Maximilian Eibl, Martin Gaedke. (Hrsg.): INFORMATIK 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 1865

A mapping approach for configuration management tools to close the gap between two worlds and to regain trust

Or how to convert from docker to legacy tools (and vice versa)

Roy Meissner[1] and Marcus Kastner[2]

Abstract: In this paper we present the tool "DockerConverter", an approach and a software to map a Docker configuration to various matured systems and also to reverse engineer any available Docker image in order to increase the confidence (or trust) into it. We show why a mapping approach is more promising than constructing a Domain Specific Language and why we chose a Docker image instead of the Dockerfile as the source model. Our overall goal is to enable Semantic Web research projects and especially Linked Data enterprise services to be better integrated into enterprise applications and companies.

Keywords: Configuration Management; Orchestration; Trust; Docker; Ansible; Chef; Puppet; Reverse Engineering; DSL

1 Introduction

As [Ar15] showed, containerization is a perfect fit to support the research areas Semantic Web and Linked Data (and possibly other ones) in order to decouple some of their environment requirements and to integrate complex systems. It also helps other projects or even companies that want to reuse results of these projects for their own purposes. Docker[3] is a containerization technology that exists for about 3 years now and is lately often used as a standard technology among research communities, recent projects and startups. Nevertheless and according to [Ri16, page 11 et seqq.], well established enterprises tend to use matured software, approaches and systems and thus have not widely utilized Docker yet, instead using similar, but more matured systems like Chef, Puppet or Ansible. Additionally, these companies are not known for their fast pace in exchanging widely used technologies. This chops a large gap between research projects dealing with state of the art approaches and on the other side matured companies. It's even larger in case these projects want to (or are forced to) integrate their solutions into established enterprises or public administration. In order to fill this gap, we have developed and now introduce "DockerConverter"[4], an

[1] Institute for Applied Informatics (InfAI) e.V. Hainstraße 11, 04109 Leipzig, Germany meissner@informatik.uni-leipzig.de
[2] Leipzig University, Augustusplatz 10, 04109 Leipzig, Germany mk97caza@studserv.uni-leipzig.de
[3] See [Ar15] for an introduction into Docker and its use in the Linked Data Cloud
[4] Github Repository of DockerConverter: `https://github.com/guitarmarx/DockerConverter`

doi:10.18420/in2017_186

approach and a software to map a Docker configuration to various matured systems and also to reverse engineer any available Docker image in order to increase the confidence into it. As the processing of semantic data is getting more and more complex, we are in need of integration solutions for complex software systems, that also need to integrate into matured hosting solutions. DockerConverter is an approach to achieve this goal.

We will introduce into related projects and work first in section 2 and dissociate our own work from these. In Section 3 we describe the implemented conversion approach, point out different alternatives, as well as their advantages and disadvantages. We describe in section 4 how we have implemented and validated the approach and then show how our software increases confidence into Docker images in section 5. This paper ends with a conclusion in section 6, as well as pointing out our future vision.

2 Related Work

"Image Layers"[5] is an open-source tool by CenturyLink Labs that reveals a Docker images layers by analysing it. This is similar to a part of our own approach, but will not uncover any files, is a WebApp and only works for the legacy API of DockerHub[6] (and only with DockerHub). Even worse, Image Layers does not provide any APIs in order to add further steps or automatically reuse its results and its documentation is sparse. According to their Github project[7], development has completely stopped in September 2016.

Similar to Image Layers is MicroBadger[8], that is, in contrast, able to work with the current DockerHub API. Apart of this, MicroBadger suffers from the same problems that we described for Image Layers and is not an open-source project.

"Image2Docker" (Windows only), announced in [Ma16][9], is a recent tool by Trevor Sullivan and enables to convert a virtual machine image (currently only WIM, VHD and VHDX formats and Windows Server 2003, -08, -12 and -16) to a Dockerfile, in order to produce an identical Docker image. This is, as compared to us, the opposite approach (kind of the "vice versa") but is very limited and lacks some major features, like support of established configuration management tools (CM-tools).

Sebastian Günther et. al. describe in [GHS10] an approach for streamlining host setups, software deployments and maintenance approaches into one Domain Specific Language (DSL). They have executed a market analysis and constructed a DSL that is similar to currently available tools, like Ansible. In contrast to our approach, they focused on a DSL as the only solution and Docker was not available at this time. Using a DSL has some major drawbacks that we describe in section 3.

[5] Image Layers Website: https://imagelayers.io/
[6] A public hosting service for Docker images, see `https://hub.docker.com/`
[7] Image Layers at Github: `https://github.com/microscaling/imagelayers`
[8] MicroBadger Website: `https://microbadger.com/`
[9] Image2Docker presentation at this years DockerCon: `https://www.youtube.com/watch?v=YVfiK72Il5A&t=559s`

3 Model Transformation or DSL?

There are mainly two possible approaches to convert from Docker to matured CM-tools. One possibility is constructing a DSL or metalanguage, including a generator, parser and interpreter to convert into the different CM-tool languages. According to [Me12] a DSL has to include all semantic elements of the sub languages and their specifications to generate fully operational configurations. Arie Van Deursen et al. showed in [DKV00] that because of the internal design, a DSL allows validation and optimization at the domain level which improves testability and the conversion.

The alternative approach is a model transformation, that can be accomplished by creating a semantic mapping between the CM-tool languages. This approach allows the conversion from one source model to multiple target models. Every CM-tool language would have its own mapping with its language specific elements like vocabulary and semantics.

Even though a DSL can be very powerful, it has, according to [DKV00, page 2 et seqq.], a large potential loss of efficiency in comparison with model transformation, due to the costs of designing, implementing and maintaining it. As stated in [MHS05], the extension of a DSL is hard to accomplish because of the internal design. A DSL is optimized to all input models which form the semantic of the DSL in the end. Adding another target model might even end in a complete revision process of the semantic and the internal compiler. Apart of this, introducing an artificial language, that acts as a superset of most of the existing languages is not advisable as potential operators would need to learn another language with a steep training curve and complex syntax because of the included sub languages. In contrast, we know about some projects that successfully developed and implemented a DSL and got the market to widely adopt it, for instance the LLVM compiler infrastructure.

In the end, we decided to use a model transformation approach for the conversion because it allows to reuse and convert already existing configurations and will not introduce the mentioned steep learning curve. This is similar to what Trevor Sullivan did in his Image2Docker approach [Ma16], mentioned in section 2 and is expected to increase the market adoption of the tool. Nevertheless, the major risk of a model transformation is the insufficient compatibility of the different models and the loss of information during the conversion process. We explain in section 4 how we handled this risk.

4 Implementation & Validation

We identified two possible alternatives that may act as a proper source model for the transformation process: the Dockerfile and its syntax or Docker images themselves. The transformation between a Dockerfile and a receipt (a collection of commands in a CM-tool) is easier to accomplish because of the similar structure and the same abstraction level. But a Dockerfile contains only a part of the resulting configuration, as a final image is often composed of several Dockerfiles. To reassemble the full configuration, one would have

to trace the composed Dockerfiles all the way up to their root and collect them all. This is sometimes impossible, as information in the chain are missing or it is time consuming, because one needs to find also all possibly added files first[10].

As a Docker image already contains all required information for the transformation, we used Docker images themselves as the source model. Unfortunately an image differs from CM-tool configurations in various and significant ways. For instance docker splits an image into different layers that contain information about executed commands and metadata, as well as files and their history. Apart of layering an image Docker optimizes commands, variables and files for some internal reasons. In order to resolve these we had to execute the following steps:

1. Split the layers of an image apart, to enable further processing
2. Match the various layers with the sequence of executed commands
3. Transform embedded Docker commands (e.g. ADD, CMD, ...) to valid bash commands
4. Extract variables and files within layers by using the information from the previous steps

Thus we transformed an image to a lower abstraction level. The different abstraction layers are depicted in figure 1.

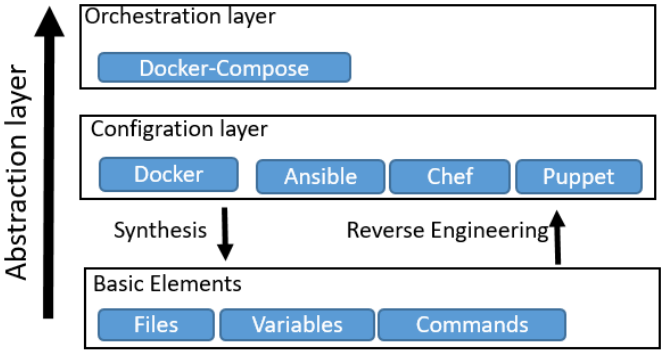

Fig. 1: Abstraction layers of DockerConverter

Speaking more abstract, a synthesis is executed first, which, according to [CH06], denotes a transformation from a higher to a lower abstraction layer. The subsequent transformation back to the configuration layer is called reverse engineering. We have used a Docker image instead of the Dockerfile and the mentioned steps to analyse it, in order to obtain a complete source model. By transforming these information to a lower abstraction layer

[10] This has also been identified by the projects Image Layers and Microbadger, mentioned in section 2

(bash commands), we were able to accomplish a transformation to other CM-tool languages without loss of information, as all CM-tool languages support bash commands.

The transformation process may be controlled via a graphical user interface (GUI), which is depicted in figure 2, or a command line interface (CLI). It is possible to look into different image components, like files and commands, before the transformation process is triggered and also to edit available variables. Additionally, the GUI enables to trigger a fast analyse mode that reconstructs the Dockerfile only. Thus, files (e.g. a HTTPD conf file) which might have been added to the image as of the Docker build process are not reconstructed. The fast mode analyses images much faster, as shown in figure 3 and might be useful if it is only important to reconstruct the Dockerfile itself, e.g. to understand an images structure.

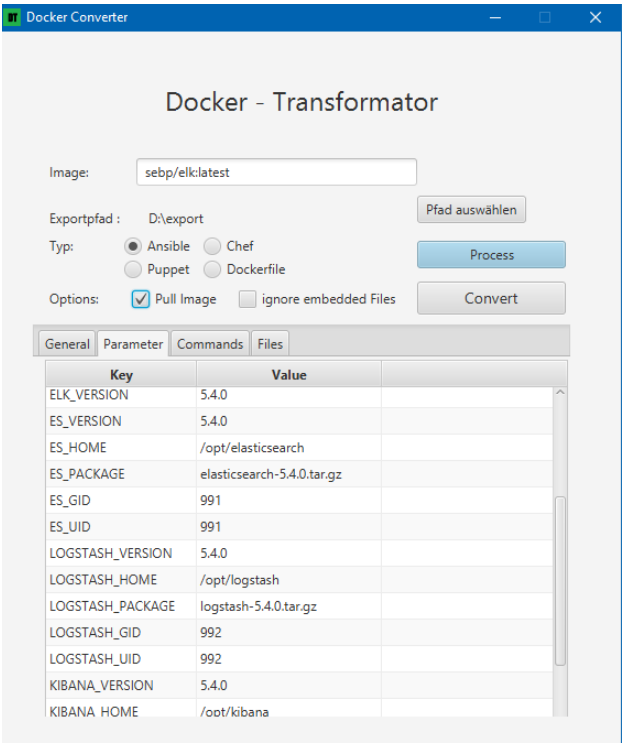

Fig. 2: Graphical User Interface of DockerConverter with an example analysis

We have tested the execution speed on a workstation (Core i5 3. Gen., SSD, 16GB DDR3) and measured both, available analysis modes (fast and full mode). The graph in figure 3 shows the transformation time in relation to the image size for the top ten images from DockerHub[11]. The figure shows that the execution time for a full transformation (full

[11] Top ten images from DockerHub: `https://www.ctl.io/developers/blog/post/docker-hub-top-10/`

analysis mode) depends heavily on the image size, as this mode contains the needed steps for reconstructing additionally used files, as described in the previous paragraph.

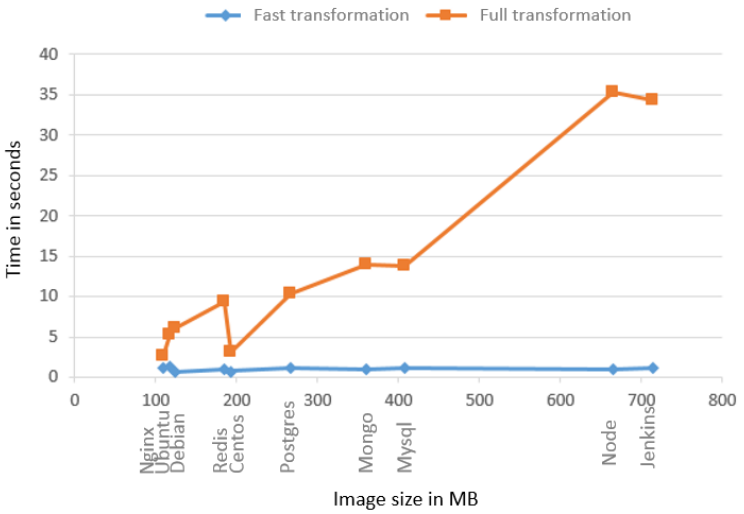

Fig. 3: Transformation time graph

We are currently using manual tests to validate the tool. It has been validated by testing it against ten images of the German company GIP, five images of the EU research project SlideWiki, as well as for 5 images of the German research project LEDS (see section 6 for more information about these). We have additionally tested it against the ten most popular images from DockerHub. In summary, we achieved for all images a transformation to all implemented CM-tools and were able to reproduce virtual machines that are equivalent to the used docker images.

5 Regain Trust

As explained in section 4, anyone with access to an Docker image is able to reverse engineer it with Docker itself. But because of how Docker structures and stores images (see section 4 for an in detail explanation) it ends up to be a time consuming and sometimes complex task to fully reverse engineer a Docker image. Especially as, for example potential malware is often hidden in default files, it is important to reconstruct them in general, as well as in the right way.

With DockerConverter we have implemented all needed steps as a program and thus automated them, so anyone is now able to reverse engineer a Docker image within seconds, essentially reproducing the Dockerfile and all needed additional files, that were used to build the image. It is even possible to convert this information to one of the more matured CM-tools in order to analyse it, in case that the person using DockerConverter is more

familiar with these systems than with Docker itself. So in case other trust or validation methods failed (either because they threw an error or were not used at all), it is now easily possible to validate an image manually or to search for bugs or possible intruders. This is also useful to simply reverse engineer an image, similar to disassembling a compiled program, in order to read and understand it. We have depicted the workflow for in figure 4.

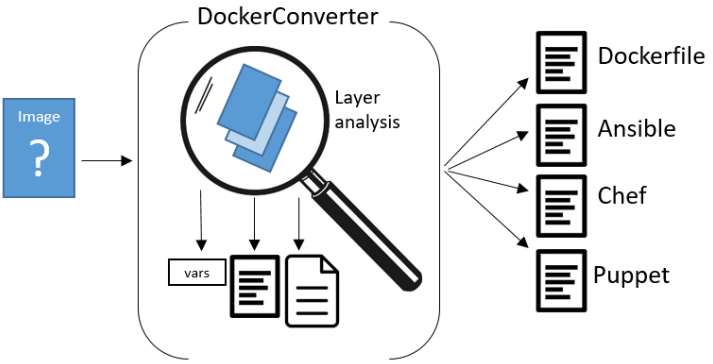

Fig. 4: Image and layer analysis and conversion

6 Conclusion & Perspective

We have implemented an extensible open-source tool for a one way conversion of Docker images (and thus also Dockerfiles) to the three most popular (according to [Ri16]) configuration management tools (CM-tools), that supports a commandline interface, a fast and a full analysis mode as well as a reverse engineering method to reconstruct the complete Dockerfile from an Docker image. DockerConverter thus enables Semantic Web research projects and especially Linked Data enterprise services to be better integrated into enterprise applications, that are more likely to build upon matured CM-tools like Ansible, Chef or Puppet. Therefore we anticipate more possible applications of Semantic Web and Linked Data research results in larger contexts and thus hope to enable better integration of these software systems at all. Additionally, DockerConverter enables to reverse engineer any image in order to gain trust into it, to find out why an already used trust method failed or to simply understand it. This will hopefully also strengthen the confidence in semantic web projects. The tool has been validated via in-use testing by the German company GIP, the EU Horizon 2020 research project SlideWiki and the German Federal Ministry for Research and Educations research project Linked Enterprise Data Services (LEDS) (see section 6 for more information about these), as well as by testing it against the ten most popular Docker images at DockerHub. We have additionally outlined possible other ways of implementing or constructing this tool and why we chose a mapping approach over the DSL approach.

As the tool is open-source and thus extensible by anyone, it is easily possible to add more mappings in order to support more CM-tools. We have noticed that the mapping of the most

basic commands between those CM-tools is bijective, therefore the mapping is revertible, but currently not implemented as we have focused on the described use case. According the former paragraph, we have only tested the tool manually. Thus, it is a future goal to create an automated test suite for continuous integration to cover unit and integration tests, as well as testing the integration between different CM-tool configurations. Thinking even more into the future, it seems to be possible to map whole orchestration configurations from Docker-Compose and Docker Swarm to the matured systems and vice versa.

Acknowledgements

This work was partly supported by the German company "Gesellschaft für innovative Personalwirtschaftssysteme mbH" (GIP), located in Leipzig, as well as the following grants from European Union's Horizon 2020 research and innovation programme for the SlideWiki Project under grant agreement No 688095 and the German Federal Ministry of Education and Research (BMBF) for the LEDS Project under grant agreement No 03WKCG11C.

References

[Ar15] Arndt, Natanael; Ackermann, Markus; Brümmer, Martin; Riechert, Thomas: Knowledge Base Shipping to the Linked Open Data Cloud. In: 11th International Conference on Semantic Systems Proceedings. International Conference on Semantic Systems Proceedings, Vienna, Austria, pp. 73–80, September 2015.

[CH06] Czarnecki, Krzysztof; Helsen, Simon: Feature-based survey of model transformation approaches. IBM Systems Journal, 45(3):621–646, 2006.

[DKV00] Deursen, Arie Van; Klint, Paul; Visser, Joost: Domain-Specific Languages: An Annotated Bibliography. Sigplan Notices, 35(6):26–36, 2000.

[GHS10] Günther, Sebastian; Haupt, Maximilian; Splieth, Matthias: Utilizing Internal Domain-Specific Languages for Deployment and Maintenance of IT Infrastructures. March 2010. Technical Report, URL: http://www2.cs.uni-magdeburg.de/inf_media/downloads/forschung/technical_reports_und_preprints/2010/TechReport04-EGOTEC-2f2p4di8gee9ocvgnihl5nauh31gad9b.pdf.

[Ma16] Marks, Mano: , Image2Docker: A new tool for prototyping Windows VM conversions, September 2016. https://blog.docker.com/2016/09/image2docker-prototyping-windows-vm-conversions/, last access: 05 May 2017.

[Me12] Mernik, Marjan, ed. Formal and Practical Aspects of Domain-Specific Languages: Recent Developments. IGI Global, September 2012. DOI: 10.4018/978-1-4666-2092-6.

[MHS05] Mernik, Marjan; Heering, Jan; Sloane, Anthony M.: When and how to develop domain-specific languages. ACM Computing Surveys, 37(4):316–344, dec 2005.

[Ri16] RightScale, Inc.: , State of the Cloud Report: DevOps Trends, January 2016.