

Refactorings and Technical Debt for Docker Projects

Emna Ksontini, Marouane Kessentini,
Thiago do N. Ferreira and Foyzul Hassan

ASE2021

発表者：松尾 和輝

背景

- **Docker**に代表されるソフトウェアコンテナは、近年、再利用可能なソフトウェア成果物を提供する主流の技術であり、産業界では現在、アプリケーションのコンテナ化を進めることが一般的な傾向となっている
- コンテナベースのプロジェクトでは、**Docker**と**Docker-compose**ファイルを含む様々なコンポーネントと、様々なコンテナを組み合わせたソースコードへのその他の依存関係が混在している

背景

- 他の複雑なシステムと同様に，コンテナベースのプロジェクトは，様々な成果物に関連した様々な品質や技術的負債の問題が発生しがちである
- しかし，コンテナベースのプロジェクトが実際にどのように進化し，維持されているかについては分かっていない

目的, アプローチ

- 目的
 - Dockerのプロジェクトに適用されるリファクタリングと, それにより軽減される技術的負債を明らかにする
- アプローチ
 - Dockerプロジェクトからリファクタリングに関わるコミットを特定し, 分析を行う

データセット

- プロジェクトの選択
 - BigQuery [11]上のGitHub公開アーカイブからDockerプロジェクトを選択, 2,342リポジトリ取得
 - 存在しないプロジェクト, 他のリポジトリからフォークされたリポジトリを削除
 - REFACTORとDOCKERというキーワードに言及したコミットメッセージが1つ以上ある
 - DockerおよびDocker-composeファイル以外のコードがほとんどJavaで書かれている

以上の条件を満たす合計68のDockerプロジェクト, 193のコミットをデータセットとする

定量的分析

- 193のコミットが行なったリファクタリングを手作業で分類
- 193のコミットのうち、44のコミットでDockerfile関連のリファクタリング、51のコミットでDocker-compose関連のリファクタリング、55のコミットでDockerfileやDocker-composeの変更に伴う通常のコード（Java, C++ など）のリファクタリングが行われていることがわかった
- 偽陽性を除く150のコミットで行われたリファクタリングから、リファクタリングの種類とリファクタリングによって返済された技術的負債を特定する

リファクタリングの分類

- 灰色がリファクタリング，紫色が技術的負債
- リファクタリングによって返済された技術的負債を線でつないでいる

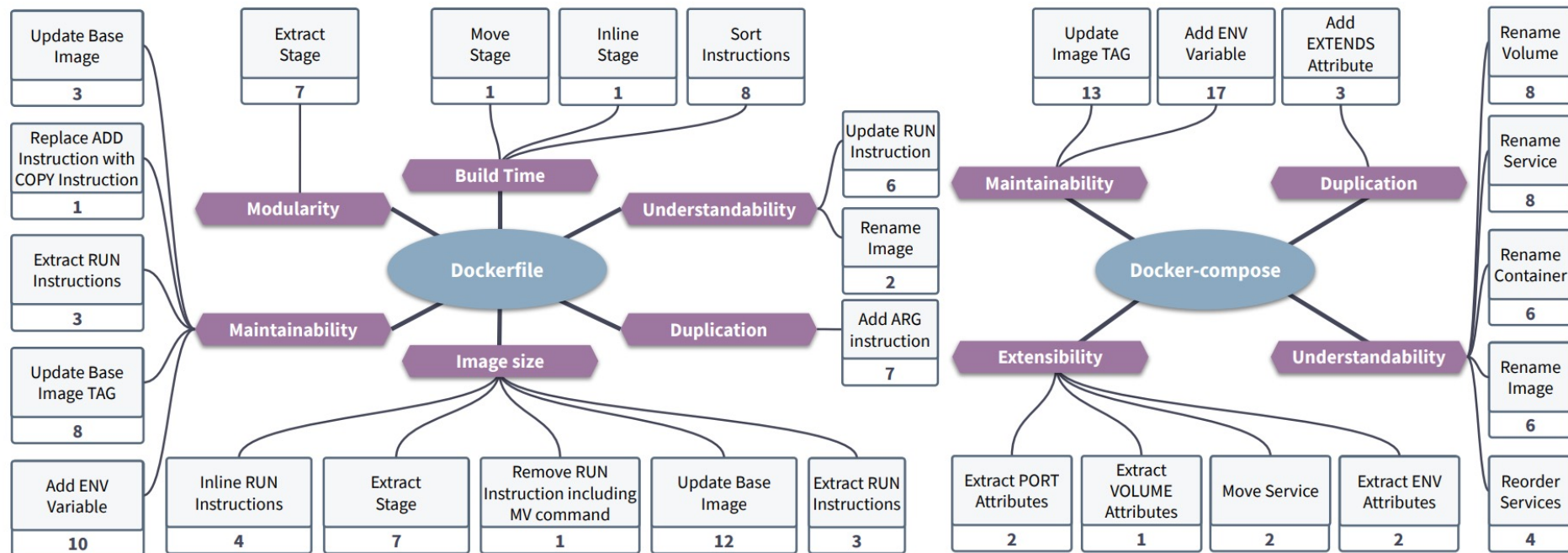


Fig. 1: Docker specific refactorings taxonomy.

リファクタリングの分類

- メンテナンス性を向上させるために行われたリファクタリングには、Update Base Image, Extract RUN Instruction, Replace ADD Instruction with COPY Instruction の3種類があった
- リファクタリングのうち、イメージサイズ（36％）を小さくすることを目的としたものが多かった
- 他には、Dockerfileをより分かりやすく、よりモジュール化するためのリファクタリングや、重複を避けてコードの再利用を促進することでコード設計を改善することを目的としたリファクタリングが行われている

リファクタリングの種類

TABLE IV: Discovered Docker-specific refactoring types

Refactoring Type	Artifact	Description
Extract stage	Dockerfile	Extract multi-stage building from a single-stage Building
Inline stage	Dockerfile	Aggregate multi-stage building into a single stage
Move stage	Dockerfile	Move a stage from a multi-stage context into another single stage context in a different Dockerfile
Sort Instructions	Dockerfile	Order Instructions sequence from the least frequently changing to the most frequently changing
Replace ADD Instruction with COPY Instruction	Dockerfile	Replace ADD instruction with COPY Instruction when files/directories need to be only copied from host to container
Extract Run Instructions	Dockerfile	Extract Run instructions commands into a separate script file
Inline Run Instructions	Dockerfile	Inline Run instructions into a single RUN instruction using && operator
Remove Run Instruction including mv command	Dockerfile	Delete RUN instruction with mv command and use previous COPY or ADD instructions to set the correct path
Update Base Image	Dockerfile	Avoid using unnecessary heavy image, replace base image with a lighter one
	Dockerfile	Avoid building and downloading dependencies on top of base image, replace base image with a larger one; if dependencies are fixed and a similar image exists in public or private repo, use an existing Image
Rename Image	Docker-compose	Set a relevant Name and TAG including the necessary information of your image (version, software..)
	Dockerfile	Add or rename Image alias
Update RUN Instruction	Dockerfile	Reformat RUN instruction commands; split commands into multiple lines where each line represents a single option/argument, order option/argument alphabetically, use backslash ...
Update Base Image TAG	Dockerfile	DRY principle, set a dynamic TAG using ARG instruction to avoid creating a new Dockerfile when TAG is only changing
	Dockerfile	Change TAG value or Replace latest TAG with an explicit TAG
Add ENV variable	Dockerfile	Add ENV variables to store useful system-wide values
	Docker-compose	
Add ARG instruction	Dockerfile	DRY principle, set dynamic instructions using ARG instruction to avoid creating a new Dockerfile for similar images
Extract Ports Attribute	Docker-compose	Extract ports attribute into an override Docker-compose file
Extract Volume Attribute	Docker-compose	Extract volume attribute into an override Docker-compose file
Extract ENV Attribute	Docker-compose	Extract ENV attribute into an override Docker-compose file or use .env file
Move Service	Docker-compose	Extract service into an override Docker-compose file
Rename Service	Docker-compose	Set a relevant Name for the service
Rename Container	Docker-compose	Set a relevant Name for the container
Rename Volume	Docker-compose	Set a relevant Name for the volume
Add Extends attribute	Docker-compose	Add Extends attribute to inherit configuration from an existing service thus avoiding duplication
Reorder Services	Docker-compose	Order services based on their dependency order
Update Image TAG	Docker-compose	Change TAG value or Replace latest TAG with an explicit TAG

技術的負債の種類

TABLE V: Discovered Docker-specific technical debt categories.

Technical Debt	Artifact	Goal	Situation	Consequence
Image size	Dockerfile	Refactorings are used to reduce the final image size	Adding new/changing components requires huge modifications and may lead to unexpected behaviors	Huge disk space, difficult to upload and a huge attack surface
Build Time	Dockerfile	Refactorings are used to reduce the build or re-build time	Docker engine takes a lot of time to build the final image	Evolving and changing the image became difficult process
Extensibility	Docker-compose	Refactorings are performed to increase the level of abstractions and improve the reusability of the Dockerfile and Docker-compose files.	Adding new components require duplication	Duplication
Duplication	Docker-compose	Refactorings are introduced to fix duplicated fragments in Dockerfile and Docker-compose.	Duplicated fragments	Adding new/changing existing components is difficult and error-prone
Maintainability	Both	Refactorings are performed to ease the modification as well as preventing large impact/spread of future bugs/unexpected behavior	Adding new/changing components requires modifying existing files	Modifications and upgrades require huge endeavors and conceivable downtime
Understandability	Both	Refactorings are applied to reduce the effort to understand code, e.g., renaming elements	Huge efforts to understand code	Slight and simple modifications will become time consuming
Modularity	Dockerfile	Refactorings are used to reduce image complexity by breaking image into various stages.	Complex image	Adding new/changing existing components is difficult and error-prone

リファクタリングの例

- Update Base Imageの例

- 最初にベースイメージとしてopenjdkを使い（5行目），RUN命令を使ってDockerizeソフトウェアがインストールされていたものを（7～12行目），公式のイメージを使うことにより（6行目）メンテナンスにかかる時間を大幅に短縮することができる

```
1 diff --git Dockerfile
2 --- a/docker/elasticsearch/Dockerfile
3 +++ b/docker/elasticsearch/Dockerfile
4 @@ -1,17 +1,12 @@
5 -FROM openjdk:8
6 +FROM jwilder/dockerize:0.6.1
7 -RUN apt-get update && apt-get install -y wget
8 -    && apt-get install -y curl
9 -ENV DOCKERIZE_VERSION v0.6.1
10 -RUN wget https://github.com/jwilder/docke[...]
11 -    && tar -C /usr/local/bin -xzvf docker[...]
12 -    && rm dockerize-linux-amd64-$DOCKERIZ[...]
```

Listing 4: Commit 4f221f9 from datahub: Refactored Dockerfile to reduce the number of pushed layers on a re-build.

まとめ

- 目的
 - Dockerプロジェクトにおけるリファクタリングと、リファクタリングによって返済される技術的負債の調査
- 方法
 - Dockerプロジェクトのリファクタリングに関するコミットから、リファクタリングと技術的負債を分類
- 結果
 - 24種のリファクタリング、7種の技術的負債を特定

所感

- Update Base ImageやAdd ENV Variableなどのリファクタリングが行われる回数が多いのか気になった
- Dockerfileのリファクタリングって必要？
 - 何度も利用しないイメージ