A Containerized Approach for Data Analysis on Omega

Aidan Sciortino
Wilson Magnet High School

Advisor: Richard Kidder
University of Rochester Laboratory for Laser Energetics

January 2019

Abstract

Scientific data analysis is critical to the work done at LLE. An important part of this is enabling easy access to data and compute resources for scientists, both on the day of a shot and afterwards. Current systems for accessing data for analysis are old, hard to access, and have long download times. This makes quick analysis between shots very difficult. This project explores possible ways to remedy this, presenting a container-based system allowing web-based development of analysis programs, with easy access to data on the webpage itself. This is part of a larger study of containerization as a way to provide compute resources across the lab, scaled according to need.

1 Introduction

1.1 Challenges at LLE

One of the major challenges that needs to be addressed at LLE is how data access is provided to scientists, both for on-site PIs as well as for visiting PIs. Currently on-site PIs connect directly to the facility SQL database, or scrape data off statically generated webpages based on technologies popular in the 1990s.

While external PIs can access these webpages beyond the firewall, they are unable to access the SQL database and must rely on on-site scientists to share any data that can't be found on the webpages with them by email.

HTML scraping or writing SQL is inefficient and inhibits productivity, forcing scientists to write extra code—and in the case of SQL learn a new programming language—instead of allowing them to focus in on their specific research. In addition, writing SQL requires somewhat intricate knowledge of the data structures within the database in order to build queries, without which pulled data may be inaccurate.

Another major challenge encountered at LLE is the amount of time it often takes for a scientist to access and analyze data from a shot. In order to access data scientists must download it to their workstations, where analysis code must then be run. While some analyses may be run well after the day of the shot, others are optimal to run in between shots in order to catch and rectify any problems in the experimental setup. The process of loading data

and running these procedures is inconsistent, varying according to many factors including but not limited to the age of the computer, the speed of the user's internet connection, and the amount of data that must be loaded. In cases where resources are unavailable, running analyses between shots is not feasible. A containerized approach for scientific data analysis provides a solution for consistent, secure access to data, services, and resources that will help scientists formulate new methods for checking data in between shots, allowing further sharing of responsibilities and better utilization of facility resources.

1.2 Modern Technologies

Since the SQL database and HTML pages were set up in the late 1990s, countless new approaches to provisioning programmatic data access for users have emerged. In addition, many new technologies have emerged to allow for easier allocation of computing resources, including the concept of containerization. Containerization builds on the concept of Unix process isolation, in which the root directory of a process is changed such that it doesn't affect other processes. This concept evolved, through the concept of "Jailing" processes by partitioning a full system into several smaller systems, each of which is assigned its own IP address and system configuration. As cloud-based hosting grew and there became more and more demand for technologies allowing allocation of compute resources for various cloud websites, the concept of the container was embraced by many providers as a system for isolating and allocating resource usage in cloud data centers.

In 2013 the Docker container system was introduced, which made containers significantly easier to create. With Docker emerged Docker Hub, a platform that allows users to share container images for other users to download and use. In 2014 Google introduced Kubernetes, an open source container orchestration platform. Kubernetes solves the problems that Docker does not, coordinating the way in which multiple containers share resources and interact.

In the past five years containerization has grown from a fringe technology used by few companies for very specialized purposes in web hosting to the de-facto standard for deploying

modern applications, databases, and other stateless infrastructure.

2 Approaches to Service Compartmentalization

2.1 Bare Metal

In a bare metal architecture (shown in figure 1) an operating system runs directly on the hardware, and holds libraries that all applications hosted on the server share.

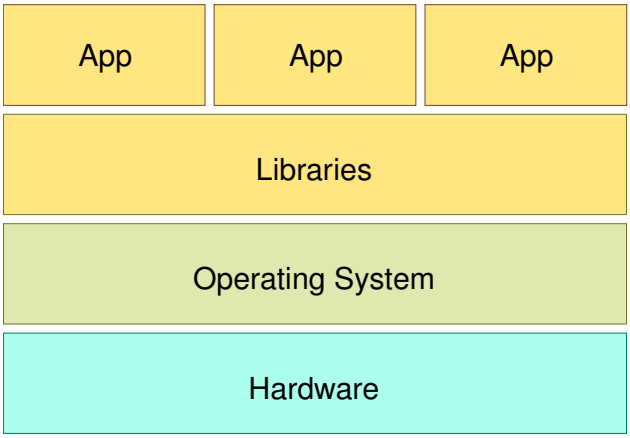

Figure 1: Traditional Bare-Metal System Architecture

This architecture prevents inefficient usage of storage space and processing power because the applications share libraries and operating system resources with each other. However, since applications share libraries, dependency conflicts (in which one application requires a different version of a library than another) are common.

2.2 Virtualization

In a virtualized architecture (shown in figure 2) an operating system still runs directly on the hardware, but its sole purpose is to run the virtualization software. The virtualization software allows for the creation of multiple virtual machines, each of which has its own operating system, and can hold its own libraries and applications.

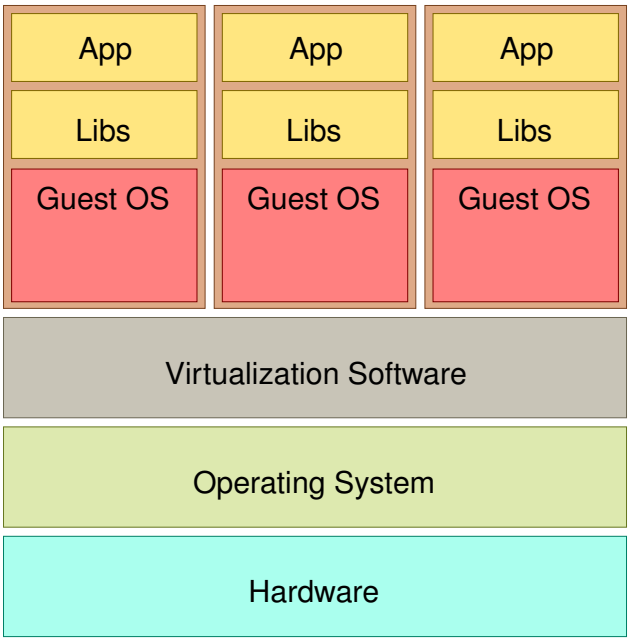

Figure 2: Virtualized System Architecture

This architecture avoids the dependency conflicts described in section 2.1 by providing each application with its own individual dependencies. In many cases this architecture is also more secure due to the compartmentalized nature of the virtual machines. If one application is compromised it does not mean that the entire system is; solely that machine. In comparison, in a bare-metal architecture if one application is compromised all other applications on the machine are vulnerable.

The primary disadvantage of virtualization is that an entire computer must be simulated and an entire operating system must be virtualized, which uses significant overhead. This results in less efficient systems and often has a negative impact on the performance of processes running in virtual machines.

2.3 Containerization

A containerized system (shown in figure 3) presents a hybrid of the two previously mentioned approaches. The operating system runs a container engine, similarly to how the operating system in a virtualized system runs virtualization software. However, this container engine

does not simulate an entire machine for the apps contained in containers. Instead it passes through core parts of the underlying host operating system, focusing on modularising only the dependencies for each app in each container.

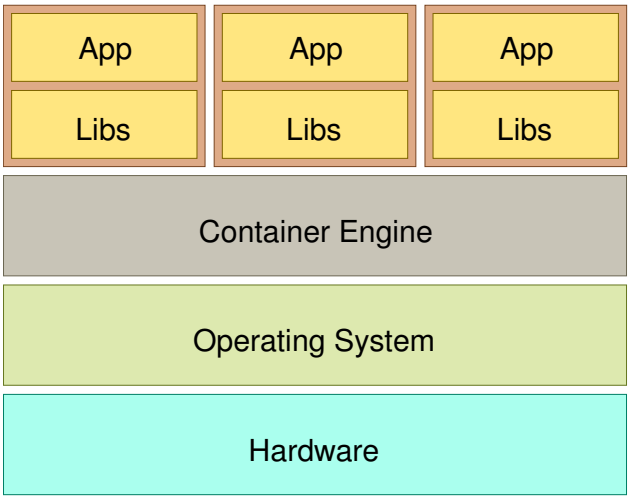

Figure 3: Containerized System Architecture

By sharing core parts of the operating system to each container, containerization avoids the inefficiency of simulating entire computers and virtualizing entire operating systems as described in section 2.2. However, it also avoids the dependency conflicts described in section 2.1 by providing individual libraries based on containers, and maintains many of the security benefits of a virtualized architecture.

In addition, by passing through core parts of the host operating system, containerization avoids the performance costs associated with virtualization.

3 Proof of Concept Application

The *Shotday* web application was planned and developed to demonstrate the possibilities that containerization offers for data analysis solutions internally and externally. The application builds upon modern container technology with modern web technologies such as Angular[1]

[1] `https://angular.io/`

and NodeJS[2], and data analysis solutions such as Jupyter[3].

3.1 Application Architecture

The application is based around the architecture depicted in figure 4. A frontend, written using the Angular web framework, connects a user to a Linux container that serves as a backend. This backend runs a Jupyter data analysis kernel that allows the user to write code in Python 3 or Matlab. This container has access to the LLE database using the user's authentication, such that code written in it can be executed anywhere, and still access facility resources.

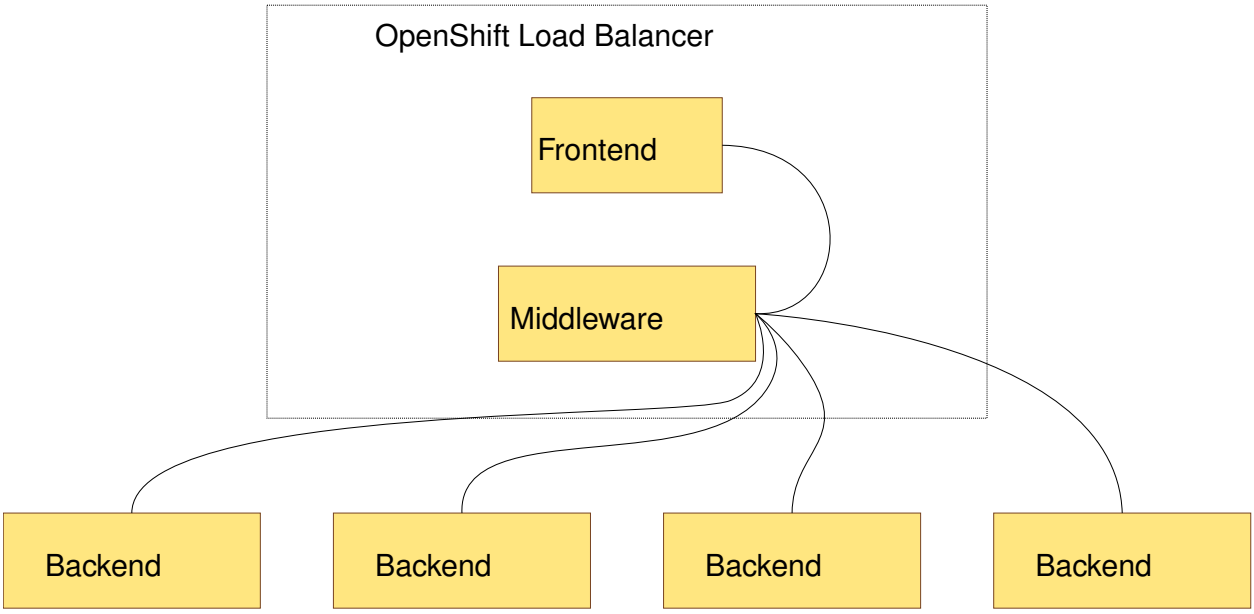

Figure 4: *Shotday* Architecture

The containers are all contained within the OpenShift[4] containerization platform. This platform extends the Kubernetes[5] container orchestration platform, which is the industry standard for managing containers in a production environment. This platform provides easy management of container infrastructure that is easily scalable. It also automates tasks such as load

[2] https://nodejs.org/
[3] https://jupyter.org/
[4] https://www.openshift.com/
[5] https://kubernetes.io

balancing and allocating resources such as processing power, memory, and storage space to containers.

The frontend and middleware are both designed to be stateless, such that the containers themselves can be destroyed and recreated without users losing data. They are also designed to be load balanced using the load balancing technologies included in OpenShift.

The backend containers are also designed to be stateless, with user programs being stored in the facility's Gitlab code management system. However, the backend containers are managed by the Jupyter Enterprise Gateway, a piece of software designed by developers at IBM and several other companies that manages Jupyter kernel deployment in High Performanc Computing (HPC) and containerized environments.

4 Integration with Current Infrastructure

4.1 Current LLE Infrastructure

Currently LLE possesses an Oracle SQL database that stores data from laser shots. This database provides data to webpages, which are used by different people across the lab for different purposes.

Many scientists run data analysis routines on personal computers—generally either workstations or laptops. The lab also has several sources of high performance computing (HPC) power, two on site as well as one off site, that are used for data analysis as well as for simulations and other HPC-based experiments.

In addition, there is an existing code management tool called Gitlab that allows easy sharing of and collaboration on code.

4.2 Proposed Containerized Architecture

The proposed containerized application provides scientists with a secure Linux container that has easy access to all of the existing resources from anywhere in the world; not solely behind the LLE firewall.

These containers have libraries that are designed to make pulling from the database simple, have access to the facility Gitlab system, and are designed with the intention of eventual expansion allowing jobs to be executed on facility HPC resources.

Containers can be managed using existing facility infrastructure including the authentication servers already in use for both existing web services and user authentication into desktop computers. In addition, containers present further security benefits as discussed in section 2.3, as such presenting a method that is as secure, and possibly even more secure, than the methods currently used for data analysis.

5 Conclusions

The proof of concept presented in this paper serves only the function of providing scientists with a web-based interface for doing data analysis. While containerization serves this specific application well, it also holds significant opportunities and advantages that could be used to enhance the lab's existing infrastructure.

For example, stateless containers could be used to provide access to the facility database via a web Application Programming Interface. This allows access to data without needing to learn SQL or other programming languages. It also allows for easier integration of modern web applications with facility data. Data services such as these are growing more common, as more and more applications become web based. Containerization presents a thoroughly future-proof method for implementing such services, and thus is a valid route forwards that should be taken into consideration for use in the lab.

6 Acknowledgements

First and foremost I'd like to thank my advisor, Richard Kidder, for all of his guidance on this project. Without Rick's vision this project would not exist at all, and without his guidance in connecting me with people around the lab, as well as his immense knowledge of the operations of the facility, I would not have been able to get anywhere close to the proof-of-concept presented in this paper.

I'd also like to thank Michael Charissis, who set up the infrastructure required for the project and from whom I've learned immense amounts about server administration and what todays enterprise technology landscape looks like.

In addition I'd like to thank the informatics team (Andrew Zeller, Tyler Coppenbarger, and Mathew Schweigardt), with whom I worked extensively investigating the feasibility of integrating the proof of concept into their existing code.

Finally I'd like to thank Dr. Craxton for organizing and running the summer intern program, as well as all of the other summer interns with whom I had countless discussions in topics in computer science, physics, and philosophy, and whom challenged me every day to think harder and longer about many problems in all spheres of my work.