ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

MEJORA A SERVIDOR WEB CONCURRENTE QUE SOPORTA APLICACIONES WEB JAVA Y MODULARIZACIÓN

PRESENTADO POR:

NICOLÁS OSORIO ARIAS

30 DE SEPTIEMBRE DE 2018

BOGOTÁ D.C.

INTRODUCCIÓN

Desde los principios de las civilizaciones, la distribución de tareas de forma correcta para el desarrollo de la sociedad ha sido un área que ha sido determinante, con en el tiempo, nos damos cuenta de que las civilizaciones evolucionaron de forma distinta, con objetivos, culturas, características y velocidades variadas. Cuando analizamos las grandes civilizaciones victoriosas, nos damos cuenta de que han sabido distribuir tanto sus recursos como a sus habitantes con el fin de conseguir los objetivos del momento. Esto, aunque puede sonar sencillo, en nuestros días los grandes cargos y trabajos más importantes de una empresa se basan en la estrategia que tienen para administrar los recursos y empleados para conseguir su objetivo común por medio proyectos que se formulan y desarrollan en el tiempo.

El verdadero problema se puede observar cuando nos damos cuenta que agregar personas a una determinada tarea no siempre tiene un efecto positivo en la productividad, así como reducir el personal tampoco ayudará siempre a que una empresa sea más productiva, todo se debe adaptar a la tarea y al verdadero objetivo o meta que se tenga. De igual manera nos damos cuenta que otro gran problema que nace con esto es la manera en la que cada componente de la empresa interactúa con los otros, generando grandes problemas como sobrecargas de trabajo para algún componente de la empresa, o grandes periodos de ocio en algunas partes de los procesos por demoras en otras zonas.

Cuando la sociedad empezó a crear los primeros ordenadores, pensar en la ejecución de instrucciones de forma secuencial resultaba ser todo un logro, sin embargo gracias a las bajas velocidades y a la necesidad de ejecutar varias acciones paralelas es que nacen los procesadores con más de un núcleo, abriendo así un nuevo panorama para la solución de los problemas.

Finalmente, con la llegada de Internet no solo se permitía unir a la sociedad en una sola red, sino también olvidar las distancias que existen entre las diferentes máquinas. Pero como siempre, un nuevo adelanto trae nuevos problemas, y así como las civilizaciones, empresas o personas, buscan organizar y administrar sus recursos de la mejor forma, los nuevos sistemas tienen que tener características que se adapten a sus objetivos.

Inicialmente esto no supone un problema, pero cuando la demanda de usuarios aumenta superando las expectativas, es que nos damos cuenta que los recursos de procesamiento y memoria son limitados; es por esta razón que los sistemas modernos no dependen de un equipo para atender a los múltiples usuarios que tienen, sino que con el paso del tiempo se han logrado crear grandes sistemas distribuidos que permiten asignar diversos papeles y trabajos a ciertos componentes del sistema, distribuyendo el computo de sus operaciones e inclusive el almacenamiento.

Es gracias a estos adelantos tecnológicos que el mismo problema que tuvieron las primeras civilizaciones vuelve a entrar a nuestros días. Tenemos que distribuir las diferentes funciones y recursos de un sistema para cumplir con los objetivos de éste, sin embargo hemos desarrollado nuevas y modernas herramientas y soluciones que pueden ayudar a facilitar la solución a este problema.

El siguiente ejercicio muestra diversas mejoras aplicadas a un Servidor Web desarrollado en Java, para que pueda cumplir con su finalidad de atender muchas solicitudes y aplicaciones WEB Java en paralelo.

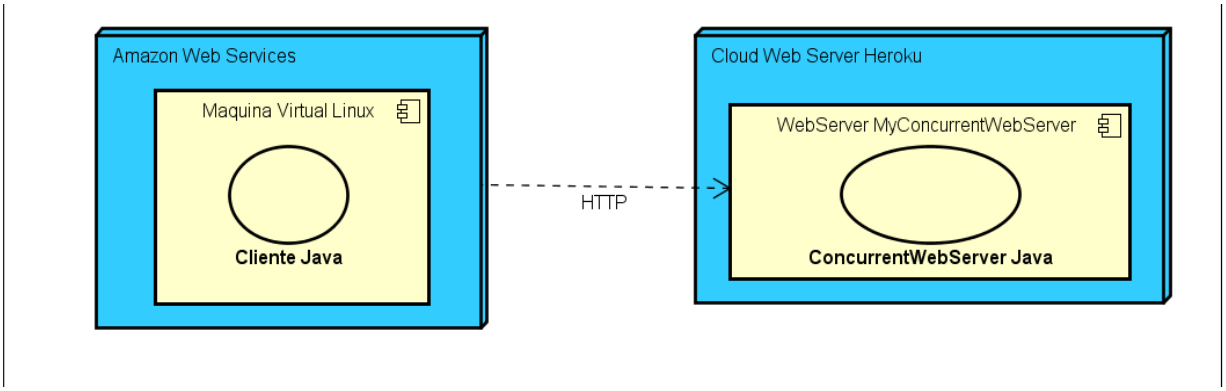
ARQUITECTURA DEL EJERCICIO

La arquitectura de este ejercicio se basa en los conceptos de modularización por medio de virtualización con las herramientas y diversas soluciones que brinda Amazon Web Services (AWS), todo esto con la finalidad de mejorar un servidor web básico hasta obtener un servidor web que soporte varias solicitudes concurrentes y aplicaciones WEB Java.

Para lograr el ejercicio propuesto es necesario obtener una máquina virtual Linux desde AWS; para esto se utiliza HTTP, se accede por medio de un navegador a Amazon Web Services, donde se procede a usar el servicio de EC2, el cual crea y ejecuta máquinas virtuales en la nube, en este caso se utilizó una maquina Linux.

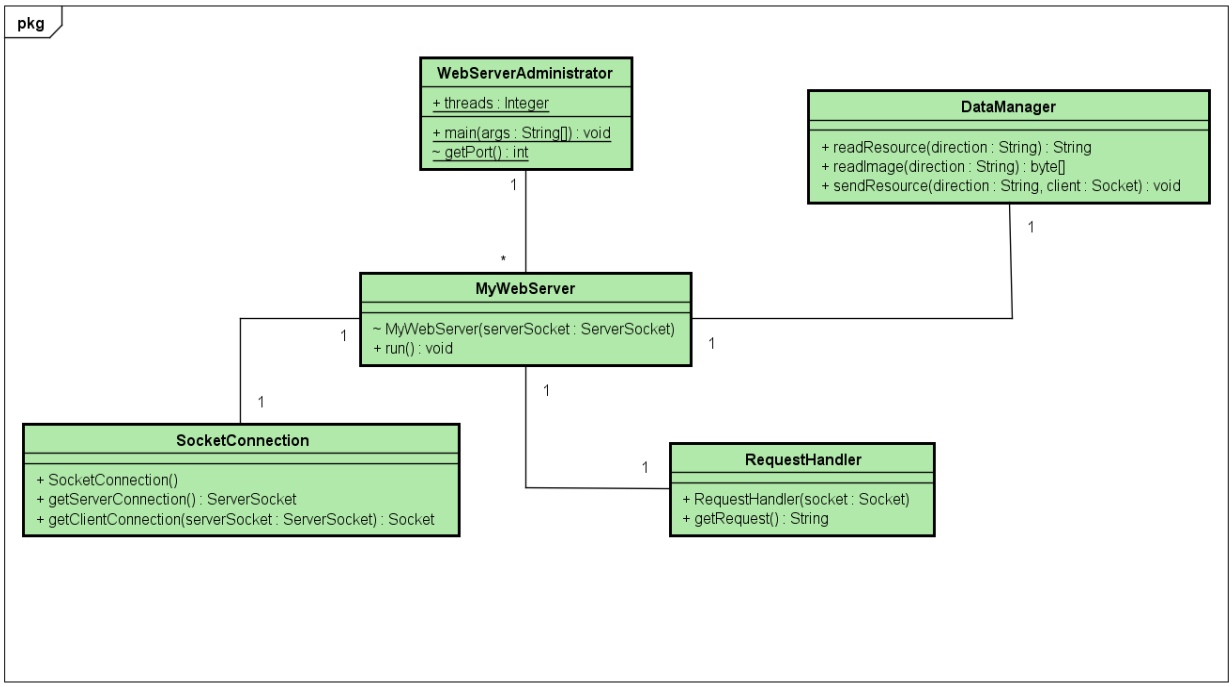
En esta máquina virtual es necesario subir un cliente Java el cual se conecta a una URL e imprime el contenido en pantalla, para lograr transferirlo a la máquina virtual, es necesario utilizar SFTP, el cual funciona sobre el protocolo SSH y nos permite conectarnos a maquinas remotamente y lograr transferir sus archivos de forma segura.

Cuando finalmente en la máquina virtual se encuentra el cliente de forma correcta, y la aplicación Java funcione de la manera esperada, es necesario conectarlo con el servidor web; previo a esto es necesario verificar que el servidor se encuentra desplegado de forma correcta en Heroku, mostrando las diversas peticiones que se hagan desde el navegador, y que se mantenga en operación permanente.

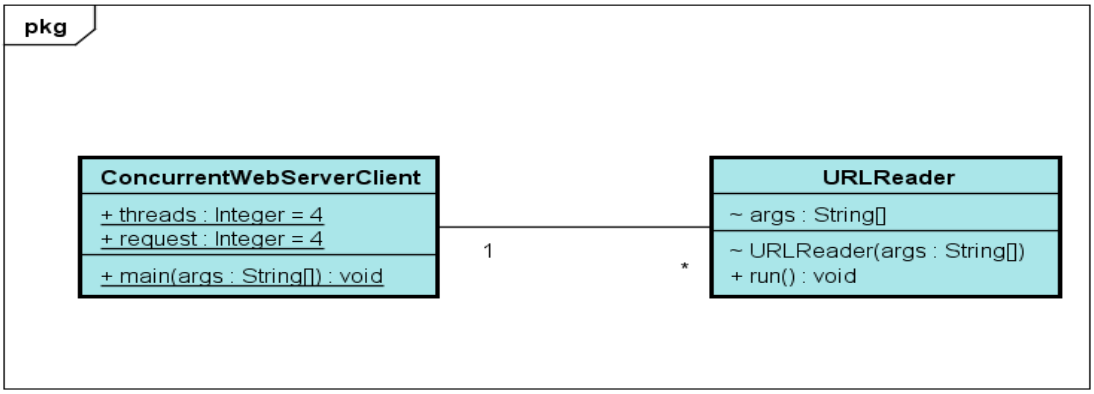


Para lograr conectar el cliente al servidor desplegado en Heroku, se usó HTTP, por medio de los argumentos que se le brindan al Cliente Java al momento de correrlo desde la máquina virtual obtenida desde AWS, con esto es posible hacer las peticiones a una URL destino, y así poder mostrar en pantalla lo que se obtiene de la URL de nuestro servidor Web desplegado en Heroku. Cuando lo anterior es probado, en el cliente se debe obtener el archivo pedido en los argumentos desde el servidor web.

Finalmente, una vez lograda la conexión se procede a ajustar el servidor web para que ahora pueda responder múltiples peticiones de manera concurrente, para esto se utilizaron Executors y ExecutorServices de la librería Java Util Concurrent, gracias a estos es posible crear un pool de Threads de longitud variable, manejando así por medio de hilos las distintas instancias y peticiones que se le hagan al servidor web y permitiendo llevar a cabo los experimentos requeridos.



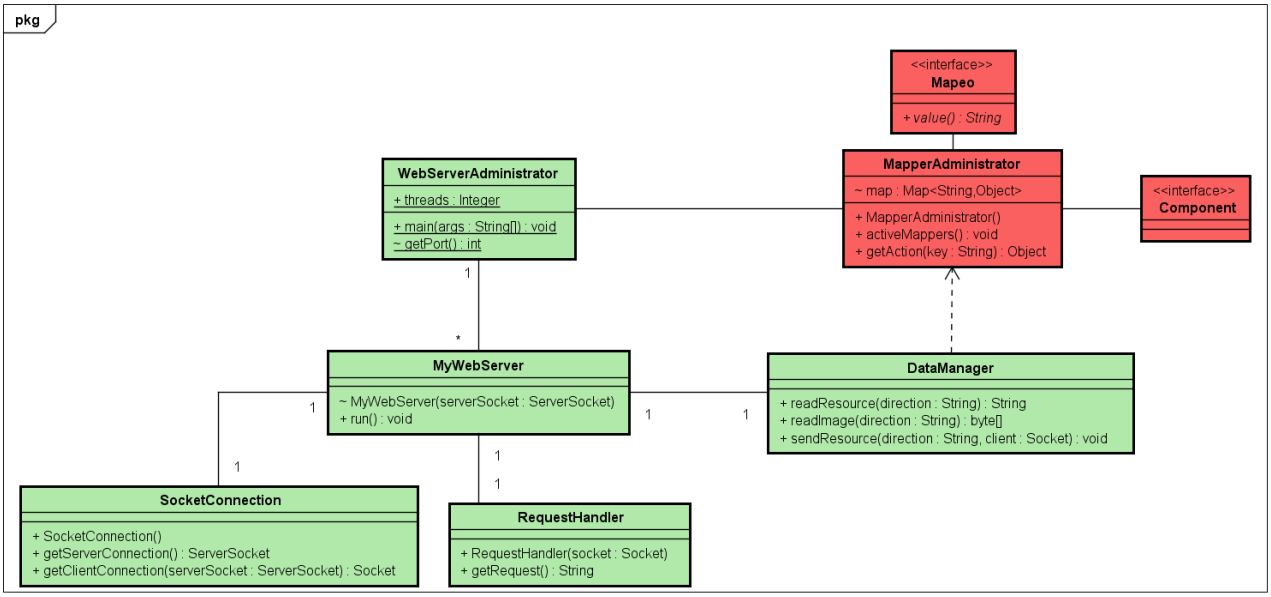
Con la finalidad de probar la concurrencia del servidor web es necesario ajustar el cliente Java para que pueda crear diversas peticiones en paralelo, esto se hace de manera muy similar a la concurrencia del Servidor, variando la forma a lo que invocan y a la forma en que terminan sus procesos.



Cuando finalmente se tiene el nuevo cliente concurrente Java en la máquina virtual, y el servidor web desplegado de forma correcta en Heroku, se procede a experimentar con el tiempo de respuesta que puede tener el servidor con respecto a los hilos que pueda manejar, para esto es necesario variar la cantidad hilos que tendrá tanto el pool de threads del cliente como del servidor.

Para finalizar el ejercicio obtener un servidor mucho más interesante, es necesario mejorarlo para que soporte aplicaciones WEB java, para esto se agrega una nueva parte a la aplicación la cual se encarga de guardar las nuevas direcciones URL a tener en cuenta por parte del servidor; esto se logra en el MapperAdministrator el cual logra almacenar las distintas rutas junto con su respectiva acción en un diccionario, pudiendo acceder a él desde el segmento de aplicación destinado al envío de peticiones de nuestro servidor, con esto lograr verificar si se encuentra mapeada en la aplicación, acompañado de ejecutar la respectiva acción necesaria.

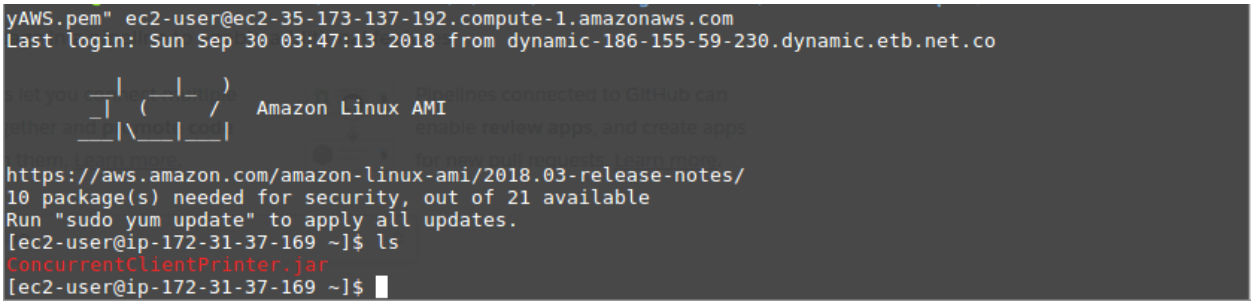
La creación de notaciones es la base de esta estructura, ya que permiten obtener los valores que ingrese el programador de la aplicación Java que se desee correr, para esto se utilizan dos interfaces para lograr reconocer los dos factores que nos interesan, el primero, si es un componente destinado para que lo ejecute el servidor web, y finalmente las diversas rutas acompañadas de las funciones que se desean vincular.



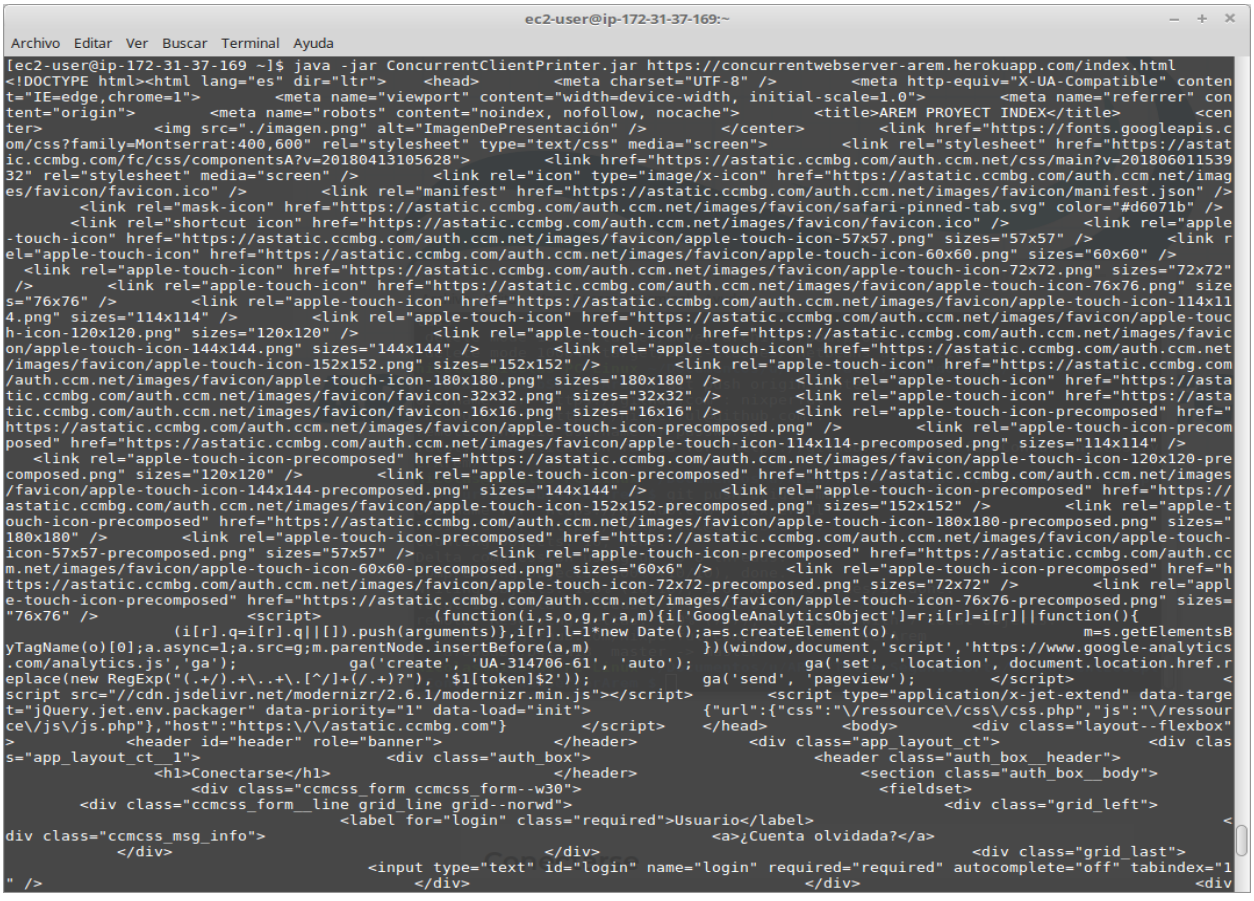
Para concluir el ejercicio es necesario efectuar una pruebas y experimentos que logren demostrar la conexión del cliente desplegado en la maquina virtual Linux de AWS con el servidor web que se encuentra desplegado en Heroku.

EXPERIMENTOS ELABORADOS:

Se procede a subir un cliente Java configurado con un (1) Hilo a una máquina virtual de Amazon Web Services en Linux llamada EC2 como se ve a continuación:

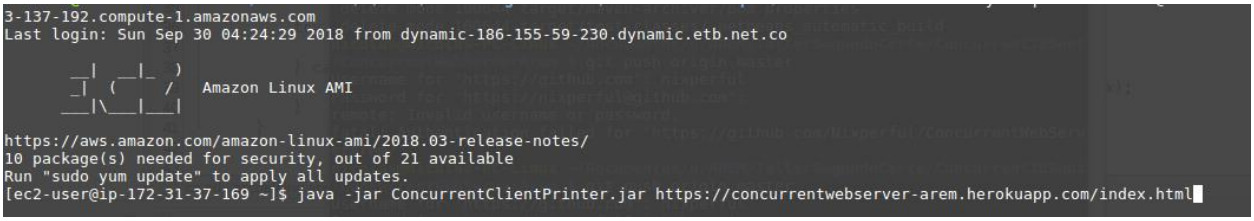


Se hace la petición al index.html de nuestro servidor desplegado en Heroku el cual tiene una configuración para aceptar un Hilo (1).



El resultado fue correcto puesto que completó la petición en 0.35 segundos. Sin embargo, es necesario experimentar con varias posibles configuraciones de hilos, tanto en el cliente como en el servidor, ya que como se pudo observar, todo depende del objetivo y el contexto de la aplicación.

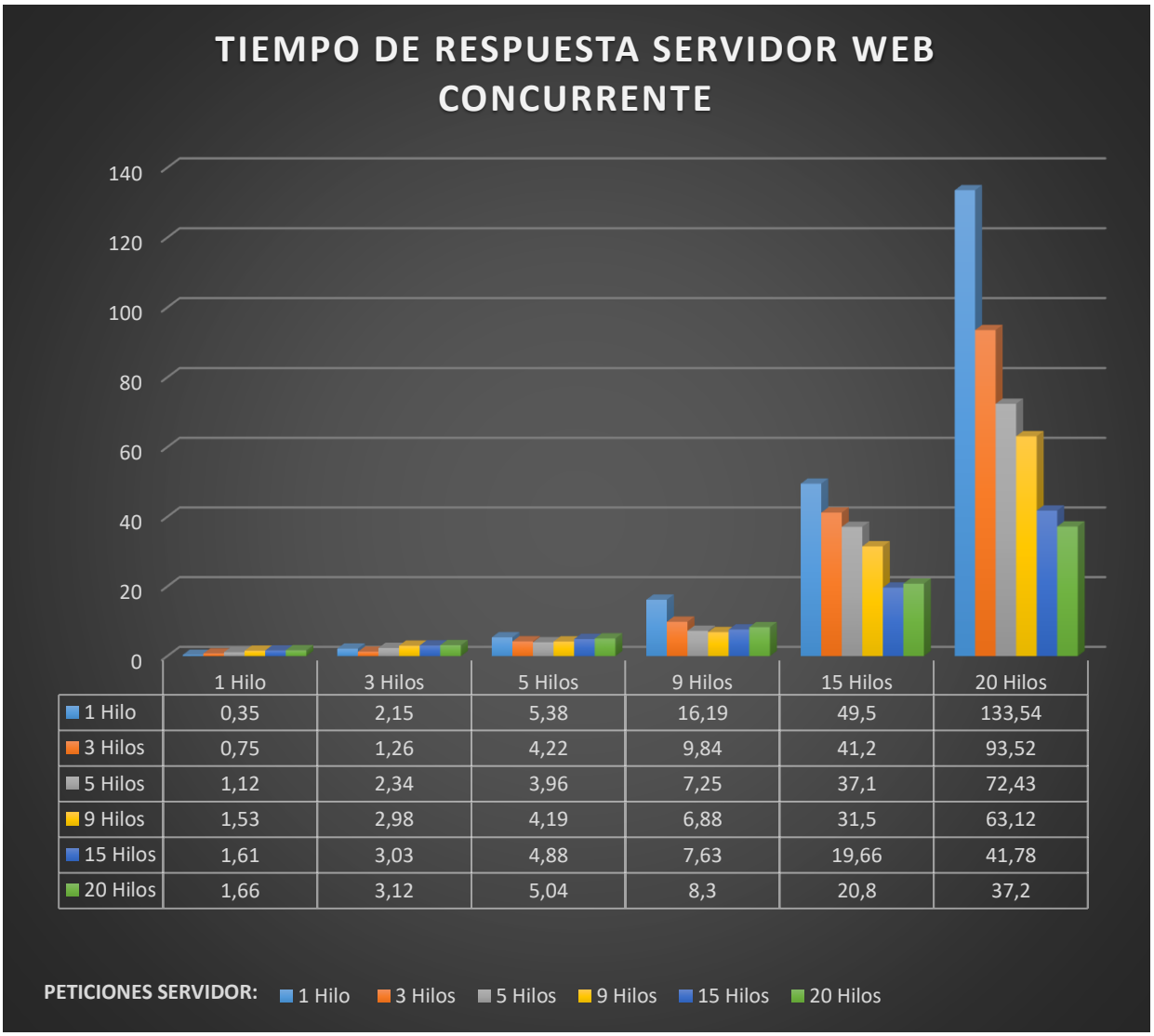
Para hacer las diversas pruebas, se procede a configurar varios clientes y a subirlos a la maquina virtual de AWS, cada uno de ellos con diversas cantidades de hilos, teniendo como máximo veinte peticiones.



De igual forma se procede a variar el número de peticiones concurrentes que soporta el servidor web, desplegando nuevamente en Heroku para poder hacer las distintas pruebas y así determinar la mejor configuración posible. Se hacen diversas pruebas incluyendo imágenes:

Finalmente, luego de varias pruebas con diversas configuraciones se pudo obtener los datos necesarios para lograr medir el desempeño del servidor, estos datos son analizados para que brinden una información más clara y concisa sobre el rendimiento final.

Con los resultados obtenidos del experimento se pudo obtener la siguiente gráfica, la cual muestra el rendimiento que tuvo el servidor respondiendo peticiones concurrentes variando el número de peticiones concurrentes que podría responder:



	1 Hilo	3 Hilos	5 Hilos	9 Hilos	15 Hilos	20 Hilos
1 Hilo	0,35	2,15	5,38	16,19	49,5	133,54
3 Hilos	0,75	1,26	4,22	9,84	41,2	93,52
5 Hilos	1,12	2,34	3,96	7,25	37,1	72,43
9 Hilos	1,53	2,98	4,19	6,88	31,5	63,12
15 Hilos	1,61	3,03	4,88	7,63	19,66	41,78
20 Hilos	1,66	3,12	5,04	8,3	20,8	37,2

CONCLUSIONES:

- Después de completar de manera exitosa las pruebas, se puede concluir que el servidor web desplegado en Heroku soporta múltiples peticiones concurrentes de manera correcta, así como también el cliente el cual se encuentra en una maquina virtual de AWS se conecta satisfactoriamente y hace las peticiones concurrentes de la manera en la cual se necesitan.
- La configuración de Threads arroja resultados muy variados que ayudan a determinar algunos patrones de desempeño del servidor web; se puede evidenciar que cuando se hace un número elevado de peticiones concurrentes al servidor, el tiempo de respuesta de éste se puede reducir significativamente, sin embargo, cuando el número de peticiones que soporta es mayor a las peticiones concurrentes que envía el cliente, el tiempo de respuesta aumenta cada vez que la diferencia entre estos aumenta.
- Se logra determinar que la mejor configuración para el servidor web concurrente depende de la cantidad de peticiones concurrentes que este pueda tener y el distinto uso que se le quiera brindar, en este caso, con 3 hilos serán suficientes.
- Se logró el ejercicio de una manera acertada, se completaron los experimentos de manera correcta, logrando crear y manejar soluciones usando Amazon Web Services, además de lograr adaptar un servidor básico a uno concurrente que soporta múltiples peticiones en paralelo.

TRABAJO FUTURO:

- Para trabajos futuros sería adecuado mejorar el cliente para que se pueda probar con distintos hilos desde los argumentos que se le mandan a la aplicación.
- Incorporar nuevas aplicaciones web Java que se adapten a este servidor web concurrente para probar nuevos problemas que se puedan encontrar.
- Refactorizar el código para tener una mejor estructura.