TP : Déploiement d’une Stack Web avec Docker et Load Balancer

Objectifs :
Mettre en place plusieurs conteneurs Nginx en back-end pour simuler un environnement web.
Configurer HAProxy comme load balancer.
Simuler une charge pour observer l’équilibrage de charge entre les conteneurs.

Pré-requis :
Connaissances de base en Docker et en ligne de commande.
Pas de connaissances préalables sur les load balancers ou Nginx nécessaires.

Étape 1 : Préparer l'environnement Docker

Créer le répertoire du projet :

```
mkdir -p ~/stack-web
cd ~/stack-web
```

Créer le fichier docker-compose.yml :

```
version: '3'

services:
  nginx1:
    image: nginx
    container_name: nginx1
    networks:
      - webnet
    ports:
      - "8081:80"
    volumes:
      - ./nginx1.conf:/etc/nginx/nginx.conf

  nginx2:
    image: nginx
    container_name: nginx2
    networks:
      - webnet
    ports:
      - "8082:80"
    volumes:
      - ./nginx2.conf:/etc/nginx/nginx.conf

  loadbalancer:
    image: haproxy:alpine
    container_name: loadbalancer
    networks:
      - webnet
    ports:
      - "8080:80"
    volumes:
      - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg

networks:
  webnet:
```

```
    driver: bridge
```

Étape 2 : Configurations Nginx complètes

Créer nginx1.conf :

```
events {}

http {
    server {
        listen 80;
        server_name localhost;

        location / {
            default_type text/plain;
            add_header Cache-Control "no-store";
            return 200 'Hello from nginx1';
        }
    }
}
```

Créer nginx2.conf :

```
events {}

http {
    server {
        listen 80;
        server_name localhost;

        location / {
            default_type text/plain;
            add_header Cache-Control "no-store";
            return 200 'Hello from nginx2';
        }
    }
}
```

Étape 3 : Configuration HAProxy

Créer haproxy.cfg :

```
global
    maxconn 200
    log stdout format raw local0

defaults
    mode http
    log global
    option httplog
    option dontlognull
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms
    option http-server-close
    option forwardfor
```

```
frontend http_front
    bind *:80
    default_backend http_back

backend http_back
    balance roundrobin
    server nginx1 nginx1:80 check
    server nginx2 nginx2:80 check
```

log stdout → permet de voir les requêtes dans docker logs loadbalancer.

option http-server-close → améliore l'alternance côté client.

Étape 4 : Lancer les conteneurs

```
docker-compose up -d
```

Vérifier :

```
docker ps
```

Ports exposés :

nginx1 → 8081

nginx2 → 8082

loadbalancer → 8080

Étape 5 : Tester

Tester nginx1 :

```
curl http://localhost:8081
```

→ Hello from nginx1

Tester nginx2 :

```
curl http://localhost:8082
```

→ Hello from nginx2

Tester load balancer :

```
curl http://localhost:8080
```

→ Alternance entre nginx1 et nginx2 visible dans les logs :

```
docker logs -f loadbalancer
```

Étape 6 : Simuler une charge

Installer Apache Bench :

sudo apt-get install apache2-utils

Simuler une charge :

ab -n 1000 -c 10 http://localhost:8080/

→ Les requêtes sont réparties entre nginx1 et nginx2.

Étape 7 : Vérification des logs

docker logs nginx1
docker logs nginx2
docker logs loadbalancer

→ On voit les requêtes HTTP et à quel serveur elles sont envoyées.

Conclusion :

Nginx fonctionne correctement avec configuration complète.

HAProxy distribue la charge entre nginx1 et nginx2.

Les logs sont visibles sur stdout, et l’alternance est immédiate dans la plupart des cas.

On a déployé une stack web simple avec Docker et load balancer.