ESIEA

4A — Décembre 2023

Projet de Virtualisation

classe 47

Semestre 7

—

2023-2024

Étudiants :

- Jean-Luc LAURENT
- Thomas COSSET
- Théo BACHELERY

SOMMAIRE

INTRODUCTION

Le projet s’inscrit dans le cours de virtualisation, dans celui-ci, nous avons étudié les méthodes de déploiement d'applications conteneurisées. L'objectif principal de ce projet est de créer et déployer une page web permettant aux utilisateurs de voter pour leur animal préféré entre chiens et chats. Pour ce faire, nous avons utilisé Docker pour la gestion des conteneurs et avons implémenté différentes technologies telles que Python, Node.js, .NET, Redis et Postgres.

ENVIRONNEMENT DOCKER

Configuration de l'environnement Docker

La mise en place de l'environnement Docker a été une étape cruciale pour le succès du projet. Nous avons choisi d'utiliser une machine virtuelle basée sur Ubuntu 20.04 pour héberger notre application. Les étapes suivantes ont été suivies attentivement en référence à la documentation officielle de Docker.

Installation de Docker :

Nous avons suivi les instructions officielles pour installer Docker sur notre machine virtuelle Ubuntu 20.04. Ces instructions garantissent une installation propre et optimisée du moteur Docker.

Documentation d'Installation Docker pour Ubuntu

Post-installation Docker :

Après l'installation, nous avons suivi les recommandations post-installation pour configurer Docker de manière à ce qu'il puisse être utilisé sans privilèges root. Cela inclut l'ajout de l'utilisateur au groupe Docker.

Documentation Post-installation Docker pour Linux

Installation de Docker Compose :

Docker Compose est un outil essentiel pour la gestion des applications multi-conteneurs. Nous avons suivi les instructions pour installer Docker Compose sur notre machine virtuelle.

Documentation Docker Compose

L'ensemble de ces étapes garantit un environnement Docker fonctionnel et prêt à héberger notre application "HumansBestFriend".

Remarques supplémentaires:

- Nous avons veillé à ce que toutes les dépendances requises soient correctement installées sur la machine virtuelle, notamment Python, Node.js, et .NET.

CONSTRUCTION DES IMAGES DOCKER

Le fichier docker-compose.build.yml joue un rôle crucial dans la construction et la publication des images Docker nécessaires pour notre application. Ce fichier orchestre le processus de création des images à partir des fichiers Dockerfile de chaque service.

Voici comment nous l'avons implémenté :

```
# Contenu du fichier docker-compose.build.yml

services:
  vote:
    image: docker/esiea_vote
    depends_on:
      redis:
        condition: service_healthy
    ports:
      - "5002:80"
    networks:
      - front-tier
      - back-tier

  result:
    image: docker/esiea_result
    depends_on:
      db:
        condition: service_healthy
    ports:
      - "5001:80"
    networks:
      - front-tier
      - back-tier

  worker:
    image: docker/esiea_worker
    depends_on:
```

```
      redis:
        condition: service_healthy
      db:
        condition: service_healthy
    networks:
      - back-tier

  redis:
    image: redis:alpine
    volumes:
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/redis.sh
      interval: "5s"
    networks:
      - back-tier

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

volumes:
  db-data:

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

La construction des images Docker est déclenchée par la commande docker-compose build à partir du répertoire où se trouve le fichier docker-compose.build.yml. Cette commande va

parcourir chaque service défini dans le fichier et construire l'image Docker correspondante en utilisant le Dockerfile spécifié dans le répertoire du service.

Après la construction des images, celles-ci peuvent être publiées sur un registre Docker pour être partagées et utilisées par d'autres. Cependant, dans notre cas, on utilise déjà des images provenant d'un dépôt similaire, donc la publication peut ne pas être nécessaire.

Remarques supplémentaires:

- Nous nous sommes bien assuré que le chemin vers le répertoire contenant le fichier docker-compose.build.yml est le répertoire actuel lorsque l'on exécute la commande de construction.

- Les dépendances entre les services, définies par depends_on, garantissent que les services qui en ont besoin sont prêts avant la construction de l'image.

Après avoir exécuté avec succès la commande de construction des images, nous sommes donc passé à la prochaine étape : le déploiement de l'application à l'aide du fichier compose.yml.

DÉPLOIEMENT DE L'APPLICATION

Le fichier compose.yml est essentiel pour le déploiement de notre application conteneurisée. Il définit la configuration des services, des dépendances, des volumes, des ports et des réseaux nécessaires à notre application. Voici comment nous avons configuré ce fichier pour "HumansBestFriend":

```
# Contenu du fichier compose.yml

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
    # use nodemon rather than node for local dev
    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:
      condition: service_healthy
  networks:
    - back-tier

redis:
  image: redis:alpine
  volumes:
    - "./healthchecks:/healthchecks"
  healthcheck:
    test: /healthchecks/redis.sh
    interval: "5s"
  networks:
    - back-tier

db:
  image: postgres:15-alpine
  environment:
    POSTGRES_USER: "postgres"
    POSTGRES_PASSWORD: "postgres"
  volumes:
    - "db-data:/var/lib/postgresql/data"
    - "./healthchecks:/healthchecks"
  healthcheck:
    test: /healthchecks/postgres.sh
```

```
      interval: "5s"
    networks:
      - back-tier

  seed:
    build: ./seed-data
    profiles: ["seed"]
    depends_on:
      vote:
        condition: service_healthy
    networks:
      - front-tier
    restart: "no"

volumes:
  db-data:

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

Déploiement Manuel :

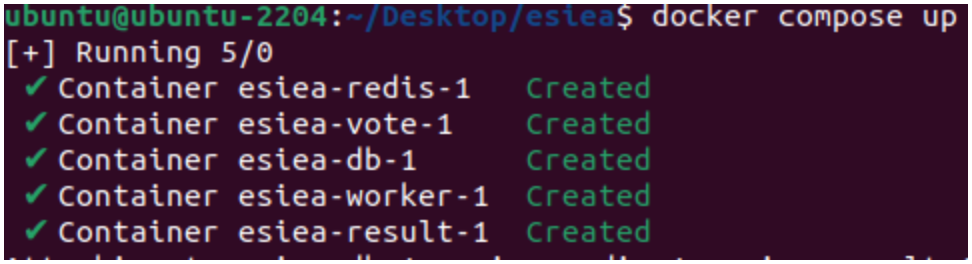
Avant tout déploiement d'application, il est primordial de lancer la commande "docker compose build" qui nous permettra d'identifier une image en mauvaise santé. Cela nous montrera si le docker-compose.build.yml est bien construit.

```
docker compose build
```

```
ubuntu@ubuntu-2204:~/Desktop/esiea$ docker compose build
[+] Building 2.4s (39/39) FINISHED                                docker:default
 => [result internal] load build definition from Dockerfile                0.1s
 => => transferring dockerfile: 525B                                       0.0s
 => [result internal] load .dockerignore                                   0.0s
 => => transferring context: 2B                                            0.0s
 => [result internal] load metadata for docker.io/library/node:18-slim     1.6s
 => [worker internal] load build definition from Dockerfile                0.1s
 => => transferring dockerfile: 1.04kB                                     0.0s
 => [worker internal] load .dockerignore                                   0.1s
 => => transferring context: 2B                                            0.0s
 => [worker internal] load metadata for mcr.microsoft.com/dotnet/runtime:  0.5s
 => [worker internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0   0.5s
 => [vote internal] load build definition from Dockerfile                  0.2s
 => => transferring dockerfile: 1.09kB                                     0.0s
 => [vote internal] load .dockerignore                                     0.1s
 => => transferring context: 2B                                            0.0s
 => [vote internal] load metadata for docker.io/library/python:3.11-slim   1.0s
 => [worker build 1/7] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:4be8f  0.0s
 => [worker internal] load build context                                   0.0s
 => => transferring context: 95B                                           0.0s
 => [worker stage-1 1/3] FROM mcr.microsoft.com/dotnet/runtime:7.0@sha256  0.0s
 => CACHED [worker stage-1 2/3] WORKDIR /app                               0.0s
 => CACHED [worker build 2/7] RUN echo "I am running on linux/amd64, buil  0.0s
 => CACHED [worker build 3/7] WORKDIR /source                              0.0s
 => CACHED [worker build 4/7] COPY *.csproj .                              0.0s
 => CACHED [worker build 5/7] RUN dotnet restore -a amd64                  0.0s
 => CACHED [worker build 6/7] COPY . .                                     0.0s
 => CACHED [worker build 7/7] RUN dotnet publish -c release -o /app -a am  0.0s
 => CACHED [worker stage-1 3/3] COPY --from=build /app .                   0.0s
 => [worker] exporting to image                                            0.1s
 => => exporting layers                                                    0.0s
 => => writing image sha256:bcbe15e21c2f1e4ab6f274dfe2aaa4fcbf499856855a3  0.0s
 => => naming to docker.io/library/esiea-worker                            0.0s
 => [result 1/7] FROM docker.io/library/node:18-slim@sha256:fe687021c0638  0.0s
 => [result internal] load build context                                   0.1s
 => => transferring context: 561B                                          0.0s
 => [vote base 1/5] FROM docker.io/library/python:3.11-slim@sha256:8f64a6  0.0s
 => [vote internal] load build context                                     0.1s
 => => transferring context: 37B                                           0.0s
 => CACHED [vote base 2/5] RUN apt-get update &&     apt-get install -y -  0.0s
 => CACHED [vote base 3/5] WORKDIR /usr/local/app                          0.0s
 => CACHED [vote base 4/5] COPY requirements.txt ./requirements.txt       0.0s
 => CACHED [vote base 5/5] RUN pip install --no-cache-dir -r requirements  0.0s
 => CACHED [vote dev 1/1] RUN pip install watchdog                         0.0s
```

On exécute la commande “docker compose up” à partir du répertoire contenant le fichier compose.yml. Cette commande démarre tous les services définis dans le fichier, construisant les images au besoin, créant les conteneurs, et les reliant selon les spécifications du fichier.

```
docker compose up
```



Les images (locales et publiques) sont bien déployés lors de la commande docker compose up

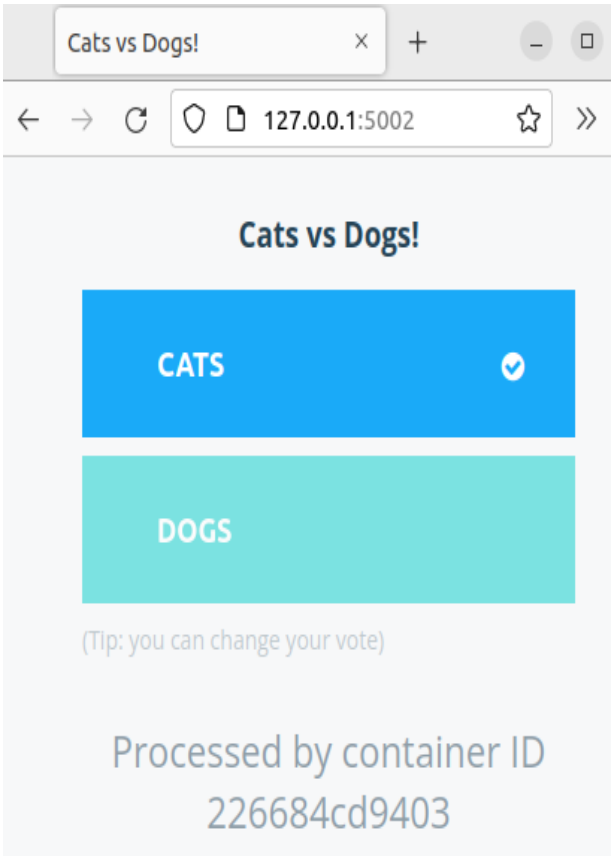
```
Attaching to esiea-db-1, esiea-redis-1, esiea-result-1, esiea-vote-1, esiea-worker-1
esiea-db-1      |
esiea-db-1      | PostgreSQL Database directory appears to contain a database; Skipping initialization
esiea-db-1      |
esiea-redis-1   | 1:C 06 Jan 2024 09:45:34.399 # WARNING Memory overcommit must be enabled! Without it, a background save or replication
ause failures without low memory condition, see https://github.com/jemalloc/jemalloc/issues/1328. To fix this issue add 'vm.overcommit_me
sysctl vm.overcommit_memory=1' for this to take effect.
esiea-redis-1   | 1:C 06 Jan 2024 09:45:34.399 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
esiea-redis-1   | 1:C 06 Jan 2024 09:45:34.399 * Redis version=7.2.3, bits=64, commit=00000000, modified=0, pid=1, just started
esiea-redis-1   | 1:C 06 Jan 2024 09:45:34.399 # Warning: no config file specified, using the default config. In order to specify a confi
esiea-redis-1   | 1:M 06 Jan 2024 09:45:34.400 * monotonic clock: POSIX clock_gettime
esiea-redis-1   | 1:M 06 Jan 2024 09:45:34.412 * Running mode=standalone, port=6379.
esiea-redis-1   | 1:M 06 Jan 2024 09:45:34.415 * Server initialized
esiea-redis-1   | 1:M 06 Jan 2024 09:45:34.421 * Loading RDB produced by version 7.2.3
esiea-redis-1   | 1:M 06 Jan 2024 09:45:34.422 * RDB age 150850 seconds
esiea-redis-1   | 1:M 06 Jan 2024 09:45:34.422 * RDB memory usage when created 0.85 Mb
esiea-redis-1   | 1:M 06 Jan 2024 09:45:34.422 * Done loading RDB, keys loaded: 0, keys expired: 0.
esiea-redis-1   | 1:M 06 Jan 2024 09:45:34.422 * DB loaded from disk: 0.006 seconds
esiea-redis-1   | 1:M 06 Jan 2024 09:45:34.422 * Ready to accept connections tcp
esiea-db-1      | 2024-01-06 09:45:34.712 UTC [1] LOG:  starting PostgreSQL 15.5 on x86_64-pc-linux-musl, compiled by gcc (Alpine 13.2.1_
esiea-db-1      | 2024-01-06 09:45:34.712 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
esiea-db-1      | 2024-01-06 09:45:34.712 UTC [1] LOG:  listening on IPv6 address "::", port 5432
esiea-db-1      | 2024-01-06 09:45:34.727 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
esiea-db-1      | 2024-01-06 09:45:34.754 UTC [23] LOG:  database system was interrupted; last known up at 2024-01-04 15:56:29 UTC
esiea-db-1      | 2024-01-06 09:45:36.181 UTC [23] LOG:  database system was not properly shut down; automatic recovery in progress
esiea-db-1      | 2024-01-06 09:45:36.196 UTC [23] LOG:  redo starts at 0/15856F0
esiea-db-1      | 2024-01-06 09:45:36.196 UTC [23] LOG:  invalid record length at 0/15857D8: wanted 24, got 0
esiea-db-1      | 2024-01-06 09:45:36.196 UTC [23] LOG:  redo done at 0/15857A0 system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed:
esiea-db-1      | 2024-01-06 09:45:36.212 UTC [21] LOG:  checkpoint starting: end-of-recovery immediate wait
esiea-db-1      | 2024-01-06 09:45:36.252 UTC [21] LOG:  checkpoint complete: wrote 3 buffers (0.0%); 0 WAL file(s) added, 0 removed, 0 r
 longest=0.004 s, average=0.003 s; distance=0 kB, estimate=0 kB
esiea-db-1      | 2024-01-06 09:45:36.272 UTC [1] LOG:  database system is ready to accept connections
esiea-result-1  | [nodemon] 3.0.2
esiea-result-1  | [nodemon] to restart at any time, enter `rs`
esiea-result-1  | [nodemon] watching path(s): *.*
esiea-result-1  | [nodemon] watching extensions: js,mjs,cjs,json
esiea-result-1  | [nodemon] starting `node --inspect=0.0.0.0 server.js`
esiea-vote-1    |  * Serving Flask app 'app'
esiea-vote-1    |  * Debug mode: on
esiea-vote-1    | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
esiea-vote-1    |  * Running on all addresses (0.0.0.0)
esiea-vote-1    |  * Running on http://127.0.0.1:80
esiea-vote-1    |  * Running on http://192.168.80.4:80
esiea-vote-1    | Press CTRL+C to quit
esiea-vote-1    |  * Restarting with watchdog (inotify)
esiea-result-1  | Debugger listening on ws://0.0.0.0:9229/aa300d31-748e-4991-8f7e-d8313bc35c1c
esiea-result-1  | For help, see: https://nodejs.org/en/docs/inspector
esiea-worker-1  | Connected to db
```

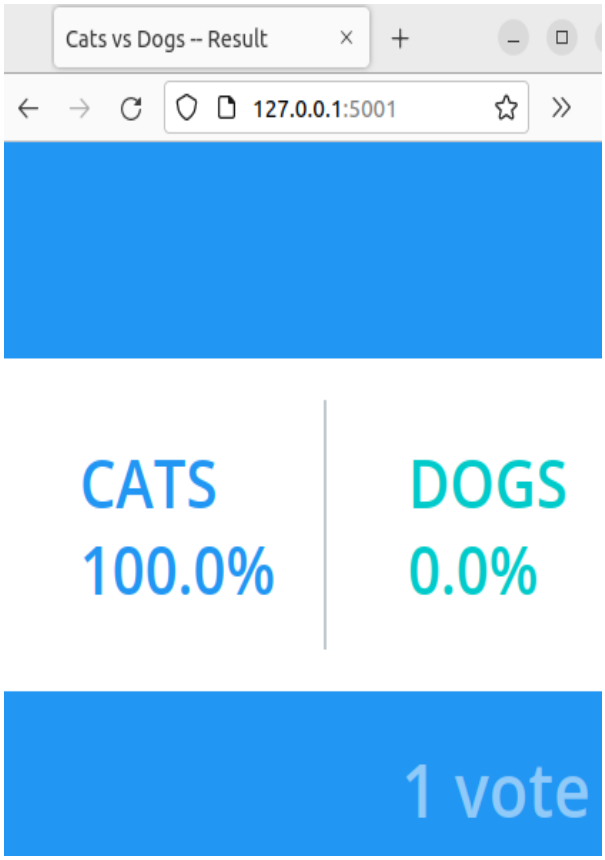
```
esiea-vote-1     | 127.0.0.1 - - [06/Jan/2024 09:45:57] "GET / HTTP/1.1" 200 -
esiea-vote-1     | 127.0.0.1 - - [06/Jan/2024 09:46:13] "GET / HTTP/1.1" 200 -
esiea-vote-1     | 127.0.0.1 - - [06/Jan/2024 09:46:28] "GET / HTTP/1.1" 200 -
esiea-vote-1     | 192.168.80.1 - - [06/Jan/2024 09:46:34] "GET / HTTP/1.1" 200 -
esiea-vote-1     | 192.168.80.1 - - [06/Jan/2024 09:46:35] "GET /static/stylesheets/style.css HTTP/1.1" 200 -
esiea-vote-1     | 127.0.0.1 - - [06/Jan/2024 09:46:43] "GET / HTTP/1.1" 200 -
esiea-vote-1     | [2024-01-06 09:46:44,059] INFO in app: Received vote for a
esiea-vote-1     | 192.168.80.1 - - [06/Jan/2024 09:46:44] "POST / HTTP/1.1" 200 -
esiea-vote-1     | 192.168.80.1 - - [06/Jan/2024 09:46:44] "GET /static/stylesheets/style.css HTTP/1.1" 304 -
esiea-worker-1   | Processing vote for 'a' by '182f348f0585cd'
```

Sur cette image, nous pouvons observer plusieurs choses intéressantes, à commencer par la requête GET envoyée par notre navigateur vers l'adresse du projet (127.0.0.1:5001) signifiant que le projet est bien accessible en local.
Nous voyons également que le système de vote accessible depuis 127.0.0.1:5002 fonctionne correctement car le vote depuis notre navigateur est bien pris en compte (Processing vote for 'a' by '182f348f0585cd')

```
esiea-db-1       | 2024-01-06 10:13:17.213 UTC [40] ERROR:  duplicate key value violates unique constraint "votes_id_key"
esiea-db-1       | 2024-01-06 10:13:17.213 UTC [40] DETAIL:  Key (id)=(182f348f0585cd) already exists.
esiea-db-1       | 2024-01-06 10:13:17.213 UTC [40] STATEMENT:  INSERT INTO votes (id, vote) VALUES ($1, $2)
```

D'ailleurs il nous est formellement interdit de voter à nouveau depuis notre navigateur pour le même choix. Cela générera une erreur de clé dans la base de données esiea-db-1.





Pour arrêter l'application, nous pouvons utiliser le raccourci clavier ctrl+c plus la commande :

```
docker compose down -v
```

```
^CGracefully stopping... (press Ctrl+C again to force)
Aborting on container exit...
[+] Stopping 5/5
 ✔ Container esiea-worker-1  Stopped
 ✔ Container esiea-vote-1    Stopped
 ✔ Container esiea-result-1  Stopped
 ✔ Container esiea-redis-1   Stopped
 ✔ Container esiea-db-1      Stopped
canceled
ubuntu@ubuntu-2204:~/Desktop/esiea$ docker compose down -v
[+] Running 8/8
 ✔ Container esiea-vote-1    Removed
 ✔ Container esiea-result-1  Removed
 ✔ Container esiea-worker-1  Removed
 ✔ Container esiea-redis-1   Removed
 ✔ Container esiea-db-1      Removed
 ✔ Volume esiea_db-data      Removed
 ✔ Network esiea_front-tier  Removed
 ✔ Network esiea_back-tier   Removed
```

Cette commande aura pour effet de supprimer les volumes déployés lors de l'utilisation de l'application. Cela aura pour effet de corriger certains problèmes de doublon.

Remarques supplémentaires :

- Les services définis dans le fichier compose.yml définissent les détails spécifiques de chaque composant de l'application, tels que le service de vote, le service de résultat, le worker, etc.

- Les dépendances entre les services, spécifiées par depends_on, garantissent que les services nécessaires sont prêts avant le démarrage.

Une fois le déploiement réussi, nous pouvions accéder à l'application "HumansBestFriend" via les URLs spécifiées dans le fichier compose.yml. Nous avons fait en sorte de vérifier l'état de l'application à l'aide des outils de surveillance et des vérifications de santé spécifiés dans les fichiers de configuration.

ARCHITECTURE DE L'APPLICATION

L'architecture de l'application "HumansBestFriend" repose sur une approche conteneurisée, utilisant Docker pour la gestion des services. Voici une description de l'architecture de l'application :

Vote App (Python):

- Cette partie de l'application fournit une interface web accessible à l'adresse http://localhost:5002. Les utilisateurs peuvent voter pour leur animal préféré entre chiens et chats. Le service utilise le langage Python et est configuré pour écouter sur le port 80 à l'intérieur du conteneur, mappé sur le port 5002 à l'extérieur.

Result App (Node.js):

- Le service Result App affiche en temps réel les résultats des votes. Il utilise Node.js et est accessible à l'adresse http://localhost:5001. Le service est configuré pour écouter sur le port 80 à l'intérieur du conteneur, mappé sur le port 5001 à l'extérieur.

Worker (.NET):

- Le Worker est implémenté en utilisant le framework .NET. Il consomme les votes générés par le service de vote et les stocke dans la base de données. Ce service dépend du fonctionnement sain des services Redis et de la base de données.

Redis (Messaging):

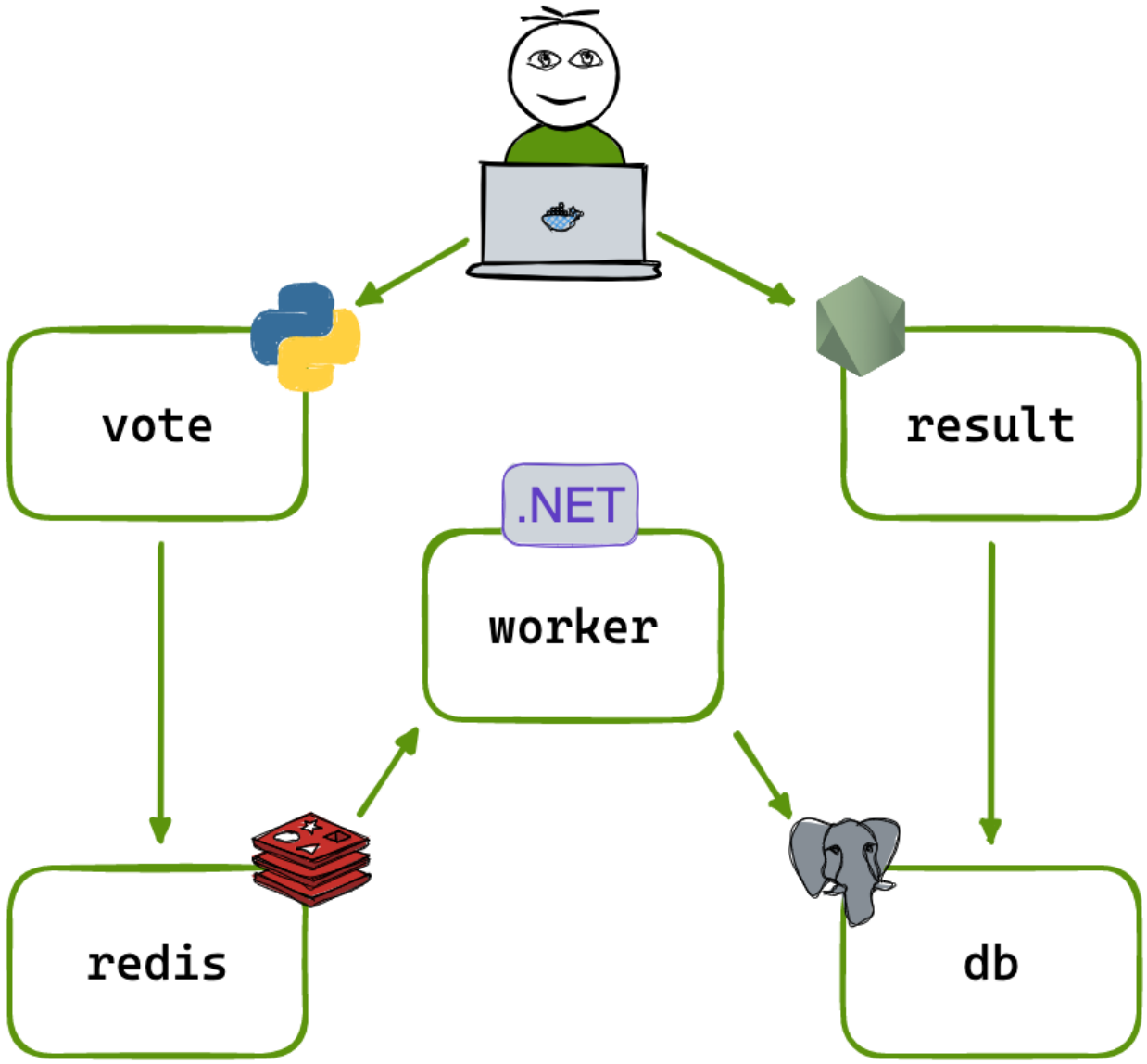
- Redis est utilisé comme système de messagerie pour collecter les nouveaux votes. Il fonctionne en tant que conteneur Docker avec l'image Redis Alpine. Un volume Docker est monté pour stocker les scripts de vérification de santé.

Postgres (Database):

- PostgreSQL est utilisé comme base de données pour stocker les votes. Le service utilise l'image Postgres Alpine, avec des scripts de vérification de santé pour s'assurer du bon fonctionnement du service. Un volume Docker est monté pour stocker les données de la base de données de manière persistante.

Seed Service:

- Un service de semence est inclus pour générer des votes initiaux. Il est construit à partir du répertoire `seed-data` et dépend du service de vote. Il est configuré pour ne pas redémarrer automatiquement après son arrêt.



L'ensemble de l'architecture est organisé dans un réseau Docker, avec des réseaux spécifiques pour la communication interne entre les services (`front-tier` et `back-tier`).

Remarques supplémentaires:

- Les services communiquent entre eux via les réseaux spécifiés dans le fichier `compose.yml`.
- Les dépendances entre les services garantissent un démarrage ordonné pour assurer une application fonctionnelle.
- L'application utilise une approche simple de messagerie, stockage, et interface utilisateur pour démontrer des concepts de conteneurisation.

Cette architecture met en œuvre une approche distribuée à petite échelle pour illustrer l'utilisation de Docker dans des environnements multi-conteneurs. Pour des scénarios de production, il est important de noter que des ajustements plus approfondis peuvent être nécessaires pour garantir une résilience et une évolutivité optimale.

CONCLUSION

En conclusion, ce projet a été une expérience enrichissante qui a renforcé notre compréhension de la virtualisation et de la conteneurisation. En mettant en œuvre une application distribuée à l'aide de Docker, nous avons acquis des compétences pratiques et des connaissances précieuses qui seront bénéfiques pour des projets futurs dans le domaine de la virtualisation et de la gestion de conteneurs.

Bien que l'application "HumansBestFriend" soit une démonstration simple, elle offre des bases solides pour explorer des concepts plus avancés tels que l'orchestration de conteneurs avec Kubernetes. Les leçons apprises dans ce projet peuvent être étendues pour aborder des scénarios plus complexes et des déploiements à grande échelle.

Lien du GITHUB
