Makefileの使い方

タスク(擬似ターゲット，ダミーターゲット)の使い方

ある特定のファイルを作るためではなく，作業を行うコマンドとして利用したい場合に用いられる.

```
.PHONY: [実行したいタスク名]

[実行したいタスク名]:
    [そのタスクを行うためのコマンド行]
```

.PHONYは，タスクターゲットを宣言するためのターゲットです．また，[そのタスクを行うためのコマンド行]の手前にもタブ1文字を入れることに注意.

例) 同じフォルダの"requirements.txt"で指定されているライブラリをpipで一括でインストールする場合は、以下のようにMakefileに書き込む。

```
.PHONY: dev
dev:
    pip install -r requirements.txt
```

こうすると，「make dev」で記載されているpythonのライブラリが一括でインストールされる.

Dockerの使い方

Dockerの基本的な概念

Dockerは、Linuxのコンテナ技術を使ったもので、よく仮想マシンと比較される。VirtualBoxなどの仮想マシンでは、ホストマシン上でハイパーバイザを利用しゲストOSを動かし、その上でミドルウェアなどを動かす。それに対し、コンテナはホストマシンのカーネルを利用し、プロセスやユーザなどを隔離することで、あたかも別のマシンが動いているかのように動かすことができる。そのため、軽量で高速に起動、停止などが可能。

Dockerの利点

1. コード化されたファイルを共有することで、どこでも誰でも同じ環境が作れる。
2. 作成した環境を配布しやすい。
3. スクラップ＆ビルドが容易にできる。

例えば、開発環境(Windows上)では動いていたけどLinuxで動かなかった、といったケースも、開発工程からDockerを活用していくことで防ぎやすくなる。そして、開発工程の中で使っていた環境をそのまま本番環境に持っていくことも可能なため、環境差分が少なく、環境による問題を減らすことができる。

Dockerの基本操作

Dockerを操作する基本的なコマンドは以下の通り

```
docker コマンド オプション
```

1. コマンド
   「run」,「start」,「stop」などの命令のこと
2. オプション
   コンテナ名を指定する「--name」など

具体的な例としては、

```
docker run -dit --name my-apache-app -p 8080:80 -v "$PWD":usr/local/apache2/htdocs
httpd:2.4
```

基本的なDockerの使い方としては、

- 使用にあたって
    1. コンテナイメージの取得(docker pull)
    2. コンテナイメージからコンテナの作成(docker create)
    3. コンテナの開始(docker start)
- 終了にあたって
    1. コンテナの停止(docker stop)
    2. コンテナの削除(docker rm)
    3. コンテナイメージの削除

Docker file

Docker file とは

ベースとなるイメージと、そのイメージに対してどのような操作を行うかを示したファイル。このファイルに記述されたとおりにへんこうや　ファイルのコピーを行うことでイメージを作成する。

Docker file の実行

Docker fileは以下のようなコマンドで実行される。

```
docker build -t
$(BASE_IMAGE_NAME):$(TRAINING_PATTERN)_$(TRAINING_PROJECT)_$(IMAGE_VERSION) -f
$(DOCKERFILE) .
```

- 「-t」オプション
    - 「イメージ名:タグ名」を指定する
- 「-f」オプション
    - docker fileの名前

Docker file の特徴、書き方

基本的なコマンド

- FROM
    - ベースイメージの指定
- ENV
    - 環境変数の定義
- WORKDIR
    - 作業ディレクトリの指定
- RUN
    - イメージのビルド
- ADD
    - イメージにファイルやフォルダを追加する

具体例：

```
FROM python:3.8-buster

ENV PROJECT_DIR /mlflow/projects
ENV CODE_DIR /mlflow/projects/code
WORKDIR /${PROJECT_DIR}
ADD requirements.txt /${PROJECT_DIR}/

RUN apt-get -y update && \
    apt-get -y install apt-utils gcc curl && \
    pip install --no-cache-dir -r requirements.txt

WORKDIR /${CODE_DIR}
```

mlflowの使い方

MLflowとは、実験、再現性、デプロイメント、セントラルモデルレジストリなど、MLライフサイクルを管理するためのオープンソースのプラットフォームです。

mlflow のインストール方法

```
pip install mlflow
```

mlflow の実行方法

実装例

- start_run
  - runIDを発行する
- log_param
  - パラメータを記録する
- log_metric
  - メトリックを記録する（ステップごとに）
- log_artifact
  - 生成物を記録する
  - UIからディレクトリ・ファイルの中身を確認できる

```
import mlflow
mlflow.start_run()

# Log a parameter (key-value pair)
mlflow.log_param("param1", 5)

# Log a metric; metrics can be updated throughout the run
mlflow.log_metric("foo", 2, step=1)
mlflow.log_metric("foo", 4, step=2)
```

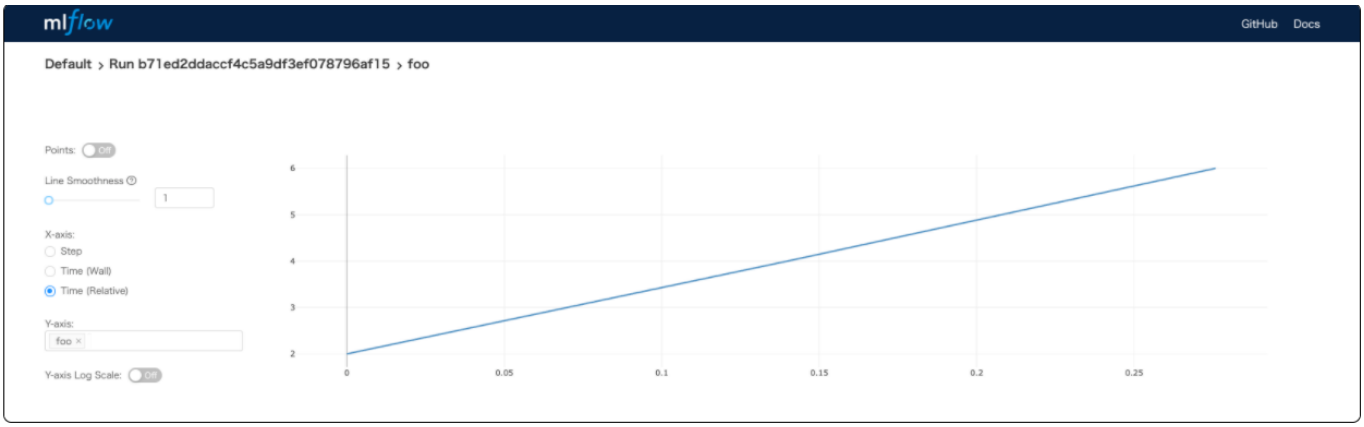
```
mlflow.log_metric("foo", 6, step=3)

# Log an artifact (output file)
with open("output.txt", "w") as f:
    f.write("Hello world!")
mlflow.log_artifact("output.txt")

mlflow.end_run()
```

UIで実験結果を確認することができる。

```
$ mlflow ui
# -> http://localhost:5000
```



MLprojectの書き方

上記のようなmlflowの一連の捜査をYAMLファイルとして保存し、実行できる。具体的には、以下のコマンドを実行することで同ディレクトリにある"MLproject"ファイルを実行する。（本来の使い方としては、github状のMLprojectも実行できる。）

```
mlflow run . --no-conda
```

MLprojectの基本的なコマンドは以下の通り。

- name
  - プロジェクトの名前(任意で良い)
- docker_env
  - 実行するdockerのコンテナを指定（指定されない場合はホスト側で実行される）
- enrty_points
  - 実行するコマンドや引数の情報が記載される

実行例) ここでは、同ディレクトリのiris_train.pyに "test_size"の引数と"target_iris"の引数をデフォルトでそれぞれ"0.3","virginica"を与えて実行する。

```
name: iris_binary

docker_env:
  image: shibui/ml-system-in-actions:training_pattern_iris_binary_0.0.1

entry_points:
  main:
    parameters:
      test_size: {type: float, default: 0.3}
      target_iris: {type: string, default: virginica}

    command: |
          python -m iris_train \
            --test_size {test_size} \
            --target_iris {target_iris}
```

実際にisis_train.pyの関数main()を参照すると、以下のような関数が用意されている。

```
def main():
    parser = ArgumentParser(description="Scikit-learn iris Example")
    parser.add_argument(
        "--test_size",
        type=float,
        default=0.3,
        help="test data rate",
    )
    parser.add_argument(
        "--target_iris",
        type=str,
        choices=["setosa", "versicolor", "virginica"],
        default="setosa",
        help="target iris",
    )
    args = parser.parse_args()
```

"parser"の"add_argument"の引数で指定された"test_size"と"target_iris"が入力されるようになっているのがわかる。

参考文献

- Makefileの書き方について

http://objectclub.jp/community/memorial/homepage3.nifty.com/masarl/article/gnu-make/rule.html

- Dockerの基本

https://knowledge.sakura.ad.jp/13265/

- mlflowについての解説

https://future-architect.github.io/articles/20200626/

https://blog.amedama.jp/entry/mlflow-projects#%E4%B8%8B%E6%BA%96%E5%82%99