Large-scale analysis of Docker registry dataset

Your N. Here
Your Institution

Second Name
Second Institution

Abstract

Docker containers are becoming increasingly popular due to their isolation properties, low overhead, and efficient packaging of execution environment. Containers are created from images which preserve software dependencies, environment configuration, and other parameters that affect application's runtime. Docker registry stores images and allows clients to push images to and pull images from it. The total amount of images stored in Docker Hub registry is 457,627 now and is continuing to increase. As the amount of images stored in public and private Docker registries increases it becomes important to study images' characteristics. Such knowledge can help to improve storage performance both at Docker registry and Docker client sides.

Our goal is to collect statistics from a large amount of Docker images and perform a large-scale characterization of Docker images. So far, we downloaded 143,784 images from Docker Hub registry (20TB, 31% of the whole dataset) and performed the first in-depth analysis for these images. We characterize images using multiple metrics, e.g., image size distribution, file size and type distribution, the number of layers per image, the amount of redundant data between images and layers, temporal trends.

1 Introduction

outline:
1. container and registry are becoming populary
2. but we dont know the statistic characterization about registry
3. This paper provide a first in-depth analysis
4. contribution: interesting findings

2 Background

TODO:
1. complete:
2. search for reference

2.1 Docker

2.1.1 Docker image

An image is a combination of a JSON manifest and individual layer files.

1. Manifest.

2. Config file.

3. Layer files.

2.1.2 Storage driver

The storage driver controls how images and containers are stored and managed on Docker host using a pluggable architecture.

1. aufs.

2. zfs.

3. overlay/overlay2.

4. devicemapper.

2.1.3 Docker container

Containers are an abstraction at the app layer that packages code and dependencies together.

1. writable layer.

2. Copy-on-write.

3. Data volumes.

2.2 Docker Registry

The Registry is a stateless, highly scalable server side application that stores and lets you distribute Docker images.

1. Registry versions.

2. Content addressability.

3. Pulling images.

2.3 Current public registries

3 Methodology

TODO:
1. Polish
2. Complete fig and table

Our methodology has two steps. The first step is to massively download the Docker images from Docker registry. When the images are downloaded, we analyze them and calculate statistics distribution for different metrics. The details of each step are covered in the following sections.

3.1 Downloader

Instead of using Docker (or Docker Engine) to download images, we wrote our own downloader python script that utilizes Docker Registry API [] to simultaneously download **original** manifests and layers. There are two reasons. First, Docker Engine (starting from version 1.10) automatically converts the manifests from schema version 1 to schema version 2, which affects our results about manifest version statistics; Second, layer content directories are not visible for some Docker storage drivers, e.g., devicemapper, which is not feasible to analyze the layer content.

Downloader can download multiple images simultaneously and within each image downloading process, layers are downloaded in parallel. To download the original manifests and layers from Docker registry, downloader embeds a Docker registry client API [**?**] which only encapsulates manifest, config file and layer downloading functions in Docker engine without extracting layer tarball and converting manifest version.

To download an image, the name should be provided. To the best of our knowledge, Docker Hub (i.e., Docker registry) doesn't provide a method to list their public images. Public images in Docker registry can be divided into official images and non-official images. The amount of official images is only xxx. While estimating and listing all the non-official images requires crawling Docker Hub. We created Crawler python script to crawl Docker Hub websites and list both official and non-official images.

3.1.1 Crawler

Docker Hub website provides search engine which indexes public images for users to search for a specific image or a list of images that contains a certain letter or string. The name of non-official public image is comprised of "$\langle namespace \rangle / \langle repositoryname \rangle$", where *namespace* is the user name. In this case, we search for '/' and obtain a list of images which contains '/'. In other words, this method lists all the non-official public images in Docker Hub. Crawler downloads all the pages which contains '/'. Once web pages are downloaded, it parses the web content and build a list of non-official images.

A interesting observation is that Crawler can get a similar list of images if we replace '/' with '*'. Note that from 5/30/2017-7/11/2017, Crawler used above method to obtain the total amount of images in Docker Hub. But after 7/11/2017, the Docker Hub removed the index of '/'. Thus, currently we search for '*' instead of '/' to obtain a list of non-official public images.

3.1.2 Downloading images

As shown in Figure **??**, downloader first obtains a list of public images through crawling Docker Hub. Then, it starts downloading process. It downloads two components: manifest and individual layer files.

The first step in downloading an image is to fetch the manifest by using the following url: $GET/v2/\langle name \rangle / manifests / \langle reference \rangle$, where *name* parameter refers to $\langle namespace \rangle / \langle repositoryname \rangle$. Note that the *namespace* of official is *library*. The reference can include a tag or digest. Note that we only downloaded the images with *latest* tag to shorten the downloading process.

Layers are stored as compressed tar archive in the registry, indexed by digest. As discussed in Section **??**, manifest consist of multiple layer digests. Note that Schema 2 version also contains a config file digest. Once the manifest is downloaded, the downloader will then use the digests to download individual layers (including config file for Schema 2 version) by using the following url: $GET/v2/\langle name \rangle / blobs / \langle digest \rangle$, where *name* refers to the image name while *digest* refers to the layer digest or config file digest

3.1.3 Docker image dataset statistics

Crawler delivered a list of xxx images on 5/30/2017. However, duplicated images exist in the list. After re-

ducing the repeated images, our image dataset consists of xxx distinct images.

3.2 Analyzer

3.2.1 Layer profile

3.2.2 Config profile

3.2.3 Image Profile

4 Results

4.1 Image

4.1.1 Growth of images at docker registry

4.1.2 Image size distribution

4.1.3 Image popularity distribution

4.1.4 Layer depth distribution

4.1.5 Repeat layer count distribution

4.1.6 File count distribution

4.1.7 Repeat file count distribution

4.1.8 Compression rate distribution

4.1.9 Other metrics, exectution envirnoment

4.2 Layer

4.2.1 Layer size distribution

4.2.2 File count distribution

4.2.3 Directory count distribution

4.2.4 Directory depth distribution

4.2.5 Repeat file distribution

4.2.6 File size distribution

4.2.7 File type distribution

4.2.8 Compression rate distribution

4.2.9 Other metrics, layer age

5 Relatedwork

do we need a relatedwork?

6 Conclusion

The downloading process took roughly 30 days to finish. Overall, we downloaded xxx images with xxx layers as shown in Table **??**. xxx of images couldn't be downloaded. There are two reasons: first, xxx of images were either deleted or empty. Second, xxx of images doesn't have tag: *latest*. As we discussed in Section **??**, we only downloaded the images with latest version to shorten the downloading process.