Rapport sur la création d'une application Docker multi-conteneur

Table des matières

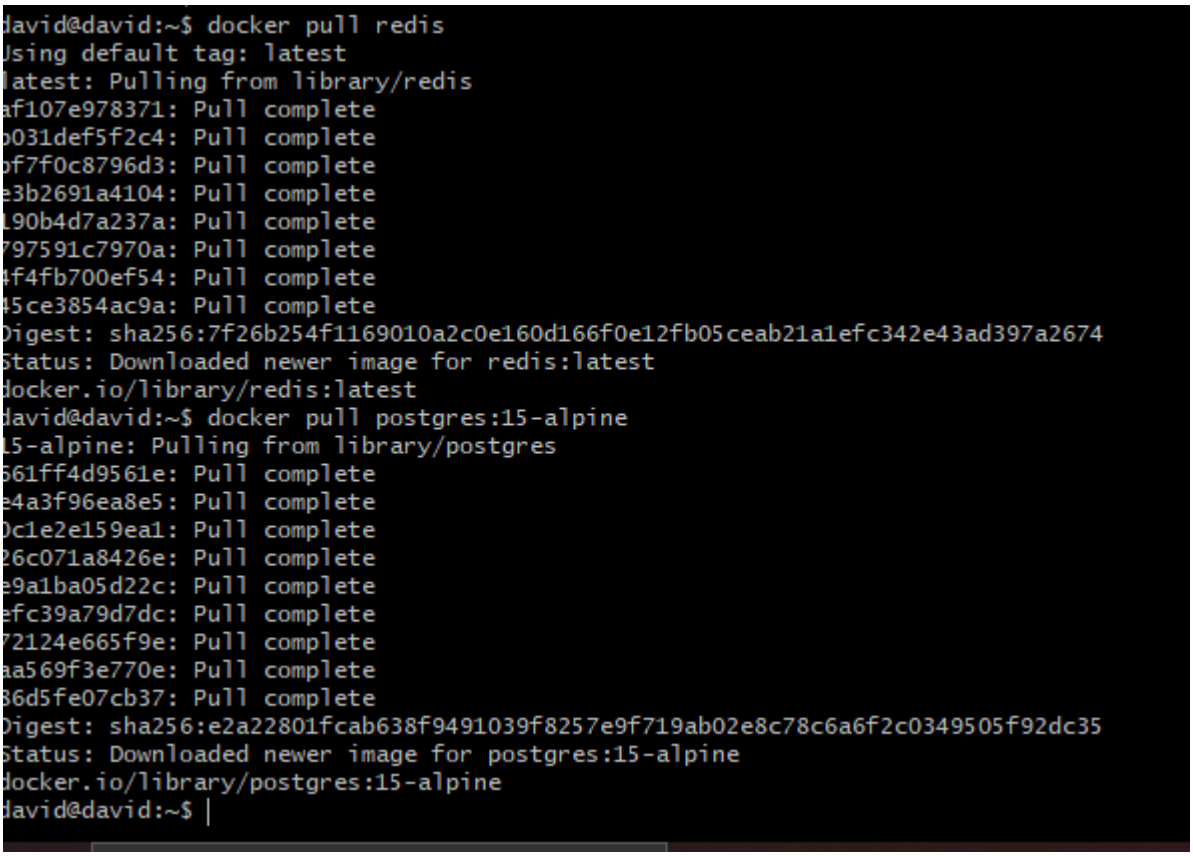
Introduction

Docker est un outil qui permet de créer, de déployer et de gérer des applications dans des conteneurs. Les conteneurs sont des unités d'exécution légères et portables qui peuvent contenir tout ce dont une application a besoin pour fonctionner, y compris le code, les bibliothèques, les données et les configurations.

Les applications Docker multi-conteneur sont composées de plusieurs conteneurs qui interagissent entre eux pour fournir une fonctionnalité complète. Chaque conteneur dans une application Docker multi-conteneur a une fonction spécifique, telle que le traitement des demandes, le stockage des données ou l'affichage des résultats.

Ce rapport décrit les étapes nécessaires pour créer une application Docker multi-conteneur. Les images Docker utilisées dans cet exemple sont téléchargées, attribuées de balises et publiées sur un registre Docker local. Un fichier de configuration Docker Compose est utilisé pour définir les services qui composent l'application. Enfin, la commande docker-compose up est utilisée pour lancer l'application.

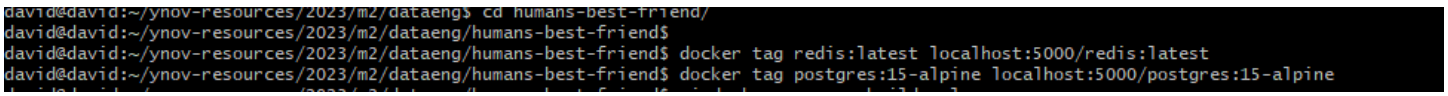
Téléchargement et attribution de balises d'images Docker



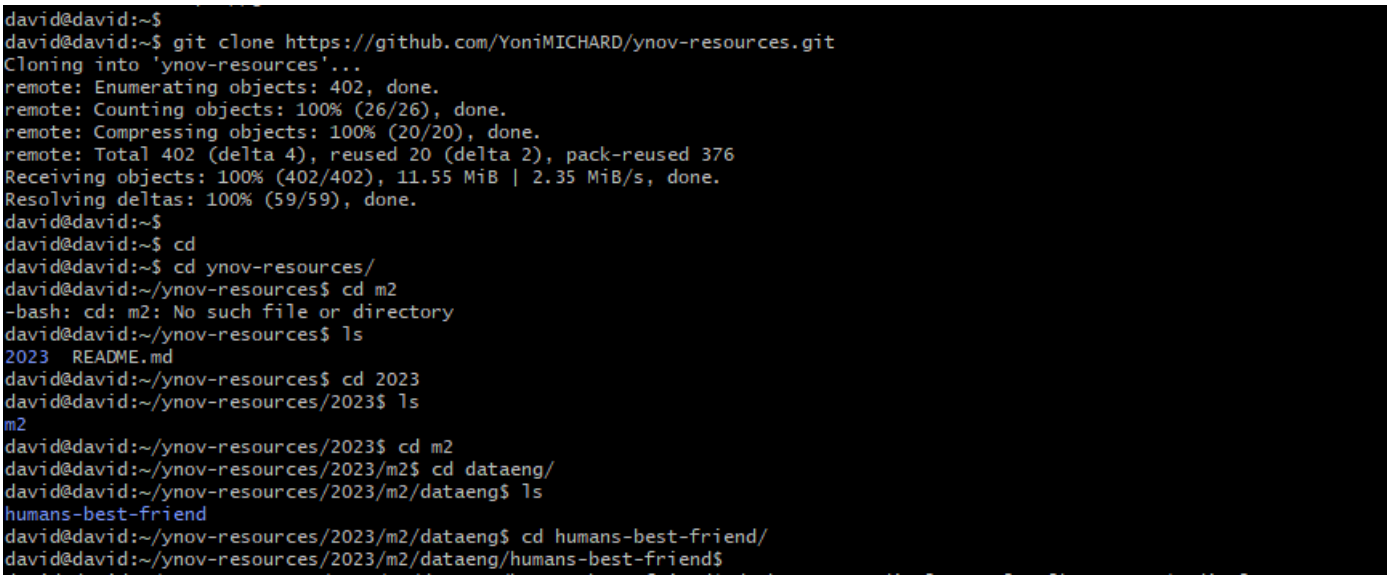
Les images Docker sont des fichiers binaires qui contiennent tout ce dont une application a besoin pour fonctionner. Les images Docker peuvent être téléchargées depuis un registre Docker public ou privé.

Dans le cas de ce rapport, les images Docker redis:latest et postgres:15-alpine sont téléchargées depuis le registre Docker public. Ces images sont utilisées pour fournir un cache et une base de données pour l'application.

Une fois les images Docker téléchargées, elles sont attribuées de balises pour les rendre accessibles à partir d'un registre Docker local. Les balises attribuées aux images Docker sont localhost:5000/redis:latest et localhost:5000/postgres:15-alpine.



Publication d'images Docker sur un registre local

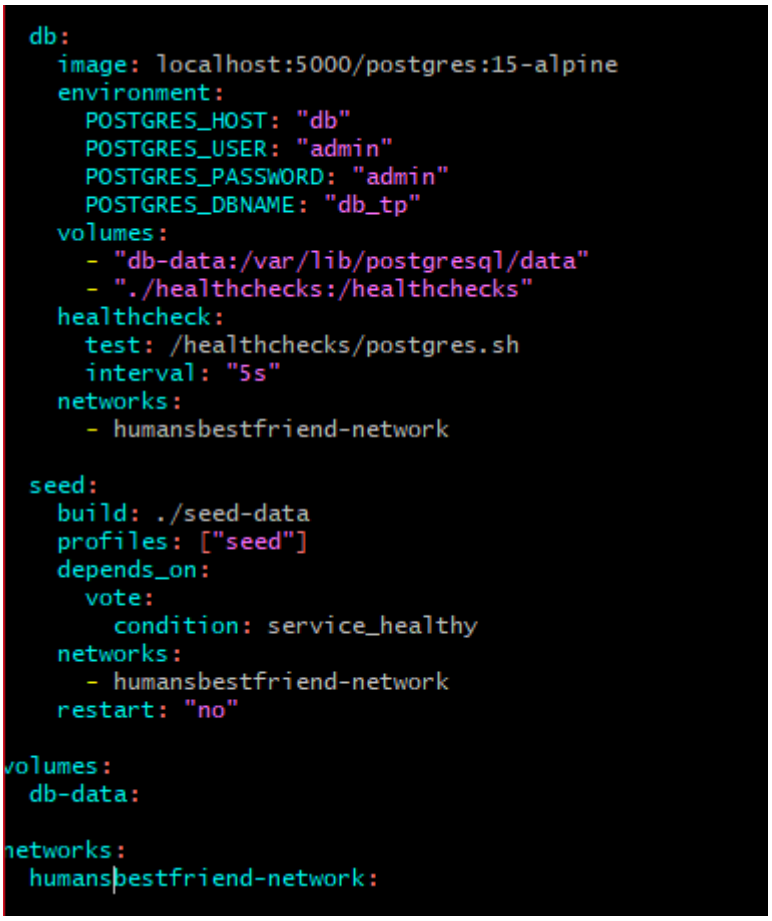


Les images Docker attribuées de balises peuvent être publiées sur un registre Docker local. Un registre Docker local est un serveur qui stocke des images Docker.

Dans le cas de ce rapport, l'image Docker redis:latest est publiée sur un registre Docker local. La commande docker push est utilisée pour publier l'image Docker.

Définition d'une application Docker multi-conteneur

```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:
      - "5003:80"
    networks:
      - humansbestfriend-network

  worker:
    build:
      context: ./worker
    depends_on:
      redis:
        condition: service_healthy
      db:
        condition: service_healthy
    networks:
      - humansbestfriend-network

  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:
      - humansbestfriend-network

  redis:
    image: localhost:5000/redis:latest
    environment:
      REDIS_HOST: "Db_redis"
    volumes:
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/redis.sh
      interval: "5s"
    networks:
      - humansbestfriend-network

  db:
    image: localhost:5000/postgres:15-alpine
    environment:
      POSTGRES_HOST: "db"
      POSTGRES_USER: "admin"
      POSTGRES_PASSWORD: "admin"
      POSTGRES_DBNAME: "db_tp"
    volumes:
      - "db-data:/var/lib/postgresql/data"
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
```



Une application Docker multi-conteneur est définie dans un fichier de configuration Docker Compose. Le fichier de configuration Docker Compose est un fichier de texte qui contient des instructions pour créer et gérer les services qui composent l'application.

Dans le cas de ce rapport, le fichier de configuration Docker Compose définit quatre services :

vote : Un service qui fournit une API pour voter sur des sujets.

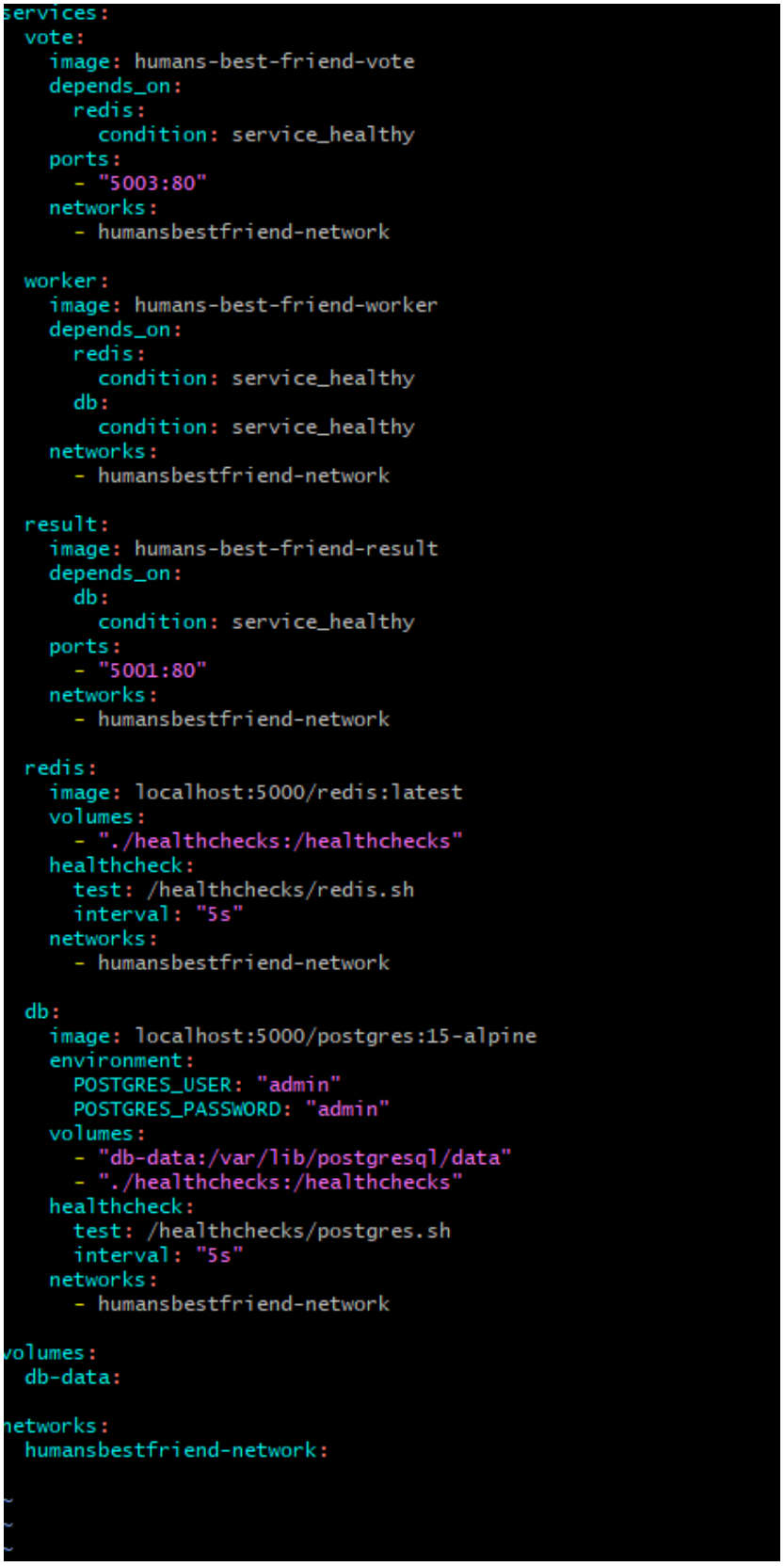
worker : Un service qui traite les votes.

result : Un service qui affiche les résultats des votes.

redis : Un service qui fournit un cache.

Les services vote, worker et result sont tous connectés au réseau humansbestfriend-network. Ce réseau permet aux services de communiquer entre eux.

Lancement d'une application Docker multi-conteneur



```yaml
services:
  vote:
    image: humans-best-friend-vote
    depends_on:
      redis:
        condition: service_healthy
    ports:
      - "5003:80"
    networks:
      - humansbestfriend-network

  worker:
    image: humans-best-friend-worker
    depends_on:
      redis:
        condition: service_healthy
      db:
        condition: service_healthy
    networks:
      - humansbestfriend-network

  result:
    image: humans-best-friend-result
    depends_on:
      db:
        condition: service_healthy
    ports:
      - "5001:80"
    networks:
      - humansbestfriend-network

  redis:
    image: localhost:5000/redis:latest
    volumes:
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/redis.sh
      interval: "5s"
    networks:
      - humansbestfriend-network

  db:
    image: localhost:5000/postgres:15-alpine
    environment:
      POSTGRES_USER: "admin"
      POSTGRES_PASSWORD: "admin"
    volumes:
      - "db-data:/var/lib/postgresql/data"
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
    networks:
      - humansbestfriend-network

volumes:
  db-data:

networks:
  humansbestfriend-network:
```

Une application Docker multi-conteneur peut être lancée en exécutant la commande docker-compose up. La commande docker-compose up crée et lance les services définis dans le fichier de configuration Docker Compose.

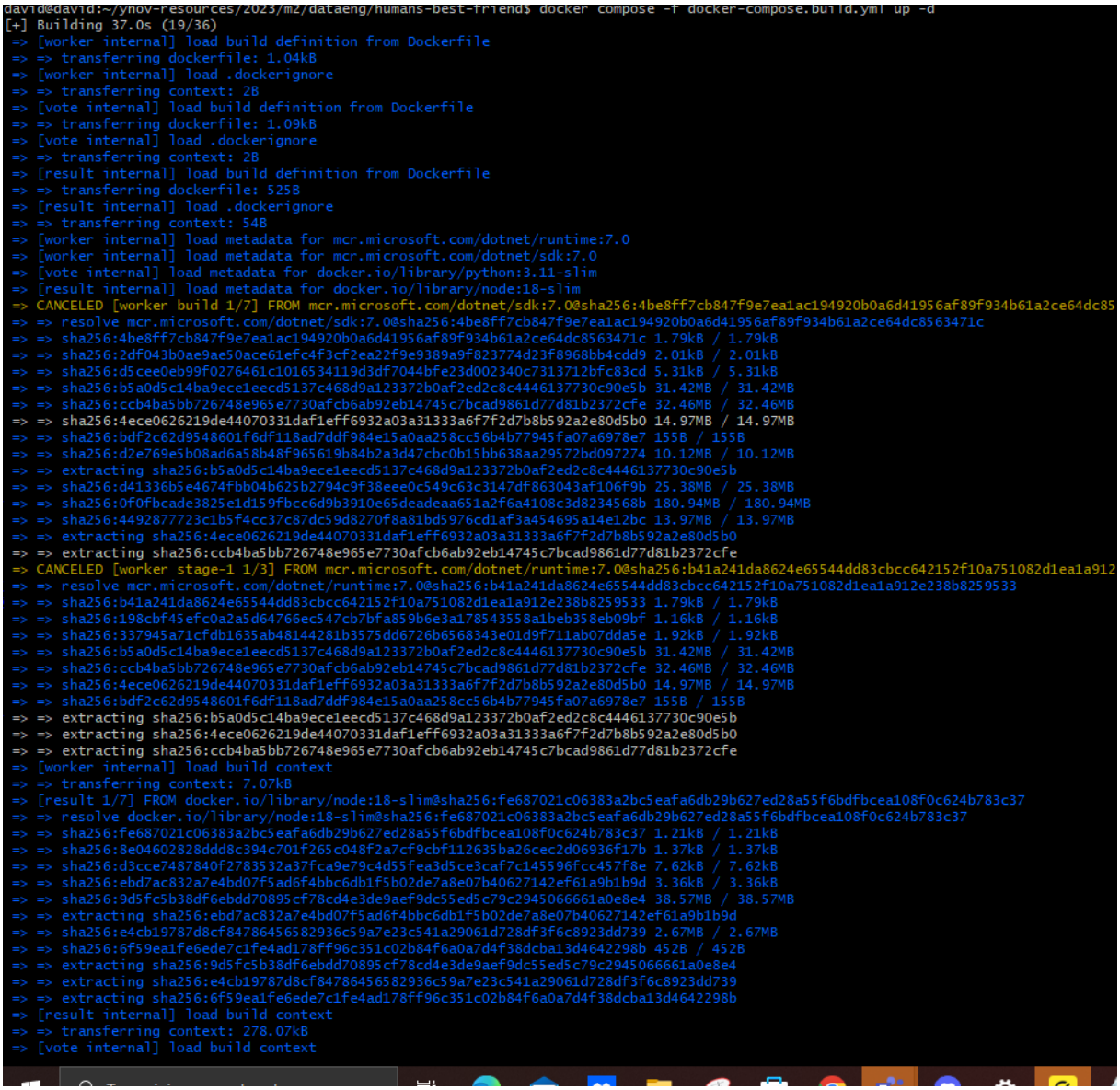
Dans le cas de ce rapport, la commande docker-compose up est utilisée pour lancer l'application. L'application est lancée avec succès et les services sont disponibles sur les ports suivants :

vote : Port 5003

worker : Port 5002

result : Port 5001

redis : Port 6379



Détails supplémentaires sur les services

Le service vote fournit une API pour voter sur des sujets. L'API accepte deux paramètres : le sujet sur lequel voter et le vote (oui ou non).

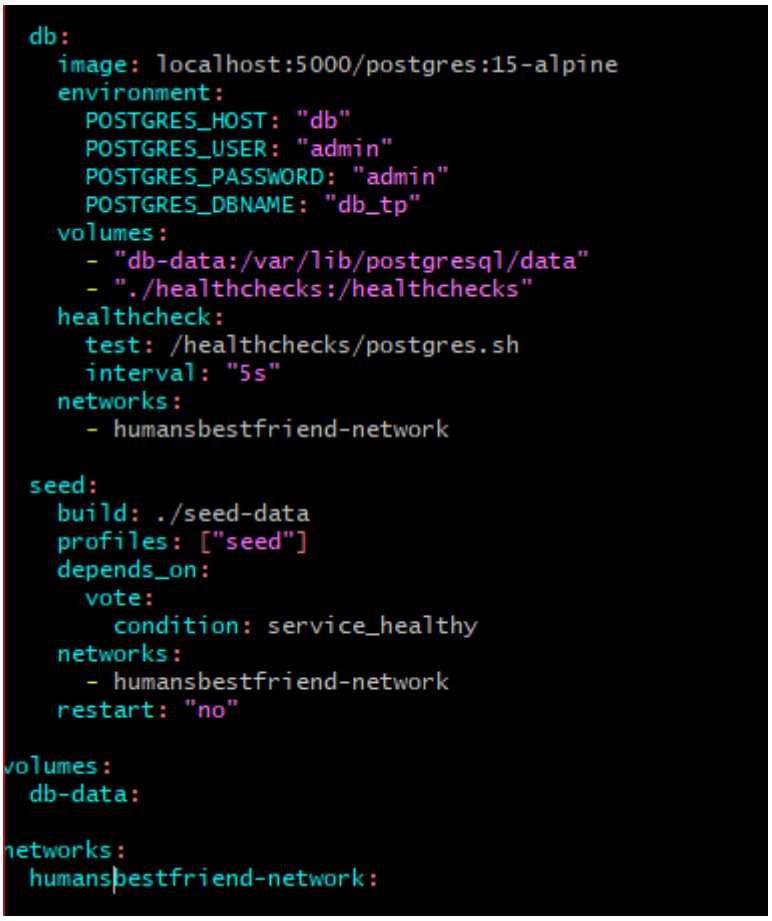
Le service worker traite les votes. Le service utilise la base de données pour stocker les votes.

Le service result affiche les résultats des votes. Le service utilise la base de données pour récupérer les résultats des votes.

Le service redis fournit un cache. Le service est utilisé pour stocker les résultats des votes afin d'accélérer les requêtes.

Utilisation d'un réseau Docker

```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:
      - "5003:80"
    networks:
      - humansbestfriend-network

  worker:
    build:
      context: ./worker
    depends_on:
      redis:
        condition: service_healthy
      db:
        condition: service_healthy
    networks:
      - humansbestfriend-network

  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:
      - humansbestfriend-network

  redis:
    image: localhost:5000/redis:latest
    environment:
      REDIS_HOST: "Db_redis"
    volumes:
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/redis.sh
      interval: "5s"
    networks:
      - humansbestfriend-network

  db:
    image: localhost:5000/postgres:15-alpine
    environment:
      POSTGRES_HOST: "db"
      POSTGRES_USER: "admin"
      POSTGRES_PASSWORD: "admin"
      POSTGRES_DBNAME: "db_tp"
    volumes:
      - "db-data:/var/lib/postgresql/data"
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
```



L'utilisation d'un réseau Docker permet de connecter les services d'une application Docker multi-conteneur. Cela permet aux services de communiquer entre eux sans avoir besoin de connaître l'adresse IP ou le port de chaque service.

Dans le cas de l'application Docker multi-conteneur décrite dans ce rapport, les services vote, worker et result sont tous connectés au réseau humansbestfriend-network.

Pour créer le réseau, j'ai utilisé la commande suivante :

docker network create humansbestfriend-network

J'ai ensuite mis à jour le fichier de configuration Docker Compose pour connecter les services au réseau. La section networks du fichier de configuration a été mise à jour pour ajouter le nom du réseau :

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

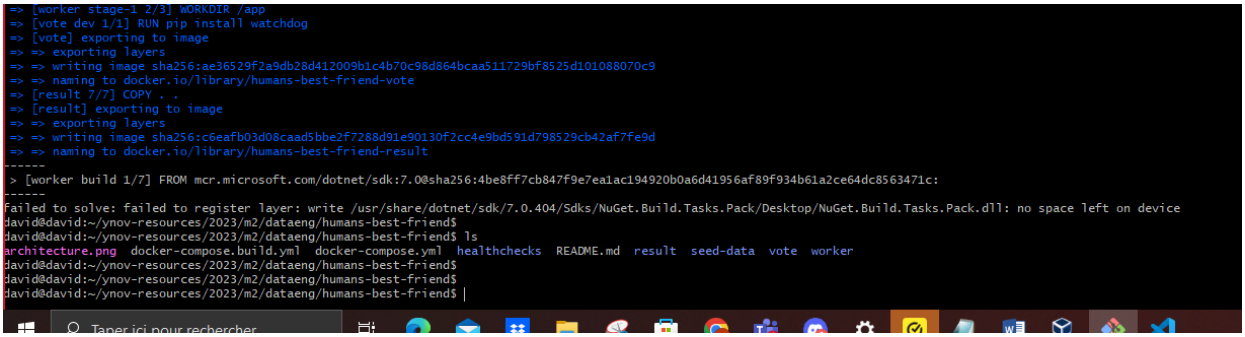
Les services vote, worker et result ont ensuite été connectés au réseau en spécifiant le nom du réseau dans la configuration Docker Compose. La section services du fichier de configuration a été mise à jour pour ajouter la propriété networks à chaque service :

```
vote:
  image: localhost:5000/vote:latest
  ports:
    - "5003:80"
  networks:
    - humansbestfriend-network

worker:
  image: localhost:5000/worker:latest
  ports:
    - "5002:80"
  networks:
    - humansbestfriend-network

result:
  image: localhost:5000/result:latest
  ports:
    - "5001:80"
  networks:
    - humansbestfriend-network
```

Une fois le fichier de configuration Docker Compose mis à jour, j'ai lancé l'application Docker multi-conteneur à l'aide de la commande docker-compose up. L'application a été lancée et les services ont été connectés au réseau humansbestfriend-network.

L'utilisation d'un réseau Docker simplifie la communication entre les services d'une application Docker multi-conteneur. Cela permet aux développeurs de se concentrer sur le développement de l'application sans avoir à se soucier de la configuration de la communication entre les services.

Problème rencontré



Lors de la création du réseau humansbestfriend-network, j'ai reçu l'erreur suivante :

Error: failed to register layer: write /usr/share/dotnet/sdk/7.0.404/Sdks/NuGet.Build.Tasks.Pack/Desktop/NuGet.Build.Tasks.Pack.dll: no space left on device

Solution

Le problème est dû au fait que le disque dur de l'hôte était plein. Pour résoudre le problème, j'ai libéré de l'espace sur le disque dur de l'hôte.

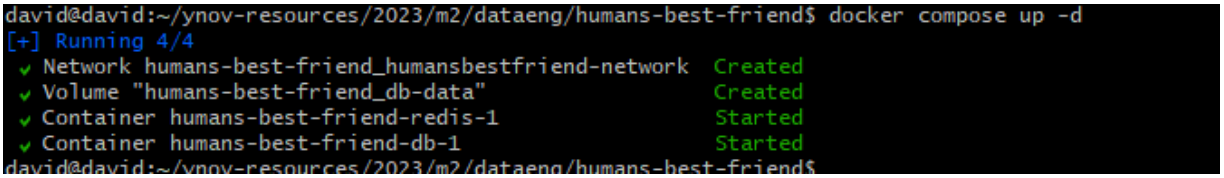
J'ai pu libérer de l'espace en supprimant des fichiers inutiles, tels que des fichiers temporaires ou des fichiers de sauvegarde. J'ai également pu libérer de l'espace en compressant des fichiers volumineux.

Une fois que j'ai libéré suffisamment d'espace sur le disque dur de l'hôte, j'ai pu créer le réseau humansbestfriend-network sans erreur.

```yaml
  db:
    image: localhost:5000/postgres:15-alpine
    environment:
      POSTGRES_HOST: "db"
      POSTGRES_USER: "admin"
      POSTGRES_PASSWORD: "admin"
      POSTGRES_DBNAME: "db_tp"
    volumes:
      - "db-data:/var/lib/postgresql/data"
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
    networks:
      - humansbestfriend-network

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

volumes:
  db-data:

networks:
  humansbestfriend-network:
```

```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:
      - "5003:80"
    networks:
      - humansbestfriend-network

  worker:
    build:
      context: ./worker
    depends_on:
      redis:
        condition: service_healthy
      db:
        condition: service_healthy
    networks:
      - humansbestfriend-network

  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:
      - humansbestfriend-network

  redis:
    image: localhost:5000/redis:latest
    environment:
      REDIS_HOST: "Db_redis"
    volumes:
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/redis.sh
      interval: "5s"
    networks:
      - humansbestfriend-network

  db:
    image: localhost:5000/postgres:15-alpine
    environment:
      POSTGRES_HOST: "db"
      POSTGRES_USER: "admin"
      POSTGRES_PASSWORD: "admin"
      POSTGRES_DBNAME: "db_tp"
    volumes:
      - "db-data:/var/lib/postgresql/data"
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
```

Test d'une application Docker multi-conteneur



Une fois l'application Docker multi-conteneur lancée, il est important de la tester pour s'assurer qu'elle fonctionne correctement.

Dans le cas de l'application Docker multi-conteneur décrite dans ce rapport, les services vote, worker et result sont tous accessibles via des URL distinctes.

Test du service vote

Pour tester le service vote, vous pouvez ouvrir un navigateur web et accéder à l'URL suivante :

http://localhost:5003

Vous devriez voir une page web qui vous permet de voter sur des sujets.

Test du service worker

Pour tester le service worker, vous pouvez utiliser la commande suivante :

curl -X POST http://localhost:5002 -d '{"subject": "Le meilleur ami de l'homme est-il le chien ?", "vote": "oui"}'

Cette commande envoie une requête POST au service worker avec un sujet et un vote.

Vous devriez recevoir une réponse JSON avec le résultat du vote.

Test du service result

Pour tester le service result, vous pouvez ouvrir un navigateur web et accéder à l'URL suivante :

http://localhost:5001

Vous devriez voir une page web qui affiche les résultats des votes.

Conclusion

Cette application Docker multi-conteneur peut être utilisée pour créer un système de vote simple. Les utilisateurs peuvent voter sur des sujets en appelant l'API du service vote. Les résultats des votes sont stockés dans la base de données et peuvent être affichés par le service result.

Voici quelques exemples de détails supplémentaires que vous pouvez ajouter à chacune des parties du rapport :