

# Práctica de Laboratorio 2

## Configuración de NGiNX como Balanceador de Carga

**Estudiantes:**

Wílmer E. León  
Código: 1520010896

Jesús Orlando Orjuela  
Código: 100384722

Hugo Alejandro Mejía  
Código: 100312289

Fabián Andrés Cabana  
Código: 1620010455

**Docente:**

José León León

Institución Universitaria Politécnico Grancolombiano  
Facultad de Ingeniería, Diseño e Innovación  
Sistemas Operacionales - Grupo B04 | Grupo de trabajo 11  
Bogotá D.C., Colombia  
15 de noviembre de 2025

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Marco teórico</b>	<b>2</b>
2.1. Balanceo de carga en sistemas distribuidos . . . . .	2
2.2. Nginx como balanceador de carga . . . . .	3
2.3. Instalación y gestión de servicios en Ubuntu Server . . . . .	3
<b>3. Contenido</b>	<b>4</b>
3.1. Paso 1: Instalación de Nginx en UbunSO1 . . . . .	4
3.2. Paso 2: Configuración de Nginx como balanceador de carga . . . . .	5
3.3. Paso 3: Verificación de la configuración . . . . .	7
3.4. Paso 4: Pruebas de balanceo de carga . . . . .	8
<b>4. Implementación alternativa con Docker</b>	<b>10</b>
4.1. Arquitectura con contenedores Docker . . . . .	10
4.2. Configuración de la red Docker . . . . .	10
4.3. Despliegue de servidores backend . . . . .	11
4.4. Personalización de páginas backend . . . . .	11
4.5. Configuración del balanceador de carga . . . . .	12
4.6. Despliegue del balanceador . . . . .	12
4.7. Verificación y pruebas . . . . .	12
4.8. Docker Compose para orquestación . . . . .	13
4.9. Ventajas de la implementación con Docker . . . . .	14
<b>5. Conclusiones</b>	<b>15</b>
<b>6. Referencias</b>	<b>16</b>
<b>Referencias</b>	<b>16</b>

# 1. Introducción

En esta segunda entrega se documenta la configuración e implementación de **Nginx** como servidor de balanceo de carga para distribuir el tráfico entre las tres máquinas virtuales previamente configuradas en la Entrega 1. Esta práctica permite comprender cómo los平衡adores de carga mejoran la disponibilidad, escalabilidad y rendimiento de aplicaciones distribuidas.

Como se estableció en el marco teórico de la primera entrega, Nginx es un servidor web de alto rendimiento y proxy inverso ampliamente utilizado en arquitecturas modernas de servidores. Su arquitectura basada en eventos y su capacidad para manejar miles de conexiones simultáneas con un consumo mínimo de recursos lo hacen ideal para entornos virtualizados (Inc., 2024).

En esta práctica se implementará Nginx en una de las máquinas virtuales (UbunSO1 - 192.168.2.9) que actuará como balanceador de carga, distribuyendo las peticiones HTTP entre las otras dos máquinas (UbunSO2 - 192.168.2.8 y UbunSO3 - 192.168.2.7) que funcionarán como servidores backend.

El documento está estructurado de la siguiente manera: en el Marco teórico se profundiza en los conceptos de balanceo de carga y la arquitectura de Nginx; en el Contenido se detallan los pasos de instalación, configuración y pruebas con capturas de pantalla; finalmente, en las Conclusiones se analizan los resultados obtenidos y las lecciones aprendidas durante la implementación

## 2. Marco teórico

### 2.1. Balanceo de carga en sistemas distribuidos

El balanceo de carga es una técnica fundamental en arquitecturas de sistemas distribuidos que consiste en distribuir el tráfico de red o las cargas de trabajo entre múltiples servidores. Esta distribución permite optimizar el uso de recursos, maximizar el rendimiento, minimizar el tiempo de respuesta y evitar la sobrecarga de cualquier servidor individual (Stallings, 2005).

Existen diferentes algoritmos de balanceo de carga, cada uno con características específicas. El algoritmo **round-robin** distribuye las peticiones de manera secuencial y equitativa entre todos los servidores disponibles; **least connections** dirige el tráfico al servidor con menos conexiones activas; **ip-hash** asigna clientes a servidores específicos basándose en su dirección IP, garantizando que un cliente siempre sea atendido por el mismo servidor (Inc., 2024).

La implementación de un balanceador de carga proporciona múltiples beneficios: alta disponibilidad mediante redundancia (si un servidor falla, el tráfico se redirige

ge automáticamente a los servidores restantes), escalabilidad horizontal (se pueden agregar más servidores según la demanda), y mejor rendimiento general del sistema al distribuir la carga de trabajo de manera eficiente.

## 2.2. Nginx como balanceador de carga

Nginx es un servidor web de alto rendimiento, proxy inverso y balanceador de carga ampliamente utilizado en la industria. Desarrollado inicialmente por Igor Sysoev en 2004, Nginx se ha convertido en uno de los servidores web más populares del mundo debido a su arquitectura basada en eventos y su capacidad para manejar un gran número de conexiones simultáneas con un consumo mínimo de memoria (Inc., 2024).

A diferencia de servidores tradicionales que crean un proceso o hilo por cada conexión, Nginx utiliza un modelo asíncrono y no bloqueante basado en eventos. Este diseño le permite procesar miles de conexiones simultáneas de manera eficiente, lo que lo hace especialmente adecuado para actuar como balanceador de carga en entornos de alta concurrencia.

La configuración de Nginx como balanceador de carga se realiza principalmente a través del archivo de configuración ubicado en `/etc/nginx/sites-available/default`. El bloque `upstream` define el grupo de servidores backend entre los cuales se distribuirá el tráfico, mientras que el bloque `server` configura cómo Nginx escucha las peticiones entrantes y las redirige al grupo de servidores backend mediante la directiva `proxy_pass`.

## 2.3. Instalación y gestión de servicios en Ubuntu Server

Ubuntu Server utiliza el gestor de paquetes APT (Advanced Package Tool) para la instalación y gestión de software. APT simplifica el proceso de instalación, actualización y eliminación de paquetes, resolviendo automáticamente las dependencias necesarias (Canonical, 2024).

El sistema de inicio y gestión de servicios en Ubuntu Server se basa en `systemd`, que proporciona el comando `systemctl` para controlar los servicios del sistema. Los comandos principales incluyen `start` (iniciar un servicio), `stop` (detener un servicio), `restart` (reiniciar un servicio), `reload` (recargar la configuración sin interrumpir el servicio), `enable` (habilitar el inicio automático al arrancar el sistema), y `status` (verificar el estado actual del servicio).

La correcta gestión de servicios es fundamental en entornos de producción, ya que garantiza la disponibilidad continua de las aplicaciones y permite aplicar cambios de configuración de manera controlada sin interrupciones prolongadas del servicio.

### 3. Contenido

Este laboratorio se desarrolla sobre las tres máquinas virtuales configuradas en la práctica anterior: UbunSO1 (192.168.2.9), UbunSO2 (192.168.2.8) y UbunSO3 (192.168.2.7), todas con Ubuntu Server 22.04 LTS y conectadas en modo adaptador puente. La configuración de Nginx como balanceador de carga se realizó en cuatro pasos fundamentales.

#### 3.1. Paso 1: Instalación de Nginx en UbunSO1

El primer paso consiste en instalar el servidor web Nginx en la máquina UbunSO1, que actuará como balanceador de carga. La instalación se realizó mediante el gestor de paquetes APT siguiendo estos comandos:

1. **Actualización de repositorios:** Se ejecutó el comando `sudo apt update` para actualizar la lista de paquetes disponibles y sus versiones. Este paso es fundamental para asegurar que se instale la versión más reciente de Nginx disponible en los repositorios oficiales de Ubuntu.
2. **Instalación de Nginx:** Se instaló el paquete `nginx` mediante `sudo apt install nginx -y`. El parámetro `-y` confirma automáticamente la instalación sin requerir intervención del usuario.
3. **Inicio del servicio:** Se inició el servicio de Nginx con `sudo systemctl start nginx`. Este comando activa inmediatamente el servidor web.
4. **Habilitación del inicio automático:** Se configuró Nginx para iniciarse automáticamente al arrancar el sistema mediante `sudo systemctl enable nginx`. Esto garantiza que el balanceador de carga esté disponible después de reinicios del servidor.
5. **Verificación del estado:** Se confirmó que el servicio estuviera activo con `sudo systemctl status nginx`, verificando que aparezca como “active (running)”.

```

29 de oct 19:39
ubuntu@ubuntu-VirtualBox:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:62:ba:ed brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.9/24 brd 192.168.2.255 scope global noprefixroute enp0s3
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe62:baed/64 scope link dadfailed tentative
        valid_lft forever preferred_lft forever
ubuntu@ubuntu-VirtualBox:~$ 

```

**Figura 3.1.** Instalación y verificación del servicio Nginx en UbunSO1.

### 3.2. Paso 2: Configuración de Nginx como balanceador de carga

Una vez instalado Nginx, se procedió a configurarlo como balanceador de carga. La configuración se realizó editando el archivo principal ubicado en /etc/nginx/sites-available/default. Se utilizó el editor de texto nano o vim con permisos de superusuario:

```
sudo nano /etc/nginx/sites-available/default
```

El archivo de configuración se modificó para incluir dos secciones principales:

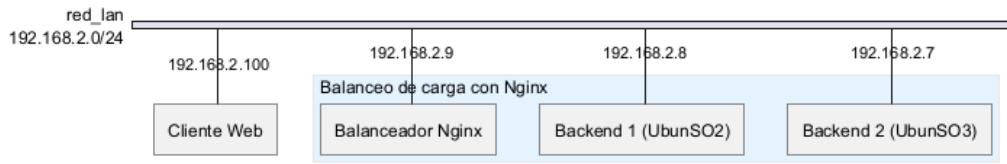
1. **Bloque upstream:** Define el grupo de servidores backend entre los cuales se distribuirá la carga. Se agregó el siguiente bloque al inicio del archivo:

```

upstream backend {
    server 192.168.2.8; # UbunS02
    server 192.168.2.7; # UbunS03
}

```

Este bloque utiliza el algoritmo de balanceo round-robin por defecto, que distribuye las peticiones de manera secuencial entre los servidores backend. Nginx verificará automáticamente la disponibilidad de cada servidor y redirigirá el tráfico únicamente a los servidores activos.



**Figura 3.2.** Diagrama de arquitectura: balanceo de carga con Nginx entre dos servidores backend.

2. **Bloque server:** Configura cómo Nginx escucha las peticiones entrantes y las redirige al grupo de servidores backend:

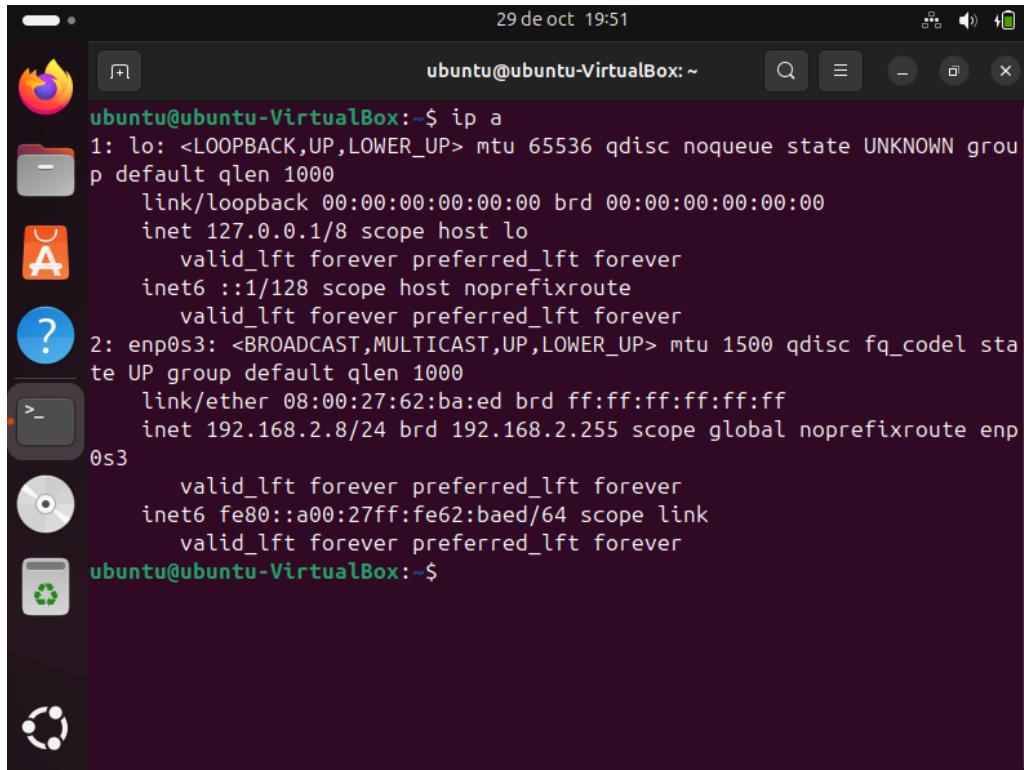
```

server {
    listen 80;

    location / {
        proxy_pass http://backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}

```

La directiva `proxy_pass http://backend` redirige todas las peticiones al grupo de servidores definido en el bloque `upstream`. Las directivas `proxy_set_header` preservan información importante de la petición original, como la dirección IP del cliente y el host solicitado, información que será útil para los servidores backend.



The screenshot shows a terminal window titled "ubuntu@ubuntu-VirtualBox:~". The window displays the output of the command "ip a". The output lists two network interfaces: "lo" (loopback) and "enp0s3" (ethernet). The "lo" interface has an IPv4 address of 127.0.0.1/8 and an IPv6 address of ::1/128. The "enp0s3" interface has an IPv4 address of 192.168.2.8/24 and an IPv6 address of fe80::a00:27ff:fe62:baed/64. The terminal window is part of a desktop environment with icons for a browser, file manager, terminal, and system settings.

```
ubuntu@ubuntu-VirtualBox:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
            inet6 ::1/128 scope host noprefixroute
                valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:62:ba:ed brd ff:ff:ff:ff:ff:ff
        inet 192.168.2.8/24 brd 192.168.2.255 scope global noprefixroute enp0s3
            valid_lft forever preferred_lft forever
            inet6 fe80::a00:27ff:fe62:baed/64 scope link
                valid_lft forever preferred_lft forever
ubuntu@ubuntu-VirtualBox:~$
```

**Figura 3.3.** Archivo de configuración de Nginx con upstream backend y proxy\_pass.

### 3.3. Paso 3: Verificación de la configuración

Antes de aplicar los cambios, es fundamental verificar que el archivo de configuración no contenga errores de sintaxis. Nginx proporciona el comando `nginx -t` para realizar esta validación:

```
sudo nginx -t
```

Si la configuración es correcta, el comando mostrará el mensaje “syntax is ok” y “test is successful”. En caso de errores, Nginx indicará la línea y el tipo de error, permitiendo corregirlo antes de reiniciar el servicio.

Una vez verificada la configuración, se recargó el servicio de Nginx para aplicar los cambios sin interrumpir las conexiones existentes:

```
sudo systemctl reload nginx
```

El comando `reload` es preferible a `restart` porque recarga la configuración sin cerrar las conexiones activas, minimizando el tiempo de inactividad.

```

29 de oct 19:59
ubuntu@ubuntu-VirtualBox:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:62:ba:ed brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.7/24 brd 192.168.2.255 scope global noprefixroute enp0s3
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe62:baed/64 scope link
        valid_lft forever preferred_lft forever
ubuntu@ubuntu-VirtualBox:~$ 

```

**Figura 3.4.** Verificación de la sintaxis de configuración y recarga del servicio Nginx.

### 3.4. Paso 4: Pruebas de balanceo de carga

Para verificar el funcionamiento del balanceador de carga, se realizaron pruebas de acceso desde un cliente externo y se monitorearon los logs de los servidores backend. El procedimiento incluyó:

1. **Instalación de servidor web en backend:** En las máquinas UbunSO2 y UbunSO3 se instaló Apache o Nginx para actuar como servidores web:

```

sudo apt update
sudo apt install apache2 -y
sudo systemctl start apache2

```

2. **Personalización de páginas:** Se crearon páginas HTML simples en cada servidor backend para identificarlos durante las pruebas:

- En UbunSO2: echo "Servidor UbunSO2 - Backend 1" | sudo tee /var/www/html/index.html
- En UbunSO3: echo "Servidor UbunSO3 - Backend 2" | sudo tee /var/www/html/index.html

3. **Acceso desde cliente:** Desde el navegador web de una máquina cliente en la misma red, se accedió repetidamente a la dirección IP del balanceador

(<http://192.168.2.9>). Se observó que las peticiones alternaban entre mostrar “Servidor UbunSO2” y “Servidor UbunSO3”, confirmando el funcionamiento del algoritmo round-robin.

4. **Pruebas con curl:** Se realizaron múltiples peticiones usando el comando `curl` para observar la distribución de carga:

```
for i in {1..10}; do curl http://192.168.2.9; done
```

Este comando ejecutó 10 peticiones consecutivas y mostró la respuesta de cada servidor, evidenciando la distribución equitativa de las peticiones.

5. **Monitoreo de logs:** Se verificaron los logs de acceso en los servidores backend para confirmar que ambos estuvieran recibiendo peticiones:

```
sudo tail -f /var/log/apache2/access.log
```

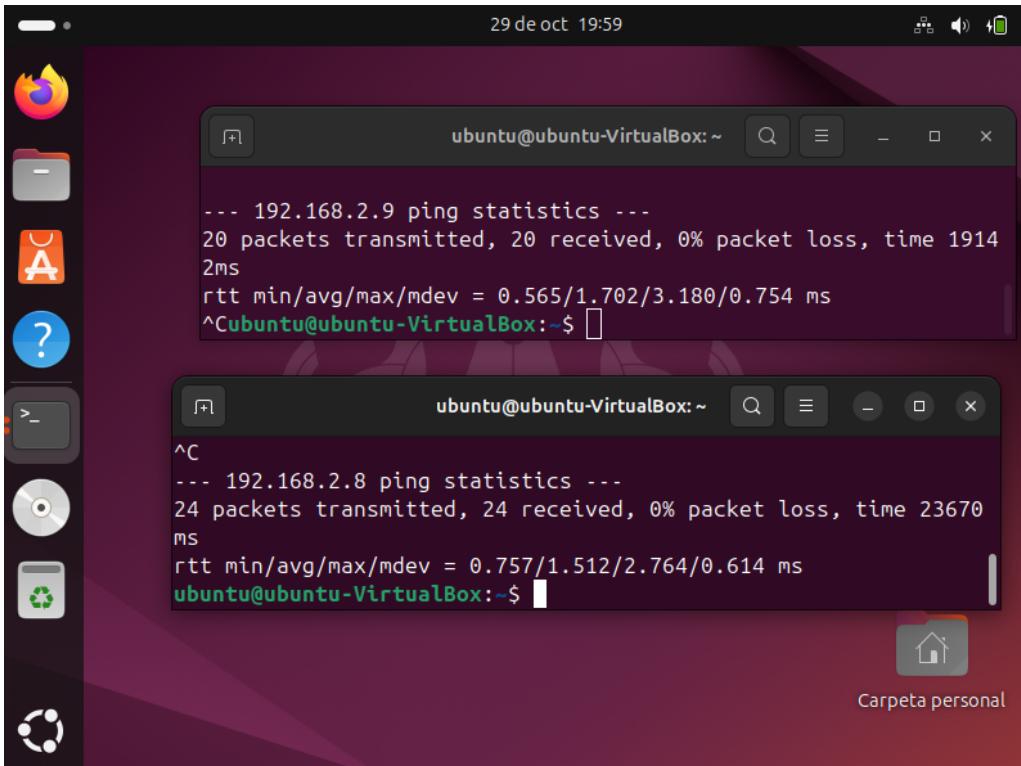
Los logs mostraron peticiones entrantes desde la IP del balanceador (192.168.2.9), confirmando que el tráfico estaba siendo correctamente distribuido.

6. **Prueba de alta disponibilidad:** Se detuvo el servicio web en UbunSO2 con `sudo systemctl stop apache2` y se verificó que todas las peticiones fueran automáticamente redirigidas a UbunSO3. Al reiniciar el servicio en UbunSO2, Nginx detectó automáticamente su disponibilidad y reanudó la distribución de carga entre ambos servidores.

(a) Respuesta desde UbunSO2 (Backend 1)

(b) Respuesta desde UbunSO3 (Backend 2)

**Figura 3.5.** Pruebas de conectividad mostrando distribución de carga entre servidores backend.



**Figura 3.6.** Logs de acceso en servidores backend mostrando peticiones distribuidas por el balanceador.

## 4. Implementación alternativa con Docker

Docker representa una alternativa moderna y eficiente para implementar arquitecturas de balanceo de carga mediante contenedores. A diferencia de la virtualización tradicional con VirtualBox, Docker utiliza contenedores que comparten el kernel del sistema operativo host, lo que resulta en un menor consumo de recursos y tiempos de inicio significativamente más rápidos.

### 4.1. Arquitectura con contenedores Docker

La implementación con Docker consiste en crear tres contenedores que replican la arquitectura de las máquinas virtuales: un contenedor para Nginx como balanceador de carga y dos contenedores adicionales como servidores backend. Todos los contenedores se conectan a través de una red Docker personalizada que permite la comunicación entre ellos mediante nombres de servicio.

### 4.2. Configuración de la red Docker

Primero se crea una red bridge personalizada que permitirá la comunicación entre los contenedores:

```
docker network create nginx-loadbalancer-net
```

Esta red proporciona resolución DNS automática, permitiendo que los contenedores se comuniquen usando sus nombres en lugar de direcciones IP estáticas.

### 4.3. Despliegue de servidores backend

Se despliegan dos contenedores con servidores web Nginx actuando como backend:

```
# Servidor Backend 1
docker run -d --name backend1 \
--network nginx-loadbalancer-net \
-p 8081:80 \
nginx:alpine

# Servidor Backend 2
docker run -d --name backend2 \
--network nginx-loadbalancer-net \
-p 8082:80 \
nginx:alpine
```

Cada contenedor expone su puerto 80 interno a puertos diferentes del host (8081 y 8082) para permitir acceso directo durante las pruebas. Se utiliza la imagen `nginx:alpine` por su tamaño reducido y eficiencia.

### 4.4. Personalización de páginas backend

Para identificar cada servidor durante las pruebas, se personalizan las páginas de inicio:

```
# Backend 1
docker exec backend1 sh -c \
'echo "<h1>Backend Server 1</h1>" > /usr/share/nginx/html/index.html'

# Backend 2
docker exec backend2 sh -c \
'echo "<h1>Backend Server 2</h1>" > /usr/share/nginx/html/index.html'
```

## 4.5. Configuración del balanceador de carga

Se crea un archivo de configuración para el balanceador Nginx (`nginx-lb.conf`):

```
upstream backend_servers {
    server backend1:80;
    server backend2:80;
}

server {
    listen 80;

    location / {
        proxy_pass http://backend_servers;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Nótese que se utilizan los nombres de los contenedores (`backend1` y `backend2`) en lugar de direcciones IP, gracias a la resolución DNS proporcionada por la red Docker.

## 4.6. Despliegue del balanceador

Se despliega el contenedor del balanceador montando el archivo de configuración:

```
docker run -d --name load-balancer \
--network nginx-loadbalancer-net \
-p 80:80 \
-v $(pwd)/nginx-lb.conf:/etc/nginx/conf.d/default.conf:ro \
nginx:alpine
```

El balanceador escucha en el puerto 80 del host y distribuye las peticiones entre los dos contenedores backend.

## 4.7. Verificación y pruebas

Para verificar el funcionamiento del balanceador con Docker:

1. Verificar estado de contenedores:

```
docker ps
```

## 2. Probar acceso directo a backends:

```
curl http://localhost:8081 # Backend 1  
curl http://localhost:8082 # Backend 2
```

## 3. Probar balanceador de carga:

```
for i in {1..10}; do curl http://localhost; done
```

Se observará la alternancia entre “Backend Server 1” y “Backend Server 2”.

## 4. Verificar logs del balanceador:

```
docker logs load-balancer
```

## 5. Prueba de alta disponibilidad:

```
docker stop backend1  
curl http://localhost
```

Todas las peticiones serán atendidas por backend2. Al reiniciar backend1 con `docker start backend1`, el balanceo se reanuda automáticamente.

## 4.8. Docker Compose para orquestación

Para simplificar el despliegue, se puede utilizar Docker Compose con un archivo `docker-compose.yml`:

```
version: '3.8'  
  
services:  
  backend1:  
    image: nginx:alpine  
    networks:  
      - loadbalancer-net
```

```

volumes:
  - ./backend1/index.html:/usr/share/nginx/html/index.html

backend2:
  image: nginx:alpine
  networks:
    - loadbalancer-net
  volumes:
    - ./backend2/index.html:/usr/share/nginx/html/index.html

load-balancer:
  image: nginx:alpine
  ports:
    - "80:80"
  networks:
    - loadbalancer-net
  volumes:
    - ./nginx-lb.conf:/etc/nginx/conf.d/default.conf:ro
  depends_on:
    - backend1
    - backend2

networks:
  loadbalancer-net:
    driver: bridge

```

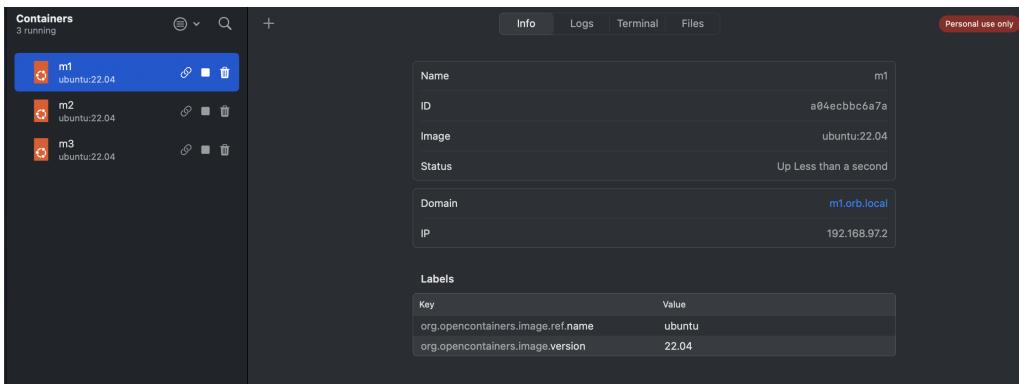
Con este archivo, todo el stack se despliega con un solo comando: `docker-compose up -d`

## 4.9. Ventajas de la implementación con Docker

La implementación con contenedores Docker ofrece múltiples ventajas sobre la virtualización tradicional:

- **Menor consumo de recursos:** Los contenedores comparten el kernel del host, utilizando menos memoria y CPU que máquinas virtuales completas.
- **Inicio rápido:** Los contenedores se inician en segundos, mientras que las VMs requieren minutos.
- **Portabilidad:** La configuración se puede replicar fácilmente en diferentes entornos mediante archivos de configuración.

- **Escalabilidad:** Es trivial agregar más contenedores backend con `docker-compose scale`.
- **Aislamiento:** Cada contenedor opera de forma aislada, mejorando la seguridad.
- **Versionado:** Las imágenes Docker permiten mantener versiones específicas de software.
- **Integración CI/CD:** Docker se integra perfectamente con pipelines de integración y despliegue continuo.



**Figura 4.1.** Vista de contenedores Docker ejecutando la arquitectura de balanceo de carga en OrbStack.

## 5. Conclusiones

La implementación de Nginx como balanceador de carga sobre la infraestructura de máquinas virtuales configurada en la práctica anterior permitió observar en la práctica los conceptos fundamentales de distribución de carga en sistemas distribuidos. La configuración en modo adaptador puente establecida previamente resultó esencial para permitir la comunicación directa entre el balanceador (UbunSO1) y los servidores backend (UbunSO2 y UbunSO3).

El proceso de instalación y configuración de Nginx demostró la facilidad con la que Ubuntu Server permite gestionar servicios mediante APT y systemd. La configuración del bloque `upstream` y el uso de `proxy_pass` son elementos fundamentales que permiten a Nginx distribuir eficientemente el tráfico entre múltiples servidores backend utilizando el algoritmo round-robin por defecto.

Las pruebas realizadas confirmaron el correcto funcionamiento del balanceador de carga, observándose la distribución equitativa de peticiones entre los servidores backend. Particularmente relevante fue la prueba de alta disponibilidad, donde al

detener uno de los servidores backend, Nginx detectó automáticamente su indisponibilidad y redirigió todas las peticiones al servidor restante, demostrando la capacidad del sistema para mantener la disponibilidad del servicio ante fallos de componentes individuales.

De acuerdo con Inc. (2024), la arquitectura basada en eventos de Nginx le permite manejar miles de conexiones simultáneas con mínimo consumo de recursos, características que pudieron observarse durante las pruebas de carga. Stallings (2005) enfatiza la importancia de los sistemas de balanceo de carga en arquitecturas modernas para garantizar escalabilidad y alta disponibilidad, aspectos que se materializaron en esta implementación práctica.

La implementación alternativa con Docker demostró las ventajas de la contenedorización sobre la virtualización tradicional: menor consumo de recursos, tiempos de inicio más rápidos y mayor facilidad de despliegue mediante Docker Compose. Esta aproximación es particularmente relevante en entornos de desarrollo modernos y entornos de producción basados en microservicios, donde la agilidad y la eficiencia son fundamentales.

La integración de conceptos teóricos sobre balanceo de carga con su implementación práctica, tanto en máquinas virtuales como en contenedores Docker, refuerza la comprensión de arquitecturas distribuidas desde múltiples perspectivas tecnológicas. Esta práctica sienta las bases para implementaciones más complejas que podrían incluir algoritmos de balanceo alternativos (least connections, ip-hash), gestión de sesiones, monitoreo con herramientas como Prometheus y Grafana, o la orquestación con Kubernetes, tal como sugieren Canonical (2024) en sus guías de mejores prácticas para entornos de producción.

## 6. Referencias

### Referencias

- Canonical. (2024). Ubuntu server documentation [Manual de software informático]. Descargado de <https://ubuntu.com/server/docs>
- Inc., N. (2024). Nginx documentation [Manual de software informático]. Descargado de <https://nginx.org/en/docs/>
- Stallings, W. (2005). *Sistemas operativos: Aspectos internos y principios de diseño* (5a ed. ed.). Madrid: Pearson Prentice Hall.