# Science as a Service

Sterling A. Baldwin, Matthew B. Harris

***Abstract*—As the threat of global warming comes out of the land of speculation and into the real world, the demand for accurate, repeatable climate science is growing at a tremendous pace. To help the ever-increasing number of researchers with their ballooning datasets, climate science tools need to improve in tandem. The current paradigm of passing terabytes of raw data over the network repeatedly and the wide variety of semi-compatible tools needs to be replaced. We are doing so with a single web application, able to handle the diverse needs and growing demands of the climate science world, following the same continuous delivery deployment model as SaaS businesses. Lawrence Livermore National Laboratory and the ACME [1] project are uniquely suited to fill this void in the scientist's tool kit; we are at the crux of data providers' networks, HPC infrastructure, researchers, and the developers building the tools. This paper will discuss the challenges faced by this effort and the prototype, which is currently in development.**

***Index Terms*— SaaS, climate science, web programming, data science, high performance computing**

## I. Introduction

Research into the climate is growing at an ever-increasing rate, but the toolkit available to scientists has remained relatively static. Each tool has marched forward in features and version numbers, but the world of climate science is ready for a shift into the world of rich web applications. The ACME Web Dashboard project's aim is to take the many multifaceted tools distributed through the scientists workflow and tie them together into a single web interface. The Dashboard will allow researchers to divorce themselves from the fine-grained details of their workflows, and automate away all of the many laborious tasks that keep them away from their actual work: doing science.

From a tools programmer's perspective, the four primary facets of a climate scientist's work are as follows: access to data and models, running simulations on that data using models, visualizing the results of the simulations, and lastly publishing their results for others to examine and use. Currently, all four of these domains are handled by different applications. All of those applications force the users to focus on learning the intricacies of individual command line tools, and deal with the various incompatibilities of those tools, both minor and major.

Manuscript received June 02, 2015; revised July XX, 20XX. This work was supported in part by the U.S. Department of Energy under grant (ESGF/ACME/AIMS funding information)

S. A. Baldwin is a Computer Science and Web Programming intern in the AIMS project at Lawrence Livermore National Laboratory, Livermore, CA 94550 USA (phone: 925-423-8954 email: baldwin32@llnl.gov)

M. B. Harris is a Computer Scientist, Mathematical Programmer in the AIMS project at Lawrence Livermore National Laboratory, Livermore CA, 94550 USA (phone: 925-423-8978 email: harris112@llnl.gov)

To retrieve data, users must either first install the Python based Earth System Grid Federation (ESGF) [2] client and its dependencies, or manually crawl through the web pages of specific data nodes, use their Open ID credentials to authorize the data, and then wait as terabytes of data are transmitted over the wire and stored on their local machine. They can run some initial analysis locally, but for any serious computation, they have to get set up with whatever HPC platforms are available to them, stage the model run, and wait until it actually wraps up. Once the simulations are completed, they bring the output to their local machines and can begin analyzing the results. To do so, they install the Ultrascale Visualization Climate Data Analysis Tool framework (UV-CDAT) [3] and its whole suite of dependencies, allowing them to visualize and interact with the data. If they are on a supported platform (modern flavors of Linux or OSX), this should go smoothly. However, if they have to build from source, this process can take several hours.

This process in its entirety is extremely complex and forces climate researchers to spend an inordinate amount of time battling with their computers instead of doing what they do best—the actual science. Our aim with this project is to create a single web interface to relieve the pain-points for scientists while also reducing the computational and bandwidth requirements for performing their jobs.

## II. Prototype Design

### A. Frontend User Interface

Although climate science is complex, the user experience doesn't need to be. One of the problems faced by the climate science community is reproducibility of results, partially because every institution has a different method for setting up and configuring their work environment. By reducing the barrier to entry and unifying the interface and toolset, we can dramatically reduce the reproducibility problem, and allow more scientists access to cutting edge tools.

The look of the user interface will go through many revisions before we are ready to publish a finished product, but the fundamentals will remain the same. The interface is built around a tiling window manager written from scratch in javascript, with each window encapsulating a service. During the early phase of the project, several open source windowing libraries were evaluated. After much deliberation, each candidate was found wanting. The up-shot of this process was that we were able to find what we liked and disliked about all the

alternatives and incorporate those decisions into our own interface. For example, we experimented with the jsPanel [6] library, which makes attractive panels with the ability to move, resize, and easily populate with content, but it was very difficult to get the panels to interact with each other. We took the lessons learned and went back to the drawing board.

Our windowing system allows the user to create many sub-windows inside the browser window proper, with the ability to move and resize the windows to whatever degree they like. Each window is bound to a service, so that all functionality based on that service lives inside the window. This allows for a simple design, while also allowing each window to easily query the status of the other services and behave accordingly. This functionality was critical to us, allowing us to easily create new windows types as we need them, and create dynamic interaction between the service windows while maintaining the autonomy of each window. For example, the visualization service needs to be able to query the data node service to find which node the user has selected, but the user should not be able to close the data window after selecting a node and still be able to use the visualizer. Because the project is still in its early stages, it's important that we use an architecture that allows for extensibility as the project grows.

Although the frontend can be seen as a wrapper for the underlying services, it needs to expose those services functionality in a way that the user understands intuitively without the need for an excessive learning curve. To this end, we strive to make the interface behave as much as possible like a native desktop application, so that all the assumptions they have about usability match with the dashboards functionality. Much of the CDAT user base has been doing climate work for years; therefore it's important to us that they are able to jump into using our product without the tedium and frustration of figuring out how a new environment works. In the best-case scenario, they should understand how the interface functions simply by looking at it.

A major design goal for the frontend was to make it simple and intuitive, while simultaneously giving the user as many options and as much customization as possible. We cannot assume that scientists will be using the tool for exactly the same thing, or have the same preferences on how they would like their environment to look.

Additionally, the site needs to function on all modern browsers and in all operating systems, allowing users to move between computers and facilities without breaking any of the features. To this end, we have implemented testing using the open source Robot framework [7] developed by Google. This allows us to quickly write tests to ensure that as we add new features we don't break older features, and that all features function across all target browsers.

### *B. Backend Servers and Service Interface*

Simultaneous to the frontend effort, the backend has been in development to manage the users interaction with the service layer. The backend is broken into two distinct parts: a) the service layer, and b) a python based Django [5] web server to manage these services and direct data between the user and the 3rd parties. Although the user would never know these three hidden layers exist, by breaking them apart we reduce overall complexity, allowing each distinct part to exist independently and speed up development by creating each service concurrently.

The service layer consists of three main components: the interaction with the ESGF high capacity data store, the visualization server (CDATWeb), and the secure high performance compute node (Velo [4]) with its on-site data store. One of the main challenges of the project so far has been developing the service layer for each of these while their APIs are still in active development. Of these three tools, only the ESGF has a stable development ready API, while CDATWeb and Velo are themselves only in the beginning of their development life cycle.

The volatility of the APIs the project depends on has repeatedly shown that going with a Django backend was the correct choice. Besides all the benefits of using a framework written in python, using a Model View Controller (MVC) architecture makes adapting to changing dependencies much less painful. By adopting the MVC framework, we can build the database model, and the users view interaction independently of the more volatile controller, allowing us to make changes to how the web server interfaces with the Velo and CDATWeb APIs while not worrying that it will break things elsewhere in the project.

The Django server's main purpose is to work as a hub interfacing with the 3rd parties—but it does so much more. In addition, it allows us to cache responses as well as minimizing data duplication, while maximizing code reusability. It works as both a request dispatcher to the service layer as well as a short-term data store, reducing the amount of times that large batches of data need to be transmitted through the network. By physically locating the Django server along side the visualization and Velo servers, we are able to cut down on network lag as well as bandwidth overhead. One of the compelling reasons for this whole project has been the issue of repeatedly moving very large datasets. By placing the service hub in the same facility as the data warehouse and HPC compute nodes, we can cut down hugely on both the scientists time wasted waiting to download these terabyte sized files as well as the amount of duplication involved.

Instead of requiring each user to have a copy of the entire dataset, the Django server can get a copy from the ESGF node, visualize it, and only send the results outside of the data center. This cuts down hugely on the amount of time and space needed by reducing the time and space requirements from scaling linearly—at least one copy per user stored and at least one copy transmitted—to a constant of one copy in the ESGF nodes and when

required, one copy maximum stored on the Django server. This creates additional benefits if two users are accessing the same data, as the Django server doesn't even need to do the step of retrieving the data from ESGF but instead can serve many users from the same dataset simultaneously. Effectively, this changes the process from a linear time/space complexity O(n), to a constant complexity O(1).

## III. Challenges of Implementation

As the ACME dashboard project enters full swing, we are continuing to discover unforeseen challenges. All three of the main services that the climate dashboard interacts with are moving targets, quickly changing as they go through their own development cycle. Fortunately, two out of three of them (ESGF and CDAT) are not only open source, but are being developed by other teams also out of the Lawrence Livermore National Labs, making collaboration much easier. The other service, Velo, has the highest bar for security as it directly interfaces with the HPC infrastructure itself.

Managing the security of credentials for all these services has posed another challenge. Each one of the external APIs have their own method of managing user permissions. For the dashboard prototype, we have been assuming that all of our users would already have credentials and permission to use each of these services, and so although we have a method of storing their credentials so we can access their accounts, we have yet to design or implement a method of creating new users. As we move towards production, working with the other teams to implement this will be a considerable challenge as they all have their own authentication mechanisms, none of which support new user creation through their APIs.

An additional challenge will be scaling the project to work with generic HPC resources. Currently, the project is targeting the Oakridge National Lab and their Titan supercomputer as the first deployment site, and the Velo service is tailored to specifically work with Titan. After we have deployed the prototype and are able to find a candidate for a second site, we may be forced to create another level of abstraction and encapsulate the Velo service in a generic file access and HPC job service wrapper. It is very possible that we will be forced to custom tailor each deployment with their specific HPC configuration.

## IV. Conclusion

Climate science is growing in importance to global political leaders as well as scientists. As interest grows, the tools to do the science needs to grow as well. The current paradigm of each researcher hand crafting their environment needs to be replaced by a fast and more usable framework of Science as a Service.

By creating a single tool encapsulating the data service, the HPC service, and the visualization service, we can greatly increase the effectiveness of individual researchers, as well as the international climate modeling effort. At the same time, we can relieve pain points in the key challenges of reproducibility and the explosion of computation resource demands. The scale of datasets and modeling complexity is ever growing, and so our toolsets must evolve to handler bigger data, and ever increasing demand.

## Acknowledgment

We would like to thank each member of our team for all their hard work and dedication to the project and its goals: Dean Williams, Renata McCoy, Matthew Harris, Jonathan Beezeley, John Harney, Samuel Fries, Bibi Raju, Brian Smith, Sterling Baldwin, Dakotah Webb and every member of the ACME project.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DEAC52-07NA27344.
LLNL-CONF-674391

## References

[1] "New Project Is the ACME of Addressing Climate Change." Lawrence Livermore National Laboratory, 19 Aug. 2014. Web. 24 June 2015.

[2] Pascoe, Stephen. "Esgf-pyclient Documentation." *Welcome to Esgf-pyclient Documentation — ESGF Pyclient 0.1.4b Documentation*. Earth System Grid Federation, 2012. Web. 24 June 2015.

[3] "Ultrascale Visualization." *Home*. Lawrence Livermore National Lab, n.d. Web. 24 June 2015.

[4] "VELO." *Computational Sciences & Mathematics*. Pacific Northwest National Labratory, May 2015. Web. 24 June 2015.

[5] "Django Overview." *Django Overview*. N.p., n.d. Web. 25 June 2015.

[6] "JsPanel a JQuery Plugin to Create Multifunctional Floating Panels."*JsPanel*. N.p., n.d. Web. 25 June 2015.

[7] "ROBOT FRAMEWORK." *Robot Framework*. N.p., n.d. Web. 25 June 2015.