Lakbiri Tarek, Nano Alexandre, Kpotogbey Killian, Diderot Hugo

Projet de Virtualisation et Conteneurisation

HumansBestFriend app? CATs or DOGs?

A simple distributed application running across multiple Docker containers.

Sommaire

Introduction

Le présent rapport détaille la réalisation d'un projet axé sur la virtualisation et la conteneurisation d'une application distribuée simple, baptisée "HumansBestFriend app? CATs or DOGs?". L'objectif principal de ce projet est de déployer cette application sur plusieurs conteneurs Docker au sein d'une machine virtuelle. Pour ce faire, nous avons suivi les directives et la documentation de Docker, telles qu'abordées durant notre cours, pour l'installation et la configuration nécessaire.

Exigences

Le projet sera réalisé à l'intérieur d'une machine virtuelle qui exécute docker et docker compose. Nous suivrons la documentation de Docker comme nous l'avons vu pendant le cours pour les instructions d'installation si vous ne l'avez pas encore.

Pour commencer

Cette solution utilise Python, Node.js, .NET, avec Redis pour la messagerie et Postgres pour le stockage.

Tâches

Nous créons un fichier appelé docker-compose.build.yml (pas de conteneurs en cours d'exécution). Ce fichier sera responsable de la construction des images de l'application à partir du contenu du fichier Docker fourni.

Worker Service

Le worker service dépend de redis et de db. On Utilise ce qui suit dans notre fichier de composition

```
depends_on:
  redis:
    condition: service_healthy
  db:
    condition: service_healthy
```

Le worker doit se trouver à l'intérieur du réseau back-tier

```
FROM --platform=${BUILDPLATFORM} mcr.microsoft.com/dotnet/sdk:7.0 as build
ARG TARGETPLATFORM
ARG TARGETARCH
ARG BUILDPLATFORM
RUN echo "I am running on $BUILDPLATFORM, building for $TARGETPLATFORM"

WORKDIR /source
COPY *.csproj .
RUN dotnet restore -a $TARGETARCH

COPY . .
RUN dotnet publish -c release -o /app -a $TARGETARCH --self-contained false --no-restore

# app image
FROM mcr.microsoft.com/dotnet/runtime:7.0
WORKDIR /app
COPY --from=build /app .
ENTRYPOINT ["dotnet", "Worker.dll"]
```

Vote Service

mapper un volume dans /usr/local/app qui se trouve à l'intérieur du conteneur. Il doit s'agir d'un montage bind, par exemple

```
volumes:
     - ./vote:/usr/local/app
```

Le service de vote écoute sur le port 80.

Le service de vote doit se trouver à l'intérieur de deux réseaux : front-tier et back-tier.

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

Ici, nous pouvons voir que le port d'écoute de nos sites webs sont 5001 et 5002.

```yaml
services:
  vote:
    build:
      context: ./vote
      target: dev
    depends_on:
      redis:
        condition: service_healthy
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost
      interval: 15s
      timeout: 5s
      retries: 3
      start_period: 10s
    volumes:
     - ./vote:/usr/local/app
    ports:
      - "5002:80"
    networks:
      - front-tier
      - back-tier

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

  worker:
    build:
      context: ./worker
    depends_on:
      redis:
        condition: service_healthy
      db:
```

L'application de vote fonctionnera à l'adresse http://localhost:5002, et les résultats seront disponibles à l'adresse http://localhost:5001.

Seed-Data

On ajoute les seed

```yaml
profiles: ["seed"]
    depends_on:
      vote:
        condition: service_healthy
    restart: "no"
```

Il se trouver à l'intérieur du réseau front-tier

Résultat

A l'intérieur du conteneur, le port est 80. Nous exposons l'extérieur avec 5001:

```
entrypoint: nodemon --inspect=0.0.0.0 server.js
    depends_on:
      db:
        condition: service_healthy
    volumes:
      - ./result:/usr/local/app
    ports:
      - "5001:80"
      - "127.0.0.1:9229:9229"
```

Les images qui sont construites doivent être publiées d'abord dans notre registre publish docker et ensuite dans un registre privé. Ensuite nous nous assurons d'avoir un frontend pour le registre privé comme nous l'avons vu dans le cours.

Nous créons un autre fichier appelé compose.yml et il sera responsable du déploiement de l'application et de tous les conteneurs nécessaires. On s'assure que les images font référence à celle de notre registre privé.

Postgres DB

Il doit se trouver à l'intérieur du réseau back-tier

Nous utilisons les éléments suivants dans votre fichier de composition. On utilise postgres:15-alpine pour l'image postgres.

```
volumes:
  - "db-data:/var/lib/postgresql/data"
  - "./healthchecks:/healthchecks"
healthcheck:
  test: /healthchecks/postgres.sh
  interval: "5s"
```

Pour le volume db-data, on s'assure de le créer dans notre fichier de composition

Redis Service

Nous utilisons le texte ci-dessous dans notre fichier de composition.

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

Il doit se trouver à l'intérieur du réseau back-tier

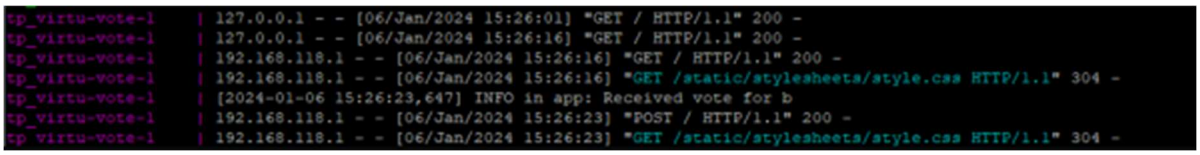
La commande *docker compose up* lance le déploiement de notre application en créant et en démarrant tous les services nécessaires spécifiés dans notre fichier docker-compose.build.yml.

```
ki@vm1:~/TP_virtu$ docker compose up
WARN[0000] Found multiple config files with supported names: /home/ki/TP_virtu/compose.yml, /home/ki/TP_virtu/docker-compose.yml
WARN[0000] Using /home/ki/TP_virtu/compose.yml
WARN[0000] Found multiple config files with supported names: /home/ki/TP_virtu/compose.yml, /home/ki/TP_virtu/docker-compose.yml
WARN[0000] Using /home/ki/TP_virtu/compose.yml
[+] Running 5/0
 ✔ Container tp_virtu-db-1      Created
 ✔ Container tp_virtu-redis-1   Created
 ✔ Container tp_virtu-result-1  Created
 ✔ Container tp_virtu-worker-1  Created
 ✔ Container tp_virtu-vote-1    Created
Attaching to tp_virtu-db-1, tp_virtu-redis-1, tp_virtu-result-1, tp_virtu-vote-1, tp_virtu-worker-1
tp_virtu-redis-1   | 1:C 06 Jan 2024 15:25:39.886 # WARNING Memory overcommit must be enabled! Without it, a background save or replication may fail unde
w memory condition. Being disabled, it can also cause failures without low memory condition, see https://github.com/jemalloc/jemalloc/issues/1328. To fix
s issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
tp_virtu-redis-1   | 1:C 06 Jan 2024 15:25:39.886 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
tp_virtu-redis-1   | 1:C 06 Jan 2024 15:25:39.886 * Redis version=7.2.3, bits=64, commit=00000000, modified=0, pid=1, just started
tp_virtu-redis-1   | 1:C 06 Jan 2024 15:25:39.886 # Warning: no config file specified, using the default config. In order to specify a config file use re
server /path/to/redis.conf
tp_virtu-redis-1   | 1:M 06 Jan 2024 15:25:39.887 * monotonic clock: POSIX clock_gettime
tp_virtu-redis-1   | 1:M 06 Jan 2024 15:25:39.888 * Running mode=standalone, port=6379.
tp_virtu-redis-1   | 1:M 06 Jan 2024 15:25:39.889 * Server initialized
tp_virtu-redis-1   | 1:M 06 Jan 2024 15:25:39.889 * Loading RDB produced by version 7.2.3
tp_virtu-redis-1   | 1:M 06 Jan 2024 15:25:39.889 * RDB age 4 seconds
tp_virtu-redis-1   | 1:M 06 Jan 2024 15:25:39.889 * RDB memory usage when created 1.09 Mb
tp_virtu-redis-1   | 1:M 06 Jan 2024 15:25:39.889 * Done loading RDB, keys loaded: 0, keys expired: 0.
tp_virtu-redis-1   | 1:M 06 Jan 2024 15:25:39.890 * DB loaded from disk: 0.001 seconds
tp_virtu-redis-1   | 1:M 06 Jan 2024 15:25:39.890 * Ready to accept connections tcp
tp_virtu-db-1      |
tp_virtu-db-1      | PostgreSQL Database directory appears to contain a database; Skipping initialization
tp_virtu-db-1      |
tp_virtu-db-1      | 2024-01-06 15:25:39.914 UTC [1] LOG:  starting PostgreSQL 15.5 on x86_64-pc-linux-musl, compiled by gcc (Alpine 13.2.1_git20231014)
.1 20231014, 64-bit
tp_virtu-db-1      | 2024-01-06 15:25:39.915 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
tp_virtu-db-1      | 2024-01-06 15:25:39.915 UTC [1] LOG:  listening on IPv6 address "::", port 5432
tp_virtu-db-1      | 2024-01-06 15:25:39.917 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
tp_virtu-db-1      | 2024-01-06 15:25:39.920 UTC [25] LOG:  database system was shut down at 2024-01-06 15:25:35 UTC
tp_virtu-db-1      | 2024-01-06 15:25:39.925 UTC [1] LOG:  database system is ready to accept connections
tp_virtu-worker-1  | Connected to db
tp_virtu-worker-1  | Found redis at 172.20.0.2
tp_virtu-worker-1  | Connecting to redis
tp_virtu-vote-1    |  * Serving Flask app 'app'
tp_virtu-vote-1    |  * Debug mode: on
tp_virtu-vote-1    | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
tp_virtu-vote-1    |  * Running on all addresses (0.0.0.0)
tp_virtu-vote-1    |  * Running on http://127.0.0.1:80
tp_virtu-vote-1    |  * Running on http://172.20.0.5:80
tp_virtu-vote-1    | Press CTRL+C to quit
tp_virtu-vote-1    |  * Restarting with watchdog (inotify)
tp_virtu-result-1  | [nodemon] 3.0.2
tp_virtu-result-1  | [nodemon] to restart at any time, enter `rs`
tp_virtu-result-1  | [nodemon] watching path(s): *.*
tp_virtu-result-1  | [nodemon] watching extensions: js,mjs,cjs,json
tp_virtu-result-1  | [nodemon] starting `node --inspect=0.0.0.0 server.js`
tp_virtu-result-1  | Debugger listening on ws://0.0.0.0:9229/7e9b4b9a-4239-49c4-9dd6-86372d512de9
tp_virtu-result-1  | For help, see: https://nodejs.org/en/docs/inspector
tp_virtu-vote-1    |  * Debugger is active!
tp_virtu-vote-1    |  * Debugger PIN: 513-437-365
tp_virtu-result-1  | Sat, 06 Jan 2024 15:25:46 GMT body-parser deprecated undefined extended: provide extended option at server.js:67:17
tp_virtu-result-1  | App running on port 80
tp_virtu-result-1  | Connected to db
```

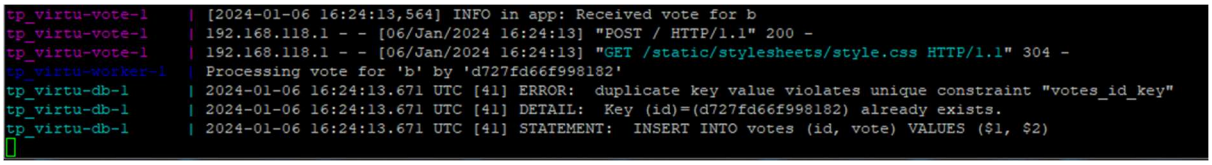
Lorsque l'on va sur le site http://192.168.118.129:5001, notre page web fonctionne et nous pouvons voir qui a voté.

Lorsque l'on va sur le site http://192.168.118.129:5002, notre page web fonctionne et nous pouvons voter.

Ici, nous pouvons voir que le choix B à reçus un vote.



Lorsque l'on essaie de voter une deuxième fois, nous sommes bloqués et nous avons une erreur pour pas qu'un utilisateur normal puisse voter plusieurs fois.



En conclusion, ce travail a renforcé nos compétences en conception, déploiement et sécurité des applications basées sur des microservices. La maîtrise des interactions entre services et la gestion des flux de données ont été des éléments clés de notre réussite, illustrant notre compréhension des défis et opportunités offerts par la virtualisation et la conteneurisation.

En résumé, ce projet a consolidé notre expertise dans ce domaine et nous a préparés à relever avec confiance des défis similaires à l'avenir, renforçant ainsi notre compréhension des technologies modernes de développement logiciel.