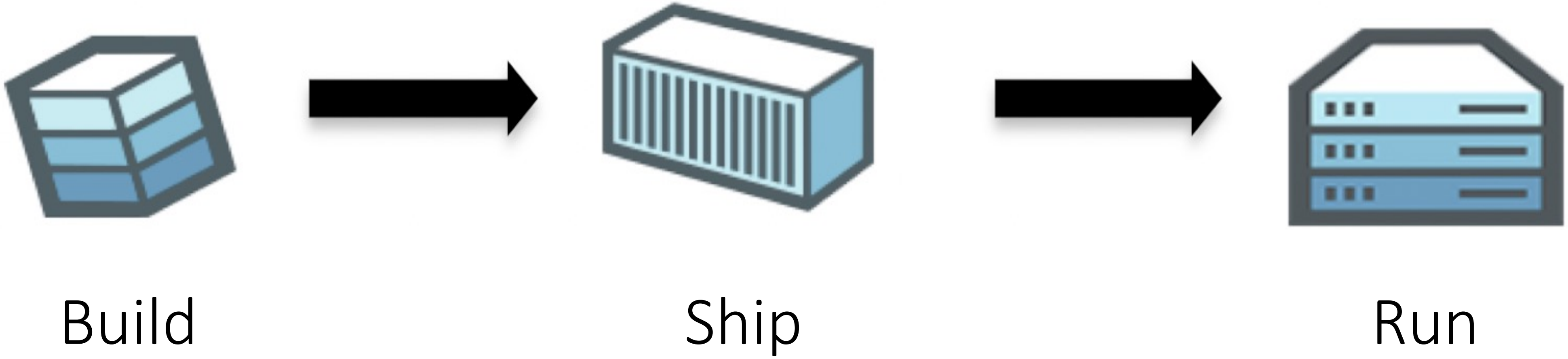
Fixing Dockerfile Smells: An Empirical Study

Giovanni Rosa, Simone Scalabrino and Rocco Oliveto

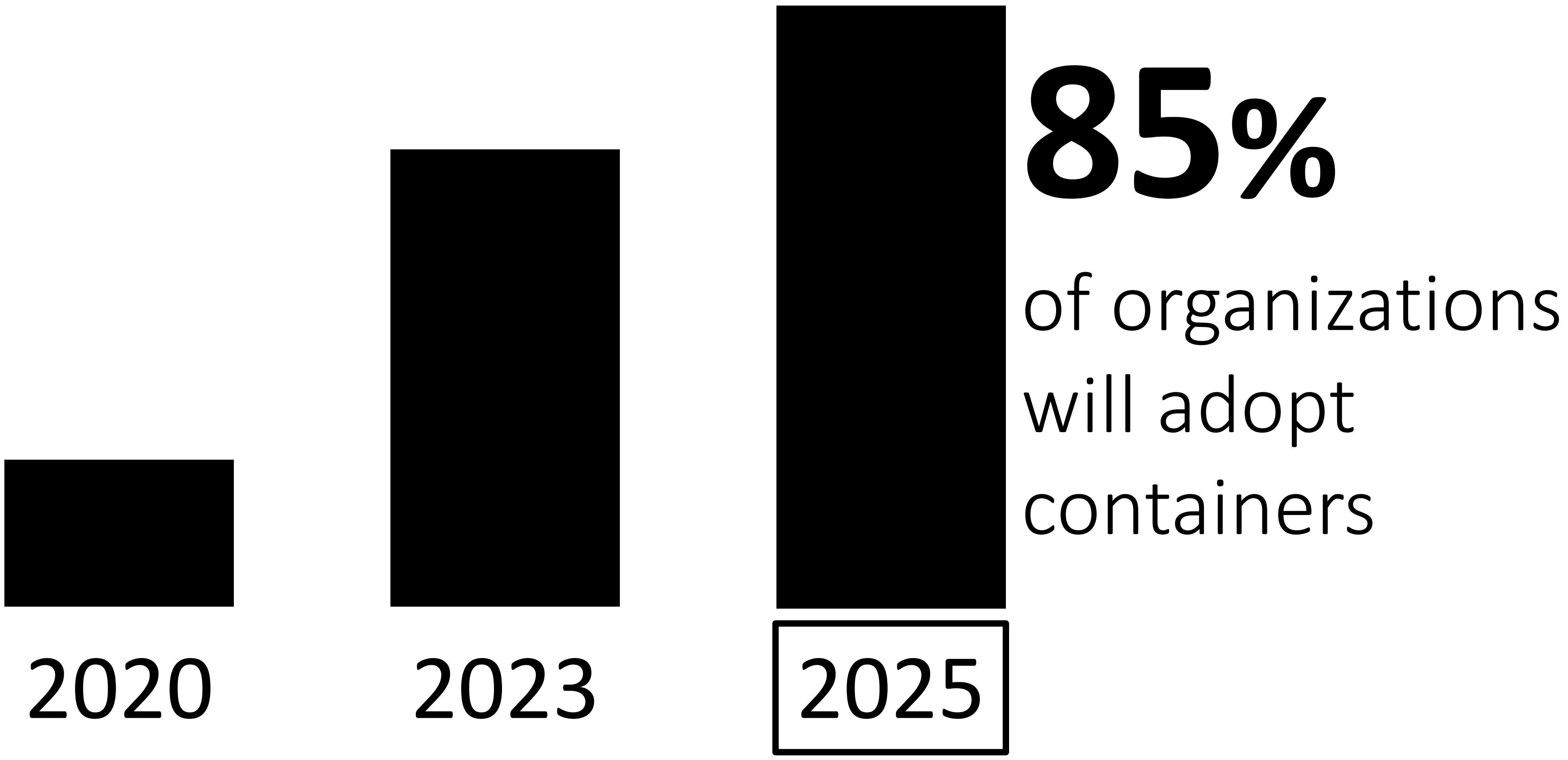
University of Molise, Italy

ICSME '22 Registered Reports - Oct 6th 2022
Limassol, Cyprus

ICSME 2022

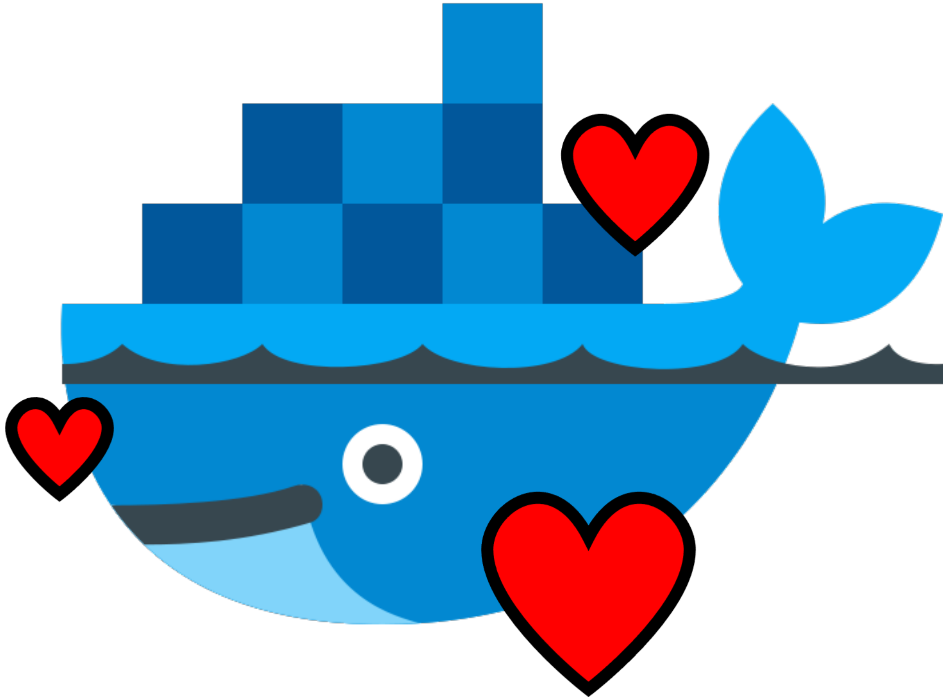


Software containers



Gartner

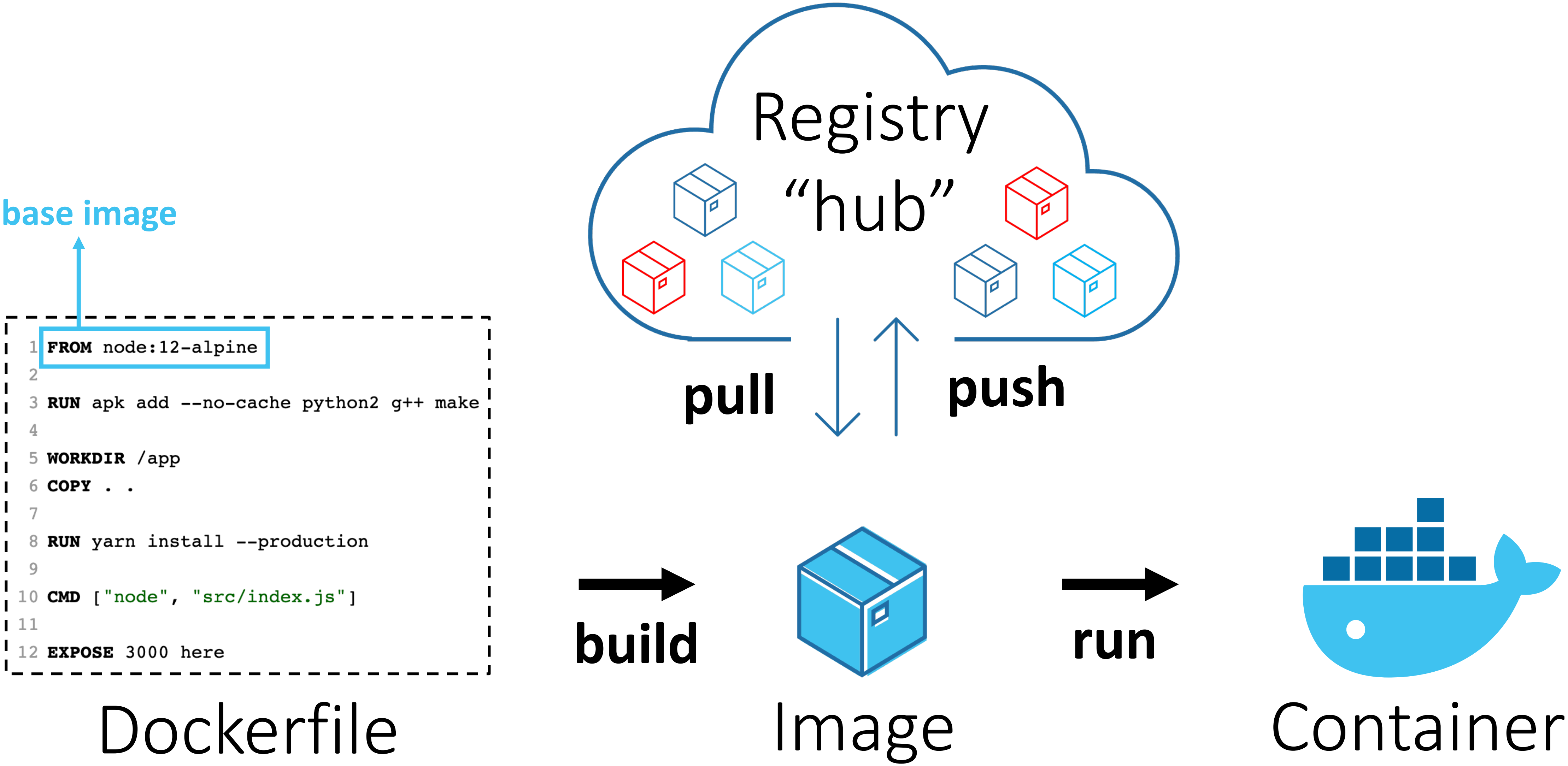
Containers in production environments



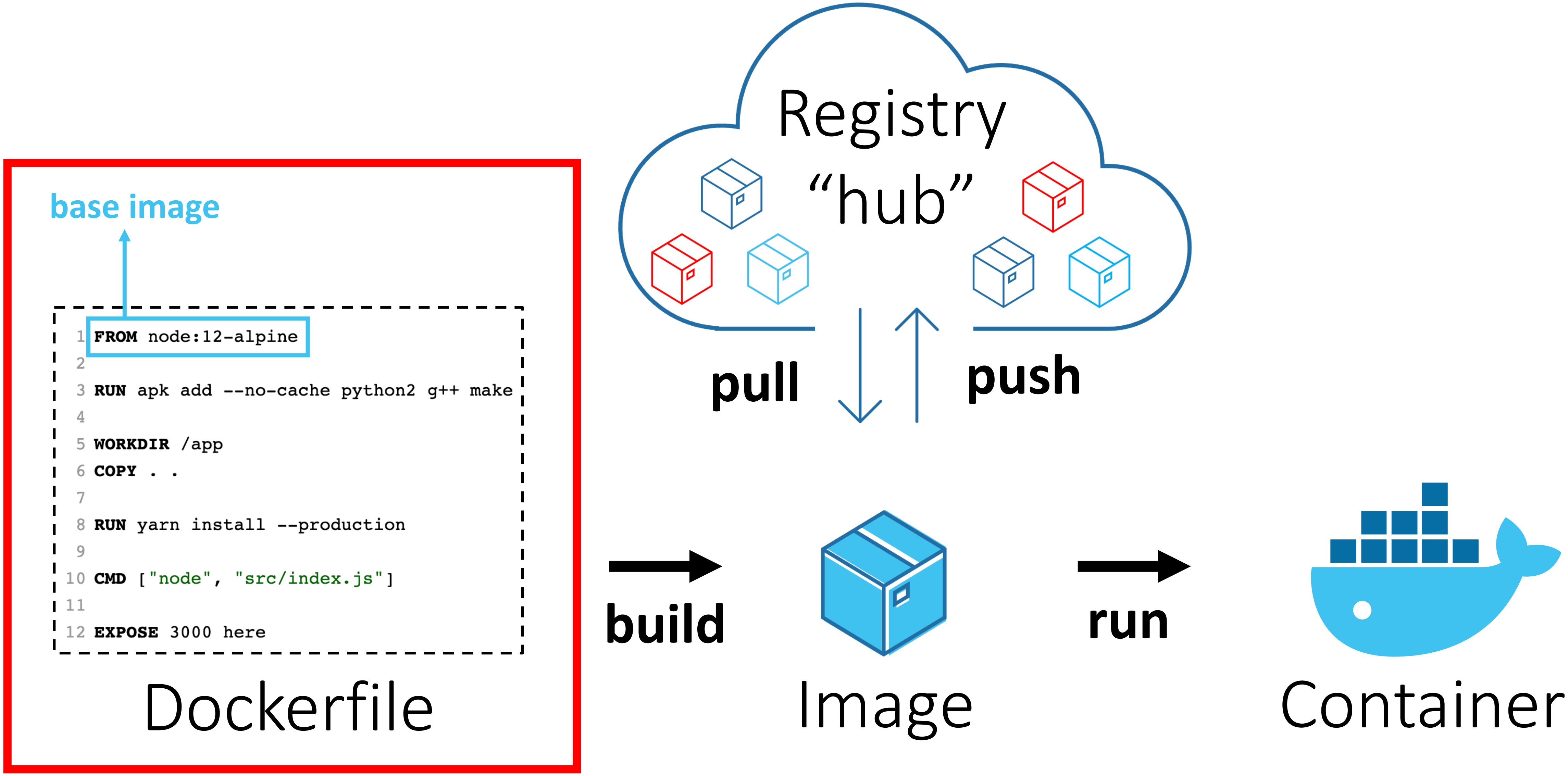
#1 Most-Wanted
and
#1 Most Loved
tool

2022 Developer Survey

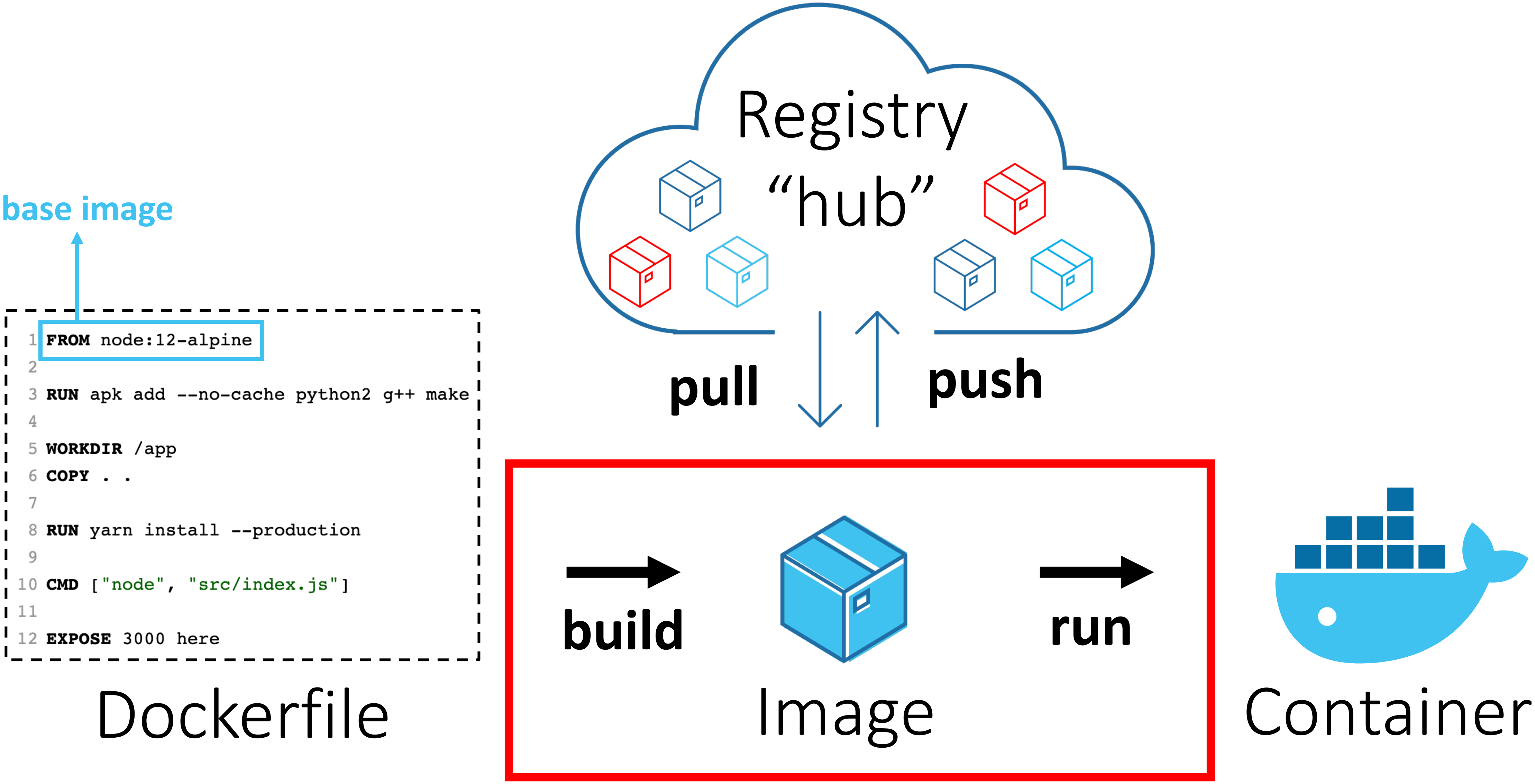
Why Docker?



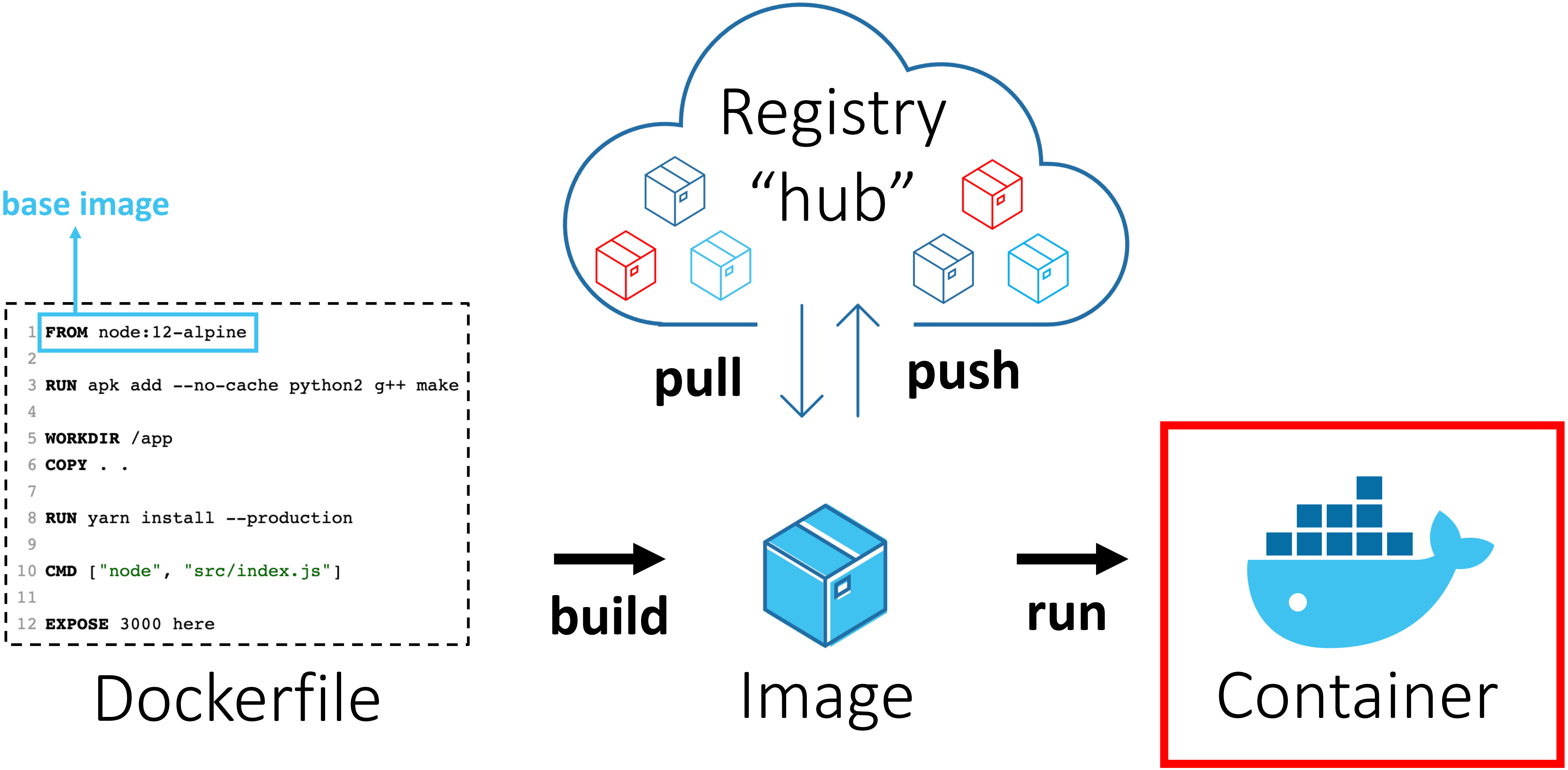
Docker in a nutshell



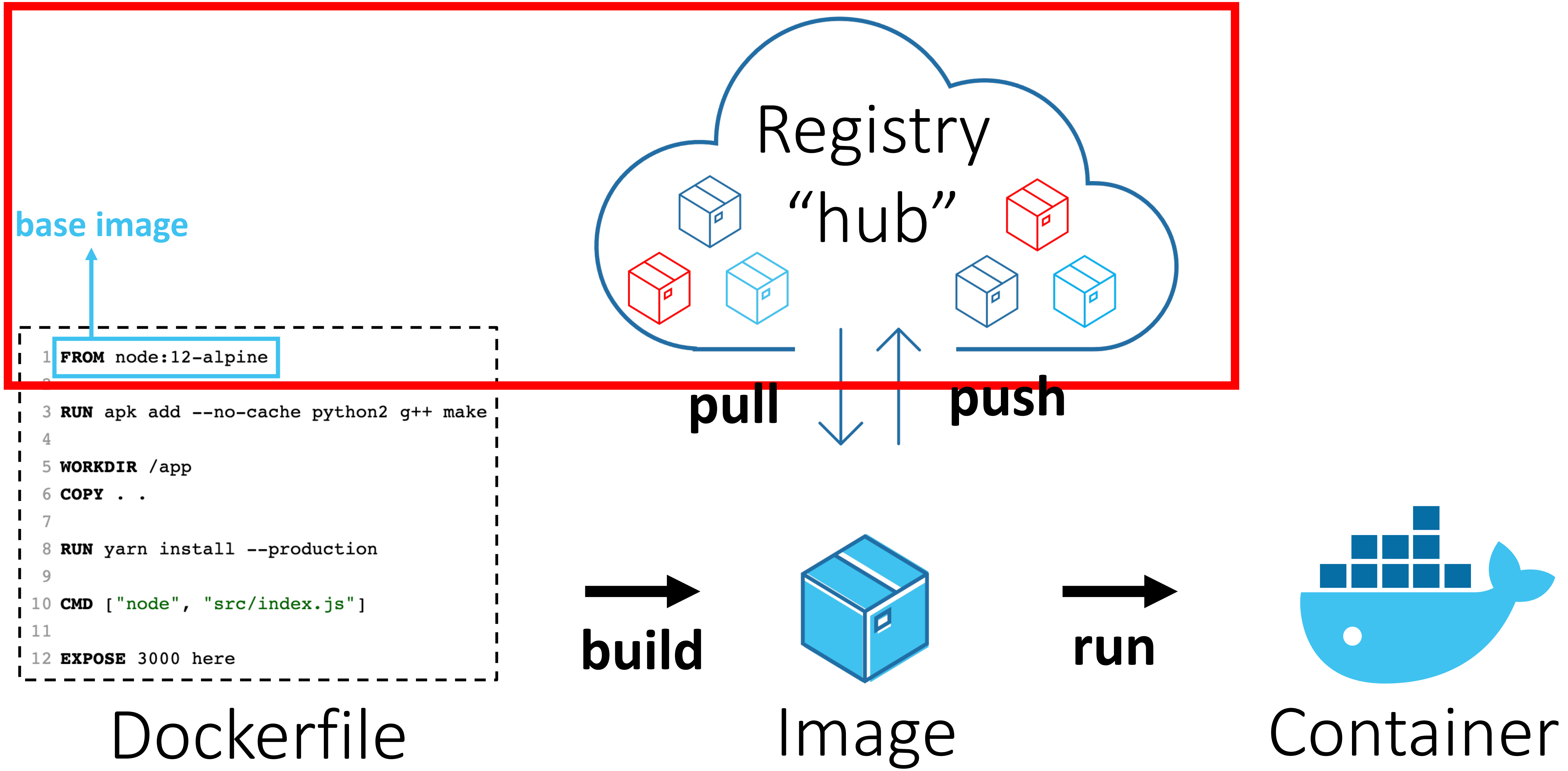
Docker in a nutshell



Docker in a nutshell



Docker in a nutshell



Docker in a nutshell

How to measure Dockerfile quality?

How to measure Dockerfile quality?

Haskell

Dockerfile **Lint**er

```
FROM python:latest

MAINTAINER Mark Red <mark.red@example.com>

RUN apt-get update -y && \
    apt-get install -y default-jdk

RUN python -m pip install --upgrade pip

WORKDIR /app

COPY requirements.txt ./

RUN pip install -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "-m", "server"]
```

DL3007: Pin the version explicitly to a release tag

DL4000: MAINTAINER is deprecated

DL3008: Pin versions in apt get install.

DL3009: Delete the apt-get lists after installing something

DL3059: Multiple consecutive 'RUN' instructions

DL3013: Pin versions in pip

DL3042: Avoid use of cache directory with pip

DL3042: Avoid use of cache directory with pip

Smells in Dockerfiles

Image size

Build reliability

Security vulnerabilities

Smells can **negatively** affect Docker images

2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)

An Empirical Analysis of the Docker Container Ecosystem on GitHub

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

* Software Evolution and Architecture Lab
University of Zurich, Switzerland
{lastname}@ifi.uzh.ch

† IBM T. J. Watson Research Center
Yorktown Heights, NY, USA
witternj@us.ibm.com

***Abstract*—Docker allows packaging an application with its dependencies into a standardized, self-contained unit (a so-called container), which can be used for software development and to run the application on any system. Dockerfiles are declarative definitions of an environment that aim to enable reproducible builds of the container. They can often be found in source code repositories and enable the hosted software to come to life in its execution environment. We conduct an exploratory empirical study with the goal of characterizing the Docker ecosystem, prevalent quality issues, and the evolution of Dockerfiles. We base our study on a data set of over 70000 Dockerfiles, and contrast this general population with samplings that contain the Top-100 and Top-1000 most popular Docker-using projects. We find that most quality issues (28.6%) arise from missing version pinning (i.e., specifying a concrete version for dependencies). Further, we were not able to build 34% of Dockerfiles from a representative sample of 560 projects. Integrating quality checks, e.g., to issue version pinning warnings, into the container build process could result into more reproducible builds. The most popular projects change more often than the rest of the Docker population, with 5.81 revisions per year and 5 lines of code changed on average. Most changes deal with dependencies, that are currently stored in a rather unstructured manner. We propose to introduce an abstraction that, for instance, could deal with the intricacies of different package managers and could improve migration to more light-weight images.**

***Keywords*-empirical software engineering; GitHub; Docker**

I. Introduction

Containerization has recently gained interest as a light-weight virtualization technology to define software infrastructure. Containers allow to package an application with its dependencies and execution environment into a standardized, self-contained unit, which can be used for software development and to run the application on any system. Due to their rapid spread in the software development community, Docker containers have become the de-facto standard format [1]. The contents of a Docker container are declaratively defined in a *Dockerfile* that stores instructions to reach a certain infrastructure state [2], following the notion of Infrastructure-as-Code (IaC) [3]. Source code repositories containing Dockerfiles, thus, potentially enable the execution of program code in an isolated and fast environment with one command. Since its inception in 2013, repositories on GitHub have added 70197 Dockerfiles to their projects (until October 2016).

Given the fast rise in popularity, its ubiquitous nature in industry, and its surrounding claim of enabling reproducibility [4], we study the Docker ecosystem with respect to quality of Dockerfiles and their change and evolution behavior within software repositories. We developed a tool chain that transforms Dockerfiles and their evolution in Git repositories into a relational database model. We mined the entire population of Dockerfiles on GitHub as of October 2016, and summarize our findings on the ecosystem in general, quality aspects, and evolution behavior. The results of our study can inform standard bodies around containers and tool developers to develop better support to improve quality and drive ecosystem change.

We make the following contributions through our exploratory study:

Ecosystem Overview. We characterize the ecosystem of Docker containers on GitHub by analyzing the distribution of projects using Docker, broken down by primary programming language, project size, and the base infrastructure (*base image*) they inherit from. We learn, among other things, that most inherited base images are well-established, but heavy-weight operating systems, while light-weight alternatives are in the minority. However, this defeats the purpose of containers to lower the footprint of virtualization. We envision a recommendation system that analyzes Dockerfiles and transforms its dependency sources to work with light-weight base images.

Quality Assessment. We assess the quality of Dockerfiles on GitHub by classifying results of a Dockerfile Linter [5]. Most of the issues we encountered considered version pinning (i.e., specifying a concrete version for either base images or dependencies), accounting for 28.6% of quality issues. We also built the Dockerfiles for a representative sample of 560 repositories. 66% of Dockerfiles could be built successfully with an average build time of 145.9 seconds. Integrating quality checks into the "docker build" process to warn developers early about build-breaking issues, such as version pinning, can lead to more reproducible builds.

Evolution Behavior. We classify different kinds of changes between consecutive versions of Dockerfiles to characterize their evolution within a repository. On average, Dockerfiles only changed 3.11 times per year, with a mean 3.98 lines of code changed per revision. However, more popular projects revise up to 5.81 per year with 5 lines changed. Dependencies see a high rate of change over time, reinforcing our findings to improve dependency handling from the analysis of the

978-1-5386-1544-7/17 $31.00 © 2017 IEEE

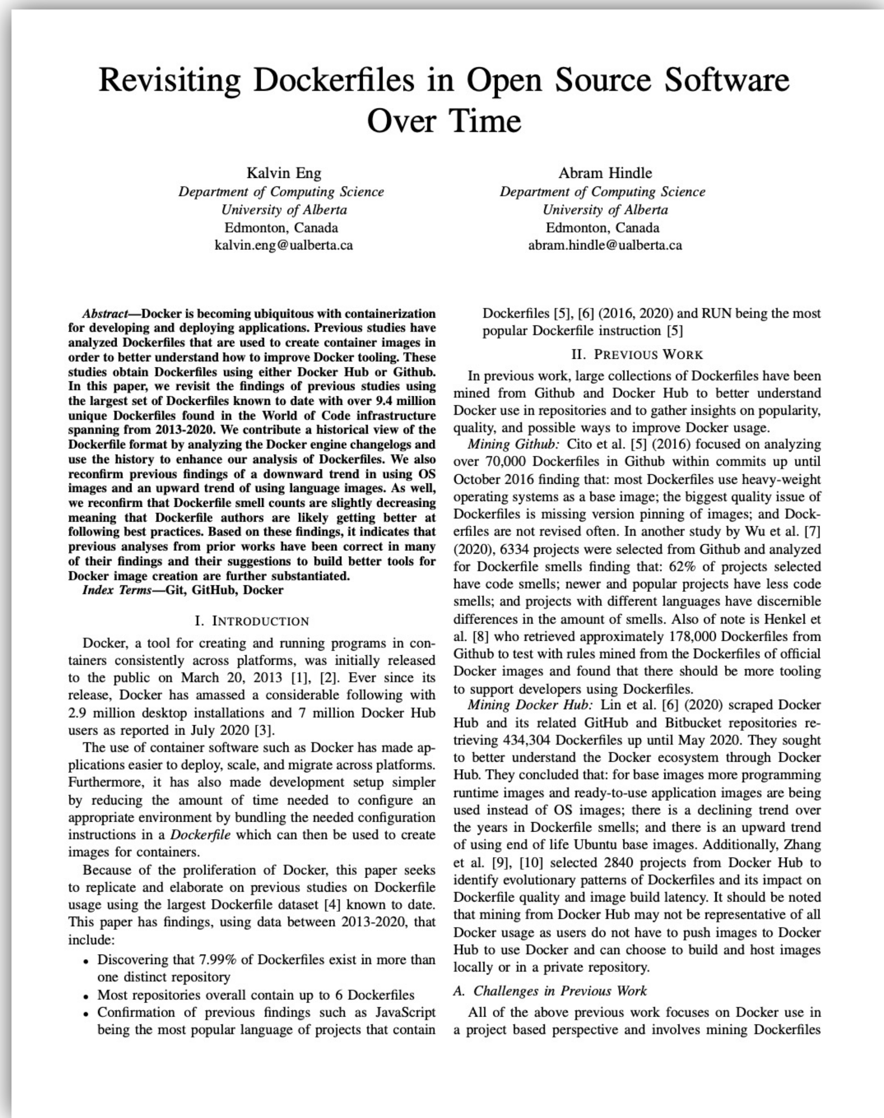
DOI 10.1109/MSR.2017.67

Cito et. al

84%

of 70000 Dockerfiles contain **smells**

2017



Revisiting Dockerfiles in Open Source Software Over Time

Kalvin Eng
Department of Computing Science
University of Alberta
Edmonton, Canada
kalvin.eng@ualberta.ca

Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Canada
abram.hindle@ualberta.ca

***Abstract*—Docker is becoming ubiquitous with containerization for developing and deploying applications. Previous studies have analyzed Dockerfiles that are used to create container images in order to better understand how to improve Docker tooling. These studies obtain Dockerfiles using either Docker Hub or Github. In this paper, we revisit the findings of previous studies using the largest set of Dockerfiles known to date with over 9.4 million unique Dockerfiles found in the World of Code infrastructure spanning from 2013-2020. We contribute a historical view of the Dockerfile format by analyzing the Docker engine changelogs and use the history to enhance our analysis of Dockerfiles. We also reconfirm previous findings of a downward trend in using OS images and an upward trend of using language images. As well, we reconfirm that Dockerfile smell counts are slightly decreasing meaning that Dockerfile authors are likely getting better at following best practices. Based on these findings, it indicates that previous analyses from prior works have been correct in many of their findings and their suggestions to build better tools for Docker image creation are further substantiated.**

***Index Terms*—Git, GitHub, Docker**

I. Introduction

Docker, a tool for creating and running programs in containers consistently across platforms, was initially released to the public on March 20, 2013 [1], [2]. Ever since its release, Docker has amassed a considerable following with 2.9 million desktop installations and 7 million Docker Hub users as reported in July 2020 [3].

The use of container software such as Docker has made applications easier to deploy, scale, and migrate across platforms. Furthermore, it has also made development setup simpler by reducing the amount of time needed to configure an appropriate environment by bundling the needed configuration instructions in a *Dockerfile* which can then be used to create images for containers.

Because of the proliferation of Docker, this paper seeks to replicate and elaborate on previous studies on Dockerfile usage using the largest Dockerfile dataset [4] known to date. This paper has findings, using data between 2013-2020, that include:

- Discovering that 7.99% of Dockerfiles exist in more than one distinct repository
- Most repositories overall contain up to 6 Dockerfiles
- Confirmation of previous findings such as JavaScript being the most popular language of projects that contain Dockerfiles [5], [6] (2016, 2020) and RUN being the most popular Dockerfile instruction [5]

II. Previous Work

In previous work, large collections of Dockerfiles have been mined from Github and Docker Hub to better understand Docker use in repositories and to gather insights on popularity, quality, and possible ways to improve Docker usage.

Mining Github: Cito et al. [5] (2016) focused on analyzing over 70,000 Dockerfiles in Github within commits up until October 2016 finding that: most Dockerfiles use heavy-weight operating systems as a base image; the biggest quality issue of Dockerfiles is missing version pinning of images; and Dockerfiles are not revised often. In another study by Wu et al. [7] (2020), 6334 projects were selected from Github and analyzed for Dockerfile smells finding that: 62% of projects selected have code smells; newer and popular projects have less code smells; and projects with different languages have discernible differences in the amount of smells. Also of note is Henkel et al. [8] who retrieved approximately 178,000 Dockerfiles from Github to test with rules mined from the Dockerfiles of official Docker images and found that there should be more tooling to support developers using Dockerfiles.

Mining Docker Hub: Lin et al. [6] (2020) scraped Docker Hub and its related GitHub and Bitbucket repositories retrieving 434,304 Dockerfiles up until May 2020. They sought to better understand the Docker ecosystem through Docker Hub. They concluded that: for base images more programming runtime images and ready-to-use application images are being used instead of OS images; there is a declining trend over the years in Dockerfile smells; and there is an upward trend of using end of life Ubuntu base images. Additionally, Zhang et al. [9], [10] selected 2840 projects from Docker Hub to identify evolutionary patterns of Dockerfiles and its impact on Dockerfile quality and image build latency. It should be noted that mining from Docker Hub may not be representative of all Docker usage as users do not have to push images to Docker Hub to use Docker and can choose to build and host images locally or in a private repository.

A. Challenges in Previous Work

All of the above previous work focuses on Docker use in a project based perspective and involves mining Dockerfiles

Eng et. al

"**Version pinning** smell
is the biggest quality issue
[...]
there is a **declining** trend of
Dockerfile smells"

2021

It is not clear what smells are relevant and need to be fixed

- It is not clear what smells are relevant and need to be fixed
- Lack of advanced supporting tools for developers

Empirical Study

Empirical Study

Smell survivability

Empirical Study

Smell survivability

Fix recommendations

RQ 1

How do developers fix Dockerfile smells?

RQ1

Dockerfile snapshots over time

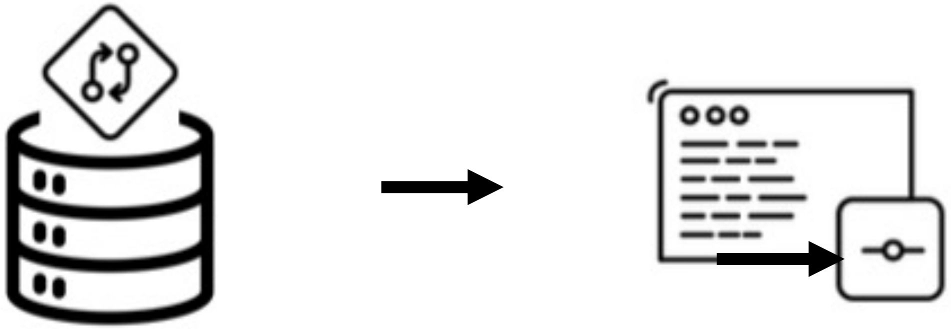
→

9.4M

unique Dockerfiles

from **2013** to **2020**

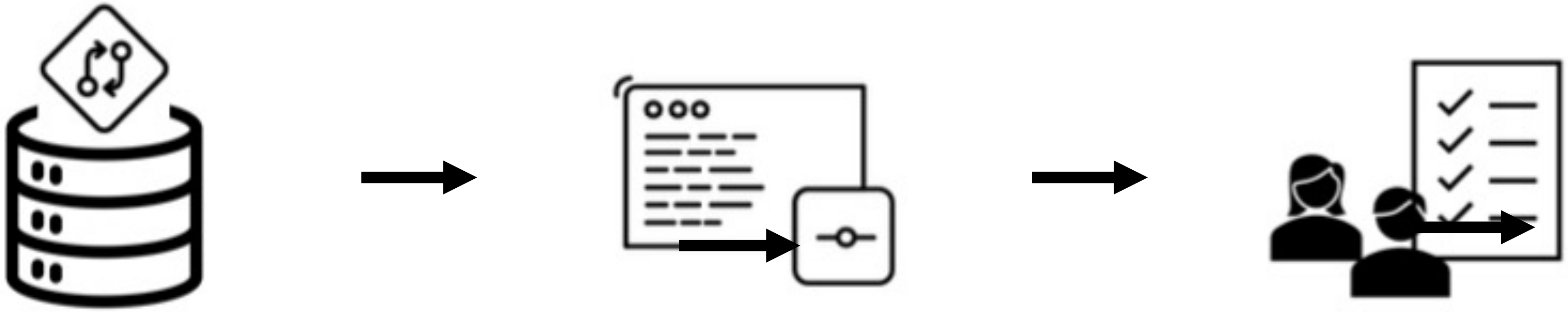
RQ1



Dockerfile snapshots over time

Extraction of smell-fixing commits

RQ1



Dockerfile snapshots over time

Extraction of smell-fixing commits

Manual validation (1000 commits)

Manual validation

```
RUN apt-get install -y \
    curl=7.* \
    git \
    && rm -rf /var/lib/apt/lists/*
```

smelly

```
RUN apt-get install -y \
    curl=7.* \

    && rm -rf /var/lib/apt/lists/*
```

not smelly

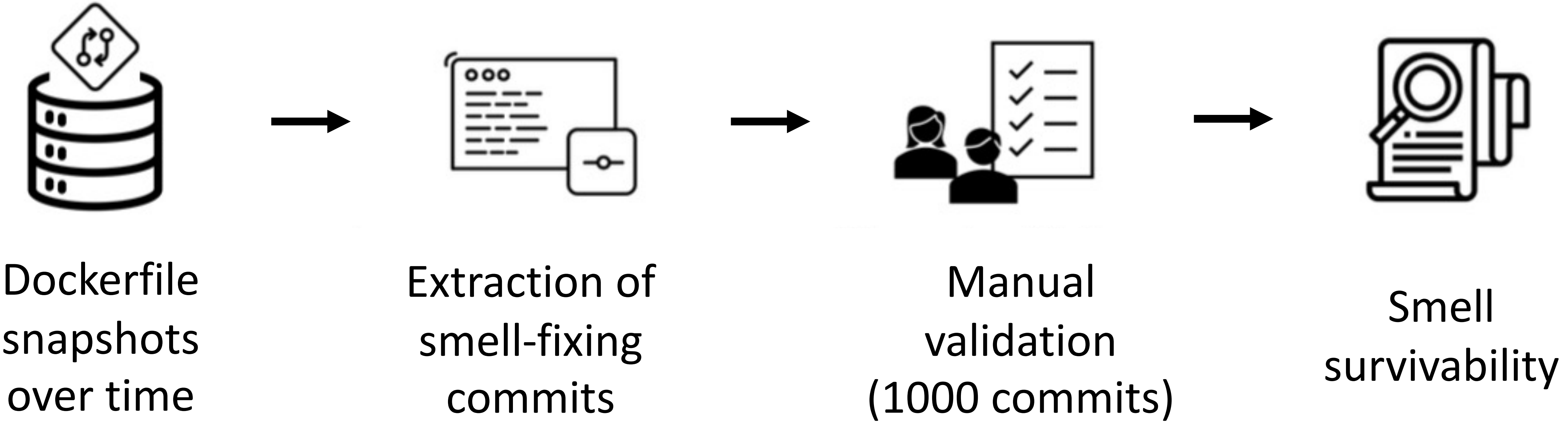
```
RUN apt-get install -y \
    curl=7.* \
    git \
    && rm -rf /var/lib/apt/lists/*
```

smelly

```
RUN apt-get install -y \
    curl=7.* \
    git=2.23 \
    && rm -rf /var/lib/apt/lists/*
```

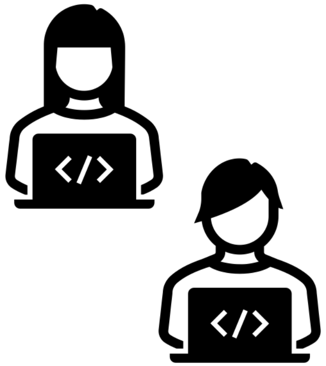
not smelly

RQ1


Dockerfile snapshots over time
Extraction of smell-fixing commits
Manual validation (1000 commits)
Smell survivability

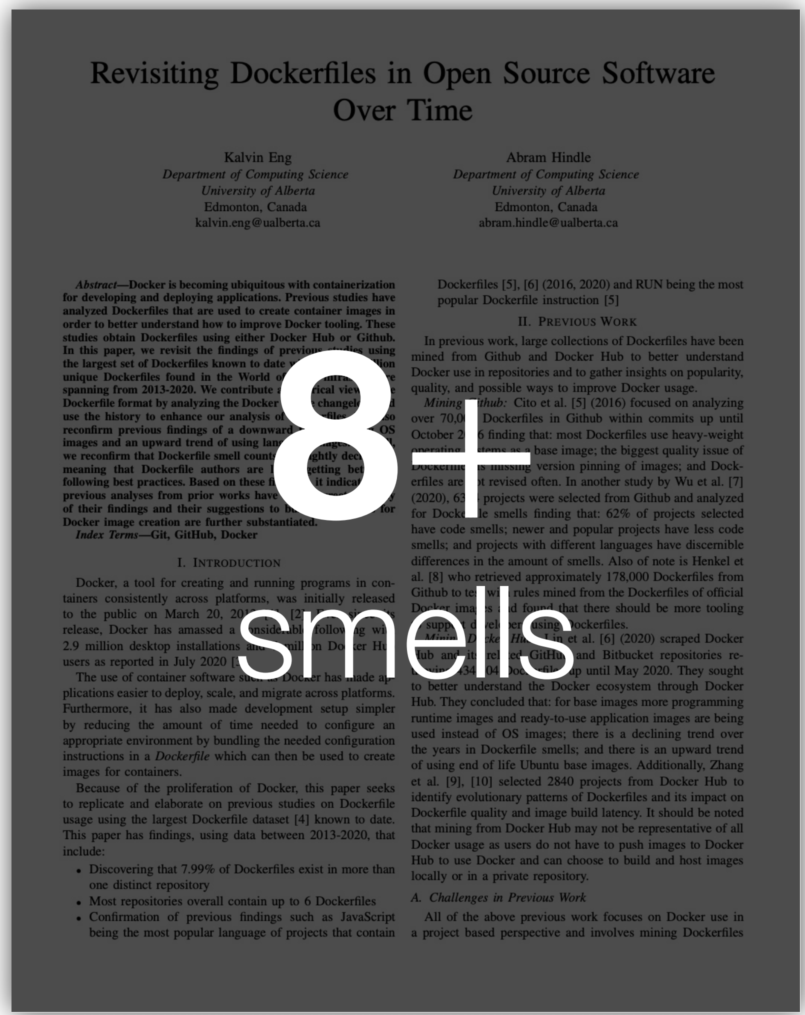
RQ 2

Which Dockerfile smells are developers willing to address?



RQ2

Most frequent and fixed smells



Revisiting Dockerfiles in Open Source Software Over Time

Kalvin Eng
Department of Computing Science
University of Alberta
Edmonton, Canada
kalvin.eng@ualberta.ca

Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Canada
abram.hindle@ualberta.ca

Eng et. al 2021

11.5M

commits

RQ2

Most frequent and fixed smells

→

Rule-based Refactoring tool

→

DL3006

José Lorenzo Rodríguez edited this page on 4 Feb 2018 · 5 revisions

Always tag the version of an image explicitly.

Problematic code:

```
FROM debian
```

Correct code:

```
FROM debian:jessie
```

DL3008

Jeroen de Bruijn edited this page on 20 Oct 2019 · 3 revisions

Pin versions in apt get install.

Problematic code:

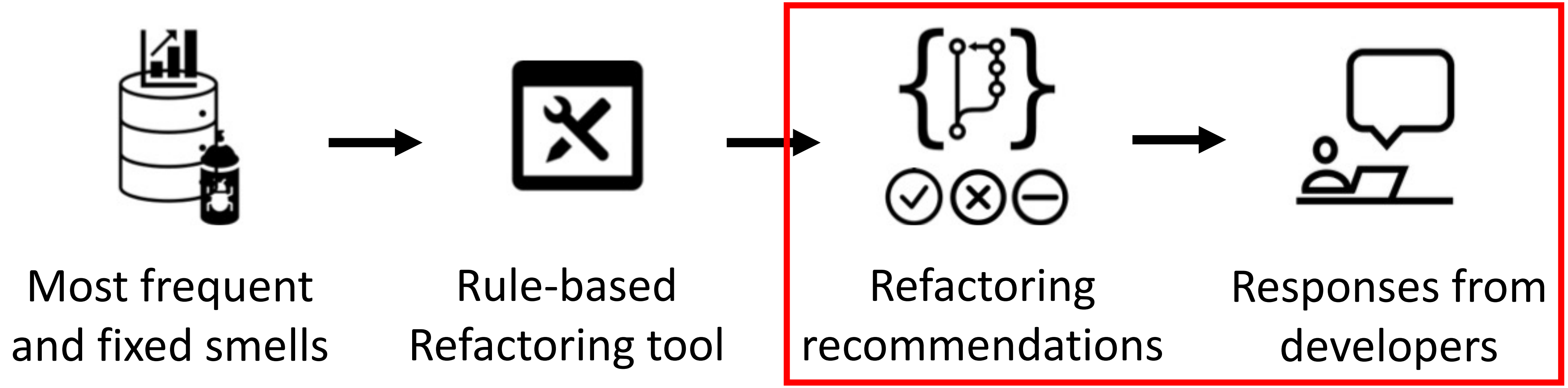
```
FROM busybox
RUN apt-get install python
```

Correct code:

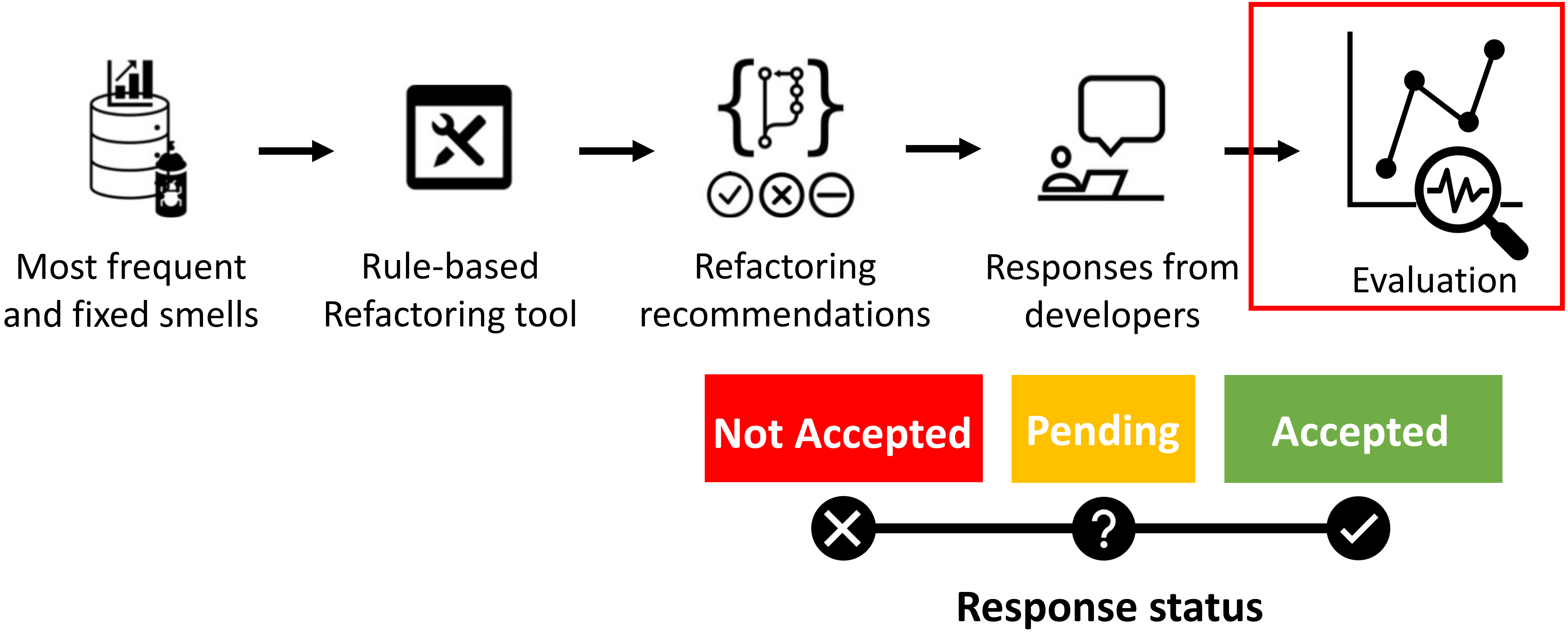
```
FROM busybox
RUN apt-get install python=2.7
```

hadolint wiki

RQ2


Most frequent and fixed smells
Rule-based Refactoring tool
Refactoring recommendations
Responses from developers

RQ2


Most frequent and fixed smells
Rule-based Refactoring tool
Refactoring recommendations
Responses from developers
Evaluation
Not Accepted
Pending
Accepted
Response status

Summary

How to measure Docker quality?

Haskell Dockerfile Linter

Image size

Build reliability

Security risks

Smells potentially affect Docker images in a **negative way**

- It is not clear what smells are relevant and need to be fixed
- Lack of advanced supporting tools for developers

Empirical Study

- Smell survivability
- **Fix recommendations**

RQ1

Dockerfile snapshots over time → Extraction of smell-fixing commits → Manual validation (1000 commits) → Smell survivability

RQ2

Most frequent and fixed smells → Rule-based Refactoring tool → Refactoring recommendations → Responses from developers → Evaluation

Not Accepted | Pending | Accepted

Response status

Giovanni Rosa

University of Molise, Italy

giovanni.rosa@unimol.it