Accelerating Gateway Development with Agave ToGo Webapps and Microsites

Rion Dooley
University of Texas at Austin
Austin, TX 78218
Email: dooley@tacc.utexas.edu

Sean Cleveland
University of Hawaii
Honolulu, HI 98622
Email: seanbc@hawaii.edu

***Abstract*—The Agave Platform is an open, Science-as-a-Service (ScaaS) cloud platform for reproducible science. Agave uses standards-based technologies and community promoted best practices to enable users to run code, manage data, collaborate meaningfully, and integrate anywhere [1]. Agave ToGo is an open source, white label, reference web application that serves as both a defacto user interface for the Platform and a launching point for developers seeking to create new gateways leveraging the Platform. In the past year, demand has increased for smaller, focused gateways that facilitate a single activity and/or serve a particular function. To meet this need, Agave ToGo Microsites were developed. In this paper, we present the Agave ToGo webapps and microsites, examine how and when they can best be used to accelerate science gateway development, and look at two recent projects adopting ToGo for their gateways.**

***Keywords*—agave, gateway, science gateway, angularjs, web service, angular, microsite, webapp, responsive, website**

I. Introduction

The Agave Platform is an open, Science-as-a-Service (ScaaS) cloud platform for reproducible science. Agave uses standards-based technologies and community promoted best practices to enable users to run code, manage data, collaborate meaningfully, an integrate anywhere [1]. Agave ToGo (ToGo) is an open source, white label, reference web application for the Agave Platform. ToGo serves as both a defacto user interface for the Platform and a launching point for developers seeking to create new gateways leveraging the Agave Platform.

The field of web development has exploded since Agave first went into production in 2011. From the first release of GatewayDNA[2] in 2013 to the first release of Agave ToGo in late 2015, to the introduction of realtime microsites today, the rate at which technology changes is only increasing. Never before have there been so many frameworks and platforms with such strongly opinionated communities evangelizing their merits. The term "Javascript fatigue" has made its way into the mainstream vocabulary of web developers everywhere as they struggle to keep up with the latest and greatest technologies.

Despite the rate of change, there is hope. If we take a moment to step back from the bleeding edge and look around, we can identify some clear winners in the field. The big winner over the last 7 years was the movement towards API driven applications using REST. The current move to microservice architectures, edge computing, and mobile-first applications only further cement REST as the standard for web interoperability.

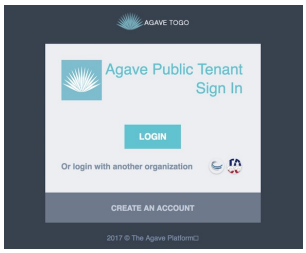
PHP still powers a significant portion of the web thanks to Wordpress, Drupal, and Laravel, but Python has gone mainstream, becoming the language choice of some of the largest websites on the web today. The new 800lb gorilla, however, is Javascript. Thanks to Google's V8 engine and Node.js, Javascript runs in everything from new microwaves to NASA's Mars Lander. Full stack application development is not only possible in Javascript, it is frequently the best choice for greenfield development.

Of the dozens of frontend Javascript frameworks appearing since 2011, Angular, React, and Vue have attracted the kind of critical mass needed from the open source community to make them safe choices for both commercial and non-commercial use. One could argue that Ember, Elm, or another language should be included in the group, but few would argue that any of three listed should not be.

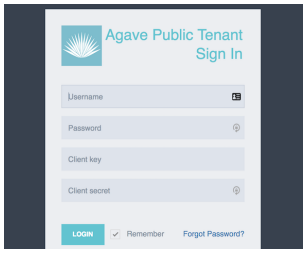
Finally, with all the uncertainty around HTML5, CSS3, and HTTP/2, the big winners were Webpack, Websockets, and Bootstrap, one of which is likely to be used nearly every time we load a web page or open a mobile app.

In response to these changes, the Agave team worked with several groups from our user community to understand how different web development trends impacted their needs and expectations. From those engagements, we identified two emerging areas of need for our users. First, we found that there was a strong, ongoing desire to deploy smaller, focused gateways that facilitate a single activity and/or serve a particular function. Second, we found that due to externally imposed constraints, gateway teams could reduce some of their risk and extend the life of their projects through the incorporation of a few common boutique APIs.

In this paper, we present our work to address each of these areas within the context of Agave ToGo. We start with an overview of Agave ToGo and look at the parallel implementations in Angular, AngularJS, Laravel, Flask, and Express. We then present our new ToGo Microsite Gateways and discuss how and when to best use them accelerate science gateway development. We finish with a look at a recent project adopting ToGo for its gateway and concluding remarks.



(a) Implicit Flow supports login to multiple tenants



(b) Resource Owner Password Credentials Flow login page

Fig. 1: Login screens for Agave ToGo

II. ToGo Webapps

Agave ToGo is the reference implementation of a science gateway built entirely on top of the Agave Platform. It was originally implemented as a static web application written in AngularJS [3][4]. This year, three new server-side implementations of ToGo will be ported onto the popular web frameworks Flask [5], Laravel [6], and Express [7]. While the implementation details will differ between languages and frameworks, every implementation of ToGo will present a consistent user interface and feature set.

A. Common features

Regardless of the implementation, all ToGo implementations have the following features.

1) Responsive dashboard layout: The user interface (UI) is a fully responsive, mobile friendly single page application built upon Twitter Bootstrap. The design utilizes an "admin dashboard" theme common to many of today's SaaS applications. A collapsing primary navigation menu is pinned to the left of the screen with content appearing in the main panel. A static header holds clickable notification icons indicating outstanding alerts, status messages, a user profile menu, and an icon button to toggle the application settings panel.

2) Integrated auth: Users must authenticate to use ToGo. Unauthenticated users are redirected to a login page where they can select the tenant they would like to use and login using the selected tenant's OAuth2 server. All frontend implementations support the Implicit Flow[8] shown in Figure 1a by default. The lifetime of implicit bearer tokens is usually 4 hours, but can vary from tenant to tenant.

While implicit bearer tokens cannot be refreshed, token refresh is supported when users provide their own client application keys and login via the Resource Owner Password Credentials Flow[8] shown in 1b. For users familiar with any of the Agave SDK or CLI, the local authentication cache file created by those tools can be dropped onto the sign in form and used to authenticate. If a refresh token is present, Agave will automatically refresh the user's bearer token when it expires.

Server-side implementations provide an additional Authorization Code Flow. This allows them to handle token refresh on behalf of their users without exposing sensitive information to the browser.

3) Multitenant aware: ToGo supports the use of multiple tenants. Tenant information can be looked up dynamically using the public Agave Tenants API, or it can be provided via a configuration file. By default, ToGo only requires the tenant code, URL to redirect the user after login, and a valid client key. Additional information can be included to support account creation and references to the help and project pages.

It is common for developers to interact with more than one tenant at a time. They may frequently interact with development and staging tenants when preparing a release. Once a user is authenticated to a tenant, their session, preferences, and request cache are namespaced with the tenant code and authenticated username. This allows transparent account switching both within and between tenants.

4) Modular API support: Unlike science gateways built for a specific domain or use case, ToGo provides a "kitchen sink" environment where users can interact with the Platform and developers can find real world examples of how the services can work together in a synergistic way.

Support of individual APIs is enabled through the inclusion their library as a dependency. In AngularJS, they are implemented as modules. In Angular, they are implemented as components. In either case, the UI dynamically builds navigation, routes, contextual menus, and page views based on the included APIs. Enabling and disabling a particular API is simply a matter of commenting out a couple lines of Javascript. Within the individual API libraries, all interaction with the Agave Platform is handled by one of the Agave SDKs. The AngularJS version of ToGo uses the AngularJS SDK, the Angular implementation uses the TypeScript SDK, etc.

Each API has basic form-based CRUD support. Additionally, wizards are available to simplify some of the more complex forms such as system registration, app definition, and job submission. Figure 2 shows the collapsible split panel view featured in every wizard. The form and JSON editor share a two-way binding so a changes to the form are immediately reflected in the editor and vice-versa. Users have found this particularly helpful to developers authoring JSON to use when building external integrations in other contexts.

5) Async and realtime support : Agave provides access to its event stream through the Realtime API and through event publication to realtime services such as PushPin, Fanout, and Pusher. Because the Realtime API is an optional service and not available in every tenant, ToGo will use a websocket connection to subscribe and listen for realtime notifications when available, and fall back on asyncronous polling otherwise.

Regardless of the implementation, it is possible that ToGo will at some point become unavailable to receive an event for one reason or another. In this situation, it will query the Notifications API for all notifications missed since the last recorded message receipt and process them in order. Regardless of how ToGo recieves messages, when a new message arrives, ToGo forwards it to an internal alerting system. Alerts will then be displayed as Toastr notifications in the top right of the page (as shown in Figure 3, an icon indicator in the top menu, and, if enabled by the user, a

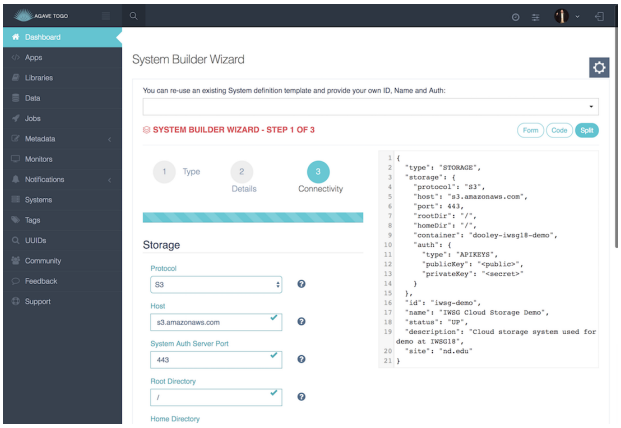


Fig. 2: Agave ToGo wizards feature two-way binding to a JSON editor allowing users to validate existing JSON or generate new JSON with a form interface.

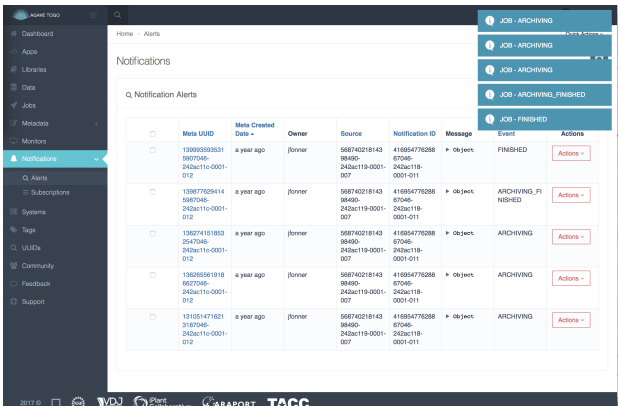


Fig. 3: Agave ToGo presents toastr notifications upon realtime and async message delivery.

desktop notification. Both message verbosity and display are configurable through user preferences.

B. Frontend implementation

ToGo was first implemented as a front-end only web application. The goal was to provide users a full-featured example of a science gateway with zero ongoing operational costs and answer the question, "How does one build a Science Gateway for under a million dollars?" [9]. The AngularJS version of ToGo accomplishes just that. In fact, for the last 2 years, the official Agave ToGo deployment at http://togo.agaveplatform.org is actually hosted directly out of the project's Github repository using Github Pages.

In addition to the financial advantages of this approach it also,

- Democratizes availability of science gateways to anyone with basic web development competency: html, css, javascript.
- Exposes students to skills and technologies heavily desired by employers.
- Dramatically lowers the time to first gateway by replacing all hosting, DNS, and publishing infrastructure with a simple fork of a Github repository.
- Removes the success penalty experienced by many science gateways when their usage outgrows their original budget and/or timeline.

1) Static frontend implementation: Since the initial development of ToGo, the next generation of AngularJS, simply called Angular, was released. Work is currently underway to port the existing AngularJS version of ToGo to Angular. This represents a significant change in the application architecture, but brings with it some welcome changes, such as the use of Typescript, ECMA 6, better modularization of individual Agave and external API support through the use of Angular components, and reduction on 3rd party libraries, and overall performance improvements.

2) React, Vue, and Ember: Just as our use of Backbone in GatewayDNA gave way to AngularJS in Agave ToGo, several other Javascript frameworks are gaining popularity in the developer community. Among them, React and Vue have seen the greatest adoption for new application development. While no work is currently underway on ToGo implementations in these frameworks, we are actively looking for opportunities to work with the user community on such implementations.

C. Full stack implementations

Over the last year, we worked with several groups who presented use cases that necessitated the use of a server-side component. Their use cases can be summarized by the following 4 scenarios:

- The gateway provides access to one or more shared resources from which, access must be granted and tracked using a privileged account.
- The gateway needs to perform one or more actions in response to an event from the Platform. The event involves manipulating the webhook's contents and using that in a request back to the Platform on behalf of a user.
- The gateway presents a view that requires information from multiple Science APIs. The gateway can greatly speed up the view rendering by handling its own caching and response generation.
- The gateway is meant to be an internal application for a specific group of users. Only those users should be able to login to the application. Once logged in, they should remain logged into the gateway on that computer until they login from somewhere else, or they manually log out.

As we continued to work with these groups, we realized that formalizing a server-side version of ToGo with general solutions to the above use cases could be something of value to the community. Over the last 6 months we have integrated code contributed back from the community into server-side versions of ToGo. This summer we will release implementations based on the 3 top languages and web frameworks as published by the Science Gateway Community Institute [10].

1) PHP and Laravel: Laravel is a free, open-source PHP web framework intended for the development of web applications following the modelviewcontroller architectural pattern. Currently on the fifth major release, Laravel has become arguably the top web application framework for PHP developers. The ToGo Laravel implementation uses the Agave PHP SDK to communicate with the Platform, Eloquent for application persistence, Larvel Echo for realtime communication, Socialite for OAuth2 authentication, and Passport for webhook authentication. Dependency management is handled by Composer. The project is available as source and as a Docker image. An included Docker Compose file will stand up a development version of the gateway out of the box. Initial gateway configuration and management are available through custom Artisan commands included with the gateway.

2) Python and Flask: Flask is a microframework for Python based on Werkzeug and Jinja 2. While somewhat less popular than Django for full blow application development, Flask is particularly well suited for light administrative apps like ToGo. A popular colloquial idiom comparing the two frameworks is that "Pirates use Flask, The Navy uses Django." Given the target audience of ToGo, Flask appears to be the appropriate choice at this time.

The ToGo Flask implementation uses the Agave Python SDK to communicate with the Platform, the native ORM for application persistence, "Flask-SocketIO" for realtime communication, "Python Social Auth" library for OAuth2 authentication, and "Flask-JWT" for webhook authentication. Dependency management is handled by pip. The project is available as source and as a Docker image. An included Docker Compose file will stand up a development version of the gateway out of the box. Initial gateway configuration and management are available through custom utility scripts included with the gateway.

3) Server-side Javascript and Express: Express is a free, minimal open source Node.js web application framework that provides a robust set of features for web and mobile applications. Express is fully asynchronous and makes heavy use of "middleware" to provide its functionality. The ToGo Express implementation uses the Agave Node SDK to communicate with the Platform, MongoDB and mongoose for application persistence, Redis for caching, Passport for OAuth2 and webhook authentication. Dependency management is handled by the Node Package Manager. The project is available as source and as a Docker image. An included Docker Compose file will stand up a development version of the gateway out of the box. Initial gateway configuration and management are available through custom npm commands included with the gateway.

III. ToGo Microsites

A growing trend for large organizations and small labs is the deployment of smaller, single-purpose applications called Microsites. Like their web service counterparts, microservices, microsites are single-paged web applications that providing a single, task-oriented experience to the user. Common use cases for microsites are job submission sites tailored for a single code, data transfer sites to schedule and move data, ETL (extract, transform, and load) applications for importing and scrubbing datasets, chat and messaging sites, and publication sites to control the release, approval, and publication of data.

While the Agave ToGo web application can, and in some cases already does solve these use cases, ToGo can feel like overkill when the use case is simply to enable a single task. In those situations, we recommend Agave ToGo Microsites.

Agave ToGo Microsites are single-purpose, single-page web applications that satisfy the basic need of many projects to run code, collaborate, and share results. Each instance is self-configurable, comes bundled with multiple theme and style options, has an optional backend server, and comes with devops tooling built in. There are currently 3 microsites available. We present each in turn.

A. Application oriented

The first microsite was developed to enable execution of a single application code. A screenshot of the microsite is shown in Figure 4. Unlike Agave ToGo, each application microsite is configured for a specific tenant, app, and storage enviroment. These values enable the site to streamline the UI, auto-generate and job submission form with client-side validation, auto-generate archive locations for each job, and create boutique notifications to update the user about their activity.

An optional Express server is included with the microsite. When activated, the microsite leverages server-side authentication, provides automated token refresh, and supports an access control list of users allowed to login to the microsite.

B. Data focused

The second microsite developed was focused on data movement. The need for reliable data transfer, sync, and replication is one of the most common use cases that bring new users to Agave. The data microsite provides a simple UI for the Agave Files and Transfers services, making it easy to move data to, from, and between multiple storage solutions. The microsite tracks transfer progress, sets notifications, and provides a simple way to set up one-off and repeating scheduled transfers. The data microsite requires a server-side component to run. The server provides automated scheduling, webhook processing, and automated token refreshing.

C. Impact dashboard

The third microsite, still under development, is a dynamic dashboard builder focused on presenting real-time stats and reporting information about the Agave Platform from a user perspective. Application publishers and tenant administrators frequently request usage and impact stats from our operations team. While this information is easily available by querying the Platform's REST APIs, users frequently ask for a way to pull together this information in views and reports similar to those found in the Agave Stats Dashboard, show in Figure 5.

The impact dashboard microsite allows users to build custom dashboards made up of graphs, tables, and widgets from

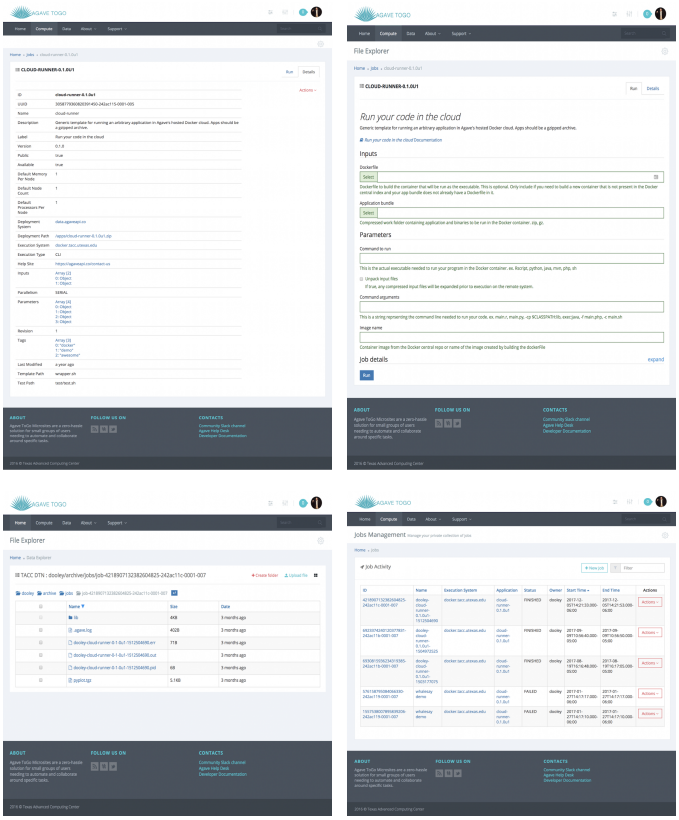


Fig. 4: Screenshots of an application oriented microsite. Clockwise from top left: execution history, app information, submission form, output browser.

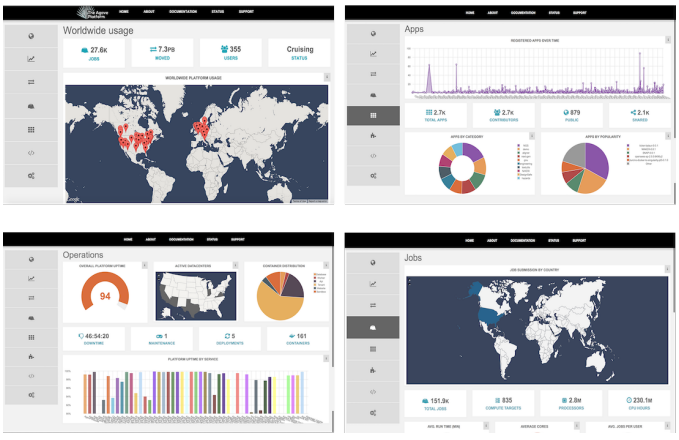


Fig. 5: Screenshots of four sample Agave Impact dashboard microsite views.

Component	Category	Description
Dial	graph	Numeric value as an animated dial
Progress bar	graph	Horizontal progress bar
Bar	chart	Color coded bar chart of a dataset
Line	chart	Color coded line chart of dataset
Pie	chart	Color coded pie chart of a value in a dataset
Counter	badge	Displays a single value with label
Ratio	badge	Displays a ratio with label
Profile	badge	Generates a user profile badge with basic info and avatar
Status bar	graph	Displays a color coded status value for a particular resource
Listing	table	Dynamic table based on fields in the response from a resource collection
Search	table	Dynamic table based on fields in the response from a search against one or more resources
Temporal	table	Search table with timepicker and predefined timeframes.

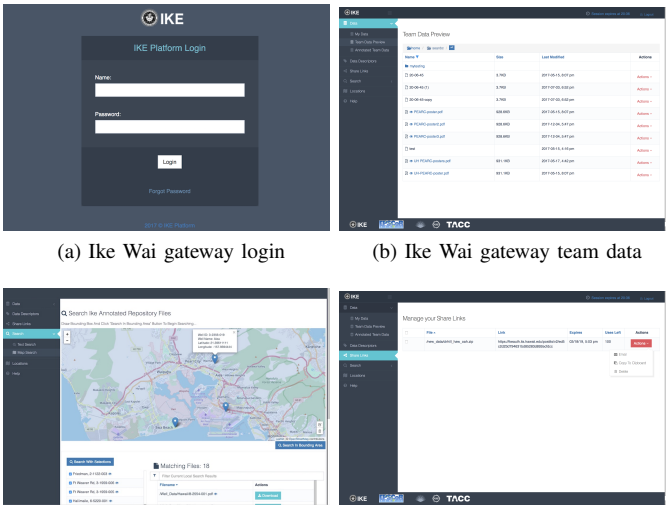
Fig. 6: Components available in the impact dashboard microsite.

a collection of predefined components. The components currently available are listed in table 6. By default, all dashboard changes are persisted in the browser's local storage. Settings can also be exported as a metadata record or flat file at the user's request. No server-side component is required.

IV. The Ike Wai Gateway

Agave ToGo was used at the University of Hawaii (UH) to build the Ike Wai gateway, Ike meaning knowledge and Wai meaning water in Hawaiian [11]. The Ike Wai gateway supports water research and provide tools in pursuit of the question of water sustainability in Hawaii. The gateway aims to provide centralized web based user interfaces to support multi-domain data management, computation, analysis and visualization tools to enable reproducible science, modeling, data discovery and decision support for the Hawaii EPSCoR Ike Wai research team and wider Hawaii water community. By leveraging Agave ToGo, UH has was able to construct a gateway that ties data and advanced computing resources together to support domains including microbiology, geochemistry, geophysics, economics and humanities with computational and modeling workflows delivered in a user friendly web based interface in a rapid manner.

The adoption of Agave ToGo enabled Ike Wai developers to quickly stand up a working gateway and establish a tight



(a) Ike Wai gateway login

(b) Ike Wai gateway team data

(c) Ike Wai gateway map search

(d) Ike Wai gateway shared links

Fig. 7: Screenshots of the Ike Wai gateway.

feedback loop with early adopters. The shorter feedback loop paved a path to iterative development in collaboration the project's end users. The end result was the identification and implementation of features that users needed much earlier in the project.

The use of Agave ToGo availed the two UH developers to a community of developers, adopters, and evangelists where they could turn to ask questions and receive advice on feature implementation such as the backend authentication service used to white list users and provide persistent token refresh. To date the Ike Wai gateway extends ToGo with interfaces to annotate data files with Agave's metadata API, perform full-text and spatial search of annotated data files, and filter search results using an interactive leaflet map.

V. Conclusion

Agave ToGo, like all software, must continue to evolve to stay relevant to the changing needs of the the Agave Platform's user community. This year we introduce two new areas of growth targeted at meeting emerging needs communicated by our community. The first is a the availability of server-side components to make offline activity, access control, and token refresh easier to implement. To address this need, we are releasing parallel implementations of ToGo in the Flask, Express, and Laravel application frameworks. The second area of growth is suppport for streamlined gateways focusing on a single task or activity. To address this need we are introducing three different Agave ToGo Microsites to provide a way to rapidly deploy low cost, low maintenace solutions to a pain point expressed by many users representing the long tail of science.

Going forward, we expect to further expand ToGo's server-side componets to make building and registering boutique APIs easier to do. We are also exploring ways to replicate the functionality of ToGo's server-side components using Agave's serverless solution, Abaco [12].

Acknowledgment

This material is based upon work supported by the National Science Foundation Plant Cyberinfrastructure Program (DBI-0735191), the National Science Foundation Plant Genome Research Program (IOS-1237931 and IOS-1237931), the National Science Foundation Division of Biological Infrastructure(DBI-1262414), the National Science Foundation Division of Advanced CyberInfrastructure (1127210), and the National Institute of Allergy and Infectious Diseases (1R01A1097403).

References

[1] R. Dooley and S. Cleveland, "The agave platform: An open science-as-a-service cloud platform for reproducible science," in *Practice & Experience in Advanced Research Computing*, Jul. 2018.

[2] D. Rion and H. M. R., "Recipes 2.0: building for today and tomorrow," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 258–270.

[3] "AngularJS Superheroic JavaScript MVW Framework." [Online]. Available: https://angularjs.org/

[4] "Agave ToGo Github repository," Apr. 2018, original-date: 2017-08-09T18:28:19Z. [Online]. Available: https://github.com/agaveplatform/agave-togo

[5] "Welcome | Flask (A Python Microframework)." [Online]. Available: http://flask.pocoo.org/

[6] "Laravel - The PHP Framework For Web Artisans." [Online]. Available: https://laravel.com/

[7] "Express - Node.js web application framework." [Online]. Available: https://expressjs.com/

[8] D. Hardt, "The OAuth 2.0 Authorization Framework." [Online]. Available: https://tools.ietf.org/html/rfc6749

[9] R. Dooley and M. R. Hanlon, "Recipes 2.0: building for today and tomorrow: RECIPES 2.0: BUILDING FOR TODAY AND TOMORROW," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 258–270, Feb. 2015. [Online]. Available: http://doi.wiley.com/10.1002/cpe.3285

[10] K. A. Lawrence and N. Wilkins-Diehr, "Roadmaps, Not Blueprints: Paving the Way to Science Gateway Success," in *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the Campus and Beyond*, ser. XSEDE '12. New York, NY, USA: ACM, 2012, pp. 40:1–40:8. [Online]. Available: http://doi.acm.org/10.1145/2335755.2335837

[11] Sean Cleveland and Jennifer Geis, "Ike Wai ToGo." [Online]. Available: https://ikewai.its.hawaii.edu/

[12] "Actor based co(mputing)ntainers," 2014, last access: 2015-11-11. [Online]. Available: https://github.com/TACC/abaco