Containerized Application Management for Cloud Based Scientific Analysis

Cameron J. Ryan
Advisor: Richard Kidder
McQuaid Jesuit High School
`cjryanwashere@gmail.com`

Laboratory for Laser Energetics
University of Rochester
May 2023

ABSTRACT

Experimental data from diagnostic reports at the Laboratory for Laser Energetics (LLE) used for statistical analysis is accessed through the LLE web Application Programming Interface (API). The analysis software development pipeline is inhibited by the lack of any existing software for loading, formatting, and abstracting data from the web API. New software needs to be created redundantly to load data from the API every time new analysis software is made, or whenever external scientists make software that analyzes data observed from experiments at LLE. An application management system using software containers was researched and developed to allow new or external scientists to securely develop analysis software with LLE's physical computational resources, as well as a code-sharing repository for the distribution of diagnostic-reading software through GitLab. This can all be accessed through an easy-to-use web interface.

1 Introduction

Scientists at the Laboratory for Laser Energetics (LLE) and other nuclear fusion laboratories continually collaborate in the creation of software that analyzes data gathered from experiments conducted at LLE. Many of these software development projects are not open-source, and so their source code cannot be made publicly available through an online code repository service, such as GitHub. This is problematic, because scientists often need to use software that has already been created, yet have no clear method of discovering what they need. Additionally, scientists working at laboratories outside of LLE that have been granted permission to use data collected from experiments at the laboratory do not have a clear way to discover or access software that has been created to read and abstract data from diagnostics. This research aims to provide a method for internal and external scientists to collaborate in the development of software [1].

2 Challenges posed by distributed software collaboration

2.1 General software development

Most software development at LLE is version-controlled and shared using GitLab, a locally hosted Git repository service [2]. GitLab, as it has been previously used, is a competent tool for allowing scientists to collaborate together on a single project; however, scientists working on different projects have no way of sharing code that performs the same purpose. Because of this, scientists often redundantly create software for some particular task, such as reading data from LLE's shot diagnostic database. For example, if a scientist wants to create software that performs analysis on data gathered for a particular experiment, they will need to write code from scratch that will load data from the database Application Programming Interface (API) and abstract the data to provide the analysis that they need. Considering that what they create will likely have already been created, this is redundant and time wasteful.

2.2 External Users

An additional problem to be considered is scientists from external laboratories who are performing experiments in collaboration with scientists at LLE and have access to the LLE database API, through an LLE-specific API access token. External users cannot use GitLab, because it creates various potential security issues, and a potential challenge due to a need to manage the authentication of all their GitLab accounts. As external users are likely to find the most difficulty in figuring out how to write code for tasks such as reading data from the database API, code sharing platforms should be inclusive of them by not directly accessing the GitLab Representational State Transfer (REST) API [3].

3 Proposed solution

The purpose of this work is to demonstrate a web platform for scientists to discover software from Git repositories that performs a desired function, so that they do not have to do it themselves. Ideally, it should accomplish the following items:

- Both internal and external scientists should be able to use the web platform to browse Git projects that have already been created, and whose creators consent to having their code shared internally.
- The web platform should allow for scientists to edit code, and utilize the functionality of Git. With Git, scientists should be able to branch and merge code for various projects.

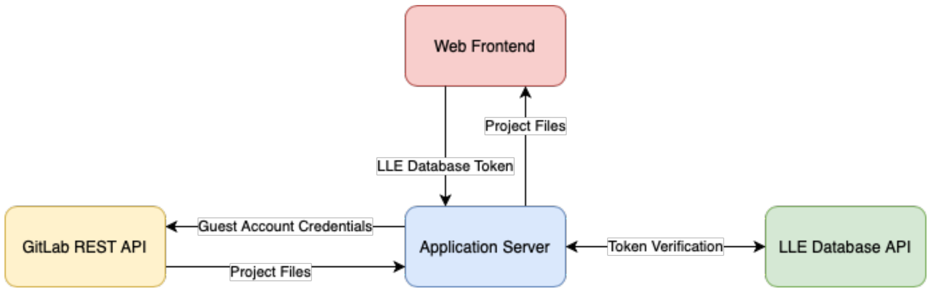


Figure 1: With indirect authentication, external users cannot directly access the GitLab REST API. However, they are able to authenticate through this project's server by sending a request with their database authentication token, which will be verified by the LLE database API, allowing them to access the GitLab projects as a guest user.

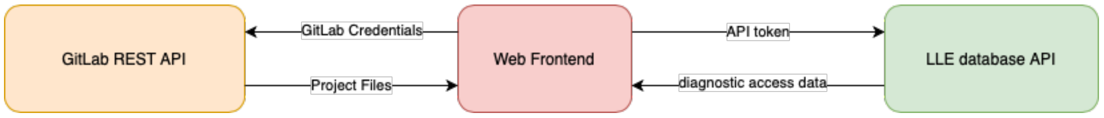


Figure 2: With direct authentication, users directly access GitLab on the web front end through the GitLab REST API. With this method, users are essentially using GitLab, except with additional features for exploring projects.

- Scientists should be able to run code that they have found or edited remotely on a server at LLE. When they run software, their computational resources should be regulated, and they should be able to save their work remotely. This can be accomplished through the use of a virtualization software.

4 Developing with GitLab

GitLab allows for developers to access many of the functions of Git through its REST API. This allows the main functionality of Git to be accessed from a web platform. Users can use their GitLab authentication credentials on the web platform, allowing them to open projects, browse files, and create new branches with their own accounts [3]. The two forms of authentication are described in the following subsections.

4.1 Indirect authentication

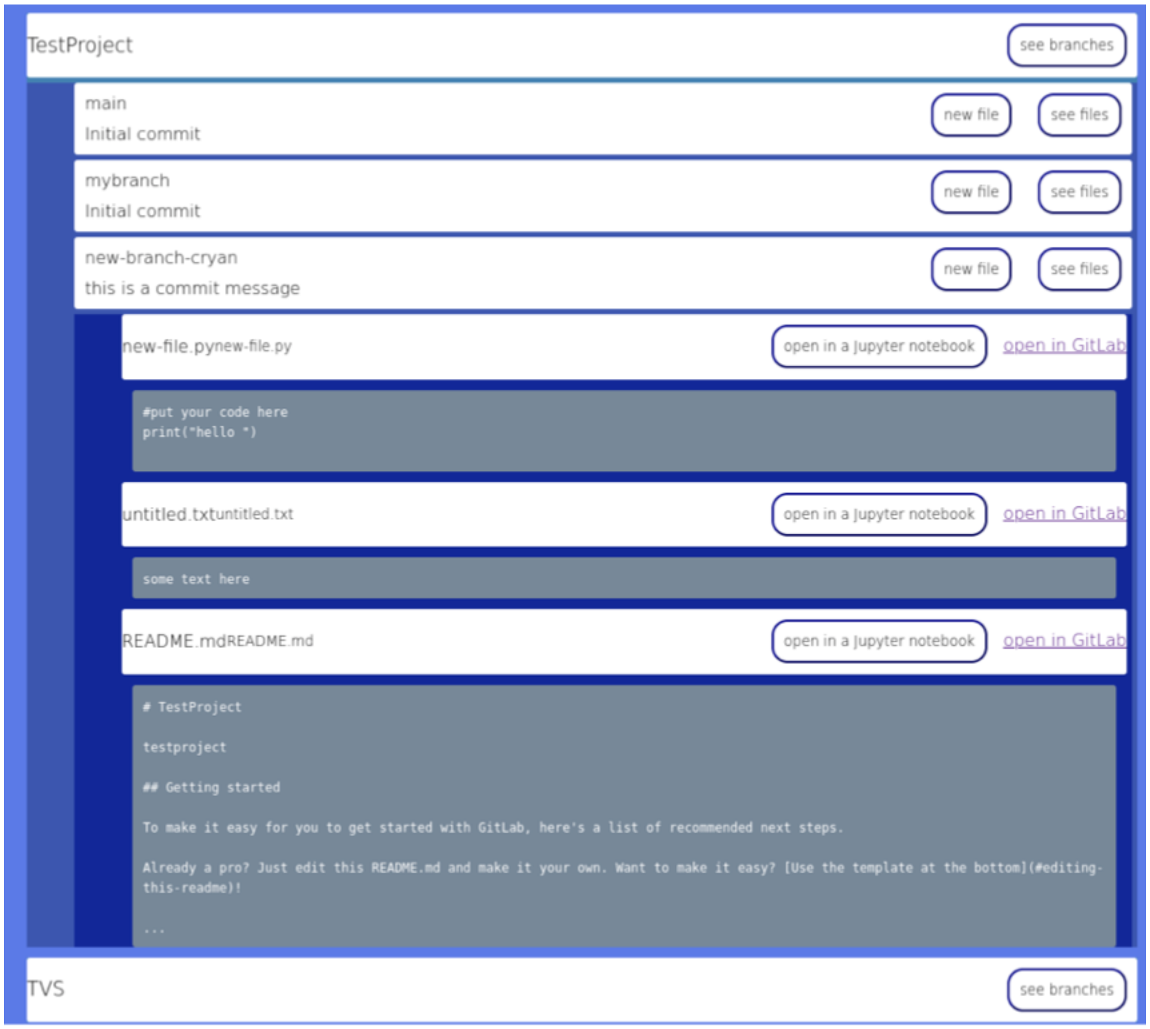
As mentioned, LLE is unable to provide GitLab accounts to all users from external laboratories. However, users can access data gathered from experiments through the database API. To allow them to access GitLab projects that contain code for reading data from diagnostics that they have access to, they can authenticate indirectly using their database access token.

When an external user is given access to data taken from certain shots, and certain diagnostics in the database, they can login to the web platform with their database access token. The server (for the web platform) will request which diagnostics the user has access to from the database API, find all GitLab projects that read data from that specific diagnostic, and return the source code for those projects to the user in the web front end. This is demonstrated in Figure 1.

Since the GitLab REST API must be accessed by a specific GitLab account, the creation of a guest user account allows for the server to access GitLab for the user who is authenticating with their database API token. Note that while this is designed for external users, it can also work for anyone with access to the database, thus internal users as well.

4.2 Direct authentication

If an internal user (LLE employee) wishes to browse GitLab projects with loading and analysis code for any particular diagnostic that they have access to, they may do so by directly accessing the GitLab REST API with their authentication credentials. If they provide their database API token, the web frontend will automatically filter what projects they have access to (based on what diagnostics they may access), and will return to them the source code for those projects. This is demonstrated in Figure 2.



```
#put your code here
print("hello ")
```

```
some text here
```

```
# TestProject

testproject

## Getting started

To make it easy for you to get started with GitLab, here's a list of recommended next steps.

Already a pro? Just edit this README.md and make it your own. Want to make it easy? [Use the template at the bottom](#editing-
this-readme)!

...
```

Figure 3: Screenshot taken from the web front end. Users can search Git projects by diagnostic, browse the file tree of the Git project, and read files in the project

5 Web interface

To facilitate the discovery of Git projects, a web front end and server were created to indirectly browse Git projects. They could also be used to safely and securely access the computational resources of the server.

Additionally, since it would be unsafe to allow users to directly run their analysis code on the operating system of the server, containers were chosen to safely compartmentalize applications run by users. The reasoning behind this decision is further elaborated in 6.

5.1 Front end

The front end was created using JavaScript, HTML, and CSS. As mentioned, it was created with functionality to access GitLab through the GitLab API (or the server). The interface is organized on the basis of what diagnostic its code analyzes (each diagnostic corresponds to a set of projects that analyze data from that diagnostic). When the user has found the diagnostic, for which they need code to load data from or analyze, they are able to browse projects and their file trees. A screenshot of the front end's web interface can be seen in Figure 3.

Early versions of the front end allowed for the user to edit files directly using an embedded code editor. Once finished with editing files in the project, they could save their edits through the GitLab API if they have access to the project, and push their changes to a new branch if they do not have access to the project. New branches will be accessible by all

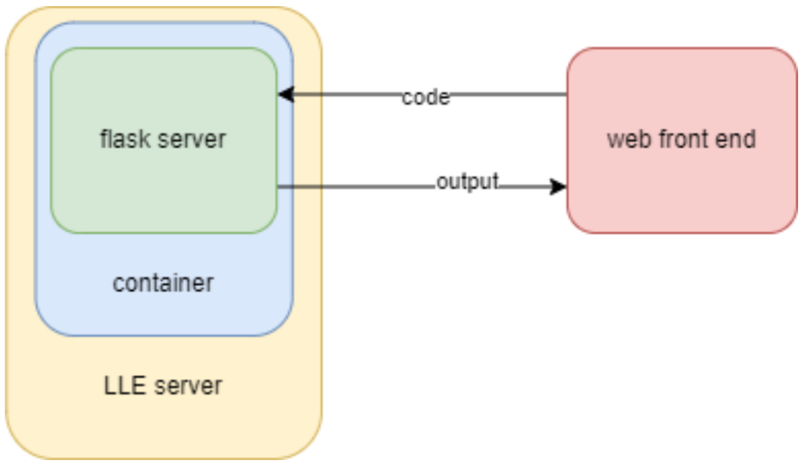


Figure 4: A demonstration of the web back end with a Flask server

users browsing a certain diagnostic, and will be owned by the user's GitLab account. If the user is an external user who is indirectly authenticating, then the branch will simply be owned by the guest user. Developing a better method for external users to save and have control over project branches they have created may be the subject of future work.

Furthermore, users would be able to directly run code on the server from the web interface as they edited it. This functionality is explained in 5.2.

5.2 Back end

The back end experienced various iterations over the course of this work. Initially the back end was a NodeJS server [4], and accessed the GitLab REST API by executing a series of python scripts that use the GitLab python library [5]. Furthermore, the back end authenticates users with their LLE database access token, so that they may access GitLab projects they have access to with the guest GitLab account.

When users would want to run code that they have found or edited, the back end would spawn a container, which runs a Python Flask server [6] in the container upon its boot process, as demonstrated in Figure 4. The python flask server would function in a way similar to a remote shell, allowing the remote user to run software in their container. The web server additionally had various built-in buttons that could run pre-made shell commands for running python [7] and Julia [8] scripts, allowing for the user to develop and run code with a minimal understanding of Linux [9] [10] shell commands.

5.3 Jupyter

While the custom, Flask-server-based back end for remote computation functioned properly, it was problematically prone to errors and insecurities due to the amount of new production-level software that needed to be created from scratch. This prompted research into a new solution for remotely editing code, and accessing computational resources in a container.

Jupyter provides a powerful text editor as well as built-in functionality for python notebooks (`.ipynb` files), which have fairly recently become an important tool for data science [11] [12]. With Jupyter, it is easy for users to remotely access containers by adding a command to start a Jupyter server in the container's `Dockerfile` [13]. Using this method, when a scientist wants to edit code, they can start a Jupyter container and then clone the GitLab repository of the project that they would like to edit.

6 Software compartmentalization

6.1 Remote scientific computation

The creation of the web-based diagnostic code sharing platform prompted an additional opportunity for streamlining the efficiency of internal software development at LLE. Given that scientists are able to find software for reading data from

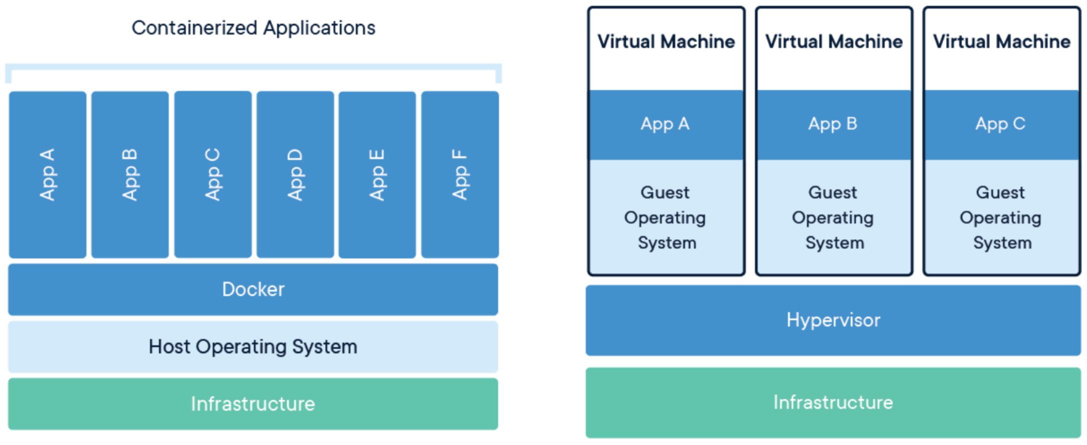


Figure 5: An intuitive visualization of containerization and virtual machines. Image sourced from [14].

diagnostics on the web platform, it is also possible that they may wish to use, modify, or further develop the software for their own personal needs, and share what they create with other scientists. This work aims to create an application management system that allows scientists to run or develop code securely and remotely.

If a scientist working on a particular project wants to use software created for another project that is stored on GitLab, such as loading diagnostic data and performing a particular analysis on it, they will need to clone the Git repository and run it locally (in this case, 'locally' means on their personal machine). They will need to install all of the dependencies of the project that they are using, and handle the issues created by conflicting software configurations on their local machine. Furthermore, if the project includes the usage of some intense scientific computation, their machine's hardware may be inadequate for its task. After all of this has been handled and the scientist has extracted the analysis data that they need, two more problems arise. Firstly, the installation of the project and its dependencies could potentially conflict with other software on the scientist's computer, and therefore have a detrimental effect on the software's ability to function properly in the future. Secondly, if the scientist wishes to return to their analysis project in the future, and use it the same way they had previously, dependencies from the project will likely have been updated, potentially causing compatibility issues, and preventing the analysis code from continuing to work properly. This is clearly a problematic system of using scientific analysis software, and this work aims to demonstrate how it can be improved through the usage of remote computing [1].

6.2 Compartmentalizing Software

Compartmentalization of software allows any given application to be run in a safe and controlled manner. One goal of this research was to find a compartmentalization tool that can be used to control the computational resources allotted to a particular application, and to easily manage the dependencies that a particular software needs so that it can be used to clone Git projects and run code from the projects.

6.3 Virtual machines

Virtual machines are operating systems that are run under the supervision of a hypervisor software. The operating systems each have an allocated portion of the physical computer's computational resources and are already commonly used for various applications [14]. This is visualized in Figure 5. While this is an attractive solution due to their powerful capabilities, virtualizing an entire operating system for the purpose of cloning a Git repository and writing code is inefficient and wasteful [1].

6.4 Containers

Containers are applications that virtualize the libraries, file system, and computational resources of an operating system, while running on a host operating system [14]. For this project, several software packages for containerization were researched.

6.4.1 Docker

Docker is an extremely popular container software for Linux. Docker utilizes a daemon, a server for developing and managing containers. Docker allows its user to create a Dockerfile, which contains set of commands that configure a container image, as well as allow for the container to run certain commands when it is spawned [14]. This was used in early versions of the back end, as the server's containers would run a python Flask server when they were spawned, allowing for the remote user to access its computational resources.

While Docker is a powerful tool, it is not supported by Red Hat Enterprise Linux (RHEL) [15], the operating system used for the server. Additionally, Docker requires root access in its operating system to develop and manage containers, which may pose a potential security risk. Because of these reasons, the decision was made to not use Docker, and look for another viable option for containerization [1].

6.4.2 Podman

Podman is a containerization software that has been created as a substitute for Docker, specifically for RHEL users. Podman does not use a central daemon server to manage and develop containers, and it does not require root privileges to be used. Furthermore, Podman's functionality is accessible by many of the same commands as Docker, and can use the same Dockerfiles that Docker uses to create container images [16].

6.5 Container and application management

While containers provide an important tool for software compartmentalization, allowing this project to function, a need was discovered for a way to manage containers, rather than simply spawning a container every time a scientist branches a Git [17] project from the web interface. It is desired that an application management system would solve the following items.

- Allow for a server administrator to manually control how many computational resources are allocated to each container. Different projects may take different levels of priority, or simply need more computational power than will be allocated to each container.
- Kill containers after they are no longer used, or when they have been left idle. There will likely arise situations when scientists accidentally leave containers on when they are no longer being used.

While Kubernetes [18] was examined to provide a solution to this, it was found that most of this functionality could be achieved through Docker and Podman alone. Additionally, for Jupyter notebooks, a software called DockerSpawner [19] was discovered and used for spawning Jupyter Notebook containers when scientists need to create a new container.

6.6 Saving containers

After a scientist has performed some sort of scientific analysis in their container, they may wish to save the file system of their container. By saving the file system of their container, they can return to their analysis code in the future, if they find themselves working on a project that requires more of the same analysis. Additionally, their container may be where they manage the Git branching and merging if their code is a contribution to a collaborative project.

A solution to this problem was not completely finished. Scientists can use Git to push their changes as a new Git repository that they own, and clone the repository when they want to use their work again. A solution for scientists to save container file systems will be explored in future work.

7 Conclusion

7.1 What has been accomplished

The purpose of this project is to demonstrate how a method of remote software collaboration can function for non open-source projects at LLE. This aims to solve many of the existing inefficiencies that exist, as well as allowing scientists from external research organizations to safely use software that has been created at LLE to analyze data that they have access to.

This project created a web interface, an indirect authentication server, and a containerized application management system to allow for users to access software they need, and run minor computational experiments in a safe and remote fashion. This proves to be advantageous, as the compartmentalized nature of containers allows for software dependencies to be easily managed remotely, thus improving the efficiency for the user.

7.2 Future Work

While this system has been demonstrated to work end-to-end, work on this project is not complete. The project has yet to be used for actual science in a practical application. The code for the server and application management must be refined to prevent insecurities that are inherently present in such a large untested project.

Future work on this project will attempt to deploy the server, and receive feedback on the server from scientists that use it. Optimistically, the web platform will become a useful tool for streamlining the efficiency of computational analysis experiments. This will allow for scientists to spend less time dealing with dependency and software compatibility issues on their local machine, so that they can focus on science.

Acknowledgments

Great thanks to the Laboratory for Laser Energetics High School Summer Research Program for allowing this research to take place. Furthermore, many thanks to Richard Kidder and Dr. R. S. Craxton for guidence during the course of this research.

References

[1] Richard Kidder. private communications.

[2] Gitlab. `https://about.gitlab.com`. Accessed on May 11, 2023.

[3] GitLab. Rest api. `https://docs.gitlab.com/ee/api/rest/`. Accessed on May 10, 2023.

[4] NodeJS. Documentation. `https://nodejs.org/en/docs`. Accessed on May 11, 2023.

[5] `python-gitlab v3.14.0`. `https://python-gitlab.readthedocs.io/en/stable/`. Accessed on May 11, 2023.

[6] Welcome to flask - flask documentation. `https://flask.palletsprojects.com/en/2.3.x/`. Accessed on May 11, 2023.

[7] Welcome to `python.org`. `https://www.python.org`. Accessed on May 11, 2023.

[8] The Julia Programming Language. `https://julialang.org`. Accessed on May 11, 2023.

[9] Wikipedia. Linux. `https://en.wikipedia.org/wiki/Linux`. Accessed on May 11, 2023.

[10] Linux. `https://www.linux.org`. Accessed on May 11, 2023.

[11] Jupyter. Project jupyter. `https://jupyter.org`. Accessed on May 11, 2023.

[12] Jupyter project documentation. `https://docs.jupyter.org/en/latest/`. Accessed on May 11, 2023.

[13] Docker. Minimal jupyter notebook. `https://hub.docker.com/r/jupyter/minimal-notebook/`. Accessed on May 11, 2023.

[14] Docker. Overview what is a container? `https://docs.docker.com/get-started/`. Accessd on May 10, 2023.

[15] Red Hat Enterprise Linux. `https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux`. Accessed on May 11, 2023.

[16] What is podman? `https://docs.podman.io/en/latest/`. Accessed on May 11, 2023.

[17] Git. `https://git-scm.com`. Accessed on May 11, 2023.

[18] Kubernetes. `https://kubernetes.io`. Accessed on May 11, 2023.

[19] Dockerspawner. `https://jupyterhub-dockerspawner.readthedocs.io/en/latest/`. Accessed on May 11, 2023.