Diouf Mohamed, Toussenel Axel
M2 Data Eng

TP Final Docker_HumansBestFriend

Intro

Ce projet a pour but de créer une application web permettant de voter pour son animal de compagnie préféré (chien ou chat), les votes seront ensuite stockés et nous pourrons consulter le résultat du vote. Pour cela, nous avons donc besoin de créer 5 containers :

- Un service “vote” pour l’interface de vote
- Une base Redis
- Une base postgres
- Un nommé “résultat” pour la gestion du résultat des votes
- Un worker pour faire le lien entre redis et postgres

Pour cela, nous avons récupéré le répertoire Github associé puis par étape :

1) Ecrire le fichier docker-compose.build.yml pour créer les images à utiliser
2) Tagger les images déjà créer et les push dans le container Registry pour utiliser nos propres images
3) Ecrire le fichier compose.yml pour créer et lancer nos 5 containers

1) Docker-compose.build.yml

Ce fichier nous permet de configurer la construction de notre application Docker et ainsi, créer les images.

Nous allons décomposer ce fichier :

```
services:
  worker:
    build:
      context: ./worker
```

Le *context* nous indique où se trouve le Dockerfile et mes fichiers nécessaires à la construction de l’image.

```
  vote:
    build:
```

```
    context: ./vote
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost"]
    interval: 15s
    timeout: 5s
    retries: 3
    start_period: 10s
```

Cette section “*healthcheck*” est utilisée pour définir un test de santé pour le service. Un test de santé permet à Docker de vérifier si le service est en cours d'exécution correctement.

On définit dans la section “*interval*” que ces tests de santé vont être fait toutes les 15 sec. “*Timeout”* est le temps d'attente maximal pour qu'un test de santé réussisse. “*Retries*" signifie nombre de tentatives avant de considérer le test de santé comme échoué.

```
seed-data:
  build:
    context: ./seed-data
  restart: "no"

result:
  build:
    context: ./result

db:
  image: postgres:15-alpine
```

Ici, on spécifie l’image Docker que l’on veut utiliser.

```
  healthcheck:
    test: /healthchecks/postgres.sh
    interval: "5s"

redis:
  image: redis
  healthcheck:
    test: /healthchecks/redis.sh
    interval: "5s"
```

2) compose.yml

Voici une configuration Docker Compose qui définit plusieurs services pour l'application "HumansBestFriend".

```
services:
  vote:
    image: localhost:5000/vote:v0
    depends_on:
      - redis
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 15s
      timeout: 5s
      retries: 3
      start_period: 10s
    volumes:
      - ./vote:/usr/local/app
```

Cette section montre de quelle manière sont configurés les volumes. Par exemple, le service “*vote*” monte le répertoire local ./vote dans le répertoire /usr/local/app du container.

```
    ports:
      - "5002:80"
```

Spécifie la liaison de ports entre l'hôte et le container. Par exemple, le service vote expose le port 80 à l'hôte sur le port 5002.

```
    networks:
      - humansbestfriend-network
```

Spécifie les réseaux auxquels chaque service est connecté. Ici, tous les services sont connectés au réseau “*humansbestfriend-network*”.

```

  result:
    image: localhost:5000/result:v0
    build: ./result
```

Pour le service “*result*”, cela spécifie le chemin vers le dossier contenant le Dockerfile pour construire l'image localement. Il indique que l'image doit être construite à partir du contexte ./result.

```

    entrypoint: ["nodemon", "--inspect=0.0.0.0", "server.js"]
```

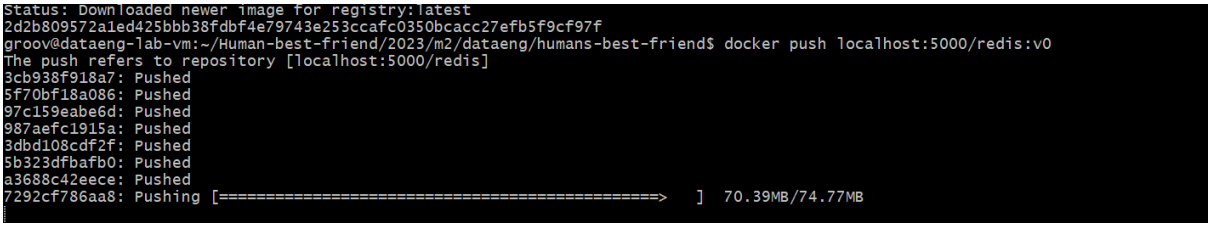
Pour le service “*result*”, cela spécifie la commande d'entrée à exécuter lorsque le conteneur démarre.

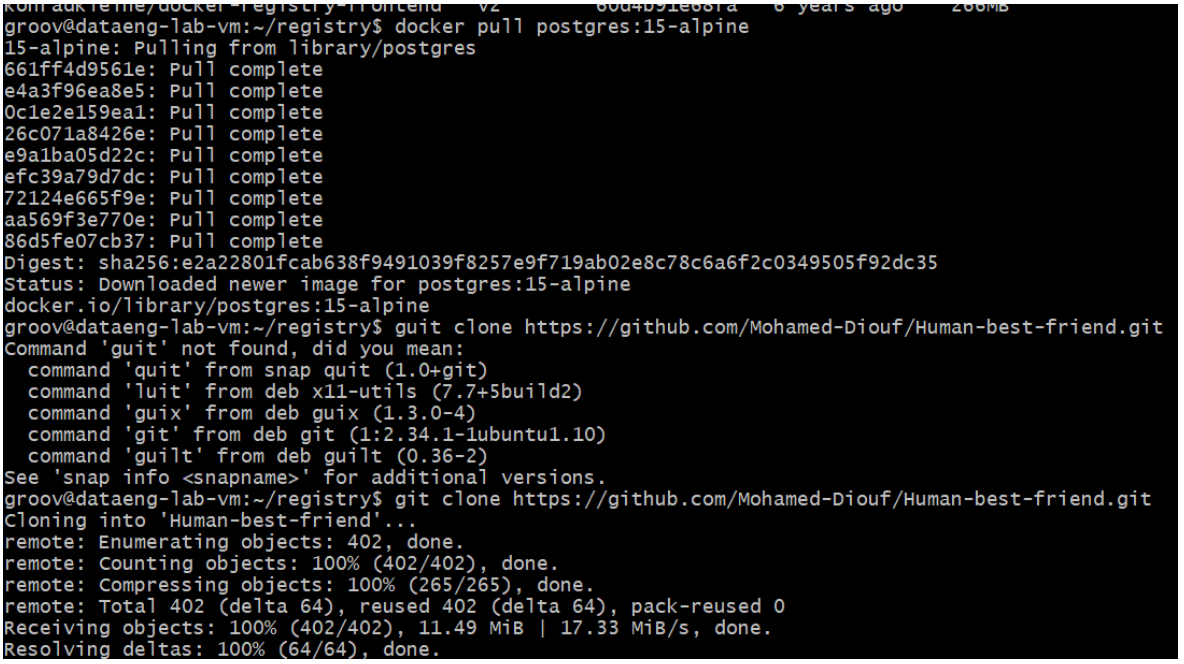
```
    depends_on:
      - db
    volumes:
      - ./result:/usr/local/app
    ports:
      - "5001:80"
      - "127.0.0.1:9229:9229"
    networks:
      - humansbestfriend-network

  worker:
    image: localhost:5000/worker:v0
    build:
      context: ./worker
    depends_on:
      - redis
      - db
    networks:
      - humansbestfriend-network
```

3) Construction de l’application

Pour build nos images, voici la commande que nous avons utilisée :

Ensuite, nous avons pull Postgres et push dans Registry :





Ensuite, nous avons exécuté la commande pour créer nos containers :