# ACTIVIDAD 4 - PROXY INVERSO

Cristóbal Suárez Abad

SEGURIDAD Y ALTA DISPONIBILIDAD - 2º ASIR

# Índice

# Descripción de las Tecnologías de Red Desplegadas

Para esta actividad vas a usar el proyecto de Docker adjunto que simula una arquitectura de producción profesional con la configuración de un Proxy Inverso. A continuación, se detallan las tecnologías y servicios que están operando dentro de cada nodo:

| Tecnología | Función | Descripción Técnica | Contenedor(es) |
|---|---|---|---|
| **Nginx (Reverse Proxy)** | **Punto de Entrada** | Intercepta peticiones externas y las redirige según reglas. | proxy |
| **Upstream (Load Balancing)** | **Balanceador** | Algoritmo **Round Robin** para repartir la carga de trabajo. | proxy (configuración) |
| **Nginx (Web Server)** | **Nodo de Backend** | Sirve el contenido final (las páginas de notas). | web1 y web2 |

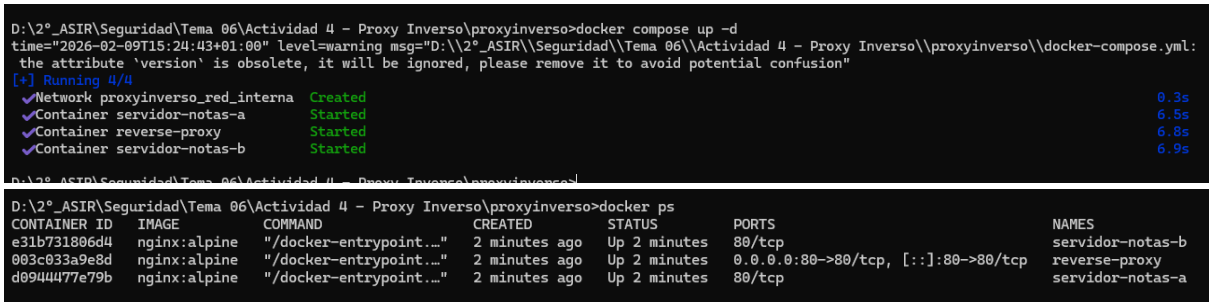
# Guía de Despliegue y Puesta en Marcha

Para que el laboratorio sea funcional, sigue estos pasos técnicos para activar los servicios:

## 1. Activación de los Servicios

Ejecuta el siguiente comando en la carpeta donde se situa el proyecto para levantar los contenedores:

Bash
docker compose up -d

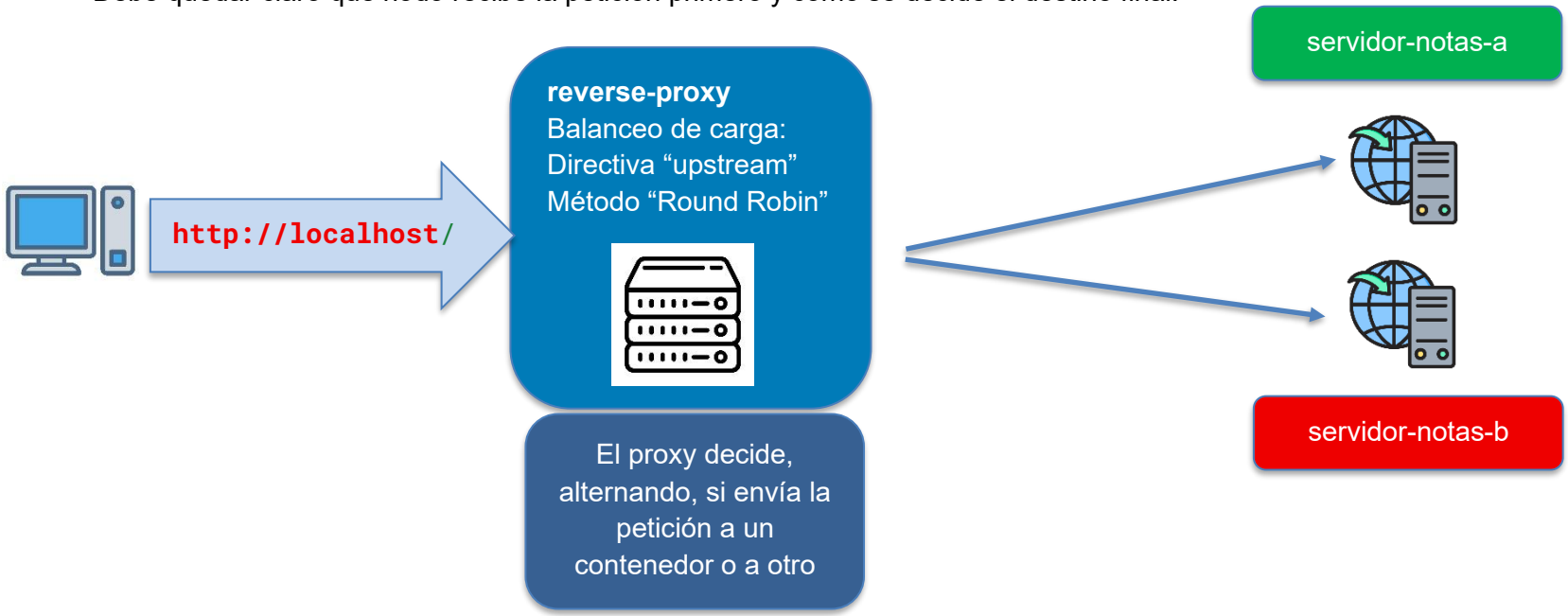


## Prueba 1: Explicación del montaje del proxy inverso y el balanceador

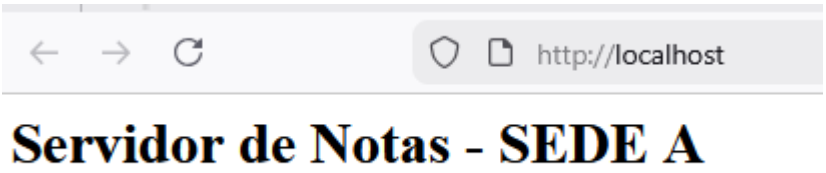
### A. Diagrama de Flujo de Tráfico

Dibuja o describe el camino que sigue una petición HTTP desde que el usuario escribe `http://localhost/` en su navegador hasta que recibe la respuesta.

- Debe quedar claro qué nodo recibe la petición primero y cómo se decide el destino final.



1. El **cliente escribe** **http://localhost/** en su navegador.
2. El **contenedor "reverse-proxy" recibe la petición**. Es el único con los puertos abiertos al exterior (servicio HTTP, puerto 80). Así que los otros no pueden recibir la orden directamente, primero debe pasar por dicho contenedor. Los tres contenedores están "unidos" al pertenecer todos a "red_interna".
3. "reverse-proxy" tiene una **configuración de balanceo de carga** para HTTP[1] que hace uso de la directiva "**upstream**", que usa el método "**Round Robin**" por defecto. Este método distribuye las peticiones a los servidores según el peso "weight" que se indique en el archivo de configuración.
4. Cuando uno de los dos contenedores que hacen de servidor de notas recibe la petición, mira en su directorio "html" y muestra el archivo "index.html".




**Servidor de Notas - SEDE A**

[1] https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/

***B. Análisis del Componente "Proxy Inverso" y "Balanceador de carga".***

*Explica la configuración del contenedor `proxy` para construir un proxy inverso y gestionar el balanceo de la carga. Debes identificar y comentar las directivas de configuración.*

La configuración de "reverse-proxy" para el balanceo de carga está en el archivo "nginx.conf".

**events { worker_connections 1024; }**

Significa que el servidor nginx puede manejar hasta 1024 conexiones simultáneas[2]. No tiene que ver con el balanceo de carga, pero es útil saberlo.

Dentro del bloque **HTTP {...}** se definirá como NGINX manejará todo lo relacionado con el protocolo de las páginas web que usen este protocolo[3].

- Primera parte: upstream.

```
upstream servidores_notas {

  server web1:80;

  server web2:80;

}
```

Usamos la directiva "upstream", donde se especificarán los servidores a los cuales se les balanceará la carga. Primero se indica un grupo ("servidores_notas") y luego, dentro de los corchetes, se indica el nombre del servidor. En este caso "web1" y "web2". Además, se especifican los puertos por donde se deben conectar.

No se indica ninguna manera por la que se deban repartir las peticiones, por lo tanto, la directiva usará el método Robin Round. Otros métodos son:

- Least Connections: se mandan las peticiones al servidor con menos conexiones activas ("least_conn;").
- IP Hash: Se envía la petición según la IP del cliente. Se usan los tres primeros octetos (IPv4) o toda la IP (IPv6) para generar un hash con el cual se decide a que servidor se manda ("ip_hash;").
- Generic Hash: El servidor al que se envía una solicitud se determina a partir de una clave definida por el usuario. Esta clave puede ser una cadena de texto, una variable o una combinación de ambas. En este caso se usa una URI:("hash $request_uri consistent;").

[2] https://medium.com/pickme-engineering-blog/nginx-the-secret-behind-handling-thousands-of-connections-with-just-a-few-workers-68c92bcd441b
[3] https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/

Ahora se define al servidor:

- “listen”, el puerto por donde escucha.
- “server_name”: nombre del servidor.

```
server {
    listen 80;
    server_name notas.iesdelgado.es;
```

Para pasar peticiones a un grupo de servidores tenemos que usar la directiva “**proxy_pass**”.
proxy_pass http://servidores_notas;

**proxy_set_header**: Es la directiva que permite mandar información al servidor que está detrás del proxy[4].

**proxy_set_header Host $host;** → Envía al backend el dominio original ($host = notas.iesdelgado.es)

**proxy_set_header X-Real-IP $remote_addr**; → Envía la IP real del cliente ($remote_addr = IP del navegador)

**proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for**; → Mantiene una lista de IPs por las que ha pasado la petición

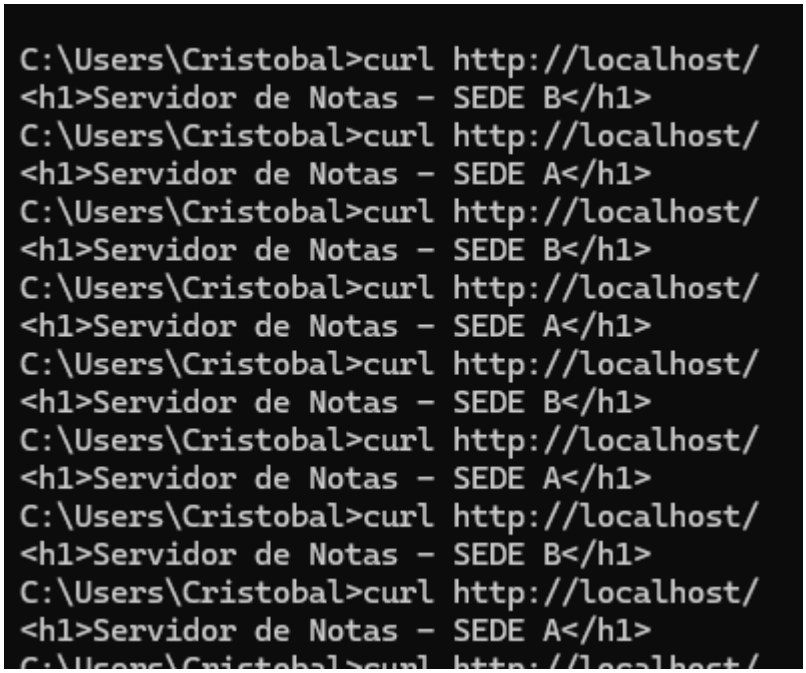
```
    location / {
        proxy_pass http://servidores_notas;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

[4] https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_set_header

## Prueba 2: Verificación del Balanceo de Carga (Round Robin)

El proxy está configurado para repartir el tráfico entre `web1` y `web2`.

- **Cómo hacerlo:** Abre una terminal y usa `curl`, o abre el navegador en modo incógnito (para evitar la caché).
- **Comando:** `curl http://localhost/`
- **Resultado esperado:** Ejecuta el comando varias veces. Deberías ver cómo una vez responde la **SEDE A** y otra la **SEDE B**.



## Prueba 3: Tolerancia a Fallos (Alta Disponibilidad)

¿Qué pasa si uno de los servidores de notas se rompe? El Proxy Inverso debería ser capaz de detectarlo.

1. **Detén un servidor:** `docker stop servidor-notas-a`

```
D:\2°_ASIR\Seguridad\Tema 06\Actividad 4 - Proxy Inverso\proxyinverso>docker stop servidor-notas-a
servidor-notas-a
```

2. **Prueba de nuevo:** Refresca el navegador o haz `curl http://localhost/`
3. **Resultado esperado:** El sitio sigue funcionando, pero **solo responde la SEDE B**. El Proxy Inverso ha detectado que `web1` está caído y redirige todo el tráfico al servidor que queda vivo. No hay error 404 ni 502 para el usuario.

```
C:\Users\Cristobal>curl http://localhost/
<h1>Servidor de Notas - SEDE B</h1>
C:\Users\Cristobal>curl http://localhost/
<h1>Servidor de Notas - SEDE B</h1>
C:\Users\Cristobal>curl http://localhost/
<h1>Servidor de Notas - SEDE B</h1>
C:\Users\Cristobal>curl http://localhost/
<h1>Servidor de Notas - SEDE B</h1>
C:\Users\Cristobal>curl http://localhost/
```

4. **Restaura:** `docker start servidor-notas-a`. Al cabo de unos segundos, el balanceo volverá a la normalidad.

```
D:\2°_ASIR\Seguridad\Tema 06\Actividad 4 - Proxy Inverso\proxyinverso>docker start servidor-notas-a
servidor-notas-a
```

```
C:\Users\Cristobal>curl http://localhost/
<h1>Servidor de Notas - SEDE A</h1>
C:\Users\Cristobal>curl http://localhost/
<h1>Servidor de Notas - SEDE B</h1>
C:\Users\Cristobal>curl http://localhost/
<h1>Servidor de Notas - SEDE A</h1>
C:\Users\Cristobal>curl http://localhost/
<h1>Servidor de Notas - SEDE B</h1>
C:\Users\Cristobal>curl http://localhost/
<h1>Servidor de Notas - SEDE A</h1>
C:\Users\Cristobal>curl http://localhost/
<h1>Servidor de Notas - SEDE B</h1>
```

### Prueba 4: Aislamiento y Seguridad (Ocultación del Backend)

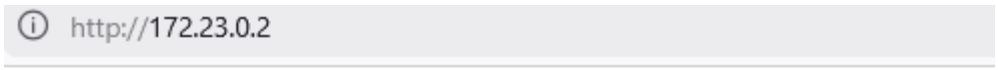
Un objetivo clave es que el servidor final esté protegido y no sea accesible

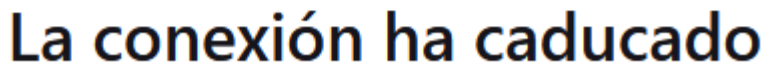
- **La prueba:** Intenta acceder a los servidores web por sus IPs internas o puertos desde tu navegador fuera de Docker.

docker inspect servidor-notas-a | findstr IPAddress → 172.23.0.2

docker inspect servidor-notas-b | findstr IPAddress → 172.23.0.4

docker inspect reverse-proxy | findstr IPAddress → 172.23.0.3




Claro, las IPs que se han mostrado antes son internas. La única conexión con el exterior es el puerto 80 del proxy inverso.

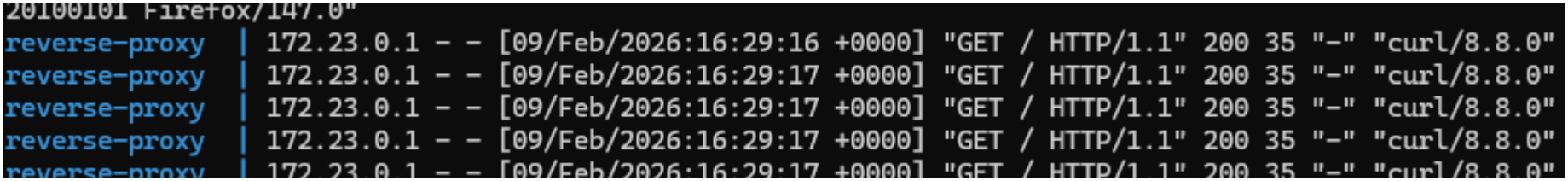
- **Resultado esperado:** No podrás. Si miras el `docker-compose.yml`, verás que **solo el proxy** tiene mapeado el puerto `:80` hacia afuera. Los servidores `web1` y `web2` son "invisibles" desde tu red real; solo el proxy puede hablar con ellos. Esto demuestra que el proxy actúa como **escudo de seguridad**.

## Prueba 5: Inspección de Cabeceras (Logs en tiempo real)

Para ver "las tripas" de lo que está pasando, observa los logs del proxy mientras navegas.

- **Comando:** `docker compose logs -f proxy`
- Explica los aspectos más importantes de lo que ves

Cuando usamos “curl”:




Se indica la IP de origen, 172.23.0.1, que es el Gateway de la red interna. Por donde le llegan al proxy inverso todos los mensajes.

El código “200” en HTTP, que indica que todo ha salido bien[5].

Y “curl/8.8.0”, que indica que hemos usado “curl”, es decir, la terminal.

[5] https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Status/200

Cuando usamos el navegador:


Parecido al anterior, solo que en este caso se usa el código “404”[6] de error, porque no encuentra el favicon.ico. En las carpetas de HTML de ambos servidores solo está el archivo index.html.

El segundo código “304”[7], el cual indica que el recurso no ha sido modificado desde la última vez que se solicitó y por lo tanto se está usando la caché.

Después se indican las características del navegador y el sistema operativo del cliente.

[6] https://support.google.com/webmasters/answer/2445990?hl=es
[7] https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Status/304