Software-as-a-Service: The iPlant Foundation API

Rion Dooley, Matthew Vaughn, Dan Stanzione, Steve Terry
Texas Advanced Computing Center
The University of Texas
Austin, USA
[dooley, vaughn, dan, sterry1]@tacc.utexas.edu

Edwin Skidmore
iPlant Collaborative
University of Arizona
Tucson, USA
edwin@iplantcollaborative.org

***Abstract*—The iPlant Foundation API (fAPI) is a hosted, Software-as-a-Service (SaaS) resource for the computational biology field. Unlike many other grid-based approaches to distributed infrastructure, the fAPI provides a holistic view of core computing concepts such as security, data, applications, jobs, and systems. It also provides the support services, such as unified accounting, provenance, metadata, and global events, needed to bind the core concepts together into a cohesive interface. Operating as a set of RESTful web services, the fAPI bridges the gap between the HPC and web worlds by supporting synchronous and asynchronous interfaces, web-based callbacks, unified access control lists (ACL), and a publication- subscription (pubsub) model that allows modern applications to interact with the underlying infrastructure using technologies and access patterns familiar to them. In its first year of operation, the fAPI has grown to support thousands of users across both the Plant and Animal Science domains. In this paper we describe the fAPI, its underlying architecture, and briefly describe its adoption before concluding with future plans.**

Keywords— cloud; biology; cyberinfrastructure; iplant; data; api; saas; web service; rest

I. INTRODUCTION

The iPlant Collaborative (iPlant) is a United States National Science Foundation (NSF) funded project that has created an innovative, comprehensive, and foundational cyberinfrastructure (CI) in support of plant biology research [1]. iPlant is developing cyberinfrastructure that uniquely enables scientists throughout the diverse fields that comprise plant biology to address Grand Challenges in new ways, to stimulate and facilitate cross-disciplinary research, to promote biology and computer science research interactions, and to train the next generation of scientists on the use of cyberinfrastructure in research and education. The iPlant cyberinfrastructure design is based on an unprecedented period of research community input. It leverages developments in high-performance computing, data storage, and cyberinfrastructure for the physical sciences, and is an open-source project with application programming interfaces that allow the community to extend the infrastructure to meet its needs.

iPlant provides a wide range of computational resources, aimed at supporting different use cases. The iPlant Data Store is a persistent, high performance, distributed storage solution built upon iRODS[2]. Atmosphere is a highly customized, on-demand virtualization solution built upon Eucalyptus [3][4]. Multiple high performance computing (HPC) resources through allocations on XSEDE [5], FutureGrid [6], the Open Science Grid (OSG) [7], and the University of Texas System [8] are provided free of charge. iPlant also manages its own high throughput computing (HTC) resources at the University of Arizona. Each of these resources comes with its own set of interfaces, allocations, and access mechanisms.

To lower the barrier of entry to these resources and facilitate broader adoption, community-led iPlant steering committees identified a need for a programmatic access layer. In-person surveys and discussions from over 100 subsequent community workshops validated this recommendation and provided an initial set of requirements for the desired access layer to meet. Obvious capabilities such as the ability to submit jobs, track provenance, move data, and share work topped the list. Additionally common requests came from the community to apply and manage metadata, interoperate with their existing private resources, interact with 3rd party services, and support ontological discovery. Other, larger development projects requested some developer-friendly features such as a Representational State Transfer (REST) [9] interface, callback notifications, token-based access, and support for asynchronous communication.

Driven by this initial set of requirements, the development team conducted an evaluation of current technologies to identify projects they could adopt or extend to provide such an access layer. This evaluation is discussed as background material in Section II.

II. BACKGROUND

The evaluation began with hosted services. Globus Online [10] provided a production quality, hosted data movement service, but at the time of this writing, required separate authentication mechanisms, provided no metadata support, and could not be embedded into existing services and applications. There is hope that these features will be provided in the near future, but until then, deep integration is not possible. The Cyberaide project [11] showed great potential as a hosted job submission service, however the project is currently unsupported, the software suffers from scalability issues associated with large input data, and the model of wrapping each individual application in a virtual appliance doesn't match well with usage scenarios involving users leveraging the service as a mechanism for rapid application development. Airavata [12] is a well supported workflow orchestration and

execution service, but currently lacks key permission and data access concepts requested by the iPlant community. Several discussions with the Airavata project team indicated that significant refactoring would be required to support these concepts and such work was not in the scope of their current roadmap.

Next, the development team looked at Infrastructure-as-a-Service (IaaS) solutions. Commercial companies such as Amazon, Rackspace, and Microsoft provided solutions with mature web service offerings that allowing customers to interact with their infrastructure. However the primary focus of their web services was the administration of the resources they own. VMWare, OpenStack, Nimbus, and Eucalyptus all provided IaaS solutions solutions similar to the hosted commercial offerings, but run on local systems. These were attractive solutions iPlant could, and did adopt to expose their underlying cloud resource, but neither the hosted nor private IaaS solutions provide services of sufficient abstraction or breadth to meet the requirements of the user community.

Finally, the development team turned their evaluation focus to popular "middleware" projects such as the Globus Toolkit[13], Grisu[14], Newt[15], SAGA[16], and Unicore [17]. Each project had its own advantages, but the focus was generally on a subset of the functionality desired by the iPlant user community. The Globus Toolkit provided solutions for cross-site authentication, data movement, job submission, and information management, but worked in a way largely disconnected from the underlying compute and data systems. Exposing the Globus GRAM scheduler as an outward facing service to iPlant users would create the possibility of a discrepancy between queue status reported by the system schedulers and the Globus scheduler. The development team learned that in other projects, this led to accounting and monitoring difficulties that could never be completely worked out. Another roadblock to adopting the Globus Toolkit as a top level solution was that interaction with the Globus services required a separate technology stack. This technology stack would be required on every system communicating with iPlant. Though much improved in the past year, the installation of this software is still challenging for novice users to install and configure. Given the 7000 users iPlant currently supports, the burden that adoption of the Globus Toolkit would potentially add to the iPlant support staff weighed significantly in the final decision to leverage some of the lower level Globus components such as GridFTP for internal use, but refrain from exposing any at the user-facing level.

Grisu is an open source project developed by the Australian Research Collaborative Service's National Grid. It provides SOAP, REST, and GUI interfaces for job submission. The approach of Grisu was in line with community requirements, however the scope of the project was much smaller than that of iPlant and the underlying technology was aging. Several of its core dependencies had been deprecated and no effort was being made to update to alternative technologies. At the time of the initial review the Grisu development team was phasing out support of the project due to lack of interest and funding. As a result, there was little opportunity to collaborate.

NEWT is a web service that allows users to access computing resources at the National Energy Research Scientific Computing Center (NERSC) through a simple REST API. NEWT exposes services for authentication, system monitoring, file uploads and downloads, directory listings, running commands, submitting jobs to batch queues, obtaining accounting information, and accessing a persistent object storage. NEWT is an internal project at NERSC and, as such, is deeply tied to the underlying NERSC infrastructure. After evaluating the NEWT architecture, it was found that while it did meet many of the requirements, it would require nearly a complete rewrite to port NEWT onto the iPlant cyberinfrastructure.

The Simple API for Grid Applications (SAGA) is a programmatic interface for managing jobs and data on a computational grid. It currently has bindings in Java, C++, and Python, however it lacks any web service iterface.

Unicore is a comprehensive middleware solution similar to the Globus Toolkit, but exposed as a set of SOAP web services. One benefit of Unicore is its use of community established protocols for job submission, information management, and authentication. The primary drawbacks to Unicore are the large buy-in required to leverage it, the technology stack required by clients to interact with the SOAP services, and the system-level installations required by both the clients and resource providers. Given iPlant's lack of administrative access to the underlying HPC hardware and the requirement for REST and web-accessible services, Unicore could not be an outward-facing technology in the desired iPlant programmatic access layer.

After completing the evaluation process, the development team determined that, for lack of a better description, there was abundant plumbing, but no kitchen sink. Despite the needs of the community, there was no solution that allowed significant re-use of existing code bases to meet the needs of the plant science community, and so the development team designed and created the Foundation API (fAPI). The remainder of this paper is organized as follows. Section 2 gives an overview of the fAPI design. Section 3 describes the individual services comprising the API. Section 4 provides usage and adoption metrics from the fAPI's first year of operation. Section 5 concludes with a brief roadmap for future development.

III. Design and Architecture

The Foundation API is designed to operate as a multitenant, cloud-based, Software-as-a-Service (SaaS) solution. Two physical instances of the fAPI services sit behind an Apache Load Balancer and direct requests between the primary instance at Texas Advanced Computing Center and the mirror instance at the University of Arizona. This single logical instance of the fAPI is available at https://foundation.iplantcollaborative.org and serves iPlant users and partner projects alike.

The fAPI services are implemented in Java and PHP as RESTful web services. The Java services leverage the Restlet [18] framework and run in a Tomcat 6 container. Hibernate is used to handle the ORM to a shared MySQL database [19].

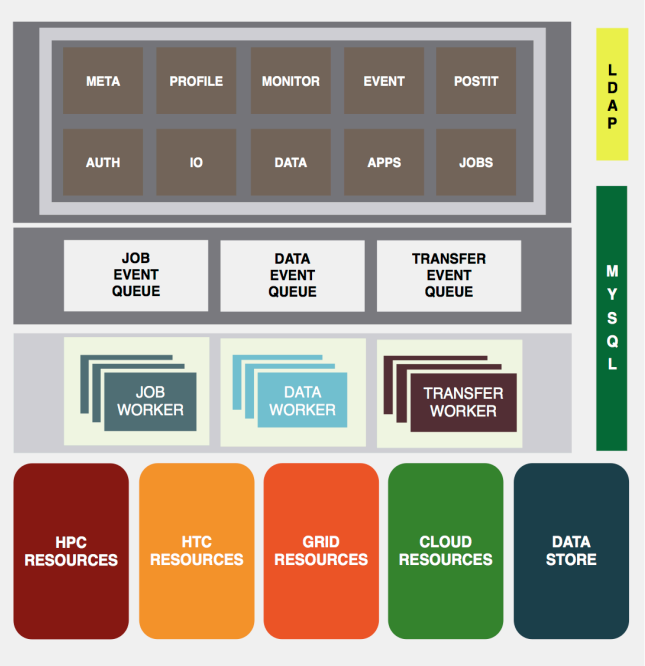

Figure 1. Conceptual overview of the Foundation API architecture.

The PHP services leverage portions of the CodeIgniter framework and utilize the native PHP 5 PDO support [20].

The fAPI has both public and protected interfaces depending on the nature of the data they expose. The protected endpoints are accessible via HTTP Basic over SSL. While iPlant uses LDAP as the backing identity store for the entire projects, the fAPI does not expose this directly, but rather supports token-based authentication. Consumers request a token from the Auth service using their client credentials and use that token to interact with the rest of the API until it expires.

Supporting the front-end services is an elastic set of worker services. These workers handle the lengthy task requests made to the front-end services such as file staging, job submission, and data transformation. The workers are implemented in Java as headless services running on distributed virtual machines. The workers do not directly communicate with the front-end services. The front-end services submit tasks to the underlying queue service and the workers watch their respective queues and claim tasks as they appear. The queue service is implemented in Java using the Quartz scheduling framework backed by a MySQL database [21]. A separate Event service complements the queue service and provides publication-subscription (pubsub) functionality to the API.

As the overall workload on the fAPI increases, scaling up is trivial. Additional worker services are started as new virtual machines (VM) are spun up. When the overall workload decreases, the VM can be shut down. If a worker fails or becomes unresponsive, another worker will claim the lost task and carry on. Fig. 1 illustrates the overall fAPI architecture.

As mentioned above, the fAPI services support both synchronous and asynchronous interaction. This allows client applications to fire and forget long running tasks such as job submission and data movement. Fig. 2 illustrates a typical flow of execution when the IO service is invoked. The scenario shown occurs when a user requests that the IO service fetch a file from an external data source, preprocess or convert it in some way, and store it in the iPlant Data Store. While fulfilling the request, the task will pass through several queues, keeping the front-end service aware of its progress. Once completed, an optional notification will be sent to the user via email or web hook alerting them that their file staging request has completed.

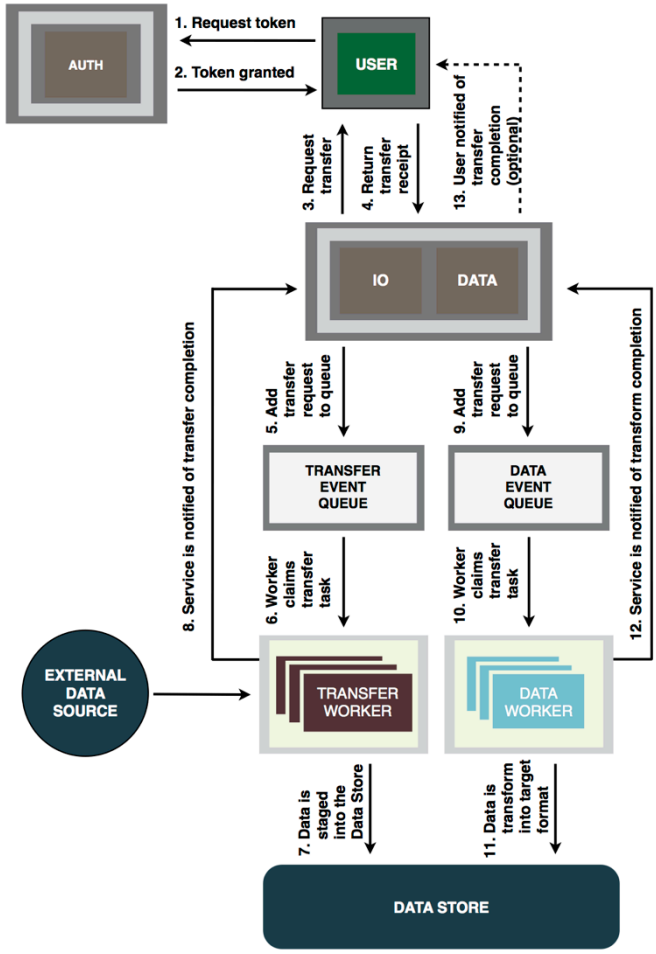

Figure 2. A typical flow of execution when an asynchronous call is made to the fAPI. In this figure, the user requests a file be staged in from an external data source, transformed, and stored in the iPlant Data Store.

Because iPlant does not own the underlying HPC, storage, or network resources needed to conduct much of the science it enables, the quickest time to production came from running the fAPI under a single community account and brokering actions on behalf of its internally vetted users. The complexity of implementing the services in this way was significantly less than supporting authentication and usage scenarios outside of the project. Using a community account also made provenance achievable across the entire enterprise, and allowed for a much smaller burden on the system administrators. The primary disadvantage of this approach was that it placed a significant security responsibility on the development team and increased the level of reporting needed at all levels of the system. It is also not a sustainable approach from a resource utilization

standpoint. This was one of the lessons learned from the CIPRES project [22]. When useful software is available to the community, community requests will quickly and forevermore out-pace the available resources. Given the already oversubscribed state of the underlying shared HPC resources, it is not possible to continue to grant larger and larger percentages of the available cycles to a single project. Thus, in the long term, the fAPI must move to enable users to charge work to their individual system allocations when appropriate. This topic is addressed more in the section on future work.

Three other considerations influenced the overall design of the fAPI without impacting the architecture. One of the most frequently requested features from the community during the requirements gathering phase was collaborative support. Users wanted a "share anything" model that would allow them to work with past, present, and future colleagues regardless of their affiliation with iPlant. As a result of this requirement, flexible access control lists (ACL) were built into all relevant pieces of the fAPI allowing users to share anything at any time with anyone.

Second, the importance of a clean, friendly API became readily apparent for a variety of reasons. A 2008 study of nearly 2000 researchers found that over 96% of those surveyed said that self-study was important when learning how to develop software while only 30% could say the same for formal education. That same study shows that 84% of the researchers surveyed claim that software development is critical to their research, yet they were only able to allocate 30% of their time to software development. [23]. Two things jump out from this study. First, the vast majority of users will not be professional programmers and will not have significant experience nor the time to learn new techniques and methodologies. In many cases users are scientists who are simply programming out of necessity. For these researchers, the learning curve required to begin building their software can be daunting. They first need to discover what services are available to them. Next, given a set of disjoint services, they must decipher how to piece them together using different communication protocols, authentication mechanisms, and access patterns. Once they figure that out, they must then familiarize themselves with several different APIs in order to write the necessary integration code. Once all that is in place, they can then begin debugging the idiosyncrasies of each scheduler, compute, and storage system to find out why their integration code doesn't work as advertised. Finally, after all that is in place, they can begin building the application they originally set out to build. Given the prevalence of graduate students doing much of the work and the inherent turnover rate of such employees, setting the barrier to entry that high is detrimental to the goal of advancing science.

The results of that study corresponded with observations made by the iPlant User Support and Engagement teams during the first 2.5 years of the project. Thus, as part of the design process, the development team took lessons from other popular APIs used in the commercial space such as Yelp, Dropbox, Flickr, Facebook, Amazon, and Paypal. User stories were constructed around mobile, web, desktop, and command-line usage scenarios. The result was an API that erred on the side of usability rather than compliance. It promoted progressive user buy-in rather than forcing vendor lock-in. It promoted a grocery store approach to adoption by allowing users to take what they need and leave the rest for later.

Lastly, the development team, though many conversations with the Discovery Environment development team as well as external projects, identified great value in providing a solution that, from a user perspective, worked the way the rest of the web worked. It should be designed in such a way that it could be easily consumed and mashed up with other services in a short amount of time. It should realize that polling for information is a bad design pattern and allow users to utilize better access patterns when they were ready. It should understand that every application has its own concept of "fresh" information and allow them to access information at whatever rate and level they felt necessary.

These three considerations were major influences on the design of the fAPI. In the next section we bring together these design decisions along with several more subtle ones to describe the individual services of the fAPI.

IV. Foundation API Services

A. Authentication Services

The first interaction users have with the fAPI is with authentication. Users access protected endpoints in the API using an authentication token. They obtain that token from the Auth service. The Auth service uses LDAP to authenticate users and issue renewable, short-term credentials in the form of tokens. The service supports token revocation, blacklisting, delegation, and fixed usage scenarios. Tokens are implemented simply as alphanumeric strings and can be used in place of a user's password when making HTTP Basic requests to the protected service endpoints.

One use case not addressed by the Auth service is that of low-trust scenarios. Often times, user workflows include interaction with a third-party services. Granting these services unlimited access to the user's account is highly undesirable. The Auth service can issue single use tokens, but it does not restrict the URL to which that token is valid. As a result, that token could be used to invoke any action using the API, not just the one intended by the user. The PostIt service was created for this purpose.

The PostIt service is a pre-authenticated URL shortening service. Similar to popular URL shortening service such as TinURL.com and Bit.ly, the PostIt service creates an obfuscated, short URL out of any endpoint in the fAPI. In the case of protected endpoints, the user can pre-authenticate the URL by authenticating to the PostIt service when creating the URL. As with the Auth service, the user can restrict lifetime and number of uses of the generated URL, but using the PostIt service, that permission will be restricted to the registered URL. When the third-party service invokes the PostIt URL, the PostIt service add the appropriate authentication token and proxies the request to the original URL. Any query parameters, form variables, or attachments are forwarded on and the response is relayed back. As with the Auth service, PostIt URLs can be revoked as needed.

B. Data Services

The IO service is the core data movement service of the Foundation API. It serves as a value-added abstraction layer on top of the iPlant Data Store.. The IO service, like the Data Store, provides basic file operations and exposes fine-grained ACL. Data can be transferred both synchronously via HTTP GET and POST, and asynchronously by requesting a data transfer from an external data source using any of a number of protocols. Current protocols supported are FTP, GridFTP, SFTP, HTTP, IRODS, and Amazon S3. When requesting data from the IO service, data can be requested in whole or in part using partial data queries. When scheduling long running transfers, a callback can be specified. Callbacks can be an email address or a web hook in the form of a HTTP endpoint to which the service will POST a response. To allow customization of the notifications, a template system is provided for use in defining the callback URL query.

The IO service also provides a chained pre-processing mechanism that allows users to transform data upon upload/import before it reaches its final destination. This allows raw data to be imported, yet arrive in a polished form. This is particularly useful when importing database dumps, raw experimental output, and data from deprecated file formats, but also handy for automatically compressing or decompressing data and running metadata extraction on datasets as they come in.

The Data service was created in response to the need for scientists to collaborate without having to learn many different software packages and data formats. The Data service acts as a Rosetta stone for plant biologists allowing them to convert their existing data from one format to another and stage it to a location they specify using any of the protocols supported by the IO service. There are currently 49 different data formats supported by the Data service. It is not possible to transform in every pairwise manner, so the data service provides a mechanism to determine the available transformations given a source data format. Like the IO service, the Data service supports both synchronous and asynchronous requests, callbacks, and partial data queries.

While the IO, Transfer, and Data services exist to aid in the management of raw data, the Meta service allows for the management of metadata. The Meta service provides metadata support for all of the fAPI services. It exists as an unstructured data store backed by a column-oriented NoSQL database[24]. Because metadata is, by definition, data about data, it has no inherent meaning. Thus, the Meta service allows users to register schemas that describe their data. Schemas are JSON or XML descriptions of data. These descriptions may be a series of tuples, or highly structured object definitions. The decision on how to describe their data is left up to the user. In order to bridge the gap between different schemas, users can register mappings between schemas. These mappings allow the Meta service to identify opportunities to do apples-to-apples comparisons between metadata associated with different schemas. This is helpful when doing global searches, constructing user-derived responses, and when using metadata to reconcile different raw data sets. The Meta service is currently in initial testing while undergoing scalability evaluation. It is anticipated that the service will hold on the order of a billion pieces of information in its first year. Querying such a dataset becomes challenging as that number grows. To improve search times, various methods of data reductions are being examined. We anticipate the Meta service to be available in full production early in 2013.

C. Execution Services

The Apps service is designed to serve as a central registry for applications deployed in the iPlant software infrastructure. In concert with the Jobs API, bioinformatic analysis applications can be deployed on a range of platforms, ranging from leadership cluster systems, to virtual machines running in an environment such as Atmosphere, to remote RESTful services. Support for truly heterogeneous, user-defined computing infrastructure such as Amazon EC2 and Google App Engine is an area of future growth in development now.

Users can search for applications by search terms such as ontological terms, tags, name, and unique ID. As a convenience for users building their own web GUIs, the Apps service also generates HTML forms for registered applications that can be posted to the jobs service for execution. Applications are available both publicly and privately. Public applications are administered and maintained by iPlant staff and made available for anyone to use. Private applications are registered by users and, by default, can be discovered and run only by their owners. Applications have their own ACL, so private applications may be shared with other users by setting the appropriate permissions.

Application registration is accomplished by posting a JSON description of the application to the Apps service. The JSON description includes basic information about the application name, version, target system, invocation method, and any required inputs, parameters, and outputs. Currently supported invocation methods include CLI, SGE, PBS, LOADLEVELER, LSF, TORQUE, and CONDOR. The description of the inputs and outputs is used exclusively for validating job submission requests and determining what information is needed by the executable to run. The process of wrapping an application is out of the scope of this paper. For more information on this process, consult the API documentation in [25].

The Job service is the core execution service in the API. Its sole purpose is to start and manage jobs across the underlying systems. Users POST a job request to the service and the service validates that request against the application description registered with the Apps service. A job handle is returned to the user and the job request is forwarded on to the submission queue. A job submission worker will pull the job off the queue, transfer in any missing data using the IO service and stage all dependencies to the remote system before submitting the job to the remote scheduler or forking the process. When the job completes, a callback will be made to the user. Like the IO service, the Job service supports a template system to enable the user to dynamically construct a callback URL at run time.

Jobs, like applications and files, have their own ACL and can be shared between iPlant users as well as the general public. To support quick access to job data, the Job service

provides convenience endpoints to easily locate and browse their output folders. This is helpful given that a job's output folder may or may not exist and changes over time. When the job is running, the folder is on the execution host. Once a job has finished, the folder is archived in the Data Store. If the job was not archived or failed right away, there will not be an output folder associated with the job.

Archiving is built into the job service. By default, all data generated during job execution is archived back to the Data Store and placed in a location specified by the user or, if not specified, in an archive folder generated by the service in the user's home space.

D. User Services

The Profile service acts as a directory service for iPlant. It allows users to view information about other iPlant users such as their name, contact info, and institutional affiliation. The identity management within iPlant is currently going through an upgrade as the project moves towards providing OAuth2 [26] support. As such, the profile service is currently available as a read-only service. Please consult the Future Plans section for more information about fAPI roadmap for identity management.

The Audit service provides detailed usage and accounting information across the iPlant cyberinfrastructure. From the Audit service, users can check their allocation status and quotas, obtain reports on their system-level usage history, their API usage history, and obtain their remaining balances across resources. The Audit service is targeted towards individual users rather than the organization as a whole. As a result, separate administrative interfaces are available for staff to create systemwide and multi-user reports.

E. Administration Services

The Systems service is a discovery endpoint for the fAPI. It provides information on what systems are currently available through the Jobs service for execution and what the current status of those systems is. This service is currently read-only. The section on Future Plans describes the ongoing work of the development team to support user-defined dynamic resource registration.

The Monitor service gives users a view into the health of the fAPI. Its main role is to tell users if any part of the fAPI is down and why. This information is obtained by running a suite of tests as a cron process that check the availability and response accuracy of the entire fAPI, its dependent services, and the underlying systems. Test results are pushed into a separate MySQL database and exposed by the Monitor service. Both the service and database are hosted physically separated from the rest of the services to ensure that it is accessibility even in the case of catastrophic failure in the rest of the fAPI.

The Track service is part of the provenance solution in the fAPI. It exists as a simple write-only service to track activity across the entire iPlant cyberinfrastructure. Every invocation of a fAPI service calls out to the Track service to create a record of its activity. The Track service provides a central place where one can mine usage information and service activity. All data from the service is currently stored in a MySQL database. This will change in the coming months as the size of data mandates a move towards a NoSQL solution.

The Event service provides pubsub capabilities to the rest of the fAPI. When a work action is taken in the fAPI such as a data or job operation, an event is triggered and pushed onto the user's event queue. The message is then available to anyone subscribed to that queue. The fAPI currently uses a custom Event service implementation based on the OGCE WS-Message project [27]. This may change going forward in anticipation of broad RabbitMQ [28] support across the rest of the project.

The Manager service is a catchall service for controlling the behavior of the other services. It provides reporting as well as the ability to shut down and drain work from the IO, data, and jobs worker services. It is a critical service in the Foundation API that we restricted exclusively to administrative users.

V. Usage and Adoption

The fAPI was released for public usage in September of 2011. Since then, over 250 users have leveraged the fAPI to enable projects representing more than 10,000 scientists worldwide. Users burned more than half a million SUs running over 4000 jobs across HPC systems at PSC, SDSC, and TACC. The fAPI data services now serve access to nearly half a petabyte of data. Fig. 3 shows the approximate usage history over the first year of operation.

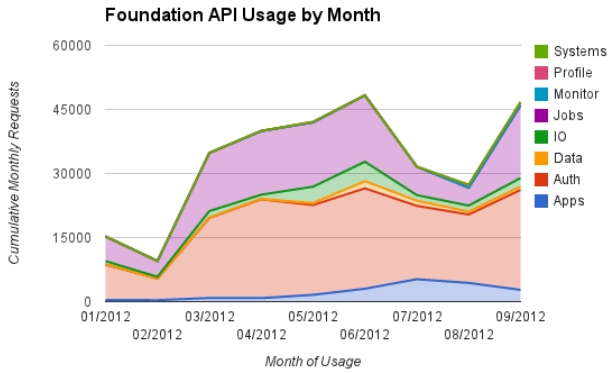

Figure 3. Foundation API monthly usage history. Usage increased consistently over the year with anticipated dips during breaks in the academic calendar.

In the remainder of this section we highlight three projects who have made significant contributions to this usage by engaging with the fAPI development team to integrate the fAPI deeply into their existing applications. These projects are the iPlant Discovery Environment, the BioExtract Server, and the Easy Terminal Alternative.

The first adopter of the fAPI was the iPlant's own Discovery Environment (DE) team [29]. Said Lead Developer Andrew Lenards, "The a la carte approach to the Foundation API was extremely attractive to an integrating system like the Discovery Environment. It primarily means that the system can leverage the portions of the framework that make the most impact. For the Discovery Environment's current implementation, this includes the Foundation API's Data, Auth, Apps, and Jobs services.

"A token-based authentication approach available from the auth service allows actions to be completed on-behalf of the Discovery Environment's users. Aspects of the data service are leveraged to enable users to manage their file-based data. And a simple "shim" script allows the Discovery Environment to off-load computationally intensive applications to the appropriate resources available at TACC. Through this simple script, the Discovery Environment can access applications deployed within the TACC and XSEDE environments, execute them, and monitor their status via commodity hardware using both the apps and jobs services.

The greatest value-added feature gained through integration with the Foundation API, however, was its monitoring capabilities. Experienced software developers are all-to-aware of the performance implications of polling, yet are often forced to implement such functionality due to lack of effective alternatives. Using a Pub-Sub model, the Foundation API does not force an integrator to poll; they just need to operate a simple end-point that allows the integrating system to subscribe to updates via web hooks available throughout the API. This makes bubbling the status of operations within the Foundation API easily achievable. The Discovery Environment has benefited tremendously from its adoption of the Foundation API and, as our needs grow over time, our adoption of other Foundation API services will increase."

The primary lesson learned working with the DE team was the cost of technological debt. The DE team started development a full year before the fAPI. As a result, they had hundreds of thousands of lines of code already deployed and working. Even small changes to their infrastructure could have significant impacts on the overall system both in terms of stability and developer effort. For them, the return on investment had to significantly outweigh the cost for any change they introduced. That placed a large responsibility on us to ensure that our services were stable and that we could clearly communicate the value of individual fAPI services so they could find integration points that fit into their existing development cycle.

The BioExtract Server is an open, Web-based system designed to aid researchers in the analysis of genomic data by providing a platform to facilitate the creation of bioinformatic workflows. Scientific workflows are created within the system by recording tasks performed by the user. These tasks may include querying multiple, distributed data sources, saving query results as searchable data extracts, and executing local and Web-accessible analytic tools. The series of recorded tasks can then be saved as a reproducible, sharable workflow available for subsequent execution with the original or modified inputs and parameter settings [30]. Over the last year, Dr. Carol Lushbough and the BioExtract team have leveraged the fAPI to enable researchers to execute iPlant analytic tools within the BioExtract Server, create iPlant workflows through the BioExtract Server, and manage analysis and provenance data across both platforms. The result of this collaboration has added a unified authentication mechanism between the two projects using the Auth service, expansion of their data capacity by several orders of magnitude using the IO and Data services, the addition of over two dozen new scientific codes to their application through the Apps service, and over 500 HPC jobs run across systems at TACC and SDSC using the Jobs service.

The biggest lessons learned from working with the BioExtract team was the need to respond quickly and manage change well. The BioExtract server already had an active user community relying on it for their daily work. Any changes made to the fAPI after integration needed to ensure backward compatibility or their application would break. As a result the development team spent time developing processes for releasing updates, versioning the services, migrating users between versions, and ensuring that provenance was pervasive across the fAPI. The processes made the fAPI a better product overall and helped manage our larger users more effectively.

Easy Terminal Alternative (ETA) is a web based front end to any infrastructure that allows novice users to submit, monitor, and share jobs [31]. ETA is designed specifically to transition users from the terminal to the web by easing the running of applications, streamlining the creation of scientific pipelines, and managing application lifecycles. The ETA development team at the Center for Genome Research and Biocomputing, led by Alex Boyd, used the fAPI Auth, Jobs, IO, and Apps services to provide overflow cycles for their campus users when their system reached full capacity. Leveraging the webhook features of the Jobs service, ETA was able to reconstruct existing pipelines using their existing workflow execution engine.

The primary takeaway from engagement with the ETA team was the need for multiple levels of documentation. While we had documentation in place for the fAPI, we didn't have the big picture description that explained the concepts of how the fAPI worked and the value proposition to its potential users. To that end, a developer's website will be released with version 2 of the fAPI.

VI. Future Plans

Looking forward, the Foundation API has an exciting roadmap. In the coming months the development team will be releasing the 2.0 version of the API, expanding functionality in several new directions while increasing performance and reliability in the existing services. Provenance will continue to be improved across the entire iPlant cyberinfrastructure. Global UUID will be given to every digital object that touches the iPlant cyberinfrastructure. The Meta service currently in testing will be rolled out as a first class service as well.

A new area of focus for the development team is authentication and identity management. A new OAuth 2 service will be deployed that will provide better accounting, provenance, and control over individual services than was previously available. At the same time, a new Systems service will be deployed that allows users to register compute and data resources external to iPlant for use through the fAPI. This will move the fAPI closer towards a true SaaS offering and help bridge the gap for users between iPlant and their local and campus resources.

Through a joint effort with the HPC and HTC system providers, a global RabbitMQ based event system will be deployed in the near future. This will allow for better

monitoring and information management across the entire system.

Finally, the Apps and Jobs services will continue to expand with additional functionality. Support for new platforms and execution mechanisms is ongoing and will bring commercial cloud support to the fAPI in the coming months. In order to support interoperability with other job execution services, additional job and application description formats will be supported. The addition of the new OAuth service will impact the Job service in a positive way. Once available, job submission using the user's personal system account and allocation will be supported.

This list of plans is in no way complete, but hopefully casts a vision of the direction in which development is currently headed. For more information on the fAPI and to follow along with its development, consult [25] and [32].

ACKNOWLEDGMENT

The iPlant Collaborative is funded by a grant from the National ScienceFoundation Plant Cyberinfrastructure Program (#DBI-0735191). This work was also partially supported by a grant from the National Science Foundation Cybersecurity Program (#1127210).

REFERENCES

[1] Stanzione, Dan, "The iPlant Collaborative: Cyberinfrastructure to Feed the World," IEEE Computer, November 2011. doi: 10.1109/MC.2011.297.

[2] Rajasekar, A., M. Wan, R. Moore, W. Schroeder, "A Prototype Rule-based Distributed Data Management System", HPDC workshop on "Next Generation Distributed Data Management", May 2006, Paris, France.

[3] iPlant Atmosphere: A Gateway to Cloud Infrastructure for the Plant Sciences. Skidmore E, Kim SJ, Kuchimanchi, S Singaram S, Merchant N, Stanzione D. Proceedings from Gateway Computing Environments 2011 at Supercomputing11 (2011)

[4] Eucalyptus. http://www.eucalyptus.com.

[5] The Extreme Science and Engineering Discovery Environment. http://xsede.org.

[6] FutureGrid: An Experimental, High-Performance Grid Test-Bed. http://portal.futuregrid.org.

[7] Mine Altunay, Paul Avery, Kent Blackburn, Brian Bockelman, Michael Ernst, Dan Fraser, Robert Quick, Robert Gardner, Sebastien Goasguen, Tanya Levshina, Miron Livny, John Mcgee, Doug Olson, Ruth Pordes, Maxim Potekhin, Abhishek Rana, Alain Roy, Chander Sehgal, Igor Sfiligoi, and Frank Wuerthwein. 2011. A Science Driven Production Cyberinfrastructure--the Open Science Grid. J. Grid Comput. 9, 2 (June 2011), 201-218.

[8] The University of Texas System. http://www.utsystem.edu.

[9] R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, Information and Computer Science, University of California, Irvine, California, USA, 2000.

[10] Ian Foster. 2011. Globus Online: Accelerating and Democratizing Science through Cloud-Based Services. IEEE Internet Computing 15, 3 (May 2011), 70-73.

[11] Kurze, T.; Lizhe Wang; von Laszewski, G.; Jie Tao; Kunze, M.; Kramer, D.; Karl, W.; , "Cyberaide onServe: Software as a Service on Production Grids," Parallel Processing (ICPP), 2010 39th International Conference on , vol., no., pp.395-403, 13-16 Sept. 2010. doi: 10.1109/ICPP.2010.47

[12] Suresh Marru, Lahiru Gunathilake, Chathura Herath, Patanachai Tangchaisin, Marlon Pierce, Chris Mattmann, Raminder Singh, Thilina Gunarathne, Eran Chinthaka, Ross Gardler, Aleksander Slominski, Ate Douma, Srinath Perera, and Sanjiva Weerawarana. 2011. Apache airavata: a framework for distributed applications and computational workflows. In Proceedings of the 2011 ACM workshop on Gateway computing environments (GCE '11). ACM, New York, NY, USA, 21-28.

[13] Ian Foster, Carl Kesselman, and Steven Tuecke. 2001. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. J. High Perform. Comput. Appl. 15, 3 (August 2001), 200-222.

[14] (grisu)Altinay, C.; Binsteiner, M.; Gross, L.; Weatherley, D.K.; , "High-Performance Scientific Computing for the Masses: Developing Secure Grid Portals for Scientific Workflows," e-Science (e-Science), 2010 IEEE Sixth International Conference on , vol., no., pp.254-260, 7-10 Dec. 2010. doi: 10.1109/eScience.2010.30

[15] S. Cholia, D. Skinner, J. Boverhof, "NEWT: A RESTful service for building High Performance Computing web applications", Gateway Computing Environments Workshop (GCE), 2010, January 1, 2010, 1--11,

[16] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C.Smith, A Simple API for Grid Applications (SAGA), OGF Document Series 90, http://www.ogf.org/documents/GFD.90.pdf.

[17] Dietmar W. Erwin , David F. Snelling, UNICORE: A Grid Computing Environment, Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing, p.825-834, August 28-31, 2001

[18] J. Louval, T. Templier, and T. Boileau. Restlet In Action. Manning Press. 2012.

[19] Hibernate. http://www.hibernate.org.

[20] PHP Data Objects. http://php.net/manual/en/book.pdo.php.

[21] Quartz Enterprise Job Scheduler. http://quartz-scheduler.org.

[22] Mark A. Miller, Wayne Pfeiffer, and Terri Schwartz. 2011. The CIPRES science gateway: a community resource for phylogenetic analyses. In Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery (TG '11). ACM, New York, NY, USA, , Article 41 , 8 pages. DOI=10.1145/2016741.2016785 http://doi.acm.org/10.1145/2016741.2016785

[23] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. 2009. How do scientists develop and use scientific software?. In Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering (SECSE '09). IEEE Computer Society, Washington, DC, USA, 1-8.Goecks, J, Nekrutenko, A, Taylor, J and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.Genome Biol. 2010 Aug 25;11(8):R86.

[24] Rick Cattell, Scalable SQL and NoSQL data stores, ACM SIGMOD Record, v.39 n.4, December 2010 [doi>10.1145/1978915.1978919]

[25] iPlant Foundation API Developer Documentation. https://foundation.iplantcollaborative.org/docs.

[26] OAuth2. http://oauth.net/2/.

[27] Y. Huang, A. Slominski, C. Herath, and D. Gannon, " WS-Messenger: A Web Services based Messaging System for Service-Oriented Grid Computing ," 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid06)

[28] Rabbitmq. http://www.rabbitmq.com/blog/.

[29] Andrew Lenards, Nirav Merchant, and Dan Stanzione. 2011. Building an environment to facilitate discoveries for plant sciences. In Proceedings of the 2011 ACM workshop on Gateway computing environments (GCE '11). ACM, New York, NY, USA, 51-58. DOI=10.1145/2110486.2110494.

[30] Lushbough, C., Jennewein, D., Brendel, V., (2011), The BioExtract Server: a web-based bioinfromatic workflow platform, Nucl. Acids Res, vol. 39, iss. W528-W532.

[31] Easy Terminal Alternative. http://eta-pub.cgrb.oregonstate.edu/about.php.

[32] iPlant Foundation API Forums. https://foundation.iplantcollaborative.org/forums.