

BALANCEADOR DE CARGA USANDO EL MÓDULO DE APACHE

MOD_PROXY_BALANCER

Diana Marcela García López
diana_ma.garcia@uao.edu.co

Iván Darío Duque Correa
ivan.duque@uao.edu.co

David Sanchez Collazos
David.sanchez_col@uao.edu.co

1. RESUMEN

Este proyecto es un balanceador de carga para servidores web, el cual distribuyen las peticiones hechas por un usuario a los diferentes servidores disponibles, estas peticiones deben ser transparentes al usuario y en tiempo real.

1.1. PALABRAS CLAVE

Balanceo de carga, servicio web.

2. ABSTRACT

This project is a load balancer for web servers, which distribute the requests made by a user to the different available servers, these requests must be transparent to the user and in real time.

2.1. KEY WORDS

Load balancing, web service.

3. INTRODUCCIÓN

Actualmente, los servicios web se encuentran en auge más que nunca, la cantidad de usuarios que solicitan estos servicios es mayor, y se requiere que todos puedan acceder a él en el momento oportuno.

En este proyecto se implementó un balanceador de carga en una máquina local usando GNU/Linux en la distribución CentOS; esto se logró gracias al mod proxy de Apache, y esta configuración la probamos con el sistema *Artillery*.

4. OBJETIVOS

4.1. OBJETIVO GENERAL

Implementar un *Cluster* de servidores web con balanceo de carga; este funciona como *front-end* y las peticiones se realizan a los servidores de *back-end*

4.2. OBJETIVO ESPECÍFICO

- Realizar la configuración de cada uno de los servidores creados por medio de máquinas virtuales.
- Probar la ejecución de cada una de las máquinas y que la configuración del balanceador de carga este correcta.
- Ejecutar pruebas de rendimiento y de peticiones por medio de *Artillery*.

5. ALTERNATIVAS DE SOLUCIÓN

En el mercado existen diferentes alternativas de solución para la implementación de un balanceador de carga, comercialmente hay varias herramientas que permiten realizar esta implementación y diferentes empresas utilizan diferentes proveedores según sus necesidades.

5.1. SEESAW

Está desarrollado en el lenguaje *Go* y funciona bien en la distribución Ubuntu/Debian. Admite *anycast* y DSR (retorno directo del servidor) y requiere dos nodos *Seesaw*. Pueden ser físicos o virtuales. [1]

5.2. HAPROXY

Uno de los más populares en el mercado es proporcionar alta disponibilidad, proxy, equilibrio de carga TCP/HTTP. *HAProxy* es utilizado por algunas de las marcas de renombre en el mundo. [1]

5.3. NEUTRINO

Neutrino es utilizado por eBay y construido con *Scala* & *Netty*. Admite algoritmos de mínima conexión y *round-robin*. [1]

5.4. NGINX

Nginx Plus es una solución de entrega de aplicaciones web todo en uno incluido el balanceo de carga, almacenamiento en caché de contenido, servidor web, WAF, monitoreo, etc. Proporciona una solución de

balanceador de carga de alto rendimiento para escalar aplicaciones para atender millones de solicitudes por segundo. [1]

5.5. POR QUE APACHE_MOD

Esta solución se utilizó por encima de las otras principalmente por su facilidad de implementación en las máquinas CentOS, su documentación, código libre y en general las ventajas para la práctica académica, su algoritmo de implementación ya integrado es muy útil y permite que se balancee la carga agregando simplemente unas líneas de código en el archivo de configuración del servidor.

6. MARCO TEORICO

6.1. BALANCEADOR DE CARGA

Un balanceador de carga permite maximizar la tasa de datos y, al mismo tiempo, reducir la carga sobre el servidor. Los balanceadores de carga son capaces de evaluar la carga, así como los tiempos de respuesta de distintos servidores y distribuir el tráfico entre varios servidores con ayuda de reglas. El uso de balanceadores de carga es práctico cuando sus aplicaciones web son puestas a disposición por varios servidores y es necesario garantizar una elevada disponibilidad. Este es el caso, por ejemplo, de páginas web con muchas visitas. [2]

Una manera gráfica de ver el proceso de balanceo de carga aplicada en este proyecto es un balanceo de carga por medio de *software* y *hardware*, el cual por medio de un algoritmo balancea las peticiones hechas por el usuario a uno de los servidores disponibles, el proceso se puede observar en la Ilustración 1.

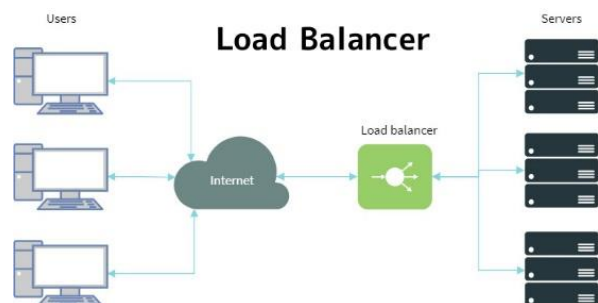


Ilustración 1. Diagrama Balanceador de carga [12]

El balanceador redirecciona las peticiones a cada uno de los servidores disponibles, balanceando el tráfico en la página, esta acción es transparente al usuario y permite mejores tiempos de respuesta del servicio y una mejor optimización del sitio web al momento de realizar peticiones.

6.2. APACHE MOD_PROXY_BALANCER

Es la modificación proporcionada por Apache para la configuración del balanceador de carga, esta modificación permite una configuración más sencilla y no empezarla totalmente desde cero; Proporciona las funciones utilizadas para enviar solicitudes HTTP y HTTPS. *mod_proxy_http* admite HTTP/0.9, HTTP/1.0 y HTTP/1.1. No proporciona ninguna capacidad de almacenamiento en caché. [3]

6.3. APACHE

El proyecto de servidor Apache HTTP es un esfuerzo por desarrollar y mantener un servidor HTTP de código abierto para los sistemas operativos modernos, incluidos UNIX y Windows. El objetivo de este proyecto es proporcionar un servidor seguro, eficiente y extensible que brinde servicios HTTP en sincronía con los estándares HTTP actuales. [4]

6.4. SERVIDOR WEB

Con "Servidor web" podemos referirnos a hardware o software, o a ambos trabajando juntos.

En cuanto a hardware, un servidor web es una computadora que almacena el software de servidor web, y los archivos que componen un sitio web (por ejemplo, documentos HTML, imágenes, hojas de estilos CSS y archivos JavaScript). Un servidor web -hardware- se conecta a internet y mantiene el intercambio de datos con otros dispositivos conectados a la web.

En cuanto a software, un servidor web tiene muchas partes que controlan cómo los usuarios de la web obtienen acceso a los archivos alojados en el servidor; es decir, mínimamente, un servidor HTTP. Un servidor HTTP es una pieza de software capaz de comprender URLs (direcciones web) y HTTP (el protocolo que tu navegador usa para obtener las páginas web). Un servidor HTTP puede ser accedido a través de los nombres de dominio de los sitios web que aloja, y entrega el contenido de esos sitios web alojados al dispositivo del usuario final. [5]

6.5. PROTOCOLO HTTP

HTTP, de sus siglas en inglés: "*Hypertext Transfer Protocol*", es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos, como pueden ser documentos HTML. Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor, esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (el cliente), normalmente un navegador Web. [6]

7. HERRAMIENTAS UTILIZADAS

En la práctica realizada se necesitaron diferentes herramientas para culminar de manera satisfactoria la propuesta de proyecto. Cada una de las mencionadas a continuación fue relevante y útil.

7.1. COMPUTADOR PERSONAL

Esta es la principal herramienta, donde se van a ejecutar todos los procesos y se crearan las máquinas virtuales para su ejecución.

Estos tienen requerimientos de funcionamiento y los mínimos de todos los integrantes del grupo de trabajo son los siguientes:

- Procesador Intel Core I3 o equivalente.
- Memoria RAM 8Gb.
- Disco Duro 120Gb.
- Vagrant y VirtualBox instalado.

7.2. VIRTUALBOX

Oracle VM VirtualBox es una aplicación de virtualización multiplataforma [7]; este es el proveedor de virtualización para la ejecución de los entornos virtuales de las prácticas;

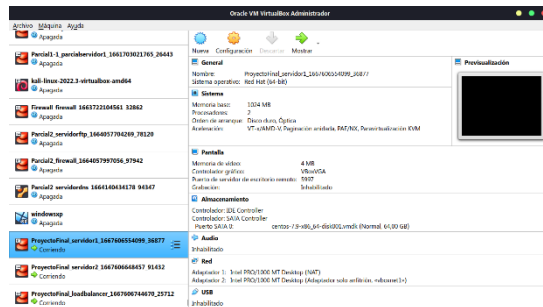


Ilustración 2. Interfaz de la implementación de VirtualBox

7.3. VAGRANT

Vagrant, una herramienta que nos permite crear cualquier entorno de desarrollo basado en máquinas virtuales. Ofrece una interfaz fácil de usar para crear servidores perfectamente configurados e independientes del sistema operativo del desarrollador. [8].

En la ilustración 3 está la captura de la línea de comandos con las máquinas instaladas y ejecutándose.

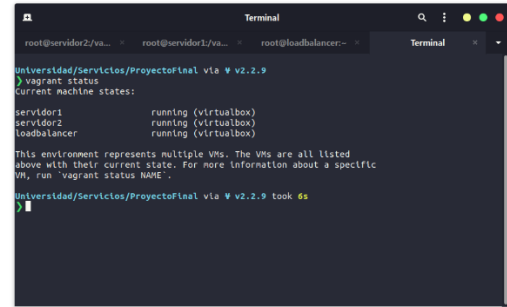


Ilustración 3. Versión y máquinas en Vagrant.

En esta práctica, se realizó el desarrollo en Vagrant en su versión 2.3.2 y 2.2.9.

7.4. CENTOS (SISTEMA OPERATIVO)

CentOS (*Community ENTERprise Operating System*) es una distribución Linux que consiste en una bifurcación a nivel binario de la distribución GNU/Linux Red Hat Enterprise Linux RHEL, compilado por voluntarios a partir del código fuente publicado por Red Hat, siendo la principal diferencia con este la eliminación de todas las referencias a las marcas y logos propiedad de Red Hat. [9]

En este proyecto se está utilizando la versión 7.9 de CentOS emulado en un entorno virtual por medio de VirtualBox y accediendo a su línea de comandos por medio de Vagrant.

Ilustración 4. Página web implementada

7.5. ARTILLERY

Artillery es un conjunto de herramientas de prueba de rendimiento moderno, potente y fácil de usar. Úsalo para enviar aplicaciones escalables que mantienen su rendimiento y resistencia bajo una carga alta. [10]

8. METODOLOGÍA

En la implementación del proyecto se procedió con la creación de tres máquinas individuales en el sistema operativo CentOS 7.9 explicado anteriormente, el esquema máquinas y de cómo se procedió al balanceo de cargas se puede ver en la Ilustración 5.

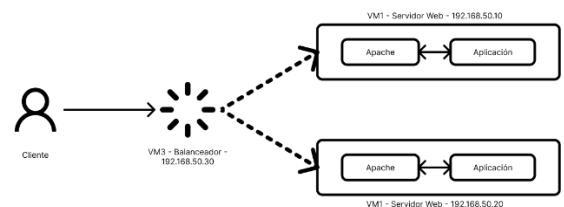


Ilustración 5. Diagrama de implementación

El servidor con el cual se ha de conectar el usuario final o en el diagrama el “Cliente” será al servidor balanceador de carga, implementado en la maquina “*loadbalancer*” con la IP 192.168.50.30, este servidor es el encargado de decidir a qué servidor realizar la petición y así mismo balancear la carga entre el clúster de máquinas disponibles.

El balanceo realizado por el servidor “*loadbalancer*” se realiza mediante el algoritmo “*byrequest*” equilibrado por medio del número de peticiones o también conocido como “Round Robin”, este algoritmo es un método para seleccionar todos los abstractos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento. [11]

Los siguientes dos servidores son los que alojan la aplicación web, el servidor1 y servidor2, para esto se dispone la aplicación en ambos servidores, teóricamente se debe poner la misma aplicación en los dos servidores, de esta forma para el usuario es indiferente a que servidor se conectó y su navegación es más fluida, sin embargo, para este ejercicio académico se implementó una página diferente en cada servidor y así poder visualizar a que servidor se conceto el cliente en ese momento, la captura de esta implementación se encuentra en la ilustración 4.



Para las pruebas de carga se realizaron con un software de pruebas de carga al servidor, el software *Artillery* se encarga de realizar peticiones al servidor para validar su rendimiento.

9. INSTALACIÓN Y CONFIGURACIÓN

Para la configuración de las maquinas utilizamos los siguientes comandos de configuración:

9.1. CONFIGURACION MAQUINA “LOADBALANCER”

Para la configuración de la maquina *Loadbalancer* aplicamos los siguientes comandos:

```
# sudo -i
# yum install vim httpd
# service httpd start
# vim /etc/httpd/conf/httpd.conf
```

Este último comando nos permite acceder a la configuración del *httpd* y agregamos las siguientes líneas de código

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module
modules/mod_proxy_http.so
```

Los últimos comandos nos permiten importar los módulos de *MOD_PROXY_BALANCER*.

```
<VirtualHost *:80>
    <Proxy balancer://clusterServicios>
        BalancerMember
            http://192.168.50.10
        BalancerMember
            http://192.168.50.20
        ProxySet lbmethod=bytraffic
    </Proxy>
    ProxyPreserveHost On
    ProxyPass "/"
        "balancer://clusterServicios/"
    ProxyPassReverse "/"
        "balancer://clusterServicios/"
</VirtualHost>
```

Esta última configuración nos permite configurar el balanceo de carga entre los servidores *back-end*, es necesario agregar un *VirtualHost*.

```
# service httpd restart
Por últimos reiniciamos el servicio.
```

9.2. CONFIGURACIÓN SERVIDOR1 Y SERVIDOR2

Para la configuración de las maquinas servidor1 y servidor2 aplicamos los siguientes comandos:

```
# sudo -i
# yum install vim httpd
# service httpd start
# vim /var/www/html/index.html
```

En esta última línea de código accedemos a la carpeta donde se encuentran los archivos a utilizar como aplicación web del servidor, y creamos el archivo *index.html*

```
# service httpd restart
Por últimos reiniciamos el servicio
```

Con estas configuraciones el balanceador de carga estaría listo y podríamos empezar a utilizarlo.

9.3. CONFIGURACIÓN ARTILLERY

Instalamos *Artillery* en la máquina del cliente con el siguiente comando

```
# npm install -g artillery@latest
```

Generamos un archivo .yaml con la siguiente configuración

```
config:
  target: "http://192.168.50.30" #
  default target
  phases:
    - arrivalRate: 25
      duration: 1
  escenarios:
    - name: Create record
      flow:
        - get:
            url: "/"
```

Luego ejecutamos el siguiente comando para iniciar las pruebas:

```
# artillery run -e staging my-
script.yaml --output
servicios.json

# artillery report servicios.json
```

El ultimo comando genera el reporte que analizaremos más adelante.

10.RESULTADOS

Por medio del software *Artillery* se realizaron las pruebas de carga al servidor, este programa nos genera unos resultados en formato *JSON* y *HTML*, con lo cual la visualización de las pruebas es más sencilla de entender y analizar.

10.1. CONFIGURACION GENERAL ARTILLERY

En la tabla 1 podremos ver los valores de prueba ingresados en el artillery, según el *arrivalRate* y el tiempo total de la prueba que se ingresaron el archivo de creación de las pruebas

Tabla 1. Configuración inicial Artillery

Configuración inicial		
Prueba	arrivalRate	Duración (s)
P1	10	50
P2	100	50
P3	100	100
P4	800	300
P5	25	1

A continuación, en la tabla 2, pondremos los resultados de las pruebas obtenidas y la configuración de estas para evaluar le rendimiento y la respuesta del balanceador.

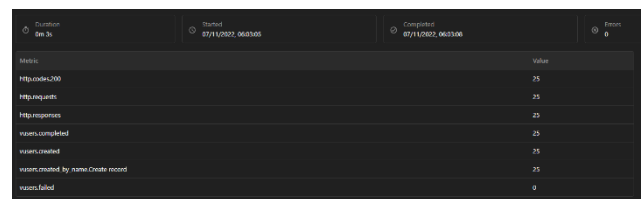
Tabla 2 Pruebas de *Artillery*

Pruebas <i>Artillery</i>		
Prueba	Peticiones	Duración
P1	500	00:00:50
P2	4670	00:01:16
P3	10000	00:01:46
P4	240020	00:05:10
P5	25	00:00:03

El programa *Artillery* nos permite generar estas pruebas variando los valores de la cantidad de *requests* y el tiempo de cada una de estas al servidor.

10.2. ESTADÍSTICAS ARTILLEY

Cada una de estas pruebas cuenta con su respectivo reporte, en este se nos indica exactamente que se intentó hacer, cuantas fueron las peticiones satisfactorias, cuantas generaron errores etcétera.



Metric	Value
http.codes.200	25
http.requests	25
http.responses	25
users.completed	25
users.created	25
users.created by name>Create record	25
users.failed	0

Ilustración 6. Estadísticas de *Artillery*

10.3. VUSERS

Estas pruebas se generan realizando usuarios virtuales o *vusers* que son los encargados de realizar las peticiones al servidor, sea solamente accediendo a la página o generando peticiones al mismo.

Tabla 3. *vusers* por prueba

Vusers por prueba			
Prueba	Creados	Completados	Error
P1	500	500	0
P2	4670	2785	1876
P3	10000	6249	3751
P4	240020	18940	221080
P5	25	25	0

Gracias a esta tabla empezamos a notar una tendencia, que a mayor sea la cantidad de *vusers* que intentan acceder al servidor mayor será la cantidad de estos que proceden a contener algún fallo en su creación.

Tabla 4. Error en los *vusers*

Vusers porcentaje de error			
Prueba	Creados	Error	Porcentaje error
P1	500	0	0%
P2	4670	1876	40.17%
P3	10000	3751	37.51%
P4	240020	221080	92.10%
P5	25	0	0%

Luego de realizados los análisis podemos observar que el balanceo es efectivo según la cantidad de usuarios que ingresen al sistema, es decir, esta acción de entrada al servidor y tiempo de petición es directamente proporcional a su efectiva respuesta, por lo tanto, es preferible que pocos usuarios se conecten mucho tiempo al servidor, a que muchos usuarios se conecten poco tiempo al mismo, esta conclusión se obtuvo principalmente comparando los tiempos de permanencia de las pruebas P1, P2, y P3, las cuales expondremos en la siguiente tabla.

Tabla 5. Comparación de peticiones y error

Comparación de tiempos			
Prueba	arrivalRate	Duración (s)	Error (%)
P1	10	50	0%
P2	100	50	40.17%
P3	100	100	37.51%

10.4. RESPUESTAS DEL SERVIDOR

Artillery nos muestra la cantidad de peticiones que obtuvieron una respuesta satisfactoria del servidor (HTTP 200) y esto va relacionado a la cantidad de peticiones realizadas. Esto se expone en la tabla 6.

Tabla 6. Respuestas OK del servidor

Respuesta OK del servidor			
Prueba	Peticiones	HTTP 200	Correctos (%)
P1	500	498	99.6%
P2	4670	2786	59.65%
P3	10000	6249	62.49%
P4	240020	18940	7.89%
P5	25	25	100%

Con esta información podemos graficar y ver la cantidad de peticiones correctas según la cantidad total de peticiones; esta información la podemos ver en la figura 1.

Esta nos demuestra que, a mayor cantidad de peticiones, mas es la cantidad de errores en el servidor, los errores presentados en la P4 son los siguientes:

- `errors.ECONNRESET` = 13
- `errors.ETIMEDOUT` = 221067

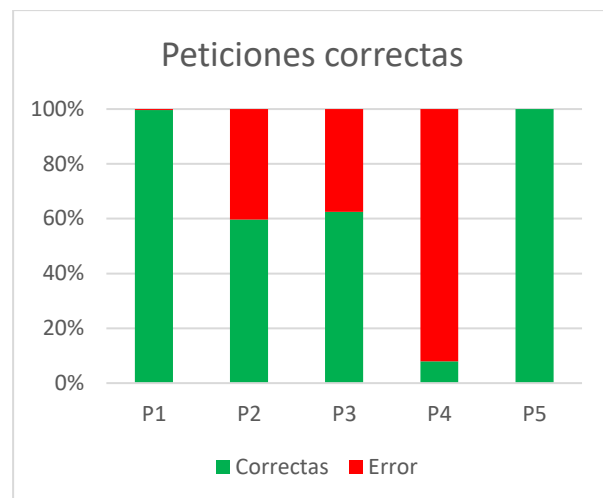


Figura 1. Peticiones correctas al servidor

10.5. TIEMPOS DE RESPUESTA

Artillery además nos permite ver los tiempos de respuesta del servidor, estos están expuestos en la tabla 7.

Tabla 7. Tiempos de respuesta

Tiempos de respuesta		
Prueba	Media (ms)	Máxima (ms)
P1	No registra	2350
P2	2101	8894
P3	2018	9977
P4	2056	9446
P5	No registra	2121

Esto nos demuestra que a mayor cantidad de usuarios intentando ingresar al servidor mayor es la cantidad de tiempo de espera, y esto se traduce en no respuesta del servidor.

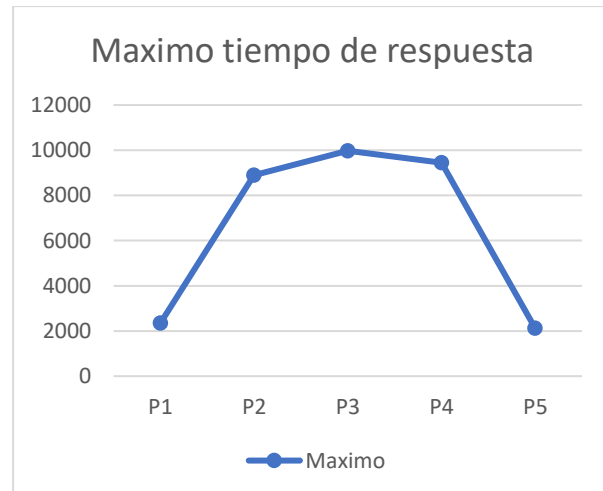


Figura 2. Tiempos máximos de respuesta

10.6. CONCLUSIONES DE LOS GRAFICOS

Con *Artillery* podemos obtener estadísticas del rendimiento de nuestro servidor, y esta pasara a ser una herramienta importante en el desarrollo de soluciones informáticas web a partir de ahora.

11. CONCLUSIONES

Este fue un proyecto muy útil, y aplicable en contextos de despliegue de aplicaciones reales, para crear y mantener un servidor se debe tener en cuenta el flujo de usuarios, el tiempo que podrían permanecer y las peticiones y solicitudes de información o datos que estos puedan realizar. El balanceador de carga es una excelente forma de optimizar los procesos de los servidores para aplicaciones que manejen un gran tráfico de usuarios o de información y con una correcta aplicación puede crear una mejor experiencia de usuario en tiempos de respuesta así mismo evitar posibles fallos o saturaciones del servidor.

BIBLIOGRAFÍA

- 1] C. Kumar, «10 balanceador de carga de código abierto para alta disponibilidad y rendimiento mejorado,» *geekflare*, 06 09 2022. [En línea]. Available: <https://geekflare.com/es/open-source-load-balancer/>. [Último acceso: 07 11 2022].
- 2] IONOS, «Crear un balanceador de carga,» 2022. [En línea]. Available: <https://www.ionos.es/ayuda/servidores-cloud/balanceador-de-carga/crear-un-balanceador-de-carga/>. [Último acceso: 07 11 2022].
- 3] The Apache Software Foundation, «Apache Module mod_proxy_http,» Apache, 2022. [En línea]. Available: https://httpd.apache.org/docs/2.4/mod/mod_proxy_http.html. [Último acceso: 07 11 2022].
- 4] The Apache Software Foundation, «The Number One HTTP Server On The Internet,» Apache, 08 06 2022. [En línea]. Available: <https://httpd.apache.org/>. [Último acceso: 07 11 2022].
- 5] MDN contributors, «Que es un servidor WEB?,» MDN web docs, 27 09 2022. [En línea]. Available: https://developer.mozilla.org/es/docs/Learn/Common_questions/What_is_a_web_server. [Último acceso: 07 11 2022].
- 6] MDN contributors, «Generalidades del protocolo HTTP,» MDN web docs, 03 10 2022. [En línea]. Available: <https://developer.mozilla.org/es/docs/Web/HTTP/Overview>. [Último acceso: 07 11 2022].
- 7] Oracle, «Oracle VM VirtualBox,» 2022. [En línea]. Available: <https://www.virtualbox.org/manual/UserManual.html#virt-why-useful>. [Último acceso: 07 11 2022].
- 8] M. A. Alvarez, «Qué es Vagrant, cómo trabajar con Vagrant,» *desarrolloweb.com*, 26 03 2022. [En línea]. Available: <https://desarrolloweb.com/articulos/trabajar-con-vagrant.html>. [Último acceso: 07 11 2022].
- 9] M. Garcia Mediavilla, J. L. Garrido Labrador, D. Gomez Chayan y A. Romero Chila, «CentOS,» 2015.
- 10] Artillery, «Why Artillery?,» Artillery, 2022. [En línea]. Available: <https://www.artillery.io/docs/guides/overview/why-artillery>. [Último acceso: 07 11 2022].
- 11] R. Arpaci-Dusseau, «Scheduling: Introduction,» *Three Easy Pieces*, 2008.
- 12] Gustavo.HdezF, Artist, *¿Qué es el balanceo de carga y cómo funciona?*. [Art]. Arquitectura de una balanceador de carga., 2022.