Noname manuscript No.
(will be inserted by the editor)

Too many Images on DockerHub! How Different are Images for the same System?

Md Hasan Ibrahim · Mohammed Sayagh · Ahmed E Hassan

Received: date / Accepted: date

Abstract Containerization is a technique used to encapsulate a software system and its dependencies into one isolated package, which is called a container. The goal of these containers is to deploy or replicate a software system on various platforms and environments without facing any compatibility or dependency issues. Developers can instantiate these containers from images using Docker; one of the most popular containerization platforms. Furthermore, many of these images are publicly available on DockerHub, on which developers can share their images with the community who in turn can leverage such publicly available image. However, DockerHub contains thousands of images for each software system, which makes the selection of an image a nontrivial task. In this paper, we investigate the differences among DockerHub images for five software systems and 936 images with the goal of helping Docker tooling creators and DockerHub better guide users select a suitable image. We observe that users tend to download the official images (images that are provided by Docker itself) when there exist a large number of image choices for each single software system on the community images (images that are provided by the community developers), which are in many cases more resource efficient (have less duplicate resources) and have less security vulnerabilities. In fact, we observe that 27% (median), 35% (median), 6% (median), and 9% (median) of the DockerHub Debian, Centos, Ubuntu, and Alpine based images are identical to another image across all the studied software systems. Furthermore, 26% (median), 49% (median), and 8% (median) of the Alpine, Debian, and Ubuntu based community images are more resource efficient than their respective official images across all the five studied software systems. 7% (median) of the community Debian based images have less security vulnera-

Md Hasan Ibrahim · Mohammed Sayagh · Ahmed E Hassan
Software Analysis and Intelligence Lab (SAIL)
Queen's University
Kingston, ON, Canada
E-mail: {ibrahim.mdhasan, msayagh, ahmed}@cs.queensu.ca

bilities than their respective official images across the four studied software systems, for which an official Debian based image exists. Unfortunately, the description of 78% of the studied images do not guide users when selecting an image (the description does not exist at all or it does not highlight the particularities of the image), we suggest that Docker tooling creators and DockerHub design approaches to distinguish DockerHub images and help users find the most suitable images for their needs.

Keywords Docker · Docker images · DockerHub · Containerization

1 Introduction

One can package a software system with all of its dependencies and required libraries into one isolated container using Docker [15]. Docker is a platform used to instantiate Docker images on containers, which can be rapidly deployed on any environment without dealing with compatibility and dependency issues. Docker helps practitioners quickly ship a software system into production by reducing the deployment time by 10-15 times compared to the manual setup of the environments [24].

Docker is one of the most popular containerization technologies due to its portability, lightweight, and self-sufficiency [2, 21, 23, 26]. According to 451 Research [1], container technologies generated a revenue of $762 million in 2016 and are expected to reach a revenue of $2.7 billion by 2020 [25], with Docker accounting for 83% of all this revenue [6]. Moreover, Datadog [10] reports that around one-quarter of their customers already adopted Docker for deploying their applications on the cloud [11].

DockerHub is the *"world's leading service for finding and sharing container images with [one's] team and the Docker community"* [18]. It is a cloud-based image registry where developers can share their images either privately with a selective group of users, or publicly with the whole community of DockerHub users. DockerHub contains a rich database of images, from which one can reuse an existing image instead of building one from scratch.

However, a large number of images exists for installing the same software system making the task of choosing an image a not so straightforward task. For example, one can obtain 57,000 images when searching for an image related to the Nginx software system in DockerHub, hence she may end up choosing an image which is not the most suitable to fulfill their needs when other images could be more suitable. Moreover, users do not always choose just a single image, they might often need to choose and connect multiple images together to build a distributed software installation or a stack of layers such as the MEAN stack, which is composed of the following software systems: MongoDB, Express.js, AngularJS, and Node.js. In fact, searching the last software systems results in more than 5,000, 9,000, 11,000 and 55,000 images for MongoDB, Express.js, AngularJS, and Node.js respectively.

In this paper, we study the differences among 936 DockerHub images for five software systems in order to help Docker tooling creators and Docker-

Hub define approaches that guide users locate the most suitable images for their varying needs. Although prior studies focus on various aspects such as security [27, 29, 33], quality [8, 35], and evolution [8, 34] of Docker images, understanding the differences among DockerHub images to help users identify a suitable image has not been explored in the literature. Borgi et al. [5] is the closest work to our paper since they proposed a prototype for a multi-attribute based search tool for DockerHub images. However, their approach is limited to searching for images based on their names, sizes, and installed software distributions. Their tool can report a large number of images while providing no assistance for users who might wish to differentiate among these reported images.

In the first part of this paper, we perform a preliminary study to understand the benefits of using a DockerHub image, as well as the efforts to identify a suitable image on DockerHub by studying five software systems and 936 DockerHub images for each of these software systems. We observe that Docker images require a considerable amount of effort to be built from scratch. However, finding a suitable image on DockerHub is not trivial since a large number (a median number of 19,280) of images are reported when searching for each software system. We also observe that the most popular DockerHub images are the official images, although they might not be the most suitable images to fulfill the needs of a user.

In the second part of this paper, we study the differences among the DockerHub images for the same software systems in terms of their installed libraries, resource efficiency, and security vulnerabilities. In this regard, we answer the following three research questions:

RQ1. How different are DockerHub images for the same software system in terms of their installed libraries?
DockerHub contains a large number of images for each software system. These DockerHub images are different from each other in terms of their installed libraries and versions. In particular, 27% (median), 35% (median), 6% (median), and 9% (median) of the Debian, Centos, Ubuntu, and Alpine based images are identical to at least one other DockerHub image across all the five studied software systems. Such differences are, unfortunately, not well documented with 58% of the images not having any description and the descriptions of 20% of the images not providing any indication about the installed libraries.

RQ2. How different are these images in terms of their size and resource efficiency?
Although official images are the most popular, 26% (median), 49% (median), and 8% (median) of the community Alpine, Debian, and Ubuntu based images are more resource efficient (i.e., have less duplicate resources) than their respective official images across all the five studied software systems. DockerHub images for the same software system have different sizes, which indicates that there are a variety of DockerHub image choices based on their used resources. Images for Cassandra based on Centos have the

largest interquartile range of 227 MB, while images for Nginx Alpine based images have the lowest interquartile range of 8 MB. Similarly to RQ1, we observe that developers do not document any particularities neither about the used resources nor about the resource efficiency of their images.

RQ3. How different are these images in terms of their security vulnerabilities?

While users prefer using official images, 7% (median) of the community images have less security vulnerabilities (within the installed libraries as reported by the Debian's security-tracker) than their respective official images across the four studied software systems, which have an associated Debian based official image. However, no indication about the security of community images is provided neither by the DockerHub UI nor by the description of these images. Note that DockerHub provides a feature that identifies the vulnerabilities of an image; however, that feature is available only for the official images.

Finally, our findings suggest to Docker tooling creators and DockerHub define approaches that help users differentiate among DockerHub images and find the suitable images of one's needs.

The rest of this paper is organized as follows: Section 2 provides the background of our study. Section 3 presents our data extraction. Section 4 discusses the results of our preliminary study. Section 5 provides the results of our study. Section 6 presents a discussion based on our observations. Section 7 presents related work. Section 8 discusses the threats to validity of our results. Finally, Section 9 concludes the paper.

2 Background

Containerization is the process of packaging a software system with all of its dependencies so that it can be run on any platform without any compatibility or dependency issues. Containerization is an approach that provides an operating system level virtualization which is more efficient and faster than the traditional virtualization platforms [2, 21, 26]. That efficiency makes Docker one of the most popular containerization platforms [23].

2.1 Docker Image

The creation of a Docker image starts with the creation of a Dockerfile as shown in Figure 1. A Dockerfile is a script that contains the required instructions to install and configure a software system. Figure 2 shows an example of a Dockerfile that builds an image to run the application *myapp.jar*, which depends on Java. That Dockerfile uses Ubuntu as its operating system (line 2), updates the installed packages (line 8), installs Java (line 11-12), copies resources into the Docker image (line 15), and runs the application *myapp.jar* (line 21).

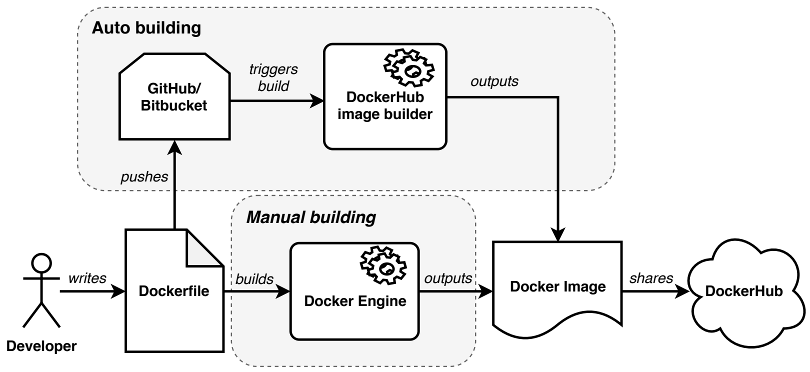
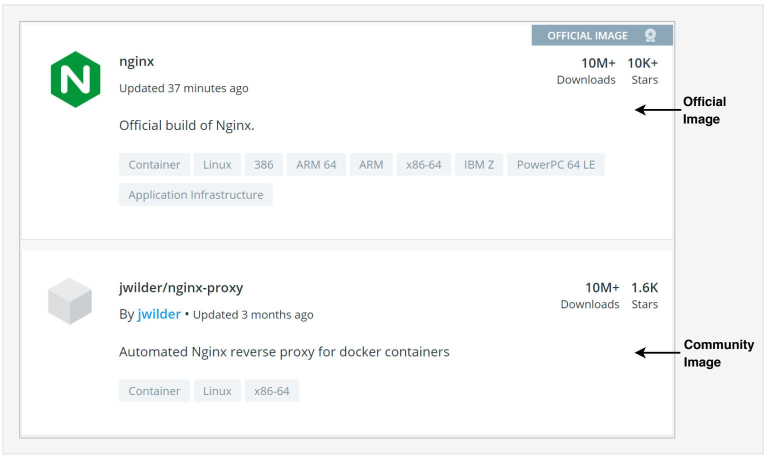


Fig. 1: Building and sharing images on DockerHub.

```
#extending base image
FROM ubuntu:18.04

#maintainer email address
LABEL maintainer="john.doe@example.com"

#updating libraries
RUN apt-get -y update

#installing java8
RUN add-apt-repository ppa:webupd8team/java && \
  apt install -y oracle-java8-installer

#copying target application package
COPY . /app/

#setting the working directory
WORKDIR /app/

#running the application
CMD ["java", "-jar", "myapp.jar"]
```

Fig. 2: An example of a Dockerfile that defines an image for *myapp.jar* that runs on the Java runtime.

Starting from a Dockerfile, a Docker image can be built manually or automatically as shown in Figure 1. In the manual process, a developer can build an image from a Dockerfile using the Docker Engine. This manual process requires a developer to build the image whenever he or she updates their Dockerfile. On the other hand, to build an image automatically, a developer needs to push their Dockerfile to a GitHub [19] or Bitbucket [4] repository and link that repository with a DockerHub repository (discussed in the next sub-



OFFICIAL IMAGE

nginx
Updated 37 minutes ago

10M+ Downloads 10K+ Stars

Official build of Nginx.

Container Linux 386 ARM 64 ARM x86-64 IBM Z PowerPC 64 LE
Application Infrastructure

jwilder/nginx-proxy
By jwilder • Updated 3 months ago

10M+ Downloads 1.6K Stars

Automated Nginx reverse proxy for docker containers

Container Linux x86-64

Fig. 3: Official and community images on DockerHub.

section). DockerHub then automatically builds and updates the Docker image when a new Dockerfile version is pushed to the version control repository.

2.2 DockerHub

DockerHub is an online cloud-based image registry platform that hosts Docker images which can be pulled and consumed by the Docker community. Each image can have a short and/or a detailed description. An image can be either public or private.

As shown in Figure 3, DockerHub hosts two types of public images: official and community images. Official images are provided by DockerHub which, in turn, ensures regular security updates for such official images [17]. On the other hand, community images are developed and shared by the developer community.

Every DockerHub image has the following meta-data associated with it as shown in Figure 5:

- *Full Name:* The full name of a DockerHub community image is composed of two parts: the owner name and the name that is provided by the developer of the image. Note that official images do not have an owner name.
- *Short Description:* Each image has a short description which briefly describes the image.
- *Full Description:* The full description provides a detailed description about the image.
- *Tags:* An image on DockerHub might have different versions, each of which has a different tag. For example, the Cassandra official image has different versions, each of which has a specific tag (e.g., latest and 3.11.6) as shown in Figure 4.

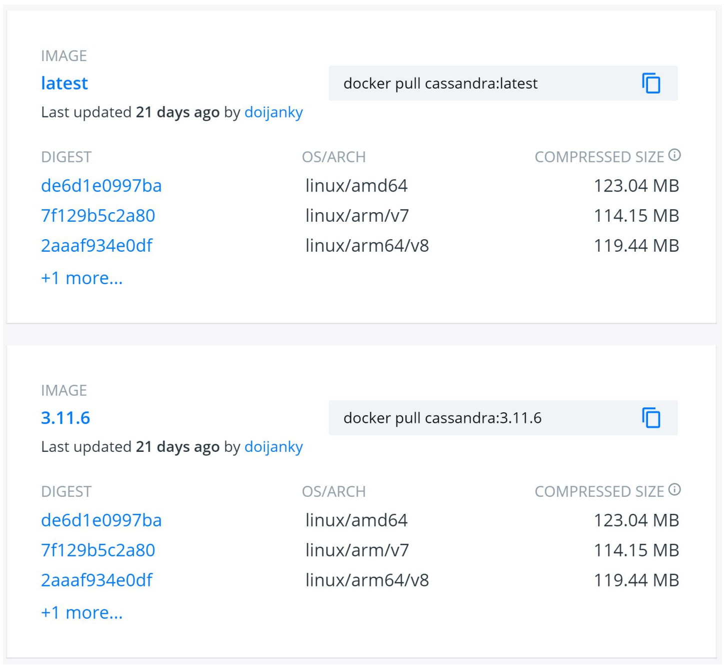


IMAGE
latest
docker pull cassandra:latest
Last updated 21 days ago by doijanky

DIGEST	OS/ARCH	COMPRESSED SIZE
de6d1e0997ba	linux/amd64	123.04 MB
7f129b5c2a80	linux/arm/v7	114.15 MB
2aaaf934e0df	linux/arm64/v8	119.44 MB

+1 more...

IMAGE
3.11.6
docker pull cassandra:3.11.6
Last updated 21 days ago by doijanky

DIGEST	OS/ARCH	COMPRESSED SIZE
de6d1e0997ba	linux/amd64	123.04 MB
7f129b5c2a80	linux/arm/v7	114.15 MB
2aaaf934e0df	linux/arm64/v8	119.44 MB

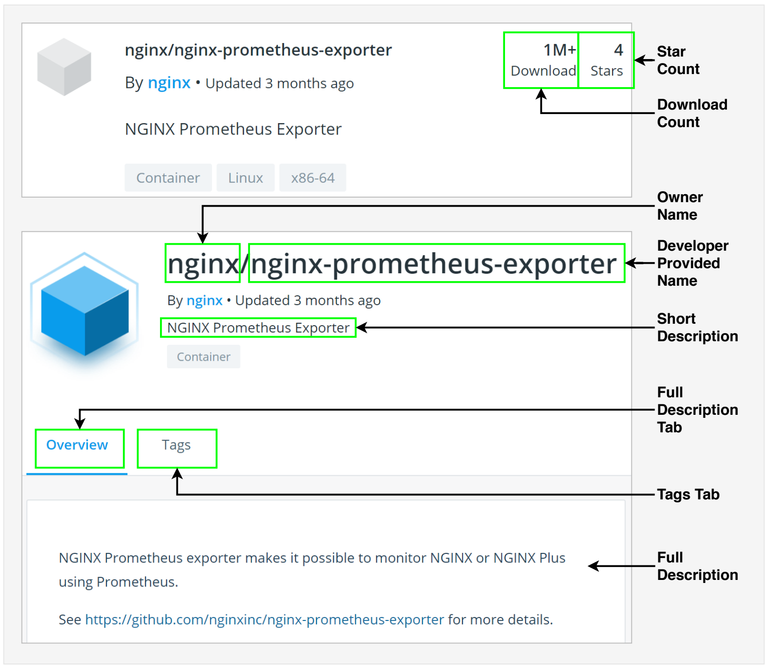
+1 more...

Fig. 4: An example of two versions (i.e., latest and 3.11.6) of the Cassandra DockerHub image.

– *Download Count:* It represents the number of times an image is downloaded. DockerHub UI shows the exact number of downloads when it is lower than 1,000, and shows that number using a thousand or million scale for images with a large number of downloads as shown in the two examples of Figure 5.
– *Star Count:* Similar to the number of downloads, DockerHub provides the number of stars on an image. Star count is also shown using a thousand or million scale when the count is equal or higher than 1,000.

3 Data Extraction

In this paper, we study the differences among DockerHub images for the same software system using three different data sources (i.e., installed libraries, resource efficiency of images, and security vulnerabilities). Our final data-set contains 936 DockerHub images, which consists of seven official images and 929 randomly-selected community images for each of the five studied software systems: Cassandra, Java, Mongo, MySQL, and Nginx. Note that there is only one official image for each software system on DockerHub.


nginx/nginx-prometheus-exporter

By nginx • Updated 3 months ago

NGINX Prometheus Exporter

Container Linux x86-64

1M+ Download 4 Stars

nginx/nginx-prometheus-exporter

By nginx • Updated 3 months ago

NGINX Prometheus Exporter

Container

Overview Tags

NGINX Prometheus exporter makes it possible to monitor NGINX or NGINX Plus using Prometheus.

See https://github.com/nginxinc/nginx-prometheus-exporter for more details.

Fig. 5: DockerHub image meta-data.

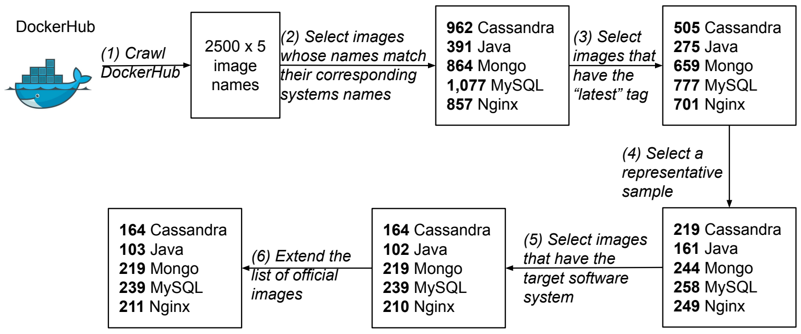

DockerHub
(1) Crawl DockerHub
2500 x 5 image names
(2) Select images whose names match their corresponding systems names
962 Cassandra 391 Java 864 Mongo 1,077 MySQL 857 Nginx
(3) Select images that have the "latest" tag
505 Cassandra 275 Java 659 Mongo 777 MySQL 701 Nginx
(4) Select a representative sample
219 Cassandra 161 Java 244 Mongo 258 MySQL 249 Nginx
(5) Select images that have the target software system
164 Cassandra 102 Java 219 Mongo 239 MySQL 210 Nginx
(6) Extend the list of official images
164 Cassandra 103 Java 219 Mongo 239 MySQL 211 Nginx

Fig. 6: The process of collecting images.

3.1 Images Selection

Our image collection (performed in January 2020) consists of several steps that are shown in Figure 6 and summarized in the following steps:

– **Step 1**: For each software system, we crawl all the image names that show up in the first 100 pages of the DockerHub search results. (DockerHub only returns 100 pages of results for each search). We end up with the names of 2,500 images for each of the studied software systems (i.e, 12,500 image names).

- **Step 2**: We select the official image as well as images whose name matches exactly each of our studied systems. Our initial observations show that some images combine our searched software system with other systems or components. For example: the “richarvey/nginx-php-fpm” might install not just “nginx”, but also “php” and “fpm”. Therefore, to compare images that provide just our software system of interest, we only consider images whose name exactly matches the name of the studied software systems. For example, an image name should contain just the word “nginx” for nginx related images such as “mailu/nginx” (owner name/image name). Note that we consider the names “mongo” and “mongodb” for the Mongo software system. We obtain 962, 391, 864, 1077, and 857 images for the Cassandra, Java, Mongo, MySQL, and Nginx software systems, respectively, as shown in Figure 6.
- **Step 3**: Our study considers the last version (i.e., the version that is tagged with the “latest” tag) of each image in order to guarantee that we study images that are at the same stage in regards to the current versions of their libraries.
- **Step 4**: We select a representative random sample (confidence level = 95%, confidence interval = 5%) from the images from Step 3 to end up with the number of images that are shown in Figure 6.
- **Step 5**: We ensure that each studied community image respects the following criterion: the image should install the target software system (indicated by its name) using one of the following package managers: the Advanced Package Tool (APT), Alpine Linux package manager (APK), or Red Hat Package Manager (RPM).
- **Step 6**: We found that Java and Nginx official images support additional versions for Java and Nginx on the Alpine Linux distribution, whereas the “latest” version of these two images are based on the Debian Linux distribution. Therefore, we extend our dataset of images with these two additional versions of official images. The official images that are related to the other software systems support just one Linux distribution [1].

Our comparisons use the following four perspectives:

Libraries: We use the *Snyk-docker-analyzer* [28] tool, which is an open source software tool that extracts the installed libraries in an image. The tool supports the APT, APK, and RMP package managers. We used the *Snyk-docker-analyzer* to extract the installed libraries for each of the 936 studied images. *Snyk-docker-analyzer* pulls each of the 936 images from DockerHub, builds a container for each of these images, executes these containers, then extracts the installed libraries in these images. The Snyk-docker-analyzer provides for each image the name of its installed libraries,

[1] Our data is publicly available in https://github.com/SAILResearch/replication_dockerhub.

their versions, and how they are installed: manually by the developer of that image, or automatically via an implicit dependency.

Resource Efficiency: We use *Dive* [30] to measure the resource efficiency for each of our selected images. *Dive* is a tool that helps users explore the content of an image, and that evaluates the resource efficiency of a Docker image based on the amount of its wasted space (duplicate resources). For example, one image may contain multiple identical files in different layers or even in the same layer which is a waste of space. The resource efficiency of an image is measured as follows:

$$E = \frac{S - W}{S} \times 100 \qquad \begin{array}{ll} E & = \text{Resource efficiency} \\ S & = \text{Size of the image} \\ W & = \text{Wasted space due to duplicate file resources} \end{array}$$

Security Vulnerabilities: We use ConPan [32] that relies on the publicly available Debian vulnerabilities dataset [13] to identify the security vulnerabilities of the installed libraries for Debian based images. ConPan reports the list of vulnerabilities that exist on a Docker image, the vulnerable libraries and versions, the vulnerability status (open or resolved), and the severity of the vulnerability (low, medium, or high).

Meta-data: We collect the size, the number of downloads, and the number of stars of each of the 936 images using the DockerHub HTTP API [16].

4 Preliminary Study: The Advantages and Abundance of DockerHub Images

The goal of our preliminary study is to understand the advantages of using an existing image on DockerHub instead of building one from scratch, as well as the difficulty of finding an image on DockerHub. We structure our preliminary study through the following preliminary research questions:

PQ1. How many libraries one has to install for a typical software system?
PQ2. How many images are available when searching for a typical software system on DockerHub?

PQ1. How many libraries one has to install for a typical software system?

Motivation: The goal of this preliminary research question is to understand the benefits of using a DockerHub image. In other words, the goal is to understand the amount of efforts that practitioners can save by using one of the available DockerHub images instead of building an image from scratch.

Approach: To measure the development and maintenance efforts that one can save using a DockerHub image, we count the number of libraries which are manually installed as well as the libraries that are automatically installed

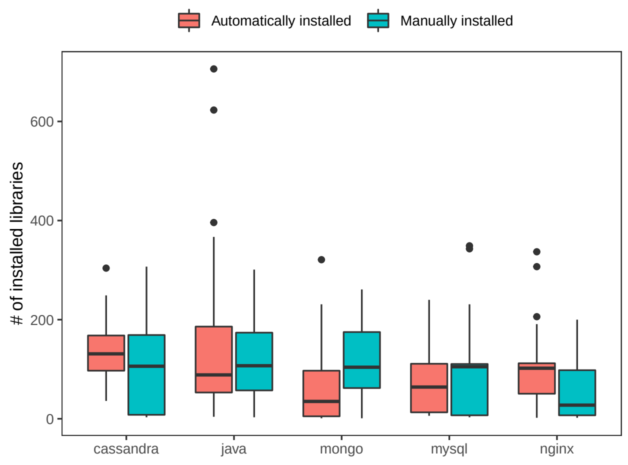


Fig. 7: The number of installed libraries in different types of images.

to quantify the required efforts for building and maintaining a single Docker image. Note that we extract the libraries that are installed in an image following the approach discussed in Section 3.

Results: Developing a new Docker image might require a considerable amount of installation and testing effort. We observe that one has to manually install a large number of libraries whose median ranges between 27 and 107 installed libraries. The situation gets even worse when it comes to building a distributed software system, which involves building and testing multiple Docker images.

Besides installing and testing an image built from scratch, maintaining an image can be more challenging. Prior studies [14, 33] suggest that many libraries can introduce bugs as well as security issues in a Docker image. Therefore, developers need to check that the libraries that they install are secure and error free. Unfortunately, developers do not only need to check the health of the manually installed libraries, but also libraries that are automatically installed via implicit dependencies. In fact, the number of these automatically installed libraries is not negligible since we observe that a median of 35 to 131 libraries are installed automatically as shown in Figure 7.

PQ2. How many images are available when searching for a typical software system on DockerHub?

Motivation: The goal of this preliminary research question is to understand the variety of choices that exist when searching for a single software on DockerHub. We believe that the larger the number of available images is, the more challenging it is to find the appropriate image. Therefore, this question investigates the number of images that are provided for each of our five studied software systems.

Approach: To understand the variety of DockerHub image choices a user has, we first report on the number of images that are reported by DockerHub when searching for the name of each of our five studied software systems.

Results: A large number of images exists when searching for each software system. We observe that DockerHub has an active community of developers as the median number of images that are provided when searching for each software system is 19,280 images. The number of available images for a software system ranges between 2,337 images for Cassandra to 57,120 images for Nginx as shown in Table 1. Therefore, choosing one out of the thousands of the available images might be challenging.

Table 1: Number of images (by March 2020) available on DockerHub when searching for images related to the five studied software systems.

Image type	**# of images**
Nginx	57,120
Java	22,621
MySQL	19,280
Mongo	12,069
Cassandra	2,337

Although a large number of community images exists, the official images are the most popular images. We observe that our 5 studied official DockerHub images have a median of 10M downloads and a median of 6,500 stars, while the most popular community images for each of our 5 studied software systems have a median of 5M downloads and a median of 30 stars (considering all of the 2,500 community images that we were able to crawl for each of our five studied software system as discussed in Section 3.1). This result shows that official images have more than 2 times downloads and nearly 216 times more stars than their most popular community images.

Summary of Preliminary Study

Using an image from DockerHub might save the efforts to install, test, and maintain a large number of libraries. However, choosing one image out of the thousands of available images is challenging. Therefore, users often download and star official images, while better community images might exist. Our results suggest a need for a study to identify how different are DockerHub images, which can indicate the number of true choices one has for a single software system.

5 The Differences among DockerHub Images

The goal of this paper is to better understand the differences among DockerHub images in order to help users find suitable Docker images. Since we find in our preliminary study that users have a large number of image choices for building a single software system on DockerHub, this paper compares the images that are provided for the same software system on DockerHub. For example, we compare images together whose names contain Nginx. We summarize our comparisons along the following perspectives:

RQ1. How different are DockerHub images for the same software system in terms of their installed libraries?
RQ2. How different are these images in terms of their size and resource efficiency?
RQ3. How different are these images in terms of their security vulnerabilities?

RQ1. How different are DockerHub images for the same software system in terms of their installed libraries?

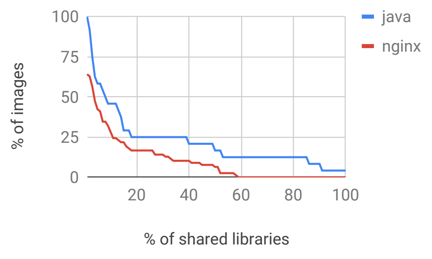
Motivation: While DockerHub provides a large number of images for each software system (PQ1) and users tend to select official images (PQ2), the goal of this research question is to understand whether there are a large variety of choices for each single software system, or if all the images are similar to the official image and to each other.

Approach: To evaluate the variety of choices one has for each software system, we compare DockerHub images based on their Linux distribution and installed libraries. We first extract the Linux distribution of an image using *Snyk-docker-analyzer*. We also compare the community images with their corresponding official image. Afterwards, we compare each pair of community images in terms of their installed libraries. For example, considering the images a, b, and c, we compare a-b, a-c, and b-c based on their installed libraries. Note that we use the *Snyk-docker-analyzer* tool to extract the installed libraries of an image as discussed in Section 3.

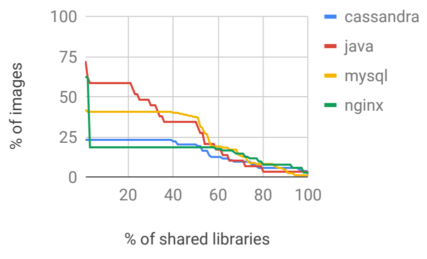
We consider images for the following Linux distributions: Alpine, Centos, Debian, and Ubuntu since we have a large number of images to compare each distribution. In addition, we do not compare images that have a different Linux distribution, we compare only images with the same distribution.

Table 2: Existing Linux distributions. (*) Indicates the Linux distribution of the official image.

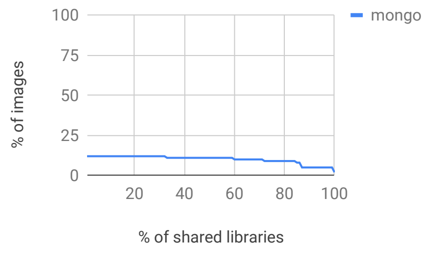
	Alpine	Centos	Debian	Fedora	Fedora-modular	Oracle	Raspbian	Rhel	Ubuntu
Cassandra	0	17	(*) 104	0	0	0	0	0	43
Java	(*) 25	4	(*) 30	2	0	0	0	0	42
Mongo	22	9	81	2	1	0	2	1	(*) 102
Mysql	14	8	(*) 158	0	0	4	0	0	55
Nginx	(*) 79	3	(*) 103	0	0	0	0	0	26



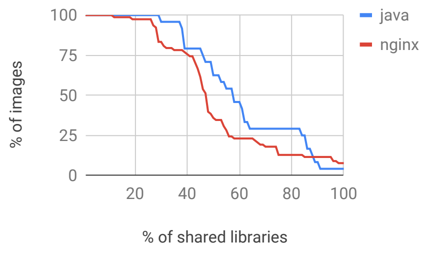
(a) Considering library versions - Alpine



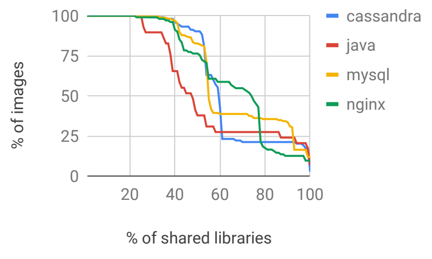
(b) Considering library versions - Debian



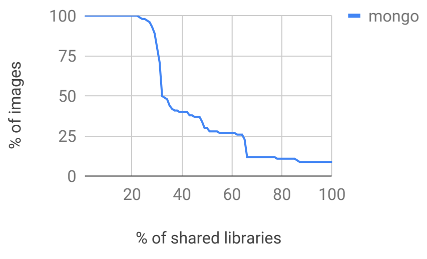
(c) Considering library versions - Ubuntu



(d) Ignoring library versions - Alpine



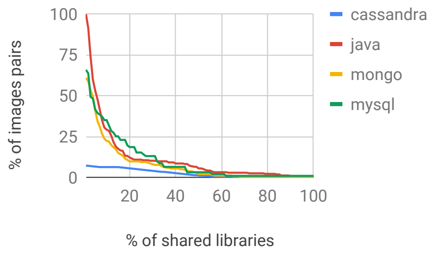
(e) Ignoring library versions - Debian



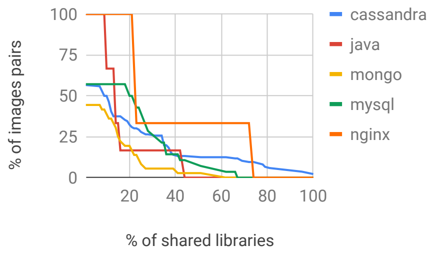
(f) Ignoring library versions - Ubuntu

Fig. 8: The percentage of libraries shared between community and official images. The x-axis represents the cumulative percentage of shared libraries between the community images and their corresponding official image. The y-axis represents the percentage of community images. For example, in Figure 8e, 51% of the Java Debian based community images share 47% of their installed libraries with their official image (when ignoring the exact versions of the libraries).

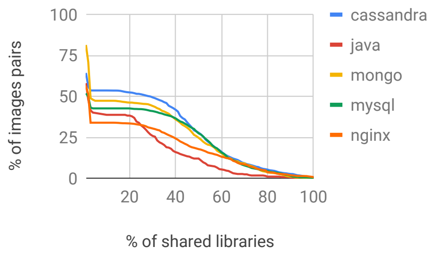
Results: We observe that the community images provide a richer variety of Linux distributions compared to the official images. While the most popular images are the official images, one can still find the same software system installed on a Linux distribution within the community images. For example, the official image provides Mongo on Ubuntu, while one can find a community image that provides Mongo on a different Linux distribution, such as Alpine and Debian as shown in Table 2.



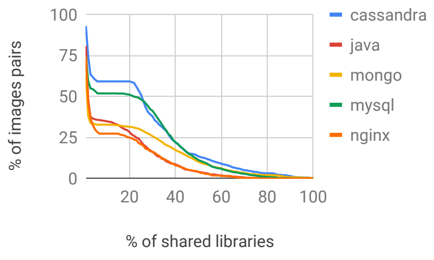
(a) Considering library versions - Alpine



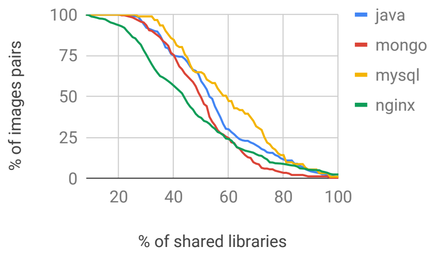
(b) Considering library versions - Centos



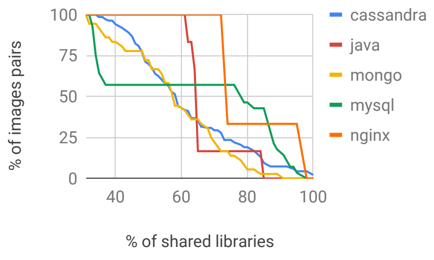
(c) Considering library versions - Debian



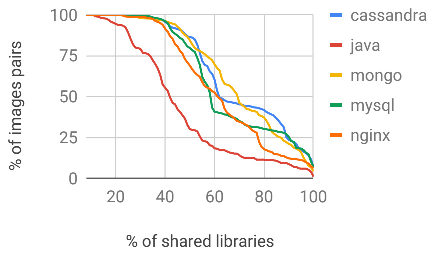
(d) Considering library versions - Ubuntu



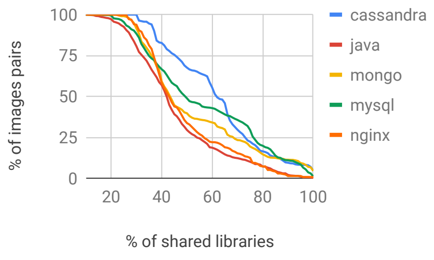
(e) Ignoring library versions - Alpine



(f) Ignoring library versions - Centos



(g) Ignoring library versions - Debian



(h) Ignoring library versions - Ubuntu

Fig. 9: The percentage of libraries that each pair of images share. The x-axis represents the cumulative percentage of shared libraries between a pair of images. The y-axis represents the percentage of image pairs. For example, in Figure 9e, 60% of the Mongo Alpine based image pairs share 46% of their installed libraries (when ignoring the exact versions of the libraries).

DockerHub images are different from each other in terms of their installed libraries considering the exact versions of these libraries. We first observe that community images are different from their corresponding official image. Only one community Java Alpine based image share 100% of its installed libraries (with exact library versions) with its corresponding official Java Alpine based image, while none of the images for the other four software systems on the different Linux distributions is identical to their corresponding official image. Furthermore, 32% (median), 20% (median), and 12% (median) of the Debian, Alpine, and Ubuntu based community images share at least 25% of their libraries with their respective official images across all the five studied software systems, as shown in Figure 9c, 9a, and 9d, respectively.

In addition to the differences between community and official images, community images are different from each other in terms of their installed libraries considering the exact versions of the installed libraries. 27% (median), 35% (median), 6% (median), and 9% (median) of the Debian, Centos, Ubuntu, and Alpine based images have at least one other image with which they share 100% of their libraries and versions across all the studied software systems. Moreover, our comparison shows that 10% (median), 27% (median), 42% (median), and 29% (median) of the Alpine, Centos, Debian, and Ubuntu based image pairs share at least 25% of the exact version of their installed libraries across all the five studied software systems, as shown in Figure 9a, 9b, 9c, and 9d. Note that we ignore systems which do not have any image on a given Linux distribution.

DockerHub images are different from each other in terms of their installed libraries independently from these libraries' versions. Ignoring the versions of the libraries, we observe that 5% (median), 8% (median), and 9% (median) of the Alpine, Debian, and Ubuntu based images are identical to their corresponding official image across the five studied software systems, as shown in Figure 8d, 8e, and 8f, respectively. While the community images are different from the official images, we observe that all the Alpine, Debian, and Ubuntu images share a maximum of 20% (median), 26% (median), 22% (median) of their respective installed libraries with their corresponding official images across the five studied software systems.

In addition to the differences between community and official images, community images are different from each other. 57% (median), 35% (median), 14% (median), and 15% (median) of the Debian, Centos, Ubuntu, and Alpine based community images share exactly the same set of libraries with at least one other image (irrespective of their versions) across all the five studied software systems. Moreover, our comparison shows that 54% (median), 72% (median), 79% (median), and 39% (median) of the Alpine, Centos, Debian, and Ubuntu based image pairs share at least 50% of the exact version of their installed libraries across the five studied software systems, as shown in Figure 9e, 9f, 9g, and 9h. Note that we ignore systems which do not have any image on a given Linux distribution.

While there are a median number of 13, 119, 0, and 54 libraries that are installed in all the Alpine, Centos, Debian, and Ubuntu based images respec-

Table 3: Distribution of installed libraries on the studied images. Number of installed libraries across all images/Number of installed libraries by all images/Number of libraries that are unique to an image. Note that we only consider libraries that are not automatically installed.

	Alpine	Centos	Debian	Ubuntu
Cassandra	0/0/0	433/94/147	216/0/48	309/66/60
Java	137/17/53	311/145/74	402/0/200	557/54/279
Mongo	148/14/72	385/68/157	250/0/95	375/50/102
Mysql	99/13/63	411/119/38	203/0/52	395/52/118
Nginx	300/10/179	204/145/57	350/0/186	291/54/57
Median	137/13/63	385/119/74	250/0/95	375/54/102

tively, a median number of 63, 74, 95, and 102 libraries are more specific to one Alpine, Centos, Debian, and Ubuntu based image, respectively, as shown in Table 3. We further investigate the category of libraries that are installed by at least 50% of the Debian based images. We observe that the most popular libraries are related to the group of libraries that are required by other programs, utilities that are used to manipulate files and the disk, and administrative utilities that are used to handle system resources and user accounts as shown in Table 4. Note that we consider just Debian images since Debian provides a categorization of its libraries based on their goals.

While DockerHub images are different from each other, **distinguishing images from their description is not straightforward as most of the DockerHub images are not well documented.** Our manual analysis for a representative sample of 272 images shows that 61 (22%) images have a good description that highlights the features of the images, and 78% are not well documented. In fact, 54 (20%) have a weak description which does not present the features of the image, while the remaining (58%) images do not have any description. Figure 10a and 10b show an example of a DockerHub image that documents its features and an image with a weak description respectively.

Summary of RQ1

There are a large variety of choices for each single software system on DockerHub images which are hard to distinguish since developers do not describe the particularities of their images.

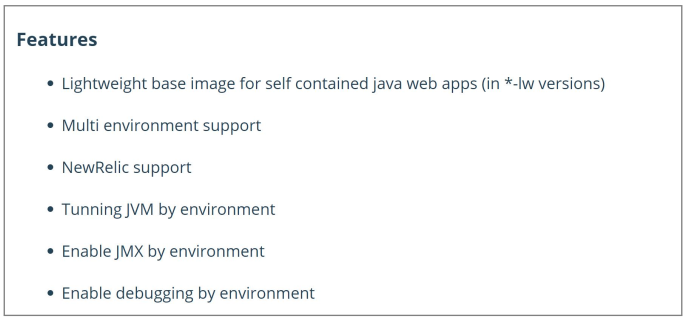
uu

RQ2. How different are these images in terms of their size and resource efficiency?

Motivation: The goal of this research question is to identify the differences among the images in terms of their size and resource efficiency. Duplicate resources in a Docker image can make the containers unnecessarily larger in size. A prior study [36] shows that container deletion time increases as the

Category	Description	Cassandra	Java	Mongo	Mysql	Nginx
Administration Utilities	"Utilities to administer system resources, manage user accounts" [12]	17	24	18	1	0
Databases	Databases related libraries	0	0	0	1	0
Interpreters	interpreters related libraries	1	1	1	0	0
Language packs	Localization support for packages	2	2	2	0	0
Libraries	"Libraries to make other programs work" [12]	28	46	28	0	0
Meta packages	Libraries that provide dependencies between packages	0	1	0	0	0
Miscellaneous	-	4	5	5	0	0
Network	Network based libraries	2	2	1	2	0
Perl	Perl based libraries	5	5	5	1	0
Shells	Command shells	2	2	2	0	0
Utilities	"Utilities for file/disk manipulation, backup and archive tools, system monitoring, input systems" [12]	15	19	15	1	1
Version Control Systems	-	0	1	0	0	0
Virtual packages	-	18	18	16	2	1
Web Servers	Web servers and their modules	0	0	0	0	1
Web Software	"Web servers, browsers, proxies, download tools etc." [12]	0	2	0	0	0

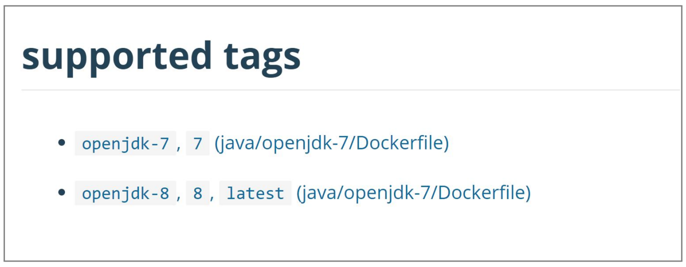
Table 4: Categories of the Debian libraries [12] that are installed by at least 50% of the Debian based images. For example 17 administration utilities related libraries are installed by at least 50% of the Cassandra Debian based images.



Features

- Lightweight base image for self contained java web apps (in *-lw versions)
- Multi environment support
- NewRelic support
- Tunning JVM by environment
- Enable JMX by environment
- Enable debugging by environment

(a) Part of a good description



supported tags

- `openjdk-7` , `7` (java/openjdk-7/Dockerfile)
- `openjdk-8` , `8` , `latest` (java/openjdk-7/Dockerfile)

(b) Weak description

Fig. 10: Two examples of DockerHub image description.

image size increases. The slower deletion time affects the performance of the container redeployment process. Moreover, the increased container size due to duplicate resources negatively affects the cost of resource provisioning in the Cloud. Hence, it is important to know the size and the resource efficiency of an image and how they are different from other similar images.

Approach: To identify how images are different in terms of their sizes and resource efficiency, we collect for each image (we used the same images that are used in RQ1) its size and calculate its resource efficiency following the approach discussed in Section 3. Note that **the resource efficiency of an image consists of the amount of wasted space due to duplicated resources**. Similarly to RQ1, we first compare the official to the community images, then community images to each other, we consider images for the following Linux distributions: Alpine, Centos, Debian, and Ubuntu, and we compare only images with the same distribution.

Results: DockerHub images are different from each other in terms of their sizes. 58% (median), 60% (median), and 56% (median) of the Alpine, Debian, and Ubuntu based community images are larger in size compared to their official images across the five studied software systems, as shown in Figure 11, which suggests that such images might contain additional resources that can differentiate images.

Table 5: Spearman correlation between image size versus library count and image size versus wasted space. Note that we did not include 4 Centos images since they do not have enough data points to statistically compare. Bold numbers indicate strong correlations (i.e., r > 0.7).

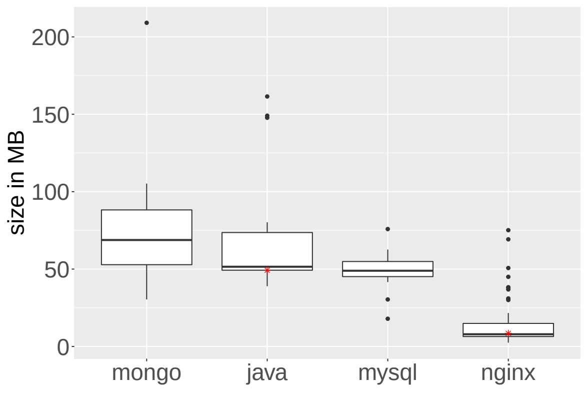
Case Study	Alpine		Centos		Debian		Ubuntu	
	library count	Wasted space	library count	Wasted space	library count	Wasted space	library count	Wasted space
Cassandra	-	-	0.09	0.20	**0.73**	0.20	-0.12	0.52
Java	**0.84**	0.09	-	-	0.23	0.22	-0.14	0.26
Mongo	0.58	0.22	-	-	0.43	0.30	0.42	0.43
MySQL	0.05	0.48	-	-	0.47	0.34	**0.70**	0.44
Nginx	**0.73**	0.66	-	-	**0.83**	0.48	**0.82**	0.43

The size of community images which build the same software system varies. Cassandra Centos based, Java Ubuntu based, then Java Debian based images differ the most in size where their interquartile range of image size is 227 MB, 217 MB and 186 MB, respectively. Although Nginx Alpine based images differ the least with an interquartile range of 8 MB since Alpine is a light-weight Linux distribution, the interquartile range is 100% of the size of their official image.

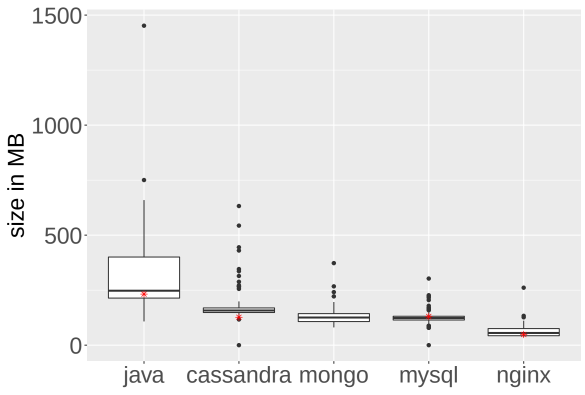
Note that the last difference among the images in terms of the size is not necessarily associated to the number of installed libraries. We observe a statistically significant difference between the number of installed libraries and the size of images for all images and Linux distributions, except for Debian based Java and MySQL images. Our finding suggests that images may contain additional resources that are not necessarily the size of the installed libraries as shown in Table 5. Therefore, images may differ on resources in addition to differing in terms of the installed libraries (RQ1).

DockerHub images are different from each other in terms of their resource efficiency. In addition to the variety of choices that exist on DockerHub images, some images are even better than their corresponding official image in terms of their resource efficiency. 26% (median), 49% (median), and 8% (median) of the community Alpine, Debian, and Ubuntu based images are more resource efficient than their corresponding official image across the five studied software systems, as shown in Table 6. For example, 75% of the Cassandra Debian based community images are more resource efficient than their official image. Besides, we do not observe any official image that is 100% resource efficient (free from any duplicate resources), while 67% (median), 2% (median), and 3% (median) of the Alpine, Debian, and Ubuntu community images are 100% resource efficient across the five studied software systems.

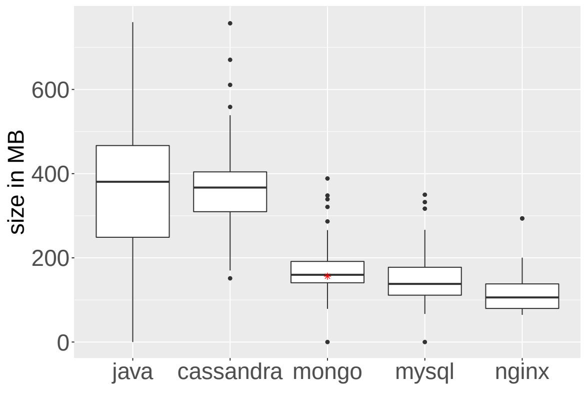
Besides the difference between community and official images, community images are also different from each other in terms of their resource efficiency. For instance, Centos and Ubuntu based community images have a median interquartile of 135.76 MB and 42.85 MB of wasted resources, respectively. While the wasted space is less important on Alpine and Debian images, these images still have extreme cases. Alpine and Debian based community images



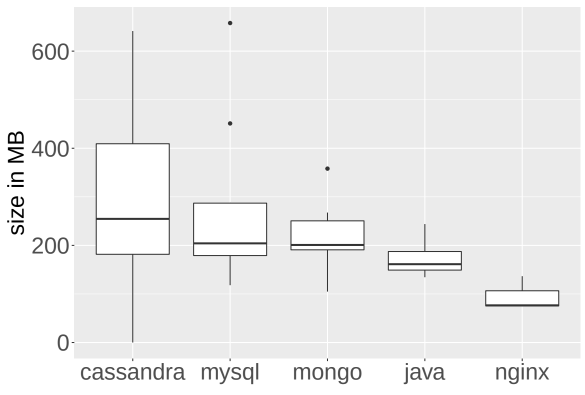
(a) Alpine



(b) Debian



(c) Ubuntu



(d) Centos

Fig. 11: Size of different types of images, where the red star shows the size of the corresponding official image.

Case Study	Alpine			Debian			Ubuntu		
	Official Image Efficiency	Median efficiency of community images	% of the community image more efficient than official image	Official Image Efficiency	Median efficiency of community images	% of the community image more efficient than official image	Official Image Efficiency	Median efficiency of community images	% of the community image more efficient than official image
Cassandra	-	-	-	98.2	98.6	75	-	91.4	-
Java	99.9	99.7	4	98.9	98.9	51	-	93.4	-
Mongo	-	99.7	-	-	98.8	-	98.6	92.4	8
MySQL	-	99.8	-	98.5	98.5	46	-	89.5	-
Nginx	98.9	98.9	48	98.4	98.2	35	-	84.9	-

Table 6: Comparison of the resource efficiency between official and community images.

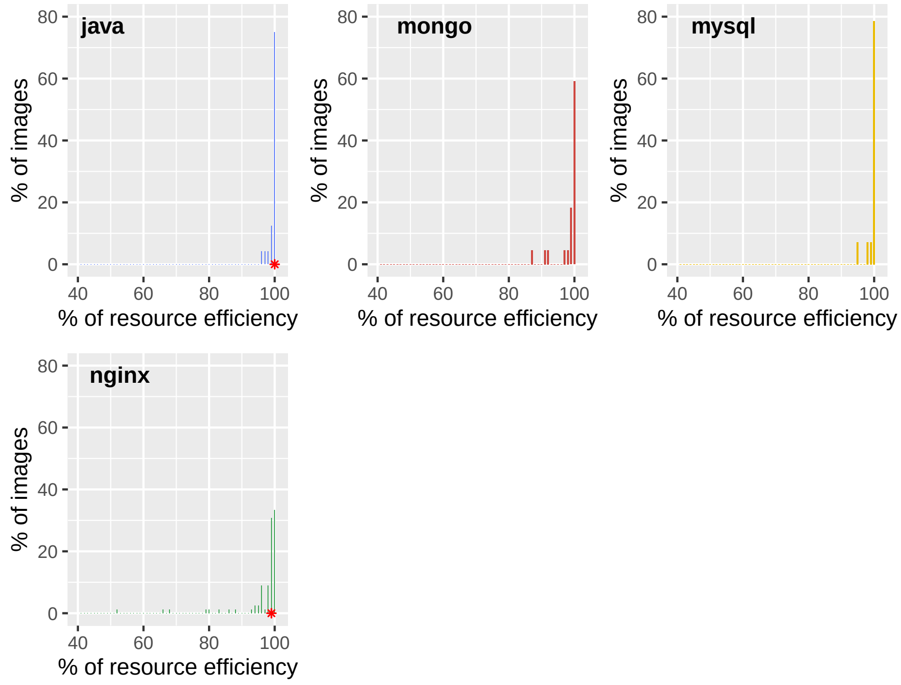


Fig. 12: Resource efficiency distributions of the Alpine based DockerHub images where the red star shows the resource efficiency of the corresponding official image.

have a median difference resource efficiency between community images of 48% (the less and the more resource efficient image have an efficiency of 51% and 99%) and 50% (the less and the more resource efficient image have an efficiency of 49% and 99%), respectively as shown in Figure 12, 13, 14, and 15. Note that there is no correlation (i.e., Spearman correlation) between the size of an image and its resource efficiency for all types of images based on different Linux distributions, except few cases that are shown in Table 5.

While most (median of 1) of the duplicated resources are specific to one image, there are resources that are duplicated across different images, as shown in Figure 16. For example, we observe that the "/etc/group" file, which is dedicated to defining groups to which users belong, is duplicated in 78 Nginx Alpine based images. The same file is also duplicated in 22, 12, and 7 Mongo, Mysql, and Java Alpine based images, respectively.

DockerHub image developers do not provide information regarding the resource efficiency of their images. By manually studying the descriptions of 50 DockerHub images (the top 10 most resource efficient images from each image type), we observe that image descriptions do not reveal any

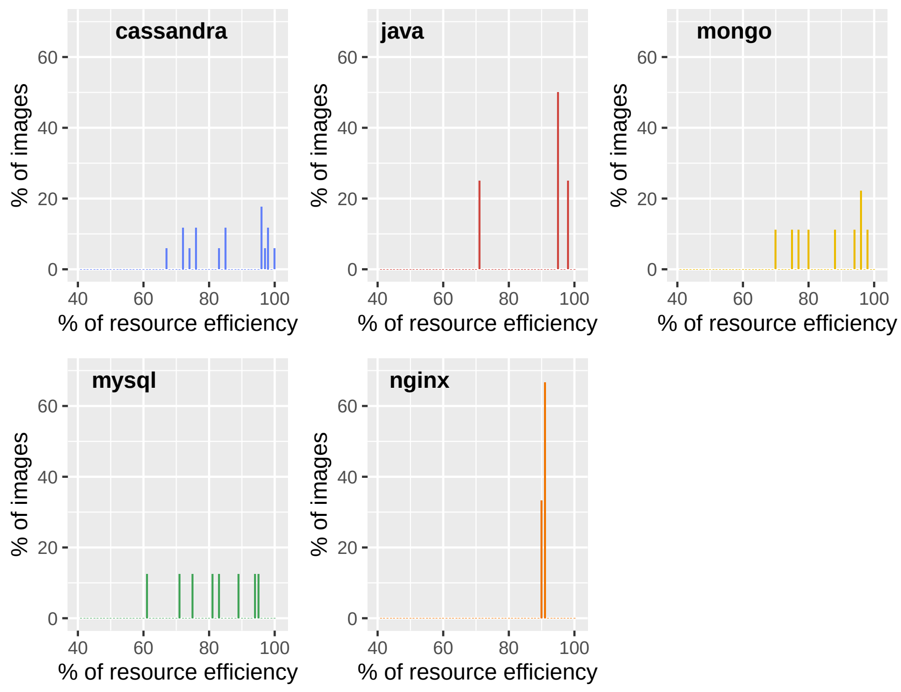


Fig. 13: Resource efficiency distributions of the Centos based DockerHub images.

information about the efficiency of the images, while 2% of the images specify the resources that they contain.

Summary of RQ2

In addition to the variety of choices in terms of the installed libraries, DockerHub images differ in terms of their resources as well. In addition, some choices can be better than other in terms of images resource efficiency. While users tend to download official images, 26% (median), 49% (median), and 8% (median) of the community Alpine, Debian, and Ubuntu based images are more resource efficient across the five studied software systems. However, finding such efficient images is not trivial since developers do not document the resource efficiency of their images.

RQ3. How different are these images in terms of their security vulnerabilities?

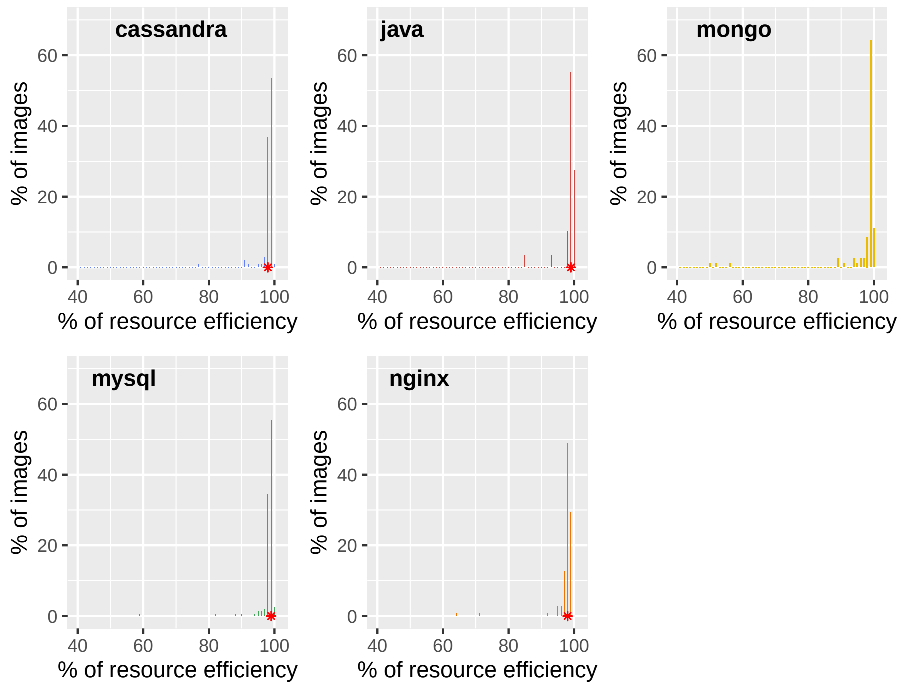


Fig. 14: Resource efficiency distributions of the Debian based DockerHub images where the red star shows the resource efficiency of the corresponding official image.

Motivation: The goal of this research question is to compare the security among DockerHub images since security is a major concern when deploying a Docker image [3].

Approach: To compare DockerHub images based on their security vulnerabilities, we extract the security vulnerabilities of the libraries that are installed on each of the Debian based DockerHub images (that are used in RQ1) using ConPan [32] as discussed in Section 3. Note that we focus on Debian based images, for which we have the vulnerabilities dataset.

Results: DockerHub images have different number of security vulnerabilities. We observe that there are community images which have less security vulnerabilities than their corresponding official image, although Docker regularly applies security updates on the official images [17]. All the four Debian based official images are not free from security vulnerabilities, and 7% (median) of the community images are more secure than their corresponding official images across the four studied systems (7% for MySQL, 0.9% for Nginx, 71% for Java, and 6% for Cassandra), for which an official Debian based image exists, as shown in Figure 17.

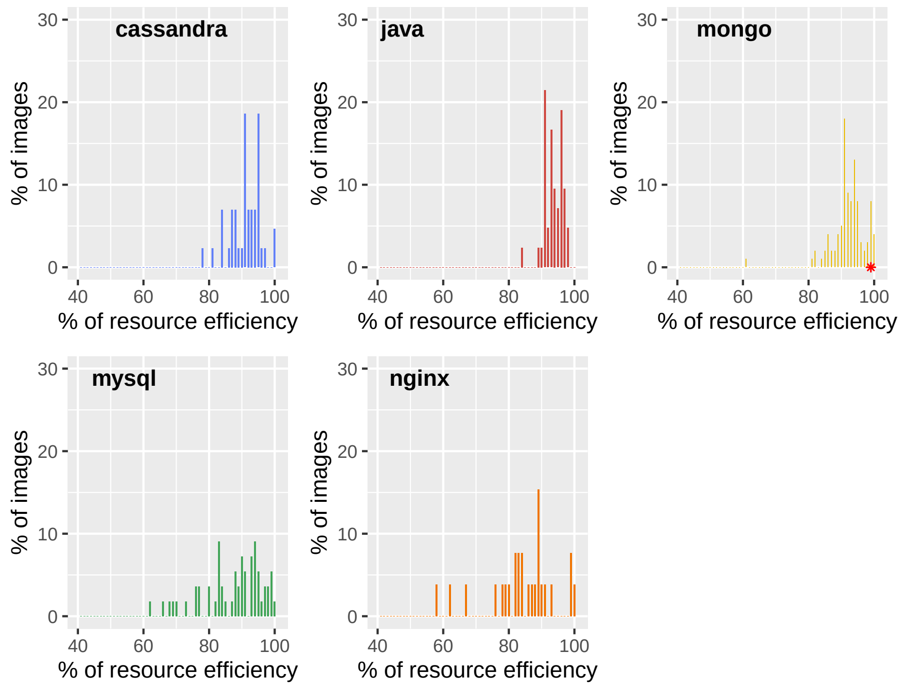
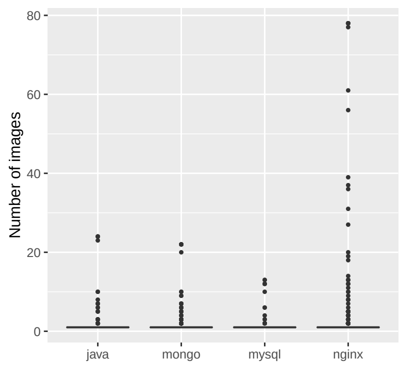


Fig. 15: Resource efficiency distributions of the Ubuntu based DockerHub images where the red star shows the resource efficiency of the corresponding official image.

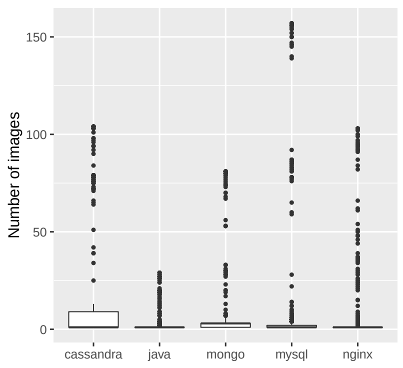
While community images and official images are different in terms of the number of security vulnerabilities they have, we observe a large variance on the number of security vulnerabilities between community images. In fact, the interquartile ranges of security vulnerabilities are 144, 39, 324, 79, 332 for Nginx, MySQL, Java, Mongo, and Cassandra images respectively. Moreover, all the images have at least one vulnerability and the median number of security vulnerabilities are 433, 132, 135, 168, and 319 for Java, Mongo, MySQL, Nginx, and Cassandra images, respectively, as shown in Figure 17.

We do not observe any vulnerabilities that are specific to official images (i.e., vulnerabilities that exist in the official image and not in the community images). We found that the same vulnerabilities can appear in a median of 3 different images, as shown in Figure 18. We observe 12, 8, 2, 7, 10 vulnerabilities that exist in all Cassandra, Java, Mongo, MySQL, and Nginx images. The "CVE-2016-2781" and "CVE-2017-18018" are shared between all the studied images. "CVE-2016-2781" allows local users to access the parent session, whereas "CVE-2017-18018" allows the arbitrary modification of file ownership.

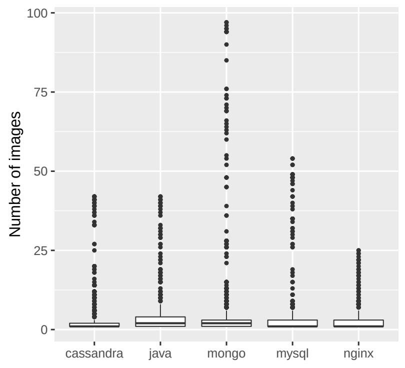
Except for the official images, DockerHub image developers do not provide any information regarding the security aspect of their



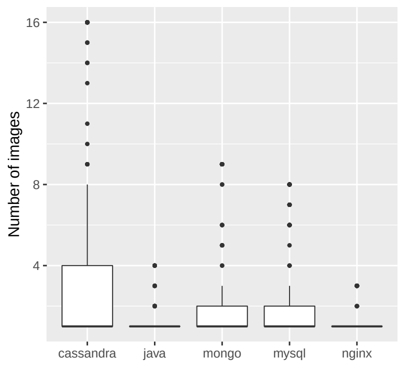
(a) Alpine



(b) Debian



(c) Ubuntu



(d) Centos

Fig. 16: The number of images (y-axis) that have the same duplicated resource.

images. By manually inspecting the descriptions of 50 DockerHub images (the top 10 most secure images from each of the five studied software systems), we do not observe any image where developers discuss its security aspect, except for two cases for which developers mention the security aspect, but not in the context of library vulnerabilities. For example, the description of the "centerx/nginx" image mentions that users *"need to add SSL to secure client interactions"*. Note that DockerHub evaluates and shows the security vulnerabilities of just the official images and not the community images.

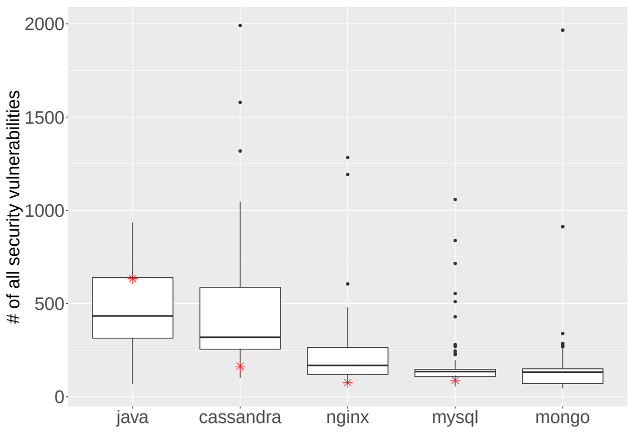


Fig. 17: Number of security vulnerabilities in the DockerHub images where the red stars show the number of security vulnerabilities in the corresponding official image.

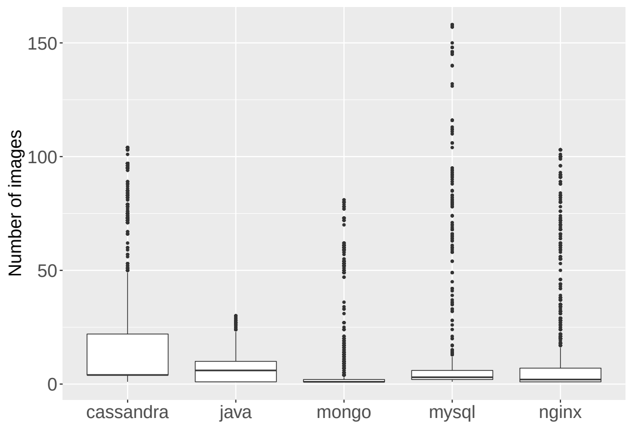


Fig. 18: The number of images (y-axis) that have the same security vulnerability.

Summary of RQ3

Distinguishing secure and insecure images on DockerHub is not straightforward. Official images, for which Docker evaluates and publicly displays their security vulnerabilities, are not always the most secure images. 7% (median) of the community images have less security vulnerabilities than their corresponding official images across the four software systems, for which an official image exists.

Table 7: The percentage of community images that are more resource efficient and have less security vulnerabilities compared to their corresponding official image. Note that there is no official Debian based image for Mongo.

Image type	**% of community images**
Cassandra	3%
Java	44%
Mongo	-
MySQL	4%
Nginx	0%

6 Discussion

We observe from our preliminary study that using an image from DockerHub can save one's efforts of building, testing, and maintaining an image from scratch. However, finding a suitable image might be challenging due to the large number of existing images for each software system on DockerHub. Thus, we conducted an empirical study that compares DockerHub images to better understand their differences.

DockerHub contains a large number of choices of each single software system, while the particularities of images are not well documented. Our study shows that there are a variety of choices of each single software system on DockerHub since DockerHub images install different libraries and have different resources. We observe that community images are different from their corresponding official images, while they also differ among themselves in terms of their installed libraries (RQ1). Besides, images provide different sizes which indicate the installation of different resources (RQ2). However, developers do not document the particularities of their images. Neither do they document which libraries are installed nor which resources are available on their images.

Some of these image choices might be better than others in terms of improved resource efficiency and reduced security vulnerabilities. 3% (median) of the Debian based community images are at the same time more resource efficient and have less security vulnerabilities compared to the official images across the five studied software systems, as shown in Table 7 and Figure 19. For example, *"gcarre/java"* has a resource efficiency of 99.7% while containing 67 vulnerabilities compared to its official image, whose resource efficiency is 98% and contains 633 vulnerabilities.

While few images are better than the official images in both security and resource efficiency, there are other images that are better than the official image on just one of the two perspectives as shown in Figure 19. 26% (median), 49% (median), and 7% (median) of the Alpine, Debian, and Ubuntu based images are more resource efficient than their corresponding official image (RQ2) across the five studied software systems, while 7% (median) of the Debian based images have less security vulnerabilities than their corresponding official image (RQ3) across the four studied software systems, for which a Debian based

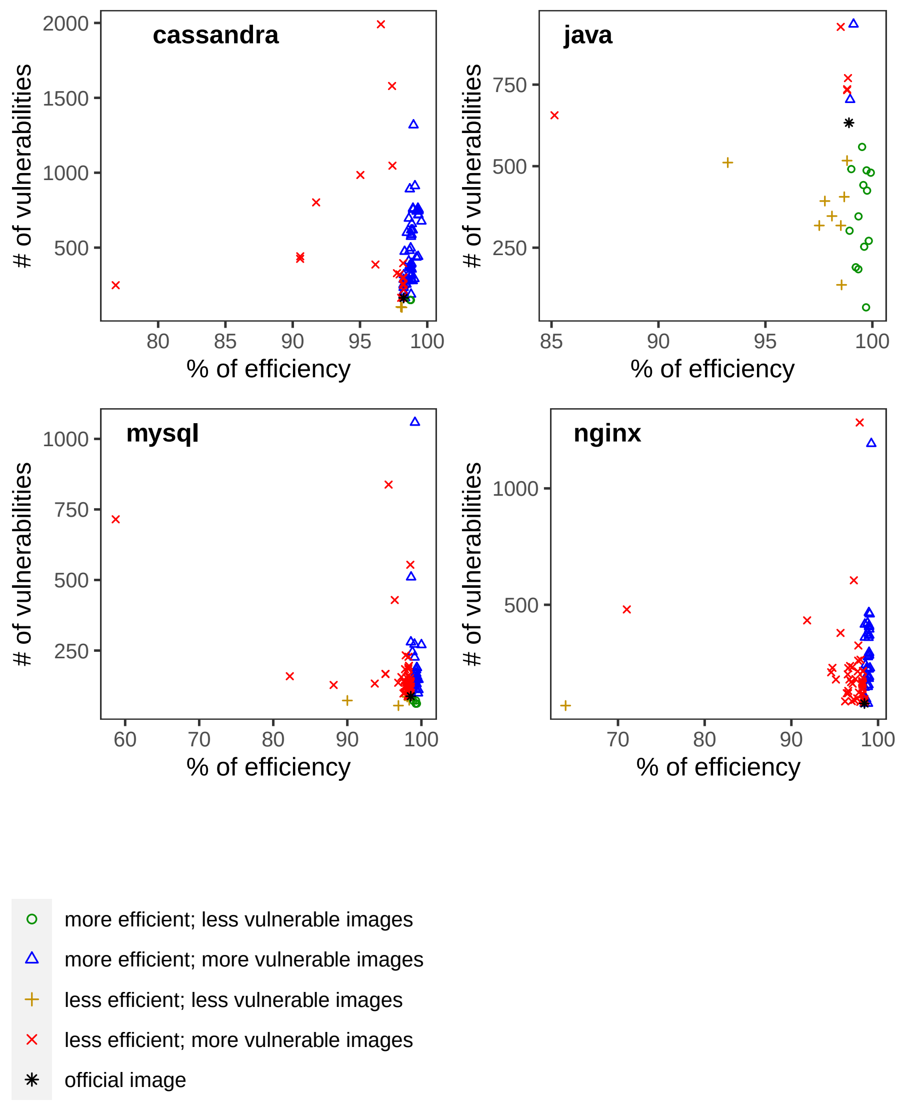


Fig. 19: The relation between the resource efficiency and security vulnerabilities of the images. We plot images based on their resource efficiency (RQ2) and number of vulnerabilities (RQ3). Note that the green images are better than the official images in terms of resource efficiency and number of vulnerabilities, while the red images are the worst images on both perspectives. Blue and yellow images are better than the official image on just one of the two perspectives. Note that we consider just Debian images since it is the only Linux distribution for which we have a dataset of vulnerabilities. We do not consider Mongo, since it does not have any Debian based official image.

Table 8: Spearman's rank correlations between different metrics of the images.

Image type	**# of libraries vs # of security vulnerabilities correlation coefficient (r_s)**	**Resource efficiency vs # of security vulnerabilities correlation coefficient (r_s)**
Cassandra	0.8	0.3
Java	0.4	-0.2
Mongo	0.6	0.08
MySQL	0.5	0.1
Nginx	0.5	0.2

official image exists. That is even though Docker regularly integrates security fixes and tracks the security of official images. From the other side, community developers do not document the efficiency and security perspectives of their images, which make them harder to identify.

While the number of security vulnerabilities might be associated with the number of installed libraries, we do not observe an association between the resource efficiency and the number of security vulnerabilities. We observe a very strong and a strong correlation between the number of vulnerabilities and the number of installed libraries on Cassandra and Mongo images, respectively, while we observe a moderate correlation for Java, Mysql, and Nginx as shown in Table 8. However, we observe a weak correlation between security vulnerabilities and resource efficiency for all the images as shown in Table 8.

The best images in terms of resource efficiency and security vulnerabilities are not explored by users. While resource efficient images and images with less security vulnerabilities compared to the official image exist, users neither download nor star these images. We do not observe any significant correlations (Spearman's correlation coefficient) neither between the resource efficiency and the number of downloads and stars nor between the number of security vulnerabilities and number of downloads and stars as shown in Table 9.

Summary

Our results suggest to Docker tooling creators and DockerHub should provide approaches to distinguish images based on their installed libraries, resources, resource efficiency, and security vulnerabilities.

7 Related Work

Prior studies on Docker images focus on their security, quality, and the evolution of Docker images. For example, Xu et al. [31] showed that mining Docker image repositories from DockerHub can provide insight into the use cases and the configurations of software systems. Tak et al. [29] observed that more than 92% of the DockerHub images are not free from any security or compliance

Table 9: Spearman's rank correlation coefficients between the resource efficiency, number of security vulnerabilities, and the popularity metrics.

Image type	Popularity metric	Resource efficiency				# of security vulnerabilities
		Alpine	Centos	Debian	Ubuntu	
Cassandra	#downloads	-	0.2	-0.1	0.02	0.06
	#stars	-	-0.02	-0.06	0.2	0.04
Java	#downloads	-0.2	-	-0.3	0.003	0.09
	#stars	-0.1	-	-0.002	-0.2	0.24
Mongo	#downloads	0.4	-	-0.2	0.04	-0.06
	#stars	0.08	-	0.05	0.06	-0.06
MySQL	#downloads	-0.4	-	-0.06	0.04	-0.03
	#stars	0.1	-	0.05	-0.1	-0.05
Nginx	#downloads	0.03	-	-0.09	0.07	-0.12
	#stars	0.05	-	-0.005	-0.08	-0.06

issue. Shu et al. [27] created a scalable Docker image vulnerability analysis (DIVA) framework which can discover, download, and analyze the images from DockerHub. They observed that on average more than 180 vulnerabilities are present on DockerHub images. Zerouali et al. [33] studied outdated and most up-to-date DockerHub images and observed that even the most up-to-date Docker images have severe security vulnerabilities. Zerouali et al. [32] proposed *ConPan*, a tool that identifies the characteristics of Docker installed packages (e.g., outdatedness, bugs, and security vulnerabilities). Zhang et al. [34] identified six different evolutionary patterns for Dockerfiles and investigated the impact of these different evolutionary patterns on the quality and the build latency of Docker images [35]. They observed that a decrease in the number of image layers and size along with more diverse instructions can significantly improve both the quality and the build time of Docker images. Lu et al. [22] address the problem of temporary files that are created during the build of an image and which increase its size.

Since DockerHub supports searching for images based on only the name, description, or username, Brogi et al. [5] proposed a tool named DOCKERFINDER which supports a multi-attribute search feature for Docker images. The tool can search Docker images based on the name, image size, or supported software distributions (e.g., python 2.7 and java 1.8) of an image. Although their research also helps users find out suitable Docker images with attribute-based searching, they did not compare the internal differences among DockerHub images. Chen et al.[7] proposed a semi-supervised model that proposes labels for a Docker image.

While prior studies conducted on Docker images focused on different aspects such as security, quality, and evolution of Docker images, we focus on identifying the differences among DockerHub images which appear to install the same software system according to their names.

8 Threats to Validity

8.1 External Validity

We selected five popular software systems then randomly selected a representative sample of their community images to mitigate any bias that can threaten the generalizability of our observations. Future studies should explore our findings on other systems as well as images.

The results are only applicable on DockerHub images since we only study images from DockerHub. The results may vary in other Docker image registry platforms such as Quay [9] or Google Container Registry [20]. Therefore, additional replication studies can be performed to identify the differences among the images on other Docker image registry platforms.

8.2 Internal Validity

A first threat to internal validity for threat concerns the collection of installed libraries. We only consider in this study the images that use the APT, APK, or RPM package manager to install their required libraries. Future studies should explore our findings for manually installed libraries as well as libraries that are installed using other package manager systems (e.g., NPM and PYPI).

Another internal threat to validity concerning the comparison of our studied images is related to the content of an image. One might install on top of a software system, additional services or even more systems, such as the "nginx-php-wordpress" image, which might contain on top of Nginx, PHP and WordPress. Thus, to mitigate this risk and fairly compare similar images, we consider images whose names are exactly the name of one of our studied software systems, as discussed in Section 3.1. That is because users might filter out the images that provide multiple services from their names when searching for images related to a software system. Future studies should investigate additional heuristics to better quantify the content of images from their available user artifacts (e.g., description and Dockerfile).

9 Conclusion

In this paper, we study the differences among DockerHub images. We observe in our preliminary study that images from DockerHub can save a considerable amount of effort when building, testing, and maintaining a Docker image. However, we observe that a large number of Docker image choices are reported when searching for any given software system on DockerHub, which makes choosing the most suitable image a nontrivial endeavor. Therefore, we empirically compare DockerHub images for the same software system in terms of their installed libraries, sizes, resource efficiency, and security vulnerabilities. We observe that users have a variety of choices for each software system in terms of installed libraries, while these choices vary in terms of resource efficiency and security. 26% (median), 49% (median), and 8% (median) of the community Alpine, Debian, and Ubuntu based images are more resource efficient than their respective official images across all the five studied software systems. 7% (median) of the community images have less security vulnerabilities than their respective official images across all the studied Debian based images. We also observe that developers do not document the particularities of their images regarding the resource efficiency and security; and users neither select the most resource efficient images nor images with less security vulnerabilities. Hence, we recommend that Docker tooling creators and DockerHub provide mechanisms for users to distinguish among images and select the most suitable images for their needs. Future studies are needed to investigate a large number of images, or other Docker image registry platforms as well as studying images along additional perspectives (e.g., outdatedness of images).

References

1. 451research. 451 research. https://451research.com, 5 2019. [Online; last accessed: 23 May, 2019].
2. A. Acharya, J. Fanguëde, M. Paolino, and D. Raho. A performance benchmarking analysis of hypervisors containers and unikernels on armv8 and x86 cpus. In *European Conference on Networks and Communications (EuCNC)*, pages 282–289, June 2018.
3. A. Bettini. Vulnerability exploitation in docker container environments. *FlawCheck, Black Hat Europe*, 2015.
4. Bitbucket. Bitbucket. https://bitbucket.org, 5 2019. [Online; last accessed: 23 May, 2019].
5. A. Brogi, D. Neri, and J. Soldani. Dockerfinder: Multi-attribute search of docker images. In *International Conference on Cloud Engineering (IC2E)*, pages 273–278, April 2017.
6. E. Carter. 2018 docker usage report. https://sysdig.com/blog/2018-docker-usage-report, 5 2019. [Online; last accessed: 23 May, 2019].
7. W. Chen, J.-H. Zhou, J.-X. Zhu, G.-Q. Wu, and J. Wei. Semi-supervised learning based tag recommendation for docker repositories. *Journal of Computer Science and Technology*, 34(5):957–971, Sep 2019.
8. J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall. An empirical analysis of the docker container ecosystem on github. In *14th International Conference on Mining Software Repositories (MSR)*, pages 323–333, May 2017.
9. CoreOS. Coreos quay. https://quay.io, 5 2019. [Online; last accessed: 23 May, 2019].
10. Datadog. Datadog. https://www.datadoghq.com, 5 2019. [Online; last accessed: 23 May, 2019].

11. Datadog. Docker adoption. https://www.datadoghq.com/docker-adoption, 5 2019. [Online; last accessed: 23 May, 2019].
12. Debian. Debian packages. https://packages.debian.org/, 2020.
13. Debian. Debian packages vulnerabilities. https://security-tracker.debian.org/tracker/data/json, 2020.
14. A. Decan, T. Mens, and E. Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *15th International Conference on Mining Software Repositories (MSR)*, pages 181–191, May 2018.
15. Docker. Docker. https://www.docker.com, 5 2019. [Online; last accessed: 23 May, 2019].
16. DockerHub. Dockerhub http api v2. https://docs.docker.com/registry/spec/api, 5 2019. [Online; last accessed: 23 May, 2019].
17. DockerHub. Official images on dockerhub. https://docs.docker.com/docker-hub/official_images, 6 2019. [Online; last accessed: 7 June, 2019].
18. DockerHub. Nginx docker images. https://www.docker.com/products/docker-hub, 3 2020. [Online; last accessed: 15 March, 2020].
19. Github. Github. https://github.com, 5 2019. [Online; last accessed: 23 May, 2019].
20. Google. Google container registry. https://cloud.google.com/container-registry, 5 2019. [Online; last accessed: 23 May, 2019].
21. Z. Li, M. Kihl, Q. Lu, and J. A. Andersson. Performance overhead comparison between hypervisor and container based virtualization. In *31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 955–962, March 2017.
22. Z. Lu, J. Xu, Y. Wu, T. Wang, and T. Huang. An empirical case study on the temporary file smell in dockerfiles. *IEEE Access*, 7:63650–63659, 2019.
23. D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. volume 2014, Mar. 2014.
24. N. Muhtaroglu, B. Kolcu, and İ. Arı. Testing performance of application containers in the cloud with hpc loads. In *Fifth International Conference On Parallel, Distributed, Grid And Cloud Computing For Engineering.* Civil-Comp, 2017.
25. Serverwatch. Container revenue growing to 2.7b by 2020. https://www.serverwatch.com/server-news/container-revenue-growing-to-2.7b-by-2020.html, 5 2019. [Online; last accessed: 23 May, 2019].
26. S. Shirinbab, L. Lundberg, and E. Casalicchio. Performance evaluation of container and virtual machine running cassandra workload. In *3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*, pages 1–8, Oct 2017.
27. R. Shu, X. Gu, and W. Enck. A study of security vulnerabilities on dockerhub. In *Seventh ACM on Conference on Data and Application Security and Privacy (CODASPY)*, pages 269–280, 2017.

28. Snyk. Snyk docker analyzer. https://github.com/snyk/snyk-docker-analyzer, 5 2019. [Online; last accessed: 23 May, 2019].
29. B. Tak, H. Kim, S. Suneja, C. Isci, and P. Kudva. Security analysis of container images using cloud analytics framework. In *2018 Web Services (ICWS)*, pages 116–133, Cham, 2018.
30. Wagoodman. Dive. https://github.com/wagoodman/dive, 5 2019. [Online; last accessed: 23 May, 2019].
31. T. Xu and D. Marinov. Mining container image repositories for software configuration and beyond. In *40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*, pages 49–52, May 2018.
32. A. Zerouali, V. Cosentino, G. Robles, J. M. Gonzalez-Barahona, and T. Mens. Conpan: a tool to analyze packages in software containers. In *Proceedings of the 16th International Conference on Mining Software Repositories*, pages 592–596. IEEE Press, 2019.
33. A. Zerouali, T. Mens, G. Robles, and J. M. Gonzalez-Barahona. On the relation between outdated docker containers, severity vulnerabilities, and bugs. In *26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 491–501, Feb 2019.
34. Y. Zhang, H. Wang, and V. Filkov. A clustering-based approach for mining dockerfile evolutionary trajectories. volume 62, pages 19101:1–19101:3. Science China Press, 2019.
35. Y. Zhang, G. Yin, T. Wang, Y. Yu, and H. Wang. An insight into the impact of dockerfile evolutionary trajectories on quality and latency. In *42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 01, pages 138–143, July 2018.
36. C. Zheng and D. Thain. Integrating containers into workflows: A case study using makeflow, work queue, and docker. In *8th International Workshop on Virtualization Technologies in Distributed Computing (VTDC)*, pages 31–38, 2015.