Lancium Compute: Green-Powered Cloud

A Technical Report submitted to the department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Samuel McBroom

April 20, 2021

Technical Project Team Members

Courtney Jacobs

On my honor as a University student, I have neither given nor received aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

signed: _____ date: April 20, 2021
Samuel McBroom

signed: _____ date: April 20, 2021
Dr. Andrew Grimshaw, Department of Computer Science

Lancium Compute: Green-Powered Cloud

Sam McBroom
sammcb@virginia.edu

Courtney Jacobs
cj5he@virginia.edu

Figure 1: Lancium logo

Abstract

Lancium offers affordable access to a cloud computing grid powered by renewable energy and designed specifically for High Throughput Computing jobs. While Lancium offers comprehensive tools for interfacing with their grid, such as a command line interface (CLI) and programming language APIs, these require a high degree of technical knowledge and the installation of external software to provide access. Web applications provide the solution to this, as they provide a user-friendly interface and can be accessed from any device with an internet connection. Lancium offered a web application, though it lacked many of the features the other tools provided. Due to its design, the web app could not be easily extended to achieve feature parity with the application programming interface (API). The new web interface enhances the web experience for users, reduces technical maintenance, and adds new functionality by leveraging the power of the existing API.

1 Introduction

The rise of cloud computing services such as Amazon Web Services and Google Cloud has allowed users to access powerful computing resources without the cost of acquiring and maintaining local servers. Specifically, these cloud providers build massive data centers and rent processing power to businesses and individuals. Data centers are currently estimated to use about 1% of the world's electricity [1]. Lancium is an affordable, green cloud provider that sources power from renewables, avoids the use of refrigerants, and reduces e-waste. Thus, our goal was to try and increase Lancium's user base to build awareness of and entice users towards clean computing options. To this end, we aimed to redesign the web interface to make it more user-friendly easier to support and maintain by connecting to Lancium's existing API.

Our development team consisted of Samuel McBroom and Courtney Jacobs. This system is our Fourth Year Engineering Research Capstone project. Our new web app was built using Vue 3, JavaScript, and Ruby. Lancium plans to switch to our new web interface sometime in the future after complete feature parity is achieved with the API.

2 Existing Access Options

In building this web app, we analyzed Lancium's existing grid access technologies to build an understanding of the features the new web app would need to support. We then determined the features and limitations of the old web interface to design an initial architecture for our new web interface design.

2.1 Command Line Tool & Language API

Lancium offers a command line tool, which is an executable that can be installed and includes commands that can be run to perform different actions on the grid. In addition, Lancium also provides an API that can be accessed from programs written in a variety of languages. These tools give complete coverage of the API functionality, but require more work on the users' end. To effectively connect to Lancium's grid, users need to manage authentication. This process involves generating a token via the web interface, securely storing that token, and then passing the key into every command executed. A web interface, by nature of its design, is able to handle all the authentication flows for the user after they log in. The command line interface does offer the ability to create automation scripts to manage compute jobs but requires a significant amount of technical knowledge to implement. We recognized that technical users would likely prefer to use the command line tool if possible. Thus, our goal for the website

was to provide convenience functionality aimed at a user running one-off commands and providing a clear interface for checking the status of submitted jobs.

2.2 Web Interface

At the time we joined the project, Lancium already provided a web interface though it had significant drawbacks compared to the CLI. Firstly, it only allowed users to submit, check the status of, and delete jobs. Secondly, the web interface provided its own implementations for API functionality which increased developer maintenance, as two code bases needed to be maintained. Thirdly, due to its duplicate code base, new code needed to be written for the web interface to match new API functionality, leading to the web interface lagging significantly behind the API. However, the web interface offered many conveniences. A price calculator was shown on the create page, and while it does not currently function because Lancium Compute is still in their beta testing period, it will be especially convenient for users to see a live price estimate as they select the configuration for their job. In addition, the web interface could provide live updates on job statuses without the need to refresh the page.

3 System Design

Lancium has a robust API covering three primary areas: managing compute jobs, managing images, and managing data. For our capstone, our scope mainly focused on covering the majority of the job API in our new web app, as configuring and running jobs are the main actions performed by Lancium users.

3.1 Single Page Application

When it was first developed, the primary function of the internet was mainly to act as a cross-computer file browser. Today, one of the most noticeable remaining artifacts of this design are page reload to retrieve new information. These can disrupt the flow of a web application and can be frustrating to the user. Many modern web frameworks now support the ability to create single page applications (SPAs). Single page applications are entirely loaded onto a users computer when they navigate to the website URL and when the move to new pages, the website performs an Ajax request that updates the page without the user creating a new page network request. The result is a web app that feels like a native desktop application and allows for convenience features like live page updates without the need to repeatedly refresh the page. We chose to use Vue to build the new web app, as this framework offers excellent SPA support and reduced the amount of boilerplate code compared to similar JavaScript-based web frameworks, such as React.

3.2 Efficient Networking

When building a scalable web app, a primary concern is limiting unnecessary requests to the underlying API. This allows the web app to support more concurrent users while reducing server load. To achieve this, we used a state management library called Vuex, which caches information on the users local computer to populate page information. We implemented a background task that automatically refreshed job information every five minutes to make sure the user never had to worry about viewing outdated data. In addition, users could reload the page manually to force an update. This system effectively balanced limiting the number of extraneous requests to the server while also providing up-to-date data to the user.

3.3 Drafts

A unique feature we decided to implement on the web interface was the ability to save job submission drafts. Similar to email drafts, this allowed a user to begin creating a job, and then to save their current configuration to submit later for any reason. This system is a unique benefit of the web application system as a job created with the CLI cannot be submitted without fully specifying its configuration options. Eventually, it is planned to expand this to a template system so web users can quickly create multiple jobs with the same configuration which CLI users accomplish with scripts.

3.4 File Uploads

By far the most technically difficult portion of the web app development was enabling non-blocking file uploads for jobs. Traditionally, due to JavaScript limitations, when uploading a file to the old website it would block the user from interacting with the website until the file finished uploading or the user cancelled the upload. Recently, JavaScript has added support for worker scripts which we leveraged to provide file uploading without blocking the user from accessing the rest of our web app. Implementing the file uploading came with other difficulties, as many situations had to be considered: if the user closed the site while they were uploading a file, if the user lost internet connection while uploading a file, and any other possible reason that might cause the file upload to partially or completely fail. This is especially significant for Lancium as their typical user might be uploading files several tens of gigabytes in size. We created a safe solution by notifying the user if their job failed to upload, alerting the user if they tried to close the page while

their file was uploading, and allowing the user to resume uploading disrupted file uploads.

3.5 Status Feedback

Finally, we built a page to list all the users jobs, which would update every five minutes showing the status of the job as it moved through the grid queue. Another important aspect of status feedback was alerting the user if a job failed or successfully submitted or if a file upload failed. To supply notifications to every page of the application, we used a toast library (a toast is a popup notification), and added a custom section to our datastore to track where to the display the notification.

4 Procedure

The new web app was designed to be both powerful and intuitive. Here we will discuss the general flow a user might take though the application and how our design enables and enhances this flow.

4.1 Authentication

A major downside to using the CLI is the need to manually and securely manage authentication tokens. Our web app design automatically fetches a token for the user after they log in to their account. In addition, to reduce any possible friction, our system automatically refreshes an expired token if the user is still logged in when any request is made to the server. This abstracts any authentication handling away from the user and creates a simple login experience.

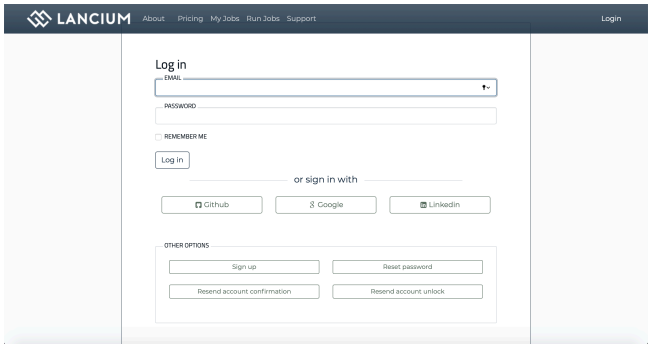

Figure 2: Login page

4.2 Create Job

The first thing a user might do when logging into the site is to create a job to run on the grid. They can supply a number of configuration options, including a job name, preconfigured operating system, specs, input files, and output files. Our web app improves on the old site by allowing the user to either save the job and submit it later or submit the job immediately. When a job is saved the user is automatically taken to the generated detail view for their job with the option to further edit, submit, or delete their job. If the job is submitted, the user is taken to the jobs overview page to view the status of their job. They can then cancel the job or wait for completion and view the results in the provided output files on the details page. These options significantly improve upon the old site, which only allowed users to submit, view the status of, and cancel jobs.

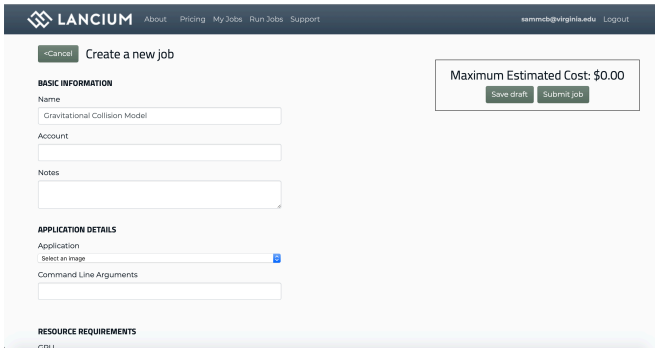

Figure 3: Create job page

4.3 Upload Files

Before submitting a job, a user might choose to upload input files. These are usually massive data files, on the order of tens of gigabytes in size. As discussed above, using JavaScript workers allows file uploading to take place in the background and lets the user continue to navigate to other pages, submit more jobs, and check the details of other jobs while their files upload. This is significantly more user-friendly than the old site, where the user would have to wait until the files finished uploading to interact with any other areas of the site. In addition, we designed the file upload system so users could get immediate feedback on the progress and status of their upload and resume an upload from a partially uploaded state if needed.

4.4 Submit Job

Submitting a job was another area we focused heavily on, because providing immediate feedback to the user if a job configuration was complete is critical in allowing the user to quickly debug issues with their submission. We chose to duplicate some of the API validation checks on the client-side, and while this results in minor amounts of duplicate code, it drastically reduces the number of network requests to the server. This both allows instant user feedback as no network requests are needed and potentially allows many

users to be interfacing with the web app simultaneously without overburdening the server.

4.5 Jobs Overview

When a user first enters the web app they are taken to the job overview page. This page displays a list of all the user's jobs and allows them to quickly view the details of or delete any compute job. The original job overview page (seen in Figure 4) displayed three lists for each stage of the job submission. This could make it difficult to find a job as it could be in any of the three sections. The old page also had basic search functionality where a user could enter a search term and would filter all lists to only display jobs with names matching or containing the term.

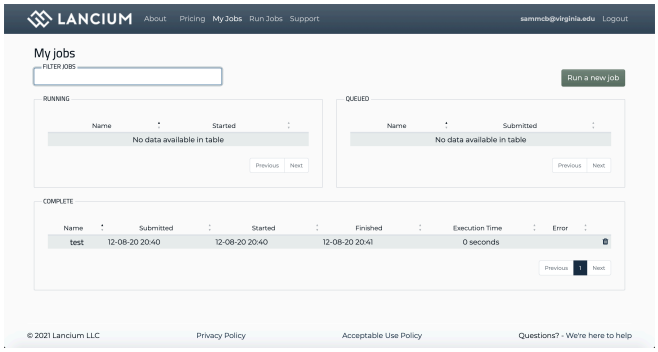

Figure 4: Old jobs overview page

When designing the new web app, we chose to condense the jobs into one list and include a "status" column. This is more consistent with traditional list pages. To allow users to quickly find jobs by name and status, we reimplemented the basic search box but also created a new status filter, whereby users could search for all jobs with a certain status. In addition, we removed pagination, as this just resulted in more clicks for the user when searching through their jobs. We also added sort functionality, so users could choose in what order jobs are displayed, with the most recently created jobs appearing at the top of the list by default. These changes, combined with automatic background refreshing and clearer styling makes this page much more convenient for both CLI users quickly checking job statues and web interface users.

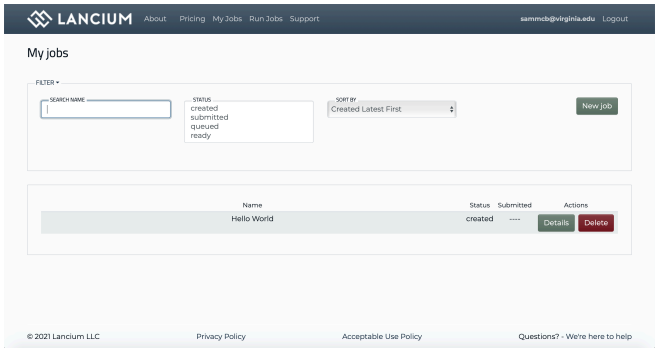

Figure 5: New jobs overview page

4.6 Delete Job

Finally, the end of a typical workflow might consist of a user deleting their job. This might be a job draft or a submitted job. When the user deletes a job draft, the results are immediate. The user is taken to the overview page if they were on another page, the job is immediately removed from their list, and request is sent to the server to delete the draft. If the user cancels a submitted job, it might take time for the grid to perform the cleanup. By interfacing directly with the API to retrieve job statues, our job overview page automatically marked the job as cancelled and removed it from the list when the server performed the cleanup.

5 Results

At the end of the semester, we presented our redesigned web application to our project supervisor, Cushing Whitney, and our capstone professor, Andrew Grimshaw. They were both pleased with the results, and approved the level of feature parity we achieved with the API. We were able to comprehensively cover nearly all of the job and file upload APIs, including proper handling of both success and all possible error resolutions.

Most importantly however, we laid the groundwork for a fully feature-rich web interface. With the removal of the duplicate logic, we not only made the code base easier to maintain but also provided most of the logic necessary to make future API support significantly simpler. We hope that a more intuitive and powerful web interface will help Lancium attract more customers and introduce them to clean and sustainable solutions Lancium's Clean Compute Centers offer.

6 References

(1) Masanet, E., Shehabi, A., Lei, N., Smith, S., & Koomey, J. (2020, February 28). Recalibrating global data center energy-use estimates. *Science, 367* (6481), 984–986.

(2) Lancium (2021). Lancium Compute Documentation - Concepts and Usage. https://lancium.github.io/compute-api-docs/lancium_cli.html#lancium-compute-api