

An Empirical Analysis of the Docker Container Ecosystem on GitHub

Jürgen Cito, Gerald Schermann, John Erik Wittern,
Philipp Leitner, Sali Zumberi, Harald C. Gall

GitHubでのDockerコンテナエコシステムの の実証分析

発表者：辻 裕真

背景

- ・コンテナ技術はソフトウェア基盤を構築する軽量仮想化技術として近年関心を集めている.
- ・Dockerのエコシステム, 一般的な品質の問題, およびDockerfileの進化を特徴づけることが目的.

コンテナ

- ・他のプロセスと隔離されたアプリケーション実行環境.
- ・仮想マシン(VM)と異なり別のホストOSをエミュレートするのではなく、基盤となるカーネルをホストと共有するため、オーバーヘッドが大幅に削減される.
- ・アプリケーションを依存関係と実行環境とともに自己完結型のユニットにパッケージ化できる.

Docker

- ・ソフトウェア開発コミュニティで急速に普及しDockerコンテナはソフトウェア開発の標準フォーマットになっている.
- ・DockerコンテナはDockerfileによって定義され, 1つのコマンドで分離された高速な環境でプログラムを実行できる.

データセット

- ・2016年10月にBigQuery2の公開GitHubアーカイブからDockerfileを含むリポジトリのリストを取得
- ・80054のプロジェクトの内の320000を超えるDockerfileから他のリポジトリから分岐したリポジトリを削除し重複が起こらないようにした.

データセット

- ・結果, 38079のプロジェクトの70197個の一意のDockerfileが抽出できた.
- ・所有者の種類, 所有者名, 使用されているプログラミング言語, プロジェクトサイズ, フォークの数, 問題, ウォッチャーの数などのデータを取得した.

Docker Ecosystem

Dockerfileを含むプロジェクトの特性を分析

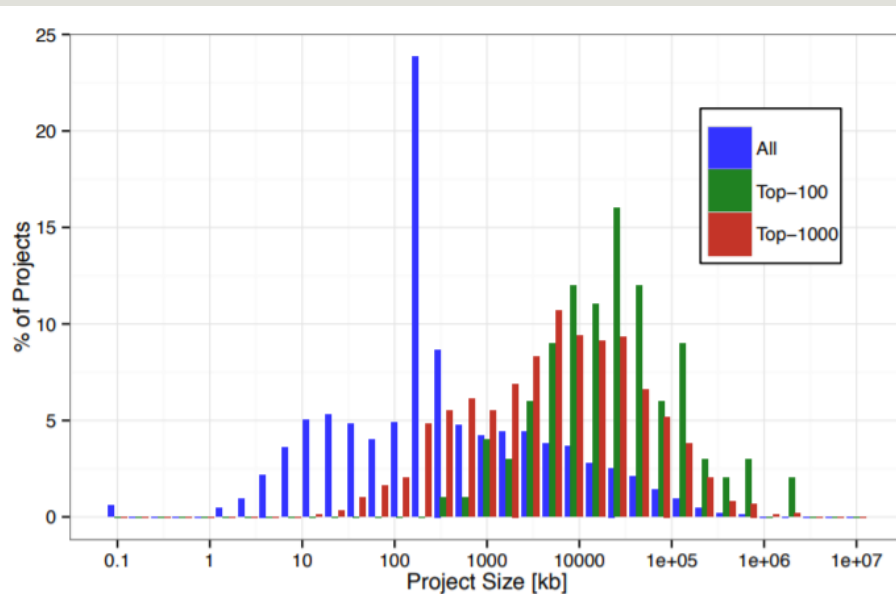


Fig. 3. Distribution of project sizes

人気のプロジェクト
(Top100及びTop1000)
のサイズは全体と比べ
大きくなっている。
しかし、これはDockerを
使用しているかに関わ
らず一般的なプロジェク
トでも言えることかもし
れない。

Docker Ecosystem

プログラミング言語の分布

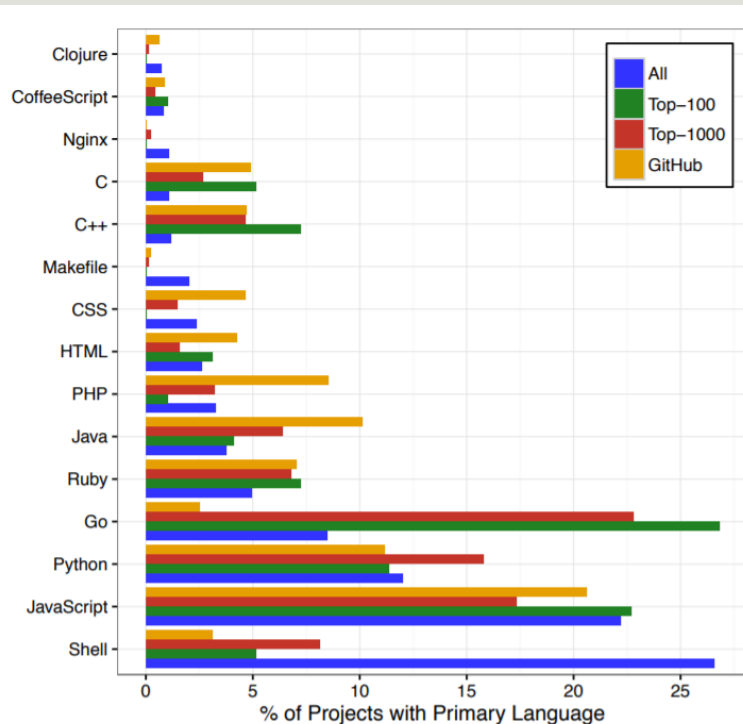


Fig. 4. Distribution of top 15 languages in our dataset

DockerプロジェクトではShellがかなりの割合を占めているがGitHubプロジェクト全体ではあまり一般的ではない。また、Goなども比較的多く使用されているが、PHPやJavaは全体と比べると比較的少ないことが分かる。

Docker Ecosystem

ベースイメージ

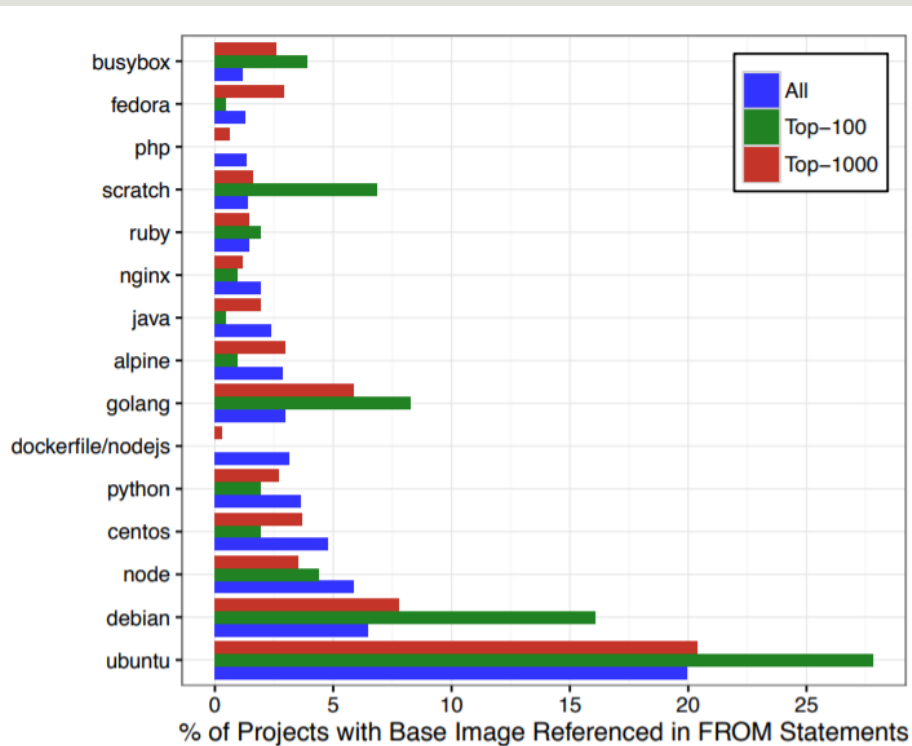


Fig. 5. Percentage of usage of 15 most commonly used base images

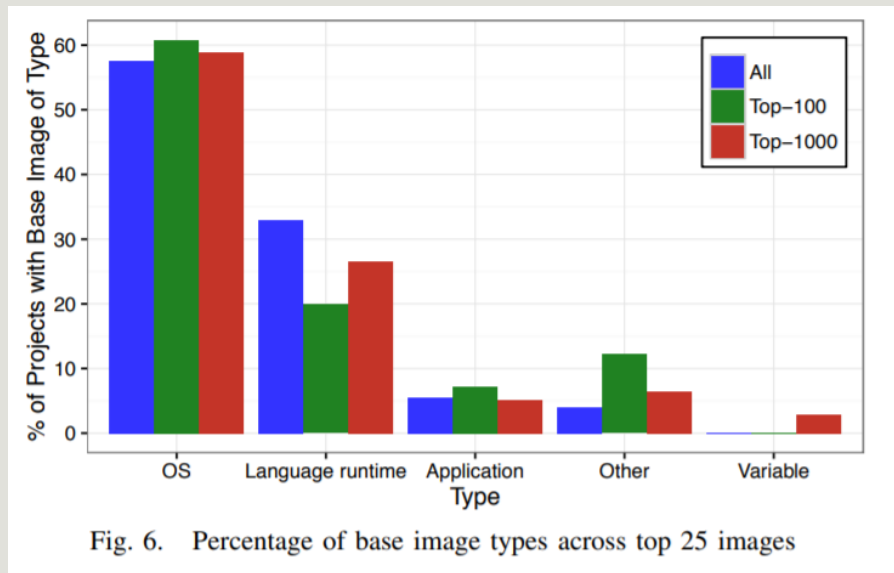
データセットのDockerfileには9298の一意のベースイメージが含まれていた

図は最も使用されるベースイメージの頻度の分布であり, 上位15のイメージのうち, 14が公式のものであった.

9298のイメージ中, 公式のものは130(1.4%)だけであった.

Docker Ecosystem

ベースイメージの要約



全体のデータ, Top100, Top1000で使用される上位25のイメージを分類.
最も使用される25のイメージが全てのイメージの使用量の64%を占めている.
図はそれら25のイメージをタイプ別に分類したものである.

Docker Ecosystem

Instruction(命令)

TABLE I
PERCENTAGES OF INSTRUCTIONS AND REPOSITORIES USING
INSTRUCTIONS AT LEAST ONCE

Instruction	Instructions			Repos using instructions (at least once)		
	All	T-1000	T-100	All	T-1000	T-100
RUN	0.40	0.41	0.48	0.87	0.87	0.86
COMMENT	0.16	0.14	0.15	0.54	0.51	0.55
ENV	0.06	0.07	0.09	0.42	0.44	0.47
FROM	0.07	0.08	0.07	0.97	0.96	0.94
ADD	0.06	0.05	0.02	0.46	0.48	0.37
CMD	0.04	0.04	0.03	0.56	0.53	0.47
COPY	0.03	0.04	0.03	0.29	0.32	0.32
EXPOSE	0.04	0.04	0.03	0.45	0.43	0.42
MAINTAINER	0.04	0.04	0.03	0.55	0.42	0.45
WORKDIR	0.03	0.03	0.03	0.45	0.53	0.57
ENTRYPOINT	0.02	0.02	0.01	0.23	0.31	0.27
VOLUME	0.02	0.02	0.01	0.17	0.19	0.16
USER	0.01	0.01	0.01	0.10	0.10	0.08

Dockerfileで最もよく使用される命令はRUNであった。RUNとCOMMENTを合わせると全体で56%を占めている。

Docker Ecosystem

RUNで実行されるコマンドの分類

TABLE II
BREAKDOWN OF RUN INSTRUCTIONS IN SIX CATEGORIES

Category	Examples	Instructions		
		All	T1000	T100
Dependencies	apt-get, yum, npm, pip, mvn	0.452	0.447	0.452
Filesystem	mkdir, rm, cd, cp, touch, ln	0.304	0.293	0.294
Permissions	chmod, chown, useradd	0.073	0.052	0.023
Build/Execute	make, python, service, install	0.053	0.083	0.135
Environment	set, export, source, virtualenv	0.006	0.01	0.002
Other		0.113	0.115	0.094

Docker Ecosystem

- ・ほとんどのDockerfileがOSをベースイメージとして使用していた.それらのOSイメージのうち, DebianやUbuntuなどの確立されたシステムはサイズがとても大きい(約125MB)が, Alpineなどは4MBほどしかない.
- ・同じアプリケーションをサイズの小さいベースイメージで実行することで全体のサイズを縮小し, ビルド時間も短縮できる.

Dockerfileの品質と標準準拠

Docker Linterの結果を分析することで
Dockerfileの品質を評価

データセット内のすべてのDockerfileに対し
てリンターを実行し, 報告されたルール違反
を調査

また, 正常にビルドされたDockerfileの数も
特定(560のサンプルで調査)

Dockerfileの品質と標準準拠

最も違反されていたルール

TABLE III

12 MOST VIOLATED RULES ACROSS ALL REPOSITORIES, TOP 100 AND TOP 1000 REPOSITORIES BASED ON THEIR STAR RATING, AND A RANDOM SAMPLE OF 560 BUILT DOCKERFILES.

	others	apt-get upgrade (DL3005)	pip version pinning (DL3013)	maintainer email (DL3012)	image :latest (DL3007)	use WORKDIR (DL3003)	double quote (SC2086)	img. version pinning (DL3006)	delete apt-get lists (DL3009)	avoid add. packages (DL3015)	copy instead of add (DL3020)	apt-get vers. pinning (DL3008)	maintainer missing (DL4000)
All (n=33670)	8.8%	1.3%	2.0%	2.2%	3.4%	4.2%	4.3%	9.7%	10.0%	11.9%	12.8%	13.5%	15.9%
Top-100	9.5%	0.6%	4.8%	1.3%	3.2%	7.1%	4.8%	4.3%	11.1%	12.2%	5.0%	14.5%	21.7%
Top-1000	7.9%	0.9%	2.2%	1.6%	2.9%	5.5%	5.3%	7.7%	9.1%	11.1%	12.2%	13.8%	19.8%
Build Failure (n=203)	10.8%	0.4%	2.3%	1.7%	3.1%	4.7%	3.5%	14.9%	7.6%	10.8%	17.5%	11.4%	11.4%
Build Success (n=357)	6.6%	0.7%	1.9%	1.7%	4.1%	3.7%	6.4%	7.7%	10.0%	13.1%	13.6%	14.8%	15.6%

表は最も違反の多い12のルールを表している。

バージョンの指定を行っていないのは重大なルール違反であるが、メンテナー情報の欠落などはそれほど重要ではない。

Dockerfileの品質と標準準拠

- ・全体では平均して3.1, Top100では3.2, Top1000では3.5のルールに違反している.
- ・バージョンの指定に関して, ベースイメージやインストールするパッケージの具体的なバージョンが存在しないことは, ビルドの失敗やコンテナの実行時の障害につながる.

Dockerfileの品質と標準準拠

- ・メンテナ情報がない違反に関して, 全体では15.9%がTop100では21.7%, Top1000では19.8%がこのルールに違反していた.

人気のあるプロジェクトでは多くの開発者が貢献するため, メンテナンスを個々の開発者に割り当てられないことからこのような結果になったと考えられる.

Dockerfileの品質と標準準拠

ビルド分析

サンプルの540のDockerfileのうち357のイメージを正常に作成できた.これはGitHub上のRubyベース(72.7%)やJavaベース(82.4%)と比べ, Dockerは66%と低くなっている.

TABLE IV
BUILD TIME STATISTICS OF FAILED AND SUCCESSFUL DOCKERFILE
BUILDS IN SECONDS.

	Failed	Successful
%	0.34	0.66
mean	90.5	145.9
median	24.7	76.7
min	0.0	0.7
max	1539.0	2742.0
sd	197.4	264.0

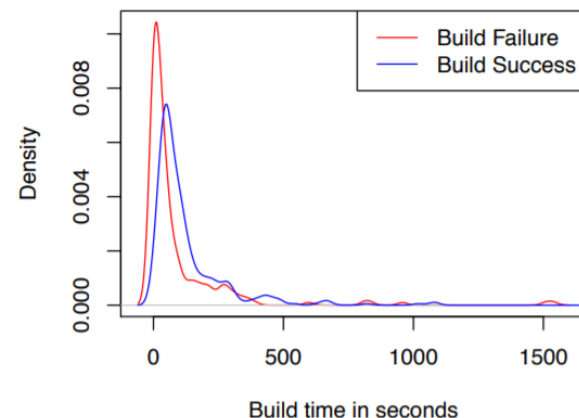


Fig. 7. Build time density

保守と進化

年間の変更の分布

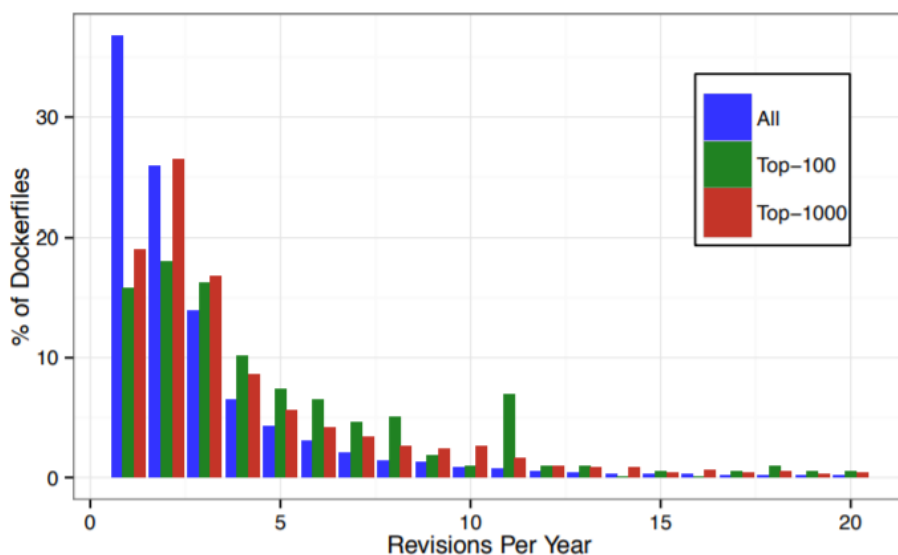


Fig. 8. Distribution of yearly revisions

全体ではDockerfileは年間平均3.11回変更されているが、Top100およびTop1000ではより頻繁に変更され、それぞれ5.81および4.70回であった。

人気のあるプロジェクトはより積極的に保守が行われていると考えられる。

保守と進化

変更のサイズ

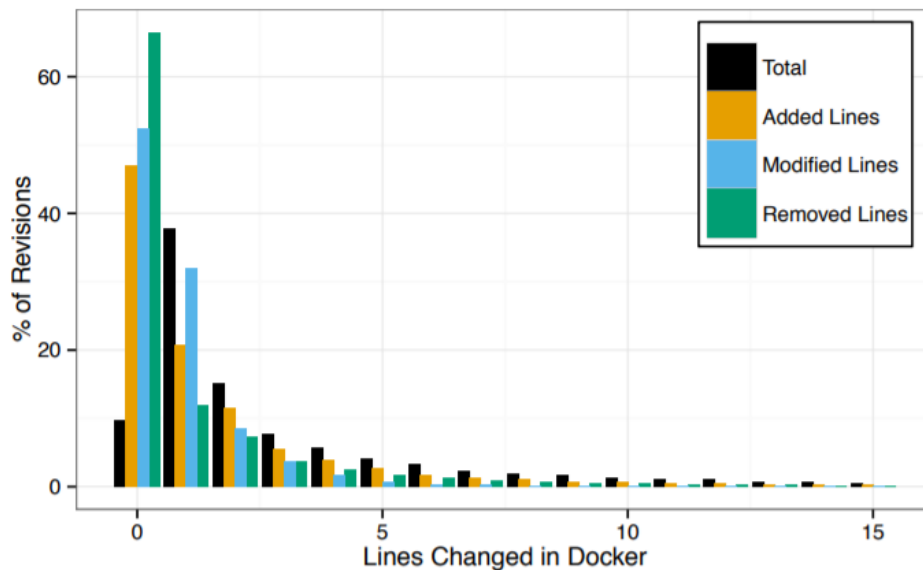


Fig. 9. Distribution of revision sizes measured in lines changed

全体のデータとTop100, Top1000のデータでは大きな違いが見られなかったため全体のデータのみプロットしている.

平均で3.98行の変更が行われており, ほとんどは命令の追加(47.44%), その後の命令の削除(33.17%), および変更(19.40%)であった

保守と進化

変更の性質

TABLE V
RELATIVE CHANGES OF ALL DOCKER INSTRUCTION TYPES.

	All				Top-1000				Top-100			
	All	Add	Mod	Rem	All	Add	Mod	Rem	All	Add	Mod	Rem
RUN	0.53	0.59	0.29	0.61	0.55	0.64	0.25	0.64	0.58	0.66	0.31	0.62
Dependencies	0.48	0.5	0.34	0.49	0.49	0.51	0.35	0.5	0.51	0.52	0.44	0.51
Filesystem	0.26	0.24	0.42	0.25	0.24	0.23	0.41	0.22	0.24	0.24	0.37	0.2
Permissions	0.06	0.06	0.07	0.06	0.05	0.06	0.04	0.05	0.03	0.03	0.0	0.04
Build/Execute	0.04	0.04	0.03	0.05	0.06	0.05	0.05	0.07	0.1	0.09	0.06	0.14
Environment	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.0	0.01	0.01	0.01	0.0
COMMENT	0.12	0.12	0.11	0.12	0.09	0.09	0.10	0.09	0.10	0.10	0.10	0.09
ENV	0.07	0.06	0.13	0.05	0.07	0.05	0.16	0.05	0.10	0.06	0.16	0.12
FROM	0.06	0.00	0.29	0.00	0.07	0.00	0.31	0.00	0.00	0.04	0.26	0.00
ADD	0.05	0.05	0.05	0.06	0.06	0.05	0.06	0.07	0.02	0.02	0.01	0.03
CMD	0.05	0.04	0.06	0.05	0.04	0.04	0.04	0.04	0.03	0.03	0.04	0.02
COPY	0.03	0.04	0.03	0.03	0.03	0.04	0.03	0.03	0.04	0.05	0.03	0.04
EXPOSE	0.02	0.02	0.01	0.02	0.01	0.02	0.01	0.01	0.02	0.02	0.01	0.01
MAINTAINER	0.01	0.01	0.02	0.00	0.01	0.00	0.04	0.00	0.01	0.00	0.02	0.01
WORKDIR	0.01	0.02	0.00	0.02	0.01	0.01	0.00	0.01	0.01	0.02	0.00	0.02
ENTRYPOINT	0.02	0.01	0.02	0.02	0.01	0.01	0.02	0.01	0.03	0.01	0.04	0.03

保守と進化

変更の性質

全ての変更の53%はRUN命令に関連している。RUN命令は全体の命令の40%しか占めていないため、RUN命令は頻繁に変更されることが分かる。

関連研究

- ・仮想化インフラストラクチャでコードを迅速に構築および展開する方法に関する研究が注目されており, ビルドシステムに関する研究(McIntosh et al)などがある.
- ・Dockerに特に関係する研究は, パフォーマンスの側面に焦点が当てられており, 多くはコンテナのパフォーマンスとオーバーヘッドを従来の仮想化技術と比較している.

結論

- ・多くのコンテナは重いOSをベースイメージとして継承しているので、コンテナ技術の利点である軽さを活かせていない。
- ・品質の問題の28.6%はバージョンの指定が行われていないことであった。これを改善することで将来の再現性と安定したビルドにつながる可能性がある。

所感

- ・興味のあるテーマだったので読みやすかった.
- ・Dockerに関して、ベースイメージについてなどコンテナの構成要素やその特徴について知ることができた.
- ・Dockerfile内の命令について知ることができた.