USING DOCKER TO SUPPORT REPRODUCIBLE RESEARCH

Ryan Chamberlain, PhD
Invenshure, LLC
Minneapolis, MN

Jennifer Schommer
Invenshure, LLC
Minneapolis, MN

July 14, 2014

Abstract

Reproducible research is a growing movement among scientists, but the tools for creating sustainable software to support the computational side of research are still in their infancy and are typically only being used by scientists with expertise in computer programming and system administration. Docker is a new platform developed for the DevOps community that enables the easy creation and management of consistent computational environments. This article describes how we have applied it to computational science and suggests that it could be a powerful tool for reproducible research.

Keywords: sustainable software, reproducible research, Docker, virtualization, code sharing, computational science, open science

Introduction

It is difficult to overstate the importance of computational algorithms to scientific research. Despite the central role that scientific computing plays in research, computational experiments are notoriously difficult to reproduce and share[1−5]. Most peer-reviewed publications - even those that serve solely to describe a novel computational algorithm - do not provide enough detail on an algorithm's implementation to enable readers to re-implement it themselves[1,5]. It is even more rare for the author to share the source code used to generate the results being published. Even if the source code is made available, it is only a single piece of a large and complex system that creates the entire computational environment that would make the algorithm of interest reproducible. Furthermore, even if all the necessary information is available to recreate the full computational environment, many researchers that focus on non-computational science specialties (i.e. medicine or economics) lack the time and expertise to manage the required installations of third-party libraries or implement from scratch state-of-the-art data processing algorithms[6−8]. Therefore, the problem of reproducible computational research is amplified the further the discipline sits from computer science expertise. This paper introduces a new project, Docker, that could be a powerful tool to overcome these obstacles and support reproducible research in scientific computing.

Existing Tools for Reproducible Research in Science

Many tools have been developed over the past decade that enable software developers to share source code (e.g., Sourceforge and GitHub). There have also been many federally funded efforts to manage tool and data sharing, such as caBIG[9] and the Cancer Imaging Archive[10], and recent projects have developed cloud-hosted computing (e.g., HubZero[11] and PiCloud[12]). The common theme among all of these projects, however, is that they require a high level of expertise in computer programming and system administration. In many cases, they also require users to adopt specific languages, data formats, or workflows.

A brute-force approach to sharing scientific computation is through virtual machines. Virtual machines (VMs) are a safe and predictable way to share a complete computational environment. However, VMs have serious drawbacks in the context of reproducible research and sustainable software for science. First, they are cumbersome to build reproducibly without a high level of systems administration knowledge. Second, they take up a lot of storage space, which makes them onerous to share and they quickly fill up disk space if a developer is using a separate VM for every algorithm or analysis. Third, they are difficult to reliably version and archive.

The ideal tool for reproducible research would be easy to use, accommodate all languages and libraries, be backwards compatible for decades, and enable easy sharing. A tool that takes a large step towards meeting this description is Docker (http://www.docker.com).

Docker

Docker is a platform that uses Linux Containers (LXC) to completely encapsulate software applications. It is an open source project backed by a private company that has been focused on developing Docker as a platform for easy and scalable web application hosting.

LXC is an operating system level virtualization technology that creates a completely sandboxed virtual environment in Linux without the overhead of a full-fledged virtual machine. Docker extends the LXC technology, makes it user-friendly, and provides easy versioning, distribution, and deployment. The overhead of a Docker container is much smaller than a traditional virtual machine because it does not replicate the full operating system, only the libraries and binaries of the application being virtualized.

A Docker container runs in a fully virtualized environment therefore an algorithm can be run using any Linux compatible language (R, Python, C, Matlab, etc.) and be compiled using any Linux compatible libraries without causing version or library conflicts with other algorithms. Furthermore, a Docker container can be exported and archived, then run anytime in the future with confidence that the computational environment will be identical, assuming the Docker project remains backwards compatible. More detail on Docker can be found at: http://www.docker.com/whatisdocker/.

Beyond providing an isolated and consistent computational environment, Docker provides many features that make it an attractive tool for reproducible research:

- Docker containers can be built using a simple script, so the container itself can be versioned, shared, and reproducibly re-built with a single text file (a Dockerfile)
- The Docker container itself can be versioned and forked, similar to a Git repository, and shared directly using Docker's hosted repository, the Docker Hub
- Unlike a VM, the Docker container's contents are restored to their original condition every time the container is launched, ensuring a consistent computational environment
- Data, documentation, and other files can be encapsulated within the Docker container, so a single vehicle can be used to share the entire computational experiment
- There is a large user base in the DevOps and web application communities that provides a wealth of information on use cases and troubleshooting
- Docker containers own only the resources they are actively consuming, so there is minimal overhead to running a Docker container compared to running a native application, unlike a VM where one needs to statically provision memory to the VM for its entire uptime
- Directories on the host file-system are easily mounted into the Docker file-system, so data and source code can be seamlessly shared between the two

Using Docker for Science

Docker was developed as a tool for web applications not scientific computing, so there is not a standardized workflow for performing scientific work using Docker. The following is a collection of workflows we have developed to use Docker in our scientific computing with minimal distraction. There are possibly alternative methods and strategies, but these have proven effective for our work.

There are two ways to build Docker containers: interactively or through a Dockerfile. When building a container interactively one is able to install libraries and configure the environment from a shell just like in a typical Linux environment. The changes can be saved through commits similar to how source codes changes are tracked in Git. Full documentation is available at https://docs.docker.com/userguide/. However, when interactively building a container, the process cannot be reliably repeated, which is not ideal for reproducible research. A better approach is to build the container entirely through commands in a Dockerfile. A Dockerfile identifies a source container to start from (typically a basic Linux installation), then records a series of commands to install libraries and configure the environment. Dockerfiles can also load other Dockerfiles allowing for environment layering and concise organization of various software development projects. Dockerfiles can be versioned using a standard source control versioning tool like Git, allowing revision tracking and archiving of the computational environment. More information can be found at https://docs.docker.com/reference/builder/.

We have broken down our workflow to two broad use cases: development and distribution. In the development workflow, we are actively working on the source code of the algorithm, and in the distribution workflow we are sharing a released version of the algorithm with others. In order to have the same reproducible computational environment in both workflows, we have adopted a system comprising three Docker containers (managed through Dockerfiles): base, dev, and release. The base Docker container contains all of the libraries and tools that the algorithm itself needs to run. The dev Docker container uses the base container as a starting point then adds tools that are useful in development (IPython and matplotlib are the most common for us), but that are not necessary for a release of the software. This architecture is summarized in Figure 1.

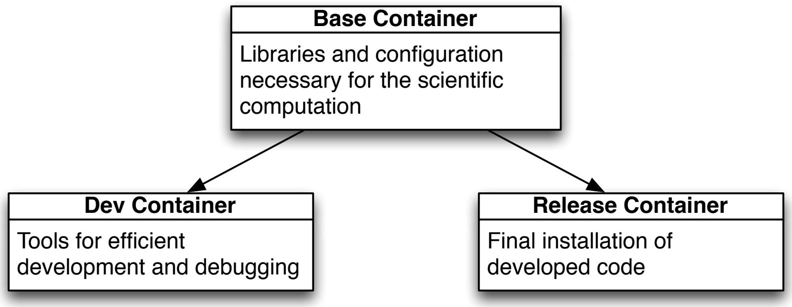
Base Container
Libraries and configuration necessary for the scientific computation
Dev Container
Tools for efficient development and debugging
Release Container
Final installation of developed code

Figure 1: Hierarchy of Docker containers

The dev Docker container does not include the source code for the algorithm under development. Instead, the source code is mounted from the host file-system. In this setup, the development environment (i.e. the text editor or IDE) is running on the host system, not within the Docker container. This allows the source code changes made during development to be reflected in the dev Docker container when it is executed. The release container also uses the base container as the starting point but then installs the released version of the algorithm within the container. Figure 2 summarizes the differences between the dev and release Docker containers. An example of this setup is available on GitHub at https://github.com/rmchamberlain/docker-repro-research.

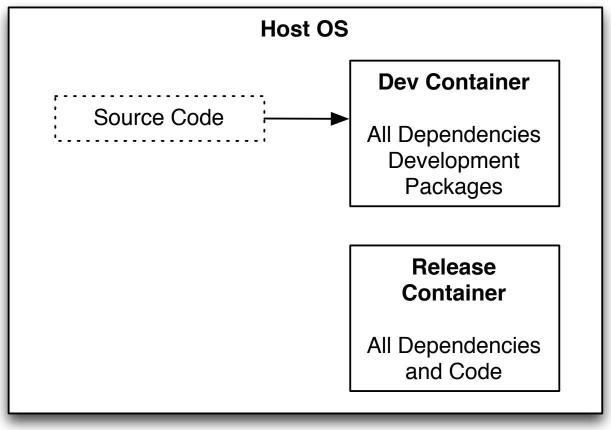

Figure 2: Summary of differences between the dev and release Docker containers

Conclusion

In order for a more widespread adoption of reproducible research practices, the required tools and workflows must become more subtle and easy to use for scientists with minimal computer skills. Docker by itself is a step forward in our view, but, more importantly, we believe that combining it with a tool like IPython, it could be a fundamental component of a platform that makes creating reproducible research simple and straightforward for the greater scientific community.

References

1. Donoho D, Maleki A. Reproducible research in computational harmonic analysis. Comput. Sci. Eng. 2009;11(1):8-18.
2. Stodden V, Bailey D, Borwein J. Setting the Default to Reproducible. stodden.net. 2013:1-19.
3. Stodden V. Trust Your Science? Open Your Data and Code. Amstat News. 2011;409:21-22.
4. LeVeque R, Mitchell I, Stodden V. Reproducible Research for Scientific Computing: Tools and Strategies for Changing the Culture. Comput. Sci. Eng. 2012;14(4):13-17.
5. Freire J, Silva C. Making Computations and Publications Reproducible with VisTrails. Comput. Sci. Eng. 2012;14(4).
6. Avila-Garcia MS, Trefethen a. E, Brady M, Gleeson F, Goodman D. Lowering the Barriers to Cancer Imaging. 2008 IEEE Fourth Int. Conf. eScience. 2008:63-70.
7. Kagadis GC, Kloukinas C, Moore K, et al. Cloud computing in medical imaging. Med. Phys. 2013;40(7).
8. Guo P. CDE: A Tool for Creating Portable Experimental Software Packages. Comput. Sci. Eng. 2012;14(4).
9. https://cabig.nci.nih.gov
10. http://www.cancerimagingarchive.net
11. http://www.hubzero.org
12. http://www.picloud.com