Installer WordPress via Docker

SOMMAIRE

Matthieu MARQUETTE
Omar SY
Docker

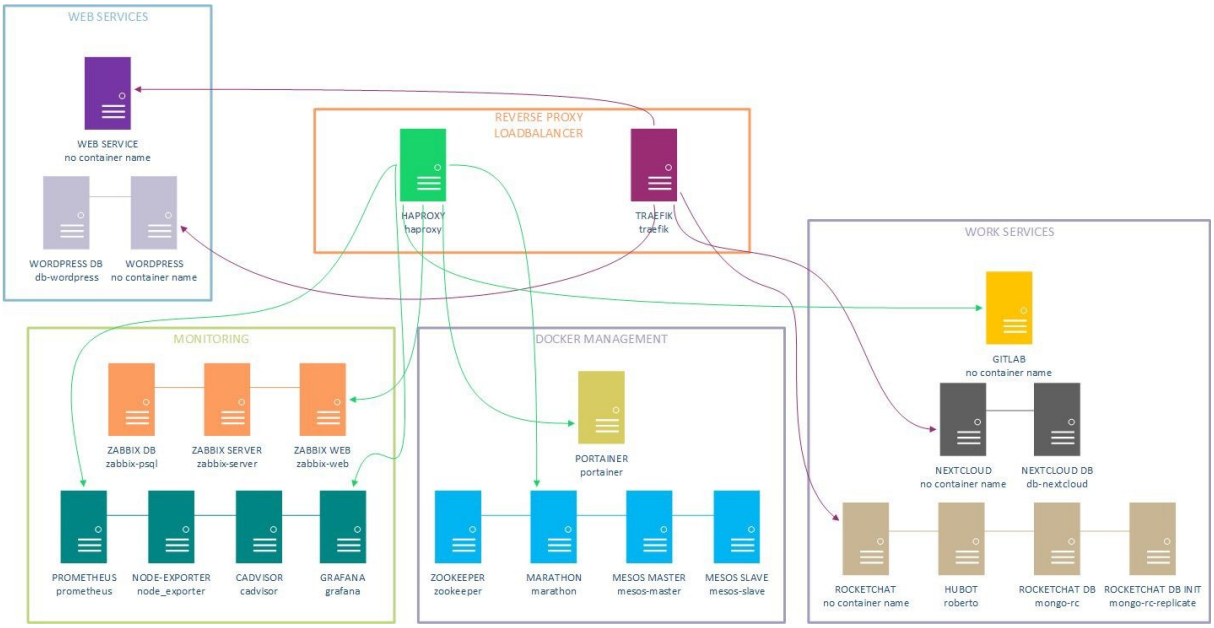
Introduction

Le but de ce tp est de faire fonctionner le cms Wordpress sous docker avec une vm Ubuntu. Pour se faire, nous allons donc créer un premier container pour pour installer WordPress, un autre pour sa base de données et enfin un dernier pour PhpMyAdmin. Il nous sera également possible de créer un autre container pour installer Portainer qui est un petit utilitaire graphique Docker.

Il va donc falloir lier les deux containers (base de données et le cms) afin de faire fonctionner correctement WordPress.

Il nous est également possible d'installer **Nginx** et faire une redirection (« **127.0.0.1** » ou « **localhost** » vers « **wordpress-docker.co** ») par exemple.

Deux méthodes sont possibles pour créer des container (ligne de commande ou docker-compose). Un peu plus dans cette procédure, je montre comment utiliser docker-compose.



Matthieu MARQUETTE
Omar SY
Docker

Création d'un container maria dB

```
root@ubuntu:~# docker run -e MYSQL_ROOT_PASSWORD=Passw0rd -e MYSQL_USER=user -e
 MYSQL_PASSWORD=Passw0rd -e MYSQL_DATABASE=wordpressdb -v /root/wordpress/datab
ase:/var/lib/mysql --name wordpressdb -d mariadb
Unable to find image 'mariadb:latest' locally
latest: Pulling from library/mariadb
5667fdb72017: Pull complete
d83811f270d5: Pull complete
ee671aafb583: Pull complete
7fc152dfb3a6: Pull complete
9f669c535a8b: Pull complete
a6de1092ee4e: Pull complete
ee37a2c88dd9: Pull complete
d927a3dd356c: Pull complete
d83c9d39c64f: Pull complete
1b0644883413: Pull complete
09a38adc2558: Pull complete
3c853415b952: Pull complete
2690cf0bfab9: Pull complete
3c68d64f060f: Pull complete
Digest: sha256:a32daf0281803fd96e86daf6b0293b4d476cede1b5ce80b18452dfa1405360ff
Status: Downloaded newer image for mariadb:latest
f2d7032d36442e1ef9318e1e8b31d8986f1e5f3275863782c963113e79e8ee3b
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
      STATUS              PORTS               NAMES
f2d7032d3644        mariadb             "docker-entrypoint.s…"   5 seconds ago
      Up 4 seconds        3306/tcp            wordpressdb
```

L'attribut -e permet de passer plusieurs variables d'environnement.

L'attribut -v nous permet de binder un nouveau volume.

L'attribut --name permet de spécifier le nom du nouveau container créé.

L'attribut -v nous permet de créer un volume persistant afin de garder les modifications lorsqu'on coupe le container. (J'ai créé les dossier /root/wordpress/database et html auparavant avec « mkdir »).

Et l'attribut -d permet de spécifier le nom de l'image (locale ou non) que l'ont souhaites récupérer.

Entrer dans le container

Pour entrer dans un container Docker, on utilise la commande « *docker exec -it <nom_du_container> bash* ».

-it permet d'entrer dans le container en mode « *interactif* ».

« *Bash* » à la fin de la ligne de commande nous permet de dire à docker qu'on veut y accéder en interface de commande (bash). PowerShell ou cmd serait également possible si nous étions sur un core Windows.

Création d’un container Wordpress

```
root@ubuntu:~# docker run -e WORDPRESS_DB_USER=wpuser -e WORDPRESS_DB_NAME=word
press_db -p 8081:80 -v /root/wordpress/html:/var/www/html --link wordpressdb:my
sql --name wpcontainer -d wordpress
Unable to find image 'wordpress:latest' locally
latest: Pulling from library/wordpress
b8f262c62ec6: Pull complete
a98660e7def6: Pull complete
4d75689ceb37: Pull complete
639eb0368afa: Pull complete
99e337926e9c: Pull complete
431d44b3ce98: Pull complete
beb665ea0e0e: Pull complete
1914f5ed0362: Pull complete
3bb658c14677: Pull complete
6a4699b1063e: Pull complete
d23f6accef3d: Pull complete
3814846efc9c: Pull complete
e14c865e4394: Pull complete
2133ee9f21fd: Pull complete
c54f25ec0676: Pull complete
b2f1c6cffd7b: Pull complete
c26c7edcc26f: Pull complete
119b6e1da171: Pull complete
555b521641eb: Pull complete
e2864c97d80f: Pull complete
dfb00b06eb36: Pull complete
Digest: sha256:661e9b06de5e9a2b9b44461206fe05362afbc66d4cc9e4f2135da083a709de9e
Status: Downloaded newer image for wordpress:latest
```

Ici, les attributs sont globalement les mêmes excepté le –link qui va nous permettre de lier le container créé avec le service du container spécifié, ici MySQL sur wordpressdb.

-p nous permet de faire une redirection de port du container vers la machine hôte.

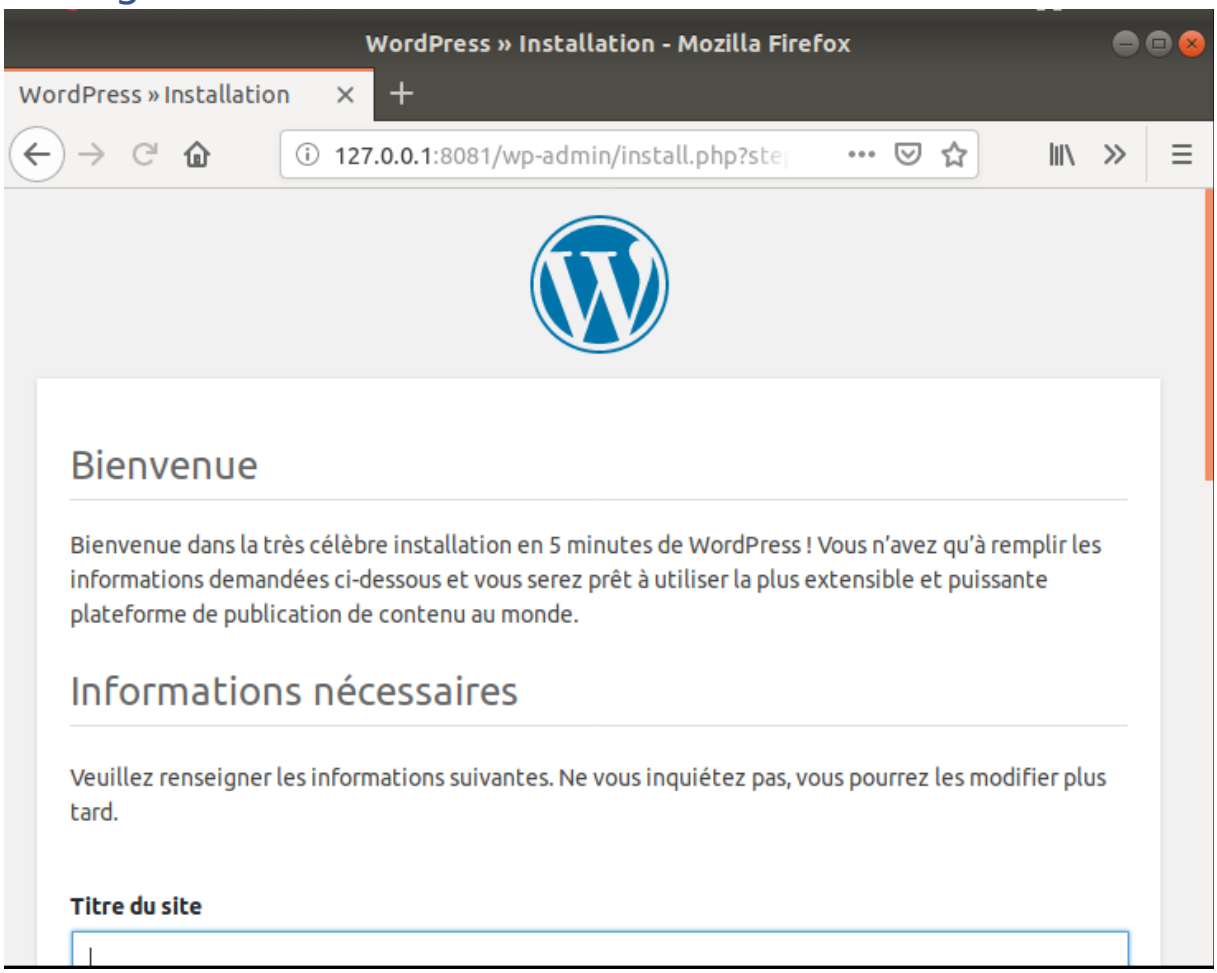
Ici, on créer un volume persistant de données grâce au -v (expliqué au-dessus).

Docker va automatiquement aller chercher l’image WordPress sur le docker hub, la télécharger en local et se baser dessus pour créer notre container wordpressdb.

```
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
      STATUS              PORTS                    NAMES
64c2ff5ba243        wordpress           "docker-entrypoint.s…"   7 seconds ago
      Up 5 seconds        0.0.0.0:8081->80/tcp     wpcontainer
f2d7032d3644        mariadb             "docker-entrypoint.s…"   12 minutes ago
      Up 12 minutes       3306/tcp                 wordpressdb
root@ubuntu:~#
```

Cette commande est assez pratique, elle nous permet de voir la liste des containers créés. Il suffit de regarder la colonne « STATUS » -> (Up / Down).

Configuration WordPress



Sur la machine hôte, on se connecte à l'adresse « 127.0.0.1:8081 », l'installation de WordPress s'affiche. Tout fonctionne correctement.

La base de données et le cms WordPress communiquent alors qu'ils sont sur deux containers différents.

Installation d'un container Portainer

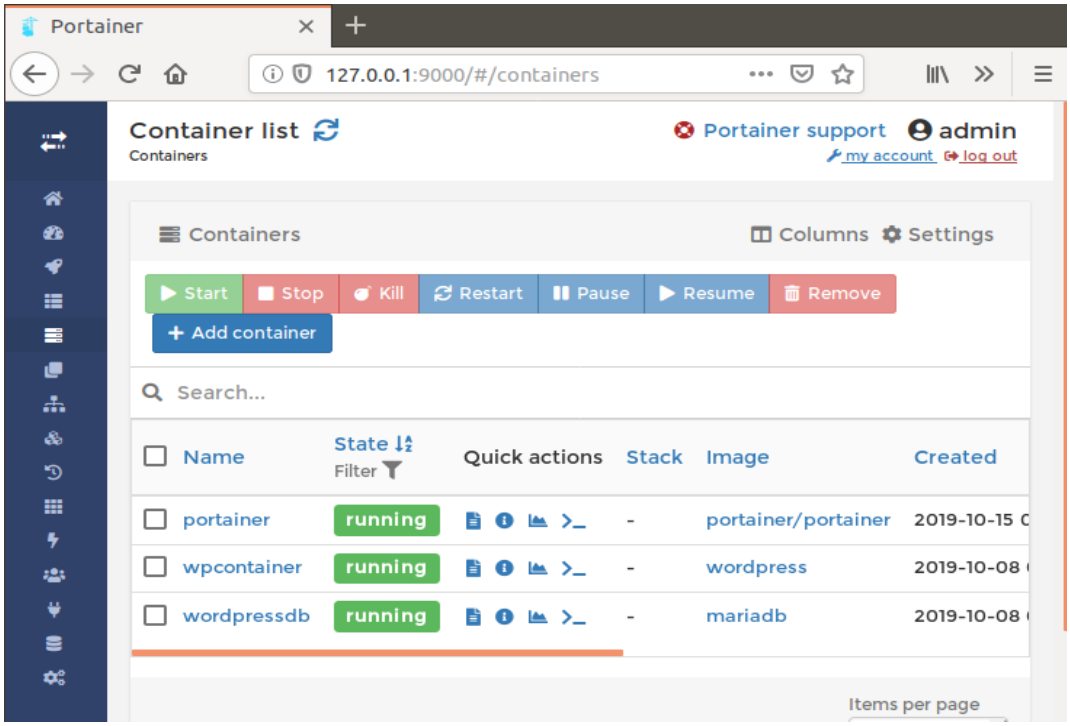
Portainer est un utilitaire relativement pratique qui permet d'avoir un aperçu visuel des containers et des images présents sur notre machine. Son installation est assez simple :

```
root@ubuntu:~# docker run --name portainer -d -p 9000:9000 -v /var/run/docker.s
ock:/var/run/docker.sock portainer/portainer
7ead1d778cdc7a1032bc702bc3c591366c2b7899afb1042ab2b53d2252a1e280
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED
        STATUS              PORTS                    NAMES
7ead1d778cdc        portainer/portainer    "/portainer"             3 seconds ag
o       Up 1 second         0.0.0.0:9000->9000/tcp   portainer
4dd2d35ede0d        wordpress              "docker-entrypoint.s…"   6 days ago
        Up 6 days           0.0.0.0:8081->80/tcp     wpcontainer
3416993fe78c        mariadb                "docker-entrypoint.s…"   6 days ago
        Up 6 days           3306/tcp                 wordpressdb
root@ubuntu:~#
```

--name pour spécifier le nom du container

-p pour attribuer le port 9000 au container « Portainer »

-v pour créer le **volume persistant** (explications en fin de procédure).



Copie d'écran de l'interface Portainer.

Il est également possible de l'installer via docker-compose (méthode ci-dessous).

Docker-Compose

Le fichier docker-compose nous permet de faire tout ce qu'on a vu au-dessus en quelques minutes. A partir d'une vm vierge (docker et docker-compose installés), nous n'avons plus qu'à exécuter ce docker-compose pour avoir les 4 container disponible en quelques minutes.

```
  GNU nano 2.9.3                  docker-compose.yml                      Modified

wordpressdb:
  image: mariadb
  volumes:
    - /root/wordpress/database:/var/lib/mysql
  environment:
    MYSQL_ROOT_PASSWORD: Passw0rd
    MYSQL_USER: user
    MYSQL_PASSWORD: Passw0rd
    MYSQL_DATABASE: wordpressdb
wordpress:
  image: wordpress
  ports:
    - 8081:80
  environment:
    MYSQL_DB_USER: wpuser
    MYSQL_DB_NAME: wordpress_db
  volumes:
    - /root/wordpress/html:/var/www/html
  links:
    - wordpressdb:mysql
```

```
portainer:
  image: portainer/portainer
  ports:
    - 9000:9000
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
phpmyadmin:
  image: corbinu/docker-phpmyadmin
  links:
    - wordpressdb:mysql
  ports:
    - 8082:80
  environment:
    MYSQL_USERNAME: root
    MYSQL_ROOT_PASSWORD: Passw0rd
```

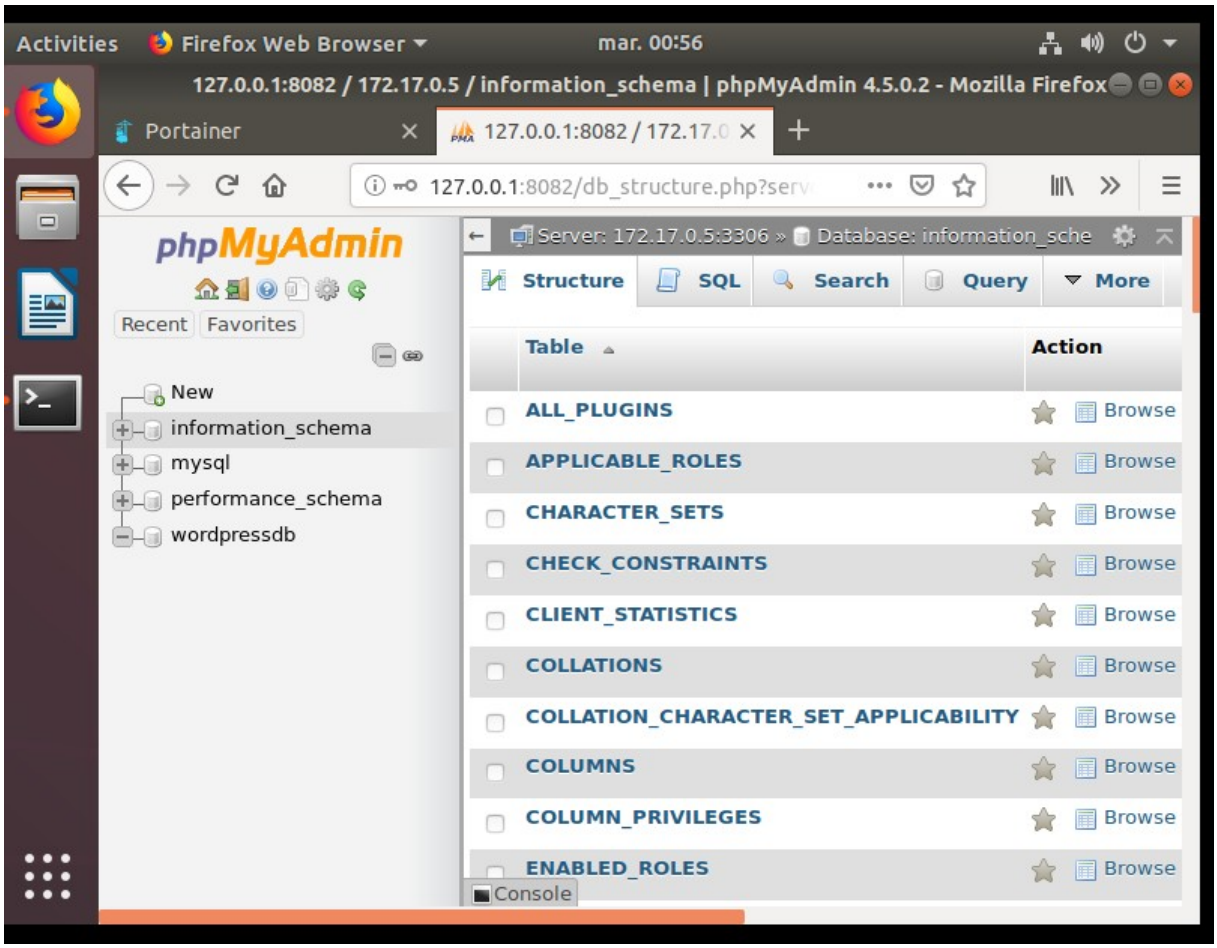
Quelques explications :

La première ligne permet de spécifier le nom du container, **image**, comme son nom l'indique spécifie l'image que l'on souhaite utiliser, **volumes** (attribut -v) nous sert à spécifier le volume persistant que l'on souhaite utiliser. **Links** (--link) nous permet de lier plusieurs containers entre eux (il est également possible d'utiliser networks). **Ports** (-p) nous permet d'attribuer un port au container créé. Enfin, **environment** (attribut -e en ligne de commande) nous permet de passer des variables d'environnement telles que l'utilisateur et le mot de passe de la base de données.

PhpMyAdmin

L'installation de phpMyAdmin a été gérée directement dans le docker-compose.yml. Comme nous pouvons le constater, le port 8082 est bien attribué au container PhpMyAdmin.

(Copie ci-dessus).



Sur la copie d'écran ci-dessus, la base de données « wordpressdb » est bien accessible.

Volumes persistants

```
File  Edit  View  Search  Terminal  Help
root@ubuntu:/root/wordpress# ls
database  html  phpmyadmin
root@ubuntu:/root/wordpress# ls database/
aria_log.00000001  ibdata1       ibtmp1             performance_schema
aria_log_control   ib_logfile0   multi-master.info  wordpressdb
ib_buffer_pool     ib_logfile1   mysql
root@ubuntu:/root/wordpress#
```

Lors de la création du container wordpressdb, nous avons précisé qu'un volume persistant devait être créé dans « /root/wordpress/database ». Voici son contenu.

L'avantage de cette méthode est que lorsqu'on éteint le container wordpressdb (qui est le container de la base de données), on ne perd aucune donnée vu qu'elles sont stockées dans le répertoire en dur « /root/wordpress/database ».

Dockerfile

Les Dockerfiles sont pratiques si l'ont souhaite passer un projet à une autre personne. Il/elle n'aura plus qu'à faire la commande « docker build -f Dockerfile . » et patienter le temps que tout soit créé.

Un Dockerfile contient toutes les lignes de commandes que nous avons utilisés pour venir à bout de notre projet.

```
  GNU nano 2.9.3                    Dockerfile                         Modified

FROM ubuntu:18.04

RUN apt update && \
    apt upgrade && \
    mkdir /root/wordpress && \
    mkdir /root/wordpress/html && \
    mkdir /root/wordpress/database
RUN docker run -e MYSQL_ROOT_PASSWORD=Passw0rd -e MYSQL_USER=user \
    -e MYSQL_PASSWORD=Passw0rd -e MYSQL_DATABASE=wordpressdb \
    -v /root/wordpress/database:/var/lib/mysql \
    --name wordpressdb -d mariadb

RUN docker run -e WORDPRESS_DB_USER=wpuser -e WORDPRESS_DB_NAME=wordpress_db \
    -p 8081:80 -v /root/wordpress/html:/var/www/html \
    --link wordpressdb:mysql --name wpcontainer -d wordpress

RUN docker run --name portainer -p 9000:9000 \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -d portainer/portainer
```

Première commande pour vérifier que la vm soit bien vide. Ensuite, on lance le Dockerfile et on attend.

```
root@ubuntu:/root# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
 STATUS              PORTS               NAMES
root@ubuntu:/root# docker build -f /root/Dockerfile .
Sending build context to Docker daemon  9.728kB
Step 1/5 : FROM ubuntu:18.04
18.04: Pulling from library/ubuntu
5667fdb72017: Pull complete
d83811f270d5: Pull complete
ee671aafb583: Pull complete
7fc152dfb3a6: Pull complete
Digest: sha256:b88f8848e9a1a4e4558ba7cfc4acc5879e1d0e7ac06401409062ad2627e6fb58
Status: Downloaded newer image for ubuntu:18.04
 ---> 2ca708c1c9cc
Step 2/5 : RUN apt update &&     apt upgrade &&     mkdir /root/wordpress &&
  mkdir /root/wordpress/html &&     mkdir /root/wordpress/database
 ---> Running in 08f76ee95c25
```

Au bout de quelques minutes, voici le résultat :

```
root@ubuntu:/root/wordpress# docker ps
CONTAINER ID        IMAGE                  COMMAND
        STATUS              PORTS                    NAMES
7ead1d778cdc        portainer/portainer    "/portainer"
        Up About a minute   0.0.0.0:9000->9000/tcp   portainer
4dd2d35ede0d        wordpress              "docker-entrypoint.s…"
        Up About a minute   0.0.0.0:8081->80/tcp     wpcontainer
3416993fe78c        mariadb                "docker-entrypoint.s…"
        Up About a minute   3306/tcp                 wordpressdb
root@ubuntu:/root/wordpress#
```

A l'aide de la commande docker ps, on a un rapide aperçu des containers créés par le Dockerfile. Nos trois containers sont bien créés avec les paramètres que nous lui avons définis auparavant.

Configuration Traefik

Nous pouvons maintenant commencer la configuration de Traefik. Celui-ci va nous permettre de gérer le « load-balancing » (utile sur les gros sites web afin de répartir le trafic sur plusieurs serveurs).

```
root@ubuntu:/root/wordpress# mkdir traefik
root@ubuntu:/root/wordpress# cd traefik/
root@ubuntu:/root/wordpress/traefik# nano traefik.toml
```

Nous allons créer un docker-compose.yml ; configuration ci-dessous :

```
 GNU nano 2.9.3                  docker-compose.yml

version: '2'
services:
  traefik:
    image: traefik:1.5
    command: --web --docker --logLevel=DEBUG
    ports:
      - "80:80"
      - "8080:8080"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
      - "./traefik/traefik.toml:/traefik.toml"

  webapp:
    image: dockercloud/hello-world
    labels:
      - "traefik.port=80"
      - "traefik.backend=hello"
      - "traefik.frontend.rule=Host:b3tlse.care"
```

On se base sur l'image traefik version 1.5, on passe l'argument –web pour lui spécifier qu'il s'agit d'un service web à interpréter et on créer un service « webapp » qui va en fait être notre page d'accueil. Celle-ci va se baser sur l'image « dockercloud/hello-world ».

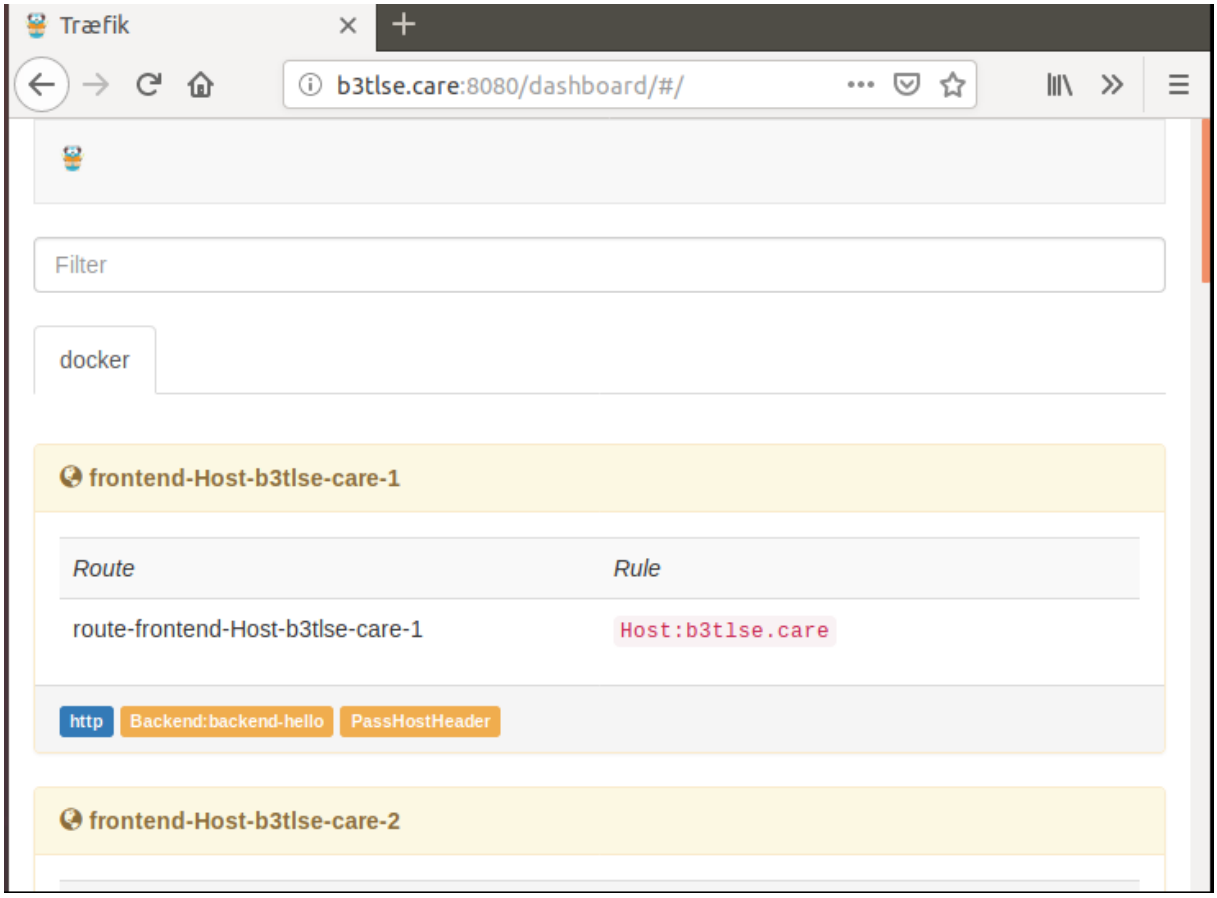
On créé également le fichier de configuration traefik.toml au même niveau que le docker-compose. Il restera vide pour l'instant.

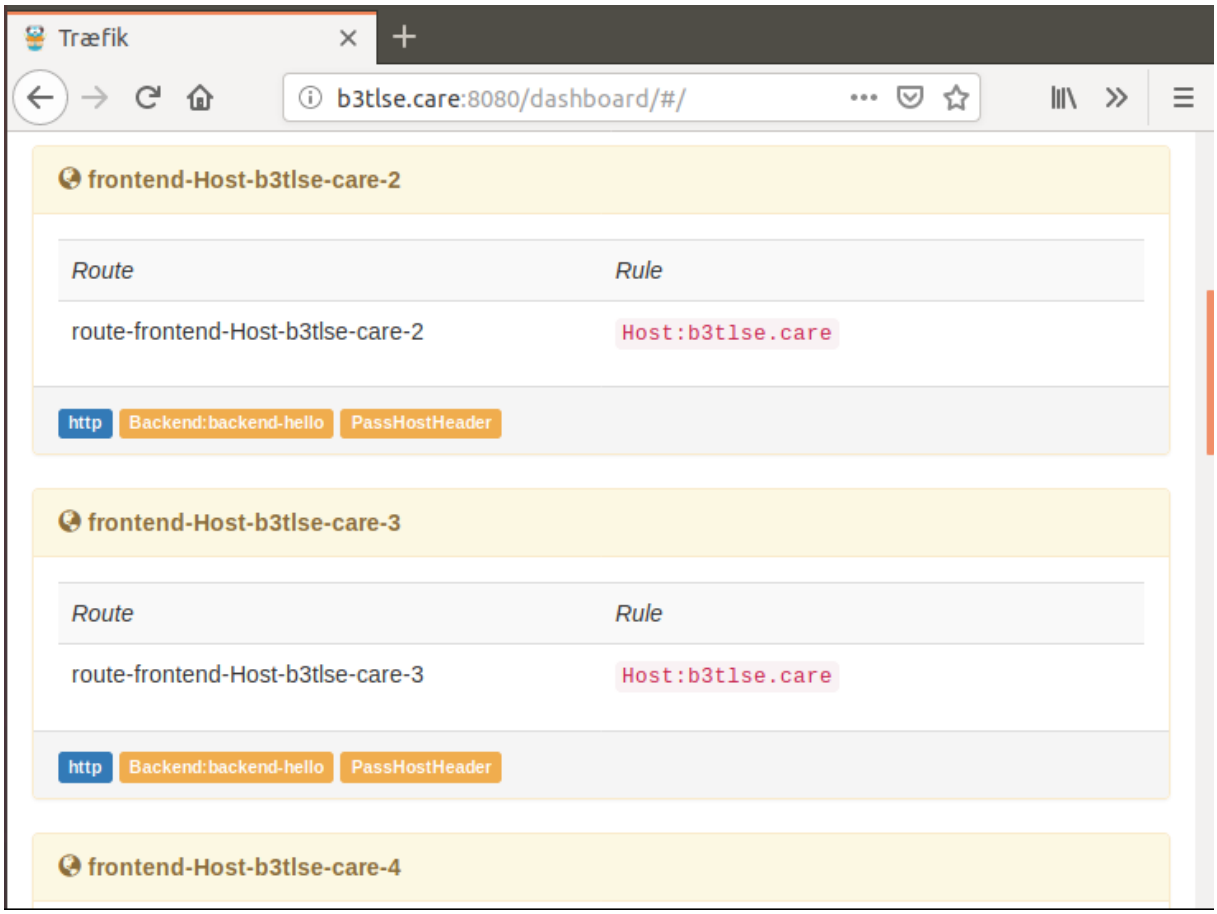
Nous pouvons désormais lancer le docker compose avec la particularité –scale qui nous permet de créer 5 containers webapp qui vont s'alterner aléatoirement lorsqu'on arrivera sur le site.

```
root@ubuntu:/root/wordpress/traefik# touch traefik.toml
root@ubuntu:/root/wordpress/traefik# docker-compose up -d --scale webapp=5
```

```
1.5: Pulling from library/traefik
5d3835484afe: Pull complete
a34d9cadf285: Pull complete
Digest: sha256:96fe09a867c29dfd5ecf240c955ae104dc7d3422bcc3d42b74a0d0fcc5a2377f
Status: Downloaded newer image for traefik:1.5
Pulling webapp (dockercloud/hello-world:latest)...
latest: Pulling from dockercloud/hello-world
486a8e636d62: Pull complete
03374a673b41: Pull complete
101d2c41032c: Pull complete
1252e1f36d2b: Pull complete
8385bb1a4377: Pull complete
f29c06131731: Pull complete
Digest: sha256:c6739be46772256abdd1aad960ea8cf6c6a5f841c12e8d9a65cd5ef23bab45fc
Status: Downloaded newer image for dockercloud/hello-world:latest
Creating traefik_traefik_1 ...
Creating traefik_webapp_1 ...
Creating traefik_webapp_2 ...
Creating traefik_webapp_3 ...
Creating traefik_webapp_4 ...
Creating traefik_webapp_5 ...
Creating traefik_traefik_1
Creating traefik_webapp_1 ... done
Creating traefik_webapp_2 ... done
Creating traefik_webapp_3 ... done
Creating traefik_webapp_4 ... done
Creating traefik_traefik_1 ... done
root@ubuntu:/root/wordpress/traefik#
```

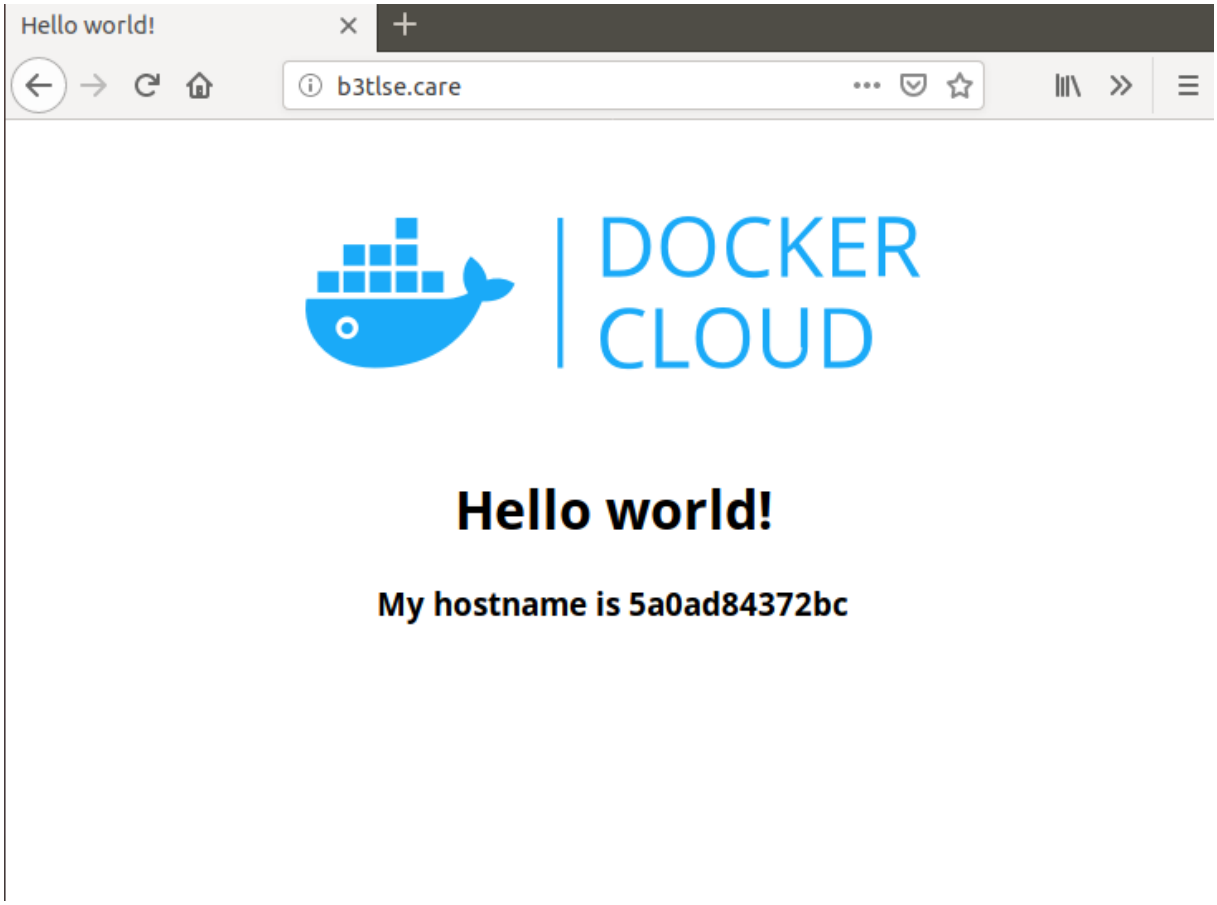
On constate que les containers ont bien étés créés.


Træfik
b3tlse.care:8080/dashboard/#/
Filter
docker
frontend-Host-b3tlse-care-1
Route
Rule
route-frontend-Host-b3tlse-care-1
Host:b3tlse.care
http
Backend:backend-hello
PassHostHeader
frontend-Host-b3tlse-care-2


Træfik
b3tlse.care:8080/dashboard/#/
frontend-Host-b3tlse-care-2
Route
Rule
route-frontend-Host-b3tlse-care-2
Host:b3tlse.care
http
Backend:backend-hello
PassHostHeader
frontend-Host-b3tlse-care-3
Route
Rule
route-frontend-Host-b3tlse-care-3
Host:b3tlse.care
http
Backend:backend-hello
PassHostHeader
frontend-Host-b3tlse-care-4

Si on va sur notre domaine :8080, on arrive bien sur la page de statistiques Traefik. On voit également que tous nos containers sont disponibles.

Si on va sur le domaine b3tlse.care, on arrive sur la page hello avec un identifiant qui change à chaque fois qu'on actualise la page.



Création d’un container HA Proxy

L’installation de HA Proxy a été gérée directement dans le docker-compose.yml. Comme nous pouvons le constater, le port 1936 est bien attribué au container HA Proxy.

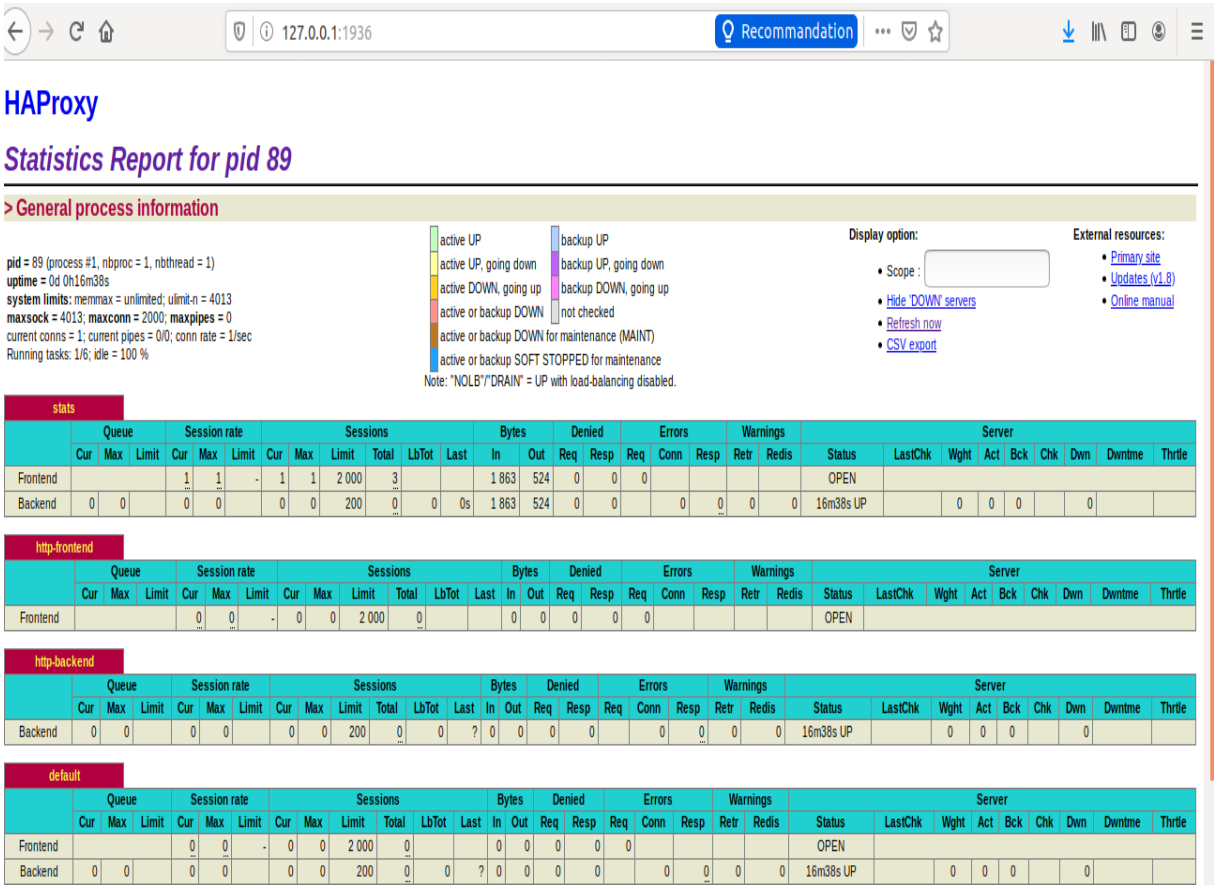
```
root@omar-HP-Laptop-15-bs0xx:~# docker ps
CONTAINER ID        IMAGE                    COMMAND
 CREATED             STATUS              PORTS
      NAMES
07135a81e98a        traefik:1.7              "/traefik"
 11 days ago         Up 7 minutes        80/tcp, 0.0.0.0:8081->8081/
tcp   traefik_traefik_1
5509cc1b790e        eeacms/haproxy           "/docker-entrypoint.…"
 11 days ago         Up 2 minutes        0.0.0.0:1936->1936/tcp
      haproxy_haproxy_1
f005bd9ef3ea        phpmyadmin/phpmyadmin    "/docker-entrypoint.…"
 12 days ago         Up 7 minutes        0.0.0.0:8082->80/tcp
      phpmyadmin_phpmyadmin_1
4befdcec18c9        mysql:5.7                "docker-entrypoint.s…"
 12 days ago         Up 7 minutes        3306/tcp, 33060/tcp
      phpmyadmin_db_1
bb25aa3d51f1        portainer/portainer      "/portainer"
 12 days ago         Up 48 seconds       0.0.0.0:9000->9000/tcp
      portainer
5f875755c6d7        wordpress:latest         "docker-entrypoint.s…"
 12 days ago         Up 7 minutes        0.0.0.0:8000->80/tcp
      wordpress_wordpress_1
d1957b49de08        mysql:5.7                "docker-entrypoint.s…"
 12 days ago         Up 7 minutes        3306/tcp, 33060/tcp
      wordpress_db_1
4d2787a7669d        nginx:latest             "nginx -g 'daemon of…"
 12 days ago         Up 8 seconds        0.0.0.0:8080->80/tcp
      web
```

Docker-compose.yml

```
version: "2"
services:
  haproxy:
    image: eeacms/haproxy
    depends_on:
    - webapp
    ports:
    # - "81:5000"
    - "1936:1936"
    environment:
      BACKENDS: "webapp"
      DNS_ENABLED: "true"
      LOG_LEVEL: "info"

  webapp:
    image: eeacms/hello
~
```

avec le port 1936 on accéde bien sur la page statistics de HAProxy en utilisant admin comme utilisateur et mot de passe



Matthieu MARQUETTE
Omar SY
Docker