



Rails7.0.8 MySQL8 Docker環境構築

type	Permanent
tag	 Docker ,  Ruby on Rails , MySQL
created	@2023年10月10日 9:59
updated	@2023年10月10日 15:03

▼ 目次

[目次](#)

[環境](#)

[各種設定ファイルの準備](#)

[ディレクトリの作成](#)

[ファイルの作成](#)

[Dockerfile](#)

[docker-compose.yml](#)

[docker-composeの解説](#)

[Gemfile](#)

[Gemfile.lock](#)

[entrypoint.sh](#)

[Rails プロジェクト立ち上げ](#)

[rails newを実行する](#)

[docker-compose build を実行する](#)

[MySQLの設定](#)

[DBの作成](#)

[コンテナの作成と起動](#)

[ローカルホストに接続](#)

環境

Ruby v3.2.2

Rails v7.0.8

MySQL v8

各種設定ファイルの準備

ディレクトリの作成

```
#ディレクトリ名はアプリケーション毎に適宜変更する
mkdir <ディレクトリ名>
cd <ディレクトリ名>
```

ファイルの作成

```
touch {docker-compose.yml,Dockerfile,Gemfile,Gemfile.lock,entrypoint.sh}
```

各種設定ファイルを作成するコマンド

Dockerfile

```
# FROM: 使用するイメージ、バージョン
FROM ruby:3.2.2
# 公式→https://hub.docker.com/_/ruby

# Rails 7ではWebpackerが標準では組み込まれなくなったので、yarnやnodejsのインストールが不要

# RUN: 任意のコマンド実行
RUN mkdir /app

# WORKDIR: 作業ディレクトリを指定
WORKDIR /app

# COPY: コピー元とコピー先を指定
# ローカルのGemfileをコンテナ内の/app/Gemfileに
COPY Gemfile /app/Gemfile
COPY Gemfile.lock /app/Gemfile.lock

#Gemfile.lockにプラットフォームを追記
RUN bundle lock --add-platform ruby
RUN bundle lock --add-platform x86_64-linux

# RubyGemsをアップデート
RUN bundle install
RUN bundle update

COPY . /app

# コンテナ起動時に実行させるスクリプトを追加
COPY entrypoint.sh /usr/bin/
RUN chmod +x /usr/bin/entrypoint.sh
ENTRYPOINT ["entrypoint.sh"]
EXPOSE 3000

# CMD: コンテナ実行時、デフォルトで実行したいコマンド
```

```
# Rails サーバ起動
CMD ["rails", "server", "-b", "0.0.0.0"]
```

docker-compose.yml

```
version: '3'
services:
  web:
    # build: .でdocker-compose.ymlと同じフォルダにDockerfileがあることを示す
    build: .
    # 毎回 rm tmp/pids/server.pid するのも手間であるため、・事前に手元で/tmp/pids/server.pidを削除する
    command: bash -c "rm -f tmp/pids/server.pid && bundle exec rails s -p 3000 -b '0.0.0.0'"
    volumes:
      - ./app
    ports:
      - 3000:3000
    depends_on:
      - db
    # railsでpryする用
    # true を指定することでコンテナを起動させ続けることができます。
    tty: true
    # stdin_openとは標準入出力とエラー出力をコンテナに結びつける設定です。
    stdin_open: true
  db:
    image: mysql:8
    # DBのレコードが日本語だと文字化けするので、utf8をセットする
    command: mysqld --character-set-server=utf8 --collation-server=utf8_unicode_ci
    volumes:
      - db-volume:/var/lib/mysql
    # 環境変数
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: root
      TZ: "Asia/Tokyo"
    ports:
      - "3306:3306"
# PC上にdb-volumeという名前でボリューム（データ領域）が作成される
# コンテナを作り直したとしてもPC上に残るようにするために設定
volumes:
  db-volume:
```

▼ docker-composeの解説

services:

`services` の中でコンテナを定義します。

今回は `MySQL` と `Rails` をコンテナ化するので、名前をわかりやすく `db`、`web` とします。

実際に作成されるコンテナ名は「アプリケーション名 + サービス名 + 連番」で

`app_db_1`、`app_web_1` となります。

image:

利用するイメージを指定し、コンテナを構築します。

enviroment:

MySQLに関する環境変数を設定します。（今回はパスワードのみ）

パスワードはご自身で決めて構いません。

ports:

ポート番号に関するを設定します。

MySQL は `3306`、 Rails は `3000` というポート番号がデフォルト値として設定されています。

Railsに関してはおなじみの `localhost:3000` の 3000 ですね。

Dockerを使わず、ローカルで環境構築した場合は直接この `localhost:3000` にアクセスすれば良かったわけですが、Dockerを使った場合、コンテナに外部からアクセスすることができないので、ちょっとした工夫が必要です。

そこで登場するのがこの `ports:` です。

`ports:` は `- ローカル側のポート番号 : コンテナ側のポート番号` で表します。

今回、 `ports: - 3000:3000` としていますが、説明をわかりやすくするため、仮にもし `ports: - 9000:3000` に設定するとします。

これはコンテナ側の `3000`ポート に `9000`ポート でのアクセスを許可するという意味になります。

つまりこの場合、 `localhost:9000` でアクセス可能になるということになります。

volumes:

ボリュームに関する設定をします。

`ボリューム` とはコンテナにおいて生成されるデータを永続的に保存するためのものです。

MySQLを例にとると、MySQLをコンテナ化し、そこに テーブル や カラム などのデータを保存したとします。

しかし、そのMySQLのコンテナを削除してしまうと、保存してあったテーブル、カラムなどのデータも一緒に削除されてしまいます。

それはまずいので、データだけローカルにも保存しておく、この仕組みを `ボリューム` といいます。

このボリュームを利用すると、仮にコンテナを削除したとしても、新しくコンテナを立ち上げたときに、そのデータを再利用することができます。

ボリュームを保存する方法は大きく分けて2つあります。

1つ目は、**保存したいディレクトリをマウントする方法**、2つ目は、**名前付きボリュームを利用する方法**です。

今回 `MySQL` は2つ目の **名前付きボリュームを利用する方法** を使います。

この方法は、ローカルに名前付きのボリュームを作成し、そこにコンテナ側の指定したディレクトリを保存するというものです。

名前付きボリュームの場合、`volumes:` は `- ボリューム名 : コンテナ側の保存させたいディレクトリ` で表します。

今回、`ボリューム名` を `mysql_data` とし、`コンテナ側の保存させたいディレクトリ` はMySQLのデータ保存場所である `/var/lib/mysql` を指定します。

結果、`volumes: - db-volume:/var/lib/mysql` という書き方になります。

そして、`version:` や `services:` と同じ段落(トップレベル)の、一番下に行に書いている `volumes:` にボリューム名を記載し、名前付きボリュームであることを明示します。

次に `Rails` は、1つ目の **保存したいディレクトリをマウントする方法** を使います。

この方法は、「ローカル側の指定したディレクトリ」と「コンテナ側の指定したディレクトリ」を同期させて、両者を常に同じ状態に保つというものです。

マウントする場合、`volumes:` は `- ローカル側の同期させたいディレクトリ : コンテナ側の同期させたいディレクトリ` で表します。

今回、`volumes: - ../app/` としています。

つまり、`.` (`docker-compose.yml`のあるディレクトリ、つまりローカル側の `myapp`配下) と `/myapp/` (コンテナ側の `myapp`配下) を同期するということです。

これにより、コンテナ起動後、ローカルの `myapp`配下のファイルなどを編集すると、コンテナ側にも編集が反映されます。

build:

`Dockerfile`のあるディレクトリをして指定します。(`docker-compose.yml`から見て同じディレクトリに `Dockerfile`が存在するので `.`)

このステップで ③で説明した `Dockerfile` を利用しオリジナルのイメージを作成し、コンテナを構築します。

command:

コンテナ起動時の実行されるコマンドを設定します。

今回設定するコマンドは、`bash -c "rm -f tmp/pids/server.pid && bundle exec rails s -p 3000 -b '0.0.0.0'"` です。

`bash -c ""` はコンテナの中に入り `""` 内のコマンドを実行するという命令です。

`rm -f tmp/pids/server.pid` で rails server が起動していた場合、停止します。(何らかの原因で rails server がすでに起動していた場合、新しい rails server が起動できないため。)

`bundle exec rails s -p 3000 -b '0.0.0.0'` で rails server を起動します。

`&&` は複数のコマンドを実行するときに用います。

depends_on:

コンテナの作成順序の設定です。

`depends_on: - db` は、`MySQLのコンテナ` が起動してから `Railsのコンテナ` を起動するという意味です。

Gemfile

```
source 'https://rubygems.org'
gem 'rails', '~>7.0.8'
```

Railsのバージョンを指定する

Gemfile.lock

空欄のままでOK

entrypoint.sh

```
#!/bin/bash
set -e

rm -f /app/tmp/pids/server.pid

exec "$@"
```

特定のファイルが既に存在する場合にサーバーの再起動を妨げる Rails 固有の問題を修正するエントリポイントスクリプトを提供します。このスクリプトは、コンテナが開始されるたびに実行されます。

Rails プロジェクト立ち上げ

rails newを実行する

```
docker-compose run --no-deps web rails new . --force --database=mysql --skip-bundle
```

アプリケーションの作業ディレクトリに移動し、`docker-compose run` コマンドで `rails new` を実行します。

- `-force` : 同じファイルがある場合、上書きする
- `-database=mysql` : データベースにMySQLを指定する
- `-skip-bundle` : `bundle install` をスキップする。

このコマンドを実行すると、`Dockerfile` を元に、イメージとコンテナを作成し、そのコンテナの中で `rails new` を実行します。

docker-compose build を実行する

```
docker-compose build
```

`rails new` で `Gemfile` が更新されたので、`bundle install` するため、`docker-compose build` を実行します。

先ほどの `docker-compose run` で作成されたイメージは `rails new` される前の `Gemfile` を `bundle install` したイメージなので、更新された `Gemfile` を `bundle install` したイメージを再度作成する必要があります。

MySQLの設定

`/config/database.yml` を編集する

```
default: &default
  adapter: mysql2
  encoding: utf8mb4
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
  username: root
  password: password
  host: db
```

`password:` と `host:` を編集します。

`password:` には `docker-compose.yml` の `MYSQL_ROOT_PASSWORD:` に書いたパスワードを記載します。

`host:` には `docker-compose.yml` で命名したMySQLのコンテナ名 `db` を記載します。

DBの作成

```
docker-compose run --rm web rails db:create
```

コンテナの作成と起動

```
docker-compose up -d
```

このコマンドでコンテナを起動します。
-dをつけることでバックグラウンドで起動します。

```
docker-compose ps
```

それぞれのコンテナが **UP** になっていれば起動成功

ローカルホストに接続

<http://localhost:3000/> にアクセスしてページが表示されたら成功。