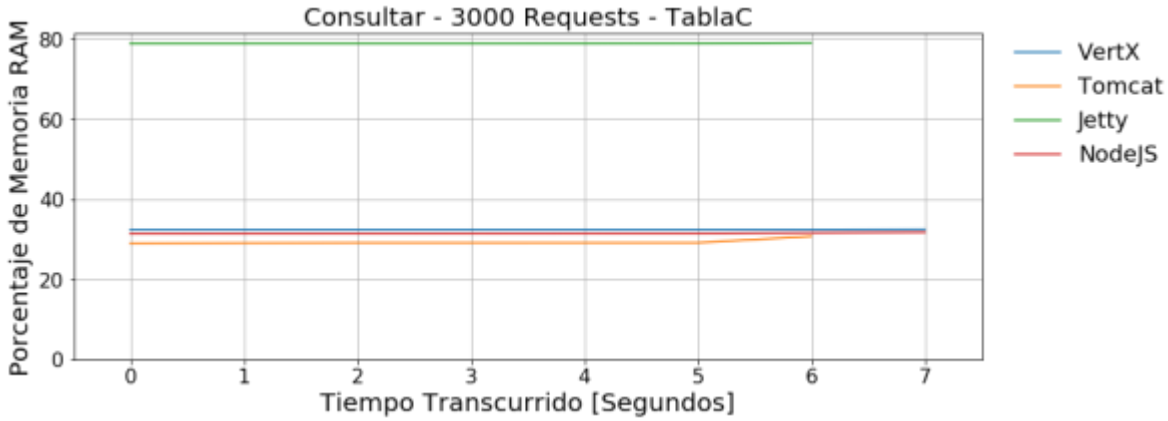
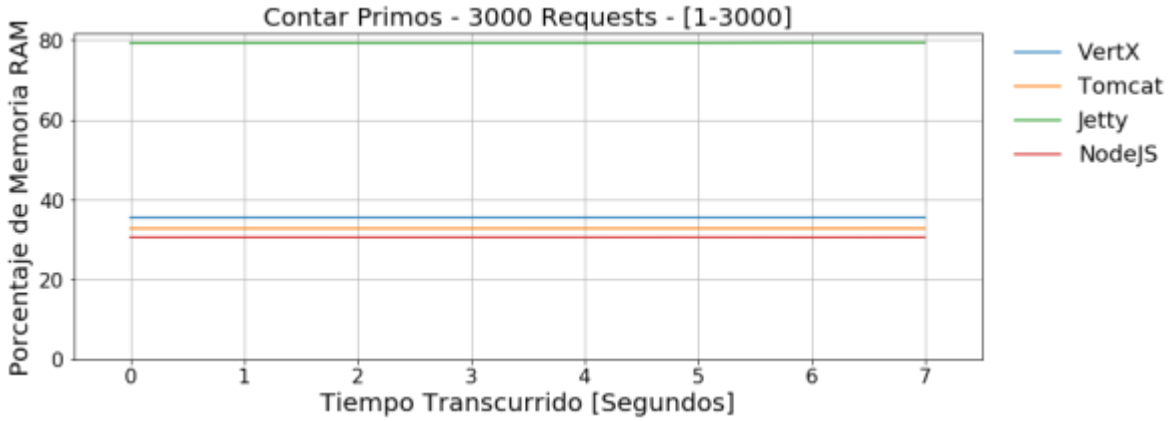


Figura 43. Consumo de memoria RAM en la aplicación tipo “Contar Primos”.



A pesar de que las aplicaciones están ofreciendo el mismo rendimiento, cada una está manejando los recursos del servidor de forma diferente. Dentro de los datos obtenidos se observa que:

- Node.js posee un consumo alto de CPU en las aplicaciones tipo “Consultar” y ”Contar Primos” y un consumo bajo de memoria RAM en todas las aplicaciones tipo, que es menor al de Vert.x y cercano al de Tomcat.

- Jetty en la aplicación tipo “Insertar” muestra un consumo alto de CPU manteniendo su valor cerca al 70% a lo largo de la prueba, en las demás aplicaciones tipo es el segundo en consumo de CPU, y es la aplicación que en general consume la mayor cantidad de memoria RAM.

- El consumo de CPU en Vert.x es inferior al de Node.js y Jetty pero cercano al de Tomcat; su consumo de memoria RAM es un poco mayor al de Tomcat y Node.js.

- La aplicación que tiende a consumir menos recursos de CPU y memoria RAM es Tomcat.

7.2 EVALUACIÓN A TRAVÉS DE LAS PRUEBAS DE CARGA

Para el análisis de las pruebas de carga se tienen en cuenta los siguientes indicadores clave del rendimiento: tiempo de respuesta, número de muestras iniciadas por segundo, número de usuarios activos, el rendimiento y el consumo de recursos de CPU y memoria RAM. El número de usuarios activos se refiere a la concurrencia de usuarios que mantiene JMeter durante los intervalos de tiempo definidos en las pruebas de carga.

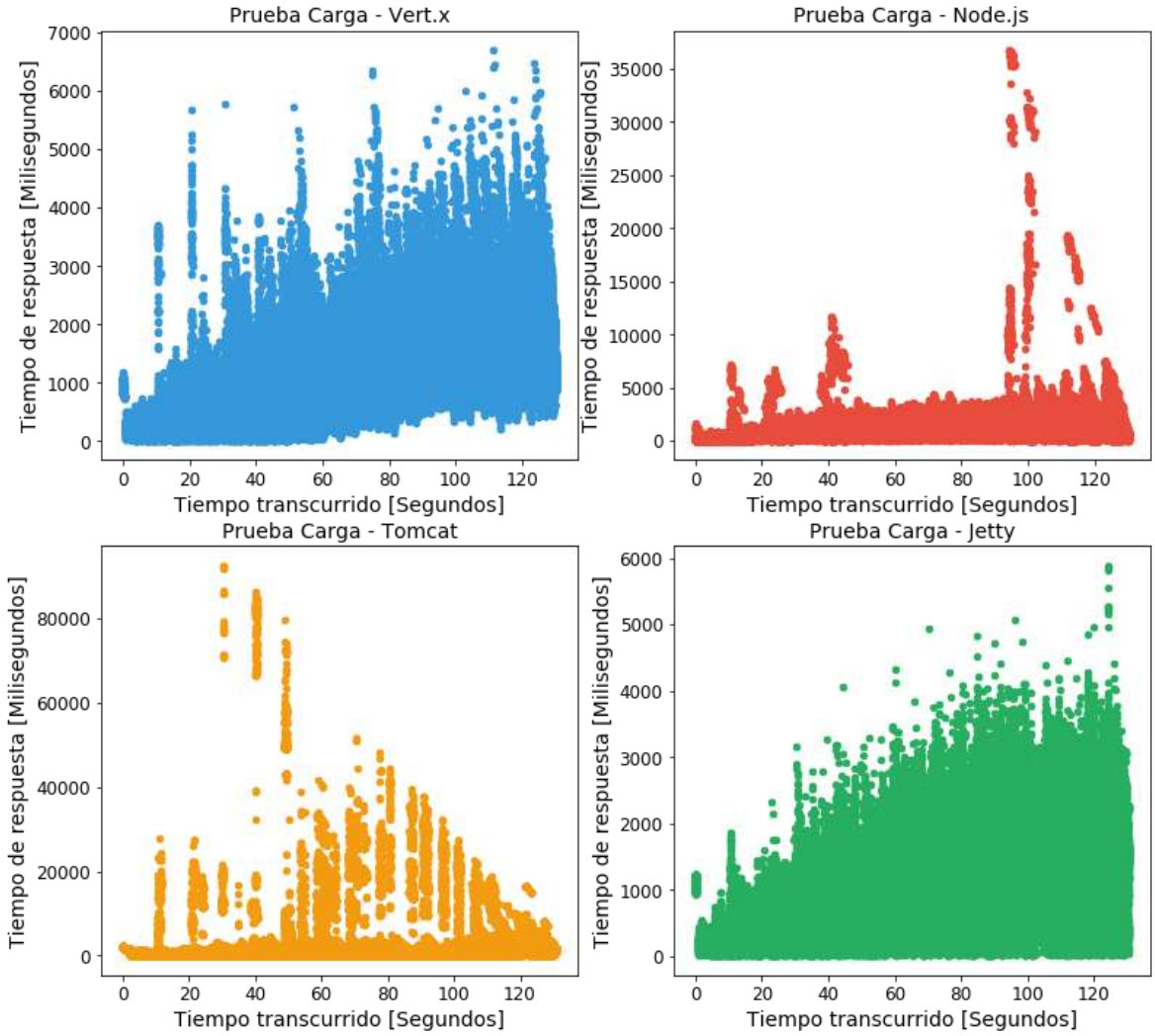
7.2.1 Tiempo de respuesta. Dado que la prueba de carga incrementa la concurrencia de usuarios cada cierto intervalo de tiempo, se pueden generar cuellos de botella que producen retrasos en la capacidad de respuesta de la aplicación. Estos cuellos de botella se detectan observando los cambios en el tiempo de respuesta de las solicitudes a lo largo de la ejecución de la prueba de carga, u observando los cambios en el número de muestras iniciadas. La causa de estos cuellos de botella puede estar relacionada con la falta de optimización en el código o la mala configuración del entorno de ejecución de la aplicación, entre otros factores.

Los cambios en el tiempo de respuesta a lo largo de la prueba de carga de cada una de las cuatro tecnologías implementadas en los modelos de programación se pueden observar en la Figura 44. Para entender estas gráficas se debe tener en cuenta que durante la prueba de carga, la concurrencia aumenta cada diez segundos hasta alcanzar un valor límite.

Si la aplicación posee un cuello de botella que limita el manejo del incremento de concurrencia, se manifestará con un aumento en los tiempos de respuesta. Mientras las aplicaciones se adaptan a este embotellamiento, la media de los tiempos de respuesta se incrementa hasta que los valores del tiempo de respuesta llegan a un punto estable, sin embargo, en este punto la dispersión de los datos es mayor.

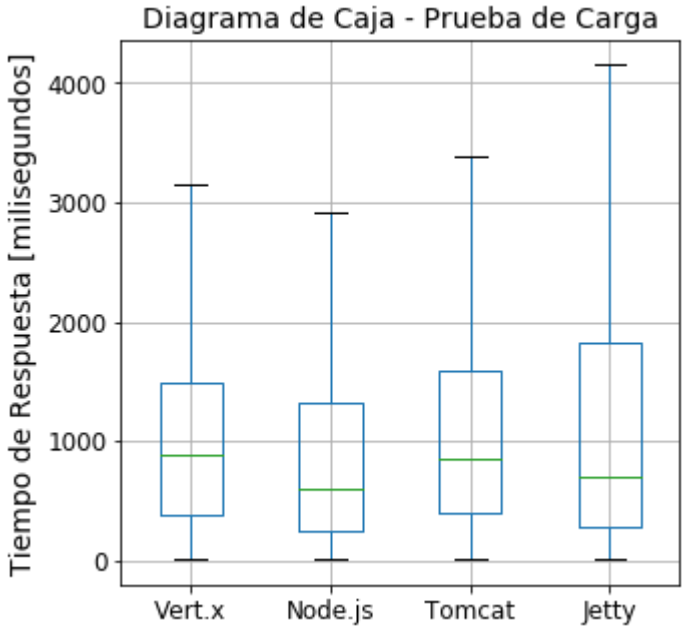
De las cuatro tecnologías, Node.js y Tomcat presentan un cuello de botella adicional que incrementa el tiempo de respuesta de algunas solicitudes de forma considerable, alcanzando valores superiores a 80.000 milisegundos en Tomcat y valores cercanos a 35.000 milisegundos en Node.js. Esto afecta de forma significativa la capacidad de respuesta de la aplicación y parece ser causado por un factor diferente al del incremento en la concurrencia, ya que no se observa este comportamiento en Vert.x y Jetty.

Figura 44. Tiempos de respuesta de la prueba de carga.



Para observar la dispersión del tiempo de respuesta en cada una de las cuatro tecnologías implementadas en los modelos de programación, y el rango de valores en el cual se concentran los datos obtenidos, se ha optado por usar un diagrama de caja que se muestra en la Figura 45. En este diagrama están representados los cuartiles, la mediana y los valores máximos y mínimos para el tiempo de respuesta de cada tecnología. Además, no se tienen en cuenta los valores atípicos, ya que son descartados durante el análisis estadístico.

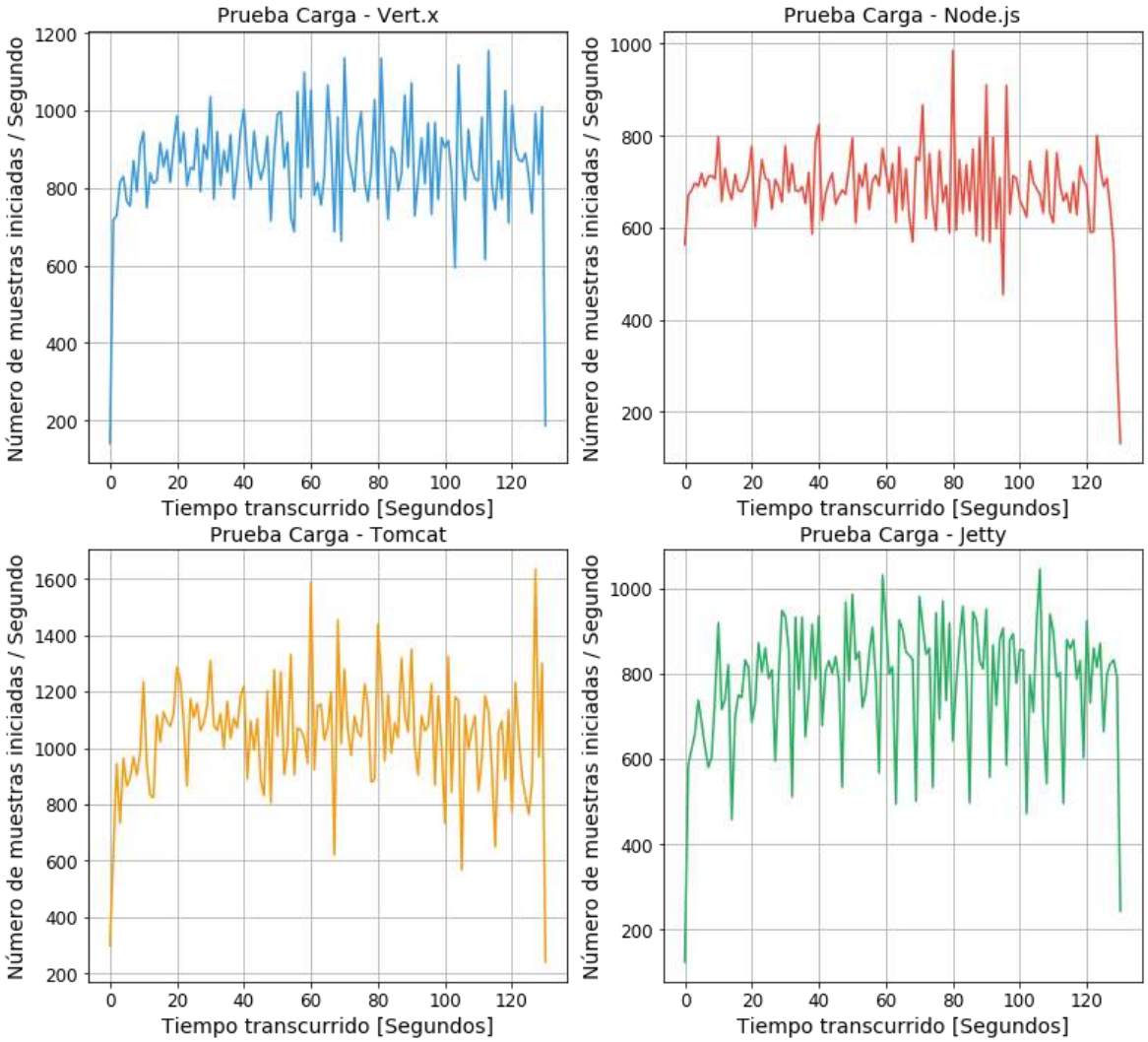
Figura 45. Distribución del tiempo de respuesta.



Para analizar el diagrama de caja se tiene en cuenta el rango de las aristas, la posición de la caja respecto al eje vertical y la ubicación de la mediana (línea de color verde). A mayor rango de aristas mayor dispersión de los datos, tal como se observa en Tomcat y Jetty. También se puede decir que Vert.x y Node.js son los que poseen menores tiempos de respuesta y menor dispersión, en comparación de lo que se observa en las otras tecnologías.

7.2.2 Número de muestras iniciadas por segundo. El número de muestras iniciadas por segundo es una aproximación al rendimiento en un instante de tiempo. El comportamiento ideal para este indicador es que no varíe a lo largo de la prueba y se mantenga en el valor máximo de rendimiento que puede ofrecer la aplicación. Pero si existe embotellamiento, el rendimiento disminuye y se genera ruido a lo largo de la prueba, produciendo los altibajos que se observan en la Figura 46.

Figura 46. Número de muestras iniciadas de la prueba de carga.



Para analizar las variaciones en el rendimiento se observan los rangos en los cuales las aplicaciones se mantuvieron a lo largo de la prueba de carga, con esto como base se tiene que:

- El rango de variación del rendimiento de Vertx se mantuvo en la mayor parte de la prueba entre [700 - 1000] y en algunas situaciones entre [500 - 1200].

- El rango de variación para Node.js creció de forma significativa cada cuarenta segundos. En los primeros cuarenta se mantuvo entre [600 - 800], luego pasó a estar entre [400 - 1000] y finalmente se mantuvo entre [200 - 1100]. Estos cambios bruscos en el rendimiento afectan el tiempo de respuesta de la aplicación, mostrando lo inestable que es al aumentar la concurrencia de usuarios.

- El rango de variación del rendimiento de Tomcat se mantuvo en la mayor parte de la prueba entre [1000 - 1200] y en algunas situaciones entre [700 - 1500].

- El rango de variación del rendimiento en Jetty se mantuvo aproximadamente entre [500 - 1000] a lo largo de toda la prueba.

Como se observa en la Figura 46, entre mayor sea el efecto del embotellamiento, mayor será el ruido causado en el número de muestras iniciadas por segundo. Este indicador permite visualizar cómo el embotellamiento afecta el rendimiento de la aplicación y en qué rango de valores se mantiene el rendimiento para un número de usuarios activos.

7.2.3 Consumo de recursos. Analizando el consumo de recursos que se observa en la Figura 47 y la Figura 48, se tiene que el consumo de CPU es similar para Vert.x, Node.js y Tomcat. Por el contrario, en Jetty se observa mucho ruido en el consumo de CPU indicando un manejo ineficiente de los recursos.

En lo que respecta al consumo de memoria RAM, Node.js mantiene un bajo consumo a lo largo de toda la prueba de carga. Por el contrario Tomcat, Jetty y Vert.x poseen un consumo elevado de memoria RAM. Es probable que la causa del embotellamiento drástico observado en Node.js (ver Figura 44 y Figura 46) sea un reflejo del uso ineficiente de la memoria RAM.

Figura 47. Consumo de CPU de la prueba de carga.

Figura 48. Consumo de memoria RAM de la prueba de carga.

7.2.4 Reporte Resumen. En la Tabla 8 se encuentra un resumen de las estadísticas obtenidas para las pruebas de carga de cada aplicación con respecto al tiempo de respuesta en milisegundos.

Se observa que las aplicaciones con mayor rendimiento con respecto a su límite y al número de muestras fueron Vert.x y Tomcat. Según la desviación estándar y los valores mínimo y máximo, las que presentaron mayor dispersión (teniendo en cuenta valores atípicos) fueron Tomcat y Node.js y según la media y la mediana las que poseen mayor simetría en sus distribuciones son Vert.x y Jetty.

Tabla 8. Resumen de la prueba de carga.

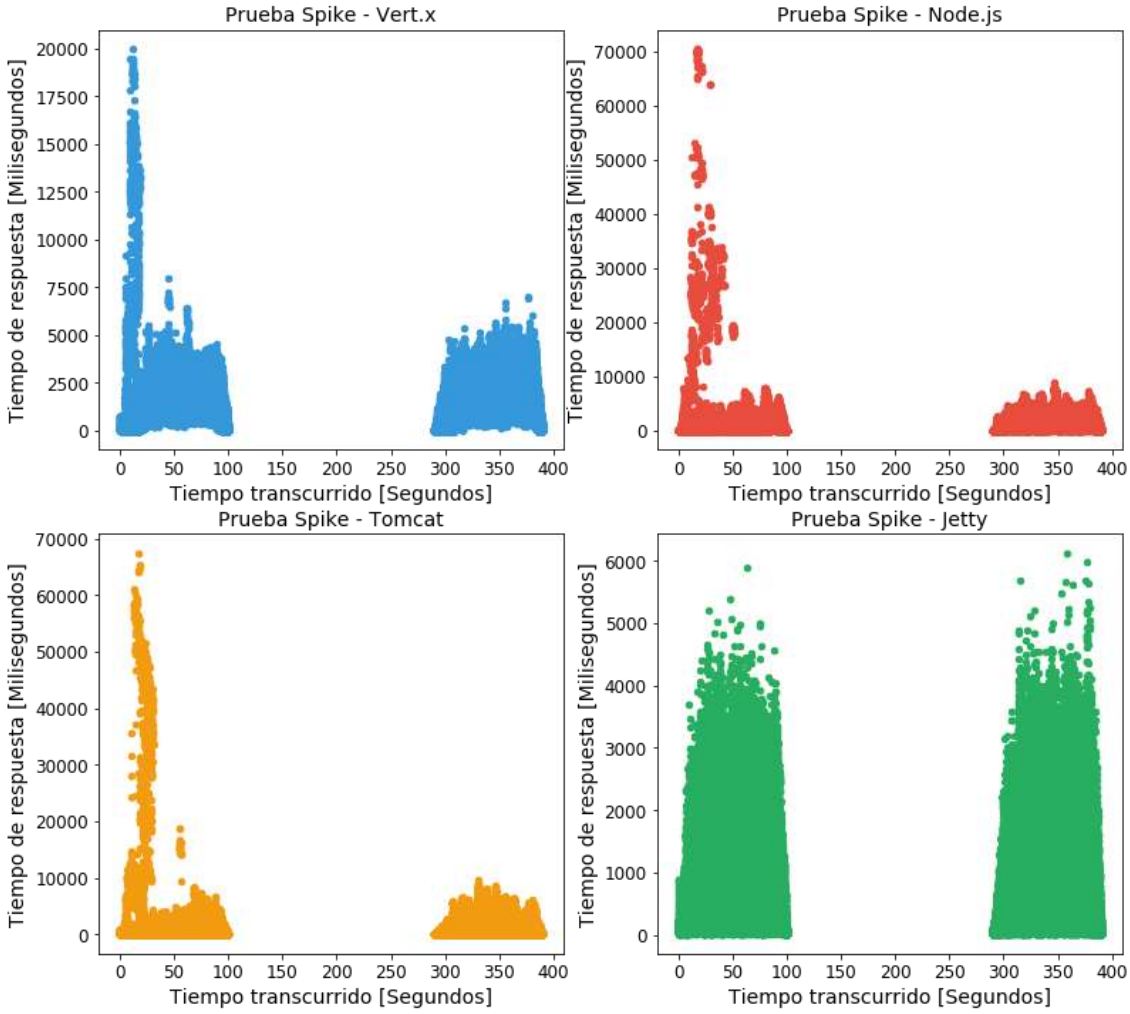
Tecnología	# Muestras	Media	Mediana	Mín	Máx	Rendimiento	Desv. Estándar	Límite
Vert.x	112628	1065	892	9	6692	856,2	851,15	1400
Tomcat	135807	1911	596	5	92540	982,0	5149,74	3000
Jetty	101997	1043	858	7	5888	769,9	771,01	1240
Node.js	89567	1227	694	8	36756	676,3	2116,14	1290

7.3 EVALUACIÓN A TRAVÉS DE LAS PRUEBAS DE ESTRÉS

La prueba de estrés (spike) se analiza observando las diferencias en el comportamiento de la aplicación al pasar por los picos de concurrencia. Los indicadores seleccionados para realizar esta comparación son el número de muestras iniciadas por segundo, el tiempo de respuesta y el tipo de error reportado.

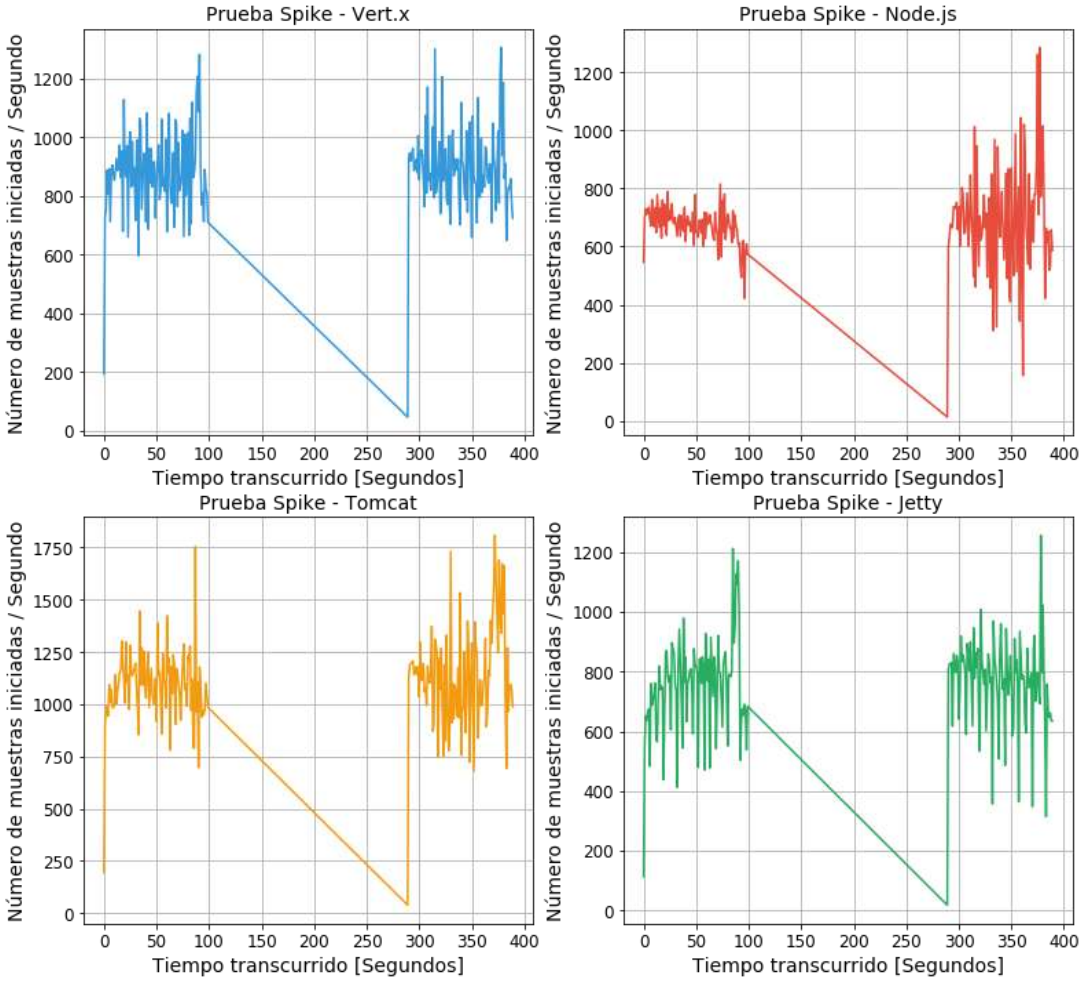
7.3.1 Tiempo de respuesta. En el primer pico de concurrencia, las aplicaciones pasan bruscamente de un estado de inactividad a uno de alta carga de trabajo, causando un embotellamiento en todas las aplicaciones excepto en Jetty (ver Figura 49). El impacto del embotellamiento en el primer pico de concurrencia fue menor para Vert.x que el observado en Node.js y Tomcat, los cuales presentaron valores de 70000 milisegundos en el tiempo de respuesta, siendo realmente altos comparados con los valores de 20000 milisegundos alcanzados en Vert.x.

Figura 49. Tiempos de respuesta de la prueba de estrés (spike).



7.3.2 Número de muestras iniciadas por segundo. Para poder observar la variación en el número de muestras iniciadas por segundo, basta con ver el rango dentro del cual se mantienen las tecnologías que representan cada modelo. Por ejemplo, al observar la Figura 50 se tiene que en Vert.x el número de muestras iniciadas por segundo en el primer pico de concurrencia varía de 600 a aproximadamente 1280 y en el segundo pico varía de 600 a aproximadamente 1300. En este ejemplo se observa que el rendimiento no se degrada por la carga de trabajo, y que su posible aumento sea debido al entrenamiento que tuvo la base de datos al pasar por el primer pico de carga.

Figura 50. Número de muestras iniciadas de la prueba de estrés (spike).



Analizando la Figura 50, se tiene que:

- La variación del número de muestras iniciadas por segundo de Vert.x se muestra similar en ambos picos de concurrencia. Además, su rango de variación se mantiene entre [600 - 1300].

- Tomcat también muestra una variación similar en ambos picos de concurrencia, sin embargo, en el segundo pico el número de muestras iniciadas por segundo alcanzó valores altos con más frecuencia que en el primero. Su rango de variación se mantiene entre [750 - 1750].

- Jetty no presenta cambios significativos en la variación del número de muestras iniciadas por segundo. Su rango se mantiene entre [400 - 1200].

- Node.js sí muestra una notable variación en el número de muestras iniciadas en los picos de concurrencia. En el primer pico, muestra una variación tenue dentro del rango [400 - 800]. Sin embargo, en el segundo pico de carga se muestran cambios bruscos entre valores altos y bajos, del número de muestras iniciadas por segundo en el rango [180 - 1300]. Esto evidencia que el rendimiento de la aplicación se degradó después de superar el primer pico de concurrencia, lo que también genera retrasos en los tiempos de respuesta.

7.3.3 Reporte resumen. En la Tabla 9, se muestra un resumen de los datos obtenidos para cada una de las aplicaciones con respecto al tiempo de respuesta en milisegundos.

Tabla 9. Resumen de la prueba de estrés (spike).

Tecnología	# Muestras	Media	Mediana	Mín	Máx	Desv. Estándar	% Error	Límite
Vert.x	178007	1198	1020	2	19955	989,90	1,67%	1324
Tomcat	223136	1063	603	2	67451	2552,60	2,50%	1500
Jetty	151701	1284	1267	2	6127	849,37	2,10%	1210
Node.js	136650	1477	950	2	120382	2950,10	2,11%	1250

El número de muestras y el valor límite que se observa en la Tabla 9 ofrece una aproximación al rendimiento. Si relacionamos estos dos indicadores por medio de la razón de cambio (# Muestras / Límite) se obtienen los resultados de la Tabla 10.

Tabla 10. Relación entre # Muestras y Límite.

	Node.js	Jetty	Vert.x	Tomcat
# Muestras / Límite	109,32	125,37	134,45	148,76

Por medio de esta relación se puede comparar el rendimiento entre las aplicaciones. Entre mayor sea el valor de la relación, mayor será el rendimiento de la aplicación presentado en la prueba de estrés.

Con los valores máximos y la desviación estándar (ver Tabla 9), se puede observar en cuál aplicación se presentan los mayores retrasos en los tiempos de respuesta. Estos retrasos bloquean la actividad de los usuarios evitando que puedan realizar más acciones. De las cuatro aplicaciones Tomcat y Node.js presentan valores máximos y desviación estándar elevados. En el caso de Tomcat que posee un alto rendimiento para esta prueba, el tener tiempos de respuesta tan elevados lo posiciona en segundo lugar, después de Vert.x como la mejor opción.

7.3.4 Reporte de errores obtenidos. Para hablar de pruebas de estrés es necesario que las aplicaciones superen su límite de concurrencia y empiecen a generar errores. Para los escenarios de prueba planteados en la prueba de estrés, el error reportado por todas las aplicaciones es el que se muestra en la Tabla 11.

Tabla 11. Errores obtenidos en la prueba de estrés (spike).

	Response Code	Response Message
Error	Non HTTP response code: java.net.NoRouteToHostException	Non HTTP response message: Cannot assign requested address (Address not available)

Una vez identificado el error, se debe entender qué lo causa para de esta forma optimizar las aplicaciones en caso de que sea necesario. La excepción arrojada

por las aplicaciones es **“java.net.NoRouteToHostException”**. Esta excepción ocurre cuando no se logra conectar a un socket en una dirección y puerto remotos. Este error solo se presenta cuando aparece uno de los siguientes escenarios:

- El host remoto no puede ser alcanzado por intervención de un firewall.
- El servidor al que se intenta conectar se encuentra caído.
- La aplicación supera el límite de conexiones TCP/IP permitidas por el sistema operativo.

Analizando cada uno de estos escenarios, se tiene que: primero, no se está implementando ningún firewall en el servidor; segundo, el servidor no se cae debido a que continúa recibiendo solicitudes, y así manteniendo un margen de error constante e inferior al 3%. Por último, el escenario más probable es aquel donde se está superando el límite de conexiones TCP/IP configuradas en el sistema operativo del servidor.

Una vez se detecta la causa del error, se procede a analizar si es necesario eliminarlo. Dentro de este análisis se tiene en cuenta la concurrencia con la que normalmente trabaja la aplicación, los riesgos secundarios generados por el error, la disponibilidad de los recursos, entre otros. En lo que respecta al error presentado, no siempre es viable aumentar la disponibilidad de los recursos al máximo, ya que dependiendo de la situación estos recursos pueden estar siendo desperdiciados con el tiempo por falta de uso.

7.3.5 Consumo de recursos. El consumo de CPU y memoria RAM de las aplicaciones en la prueba de estrés se muestra en la Figura 51 y Figura 52. Se puede ver que el consumo de CPU es similar al presentado en la prueba de carga, donde Jetty continúa mostrando ruido durante su consumo, lo que refleja ineficiencia en el manejo de procesos. En lo que se refiere al consumo de

memoria RAM durante la prueba de estrés, Node.js es el único que no logra aprovechar este recurso al máximo.

Figura 51. Consumo de CPU de la prueba de estrés.

Figura 52. Consumo de memoria RAM de la prueba de estrés.

Con el análisis realizado en el consumo de recursos de las pruebas de rendimiento, de carga y de estrés, se observa que las aplicaciones tipo implementadas en Node.js mantienen un bajo consumo de memoria RAM sin importar la carga de trabajo o el tipo de tareas que ejecute, dejando la mayor parte del trabajo al CPU. Además, el consumo de recursos observado durante la ejecución de todas las pruebas, permitió observar que Vert.x y Tomcat administran

de forma más eficiente el manejo de los procesos en la CPU y la memoria RAM, dado que solo se aumenta su consumo si la situación lo requiere.

Por último, se observa que Jetty mostró un alto consumo de RAM cuando no existía mucha concurrencia de usuarios y también mostró inestabilidad en el manejo de la CPU cuando se llevó a su límite máximo de concurrencia. Esto quiere decir que necesita mayor optimización que Tomcat en el manejo de los recursos disponibles en el servidor, a pesar de que ambas aplicaciones se desarrollaron de forma similar y manteniendo las mismas características en la mayor parte del código.

8. CONCLUSIONES

- La evaluación de rendimiento de una aplicación web requiere analizar diversos aspectos como el tipo de aplicación, el comportamiento de la demanda, el comportamiento de la aplicación bajo situaciones de estrés o carga normal, entre otras. En este proyecto se abarcó un conjunto importante de escenarios de prueba para analizar y caracterizar el comportamiento de las aplicaciones desde el punto de vista del rendimiento (tiempo medio de respuesta, throughput, uso de memoria RAM y procesador).

- La evaluación de alternativas tecnológicas es uno de los principales procesos de la ingeniería, siendo un factor estratégico del cual depende en gran medida el éxito del proyecto a desarrollar. Este proceso es aún más complejo en la ingeniería de software, dada la variedad de herramientas, tecnologías y modelos existentes. En este proyecto se abordó la evaluación tecnológica a lo largo de la metodología y el análisis planteado en la comparación del rendimiento.

- La escalabilidad de las aplicaciones implementadas se vio limitada por diferentes tipos de factores, y la única forma de detectarlos fue por medio de las pruebas de rendimiento y sus variantes. Cabe resaltar, que el proceso de probar las aplicaciones durante su etapa de desarrollo consume una considerable cantidad de tiempo, dada la cantidad de pruebas necesarias para inspeccionar los cambios realizados.

- El objetivo de este proyecto es el de evaluar el rendimiento de diferentes modelos de programación web, sin embargo, este aspecto se ha visto afectado por otros factores, tales como la configuración óptima de las

distintas herramientas empleadas, la selección del servidor donde se alojan las aplicaciones, entre otros. La determinación de estos factores y su impacto se basó en un proceso repetitivo de prueba y análisis del comportamiento, siendo este un proceso complejo que consume una cantidad considerable de tiempo. Los procesos del tipo DevOps cobran importancia al acercar los equipos de desarrollo y operaciones para ofrecer mejores resultados al durante el desarrollo del software.

- Se concluye que el proceso de desarrollo de cualquier aplicación siempre debe ir de la mano con un proceso de pruebas controlado y automatizado, ya que esto permite detectar factores de riesgo en etapas tempranas, ahorrando de esta forma tiempo y dinero.

- El límite en las pruebas de estrés y de carga es relativo, ya que según el análisis realizado para la prueba de estrés, el error reportado por las aplicaciones está relacionado con el límite de conexiones TCP/IP permitidas en el sistema operativo del servidor. Por ende, la diferencia en el límite de concurrencia que soportan las aplicaciones en las pruebas de carga y estrés está directamente relacionada con la forma en la que se manejan las conexiones en cada tecnología, siendo algunas más rápidas que otras.

- Se concluye que los modelos asíncronos (Vert.x y Node.js), mostraron menos dispersión en los tiempos de respuesta que los modelos síncronos (Tomcat y Jetty), esto quiere decir que la mayoría de las solicitudes se atendieron en tiempos relativamente cortos.

9. RECOMENDACIONES

- Una sugerencia para trabajos futuros de pruebas de rendimiento es agregar más aplicaciones tipo como actualizar y eliminar registros, con un grupo de hilos más grande en una base de datos no relacional como MongoDB.

- Es recomendable ejecutar pruebas de carga sobre las aplicaciones antes de las pruebas de rendimiento finales, con el fin de detectar cuellos de botella a tiempo y así mejorar el rendimiento de la aplicación.

- Hacer una comparativa de nuevos Frameworks para Front-end con pruebas de rendimiento.

- Si se van a ejecutar pruebas de carga muy pesadas en JMeter, se debe aumentar su “Heap Size”, que por defecto se encuentra en 512 MB siendo bastante bajo. También, es importante ejecutar JMeter en modo no GUI, esto con el fin de bajar el consumo de RAM y evitar que salgan errores como “java.lang.OutOfMemoryError”.

- Realizar la evaluación del rendimiento de las aplicaciones enfocándose en diferentes escenarios de configuración, aplicando la metodología de afinamiento de rendimiento de sistemas (performance tuning).

BIBLIOGRAFÍA

ALCÓCER GARCÍA, Alejandro Carlos. Redes de computadoras: Servicios de aplicación: TELNET, FTP, TFTP, NFS, SMTP, POP3, IMAP4, WWW Y GOPHER. [En línea]. Perú: Infolink, 2000. p. 318 - 337. (Recuperado en 16 marzo 2018). Disponible en http://repositorio.pucp.edu.pe/index/bitstream/handle/123456789/28691/Redes_Cap22.pdf?sequence=22

ÁLVAREZ, Cecilio. ¿Cómo funciona Node.js?. [En línea]. En: Genbeta, 2014. (Recuperado en 9 septiembre 2018). Disponible en https://www.genbeta.com/desarrollo/como-funciona-node-js

ÁLVAREZ, Esaú. Qué son Microservicios y ejemplos reales de uso. [En línea]. En: OpenWebinars, 2016. (Recuperado en 16 marzo 2018). Disponible en https://openwebinars.net/blog/microservicios-que-son/

AMAZON WEB SERVICES. ¿Qué es Amazon EC2? - Amazon Elastic Compute Cloud. [En línea]. En: Documentación de AWS. (Recuperado en 25 enero 2019). Disponible en https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html

--------. Tipos de instancias de Amazon EC2. [En línea]. En: Amazon Web Services. (Recuperado en 25 enero 2019). Disponible en https://aws.amazon.com/es/ec2/instance-types/

APACHE SOFTWARE FOUNDATION. Apache JMeter. [En línea]. En: Apache JMeter. (Recuperado en 22 septiembre 2018). Disponible en https://jmeter.apache.org/

--------. Apache Tomcat. [En línea]. En: Apache Tomcat, 2019. (Recuperado en 13 septiembre 2018). Disponible en http://tomcat.apache.org/

--------. Maven – Introduction. [En línea]. En: Apache Maven Project, 2019. (Recuperado en 19 septiembre 2018). Disponible en https://maven.apache.org/what-is-maven.html

ARIAS FISTEUS, Jesús. Servlets. [En línea]. En: Universidad Carlos III de Madrid - Ingeniería Telemática, 2008. (Recuperado en 5 septiembre 2018). Disponible en https://www.it.uc3m.es/labttlat/2007-08/material/servlets/

BANSODE, Prashant, *et al.* Performance Testing Guidance for Web Applications. [En línea]: Chapter 2 – Types of Performance Testing. En: Microsoft, 2010. (Recuperado en 17 marzo 2018). Disponible en https://msdn.microsoft.com/en-us/library/bb924357.aspx

BELLIDO, Félix y GÓMEZ FONTANILLS, David. Herramientas de Evaluación Tecnológica en Gestión de la tecnología. [En línea]. En: Wiki EOI, 2012. (Recuperado en 17 marzo 2018). Disponible en http://www.eoi.es/wiki/index.php/Herramientas_de_Evaluaci%C3%B3n_Tecnol%C3%B3gica_en_Gesti%C3%B3n_de_la_tecnolog%C3%ADa

BLONDEAU, Antoine. FinistDevs April Edition: About Microservices and Infrastructure Automation. [En línea]. En: A Medium Corporation, 2018. (Recuperado en 25 enero 2018). Disponible en https://medium.com/@ant.blondeauw/finistdevs-april-edition-about-microservices-and-infrastructure-automation-664b66d34f48

BONÉR, Jonas, *et al.* The Reactive Manifesto. [En línea]. En: Reactive Manifesto. (Recuperado en 18 marzo 2018). Disponible en https://www.reactivemanifesto.org/

DB-Engines Ranking. [En línea]. En: DB-Engines, 2018. (Recuperado en 18 septiembre 2018). Disponible en http://db-engines.com/en/ranking

DEITEL, Paul y DEITEL, Harvey. Visual C# How to Program: Chapter 28 Asynchronous Programming with async and await. 5 ed. Upper Saddle River, NJ: Pearson Education, 2014. p. 1 - 14.

DIRKSEN, Jos. Jetty. [En línea]: A Lightweight, Open-Source Web Server and Servlet Container. Cary: DZone Refcardz, 2012. (Recuperado en 5 septiembre 2018). Disponible en https://dzone.com/refcardz/jetty

DOCKER INC. Java. [En línea]. En: Docker Hub. (Recuperado en 15 noviembre 2018). Disponible en https://hub.docker.com/_/java

--------. Jetty. [En línea]. En: Docker Hub. (Recuperado en 15 noviembre 2018). Disponible en https://hub.docker.com/_/jetty

--------. MySQL. [En línea]. En: Docker Hub. (Recuperado en 15 noviembre 2018). Disponible en https://hub.docker.com/_/mysql

--------. Node. [En línea]. En: Docker Hub. (Recuperado en 15 noviembre 2018). Disponible en https://hub.docker.com/_/node/

--------. Tomcat. [En línea]. En: Docker Hub. (Recuperado en 15 noviembre 2018). Disponible en https://hub.docker.com/_/tomcat

ECLIPSE FOUNDATION. Jetty - Servlet Engine and Http Server. [En línea]. En: Eclipse, 2016. (Recuperado en 6 septiembre 2018). Disponible en http://www.eclipse.org/jetty/

ESCOFFIER, Clement. Building Reactive Microservices in Java Asynchronous and Event-Based Application Design. Sebastopol: O'Reilly Media, 2017. p. 3 - 4.

EVANS, Clark; BEN-KIKI, Oren y DÖT NET, Ingy. YAML 1.2. [En línea]. En: The Official YAML Web Site, 2009. (Recuperado en 23 noviembre 2018). Disponible en https://yaml.org/

FONSECA, Rodrigo. Introduction to Asynchronous Programming. [En línea]. En: Computer Science at Brown University. p. 1 - 6. (Recuperado en 19 marzo 2018). Disponible en http://cs.brown.edu/courses/cs168/s12/handouts/async.pdf

FREE SOFTWARE FOUNDATION. GNU Bash. [En línea]. En: GNU, 2017. (Recuperado en 15 noviembre 2018). Disponible en https://www.gnu.org/software/bash/

JUNTA DE ANDALUCÍA. Ejecución de Pruebas con JMeter. [En línea]. España: Marco de Desarrollo de la Junta de Andalucía. (Recuperado en 23 septiembre 2018). Disponible en http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/401

KULANDAI, Joseph. What is Servlet. [En línea]. En: Javapapers, 2014. (Recuperado en 10 septiembre 2018). Disponible en https://javapapers.com/servlet/what-is-servlet/

MALDONADO, Viviana Andrea. Performance-MW. [En línea]. En: GitHub. (Recuperado en 29 enero 2018). Disponible en https://github.com/vivianamaldonado/Performance-MW

MTOP: MONITOREO de carga en MySQL [Anónimo]. [En línea]. Gabriel Barrera. 28 de octubre de 2009. (Recuperado en 18 septiembre 2018). Disponible en http://tecnocacharrero.blogspot.com/2009/10/mtop-monitoreo-de-carga-en-mysql.html

MUNTZ, Richard. Performance Measurement and evaluation. En: Encyclopedia of Computer Science. Enero, 2003, p. 1385-1390.

MUÑOZ, José Domingo. Introducción a docker. [En línea]. En: PLEDIN 3.0, 2015. (Recuperado en 20 septiembre 2018). Disponible en https://www.josedomingo.org/pledin/2015/12/introduccion-a-docker/

--------.-------- Primeros pasos con Docker. [En línea]. En: PLEDIN 3.0, 2016. (Recuperado en 21 septiembre 2018). Disponible en https://www.josedomingo.org/pledin/2016/02/primeros-pasos-con-docker/

OKUR, Semih, *et al.* A Study and Toolkit for Asynchronous Programming in C#. En: Proceedings of the 36th International Conference on Software Engineering. Mayo - junio, 2014. p. 1 - 12.

ORACLE. The Java Servlet API White Paper. [En línea] En: Oracle. (Recuperado en 10 septiembre 2018). Disponible en https://www.oracle.com/technetwork/java/whitepaper-135196.html

OXFORD UNIVERSITY PRESS. Reactive. [En línea]. En: Oxford Dictionaries. (Recuperado en 18 de marzo 2018). Disponible en https://en.oxforddictionaries.com/definition/reactive

PONGE, Julien; SEGISMONT, Thomas y VIET, Julien. A gentle guide to asynchronous programming with Eclipse Vert.x for Java developers. [En línea]. En: Eclipse Vert.x, 2018. (Recuperado en 23 septiembre 2018). Disponible en https://vertx.io/docs/guide-for-java-devs/#_core_vert_x_concepts

--------.-------- A gentle guide to asynchronous programming with Eclipse Vert.x for Java developers. [En línea]. En: Eclipse Vert.x, 2018. (Recuperado en 23 septiembre 2018). Disponible en https://vertx.io/docs/guide-for-java-devs/#_what_is_vert_x

POSA, Rambabu. Node JS Architecture - Single Threaded Event Loop. [En línea]. En: JournalDev, 2015. (Recuperado en 15 septiembre 2018). Disponible en https://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop

REDMOND, Eric. The Maven 2 POM demystified. [En línea]. En: JavaWorld, 2006. (Recuperado en 19 septiembre 2018). Disponible en

https://www.javaworld.com/article/2071772/java-app-dev/the-maven-2-pom-demystified.html

SHAPIRA, Yoav; ARCAND, Jeanfrancois y HANIK, Filip. The Apache Tomcat 5.5 Servlet/JSP Container. [En línea]: Tomcat Architecture. En: Apache Tomcat, 2012. (Recuperado en 14 septiembre 2018). Disponible en https://tomcat.apache.org/tomcat-5.5-doc/architecture/overview.html

SOTO, Jason. Arquitectura de Docker. [En línea]. En: GitBook - Jsitech1, 2016. (Recuperado en 20 septiembre 2018). Disponible en https://jsitech1.gitbooks.io/meet-docker/content/arquitectura_de_docker.html

--------.-------- Docker Compose. [En línea]. En: GitBook - Jsitech1, 2016. (Recuperado en 21 septiembre 2018). Disponible en https://jsitech1.gitbooks.io/meet-docker/content/docker_compose.html

SYNCHRONOUS VS. Asynchronous Execution [Anónimo]. [En línea]. En: Software Bisque. (Recuperado en 01 abril 2018). Disponible en https://www.bisque.com/products/orchestrate/RASCOMHelp/RASCOM/Synchronous_vs_Asynchronous_Execution.htm

TOMCAT HOSTING with JVM Host - your Java and Tomcat Host [Anónimo]. [En línea] En: JVMHost, 2013. (Recuperado en 13 septiembre 2018). Disponible en https://www.jvmhost.com/tomcat-hosting/

TORO, Luigys. Jupyter notebook: documenta y ejecuta código desde el navegador. [En línea]. Blog DESDELINUX. 21 de septiembre de 2017. (Recuperado en 27 enero 2019). Disponible en

https://blog.desdelinux.net/jupyter-notebook/

TSVETINOV, Nickolay. Learning Reactive Programming With Java 8. Kindle ed. Packt Publishing, 2015, p. 2 - 4.

VÉLEZ REYES, Javier. Programación asíncrona en Node JS [diapositivas]. Slideshare. 29 de mayo de 2014, diapositivas 1-53. (Recuperado en 19 marzo 2018). Disponible en https://es.slideshare.net/jvelez77/presentacion-35264918

--------.-------- Programación asíncrona: paso de continuadores, eventos, promesas y generadores. [En línea]. En: TodoJS, 2015. (Recuperado en 19 marzo 2018). Disponible en https://www.todojs.com/programacion-asincrona-paso-de-continuadores-eventos-promesas-y-generadores/

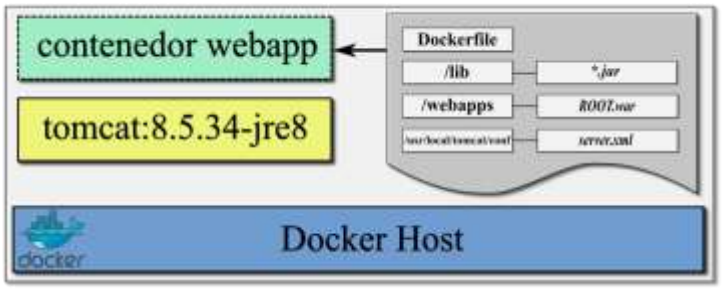
WALSH, Daniel. ¿Qué es DOCKER?. [En línea]. En: Red Hat. (Recuperado en 20 septiembre 2018). Disponible en https://www.redhat.com/es/topics/containers/what-is-docker

ANEXOS

Anexo A. Estructura de las imágenes Docker implementadas en el proyecto

La primera imagen de Docker diseñada es para la aplicación de Tomcat y utiliza la imagen oficial de Tomcat, versión '8.5.34-jre8' de Docker Hub [47]. Con Maven se crea el '.war' de la aplicación que es agregado en la carpeta 'webapps' donde se alojan las aplicaciones de Tomcat en la imagen Docker, además del archivo de configuración 'server.xml' en el directorio '/usr/local/tomcat/conf/'. Estas instrucciones están especificadas en el Dockerfile de tomcat. La Figura 53 muestra la estructura de la imagen diseñada y su contenido.

Figura 53. Estructura de la imagen Docker para la aplicación de Java Servlets con Tomcat.



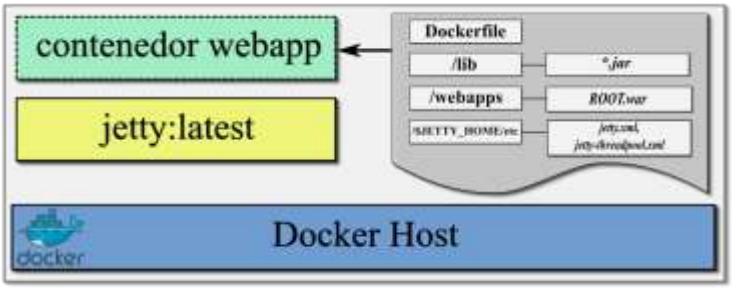
La segunda imagen de Docker creada es para la aplicación de Servlets con Jetty. Esta imagen utiliza la última versión oficial de Jetty disponible en Docker Hub [48]. A través de Maven se crea el '.war' de la aplicación que se agrega en la carpeta 'webapps' de la imagen Docker. Se agregan los archivos de configuración jetty.xml y jetty-threadpool.xml en el directorio '/$JETTY_HOME/etc', y se especifica el

[47] DOCKER INC. Tomcat. [En línea]. En: Docker Hub. (Recuperado en 15 noviembre 2018). Disponible en https://hub.docker.com/_/tomcat

[48] DOCKER INC. Jetty. [En línea]. En: Docker Hub. (Recuperado en 15 noviembre 2018). Disponible en https://hub.docker.com/_/jetty

puerto en el que se escuchará. Estas instrucciones se encuentran en el Dockerfile de Jetty. La Figura 54 muestra el diseño de la imagen Docker para Jetty.

Figura 54. Estructura de la imagen Docker para la aplicación de Java Servlets con Eclipse Jetty.

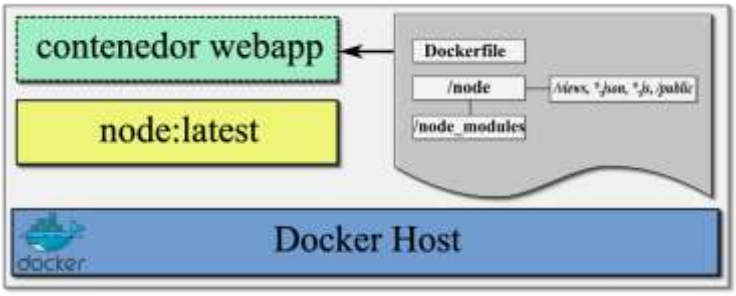


La tercera imagen de Docker creada es para la aplicación de NodeJS, a partir de la última versión de la imagen oficial de NodeJS disponible en Docker Hub [49]. Se crea un directorio llamado '/node' que será el directorio de trabajo para la aplicación. A continuación, se agrupa el código fuente de la aplicación dentro de la imagen Docker, se instalan las dependencias de NodeJS ejecutando 'npm install', se especifica el puerto y el comando para hacer el despliegue de la aplicación.

Las indicaciones anteriores se encuentran escritas en el Dockerfile para la imagen diseñada de NodeJS. La Figura 55 muestra la estructura de de la imagen Docker que se ha creado para NodeJS.

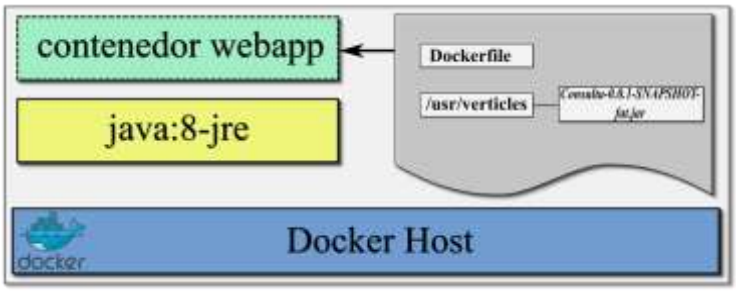
[49] DOCKER INC. Node. [En línea]. En: Docker Hub. (Recuperado en 15 noviembre 2018). Disponible en https://hub.docker.com/_/node/

Figura 55. Estructura de la imagen Docker para la aplicación de Node.js.



Para la creación de la cuarta imagen de Docker de la aplicación de Vert.x, se utiliza la imagen oficial de Java, versión '8-jre' de Docker Hub [50]. Se crea un directorio llamado '/usr/verticles' en donde se agrega a la imagen Docker el archivo '-fat.jar' de la aplicación, generado por Maven. El puerto es especificado y seguido a esto, se ejecuta el comando java para desplegar la aplicación de Vert.x. Estas instrucciones se encuentran en el Dockerfile de la imagen creada y su estructura se puede visualizar en la Figura 56.

Figura 56. Estructura de la imagen Docker para la aplicación de Vert.x.



Finalmente, se crea la quinta imagen de Docker para MySQL, utilizando la imagen oficial de MySQL en su versión '5.7.23' de Docker Hub [51]. Se agrega a la imagen Docker el archivo '.sql' que contiene las instrucciones para crear la base de datos

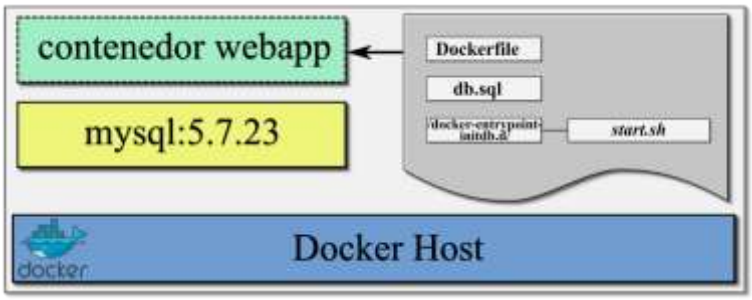
[50] DOCKER INC. Java. [En línea]. En: Docker Hub. (Recuperado en 15 noviembre 2018). Disponible en https://hub.docker.com/_/java

[51] DOCKER INC. MySQL. [En línea]. En: Docker Hub. (Recuperado en 15 noviembre 2018). Disponible en https://hub.docker.com/_/mysql

y los procedimientos almacenados del proyecto. Luego, se agrega un script de bash [52] con las instrucciones para crear un usuario con todos lo privilegios y leer el archivo '.sql', dentro de la carpeta '/docker-entrypoint-initdb.d/'.

Una vez se haga el despliegue del contenedor Docker de la imagen diseñada, se ejecutarán todos los archivos con extensión '.sql' y '.sh' que se encuentren en esta carpeta. En el Dockerfile de MySQL se encuentran cada una de las instrucciones mencionadas y se puede ver la estructura de la imagen diseñada en la Figura 57.

Figura 57. Estructura de la imagen Docker de MySQL.



[52] FREE SOFTWARE FOUNDATION. GNU Bash. [En línea]. En: GNU, 2017. (Recuperado en 15 noviembre 2018). Disponible en https://www.gnu.org/software/bash/

Anexo B. Diagrama de flujo del script de automatización

Figura 58. Diagrama de flujo del script para la automatización de pruebas. Parte 1.

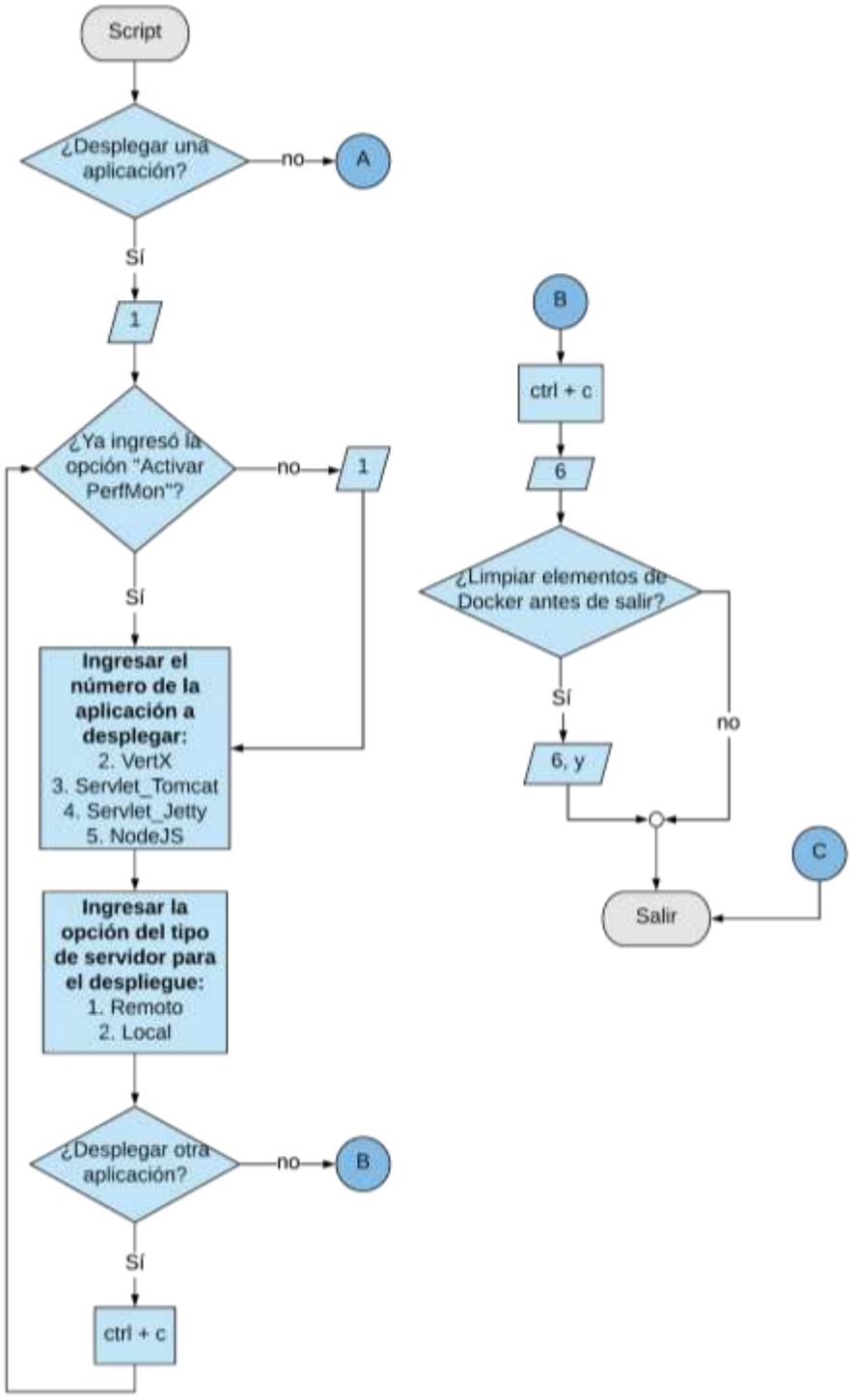
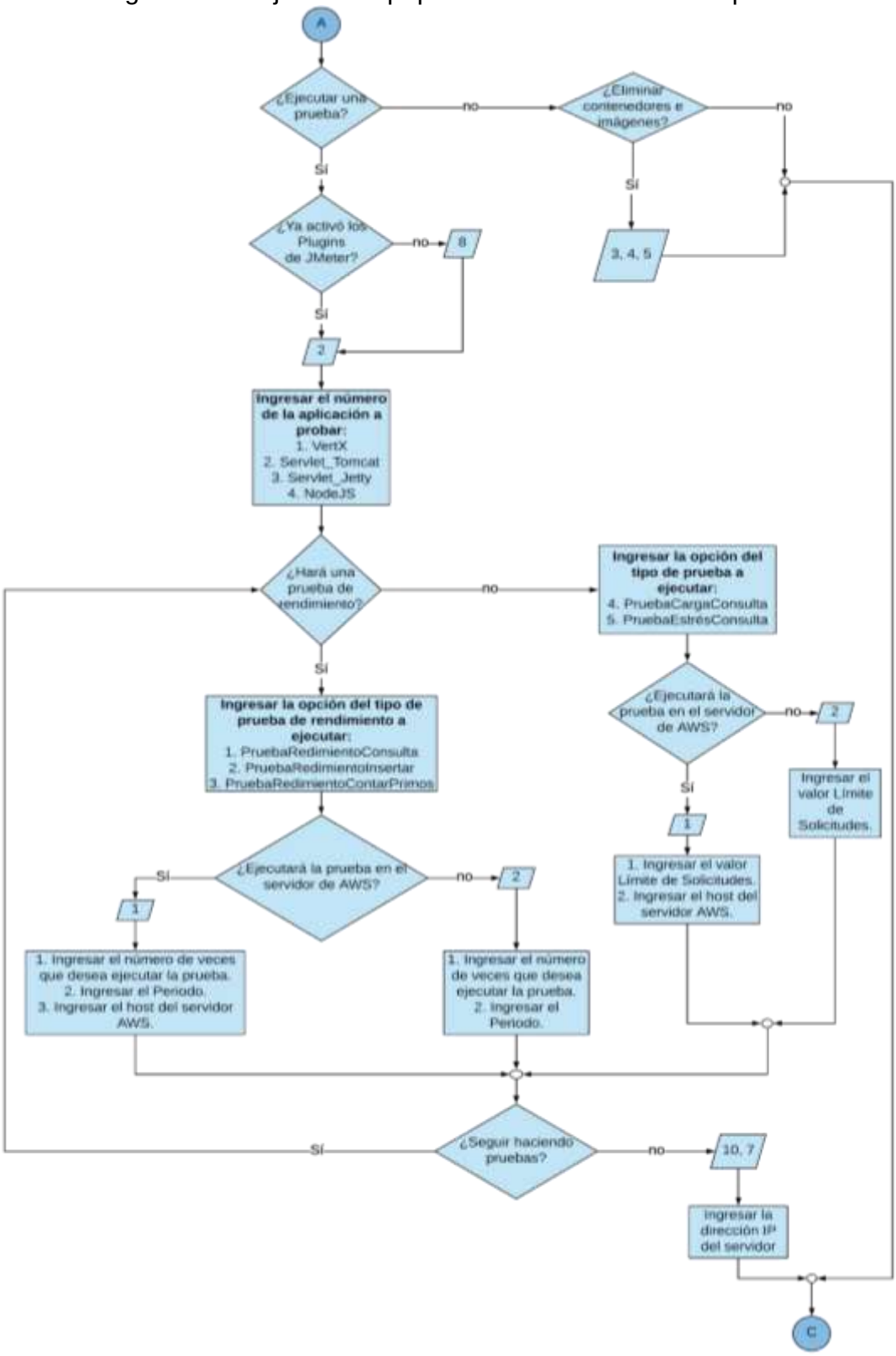


Figura 59. Diagrama de flujo del script para la automatización de pruebas. Parte 2.



Anexo C. Mapa de calor de las desviaciones estándar en la prueba de rendimiento

Figura 60. Desviación estándar para “Insertar” con Periodo 6.

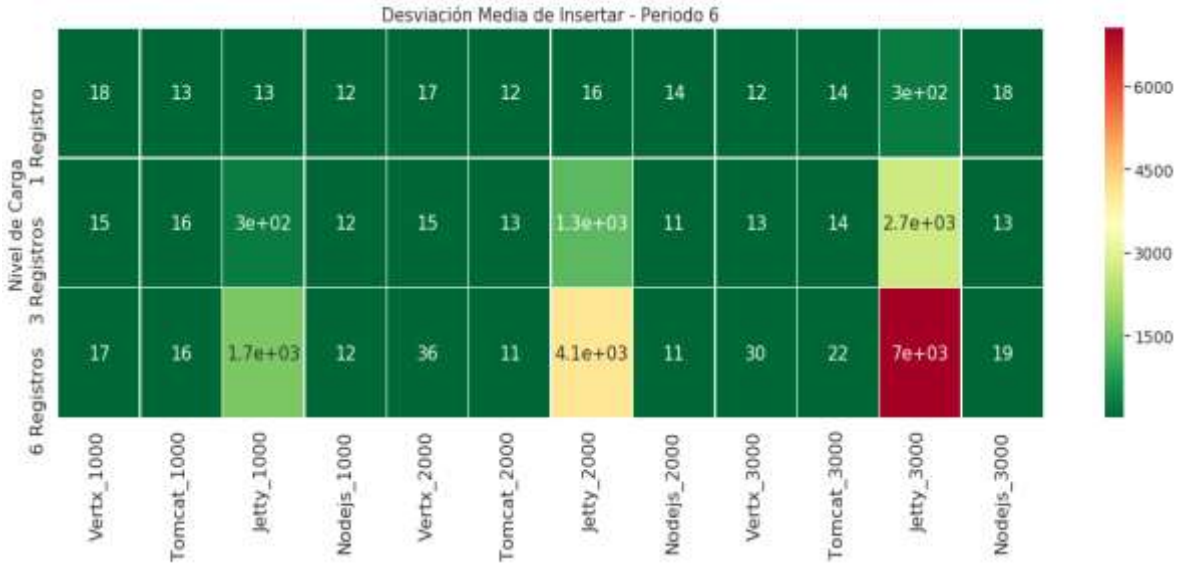


Figura 61. Desviación estándar para “Insertar” con Periodo 8.

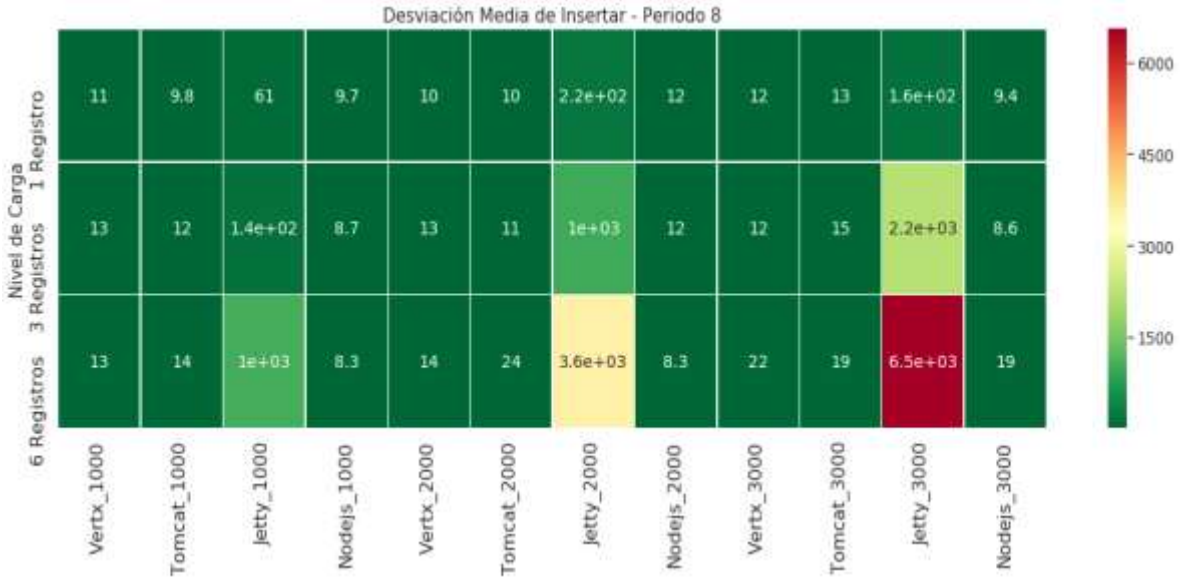


Figura 62. Desviación estándar para “Insertar” con Periodo 10.

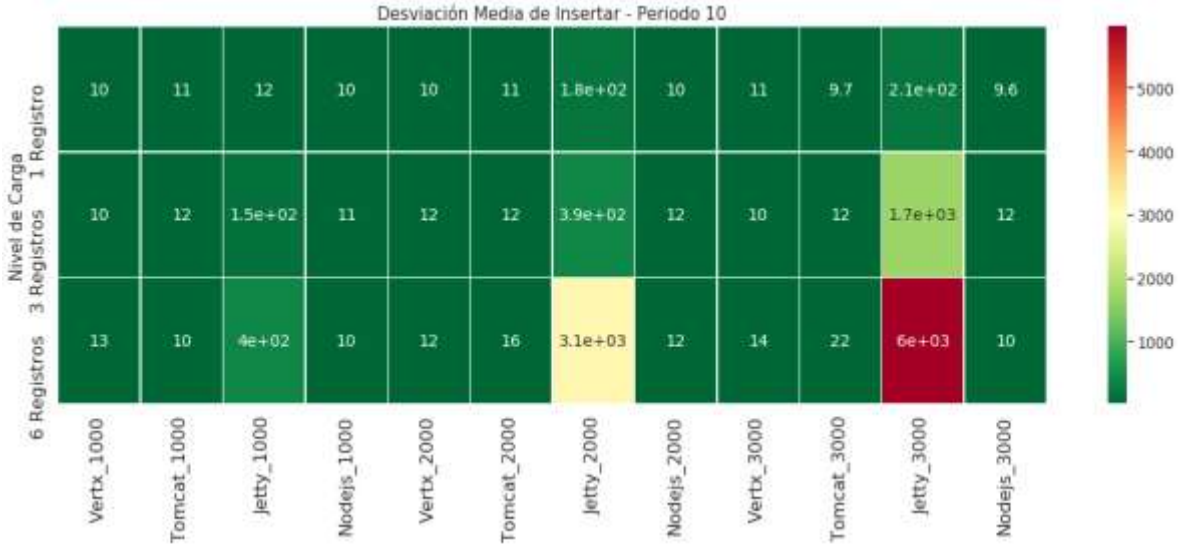


Figura 63. Desviación estándar para “Consultar” con Periodo 6.

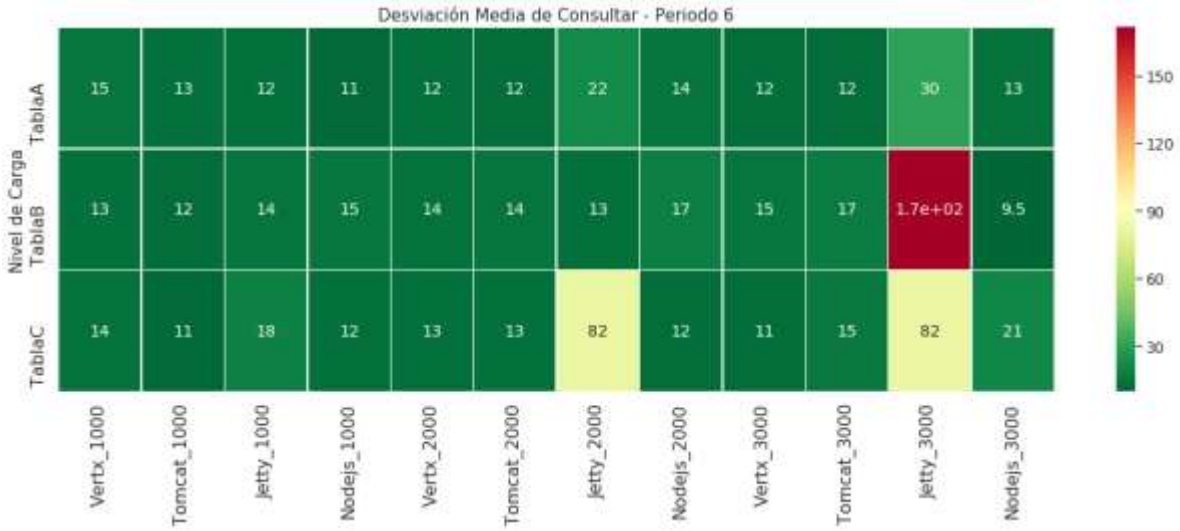


Figura 64. Desviación estándar para “Consultar” con Periodo 8.

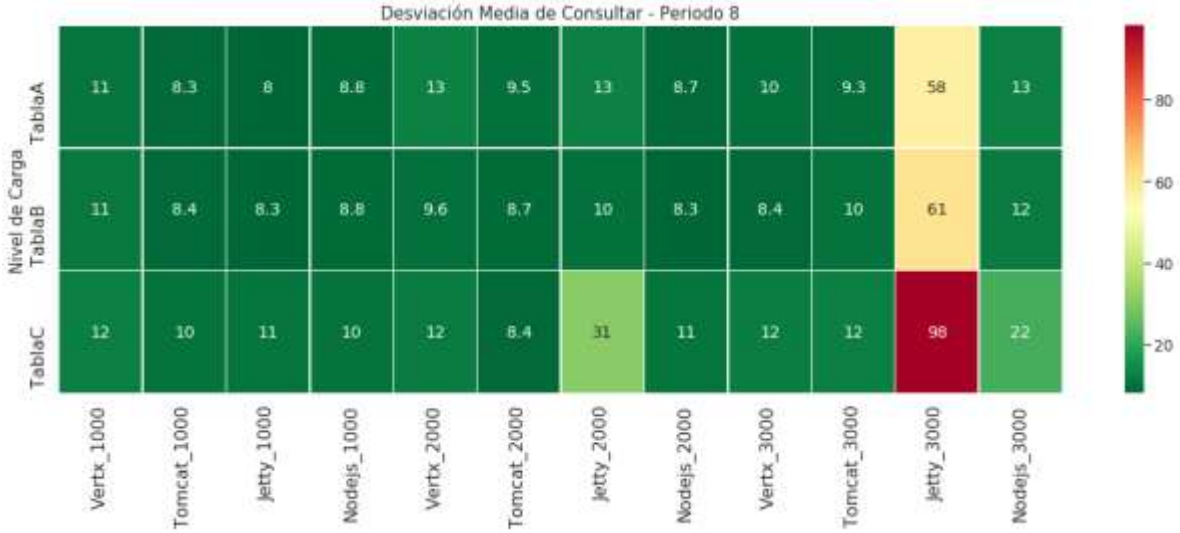


Figura 65. Desviación estándar para “Consultar” con Periodo 10.

Figura 66. Desviación estándar para “Contar Primos” con Periodo 6.

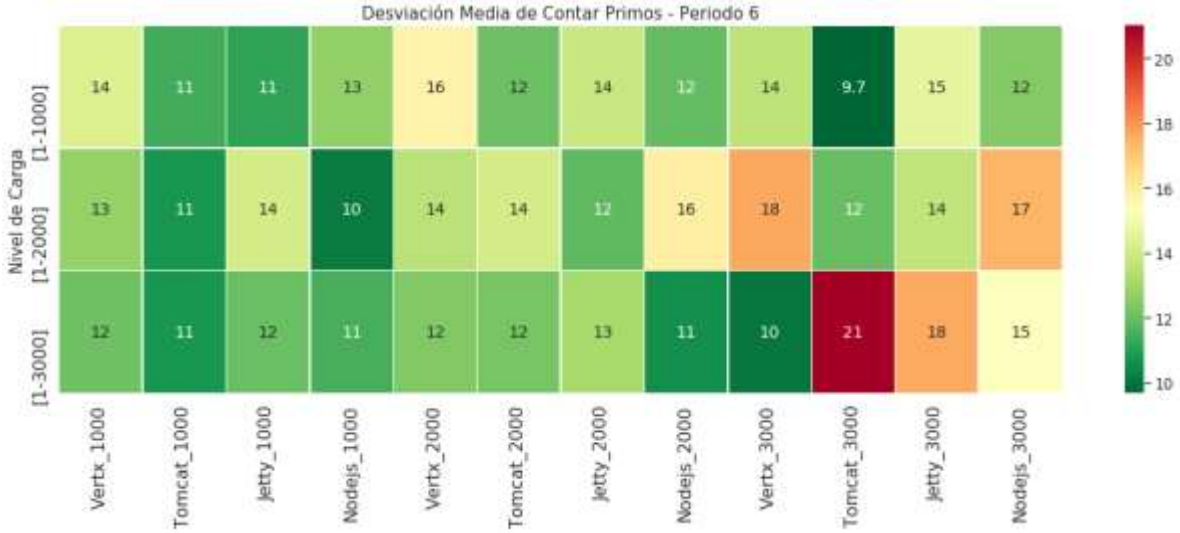


Figura 67. Desviación estándar para “Contar Primos” con Periodo 8.

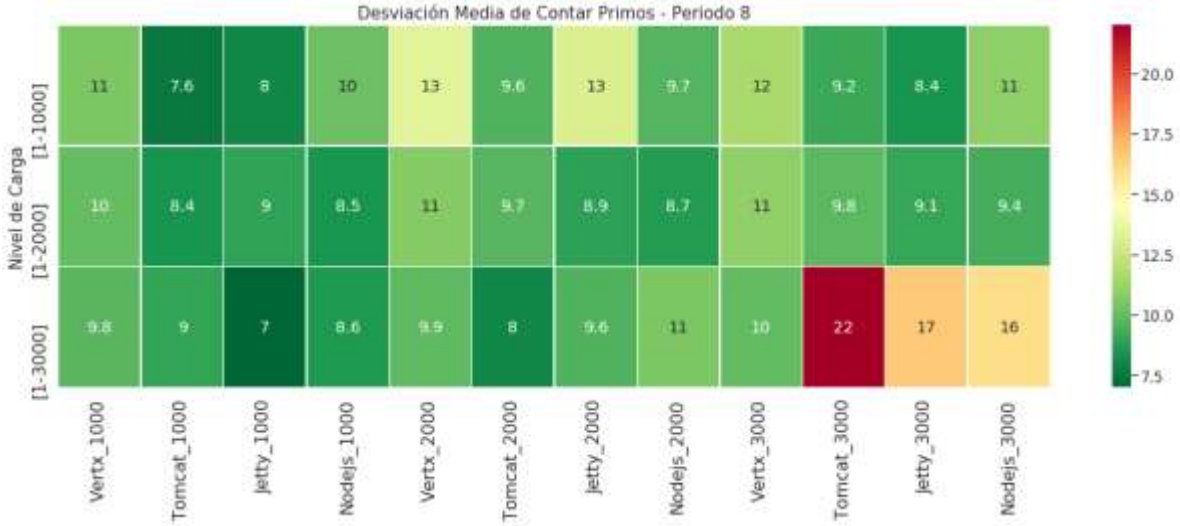


Figura 68. Desviación estándar para “Contar Primos” con Periodo 10.