Smart Container: an ontology towards conceptualizing Docker

Da Huo[1], Jaroslaw Nabrzyski[1], and Charles F. Vardeman II[1]

University of Notre Dame
{dhuo,naber,cvardema}@nd.edu

Abstract. Because of growing demand to preserve and share reproducible computational experiments in scientific community, there has been interest in using Docker Linux Containers as a preservation mechanism. However, this is insufficient to help researches to comprehend "Dockerized" experiments and connect computational artifacts with concepts in peer-reviewed publications. We present here an ontology and software, Smart Container, that can conceptualize Docker artifacts by and is aligned with other existing vocabularies such as the well known W3C prov vocabulary.

1 Introduction

As modern science becomes computation-driven and data intensive, the need for reproducibility of scientific research requires an efficient approach to preserve and share computational experiments. Virtual Machines and workflow languages are two mechanisms that have been explored as general solution to the preservation problem. But VMs have large software dependency requirements to distribute and workflow techniques do not capture third party resource dependencies [4]. We believe Docker provides a more portable and light-weighted approach that, in fact, the European Organization for Nuclear Research, known as CERN[1] already adopted to preserve high energy physics experiments.

We present here are initial work towards building an ontology called Smart Container(SC) that conceptualizes Docker software objects and is aligned with other existing ontologies, PROV-O[2] and CSO[3]. We are aiming to provide a mechanism that can capture the provenance of Docker containers themselves and potential enable sharing Docker information on the Semantic Web via Linked Data principles. We describe how we can capture the provenance trail of Docker images as artifacts themselves and how we can extend this provenance to include artifacts that are contained *within* a container object.

2 Formalize the SC Domain

Given the complexity of software systems, an initial ontology certainly cannot cover every single aspect related to Docker. Therefore, we focused on conceptualizing essential terms involving running a computational experiment in Docker

[1] https://twiki.cern.ch/twiki/bin/view/Main/DockerCVMFS
[2] http://www.w3.org/ns/prov#
[3] http://cos.ontoware.org/cso#

importing existing ontologies and conceptual terms into an ontology pattern. The purpose of this work is to fill the provenance gap between Docker infrastructure and scientific experiment artifacts and providing a vocabulary prototype that is capable for future extension.

2.1 Background

The Smart Container Ontology was constructed from a systematic alignment between the main concepts present in existing Docker meta-data utilizing and existing vocabulary terms where possible to contextualize those meta-data concepts. In our work, PROV and CSO, two widely-used vocabularies, were introduced to construct the ground of Smart Container Ontology.

PROV-O is a W3C recommendation that describes the interactions of provenance generated in different systems and under different contexts[2]. Three main types of concepts: prov:Entity, which represents objects; prov:Activity, which describes an event happened over time involving entities; and prov:Agent, which is responsible for an activity or an entity, constructed PROV-O. PROV-O has been demonstrated to have reasonable flexibility and has been shown to enable alignment between other ontologies [1]. Therefore, we choose to use PROV-O as the foundational "upper" ontology for the Smart Container Ontology to facilitate connections with other vocabularies.

The Core Software Ontology(CSO)[3] is an ontology formalizing common concepts in software engineering, such as data, software with its different shades of meaning classes and methods. CSO uses DOLCE[4] as a foundational ontology and its extensions: Descriptions&Situations(DnS)[5], the Ontology of Plans(OoP)[6] and the Ontology of Information Objects(OIO)[7]. CSO provided us with a formalization of "software" concepts that we can apply to Smart Container domains. However, because of the complexity of DOLCE, we do not import CSO directly to avoid entailment of relations beyond the scope of this application.

Docker is an application based on Linux Containers(LXC). It isolates an application with its dependencies in a single process which is more light-weighted than full hypervisor virtualization of guest operating system. It can be provisioned by a simple Dockerfile text based workflow. Docker also adopted a layer file system way to achieve versioning and component re-use. A Docker image is a read-only layer which is stateless. A container has states: when it is running, it represents a tree of processes isolated from other processes on the host; when it exits, it represents a read-write layer generated by the process along with its all underneath stateless images. We differentiated these two concepts in our ontology.

---

[4] `http://purl.org/ifgi/dolce#`
[5] `http://www.loa.istc.cnr.it/ontologies/ExtendedDnS.owl`
[6] `http://www.loa.istc.cnr.it/ontologies/Plans.owl`
[7] `http://www.ontologydesignpatterns.org/ont/dul/IOLite.owl`

2.2 Alignment Pattern

From inspection, a Docker image is a digital object with some attributes. It matches the description of a prov:Entity: a physical, digital, conceptual, or other kind of thing with some fixed aspects[2]. The Dockerfile is a text file with lines of Dockerfile commands. Docker fetches these commands and invokes the relevant software to execute them. Each line in the Dockerfile generates an execution inside the container. Because a container has states, we treat static (just created or exited) container as a prov:Entity, which is similar to a docker image. A running container is represented by a prov:Activity, which represents something that occurs over a period of time and acts upon or with entities[2]. For each Docker container, a software can be bash, python, Docker itself or any other agent executes commands. We extracted the software responsible for each command execution as a prov:SoftwareAgent, which is a subclass of prov:Agent. From a macro perspective, a series of operation in a computational experiment is always associated with a human user. We aligned the human user with prov:Person, sub-classing prov:Agent and use the prov:actedOnBehalfOf to connect the two.

In computational experiments, we have to be very careful about some special concepts from the computer science domain. The encoding of the whole computational experiment, such as the Dockerfile, is similar to InformationObject concept from CSO. The Docker itself in the experiment, on the other hand, is similar to a form of CSO:InformationRealization which is a realization of code in the machine. If we break the Dockerfile line by line, we also can treat each line of command as a smaller InformationObject. The running container is analogous to ComputationalActivity in CSO where the software manifests itself by a sequence of tasks contained in a plan. Our approach is *Ontology Pattern* based creating our own specialization but apply rdfs:seeAlso to terms in CSO, the weak sense of identity without making strong ontological commitments based on DOLCE.

In fig 1, sc:Image, representing a Docker Image, is a specialization of prov:Entity. Docker containers were divided into three parts: sc:startContainer, sc:runningContainer and sc:endContainer. sc:startContainer and sc:endContainer subclass prov:Entity representing static conditions. sc:RunningContainer, on the other hand, subclasses prov:Activity as an event over time. An Dockerfile, referenced as sc:Dockerfile, and a line of command, referenced as sc:Command, both are subclasses of prov:Plan. The human user of the Dockerfile is identified as sc:User which subclassing prov:Person. sc:SoftwareAgent is a direct subclass of prov:SoftwareAgent standing for the software executes commands. We use a technique similar to TrustURI's by using the Docker image 64 digit code uniquely can be resolved by a uniform resource name(URN) with specific protocols to create a URI for a static image. Each running container can be exposed by a HTTP address which is dereferenceable so we construct Uniform Resource Locator(URI) in the normal manner. We identify a human agent using URI's constructed from ORCID(Open Researcher and Contributor ID)identifier, a non-proprietary alphanumeric code to uniquely identify scientific and other academic authors, facilitating investigator and potential publication identities to be propagated.

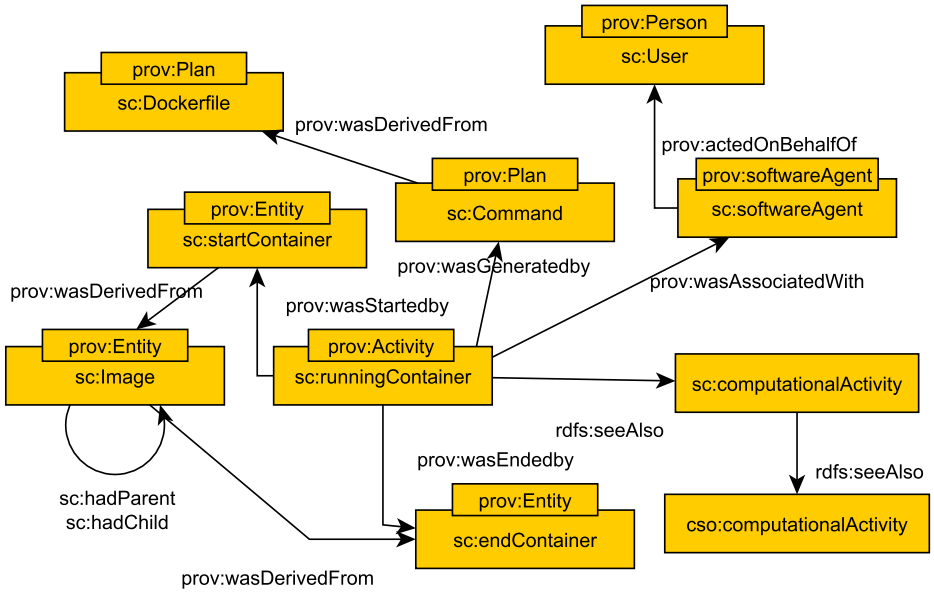

Fig. 1. Boxes represent concepts with inheritance represented by the smaller boxes

3 Conclusion

In this paper, we present an ontology pattern named Smart Container that contextualizes the docker software system acting as an infrastructure for computational experiments. We populated our ontology design pattern by analyzing main concepts in Docker and aligned with PROV-O and CSO to provide possibilities for wider extensions.

Acknowledgements. We acknowledge funding from NSF grant PHY-1247316 "DASPOS: Data and Software Preservation for Open Science."

References

1. Compton, M., Corsar, D., Taylor, K.: Sensor data provenance: Ssno and prov-o together at last. In: To appear 7th International Semantic Sensor Networks Workshop (October 2014) (2014)
2. Lebo, T., Sahoo, S., McGuinness, D., Belhajjame, K., Cheney, J., Corsar, D., Garijo, D., Soiland-Reyes, S., Zednik, S., Zhao, J.: Prov-o: The prov ontology. W3C Recommendation 30 (2013)
3. Oberle, D., Grimm, S., Staab, S.: An ontology for software. In: Handbook on ontologies, pp. 383–402. Springer (2009)
4. Zhao, J., Gomez-Perez, J.M., Belhajjame, K., Klyne, G., Garcia-Cuesta, E., Garrido, A., Hettne, K., Roos, M., De Roure, D., Goble, C.: Why workflows breakunderstanding and combating decay in taverna workflows. In: E-Science (e-Science), 2012 IEEE 8th International Conference on. pp. 1–9. IEEE (2012)