A Report on the "**Symposium on Challenges and Opportunities in Containerization of Software and Data Products"**

Held on July 23, 2023, 9 AM to 5:00 PM, at Portland, Oregon

Co-Located with PEARC23

Event Website: https://sites.google.com/view/containerization2023/home

Submitted by
Ritu Arora, Wayne State University (ritu@wayne.edu)

Symposium Participants/Contributors

Aragorn Steiger (+1), Wayne State University
Dan Harris, National Renewable Energy Laboratory
Daniel Perrefort, University of Pittsburgh
Daryl Williams, IBM
Dmitry Duplyakin, National Renewable Energy Laboratory
Frank Lee, IBM
Geoffrey Lentner, Purdue University
Guangzhen Jin, Purdue University
Irfan Elahi, UCAR
Izzat Alsamadi, Texas A&M University, San Antonio
Jason Nucciarone, Canonical
Joanna Wong, IBM
John Yun, Ansys
Joseph Latessa, Wayne State University
Kevin Menear, National Renewable Energy Laboratory
Lev Gorenstein, Purdue University
Matthew Leon Curry, Sandia Laboratory
Michael Reekie, Harvard University
Moaz Reyad, DePaul University
Ritu Arora, Wayne State University
Ronald O Rahaman, Georgia Institute of Technology
Sanjeeth Boddinagula, Wayne State University
Sukrit Sondhi, Macmillan
Tanu Malik, DePaul University
Tom Boyle, Kennesaw State University
Wei Fienstein, Lawrence Berkeley National Laboratory
Yucheng Zhang, Purdue University

Executive Summary

The "containerization" of software and data products future-proofs them, helps in their long-term preservation, makes them portable across different hardware platforms, ensures reproducible results, and makes them convenient to disseminate. Docker and Singularity are two popular software technologies for containerizing scientific applications and are widely supported on different hardware platforms. This symposium was planned to be a summit of talks and activities on challenges and opportunities related to containerization using Docker and Singularity. Some of the topics that were in the scope of this symposium included: (1) processes for developing optimized, secure, and trustworthy, Docker/Singularity images, (2) test cases that can benefit from containerization, (3) using containers on HPC and cloud computing platforms, (4) best practices in containerization, (5) selecting licenses for Docker/Singularity images, (6) tools and techniques for facilitating the containerization and dissemination of software and data products, and (7) adoption and learning curve related to containerization.

The symposium brought together (1) the colleagues interested in containerizing their software and data products, and (2) the service providers who can help in meeting such needs. The symposium was co-located with PEARC23, was scheduled before the conference began, and there was no cost/fee to attend this event. Travel support was provided to some individuals for attending the symposium. The following activities were covered during the event:

1. Ice-breaking session, introductions, formation of groups for the brainstorming session
2. Invited talks
3. Brain-storming sessions
4. Lunch
5. Presentations from each group engaged in the brainstorming session

The complete agenda of the event is available at the following URL:
https://sites.google.com/view/containerization2023/schedule

The presentation slides, where available, are linked to the schedule mentioned at the aforementioned URL. A summary of the presentations related to the brainstorming sessions is included in this report. A LinkedIn.com group was formed to connect those interested in networking with each other even after the event was over.

Summary of Brainstorming Sessions

Groups were formed early during the event, and each group discussed challenges and opportunities related to the containerization of applications using Docker and Singularity. They were requested to discuss and document test cases related to containerization, and prepare a presentation summarizing their findings. A summary of each group's discussion and findings is presented below.

Group 1

This group identified the following challenges in containerization:

1. Building MPI inside the containers with Podman (a daemonless container engine developed by Red Hat and is an alternative to Docker/Singularity) and building and running multi-node MPI jobs can be challenging.
2. Building GUI apps in containers can be tricky.
3. Building different containers for different CPU architectures - multi-arch containers - can be difficult, and can increase the number of images to maintain.
4. The containers lack version metadata inside them.
5. Images/containers too large in size can be challenging to handle.
6. The process of cache image registry may not be clear to all.
7. Using specific datasets with a container can be tricky.
8. Building a dataset container having too many files can be challenging.
9. Containerized apps are sometimes not performant.
10. Best practices to build apps with different runtime for different containers are lacking/adopted.

The following opportunities were listed by the group:

1. Use Podman (it is more like Docker with rootless ability) to build containers.
2. Use a building tool named Apptainer (previously named Singularity) on HPC systems.
3. Use Spack to containerize environments.
4. Users have a hard time building container images on HPC clusters. They need to find their own Linux machines to build their own images.
5. Add an extra layer to run apps outside of containers.
6. Share the build recipes with each other.
7. Advanced training for containerization.

The biocontainer wiki page from the Purdue University was shared as an example of the containerization effort in the community: https://www.rcac.purdue.edu/knowledge/biocontainers

Group 2

This group discussed the best practices for physical storage and noted the need for developing a matrix of options for selecting the different filesystems for storage solutions. The group mentioned bioinformatics as a sample domain. It was noted that there is a lot of reference data in this domain. Gathering data in one place rather than separately downloading each dataset or image containing datasets should be supported. The group also wondered as to how multiple containers would share the same data. The author notes that one solution for this scenario is to build shared volumes that can be mounted in different containers and Docker provides a feature for linking volumes between running containers.

The group also discussed the topic of security in containers and wondered if running containers without any special privileges is sufficient for ensuring the security of the system.

There was a discussion of different types of storage strategies for different workflows. The applications or workflows that require data streaming (or real-time data ingestion and

processing), and the ones that require decoupled steps for data ingestion and processing need to be handled during the containerization effort. There should be mechanisms for handling concurrent data processing and supporting hybrid modes of data ingestion and processing.

The topic of compatibility of the chosen operating system during containerization with different libraries was discussed (e.g., GLIBC compatibility on Centos 7 / RHEL 7).

The topic of developing best practices for containerization also emerged.

Group 3

This group discussed the different categories of applications that can be containerized, such as web applications and scientific software. It was noted that the “Twelve-factor principle” for developing software-as-as-service is applicable for containerization as well and could be used as best practices for containerization. The twelve factors (https://12factor.net/) as related to the twelve-factor principle are as follows:

1. There should be one codebase - tracked in a version control system - and multiple deployments (such as, production, testing, development).
2. The software dependencies should be explicitly declared and should be isolated from the system-wide dependencies.
3. The configurations (such as database passwords and IP addresses for websites) should be stored as environment variables and should not be made a part of the code.
4. The backing services, or any services that the applications use over the network for normal operations (such as, SQL databases), should be treated as attached resources that can be plugged in and out without impacting the code of the applications.
5. The build, release, and run stages for code development and deployment should be separated from each other.
6. The application processes should be stateless and share-nothing, and persistent data should be stored in a stateful backing service (such as a database).
7. The applications should be self-contained and all services should be exported via port binding and listening to specific ports.
8. Scaling out an application via the process model should be supported. As an individual VM can only grow to a certain extent, the applications should be able to scale out such that they are able to span across multiple processes running on multiple physical machines.
9. The application processes should be disposable such that they can be quickly started and shutdown gracefully to support elastic scaling and fast deployments.
10. The development, staging, and production environments should be as similar as possible.
11. The application should not be made responsible for writing or managing logfiles. Instead, each running process should be made responsible to write its event stream, unbuffered, to

stdout that can then be used as logs and archived and visualized later in a persistent storage.

12. One-off administration processes should be run in the environment that is identical to the long-running processes in the deployment environment, and the code for system administration should be shipped along with the rest of the application code to avoid any synchronization issues.

Some examples of best practices specifically for containerization include: (1) adding tags to the images, (2) build staging - caching tasks to optimize build time, (3) integration with CI/CD systems to build images (GitHub Actions, GitLab runners, etc.), (4) agile development: images can be built iteratively, and (5) autoscaling and using sidecar services.

Having a global registry for Apptainer (Singularity) images was noted as an opportunity for community development. Education around containerization and having user-friendly documentation were noted as well. It was noted that typically domain scientists may not be using containers already and may require support for application containerization. Caching the steps for building images and integrating support for different hardware architectures were also noted as opportunities in the area of containerization.

The group noted that it can be challenging to replicate issues related to the software applications that can be used or run on heterogeneous systems. Containerization can be extremely useful in such scenarios. Additional key advantages noted for using containerization include: supporting reproducibility, distributing sandboxed code for workshops, training and education purposes. Researchers working in different domains will find containerization useful for sharing their software with others.

Conclusions

Containerization of scientific applications and workflows will continue to be important for their deployment, management, and sharing. Scaling of containerized applications, optimization of container images, training, and community-building are some of the dimensions that need further attention and effort. The discussions on best practices for containerization should continue and the state-of-the-practice should continue to evolve with the evolution in the landscape of the tools and technologies required for containerization.

Acknowledgement

We are grateful to the National Science Foundation (awards #2209946 and #2314201), IBM, and Venra Tech for sponsoring the “Symposium on Challenges and Opportunities in Containerization of Software and Data Products". We are grateful to Wayne State University for help in processing the reimbursements for travel support.