@passol78 2017年08月12日に更新

DockerでGemfileの更新時のビルド時間を短縮する

Rails docker

この記事は最終更新日から1年以上が経過しています。

前提

- すでにRailsプロジェクトは存在済みであることとします。
- バッドノウハウだと思いますので、ご注意ください。
- 余計なファイルを管理したくない場合はこの記事を見ない方がいいです。
- 高速に開発できる Docker + Rails開発環境のテンプレートを作った がいい記事ですので、こちらを利用して開発を進めるのもいいと思います。
  - Entrykitについては、 Entrykitのすすめ がわかりやすくていいです。

はじめに

みなさん、Railsの開発をDocker上で開発する際に、Gemfileの更新をしますよね？

その場合Gemfileのcopyからやり直しになるので、bundle installが始めから走ってしまい、結構長い時間のロスになってしまうこともあるかと思います。

ここでは、決して褒められた方法ではないですが、Gemfile更新時のDockerimageの再ビルドの短縮方法について記載させていただきます。

なぜ Gemfile を更新すると再buildに時間がかかってしまうのか？

例えば、以下のようなDockerfile[参考]があったとします

Dockerfile

```
FROM ruby:2.3

RUN apt-get install hoge

# 中略

WORKDIR /app

# Gemfileのコピー
COPY Gemfile /app
COPY Gemfile.lock /app

RUN bundle install

# アプリのcopy
COPY . .

CMD bin/rails s
```

上記のようなDockerfileの場合、1回image化するとGemfile及びGemfile.lockの変更がない間は、「RUN bundle install」までがimageのcache化され、imageの再作成が早くなります。

しかし、GemfileやGemfile.lockが編集されてしまうと、「COPY Gemfile /app」や「COPY Gemfile.lock /app」も再作成のbuild対象となってしまうためそれより下の 「RUN bundle install」もcacheされていないこととなってしまい、再実行されてしまうため、時間がかかってしまうのです。

Gemfileの修正があまり発生しない場合はいいですが、比較的多めにGemfileの修正がある場合にはこの時間のロスは大きくなってしまいます。

じゃあどうするか？

色々方法はあるかと思うのですが、以下のような方法でGemfileの更新の時間を短縮できます。

1. 現在時点のGemfile及びGemfile.lockを別名で保存する

```
% cp Gemfile .Gemfile.cache
% cp Gemfile .Gemfile.lock.cache
```

2. Gemfileでbundle installを2回実行するように修正する

Dockerfile

```
FROM ruby:2.3

RUN apt-get install hoge

# 中略

WORKDIR /app

# Gemfileのキャッシュ化
COPY .Gemfile.cache /app
COPY .Gemfile.lock.cache /app

RUN bundle install

# Gemfileのコピー
COPY Gemfile /app
COPY Gemfile.lock /app

RUN bundle install

# アプリのcopy
COPY . .
```

```
CMD bin/rails s
```

以上

なぜ短縮されるのか？

単純な話なのですが、.Gemfile.cache、.Gemfile.lock.cacheはGemfile及びGemfile.lockが修正されたとしてもcacheされたままですので、1回目の「RUN bundle install」もcache化されたままとなります。

GemfileとGemfile.lockの修正はcache化されませんが、1回目の「RUN bundle install」がcache化されているので、2回目の「RUN bundle install」は差分のinstallとなりますから、imageの再生成の速度がだいぶ短縮されます。

最後に

Dockerは便利ですよね。
いいDockerライフを過ごせますように。