鹿児島.mk #29

Dockerを体験しよう

福盛　照太郎（ふくもり しょうたろう）てるみっと

やってきたこと

- 2016.04～2020.03：市役所職員（農業技術職員）
- 2020.04～2020.08：SESとして勤務
- 2020.11～2022.11：某家具メーカーに勤務

好きなこと

- 釣り🎣
- ドライブ🚙🏍
- 食べ歩き🍜
- お酒🍻

鶴ケ野　宏祐(つるがの　こうすけ)　つるP

やってきたこと

- 2022年2月まで、理学療法士として約 12年間勤務
- 2022年3月～11月まで、受託・自社開発企業で勤務
- 2022年11月中旬からGMOペパボ株式会社に入社

好きなこと

- アニメ、犬、ゲーム、サーフィン
- お酒：飲みすぎるので注意してます

今日の目標！

- Dockerとはどんな技術かを知ってもらう

- Docker composeを使って、RailsとMySQLを動作させる

Play with Dockerではコピペの仕方が違う！！

Windows = コピー：Ctrl + Insert

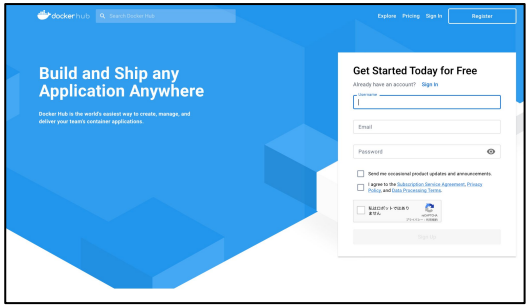
ペースト：Ctrl + Insert

Mac = コピー：command + c

ペースト：command + v

出典元：https://zenn.dev/kazurof/articles/a2de4a9fcf5dc1

Docker Hubアカウント作成



1. ブラウザで Docker Hub のサインアップ・ページ を開きます。
    a. サインアップ・ページのフォームを入力します。
        i. 登録にはユーザ名、パスワード、メールアドレスが必要です。
    b. Signup（サインアップ）を押します。
        i. ブラウザには Docker Hub のウェルカム・ページが表示されます。
2. メールの受信箱を確認します。
    a. メールが届いていなければ、迷惑メール用のフォルダに入っていないか確認するか、メールが到着するまでお待ちください。
3. メールの本文にあるボタンをクリックし、メールアドレスの確認を済ませます。

出典元：https://docs.docker.jp/mac/step_five.html

※今回の学習会での注意点

- Dockerについて学ぶ際に使うツールとして「Play With Docker」を使用
- 理由としてDockerを利用する環境構築をおこなう際、Mac・Windowsでの構築手順に違いがあり、今回の体験会では時間の都合から難しいと判断したため
- Dockerの環境構築がお済みの方に関しては、ご自分の環境で実施していただいて問題ありません

Play With Docker（URL）：https://labs.play-with-docker.com/

Dockerとは

- Dockerは、アプリケーションの開発、出荷、実行するためのオープンプラットファームです。（以下略）
  - 出典元：https://docs.docker.com/get-started/overview
- Docker社が開発したコンテナを用いた仮想環境を作成・配布・実行するためのプラットフォーム
  - 出典元：https://tech-blog.rakus.co.jp/entry/20221007/docker

Docker：コンテナとは

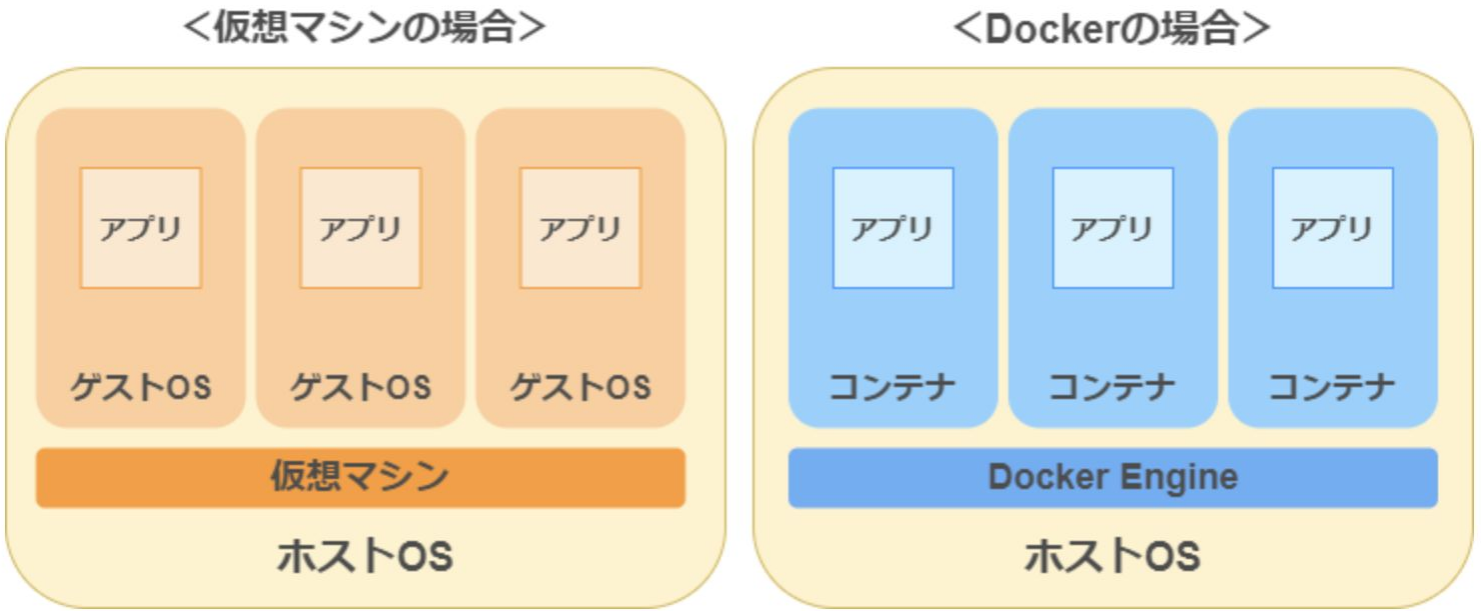
- コンテナというフレーズから列車や船に積まれる容器を思い浮かべるかと思います。
- そこで、物流における安全かつ効率的に輸送するコンテナからヒントを得て「プロセスだけを分離して動かす技術」として誕生したのがDockerのコンテナと言われています。
    - 出典元：https://www.winserver.ne.jp/column/docker-container/

仮想環境とは

- 仮想環境とは、PCやサーバーなどの一つのハードウェアの中で、仮想的な環境を構築したものをいいます。CPUやメモリなどの物理的な数しかありませんが、論理的にそれらを割り当てて、環境をしたり統合したりする技術です。
    - 例としてWindows環境の中で、仮想的に作られたLinux OS環境など...（以下省略）
- 一般的に仮想環境の土台となるOSを「ホストOS」、仮想環境上のOSを「ゲストOS」と呼びます
    - 出典元：https://bcblog.sios.jp/what-is-virtualenvironment-vmware/

仮想：事実でないことを仮にそう考えること。
　　仮定しての想像。

出典元：https://tech-blog.rakus.co.jp/entry/20221007/docker



仮想マシンとDockerの違い

Dockerのメリット

- 誰でも開発環境を揃えることができる
  - Docker（コンテナ技術）を使うことで、ローカル PCの影響を受けずに開発における言語のバージョンなどによるバグや開発環境を整える時間を少なくすることができる。
  - その他：仮想環境よりも軽量化・高速化、インフラコストの削減等がある。
  - しかし、状況によってはDockerの恩恵をあまり感じることができないこともある？

Dockerのデメリット

- 習得に時間がかかる
  - Dockerfileの作成や構築について理解する必要がある

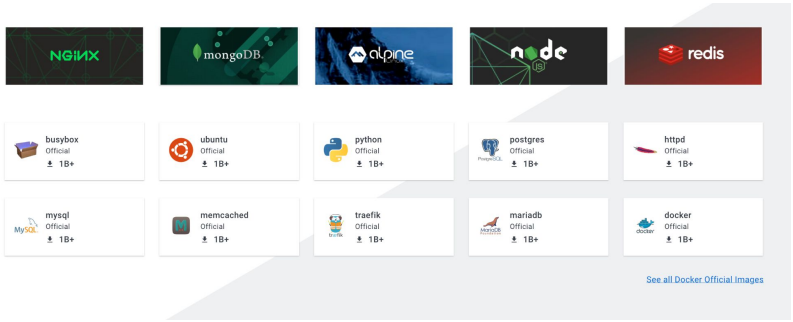
Dockerのメリット・デメリット：出典元

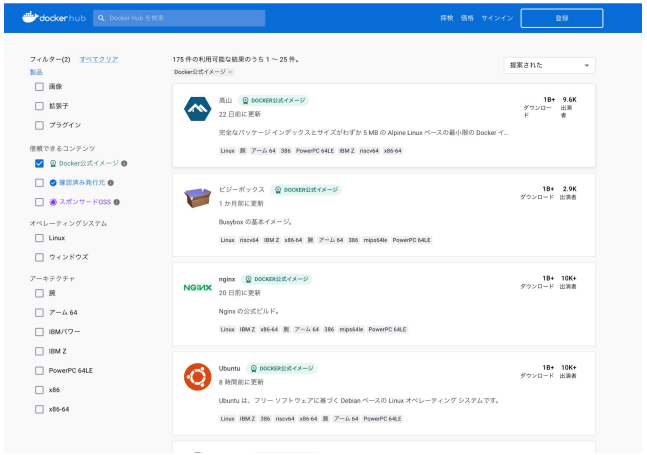
- メリット
  - https://www.geekly.co.jp/column/cat-technology/1902_047/
- デメリット
  - https://www.fenet.jp/infla/column/server/docker%E3%81%A8%E3%81%AF%E3%81%AA%E3%82%93%E3%81%AA%E3%81%AE%E3%81%8B%EF%BC%9F%E5%88%A9%E7%94%A8%E3%81%99%E3%82%8B%E3%83%A1%E3%83%AA%E3%83%83%E3%83%886%E3%81%A4%E3%82%84%E4%BD%BF%E3%81%84%E6%96%B9/

Dockerfileとは

- **一言で説明すると、Dockerイメージの設計図！！**
  - Dockerfileとは、新規にDockerイメージを作成するための手順を記したテキストファイルです。
  - Dockerfileというテキストファイルには、どのようなコンテナにするかをコマンドを書いて表現します。
- 出典元：https://aiacademy.jp/media/?p=1386

Dockerイメージは取得できる！！

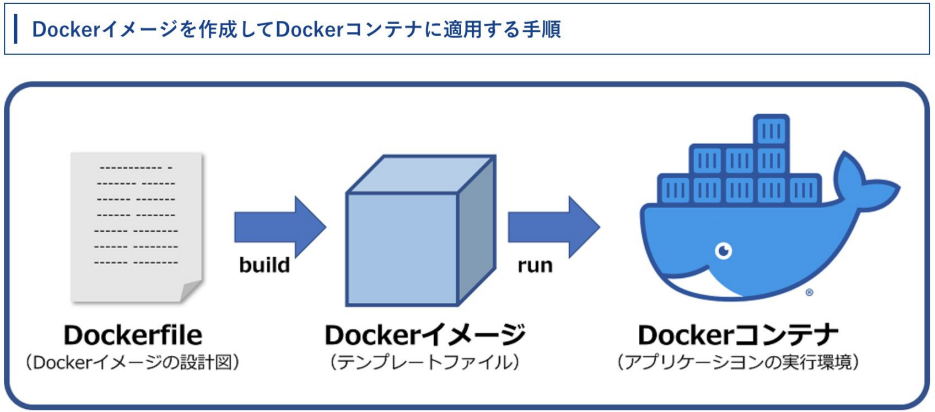




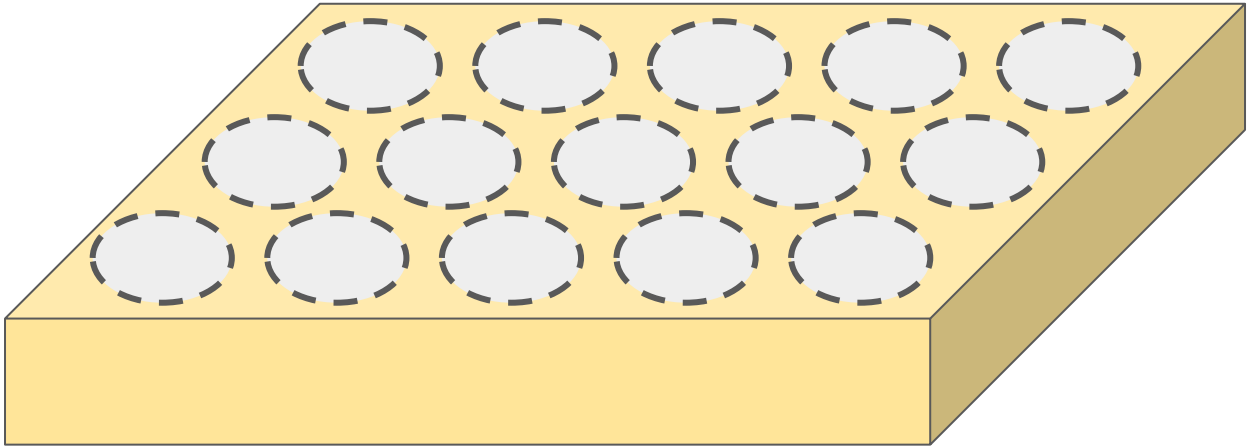
- Dockerイメージについては、Docker Hubから取得することができる。
  - Dockerイメージが公式・非公式に公開されている
  - 出典元：https://hub.docker.com/
  - 出典元：https://hub.docker.com/search?image_filter=official&q=&type=image

Dockerイメージとは

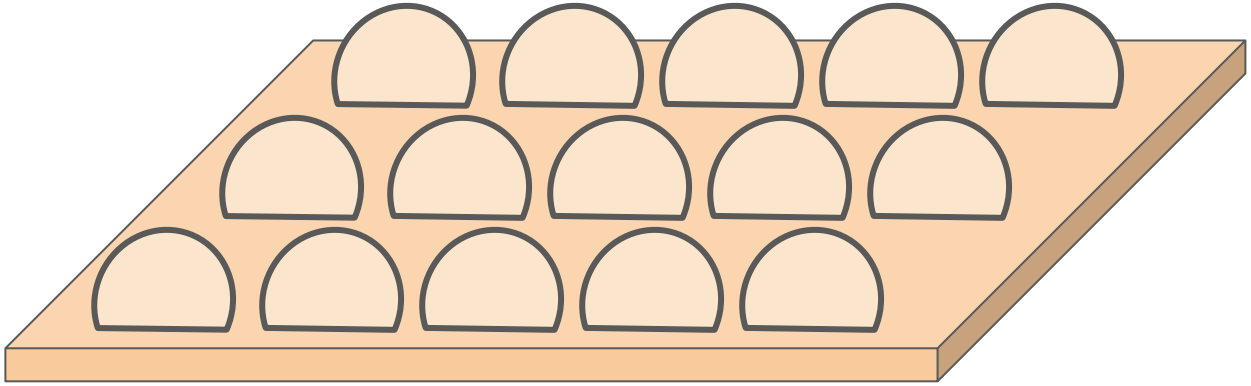
- Docker イメージ（image）とは、Docker コンテナの実行に必要な概念としてのパッケージ（ファイルやメタ情報の集合体）です。
  - 少し想像しにくい....(T ^ T)
  - 出典元：https://qiita.com/zembutsu/items/24558f9d0d254e33088f
  - 出典元：https://www.kagoya.jp/howto/rentalserver/dockerimage/



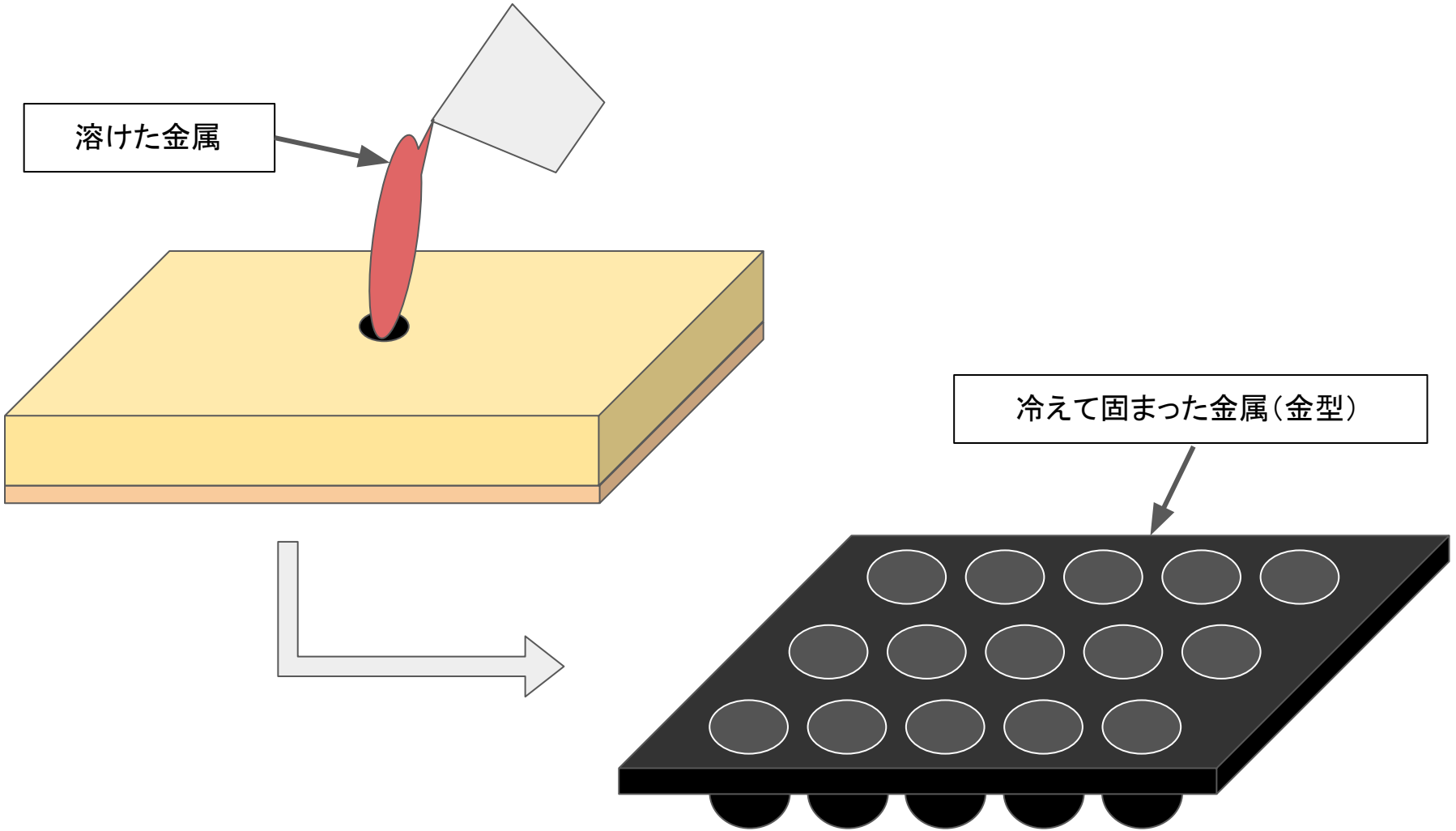
鋳造（ちゅうぞう）：溶かした金属を形成したい型に流し込み金型を生成する方法

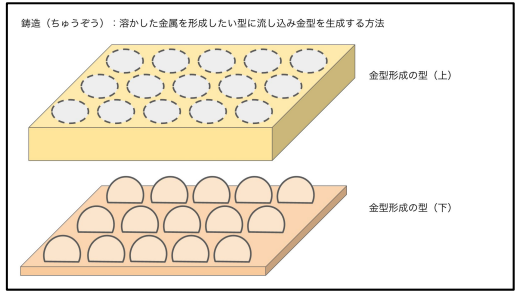


金型形成の型（上）



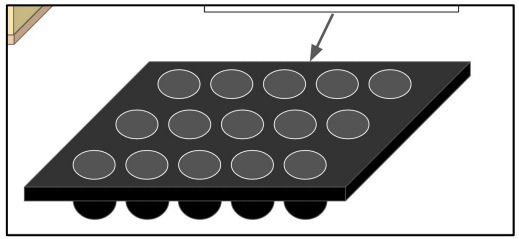
金型形成の型（下）


溶けた金属
冷えて固まった金属（金型）



＝

Dockerfile＝金型の設計書指示書



＝

Dockerイメージ＝金型の設計書を元に成型した型

＝

Dockerコンテナ＝成型された金型で作られたもの

Dockerコンテナを使ってRailsを立ち上げに必要な構成

1. Dockerfile
2. Gemfile
3. Gemfile.lock
4. docker-compose.yml

出典元：https://qiita.com/jibiking/items/fc7b0141af4b13a32ec3

Play With Docker上のターミナルで実施

```
[node1] (local) root@192.168.0.28 ~
$ mkdir docker-rails
[node1] (local) root@192.168.0.28 ~
$ ls
docker-rails
[node1] (local) root@192.168.0.28 ~
$ cd
.ssh/          docker-rails/
[node1] (local) root@192.168.0.28 ~
$ cd docker-rails/
[node1] (local) root@192.168.0.28 ~/docker-rails
$ touch Dokerfile
[node1] (local) root@192.168.0.28 ~/docker-rails
$ touch Gemfile
[node1] (local) root@192.168.0.28 ~/docker-rails
$ touch Gemfile.lock
[node1] (local) root@192.168.0.28 ~/docker-rails
$ touch docker-compose.yml
[node1] (local) root@192.168.0.28 ~/docker-rails
$ ls
Dokerfile            Gemfile              Gemfile.lock         docker-compose.yml
[node1] (local) root@192.168.0.28 ~/docker-rails
$
```

ディレクトリ・フォルダの作成

- ディレクトリ、フォルダの作成

  $ mkdir docker-rails

  $ cd docker-rails

作成したディレクトリ・フォルダないに空のファイルを作成

$ touch Dockerfile

$ touch Gemfile

$ touch Gemfile.lock

$ touch docker-compose.yml

Dockerfile記載例

※出典元を参考に一部変更箇所があります。ご容赦ください。

```
FROM ruby:3.2.1
RUN apt-get update -qq && apt-get install -y nodejs
RUN mkdir /myapp
WORKDIR /myapp
ADD Gemfile /myapp/Gemfile
ADD Gemfile.lock /myapp/Gemfile.lock
RUN bundle install
ADD . /myapp
```

Dockerfileの記述例（Railsを例とする）

```
FROM ruby:3.2.1
```

- ベースとなるイメージ（DockerHubで公開中の元イメージを指定）
  - rubyのバージョン指定

出典元：https://www.tohoho-web.com/docker/dockerfile.html

出典元：https://di-acc2.com/system/23152/#index_id3

出典元：https://kitsune.blog/dockerfile-summary

```
RUN apt-get update -qq && apt-get install -y nodejs
```

- docker build時に実行するコマンド（イメージをBuildする際に実行するコマンド）
  - aptコマンド = Linux（Debian系）のパッケージ管理ツールでのコマンド
  - apt-get update = インストール可能なパッケージの「一覧」を更新する
  - -qq = エラー以外は表示しない
  - && = AかつBの「かつ」の部分（AND）
  - apt-get install = 指定したパッケージをインストールする
  - -y = 全ての問い合わせに Yes で答える
  - nodejs = nodejsのパッケージ

```
RUN apt-get update -qq && apt-get install -y nodejs
```

出典元：https://webkaru.net/linux/apt-get-command/

出典元：http://www.ne.jp/asahi/it/life/it/linux/linux_command/linux_apt-get.html

出典元：https://qiita.com/uma002/items/28be70b674e24427491e

```
RUN mkdir /myapp
```

- docker build時に実行するコマンド（イメージをBuildする際に実行するコマンド）
  - myappの名前のディレクトリ・フォルダの作成

```
WORKDIR /myapp
```

- ワークディレクトリ
- （RUN、CMD、ENTRYPOINT、ADD、COPYの際の作業ディレクトリ）

```
ADD Gemfile /myapp/Gemfile
```

- ホストからコンテナへのファイルコピー
- （イメージにファイルを追加。圧縮ファイル指定時は圧縮まで実行される。）

```
ADD Gemfile.lock /myapp/Gemfile.lock
```

- ホストからコンテナへのファイルコピー
- （イメージにファイルを追加。圧縮ファイル指定時は圧縮まで実行される。）

```
RUN bundle install
```

- docker build時に実行するコマンド
- （イメージをBuildする際に実行するコマンド）
- bundleのインストール

```
ADD . /myapp
```

- ホストからコンテナへのファイルコピー
- （イメージにファイルを追加。圧縮ファイル指定時は圧縮まで実行される。）

Gemfile

```
source 'https://rubygems.org'
```

- ホスティングサーバーを指定

```
gem 'rails', '7.0.4.2'
```

- バージョンの指定もできる
- バージョンの指定をしない場合、最新のバージョンはインストールされる

Gemfile.lock

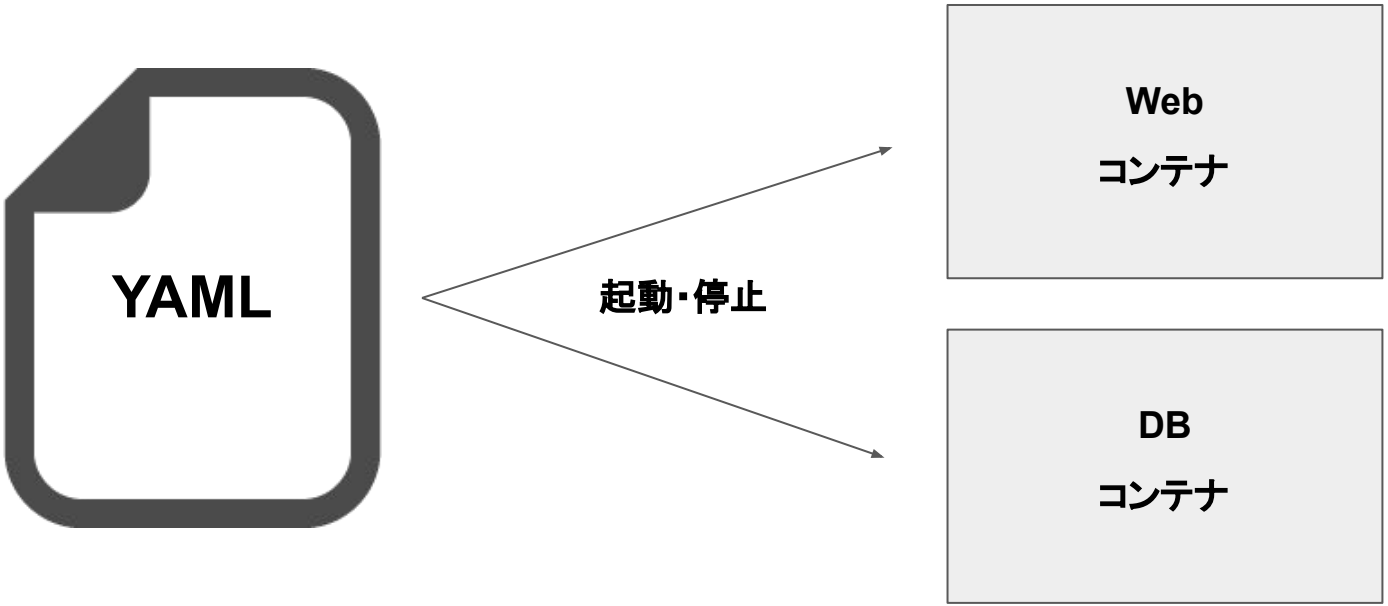
- 記載なしでOK

出典元
:https://nishinatoshiharu.com/difference-gemfile-gemfilelock/#:~:text=%E3%81%A7%E3%81%99%E3%81%AE%E3%81%A7%E3%80%81Gemfile%E3%81%AF%E3%80%8C%E3%82%A2%E3%83%97%E3%83%AA,%E3%81%8C%E8%A8%98%E8%BF%B0%E3%81%95%E3%82%8C%E3%81%A6%E3%81%84%E3%81%BE%E3%81%99%E3%80%82

docker compose

Docker Composeとは

アプリケーションの定義（YAMLファイルに記述された内容）を使って、簡単に複数のコンテナを立ち上げたり操作（起動・停止など）したりできるようになる。



DockerとDocker Composeの違い

- 同時に複数のコンテナを起動できる。
- docker-compose.ymlの内容を元に起動する事で、操作ミスによる失敗を減らすことができる。
- アプリケーションの定義時、Dockerがコンテナ間で内部通信するために必要な「ネットワーク」と、コンテナ間でデータを共有するために必要な「ボリューム」も定義できる。

＊ボリューム：Dockerコンテナで扱うデータを永続化する仕組み。ボリュームをホスト側と共有しなければコンテナが削除されると、データが消えてしまう。

Docker Compose を使わなかったら？

一つ一つを起動し、コマンドライン上でも設定を書く必要がある。

例:

DBの立ち上げ

```
$ docker run -ti
```

OSをDBへのリンクを指定

```
$ docker run -ti --rm --link berserk_brown:mongo centos:6
```

Docker Composeを使うには

- Docker Desktopがセットアップ済みの場合は、自動的にDocker Compose（docker-composeバイナリ）もセットアップされています。そのため、何か特別なセットアップを行う必要はありません。
- Linux 版 Docker もありますがここでは割愛します。

今回はDocker Desktopまたは、オンラインのDocker実行環境Play with Dockerを使います。

＊環境構築に関しては今回の体験会では割愛させていただきます。

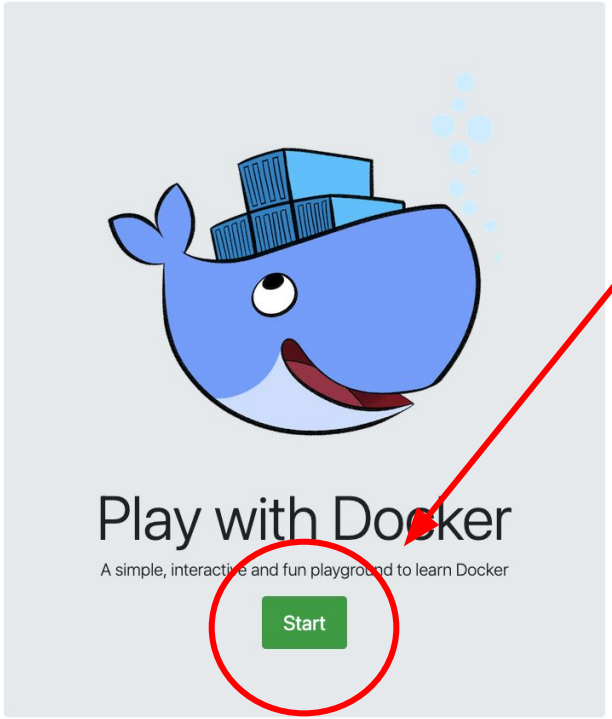
Docker Compose で WordPress を起動しよう

- 参考資料の内容をコピペして、実際に動くことを体験してみましょう！

Play with Docker の注意点

- PWD環境の制限時間は4時間です。4時間経過すると環境が削除されるので新たに作成しなおす必要があります。
- セキュリティは考慮されていないので重要なデータを置くことはできません。

手順1

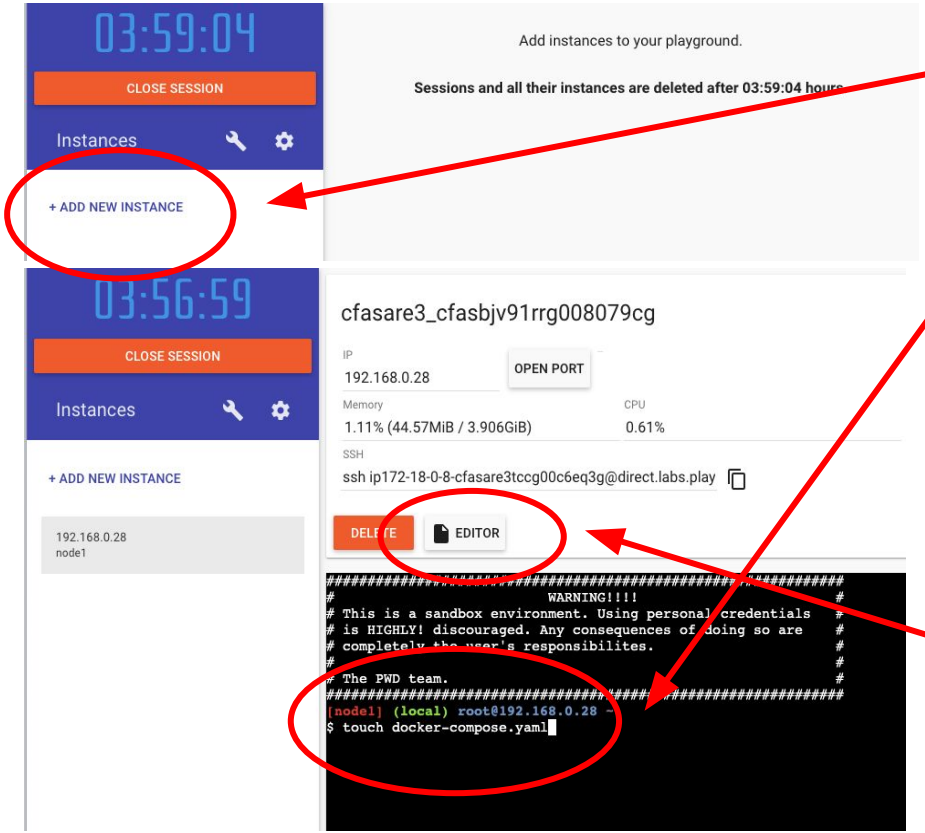


ログインボタンをクリック

（GitHubアカウントが必要）

https://labs.play-with-docker.com/

手順2



「ADD NEW INSTANCE」をクリック

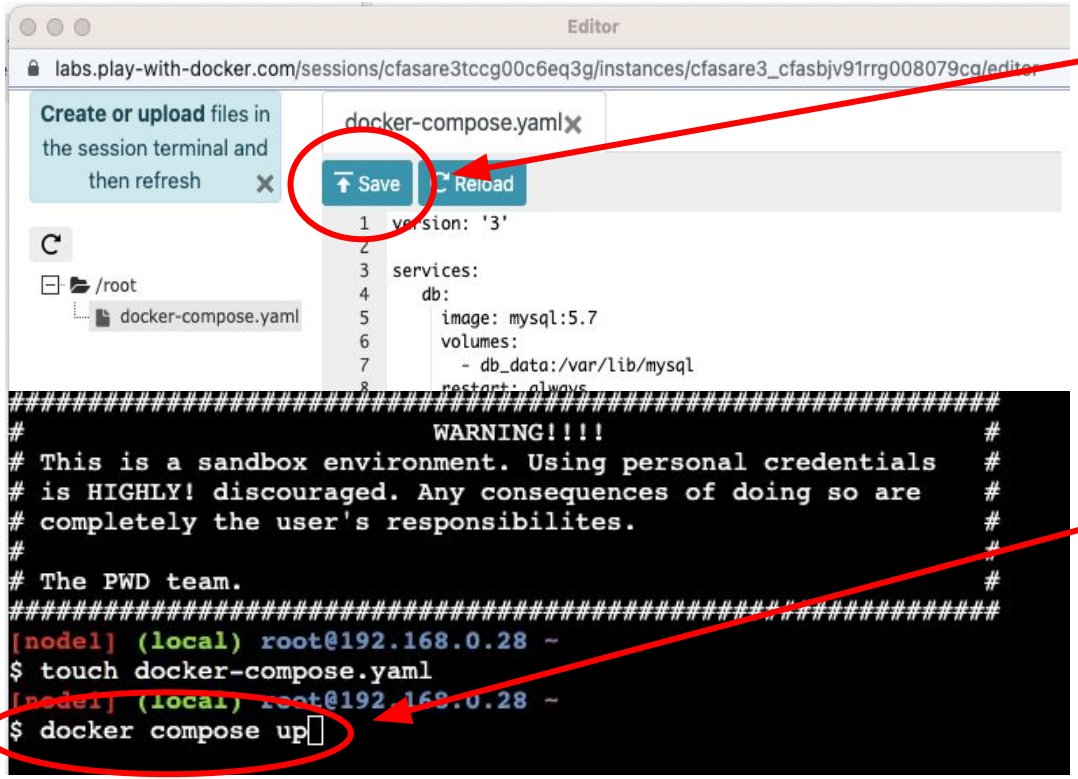
コンソール画面が出たら

「touch docker-compose.yml」

と入力

「EDITOR」をクリック

手順3



参考資料の内容を貼り付けて

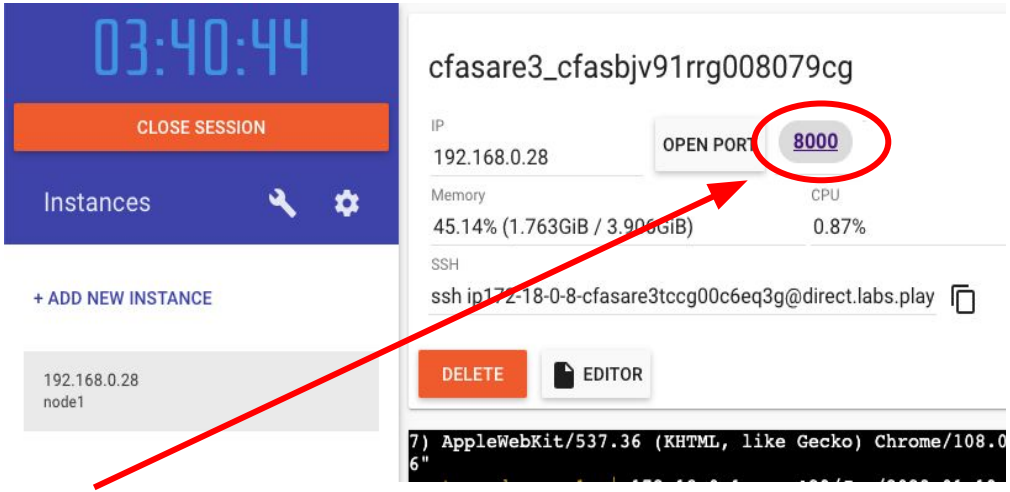
「Save」をクリック

Editorを閉じて、先ほどのコンソール画面で

「docker compose up」

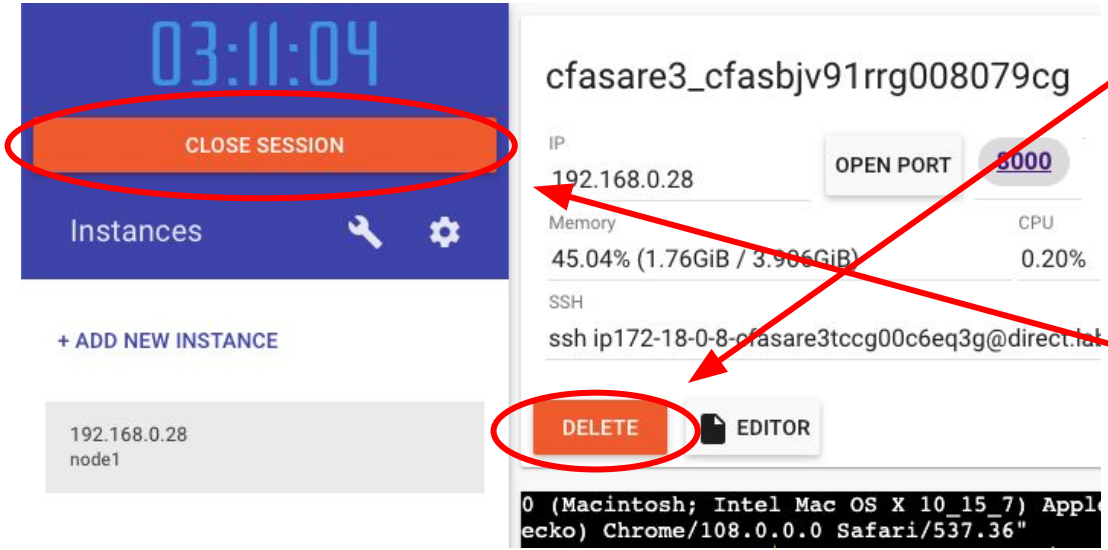
と入力

手順4



「8000」をクリックして、Wordpressが立ち上がり、登録できると成功！

終了する方法



「DELETE」をクリック終了

「CLOSE SESSION」をクリックして閉じる

docker-compose.ymlの内容1

```
version: '3'①

services:②
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
```

①version

docker-composeで使用するバージョンを定義しています。

現在公式サイトによると3系が最新バージョンとなっています。

②services

全体の構成をするコンテナの設定を記述する。
今回だとdb（コンテナ）、WordPress（コンテナ）

docker-compose.ymlの内容2

```
db:
    image: mysql:5.7①
    volumes:②
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWO…
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
…

volumes:
    db_data:
```

①image
DockerHubからイメージを利用する場合は imageと記述した後に、使用したいイメージを記述する。
自分で作成したDockerFileを元にイメージを作成する場合には「**build**」と書いて、相対パスで記述する。
例：
build: ./ruby　など

②volumes
volumesの作成場所を指定。自由に名前をつけることができ、最後の volumesで同じ名前を記述することで、毎回同じボリュームを使える。

docker-compose.ymlの内容3

```
db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always①
    environment:②
      MYSQL_ROOT_PASSWO…
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
…
```

①restart:always

コンテナが立ち上がった際に、明示的に stop がされない限り、終了ステータスに関係なく常に再起動が行われる。

なぜ必要？

修復や維持を考えるより新しく作り直す方が良い。

との思想で設計されている。

②environment
Compose 内の環境変数を定義する場所

docker-compose.ymlの内容4

```
wordpress:
  depends_on:①
    - db
  image: wordpress:latest
  ports:
    - "8000:80"
  restart: always
  environment:
    WORDPRESS_DB_HOST: db:3306
    WORDPRESS_DB_USER: wordpress
    WORDPRESS_DB_PASSWORD:w…
```

①depends_on
サービス間の依存関係を示す。起動順を表す。
今回だと、
起動：db→wordpress
停止：wordpress→db

あくまで起動の順番を制御してくれるが、dbが「準備」状態になるまで待つわけではない。

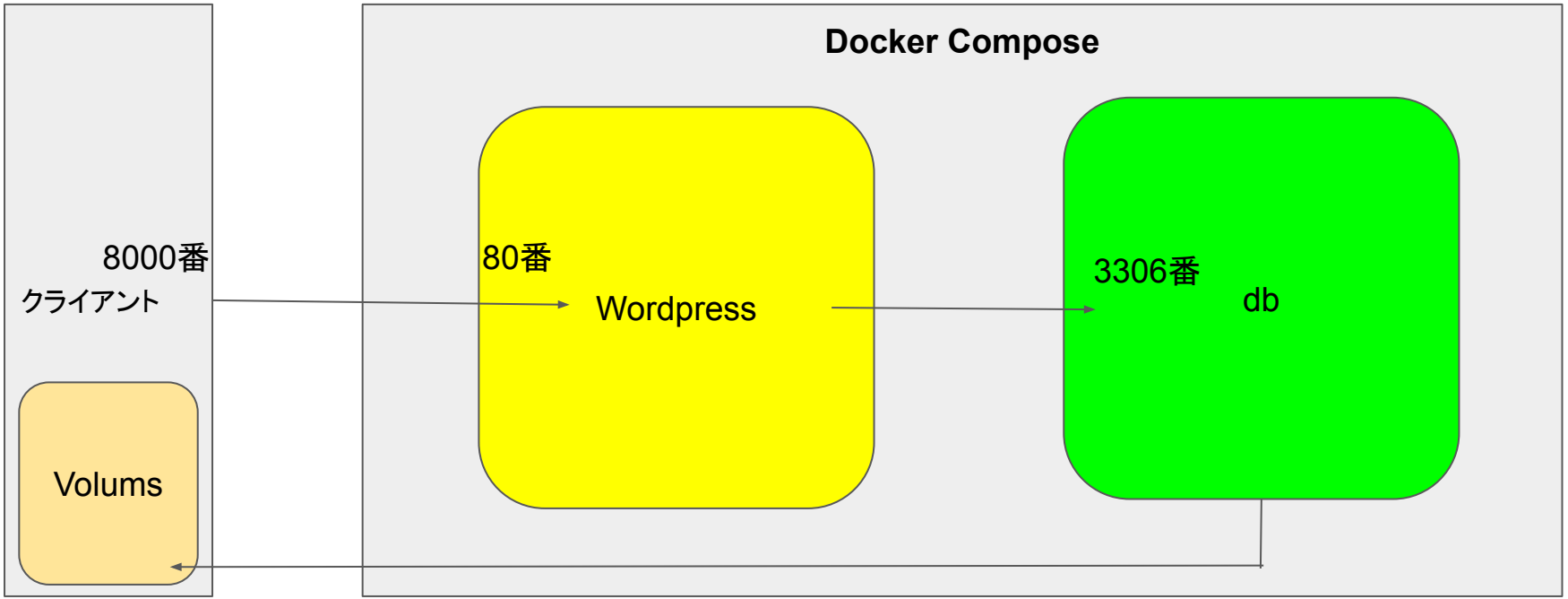
対策として接続が確立されるまで再接続を繰り返す方法や、コンテナの状態をかんしする方法がありますが、今回は割愛します。

docker-compose.ymlの内容5

```
wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:①
     - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306②
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD:w…
```

①ports
接続するポート番号を指定する。今回の場合クライアントの8000番ポートからwordpressコンテナの80番ポートに接続する。という内容になっています。
②では、wordpressコンテナからdbコンテナの3306番ポートに接続する。という内容になっています。
dbという名前を指定することで、本来であれば煩雑な作業をDockerがいい感じにしてくれている。

イメージ図



名前空間（namespace）

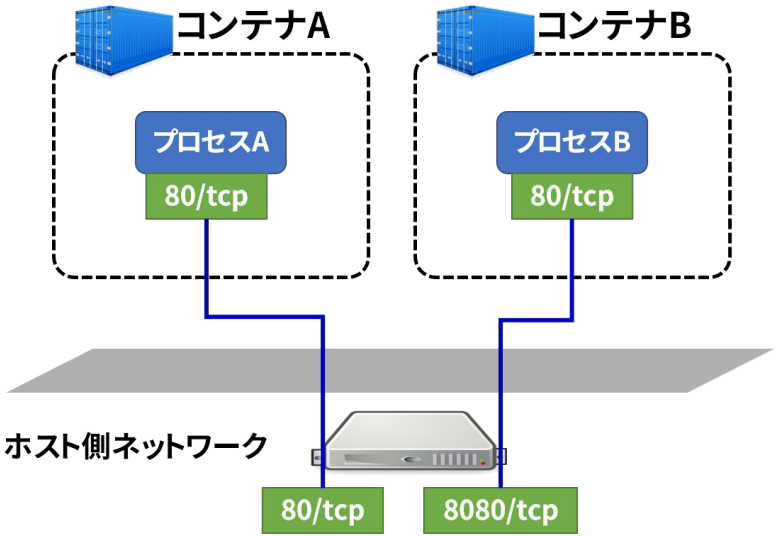


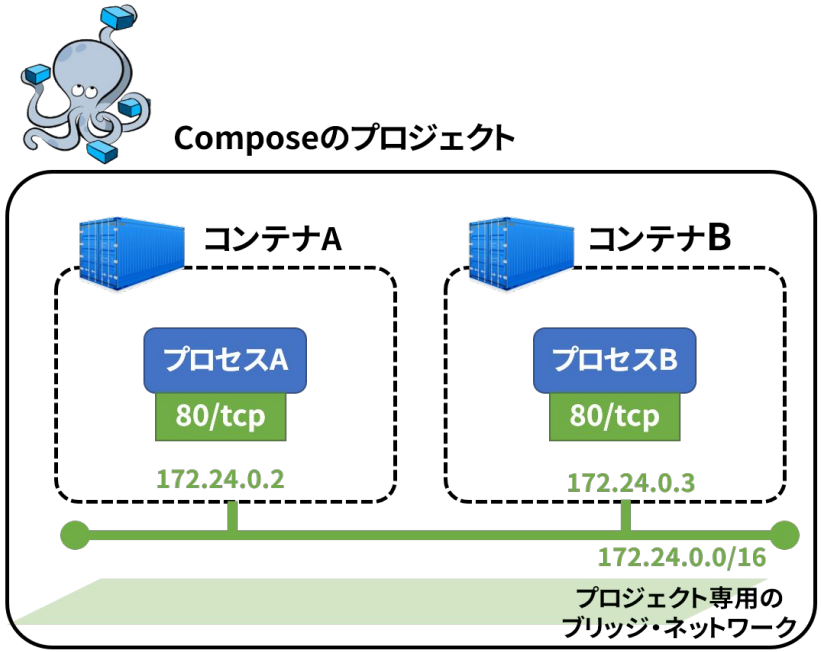
図1：Dockerコンテナはデフォルトで隔離された状態

通常のLinuxホスト上では、複数のプロセスがポートを重複して開くことはできません。

コンテナごとにネットワークも隔離されるため、コンテナ内で各々のプロセスがポートをリッスンしていたとしても、（ホスト側で使用するポート番号が重複しなければ）お互いに影響を与えず起動し続けられます。

参考URL：さくらのナレッジ
https://knowledge.sakura.ad.jp/23899/

コンテナ間通信



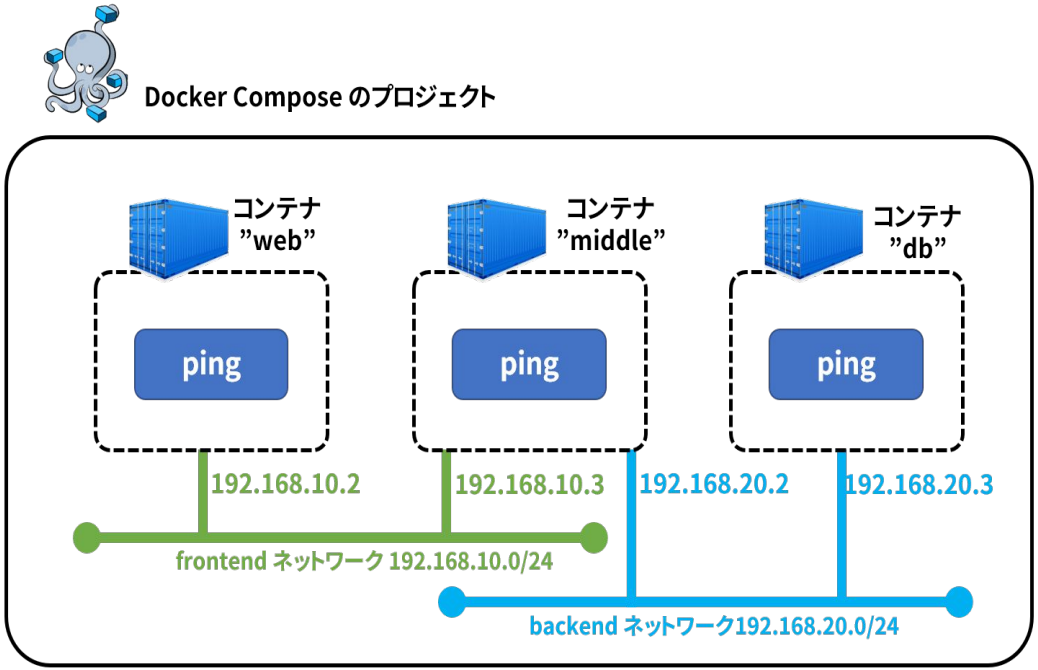
同じComposeのプロジェクト内に存在するコンテナ間（サービス）で通信可能なネットワークを作成します。

これはユーザ定義・ブリッジ・ネットワークと呼ばれ、コンテナ間での通信を行うだけでなく、インターネットなどホスト上にあるネットワークとの通信もやりとり（ブリッジ）します。

図2：Docker Composeはプロジェクト単位で内部通信用のネットワークを持つ

参考URL：さくらのナレッジ
https://knowledge.sakura.ad.jp/23899/

Docker Compose のネットワーク



図のような構成を組むことも可能。

middlewareを置いて、frontとbackendを直接通信させないことも可能。

参考URL：さくらのナレッジ
https://knowledge.sakura.ad.jp/26522/

Rails、MySQLをDocker Composeで起動しよう

準備するもの

Dockerfile：Rubyを動かす環境を記述

Gemfile：使うRailsのバージョンを指定

Gemfile.lock：空のファイル

docker-compose.yml：RailsとMySQLを使うために必要な設定を記述

それぞれの内容をコピペしてください。

次のコマンドを実行

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

docker-compose .yml内で記述したWebという名前のコンテナ内で

```
rails new . --force --no-deps --database=mysql
```

を実行するという意味

```
docker compose build
```

docker-compose.ymlに記述された内容でbuild（イメージを作成）

database.ymlに必要な内容を記述

この時hostをdbに変更する

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

先ほどと同様の手順でdb:createする

```
docker compose up
```

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

Docker Composeを終了さる

終了しないとどうなるのか？

- Docker を利用してコンテナのメモリ使用率が高まると OS がハングアップ（フリーズ）してしまう。
- 対応として docker-compose.yml や、Docker DeskTopなどでのメモリ使用量を制限するなどの対応があります。
- 基本的に使わないコンテナは停止させる。使わないイメージは削除する。

まとめ

Dockerには今回紹介した以外にも、コマンドや利用方法が沢山あります。

習得は大変ですが、使えるようになると大変メリットのある技術となっています。

今回の鹿児島.mkで少しでも理解する為の手助け、興味を持つきっかけになれたなら幸いです。

ご清聴ありがとうございました！

Docker Compose引用元

・さくらのナレッジ（Docker Compose入門（1）～（4））

https://knowledge.sakura.ad.jp/21387/

・Docker-docs-ja（クィックスタート: Compose と Rails）

https://docs.docker.jp/compose/rails.html#compose-rails