

Dockerfileの自動修正に向けた Dockerイメージの継承と ビルドエラーの関係の調査

2022/7/19

坂本廉也

背景

▶ 背景

- ソフトウェア開発現場では，テストと本番の実行環境の違い
 - ex)パスが通っていない，ポートが違う，必要なパッケージがない
- 組織・企業のメンバーそれぞれの開発環境の違い
 - ex)メンバー内で1人だけOSが違う



開発現場ではDockerが広く利用されている

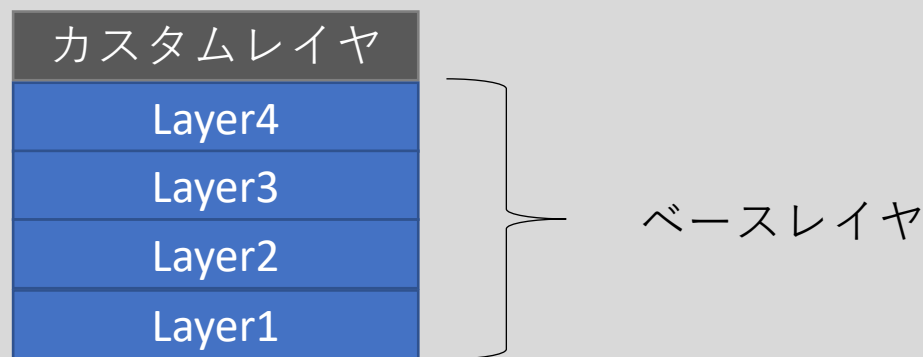
▶ Docker

- コンテナ型仮想化技術
 - コンテナ：開発のためのミドルウェア，OSがパッケージ化された仮想環境
- 複数のコンテナが軽量で高速に動作可能
- 組織・企業で同様の環境を作成可能

Dockerイメージとレイヤ構造

▶ Dockerイメージ

- コンテナの実行に必要なファイルの集合
- イメージを基にDockerコンテナを作成
- レイヤ構造としてイメージが生成



▶ レイヤ構造

- ベースイメージ：カスタムレイヤより下のイメージ
- カスタムレイヤ：ユーザが最終的に見えているイメージレイヤ

Dockerfile

- ▶ Dockerイメージを作成するための命令が記述されたファイル
 - 命令を順番に逐次実行し，1命令ごとに1つのレイヤを構成

```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

- ▶ FROM：新しいビルドステージを構築(イメージとタグを指定)
 - ベースイメージを指定，最小限のコンテナ作成も可能
- ▶ COPY：ファイル・ディレクトリをイメージ側にコピー
- ▶ RUN：ビルド時に実行するシェルコマンドを指定
- ▶ CMD：コンテナ内で実行するコマンドを指定

先行研究

▶ Dockerfileのビルドエラー自動修正の必要性

- Dockerfileのビルドは34%以上失敗している[1]
- ビルドエラーの修正は他の言語よりも時間がかかる[2]

▶ Shipwright[3]

- Dockerfileを自動修正するためのツール
- Dockerfile内の命令間の依存関係のみを考慮
- ビルドエラーが発生するDockerfileのうち約19%のみを自動修正可能

Shipwrightの問題点

Dockerfile内での依存関係のみを考慮して自動修正を行っており、親イメージとの継承関係が考慮されていない

[1]Jürgen Cito, Gerald Schermann, John Erik Wittern, Philipp Leitner, Sali Zumberi and Harald C. Gall. An Empirical Analysis of the Docker Container Ecosystem on GitHub. Proceedings of the IEEE/ACM 14th International Conference on Mining Software Repositories, pp. 323-333, 2017

[2]Yiwen Wu, Tao Wang, Yang Zhang and Huaimin Wang. An Empirical Study of Build Failures in the Docker Context. Proceedings of the 17th International Conference on Mining Software Repositories, pp. 76-80, 2020

イメージ間の依存関係：継承関係

▶ ベースイメージとの依存関係でビルド

- バージョン管理の欠落によりビルドエラーになる可能性[1]

A:latest

pip 20.3.4
Python2.7
Ubuntu:16.04



Dockerfile

```
FROM A:latest  
RUN pip install numpy==1.18.1
```

イメージ(A)で使用しているPython2.7とnumpyのバージョンの整合性がない



ビルドエラー

本研究の着眼点

▶ 着眼点

- ビルドの修正に時間がかかるのは、**Dockerfile**内の命令、イメージ間の依存関係が複雑だからではないか

▶ 動機

- イメージ間の依存関係を明らかにすることで、**Dockerfile**の迅速な修正に役立つ

目的とRQ

▶ 目的

- 依存関係（イメージ間, 命令間の互換性）とビルドエラーの関係を明らかにする

▶ RQ

- RQ1：ビルドエラーと継承の深さに関係があるか？
- RQ2：ビルドエラーとDockerfile内の命令の互換性に関係があるか？

RQ1

RQ1：ビルドエラーと継承の深さに関係はあるか？

▶ データセットの3217件のDockerfileを目視で確認

- FROM命令のみで構成されているDockerfileである45件を除外
- GitHubのdocker-library上にあるイメージ[3]をオフィシャルイメージとして定義

Dockerfileの分類	件数（割合）	ビルド失敗率(%)
オフィシャルイメージを継承しているDockerfile	1548（48.8%）	32.4
オフィシャルイメージ以外を継承しているDockerfile	1142（36.0%）	52.9
継承をしていないDockerfile	98（3.0%）	71.7
その他	384（11.9%）	54.4

その他：FROM命令がない、ARG命令でビルド時にイメージを指定、マルチステージビルド

RQ1

RQ1：ビルドエラーと継承の深さに関係はあるか？

▶ 継承の深さとビルド失敗率の調査

- 継承の深さ：幾つのイメージを継承しているか
- オフィシャルイメージ以外を継承しているDockerfileが対象(1142件)
- 目視で確認

継承の深さ	件数(件)	失敗率(%)
1	36	61.1
2	371	32.8
3	82	40.2
4	28	39.2
5	11	54.5

RQ1

RQ1：ビルドエラーと継承の深さに関係はあるか？

継承の深さが1のDockerfileのビルド失敗率が高い



しかし．．．

深さ2のDockerfileの失敗率は、オフィシャルイメージを
継承しているDockerfileとほぼ同じ

▶ 深さ2を超えると継承が深くなるほど失敗率が高くなる傾向

RQ2

RQ2：ビルドエラーとDockerfile内の命令の互換性に関するか？

- ▶ 命令間の依存関係に問題があるエラーのパターンを作成
 - Dockerfile内に存在する依存関係はイメージ間にも存在する可能性
- ▶ Shipwrightのエラー分類を基にビルドログを目視（103件）
 - Dockerfile内の命令間の依存関係が原因で発生しているエラーは22件
 - 目視の結果から14個のルールを作成



データセットにルールを適用

162件のDockerfileを抽出

ビルドに失敗したDockerfileの11.6%， Shipwrightで分類されたエラーの28%

RQ2(補足)

RQ2：ビルドエラーとDockerfile内の命令の互換性に関するか？

- ▶ ルールには警告メッセージを多く使用
- ▶ Shipwrightでは未分類だがビルドに失敗しているDockerfile
 - 分類されているものと同様に目視
 - 1件のみルールを作成

今後について

- ▶ 継承によって起こるビルドエラーの特徴を調査
 - 具体的な分析方法は未定