8INF911: Mini-Projets-Docker

Ce dépot contient les ressources des quatres mini-projets réalisés par Léo VANSIMAY et Gaël LECONTE dans le cadre du cours **Architecture Cloud et Méthodes DevOps, outils pour le Cloud-Gaming** pour le trimestre d'été 2022 à l'UQAC.

Table des matières

1. VISUALISATION DE DONNEES IOT – FASTAPI / INFLUXDB
2. DISPONIBILITE DE SERVICE
3. HEBERGEMENT WEB (VERSION NGINX)
4. HEBERGEMENT WEB (VERSION TRAEFIK)

1. VISUALISATION DE DONNEES IOT – FASTAPI / INFLUXDB

Notre idée initiale était de créer deux images, une pour utiliser l'API *pyflux*, et l'autre pour lancer *InfluxDB* et le configurer automatiquement. Mais nous avons remarqué qu'une image *InfluxDB* était disponible sur DockerHub. Voici le `Dockerfile` du conteneur *InfluxPro*:

```
FROM influxdb

ENV DOCKER_INFLUXDB_INIT_BUCKET=$DOCKER_INFLUXDB_INIT_BUCKET
ENV DOCKER_INFLUXDB_INIT_ORG=$DOCKER_INFLUXDB_INIT_ORG
ENV DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN
ENV DOCKER_INFLUXDB_INIT_PASSWORD=$DOCKER_INFLUXDB_INIT_PASSWORD
ENV DOCKER_INFLUXDB_INIT_USERNAME=$DOCKER_INFLUXDB_INIT_USERNAME
ENV DOCKER_INFLUXDB_INIT_MODE=setup
ENTRYPOINT influx setup -b DOCKER_INFLUXDB_INIT_BUCKET -o DOCKER_INFLUXDB_INIT_ORG
-t DOCKER_INFLUXDB_INIT_ADMIN_TOKEN -u DOCKER_INFLUXDB_INIT_USERNAME -p
DOCKER_INFLUXDB_INIT_PASSWORD
```

Si certaines variables d'environements sont configurées, alors le service pourra se configurer tout seul. On fixe ainsi la variable `DOCKER_INFLUXDB_INIT_MODE` sur `setup` dans le fichier `docker-compose.yml`.

```
version: "3.9"

services:
  pyflux:
    build:
      context: ./pyflux
      dockerfile: Dockerfile
    hostname: pyflux
    environment:
      - DOCKER_INFLUXDB_INIT_URL=${DOCKER_INFLUXDB_INIT_URL}
      - DOCKER_INFLUXDB_INIT_ORG=${DOCKER_INFLUXDB_INIT_ORG}
      - DOCKER_INFLUXDB_INIT_BUCKET=${DOCKER_INFLUXDB_INIT_BUCKET}
```

```
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=${DOCKER_INFLUXDB_INIT_ADMIN_TOKEN}
    networks:
      - front
      - back
    ports:
      - "3000:3000"
  influxpro:
    image: influxdb
    hostname: influxpro
    environment:
      - DOCKER_INFLUXDB_INIT_ORG=${DOCKER_INFLUXDB_INIT_ORG}
      - DOCKER_INFLUXDB_INIT_BUCKET=${DOCKER_INFLUXDB_INIT_BUCKET}
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=${DOCKER_INFLUXDB_INIT_ADMIN_TOKEN}
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=${DOCKER_INFLUXDB_INIT_USERNAME}
      - DOCKER_INFLUXDB_INIT_PASSWORD=${DOCKER_INFLUXDB_INIT_PASSWORD}
    networks:
      - back

networks:
  front:
  back:
```

Pour des raisons de sécurité, aucune information confidentielle n'est renseignée dans ce fichier, nous utilisons à la place le fichier `.env` qui est par défaut chargé par docker-compose pour fixer les variables d'environnement.

```
DOCKER_INFLUXDB_INIT_URL="http://influxpro:8086"
DOCKER_INFLUXDB_INIT_ORG="my-org"
DOCKER_INFLUXDB_INIT_BUCKET="my-bucket"
DOCKER_INFLUXDB_INIT_ADMIN_TOKEN="my-token"
DOCKER_INFLUXDB_INIT_USERNAME="my-user"
DOCKER_INFLUXDB_INIT_PASSWORD="my-password"
```

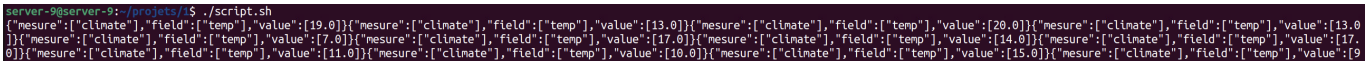
N-B: Dans le contexte pédagogique, le `.env` est présent sur le dépôt git, mais dans une situation de production, ce fichier ne doit pas être communiqué, étant donné qu'il sert à configurer un produit pour un client.

Fonctionnement du projet

Une fois la commande `docker-compose up` lancée, le port 3000 du conteneur *pyflux* est exposé et mappé sur le port 3000 de la machine hôte, on peut donc joindre l'API en envoyant des requêtes à l'adresse http://localhost:3000. Pour populer la base de données, on se sert du script `script.sh`:

```
#!/bin/bash
while true
do
   v=$(shuf -i 5-20 -n 1)
```

```
    curl -X POST http://localhost:3000/climate/temp/$v
    sleep 1
done
```



2. DISPONIBILITE DE SERVICE

Ici l'idée est de mettre en évidence la disponibilité applicative en utilisant le reverse-proxy de Nginx. L'objectif est de créer un conteneur qui sera utilisé en frontal d'un ensemble de conteneurs, situé sur leur réseau propre, qui pourront être mis à l'échelle par Docker.

Pour la création des conteneurs un docker-compse.yml sera réalisé à la racine.

```
services:
  reverseproxy:
    build:
      context: ./reverseproxy
      dockerfile: Dockerfile
    ports:
      - 80:80
  whoami:
    image: containous/whoami
    hostname: whoami
```

On utilise Nginx (documentation: Nginx) pour gérer l'équilibrage de charge sur les conteneurs whoami (documentation: Whoami). Pour la configuration du reverse proxy, le fichier de configuration de base de nginx situé dans /etc/nginx/conf.d à été écrasé par `nginx.conf`:

```
FROM nginx

COPY nginx.conf /etc/nginx/nginx.conf
```

Fichier `nginx.conf`:

```
event{}
http{server {
   listen 80;
   location / {
      proxy_pass http://whoami;
      proxy_set_header   Host             $host;
      proxy_set_header   X-Real-IP        $remote_addr;
      proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
   }
}
}
```

Fonctionnement du projet

Une fois la commande `docker-compose up` lancée, le port 80 du conteneur *nginx* est exposé et mappé sur le port 80 de la machine hôte, on peut donc joindre les conteneurs *whoami* en envoyant des requêtes à l'adresse http://localhost. Le fichier de configuration `nginx.conf` redirige ainsi toute requête reçu sur le port 80 aux conteneurs.

```
server-9@server-9:~/projets/2/reverseproxy$ curl http://localhost
Hostname: whoami
IP: 127.0.0.1
IP: 192.168.160.7
RemoteAddr: 192.168.160.6:38214
GET / HTTP/1.1
Host: localhost
User-Agent: curl/7.81.0
Accept: */*
Connection: close
X-Forwarded-For: 192.168.160.1
X-Real-Ip: 192.168.160.1

server-9@server-9:~/projets/2/reverseproxy$ curl http://localhost
Hostname: whoami
IP: 127.0.0.1
IP: 192.168.160.3
RemoteAddr: 192.168.160.6:56454
GET / HTTP/1.1
Host: localhost
User-Agent: curl/7.81.0
Accept: */*
Connection: close
X-Forwarded-For: 192.168.160.1
X-Real-Ip: 192.168.160.1

server-9@server-9:~/projets/2/reverseproxy$ curl http://localhost
Hostname: whoami
IP: 127.0.0.1
IP: 192.168.160.5
RemoteAddr: 192.168.160.6:51188
GET / HTTP/1.1
Host: localhost
User-Agent: curl/7.81.0
Accept: */*
Connection: close
```

3. HEBERGEMENT WEB (VERSION NGINX)

L'objectif de ce mini projet est de créer un service Docker permettant l'hébergement d'un site web avec Nginx. Le site ne sera pas présenté frontalement mais sera accessible à travers un service de proxy Nginx. Un certificat électronique sera automatiquement généré en utilisant l'API de l'hébergeur OVH. Pour réaliser ce projet, vous devez être propriétaire d'un nom de domaine OVH. Le certificat électronique sera généré par LetsEncrypt qui procèdera à une vérification de la possession du nom de domaine pour lequel vous demandez à créer un certificat. Nous paramètrerons l'utilitaire de génération de certificat certbot pour réaliser un challenge DNS auprès de LetsEncrypt.

Arborescence de travail

Dans un dossier *Nginx* nous insérons un fichier de configuration du service `Dockerfile`, ainsi qu'un dossier `site`, prévu pour stocker l'arborescence du site web. Dans le `Dockerfile` le dossier de stockage sera dans */home/site* en lecture seule.

```
FROM alpine:latest

RUN apk update
RUN apk add nginx
RUN apk add openrc

RUN openrc
RUN touch /run/openrc/softlevel

RUN rc-update add nginx

VOLUME /home/server-9/projets/3/nginx/site /home/site:ro

RUN service nginx start

EXPOSE 80/TCP
ENTRYPOINT /bin/sh
```

Le site web généré sera un site issu de HTML5Up dont nous avons placé l'archive dans `nginx/site`

Conteneur de service

Dans un dossier *Cert* nous insérons un fichier de configuration du service `Dockerfile` avec les caractéristiques suivantes :

```
FROM debian:latest

RUN apt-get update && apt-get upgrade -y
RUN apt-get install certbot python3-certbot-dns-ovh python3-certbot-nginx -y

CMD [ "/bin/sh", "-c", "while true; do sleep 1; done" ]
```

Paramétrage de l'API OVH

Apres avoir acheté un nom de domaine, il faut créer un token. Ce token doit posséder les droits *GET PUT DELETE POST*. Le tokent est stocké dans un fichier ovh.ini situé à l'interieur du dossier *cert*.

Ensuite à l'aide de la commande suivante, on génère les certificats : `certbot certonly --dns-ovh --dns-ovh-credentials /ovh/.ovh.ini -d <fqdn>`

Docker-compose.yml

À la racine du projet, on créer un fichier *docker-compose.yml* où l'on déclare les conteneurs *nginx* et *cerbot*.

```
version: "3.9"

services:
  certbot:
    build:
      context: ./cert
      dockerfile: Dockerfile
    volumes:
      - ./cert/ovh/:/ovh:ro
      - certs:/etc/letsencrypt
  nginx:
    image: nginx
    hostname: nginx
    volumes:
      - ./nginx/site.conf:/etc/nginx/conf.d/site.conf
      - ./nginx/site/html5up-story:/home/site:ro
      - certs:/etc/letsencrypt:ro
    ports:
      - "80:80"
      - "443:443"

volumes:
  certs:
```

Les ports 80 et 443 sont mappé pour http et https.

Un volume *certs* est déclaré et est utilisé dans le conteneur *certbot* en le mappant sur l'arborescence du conteneur /etc/letsencrypt. Ce volume sera présenté aussi dans le conteneur nginx sur le même point de montage, mais en lecture seule.

Service HTTPS

Ce service est paramétré dans le fichier *nginx/site.conf*.

```
server{
 listen 80;
 listen 443 ssl http2;
 listen [::]:443 ssl http2;
 server_name www.projet.keleranv.ovh;
```

```
    # SSL
    ssl_certificate /etc/letsencrypt/live/www.projet.keleranv.ovh/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/www.projet.keleranv.ovh/privkey.pem;
    ssl_trusted_certificate /etc/letsencrypt/live/www.projet.keleranv.ovh/chain.pem;
    location / {
     root /home/site;
     }
   }
```

Résultat

On observe bien les ports d'écoute 80 & 443 du conteneur nginx.

```
root@server-9:/home/server-9/projets/3# docker-compose ps
   Name                 Command              State                                  Ports
-------------------------------------------------------------------------------------------------------------------------------
3_certbot_1   /bin/sh -c while true; do  ...   Up
3_nginx_1     /docker-entrypoint.sh ngin ...   Up      0.0.0.0:443->443/tcp,:::443->443/tcp, 0.0.0.0:80->80/tcp,:::80->80/tcp
```

Après requête à l'adresse `https://www.projet.keleranv.ovh`, le site s'affiche.

```
root@server-9:/home/server-9/projets/3# curl https://www.projet.keleranv.ovh
<!DOCTYPE HTML>
<!--
        Story by HTML5 UP
        html5up.net | @ajlkn
        Free for personal and commercial use under the CCA 3.0 license (html5up.net/license)
-->
<html>
        <head>
                <title>Story by HTML5 UP</title>
                <meta charset="utf-8" />
                <meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=no" />
                <link rel="stylesheet" href="assets/css/main.css" />
                <noscript><link rel="stylesheet" href="assets/css/noscript.css" /></noscript>
        </head>
        <body class="is-preload">

                <!-- Wrapper -->
                        <div id="wrapper" class="divided">

                                <!-- One -->
                                        <section class="banner style1 orient-left content-align-left image-position-right fullscreen onload-image-fade-in onload-content-fade-right">
                                                <div class="content">
                                                        <h1>Story</h1>
                                                        <p class="major">A (modular, highly tweakable) responsive one-page template designed by <a href="https://html5up.net">HTML5 UP</a> and re
leased for free under the <a href="https://html5up.net/license">Creative Commons</a>.</p>
                                                        <ul class="actions stacked">
                                                                <li><a href="#first" class="button big wide smooth-scroll-middle">Get Started</a></li>
                                                        </ul>
                                                </div>
                                                <div class="image">
                                                        <img src="images/banner.jpg" alt="" />
                                                </div>
                                        </section>
```

4. HEBERGEMENT WEB (VERSION TRAEFIK)

L'objectif de ce 4 ième mini projet est de créer un service Docker permettant l'hébergement d'un site web avec Nginx. Le site ne sera pas présenté frontalement, mais sera accessible à travers un service de proxy Traefik. Un certificat électronique sera automatiquement généré en utilisant l'API de l'hébergeur OVH.

Arborescence de travail

Dans un dossier *Nginx* (support de notre conteneur web) nous insérons un fichier de configuration du service `Dockerfile`, ainsi qu'un fichier `site.conf` dans un dossier `site`, prévu pour stocker l'arborescence du site web.

```
  server{
   listen 80;
   location / {
```

```
    root /home/site;
    }
}
```

Dans le `Dockerfile` le dossier de stockage sera dans */home/site* en lecture seule.

```
FROM alpine:latest

RUN apk update
RUN apk add nginx
RUN apk add openrc

RUN openrc
RUN touch /run/openrc/softlevel

RUN rc-update add nginx

VOLUME /home/server-9/projets/4/nginx/site /home/site:ro

RUN service nginx start

ENTRYPOINT /bin/sh
```

Docker-compose.yml

Le fichier Docker-compose.yml créer deux service :

- reverse-proxy : Ce conteneur utilise l'image de traefic (documentation: Traefik) qui reçoit les demandes au nom du système et trouve quel service est responsable de traiter la requête. Ici nous n'avons créé qu'un seul service, le service web nginx.
- nginx : Ce conteneur correspond à notre serveur web avec une configuration très basique, il écoute sur son port 80 et met à disposition le site web.

Génération du certificat avec l'API OVH

Pour générer un certificat pour Traefik, on utilise ses options permettant l'accès à l'API d'OVH, une fois un token d'API généré, on renseigne ce token dans le `.env` afin qu'il soit chargé par `docker-compose`.

Fonctionnement de l'API OVH

À l'aide du token généré, Traefik va générer un challenge, et stocker la réponse dans une entrée TXT du domaine DNS `keleranv.ovh`, et si les serveurs de certification sont capables de retrouver la réponse dans les entrées DNS, alors le certificat est généré.

Au niveau du reverse-proxy seul les ports 8080 pour l'ui traefik et le port 443 pour les requêtes HTTPS sont autorisées.

```
version: '3'

services:
  reverse-proxy:
    # The official v2 Traefik docker image
    image: traefik:v2.7
    # Enables the web UI and tells Traefik to listen to docker
    command:
      #- "--log.level=DEBUG"
      - "--api.insecure=true"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
        #- "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
      - "--certificatesresolvers.myresolver.acme.dnschallenge=true"
      - "--certificatesresolvers.myresolver.acme.dnschallenge.provider=ovh"
      #- "--certificatesresolvers.myresolver.acme.caserver=https://acme-staging-
v02.api.letsencrypt.org/directory"
      - "--certificatesresolvers.myresolver.acme.email=l.vansimay@gmail.com"
      - "--certificatesresolvers.myresolver.acme.storage=/letsencrypt/acme.json"
    ports:
      # The HTTP port
      #- "80:80"
      # The Web UI (enabled by --api.insecure=true)
      - "8080:8080"
      # The HTTPS port
      - "443:443"

    environment:
      - OVH_ENDPOINT=${DNS_OVH_ENDPOINT}
      - OVH_APPLICATION_KEY=${DNS_OVH_APPLICATION_KEY}
      - OVH_APPLICATION_SECRET=${DNS_OVH_APPLICATION_SECRET}
      - OVH_CONSUMER_KEY=${DNS_OVH_CONSUMER_KEY}

    volumes:
      # So that Traefik can listen to the Docker events
      - /var/run/docker.sock:/var/run/docker.sock
      - ./letsencrypt:/letsencrypt
   nginx:
    image: nginx
    hostname: nginx
    volumes:
      - ./nginx/site.conf:/etc/nginx/conf.d/default.conf
      - ./nginx/site/html5up-story:/home/site:ro
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.nginx.rule=Host(`www.projet.keleranv.ovh`)"
      - "traefik.http.routers.nginx.entrypoints=websecure"
      - "traefik.http.routers.nginx.tls.certresolver=myresolver"
```

Résultat

On observe bien les ports d'écoute 8080 & 443 du conteneur Traefik.

```
server-9@server-9:~/projets/4$ sudo docker-compose up -d
Starting 4_nginx_1         ... done
Starting 4_reverse-proxy_1 ... done
server-9@server-9:~/projets/4$ sudo docker-compose ps
      Name                     Command               State                                        Ports
-------------------------------------------------------------------------------------------------------------------------------------------------
4_nginx_1           /docker-entrypoint.sh ngin ...   Up      80/tcp
4_reverse-proxy_1   /entrypoint.sh --api.insec ...   Up      0.0.0.0:443->443/tcp,:::443->443/tcp, 80/tcp, 0.0.0.0:8080->8080/tcp,:::8080->8080/tcp
```

Après requête à l'adresse `https://www.projet.keleranv.ovh`, le site s'affiche.

```
nginx_1           | 2022/06/22 13:57:49 [notice] 1#1: start worker process 30
nginx_1           | 2022/06/22 13:57:49 [notice] 1#1: start worker process 31
nginx_1           | 172.31.0.2 - - [22/Jun/2022:13:57:59 +0000] "GET / HTTP/1.1" 200 16045 "-" "curl/7.81.0" "172.31.0.1"
```

```
root@server-9:/home/server-9/projets/4# curl https://www.projet.keleranv.ovh
<!DOCTYPE HTML>
<!--
        Story by HTML5 UP
        html5up.net | @ajlkn
        Free for personal and commercial use under the CCA 3.0 license (html5up.net/license)
-->
<html>
        <head>
                <title>Story by HTML5 UP</title>
                <meta charset="utf-8" />
                <meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=no" />
                <link rel="stylesheet" href="assets/css/main.css" />
                <noscript><link rel="stylesheet" href="assets/css/noscript.css" /></noscript>
        </head>
        <body class="is-preload">

                <!-- Wrapper -->
```