docker compose勉強会用リポジトリ

はじめに

前提条件

- WSL または macOS 環境があること
- dockerコマンドが使用できること
- gitコマンドが使用できること

やること

- Gitリポジトリのクローン(復習)
- Dockerfileからコンテナを作成(復習)
- docker-compose.ymlの利用
- DB初期化（初級編）
- DB初期化（上級編）

やらないこと

- expressの環境構築
- フロントエンドの用意

[ステップ1] 復習

リポジトリのクローン

```
git clone https://github.com/kaneko555/docker-compose-tutorial.git
```

まずは前回の復習も兼ねてdockerコマンドでコンテナを起動してみましょう

バックエンドコンテナを起動

```
docker-compose-tutorial$ docker build -t backend-app ./backend
docker-compose-tutorial$ docker run -d \
  --name backend-app \
  -p 8080:8080 \
  -v ./backend:/app \
  -w /app \
  backend-app \
  node index.js
```

バックエンド疎通確認

```
docker-compose-tutorial$ docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
8733b5b10c97   backend-app   "docker-entrypoint.s…"   23 minutes ago   Up 23 minutes   0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp            backend-app
```

```
docker-compose-tutorial$ curl http://localhost:8080
Hello from Express!
```

MySQLコンテナを起動

```
docker-compose-tutorial$ docker run -d \
  --name mysql \
  --restart always \
  -e MYSQL_ROOT_PASSWORD=pass \
  -e MYSQL_DATABASE=testdb \
  -p 3306:3306 \
  -v $(pwd)/docker/mysql/data:/var/lib/mysql \
  -v $(pwd)/docker/mysql/init:/docker-entrypoint-initdb.d \
  mysql:8
```

```
docker-compose-tutorial$ docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS                                                   NAMES
d3acf04de1a0   mysql:8       "docker-entrypoint.s…"   23 minutes ago   Up 23 minutes   0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp, 33060/tcp   mysql
8733b5b10c97   backend-app   "docker-entrypoint.s…"   23 minutes ago   Up 23 minutes   0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp              backend-app
```

mysqlテーブル確認

```
docker-compose-tutorial$ docker exec -it mysql bin/bash
bash-5.1$ mysql -u root -p # pass
mysql>$ use sample_db;
mysql>$ select * from users;
mysql>$ exit
bash-5.1$ exit
```

カスタムネットワークを作成して接続

ただし、今の状態では `backend-my-app` と `mysql` は通信できません。

```
docker-compose-tutorial$ curl http://localhost:8080/users
{"message":"Internal server error"}
```

なぜなら先ほど起動したコンテナはそれぞれ独立しており、紐づいていないからです。

その為、カスタムネットワークを作成して先ほどのコンテナを同一ネットワークに接続させる必要があります。

```
docker-compose-tutorial$ docker network create my-network
docker-compose-tutorial$ docker network connect my-network mysql
docker-compose-tutorial$ docker network inspect my-network
```

さらにバックエンドにMySQL接続に必要な環境変数を追加してかつ、DBと同じネットワークで起動させなければなりません。

```
docker-compose-tutorial$ docker stop backend-app
docker-compose-tutorial$ docker rm backend-app
docker-compose-tutorial$ docker run -d \
  --name backend-app \
  --network my-network \
  -p 8080:8080 \
  -v ./backend:/app \
  -w /app \
  -e DB_HOST=mysql \
  -e DB_USER=root \
  -e DB_PASSWORD=pass \
  -e DB_NAME=sample_db \
  backend-app \
  node index.js
```

```
docker-compose-tutorial$ docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS                      PORTS                                                NAMES
ad585e4d12e9   backend-app   "docker-entrypoint.s…"   4 seconds ago    Exited (1) 3 seconds ago                                                         backend-app
d3acf04de1a0   mysql:8       "docker-entrypoint.s…"   43 minutes ago   Up 43 minutes               0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp, 33060/tcp   mysql
```

動作確認

```
docker-compose-tutorial$ curl http://localhost:8080/users
[{"id":1,"name":"taro"},{"id":2,"name":"hanako"}]
```

コンテナの停止/削除

```
docker-compose-tutorial$ docker stop backend-app mysql
docker-compose-tutorial$ docker rm backend-app mysql
```

10分 休憩

[ステップ2] docker composeを使用してみる

dockerコマンドで複数のコンテナを扱おうとすると**非常**に手間である

- コンテナのビルド
- コンテナの起動（長いコマンドなど）
- ネットワークの管理

それを毎回、毎日行うのは面倒すぎる...
その為に `docker compose` という物が用意されています。

実際に使用してみましょう。

```
docker-compose-tutorial$ cd docker
docker-compose-tutorial/docker$ docker compose up -d
docker-compose-tutorial/docker$ docker ps -a #docker compose ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED         STATUS         PORTS                                                  NAMES
d9df5a532f21   docker-backend-app   "docker-entrypoint.s…"   2 seconds ago   Up 2 seconds   0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp            backend-app
f40bd2e79313   mysql:8              "docker-entrypoint.s…"   2 seconds ago   Up 2 seconds   0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp, 33060/tcp   mysql
```

これだけで簡単に複数のコンテナを管理・構成・起動できる！

動作確認

```
docker-compose-tutorial$ curl http://localhost:8080/users
[{"id":1,"name":"taro"},{"id":2,"name":"hanako"}] # エラーが出たら少し時間を置く(1分程度)
```

ただしデメリット/注意点も存在します

docker composeを使用する場合のデメリット

- あくまでも開発環境用である
  - 開発・検証用途向けに設計されているので、本番等では Docker Swarm や Kubernetes の使用を検討した方が良い
- 新入社員や初心者にはやや難しい
  - 設定ファイルの作成にはdockerの知識が必要
  - 複数コンテナが同時に起動・終了するため、トラブル時にどのコンテナが原因か特定しにくい
- 設定ファイルが複雑化しやすい
  - 規模が大きくなると、`docker-compose.yml` が肥大化・複雑化する。

[ステップ3] DB初期化（初級編）

DBにテーブルを用意したり初期値を入れたいケースもあるかと思います。

dockerのmysql公式ドキュメントを見ると、その為の機能が実装されています。

> `/docker-entrypoint-initdb.d` にある拡張子が `.sh`、`.sql`、`.sql.gz` のファイルも実行されます。SQLダンプをそのディレクトリにマウントすることで、サービスを簡単に設定できます。

[ステップ4] DB初期化（上級編）

ステップ4で説明した方法は、以下のケースだと使用できません。

- 公式イメージで自動的に初期化される仕組みが存在しない場合
  - DynamoDB 等
- 初期化タイミングの制御が必要な場合
  - `init.sql` は MySQLコンテナの初回起動時 にしか実行されない
- 制御ロジックが必要な場合
  - `init.sql` 等では単純なSQL実行しかできない
  - 条件分岐、マイグレーションのバージョン管理、ロールバックなどの制御は記載できない

- 他サービスの準備が必要な場合
  - `init.sql` はMySQLの「内部」で実行されるため、他サービスとの連携ができない
- 初期化処理がSQL以外を含む場合
  - CSVやjsonファイルを読み込んでINSERTしたいケースなど

この様な場合は、別コンテナからスクリプトを実行してデータの加工やINSERTを行います。

中規模以上のプロジェクトではこちらが採用されることが多いかと思います。

実際にスクリプト実行用のコンテナを作成してみましょう

usersテーブルのレコードが10件未満なら適当なレコードを5件挿入したいです。

- ヒント１：mysqlコマンドが実行できるコンテナを追加する
- ヒント２：`./docker/mysql/scripts/insert.sh` を利用する
- ヒント３：上記スクリプトにmysqlの接続情報を渡す
- ヒント４：mysqlの起動後に実行される様にする

おまけ1

コンテナ内でコードを修正しましたが、正しく動作するか確認したいです。

今起動しているバックエンドは `docker compose up` でサービスまで起動しますが、ホットリロードに対応していません。

※ ホットリロード：アプリケーションの状態を保持したまま、変更されたモジュールだけを差し替える機能（React等）

この様な場合どうすれば良いでしょうか？

1. そのまま動作確認する
2. `docker compose restart` 後、動作確認する
3. バックエンドのコンテナに入って `npm run dev` を実行後、動作確認する
4. バックエンドのコンテナに入ってサービスをKILL後、`npm run dev` を実行後、動作確認する
5. 自信があるので動作確認しない

他にはどの様な手段が考えられるでしょうか..？

おまけ2

現在の `docker-compose.yml` には幾つかの問題点があります。

今回は以下の点を修正してみましょう。

level 1

現在の実装では、mysqlの接続情報をハードコーディングで直接記載してしまっています。

これでは、設定の再利用性・保守性が損なわれてしまっています。

- mysqlの接続情報を外部ファイルから読み込む様にしてみましょう

ヒント：`.env` を使用してみよう

level 2

現在の実装では、`docker compose up` 後すぐに `http://localhost:8080/users` にリクエストを送るとエラーとなる可能性があります。

- エラーが発生するリスクを無くしてみましょう

ヒント：`depends_on` はコンテナの起動順序の制御のみです

level 3

おまけ1で説明した通り、コード修正を反映させるのに毎回コンテナを起動し直すのは手間です。

サービスの起動をコンテナ起動とは分離する事で、開発作業を柔軟に対応する事ができます。

- `docker compose up` ではコンテナの起動のみで、コンテナに入ってからサービスを自由に起動/終了できるようにしてみましょう

おまけ3

便利なコマンド

- docker stop $(docker ps -q)
- docker rm $(docker ps -q -a)