Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

A Tool for Analyzing and Browsing Docker Registries

Relatore: *Francesca Arcelli Fontana*

Co-relatore: *Marco Zanoni*

Relazione della prova finale di:

Simone Erba

Matricola 793250

Anno Accademico 2016-2017

Summary

Figures Index

1. Introduction

1.1. Context

Docker (DockerDocs, 2017) is the world's software container platform. Docker is based on the image and container mechanism. An image is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, operating system’s base libraries, environment variables, and configuration files. An image it's always based on other images. The image on which an image is based is called parent image. A base image is an image with no parents. An official image is an image maintained by the Docker team.

A container is a runtime instance of an image. It runs completely isolated from the host environment by default, allowing access only to host files and ports if configured to do so. Containers encapsulate a whole operating system.

Docker use the host machine’s kernel, that is the only kernel in execution. Containers are instantiated as processes.

Docker allows Developers to eliminate local environment problems when collaborating on code with co-workers. Operators use Docker to run and manage apps side-by-side in isolated containers to get better compute density. Enterprises use Docker to build agile software delivery pipelines to ship new features faster, more securely and with confidence for both Linux and Windows Server apps.

In order to manage image versioning, it’s crucial to use tags. Tags are meta-data used to distinguish versions of Docker images, they are useful to preserve older copies or variants of a primary build.

To provide connections between users and images Docker created the Docker Hub (DockerDocs, 2017). Docker Hub is a cloud-based registry service that allows to link to code repositories, build images and test them, stores manually pushed images, and links to Docker Cloud (a service for managing containers hosted on cloud providers) so it's possible to deploy images to the hosts. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

Docker Hub is not the only existing registry. For example, Quay.io is another registry. It is also possible for users to create private registries and to pull images from public registries. Nevertheless, Docker Hub is the biggest public registry at the moment.

Docker can build images automatically by reading the instructions from a Dockerfile (DockerDocs, 2017). A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Images can be built from a Dockerfile using the 'docker build' command, using it users can create an automated build that executes several command-line instructions in succession.

The advantage of a Dockerfile over just storing the binary image (or a snapshot/template in other virtualization systems) is that the automatic builds will ensure you have the latest version available. This is a good thing from a security perspective, because it guarantee that the software is not vulnerable. It also allows light way replicability of production and development environment. This approach is derived and inspired by configuration management systems (like puppet, chef, ansible, or vagrant).

As shown in the figure 1, Docker containers don't contain Guest OS, so the waste of CPU-resource and storage is small. For this reason, boot-time, time of generating and distributing images are short. This is the benefit of using Docker cloud when compared with VM Cloud. Of course, it has weakness as well. VM is operated individually enough to be expressed like it provides new computer to the user. Because of this, it is easy to manage and apply the policy of system, network, user, security. In addition, user can use various virtualization OS regardless of Host OS. The startup time for a virtual machine is usually around some minutes, while for containers is few seconds. (Kyoung-Taek Seo, 2014)

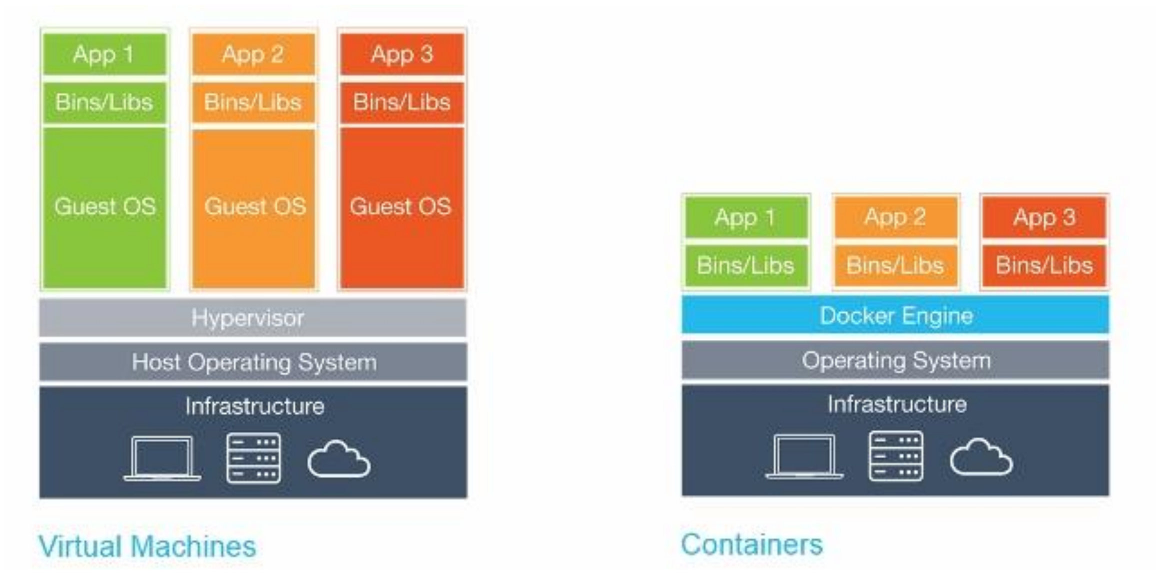


Figure 1 Virtual machines and Containers [https://www.docker.com/what-docker]

1.1.1. Virtualization

Hardware virtualization refers to the creation of a virtual machine that acts like a real computer with an operating system. Software executed on these virtual machines is separated from the underlying hardware resources.

Virtualization is a technology that allows to create multiple simulated environments or dedicated resources from a single, physical hardware system. Software called a hypervisor connects directly to that hardware and allows to split one system into separate, distinct, and secure environments known as virtual machines (VMs). These VMs rely on the hypervisor’s ability to separate the machine’s resources from the hardware and distribute them appropriately.

The original, physical machine equipped with the hypervisor is called the host, while the many VMs that use its resources are called guests. These guests treat computing resources like CPU, memory, and storage as a hangar of resources that can easily be relocated. Operators can control virtual instances of CPU, memory, storage, and other resources, so guests receive the resources they need when they need them. (VirtualizationRedHat, 2017)

Virtualization is used in software development too. The Java Virtual Machine or the .NET Framework are virtual machines used to compile, link and run software. They can hide the host’s physical resources and operative system in order to make the software execution independent from the local environment.

1.1.2. DevOps

DevOps is a software development process that emphasizes communication and collaboration between product management, software development and professional operations. DevOps also automates the process of software integration, testing, deployment and infrastructure changes. It aims to establish a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably.

In traditional, functionally separated organizations, there is rarely a cross-departmental integration of these functions with IT operations. DevOps promotes a set of processes and methods for thinking about communication and collaboration between departments of development, quality assurance, and IT operations. In some organizations, this collaboration involves embedding IT operations specialists within software development teams, thus forming a cross-functional team.

There are several advantages of using Docker in DevOps instead of traditional development using external libraries. First, libraries can be very big in term of space usage, while Docker containers are very light. Using external libraries can be dangerous because they may have malicious code inside them. Docker containers are isolated and resources are limited, even if one application is hacked, it will not affect others containers. In addition, there are always minor differences between environments in development and environments in the release, this can bring to unexpected behavior from the software. Libraries used in the traditional development process can be incompatible with the user's environment, while Docker containers are portable and can be continuously integrated and tested.
(Nachmani, 2015)

1.2. Thesis motivation

There are plenty of tools and programs made for help Docker users and developers. Here is a table with the most relevant ones:

Table 1 Most relevant tools to help with Docker deploy and management

Quay.io	Secure hosting for private Docker repositories.
Devstep	Development environments powered by Docker
Clocker	Clocker creates and manages a Docker cloud infrastructure. Clocker supports single-click deployments and runtime management of multi-image

	applications.
Dockunit	Docker based integration tests. A simple Node based utility for running Docker based unit tests.
deploy	Git and Docker deployment tool. A middle ground between simple Docker composition tools and full blown cluster orchestration
Captain	Convert your Git workflow to Docker containers ready for Continuous Delivery

Here there is the full list of them: https://veggiemonk.github.io/awesome-docker/#developer-tools. Some of them are services to host images like the Docker Hub. Others are secure, light and trusted base images ready to being used. There are a lot of image builders and tools for manage and log local images and containers. All the services available for Docker users are focused on the local deployment and don't provide a larger understanding of the interactions between images. The motivation for this work comes from this fact.

This tool wants to be a helpful service for Docker users, programmers and system administrators. It offer a graphical representation of all the dependencies between Docker images. It allows to navigate, analyze and understand connections between images. Often Docker users don't have enough understanding of other images. They may also not use the best parent images for their projects, this tool want to help them with this important choice. This project also wants to help Docker experts in perform analysis about how Docker users use the dependency mechanism. The motivation is to help the Docker Community.

Personally, I chose this project because it allowed me to don’t focus in only one argument, but to face many problems and solve them. I put together concepts from different courses and disciplines to have a larger view. I created a tool that will be useful in a real life scenario and I gained knowledge about a new popular technology like Docker. Most important, I understood the model behind a popular container software platform. It helped me in managing my time and resources, in being more independent and confident.

1.3. Purpose

The purpose is to find relationships between all the existing projects on the Docker Hub, save them and provide an easy to browse representation of them. I called this tool DockerSurfer. It wants to make the Docker dependency mechanism and its effects more understandable to humans. It also can help Docker users to find new images and the most popular and stable ones. The purpose is to write a fast, easy to understand and useful tool for browsing and analyzing direct, indirect and inverse relationships between Docker images. DockerSurfer also wants to make Docker users aware of changes in images that they are interested in and wants to provide useful information about images, that can’t be found elsewhere.

\

1.4. Outline

Chapter 2: Overview of the tool

In this chapter the structure of the DockerSurfer will be introduced along with some technologies used by it, such as graph databases. The interactions between users, DockerSurfer and data sources will be also showed

Chapter 3: Design of the tool

In this chapter the focus will be on the Web Application, will be shown the user experience with some examples and a better explanation on how the Web Application is made will be provided.

Chapter 4: Details of the tool

In this chapter, the tool will be analyzed in a deeper way. Will be showed design choices, problems faced and their solutions. In addition, graph indexes will be presented.

Chapter 5: Results

In this chapter, new values will be calculated in order to analyze the graph and images better. This values will be motivated and analyzed. The graph structure will be discussed.

Chapter 6: Conclusions

Summary of the results, future additions that will make the tool better. What DockerSurfer can't do, its limits.

2. Overview of Docker Surfer

2.1. Structure of the tool

This tool gathers data from the Docker Hub (See Section 1.1), manipulates it to provide a fast and connection-oriented graph database, analyzes it to understand properties of the images and visualizes it through an easy to browse interface.

As it’s possible to see in the figure 2, it's composed by a web service and a program that runs on the server side, in order to download, analyze and manage the information. It's an application running on a web server. It stores the data of all the DockerHub projects in a Neo4j database (https://neo4j.com/), which is a graph database. The Java program running on the server scans the DockerHub to maintain the local database updated frequently. The web service allow users to have a graphical representation of the database structure. The tool has to download a big amount of data, insert it into the graph, provide pages to view the graph structure and perform some statistical analysis.

The figure 2 is a graphical representation of the tool:

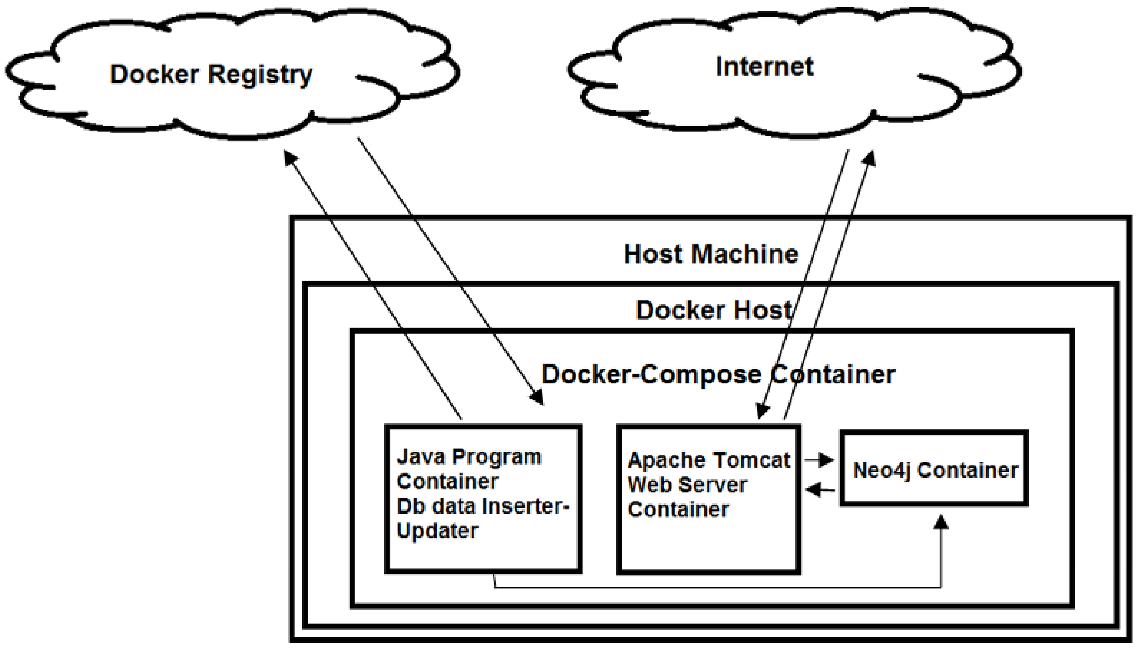


Figure 2 The tool structure

The main parts of the tool:

1. **Graph database:**

A graph database (Neo4jDocumentation, 2017) is a database that embraces relationships as a core aspect of its data model and it's able to store, process, and query connections efficiently. While other databases compute relationships expensively at query time, a graph database stores connections as first class citizens, readily available for any navigational operation. Accessing those already persistent connections is an efficient, constant-time operation and allows to quickly traverse millions of connections per second per core.

Independent of the total size of the dataset, graph databases excel at managing highly connected data and complex queries. Armed only with a pattern and a set of starting points, graph databases explore the larger neighborhood around the initial starting points — collecting and aggregating information from millions of nodes and relationships — leaving the billions outside the search perimeter untouched. This is very useful when trying to view the images connected to an image.

The fundamental units that form a graph are nodes and relationships. In Neo4j, both nodes and relationships can contain properties. Nodes are often used to represent entities. (Neo4jDocumentation, 2017)

Properties are key-value-pairs used for store information. Nodes can have an arbitrary number of properties.

Relationships in Neo4j are links between nodes. They can be traversed in both directions with the same speed. They can store information in properties like nodes.

2. **User experience:**

The user isn't allowed to modify the structure of the database. He can only retrieve his structure. The only way to modify the data is to modify the images in the DockerHub. Users can only access to a web service that will provide them html pages with information about the data and the dependencies. Users can only make GET requests. The user interface is structured as a REST Service, to guarantee an easy to browse and share interface. The tool provide an Index HTML page that allows performing searches by user, image and tag. The REST interface is helpful for sharing links or for perform searches directly in the URL address.

3. **Data sources:**

Having old data in the database can lead to some problems for the users, they can choose images that are no longer maintained, they can not be aware of changes in the images that they are using or they can miss new images. In order to have information about newest images and changes, it's critical to update the graph frequently.

2.2. Usefulness of this tool

Docker Surfer provides to the users the ability to understand the Docker dependency graph in a new way. They can see changes on images that they are using, see the most used and stable images, receive help in the choice of parent images for their projects.

There are some problems with not choosing parent images wisely. Using old tags that are no longer supported will not guarantee updates and will lead to some usability and security problems. New updates are released daily, old software versions will make images affected by virus and bugs. It's important to avoid missing updates, choosing the right tag of the parent image.

The problems of pointing to images that are no longer updated or that change frequently are various. Images that change frequently will give often to the users unexpected changes on the environment, such as programs version number, environment variables value, modifications of network configuration and shared data location. It’s always annoying and also dangerous to don’t know which changes are happening on the images we are using because the changes will reflect on our images too. Images that are no longer updated will not be only weaker against viruses, bugs and vulnerabilities but also if a new security issue is found they will not be updated. This problem can happen in traditional software development too. If the library is not maintained anymore, is not stable or change frequently the user can have similar problems.

Docker Surfer can also help users on being aware of changes on the graph structure. They can see changes on the images that they are pointing to. For example, they can notice if an image that they are using lose popularity for some reason, for example if a better image is released or if a bug or vulnerability is found on the image. In addition, they can be aware of changes in the graph that influence their images. If their parent image change for some reason, like a changes in the image that it is using, they can be notified and they can choose if change parent image or not.

Developers can also see who is using their images so they can have a contact with their users. They can also see when their images lose or gain popularity and take decisions relative to it.

Users can also be aware of changes on their images stability caused by their parent images. They can then search for more stable images and see details about them. In addition, they can see images that use an image that they are interested in and they can see if some of those images are better of the parent image, for example if they add new commands that they wanted to add too. Users can discover images that they probably couldn’t have otherwise seen. Another useful feature of this tool is that users can see how an image is used by other users, this can be crucial if the documentation of the image is inadequate or if the user don’t know very well the software that the image use.

3. Design DockerSurfer

Now that it’s clear the structure of the tool we can take a better look at the interface for the user. The Web application is accessible from the Internet to everyone. Users can surf the structure of the graph (See Section 2.1) through a REST interface. Users can’t modify the database structure and data.

3.1. The Rest service

The Web application use REST. REST is an architectural style for distributed systems, it provides definitions of resources. Resources are uniquely defined by URIs and are manipulated through their representations. REST provides methods to apply to objects. REST can be implemented by the HTTP Protocol. HTTP methods are GET, PUT, POST, DELETE, HEAD, OPTIONS.
Requests are stateless, because objects don't have shared resources. Change resources is made by URLs. REST over HTTP provides a lightweight and layered mechanism for data and service integration.

This tool has two Rest interfaces: `/rest/json/<user>/<image>/<tag>` and `/rest/res/<user>/<image>/<tag>`. The res path provides a human readable representation of the resource made of HTML pages (see Section 3.2). The json path provide a JSON representation of the resource, and it's used by the graphical library (See Section 3.2) to represent it in an animated way.
The /popular path shows the 100 most popular images on the graph, according to the page Rank index (See Section 4.3.3).

3.2. The GUI

The GUI wants to be essential and easy to understand. Users can ask for:

1. **Username**

When asked for a user the web service answer with an HTML page that shows links to all the repositories for the given user. Clicking on the links will open the HTML page of that repository.

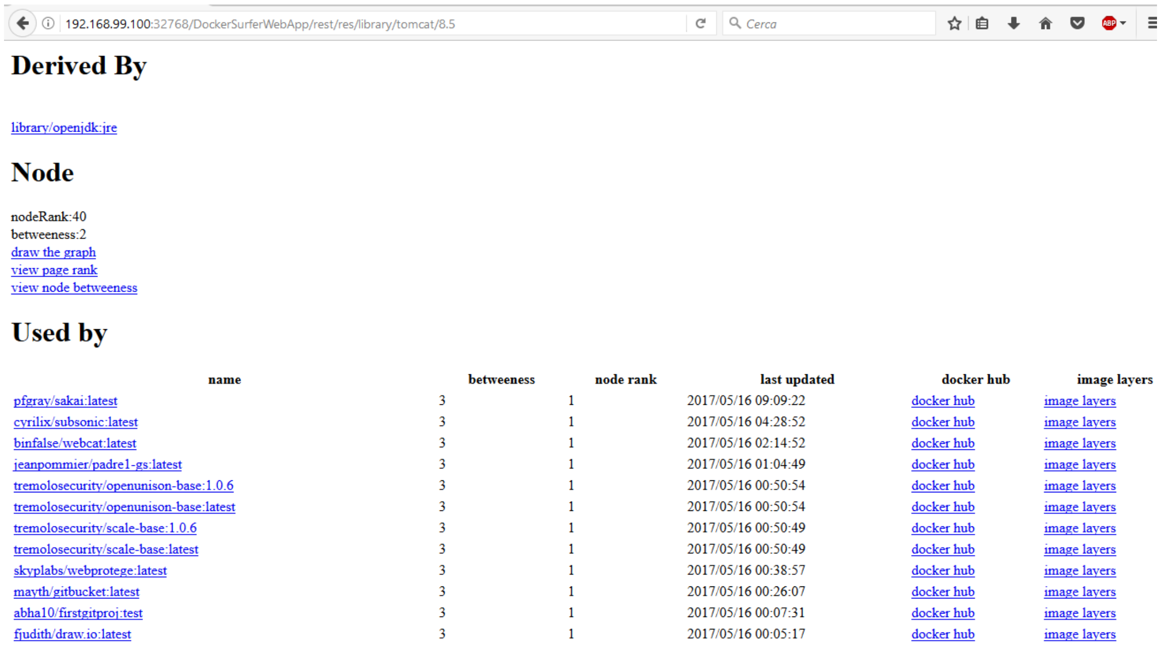
2. **Repository name**

The repository page provides links to every tag of the specific repository. It is useful to see all the tag for an image, for example to choose the best tag or to find new tags.

3. **Tag**

The tag page displays information about the tag and its connections, as showed in the figure 3. It shows the parent of the tag and its information. It also shows the tags that use the image with their information. Information is composed by the last update for the tag, indexes such as page rank and betweenness centrality (see Section 4.3.3), links to Docker Hub and ImageLayers (https://imagelayers.io/) pages about the tag. ImageLayers is an online tool that allows to visualize Docker images and layers that compose them, see how each command in the Dockerfile contributes to the final image, and discover which layers are shared by multiple images.

The figure 3 is an example of a tag page. It is the page about the official image of Tomcat:


192.168.99.100:32768/DockerSurferWebApp/rest/res/library/tomcat/8.5
Cerca

Derived By

library/openjdk:jre

Node

nodeRank:40
betweeness:2
draw the graph
view page rank
view node betweeness

Used by

name	betweeness	node rank	last updated	docker hub	image layers
pfgray/sakai:latest	3	1	2017/05/16 09:09:22	docker hub	image layers
cyrilix/subsonic:latest	3	1	2017/05/16 04:28:52	docker hub	image layers
binfalse/webcat:latest	3	1	2017/05/16 02:14:52	docker hub	image layers
jeanpommier/padre1-gs:latest	3	1	2017/05/16 01:04:49	docker hub	image layers
tremolosecurity/openunison-base:1.0.6	3	1	2017/05/16 00:50:54	docker hub	image layers
tremolosecurity/openunison-base:latest	3	1	2017/05/16 00:50:54	docker hub	image layers
tremolosecurity/scale-base:1.0.6	3	1	2017/05/16 00:50:49	docker hub	image layers
tremolosecurity/scale-base:latest	3	1	2017/05/16 00:50:49	docker hub	image layers
skyplabs/webprotege:latest	3	1	2017/05/16 00:38:57	docker hub	image layers
mayth/gitbucket:latest	3	1	2017/05/16 00:26:07	docker hub	image layers
abha10/firstgitproj:test	3	1	2017/05/16 00:07:31	docker hub	image layers
fjudith/draw.io:latest	3	1	2017/05/16 00:05:17	docker hub	image layers

Figure 3 The tag page for an image

The tag page also have a link to a graphical representation of the dependencies made using Cytoscape.js, a JavaScript external library.

JavaScript is a programming language used to make web pages interactive. It runs on the client side so it’s fast, because it doesn’t have to wait for the server response.

Cytoscape is an open source software platform written in JavaScript for complex network analysis and visualization. It allows to easily display and manipulate rich, interactive graphs. (cytoscape, 2017)

Cytoscape is a good choice because it provides to the application the ability to answer to the users clicks on the graph, making the graph representation interactive. He can browse the graph very fast clicking on the nodes, because the click on the node will open the page relative to the node.

The figure 4 is the graphical view of the dependencies of a tag using Cytoscape:

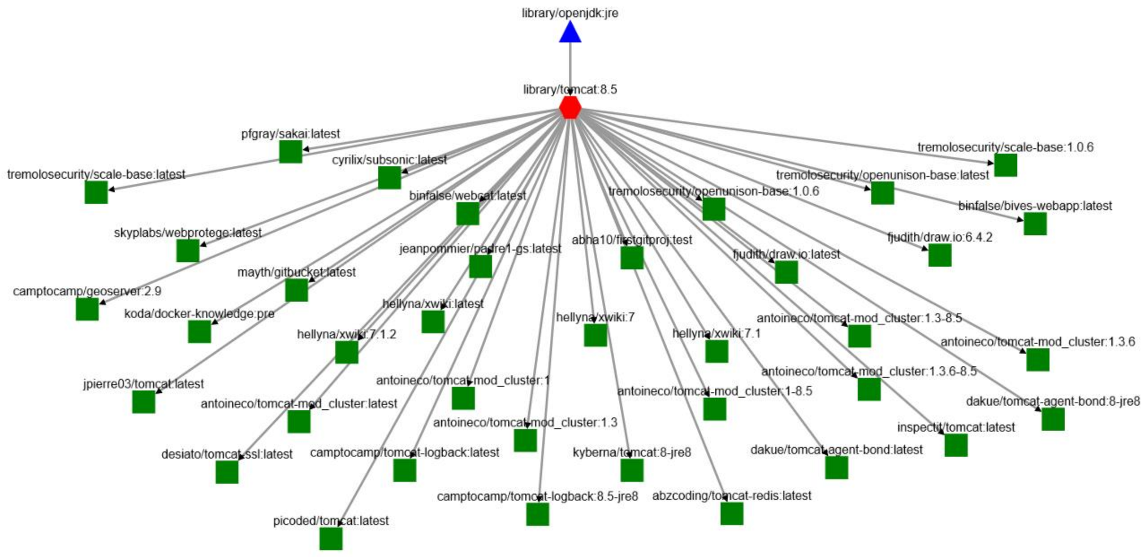

library/openjdk:jre
library/tomcat:8.5
pfgray/sakai:latest
tremolosecurity/scale-base:latest
cyrilix/subsonic:latest
binfalse/webcat:latest
skyplabs/webprotege:latest
camptocamp/geoserver:2.9
mayth/gitbucket:latest
jeanpommier/padre1-gs:latest
koda/docker-knowledge:pre
hellyna/xwiki:latest
hellyna/xwiki:7.1.2
jpierre03/tomcat:latest
antoineco/tomcat-mod_cluster:latest
antoineco/tomcat-mod_cluster:1
desiato/tomcat-ssl:latest
camptocamp/tomcat-logback:latest
antoineco/tomcat-mod_cluster:1.3
picoded/tomcat:latest
camptocamp/tomcat-logback:8.5-jre8
kyberna/tomcat:8-jre8
abzcoding/tomcat-redis:latest
hellyna/xwiki:7
abha10/firstgitproj:test
tremolosecurity/openunison-base:1.0.6
hellyna/xwiki:7.1
antoineco/tomcat-mod_cluster:1-8.5
fjudith/draw.io:latest
antoineco/tomcat-mod_cluster:1.3-8.5
antoineco/tomcat-mod_cluster:1.3.6-8.5
dakue/tomcat-agent-bond:latest
inspectit/tomcat:latest
tremolosecurity/openunison-base:latest
fjudith/draw.io:6.4.2
tremolosecurity/scale-base:1.0.6
binfalse/bives-webapp:latest
antoineco/tomcat-mod_cluster:1.3.6
dakue/tomcat-agent-bond:8-jre8

Figure 4 Cytoscape representation for a tag

3.3. Differences with other applications

Others applications such as DockerHub or ImageLayers can provide a good overview of an image, showing layers in details with commands that made them. They can also show the images for every user, the tags for every image and the Dockerfile. This tool aims to integrate already existent services with some feautures that they don't have. It can show dependencies between images, provide indexes to understand the position and the importance of the image in the graph. It also gives a meaning to the data, by linking images together and providing graph indexes. Data is organized in a graph that is relationships oriented and not image oriented.

4. Details of the tool

In the previous chapters, the general architecture of the tool is explained (Chapter 2), and also its interactions with the users (Section 3.2). In the following, the choices made during the design of the tool, and their motivations, are explained. We will see how the tool retrieves images data, and how it uses it to build the dependency graph among images.

4.1. How data are retrieved

Docker images have:

1. A user name, that is the user that made them.
2. A repository name, that is the name of the project
3. A tag name, that uniquely identify the image
4. A description, that helps in understanding the image purpose and usage
5. A list of layers, that uniquely represent commands made in the Dockerfile (Section 1.1)

We are going to retrieve data about images users, names, tags and their layers. (see Section 1.1) They are took from the Docker Hub, which is the central repository for Docker images. A Java program will download the data.

4.1.1. The search problem

To retrieve information about images, Docker provides a registry, called the Docker Registry. The Docker Registry is a stateless, highly scalable server side application that stores Docker images. The Registry APIs are an interface for programmers that use them to make requests to the Registry. APIs provides a complete and well written documentation. For this work, we need information about all the images on the Docker Hub.

To get information about the images, using the ‘pull’ method provided by the Registry APIs, we first have to know the image names. To discover all the Docker images names the Registry APIs don’t provide an easy way. There is a search function that return all the images that matche the searched word in the name or in the description. Calling the “search” function with the special character '*' should return all the images, but Docker will block the request, probably to avoid returning too large results. The solution is to query the Registry with all the existing English words and with some technologies or programming language names. In this way, very few images will be missed and the Docker Registry will not block the calls. After that, it’s recommended to extract user and image names from the images and search again the Registry using them as queries, in order to find the images that weren't returned by the previous searches.

Quay.io is an alternative to the Docker Hub. It provides a very fast way to download all the repositories names, but doesn't allow to pull the information about images.

4.1.2. Pulling manifests problems

I downloaded 2 millions and an half Docker images. In order to retrieve information about every image, we need 5 millions of GET calls, because half of them are required to get the access token to authenticate. Here the specifications: (https://docs.docker.com/registry/spec/auth/token/).

With a single Thread that uses synchronous calls this process will take approximately 28 days.

A possible solution is to use the multithreading. Multithreading is a technique by which a single set of code can be used by several processors at different stages of execution. A fixed thread pool is a code executor that guarantee that a specified

number of threads will always run. It also manage threads' lifecycle. Fixed thread pool scan speed up the process of the information downloading. They receive tasks (called jobs) in input and assign them a free thread.
A thread pool it's focused on downloading the access tokens needed to access the information and to open the connections. Another thread pool download the information and write the data in the database. The writing process use a synchronized writing, because only one thread can access the database at a time. Synchronized writings guarantee integrity to the database.

Why not asynchronous calls

Asynchronous calls are a good way to speed up the downloading process. In an asynchronous call, the receiver doesn't block its execution until the answer is submitted by the server, but it continue to run. When the call returns from the server the client will call a callback function. In this way the application can make several requests without have to wait for the server to answer. Even if using asynchronous calls is a good way to speed up the downloading process, it requires time to be implemented and isn't trivial to manage errors, such as time out exceptions, that are quite frequents when querying the Docker Hub. I tried the vertx library (http://vertx.io/) for java, but I found the previous cited problems, so I preferred multithreading.

4.2. Modeling dependencies between images

To create dependencies between images we have to understand the dependency mechanism in Docker. An image can use only one other image. An image that use another image will inherit its layers and will add new layers on top of them. Inheriting layers of another image means that it inherit the Dockerfile (See Section 1.1) and all the commands called in it. The image will inherit the code, a runtime, operating system’s base libraries, environment variables, and configuration files. In the figure 5 the images have three layers in common.

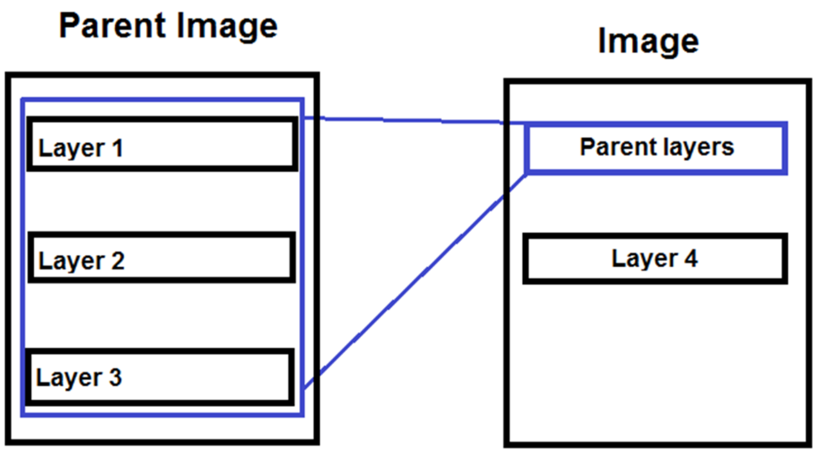


Figure 5 Layers with dependencies

A first way to create dependencies is to browse all the images in order to find a compatible image. A compatible image is an image that has at least one layer in common with the image that needs to be inserted. The layer in common should be the first in chronological order, so the first that was created. This will guarantee that the images are based on the same layer, that comes from the definition of dependency. Base layers are the first n layers in chronological order, where n is arbitrary. After a compatible image is found, its dependency tree will be scanned in a recursive way in order to find the correct position for the image. We can be sure that the image will find his place in the tree, because his parent must have the same base layers; all the images in a tree share the same base layers. Computationally, this approach based on trees navigation doesn't require much space because images are inserted and then deleted from the heap memory. Images are millions and they require Gigabytes of memory so this is the reason why at a first time this was the approach used. But it has O(n^2) execution time, where n is the number of nodes, because for every image it scan all the graph until the correct base image is found. It means that the time required grow following a quadratic

function as the number of nodes grow. It’s needed a new approach where the execution time has to be reduced to linear or logarithmic. This will of course lead to more space usage. The operation that require a lot of time is to find the image with the right layers, because we can search fast for names but not for layers. A list of couples layers-image will not solve the problem, because it will require to browse the list every time we search for layers.
It will be very useful in order to speed up the layers look up to have a data structure that will directly return the image with an input of layers.

A possible solution is to use a map where keys are a representation of the layers and the data stored is the node on the graph. In this way, if the map is implemented as a hash map, accessing a specific image knowing its layers will require constant time.
A Map is a type of fast key lookup data structure that offers a flexible means of indexing into its individual elements. Indices into the elements of a Map are called keys. These keys, along with the data values associated with them, are stored within the Map. Each entry of a Map contains exactly one unique key and its corresponding value.
Given an image, it's easy to find its parent, the operation to perform is to remove layers from list of layers identifying the the image list of layers identifying the until an image is found. That image will be its parent, according to the definition. Removing layers is needed because an image can add several operations to its parent's build, so in order to have the same layers it's sometimes required to remove more than one layer. This will not be a physical remove but only a logical remove.
Unfortunately, setting up a map of millions of elements where keys are long arrays it’s very expensive in terms of memory. That’s why it’s impossible to store such a big information in a classic map. The application will run out of heap memory. (Kotek, 2012). It's possible to increase the heap memory and the application will

work for the images on the Docker Hub in a performant machine, but it will not work for bigger data sources or on different machines, so a solution it's needed.

By using MapDb it’s possible to create the map off the heap needed to create relationships in the graph. Time performance is near at the time performance of a standard Java Map.

MapDB is an open source project. It provides concurrent Maps, Sets and Queues backed by disk storage or off-heap-memory. It is a fast and easy to use embedded Java database engine. It's very easy to understand and to use, not like others projects with the same purpose.
(MapDb, 2017)

4.3. How data are stored

In the database there is only one type of relationship, because we are only storing the dependency relations. Nodes are only of one type: images. An image, by definition, can only have a parent (See Section 1.1). Many images can use an image.

4.3.1 Relational Databases and Graph Databases

Graph databases were introduced in Section 2.1.
A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.

While relational databases have been providing storage support for decades, graph databases are a new technology. So relational databases are more stable and mature.

Relational databases have extensive multi user support. Graph databases don’t have any built in mechanisms for managing security restrictions and multiple users. It presumes a trusted environment. Even if relational databases are more stable and secure, their schema is fixed, which makes harder to manage data that change over time. Also, in a graph database, there is no need to restructure the entire schema every time a new relationship is added; only a few edges and nodes are added to the graph. When the data set becomes larger, relational databases require a lot of join operations, which is an expensive operation. On the other hand, a graph database only search in the connected nodes, that makes the operation faster.

(Shalini Batra, 2012)

In this project the database is accessible only to the web Service and to the Java application that modify it. The Web service has only reading access to the database and the Java application is not accessible from the Web, so there aren't big security problems.

4.3.2 Properties that are relevant and why

Properties are key-value-pairs used for store information. Nodes can have an arbitrary number of properties. In this case, the properties that we have to save are: the repository’s user, name and tag, list of layers, page rank value, betweenness centrality value and the last time that we updated the image. Page rank value and betweenness centrality value will be explained in the next Chapter. User name, repository name and tag are useful for uniquely find the image and to perform searches based on these parameters. The layer list is required to find the relationships between images. The page rank value and betweenness centrality value are used for analyze the graph structure and to have information about the image. The last time that we updated the image it's useful for knowing how old is the image and if it's time to check if there are updates, because images can change quite often.

4.3.3 Page rank and betweenness centrality

To increase the value of the database we can use some graph indicators. This tool use page rank (Franceschet, PageRank Centrality, 2014) and betweenness centrality (Franceschet, Betweenness Centrality, 2014), because they fit well with the graph structure (See Section 5.3) and with the dependency mechanism (See Section 4.2.1).
PageRank is an algorithm used by Google Search to rank websites in their search engine results. PageRank is a way of measuring the importance of website pages. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites
Moving this concept to our situation, we treat dependencies as links and images as pages. Page Rank is used to determine the most used images, giving a higher score to images used by many other images.

In graph theory, betweenness centrality is a measure of centrality in a graph based on shortest paths. For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that the number of edges that the path passes through is minimized. The betweenness centrality for each vertex is

the number of these shortest paths that pass through the vertex. Betweenness centrality is an indicator of how much a node is central in a graph.
Moving this concept to our situation, betweenness centrality becomes an indicator of how many images influence an image.

4.3.4 Indexes

Searching for a specific user, image or tag (See Section 1.1) can be slow, because the database has a big number of nodes (See Section 2.1). Neo4j (See Section 2.1) help us with this problem, providing database indexes.

A database index is a redundant copy of information in the database with the purpose of making retrieving data more efficient. This comes at the cost of additional storage space and slower writes, so deciding what to index and what not to index is an important and often non-trivial task. (Neo4jDocumentation, 2017)

This database should provide fast look-ups of image names, users and tags, so the properties that are indexed are user, image and tag. There is no need to index the page rank value or the betweenness centrality value, because users will not perform queries based on these parameters.

4.4. Finding updates

It’s crucial to have the database kept up to date, showing old data to the users can be useless and even dangerous, because they can have a wrong idea of what they are using. Fortunately, Docker images don’t change very often, so we can have data old of some days. The program that runs in background to find updates is designed to run forever. It doesn’t have to be very fast, so it only has three Threads even if it has to work with a lot of data. Every Thread has a task, they are:

1. **Find new images**

It's the easiest task, the only thing to do is to query the Docker Index (See Section 1.1.2) with user names and the response for every user will be his images. If an image isn't present in the database the image will be inserted.

2. **Find new users**

In order to find new users a list of all the users is used and the operation to do is to query all the Index again with the vocabulary (See Section 4.1.1) and check if there is an image by a new user.

3. **Find updates in the images layers**

Now that the application has all the images, it will pull every image again to check for changes in the layers (See Section 1.1). If it find changes, the image will be removed from the graph and inserted in the new correct position in the graph. To do so, we can't use the map with layers as keys, because we are not inserting all the images and we don't have all the images in the application memory. We can read all the nodes and insert them in a map but it will be expensive in terms of time and memory. In addition, it's not a good choice, primary because this is an application that will run forever in order to have an updated database, so it's preferable to keep the memory free for the other programs on the server. While in the program that create the relationships the map is created, used and later deleted, here the map will be kept forever.

So the application compare the roots node (the base images) with the image to insert. If they have at least a layer in common the image will be placed somewhere in that image tree. If they don't have layers in common, the image will be compared with another root node. When a compatible image is found, the image will be compared with all the images that use the compatible image, in a recursive way. When parent images can't be found anymore the last compatible image will be the parent. The application will search in the repositories that use the parent image for repositories that use the new image and will update the dependencies.

5. Results

Now that the graph (See Chapter 2) is designed and filled up with the information about images (See Section 4.2), we can perform data analysis in order to better understand dependencies and relationships between images.

5.1. Page rank and betweenness centrality motivations

The stability of an image (See Section 1.1) is the probability of that image to don't have unexpected changes. Changes are new values for environment variables, new program installed, different programs versions and everything can be defined in a Dockerfile (See Section 1.1). Unexpected changes can be dangerous for the image's security if the user is not aware of them. In order to understand the problem of image stability, we need to introduce the concept of dependency chain.
A dependency chain happens when an image use another image, that is used by another image, and so on. If a change occurs in an image that is above in the chain dependencies, the change will be applied to all the images below in the dependency chain. This event can be dangerous for the images stability, so it's important when choosing a parent image, to choose images with low betweenness centrality, this will guarantee that the image will have a short dependency chain. Betweenness centrality is recursively calculated adding one to the value of the parent image. Base images have betweenness centrality value equal to zero. In this way, the betweenness centrality value will be the number of images that can change the image's layers if some changes occur.

It's also important to choose one of the most used images as base of our projects. If many users use an image, there is a high probability that the image is stable and safe. The page rank index is an index that helps with this problem. It is calculated adding one to the sum of the page rank values of the images that use the image. Images not used by anyone will have page rank equal to one. In this way the page rank index will tell how much images will be modified if the image change.

5.2. Data analysis

A frequency distribution is an orderly arrangement of data classified according to the magnitude of the observations. When the data are grouped into classes of appropriate size indicating the number of observations in each class we get a frequency distribution. By forming frequency distribution, we can summarize the data effectively. It is a method of presenting the data in a summarized form.

Page rank values and betweenness centrality values (See Section 4.3.3) are frequency distributions.
The variance for page rank values is very high, because values are far from each others. The maximum page rank value is 246.029 and the minimum is one. 90% of the images have page rank value equal to 1. This mean that they don't have images that use them. In order to analyze values I created two tables of value distribution: one for page rank value and another for central betweenness value. For every value, I associated its absolute frequency. Considering that 90% of the images have page rank value equal to one, the histogram isn’t significant. For this reason, I analyzed only the images used by at least one image. So images with page rank value major than 1. The figure 6 is the histogram obtained. On the horizontal axis there are the page rank values and on the vertical axis the number of occurrence of the specified value or interval of values. The figure 7 is the histogram for the betweenness centrality value distribution. The figure 8 is the pie chart relative to the influence of the 100 most influent images.

From the figure 6 we can see that the majority of the images have low page rank values. This mean that there are few popular images. We can see from the figure 8 that the top 100 image with more influence in the graph can affect the half of the images. From the figure 7 we can see that there is a good balance in the betweenness centrality values. They are in the 83% of the images between zero and three. This mean that the graph has a good balance between stable images and images that are more specialized. A specialized image is an image that has a general purpose image as parent and that add some specific commands to its Dockerfile (See Section 1.1.2). A specialized image has often a high value of the betweenness centrality. It is often not official and it’s maintained by an user. It’s always more risky to use not official images (See Section 1.1.2) but it can be useful because official images are often general purpose.

Betweenness centrality values of three or more can bring to stability problems for the users, but this mean that Docker users sometimes need to use more specialized images even if they are less stable and less trusted.

The 90% of the images are leaf nodes in the graph, so they don't have image that use them. This mean that there are few images used by the majority of the images. This a reasonable fact, because it leads to the conclusion that there are few images considered trusted and used by the majority of the Docker users.

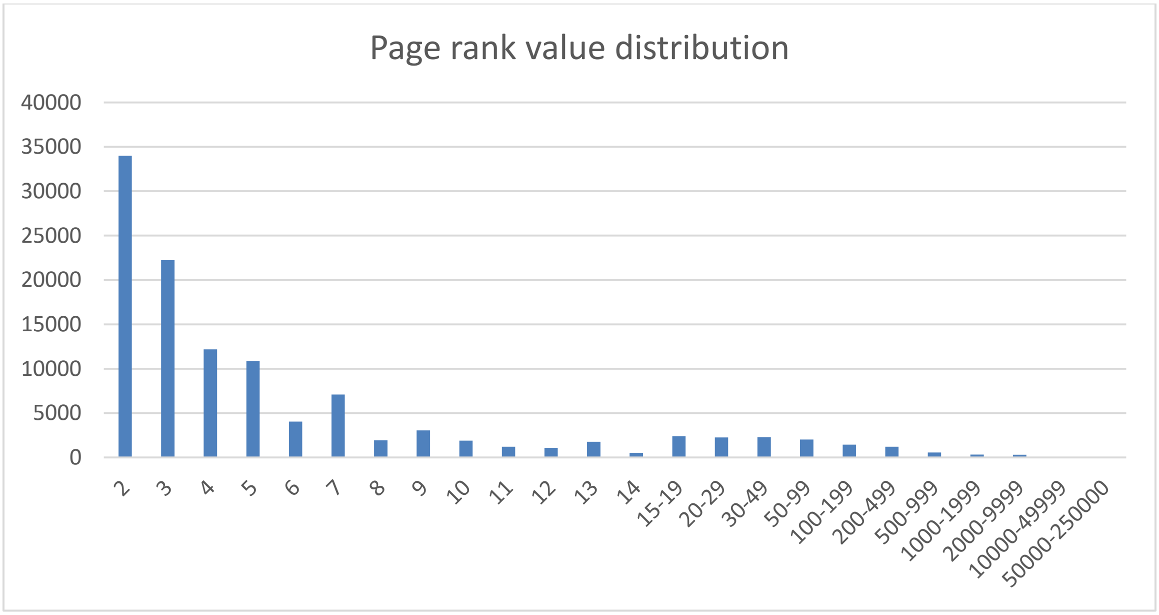


Figure 6 Page rank value distribution

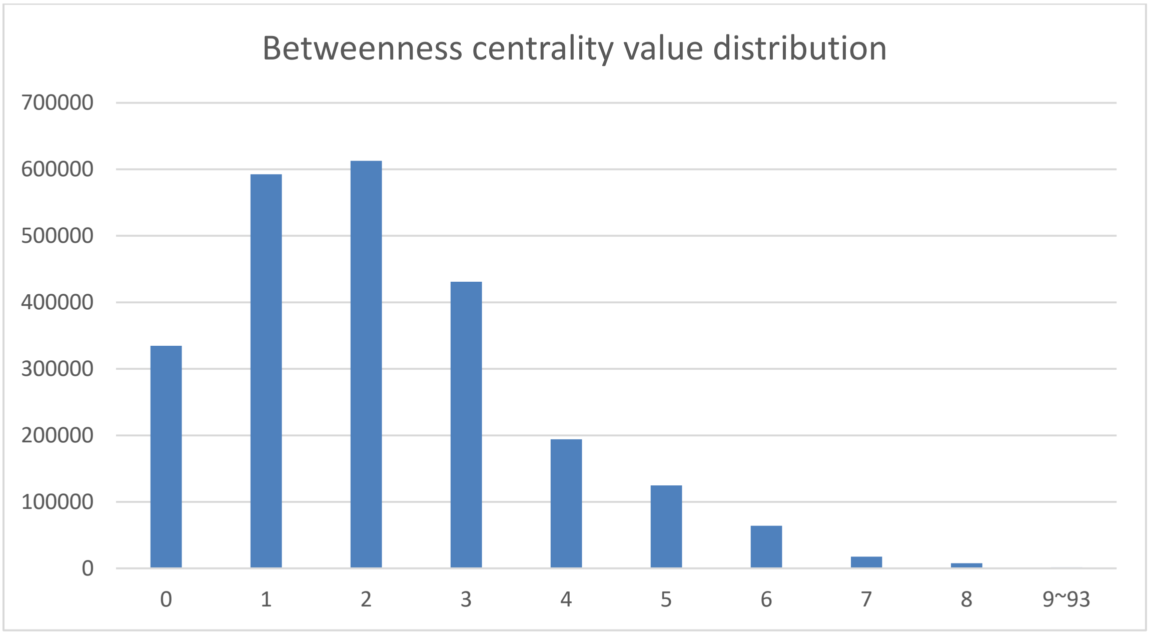


Figure 7 Betweenness centrality value distribution

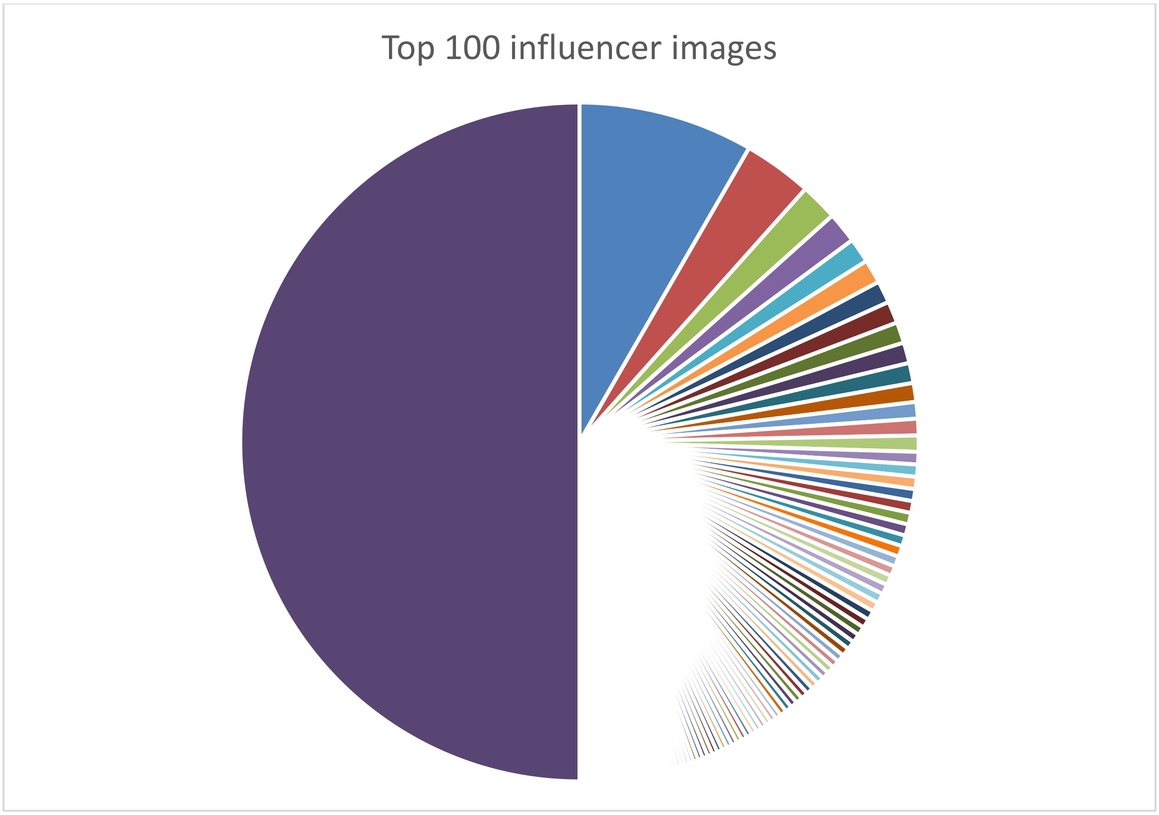


Figure 8 Top 100 influencer images

5.3. Structure of the graph

Analyze the structure of the graph can give some useful information. Images, by definition, can only have a parent (See Section 1.1.2), but they can have an infinite number of images pointing to them. Also, images can't point to an image that point to them by definition. This facts eliminate cycles from the graph. The graph (See Section 2.1) has a forest structure, because it is made of acyclic and connected graphs. Base images are roots nodes for the trees.

Height for a tree is the maximum distance of any node from the root. If a tree has only one node (the root), the height is zero. In this project trees height is a low value, usually between one and three, but every tree have a lot of nodes in it. The graph is not a single tree, but a number of different trees, because there are a lot of different base images that can be used. Users should keep an eye on the structure of the tree that they are using. If a tree has only nodes with a low height it means that different users are probably doing the same things with their images, without using the dependency mechanism in the best way. If a tree has nodes with high height values it means that the image is too general for them and they should use a different base image.

6. Conclusions

There aren’t similar researches in this field. This is the first tool with the purpose of analyze Docker dependencies. In this work I made a first step in this direction, saving the data of all the images in a database that is public, defining how to find dependencies between images and defining some terminology, like dependency chain (See Section 5.1) and specialized image (See Section 1.1). This work can be the base for future works, because it is completely accessible and can be integrated.

Results of this work:

- **Provide an easy to browse interface**

  With this work I provided a tool for browse Docker dependencies and to monitor images. Users can navigate the database structure and be aware of gaining and losing of popularity on the images that they are interested into. They can find the stablest images and also find changes in the image that they are using. They can also understand how to use an image simply watching how other users use that image.
- **Share data and the database structure**

  A very useful thing about this work is that it gathers a big amount of data that is an expensive task in terms of time and effort. This database is public and can be download by everyone. People can also download the unstructured data without the database, for their own use. Data can be integrated and it's easy to understand its structure, because there is only one type of relation between nodes (See Section 4.2) and nodes are all of the same type.
- **Analyze indexes**

  Analyzing the indexes and the graph structure (See Chapter 5) we can see that relationships between Docker images are structured in a well-designed way. There are not long dependencies chains (See Section 5.1) and users use popular and specialized images both, which is a good practice. Popular images are often more general, but they are more stable and secure, because they are maintained by Docker or by trusted users.

6.1. Future developments

Json representation of the dependencies for automated queries

Services on the internet are often implementing only a human readable response for their GET requests. To increase automation and data sharing, it will be helpful to implement a rest path that will return a JSON representation of the requested resource, in order to be understandable to machines. This can bring to automated queries to retrieve the graph structure.

Download private registries

The tool will provide to the user the ability to analyze his own private registry. Private registry act like the Docker Registry (See Section 1.1) but they are not accessible from other users. Private registry are useful because sometimes users can’t share their repositories because they contain proprietary code or confidential information. This addition will help the user in manage his private images and see the relationships between them and the other images on the Docker Hub.

Notifications

When an image that you are using lose page rank value (See Section 4.3.3) fast the tool will send a notification with the name of the image that users are migrating to. The tool will also send a notification when an image that you are directly or indirectly using change its layers. An indirect dependency happens when an image can influence an image that don’t use it directly. This can be useful for knowing unexpected changes on the images.

6.2. Limits of the program

The tool can't provide a graphical view of the dependencies for images with a lot of connections, because the external library that it’s used takes long time to process the request. The limit is set to 50 dependencies. The tool can't show the Dockerfile (See Section 1.1.2) of the images and can't show the operations made to build them. But there is a link to image layers that will show the history of an image and there is a link to Docker Hub that will show the Dockerfile. The idea is to doesn't write features that already exists, but to use them to make the project better.

7. Bibliography

cytoscape. (2017). Taken from cytoscape: www.cytoscape.org

DockerDocs. (2017). Taken from Docker: https://docs.docker.com/

Franceschet, M. (2014). *Betweenness Centrality*. sci.unich.it: https://www.sci.unich.it/~francesc/teaching/network/betweeness.html

Franceschet, M. (2014). *PageRank Centrality*. www.sci.unich.it: https://www.sci.unich.it/~francesc/teaching/network/pagerank.html

Kotek, J. (2012). *3 billion items in Java Map with 16 GB RAM.* http://kotek.net: http://kotek.net/blog/3G_map

Kyoung-Taek Seo, H.-S. H.-Y.-J. (2014). Performance Comparison Analysis of Linux Container. *Networking and Communication 2014*, (p. 105-111). onlinepresent: http://onlinepresent.org/proceedings/vol66_2014/25.pdf

MapDb. (2017). Tratto da MapDb: www.mapdb.org

Nachmani, O. (2015). *5 Key Benefits of Docker: CI, Version Control, Portability, Isolation and Security .* Dzone: https://dzone.com/articles/5-key-benefits-docker-ci

Neo4jDocumentation. (2017) Neo4j: docs.neo4j.org

Shalini Batra, C. T. (2012). Comparative Analysis of Relational and Graph Databases. *International Journal of Soft Computing and Engineering*, (p. 509-512).

VirtualizationRedHat. (2017). RedHat: https://www.redhat.com/it/topics/virtualization