Reproducible eScience: The Data Containerization Challenge

Tanu Malik
School of Computing
DePaul University
Chicago, IL USA
tanu.malik@depaul.edu

***Abstract*—Computational reproducibility is the cornerstone of the scientific method. We are witnessing a surge of reproducible practices emerging in scientific disciplines. Use of containers is a common theme across several of these practices. Containers isolate applications and make it simple to share code, data, and experiment settings. This paper presents current eScience infrastructures that use containers for reproducible research. We then present our view of the important research issues in data containerization for reproducible eScience. Containerization requires considering packaging, sharing, security, and language choices to ensure reproducibility and preservation. Despite its benefits, containerization can be inefficient and impact experiment reproducibility when users stop using it or transition the experiment to different environments. We describe *reproducible containers* that in addition to containerization include services to ensure and maintain long-term reproducibility and lay forward a vision based on reproducible containers.**

I. Introduction

Translational workflows rely heavily on computation, data, and community resources to accelerate scientific discovery. Efficient management of translational workflows is crucial for solving global problems in health, climate, and governance (1; 2).

Attaining consistent outcomes is a crucial aspect of managing translational workflows. Consistent outcomes create trust in adopted research methods and workflows. However, in computing, even getting the same results with similar data and conditions is sometimes challenging. Reproducibility of workflows has received wide attention in recent times (3). Obstacles to reproducibility are, often, the use of randomization and asynchronous distributed computation, but also evolving libraries and packages on which the program depends. In translational workflows, the reproducibility challenge is particularly increased due to two factors: the use of abstracted modules in workflow construction and the frequent changes made to workflows during execution. The building blocks of a translational workflow are often services. Fragile services often introduce mistakes that result in failures and irreproducibility of workflows. Even with robust services, translational workflow may need changes in execution, like using different computing systems or data providers. It is often challenging to guarantee the reproducibility of a translational workflow with abstracted services and changes in execution often with no pre-recorded comparative baseline.

Containerization has emerged as a mechanism to resolve some of the emerging irreproducibility challenges. Container runtimes protect software from changes in the environment. They do so by encapsulating the software in an image and isolating images from each other (4). This means that containers are completely isolated from one another and the host operating system, and they can run anywhere, regardless of the environment. Container runtimes can now package content declaratively, share images like Git, and schedule cloud-based images. This ensures that applications result in consistent execution on different operating systems. Thus, describing a workflow using containers makes it more reusable and reproducible.

Several workflow management systems (5; 6) support containerization and claim reproducibility of workflows. But use of containers in such systems is akin to a double-edge sword. Containerization does improve translational workflow reusability, but introduces further layers of abstraction, making it harder to identify failures if a fragile service fails. If changes to code, data, and runtime scale are made either to locate or resolve faults, isolation due to containers introduces significant build times which significantly reduces the efficiency of translational workflows.

Our paper presents current eScience infrastructures using containers for reproducible research. We then consider important research issues in data containerization for reproducible eScience. We consider both improvements needed in container technology to better support reproducibility of translational workflows, and improvements needed within reproducible eScience cyber infrastructure (CI) to use containers for conduct of reproducible research. Finally, we show how *reproducible containers* and accompanying services address some of these challenges.

II. Containerized CyberInfrastructure—The Current State-of-the-Art

We start by looking at various containerization software and services that make eScience cyber infrastructure reproducible.

A. Containerization Software and Services

Containerization is a cloud computing service that allows customers to provision, run and manage a bundle of computing

TABLE I: Prominent Container Registries

Container Registries	Domain	Image (I) or Specification (S)	Documentation Support	Version Control	Testing Support	Reproducible Comparison	Workflow Integration
BioCont https://biocontainers.pro/	Bioinformatics	Spec.	✓	✓	✗	✗	✓
Nucleotid.es http://nucleotid.es/	Genomics	Spec.	✓	✗	✗	✗	✓
CERN Container Registry https://hub.docker.com/u/cern	High-energy Physics	Image	✓	✓	✓	✗	✗
Nvidia Container Registry http://bit.ly/45xorKn	HPC/ML/AI	Image	✗	✓	✗	✗	✗
Rocker https://rocker-project.org	Geosciences	Image	✗	✗	✗	✗	✗

platform and applications without the complexity of building and maintaining cloud infrastructure. Containerization falls under the general category of platform-based services (4). The container runtime enables the platform at the operating system-level and ensures isolation of compute, memory, disk and network resources used by an application. Multiple container runtimes, including LXC [1], Docker [2], Apptainer [3], Rocket [4], Mesos [5], and OpenShift [6], are available to facilitate the creation of software application containers and their isolation within target environments. These runtimes ensure isolation and several of them provide create, read, update, and delete (CRUD) services for management of container images and orchestration of containers on multiple nodes. Amongst the available container runtimes, Docker is the most popular containerization platform with the biggest user community.

B. Container Registries in Science Domains

Several science domains have started porting their software in the form of container images. By using containerization they have taken the first step of maintaining reusable software. Some science disciplines have taken a further step in creating community registries of container images. A registry improves sharing and collaboration on common software in a reusable form. They also preserve building blocks of translational workflows. Table I shows the science disciplines that host container registries.

The widest use of containers currently exists within bioinformatics and more specifically in genomics. In high-energy physics, the CERN Container Registry [7] provides container images for various tools and libraries that are commonly used in particle physics, including software for analysing particle collision data and simulations. The NASA Center for Climate Simulation (NCCS) [8] provides a non-public registry of container images. NCCS also uses the public Nvidia NGC Container registry [9] which provides easy access to GPU-optimized containers for deep learning (DL), machine learning (ML), and high performance computing (HPC) applications. Finally, our methodical search and involvement with the geoscience community shows that there is a visible lack of container registries within geoscience. The ArcGIS [10] provides a comprehensive container image of GIS tools and CyberGISX [11] operates a variety of user-contributed analyses using containers. Yet, there is no specific way for users to contribute their own software and tools via containers. The Rocker project [12] provides a collection of R-specific container images for users to build their own images but do not accept contributions from the community.

Since use of containers improves conduct of reproducible research, we evaluate whether registries or the authors contributing containers adhere to reproducible practices such as detailed documentation, version control, testing of containers via practices such as continuous integration, recommendation for composition with other containers, and any included software to allow comparisons with past executions. Table I shows that few registries qualify on all these dimensions. Indeed the current registries are best-effort in that they provide a place to host container specifications or images similar to other software repositories such as GitHub [13] and Zenodo [14] and Figshare [15], but provide no guarantee that the provided software is either reusable or reproducible. We further observe that the verification of these images can be challenging since several containers images are dependent on opaque third party libraries and binary images.

C. Container-based Reproducible CI

Containerization solves the much needed requirement of packaging and sharing software. However, container interfaces are complex—they require semantic understanding of the software application and its environment, programming this

[1]https://linuxcontainers.org
[2]https://hub.docker.com
[3]https://apptainer.org
[4]https://github.com/rkt/rkt
[5]https://mesos.apache.org
[6]https://openshift.com/
[7]https://hub.docker.com/u/cern
[8]https://www.nccs.nasa.gov/nccs-users/instructional/containers
[9]http://bit.ly/45xorKn
[10]https://arcgis.com
[11]http://cybergisxhub.cigi.illinois.edu
[12]https://rocker-project.org
[13]https://github.com
[14]https://zenodo.org
[15]https://figshare.com

semantic understanding into languages supported by the runtime, understanding security issues associated with namespace configuration, managing file system layers and corresponding space requirements, and finally provisioning them on different environments. Container runtimes, so far, do not provide any support beyond isolation that furthers the conduct of reproducible science. New CI and services have emerged given the lack of support within container runtimes to support reproducible research.

Binder (7) and CodeOcean (8) are two commercial software projects that enable scientists to deposit computational artifacts associated with published research and repeat them on cloud resources without interacting with details of containerization. These software projects build upon several research prototypes proposed in the past (9; 10). While both the projects hide the ugly details of containerization and emphasise on reusability of computational artifacts, the supported notion of reproducibility is weak. First neither platforms monitors fine-grained executions and thus cannot assert if repeated executions were the same, leaving it on the user to decide on the result of a repeated artifact. Second hosted computational artifacts cannot be reused portability outside the framework, as the exported artifacts are not container images but simple zip files. Our examination shows that the export often lacks complete description of the environment (11). In other words by hiding the details of containerization, we loose the benefits of containerization.

III. Addressing Reproducibility with Data Containerization—Emerging Challenges

The previous section illustrates that large number of hosted containers do not address reproducibility. Further, current CI for conduct of reproducible science does not offer full benefits of containerization. This section outlines the challenges that need to be solved for data containerization to fully address reproducibility.

A. The Packaging Challenge: What to containerize?

The challenge in packaging is to determine the container's contents automatically. Container runtimes solve this challenge manually: they make container developers responsible for programming the container's contents using a specification. Practically, this method introduces errors and compromises the reproducibility of the containerized software. New methods (12; 13) can detect container contents by analyzing how the application is executed. These tools have shown several benefits to the scientific community. Currently commercial systems such as Binder and Code Ocean do not analyze how the application is executed. They rely on the user or simplify the programmatic creation of container specifications via interactive visual-interfaces.

B. The Data Sharing Challenge: How to share data?

A clear advantage of using containers is that they are more light-weight in virtual images. However, container runtimes are oblivious if the content containerized is excessive. A container encapsulating unnecessary files is slow to distribute and wastes significant quantities of disk storage and network bandwidth at each client. New methods (14) can remove unnecessary files from containers. But these methods treat files as independent entities from the code that uses them. Consequently, the manner in which files are accessed and used is not exploited—that is, no savings are realized when only a small fraction of large files are needed for a particular deployment of the software or if the container is interacting with other containers and thus require data to be preserved.

C. The Audit Challenge: How and where to audit?

Provenance is essential for automatically validating the results generated by repeated execution (15). However, auditing provenance is still not recommended in container practices for reproducible research.. Thus, container developers often do not audit authoritative provenance with which users or systems can compare re-executions. CodeOcean verifies workflow using past history. However, more research is needed to determine how provenance tools can be integrated without affecting the performance of the containerized software. While Docker suggests sidecar patterns for logging of data (16), these patterns are yet to be put in practice for reproducible research or be included as part of a CI. A subchallenge of provenance collection is how to use it to compare containerized workflows. Docker can differentiate container contents but not application execution. Thus automatically determining if any results were not consistently reproduced due to unintentional side-effects or hardware configurations is challenging.

D. The Environment Challenge: How to transition across container-based and non-container-based eScience infrastructures?

Translational workflows are often created in an ad hoc manner. Scientists find that quick workflows with fixed scripts and messy folders save time. If order prevails, scientists may create makefiles and include some READMEs. To restore a reproducible artifact, scientists often use tools later to recover from the reproducibility debt (). While not all debt can be recovered, especially if the project was not versioned, debt due to lack of use of containers can still be recovered. There are two ways to recover the debt: Follow a convention or automate based on the current snapshot of the artifact. The Popper convention [16] shows how containers can be an integral part of such ad hoc workflows but requires additional features such as automated validation and a repository of experiment templates so that environments can be carefully preserved. However, Popperizing of experiments itself requires manual effort and is suitable for non-interactive workflows. Projects such as Chameleon [17] have used a different convention comprising of notebooks integrated with virtualized resources to demonstrate reproducibility of several workflows. However transitioning

[16]https://popperized.github.io/swc-lesson/index.html

[17]https://chameleoncloud.org/

across container-based and non-container-based infrastructures is still a challenge.

An alternative approach is to create container images or specifications from the snapshot of the artifact. Several rules, best practices and expert knowledge can then be used for creating container specifications. Projects such as Basil [18] are exploiting these rules to create optimized images, which have reduced build times. Despite these advancements, creating optimized images from computational artifacts that reduce build time across multiple environments is still an open problem.

IV. Reproducible Containers

Given the above described challenges, we propose the concept of reproducible containers. Reproducible container is not a new way to isolate but a host of services that must be accompanied with a container image, which assert the isolation and reproducibility of the containerized software at all times. We believe reproducible containers can address several of the above mentioned challenges.

In the rest of this section, we describe some tools and services that we have developed which create a reproducible container. The description is more of an overview and we request the reader to further peruse individual references.

1) Addressing the Packaging Challenge: Despite the wide popularity of commercial container technology in the broader DevOps community, our experience is that for most scientists, using containerization as a reproducibility tool involves a steep learning curve that hinders wide adoption. For instance, basic reproducibility questions, such as 'what should be isolated?' — that is, which elements of the application to encapsulate in a container, 'did the container reproduce results similar to those from the host environment?', and 'what if the container has changed since the last time I used it?', are not addressed by the isolation properties provided by the namespaces used to construct containers. We have developed Sciunit (17; 18; 19), which adopts a systematic container-centric approach to address reproducibility for computational and data science applications. The approach combines tracking the reference execution of applications with data provenance and data deduplication (18; 20). It systematically accounts for changes in the application using a data-flow language (17). Sciunit also provides a statement of guarantee when an application reproduces results, and when it doesn't.

2) Addressing the Data Sharing Challenge: There are a variety of methods that can be employed to address the data sharing challenge. Currently, we have consider compile trace monitoring to determine redundant data (21), compiler optimizations to determine which I/O calls are executed (22), and fuzzing to determine the maximum extent of data that must be containerized (23).

3) Addressing the Audit Challenge: We consider the type of auditing necessary to determine differences across workflow runs. To compare or determine differences between individual modules, we show that program specification and provenance traces are necessary to to inform about intra-module differences. We show how program specification and traces can be be used to infer loop differences (20; 24).

4) Addressing the Environment Challenge: We currently address the environment challenge in the context of notebooks. Reproducing notebooks in different target environments, however, is a challenge. Notebooks do not share the computational environment in which they are executed. FLINC (25; 26) shows to share notebooks with its environment which can then be part of any workflow pipeline. CHEX (27) shows how such notebooks can be replayed efficiently.

[18] https://icompute.us/entry

V. Conclusions

In this paper, we have highlighted the need for reproducible containers. We examined containers in science and eScience infrastructure, but found that current usage doesn't guarantee application reproducibility. We have highlighted some challenges of using container technology for reproducible research. Lastly, we presented some emerging research prototypes that tackle reproducibility challenges.

VI. Acknowledgements

This work is supported by National Science Foundation under grants CNS-1846418, NSF ICER-1639759, ICER-1661918 and by the National Aeronautics Space Agency under grant AIST-21-0095-80NSSC22K1485. We acknowledge the collaboration of faculty members Ian Foster, Ashish Gehani, Iyad Kanj, David Tarboton Jonathan L. Goodall, Amitabha Bagchi, Amitabh Chaudhary and Boris Glavic. We acknowledge the collaboration of students and postdocs Zhihao Yuan, Ton That Dai Hai, Moaz Reyad, Quan Pham, Yuta Nakamura, Raza Ahmad, Nithin Manne, Chaitra Niddodi, Aniket Modi, Rohan Tikmany, YoungDon Choi, Binata Roy, And Bakinam Essawy.

References

[1] M. Parashar, A. Friedlander, E. Gianchandani, and M. Martonosi, "Transforming science through cyberinfrastructure," *Commun. ACM*, vol. 65, no. 8, p. 30–32, jul 2022. [Online]. Available: https://doi.org/10.1145/3507694

[2] S. H. Woolf, "The meaning of translational research and why it matters," *Jama*, vol. 299, no. 2, pp. 211–213, 2008.

[3] E. National Academies of Sciences and Medicine, *Reproducibility and Replicability in Science*. Washington, DC: The National Academies Press, 2019. [Online]. Available: https://nap.nationalacademies.org/catalog/25303/reproducibility-and-replicability-in-science

[4] C. Pahl, "Containerization and the PAAS cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015.

[5] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[6] J. Köster and S. Rahmann, "Snakemake—a scalable bioinformatics workflow engine," *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, 2012.

[7] "Binder:turn a git repo into a collection of interactive notebooks," https://mybinder.org/, accessed 3/4/2019, 2019.

[8] "Code ocean: Discover and run scientific code," https://codeocean.com/, accessed 3/4/2019, 2019.

[9] V. Stodden, C. Hurlin, and C. Pérignon, "Runmycode. org: A novel dissemination and collaboration platform for executing published computational results," in *2012 IEEE 8th International Conference on E-Science*. IEEE, 2012, pp. 1–8.

[10] A. Brinckman, K. Chard, N. Gaffney, M. Hategan, M. B. Jones, K. Kowalik, S. Kulasekaran, B. Ludäscher, B. D. Mecum, J. Nabrzyski *et al.*, "Computing environments for reproducibility: Capturing the "whole tale"," *Future Generation Computer Systems*, vol. 94, pp. 854–867, 2019.

[11] J. Chuah, M. Deeds, T. Malik, Y. Choi, and J. L. Goodall, "Documenting computing environments for reproducible experiments," in *Parallel Computing: Technology Trends*. IOS Press, 2020, pp. 756–765.

[12] T. Malik and et. al., "Sciunit," https://sciunit.run/, 2017, [Online; accessed 20-July-2021].

[13] F. Chirigati, R. Rampin, D. Shasha, and J. Freire, "ReproZip: Computational reproducibility with ease," in *SIGMOD'16*, 2016, pp. 2085–2088.

[14] J. Thalheim, P. Bhatotia, P. Fonseca, and B. Kasikci, "Cntr: Lightweight {OS} containers," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 199–212.

[15] S. B. Davidson and J. Freire, "Provenance and scientific workflows: challenges and opportunities," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1345–1350.

[16] B. Burns and D. Oppenheimer, "Design patterns for container-based distributed systems," in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[17] Z. Yuan, D. H. Ton That, S. Kothari, G. Fils, and T. Malik, "Utilizing provenance in reusable research objects," *Informatics*, vol. 5, no. 1, 2018.

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

[19] A. Youngdahl, D. H. Ton That, and T. Malik, "SciInc: A Container Runtime for Incremental Recomputation," in *IEEE eScience*, 2019.

[20] Y. Nakamura, T. Malik, and A. Gehani, "Efficient Provenance Alignment in Reproduced Executions," *12th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2020. [Online]. Available: http://www.csl.sri.com/users/gehani/papers/TaPP-2020.Alignment.pdf

[21] C. Niddodi, A. Gehani, T. Malik, J. Navas, and S. Mohan, "MiDas: Containerizing Data-Intensive Applications with I/O Specialization," *3rd ACM Workshop on Practical Reproducible Evaluation of Computer Systems (P-RECS)*, 2020. [Online]. Available: http://www.csl.sri.com/users/gehani/papers/PRECS-2020.MiDas.pdf

[22] C. Niddodi, A. Gehani, T. Malik, S. Mohan, and M. Rilee, "IOSPReD: I/O Specialized Packaging of Reduced Datasets and Data-Intensive Applications for Efficient Reproducibility," *Access*, vol. 11, 2023. [Online]. Available: http://www.csl.sri.com/users/gehani/papers/Access-2023.IOSPReD.pdf

[23] "Kondo," https://github.com/depaul-dice/kondo, 2023.

[24] R. A. Yuta Nakamura and T. Malik, "Content-defined merkle trees for efficient container delivery," in *27th International Conference on High Performance Computing, Data, and Analytics*. IEEE, Jun. 2020.

[25] R. Ahmad, N. N. Manne, and T. Malik, "Reproducible notebook containers using application virtualization," in *2022 IEEE 18th International Conference on e-Science (e-Science)*. IEEE, 2022, pp. 1–10.

[26] "FLINC," 2022, [Online; accessed 1-Sep-2022]. [Online]. Available: https://github.com/depaul-dice/Flinc

[27] N. N. Manne, S. Satpati, T. Malik, A. Bagchi, A. Gehani, and A. Chaudhary, "Chex: Multiversion replay with ordered checkpoints," *Proc. VLDB Endow.*, vol. 15, no. 6, p. 1297–1310, feb 2022. [Online]. Available: https://doi.org/10.14778/3514061.3514075