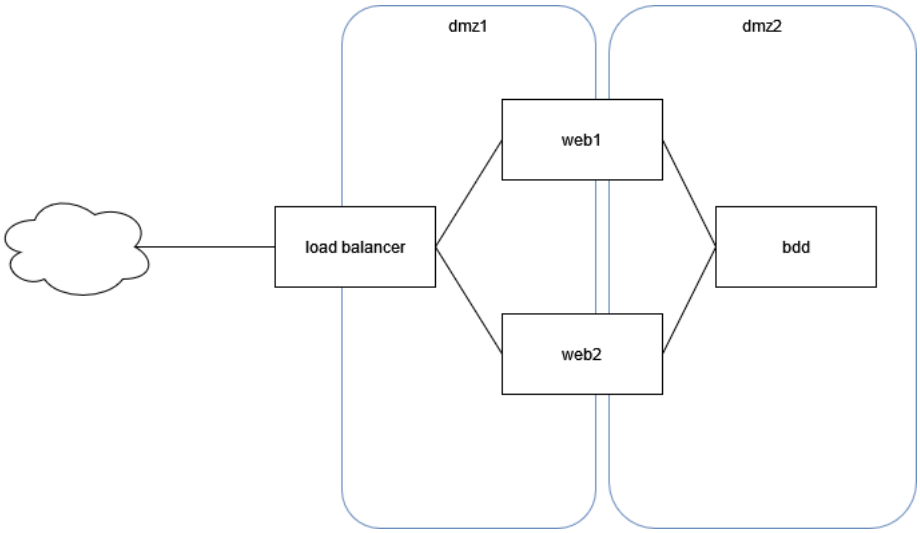
Documentation technique TP1



Sommaire

Table des matières

Initialisation des services customisés

Service Web

Le service web consiste à mettre à disposition notre application web.

Notre service web part d'une image debian 11

```
# docker/web/Dockerfile
FROM debian:bullseye
```

On viens ensuite ajouter nginx pour héberger notre application

```
# docker/web/Dockerfile
RUN apt-get update -y && \
    apt-get install -y nginx
```

Puis installer ssh et rsync (et d'autres outils utilitaires)

```
# docker/web/Dockerfile
RUN apt-get install -y openssl openssh-server rsync cron curl nano
```

Et on configure ensuite nos droit de connection en SSH

- On autorise la connexion en root
- On désactive la connexion par mot de passe
- On autorise la connexion par clef publique
- On indique notre fichier de clefs authorisés

```
# docker/web/Dockerfile
RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config
RUN sed -i 's/#PasswordAuthentication yes/PasswordAuthentication no/' /etc/ssh/sshd_config
RUN sed -i 's/#PubkeyAuthentication yes/PubkeyAuthentication yes/' /etc/ssh/sshd_config
RUN sed -i 's/#AuthorizedKeysFile/AuthorizedKeysFile/' /etc/ssh/sshd_config
```

On fini par ajouter nos clefs et nos configurations

```
# docker/web/Dockerfile
COPY key/web.key /root/.ssh/web.key
COPY key/web.key.pub /root/.ssh/web.key.pub
COPY ssh.config /root/.ssh/config
RUN cat /root/.ssh/web.key.pub >> /root/.ssh/authorized_keys
RUN chmod 600 /root/.ssh/authorized_keys
RUN chmod 600 /root/.ssh/web.ke
```

Il ne reste plus qu'a ajouter notre entrypoint et on expose nos ports

```
# docker/web/Dockerfile
COPY entrypoint.sh /root/entrypoint.sh

EXPOSE 22 80

CMD ["bash", "/root/entrypoint.sh"]
```

L'entrypoint en question consiste à créer un html différent entre nos deux services web et créer notre tâche sur le service web2 qui va copier notre dossier html de web2 à web1 toute les 5minutes, on lancera ensuite les services cron, ssh et nginx

```
# docker/web/entrypoint.sh
#!/bin/bash

rm -Rf /var/www/html/*;
echo "<html><body><h1>$NAME</h1><p>Mon super site</p></body></html>" > /var/www/html/index.html;

if [ "$NAME" = "web2" ]; then
    ssh-keyscan web1 >> /root/.ssh/known_hosts;
    echo "*/5 * * * * rsync -avz -e ssh /var/www/html/* root@web1:/var/www/html/." > /etc/cron.d/rsync;
    crontab /etc/cron.d/rsync;
    service cron start;
fi

service cron start;
service ssh start;
nginx -g "daemon off;"
tail -f /dev/null;
```

On n'oubliera pas d'ajouter nos clef **web.key** et **web.key.pub** dans notre dossier key

Service Load balancer

Le service load balancer a pour but de partager les requêtes de nos utilisateurs entre nos plusieurs services web (web1 et web2)

Notre service load balancer part d'une image nginx

```
# docker/loadbalancer/Dockerfile
FROM nginx:latest
```

Puis on installe openssl, on installe notre configuration nginx et on créer un certificat auto signé

```
# docker/loadbalancer/Dockerfile
RUN apt-get update -y && apt-get install -y openssl

COPY conf/loadbalancer.conf /etc/nginx/conf.d/loadbalancer.conf
RUN rm /etc/nginx/conf.d/default.conf

run openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/conf.d/loadbalancer.key -out /etc/nginx/conf.d/loadbalancer
```

Notre configuration nginx permet de rediriger nos requêtes vers le port 443 pour une connexions via https

```
# docker/loadbalancer/conf/loadbalancer.conf
upstream backend {
    least_conn;

    server web1:80;
    server web2:80;
}

server {
    listen 80;
    server_name localhost;

    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name localhost;

    ssl_certificate /etc/nginx/conf.d/loadbalancer.crt;
    ssl_certificate_key /etc/nginx/conf.d/loadbalancer.key;

    location / {
        proxy_pass http://backend;
    }
}
```

On fini par exposer nos port 80 et 443 et on lance nginx

```
# docker/loadbalancer/Dockerfile
EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]
```

Docker compose

Notre docker-compose.yml permet de créer notre infrastructure

Réseau

Comme indiqué sur notre schéma, notre infrastructure est composé de 3 réseau

- default ⇒ Notre connexion classique vers internet
- dmz1 ⇒ contient nos services web et loadbalancer
- dmz2 ⇒ contient notre service loadbalancer et la base de données

On indiquera aussi un sous réseau pour nos dmz

- dmz1 ⇒ 192.168.10.0/24
- dmz2 ⇒ 192.168.20.0/24

```
# docker-compose.yml
networks:
  dmz1:
    name: dmz1
    driver: bridge
    external: false
    ipam:
      config:
        - subnet: 192.168.10.0/24
```

```
  dmz2:
    name: dmz2
    driver: bridge
    external: false
    ipam:
      config:
        - subnet: 192.168.20.0/24
  default:
```

Services

On instancie ici nos services

- web1 ⇒ service web n°1
- web2 ⇒ service web n°2
- loadbalancer ⇒ service pour le load balancing
- bdd ⇒ service de base de données

Chaque service a un nom de container et leurs réseaux et leurs ip, ainsi qu'une option build ou image afin d'indiquer soit un Dockerfile soit une image de docker hub.

Nos services web ont aussi une variable d'environnement afin de les distinguer une sur la machine.

Notre service bdd aura aussi ces variables pour les connexions à la base de données, ainsi qu'un volume sur docker/bdd qui permettra de sauvegarder notre base de données sur notre machine hôte.

```
# docker-compose.yml
services:
  web1:
    container_name: web1
    build: docker/web
    networks:
      dmz1:
        ipv4_address: 192.168.10.11
      dmz2:
        ipv4_address: 192.168.20.11
    environment:
      - NAME=web1

  web2:
    container_name: web2
    build: docker/web
    networks:
      dmz1:
        ipv4_address: 192.168.10.12
      dmz2:
        ipv4_address: 192.168.20.12
    environment:
      - NAME=web2

  loadbalancer:
    container_name: loadbalancer
    build: docker/loadbalancer
    ports:
      - "80:80"
      - "443:443"
    networks:
      dmz1:
        ipv4_address: 192.168.10.10
      default:

  bdd:
    container_name: bdd
    image: mariadb:latest
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: test
      MYSQL_USER: test
      MYSQL_PASSWORD: test
    volumes:
      - ./docker/bdd:/var/lib/mysql
    networks:
      dmz2:
        ipv4_address: 192.168.20.5
```