TC3 - TD47

Rapport de projet: Virtualisation et Conteneurisation

Equipe :
LOUISIN-MANIGAT Ayinde
SATOURI Amani
RABESAHALA Mahefa Hyacinthe

2023-2024

PLAN

I. Introduction

Ce projet “humans best friends” est une application qui utilise l’architecture microservices avec des conteneurs Docker. Pour le déploiement de cette application nous faisons appel à Python, Node.js et NET associées à Redis.
L’objectif de ce projet est de montrer comment l’utilisation de Docker en tant que plateforme de conteneurisation permet le déploiement et la gestion d’applications.

Dans ce rapport, nous allons voir comment la portabilité est assurée et comment les processus de développement, d’intégration et de déploiement sont simplifiés par l’utilisation de Docker.

Pour ce faire, nous allons détailler dans les parties suivantes comment nous avons mis en place ces processus afin de réaliser le projet.

Pour ce projet, on va configurer l’infrastructure de la sorte: nous allons utiliser comme plateforme de virtualisation VMware ESXi et y déployer une machine virtuelle Ubuntu. Nous allons nous baser sur le système d’exploitation Ubuntu afin qu’il soit l'hôte pour installer Docker.

Pour résumer, nous aurons besoin d’utiliser les outils suivants pour ce projet:

- VMware
- ESXi
- VM Ubuntu
- Docker et docker-compose (installé sur le système d'exploitation Ubuntu)

II. L'Environnement Docker

Nous allons partir de zéro. Nous avons actuellement une machine virtuelle ubuntu installé sur un ESXI. Commençons l'installation de l'environnement docker

Nous devons d'abord désinstaller les anciennes versions des packages docker. Même si c'est la première installation de docker sur la machine virtuelle, par sécurité il est préférable de s'assurer qu'aucun package docker est installer avec la commande suivante :

- **for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done**

Avant d'installer Docker pour la première fois sur notre machine, nous devons configurer le repository docker. Par la suite, nous pourrons installer et mettre à jour Docker à partir du repository.

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

Après toutes ces étapes, nous pouvons installer docker et docker-compose avec la commande suivante :

- **sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin docker-compose**

Pour s'assurer que docker est bien installer sur notre machine virtuelle, nous allons lancé un conteneur qui affichera hello-world.

- **sudo docker run hello-world**

Ce message nous indique que docker est bien installer

```
esiea@esiea:~/tp/esiea-ressources$ sudo docker run hello-world
[sudo] password for esiea:
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:ac69084025c660510933cca701f615283cdbb3aa0963188770b54c31c8962493
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Les étapes suivantes ont pour but de faciliter la suite de notre travail, nous allons:

- s'assurer du bon lancement des services dockers sur linux ubuntu.
    - **sudo systemctl enable docker.service**
    - **sudo systemctl enable containerd.service**
- empecher d'écrire sudo à chaque commandes docker
    - **sudo groupadd docker**
    - **sudo usermod -aG docker <votre nom utilisateur>**

Il est important d'exécuter ces commandes.

Maintenant que Docker est installer et configuré sur notre machine virtuelle, nous pouvons commencer le build de notre projet

III. Construction des images avec Docker et Compose build

Dans cette partie, nous voulons build les images. Nous avons déjà les Dockerfiles à notre disposition mais chacune se trouve dans son dossier. Dans le fichier docker-compose.build.yml affiché ci-dessus, nous avons donc spécifié dans chaque services le context et le chemin d'accès aux fichiers Dockerfiles.
Les sous-réseaux front-tier et back-tier ont aussi été ajouté. Certains services ont une exposition de port vers l'extérieur.
Le lancement de certains services dépendent d'autres (ex: le lancement du service worker dépend de celui de redis et de postgres).

Pour lancer le build des images, nous utilisons la commande suivante:

- **docker-compose -f docker-compose.build.yml build**

Ci-dessous, le code de docker-compose.build.yml

```yaml
version: '3.8'
services:
  worker:
    build:
      context: ./worker
      dockerfile: Dockerfile
    networks:
      - back-tier
    depends_on:
      redis:
        condition: service_healthy
      db:
        condition: service_healthy
  vote:
    build:
      context: ./vote
      dockerfile: Dockerfile
    volumes:
      - ./vote:/usr/local/app
    ports:
      - "5002:80"
```

```
    networks:
      - front-tier
      - back-tier
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 15s
      timeout: 5s
      retries: 3
      start_period: 10s


  seed-data:
    build:
      context: ./seed-data
      dockerfile: Dockerfile
    depends_on:
      vote:
        condition: service_healthy
    restart: "no"
    networks:
      - front-tier
  redis:
    image: redis:latest
    volumes:
      - "./healthchecks/redis.sh"
    healthcheck:
      test: /healthchecks/redis.sh
      interval: "5s"
    networks:
      - back-tier

  db:
    image: postgres:15-alpine
    volumes:
      - db-data:/var/lib/postgresql/data
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
    networks:
      - back-tier
```

```
  result:
    build:
      context: ./result
      dockerfile: Dockerfile
    volumes:
      - ./result:/usr/local/app
    ports:
      - "5001:80"
      - "127.0.0.1:9229:9229"
    entrypoint: nodemon --inspect=0.0.0.0 server.js
    networks:
      - back-tier
    depends_on:
      db:
        condition: service_healthy

volumes:
  db-data:

networks:
  front-tier:
  back-tier:
```

IV. Automatisation du déploiement de l'application avec Docker Compose

Reprenons depuis le docker-compose.build.yml.

Dans notre situation, nous ne pouvons pas push en l'état nos images à l'extérieur. Nous allons mettre en place un registre local et y push nos images afin de pouvoir exposer l'application à l'extérieur, c'est à dire au localhost.

- **docker run -d -p 5000:5000 --name registry registry:2.7**

Après l'exécution de cette commande, nous créons pour les images build vote, worker, seed-data et result un tag, ce qui nous facilitera la gestion et la référence des images docker. Les tags nous seront surtout utiles dans le fichier compose.yml lorsque nous devrons ciblé les build.

- **docker tag esiea-ressources_vote:latest localhost:5000/esiea-ressources_vote:latest**
- **docker tag esiea-ressources_worker:latest localhost:5000/esiea-ressources_worker:latest**
- **docker tag esiea-ressources_seed-data:latest localhost:5000/esiea-ressources_seed-data:latest**
- **docker tag esiea-ressources_result:latest localhost:5000/esiea-ressources_result:latest**

Rappelons nous, nous avons crée un registre afin d'exposer l'applicatoin en dehors de la machine virtuelle, donc sur notre machine. Il faut fournir les builds de l'application dans le registre. Nous utilisons la commande docker push qui permet de fournir des images docker dans un registre

- **docker push localhost:5000/esiea-ressources_vote:latest**
- **docker push localhost:5000/esiea-ressources_worker:latest**
- **docker push localhost:5000/esiea-ressources_seed-data:latest**
- **docker push localhost:5000/esiea-ressources_result:latest**

Maintenant vérifions que les images builds sont bien dans le registre et "exposer" sur le localhost

curl http://localhost:5000/v2/_catalog

```
esiea@esiea:~/tp/esiea-ressources$ curl http://localhost:5000/v2/_catalog
{"repositories":["esiea-ressources_result","esiea-ressources_seed-data","esiea-ressources_vote","esiea-ressources_worker"]}
```

Nous pouvons maintenant lancer l'application avec la commande suivante

- **docker compose up.**

Ci-dessous, le fichier compose.yml

```yaml
version: '3.8'


services:
  worker:
    image: localhost:5000/esiea-ressources_worker:latest
    depends_on:
      redis:
        condition: service_healthy
      db:
        condition: service_healthy
    networks:
      - cats-or-dogs-network
  vote:
    image: localhost:5000/esiea-ressources_vote:latest
    volumes:
      - ./vote:/usr/local/app
    ports:
      - "127.0.0.1:5002:80"
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 15s
      timeout: 5s
      retries: 3
      start_period: 10s
    networks:
      - cats-or-dogs-network

  seed-data:
    image: localhost:5000/esiea-ressources_seed-data:latest
    depends_on:
      vote:
        condition: service_healthy
    restart: "no"
    networks:
      - cats-or-dogs-network

  result:
    image: localhost:5000/esiea-ressources_result:latest
    volumes:
      - ./result:/usr/local/app
```

```
    ports:
      - "127.0.0.1:5001:80"
      - "127.0.0.1:9229:9229"
    depends_on:
      db:
        condition: service_healthy
    networks:
      - cats-or-dogs-network

  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_PASSWORD: esiea
    volumes:
      - "db-data:/var/lib/postgresql/data"
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
    networks:
      - cats-or-dogs-network

  redis:
    image: redis:latest
    volumes:
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/redis.sh
      interval: "5s"
    networks:
      - cats-or-dogs-network

volumes:
  db-data:

networks:
  cats-or-dogs-network:
```

VI. Conclusion

En conclusion, ce déploiement de l'application HumansBestFriend à l'aide de Docker et Docker Compose a été une expérience enrichissante. Nous avons pu mettre en œuvre une architecture distribuée avec différents services interagissant harmonieusement. Cependant, un défi récurrent auquel nous avons été confrontés était la vérification de l'état de santé des conteneurs Redis et Postgre. Les vérifications de santé fournies entraînaient souvent la déclaration d'un des deux conteneurs comme étant "unhealthy", bloquant ainsi le déploiement de l'application. Cela nécessitait alors un redémarrage du conteneur concerné à chaque fois. Cette situation souligne l'importance cruciale des mécanismes de surveillance et de gestion de la santé des conteneurs dans des déploiements complexes.