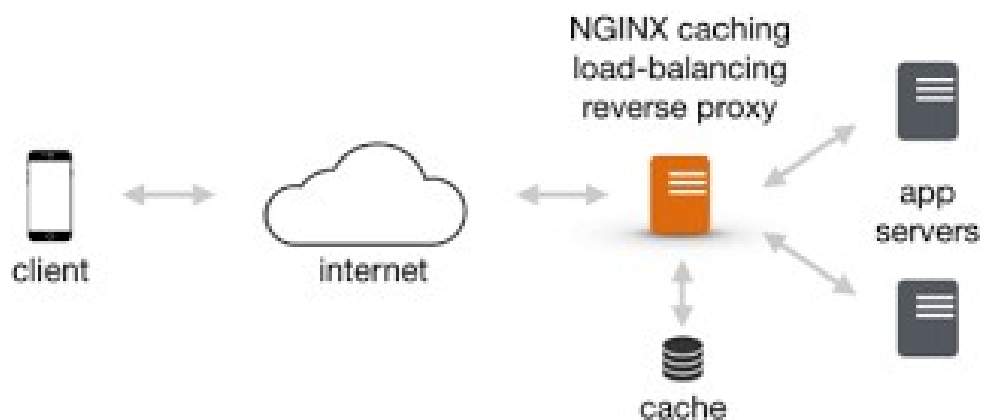


ACTIVIDAD GRUPALES TEMA 7

Docker Compose – Reverse proxy con node y nginx



Version 1.0

Veision: V1 Miguel

Fecha 15/02/24

Revisor	Estado	Fecha	Notas
Miguel Soler Bataller	Revisada	15/02(24	Configuración de nginx como reverse proxy con Docker Compose.
Hasna Zalmat	Revisada	15/02/24	Actualización detalles.
	No iniciada		

ÍNDICE

Descripción.....	2
Finalidad del servicio.....	2
Máquinas en las que se aloja.....	3
Paquetes usados y versión de los mismos.....	3
Servidor:.....	3
Configuración.....	4
Verificación del funcionamiento.....	10
Conectividad.....	12
Proceso y estado del servicios.....	12
Alternativas.....	13
Fuentes.....	13


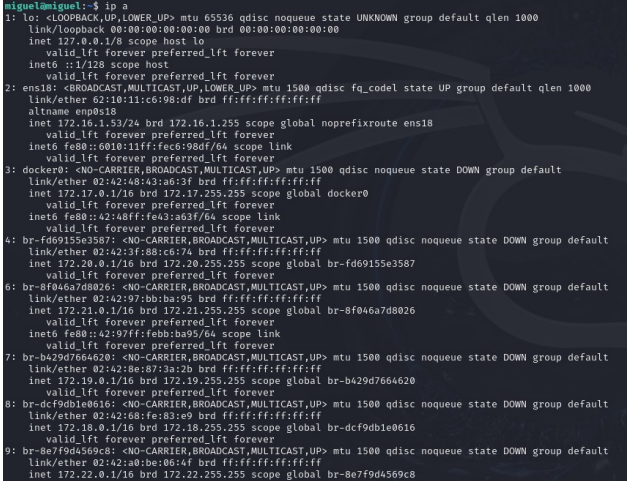
Descripción

Finalidad del servicio

Un reverse proxy con Docker Compose tiene como finalidad principal mejorar la eficiencia y seguridad del tráfico web. Actúa como intermediario entre clientes y servidores backend, distribuyendo la carga, protegiendo contra amenazas y facilitando la administración de dominios. Además, permite redirecciones, reescritura de URL, caché y escalabilidad flexible sin necesidad de gestionar SSL/TLS directamente.

Máquinas en las que se aloja

Máquina 1 (Docker)

<u>Sistema Operativo</u>	Ubuntu Desktop 22.04
<u>Hostname</u>	
<u>Nucleos</u>	4 cores
<u>RAM</u>	4 GB
<u>Configuración RED</u>	
<u>Adaptador de red</u>	DMZ

Paquetes usados y versión de los mismos

Servidor:

Docker

```
miguel@miguel:~$ docker -v
Docker version 25.0.3, build 4debf41
```

Configuración

Servidor:

-Para empezar, he hecho en local una carpeta del proyecto totalmente personalizada y hecha por mi. Se compone del siguiente arbol

```
root@miguel:/ED2Reverse# tree -d
.
├── servernginx
└── servernode
```

-He creado dos carpetas para los dos containers diferentes que crearemos, en una tenemos el servidor nginx, que será el que haga de reverse proxy y añada encabezados a el servidor node que será quien sirva la pagina web.

-Dentro del servernginx, tenemos dos archivos, el Dockerfile, las instrucciones para la creación del container de nginx, y el default.conf.template, que será el que se inyecte en la carpeta template, que posteriormente cogerá nginx para sustituirlo en /etc/nginx/conf.d/default.conf. Por eso tenemos que tenerlos ya configurados.

```
root@miguel:/ED2Reverse/servernginx# ls
default.conf.template Dockerfile
```

-Este es el Dockerfile

```
root@miguel:/ED2Reverse/servernginx# cat Dockerfile
FROM nginx:1.18.0-alpine
WORKDIR /etc/nginx/templates
COPY default.conf.template /etc/nginx/templates/default.conf.template
```

- Como vemos, estamos indicándole en el FROM, que su imagen base para construir la nueva imagen es nginx con un sistema base de alpine, en concreto la versión que aparece.

En el Workdir, establecemos el directorio de trabajo dentro del contenedor, esto nos indica que los comandos que vayan después de esté todos se ejecutarán desde ese directorio.

Y el COPY, le estamos diciendo que copie el archivo del local que tenemos en concreto el default.conf.template, a la ruta que le pasamos que es /etc/nginx/templates/default.conf.templates. Que es lo que he explicado anteriormente.

-Y este es el default.conf.template

```
root@miguel:/ED2Reverse/servernginx# cat default.conf.template
server {
    listen ${NGINX_PORT};
    server_name ${NGINX_HOST};
    location / {
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_pass http://${SERVER_HOST}:${SERVER_PORT};
        proxy_http_version 1.1;
    }
}
```

(Aclaración- He usado variables de entorno para configurar varios parametros tanto para el archivo composer, como el archivo este el default, como el index.js de node.)

En este caso tenemos lo siguiente:

-Listen \${NGINX_PORT} : Indica al servidor Nginx escuchar en el puerto especificado por la variable de entorno NGINX_PORT. Este es el puerto en el que Nginx espera las conexiones entrantes.

-server_name \${NGINX_HOST}: Define el nombre del servidor al que Nginx responderá. Esto se basa en la variable de entorno NGINX_HOST.

-location / : Este bloque define cómo Nginx manejará las solicitudes que coincidan con la ubicación /.

-proxy_set_header: Establece encabezados para las solicitudes proxy.

-X-Forwarded-For \$proxy_add_x_forwarded_for: Agrega la dirección IP del cliente original al encabezado X-Forwarded-For.

-Host \$host: Pasa el encabezado Host sin cambios. Esto ayuda a mantener el nombre de host original.

-Upgrade \$http_upgrade; y Connection "Upgrade": Son configuraciones específicas para admitir protocolos de actualización como WebSocket. Esto es útil para aplicaciones en tiempo real.

-proxy_pass http://\${SERVER_HOST}:\${SERVER_PORT}: Especifica la dirección del servidor de destino al que se reenviarán las solicitudes. Esta dirección se construye utilizando las variables de entorno SERVER_HOST y SERVER_PORT.

-proxy_http_version 1.1: Indica la versión del protocolo HTTP que se utilizará en las solicitudes proxy. En este caso, se utiliza HTTP/1.1.

-Este es el archivo .env que contiene las variables de entorno o environment para que todos los archivos reciban.

```
root@miguel:/ED2Reverse# cat .env +
SERVER_PORT=8888
NGINX_PORT=9999
SERVER_HOST=ed2Node
NGINX_HOST=ed2Nginx
```

-Ahora nos iremos a la carpeta servernode, que almacena todas las dependencias para servir la web. Como vemos en la imagen, contiene un Dockerfile personalizado para crear la imagen y node_modules, package.json y package-lock.json que hemos inicializando un proyecto con npm usando ECMAScript 6 (es6).

```
root@miguel:/ED2Reverse/servernode# ls
Dockerfile  node_modules  package.json  package-lock.json  src
```

-Este es el archivo Dockerfile para crear la imagen

```
root@miguel:/ED2Reverse/servernode# cat Dockerfile
FROM node:16-alpine
WORKDIR /server
COPY package.json .
RUN npm i
COPY . .
CMD ["npm", "start"]
```

Como vemos tenemos:

FROM, que nos indica que la imagen base es de node con un sistema base de alpine versión 16

WORKDIR, que nos indica que dentro del contenedor estaremos trabajando en el directorio /server. Y todas las instrucciones que ejecutemos a posterior se harán en ese directorio de trabajo.

COPY, que copiará el archivo local package.json dentro del directorio de trabajo del contenedor.

RUN, que instalará las dependencias del proyecto en el contenedor

Copy, Copiará todos los archivos de la carpeta local actual al contenedor

CMD, Ejecutará npm start al inicio del contenedor, que cuando os muestre ahora el package.json es un script creado que lo que hará será iniciar el servidor node fijandose en /src/index.js

-Ahora pasaré a explicar el package.json

```
root@miguel:/ED2Reverse/servernode# cat package.json
{
  "name": "",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "license": "AGPL-version-3.0",
  "private": false,
  "engines": {
    "node": "≥ 14.0.0",
    "npm": "≥ 6.0.0"
  },
  "homepage": "",
  "repository": {
    "type": "git",
    "url": ""
  },
  "bugs": "",
  "keywords": [],
  "author": {
    "name": "",
    "email": "",
    "url": ""
  },
  "contributors": [],
  "scripts": {
    "start": "node ./src/index.js",
    "dev": "",
    "test": ""
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

En este caso, como ya he explicado en el paso anterior, lo único que he cambiado aquí fue dentro de scripts, crear el script start, que ejecutará el node cogiendo el index.js de dentro de src.

```
"scripts": {
  "start": "node ./src/index.js",
  "dev": "",
  "test": ""
}
```

-Para seguir explicando, iremos a la carpeta src, la cual contiene el index.js

```
root@miguel:/ED2Reverse/servernode/src# cat index.js
import express from "express";
const app = express();

app.get('/', (req, res) => {
  const headers = req.headers;
  res.status(200).send(headers);
});

app.listen(process.env.SERVER_PORT, () => {
  console.log('Express listening on port 8888');
});
```

En este caso, estamos creando una app en Node que utiliza el framework Express, para crear un servidor HTTP simple.

import express from "express" : Esto lo que hará será importar el módulo express y crear una instancia de la aplicación app

app.get('/', (req, res) => { : Define una ruta para manejar solicitudes HTTP GET a la raíz ('/'). Cuando se recibe una solicitud GET en la raíz, se ejecuta la función de devolución de llamada con los objetos req (solicitud) y res (respuesta).

const headers = req.headers : Obtiene los encabezados de la solicitud HTTP y los almacena en la variable headers.

res.status(200).send(headers) : Devuelve una respuesta al cliente con un código de estado 200 (OK) y envía los encabezados de la solicitud como cuerpo de la respuesta.

app.listen(process.env.SERVER_PORT, () => { ... }) : Inicia el servidor y lo hace escuchar en el puerto especificado por la variable de entorno SERVER_PORT. La función de devolución de llamada se ejecuta una vez que el servidor ha comenzado a escuchar y muestra un mensaje en la consola.

-Y por último explicar el archivo docker-compose.yaml que será el encargado de crear las imágenes.

```
root@miguel:/ED2Reverse# cat docker-compose.yaml
version: "3.8"
services:
  servernode:
    image: servernode
    build:
      context: ./servernode
      dockerfile: Dockerfile
    container_name: ${SERVER_HOST}
    env_file: .env
    ports:
      - ${SERVER_PORT}:${SERVER_PORT}
  servernginx:
    image: servernginx
    restart: always
    build:
      context: ./servernginx
      dockerfile: Dockerfile
    container_name: ${NGINX_HOST}
    env_file: .env
    ports:
      - ${NGINX_PORT}:${NGINX_PORT}
```


version: "3.8": Especifica la versión de la sintaxis de Docker Compose que estás utilizando.

Servicio servernode:

image: servernode: Especifica la imagen a utilizar para el servicio servernode. Se espera que esta imagen se construya a partir del contexto ./servernode utilizando el archivo Dockerfile en ese contexto.

build: ...: Define la configuración de construcción para la imagen del servicio servernode.

context: ./servernode: Especifica el contexto de construcción, que es el directorio donde se encuentran los archivos para construir la imagen.

dockerfile: Dockerfile: Especifica el archivo Dockerfile a utilizar para construir la imagen.

container_name: \${SERVER_HOST}: Define el nombre del contenedor como la variable de entorno SERVER_HOST.

env_file: .env: Lee variables de entorno desde el archivo .env.

ports: ...: Mapea los puertos del contenedor al host, utilizando las variables de entorno SERVER_PORT para ambos extremos.

Servicio servernginx:

image: servernginx: Especifica la imagen a utilizar para el servicio servernginx. Se espera que esta imagen se construya a partir del contexto ./servernginx utilizando el archivo Dockerfile en ese contexto.

restart: always: Indica que el contenedor debe reiniciarse siempre que se detenga, a menos que se haya detenido explícitamente por el usuario.

build: ...: Define la configuración de construcción para la imagen del servicio servernginx.

context: ./servernginx: Especifica el contexto de construcción, que es el directorio donde se encuentran los archivos para construir la imagen.

dockerfile: Dockerfile: Especifica el archivo Dockerfile a utilizar para construir la imagen.

container_name: \${NGINX_HOST}: Define el nombre del contenedor como la variable de entorno NGINX_HOST.

env_file: .env: Lee variables de entorno desde el archivo .env.

ports: ...: Mapea los puertos del contenedor al host, utilizando las variables de entorno NGINX_PORT para ambos extremos.

Verificación del funcionamiento

Comprobaciones:

Servidor:

-Para poder verificar su funcionamiento, ejecutaremos el composer con la siguiente orden(tenemos que estar en la carpeta que se ubique el archivo docker.compose.yaml):

```
root@miguel:/ED2Reverse# docker compose up -d
[+] Running 2/2
 ✓ Container ed2Nginx Started 0.8s
 ✓ Container ed2Node Started 0.8s
```

-Y ahora con un docker ps, observaremos que nos ha creado los containers

```
root@miguel:/ED2Reverse# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
36ba80d4fe5a   servernode    "docker-entrypoint.s..." About a minute ago Up About a minute 0.0.0.0:8888→8888/tcp, :::8888→8888/tcp
a1087bf3b5bc   servernginx    "/docker-entrypoint..." About a minute ago Up About a minute 80/tcp, 0.0.0.0:9999→9999/tcp, :::9999→9999/tcp
Names         ed2Node      ed2Nginx
```

-Estos son sus respectivos logs:

```
root@miguel:/ED2Reverse# docker logs -f ed2Node

> start
> node ./src/index.js

Express listening on port ${process.env.SERVER_PORT}
```

```
root@miguel:/ED2Reverse# docker logs -f ed2Nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
20-envsubst-on-templates.sh: Running envsubst on /etc/nginx/templates/default.conf.template to /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Configuration complete; ready for start up
```

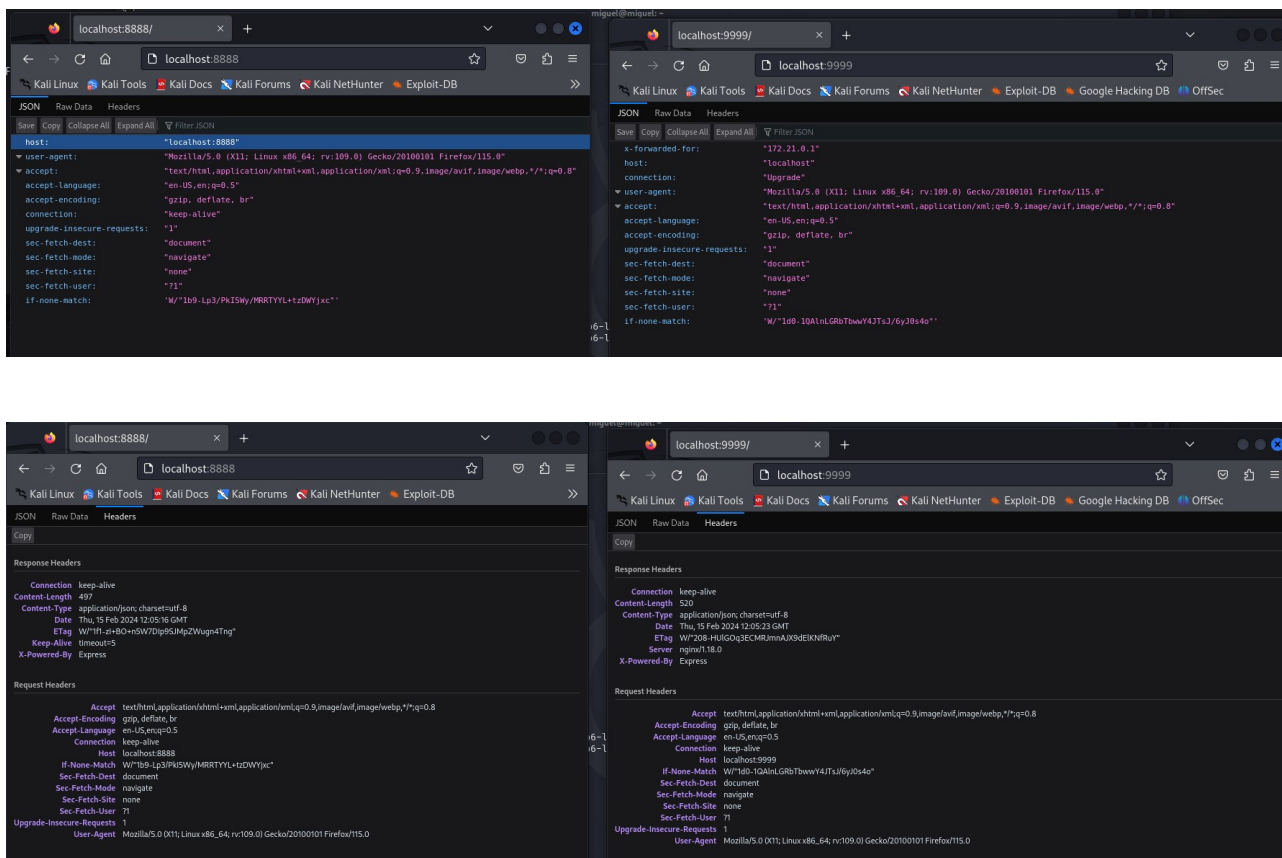
-Como hemos visto en el docker ps, el nginx está funcionando en el puerto 9999 del contenedor redirigido localmente al 9999, y el Node, en el 8888 de contenedor y 8888 localmente.

-Para poder hacer túneles y así verlo desde el navegador local, ejecutaremos el siguiente comando que es un ssh para poder redirigir el localhost:9999 y 8888 de la máquina Docker, a el localhost:9999 y 8888 de la máquina local.

```
# ssh -L 9999:localhost:9999 -p 52360 miguel@pro.ausiasmarch.es

# ssh -L 8888:localhost:8888 -p 52360 miguel@pro.ausiasmarch.es
```

-Y estas son las páginas que sirven, son muy básicas, pero gracias a nginx, podemos ver que el proxy inverso funciona porque le añade las cabeceras anteriormente descritas y tiene un mayor tamaño.



Como vemos, los headers de el localhost:8888 (página que sirve node directamente), son mucho mas simple y no tienen los headers que hemos añadido al nginx, en especial el forwarder. Pero si nos vamos al localhost:9999, vemos que nos pone el forwarded que nos indica de que dirección ip del cliente original.

Conectividad

-Meternos de maquina Docker al contenedor ed2Node

```
root@miguel:/ED2Reverse# docker exec -it ed2Node sh
/server # ls
Dockerfile      node_modules    package-lock.json  package.json      src
```

-Meternos de maquina Docker al contenedor ed2Nginx

```
root@miguel:/ED2Reverse# docker exec -it ed2Nginx sh
/etc/nginx/templates # ls
default.conf.template
```

Proceso y estado del servicios

Local

-Como vemos está cogiendo la redirección de puertos desde Docker de 8888 y 9999

```
netstat -tulnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:8888          0.0.0.0:*               LISTEN      28328/ssh
tcp        0      0 127.0.0.1:9999          0.0.0.0:*               LISTEN      28132/ssh
tcp6       0      0 :::1:9999                :::*                    LISTEN      28132/ssh
tcp6       0      0 :::1:8888                :::*                    LISTEN      28328/ssh
```

Docker Máquina

-Como vemos está cogiendo la redirección de puertos desde los contenedores que le hemos indicado en el composer

```
miguel@miguel:~$ netstat -tulnp
(No todos los procesos pueden ser identificados, no hay información de propiedad del proceso
no se mostrarán, necesita ser superusuario para verlos todos.)
Conexiones activas de Internet (solo servidores)
Proto  Recib  Enviad  Dirección local      Dirección remota      Estado      PID/Program name
tcp    0      0 0.0.0.0:22            0.0.0.0:*             ESCUCHAR   -
tcp    0      0 0.0.0.0:8888          0.0.0.0:*             ESCUCHAR   -
tcp    0      0 127.0.0.1:631         0.0.0.0:*             ESCUCHAR   -
tcp    0      0 127.0.0.53:53         0.0.0.0:*             ESCUCHAR   -
tcp    0      0 0.0.0.0:9999          0.0.0.0:*             ESCUCHAR   -
tcp6   0      0 :::22                 :::*                   ESCUCHAR   -
tcp6   0      0 :::8888                :::*                   ESCUCHAR   -
tcp6   0      0 :::1:631               :::*                   ESCUCHAR   -
tcp6   0      0 :::9999                :::*                   ESCUCHAR   -
udp    0      0 0.0.0.0:5353          0.0.0.0:*             -          -
udp    0      0 127.0.0.53:53         0.0.0.0:*             -          -
udp    0      0 0.0.0.0:631           0.0.0.0:*             -          -
udp    0      0 0.0.0.0:56326         0.0.0.0:*             -          -
udp6   0      0 :::5353                :::*                   -          -
udp6   0      0 :::46651               :::*                   -          -
```

Container ed2Node

-Como vemos está sirviendo el node en el puerto 8888

```
/server # netstat -tulnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.11:34375       0.0.0.0:*               LISTEN      -
tcp        0      0 :::8888                :::*                    LISTEN      18/node
udp        0      0 127.0.0.11:41369       0.0.0.0:*               -          -
```

Container ed2Nginx

-Como vemos está sirviendo el nginx reverse proxy en el puerto 9999

```
/etc/nginx/templates # netstat -tulnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.11:34219       0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:9999           0.0.0.0:*               LISTEN      1/nginx: master pro
udp        0      0 127.0.0.11:34892       0.0.0.0:*               -          -
```

Alternativas

Una alternativa podría ser Traefik (proxy inverso y balancer de carga), Caddy (Servidor web con funcionalidades de proxy inverso), HAProxy (software de balanceador de carga y proxy inverso de alto rendimiento).

Fuentes

- Autoconocimientos
- PDF de clase
- Dockerhub
- Github