Mining Container Image Repositories for Software Configuration and Beyond*

Tianyin Xu and Darko Marinov
University of Illinois at Urbana-Champaign
{tyxu,marinov}@illinois.edu

ABSTRACT

This paper introduces the idea of mining container image repositories for configuration and other deployment information of software systems. Unlike traditional software repositories (e.g., source code repositories and app stores), image repositories encapsulate the entire execution ecosystem for running target software, including its configurations, dependent libraries and components, and OS-level utilities, which contributes to a wealth of data and information. We showcase the opportunities based on concrete software engineering tasks that can benefit from mining image repositories. To facilitate future mining efforts, we summarize the challenges of analyzing image repositories and the approaches that can address these challenges. We hope that this paper will stimulate exciting research agenda of mining this emerging type of software repositories.

CCS CONCEPTS

• **Software and its engineering** → Software libraries and repositories; Software post-development issues; Software configuration management and version control systems;

KEYWORDS

Container; image; Docker; configuration; software repository

ACM Reference Format:
Tianyin Xu and Darko Marinov. 2018. Mining Container Image Repositories for Software Configuration and Beyond. In *ICSE-NIER'18: 40th International Conference on Software Engineering: New Ideas and Emerging Results Track, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183399.3183403

1 INTRODUCTION

Mining software repositories (MSR) has been proven to be an effective approach for discovering, characterizing, and understanding software engineering practices, towards improving software productivity and quality. Existing MSR studies mostly focus on *software development* by mining code repositories (including source code, commit histories, bug reports, and documentation) [4, 5, 8] and *software release* by mining app stores and package repositories [1, 3].

Table 1: A comparison of image versus code repositories.

	Container image repo	Source code repo
Usage	system operation	software development
User	sysadmins and operators	developers
Store	Docker Hub, Docker Store, ...	GitHub, Sourceforge, ...
Content	executables + exec. context	source code
Configuration	customized	default/customizable
Scope	entire software stack	specific project
Evolution	different software versions	source code changes

Few studies cover field configurations of software systems (e.g., for deployment and orchestration). In fact, information of field configurations is highly desired, not only by operators and sysadmins to learn best practices, but also by developers and DevOps engineers to measure software's usability and manageability.

One fundamental obstacle to the study of configurations lies in the fact that traditional software repositories such as source code repositories and app stores contain little information of how the software is actually being used in the wild. Historically, studying field configurations used ethnographic methods and manual data collection from second-hand data sources [9]. For example, a study of how software is configured in the field [9] took six person-month to collect configuration files attached in issue reports on mailing lists and online forums. However, this dataset, despite the only one of its kind, is highly biased to misconfiguration cases and is incomplete—it is hard to determine the values referencing to execution context (e.g., environment variables and file content).

In this paper, we advocate that container image repositories, as an emerging type of software repositories, provide a plethora of opportunities to study configurations and other field operations for a variety of software. Unlike source code repositories for software development, container images are used for operations. A container image is defined as a stand-alone, executable package of a piece of software[1] that includes everything needed to run it: binary code, configuration files, system libraries, language runtime, and management tools. Most of this information is not directly included in traditional software repositories. Table 1 compares container image repositories with traditional source code repositories.

Most importantly, the wide adoption of containerization techniques drives the proliferation of image sharing. According to Docker Hub's statistics, it has hosted 100K+ public image repositories contributing to 900K+ images, serving 12+ billion image pulls per week. Besides a small number of *official* image repositories from certified software vendors (e.g., Apache, Oracle, and Red Hat), most of the repositories are shared by individual users and organizations,

*An extended version can be found at [11].

[1] Containers are often designed for the microservice architecture in which each container runs one software service, so each image has its target software.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE-NIER'18, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5662-6/18/05…$15.00
https://doi.org/10.1145/3183399.3183403

containing various customization, integration, and orchestration, to serve their own use cases. Therefore, these image repositories form a massive information base of configuration and operation practices for mining and analysis.

We present the *opportunities* and *methods* for mining image repositories based on our experience of working with image data. We focus on repositories of Docker-based container images (a.k.a., *Docker images*), the *de facto* image format adopted in industry, and Docker Hub as the current largest online registry service for *public* Docker images. Our objective is to showcase the rich data and information encoded in image repositories, and more importantly, describe how several software engineering tasks—ranging from configuration design to software orchestration to combinatorial testing—can potentially benefit from or be enabled by mining these repositories (cf. §3). To facilitate future mining efforts, we summarize the methods of mining image repositories (refer to [11] for the challenges and limitations). We hope that this paper will stimulate exciting research agenda of mining the emerging image repositories.

2 IMAGE REPOSITORIES

This section goes over several preliminaries of container images and their repositories from the perspective of mining and analysis, which establishes the context necessary to understand the technical content presented in the subsequent sections.

Image organization. In essence, an image is a filesystem-level snapshot that includes all the files needed for launching a running system instance (i.e., a container). For Docker, images are organized as a series of *layers* stacking on top of one another. Each layer is created by a build instruction specified in the image's Dockerfile (i.e., Docker's build file for specifying the instructions that can be executed to assemble an image, similar to Makefile for building an executable from source code). Each layer consists of the filesystem `diff` (files added or deleted) introduced by executing that instruction on the layer below it. Stacking all the layers comprises the unified view of the image. Note that layers (identified by unique LayerIDs) can be shared across multiple images, e.g., one can create a new image by adding new files onto the `ubuntu` image, and the new image shares all the layers of `ubuntu`.

An image can be `pull`ed from and `push`ed to registry services such as Docker Hub. The image's metadata and Dockerfile can be fetched through the `inspect` command or the Docker's REST APIs.

Image repositories. An image repository on Docker Hub contains multiple images with different *tags* (typically used for annotating versions). An image on Docker Hub is identified by the repository name and the tag, for example, `ubuntu:16.04` refers to the image with the tag `16.04` in the `ubuntu` repository. All the tags, together with other metadata of the repositories (e.g., description, maintainer, community rating and comments, and update time) can be queried through the Docker's REST APIs.

Image repositories can be searched based on keywords. Although Docker Hub does not provide the entire list of image repositories, Shu et al. [7] show that a dictionary-based search method can collect the vast majority of public repositories on Docker Hub.

Containers. Containers are runtime instances launched by `docker run` images. A container's flat filesystem differs from the original image which is organized in layers. Moreover, the container creates (virtual) files for device drivers and `procfs` (`/dev` and `/proc`) based on the host OS, which are not included in images. Also, containers typically execute initial instruction (specified in Dockerfiles) to run the target software, which creates new files (e.g., logs and traces).

3 OPPORTUNITIES

This section showcases the research opportunities for software engineering enabled by the unique data encoded in container image repositories. Note that container images are supposed to run out of the box, without the need of additional configuration efforts—the data in image repositories are working samples rather than demos.

Creating a feedback loop for configuration design. One key aspect of configuration design is the trade-off between flexibility (configurability) and complexity (usability), which should be carefully made with a user-centric design philosophy, as configuration is essentially an interface for users to control and customize software behavior [9]. Feedback loops should be created to help developers understand how their software is configured in the wild, in order to tune the usability accordingly.

Furthermore, as shown in prior work [13], configuration requirements can change over time—a correct configuration value in an old version could be obsolete or become invalid (producing undesired behavior) after software upgrade. Understanding the characteristics of configuration changes through software evolution is critical to software configuration design and maintenance.

Historically, attacking the above problems is difficult, especially for open-source software projects, due to the lack of publicly available datasets [12]. Unlike source code for which there are many open-source online repositories (e.g., GitHub), software configurations are independently maintained by sysadmins and operators who have no incentives to share their settings. Some companies do collect customers' configuration values, but few of them are willing to open such data to public as configuration settings often contain sensitive, confidential information. Therefore, existing studies on field configurations are either by the companies (which are specific to one or two products), or based on tedious, time-consuming data collection effort (as discussed in §1).

With container image repositories, the usage statistics of configuration parameters can be collected by analyzing the configuration files in the image repositories built for the same piece of software. For popular software (e.g., those studied in [9]), there are typically thousands of image repositories made for different use cases and scenarios, containing a diverse set of configuration settings.[2] Moreover, as image repositories contain different versions of the target software and the configurations working for each version, mining these repositories enables the opportunities to understand software configuration with software evolution in depth.

Modeling cross-component configuration dependencies. Misconfigurations across multiple software stacks or components are among the most urgent but thorny problems in software reliability [12]. One fundamental obstacle in dealing with these misconfigurations lies in the challenges of understanding and modeling

[2] As a comparison, `mysqld` and `httpd` studied in [9] have 9133 and 2006 image repositories (which contain the corresponding configuration files with different versions of the software) on Docker Hub, respectively, while the dataset in [9] only contains 823 and 311 configuration files of these two software projects, respectively.

dependencies of configurations across components. Existing studies attempt to understand cross-component dependencies based on user-reported issues posted on mailing lists and online forums [6]. However, the user-generated data cannot help understand the unknown unknowns or model the complete dependency information, not to mention the tremendous overhead of collect them.

Mining image repositories provides opportunities of unraveling such information, as images encapsulate the complete environment for running target software from the OS kernel to user-level applications. Many images are built for system infrastructure made up of different components (each as a microservice) that have been configured to work together. Therefore, images provide an open dataset of rich, extensive, and concrete configuration values recorded in configuration files, databases, and system environment. Unlike a second-hand dataset in which configuration values are treated as isolated string literals, image repositories associate these values with their context, including the executable code, resources/entities referenced by these values, and dependent software components.

Discovering software orchestration. Unlike source code repositories dedicated for a specific piece of software, image repositories often serve as building blocks for large-scale, complex systems composed of multiple software components. These software components can either be packed into a single image (e.g., the image `wordpress:php7.1-apache` as a web stack), or form distributed systems running on top of multiple images maintained in separate repositories (e.g., the Hadoop-based data processing framework published by `uhopper` that is composed by `hadoop-namenode`, `hadoop-datanode`, `hadoop-spark`, and other `hadoop-*` repositories). Therefore, image repositories are great resources for studying how different software components (and their versions) are glued together and orchestrated as a service. Such study can not only reveal the glue logic planned by software developers, but also potentially discover spontaneous use cases invented by power users.

Note that for the case of multiple images, it takes additional effort to collect orchestration information of these images, as each image by itself does not explicitly specify the other images it connects to. One data source are "compose files" used by `docker compose` which specify how multiple containers are orchestrated from images.

Improving combinatorial testing and tuning. Mining image repositories can be used to understand common combinations and value distributions of binaries and configurations, in order to help test prioritization, performance tuning, and/or security auditing.

Testing of configurable software (e.g., a software product line, SPL) requires not only executing the software for certain inputs but also applying these inputs with various combinations of features. One key challenge is to select the subset of combinations that are representative and cover typical use cases, as testing all possible combinations is not feasible (e.g., an SPL with 10 configurable features can have more than 2^{10} distinct configurations). While combinatorial methods can explore various combinations of configurations, they are still quite costly, and may focus on irrelevant combinations rarely used in practice. Mining image repositories can discover combinations that are actually used, allowing both speeding up testing and finding bugs for relevant configurations.

While combinatorial testing for functional correctness requires checking all combinations that arise in practice, performance tuning can be biased toward the most frequent configuration settings to optimize expected runtime (over the distribution of configurations). Understanding how the software is actually used can also help developers better tune the performance of the software by focusing on common systems environment and configuration settings.

Using images as test beds for software engineering tools. Image repositories can serve as real-world test beds for research tools, including misconfiguration detection, binary analysis for malware detection, portability testing, performance auto-tuning, etc. Taking misconfiguration detection as an example, existing research efforts mostly evaluate the proposed methods and tools on self-injected errors or a small set of known misconfigurations [10]. However, it is hard to measure the actual benefits in large-scale real-world deployments. Image repositories can be used to quantitatively answer such questions, as they form a diverse, comprehensive dataset of real-world configurations and their context. We envision such test beds to be built on top of existing image repositories.

4 MINING METHODS

This section describes the methods for analyzing container image repositories, including the process and techniques for addressing the challenges derived from the characteristics of images [11].

Stream-based mining. Due to the large sizes of image repositories [11], image repositories mining needs to adopt the *stream*-based process if it cannot afford mirroring all the repositories locally. A stream-based method extracts the target information continuously after images are loaded into memory/disks, and then removes these images to make space [7]. This can be done by either *static* or *dynamic* method based on whether to `run` the images:

- *static* methods analyze the `tar` archive of an image saved on local storage. As introduced in §2, an image is organized as a series of layers in the form of filesystem `diffs`, which can be composed to create a unified filesystem hierarchy. The files of interest can be extracted;
- *dynamic* methods first launch containers from the target images and then collect information of interest by invoking mining and analysis code inside the containers (which requires to `copy` the code into the container's filesystem and `copy` the analysis results from the container out to the host). The code has the capability to invoke the commands/utilities in the container.

In comparison, static methods are more lightweight (without the need to run containers); they are also conceptually simpler as all information is encoded in the files inside `tar` archives and can be analyzed through a uniform file-based processing framework. Dynamic methods can precisely capture runtime information, with the cost of complexity due to the heterogeneity of containers.

Downloading images with shared layers. Images are organized in the granularity of layers (cf. §2). Each layer of an image is pulled down separately, and stored in the host machine. If multiple images share the same layers (e.g., built upon the same OS image), these layers only need to be downloaded once. As a result, downloading images with shared layers in batches can save significant storage and downloading overhead, compared with treating each image independently. Typically, images from the same repositories share common layers and can be batched together, as they likely share

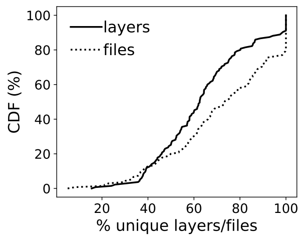

Figure 1: % unique layers (files) among all layers (files) in each official image repository.

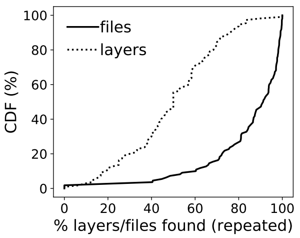

Figure 2: % files (layers) needed to analyze with official images as the knowledge base.

many base layers. Figure 1 shows the percentage of unique layers across all the layers in each official repository on Docker Hub (there are 143 official repositories)—batching the downloads can save 35+% layers for 50+% repositories. A more sophisticated approach is to leverage the `FROM` instruction in Dockerfile that specifies the base image, from which the target images were built.

Layer-based analysis. Similar to downloading, the image mining/analysis should be designed and implemented based on layers. Layers that have been processed should be recorded to avoid repeated computing effort. To illustrate the efficacy of layer-based analysis, we pull the 143 official image repositories from Docker Hub, and record the MD5 checksum of every file in each layer in a database (serving as the knowledge base). Then, we randomly sample 100 image repositories from Docker Hub and select the `latest` image in each repository. Figure 2 shows the files with MD5 found in the knowledge base—on average, 83.4% of the files hit a small base of 143 official repositories, even though the coverage of exact layers (based on LayerIDs) is much lower. The main reason of such significant coverage of files is that most files in an image come from the OS and libraries. As there are limited OS distributions and library versions, layer-based mining can lead to significant savings.

Selective mining. Not every image in a repository is worth mining for a specific software engineering task. For example, many images are for the same application binaries and configurations, but wrapped around different OS distributions or libraries. If the information of interest lies in the application itself, only one of the images needs to be downloaded and analyzed.[3]

Leveraging Dockerfile. A Dockerfile records how an image is created (cf. §2). The Dockerfile of an image can be fetched through Docker's REST API if available. A lot of information of images can be collected and inferred by analyzing Dockerfiles, without the need to download and mine the images. Unfortunately, as reported in [2], many Dockerfiles are not reproducible due to missing version pinning—34% of Dockerfiles were not able to build the images.

5 RELATED WORK

Prior studies on Docker images mostly focus on analyzing Dockerfiles as a special type of code [2] and the security implications of adopting Docker images [7]. Differently, our focus is not about how they were created and how secure to deploy them, but about the data and information that can be distilled from the images for the good and evil of software engineering research.

Prior studies on mining software repositories mainly focus on source code repositories (including version-control systems and bug databases), archived communications, and app stores [1, 3–5, 8]. With the wide adoption of containerization techniques, container images have become emerging data which encode information unavailable in traditional software repositories. This paper advocates opportunities of mining container image repositories, as a special type of software repositories, to compliment prior work.

Besides Docker images, virtual machine (VM) images are also available online, such as AMI (Amazon Machine Images) used for deploying VMs on Amazon EC2. On the other hand, AMIs do not have the same level of popularity as Docker Hub. Moreover, AMIs do not have the notion of "repositories" but are traditional disk images which contain less semantic information.

6 CONCLUDING REMARKS

In this paper, we advocate for mining container image repositories, as a special and emerging type of software repositories, for understanding configurations and use cases of software systems. The motivation derives from the observation that few existing studies have paid attention to container image repositories, or have explored the unique, rich data and information which is not available in traditional software repositories. To stimulate future research, we have discussed the opportunities of mining container image repositories, followed by the mining methods. We hope that image repositories mining can fill the gap between in-house software development and the operations of software systems.

Acknowledgement. We thank the anonymous reviewers of ICSE NIER for their valuable comments that help improve the presentation. Darko Marinov's group is supported by NSF grants CCF-1409423, CCF-1421503, CNS-1646305, and CNS-1740916.

[3] Specifically, `Alpine` Linux is the OS distribution officially adopted by Docker since 2016, which is an order of magnitude smaller than `ubuntu` (the previous default). Therefore, images based on `Alpine` are often the choice for downloading and analysis.

REFERENCES

[1] Abate, P., Cosmo, R. D., Gesbert, L., Fessant, F. L., Treinen, R., and Zacchiroli, S. Mining Component Repositories for Installability Issues. In *MSR'15*.

[2] Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S., and Gall, H. C. An Empirical Analysis of the Docker Container Ecosystem on GitHub. In *MSR'17*.

[3] Harman, M., Jia, Y., and Zhang, Y. App Store Mining and Analysis: MSR for App Stores. In *MSR'12*.

[4] Hassan, A. The Road Ahead for Mining Software Repositories. In *2008 Frontiers of Software Maintenance*.

[5] Nagappan, N., Zimmermann, T., and Zeller, A. Guest Editors' Introduction: Mining Software Archives. *IEEE Software 26*, 1 (January 2009), 24–25.

[6] Sayagh, M., Kerzazi, N., and Adams, B. On Cross-stack Configuration Errors. In *ICSE'17*.

[7] Shu, R., Gu, X., and Enck, W. A Study of Security Vulnerabilities on Docker Hub. In *CODASPY'16*.

[8] Xie, T., Thummalapenta, S., Lo, D., and Liu, C. Data Mining for Software Engineering. *IEEE Computer 42*, 8 (August 2009), 55–62.

[9] Xu, T., Jin, L., Fan, X., Zhou, Y., Pasupathy, S., and Talwadker, R. Hey, You Have Given Me Too Many Knobs! Understanding and Dealing with Over-Designed Configuration in System Software. In *ESEC/FSE'15*.

[10] Xu, T., Jin, X., Huang, P., Zhou, Y., Lu, S., Jin, L., and Pasupathy, S. Early Detection of Configuration Errors to Reduce Failure Damage. In *OSDI'16*.

[11] Xu, T., and Marinov, D. Mining Container Image Repositories for Software Configuration and Beyond. *arXiv:1802.03558*.

[12] Xu, T., and Zhou, Y. Systems Approaches to Tackling Configuration Errors: A Survey. *ACM Computing Surveys (CSUR) 47*, 4 (July 2015).

[13] Zhang, S., and Ernst, M. D. Which Configuration Option Should I Change? In *ICSE'14*.