



31



33



この記事は最終更新日から3年以上が経過しています。



@snyt45 (株式会社diddyworks)

Organization

投稿日 2019年10月11日 更新日 2019年10月15日

【Docker環境構築】PHP5.3.3 + Apache + MySQL + phpMyAdminにFuelPHPをインストール

MySQL, Apache, FuelPHP, PHP5.3, Docker

概要

既存の社内システム改修をする必要があり、PHP5.3.3、FuelPHP 4.7.2 1.7.3、Apache、MySQLという構成で、Dockerで環境構築を行いました。

調べたところPHP5.3.3は2010年頃にリリースされたらしくかなりレガシーなバージョンになるようで、記事があまり見つからない中のDocker環境構築は大変でした(最終的には色々な記事を参考になんとか構築できました・・・)。

自分が1から再構築できるようにDockerfile、docker-compose.ymlの解説や手順等を改めてまとめておこうと思います。

先に下記の先人のおかげでこの構成ができたことを感謝申し上げます！

自分の記事は先人の記事をほぼ組み合わせた感じになります。

PHP5.3.3環境を2017年に用意する方法 - Qiita

docker-compose で PHP7.2 + Apache + MySQL + phpMyAdmin 環境を構築 - Qiita

Docker 仮想CentOS6を動かす - @//メモ

ディレクトリ構成と手順

すぐ動かしたい方は、本記事と同じ構成のものをGithubに上げているのでそちらをご覧ください。本記事では、この構成を作成するまでの流れをやっていきます

追記情報は最後の方で確認してください

```
└─ docker
  └─ httpd
    └─ httpd.conf
  └─ php
    └─ Dockerfile
```

```
|   └─php.ini
└─docker-compose.yml
```

docker-compose.yml

docker-compose.yml

```
version: '3'

services:
  php:
    build:
      context: ./docker/php/
      dockerfile: Dockerfile
    volumes:
      - ./app
      - ./docker/php/php.ini:/etc/php.ini
      - ./docker/httpd/httpd.conf:/etc/httpd/conf/httpd.conf
    ports:
      - 8080:80
  mysql:
    image: mysql:5.7
    volumes:
      - ./docker/mysql/volumes:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=test
      - MYSQL_USER=test
```

```
- MYSQL_PASSWORD=test

phpmyadmin:
  image: phpmyadmin/phpmyadmin
  environment:
    - PMA_ARBITRARY=1
    - PMA_HOST=mysql
    - PMA_USER=root
    - PMA_PASSWORD=root
  links:
    - mysql
  ports:
    - 4040:80
  volumes:
    - ./docker/phpmyadmin/sessions:/sessions
```

docker/httpd/httpd.conf

httpd.conf

～～長いので省略～～

初期設定からの変更点①

#ServerName www.example.com:80

↓

ServerName www.example.com:80

初期設定からの変更点②

DocumentRoot "/var/www/html"

↓

DocumentRoot "/app"

docker/php/Dockerfile

Dockerfile

```
FROM centos:6.9
```

```
ENV PHP_VERSION 5.3.3
```

```
RUN yum install -y \  
    php \  
    php-mbstring \  
    php-pgsql \  
    php-mysql \  
    php-gd \  
    zip \  
    unzip \  
    git \  
    httpd && \  
    yum clean all
```

```
RUN mkdir /app
```

```
WORKDIR /app
```

```
EXPOSE 80
```

```
CMD ["/usr/sbin/apachectl","-DFOREGROUND"]
```

docker/php/php.ini

php.ini

```
[Date]
date.timezone = "Asia/Tokyo"

[mbstring]
mbstring.internal_encoding = "UTF-8"
mbstring.language = "Japanese"
```

PHP5.3.3が動く環境を用意する

PHP5.3.3をまずインストールしなければいけません。

自分はすんなり@suinさんの記事をすぐに見つけれられたので幸運です。

PHP5.3.3環境を2017年に用意する方法 - Qiita

うまくいった方法はCentOS 6.9でyum install phpすること。なんとRedHatは未だにPHP5.3をサポートしている。

一応、まだ過去バージョンで5.3.3もインストールできるみたいでした(僕は試していませんが...)。

[PHP: Releases](#)

[windows.php.net - /downloads/releases/archives/](http://windows.php.net/downloads/releases/archives/)

参考：旧バージョンのPHPをダウンロードする方法 | jMatsuzaki

CentOS 6.9のDockerイメージダウンロード

CentOS 6.9でphp5.3がインストールできるみたいなので、CentOS 6.9のイメージからコンテナを起動してコンテナ内でphpのインストールを試して、上手く行ったらDockerfileに記載していくという方法でやってみます。

CentOS 6.9をインストールするDockerfileを用意

Dockerfile

```
FROM centos:6.9
```

CentOS 6.9のDockerイメージをビルド

用意したら、Dockerfileがあるディレクトリに移動後、イメージをビルドします。

```
$ docker build .
```

ビルドされるとエラーが表示されました。

```
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitive files and directories.
```

セキュリティ警告：WindowsからWindows以外のDockerホストに対してDockerイメージを構築しています。すべてビルドコンテキストに追加されたファイルとディレクトリには、「-rwxr-xr-x」権限が付与されます。お勧めです機密性の高いファイルとディレクトリのアクセス許可を再確認してリセットします。

windowsからwindows以外のイメージを作成したため、パーミッションの警告のようです。とりあえず今回は無視します。

イメージの確認

確認できました。

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
centos	6.9	2199b8eb8390	7 months ago	195MB

コンテナ起動

イメージ名を指定して、コンテナを起動しログインします。

```
$ docker run -it centos:6.9 /bin/bash
```

コンテナでphpインストールまで

ログインしたら、環境変数を設定します。

```
$ export PHP_VERSION=5.3.3  
$ echo $PHP_VERSION  
=> 5.3.3
```

phpをインストールします。

```
$ yum install -y php
```

phpのバージョン確認。php5.3.3がインストールされました！

```
$ php -v  
PHP 5.3.3 (cli) (built: Mar 22 2017 12:27:09)
```

Copyright (c) 1997-2010 The PHP Group

Zend Engine v2.3.0, Copyright (c) 1998-2010 Zend Technologies

あとは必要なモジュールを同じように確認しながらインストールしていきます(モジュールは環境に応じて適宜インストール。
zip、unzip、git、httpdはFuelPHPを使う際に必須でした。ないとエラーが出ます。)。

```
$ yum install -y php-mbstring
$ yum install -y php-pgsql
$ yum install -y php-mysql
$ yum install -y php-gd
$ yum install -y zip
$ yum install -y unzip
$ yum install -y git
$ yum install -y httpd
```

Apacheが起動するか確認

コンテナにログインした状態で作業を続けます。

```
$ /usr/sbin/apachectl -DFOREGROUND
```

ServerNameのエラーが出ると思いますがここでは無視します。

Enterを押して、うごけばとりあえずOKです。

確認したら、Ctrl + Cで起動を停止します。

コンテナでの作業は終わりですので、Ctrl + Dで抜けておきましょう。

動作確認したコマンドをDockerfileに反映

先程、コンテナで実際に入力したコマンドをDockerfileを反映すると下記のようになります。

完成したDockerfile

```
FROM centos:6.9

ENV PHP_VERSION 5.3.3

RUN yum install -y \
    php \
    php-mbstring \
    php-pgsql \
    php-mysql \
    php-gd \
    zip \
    unzip \
    git \
    httpd && \
    yum clean all

RUN mkdir /app

WORKDIR /app
```

```
EXPOSE 80
```

```
CMD ["/usr/sbin/apachectl","-DFOREGROUND"]
```

先程のコンテナ時点でなかったものがあると思います。
そちらの説明です。

`yum clean all` はDockerイメージの容量を減らすためのよくあるTIPSらしいです。

`RUN mkdir /app` `WORKDIR /app` はappフォルダを作成して、作業フォルダをappに指定しています。のちほど、appフォルダにfuelphpのファイルを展開するときに必要になります。

`EXPOSE 80` は、コンテナのポート80番を使うよみたいな感じです。実際にポートが公開されるわけではなさそうです。わかりやすいように明示しておきます。

再度、ビルドを行いApacheが起動することを確認する

ここは一気にやってしまいましょう。

```
# ビルド時にイメージ名をつける  
$ docker build -t cnetos_php:6.9 .
```

```
# -p: ホストのポート8080 を コンテナのポート 80 にバインド（割り当て）します。  
# -d: バックグラウンドで起動
```

```
$ docker run -p 8080:80 -d cnetos_php:6.9
```

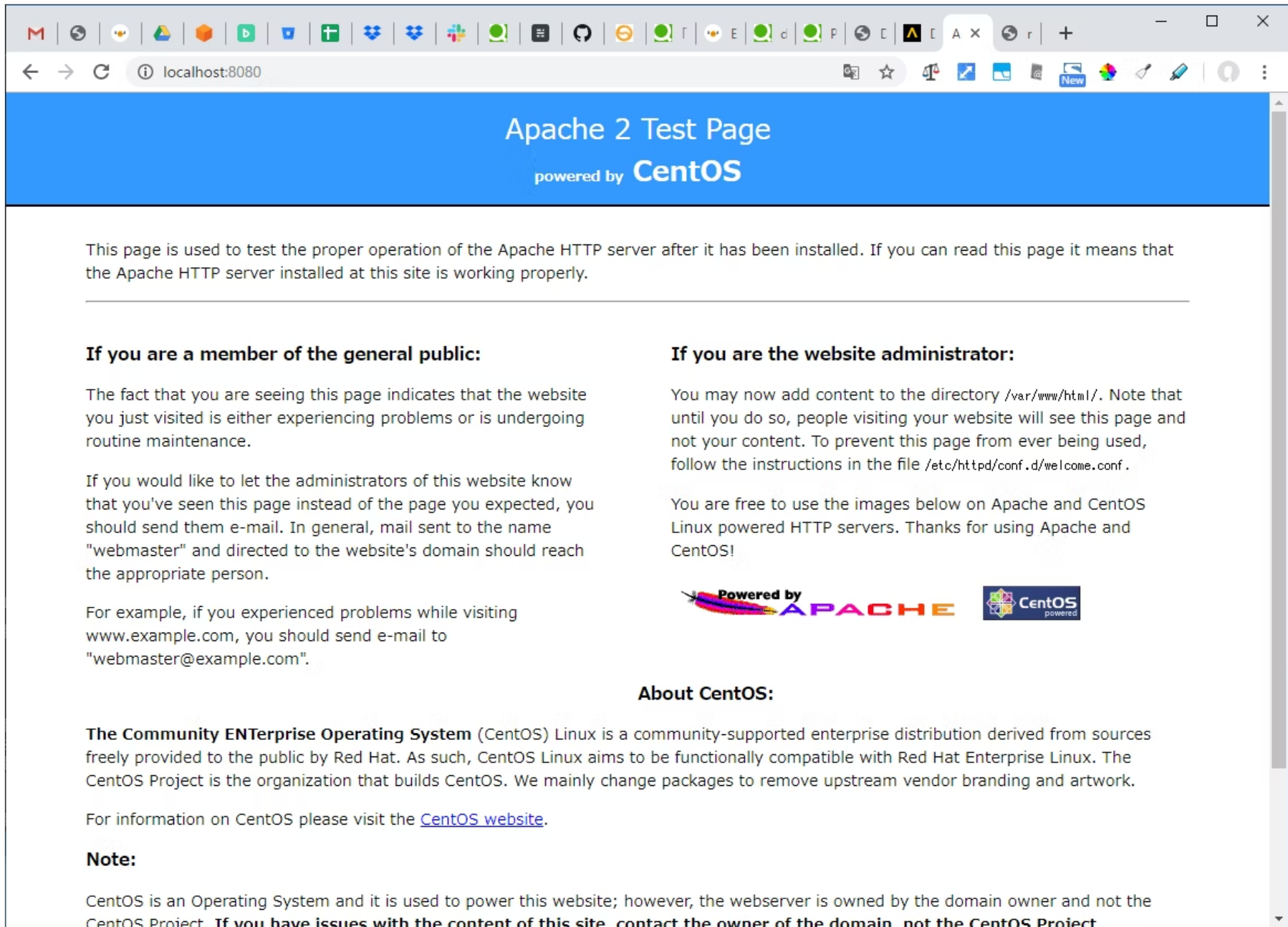
```
# コンテナが起動していることを確認
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a9337cd1cef7	cnetos_php:6.9	"/usr/sbin/apachectl..."	7 seconds ago	Up 5 seconds	0.0.0.0:8080->80/tcp	inspiring_

<http://localhost:8080/> にアクセス。

以下の画面が表示されていれば成功です！



起動が確認できたら、コンテナを停止しておきます。

```
docker stop コンテナ名またはコンテナID
```

php用のdocker-compose.ymlを用意

最終的にはdocker-compose up -dで一発起動するようにするので、docker-compose.ymlを作っていきます。

ディレクトリ構成

まず、以下のようなディレクトリ構成にします。

作成したDockerfileはdocker/php配下に配置してください。

ないフォルダは作成してください。

その他のファイルは次で説明します。

```
└─docker
  │   └─httpd
  │       └─httpd.conf
  │   └─php
  │       └─Dockerfile
```



```
|      └─php.ini
└─docker-compose.yml
```

docker-compose.yml

docker-compose.ymlは以下の通りです。

docker-compose.yml

```
version: '3'

services:
  php:
    build:
      context: ./docker/php/
      dockerfile: Dockerfile
    volumes:
      - ../app
      - ./docker/php/php.ini:/etc/php.ini
      - ./docker/httpd/httpd.conf:/etc/httpd/conf/httpd.conf
    ports:
      - 8080:80
```

サービス名は `php` にしています。

buildでは、`./docker/php/` の `Dockerfile` を指定。

volumesでは、dockerフォルダのphp.iniとhttpd.confの設定をコンテナに反映し、コンテナのappフォルダをマウントしホストから今後ファイル編集できるようにしています。

portsでは、ホストのポート8080番とコンテナのポート80番をつなげています。

これでホストの <http://localhost:8080/> にアクセスすると、コンテナの80番に転送され、`/app` の内容が表示されます (documentrootはhttpd.confで設定しています)。

php.ini

php.iniは以下の通りです。

php.ini

```
[Date]
date.timezone = "Asia/Tokyo"

[mbstring]
mbstring.internal_encoding = "UTF-8"
mbstring.language = "Japanese"
```

httpd.conf

httpd.confは、コンテナ内部からhttpd.confファイルをコピーしてきます。

```
# コピーするには「docker cp」コマンドを使用します。  
# 引数は「コンテナID: コンテナパス ローカルパス」と指定します。  
  
$ docker cp コンテナID:/etc/httpd/conf/httpd.conf docker/httpd/
```

コピーできたら、下記の部分を変更します。

httpd.confの変更点は以下の通りです。

httpd.conf

~~長いので省略~~

初期設定からの変更点①

#ServerName www.example.com:80

↓

ServerName www.example.com:80

初期設定からの変更点②

DocumentRoot "/var/www/html"

↓

DocumentRoot "/app"

php用のdocker-compose.ymlを起動

先程のdocker-compose.ymlをもとに、ビルドして起動します。

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

<http://localhost:8080/> にアクセスすると、Apacheの画面が表示されると思います。

ホストのfuel-app/の中にindex.phpを作って、再度アクセスしてみましょう。

index.php

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <title>php5.3.3-apache</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
<?php phpinfo(); ?>
</body>
</html>
```

以下の画面が表示されたら、成功です！

コンテナは止めておきましょう。

```
docker-compose stop
```

ここまでで、PHP5.3.3とApacheの環境ができました。

残るMySQLとphpMyAdminが動く環境を用意する。

残りのMySQLとphpMyAdminはどちらも公式イメージを使うので、docker-compose.ymlで完結します。

MySQLのdocker-compose.yml

以下の内容を追加するだけです。

```
docker-compose.yml
```

```
mysql:
  image: mysql:5.7
  volumes:
    - ./docker/mysql/volumes:/var/lib/mysql
  environment:
    - MYSQL_ROOT_PASSWORD=root
    - MYSQL_DATABASE=test
```

```
- MYSQL_USER=test
- MYSQL_PASSWORD=test
```

imageは、公式のイメージを使用しています。

volumesは、コンテナの `/var/lib/mysql` にMySQLな大事な設定ファイルが入るのでそれをホスト側で見れるにしています。また、コンテナを停止してもvolumesは削除しない限り消えないので、データが永続化されます。

environmentでは、環境変数を設定しています。意味は、docker hubを見るとわかりやすいです。

mysql - Docker Hub

これで、docker-compose up -dすると、mysqlコンテナが出来上がります。
コンテナログイン後、`mysql -uroot -proot` でmysqlにログインできます。

phpmyadminのdocker-compose.yml

以下の内容を追記するだけです。

```
docker-compose.yml
```

```
phpmyadmin:
  image: phpmyadmin/phpmyadmin
  environment:
    - PMA_ARBITRARY=1
    - PMA_HOST=mysql
```

```
- PMA_USER=root
- PMA_PASSWORD=root

links:
- mysql

ports:
- 4040:80

volumes:
- ./phpmyadmin/sessions:/sessions
```

imageは、公式のイメージを使っています。

environmentでは、環境変数を設定しています。これもdocker hubを見るとわかりやすいです。

[phpmyadmin/phpmyadmin - Docker Hub](#)

これで、docker-compose up -dして、<http://localhost:4040/> にアクセスするとphpMyAdminが開きます！

localhost:4040

phpMyAdmin

最近 お気に入り

- information_schema
- test

サーバ: mysql

データベース SQL 状態 エクスポート インポート 設定 変数 文字セット その他

一般設定

サーバ接続の照合順序: utf8mb4_unicode_ci

外観の設定

言語 - Language: 日本語 - Japanese

テーマ: pmahomme

フォントサイズ: 82%

詳細設定

データベースサーバ

- サーバ: mysql via TCP/IP
- サーバの種類: MySQL
- サーバの接続: SSLは使用されていません
- サーバのバージョン: 5.7.27 - MySQL Community Server (GPL)
- プロトコルバージョン: 10
- ユーザ: test@172.19.0.4
- サーバの文字セット: cp1252 West European (latin1)

ウェブサーバ

- Apache/2.4.38 (Debian)
- データベースクライアントのバージョン: libmysql - mysqlnd 5.0.12-dev - 20150407 - \$Id: 3591daad22de08524295e1bd073aceeff11e657\$
- PHP 拡張: mysqli curl mbstring
- PHP のバージョン: 7.2.22

phpMyAdmin

- バージョン情報: 4.9.1 (最新版)
- ドキュメント
- phpMyAdmin オフィシャルサイト

コンソール

最終的なdocker-compose.yml

最終的には下記のようになります。

docker-compose up -dして正常に動くか確認してみてください。

docker-compose.yml

```
version: '3'

services:
  php:
    build:
      context: ./docker/php/
      dockerfile: Dockerfile
    volumes:
      - ../app
      - ./docker/php/php.ini:/etc/php.ini
      - ./docker/httpd/httpd.conf:/etc/httpd/conf/httpd.conf
    ports:
      - 8080:80
  mysql:
    image: mysql:5.7
    volumes:
      - ./docker/mysql/volumes:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=test
      - MYSQL_USER=test
```

```
- MYSQL_PASSWORD=test

phpmyadmin:
  image: phpmyadmin/phpmyadmin
  environment:
    - PMA_ARBITRARY=1
    - PMA_HOST=mysql
    - PMA_USER=root
    - PMA_PASSWORD=root
  links:
    - mysql
  ports:
    - 4040:80
  volumes:
    - ./docker/phpmyadmin/sessions:/sessions
```

FuelPHPをインストールしてFuelPHPの画面を表示するまでの流れ

[GitHubのREADME.md](#)に詳しく書いてあるのでそちらを参照してください。

さいごに

ここまでくるのに約2日かかりました。。

fuelphp自体が初めてで、何が必要か理解するまでに時間がかかりました。

fuelphpのインストールをシェルスクリプトで完結させたり、`httpd.conf`をわざわざ2回修正したり(`documentroot`は存在しているディレクトリである必要がある)しているので、もうすこしい感じにできたらしたいなーと思います。

→ `httpd.conf`はVirtualHost機能を使うことで存在しないディレクトリを指定しても、コンテナが立ち上がるようになったので解決！

あと、正直なところがつつりfuelphpで動かしてみていないのでMySQLの連携はこれで大丈夫か心配だったりします。

とりあえずfuelphpのチュートリアルをいくつかやってみて不具合があれば修正していきます。

2019.10.12追記

phpコンテナで `php oil refine migrate` でマイグレーションを実行するとエラーになる。

原因は、phpコンテナとmysqlコンテナはつながっていないため、

phpコンテナから `/var/lib/mysql/mysql.sock` を見に行ってもないと怒られていた。

```
Uncaught exception Fuel\Core\Database_Exception: SQLSTATE[HY000] [2002] Can't connect to local MySQL server through socket '/var/lib/mysql/
Callstack:
```

```
#0 /app/fuel-app/fuel/core/classes/database/pdo/connection.php(193): Fuel\Core\Database_PDO_Connection->connect()
#1 /app/fuel-app/fuel/core/classes/database/query.php(305): Fuel\Core\Database_PDO_Connection->query(1, 'SELECT * FROM `...`, false)
#2 /app/fuel-app/fuel/core/classes/dbutil.php(644): Fuel\Core\Database_Query->execute(NULL)
#3 /app/fuel-app/fuel/core/classes/migrate.php(608): Fuel\Core\DBUtil::table_exists('migration')
#4 /app/fuel-app/fuel/core/classes/migrate.php(74): Fuel\Core\Migrate::table_version_check()
#5 [internal function]: Fuel\Core\Migrate::_init()
#6 /app/fuel-app/fuel/core/classes/autoloader.php(375): call_user_func('Migrate::_init')
```

```
#7 /app/fuel-app/fuel/core/classes/autoloader.php(249): Fuel\Core\Autoloader::init_class('Migrate')
#8 [internal function]: Fuel\Core\Autoloader::load('Migrate')
#9 /app/fuel-app/fuel/core/tasks/migrate.php(235): spl_autoload_call('Migrate')
#10 /app/fuel-app/fuel/core/tasks/migrate.php(161): Fuel\Tasks\Migrate::_run('default', 'app')
#11 /app/fuel-app/fuel/core/base.php(455): Fuel\Tasks\Migrate->__call('run', Array)
#12 /app/fuel-app/fuel/core/base.php(455): Fuel\Tasks\Migrate->run()
#13 /app/fuel-app/fuel/packages/oil/classes/refine.php(108): call_fuel_func_array(Array, Array)
#14 [internal function]: Oil\Refine::run('migrate', Array)
#15 /app/fuel-app/fuel/packages/oil/classes/command.php(126): call_user_func('Oil\Refine::run', 'migrate', Array)
#16 /app/fuel-app/oil(68): Oil\Command::init(Array)
#17 {main}
```

Previous exception:

Uncaught exception PDOException: SQLSTATE[HY000] [2002] Can't connect to local MySQL server through socket '/var/lib/mysql/mysql.sock' (2)
Callstack:

```
#0 /app/fuel-app/fuel/core/classes/database/pdo/connection.php(95): PDO->__construct('mysql:host=loca...', 'root', 'root', Array)
#1 /app/fuel-app/fuel/core/classes/database/pdo/connection.php(193): Fuel\Core\Database_PDO_Connection->connect()
#2 /app/fuel-app/fuel/core/classes/database/query.php(305): Fuel\Core\Database_PDO_Connection->query(1, 'SELECT * FROM `...`, false)
#3 /app/fuel-app/fuel/core/classes/dbutil.php(644): Fuel\Core\Database_Query->execute(NULL)
#4 /app/fuel-app/fuel/core/classes/migrate.php(608): Fuel\Core\DBUtil::table_exists('migration')
#5 /app/fuel-app/fuel/core/classes/migrate.php(74): Fuel\Core\Migrate::table_version_check()
#6 [internal function]: Fuel\Core\Migrate::_init()
#7 /app/fuel-app/fuel/core/classes/autoloader.php(375): call_user_func('Migrate::_init')
#8 /app/fuel-app/fuel/core/classes/autoloader.php(249): Fuel\Core\Autoloader::init_class('Migrate')
#9 [internal function]: Fuel\Core\Autoloader::load('Migrate')
#10 /app/fuel-app/fuel/core/tasks/migrate.php(235): spl_autoload_call('Migrate')
#11 /app/fuel-app/fuel/core/tasks/migrate.php(161): Fuel\Tasks\Migrate::_run('default', 'app')
#12 /app/fuel-app/fuel/core/base.php(455): Fuel\Tasks\Migrate->__call('run', Array)
#13 /app/fuel-app/fuel/core/base.php(455): Fuel\Tasks\Migrate->run()
#14 /app/fuel-app/fuel/packages/oil/classes/refine.php(108): call_fuel_func_array(Array, Array)
#15 [internal function]: Oil\Refine::run('migrate', Array)
```

```
#16 /app/fuel-app/fuel/packages/oil/classes/command.php(126): call_user_func('Oil\Refine::run', 'migrate', Array)
#17 /app/fuel-app/oil(68): Oil\Command::init(Array)
#18 {main}
```

まず、mysqlコンテナを調べると、`/var/lib/mysql/mysql.sock` はなくあったのは `/var/run/mysqld/mysqld.sock`

この場合、`/var/run/mysqld/mysqld.sock` をphpコンテナから見えるようにしてあげればいい。

ただ、phpコンテナから見ているのは、`/var/lib/mysql/mysql.sock` であったため、`/var/run/mysqld/mysqld.sock` にするとphpコンテナを修正する手間があった。

そのため、今回phpコンテナ側は修正をしなくなかったので、`/var/lib/mysql/mysql.sock` に合わせることにした。

①mysqlコンテナの `/etc/mysql/my.cnf` をホストの `/docker/mysql/my.cnf` でマウント。

設定は、`/var/lib/mysql/mysql.sock` を見るようにした。

② ①の対応をした上で、phpコンテナからも `/etc/mysql/my.cnf` をホストの `/docker/mysql/my.cnf` でマウント。

phpコンテナからも `/var/lib/mysql/mysql.sock` が見えるようになったことでMySQLとの通信が成功し、マイグレーションも成功した。

変更ファイルは以下の通り。

my.cnf

```
[mysqld]
socket=/var/lib/mysql/mysql.sock
```

docker-compose.yml

```
version: '3'

services:
  php:
    build:
      context: ./docker/php/
      dockerfile: Dockerfile
    volumes:
      - ./app
      - ./docker/php/php.ini:/etc/php.ini
      - ./docker/httpd/httpd.conf:/etc/httpd/conf/httpd.conf
      - ./docker/mysql/volumes:/var/lib/mysql
    ports:
      - 8080:80
  mysql:
    image: mysql:5.7
    volumes:
      - ./docker/mysql/my.cnf:/etc/mysql/my.cnf
      - ./docker/mysql/volumes:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=test
      - MYSQL_USER=test
      - MYSQL_PASSWORD=test
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    environment:
      - PMA_ARBITRARY=1
      - PMA_HOST=mysql
      - PMA_USER=root
```

```
- PMA_PASSWORD=root
links:
  - mysql
ports:
  - 4040:80
volumes:
  - ./docker/phpmyadmin/sessions:/sessions
```

2019.10.13追記その1

fuelphpのディレクトリ構成を変更したので、それに合わせてhttpd.conf修正しました。

fuelphpを含んだディレクトリ構成は下記のようにになりました。

```
├─docker
|   ├─httpd
|   |   └─httpd.conf
|   ├─mysql
|   |   └─my.conf
|   └─php
|       └─Dockerfile
|           └─php.ini
├─sample(fuelphp)
└─docker-compose.yml
```


httpd.confの変更点は以下の通りです。

httpd.conf

```
# ①ServerNameは、VirtualHostを使うのでコメントアウト
#ServerName www.example.com:80

# ②Documentorootも、VirtualHostを優先して使わないのでデフォルト設定に戻した
DocumentRoot "/var/www/html"

# ③NameVirtualHostをコメントイン。localhost:80に対して、仮想ホストを設定
NameVirtualHost *:80

# ④VirtualHostの設定を追加。localhost:8080 → localhost:80に転送され、下記の設定が有効になる。htaccessを有効にするために、AllowOverride All
<VirtualHost *:80>
    DocumentRoot /var/www/html/public
    ServerName localhost
    <Directory /var/www/html/public>
        Order Allow,Deny
        Allow from All
        Options All
        AllowOverride All
        DirectoryIndex index.php index.html
    </Directory>
</VirtualHost>
```

2019.10.13追記その2

いざFuelPHP触ってチュートリアルでもやろうかなーと思ったらデバッガー入れてないと思い、試行錯誤して入れました。
変更点は下記です。

Dockerfile

```
FROM centos:6.9
```

```
ENV PHP_VERSION 5.3.3
```

```
RUN yum install -y \  
    php \  
    php-mbstring \  
    php-pgsql \  
    php-mysql \  
    php-gd \  
    zip \  
    unzip \  
    git \  
    httpd \  
    php-devel \  
    httpd-devel \  
    php-pear \  
    gcc && \  
    yum clean all
```

```
RUN pecl channel-update pecl.php.net && \  
    pecl install xdebug-2.2.7
```

```
RUN mkdir /app
```

```
WORKDIR /app
```

EXPOSE 80

```
CMD ["/usr/sbin/apachectl","-DFOREGROUND"]
```

peclをインストール必要があったので、php-devel、httpd-devel、php-pearをインストール。

CentOS 6、peclコマンドを使えるようにする | マコトのおもちゃ箱 ～ぼへぼへ自営業者の技術メモ～

peclでxdebugをインストールする際にgccが必要だったので、gccもインストール。インストール時にpeclのエラーがでたので、pecl channel-update pecl.php.netを追加。

xdebugは5.3系が使える2.2.7を指定。

PHPでモジュールを読み込むよう設定ファイルを作成しました。

PHP5.3系にxdebugをインストールする際のあれこれ | Skyarch Broadcasting

xdebug.ini

```
zend_extension=/usr/lib64/php/modules/xdebug.so
```

ホスト側のxdebug.iniを読み込むようにdocker-compose.ymlのvolumesに追加。

docker-compose.yml

```
version: '3'
```

```
services:
```

```
php:
  build:
    context: ../docker/php/
    dockerfile: Dockerfile
  volumes:
    - ../app
    - ../docker/php/php.ini:/etc/php.ini
    - ../docker/httpd/httpd.conf:/etc/httpd/conf/httpd.conf
    - ../docker/mysql/volumes:/var/lib/mysql
    - ../docker/php/xdebug.ini:/etc/php.d/xdebug.ini
  ports:
    - 8080:80
mysql:
  image: mysql:5.7
  volumes:
    - ../docker/mysql/my.cnf:/etc/mysql/my.cnf
    - ../docker/mysql/volumes:/var/lib/mysql
  environment:
    - MYSQL_ROOT_PASSWORD=root
    - MYSQL_DATABASE=test
    - MYSQL_USER=test
    - MYSQL_PASSWORD=test
phpmyadmin:
  image: phpmyadmin/phpmyadmin
  environment:
    - PMA_ARBITRARY=1
    - PMA_HOST=mysql
    - PMA_USER=root
    - PMA_PASSWORD=root
  links:
    - mysql
```

```
ports:
  - 4040:80

volumes:
  - ./docker/phpmyadmin/sessions:/sessions
```

あとは、xdebugがVScodeが使えるようにするための設定を追加しています。

Docker × Visual Studio CodeでPHP開発【デバッグ実行】 - Qiita

php.ini

```
[Date]
date.timezone = "Asia/Tokyo"

[mbstring]
mbstring.internal_encoding = "UTF-8"
mbstring.language = "Japanese"

[xdebug]
xdebug.remote_enable=1
xdebug.remote_autostart=1

; ホスト側のIP
; host.docker.internalはdockerのhostマシンのIPを解決してくれます。
; hostマシン以外のIPの場合は適宜IPを調べて設定してください。
xdebug.remote_host=host.docker.internal

; 空いているport番号（xdebugのデフォルトは9000）。私の場合は他と競合していたので9001に設定。
xdebug.remote_port=9001

; xdebugの出力するログの場所。今回は適当に/tmp配下に。
xdebug.remote_log=/tmp/xdebug.log
```

launch.json

```
{  
  // IntelliSense を使用して利用可能な属性を学べます。  
  // 既存の属性の説明をホバーして表示します。  
  // 詳細情報は次を確認してください: https://go.microsoft.com/fwlink/?linkid=830387  
  "version": "0.2.0",  
  "configurations": [  
    {  
      "name": "Listen for XDebug",  
      "type": "php",  
      "request": "launch",  
      "port": 9001,  
      "pathMappings": {  
        // Dockerコンテナのdocument root : 開発環境のdocument root  
        "/app/sample/public": "${workspaceFolder}/sample/public"  
      }  
    }  
  ]  
}
```

2019.10.15追記

windows環境で構築してみると、windows特有のエラーに遭遇。

Macでは問題ないが、WindowsだとMySQLのデータディレクトリ以外の場所にしないといけないらしい。

つまり、/var/lib/mysql/mysql.sockなどをvolumesにすることができない。
そのため、/tmp/mysql.sockに変更をおこなった。

MySQL :: MySQL 5.7 Reference Manual :: 2.5.7.3 Deploying MySQL on Wind...

<https://dev.mysql.com>

また、色々作業している中で副作用があり下記で解決したのでメモ。

MySQL (MariaDB) のコンテナでは、データベースという特性上、Dockerのボリューム機能を合わせて使うことが多いです。

そのため、docker-compose.yml で環境変数をいじったり、初期化スクリプトをちょっと修正して再実行しても初回起動ではないと判定されて、全くデータベースに反映されないということが普通に起こります。

DockerオフィシャルのMySQL (MariaDB) コンテナの挙動をDockerfile周辺から読み解く - Qiita

<https://qiita.com>

Dockerオフィシャルの
MySQL (MariaDB) コンテナの
挙動をDockerfile周辺から読み解く

@kazuyoshikakihara

Qiita

最終的なファイルは下記。

docker-compose.yml

```
version: '3'
```

```
services:
```

```
php:
  build:
    context: ../docker/php/
    dockerfile: Dockerfile
  volumes:
    - ../app
    - ../docker/php/php.ini:/etc/php.ini
    - ../docker/httpd/httpd.conf:/etc/httpd/conf/httpd.conf
    - ../docker/mysql/volumes:/var/lib/mysql
    - ../docker/php/xdebug.ini:/etc/php.d/xdebug.ini
    - ../docker/mysql:/tmp/mysql.sock

  ports:
    - 8080:80

mysql:
  image: mysql:5.7
  volumes:
    - ../docker/mysql/my.cnf:/etc/mysql/my.cnf
    - ../docker/mysql/volumes:/var/lib/mysql
    - ../docker/mysql:/var/run/mysqld/mysqld.sock

  environment:
    - MYSQL_ROOT_PASSWORD=root
    - MYSQL_DATABASE=test
    - MYSQL_USER=test
    - MYSQL_PASSWORD=test

phpmyadmin:
  image: phpmyadmin/phpmyadmin
  environment:
    - PMA_ARBITRARY=1
    - PMA_HOST=mysql
    - PMA_USER=root
    - PMA_PASSWORD=root
```



```
links:
  - mysql

ports:
  - 4040:80

volumes:
  - ./docker/phpmyadmin/sessions:/sessions
```

```
[client]
socket=/tmp/mysql.sock

[mysqld]
socket=/tmp/mysql.sock
```

また、windowsで構築する際db.phpは下記のようにする必要がある。
dsn部分のみ抜粋。

db.php

```
'dsn' => 'mysql:host=mysql;dbname=fuel_dev;unix_socket=/tmp/mysql.sock',
```

これでマイグレーションまで動いた。



新規登録して、もっと便利にQiitaを使ってみよう

1. あなたにマッチした記事をお届けします
2. 便利な情報をあとで効率的に読み返せます

[ログインすると使える機能について](#)

新規登録

ログイン



snyt 45

@snyt45

フォロー



株式会社diddyworks

【スポーツと生きれる世界を創る】をミッションに、スポーツマッチングプラットフォーム「Liss（リス）」を運営してます。スポーツするなら、Liss！

<https://diddyworks.co.jp/>

フォロー



関連記事 Recommended by LOGLY



docker-compose を用いて Apache ・ PHP ・ MySQL の開発環境を構築してみた

by sugurutakahashi12345



はじめてのLaravel5 + Docker

by kukimo



Docker Composeで手軽に開発環境を構築 (PHP+MySQL+Elasticsearch...)

by acro5piano



Dockerを利用してFuelPHP開発環境構築

by RyosukeKamei



目に直接レンズを入れる視力矯正。眼内コンタクトレンズのICL

PR スター・ジャパン合同会社



PagerDuty×Qiita対談！インシデント対応ソリューションの導入事例と効果

PR PagerDuty株式会社

コメント

この記事にコメントはありません。

あなたもコメントしてみませんか :)

新規登録

すでにアカウントを持っている方は[ログイン](#)

記事投稿キャンペーン開催中

RubyKaigi 2023 連動企画

みんなで
Rubyの知見
を共有しよう

5.1.mon~5.31.wed



presented by Qiita

【RubyKaigi 2023連動イベント】 みんなでRubyの知見を共有しよう

2023/05/01~2023/05/31

詳細を見る

\ 最新技術について /
Udemyで学んだこと
をシェアしよう!

5.1.mon~5.31.wed

presented by Udemy

最新技術についてUdemyで学んだことをシェアしよう！

2023/05/01~2023/05/31

詳細を見る

すべて見る ➔

Qiita

How developers code is here.

© 2011-2023 Qiita Inc.

ガイドとヘルプ

About

利用規約

プライバシーポリシー

ガイドライン

デザインガイドライン

ご意見

ヘルプ

広告掲載

コンテンツ

リリースノート

公式イベント

公式コラム


募集


アドベントカレンダー

Qiita 表彰プログラム


API

SNS

 Qiita（キータ）公式

 Qiita マイルストーン

 Qiita 人気の投稿

 Qiita（キータ）公式

Qiita 関連サービス

Qiita Team

Qiita Jobs

Qiita Zine

Qiita 公式ショップ

運営

運営会社

採用情報

Qiita Blog