GitHub 上にアップロードされた.dockerignore ファイルをもとにしたテンプレートの作成

圖齋 雄治[1]　坂本 一俊[1]　串田 高幸[1]

概要：Docker の使用において, .dockerignore ファイルを記述しておく事は docker イメージサイズの削減や, ビルド時間の短縮に繋がる．しかし, .dockerignore ファイルは docker コンテナの制作者が自ら用意する必要があり，経験の無い Docker 初心者にとって負担となる．本研究では GitHub にアップロードされている.dockerignore ファイルを参照して, 多くのファイルに記述されているものを調査して.dockerignore ファイルのテンプレートを作成した. 評価のために，テンプレート作成に使用したデータとは別の，GitHub 上の 3 つの.dockerignore ファイルの記述をどの程度網羅しているかを評価した．結果として，網羅率は 18%，100%，22%であった．

1. はじめに

背景

コンテナ技術は, アプリケーションの開発工程で導入されおり, アプリケーションのリリースライフサイクルの重要な部分になりつつある [1]. コンテナ化によって, 巨大なアプリケーションをマイクロサービスと呼ばれる断片に分割しデプロイすることで, それぞれのアプリケーションをシステムの他の部分から独立させながら, アップグレードやスケーリングを行えるようになることで, システム全体の管理を容易にする．結果として, システムの継続的な改善や運用・開発コストの削減, DevOps の実現などが可能となるためである [2][*1][*2][*3].

コンテナは, それ以前に使用されてきた仮想マシンとは異なり, 同一のオペレーティングシステム・カーネルを共有する．これによりプロセッサやメモリの使用を抑えることができ, 起動時間が短くなる [3].

コンテナフレームワークの一例として, Docker がある．Docker は, 仮想のコンテナを自由に作成, 破棄, 運用できるオープンソースのテクノロジーとして, アプリケーション開発に使用されている[*4]．一般的に, Docker においてコンテナをビルドする際には, 自分で記述する, または GitHub から用意した Dockerfile を用いる．Dockerfile を自分で用意する利点として, ビルド後の docker イメージサイズの縮小, docker イメージのビルド時間の短縮が挙げられる．その後, Dockerfile の記述を基に, 他のファイルやディレクトリをデーモンに送信し, docker イメージを作成する．最終的に, この docker イメージに対して, run コマンドを用いて, コンテナを起動させる．

この時，ビルドに必要であるかを問わず，Dockerfile が存在するディレクトリ内や，COPY や ADD 命令で記述された全てのデータがデーモンに送信される．その中には, モジュールやログファイルなどの本番環境に不必要なファイルが存在している．これらのファイルやディレクトリが docker イメージに含まれてしまうと, コンテナイメージのサイズ肥大化だけでなく, ビルド時間の増大やセキュリティホールの増加に繋がる．そのため, ベストプラクティスとして, .dockerignore ファイルを用いる．

課題

背景でも述べたように, Dockerfile を用意してコンテナを作成する際には, 図 1 のように，Dockerfile で COPY 命令や ADD 命令を受けたファイルやディレクトリ, Dockerfile が置かれているディレクトリの全てのファイルがデーモンに送られて docker イメージになる．

そして, Dockerfile からコンテナを作成する時には，イメージサイズの抑制やビルド時間の短縮，セキュリティ強化を目的として, 図 2 のように.dockerignore ファイルを記述する．

しかし, .dockerignore ファイルは docker イメージを作成

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.redhat.com/ja/topics/containers/whats-a-linux-container
*2　https://circleci.com/ja/blog/benefits-of-containerization/
*3　https://www.fujitsu.com/jp/products/software/resources/feature-stories/cloud/background/
*4　https://www.docker.com/resources/what-container/

© 2023 Cloud and Distributed Systems Laboratory

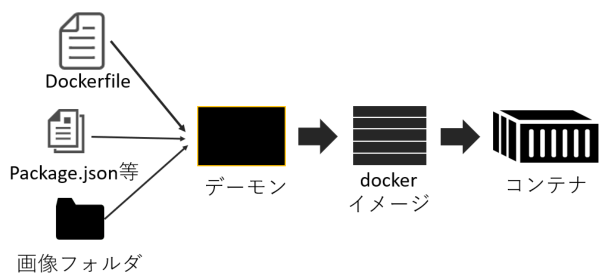

図 1　コンテナ作成の流れ

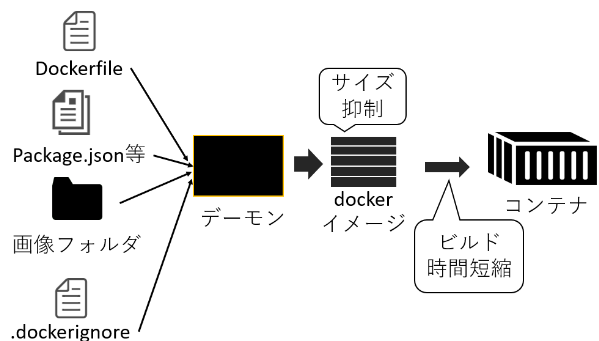

図 2　.dockerignore ファイルの存在意義

する人間が個人個人で記述する必要がある．これでは, 作成した各ファイルが本番環境に必要かを確認する手間が増える上に，.dockerignore ファイルに不要なディレクトリやファイルを書き込む時間が増加する．また，.dockerignore ファイルに記述する内容は, 経験則で書かれるため, Docker 開発初心者にとっては難しいと言える．

各章の概要

本テクニカルレポートは以下のように構成される．第 1 章では, 本稿の背景と課題について述べる．第 2 章では, 本稿の関連研究について述べる．第 3 章では, 本稿での課題を解決するための提案手法について述べる．第 4 章では, 提案した手法の実装について述べる．第 5 章では, 提案手法に対しての実験内容と, その評価について述べる．第 6 章では, 提案手法の議論を述べる．第 7 章は, 本稿のまとめである．

2.　関連研究

TOSCA(Topology and Orchestration Specification for Cloud Applications)*5という，既に存在するアプリケーションをテンプレートとして利用しようとする標準仕様を用いて，コンテナを作成しようとした研究がある．Klinbua らは, Docker 開発初心者に対して, すでに完成されている TOSCA のサービステンプレートを docker-compose YAML ファイルに変換するツールを提案した [4]．これは, TOSCA の構成要素を幾つかのグループに再定義し，このグループに docker-compose の構成アイテムを分類した後, グループごとに振り分けた docker-compose のアイテムを記述して TOSCA メタモデルを表す新しいタイプを作成する．そして, ANTRL における TOSCA の文法を定義し, リスナーとしてパーサを生成する．このパーサと先程作成したタイプを用いて dokcer-compose に記述をしていくことで, 初心者でも簡単に docker-compose YAML ファイルを作成できる．

また, Chareonsuk らは, 既に定義された TOSCA モデルを, ANTLR を用いて Kubernetes オブジェクトに変換することを提案した [5]．これは, TOSCA モデルと Kubernetes オブジェクトの関係を分析してマッピングし，ANTLR4 を使用して YAML 文法に従って Kubernetes オブジェクトに変換している．しかし, これら研究は中規模から大規模のシステムを簡単に作成できるものの, コンテナを複数必要としないシステムやアプリケーションでは使用できない. また, 自分の作成するシステムと似たネットワーク構成やソフトウェアを使用する TOSCA テンプレートを用意する必要があり, 柔軟性に欠ける．

Vaillancourt らは, さまざまな計算インフラストラクチャ上で移植可能かつ再現可能な方法で科学的ソフトウェアを開発および展開するための柔軟性を提供するソリューションとして，Docker または Singularity with Nix を使用して科学ソフトウェアをコンテナー化するための，使いやすく再現可能なテンプレートを作成した [6]．これは，Nix というパッケージビルドと依存関係を指定するためのパッケージマネージャーおよび関連言語を使用して，コンパイラやパッケージ間の依存関係を取得する．これらの情報を Dockerfile に整理し，Docker Hub にアップロードしている．ただし，この研究はノードが複数存在する場合を主に説明している．また，作成した Dockerfile や docker イメージがベストプラクティスに基づくものかどうかが不明である．

3.　提案

提案方式

本研究では, テンプレートを作成するにあたり, GitHub にアップロードされている.dockerignore ファイルの記述を参考にする事とした．図 3 のように，Github にアップロードされている.dockerignore ファイルを収集し，それぞれのファイルの記述を分析する．分析の結果，多数の.dockerignore ファイルに共通して書かれているファイルやディレクトリをテンプレートに記述する．特定の記述をテンプレートに記載する際の判断基準として，本研究では参考にした.dockerignore ファイルを記述しているリポジトリの内，過半数である 50%のリポジトリに記述されている内容を，テンプレートととした．この理由として，.dockerignore ファイルの記述の内容は，開発者の環境

*5 http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.pdf

や嗜好，開発するソフトウェアによって変化し，全ての開発者が同じような記述を書くことが少ないため，今回調査した.dockerignore ファイルの記述では，頻出する記述が少なかったためである．

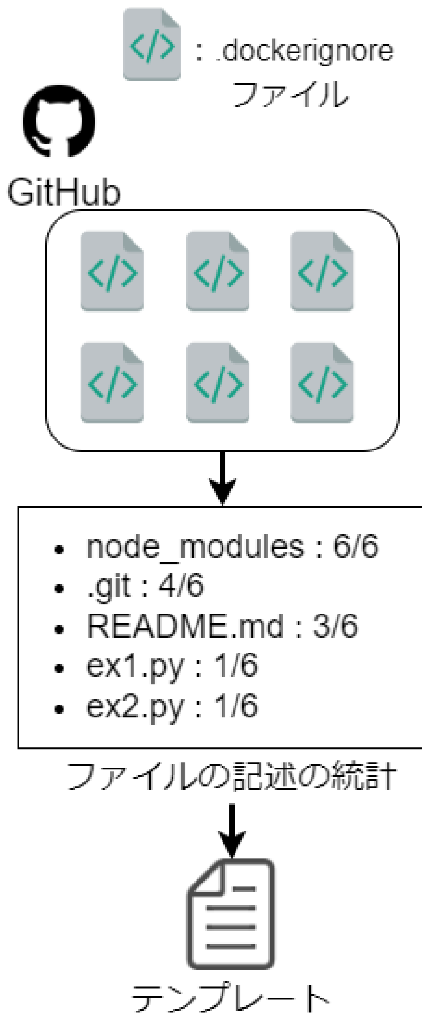

図 3　提案の流れ

また，統計の結果に関わらず，「.*Dockerfile.*」と「**/.*docker-compose.y*ml」と「**/.*.md」をテンプレートに加える．この理由として，Dockerfile と docker-compose.yml ファイルは，docker イメージの構築に参照されるファイルであるため，実際に構築された環境では使用しないからである．また，「**/.*.md」で網羅したい README.md ファイルは，GitHub での閲覧を目的としているファイルのため，実際に構築された環境では使用しないからである．

ユースケース・シナリオ

Node.js は Ruby や Python のように OS 上で動作する JavaScript 環境で，ネットワークアプリケーションを構築するために設計された．本研究では，図 4 のように，Node.js でブログを書いているユーザが，Node.js の環境を Docker に移動するというシナリオを考える．

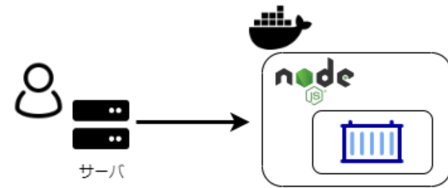

図 4　ユースケース

4.　実装

4.1　自動で GitHub 上の.dockerignore ファイルを収集

テンプレートを作成するためのデータとなる.dockerignore ファイルを，GitHub 上からを自動で取得するプログラムを Python で実装する．図 5 に実装するプログラムのフローチャートを示す．プログラムが実行されると，Github REST API を使用して，Github 上で検索を行い，その結果を取得する．Github の API は 1 分間に使用できる数に限りがあるので，この上限を越えた場合，Python の sleep() 関数を使用して一定時間待機する．制限がかかっていないのなら，取得した検索結果を一つずつ取り出し，辞書形式でリストに保管していく．最後にこのリストを.json ファイルに書き込む．　このプログラムを使用すれば、テンプレート作成に必要な.dockerignore ファイルを自動で収集する事ができる．しかし，このプログラムでは API 制限がかかってしまった場合，API 制限が解かれるまで待機した後，再び検索をかけるため，検索で API 制限がかかれば、

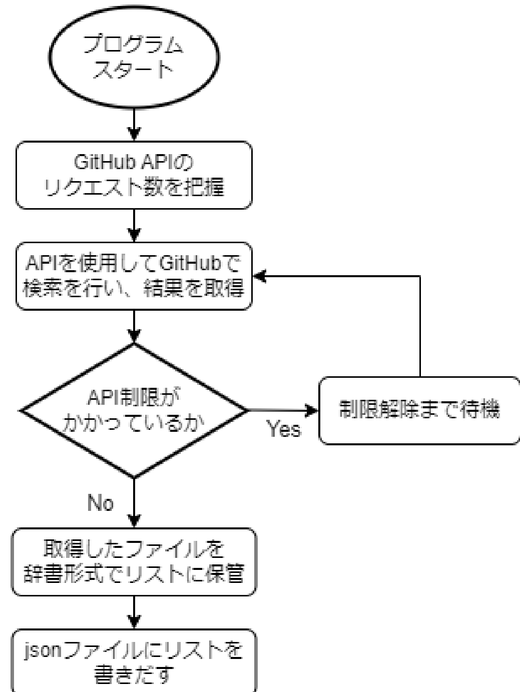

図 5　実装するプログラムのフローチャート

ループになる可能性がある．また，辞書形式でリストに保管する時にも API を使用するため，ここでも API の制限がかかる恐れがある．もしここで API 制限になると，再び検索フェーズに戻ってしまうため，安定してデータを収集できなかった．今回は，このプログラムでのデータ収集を諦め，別の手法で集めたデータを使用した．

4.2　テンプレート作成に用いたデータについて

上記 4.1 で述べたように, GitHub 上の.dockerignore ファイルを自動で収集するプログラムで取得したデータを使用せず，別の方法で収集したデータを利用した．GitHub で「Dockerfile」と検索し, ヒットするリポジトリの中から, メインの言語が JavaScript で構成されているリポジトリの内, 上位 99 個に該当するスターを 7 個獲得したリポジトリを対象に調査を行った. 結果として, 全 99 リポジトリの内, Dockerfile で Node.js 公式の Docker イメージを FROM 命令で用いていたのは 46 個あり, その内.dockerignore ファイルを記述していたのは, 22 リポジトリであった. 結果的に参照した.dockerignore ファイルの総数は 46 個であった.

4.3　テンプレート

上記 4.2 で収集したデータから，.dockerignore ファイルのテンプレートを作成した．作成したテンプレートの記述はソースコード 1 の通りである．

ソースコード 1　.dockerignore ファイルのテンプレート

```
1  node_modules
2  **/.git.*
3  .*log.*
4  .*Dockerfile.*
5  **/.*docker-compose.*
6  **/.*.md
```

5.　実験と分析

作成したテンプレートを評価していく．評価方法としては，今回統計データの対象とした GitHub 上のリポジトリとは違うリポジトリに記述されている.dockerignore ファイルを対象に，作成したテンプレートが.dockerignore ファイルをどの程度網羅しているか評価を行った．

実験環境

本研究では, テンプレートを作成するために GitHub から.dockerignore ファイルを収集して分析した. また，テンプレートを評価する為に，分析したデータ以外の.dockerignore ファイルを使用して評価した．テンプレートで網羅できたものは，ソースコードの記述の横に「網羅」と書いた．

実験結果と分析

まずは，Ethereal Engin の「XREngine」[*6]を対象に評価した．このリポジトリには公式の Node.js を FROM した Dockerfile があり，.dockerignore ファイルを記述している．2023 年 1 月 12 日時点で，スター数は 545 であった．このリポジトリの.dockerignore ファイルの記述はソースコード 2 の通りであった．結果として，テンプレートは 4 つの記述を網羅しており，網羅率は約 18%であった．この.dockerignore ファイルでは,「awscliv2.zip」や「kubectl」といった，テンプレート作成に当たって参考にした統計データに全く出てこなかった記述が書かれていたことに加えて,「.env」などのデータには存在したが少数だった記述が書かれている事が原因である．

ソースコード 2　.dockerignore ファイルのテンプレート

```
1  aws/
2  awscliv2.zip
3  .git/    網羅
4  *docker-compose* 網羅
5  node_modules     網羅
6  kubectl
7  lib/
8  packages/client/dist
9  packages/native-plugin-example
10 packages/ops
11 readme/
12 tests/
13 vendor/
14 packages/**/node_modules     網羅
15 package-lock.json
16 packages/**/package-lock.json
17 kubernetes/**
18 .vscode
19 config/prod.json
20 .env
21 .env.local
22 .idea/
```

続いて，Contentful の「the-example-app.nodejs」[*7]を対象に評価する．このリポジトリには，公式の Node.js を FROM した Dockerfile があり，.dockerignore ファイルを記述している．2023 年 1 月 12 日時点で，スター数は 399 であった．the-example-app.nodejs の.dockerignore ファイルの記述は「node_modules」と「npm-debug.log」だった．この結果として，テンプレートはこの記述を二つとも網羅しており，網羅率は 100%であった．これは，対象にした.dockerignore ファイルの記述がそもそも二つしかないため，網羅率が高くなったと言える．

続いて，Nylas の「nylas-mail」[*8]を対象に評価した．このリポジトリには公式の Node.js を FROM した Dockerfile

[*6] https://github.com/XRFoundation/XREngine
[*7] https://github.com/contentful/the-example-app.nodejs
[*8] https://github.com/nylas/nylas-mail

ソースコード 3 .dockerignore ファイルのテンプレート

```
1 node_modules     網羅
2 npm-debug.log     網羅
```

があり，.dockerignore ファイルを記述している．2023 年 1 月 12 日時点で，スター数は 24711 であった．このリポジトリの.dockerignore ファイルの記述は以下の通りであった．結果として，テンプレートは 4 つの記述を網羅しており，網羅率は約 22%であった．これは，今回の評価対象が「!」を使用して ignore をしようとした記述が 3 つあったことが挙げられる．また，7 行目と 12 行目が同じ記述であり，重複していることも原因として挙げられる．

ソースコード 4 .dockerignore ファイルのテンプレート

```
1 .arc*
2 .git*    網羅
3 arclib
4 **/node_modules     網羅
5 packages/client-*
6 !packages/client-app/.babelrc
7 *.swp
8 *~
9 .DS_Store
10 **/npm-debug.log     網羅
11 **/lerna-debug.log     網羅
12 *.swp
13 *.swo
14 .elasticbeanstalk/*
15 !.elasticbeanstalk/*.cfg.yml
16 !.elasticbeanstalk/*.global.yml
17 /packages/client-sync/spec-saved-state.json
18 n1_cloud_dist
```

6. 議論

本研究の提案において，テンプレートを作成するにあたって収集したデータは，公式の Node.js を使用する Dockerfile を作成している.dockerignore ファイルを対象にした．また，評価についても，同様に公式の Node.js イメージを使用しているリポジトリを対象に評価した．しかし，実際の Docker 開発では，他のユーザーが公式の Node.js を改造して作成した，オリジナルの docker イメージを FROM 命令で使用する Dockerfile もある．テンプレートの作成に際して，公式のイメージではないリポジトリの.dockerignore ファイルも対象にして分析する必要がある．同じように，テンプレートがオリジナルの docker イメージにも適用可能か評価する必要もある．

また，本研究で.dockerignore ファイルのテンプレートを作成するために，GitHub にアップロードされている.dockerignore ファイルを 99 ファイル分析したが，GitHub の検索によっては，他にも公式の Node.js を使用する Dockerfile を対象とした.dockerignore ファイルが存在することを確認しているので，これらのファイルも分析して，より統計の母数を増やすことも必要である．統計のデータ数が増加すれば，今回の評価では網羅できなかった記述を網羅できる可能性がある．テンプレートのための統計データを増やす手法として，実装の章で説明した，GitHub 上の.dockerignore ファイルを自動で収集するプログラムを稼働させる必要がある．4.1 でも述べた通り，GitHub の REST API の制限が発生すると，GitHub での検索からやり直されてしまうため，それまで書き込まれた.dockerignore ファイルの情報が消えてしまう．また，GitHub の検索システムは，同じワードで検索しても，該当のファイルが毎回同じ順番で表示されない．今後は，先に検索結果を保持しておき，API 制限が発生しても検索をやり直さないようにする，API 制限に近づいたら sleep() 関数で一定時間待機するなどの対策を行う．

7. おわりに

Docker においてコンテナをビルドする際に.dockerignore ファイルを記述することは docker イメージの削減や，ビルド時間の短縮，セキュリティ強化に繋がる．一方で.dockerignore ファイルへの記述は煩雑である．本研究では，公式の Node.js イメージに適用できる.dockerignore ファイルのテンプレートを，GitHub にアップロードされている，公式の Node.js を使用しているリポジトリの.dockerignore ファイルを基に作成した．このテンプレートの評価として，テンプレート作成に使用したデータとは別の，3 つの.dockerignore ファイルの記述に対してどの程度網羅出来ているかを評価した．結果として，網羅率は 18%，100%，22%であった．

謝辞 本稿の作成に当たり，本テクニカルレポートの執筆にあたりご助言を賜りました東京工科大学大学院バイオ・情報メディアコンピュータサイエンス専攻の飯島貴政さんと，東京工科大学コンピュータサイエンス学部の遠藤睦実さんに感謝申し上げます．

参考文献

[1] 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).

[2] Deshpande, U.: Caravel: Burst Tolerant Scheduling for Containerized Stateful Applications, *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1432–1442 (online), DOI: 10.1109/ICDCS.2019.00143 (2019).

[3] Felter, W., Ferreira, A., Rajamony, R. and Rubio, J.: An updated performance comparison of virtual machines

© 2023 Cloud and Distributed Systems Laboratory

and Linux containers, *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172 (online), DOI: 10.1109/ISPASS.2015.7095802 (2015).

[4] Klinbua, K. and Vatanawood, W.: Translating TOSCA into docker-compose YAML file using ANTLR, *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 145–148 (online), DOI: 10.1109/ICSESS.2017.8342884 (2017).

[5] Chareonsuk, W. and Vatanawood, W.: Translating TOSCA Model to Kubernetes Objects, *2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 311–314 (online), DOI: 10.1109/ECTI-CON51831.2021.9454890 (2021).

[6] Vaillancourt, P. Z., Coulter, J. E., Knepper, R. and Barker, B.: Self-Scaling Clusters and Reproducible Containers to Enable Scientific Computing, *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8 (online), DOI: 10.1109/HPEC43674.2020.9286208 (2020).

© 2023 Cloud and Distributed Systems Laboratory