

# Dockerfile の開発を支援する インタラクティブツールの提案

---

稲田 司

February 16, 2023

鵜林・亀井研究室

# 目次

導入

背景と目的

提案ツールの特長

簡単な使い方

主な機能

まとめ

# 目次

導入

背景と目的

提案ツールの特長

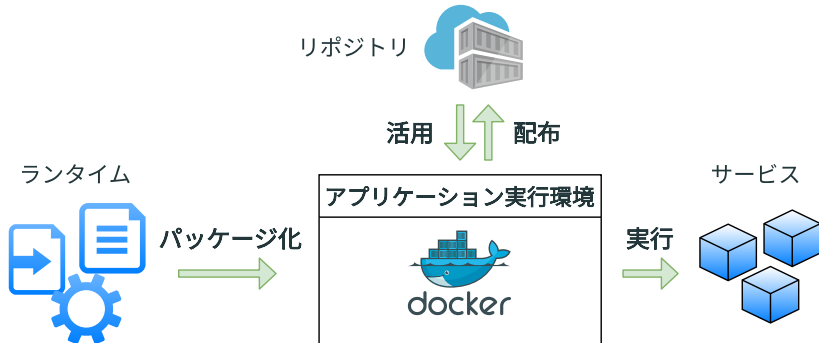
簡単な使い方

主な機能

まとめ

# Docker とは何か？

アプリケーション実行環境を構築，共有，起動するためのソフトウェアプラットフォーム．



# Dockerfile とは何か？

Docker において、

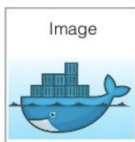
**イメージ** アプリケーション実行環境のスナップショット

**コンテナ** イメージから生成するアプリケーション実行環境

Dockerfile は、イメージ構築を自動化する一連の命令列が記載されたテキストファイル。

```
FROM ubuntu:14.04
MAINTAINER John Doe <john.doe@example.com>
RUN apt-get update && apt-get install -y python3
RUN python3 --help
CMD ["python3", "--help"]
```

build

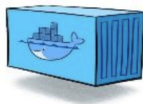


イメージ

run



(commit)



コンテナ

Dockerfile

# 目次

導入

背景と目的

提案ツールの特長

簡単な使い方

主な機能

まとめ

# イメージが抱える課題とその解決手法

ビルド時間の短縮

イメージサイズの削減

イメージ分析

サイバー攻撃への対策

同一性の確認

再現性の確保

イメージ開発の効率化

保守性の確保

# イメージが抱える課題とその解決手法

- ビルド時間の短縮 → キャッシュの利用, BuildKit
- イメージサイズの削減 → マルチステージビルド, Slim
- イメージ分析 → dive, dlayer
- サイバー攻撃への対策 → Distroless イメージ, BuildKit
- 同一性の確認 → ハッシュによる確認, イメージ署名
- 再現性の確保 → ?
- イメージ開発の効率化 → ?
- 保守性の確保 → ?



# イメージが抱える課題とその解決手法

ビルド時間の短縮	→ キャッシュの利用, BuildKit
イメージサイズの削減	→ マルチステージビルド, Slim
イメージ分析	→ dive, dlayer
サイバー攻撃への対策	→ Distroless イメージ, BuildKit
同一性の確認	→ ハッシュによる確認, イメージ署名
再現性の確保	→ ?
イメージ開発の効率化	→ ?
保守性の確保	→ ?

提案ツールで解決したい

# イメージが抱える課題とその解決手法

- ビルド時間の短縮 → キャッシュの利用, BuildKit
- イメージサイズの削減 → マルチステージビルド, Slim
- イメージ分析 → dive, dlayer
- サイバー攻撃への対策 → Distroless イメージ, BuildKit
- 同一性の確認 → ハッシュによる確認, イメージ署名
- 再現性の確保 → ?
- イメージ開発の効率化 → インタラクティブツール
- 保守性の確保 → リファクタリング・最適化の機能

# 目次

導入

背景と目的

提案ツールの特長

簡単な使い方

主な機能

まとめ

# なぜ、イメージ開発の効率化が必要なのか？

下のギャップが、開発を非効率にしている。

イメージ開発 Dockerfile を作成する。

動作確認 コンテナ内で行う。

現状の動作確認の手段

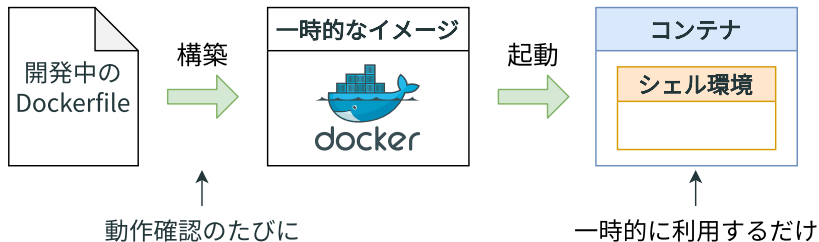
1. 開発中の Dockerfile から逐一起動するコンテナを使う。
2. 同じベースイメージから起動したコンテナを使う。

どちらも、問題あり。

# 開発中の Dockerfile から逐一起動するコンテナで動作確認しながら、Dockerfile を開発する

Dockerfile を修正する度に、一時的なイメージを構築する。

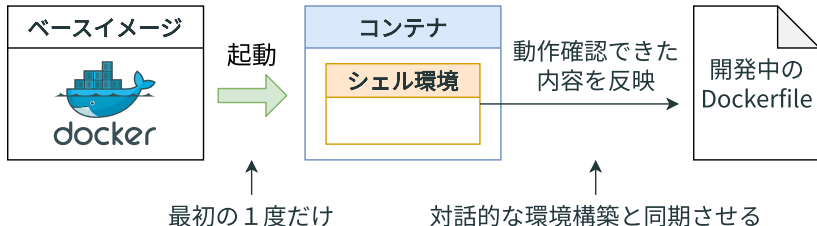
→ 待ち時間が発生する



# 同じベースイメージから起動したコンテナで 動作確認しながら、**Dockerfile** を開発する

目的の環境構築と Dockerfile の編集作業を同時進行させる。

→ 開発者の負担が大きい



# 提案ツールを用いて、Dockerfile を開発すると

## 対話的な環境構築 in fedora

```
cd root
```

```
export URL=http://example.com
```

```
URL=http://example.com/man.tar.gz
```

```
curl -LO "${URL}"; ls | grep -F tar.gz
```



自動的に

## Dockerfile

```
FROM fedora:latest
WORKDIR /
SHELL [ "/bin/bash", "--posix", "-c" ]
WORKDIR root
ENV URL=http://example.com
ARG URL=http://example.com/man.tar.gz
RUN curl -LO "${URL}"
```

# 提案ツールの特長

Dockerfile の効率的な開発手法が必要.



インタラクティブツールとしての側面

ユーザがコンテナ内で動作確認しながら環境構築することで、  
自動的に Dockerfile を生成してくれる.



# これだけでは優れた Dockerfile にはならない

## 修正前の Dockerfile

```
FROM fedora:latest
WORKDIR /
SHELL [ "/bin/bash", "--posix", "-c" ]
WORKDIR root
ENV URL=http://example.com
ARG URL=http://example.com/man.tar.gz
RUN curl -LO "${URL}"
RUN tar -xvf man.tar.gz
RUN rm -f man.tar.gz
WORKDIR man
```

これだけでは優れた Dockerfile にはならない

## 修正前の Dockerfile

```
FROM fedora:latest
WORKDIR /
SHELL [ "/bin/bash", "--posix", "-c" ]
WORKDIR root
ENV
ARG
RUN curl -LO ${URL}
RUN tar -xvf man.tar.gz
RUN rm -f man.tar.gz
WORKDIR man
```

見づらい

# これだけでは優れた Dockerfile にはならない

## 修正前の Dockerfile

```
FROM fedora:latest
WORKDIR /
SHELL [ "/bin/bash", "--posix", "-c" ]
WORKDIR root
ENV URL=http://example.com
ARG URL=http://example.com/man.tar.gz
RUN curl -LO "${URL}"
RUN tar -xvf man.tar.gz
RUN rm -f man.tar.gz
WORKDIR man
```

絶対パス指定の方がいい

集約した方がいい

# Dockerfile のリファクタリング・最適化を行ってみる

## 修正後の Dockerfile

```
FROM fedora:latest
SHELL [ "/bin/bash", "--posix", "-c" ]
WORKDIR /root
ENV URL=http://example.com/man.tar.gz

RUN set -ex; \
    curl -LO "${URL}"; \
    tar -xvf man.tar.gz; \
    rm -f man.tar.gz;

WORKDIR /root/man
```

# 提案ツールの特長

## インタラクティブツールとしての側面

ユーザがコンテナ内で動作確認しながら環境構築することで、  
自動的に Dockerfile を生成してくれる。

## リファクタリング・最適化の機能

上のようにして生成した Dockerfile に、ベストプラクティスに基づいたリファクタリング・最適化を行うことができる。

# 目次

導入

背景と目的

提案ツールの特長

簡単な使い方

主な機能

まとめ

## 手順 1 / 4

### 「提案ツールのソースコードを取得する」

提案ツールのリポジトリをクローンする.

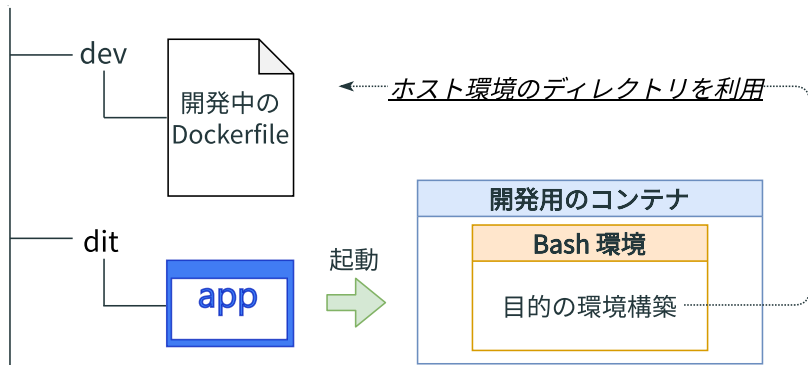
#### 実行例

```
URL=https://github.com/posl/dit.git  
git clone --depth 1 "${URL}"
```

## 手順 2 / 4

### 「開発用のコンテナを起動し、開発を開始する」

開発用のディレクトリを設定しながら、`exec.sh` を実行する。  
実行後は、指定したベースイメージに `bash` で入った状態。





## 手順 3 / 4

### 「動作確認しながら、目的の環境構築を行う」

コンテナ内で動作確認しながら、環境構築することで、それと同時に、**自動的に** Dockerfile が作られていく。

#### コマンドラインの実行

ex.

```
curl -LO "${URL}";  
ls | grep -F tar.gz
```

対話的に



#### Dockerfile の更新

ex.

```
RUN curl -LO "${URL}"
```

## 「Dockerfile を完成させる」

次の処理は、提案ツールの機能を使う。

**dit package** パッケージのインストール

**dit copy** ホスト環境からのファイルのコピー

**dit optimize** Dockerfile のリファクタリング・最適化

最後に、Dockerfile のリファクタリング・最適化を行う。

これで、目的の Dockerfile が完成する。

# 目次

導入

背景と目的

提案ツールの特長

簡単な使い方

主な機能

まとめ

# 機能要件

1. 任意のコマンドラインを実行するたびに、その必要性を判断して、対応する命令を Dockerfile に追加する機能
2. CUI 上で Dockerfile を編集する機能
3. ベストプラクティスに基づいて、作成された Dockerfile のリファクタリング・最適化を行う機能
4. Dockerfile の開発を任意のタイミングで中断・再開できるようにする機能

1. 任意のコマンドラインを実行するたびに、その必要性を判断して、対応する命令を Dockerfile に追加する機能
2. CUI 上で Dockerfile を編集する機能
3. ベストプラクティスに基づいて、作成された Dockerfile のリファクタリング・最適化を行う機能
4. Dockerfile の開発を任意のタイミングで中断・再開できるようにする機能

## 機能 1

「任意のコマンドラインを実行するたびに処理を行う」

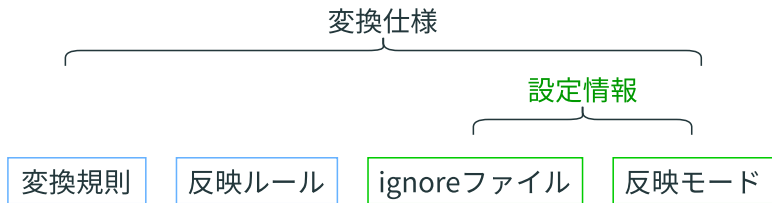
Bash のシェル変数 `PROMPT_COMMAND` を使用する.

### 使用例

```
# log each time the shell prompt is updated  
PROMPT_COMMAND='mylog > "/tmp/${date -Ins}"'
```

# 機能 1

「変換仕様により，コマンドの反映の要否を判断する」



- 
- |              |                            |
|--------------|----------------------------|
| • 変換規則       | コマンドと Dockerfile の命令を対応づける |
| • 反映ルール      | 構文情報からコマンドの反映の要否を決定        |
| • ignoreファイル | 反映しないコマンドとその条件を記述          |
| • 反映モード      | 反映ルール・ignoreファイルの使い方を変更    |
-

## 機能 1 変換規則の一覧

変換前	変換後	備考
cd pushd popd	WORKDIR 命令	作業ディレクトリが変更された場合
代入文 declare typeset readonly printf	ARG 命令	コマンド実行環境を変更する場合 左のコマンド以外には未対応
export	ENV 命令	コマンド実行環境を変更する場合 “declare -x” は ARG 命令に変換する
それ以外	RUN 命令	複数に変換されることはない



## 機能1 デフォルトの反映ルール（一部抜粋）

Bash の構文	反映ルール
単純なコマンド	出力のリダイレクションを含む場合は反映する
算術式	let に対する ignore ファイルの設定を使う
条件式	単独で使われた場合は反映しない
パイプライン	反映するコマンドより左にあるものは反映する
条件付きリスト	反映するコマンドを含む場合、まとめて反映する
その他のリスト	各コマンドの反映の可否を個別に決定する

※ 後述の反映モードの設定により、内容が少し変動する。

## 機能1 ignore ファイルの例

反映しないコマンドとその条件を記述する.

```
1 {
2     "ls": null,
3     "dir": "ls",
4     "wget": {
5         "short_opts": "0:",
6         "long_opts": {
7             "output-document": 1
8         },
9         "optargs": {
10             "output-document": "0",
11             "0": [
12                 "_"
13             ]
14         },
15         "detect_anymatch": true
16     }
17 }
```

## 機能1 反映モードの一覧

**no-reflect** Dockerfile に命令を追加しない

**strict** 処理の流れに影響しない部分は特別視しない

**normal** 処理のまとまりを考え、反映の可否を変更する

**simple** 単純なコマンド<sup>1</sup> 以外はそのまま反映する

**no-ignore** 実行されたコマンドラインは必ず反映する

strict と normal の違い（例：条件実行の “left && right”）

**strict** 左のコマンドだけを反映する可能性がある

**normal** ひとまとめに反映するか、全く反映しないか

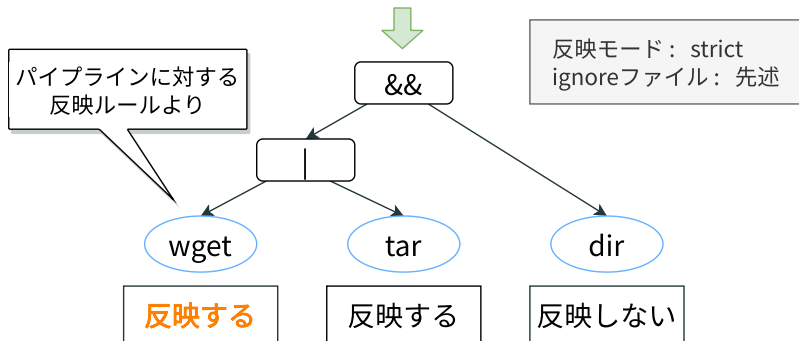
---

<sup>1</sup>Bash の基本構文.

# 機能 1 変換処理の流れ

## 変換前のコマンドライン (成功)

```
wget -O - "${URL}" | tar -xvz && dir -Af1 bash-5.1.16
```



## 変換後の RUN 命令

```
RUN wget -O - "${URL}" | tar -xvz
```

## 機能 1 変換処理の実行例 1 / 3

```
export URL=http://example.com; \  
URL="${URL}"/man.tar.gz;
```



変換処理

```
ENV URL=http://example.com  
ARG URL=http://example.com/man.tar.gz
```

## 機能1 変換処理の実行例 2 / 3

```
wget -O - "${URL}" | tar -xvz && \
ls -A man | grep -E '*.md'
```



変換処理

```
RUN wget -O - "${URL}" | tar -xvz
```

## 機能1 変換処理の実行例 3 / 3

```
cd /root/src || exit 1; \  
CFLAGS='-O2' make CC='gcc' && \  
mv -f a.out ../mycmd; \  
mycmd --version; cd ..; rm -fr src;
```



変換処理

```
RUN cd /root/src || exit 1; \  
CFLAGS='-O2' make CC='gcc' && \  
mv -f a.out ../mycmd; \  
cd ..; rm -fr src;  
WORKDIR /root
```

1. 任意のコマンドラインを実行するたびに、その必要性を判断して、対応する命令を Dockerfile に追加する機能
2. CUI 上で Dockerfile を編集する機能
3. ベストプラクティスに基づいて、作成された Dockerfile のリファクタリング・最適化を行う機能
4. Dockerfile の開発を任意のタイミングで中断・再開できるようにする機能



## 機能 2

### 「ホスト環境からファイルをコピーする」

#### dit copy

ホスト環境からコンテナ内へのファイルのコピーを行い、その内容を COPY・ADD 命令として Dockerfile に反映する。

このコマンドを使うと、

- COPY 命令と cp コマンドの仕様の違いを意識せず済む。
- ADD 命令の tar 展開機能を利用できる。

## 機能 2

### 「パッケージをインストールする」

#### dit package

最適化された形式で、パッケージのインストールを行い、その内容を RUN 命令として Dockerfile に反映する。

このコマンドを使うと、

- イメージサイズの削減に効果的な方法で実行される。
- パッケージマネージャの違いをあまり意識せず済む。

## 機能 2

### 「Dockerfile に命令を追加する」

#### dit reflect

各種ログを取りながら，Dockerfile に命令を追加する．

#### 実行例

```
# reflects the contents of './instr.txt' in Dockerfile  
dit reflect -d instr.txt
```

```
# reflects the input contents as they are in Dockerfile  
dit reflect -dp -
```

## 機能 2

### 「Dockerfile から指定した行を削除する」

#### dit erase

条件にマッチする行を，Dockerfile から削除する．

#### 実行例

```
# deletes the lines added just before from Dockerfile
dit erase -d
```

```
# deletes all LABEL instructions from Dockerfile
dit erase -diy -E '^LABEL[[:space:]]'
```

# 機能要件

1. 任意のコマンドラインを実行するたびに，その必要性を判断して，対応する命令を Dockerfile に追加する機能
2. CUI 上で Dockerfile を編集する機能
3. ベストプラクティスに基づいて，作成された Dockerfile のリファクタリング・最適化を行う機能
4. Dockerfile の開発を任意のタイミングで中断・再開できるようにする機能

# 「Dockerfile のリファクタリング・最適化を行う」

### dit optimize

下書きの Dockerfile から，完成版の Dockerfile を生成する．

#### 処理内容

- 各命令に特有のリファクタリング
- 連続する同系統の命令列に対する最適化
- 環境の整合性を保つために必須の集約処理
- 順序に依存しない命令の並べ替え

### 機能3 リファクタリング・最適化の実行例 1 / 3

```
WORKDIR /  
WORKDIR root
```



リファクタリング

```
WORKDIR /root
```

### 機能3 リファクタリング・最適化の実行例 2 / 3

```
COPY --chown=root /etc/conf.d ./src  
COPY *.c *.h ./src/
```



```
COPY /etc/conf.d *.c *.h ./src/
```



## 機能3 リファクタリング・最適化の実行例 3 / 3

```
RUN curl -LO "${URL}"; \  
RUN tar -xvf man.tar.gz; \  
RUN rm -f man.tar.gz;
```



```
RUN set -ex; \  
  curl -LO "${URL}"; \  
  tar -xvf man.tar.gz; \  
  rm -f man.tar.gz;
```

# 機能要件

1. 任意のコマンドラインを実行するたびに，その必要性を判断して，対応する命令を Dockerfile に追加する機能
2. CUI 上で Dockerfile を編集する機能
3. ベストプラクティスに基づいて，作成された Dockerfile のリファクタリング・最適化を行う機能
4. Dockerfile の開発を任意のタイミングで中断・再開できるようにする機能

## 機能 4

### 「Dockerfile の開発を中断・再開できるようにする」

コマンドラインの実行履歴を保持する，履歴ファイルを導入することで実現する．

**開発中** Dockerfile と同じように履歴ファイルを編集する．

**再開時** 履歴ファイルを実行して，環境を再現する．

# 目次

導入

背景と目的

提案ツールの特長

簡単な使い方

主な機能

まとめ

## 背景

- Dockerfile を効率的に開発するためのツールがない。
- Dockerfile のリファクタリング・最適化ツールがない。

## 提案ツール

- コンテナ内の Bash 環境で，対話的に作業する。
- 動作確認に専念して，Dockerfile を開発できる。
- リファクタリング・最適化後の Dockerfile が得られる。