John Ardila
Rogelio García
Felipe Plazas
Diana Solano

Experimento 2

Grupo St. ValenTeam

Arquitectura

A continuación se presenta una lista de las IPs de las máquinas virtuales en la que están corriendo las 6 instancias del servidor.

- 35.157.224.43:9000
- 35.156.78.146:9000
- 35.157.57.115:9000
- 35.157.240.165:9000
- 35.158.29.108:9000
- 35.157.76.14:9000

Los patrones incluídos a la arquitectura para satisfacer el atributo de calidad de disponibilidad fueron los siguientes:

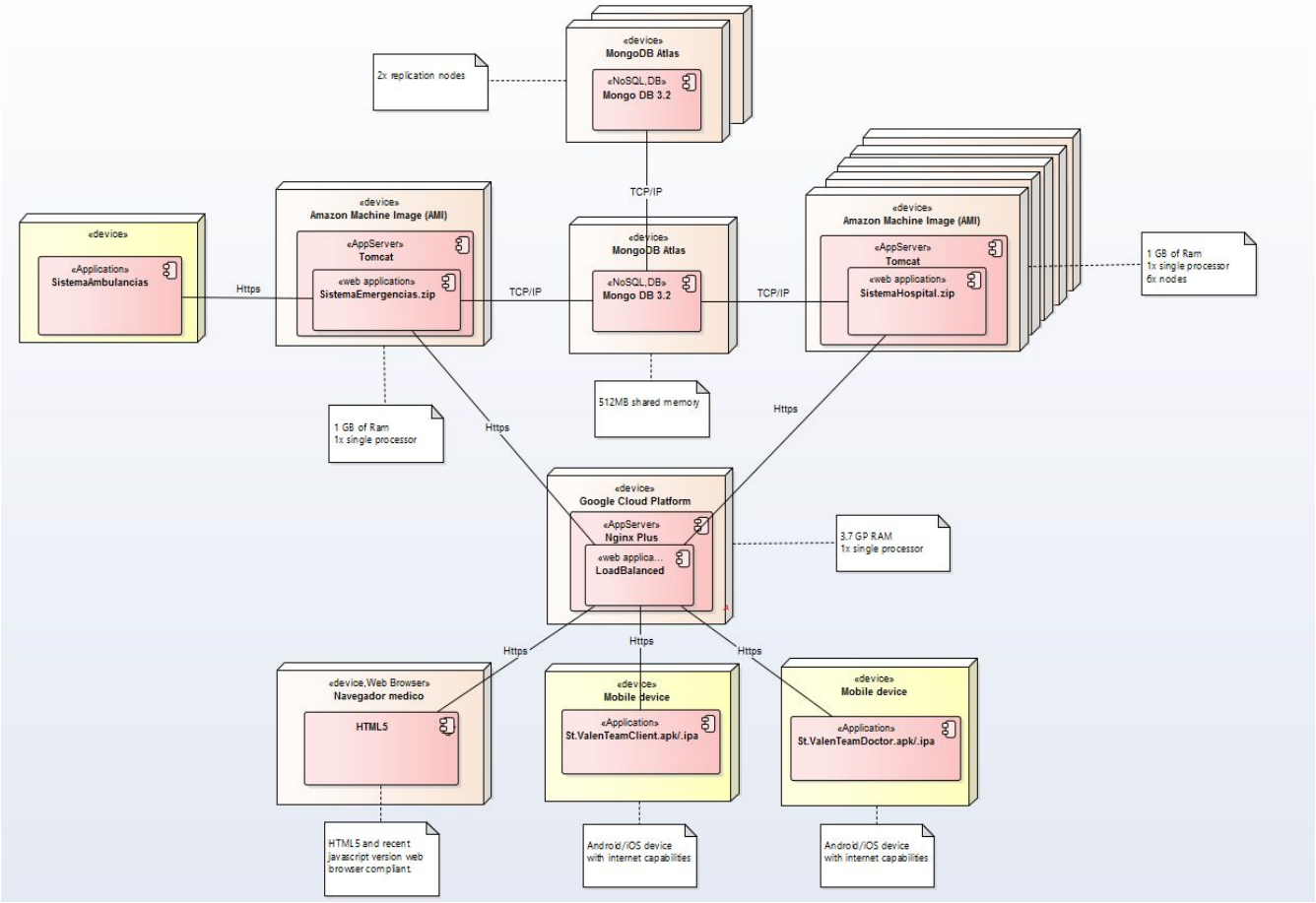
- Se realizaron múltiples copias de computación
- Se introdujeron múltiples copias de datos

Adicionalmente, se separó la aplicación en dos partes; una máquina dedicada exclusivamente a la atención de emergencias y las otras cinco empleadas para atender las demás solicitudes y casos de uso. Esto tiene como objetivo satisfacer el escenario de calidad de poder atender las emergencias en un bajo tiempo, incluso cuando el servidor esté sobrecargado.

Para lograr separar el funcionamiento de la aplicación y poder introducir múltiples copias de computación, se implementó un balanceador de carga por medio de Nginx. Dado que todos los componentes de la aplicación están en la nube y para facilitar las pruebas mediante herramientas que no dependieran de nuestros recursos físicos computacionales, se decidió implementar nuestro servidor de balanceador de carga en la nube haciendo uso de los servicios de Google Cloud Platform.

John Ardila
Rogelio García
Felipe Plazas
Diana Solano

A continuación se presenta el **diagrama de despliegue**:



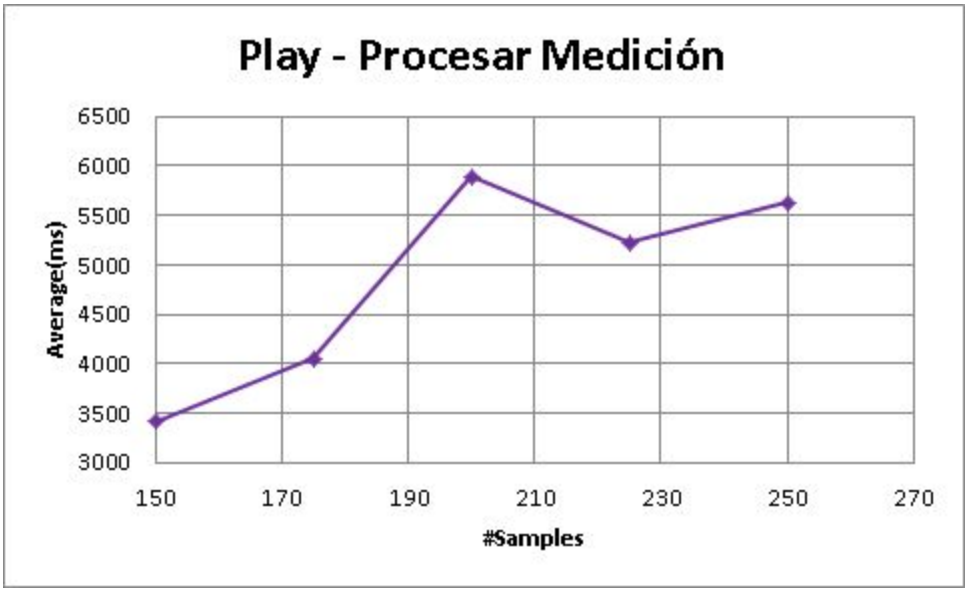
Pruebas de carga

Se volvieron a ejecutar las pruebas de carga propuestas en el experimento 1 teniendo como resultado una degradación general en el comportamiento del sistema. A continuación se pueden ver los resultados y una clara de explicación del deterioro en los resultados:

John Ardila
Rogelio García
Felipe Plazas
Diana Solano

- **Peticion tipo POST:**

Play: procesar Medición - Experimento No. 2							
#Samples	Average(ms)	Min(ms)	Max(ms)	Error (%)	Throughput	Received	Sent
150	3416	772	40003	0,0	2,5	230,18KB	714,50KB
175	4051	704	30204	0,0	2,9	263,24KB	810,37KB
200	5893	844	39689	0,0	3,3	352,01KB	965,81KB
225	5222	763	40001	0,3	3,8	335,66KB	958,93KB
250	5624	720	40003	0,4	4,2	367,28KB	1,05MB



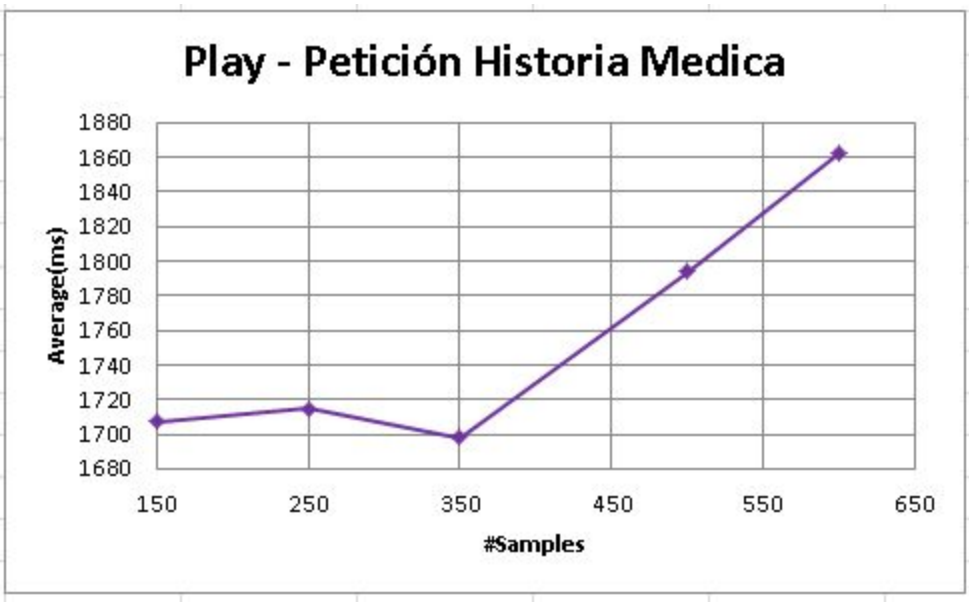
Analisis:

Como se puede observar en la gráfica anterior, el tiempo promedio aumentó a 5.200 milisegundos aproximadamente. Además se evidencia que solo es posible garantizar el procesamiento de 200 solicitudes concurrentes en 1 segundo sin presentar error. Lo anterior se debe a las especificaciones de las nuevas máquinas utilizadas ya que presentan un menor rendimiento que las proporcionadas por Heroku.

John Ardila
Rogelio García
Felipe Plazas
Diana Solano

- **Petición tipo GET:**

Play: petición Historia Medica - Experimento No. 2							
#Samples	Average(ms)	Min(ms)	Max(ms)	Error (%)	Throughput	Received	Sent
150	1707	27	10031	0,0	2,5	1,00MB	1,00MB
250	1715	38	10031	0,0	4,2	1,68MB	1,68MB
350	1698	28	17041	0,0	5,8	3,26MB	2,53MB
500	1794	28	17043	0,0	8,3	4,06MB	2,96MB
600	1862	28	17044	1,1	10,0	4,07MB	2,99MB



Analisis:

Con respecto a la petición de historia médica se evidencia que es posible procesar hasta 500 peticiones concurrentes por segundo. Se comienza a presentar error a partir de 600 solicitudes por segundo. A partir de lo anterior se evidencia que la aplicación sigue soportando las 3000 solicitudes por minuto requeridas pero el desempeño se deterioró significativamente.

John Ardila
Rogelio García
Felipe Plazas
Diana Solano

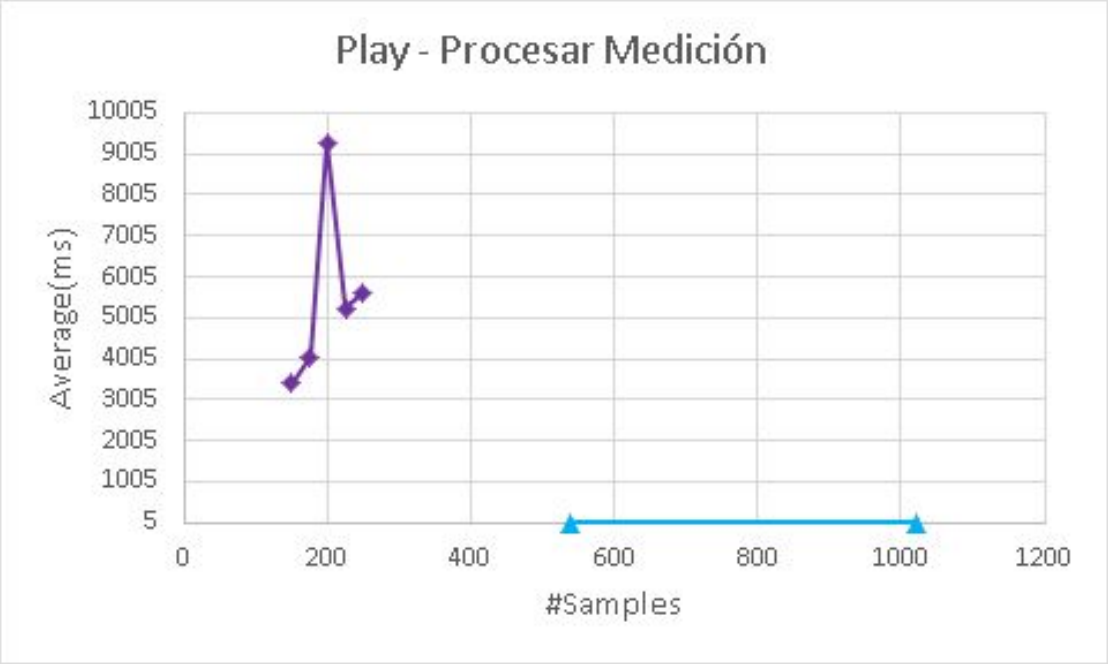
Comparación pruebas de carga

A continuación se presenta la comparación de las pruebas de carga entre el Experimento 1 y la Entrega 1 del Experimento 2, al final se expone una análisis sobre los resultados.

- **Petición tipo POST: Procesar Medición**

Experimento 1		
#Samples	Average(ms)	Error (%)
540	6	0,0
1020	7	0,0
10020	7	0,0
20038	9	0,0
40019	28	0,0
55588	326	0,4

Experimento 2		
#Samples	Average(ms)	Error (%)
150	3416	0,0
175	4051	0,0
200	9271	0,0
225	5222	0,3
250	5624	0,4
---	---	---

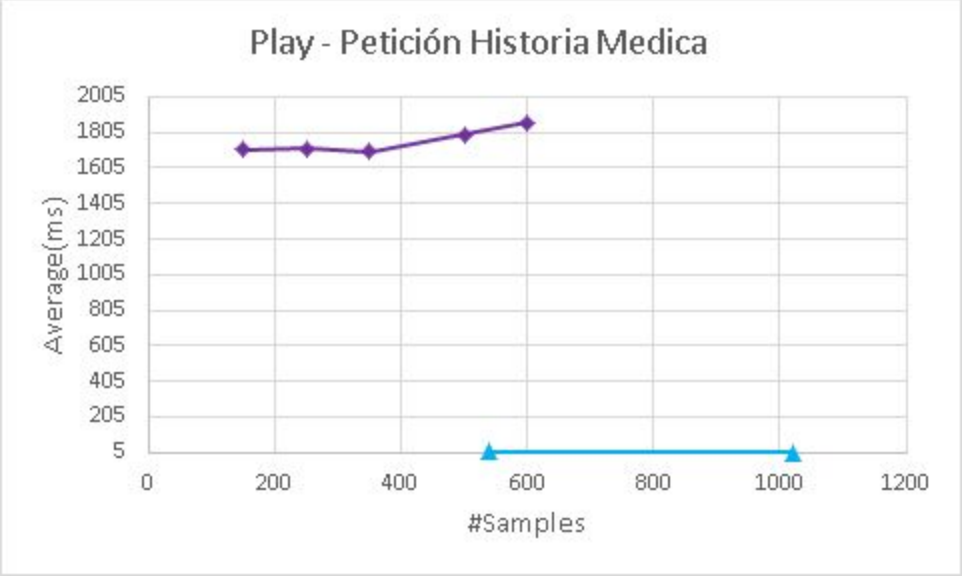


John Ardila
Rogelio García
Felipe Plazas
Diana Solano

- **Peticion tipo GET: Peticion Historia Medica**

Experimento 1		
#Samples	Average(ms)	Error (%)
540	7	0,0
1020	6	0,0
10020	6	0,0
20038	17	0,0
40019	10	0,0
55588	8	0,0
90000	25	0,0
120000	189	2,6

Experimento 2		
#Samples	Average(ms)	Error (%)
150	1707	0,0
250	1715	0,0
350	1698	0,0
500	1794	0,0
600	1862	1,1
---	---	---
---	---	---
---	---	---



Análisis:
Las gráficas anteriores sólo incluyen las dos primeras muestras de los resultados del Experimento 1, esto se debe a la significativa diferencia de usuarios simultáneos que soporta la aplicación con los nuevos ajustes (atributo disponibilidad) frente a la aplicación de la entrega anterior. Dicha diferencia es de 2 órdenes de magnitud en la petición POST y de 3 ordenes en la petición GET, ambos a favor de la aplicación del Experimento 1. De colocar la totalidad de datos el gráfico se vuelve confuso y difícil de interpretar.

Es evidente que los cambios propuestos para satisfacer el atributo de disponibilidad han afectado de manera importante el desempeño de la aplicación, esto de debe a las siguientes razones:

- **Primero:** para configurar el balanceador de carga Nginx, dispuesto en la nube, fue necesario trasladar la aplicación a un nuevo servicio de máquinas virtuales gratuitas, también dispuestas en la nube, que no solo proporcionarán el dominio de la aplicación sino también una ip. El traslado fue hacia máquinas virtuales gratuitas AWS (Amazon Web Services), que al compararlas con la máquina proporcionada por la plataforma en la nube Heroku (servicio usado en el Experimento 1) tienen una menor poder computacional, lo que significa una menor capacidad para atender peticiones.
- **Segundo:** el servicio Nginx acondicionado en la nube para trabajar sobre nodos igualmente dispuestos en la nube, en su versión gratuita, tiene una baja memoria RAM y una baja capacidad de procesamiento. Estas caracteristicas limitan drásticamente la concurrencia de eventos que puede manejar.

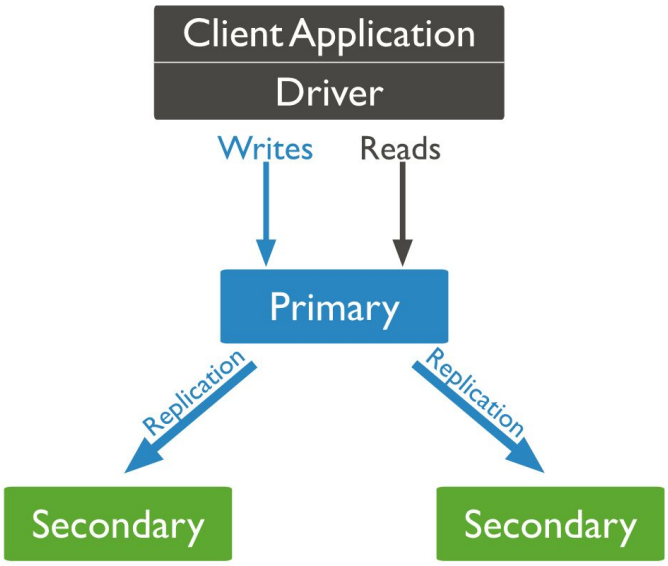
Aunque el error para la petición tipo POST pasó de 0.4% para 55588 peticiones en un segundo a 0.4% para 250 y para la petición tipo GET pasó de 2.6% para 120000 peticiones en un segundo a 1.1% para 600; no se puede concluir que la realización de un balanceador de carga con 5 nodos para la escucha de peticiones no es conveniente para la aplicación. Para esto no solo es necesario realizar pruebas sobre máquinas virtuales de mejores características, sino además tener en cuenta que disponer de múltiples copias computacionales permite que en caso de la caída de una de ellas las demás puedan seguir recibiendo peticiones sin perder información crucial para el negocio.

Replicación de datos

Dado que el proyecto y todas sus componentes han sido desplegadas en la nube, surge la preocupación de una eventual indisponibilidad en los servidores del proveedor de la base de datos. Por esta razón, hemos decidido cambiar la arquitectura actual de datos que estaba basada en un único nodo de Mongo que corría en los servidores de mLab.

La nueva arquitectura hace uso del concepto de replicación, con el fin de proveer al sistema redundancia y una alta disponibilidad de datos. La replicación en Mongo se logra a través de la implementación de un *replica set*, que se conforma de un grupo de instancias de Mongo que mantienen la misma información. Este conjunto de replicación contiene dos nodos secundarios y un nodo principal; este último es el encargado de recibir las operaciones de escritura, ejecutar la operación y posteriormente replicar la acción en los 2 nodos secundarios.

John Ardila
Rogelio García
Felipe Plazas
Diana Solano



Si por alguna razón el nodo principal deja de estar disponible, un nodo secundario seleccionado al azar pasa a tomar el lugar del nodo primario. Esto se logra gracias a que todos los nodos mantienen un chequeo constante de los otros mediante heartbeats.