# DockerfileにおけるRUNとcurlを用いたイメージサイズの削減

山野 倖平[1]　遠藤 睦実[1]　串田 高幸[1]

**概要**：GitHub 上に公開されているオープンソースの Dockerfile では，ADD コマンドを用いてイメージ内にファイルを取り込んでいるケースが存在する．curl コマンドと tar コマンドを用いず，ADD コマンドを使用してイメージ内にファイルを追加すると Dockerimege 上で中間レイヤーが肥大化してしまう．ADD コマンドを使用した際に Dockerimage のイメージサイズにどの程度影響を与えるかを調べるため，基礎実験を行った．その結果，ADD コマンドを使用してファイルの追加を行う場合イメージサイズが増加することがわかった．そのため，ADD コマンドを使用した Dockerfile をそのままビルドすると，Dockerimage のサイズが増加するという課題がある．本項ではこの課題を解決するために，Dockerfile 実行時に自動的に ADD コマンドを curl コマンドと tar コマンドを使用した形へと自動的に変更するソフトウェアを提案する．提案ソフトウェアによって，ADD コマンドを使用していた時に加えリモート URL から取得するファイル分の容量が中間レイヤーが削除され，イメージサイズが削減される．本提案を評価するために,Github 上からクローンした docker-project という Dockerfile を対象として提案適用前と適用後の Dockerfile をそれぞれビルドし，Dockerfile のイメージサイズを比較した. その結果, 提案適用前は 245MB, 提案適用後は 239MB であり, 提案適用後のイメージサイズをは, 提案適用前のイメージサイズをより 6MB 削減できた．

## 1. はじめに

### 背景

Dockerfile とは Dockerimage のビルド手順を記述したものである [1]．Dockerfile をビルドすることで作成される Dockerimage は，Dockerfile の各命令事に作成される個別のレイヤーにより構成されている [2]．Dockerfile の記述にはベストプラクティス[*1]が存在し，Docker の公式ドキュメントのベストプラクティスに記述されている [3].

しかし，GitHub 上にある Dockerfile のうち約 83.8%の Dockerfile は何らかの形でベストプラクティスに違反しており，その中で約 4.4%は全命令で反ベストプラクティスな書き方がされている．逆に全てのコードがベストプラクティス通りに書かれた完璧なものは約 16.2%しか存在しない [4]．Dockerfile には様々なコマンドが存在し，その中でも使用するコマンドの中で似たような機能を持つが，内部処理が異なるものが存在する．代表例として ADD コマンドと COPY コマンドが挙げられる [5].

ADD コマンドの誤用は Dockerfile のベストプラクティス違反の 9.04%を占めている [6]．GitHub 上に公開されている Dockerfile の中には ADD コマンドを使用する必要のないケース，外部ファイルを Dockerimage に追加する．リモート URL の読み込みを行いファイルを追加するといったケースで，tar ファイルを展開しない場合は ADD コマンドを使用する必要がない．こういったケースでも ADD コマンドで記述されている．これによりディレクトリ構造の変更が発生する可能性，セキュリティリスク，中間レイヤーが削除できないためイメージサイズが増加するという問題がある．

このことについて Docker の公式ドキュメントでは ADD コマンドの使用は控えるべきだと明示されている[*2].

### 課題

Dockerfile において ADD コマンドは，Dockerimage を作成する際に外部のリソースを取得するために使用される．ADD コマンドを用いてリモート URL からファイルをダウンロードもしくはリポジトリ内から外部ファイルを呼び出し，Dockerimage 内に追加することが可能である．しかし，ADD コマンドを使用して外部ファイルを追加すると，Dockerimage のイメージサイズが大きくなり，

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1

[*1] https://www.nttdata-gsl.co.jp/related/column/what-is-best-practice.html

[*2] Docker 公式ドキュメント（日本語)https://docs.docker.jp/engine/userguide/eng-image/Dockerfile$_{b}est - practice.html$

Dockerimage の実行時間が増加する．

Dockerfile をビルドする際に一時的に必要なファイルが存在する．一般的な範囲として，ソフトウェアのイントールに使用するパッケージファイル，ソフトウェアのビルドツール，一時ファイル，テストデータ，デバッグ情報，ビルドキャッシュ，一時的な設定ファイル，ログファイルが使用される．これらのファイルは Dockerfile をビルドする際には必要だがそれ以降は不要となるため，イメージサイズを削減するために同一命令内で削除することが推奨されている．

ADD コマンドを使用した，外部からのファイル追加を行うと，追加したファイルのレイヤーが中間レイヤーとして新たに追加される．ビルド時に Dockerfile 内の命令を処理していく際，外部ファイルの取り込みの命令の後に，異なる命令で追加した一時ファイルを削除しても，ADD コマンドでファイルを追加した時点で中間レイヤーに追加されないたものは削除されない．これにより ADD コマンドにより作成された中間レイヤーが大きくなる．ADD コマンドで追加されたレイヤーの分だけイメージサイズが大きくなり，作成される Dockerimage のイメージサイズも大きくなる．curl コマンドと tar コマンドを使用して同一の処理を行う場合は，ファイルやディレクトリ，リモート URL からのファイルダウンロードを行う際，ダウンロードと削除を同じ単一の RUN コマンド内で行うことが可能なため，削除されたファイルの大きさが Dockerimage に含まれないのでイメージサイズが大きくなることはない．

図 1 は，ADD コマンドを使用した際の Dockerimage のレイヤーを表したものである．ADD コマンドを使用した場合の Dockerimage はリモート URL を ADD コマンドで展開した際に発生した中間レイヤーのサイズが大きくなっていることで，Dockerimage 全体が肥大化している．

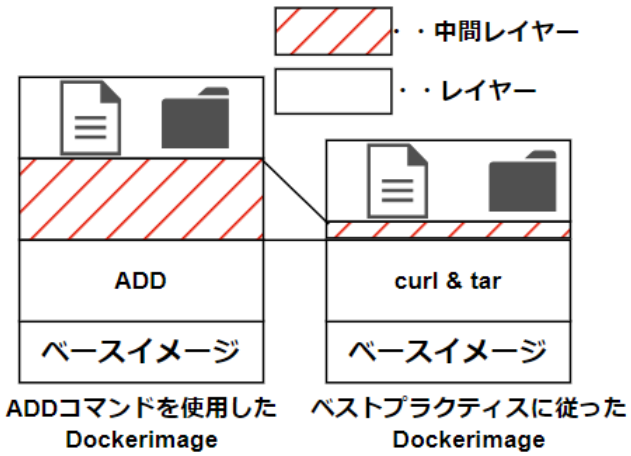


**図 1**　ビルド時における ADD コマンドの問題点

**基礎実験**

基礎実験として，リモート URL からビルド時に必要なファイルをダウンロードし削除する処理を ADD コマンドを用いて記述した Dockerfile と RUN コマンドと curl コマンドを用いて記述した Dockerfile をそれぞれビルドし比較調査を行う．

比較調査を行う際に使用するリモート URL として python 公式の提供する python3.7.3 のソースコードのダウンロード用 URL[*3]を使用する．ADD コマンドで使用した Dockerfile と RUN コマンドにて curl コマンドと tar コマンドを使用した Dockerfile をビルドした．表 1 はその結果である．RUN と curl コマンドを使用した Dockerfike のイメージサイズは 86.2MB，ADD コマンドを使用した Dockerfike のイメージサイズは 103MB と 16.8MB のイメージサイズ差が発生した．

**表 1**　Dockerfile 実行結果

| 使用コマンド | イメージサイズ |
|---|---|
| ADD https://www.python.org/ftp/python/<br>3.7.3/Python-3.7.3.tar.xz /usr/src/things/<br>RUN tar -xJf /usr/src/things/<br>Python-3.7.3.tar.xz -C /usr/src/things | 106MB |
| RUN mkdir -p /usr/src/things<br>&& curl -SL https://www.python.org/<br>ftp/python/3.7.3/Python-3.7.3.tar.xz<br>— tar -xJC /usr/src/things | 86.2MB |

**各章の概要**

本稿は以下のように構成される． 第 2 章では， 本稿の関連研究について記述する． 第 3 章では， 本稿で挙げた課題を解決するための提案について記述する． 第 4 章では， 提案した手法の実装について記述する． 第 5 章では，提案手法に対しての実験内容と， その評価について記述する． 第 6 章では， 提案手法の議論を記述する． 第 7 章は， 本稿のまとめについて記述する．

## 2. 関連研究

GitHub 上の Dockerfile の状態と進化に関する構造化された情報を収集し，ユニークなデータセットを作成した研究が存在する [7]. この研究で作成されたデータセットの調査結果として Dockerfile の品質，成功率，変化のパターンや傾向調査のデータ収集プロセス中に，各 Dockerfile に対してを調査を実行して，ベストプラクティスの遵守状況を分析し，潜在的な品質問題や頻出するベストプラクティス違反について明らかにした．

[*3] python 公式 python3.7.3 ソースコードダウンロードリンク https://www.python.org/ftp/python/3.7.3/Python-3.7.3.tar.xz

Hadolint や KICS などの最先端のスキャンツールを使用して，Dockerfile smells を効率的に検出する研究が存在する [8]．Dockerfile smells とは Dockerfile において最適化されていない部分や，パフォーマンスやセキュリティに影響を及ぼす可能性のある部分のことである．Dockerfile smells の検出ツールは現状不足しているため，11 種類の Dockerfile セキュリティ匂いタイプの修正を提案する自動修復ツール，DockerCleaner を紹介している．この論文で提案されるツールでは Dockerfile を作成するためのセキュリティのベストプラクティスにヒントを得た修復アクションを採用して実装した．評価結果は、DockerCleaner が Dockerfile の 92.67%から人為的に注入されたセキュリティ臭を除去し．99.33%のビルド可能性を保証できることを示した．

高度な要件を含む Dockerfile の自動生成ツールの作成を深層学習を用いてコーディングタスクにどの程度使用できるかの検証を目的とした研究が存在する [9]．深層学習を利用し Dockerfile に必要な要件を要件を求め Dockerfile を自動生成を行い，想定した仕様と合致したかで評価を行った．結果として 500 件中 170 件が仕様通りに動作し全体の 34%の自動生成した Dockerfile がビルドされた．

## 3. 提案

### 提案方式

本提案は，開発者が Dockerimage を使用するために，Dockerfile を使用してビルドを行う際に，自動的に Dockerfile 内のリモート URL を取得する ADD コマンドを curl と tar へと修正してからビルドするソフトウェアを提案する．本提案は Linux 環境で作業するユーザーが Dockerfile をビルドする際に，提案ソフトウェアを用いてビルドを行う．ビルドを行う前に同一リポジトリ内に存在する Dockerfile を読み取り内部のテキストを確認する．Dockerfile 内のリモート URL を使う ADD コマンドの記述を RUN コマンドを用いて，curl と tar へと置き換える．ADD コマンドを RUN コマンドを用いた形へと変換し，そのコマンド内でそのファイルを削除することで中間レイヤーを削減する．クローン Dockerfile を作成し，そのクローン Dockerfile をビルドする．その際，元々対象となっていた Dockerfile はビルドせず変更も加えない．ADD コマンドが含まれていない，ADD コマンドが RUN に置き換える必要がある場合，ADD コマンドを使用した際に取得するデータを削除しない場合は元々ビルドする対象だった Dockerfile をビルドする．本提案を利用することで，ADD コマンドの使用を最小限に抑え，可能な限り RUN コマンドを使用することで，ビルドプロセスをより明確にし，再現性を向上させることを目標として，Dockerfile のビルドプロセスを最適化が可能となる．

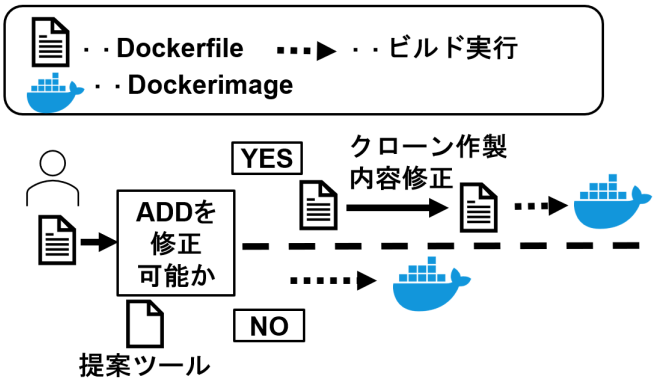


図 2　提案方式

### 判定方式

本提案において，ADD コマンドを置換するか否かの判断基準について記述する．まず，Dockerfile をテキストとして読み込み，その中に ADD コマンドが含まれているかを判定する．含まれていなかった場合は処理を終了し Dockerfile をそのままビルドする．含まれていた場合，その ADD コマンドで追加されたファイルが Dockerfile 内で削除されているかの判定を行う．ADD コマンドを変換する際の基準として，ADD コマンドで追加されたファイルが削除されているかという点で判定を行う．これは ADD コマンドを用いてファイルを Dockerimage に追加する場合，そこで作成された中間レイヤーが肥大化するからである．このためビルド後に削除されるファイルの追加は展開は RUN コマンドと curl コマンドと tar コマンドで行う方が適切である．以上より RUN コマンドと curl コマンドと tar コマンドへと ADD コマンドフを用いてァイルを追加する機能を置き換える判断を行う．

### ユースケース・シナリオ

開発者は Github 上にある Dockerfile を使用するために Github 上のリポジトリをクローンした．しかし，クローンした Dockerfile はベストプラクティスを尊守せず ADD コマンドを使用して，リモート URL からファイルをダウンロードしていた．その Dockerfile をビルドして Dockerimage を作成した際，ダウンロードしたファイルが大きく，不必要なファイルがイメージに含まれた結果，Dockerimage のサイズが膨らんだ．その結果，Dockerimage の実行時間も長くなり，作業効率が低下した．

この問題を解決するために，提案ソフトウェアを使用した．図 3 は提案適用後の図である．

開発者はこのプログラムに作成した Dockerfile を使用することで Dockerfile 内の ADD コマンドで記述されていたコマンドが COPY コマンドや RUN，curl コマンドに Dockerimage の動作は変更せずに置き換わる．この提案ソフトウェアの導入により，開発者は Dockerfile をビルドするだけで，イメージサイズが減少し，Dockerimage の実行

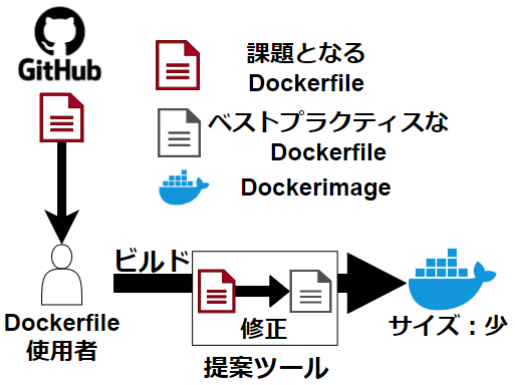


**図 3** 提案使用後のユースケース図

時間が減少する．

## 4. 実装

提案手法を基に, ビルド時に Dockerfile の自動修正を行うソフトウェアを Python を用いて新たに作成する．本提案ソフトウェアは Linux 上でコマンドとして呼び出されることを想定している．

### Dockerfile のビルド時処理

この処理は，Dockerfile を含むリポジトリ内での実行を前提としている．ユーザーがコマンドを実行すると，そのコマンドが実行されたディレクトリ内の「Dockerfile」という名前のファイルを検索し，そのファイルに対して処理を行う．処理の内容として，対象となる Dockerfile の内容を読み取り，その中に ADD コマンドが含まれているかを判定する．ADD コマンドが含まれていた場合，Dockerfile の各行に対して処理を行う．Dockerfile を 1 行づつ読み込み，ADD コマンドが含まれている行か否かの判定を行う．ADD コマンドが含まれていた場合，まず ADD コマンド内に tar が含まれているか判定を行う．含まれていた場合，その ADD コマンドへの処理は中止される．次に，ADD コマンドで追加されているファイルが Dockerfile 上での名称を特定する．特定手法として，ADD コマンドの内容を空白を区切り文字として分割して，２つ目の要素に対して処理を行う．これは ADD コマンドは，ADD 追加するファイル 追加する場所という構成となっているためである．その後「https://」「http://」が含まれていた場合削除する．最後に [/] を区切り文字として分割を行い，分割を行ったリストの最後の要素を取得した Dockerfile 上でのファイルの名称と判定する．最後に，ファイルの削除コマンドである rm が含まれており，ADD コマンドで追加されたファイルの名称が含まれる RUN コマンドが存在するか判定を行う．これらの条件を満たした場合，Docker ファイルの置き換え処理を行う．ファイルの全行に対する処理が完了した時点で，置き換え処理が行われ Dockerfile の内容が変更されていた場合，修正内容を反映した Dockerfile のクローンを同じディレクトリ内に作成し，そのクローンの Dockerfile でビルドを実行する．一方，置き換えが行われなかった場合，元々の Dockerfile をそのままビルドする．

### ADD コマンドの置き換え処理について

本処理は Dockerfile に対して一行ずつ処理を行う．以下に各行に対する処理を記述する．図 4 は，置き換え処理の流れである．

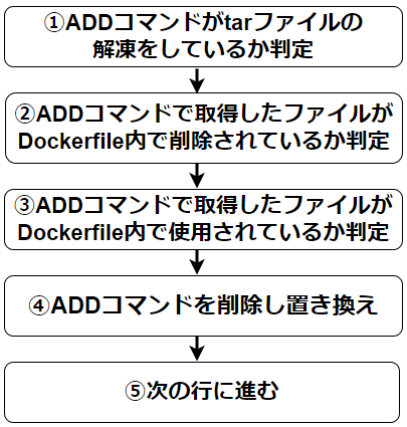


**図 4** 各行に対する処理

① tar ファイルを展開する処理が含まれているか判断を行う．含まれていた場合，その ADD コマンドに対する判定は行なわず，次の行に移行する．

② ADD コマンドから，ADD コマンドで追加されたファイルが削除された RUN コマンドの行までに存在する RUN コマンドを全て取得する．

③ 取得した RUN コマンドの行で ADD コマンドで追加されたファイルが使用されている場合，書き込み用の RUN コマンドとして保持する．

④ ADD コマンドを RUN コマンドと corl コマンドを使用して同一の処理を行う命令文へと変換する，その命令内にで作業した書き込み用の RUN コマンドを結合し，結合した RUN コマンドを元々の行から削除する．完成したものを処理を行っている ADD コマンドの行と差し替える．

⑤ 次の行の処理へと進む．

上記の手順で判定を行い全ての判定基準を満たした場合のみ，ADD コマンドを置き換える処理を行う．どれか一つの基準でも満たさない場合は次の行へと進む．

## 5. 評価実験

評価実験として，Dockerfile を用いてそのままビルドし

た Dockerimage のイメージサイズと提案ソフトウェアを用いて修正した Dockerfile を用いてビルドした際の Dockerimage のイメージサイズを比較する実験を行う．

### 実験環境

実験対象とするリポジトリとして，Github から docker-project[*4]を git から clone して使用する．このリポジトリでは ADD コマンドで追加したファイルを Dockerfile 内で削除しているという条件を満たしており，RUN コマンドが統合されていないため，今回の実験で使用した．

### 実験結果と分析

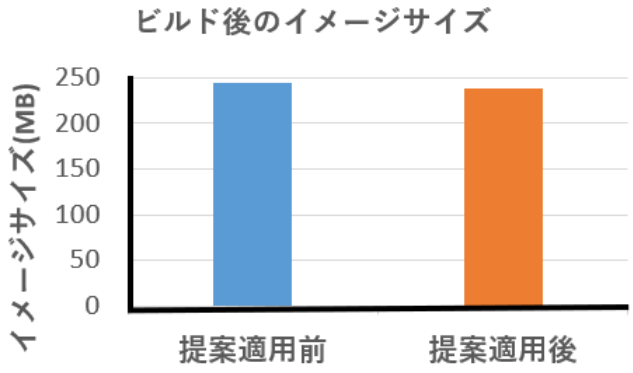


**図 5** 実験結果

図 5 は，実験結果を棒グラフで表したものである．縦軸は Dockerfile をビルドした Dockerimage のイメージサイズを表している．横軸は提案使用前か後かを表している．提案適用前の Dockerimage のイメージサイズは 245MB であり，提案適用後の Dockerimage は 239MB である．提案適用前と提案適用後の Dockerimage のイメージサイズを比較すると，提案適用後のほうがイメージサイズが 6MB 削減されている．

本提案で使用した Docker ファイルでリモート URL から取得したファイルのサイズ 2.43MB であり，このファイルを展開すると 3.56MB である．中間レイヤーから中間ファイルを削除することが本提案の目的である．削減されたイメージサイズは 6MB であり，ダウンロードしたファイルの展開前と展開後の合計分である 5.99MB 以上の容量を削減できたため目的は達成された．これは，Dockerfile 内で外部から追加されていたファイルの大きさ分中間レイヤーが削減されたものと，RUN コマンドを統合したことにより中間レイヤーが削減されたことで，Dockerimage に含まれるレイヤーの大きさが小さくなった．提案の適用によって外部から追加されていたファイルのサイズ分 Dockerimage が軽量化することができた．

[*4] https://github.com/zafar-khan123/docker-project/tree/master

## 6. 議論

本稿の提案では，提案ツールを使用し Dockerfile の内容を読み取った際に例外的な記述をされている場合に本提案が適用できない．例えば，ADD コマンドを連結し 2 行で記述している場合では ADD コマンドを置き換える判定を行う際に正常に判断できない．Github 上のオープンリポジトリで ADD コマンドを使用している場合，多くがベストプラクティスではないファイルであるため例外的な記述を全て網羅することは不可能である．解決作として，行数ではなく次にコマンドが実行されている行までをひとくくりとして判定を行う．

本稿の提案では，提案ツールを使用した際に同一ディレクトリ上にある「Dockerfile」という名称のファイルを提案の対象となる Dockerfile として定義しているため，この名称以外が用いられた Dockerfile が存在するリポジトリに対しては，本提案を適用できない．Dockerfile とは，Dockerimage をビルドするための命令が記述されたテキストファイルである．そのため独自の拡張子が存在せず非公式的に「Dockerfile」という名称が使用されている[*5]．そのため，本提案を「Dockerfile」以外の名称の Dockerfile にも適応する場合は，提案の対象となる Dockerfile をディレクトリ内から特定する必要がある．特定を行う場合，ファイルの名称や拡張子から判別が行えないため拡張子の存在しないファイルの中身を読み込み Dockerfile に必須である FROM コマンドで始まることに加え [10]，ADD コマンド，COPY コマンド，RUN コマンドといった Dockerfile で使用されるコマンドが記述されているかという形で特定を行う必要がある．

今回は一つのリポジトリを対象として実験を行ったため，他の Dockerfile でもこの提案が適用できるかが評価できなかった．Github 上でビルド可能であり，ADD コマンドで外部からファイルを追加しているファイルのなかで，複数のファイルを追加している Dockerfile や．追加しているファイルが tar や png など拡張子の異なるファイル，直接リポジトリ内の外部ファイルを指定しているファイルを対象として追加の実験を行うことで，提案を適用できるユースケースを広げる．

本稿の提案を更に有効なものにするための手法として，ADD コマンドを RUN コマンドに変換したのちに Dockerifle 内の RUN コマンドを全て統合することで RUN コマンドを使用するたびに作成される中間レイヤーを削減し，更なるイメージサイズ削減を行える．

## 7. おわりに

課題は，Github 上からクローンした Dockerfile をビルド

[*5] https://zenn.dev/thaim/scraps/7e8152734ee340

する際に，ADD コマンドが使用されていることで，Dockerimage が大きくなり，実行時間が長くなることで作業が停滞してしまう点である．課題を解決するために，Dockerfile のビルド時に Dockerfile 内の外部ファイルを追加する ADD コマンドを，RUN コマンドと curl コマンドを使用した形に内容修正を行うソフトウェアを提案した．評価実験として, 提案適用前の Dockefiler と提案適用後の修正した Dockerfile をそれぞれビルドし，その Dockerimage のイメージサイズを比較した．結果として提案適用後のイメージサイズが提案適用前より 6MB 削減された．

## 参考文献

[1] Ayed, A. B., Subercaze, J., Laforest, F., Chaari, T., Louati, W. and Kacem, A. H.: Docker2RDF: Lifting the Docker Registry Hub into RDF, *2017 IEEE World Congress on Services (SERVICES)*, pp. 36–39 (online), DOI: 10.1109/SERVICES.2017.15 (2017).

[2] Zhao, N., Tarasov, V., Albahar, H., Anwar, A., Rupprecht, L., Skourtis, D., Paul, A. K., Chen, K. and Butt, A. R.: Large-Scale Analysis of Docker Images and Performance Implications for Container Storage Systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 32, No. 4, pp. 918–930 (online), DOI: 10.1109/TPDS.2020.3034517 (2021).

[3] Wu, Y.: Exploring the Relationship between Dockerfile Quality and Project Characteristics, *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 128–130 (2020).

[4] Wu, Y., Zhang, Y., Wang, T. and Wang, H.: Characterizing the Occurrence of Dockerfile Smells in Open-Source Software: An Empirical Study, *IEEE Access*, Vol. 8, pp. 34127–34139 (online), DOI: 10.1109/ACCESS.2020.2973750 (2020).

[5] 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, *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 323–333 (online), DOI: 10.1109/MSR.2017.67 (2017).

[6] Lin, C., Nadi, S. and Khazaei, H.: A Large-scale Data Set and an Empirical Study of Docker Images Hosted on Docker Hub, *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 371–381 (online), DOI: 10.1109/ICSME46990.2020.00043 (2020).

[7] Schermann, G., Zumberi, S. and Cito, J.: Structured Information on State and Evolution of Dockerfiles on GitHub, *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pp. 26–29 (2018).

[8] Bui, Q.-C., Laukötter, M. and Scandariato, R.: DockerCleaner: Automatic Repair of Security Smells in Dockerfiles, *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 160–170 (online), DOI: 10.1109/ICSME58846.2023.00026 (2023).

[9] Rosa, G., Mastropaolo, A., Scalabrino, S., Bavota, G. and Oliveto, R.: Automatically Generating Dockerfiles via Deep Learning: Challenges and Promises, *2023 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pp. 1–12 (online), DOI: 10.1109/ICSSP59042.2023.00011 (2023).

[10] Oumaziz, M. A., Falleri, J.-R., Blanc, X., Bissyandé, T. F. and Klein, J.: Handling Duplicates in Dockerfiles Families: Learning from Experts, *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 524–535 (online), DOI: 10.1109/ICSME.2019.00086 (2019).