



@yoshiokaCB 2018年07月15日に更新



最低限知っておきたいDockerとdocker-composeの使い方

[Ruby](#) [Rails](#) [docker](#) [dockerfile](#) [docker-compose](#)



0



Dockerの基本的な使い方(よく使うコマンド)

docker pull (イメージの取得)

```
$ docker pull [REPOSITORY:TAG]
```

docker images (イメージの確認)

```
$ docker images
```

docker run (コンテナの起動)

```
$ docker run --name [任意の名前] -itd [REPOSITORY:TAG] /bin/bash
```

よく使うオプション

オプション	内容
-i	ホストの入力をコンテナの標準出力をつなげる（おまじない
-t	コンテナの標準出力とホストの出力をつなげる（おまじない
-d	バックグラウンドで起動
--rm	コマンド実行後にコンテナの削除も行う
--name	任意のコンテナ名をつける（指定しない場合は自動で名前がつけられる）

docker ps（起動しているコンテナの確認）

```
$ docker ps
```

docker ps -a（停止しているコンテナも含めた確認）

```
$ docker ps -a
```

docker exec (sshでログイン、各種コマンドの実行)

```
# ログイン
```

```
$ docker exec -it [CONTAINER ID or NAMES] /bin/bash
```

```
# ログインせずにコマンド実行
```

```
docker exec -it [CONTAINER ID or NAMES] [COMMAND]
```

※ NAMESは docker run で指定した名前を入力する

使用例 1

使用例 1

```
# docker pull でイメージをダウンロードする
```

```
#####
```

```
$ docker pull ruby:2.5.0
```

```
2.5.0: Pulling from library/ruby
```

```
3e731ddb7fc9: Pull complete
```

```
47cafa6a79d0: Pull complete
```

```
79fcf5a213c7: Pull complete
```

```
68e99216b7ad: Pull complete
```

```
4822563608bb: Pull complete
```

```
9d614f26bec1: Pull complete
```

```
1c758cfc0888: Pull complete
```

```
8a4fbc3666ca: Pull complete
```

```
Digest: sha256:ed5fc221d5d03d89e1f8c1f7780b98bc708e68b4d8dba73594d017e999156619
```

```
Status: Downloaded newer image for ruby:2.5.0
```

```
# ローカルにイメージが保存されているか確認する
```

```
#####
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ruby	2.5.0	bae0455cb2b9	2 weeks ago	80MB

```
# コンテナの起動(起動と同時にコンテナにログイン)
```

```
#####
```

```
$ docker run --name ruby25 -it ruby:2.5.0 /bin/bash
```

```
# ログイン後、Rubyの動作確認
```

```
root@6ed5f4e75858: /#
```

```
root@6ed5f4e75858: /# ruby -v
```

```
ruby 2.5.0p0 (2017-12-25 revision 61468) [x86_64-linux]
```

```
ruby 2.5.0p0 (2017-12-25 revision 61468) [x86_64-linux]
```

```
root@6ed5f4e75858:/# exit
```

起動中のコンテナ(プロセス)の確認

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
6ed5f4e75858	ruby:2.5.0	"/bin/bash"	About a minute ago	Up

停止中のコンテナ(プロセス)の確認

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
6ed5f4e75858	ruby:2.5.0	"/bin/bash"	About a minute ago	Up

コンテナを一旦削除する(rmコマンドについては後ほど説明)

```
$ docker rm ruby25
```

コンテナの起動(バックグラウンドで起動)

```
#####
```

```
$ docker run --name ruby25 -itd ruby:2.5.0 /bin/bash
```

```
d7583a4844bfcfb3e3d706360f20228bb84bee5f455c31bc293657f07a17101e
```

コンテナ(プロセス)が起動していることを確認

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
d7583a4844bf	ruby:2.5.0	"/bin/bash"	13 seconds ago	Up

起動中のコンテナ(プロセス)にログインし、Rubyが動くことを確認する

```
$ docker exec -it ruby25 /bin/bash
```

ログイン後、Rubyの動作確認

```
root@d7583a4844bf:/# ruby -v
```

```
ruby 2.5.0p0 (2017-12-25 revision 61468) [x86_64-linux]
```

```
root@d7583a4844bf:/# exit
```

ログアウト後 コンテナ起動していることを確認

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
d7583a4844bf	ruby:2.5.0	"/bin/bash"	4 minutes ago	Up

ログインせずにコマンドを実行

```
#####
```

```
$ docker exec -it ruby25 ruby -v
```

```
ruby 2.5.0p0 (2017-12-25 revision 61468) [x86_64-linux]
```

docker stop (コンテナの停止)

```
$ docker stop [CONTAINER ID or NAMES]
```

コンテナの削除

```
$ docker rm [CONTAINER ID or NAMES]
```

イメージの削除

```
$ docker rmi [IMAGE ID]
```

使用例 2

使用例 2

```
# 停止する
```

```
#####
```

```
# 起動中であることを確認
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
d7583a4844bf	ruby:2.5.0	"/bin/bash"	26 minutes ago	Up

```
# 停止する
```

```
$ docker stop ruby25
```

```
ruby25
```

```
# 停止したか確認
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
--------------	-------	---------	---------	--------

コンテナの削除

#####

停止中のコンテナを確認

\$ docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
d7583a4844bf	ruby:2.5.0	"/bin/bash"	32 minutes ago	Exited

コンテナの削除

\$ docker rm ruby25

ruby25

削除されたか確認

\$ docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
--------------	-------	---------	---------	--------

イメージの削除

#####

イメージの確認

\$ docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ruby	2.5.0	bae0455cb2b9	2 weeks ago	80MB

イメージの削除

\$ docker rmi bae0455cb2b9

Untagged: ruby:2.5.0

Untagged: ruby@sha256:ed5fc221d5d03d89e1f8c1f7780b98bc708e68b4d8dba73594d017e999150

Deleted: sha256:bae0455cb2b9010f134a2da3a1fba9d217506beec2d41950d151e12a3112c418

Deleted: sha256:95ddde1e68dbec36e6500587eb95075e88ef698b05b3efd151da57958954ecd7

Deleted: sha256:fc853360468acdce21e9b74e6fb3be6a7007ed6cdccfa3cd174169693754486

Deleted: sha256:af2d359f555f399b7d0ab4ad59d904b1a89d0033e8a03a68c070359ab3b25f45

Deleted: sha256:693965653c171e528e1d10b4feb84337a990c885ebaab89e7b87b2b2414cceab

Deleted: sha256:e7b2522baa0c6b3d21f3bfde6a15e64f4b9a005e0662773a9c6fb483deacd57f

Deleted: sha256:8d8b933a99c7c507cc21924b9e71ba9a4c9bee836ddfbf5a92678259ce0cc881

Deleted: sha256:fb65b9418cd06c4efac7ccb9740ef4bb006cc34b79819bb6e841a2c6d60d1a26

Deleted: sha256:8568818b1f7f534832b393c531edfcb4a30e7eb40b573e68fdea90358987231f

イメージが削除されたか確認

\$ docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
------------	-----	----------	---------	------

イメージの検索

```
$ docker search [キーワード]
```

```
$ docker search ruby
```

NAME	DESCRIPTION	STARS
ruby	Ruby is a dynamic, reflective, object-orient...	1288
redmine	Redmine is a flexible project management web...	528
jruby	JRuby (http://www.jruby.org) is an implement...	62
phusion/passenger-ruby22	Base image for Ruby, Python, Node.js and Met...	39
circleci/ruby	Ruby is a dynamic, reflective, object-orient...	25
heroku/ruby	Docker Image for Heroku Ruby	21
starefossen/ruby-node	Docker Image with Ruby and Node.js installed	15
frolvlad/alpine-ruby	The smallest Docker image with Ruby (17MB)	13
bitnami/ruby	Bitnami Ruby Docker Image	11
iron/ruby	Tiny Ruby image.	7
centos/ruby-23-centos7	Platform for building and running Ruby 2.3 a...	4
drecom/centos-ruby	centos ruby	4
openshift/ruby-20-centos7	DEPRECATED: A Centos7 based Ruby v2.0 image ...	3
agideo/ruby	ruby	2
instructure/ruby	Instructure base ruby image	2
centos/ruby-22-centos7	Platform for building and running Ruby 2.2 a...	1
centos/ruby-24-centos7	Platform for building and running Ruby 2.4 a...	1
instructure/ruby-passenger	Instructure ruby passenger images	1
appsvc/ruby	ruby	0
appsvctest/ruby	ruby build	0
rylnd/ruby	Fork of iron/ruby. Adds libffi and libcurl, ...	0
aptible/ruby	https://github.com/aptible/docker-ruby	0
articulate/articulate-ruby	Base images for articulate ruby projects.	0
liaisonintl/ruby	Base image of Ruby with commonly used package	0
bedrockibm/ruby-runtime	Prepackaged ruby runtime.	0

※ なるべく OFFICIAL のイメージを使用すると安心

Dockerfileを使ってみる

Nginxを動かしてブラウザからアクセスしてみる。

Dockerfileを作成する。

```
$ vi Dockerfile
```

Dockerfileに以下のコードを記述する。

```
Dockerfile

FROM centos:centos7.4.1708

RUN rpm --import http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-7
RUN rpm --import http://dl.fedoraproject.org/pub/epel/RPM-GPG-KEY-EPEL-7

RUN yum update -y \
&& yum install -y epel-release \
&& yum install -y nginx \
&& yum clean all

# Change default locale to ja-JP.UTF-8
ENV LANG=ja_JP.UTF-8
RUN localedef -f UTF-8 -i ja_JP ja_JP.UTF-8

RUN \cp -p /usr/share/zoneinfo/Japan /etc/localtime \
&& echo 'ZONE="Asia/Tokyo"' > /etc/sysconfig/clock

RUN systemctl enable nginx

# Expose ports.
EXPOSE 80
```

よく使う命令文

命令	使い方
FROM	使用するイメージを指定 例： FROM centos:centos7.4.1708

命令	使い方
RUN	build時に実行するコマンドを記述 例: RUN yum update -y
ENV	環境変数を指定
EXPOSE	外部に解放するポートを指定する 例: EXPOSE 80
ADD	ホスト側にあるファイルをイメージにコピーする 例: ADD ./Gemfile /myapp/Gemfile

ビルドする(イメージを作成する)

```
# イメージをビルドする(イメージを作成する)
$ docker build -f ./Dockerfile -t centos:nginx-test --no-cache .
```

よく使うオプション

オプション	内容
-f	Dockerfileのファイル指定
-t	repositoryとtagを指定
--no-cache=true	ビルド時にキャッシュを利用しない

コンテナを起動する

```
# ビルドしたイメージを使用してコンテナを起動する。
$ docker run -itd --privileged --name nginx-test -p 8080:80 centos:nginx-test /sbin
```

<http://localhost:8080> にアクセスしてみる。

```
# 起動したコンテナにログインする
$ docker exec -it nginx-test /bin/bash
```

docker-composeについて

docker compose がなんたるかについては [Docker-docs-ja](#) を確認ください。

また、下記のサンプルは [クイックスタート・ガイド：Docker Compose と Rails](#) を参考にして、構成を作りました。読み比べてみるとより理解が深まるかと思います。

大きな変更点としては、

1. DBをmysqlに変更
2. DBの永続化（buildし直してもデータが残るように変更）

Railsの環境構築

ファイル構成

Dockerfile

```
# Ruby 2.5.0 がインストールされたイメージを使用する
FROM ruby:2.5.0

# 必要なライブラリーのインストール
RUN apt-get update -qq && apt-get install -y build-essential mysql-client nodejs l

# Railsのインストール作業
ARG APP_NAME=/myapp/

RUN mkdir $APP_NAME
WORKDIR $APP_NAME
ADD Gemfile $APP_NAME
ADD Gemfile.lock $APP_NAME
RUN bundle install

ADD . $APP_NAME
```

```
# 文字コード指定
```

```
ENV LANG ja_JP.UTF-8
ENV LC_ALL ja_JP.UTF-8
ENV LC_CTYPE ja_JP.UTF-8
ENV LANGUAGE ja_JP.UTF-8
RUN localedef -f UTF-8 -i ja_JP ja_JP.UTF-8
RUN update-locale LANG=ja_JP.UTF-8
```

```
# タイムゾーン指定
```

```
ENV TZ Asia/Tokyo
#RUN echo "Asia/Tokyo" > /etc/timezone \
#&& dpkg-reconfigure -f noninteractive tzdata
```

docker-compose.yml

```
version: '3'
services:
  db:
    image: mysql:5.7.21
    command: mysqld --character-set-server=utf8 --collation-server=utf8_unicode_ci
    ports:
      - "3306:3306"
    volumes:
      - ./tmp/db:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=password
      - "TZ=Japan"

  web:
    build: .
    command: bundle exec rails s -p 3000 -b '0.0.0.0'
    volumes:
      - ./myapp
    ports:
      - "3000:3000"
    depends_on:
      - db
```

Gemfile

```
# rails new を実行するときに書き換えるのでrailsだけ記述しておく
source 'https://rubygems.org'
gem 'rails', '~> 5.1.5'
```

Gemfile.lock

```
# からのGemfile.lockを作成
$ touch Gemfile.lock
```

Rails プロジェクト作成

```
$ docker-compose run --rm web rails new . --force -d mysql
```

config/database.yml の書き換え

```
$ vi config/database.yml
```

config/database.yml

```
default: &default
  adapter: mysql2
  encoding: utf8
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
  username: root
  password: password
  host: db

development:
  <<: *default
  database: myapp_development

test:
  <<: *default
  database: myapp_test

production:
  <<: *default
```

```
database: myapp_production
username: myapp
password: <%= ENV['MYAPP_DATABASE_PASSWORD'] %>
```

サーバを立ち上げてみる

```
$ docker-compose build
$ docker-compose run --rm web rails db:create
$ docker-compose up
```

<http://localhost:3000> にアクセスしてみる。

Railsでscaffold してみる

```
$ docker-compose run --rm web rails g scaffold tasks name:string description:text :
$ docker-compose run --rm web rails db:migrate
```

<http://localhost:3000/tasks> にアクセスしてみる。

以下のコマンドが実行できるか試してみる

```
$ docker-compose run --rm web rails c
$ docker-compose run --rm web rails routes
$ docker-compose run --rm web rails test
```

注意点

1. Gemfileを書き換えるたびに `docker-compose build` をやり直す必要がある。
2. `xxxx_web_1 exited with code 1` と出る場合は、プロセスが停止しているのに `tmp/pids/server.pid` が残っていることが原因なので削除する

```
$ rm -f tmp/pids/server.pid
```

停止・削除

```
# 停止
```

```
$ docker-compose stop
```

```
# 停止中のコンテナを削除
```

```
$ docker-compose rm
```

```
# コンテナの停止・削除
```

```
$ docker-compose down
```

```
# --rmi all をつけるとコンテナの停止・削除と一緒にイメージにも削除
```

```
$ docker-compose down --rmi all
```

 編集リクエスト

 ストック

 いいね 0



TAKAYUKI YOSHIOKA @yoshiokaCB

フォロー

関連記事 Recommended by



slack でのいろんな表記方法

by sheercat



SQL素人でも分かるテーブル結合(inner joinとouter join)

by naoki_mochizuki



[Linux]ファイルの圧縮、解凍方法

by supersaiakujin



Markdown記法 サンプル集

by thnrr



パイオニア企業がDX領域のエンジニアを育てる理由

PR パソナテック



水俣病の被害拡大はなぜ止められなかったのか

PR 國學院大學