Globus: A Case Study in Software as a Service for Scientists

Conference Paper · June 2017

DOI: 10.1145/3086567.3086570

8 authors, including:

Ian Foster
University of Chicago
918 PUBLICATIONS **81,878** CITATIONS
SEE PROFILE

Ravi K Madduri
Argonne National Laboratory
79 PUBLICATIONS **802** CITATIONS
SEE PROFILE

Jim Pruyne
University of Chicago
48 PUBLICATIONS **2,338** CITATIONS
SEE PROFILE

Steven John Tuecke
University of Chicago
136 PUBLICATIONS **29,016** CITATIONS
SEE PROFILE

Globus: A Case Study in Software as a Service for Scientists

Bryce Allen, Rachana Ananthakrishnan, Kyle Chard, Ian Foster, Ravi Madduri, Jim Pruyne, Stephen Rosen, and Steve Tuecke
Computation Institute, University of Chicago and Argonne National Laboratory
5735 S Ellis Ave
Chicago, Illinois 60637

ABSTRACT

While some properties of SaaS have long been leveraged in science, particularly in science gateways, there is yet to be widespread adoption of SaaS models. For example, few scientific SaaS providers are publicly available, few leverage elastic cloud platforms, and none—with the exception of Globus—implement subscription-based models to recoup operations costs. Globus has employed the SaaS model for seven years and is fast approaching subscription levels that will support long-term sustainability. In this paper we discuss the SaaS paradigm and explore its suitability to scientific domains. We then describe how production Globus SaaS services are implemented, deployed, and operated.

CCS CONCEPTS

•Software and its engineering → Distributed systems organizing principles; •Security and privacy → *Security services;* **•Computer systems organization →** *Distributed architectures;*

KEYWORDS

Science as a service, Globus, research data management

ACM Reference format:
Bryce Allen, Rachana Ananthakrishnan, Kyle Chard, Ian Foster, Ravi Madduri, Jim Pruyne, Stephen Rosen, and Steve Tuecke. 2017. Globus: A Case Study in Software as a Service for Scientists. In *Proceedings of Workshop on Scientific Cloud Computing, Washington, DC USA, June 2017 (ScienceCloud'17),* 8 pages.
DOI: http://dx.doi.org/10.1145/3086567.3086570

1 INTRODUCTION

Software as a service (SaaS) seeks to overcome challenges associated with software installation and user experience by increasing the degree of automation in software delivery. With SaaS, software is operated on the user's behalf by a SaaS provider. Users then access the software over the network, often from a web browser. They need not install, configure, nor update any software, and software providers only have to support a single software version, which they can update at any time to correct errors or add features. SaaS providers typically use techniques such as replication to achieve high reliability, so that the service is not liable to be canceled by one single error. SaaS advocates argue that this approach reduces both complexity for software consumers and costs for providers. Subscription-based payment schemes further reduce friction for consumers and enable providers to scale delivery with demand. Thousands of commercial SaaS providers now operate software in this manner.

We first dissect why it is that SaaS is conventionally defined as both a technology and business model. We then review approaches to on-demand software in science, including the popular concept of a science gateway [27], a form of remote software access that has primarily targeted research supercomputer systems for computation. We then look at more cloud native versions of the science gateway SaaS concept, using two examples to illustrate how SaaS solutions are developed and operated. The first, is the Globus research data management service, that provides a variety of data management capabilities to the community. The second, Globus Genomics, provides on-demand access to bioinformatics pipelines. Both systems build on Amazon cloud services, leveraging a smorgasbord of Amazon and Globus services. A key message presented in this paper is that leveraging cloud platform services can facilitate the creation of SaaS offerings that are reliable, secure, and scalable in terms of the number of users supported and amount of service delivered. Finally, we describe the operations model used to manage Globus services, focusing specifically on the many systems and processes used to ensure high availability and good performance.

2 SAAS

Gartner defines SaaS as software that is "owned, delivered, and managed remotely by [a provider who] delivers software based on one set of common code and data definitions that is consumed in a one-to-many model by all contracted customers at any time, on a pay-for-use basis or as a subscription" [12].

This definition is as much about users' view and means of acquiring the software as it is about technology. From a technology perspective, it speaks to a delivery model in which the provider runs a single version of the software, deployed so as to permit access over the Internet with interfaces suited both for humans (using web browsers) and programmatic access (via HTTP-based Application Programming Interfaces (APIs)). Any SaaS-based offering must be *multitenant*, that is multiple consumers can access the software at the same time without interfering with each other. Such multitenancy insures the most fundamental benefit of the SaaS model which is the ease and expediency with which new users can begin using the software. If users are not able to share the common installation, there is considerable friction involved with each new user or usage. From a user's perspective SaaS delivery speaks to software that consumers do not buy, as they would a home appliance, but

ScienceCloud'17, June 26, 2017, *Washington, DC USA*

DOI: http://dx.doi.org/10.1145/3086567.3086570

instead pay per use, as they would a movie online, or subscribe to, as they would a newspaper.

This juxtaposition of technology and business model may seem odd, but in fact it is central to the success of SaaS, in industry at least. (The fact that some SaaS providers use advertising revenue rather than subscriptions to cover their costs does not change the essential economics.) In brief, the centralized operation model has allowed SaaS providers to slash per-user costs relative to software distributed via traditional channels, because there is, for example, no longer a need to support multiple computer architectures and versions. It has also greatly reduced barriers to access: most SaaS software is accessible to anyone with a Web browser, in contrast to much enterprise software that might require specialized hardware and expertise to install and run. These two factors mean that SaaS providers can deliver software to many more people at far lower costs than were previously possible: literally cents on the dollar.

While the cost of serving each new customer may be low, it is not zero. Furthermore, the upfront cost of establishing and operating a SaaS system involves significant fixed costs. (For example, 24x7 monitoring to ensure high availability.) The SaaS industry has determined that pay-per-use or subscription-based payment models are the best way to recoup these costs. Such approaches provide a low barrier to entry (anyone with a credit card can access a service) and mean that revenue scales linearly with usage. Thus, many users and per-user payments make SaaS sustainable by providing positive returns to scale: more users means more income that can pay for the scaled-up operations and/or reduce subscription charges, encouraging yet broader adoption.

2.1 SaaS architecture

In general, the goal of a SaaS architect is to create a system that can deliver powerful capabilities to many customers at low cost and with high reliability, security, and performance. This overarching goal allows for many tradeoffs: for example, between capabilities and reliability, or between optimizing for base cost or per-user cost. Nevertheless, some basic principles can be identified.

The most sophisticated SaaS systems are often architected using a *microservice* architecture [21], in which state is maintained in one or more persistent, replicated storage services, and computation is performed in short-lived, stateless services that can be rerun if necessary. This architecture provides for a high degree of fault tolerance and also facilitates scaling: more virtual machines can be allocated dynamically as load increases.

Of course, there are many other decisions to be made when designing a SaaS system. For example, SaaS that executes a particular piece of software (e.g., a scientific model) can be implemented in one of several ways: a web form could be deployed and used to execute the software on a workstation; the software could be adapted such that it can be instantiated on a cloud-hosted virtual machine instance for each request; or each of the core components of the software could be deployed into separate cloud instances with user requests allocated via a management service. Clearly, the first approach will not scale as usage increases. The second approach may be costly and unreliable as individual instances are used for each request. If an error occurs a new instance must be started and the work must be restarted from the beginning. The third approach can improve cost, speed, and reliability, however it requires significant development effort up front.

2.2 SaaS and science

The scientific community has a long history of providing online access to software. After all, the original motivation for the ARPANET, precursor to today's Internet, was to enable remote access to scarce computers [26]. With the advent of high-speed networks followed by the World Wide Web, many such experiments were conducted [11, 22]. One early system, the Network Enabled Optimization Server (NEOS), has been in operation for more than 20 years [9], solving optimization problems delivered via email or the web.

The term "science gateways" has become increasingly often used to denote a system that provides online access to scientific software [27]. In general, a science gateway is a (typically web) portal that allows users to configure and invoke scientific applications, often on supercomputers, providing a convenient *gateway* to otherwise hard-to-access computers and software.

The impact of such systems on science has been considerable. For example, the MG-RAST metagenomics analysis service, which provides online analysis of genetic material in environmental samples [19], has more than 22,000 registered users as of 2017, who have collectively uploaded for analysis some 280,000 metagenomes containing more than 10^{14} base pairs. Other successful systems, such as CIPRES [20], which provides access to phylogenetic reconstruction software; CyberGIS [16], for collaborative geospatial problem solving; and nanoHUB [14], which provides access to hundreds of computational simulation codes in nanotechnology, also have thousands of users and correspondingly large impacts on both science and education. A recent survey [15] provides further insights into how and where science gateways are used.

While it is hard to generalize across such a broad spectrum of activities, we can state that the typical science software service has some but not all of the properties of SaaS as commonly understood. First, from a technology perspective: Most such services commonly make a single version of a science application available to many people, and many leverage modern web interface technologies to provide intuitive interactive interfaces. Some also provide REST APIs and even SDKs to permit programmatic access. On the other hand, many are less than fully elastic, due to a need to run on specialized and typically overloaded supercomputers, and few are architected to leverage the power of modern cloud platforms. Thus, they handle modest numbers of users well, but may not scale.

From a business model perspective, few science software systems implement pay-by-use or subscription-based payment schemes. Instead, they typically rely on research grant support and/or allocations of compute and storage resources on scientific computing centers. This lack of a business model can be a subject of concern, because it raises a question about their long-term sustainability (what happens when grants end?) and also hinders scaling (an allocation of supercomputer time may be enough to support 10 concurrent users, but what happens when demand increases to 1000 concurrent users? 10,000?).

We next use two example systems that have each taken a different approach to science SaaS from both technology and business model perspectives: Globus and Globus Genomics.

3 THE GLOBUS RESEARCH DATA MANAGEMENT SERVICE

Globus [6, 10], developed at the University of Chicago over the past seven years, leverages SaaS methods to deliver data management capabilities to the research community. As shown in Figure 1, those capabilities, which include data transfer, sharing, publication, and discovery as well as identity and credential management, are implemented by software running on the Amazon cloud. Globus Connect software deployed on file systems at research institutions and on personal computers enable those systems to participate in the Globus file sharing network. REST APIs and a Python SDK support programmatic access to the various Globus capabilities.

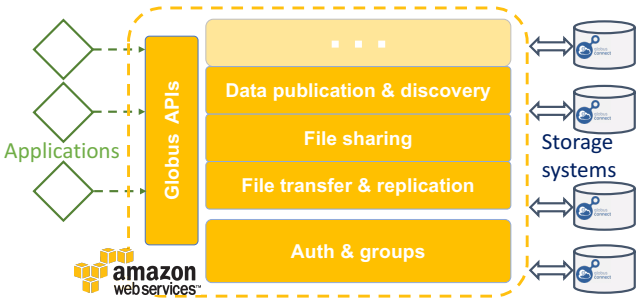

Figure 1: Globus SaaS provides authentication and data transfer, sharing, publication, and discovery capabilities, accessible via APIs (left) and web clients (not shown). Globus Connect software on storage systems enables access to data in many locations.

Globus is popular because researchers and developers of research tools alike can hand off to the Globus service responsibility for otherwise time-consuming tasks, such as babysitting file transfers. For example, consider a researcher who wants to transfer data from site *A* to *B*. With Globus, the researcher can simply make a request to the cloud-hosted service, via API or web interface. The Globus service then handles user authentication, negotiation of access at sites *A* and *B*, configuration of the transfer, and monitoring and control of the transfer activity.

As a SaaS offering, many important projects depend on Globus for authentication, authorization, data access, and other purposes. Thus, high availability is essential, and the Globus implementation leverages public cloud services to replicate state in multiple locations, operate redundant servers with dynamic failover, monitor service status, and so forth. Table 1 provides a partial list of the Amazon services used by Globus.

3.1 Globus service architecture

The Globus SaaS is broken down into logical units or *microservices*. Each service comprises three key components: a **REST API**, a set of one or more **backend task workers**, and a **persistence layer**. Additional components may be utilized by some services, and some components may be co-located to save cost or complexity. Having this common breakdown, and exposing the services to one another only via their REST APIs, provides several key properties that allow different parts of the SaaS to scale independently of one another.

Table 1: Some of the Amazon cloud services used in the Globus SaaS implementation.

Service	Use made by Globus
EC2	Provide high availability instances of Globus services; serve web APIs; run background tasks; internal infrastructure
RDS	Store Globus service state with high availability and durability
DynamoDB	Store Globus service state with high availability and durability
VPC	Establish private Amazon cloud with secure virtual network
ELB	Direct client requests to an available service instance
S3	Store state of in-progress tasks, service data backups, static web content
IAM	Manage access to Amazon resources within Globus
CloudWatch	Monitor the status of Globus resources
SNS	Send notifications to Globus staff
SES	Deliver email to users

Globus servers exposing REST APIs are typically deployed on EC2 instances. All logic used to handle REST API requests is performed synchronously: any asynchronous or long-term activity requested of a service is handled by creating records of the desired activity in the persistence layer. The REST API handlers do not wait for these actions to be completed, but simply register the desired activity in persistent storage and terminate. Further processing is then handled by the backend task workers, which either poll the persistence layer or are notified by API workers. This approach gives the REST API instances the powerful property of being stateless: their contents on disk and in memory are ephemeral, and any REST API instance can process any request. As a result, these microservices can scale up and down trivially, allowing the Globus team to add or remove capacity to serve APIs in direct proportion to observed system load. So long as the backend task workers rely on the persistence layer and are also stateless, they can scale up or down just as easily.

Globus' microservices rely heavily on one another to avoid duplicating functionality. All inter-microservice communication employs the same REST APIs exposed to the public and thus no two Globus services are tightly coupled. Each service can scale, rearrange infrastructure, and alter core service components without impacting one another at all. This separation of concerns is key to Globus operations, and allows improvements to be made safely, easily, and frequently.

The persistence layer is implemented on Amazon storage services, leveraging their replication across availability zones for fault tolerance and creating periodic remote snapshots for disaster recovery. Globus uses, in particular, S3 and the PostgreSQL Relational Database Service (RDS). The various system components are encapsulated in Virtual Private Clouds (VPCs), which allow for the provisioning of logically isolated sections of the Amazon cloud

within which Amazon resources can be launched in a managed virtual network.

In the remainder of this section we describe four Globus microservices and outline their architectures and deployment models.

3.2 Globus Auth

Globus Auth [25] is a low-level service that provides identity brokering capabilities. That is, it mediates the process of user authentication using one of the many identity providers it supports (e.g., institutional identities, ORCID, Google). It also implements an identity federation model in which identities can be linked together and then used as a set. Globus Auth is designed to be leveraged by external applications and services it does so via common interfaces, such as OpenID Connect and OAuth 2. It also provides advanced, delegated authorization models that allow services to act on behalf of other services and users. Globus Auth plays a critical role in the orchestration of all Globus microservices by providing the basis upon which all REST API calls can be authenticated and associated with a user identity.

Globus Auth is implemented as a Python web application that exposes a REST API and template-based web interface, and a stateful database. The core Python application layer represents and manages all resources in the system such as identities, accounts, and clients. The application exposes a REST interfaces to register and configure clients, obtain information about users and tokens, and retrieve linked identities. It also implements standard OAuth 2 and OpenID Connect protocols for integration with third-party applications. The stateless application is deployed across several EC2 instances with an ELB used to distribute requests across the pool of active instances. Each instance connects to a shared stateful PostgreSQL database hosted by RDS. The database is configured to be distributed across availability zones for reliability purposes. The database stores all state including identities, accounts, clients, consents, and identity providers. However, because it acts as an identity broker, Globus Auth does **not** store any user credentials such as passwords.

3.3 Globus Groups

Globus Groups provides a flexible, user-oriented groups model and a collection of specialized workflows related to management of group membership. Originally, Globus Groups was developed as a component of the Globus Nexus service [4]—the service that, prior to Globus Auth, managed all Globus users, profiles, and groups. The service allows groups to be defined with policies regarding visibility, membership requirements, workflows (invitation, acceptance, suspension, etc.), roles, and even the email templates used in workflows. Globus Groups implements a hierarchical group model, in which child groups inherent the policies of their parents.

Globus Groups is implemented as a stateless Python application using the Pyramid web framework. The application exposes only a REST API which may be consumed by external users and the Globus web application. The REST API exposes resources (e.g., groups, memberships) and the interfaces to manipulate these resources. An overarching authorization model governs who may access resources. The Groups Python application is deployed on a collection of EC2 instances behind an ELB. Requests are distributed across the active pool of (typically three) deployed instances. All state is stored in a PostgreSQL RDS instance that is replicated across availability zones to provide high availability and reliability. The groups model is serialized into a relational model in which typed nodes (users, groups) and relationships (e.g., "is member") are encoded. Globus Groups leverages other AWS services including SES for email and SNS for internal notifications.

3.4 Globus Transfer

Globus Transfer [2] provides high-performance file transfer and synchronization. It also facilitates the secure sharing of data between Globus users. Globus Transfer aims to simplify the process of moving large amounts of data between two storage systems. As such, it manages the difficult aspects of data transfer on behalf of users. That is, it allows users to easily start and manage transfers between endpoints, while automatically tuning parameters to maximize bandwidth usage, managing security configurations, providing automatic fault recovery, and notifying users of completion and problems. It relies on Globus Connect software deployed on storage systems ranging from supercomputing centers to PCs to orchestrate the secure third party transfer of data. Data does not pass through the Globus Transfer service, rather a secure data channel is created between endpoints. The Globus service manages the flow of data over this channel.

Globus Transfer is implemented as a collection of Python applications deployed across several EC2 instances. One instance implements two transfer interfaces (REST API and interactive command line) while another instance hosts an elastic pool of transfer workers (processes that manage individual transfers). Rather than utilize an existing web framework, the Transfer service implements an optimized library for parsing and managing requests. It also leverages a custom networking library (written in C) to manage data transfers using the GridFTP protocol [1]. Separate processes are used to isolate errors that may occur in the networking library.

Globus Transfer manages a large amount of stateful data including not only historical and ongoing transfers, but also individual, fine-grain events within a transfer such as performance markers, error conditions, and checksums. Given the amount of state to be stored, Globus Transfer employs a multi-tier storage model that uses both S3 and RDS. The PostgreSQL RDS instance is deployed across availability zones and manages high level transfer information (e.g., user, endpoints, and aggregate statistics). SQLAlchemy table definition, expression language, and database interfaces are used to access the database. S3 is used to store detailed transfer information such as file lists and performance markers. Detailed transfer information is cached on the service's file system as needed. SES is used for all email communication including transfer status, summaries, and error conditions. Globus Transfer uses Globus Auth for authentication and Globus Groups for managing group-based authorization, roles, and sharing.

3.5 Globus Data Publication

Globus Data Publication [5] supports user-managed publication of arbitrarily large amounts of data. Administrators create policy-controlled "collections" by specifying where data will be stored, what metadata should be collected, what form of persistent identifier will be applied, what curation workflow will be used, and

who can submit, curate, and access published data. The Globus Data Publication service enforces these policies, implementing a workflow that guides users through the publication process. Globus Data Publication is used as the underlying service for the Materials Data Facility (MDF) [3].

Globus Data Publication is implemented on top of the DSpace institutional repository system [23]. The core application functionality and REST APIs are implemented as a collection of Java Servlets. The web interface is generated by JavaServer Pages (JSP). The modifications to DSpace replace built-in functionality with use of other Globus microservices: user and group management are replaced with Globus Auth and Globus Groups, respectively; and data management is handled using Globus Transfer including the enforcement of data access policies (ACLs) for data placed into a collection. In this way, Globus Data Publication is an exemplar for other services which may build upon Globus as a platform: it uses only services and APIs which would be accessible to any other service which would be designed to build upon Globus.

Given the DSpace-based architecture of this service, it is not possible to distribute the core service across many EC2 instances. Instead, the service is deployed to a moderately-size EC2 instance with the knowledge that automated deployment scripts can be used to deploy a new instance of the software within minutes if the need for scale or reliability arises. Custom metadata schemas and input forms are stored in a versioned GitHub repository which are synchronized to the ephemeral storage of the host EC2 instance. All state is stored in a PostgreSQL RDS instance which is replicated across availability zones. A copy of all published metadata is stored in S3. These metadata files are also copied to the publication's endpoint upon completion of the workflow.

3.6 Globus as SaaS

Globus implements a traditional SaaS model both from a technology and a business model perspective. As described, Globus is comprised a set of microservices, each providing capabilities to users and to one another. Each is provisioned for scale and reliability ensuring that no single microservice hinders the operation of another. For ubiquitous ease-of-use, a web application presents a unified interface via which all microservices are used in concert.

From a business model perspective, Globus implements a common "freemium" model in which the majority of Globus capabilities are provided freely for any non-profit research and educational purposes while advanced capabilities are offered to those with a subscription. Subscriptions provide access to enhanced features such as file sharing and data publication, customized interfaces such as a branded web presence and support for alternative identity providers, support for specialized storage systems via premium storage connectors, advanced management features for system administrators, and enhanced support from the Globus team. These subscriptions pay for not only the AWS infrastructure on which Globus is hosted but also the development and operations of the services. Unlike traditional SaaS providers, Globus subscriptions are not directly available to researchers, rather they are available to research institutions and commercial entities, who in turn pass on advanced capabilities to their users. Pricing is based on the size of the institution and the number of users and level of usage.

4 GLOBUS GENOMICS BIOINFORMATICS

Globus Genomics [17, 18] is a cloud-hosted software service for the rapid analysis of biomedical, and in particular next generation sequencing (NGS), data. The basic idea is as follows: a customer (individual researcher, laboratory, community) signs up for the service. The Globus Genomics team then establishes a service instance, configured with applications and pipelines specific to the new customer's disciplinary requirements. Access to this instance is managed via a Globus Group. Any authorized user can then sign on to the instance, use its Galaxy interface to select an existing application or pipeline (or create a new pipeline), specify the data to be processed, and launch a computation that processes the specified data with the specified pipeline. Computational results can be maintained within the instance or, alternatively, returned to the user's laboratory for further processing or long-term storage.

By providing research teams with a personal cloud-powered data storage and analysis "virtual computer," Globus Genomics allows researchers to perform fully automated analysis of large genetic sequence datasets from a web browser, without any need for software installation or indeed any expertise in cloud or parallel computing. In one common use case, a researcher sends a biological sample to a commercial sequencing provider, has the resulting data communicated over the network to cloud storage (e.g., Amazon S3); and then accesses and analyzes the data by an analysis pipeline running within Globus Genomics.

4.1 Architecture and implementation

As shown in Figure 2, the Globus Genomics implementation comprises six components, all deployed on a single Amazon EC2 instance: Galaxy and web server for workflow management and user interface; HTCondor and Elastic Provisioner for computation management; and Globus Connect Server (GCS) and shared file system for data management. These services themselves engage other cloud services, notably Globus identity and data management services for user authentication and to initiate data transfers; Amazon EC2 to create and delete the virtual machine instances on which user computations run; and RDS and Elastic File Service (EFS) for storing user data that needs to persist over time. We describe each element in turn.

The **Galaxy** system [13], a Python-based web application, supports construction and execution of workflows. A user signs on to the cloud-hosted Galaxy instance. They can then select an existing workflow or create a new one, identify data to be processed, and launch computational tasks. The **web server**, an integral part of the Galaxy system, serves the Galaxy user interface to Globus Genomics users. Users need only a web browser to access Globus Genomics capabilities.

The **HTCondor** system [24] maintains a queue of tasks to be executed, dispatches tasks from that queue to available EC2 worker nodes, and monitors those tasks for successful completion or failure.

The **Elastic Provisioner** [7] manages a pool of worker nodes, allocating nodes of the right type for the tasks that are to be executed, increasing the number of nodes when the HTCondor queue becomes long, and de-allocating nodes when there is little or no work to do. The elastic provisioner is designed to use spot instances

where possible, in order to reduce costs. It uses resource profiles [8] to determine which instance types are suitable for a given task.

Globus Connect Server (GCS), implements the protocols that the Globus cloud service uses to manage data transfers between the service and data sources/destinations.

The **shared file system** uses the Network File System (NFS) to provide a uniform file system name space across the manager and worker nodes. This mechanism simplifies the execution of Galaxy workflows, which are designed to run in a shared file system environment.

4.2 Globus Genomics as SaaS

Globus Genomics has many SaaS attributes; however it does not adhere to several properties of the SaaS model. From a technology perspective, it is accessible remotely over the network, runs a single copy of its software (Galaxy, the various genomics analysis tools and pipelines), and leverages cloud platform services provided by Amazon and Globus for scalability and to simplify its implementation.

Globus Genomics is not multitenant: it creates a separate instance of the system (the manager node in Figure 2) for each customer, rather than having one scalable instance serving all customers. This characteristic of the Globus Genomics system is not a problem for users: indeed, a single-tenant architecture may appear preferable to some due to (at least an appearance of) increased security and the clean, transparent billing for Amazon cloud charges that it allows. However, single tenancy increases costs for the Globus Genomics team over time, as each new customer requires the instantiation of a complete new Globus Genomics configuration, increasing Amazon usage and other operations costs, and different customers cannot share compute instances.

Globus Genomics also relies on a single instance of the manager node in each deployment. This single instance creates a single point of failure and impacts availability and reliability. The Globus Genomics team can detect such failures and restart the service, but the failure is not transparent to users.

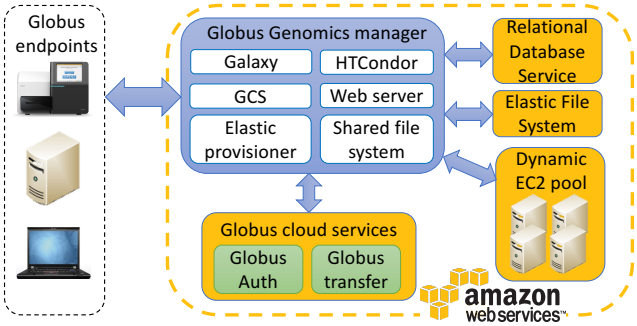

Figure 2: The Globus Genomics system dispatches tasks to a dynamically instantiated HTCondor pool, with virtual nodes added and removed by the elastic provisioner in response to changing load.

From a business model perspective, Globus Genomics also has SaaS characteristics in that its use is supported by a subscription and pay-for-use billing model. A lab or individual user signs up for a Globus Genomics subscription that covers the base cost of operating their private instance. As part of the configuration of a Globus Genomics instance, Amazon account details are provided so that resources consumed by any users granted access to that instance can be charged to that account. However, unlike other SaaS services, these costs are not transparent to users. Rather than being charged for high level operations (e.g., the analysis of a genome) instead they are charged for the resources and services used by the Globus Genomics instance.

5 OPERATIONS

One of the unique requirements of delivering SaaS is the need to host and operate services with high levels of reliability and availability. Globus accomplishes these goals through specialized software, dedicated support personnel, and well-defined policies and practices. To reduce overheads and maintain uniformity Globus leverages a number of common practices across all of its services. In this way, operational and infrastructural improvements are applied across the entire product offering and their effects are amplified. Internal components that are shared across multiple or all Globus services include:

- Continuous integration and continuous delivery pipelines
- Service health and performance monitoring
- Security monitoring and intrusion detection
- Log aggregation
- Configuration management
- Backups and disaster recovery

5.1 Continuous integration and delivery

Globus software and services are developed with a suite of automated tests. These tests are used as part of a Continuous Delivery pipeline to guarantee safety prior to any service update. Before any code is considered for release, it must pass the team's internal review standard, including successful completion of a rigorous test suite. The process, driven largely off of Continuous Integration (CI) servers running in EC2, makes it impossible to deploy code which does not pass its tests.

Once code has passed internal review, the CI service will usually load deployable build results, "build artifacts," (in some cases Amazon Machine Images) into S3. At that point developers and administrators can download and deploy the build artifacts into preproduction environments for manual testing or acceptance testing prior to release. Once the build passes the testing phase, it can be "promoted" by the CI servers – the build artifact is copied within S3 to a location for production deployment.

Globus operates several deployment "stacks" that emulate a production environment. These stacks allow developers and test engineers to fully evaluate software before it is released. Each stack includes a complete instance of each microservice, enabling modifications which impact other microservices to be evaluated collectively. At a minimum, releases must pass through an environment for testing against production versions of all other services, as well as a clone of the production environment in which a dry run of deployment activities and final verification takes place. This final check also serves to protect against rare bugs introduced or hidden by automated or iterative testing of release candidates, in

which the state of the testing environment may be corrupted by faulty release candidates. Each environment has fully independent resources including databases, credentials, and servers, to ensure that testing activities cannot impact production services.

5.2 Uptime and health monitoring

There are numerous activities that are important to achieving high availability, but few are as critical as actively monitoring the health of Globus servers and services.

Using a number of tools, including Nagios, AWS CloudWatch Alarms, cron, and AWS CloudWatch Metrics, the Globus Team monitors instances for unusually high resource consumption on servers, tracks metrics on resource consumption, alerts team members to dangerous conditions, and regularly exercises common or essential service features. The collected metrics and health checks range from highly granular checks on individual servers, akin to "was Server X above 85% memory utilization for more than 5 seconds?," to end-to-end tests of service functionality. By way of example, Globus employs custom monitors which regularly simulate Globus Auth logins and which execute and monitor Transfer tasks.

Almost all of these health monitors, many of them custom, feed back into notifications to Globus staff. Leveraging Amazon Simple Notification Service (SNS) and Simple Event Service (SES), text or email alerts are sent to service adminisrators and developers when thresholds are exceeded or healthchecks fail. Careful tuning of the alerting thresholds helps reduce the false negative rate and combats monitoring fatigue, with the ultimate goal of making every alert actionable.

5.3 Security monitoring and log analysis

Security monitoring for Globus is effectively be applied at three layers: the security of the EC2 servers running the services, the security of AWS resources (IAM users, EC2 security groups, VPCs, etc), and the security of applications built on all or part of the Globus Platform. Monitoring network activity poses peculiar problems on EC2, as it is difficult to strategically position Network Intrusion Detection systems and firewalls. Instead, Globus uses tightly specified EC2 Security Groups and focuses primarily on other threat detection techniques.

Host intrusion detection runs throughout Globus infrastructure, monitoring filesystem changes, system logs, and alerting administrators to potential threats. In addition to monitoring the EC2 servers used to host services, Globus employs monitors which check that AWS security resources, such as security groups and IAM roles, conform to internal security policies.

Finally, all Globus backend components are wired into centralized logging systems like CloudWatch Logs and rsyslog, which allow bulk log collection and analysis for any attempts to compromise Globus resources.

5.4 Configuration management

Globus uses two methods for Configuration Management. The first is a custom toolchain combined with EC2 images (AMIs). AMIs are built, tested, and shipped as units of immutable Infrastructure.

However, this approach is not suitable in all scenarios. Globus also leverages Chef for configuration management. Chef ensures that all EC2 instances for a given service have, and keep, uniform configuration. Perhaps most importantly, Chef provides a declarative, and extensible, language for provisioning new servers, along with discoverability features based on its search capabilities. Service nodes can autodetect one another and form clusters based on Chef search results. As much as possible, Chef components are developed as modular components so that they may be broken apart, reused, or layered as the need arises.

Chef also allows Globus to employ Chef Vault for highly granular management of secret data. Chef Vault uses the existing credentials that servers use to secure access to Chef to encrypt shared secrets, allowing per-server access control. Plans to allow Chef Vault to use AWS Key Management Service (KMS) would allow even more varied controls on secrets like database passwords and signing keys.

5.5 Backups and disaster recovery

Every Globus component with a notion of data durability performs regular backups. Depending on the service, and the persistence layer used, Globus relies on EBS Snapshots, RDS Snapshots, filesystem backups stored in S3, and a myriad of cron-scheduled tasks and monitors to ensure that once a transaction with the service is complete, its results are persisted even if the service fails and needs to be restored.

However, these backups are often, of necessity, accessible from the systems on which or from whence they are made. That is, primiary backups are stored within AWS, and even sometimes co-located with the components they are backing up. To mitigate the risk of entire AWS data center failure, Globus employs the industry standard practice of performing secondary off-site backups. Using separate AWS accounts, strongly isolated credentials, and different AWS regions, many of the benefits of secondary backups can be achieved on the same types of Amazon infrastructure that run Globus services.

5.6 Availability

The stringent development and operations processes employed by Globus is done to provide high availability and reliability. To evaluate the ability of these processes to meet these goals, detailed uptime information (including planned and unplanned downtime) of all Globus services is recorded. Over the past 7 years, Globus has achieved greater than 99.9% availability across all services. In 2016, Globus services experienced a total of 92 minutes of unplanned downtime due to errors, testing failures, and unplanned outages. Individual Globus services in this period achieved the following total (planned and unplanned) availability: Globus Transfer: 99.89%; Globus Auth: 99.98%; Globus Groups: 99.89%; and the Globus web application 99.90%. The primary reasons for unplanned downtime included infrastructure errors (often as the result of AWS outages), configuration errors, and software bugs. In most cases, these issues were resolved quickly by deploying new instances, "hotfixing" code, or "rolling back" deployments to previous states.

6 SUMMARY

Our goal in providing this brief review of SaaS methods is to present a framework for thinking on software as a service and its role in science. True SaaS, realized as cloud-hosted software with support

Table 2: Globus services' use of platform services.

Service	Auth	Transfer	Groups	Publication	Genomics
EC2	X	X	X	X	X
RDS	X	X	X	X	X
VPC	X	X	X	X	X
ELB	X	X	X		
S3	X	X	X	X	
SNS	X	X	X		
SES	X	X	X	X	
EFS					X
Auth		X	X	X	X
Transfer				X	X
Groups		X		X	X

from pay-for-use or subscriptions, can address in a convenient manner the three major challenges of science software, namely usability, scalability, and sustainability. But a certain scale of use is required to justify the costs of multitenant architecture, and not all software will generate the interest and subscription income required to support that scale of use.

Globus has successfully applied the SaaS model for 7 years and continues to add new services. The ease by which new services can be developed and deployed is in part based upon the use of other services, primarily those offered by AWS and Globus (Table 2). Of course, SaaS is not a silver bullet, as evidenced by Globus' deployment and operations processes. Considerable effort is needed to develop and operate services that are reliable, available, and efficient. The approaches used by Globus have resulted in minimal planned and unplanned downtime, and achieved 99.9% availability across Globus services.

ACKNOWLEDGMENTS

We thank Globus subscribers for supporting the operation and development of Globus, and users of Globus services for their continued support. This research was supported in part by NSF grant ACI-1148484 (SciDaaS) and US Department of Energy contract DE-AC02-06CH11357.

REFERENCES

[1] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. 2005. The Globus Striped GridFTP Framework and Server. In *ACM/IEEE Conference on Supercomputing (SC '05)*. IEEE Computer Society, Washington, DC, USA, 54–. DOI: http://dx.doi.org/10.1109/SC.2005.72

[2] Bryce Allen, John Bresnahan, Lisa Childers, Ian Foster, Gopi Kandaswamy, Raj Kettimuthu, Jack Kordas, Mike Link, Stuart Martin, Karl Pickett, and Steven Tuecke. 2012. Software As a Service for Data Scientists. *Commun. ACM* 55, 2 (Feb. 2012), 81–88. DOI: http://dx.doi.org/10.1145/2076450.2076468

[3] B. Blaiszik, K. Chard, J. Pruyne, R. Ananthakrishnan, S. Tuecke, and I. Foster. 2016. The Materials Data Facility: Data Services to Advance Materials Science Research. *Journal of the Minerals, Metals & Materials Society (JOM)* 68, 8 (2016), 2045–2052. DOI: http://dx.doi.org/10.1007/s11837-016-2001-3

[4] Kyle Chard, Mattias Lidman, Brendan McCollam, Josh Bryan, Rachana Ananthakrishnan, Steven Tuecke, and Ian Foster. 2016. Globus Nexus: A Platform-as-a-Service provider of research identity, profile, and group management. *Future Generation Computer Systems* 56 (2016), 571–583. DOI: http://dx.doi.org/10.1016/j.future.2015.09.006

[5] K. Chard, J. Pruyne, B. Blaiszik, R. Ananthakrishnan, S. Tuecke, and I. Foster. 2015. Globus Data Publication as a Service: Lowering Barriers to Reproducible Science. In *11th IEEE International Conference on e-Science*. 401–410. DOI: http://dx.doi.org/10.1109/eScience.2015.68

[6] K. Chard, S. Tuecke, and I. Foster. 2014. Efficient and Secure Transfer, Synchronization, and Sharing of Big Data. *IEEE Cloud Computing* 1, 3 (Sept 2014), 46–55. DOI: http://dx.doi.org/10.1109/MCC.2014.52

[7] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster. 2015. Cost-Aware Cloud Provisioning. In *11th IEEE International Conference on e-Science*. 136–144. DOI: http://dx.doi.org/10.1109/eScience.2015.67

[8] R. Chard, K. Chard, B. Ng, K. Bubendorfer, A. Rodriguez, R. Madduri, and I. Foster. 2016. An Automated Tool Profiling Service for the Cloud. In *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 223–232. DOI: http://dx.doi.org/10.1109/CCGrid.2016.57

[9] Joseph Czyzyk, Michael P Mesnier, and Jorge J Moré. 1998. The NEOS server. *IEEE Computational Science and Engineering* 5, 3 (1998), 68–75.

[10] I. Foster. 2011. Globus Online: Accelerating and Democratizing Science through Cloud-Based Services. *Internet Computing, IEEE* 15, 3 (May 2011), 70–73. DOI: http://dx.doi.org/10.1109/MIC.2011.64

[11] Ian Foster and Carl Kesselman. 2011. The History of the Grid. In *High Performance Computing: From Grids and Clouds to Exascale*. IOS Press, 3–30.

[12] Gartner Research. 2017. Software as a Service (SaaS). http://www.gartner.com/it-glossary/software-as-a-service-saas. (2017). Web site. Accessed: March, 2017.

[13] J Goecks, A Nekrutenko, J Taylor, and The Galaxy Team. 2010. Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol* 11, 8 (2010), R86.

[14] Gerhard Klimeck, Michael McLennan, Sean P Brophy, George B Adams III, and Mark S Lundstrom. 2008. nanohub.org: Advancing education and research in nanotechnology. *Computing in Science & Engineering* 10, 5 (2008), 17–23.

[15] Katherine A Lawrence, Michael Zentner, Nancy Wilkins-Diehr, Julie A Wernert, Marlon Pierce, Suresh Marru, and Scott Michael. 2015. Science gateways today and tomorrow: Positive perspectives of nearly 5000 members of the research community. *Concurrency and Computation: Practice and Experience* 27, 16 (2015), 4252–4268.

[16] Yan Liu, Anand Padmanabhan, and Shaowen Wang. 2015. CyberGIS Gateway for enabling data-rich geospatial research and education. *Concurrency and Computation: Practice and Experience* 27, 2 (2015), 395–407.

[17] R. Madduri, K. Chard, R. Chard, L. Lacinski, A. Rodriguez, D. Sulakhe, D. Kelly, U. Dave, and I. Foster. 2015. The Globus Galaxies Platform: Delivering Science Gateways as a Service. *Concurrency and Computation: Practice and Experience* 27, 16 (2015), 4344–4360.

[18] Ravi K. Madduri, Dinanath Sulakhe, Lukasz Lacinski, Bo Liu, Alex Rodriguez, Kyle Chard, Utpal J. Dave, and Ian T. Foster. 2014. Experiences Building Globus Genomics: A Next-Generation Sequencing Analysis Service using Galaxy, Globus, and Amazon Web Services. *Concurrency and Computation: Practice and Experience* 26, 13 (2014), 2266–2279.

[19] Folker Meyer, Daniel Paarmann, Mark D'Souza, Robert Olson, Elizabeth M Glass, Michael Kubal, Tobias Paczian, A Rodriguez, Rick Stevens, Andreas Wilke, and others. 2008. The metagenomics RAST server–A public resource for the automatic phylogenetic and functional analysis of metagenomes. *BMC bioinformatics* 9, 1 (2008), 386.

[20] Mark A Miller, Wayne Pfeiffer, and Terri Schwartz. 2010. Creating the CIPRES Science Gateway for inference of large phylogenetic trees. In *Gateway Computing Environments Workshop*. 1–8.

[21] Sam Newman. 2015. *Building Microservices* (1st ed.). O'Reilly Media, Inc.

[22] Rick Stevens, Paul Woodward, Tom DeFanti, and Charlie Catlett. 1997. From the I-WAY to the National Technology Grid. *Commun. ACM* 40, 11 (1997), 50–60.

[23] Robert Tansley, Mick Bass, David Stuve, Margret Branschofsky, Daniel Chudnov, Greg McClellan, and MacKenzie Smith. 2003. The DSpace Institutional Digital Repository System: Current Functionality. In *3rd ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL '03)*. IEEE Computer Society, Washington, DC, USA, 87–97. http://dl.acm.org/citation.cfm?id=827140.827151

[24] Douglas Thain, Todd Tannenbaum, and Miron Livny. 2005. Distributed computing in practice: The Condor experience. *Concurrency and computation: practice and experience* 17, 2-4 (2005), 323–356.

[25] S. Tuecke, R. Ananthakrishnan, K. Chard, M. Lidman, B. McCollam, S. Rosen, and I. Foster. 2016. Globus Auth: A research identity and access management platform. In *12th IEEE International Conference on e-Science (e-Science)*. 203–212. DOI: http://dx.doi.org/10.1109/eScience.2016.7870901

[26] M Mitchell Waldrop. 2001. *The Dream Machine: JCR Licklider and the Revolution that Made Computing Personal*. Viking Penguin.

[27] Nancy Wilkins-Diehr, Dennis Gannon, Gerhard Klimeck, Scott Oster, and Sudhakar Pamidighantam. 2008. TeraGrid science gateways and their impact on science. *Computer* 41, 11 (2008).