Docker Hands-on Guide

はじめに

想定する環境

- Windows + WSL(Ubuntu) + Docker
- Mac + Docker(Ubuntu) + Docker

  DockerDesktopを使用する場合は、Dockerコンテナを起動する必要はありません

  ※業務使用は条件次第で有償

やること

- Gitリポジトリのクローン
- Dockerfileからコンテナを作成
- Dockerコンテナの各種操作

やらないこと

- gitの細かい操作 (add, commit, pushなど)
- docker compose を使用した複数コンテナの起動と連携
- Flaskアプリケーションの構築方法

手を動かしましょう

Gitリポジトリのクローン

ここから先は WSL(Ubuntu)での操作となります。 (Macの場合はUbuntuコンテナ内)

WSL or MacDocker内 で任意の場所にリポジトリを作成してください。

```
$ git clone https://github.com/Daisuke-Ito-fwd/docker_sample.git

// WSLにGitがない場合
$ sudo apt update
$ sudo apt install git
$ git config user.name "Your Name"
$ git config user.email "sample@example.com"
```

ディレクトリ構成

```
Windows
└ 3001ポート – WSL Ubuntu
                └─ docker_sample/
                    ├─ app/ # Flaskアプリケーション
                    ├─ sh/ # シェルスクリプト
                    ├─ forMac/ # Mac用のDockerfile
                    ├─ Dockerfile.dev # 開発環境用
                    ├─ Dockerfile.prd # 本番環境用
                    ├─ nginx.conf # Nginx設定ファイル
                    ├─ ReadMe.md
                    └─ requirements.txt # Python依存関係
```

```
Mac
└ 3001ポート – Ubuntu
                └─ docker_sample/
                    ├─ app/
                    ├─ sh/
                    ├─ forMac/
                    ├─ Dockerfile.dev
                    ├─ Dockerfile.prd
                    ├─ nginx.conf
                    ├─ ReadMe.md
                    └─ requirements.txt
```

Dockerのインストール (便利シェル)

公式(https://arc.net/l/quote/dqmxodli)から提供されています。

1. スクリプトをダウンロードして実行

```
$ pwd
/mnt/docker_sample
$ cd sh
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
```

2. Dockerデーモンをバックグラウンドで起動

```
$ dockerd &
```

Dockerイメージをビルドする

1. ディレクトリ移動

```
$ cd ..
$ pwd
/mnt/docker_sample
```

2. イメージをビルド Dockerfile.devを使用します

```
# -t タグオプション -f ファイル指定
$ docker build -t docker_dev -f Dockerfile.dev .
```

3. 接続エラー時の対処

```
$ cat /etc/resolv.conf
$ echo "nameserver 8.8.8.8" > /etc/resolv.conf
```

コンテナの作成 & 起動 (開発環境)

1. コンテナを作成・起動

```
$ pwd
/mnt/docker_sample
$ docker run -dit --name docker_dev -p 3001:3001 -v src:/mnt/src docker_dev
```

2. ブラウザでアクセス
    - http://localhost:3001

dockerコマンドとdockerfileの解説

```
$ docker run -dit --name docker_dev -p 3001:3001 -v src:/mnt/src docker_dev
```

-d: コンテナをバックグラウンドで実行（Detachedモード）

-i: 標準入力をオープンにしておく（Interactiveモード）

-t: 仮想端末（TTY）を割り当てる

--name docker_dev: コンテナに「docker_dev」という名前を付ける

-p 3001:3001: ホストのポート3001とコンテナのポート3001をバインドする

-v src:/mnt/src: ホストの「src」ディレクトリ（またはDockerボリューム）をコンテナ内の「/mnt/src」にマウントする

docker_dev: 使用するイメージの名前

➡️ ポートやマウントが少ないうちは問題にならないが、増えてくるとコマンドが長くなってしまう

docker compose

1. 複数コンテナの管理が煩雑

複数のコンテナを手動で起動・停止する場合、個別にdocker runコマンドを実行する必要があり、依存関係や順序の管理が難しい。
➡️ docker-compose.ymlファイルに複数のコンテナ設定を記述し、一括で管理・操作が可能。

2. 長いコマンドの記述

docker runコマンドでポートマッピング、ボリュームマウント、環境変数などを指定すると、コマンドが非常に長くなってしまう。
➡️ Docker Composeでは設定をYAMLファイルに記述するため、コマンドが簡潔。

3. 環境の再現性が低い

手動でコンテナを起動する場合、設定ミスや環境の違いにより、同じ環境を再現するのが難しい。
➡️ Docker Composeでは、設定ファイルを共有することで、チーム全体で同じ環境を簡単に再現可能。

設定ファイルさえ整っていれば

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

これだけで環境の構築と起動が可能。

コンテナ操作

1. コンテナの状態を確認・起動

```
$ docker ps -a

# docker start <container_id or container_name>
$ docker start docker_dev
```

2. コンテナにアクセス

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

3. コンテナを停止

```
$ docker stop docker_dev
```

1. コンテナを削除

```
$ docker rm docker_dev
```

2. イメージを削除

```
$ docker rmi <image_id>
```

Dockerビルド(本番向け) Flask + Gunicorn + Nginx

1. Dockerfileをビルド

```
$ docker build -t docker_prod -f Dockerfile.prd .
```

2. コンテナを作成・起動

```
$ docker run -dit --name docker_prod -p 3001:3001 -v src:/mnt/src docker_prod
```

3. ブラウザでアクセス
   - http://localhost:3001
4. コンテナにアクセス

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

Docker関連 VSCodeの便利拡張機能

- Docker
- Remote - WSL
- Remote - Containers
- Remote - SSH

その他 VSCodeの便利拡張機能

- Git Graph
- GitHub Copilot
- GitHub Pull Requests and Issues
- Marp for VS Code

質疑応答

ありがとうございました