Balanceo de carga de servidores web

Jose Manuel Gomez Garcia (2190257), Daniel Hoyos Castillo (2180774), John Edward León (2211517), Álvaro Jose Santos (2195323).
jose_manuel.gomez@uao.edu.co , daniel.hoyos_cas@uao.edu.co, john.leon@uao.edu.co , alvaro.santos@uao.edu.co.
Universidad de Autónoma de Occidente - Cali

Resumen – En el presente documento se incluye la información teórica del balanceo de carga en servidores web, incluyendo los módulos usados como lo son Proxy Balancer y Artillery. Para el desarrollo del proyecto final del curso Telecomunicaciones 3 en el cual se implementan temas conocidos como lo son Apache y Firewall.

Abstract – This document includes theoretical information on load balancing in web servers, including the modules used such as Proxy Balancer and Artillery. For the development of the final project of the Telecommunications 3 course in which well-known topics such as Apache and Firewall are implemented.

I. INTRODUCCION

En el entorno tecnológico actual, el rendimiento y la disponibilidad de los servidores web son fundamentales para garantizar una experiencia fluida a los usuarios. Una estrategia eficaz para lograrlo es utilizar balanceadores de carga, que distribuyen de manera equitativa la carga de trabajo entre varios servidores. En este caso, vamos a explorar cómo implementar balanceadores de carga de servidores web en un entorno basado en CentOS 8 utilizando el módulo proxy balancer y la herramienta de pruebas de carga Artillery. CentOS 8, un sistema operativo ampliamente utilizado en servidores proporciona una base sólida y estable para implementar soluciones de balanceo de carga. El módulo proxy balancer es una extensión del servidor Apache en CentOS 8 que permite distribuir las solicitudes de los clientes entre múltiples servidores backend. Esto, mediante diferentes algoritmos, permite garantizar una carga de trabajo equitativa y mejorar el rendimiento y la disponibilidad de los servicios web.

Por otro lado, Artillery es una herramienta de pruebas de carga flexible y potente que nos permitirá evaluar el rendimiento y la resistencia de nuestros servidores web. Con Artillery, podemos simular una carga de trabajo realista enviando múltiples solicitudes concurrentes a los servidores y analizando su tiempo de respuesta, capacidad de procesamiento y estabilidad, brindando información valiosa para optimizar la configuración de balanceadores de carga y verificar que funcionan de manera eficiente. En este trabajo, se explora los pasos necesarios para configurar un entorno de balanceo de carga utilizando el módulo proxy balancer en CentOS 8. Se plantea el paso a paso para instalar y configurar el módulo proxy balancer, y cómo utilizar Artillery para realizar pruebas de carga y evaluar el rendimiento de los servidores web.

II. MARCO TEORICO

¿Qué son los balanceadores de carga de servidores web?

El balanceo de carga de servidores web es una técnica utilizada para distribuir la carga de trabajo entre múltiples servidores web con el objetivo de mejorar el rendimiento, la disponibilidad y la capacidad de escalabilidad de una aplicación o sitio web.

Cuando un servidor web recibe una gran cantidad de solicitudes de los usuarios, puede llegar a su capacidad máxima y generar una respuesta lenta o incluso un tiempo de espera agotado. En lugar de tener un solo servidor que maneje todas las solicitudes, el balanceo de carga distribuye la carga de trabajo entre varios servidores, lo que permite compartir la carga y evitar la saturación de recursos en un solo servidor.

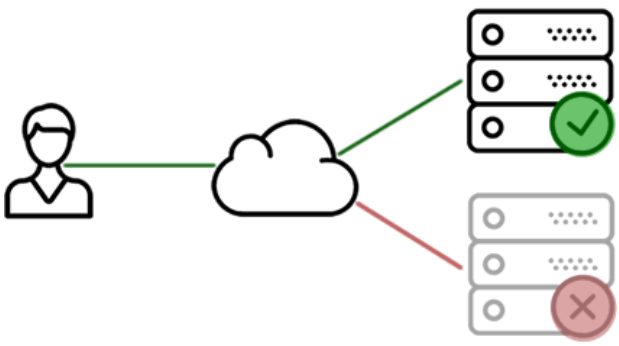


Fig.1 Ilustración grafica del balanceo de carga
Fuente: Balanceadores de carga y failover - WNPower

Existen diferentes métodos de balanceo de carga que se pueden utilizar, como:

1. Balanceo de carga basado en DNS: El servidor DNS distribuye las solicitudes de los clientes a diferentes servidores según un algoritmo de

balanceo predefinido. Cada vez que un cliente hace una solicitud, se le asigna la dirección IP de uno de los servidores disponibles.
2. Balanceo de carga basado en capa de red (TCP/IP): Se utiliza un dispositivo intermediario, como un balanceador de carga, para distribuir las solicitudes entrantes a diferentes servidores en función de la dirección IP y el puerto de destino.
3. Balanceo de carga basado en capa de aplicación (HTTP): El balanceador de carga examina el contenido de las solicitudes y decide a qué servidor enviarlas en función de ciertos criterios, como la carga actual del servidor, la capacidad de procesamiento o la ubicación geográfica.

Modulo Proxy Balancer

El módulo Proxy Balancer es un componente de Apache HTTP Server que permite realizar el balanceo de carga en un entorno web. Este módulo permite distribuir las solicitudes de los clientes entre múltiples servidores backend, optimizando el rendimiento y la disponibilidad de un sitio web o aplicación.

El funcionamiento del módulo Proxy Balancer se basa en el algoritmo de balanceo de carga que se configure. Este algoritmo determina cómo se asignan las solicitudes entrantes a los servidores backend disponibles. El metodo de balanceo se puede puede seleccionar con añadiendo la directiva ***Proxyset lbmethod*** [5] seguido del metodo que se desea el cual puede ser uno de los siguientes:

1. byrequest : En este tipo de balanceo se distribuye la carga por petición entrante, no toma en cuenta ni el peso ni el tiempo de respuesta del servidor, es posible determinar pesos distintos para cada servidor, lo que indica la proporcion que recibe respecto a las otras máquinas. Esto se logra con el atributo ***"loadfactor = #" donde*** el numero corresponde a la proporción. En caso de que todos los ***loadfactor*** sean iguales, significa que las peticiones se distribuyen por igual.[6]
2. bytraffic : A diferencia del método ***byrequest***, en este caso él se balancea teniendo en cuenta la cantidad de bytes que se van a procesar. En este caso, el ***loadfactor*** indica en que proporción de bytes I/O van a trabajar las maquinas. Por ejemplo, si se tienen ***loadfactor*** de 1 y 2 para la maquina A y B respectivamente, la maquina B procesara el doble de bytes que la maquina A [7]
3. bybusyness : Este método es similar al byrequest, se diferencia en el hecho que distribuye las peticiones a los servidores según la cantidad que tienen en su pila de request. En este caso se envía el request al servidor que tenga la menor cola. No toma en cuenta la cantidad de bytes ni el tiempo de respuesta. [8]
4. heartbeat : Este es un método más complejo de implementar, pero tiene como ventaja que balancea la carga según el tiempo de disponibilidad de los servidores [9]. Para este caso se debe configurar un heartbeat monitor en el servidor proxy el cual estar escuchando la respuesta de los servidores, Esto significa que es necesario configurar también los servidores para que envíen una señal indicativa de su estado actual.[10]

Artillery

Artillery es una herramienta de prueba de carga y estrés de código abierto que se utiliza para evaluar el rendimiento, la confiabilidad y la escalabilidad de aplicaciones y servicios web. Con Artillery, puedes simular cargas de trabajo de usuarios concurrentes y medir cómo responde tu sistema bajo diferentes condiciones [11].

Algunas características y funcionalidades clave de Artillery incluyen:

1. Escenarios flexibles: Puedes definir escenarios de prueba personalizados utilizando un lenguaje de configuración simple. Esto te permite simular diferentes tipos de solicitudes y comportamientos de los usuarios.
2. Protocolos admitidos: Artillery admite protocolos comunes como HTTP, HTTPS, WebSocket y TCP.
3. Generación de carga: Puedes generar una carga de trabajo de alta concurrencia enviando solicitudes HTTP simultáneas a tu aplicación o servicio.
4. Medición de métricas: Artillery proporciona métricas detalladas en tiempo real, como el tiempo de respuesta, la tasa de errores, el rendimiento de la CPU y la memoria. Esto te ayuda a identificar cuellos de botella y puntos débiles en tu sistema.
5. Escalabilidad: Puedes aumentar gradualmente la carga de trabajo para evaluar cómo responde tu sistema a medida que aumenta el número de usuarios concurrentes.
6. Integración con herramientas de monitoreo: Artillery se integra con herramientas populares de monitoreo y generación de informes, como

Grafana e InfluxDB, para que puedas visualizar y analizar los resultados de las pruebas.

Aparte de Artillery, actualmente están otras herramientas como:

- **Apache JMeter: JMeter:** Herramienta de código abierto desarrollada por Apache que permite realizar pruebas de carga y estrés en aplicaciones web y servicios. Ofrece una interfaz gráfica fácil de usar y es altamente configurable y extensible.
- **Gatling: Gatling:** Herramienta de prueba de rendimiento de código abierto que se centra en la simulación de usuarios concurrentes. Utiliza un lenguaje de dominio específico (DSL) llamado Gatling DSL para definir escenarios de prueba y ofrece una interfaz de informes en tiempo real.
- **Locust: Locust:** Herramienta de prueba de carga de código abierto escrita en Python. Permite definir escenarios de prueba en forma de código Python y simular la carga generada por múltiples usuarios concurrentes.
- **Vegeta:** Herramienta de prueba de carga de línea de comandos escrita en Go. Se utiliza para realizar pruebas de estrés en servicios HTTP y mide las tasas de solicitudes por segundo y los tiempos de respuesta.

III. CASO DE ESTUDIO

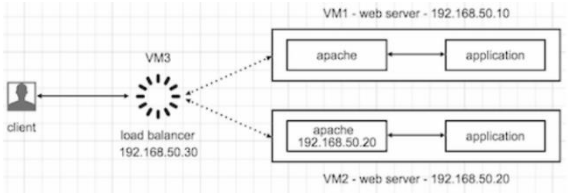
Descripción general del problema:

Se requiere implementar una solución para una página web que consume un api que lista emails y contraseña de los 2500 usuarios registrados, los datos de cada usuario pesan en promedio 60 bytes. El cliente indica que él cuenta con 3 servidores para esta solución, pero que son de bajo rendimiento. Cuentan con tan solo 512mb de RAM y baja capacidad de procesamiento. No obstante, el cliente también indica que esta página web no va a contar con más de 2500 peticiones por minuto, por lo que la solución debe ser capaz de responder a esta cantidad de peticiones teniendo en cuenta las capacidades de los servidores disponibles.

Requerimientos y arquitectura de posible solución

A partir de la descripción del problema es posible obtener los requerimientos y restricciones que se tienen para la solución del problema. Como requerimientos se tiene que la arquitectura debe soportar al menos 2500 peticiones por minuto y también la capacidad de transmitir 15000 bytes por petición. Como restricciones se tiene que: se tienen 3 servidores, cada uno con 512mb de ram y bajo poder de procesamiento.

Teniendo en cuenta os requerimientos y restricciones se planteó la siguiente arquitectura:



Esta arquitectura hara uso del balanceador de carga ***mod_proxy_balancer*** incluido en los módulos de apache. Este fue elegido teniendo en cuenta que los servidores tienen baja capacidad de procesamiento y RAM por lo que es preferible utilizar módulos que no impacten fuertemente al desempeño de la máquina, por lo que no se utilizan balanceadores externos como Nginx o ha_proxy. Como se explicó anteriormente, los métodos de balanceo de ***mod_proxy*** son sencillos y no requieren cálculos complejos por lo que se ajustan perfectamente a la solución. Además, se tuvo en cuenta que solo se cuenta con 3 servidores por lo que se destinó uno de ellos a trabajar como proxy balanceador y los otros dos como servers backend que atienden las peticiones.

Esta arquitectura de solución también permite a los servidores alivianar su carga de procesamiento por lo que, en condiciones ideales, las 2500/min se transforman a 1250/min por servidor backend.

A continuación, se va a detallar como se implementó esta arquitectura y como se plantearon los escenarios de pruebas para aproximarse a las necesidades del cliente.

IV. IMPLEMENTACION

1. Instalar máquinas virtuales con el vagrantFile. Esto simulara las 3 máquinas virtuales que se establecieron en la arquitectura

```
# -- mode: ruby --
# vi: set ft=ruby :

Vagrant.configure("2") do |config|

  if Vagrant.has_plugin? "vagrant-vbguest"
    config.vbguest.no_install  = true
    config.vbguest.auto_update = false
    config.vbguest.no_remote   = true
  end

  config.vm.define :servidorRest do |servidorRest|
    servidorRest.vm.box = "generic/centos8"
    servidorRest.vm.network :private_network, ip: "192.168.60.3"
    servidorRest.vm.hostname = "servidorRest"
  end
  config.vm.define :servidor1 do |servidor1|
    servidor1.vm.box = "generic/centos8"
    servidor1.vm.network :private_network, ip: "192.168.60.4"
    servidor1.vm.hostname = "servidor1"
  end
  config.vm.define :servidor2 do |servidor2|
    servidor2.vm.box = "generic/centos8"
    servidor2.vm.network :private_network, ip: "192.168.60.2"
    servidor2.vm.hostname = "servidor2"
  end
end
```

2. Desactivar selinux y firewalld.

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of these three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

3. Desactivar firewalld con el siguiente comando: service firewalld stop

4. Instalar mod_proxy_balance. Ingresando este comando:

sudo dnf install httpd httpd-tools mod_ssl

5. Configurar el módulo para habilitar el módulo Proxy balancer

sudo vim /etc/httpd/conf/httpd.conf

```
# Example:
# LoadModule foo_module modules/mod_foo.so
#
Include conf.modules.d/*.conf
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
#
# If you wish httpd to run as a different user or group, you must run
# httpd as root initially and it will switch.
#
```

LoadModule proxy_balancer_module modules/mod_proxy_balancer.so

LoadModule proxy_module modules/mod_proxy.so

LoadModule proxy_http_module modules/mod_proxy_http.so

6. Agregar la configuración del balanceador de carga en el archivo de configuración de Apache ingresando con el siguiente comando:

sudo vim /etc/httpd/conf.d/proxy-balancer.conf

```
<Proxy balancer://mycluster>
        BalancerMember http://10.0.0.1:80
        BalancerMember http://10.0.0.2:80
</Proxy>

ProxyPass /test balancer://mycluster
ProxyPassReverse /test balancer://mycluster
```

7. Luego en la siguiente ruta sudo vim /etc/httpd/conf/httpd.conf colocar lo siguiente:

```
<Directory />
    AllowOverride none
    Require all denied
</Directory>

<Proxy balancer://mycluster>
    BalancerMember http://192.168.60.4:80
    BalancerMember http://192.168.60.2:80
    ProxySet "lbmethod=bybusyness"
    # Agrega tantos BalancerMember como servidores de
    # También puedes agregar opciones de balanceo, co
</Proxy>

# Configura tu virtual host para utilizar el cluster
<VirtualHost *:80>
    ServerName myserver.com
    ProxyPreserveHost On

    ProxyPass / balancer://mycluster/
    ProxyPassReverse / balancer://mycluster/

    DocumentRoot /var/www/html
    <Directory /var/www/html>
        AllowOverride All
        Require all granted
        DirectoryIndex main.html index.html
    </Directory>
```

8. Reiniciar Apache

sudo systemctl restart httpd

9. Para el siguiente paso cabe resaltar que se debe hacer en todas las maquinas del vagrant file (No en el principal donde se instaló lo anterior)

- sudo dnf install httpd
- sudo systemctl start httpd
- En la ubicación /var/www/html/ crear el archivo index.html y añadir:

```
<!DOCTYPE html>

<html>

        <head>
          <title>¡Hola, mundo!</title>
        </head>

        <body>
          <h1>¡Hola, mundo!</h1>
          <p>Este es un ejemplo de página web.</p>
        </body>

</html>
```

Si se crean más de tres servidores repetir los pasos, si se quiere abrir desde un navegador se sugiere cambiar el contenido para visualizar el redireccionamiento entre las páginas.

10. Ahora ya teniendo lo que se mostrará en cada dirección ip procederemos a descargar el artillery él nos permitirá realiza pruebas de esfuerzo para simular las 2500 peticiones por minuto

- Instalación Node Js 16

sudo dnf update

curl -sL https://rpm.nodesource.com/setup_16.x | sudo bash -

sudo dnf install nodejs

node –v

- Instalación de npm 9.6.6

sudo yum install npm

npm install -g npm@latest

npm –v

- Instalación de Artillery: 2.0.0-dev9

npm view artillery versions

npm install -g artillery@2.0.0-dev9

11. Ahora para comprobar que se instaló se debe crear:

archive.yml en este caso test.yml en una ruta definida, donde se puede usar /home/vagrant

Y ahora dentro de este archivo ingresamos estos comandos:

```
config:
  target: "http://192.168.60.3"
  phases:
   - duration: 20
     arrivalRate: 1000
  defaults:
    headers:
      User-Agent: "Artillery"
scenarios:
  - name: "Test"
    flow:
      - get:
          url: "/"
```

Nota: Tener cuidado con los espacios y alineación ya que al correr Artillery generara errores. Debe quedar tal cual como se ilustra en la anterior imagen.

12. Por último solo queda activar el artillery con este comando:

artillery run test.yml

V. PRUEBAS TENIENDO EN CUENTA ARQUITECUTRA DEL DISEÑO.

Para realizar las pruebas teniendo en cuenta la arquitectura propuesta, primero se diseñó una api que simule el funcionamiento expresado por el cliente. Este api puede ser encontrada en el repositorio https://github.com/bipopa/apiUsers. Para la prueba se descargó el api en cada uno de los servidores backend con los siguientes comandos:

sudo yum install git -y

cd /home/vagrant

git clone https://github.com/bipopa/apiUsers.git

cd apiUsers/

npm i

node src/users.js

De esta forma el API queda funcionando, este API envía una lista en JSON de los email y contraseña de los usuarios registrados y también permite realizar post para registrar nuevos usuarios. Primero se utilizó aritllery para crear una base de 2600 usuarios y se atacó cada uno de los servers para que terminara con la misma cantidad de usuarios. Esta prueba se realizó con un flujo de usuarios pequeños para garantizar que todas las peticiones se registraran

```
config:
  target: "http://192.168.50.20:4000/api"
  phases:
    - duration: 130
      arrivalRate: 20
      name: Warm up


scenarios:
  - name: "test1"
    flow:

      - post:
          url: "/signup"

          json:
            email: "correoGenerico@email.com"
            password: "Contraseña123"
```

Luego se implementó el balanceador de carga con la siguiente línea de comandos en la configuración de mod_proxy:

```
<IfModule mod_proxy.c>
    ProxyRequests Off
    <Proxy *>
        Require all granted
    </Proxy>
    <proxy balancer://users>
        BalancerMember http://192.168.50.20:4000/api/users
        BalancerMember http://192.168.50.10:4000/api/users
    </proxy>
ProxyPass "/users" "balancer://users/"
ProxyPassReverse "/users" "balancer://users/"
    <proxy balancer://signup>
        BalancerMember http://192.168.50.20:4000/api/signup
        BalancerMember http://192.168.50.10:4000/api/signup
        ProxySet lbmethod=bybusyness
    </proxy>
ProxyPass "/signup" "balancer://signup/"
ProxyPassReverse "/signup" "balancer://signup/"

    <proxy balancer://frontend>
        BalancerMember http://192.168.50.20
        BalancerMember http://192.168.50.10
        ProxySet lbmethod=bytraffic
    </proxy>
ProxyPass "/" "balancer://frontend/"
ProxyPassReverse "/" "balancer://frontend/"
</IfModule>
```

Note que se crearon 3 proxys, uno para las peticiones GET, el cual funciona cuando alguien consulta el servidor proxy con la dirección 192.168.50.30/users. Y redirige la petición a las apis que están alojadas en el puerto 4000 de cada uno de los servidores backend. El otro proxy utiliza el método POST del API y permite registrar un usuario, así accede con la dirección 192.168.50.30/signup. Por último, cuando se accede a la página principal, esta redirige la petición a uno de los dos servidores de backend. Estos tienen como frontend una página que consume la API y lista los usuarios creados.

Las páginas de llegada de los servidores backend tienen el siguiente Código

```
<!DOCTYPE html>
<html>
        <body>
                <h1>Servidor Backend2</h1>
                <div id="app"></div>
                <script src="index.js" type="module"></script>
        </body>
</html>
~
~
```

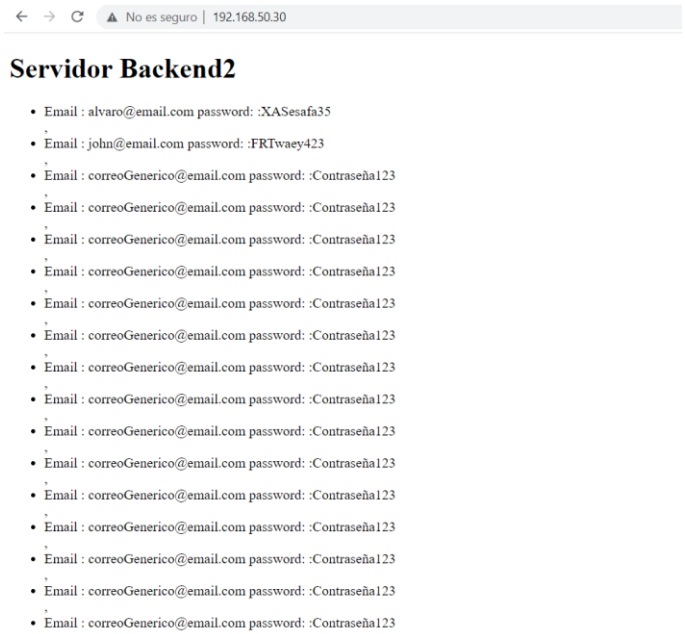
Como se observa es una página sencilla que llama al script index.js el cual consume el API, este script tiene el siguiente contenido y genera el siguiente resultado al realizar una petición a la dirección 192.168.50.30/

```
const API_URL = 'http://192.168.50.20:4000/api/users';

const xhr = new XMLHttpRequest();

//función manejadora

function onRequestHandler() {
 if(this.readyState == 4 && this.status == 200){
        const data = JSON.parse(this.response);
        const HTMLResponse = document.querySelector("#app");
        const tpl = data.map(user => `<li>Email : ${user.email}   password: :${user.password}</li>`);
        HTMLResponse.innerHTML = `<ul>${tpl}</ul>`
        }
}
xhr.addEventListener("load", onRequestHandler);
xhr.open("GET", `${API_URL}`);
xhr.send();
~
~
~
```



Escenarios de pruebas

Una vez se llenaron las bases de datos con la misma cantidad de información, se realizaron diferentes escenarios de pruebas para comprobar la eficiencia de los métodos de balanceo. La primera prueba que se realizó fue llevar los servidores backend a su rendimiento máximo generando una prueba de estrés que no puedan superar. Esto con el fin de demostrar la eficiencia del balanceador. A continuación, se muestra el script que utiliza artillery para esta prueba.

```
config:
  target: "http://192.168.50.10:4000/api"
  phases:
    - duration: 60
      arrivalRate: 50
      name: Warm up


scenarios:
  - name: "test1"
    flow:

      - get:
          url: "/users"
```

A continuación se realizó la misma prueba pero con utilizando el proxy y se verifico la eficiencia del balanceador. Para esto se utilizó el mismo script anterior pero la url es http://192.168.50.30 .Los resultados de estas pruebas se pueden observar en las tablas de 50 peticiones por segundo sin y con balanceo respectivamente.

También se realizaron diferentes pruebas cargando más datos a un servidor que a otro y utilizando el método bybusyness y se verifico su eficiencia. No obstante, este

método no es relevante para los requerimientos del cliente pues cada uno de los servidores porta la misma cantidad de datos y capacidad de procesamiento. Por lo tanto, se decidió utilizar el método by request para el balanceador, este ofrece un algoritmo sencillo y una solución eficiente.

VI. RESULTADOS

Pruebas para duración de 15 seg, 500 solicitudes por segundo

Parámetro	**192.168.60.2**	**192.168.60.3 bybusynes**
http.request_rate:	335/sec	511/sec
http.response_rate:	260/sec	472/sec
users.created_by_name.Test:	7500	7500
vusers.created.total:	7500	7500
Vusers.complete:	6175	7500
vusers.failed:	1325	
http.requests:	7500	7500
http.codes.200:	6175	7469
http.responses:	6175	7500
http.codes.502		31
errors.ECONNRESET	1233	
errors.ETIMEDOUT	92	
vusers.session_length:		
min:	41,1	4,5
Max:	9919,5	2059,5
Median:	2018,7	219,2
p95	7117	539,2
P99	8024,5	837,3
http.response_time:		
Min:	33,1	3,8
Max:	9919	2058,9
Median:	2018,7	219,2
P95:	7117	539,2
P99:	8024,5	837,3

En la anterior tabla se pueden observar los datos específicos pero lo más relevantes a considerar son los percentiles del 95 y 99% respectivamente en el tiempo de respuesta. La segunda columna es la prueba sin el balanceador donde se obtiene un tiempo de respuesta de 7117 milisegundos y cuando realizamos la prueba con el balanceador que es la columna tres respectivamente obtenemos un tiempo de 539 milisegundos. Por ello podemos corroborar que el balanceador de carga funciona correctamente y atiene la cantidad de solicitudes simuladas con Artillery.

Pruebas para duración de 60 seg, 600 solicitudes por segundo

Parametro	**192.168.60.2**	**192.168.60.3 bybusynes**
http.request_rate:	589/sec	517/sec
http.response_rate:	331/sec	488/sec
users.created_by_name.Test:	36000	34151
vusers.created.total:	36000	34151
vusers.completed:	21740	31746
vusers.failed:	14260	1527
http.requests:	36000	34151
http.codes.200:	21740	31627
http.responses:	21740	31746
http.codes.502		119
errors.ECONNRESET	7674	1388
errors.ETIMEDOUT	6586	139
vusers.session_length:		
min:	39,6	39,8
Max:	10004,7	9891
Median:	3395,5	1353,1
p95	8692,8	4316,6
P99	9607,1	7407,5
http.response_time:		
Min:	32,1	37,8
Max:	10004,3	9890,3
Median:	3395,5	1353,1
P95:	8692,8	4316,6
P99:	9607,1	7407,5

De igual manera se realiza otra prueba en el cual también se evidencia que el tiempo de respuesta del balanceador es mucho menor que realizando la prueba sin el balanceador, el tiempo de respuesta se reduce a la mitad aproximadamente lo cual es un rendimiento adecuado y óptimo para no sobrecargar solicitudes hacia un solo servidor.

Ahora se evidencian los resultados realizando otras pruebas con la API rest implementada de manera alternativa como se evidencian en las siguientes tablas:

- Pruebas sin balanceo 50request/sec

errors.ETIMEDOUT	2788
http.codes.200	416
http.request_rate	20/sec
http.requests	3000
http.response_time	
min	4
max	9952
median	4965.3
p95	9607.1
p99	9801.2
http.responses	416
vusers.completed	212
vusers.created	3000
vusers.created_by_name.test1	3000
vusers.failed	2788
vusers.session_length	
min	1013.3
max	10379.1
median	3072.4
p95	6976.1
p99	7260.8

- Pruebas con balanceo 50request/sec

errors.ETIMEDOUT	2922
http.codes.200	78
http.request_rate	50/sec
http.requests	3000
http.response_time	
min	1
max	6858
median	671.9
p95	4147.4
p99	6312.2
http.responses	3000
vusers.completed	3000
vusers.created	3000
vusers.created_by_name.test1	3000
vusers.failed	0
vusers.session_length	
min	1005.8
max	9384.4
median	1978.7
p95	7117
p99	8692.8

Como se observa en las tablas, al utilizar un proxy es posible procesar casi el total de las 3000 peticiones, más de las especificadas por el cliente, sin comprometer la integridad de los servicios. Mientras que si se utilizaran estos servidores directamente el servidor no es capaz de responder y termina descartando la mayoría de las peticiones.

VII. CONCLUSIONES

El uso del módulo proxy balancer en el balanceo de carga de servidores web brinda beneficios significativos en términos de rendimiento y disponibilidad. Al distribuir equitativamente las solicitudes entrantes entre múltiples servidores backend, se logra una mejor utilización de los recursos y se evita la sobrecarga en un solo servidor. Esto resulta en una mayor capacidad de procesamiento y una respuesta más rápida a las solicitudes de los usuarios. Además, el módulo proxy balancer permite una fácil configuración y escalabilidad, lo que facilita la adaptación a las necesidades cambiantes del entorno tecnológico.

Además, dentro del proxy balancer hay distintos métodos de balanceo dependiendo la aplicación. Cuando se cuenta con servidores más potentes que otros, la mejor opción sería byheartbeeat pues este utiliza el dato de tiempo libre de cada servidor. Por otro lado, si se cuenta con servidores iguales que desempeñan la misma función, el método byrequest es el indicado. Por último, se no que el método bybusyness es mejo cuando se tiene servidores que tienen distintas funciones y tienen diferentes tiempos de respuesta.

Para finalizar Artillery, como herramienta de pruebas de carga, desempeña un papel crucial en la optimización del rendimiento y la resistencia de los servidores web. Al simular una carga de trabajo realista mediante el envío de múltiples solicitudes concurrentes, Artillery proporciona información valiosa sobre el rendimiento de los servidores y su capacidad para manejar una carga significativa. Esto permite identificar posibles cuellos de botella, realizar ajustes en la configuración y mejorar la capacidad de respuesta de los servidores web. Con su enfoque flexible y su capacidad de generar informes detallados, Artillery se convierte en una herramienta indispensable para asegurar la eficiencia y la estabilidad de los sistemas basados en balanceadores de carga.

VIII. REFERENCIAS

[1] H. Wacker. "¿Qué es el balanceo de carga?" ComputerWorld | Innovación, negocio y tecnología. https://www.computerworld.es/tendencias/que-es-el-balanceo-de-carga (accedido el 1 de mayo de 2023).

[2] Copyright IBM Corporation 2012. "Module mod_proxy_balancer". IBM - Deutschland | IBM. https://www.ibm.com/docs/en/i/7.1?topic=ssw_ibm_i_71/rzaie/rzaiemod_proxy_balancer.html (accedido el 1 de mayo de 2023).

[3] R. Aguilera. "Tests de rendimiento con Artillery". Adictosaltrabajo.com.https://www.adictosaltrabajo.

com/2018/02/22/tests-de-rendimiento-con-artillery/ (accedido el 1 de mayo de 2023).

[4] "Apache Module mod_proxy," mod_proxy - Apache HTTP Server Version 2.4, https://httpd.apache.org/docs/2.4/mod/mod_proxy.html.

[5] "Apache Module mod_proxy_balancer," mod_proxy_balancer - Apache HTTP Server Version 2.4, https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html.

[6] "Apache Module mod_lbmethod_byrequests," mod_lbmethod_byrequests - Apache HTTP Server Version 2.4, https://httpd.apache.org/docs/2.4/mod/mod_lbmethod_byrequests.html.

[7] "Apache Module mod_lbmethod_bytraffic," mod_lbmethod_bytraffic - Apache HTTP Server Version 2.4, https://httpd.apache.org/docs/2.4/mod/mod_lbmethod_bytraffic.html.

[8] "Apache Module mod_lbmethod_bybusyness," mod_lbmethod_bybusyness - Apache HTTP Server Version 2.4, https://httpd.apache.org/docs/2.4/mod/mod_lbmethod_bybusyness.html.

[9] "Apache Module mod_lbmethod_heartbeat," mod_lbmethod_heartbeat - Apache HTTP Server Version 2.4, https://httpd.apache.org/docs/2.4/mod/mod_lbmethod_heartbeat.html.

[10] "Apache Module mod_heartmonitor," mod_heartmonitor - Apache HTTP Server Version 2.4, https://httpd.apache.org/docs/2.4/mod/mod_heartmonitor.html.

[11] "Apache Module mod_proxy," mod_proxy - Apache HTTP Server Version 2.4, https://ucuenca.edu.ec/manual/en/mod/mod_proxy.html#proxypass .

[12] "Why artillery?," Artillery, https://www.artillery.io/docs/guides/overview/why-artillery.

[13] "Installing artillery CLI," Artillery, https://www.artillery.io/docs/guides/getting-started/installing-artillery.

[14] "Test scripts," Artillery, https://www.artillery.io/docs/guides/guides/test-script-reference.