Semantically Organized Containers for Reproducible Research

Andrew Youngdahl, Zhihao Yuan, Dai Hai Ton That, Tanu Malik, Ivo Jimenez, Carlos Maltzahn

School of Computing, DePaul University, Chicago, IL, USA
Computer Science, University of California, Santa Cruz, CA, USA
Email: {*ayoungdahl, zhihao.yuan, dtonthat, tmalik1*}*@depaul.edu*, {*ivo,carlosm*}*@ucsc.edu*

Abstract

Experiments are a key component in systems and high performance computing(HPC)-related research. They help validate new ideas and concepts. Sharing and reproducing experiments, however, is a challenge, especially when computational experiments reside in multiple computing environments, are disorganized into multiple directories, are disconnected from each other, or lack sufficient documentation.

In this paper we show how sharing, porting, and reproducing distributive and iterative experiments can be simplified by using an automatic containerization tool for capturing/repeating an experiment and a convention for organizing repeated runs of an experiment. Using a simulation-analysis workflow, we show how semantically organized containers can help a reviewer find all experiments for a given result and re-execute all experiments with fail-proof guarantee. We discuss outstanding challenges of adopting this method as an artifact evaluation mechanism.

1 Introduction

Experiments in systems research are vital for establishing and validating an idea. A systems experiment can be described as having three components: goals, means, and observations [2]. The goals and observations are typically text-based, but the means of an experiment, which describe the particularities of how the experimental environment and procedures are carried out, are primarily computational. While goals and observations are easily described in a text-based publication, means are typically not well-documented. Increasingly, authors are required to share the means of an experiment as accompanying computational artifacts so that reviewers can validate artifacts for its stated goal and observations.

Sharing the means of an experiment, however, is a challenge–a challenge often more evident when an experiment fails to execute. Consider the following scenario:

Scenario: Alice is an urban scientist who has published a paper about how climate change will affect city landscapes. Bob reads her paper and, in particular, would like to repeat her experiment that predicts landscape change as a measure of climate change. The paper refers to Alice's simulation software available from Github and observation data that were downloaded from U.S. Government's open data service[1]. Re-running the simulation on XSEDE[2], Alice's original computing environment, however, results in unexpected/unreported outliers. Bob tries simpler experiments such as pre-processing the observation data but is not even able to build the code; the build reports a failure about certain C++ symbols not found.

Containerization provides an effective solution to share such experiments. In this scenario, Alice failed to describe the necessary and sufficient dependencies for her data-preprocessing step, and failed to note the state of dependencies in the XSEDE environment.

[1] www.data.gov
[2] https://www.xsede.org/

An update, in-between Alice's and Bob's run of the experiments, resulted in differing values. If both environments had been isolated and preserved in a container, Bob would have been in an easier position to replicate.

But containers do not reduce all the effort that Bob will need to spend to reproduce Alice's results. In particular, Bob had read Alice's paper and was only interested in the prediction experiment (not all of them). For Bob, it is still a challenge to find all files relevant to the prediction experiment in Alice's containers. Containers typically layer their contents or use native file system storage without semantic organization, making them inconvenient for reproducible analysis.

This raises the question if, and how, semantically organized containers can improve the conduct of reproducible research, especially in the artifact evaluation process. Our vision is an automatically organized container in which all experiments related to a paper are packaged such that each individual experiment can be partially or entirely used for reproducible analysis. We envision authors to share such semantically organized containers and reviewers to instantly find and repeat and reproduce specific experiments from this container. Towards this vision, in this paper, we outline a container organization method. The method creates semantically organized user-defined containers of experiment runs, making it easy for a reviewer to find all experiments (and their containers) pertaining to a result and making it easy to reproduce the result. Our method is based on Sciunit [5, 8], an automatic containerization tool used for reproducible analysis, and Popper [3], a convention for managing experiments as a software project. AIM [6] is an abstraction proposed to consistently modularize preprocessing and training steps in machine learning works, while [4] proposes a file structure convention to promote modularity and transparency in scientific computation.

Our initial experiment on a simulation-analysis workflow qualitatively shows that organizing containers makes it far easier to reproduce experiments by simply changing input datasets and parameters. We describe challenges in adopting and using the method for large-scale reproducible analyses.

The rest of the paper is organized as follows. Section 2 describes a use case in which we illustrate the different ways an experiment can be containerized and organized. We then describe how to partially automate this containerization using Sciunit and Popper (Section 3). We conclude in Section 4.

2 Semantic Organization of Containers

To illustrate the needs for semantic organization of containers, we consider an HPC variant of an experiment known to the authors. Food Inspection Evaluation (FIE) [1] is a predictive model comprised of a set of R scripts which aim to predict how likely a food establishment is to be in violation of health codes based on data accumulated about the establishment and its peers. Figure 1 shows the general workflow of FIE, which consists of a preprocessing phase on large amounts of data, a random forest model execution (done locally or in parallel on an HPC cluster), as well as a phase for model evaluation. After evaluation, report and paper generation scripts use the output from the model to generate the human readable results. In particular, FIE includes three kinds of analysis experiments, each of which follow the same general workflow, but may alter the type of the model or the types of data pre-processed, or use the different metrics for evaluation. To generate results for the paper (which a reviewer may want to reproduce), the experiments of FIE are further used in a specific order as shown in Figure 1.

There are two extremes to semantically organizing the FIE experiment. In one extreme, the author can instantiate one container for the entire workflow, conduct all experiments within this container, but let the reviewer determine the sequence of steps that generate a result. Alternatively, the author can instantiate a container before the start of each experiment, but write scripts that describe how to instantiate, build, and run multiple containers to generate a specific result. In other words, the author must make a decision about the granularity at which to share FIE–at the workflow level or the experiment level or some hybrid combination. The first approach is typically faster for the author, and is entirely portable on reviewer's

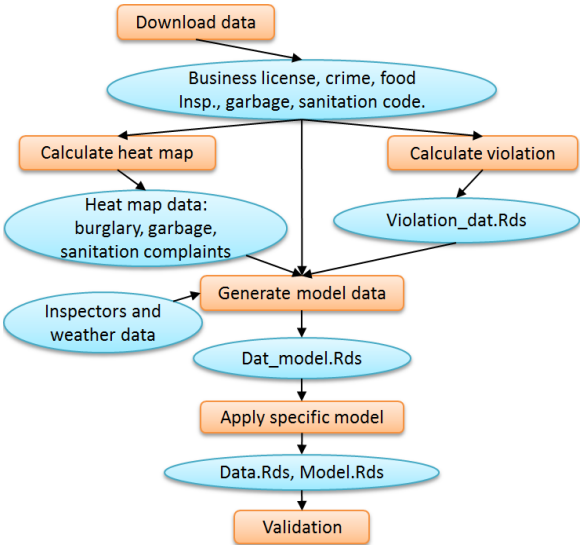

Figure 1: The general workflow of FIE [1]

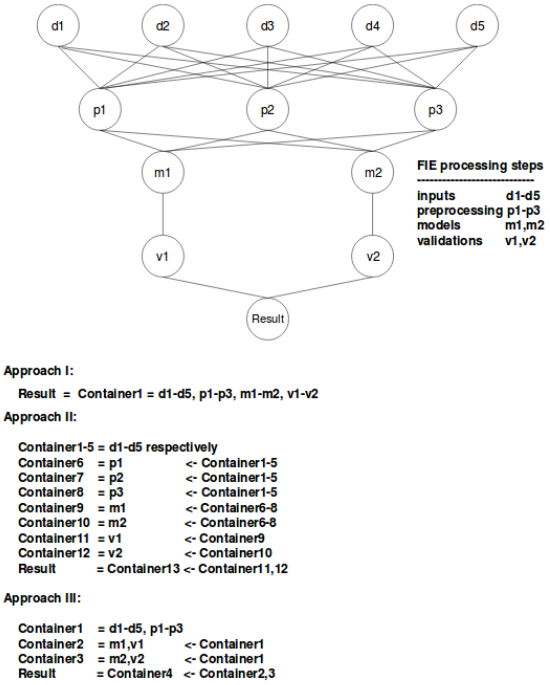

Figure 2: Result can be computed either by (I) executing all steps in one container or by (II) executing each step in its own container and pulling results from previous containers. Another alternative (III) would be to break processing into three logical steps.

environment, but essentially puts the burden on the reviewer to determine the precise scripts to run for the results. In the second or hybrid approach, the author takes more burden easing the task of the reviewer, but by creating multiple containers looses the portability benefit of the entire workflow. Often such organization requires creating scripts that may introduce non-portable commands. For instance, in using the second approach to regenerate a result, we used wget to download containers, but the command was unavailable in a newly instantiated cloud machine. Determining an ideal organization seems a tricky endeavor requiring the reconciliation of two dueling objectives: Create smaller containers which facilitate understanding, but increase the amount of connecting (and potentially non-portable) "glue" scripts. Create larger containers to facilitate portability, but decrease the transparency of the work. Figure 2 shows the two extremes as well as a subjectively assessed ideal organization.

3 Automating Container Organization

Before describing how to automatically create and organize the containers using Sciunit and Popper, we provide a brief overview of Sciunit and Popper.

3.1 Sciunit

Sciunit [5, 8] automatically creates a container of an application by monitoring an application during run-time using *ptrace*, creating a manifest and provenance log of identified dependencies, and packaging all necessary and sufficient dependencies including the provenances of their runs into a container. It further uses *ptrace* during application re-execution time to intercept system calls and redirect them within the container to provide isolation from the host environ-

ment and re-execution guarantees. By automatically creating containers of application execution, Sciunit provides portability (currently across Linux kernels) and obviates the need to programatically create containers (such as in 3^{rd}-party container tools (Docker, Singularity, etc.) that require use of scripts to create containers). It precisely preserves the dependencies used in application execution for later re-execution.

3.2 Popper

Popper [3] uses the concept of pipelines to organize a common generic analysis/experimentation workflow. Popper does not mandate the content of a pipeline, but suggests to build a self-contained pipeline using 'Devops' tools. Given a devops tool that can self-contain the content of a pipeline such as scripts, dependencies, parameterization, results and validations, Popper will allow use of such a tool and term the resulting pipeline as Popper-compliant. By using the concept of pipelines, Popper provides a way to index pipelines by semantic annotations, executes pipelines in their entirety, keeps a history of pipeline executions and results, and a history of any changes to the pipeline or its inputs.

3.3 Sciunit-Popper

Since Sciunit allows flexibility in creating containers from arbitrary grouping of experiments, and Popper provides flexibility in organizing containers, one way to organize the FIE workflow is to use command-line commands as a guide to automatically create and organize containers. For instance if user uses `./script1` to run `d1-d5; p1-p3` then the Popper system can automatically create a container of this script. When the user orchestrates these containers to create results, then using the provenance of the result (present in the Sciunit containers) can be used to instantiate a Popper pipeline. By creating containers, and using multiple containers as part of a Popper pipeline, a reviewer can easily index into a specific result, and re-run that pipeline for generating that result using sciunit containers. While currently Sciunit and Popper are not integrated to automatically create such a pipeline, it is not impossible to do so manually. Currently, we have performed such a manual integration and Fig 3 shows the result of a such a popperized workflow comprising of multiple containers. This popperized workflow is also available via [7]. In future, we plan to use machine learning approaches to learn for which commands to create containers, and how to orchestrate multiple containers as a pipeline.

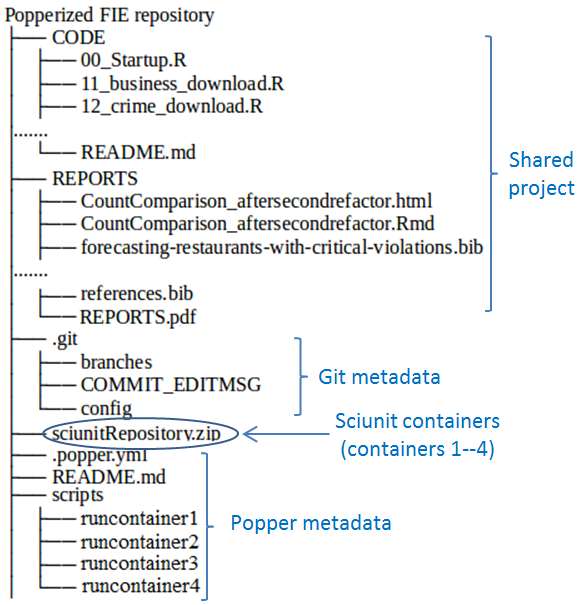

Figure 3: FIE Github repository in Sciunit-Popper

4 Conclusions

We put forward a vision of semantically organized containers for sharing and reproducibility of computational experiments. Containerization is vital for sharing necessary and sufficient dependencies of experiments, and organization is vital for reviewers to find and repeat specific experiments. We describe a method, using Sciunit and Popper, that partially automates the organization of containers. We aim to investigate machine learning approached to achieve full automation.

References

[1] City of Chicago. Food Inspection Evaluation. `https://chicago.github.io/food-inspections-evaluation/`, 2017.

[2] I. Jimenez, C. Maltzahn, J. Lofstead, A. Moody, K. Mohror, R. Arpaci-Dusseau, and A. Arpaci-Dusseau. Tackling the reproducibility problem in storage systems research with declarative experiment specifications. In *PDSW '15*, pages 25–30, New York, NY, USA, 2015. ACM.

[3] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. The popper convention: Making reproducible systems evaluation practical. In *IPDPSW'17*, pages 1561–1570, May 2017.

[4] V. Sochat. The scientific filesystem (scif). *GigaScience*, page 100420, 2018.

[5] D. H. Ton That, G. Fils, Z. Yuan, and T. Malik. Sciunits: Reusable research objects. In *IEEE eScience*, Auckland, New Zealand, 2017.

[6] X. W. Victoria Stodden and V. Sochat. Aim: An abstraction for improving machine learning prediction. In *DSW '18*, pages 150–154. IEEE, 2018.

[7] A. Youngdahl. FIE in Sciunit-Popper. `https://github.com/ayoungdahl/sciPopper`, 2018.

[8] Z. Yuan, D. H. Ton That, S. Kothari, G. Fils, and T. Malik. Utilizing provenance in reusable research objects. *Informatics*, 5(1), 2018.