TP3: Autour de Nginx

Ifrim Vasile-Alexandru
M2 SSI

L'objectif de ce TP est la mise en place d'une infrastructure complète de load balancing avec Nginx et des serveurs Express.js dans un environnement Docker.

1. Configurez un serveur Nginx en tant que load balancer pour répartir les requêtes vers un backend constitué de trois serveurs Express. Chaque serveur Node.js fonctionne sur les ports suivants:
   — localhost:3001
   — localhost:3002
   — localhost:3003
   Le load balancer doit utiliser la stratégie de répartition par défaut de Nginx.

To configure a Nginx server to balance the load of 3 backend servers, we need to define how HTTP traffic is handled by it. In the `nginx.conf` file, in the http block we first define and name the group of servers where we "upstream" the traffic, their addresses and ports. The addresses "express1/2/3" are set later in the docker-compose configuration file through the given context name (because each backend server is located in a subdirectory called "express1/2/3", for docker-compose to be able to build, it must be instructed the correct context/path to each service). By default, Nginx uses a round-robin strategy, meaning it will distribute incoming requests sequentially across these servers.

Next, the server block configures how Nginx handles incoming client requests: listen on port 80 and proxy all requests for the root path to the backend_servers group using HTTP/1.1.

```
# load-balancer for a group of backend servers
# use default round-robin strategy
events {}
http {
    upstream backend_servers {
        server express1:3001;
        server express2:3002;
        server express3:3003;
    }
    server {
        listen 80;
        location / {
            proxy_pass http://backend_servers;
            proxy_http_version 1.1;
        }
    }
}
```

The `Dockerfile` for nginx server:

```
FROM nginx:latest
COPY nginx.conf /etc/nginx/nginx.conf
```

2. Écrivez une application Express qui écoute sur le port 3001 et retourne un message indiquant le serveur qui traite la requête.

We will start by creating the first Express web server. For the other 2, there are only small changes like port number and file names.

Express is a web framework for Node.js that makes launching web servers much easier and cleaner. So we will be running some JavaScript in a container enabled with the Node.js runtime environment.

Below is the `server1.js` file containing our web server logic. First it imports the Express library and creates an Express app instance. Then it defines a route to the root URL on which the app will listen for GET requests, and how it will respond to them. Finally, we start the server to listen on its respective port number.

```
const express = require('express');
const app = express();
const port = 3001;

app.get('/', (req, res) => {
        res.send('Hello from server ' + port);
});

app.listen(port, () => {
        console.log('Server running on port ' + port);
});
```

First express server js; change port number for the other backend servers

The `Dockerfile` for first backend server:

```
FROM node:14
WORKDIR /app
COPY server1.js .
RUN npm install express
CMD ["node", "server1.js"]
```

3. Comment démarrer plusieurs serveurs Express sur les ports 3002 et 3003 pour compléter la configuration de load balancing Nginx?

For the other Express servers we just copy and modify the javascript app and the Dockerfile done in the exercise above. Each server will be running in a separate container, all being coordinated through docker-compose. In the root of our project we will be later creating (see ex. 5) a `docker-compose.yaml` file where we define the multi-container setup.

4. Comment configurer Nginx pour activer le protocole HTTP/2 uniquement entre le client et le serveur Nginx, tout en gardant la communication entre Nginx et les serveurs backend Express en HTTP/1.1?

A prerequisite is that nginx must be compiled with HTTP/2 support. To verify:

```
nginx -V 2>&1 | grep -- '--with-http_v2_module'
```

To naively enable HTTP/2 just over TCP (also known as h2c, http2 over cleartext), only on the client-facing side of Nginx, the server block in `nginx.conf` should look like this:

```
server {
        listen 80 http2;
        location / {
            proxy_pass http://backend_servers;
            proxy_http_version 1.1;
            add_header X-Client-Protocol $http2; # optional, for checking
        }
}
```

The last directive tells Nginx to add the following header if HTTP/2 is used; if it's not, it will be missing from the response headers. To test this, we can do

```
curl -I –http2-prior-knowledge localhost
```

The `--http2-prior-knowledge` flag in curl explicitly initiates an HTTP/2 connection from the start, without the usual HTTP/1.1 upgrade negotiation, but by default, curl will try to connect using HTTP/1.1, and nginx may respond accordingly.

```
alex@alex-VMware:~/Reseau/TP3$ curl -I --http2-prior-knowledge http://localhost
HTTP/2 200
server: nginx/1.27.2
date: Sat, 02 Nov 2024 19:38:09 GMT
content-type: text/html; charset=utf-8
content-length: 22
x-powered-by: Express
etag: W/"16-T+vjs2U7AzvNPrnjOK4f/d81Ehc"
x-client-protocol: h2c
```

To enable HTTP/2 over TLS we have to add SSL certificates and switch to HTTPS. We set up a self-signed certificate and key with the help of openssl library. In the nginx `Dockerfile`, we'll have to copy them to the container, and in `nginx.conf` we'll modify to listen on port 443 with ssl and http2, indicating the paths to our self-signed certificate and key.

```
# load-balancer for a group of backend servers
# use default round-robin strategy
events { }

http {
    upstream backend_servers {
        server express1:3001;
        server express2:3002;
        server express3:3003;
    }

    server {
        listen 443 ssl http2;
        server_name localhost;

        ssl_certificate /etc/nginx/selfsigned.crt;
        ssl_certificate_key /etc/nginx/selfsigned.key;

        location / {
            proxy_pass http://backend_servers;
            proxy_http_version 1.1;
            proxy_set_header Host $host;
            add_header X-Client-Protocol $http2;
```

nginx.conf

```
FROM nginx:latest
COPY nginx.conf /etc/nginx/nginx.conf
COPY selfsigned.crt /etc/nginx/selfsigned.crt
COPY selfsigned.key /etc/nginx/selfsigned.key
```

nginx Dockerfile

Finally, our docker-compose environment configuration will need to be indicated to map the ports 443:443 for the nginx service.

```
version: '3'

services:
  nginx:
    build:
      context: ./nginx
      dockerfile: Dockerfile
    ports:
      - "443:443"

  express1:
```

docker-compose.yaml

To test our new configuration, we build the project again and test by

```
curl -I --http2 https://localhost --insecure
```

In the `x-client-header` that we've added we can observe the value of "h2" (HTTP2 over TLS), compared to "h2c" in the previous HTTP/2 over cleartext configuration.

```
alex@alex-VMware:~/Reseau/TP3$ curl -I --http2 https://localhost --insecure
HTTP/2 200
server: nginx/1.27.2
date: Sun, 03 Nov 2024 12:52:16 GMT
content-type: text/html; charset=utf-8
content-length: 22
x-powered-by: Express
etag: W/"16-q/TOHN4JTsuIEoLeDXUQ8FArtQA"
x-client-protocol: h2
```

The `--insecure` flag bypasses SSL verification, which is necessary with a self-signed certificate

5. Comment pouvez-vous utiliser Docker pour conteneuriser le serveur Nginx ainsi que les trois serveurs Express, et orchestrer le tout avec Docker Compose?

The `docker-compose.yaml` file defines the list of services ran in individual containers. For each service we specify the Docker build context (directory) and what dockerfile to use (in that context), then we map ports from the host machine (outside) to the container (inside):

```
version: '3'
services:
  nginx:
    build:
      context: ./nginx
      dockerfile: Dockerfile
    ports:
      #- "80:80"
      - "443:443"
  express1:
    build:
      context: ./express1
      dockerfile: Dockerfile
    ports:
      - "3001:3001"
  express2:
    build:
      context: ./express2
      dockerfile: Dockerfile
    ports:
      - "3002:3002"
  express3:
    build:
      context: ./express3
      dockerfile: Dockerfile
    ports:
      - "3003:3003"
```

Finally, to spin up the application we run:

```
docker compose up --build
```

and to test load-balancing, we repeatedly curl the localhost:

```
alex@alex-VMware:~$ curl localhost
Hello from server 3001alex@alex-VMware:~$ curl localhost
Hello from server 3002alex@alex-VMware:~$ curl localhost
Hello from server 3003alex@alex-VMware:~$ curl localhost
Hello from server 3001alex@alex-VMware:~$ curl localhost
Hello from server 3002alex@alex-VMware:~$ curl localhost
Hello from server 3003alex@alex-VMware:~$ curl localhost
Hello from server 3001alex@alex-VMware:~$
```

Resources:

- https://expressjs.com/en/starter/installing.html
- https://expressjs.com/en/starter/hello-world.html
- https://sslinsights.com/how-to-enable-http2-on-nginx-web-server/