

Dockerfileにおけるビルドエラー 修正ルールの自動生成手法の構築

オープンソースソフトウェア工学研究室
坂本 廉也

Dockerの普及

| 社会の急速な情報化

- ソフトウェア開発競争が激化
- 開発現場はスピーディかつ柔軟な開発方式へ移行

スピーディかつ柔軟な開発方式と相性がいいDockerが普及

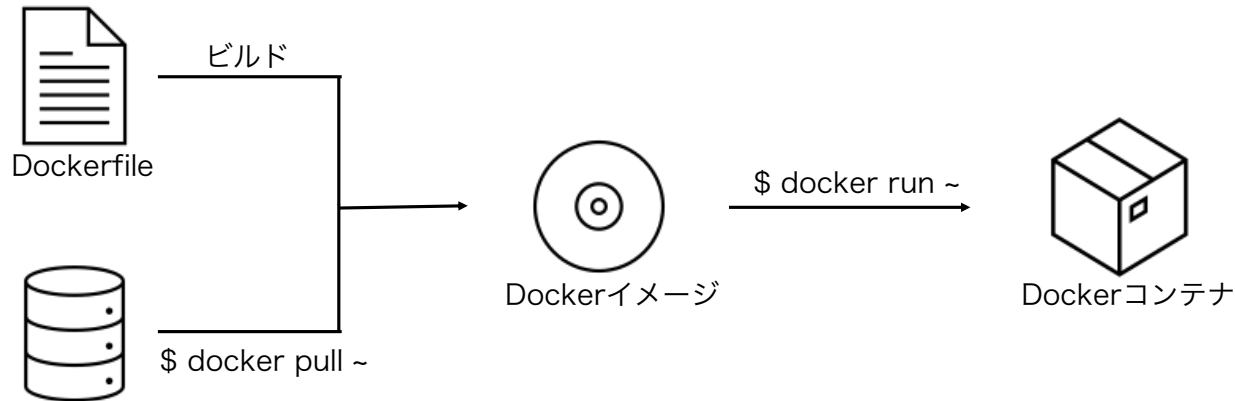
| Docker^[1]とは

- コンテナ型仮想環境を提供するためのソフトウェア
 - 独立した仮想アプリケーション環境（Dockerコンテナ）を動作
 - 軽量かつ高速に動作
- 多くの企業が利用^[2]

[1] <https://www.docker.com/>

[2] <https://portworx.com/blog/2017-container-adoption-survey/>

Dockerコンテナ作成までのながれ



| Dockerイメージ（以下、イメージ）

- Dockerコンテナの動作に必要な環境を含むイメージファイル
- **Dockerfileをビルド** or DockerHub^[3]から取得

| Dockerfile

- イメージに含める内容を指示するためのテキストファイル
- 様々な命令を組み合わせて、基本的に1行に1命令を記述

[3] <https://hub.docker.com/>

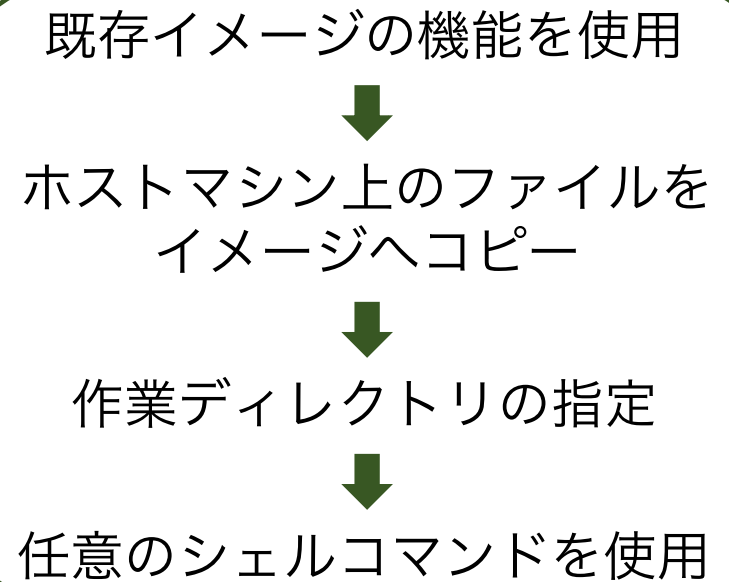
Dockerfile

```
FROM python:3-alpine

COPY [".", "/app/src/"]

WORKDIR /app/src/

RUN apk --update add ca-certificates \
    && update-ca-certificates \
    && rm -rf /var/cache/apk/*
```



| Dockerfileのビルド

- Dockerfile内の命令を上から順に実行
- 先に実行された命令を親として命令同士が親子関係を持つ
 - 前の命令との互換性を意識して次の命令を記述する必要

Dockerfileのビルドエラー

| Dockerfileのビルドは失敗することがある

- 他の言語のビルドよりもビルドエラー発生率が高い^[4]
 - 34%のDockerfileでビルドエラーが発生
- Dockerfileのビルドエラー修正には多くの時間が費やされる^[5]
- スピーディなソフトウェア開発の障壁になる

Dockerfileのビルドエラーを自動で修正することで
ソフトウェアの迅速な開発へ貢献できる

[4] Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S. and Gall, H. C.: An Empirical Analysis of the Docker Container Ecosystem on GitHub, Proceedings of MSR '17, pp.323–333 (2017).

[5] Wu, Y., Zhang, Y., Wang, T. and Wang, H.: An Empirical Study of Build Failures in the Docker Context, Proceedings of MSR '20, pp. 76–80 (2020).

先行研究 | Shipwright

| Shipwright^[6]

- ルールベースのDockerfileのビルドエラー修正手法
 1. ビルドログをクラスタリング
 2. クラスタリング結果から, 10個の修正ルールを作成し, 18.9%のビルドエラーを修正

| Shipwrightの課題

- 修正率が低い
 - 大きいクラスタのみに対してルールを作成しているため
- 手作業によるルール作成・メンテナンスが手間
 - Dockerfileは時間経過により陳腐化するため頻繁に更新が必要^{[4][7]}

[6] Henkel, J., Silva, D., Teixeira, L., d'Amorim, M. and Reps, T.: Shipwright: A Human-in-the-Loop System for Dockerfile Repair, In Proceedings of ICSE '21, pp. 1148–1160 (2021).

[7] Hassan, F., Rodriguez, R. and Wang, X.: RUDSEA: Recommending Updates of Dockerfiles via Software Environment Analysis. In Proceedings of ASE'18. pp796 - 801

目的 & アプローチ

目的

ビルドエラー修正率の向上
パッチ生成・メンテナンスの手間削減

| アプローチ

- アソシエーションルールマイニング^[8]による修正ルールの作成
 - ビルドエラー発生行以前の命令を前提部
 - ビルドに成功したDockerfileからアソシエーションルールを作成
- 作成したルールの評価
 - アソシエーションルールと実際の修正の比較評価

[8] Agrawal, R., Srikant, R.: “Fast Algorithms for Mining Association Rules in Large Databases”, In Proceedings of VLDB '94, pp. 487 - 499

アソシエーションルールマイニング

| データ中の要素間で、関連性や同時性の高い組合わせを効率よくルールとして抽出する手法

- ある前提Aが発生した場合、帰結Bが発生する

[男性, 紙おむつ] → [ビール] @スーパーマーケット

1. 頻出アイテム集合の探索

- 指定したSupportを満たす組合わせアイテムを探索

2. 相関ルールの探索

- 1. の結果からConfidence, Liftを用いて相関ルールを抽出

アソシエーションルールマイニング

| Support : 全データでのアイテム出現割合

Support

$$\frac{\text{事象Aと事象Bが同時に発生するデータ数}}{\text{全データ数}}$$

| Confidence : アイテム同時出現割合

Confidence

$$\frac{\text{事象Aと事象Bが同時に発生するデータ数}}{\text{事象Aが発生するデータ数}}$$

| Lift : 2つのアイテム間の相関

Lift

$$\frac{\text{事象Bだけが発生するデータ数}}{\text{Confidence}}$$

Dockefrileへの適用

```
FROM python:3-alpine
```

```
COPY [".", "/app/src/"]
```

```
WORKDIR /app/src/
```

```
RUN apk --update add ca-certificates \
    && update-ca-certificates \
    && rm -rf /var/cache/apk/*
```



アソシエーションルール
を適用

```
FROM python:3-alpine
```

```
COPY [".", "/app/src/"]
```

前提部

```
WORKDIR /app/src/
```

帰結部

```
RUN apk --update add ca-certificates \
    && update-ca-certificates \
    && rm -rf /var/cache/apk/*
```

| Dockerfileは、前の命令を考慮した命令の記述が必要

- ビルドエラー発生行以前の命令を前提部としたアソシエーションルールを作成

| 期待すること

- 多くの修正ルールを作成できる
- 自動でルールの収集が可能

データセット

- | Shipwrightのデータセットをフィルタリングして使用
- スター数10以上, 単一のDockerfileを持つGitHubリポジトリ
1. 破損しているリポジトリ
 2. 現在は存在していないリポジトリ
 3. マルチステージビルドを用いたDockerfileを持つリポジトリ
 - 今はとりあえず, 含めても問題ないと考えている

Dockerfile件数	ビルドエラー発生件数	ビルドエラー発生割合
16,856件	4,555件	27.0%

Dockerfileの前処理

| 余分な行を削除し，完全に1行に1命令に変換

```
# Install software and add share folder
RUN apk --update add \
    bash \
    iptables \
    docker \
    samba \
    && rm -rf /var/cache/apk/* \
    && mkdir /containers
```



```
RUN apk --update add
RUN apk --update bash
RUN apk --update iptables
RUN apk --update docker
RUN apk --update samba
RUN apk rm -rf /var/cache/apk/*
RUN mkdir /containers
```

1. DockerfileをTree-Sitterで構文解析

- － 各命令を取り出す

2. RUN命令をTree-Sitterで構文解析

- － 分解して1命令に1コマンドに変換
- － 命令の順番を考慮するため

現在までの進捗

| アソシエーションルールマイニングの実行

Support = 0.012, Confidence = 0.01, Lift = 1.0

- 手元のMacではこれが限界
- Google Colaboに移行してスクリプトを実行(中)

| Dockerfileの前処理で苦戦

- コマンドごとにルールを作成するしかない

これから

| Dockerfileの前処理を進める

| 作成されたルールの評価