Representing Dockerfiles in RDF

Riccardo Tommasini[1], Ben De Meester[2], Pieter Heyvaert[2], Ruben Verborgh[2], Erik Mannens[2], Emanuele Della Valle[1]

[1] Politecnico di Milano, DEIB, Milan, Italy
{riccardo.tommasini,emanuele.dellavalle}@polimi.it
[2] Ghent University – imec – IDLab,
Department of Electronics and Information Systems, Ghent, Belgium
{ben.demeester,pheyvaer.heyvaert,ruben.verborgh,erik.mannens}@ugent.be

Abstract. Containers – lightweight, stand-alone software executables – are everywhere. Industries exploit container managers to orchestrate complex cloud infrastructures and researchers in academia use them to foster reproducibility of computational experiments. Among existing solutions, Docker is the de facto standard in the container industry. In this paper, we advocate the value of applying the Linked Data paradigm to the container ecosystem's building scripts, as it will allow adding additional knowledge, ease decentralized references, and foster interoperability. In particular we defined a vocabulary *Dockeronto* that allows to semantically annotate Dockerfiles.

Keywords: container, Docker, Linked Data, vocabulary

1 Introduction

Linux Containers[3] (e.g., LXC) are an operating system-level virtualization technique that revolutionized the way software is packaged and distributed. Companies exploit LXC to manage complex infrastructures, either internally, e.g., by means of OpenStack[4], or deployed on one of the available cloud solutions, e.g., Microsoft Azure[5] and Amazon Web Services[6]. Among the available container solutions, *Docker*[7] rapidly became the de facto standard and, more recently, it started influencing academic research, because it helps solving a number of fundamental concerns that address reproducibility and repeatability of experiments, as put forward by Boettiger [2]. *Docker* guarantees (i) modular reuse of software packages, (ii) a portable environment, (iii) public sharing by means of web repositories (*Docker Registry*), and (iv) versioning.

Docker provides a set of concepts for the creation and initialization of containers: (i) a *Docker Image* is a software package containing a single application,

[3] https://linuxcontainers.org/
[4] https://www.openstack.org/
[5] https://azure.microsoft.com/
[6] https://aws.amazon.com/
[7] https://www.docker.com/

```
FROM ubuntu:latest
RUN apt-get update  apt-get install -y python python-pip wget
RUN pip install Flask
ADD hello.py /home/hello.py
```

Listing 1.1: A Dockerfile that installs a Python application on Ubuntu.

(ii) a *Dockerfile* is the script that contains the instructions used to build the image, and (iii) a *Docker Container* is a runable instance of an image.

The build instructions are at the core of what functionality a container offers. Although, this works in the *Docker* ecosystem, outside this ecosystem these instructions are not sharable and extendable in a machine-understandable manner. For example, (i) providing additional information about specific instructions is only limited through the use of comments in the script, (ii) refering to specific instructions outside of the complete context of its specific script is not easily achievable, and (iii) machine-understandibility is limited as the build instructions are not self-descriptive. Therefore, we advocate the use of Linked Data principles[8] to make these build instructions available. However, to apply these principles a vocabulary is required to semantically annotate these instructions.

Therefore, in previous efforts, Label-Schema.org[9] and Smart Containers [4] were introduced. However, they do not consider the build instructions. Label-Schema.org proposes a set of build-time labels for containers in the form of `org.label-schema.[key]=[value]` that can be used to add metadata to the built *Docker Image*. Smart Containers model *Docker* concepts using PROV-O [1], focusing on the environment where computational experiments are executed, but they remain high level.

In this paper, we present *Dockeronto*[10]. It is a vocabulary that builds on the idea of Smart Containers to semantically annotate *Dockerfile*s. Furthermore, it uses the Function Ontology (FnO) [5] to represent a *Dockerfile*'s instructions.

2 *Dockeronto* in a Nutshell

In this section, we introduce *Dockeronto* via an example[11]. The *Dockerfile* of Listing 1.1 installs and runs a Python application on top of the latest available Ubuntu image and executes the following types of instructions:

1. **FROM** specifies the base image from which the current *Dockerfile* inherits all the functionalities,
2. **RUN** executes a command within the image (at build time), and
3. **ADD** copies a file from the host file systems into the image file system.

[8] `https://www.w3.org/DesignIssues/LinkedData.html`
[9] `http://label-schema.org/rc1/`
[10] `https://github.com/riccardotommasini/dockeronto`
[11] The documentation and more elaborate examples are available at `https://github.com/riccardotommasini/dockeronto`.

```
 1  do:from a fno:Function, do:Instruction ;
 2    fno:expects  ( do:imageInputParam  ) ;
 3    fno:returns  ( do:imageOutputParam ) .
 4
 5  do:run  a fno:Function, do:Instruction ;
 6    fno:expects  ( do:imageInputParam  do:runInputCommand ) ;
 7    fno:returns  ( do:imageOutputParam ) .
 8
 9  do:runInputCommand a fno:Parameter;
10    fno:predicate do:runCmd;           fno:type do:Command .
11  do:imageInputParam a fno:Parameter;
12    fno:predicate do:imageInput;       fno:type do:Image   .
13  do:imageOutputParam a fno:Output;
14    fno:predicate do:imageOutput;      fno:type do:Image   .
```

Listing 1.2: FROM and RUN instruction in *Dockeronto*

```
ex:dockerfile1 a do:Dockerfile;
        do:contains ( ex:ins1 ex:ins2 ex:ins3 ex:ins4 );

ex:ins1 fno:executes do:from; do:fromValue <ubuntu:latest>;
        rdfs:label "Install latest Ubuntu";
        rdfs:comment "We always want to have the latest Ubuntu updates.";
        dcterms:creator ex:riccardo.
ex:ins2 fno:executes do:run; do:runCmd "apt-get update && ...".
        rdfs:label "Install necessary dependencies."
ex:ins3 fno:executes do:run; do:runCmd    "pip install Flask".
        dcterms:creator ex:ben.
ex:ins4 fno:executes do:add; do:src "hello.py"; do:dst "/home/hello.py";
        rdfs:comment "This script shows hello world.".
```

Listing 1.3: RDF representation of a Dockerfile with *Dockeronto*.

We represent these instructions using the Function Ontology (FnO) [5]. Listing 1.2 shows the FnO descriptions for FROM and RUN. For example, the RUN **instruction** expects an **image** and a **command** as input parameters (line 6): the image is the intermediate image from the previous instruction, and the command is, e.g., `apt-get update && apt-get install -y python python-pip wget` (Listing 1.1, line 2). Note that the modeling of the individual run commands is not in scope of this work, as this is not specific to the *Dockerfile* syntax. For the time being, they are described as string values. The instruction's output is a new image (line 7), either to be used for the next instruction, or as the resulting image of the *Dockerfile*.

Listing 1.3 shows the Turtle serialization of the example *Dockerfile* using *Dockeronto*. This representation is queryable yet still executable, because we use an `rdf:List` to retain the ordering of the *Dockerfile* instructions since it influences the output *Docker Image*. We consider intermediate images, which are generated during build time, but since they can be inferred we did not describe them explicitly. Last but not least, we added additional information to the instructions, such as labels, comments, and their creators.

Outside the context of this *Dockerfile* it is also possible to refer to specific instructions without the need to know the complete *Dockerfile* or even the fact

```
ex:riccardo dbo:created ex:ins2 , ex:ins4 ;

ex:reviewIns3  a schema:Review ;
               schema:itemReviewed ex:ins3 ;
               schema:contributor ex:riccardo .
```

Listing 1.4: Knowledge about specific instructions outside the *Dockerfile* context.

that the instructions are related to *Docker*. For example, in Listing 1.4 additional knowledge about who created the instructions is provided, together with a review of one specific instruction.

3 Conclusion

The development of *Dockeronto* is an important step to improve the use of *Docker* build instructions outside the context of a *Dockerfile*. This in turn allows to work towards applying the Linked Data principles. The extensibility and shareability is improved, and the build instructions are now self-descriptive. As was shown in the example, (i) additional knowledge can be easily added to the instructions, (ii) (references to) instructions can be shared outside the context of a *Dockerfile*, and (iii) the use of semantic annotations via *Dockeronto* allows for self-descriptive instructions that are understandable even outside of the *Docker* ecosystem.

For the future, we envision an Linked Container ecosystem, where semantic technologies are used to empower development workflows, track provenance and develop semantic microservices [3].

References

1. Belhajjame, K., Cheney, J., Corsar, D., Garijo, D., Soiland-Reyes, S., Zednik, S., Zhao, J.: PROV-O: The PROV Ontology. Recommendation, World Wide Web Consortium (W3C) (Apr 2013), `https://www.w3.org/TR/prov-o/`, accessed June 14th, 2017
2. Boettiger, C.: An introduction to docker for reproducible research. Operating Systems Review 49(1), 71–79 (2015), `http://doi.acm.org/10.1145/2723872.2723882`
3. Fernández-Villamor, J.I., Iglesias, C.A., Garijo, M.: Microservices - lightweight service descriptions for REST architectural style. In: ICAART 2010 - Proceedings of the International Conference on Agents and Artificial Intelligence, Volume 1 - Artificial Intelligence, Valencia, Spain, January 22-24, 2010. pp. 576–579 (2010)
4. Huo, D., Nabrzyski, J., Vardeman, C.: Smart container: an ontology towards conceptualizing docker. In: Proceedings of the ISWC 2015 Posters & Demonstrations Track, Bethlehem, PA, USA, October 11, 2015. (2015)
5. Meester, B.D., Dimou, A., Verborgh, R., Mannens, E.: An ontology to semantically declare and describe functions. In: The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers. pp. 46–49 (2016), `https://doi.org/10.1007/978-3-319-47602-5_10`