

Docker環境構築

バージョン	0.1
作成日	2022/08/26
作成者	k.eguchi
更新日	2022/08/26
更新者	k.eguchi

※本資料は著作物のため転載禁止です※

◆ 改訂履歴

No	バージョン	改訂日	改訂者	備考
1	0.1	2022/08/26	k.eguchi	新規作成
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

目次

1. イントロダクション	p.4
2. Dockerのインストール	p.5
3. プロジェクトの作成	p.11
4. プロジェクトファイルについて	p.13
5. コンテナ構築	p.16
6. Laravelプロジェクトの作成	p.18
7. 疎通確認	p.19

◆イントロダクション

○この章について

システム自作を行うための環境構築として、下記の順番に準備を進めます。

- ①Dockerのインストール(Docker Desktop導入)
- ②プロジェクトファイルの作成
- ③コンテナ構築
- ④Laravelプロジェクトの作成
- ⑤.envファイルの修正(疎通確認)

少し多いですが、1つずつ進めていきましょう。

◆Dockerインストール

○Docker Desktopの導入

まずは仮想環境の土台となる、Dockerの導入を進めていきます。

導入手順はMacとWindowsで分かりますので、ご自身のPCと一致する手順を参照して進めてください。

①**Windows**の方 → P.6~P.9

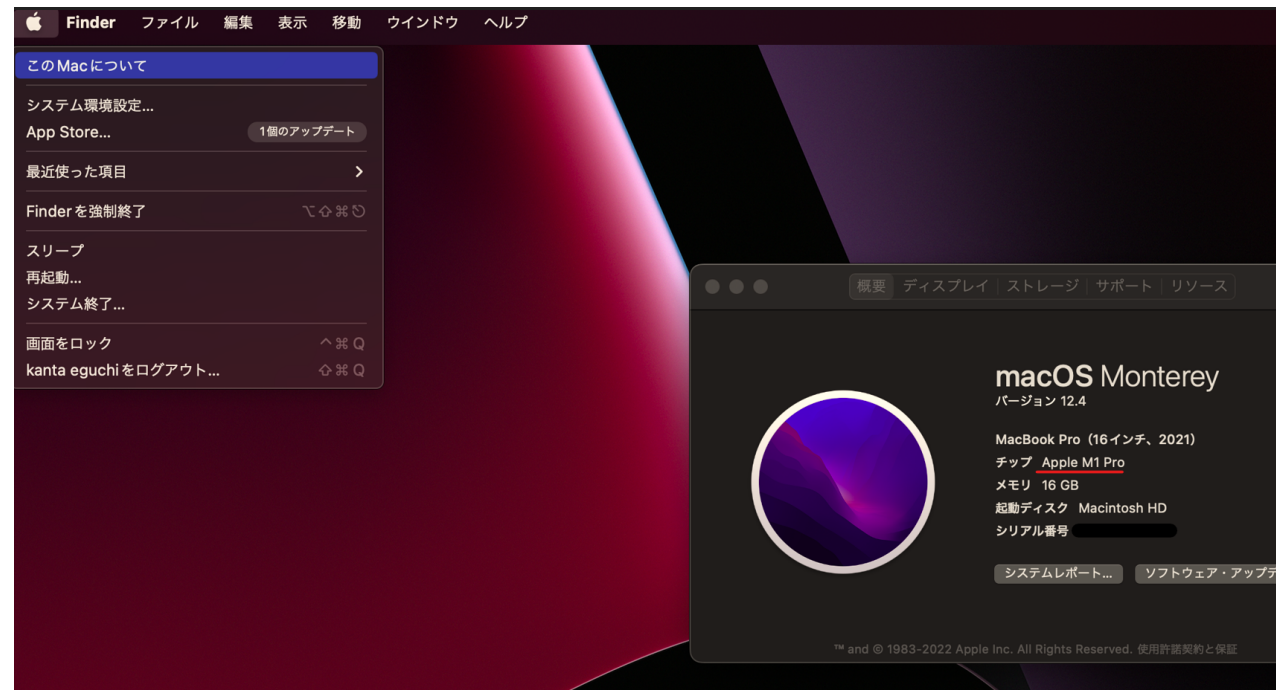
Macを使用されている方は更に2パターンに分かれます。

②**Intel**チップ

③**M1**チップ

Macのチップ確認は右図を参考にしてください。

右上のリンゴマークから、
[この**Mac**について]
でチップのタイプを確認できます。



◆Dockerインストール: Windows編

○WSL2の導入

Windowsの場合は、Dockerの導入に先立って、WSLというものを導入する必要があります。

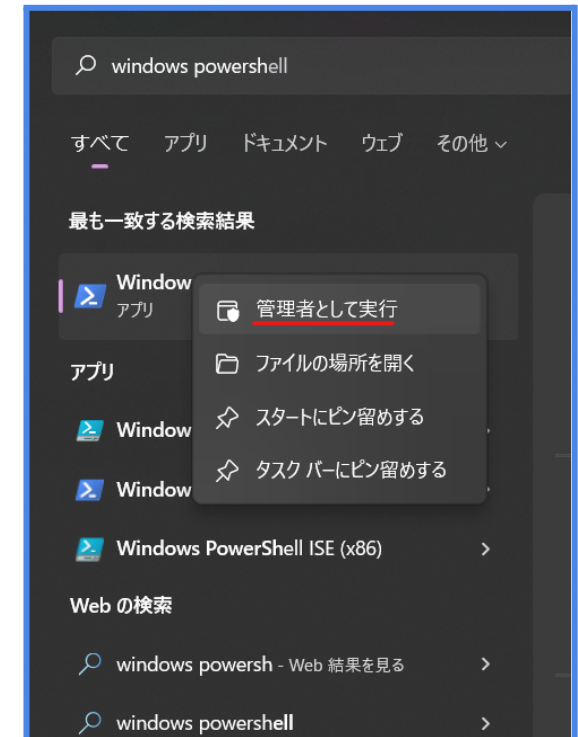
WSLの導入はWindows PowerShellを使用して行います。

Windows PowerShellを管理者として実行(右図)し、
wsl --install
とコマンドを実行してください。

諸々のインストール完了後、PCを再起動します。

再起動後、**Ubuntu**というものが立ち上がります。
(無かったらWindows StoreよりDLしてください。)

ユーザー名の設定等を求められるので、表示に従って進めてください。

A screenshot of the Ubuntu WSL2 installation progress window. The text inside reads: "Installing, this may take a few minutes... Please create a default UNIX user account. The username does not need to match your Windows username. For more information visit: https://aka.ms/wslusers Enter new UNIX username: _". The window has a blue border and a dark background.

◆Dockerインストール: Windows編

○WSL2の導入

Ubuntuの設定が完了したら、改めてWindows PowerShellを開き、[`wsl -l -v`]とコマンドを打ってみましょう。
Ubuntuの表示がされていればOKです。(画像ではユーザー名の部分を白塗りしています)

```
PS C:\Users\ [ユーザー名] > wsl -l -v
```

NAME	STATE	VERSION
* Ubuntu-20.04	Running	2

WSLの導入は完了しました。

ここからは追加要素ですが、
UbuntuとPowerShellでは使用できるコマンドが異なります。

これはPowerShellはWindowsシステム、UbuntuはLinuxというシステムの元で動いているためです。
コマンドによってアプリを切り替えてコマンド実行するのは面倒です。

Windows TerminalというCLIをダウンロードしておくと便利です。

◆Dockerインストール: Windows編

○Docker Desktopインストール

Dockerページ(<https://docs.docker.com/desktop/install/windows-install/>)に移動し、インストーラをダウンロードします。

ダウンロードが完了したら、インストールを実行します。

設定を聞かれた際は、
下記のようにWSL2を使用することを指定します。

Configuration

- ☒ Install required Windows components for WSL 2
- ☒ Add shortcut to desktop

インストールが完了したら、DockerDesktopが起動します。

CLIで [**docker --version**]を実行し、バージョン情報が表示されたら導入完了です。

```
PS C:\> docker --version
Docker version 20.10.17, build 100c701
PS C:\>
```



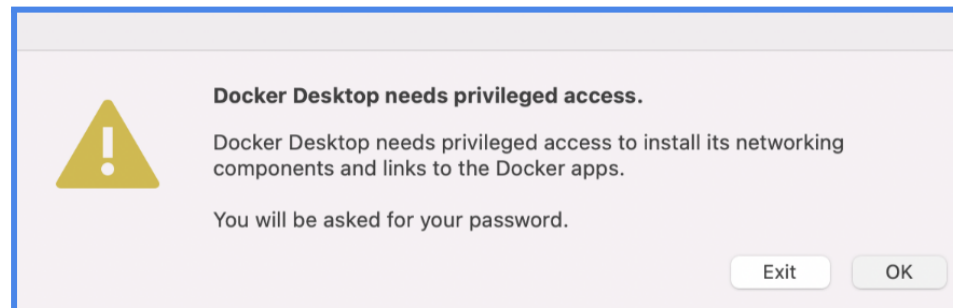
◆Dockerインストール:Mac編

○Docker Desktopインストール

Dockerのページ(<https://matsuand.github.io/docs.docker.jp.onthefly/desktop/mac/install/>)に移動し、自身のチップタイプに合うインストーラをダウンロードしてください。

Intel版のMacを使用している方はこのままインストールを進めてください。

インストール時にアクセス権限を求める表示が出ることがあります。



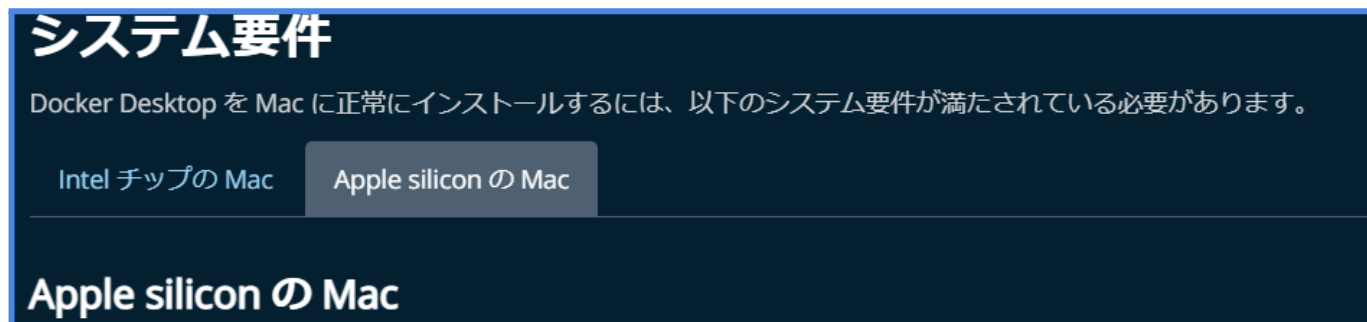
OKをクリックした後にパスワードの入力を求められるので、指示に従って入力してください。



◆Dockerインストール:Mac編 - M1 チップ

○Docker Desktopインストール

M1チップ使用の方は先ほどのDockerのページ(<https://matsuand.github.io/docs.docker.jp.onthefly/desktop/mac/install/>)より、[システム要件]の項目から[**Apple Silicon の Mac**]のタブを確認してください。



M1チップでDockerDesktopを動作させる際、一部使えないコマンドが存在します。
そうした問題を回避するため**Rosetta 2**というソフトウェアをインストールしておきます。

CLIより **softwareupdate --install-rosetta** を実行することでインストールできます。

問題の詳細については<https://matsuand.github.io/docs.docker.jp.onthefly/desktop/mac/apple-silicon/>を確認してください。

(Rosetta2はIntelMac用のバイナリデータをM1用に変換するエミュレータです)

Rosetta2のインストールが完了したら、後はIntel版と同様にインストールを進めてください。

◆プロジェクトの作成

○プロジェクトの構成

Dockerの準備が出来たので、ここからはプロジェクトの準備を進めていきます。

今回の構成は

Nginx + Laravel + MySQL + phpMyAdmin
です。

Nginxはサーバーシステムです。

PHP等のプログラムはサーバーにコードを渡し、そこにアクセスすることで実行されます。

以前のカリキュラムで使用していたMAMPでは、**MAMP**内で**Apache**サーバーシステムを走らせていました。

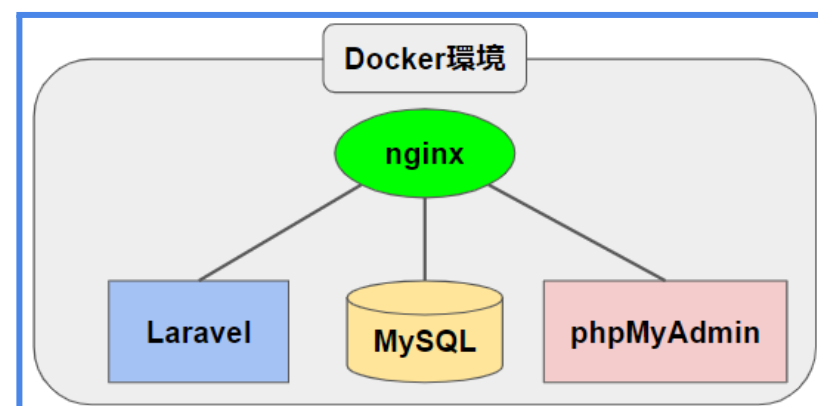
今回はそれと同様に、**Docker**環境内で**Nginx**サーバーを走らせます。

その他についてはご存じの通り、プロジェクト本体(Laravel)、DB、DB管理ツールです。

今回のプロジェクトの模式図を右に示しました。

LaravelやMySQL, phpMyAdminそれぞれがNginxを介してやり取りを行います。

これらがそれぞれが今回のプロジェクトの**Docker**コンテナになります。



◆プロジェクトの作成

○プロジェクトファイルの作成

自作プロジェクト用のフォルダを作成してください。
そのディレクトリに、資料同梱の**docker-env**フォルダの中身を展開してください。

構成は右図、内容は以下の通りです。

appディレクトリ

…Laravelプロジェクトファイル群が入ります

dbディレクトリ

…DB関連のファイル群が入ります

dockerディレクトリ

…コンテナのDockerfileや設定ファイルの置き場です

docker-compose.yml

…コンテナ群の関連性や使用するイメージを明記したファイルです。詳細は後ほど解説します。

```
C:.\n|   docker-compose.yml\n|\n+---app\n+---db\n\---docker\n    +---nginx\n    |       default.conf\n    |\n    \---php\n            Dockerfile
```

◆プロジェクトファイルについて

○Dockerfile

docker/php にあるDockerfileを以下に示しています。

```
1 FROM php:7.4-fpm
2
3 COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
4
5 RUN apt-get update && apt-get install -y \
6 git \
7 && zip unzip \
8 && docker-php-ext-install pdo_mysql
```

Dockerfileで頻出の最重要ポイントは

FROM と **RUN**

の2つです。

FROMでは使用するDockerイメージ、つまりコンテナの素材となるモノの指定を行います。
このコンテナではLaravelを動かしたいので、PHPのイメージを使用してコンテナを立ち上げます。

RUNでは、イメージを使用してコンテナを立ち上げる際に自動的に実行されるコマンドです。
今回はLaravelプロジェクトを立ち上げる際に必要なパッケージや、DBドライバのインストールを行っています。

◆プロジェクトファイルについて

○docker-compose.yml

docker-compose.ymlは **docker-compose**コマンドを使用する際の指示書のようなものです。

docker-composeコマンドは複数のコンテナから構成されるコンテナ群を関連付けて扱いやすくするためのコマンドです。

このコンテナ間の関連性や使用するイメージ、共有ファイル(後述)等を記述します。

このファイルはインデント(段落)が意味を持ちます。

右の例で行くと、

3行目以降がservices(コンテナ群)の記述、
4行目から13行目までのブロックがnginxコンテナの記述、
という風にインデントで一つのブロックになっています。

大体の記述内容は同梱のファイルにコメントアウトで記述しています。

次ページでは重要なポイントである

volumes:
の部分について解説します。

※重要

Mac M1チップを使用されている方は、docker-compose.ymlの
右部分のコメントアウトを解除して作業を進めてください。

```
1  version: '3.8'
2
3  services:
4    nginx:
5      container_name: curriculum-nginx
6      image: nginx:latest
7      volumes:
8        - ./docker/nginx:/etc/nginx/conf.d
9        - ./app:/var/www/html
10     ports:
11       - "80:80"
12     depends_on:
13       - php
14
15     php:
16       container_name: curriculum-laravel
```

```
23  php:
24    container_name: curriculum-laravel
25    #####
26    # Mac M1チップユーザーは以下のコメントアウトを解除して作業を進めてください
27    # platform: linux/x86_64
28    #####
29
```

◆プロジェクトファイルについて

○docker-compose.yml – volumes:

volumes: の記述は同期ファイルの指定を行っています。

同期ファイルの説明の前に、仮想環境とホストマシン(PC本体)の関連を考えてみます。

仮想環境は模型ハウスという例えをしましたが、置いてある場所は広く見れば家(ローカル)の中です。

Docker上に新しい仮想のマシンを用意したとしても、ファイル類が保存されるのはローカルPC上です。

同期ファイルは、コンテナ上に反映させるローカルファイルのことです。

volumes以下では

- ローカルのフォルダ: コンテナ内でのパス

という記述の仕方をします。

コンテナと同期しておくことで、ファイルの編集を即時コンテナにも反映することができるようになります。

```
1  version: '3.8'
2
3  services:
4    nginx:
5      container_name: curriculum-nginx
6      image: nginx:latest
7      volumes:
8        - ./docker/nginx:/etc/nginx/conf.d
9        - ./app:/var/www/html
10     ports:
11       - "80:80"
12     depends_on:
13       - php
14
15     php:
16       container_name: curriculum-laravel
```

◆コンテナ構築

○docker-compose up

ファイル類の確認が終わったら、いよいよコンテナの立ち上げです。

コンテナの立ち上げや操作は基本的に**docker-compose.yml**ファイルのある階層で行います。
まずは **cd** コマンドを使ってdocker-compose.ymlのある階層に移動します。

現在の階層に存在するファイルの確認には **ls** コマンドが便利です。

目的の階層まで移動が出来たら
docker-compose up --build -d
とコマンドを実行します。

コンテナの構築(ビルディング)が始まるので、
しばらく待ちます。

完了すると右のようにコンテナの構築が完了したと
メッセージが出ます。

```
PS C:\curriculum\docker-env> ls

ディレクトリ: C:\curriculum\docker-env

Mode                LastWriteTime         Length Name
----                -
d-----          2022/08/26      18:35         app
d-----          2022/08/27      13:17         db
d-----          2022/08/27      19:17        docker
-a-----          2022/08/29       2:20       1580 docker-compose.yml
```

```
PS C:\curriculum\docker-env> docker-compose up --build -d
Building php
[+] Building 2.8s (4/7)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/php:7.4-fpm
=> FROM docker.io/library/composer:latest
```

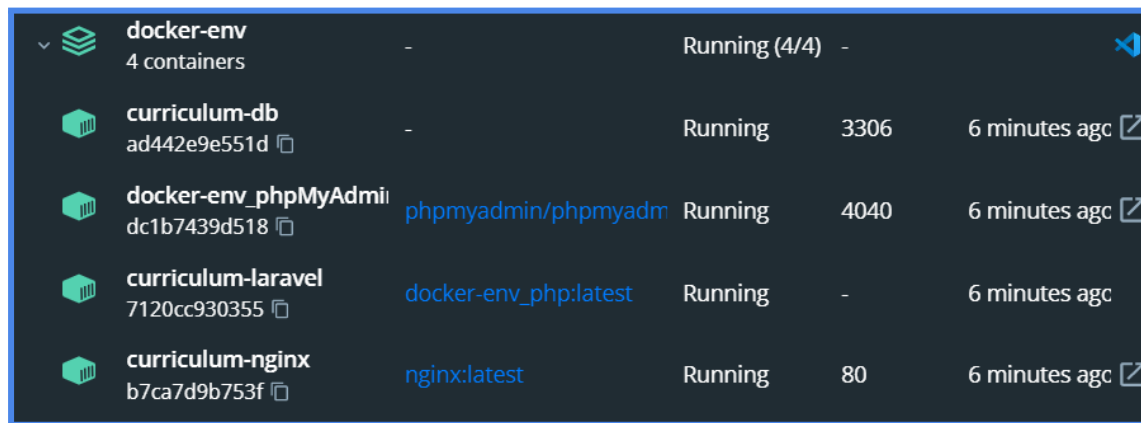
```
=> => naming to docker.io/library/docker-env_php
Creating curriculum-db ... done
Creating curriculum-laravel ... done
Creating docker-env_phpMyAdmin_1 ... done
Creating curriculum-nginx ... done
```







◆コンテナ構築

○コンテナ状況の確認

コンテナの構築が完了したらDockerDesktopからコンテナ状況を確認します。

下図のようにコンテナ全てが緑(稼働状態)を示していたらOKです。



▼		docker-env 4 containers	-	Running (4/4)	-	
		curriculum-db ad442e9e551d	-	Running	3306	6 minutes ago
		docker-env_phpMyAdmin dc1b7439d518	phpmyadmin/phpmyadm	Running	4040	6 minutes ago
		curriculum-laravel 7120cc930355	docker-env_php:latest	Running	-	6 minutes ago
		curriculum-nginx b7ca7d9b753f	nginx:latest	Running	80	6 minutes ago

また、一番上のdocker-envの部分は、プロジェクトフォルダ名が表示されます。

ここまで完了したらDockerによる仮想環境の準備は完了です。

◆Laravelプロジェクトの作成

○コンテナ内での作業

仮想環境上でLaravelプロジェクトを作成したい場合、コンテナ内でコマンドを実行する必要があります。
Laravelプロジェクトを作成するため、**PHPコンテナ**に入って作業します。

コンテナに入る、というのは操作をローカル側からではなくコンテナ側から行うという意味です。

以下のコマンドを実行すると、コンテナ内に入ることができます。

```
PS C:\curriculum\docker-env> docker-compose exec php bash
root@7120cc930355:/var/www/html#
```

ここで、Laravelプロジェクトを作成するコマンド

```
root@8ec67427c68c:/var/www/html# composer create-project "laravel/laravel=6.*" . --prefer-dist
```

を入力します。(プロジェクト名は指定せず上記コマンドのまま作成してください。)

諸々の処理が完了したのち、<http://127.0.0.1> または <http://localhost> にアクセスします。

Laravelのトップページが表示されたら、Laravelプロジェクトの構築は完了です。

※右のエラーが出た場合は、コンテナ内で

[**chown www-data storage/ -R**]

を実行してください。

UnexpectedValueException

The stream or file "/var/www/html/storage/logs/laravel.log" could not be opened in append mode: failed to open stream: Permission denied The exception occurred while attempting to log: The stream or file "/var/www/html/storage/logs/laravel.log" could not be opened in append mode: failed to open stream: Permission denied The exception occurred while attempting to log: The stream or file "/var/www/html/storage/logs/laravel.log" could not be opened in append mode: failed to open stream: Permission denied

<http://localhost/>

◆疎通確認

○.envファイルの修正

最後に環境設定を行う.envファイルを修正します。

Laravelプロジェクト内のenvファイルにある、**DB_HOST**を書き換えます。
DB_HOSTはdockerで作成したDBコンテナの名前である[**db**]に変更します。

以下のように変更できていればOKです。

```
DB_CONNECTION=mysql
DB_HOST=db
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=root
```

修正後は変更を反映するためにコンテナの再起動を行います。
コンテナの外に出て(docker-compose.ymlの階層)から、[**docker-compose up -d php**]としてphpコンテナを再起動します。

DB_DATABASEは、[接続先データベース名]になります。(今回は変更しません)

◆疎通確認

○phpMyAdminの確認

Laravelの表示、.envファイルの修正が完了したら、
<http://127.0.0.1:4040> または <http://localhost:4040> にアクセスしてください。

phpMyAdminが表示されたでしょうか。

ここから自作で使用するDBを作成してください。

自作ではDB名は「**laravel**」で統一してください。

これでDocker x Laravelの環境構築は完了です。