Sustainability in the Tapis Framework

Joe Stubbs
University of Texas
jstubbs@tacc.utexas.edu

Richard Cardone
University of Texas
rcardone@tacc.utexas.edu

Mike Packard
University of Texas
mpackard@tacc.utexas.edu

Anagha Jamthe
University of Texas
ajamthe@tacc.utexas.edu

Smruti Padhy
University of Texas
spadhy@tacc.utexas.edu

Steve Terry
University of Texas
sterry1@tacc.utexas.edu

Julia Looney
University of Texas
jlooney@tacc.utexas.edu

Joseph Meiring
University of Texas
jmeiringd@tacc.utexas.edu

Steve Black
University of Texas
scblacke@tacc.utexas.edu

Maytal Dahan
University of Texas
maytal@tacc.utexas.edu

Sean Cleveland
University of Hawaii
seanbc@hawaii.edu

Gwen Jacobs
University of Hawaii
gwen@hawaii.edu

Abstract

As more research depends fundamentally on software, sustainability becomes increasingly critical. Nevertheless, despite valiant efforts from a growing number of researchers and practitioners, a basic understanding of best-practices for sustainable software remains elusive. In this paper, we review the specific practices and strategies that have helped to sustain Tapis, a cyberinfastructure project that has been in use for over a decade. The Tapis framework is an open-source, software-as-a-service Application Programming Interface (API) for collaborative, automated, reproducible, computational research which began as the Foundation API for the iPlant Collaborative Project in 2008. Today Tapis is used by tens of thousands of individuals across more than a dozen active projects. This paper describes our multi-faceted approach to sustaining an increasingly complex ecosystem of software, documentation and other digital assets, including both technical and organizational strategies for minimizing the cost of sustainment while maximizing available resources for sustainment activities.

1. Introduction

Increasingly, research relies upon and produces software. In a 2017 survey of members of the US Postdoctoral Association, 95% of individuals who responded used software in their research, and 63% said they could not do their research without software [1]. A study of all Nature papers published between January 1 and March 31, 2016, found that 80% of the papers mentioned software, with an average of 7 distinct software packages per paper [2]. The international study "Charting the Digital Transformation of Science", from 2018, found that nearly 70% of research that resulted in scientific publications produced data and code [3].

The sustainment of such software — the activity of ensuring that the software continues to be available and usable — is therefore essential to the vitality of a significant portion of all research. But while the criticality of this challenge has been recognized by leading institutions and funding agencies, a consensus on techniques and best practices for sustainable software has yet to emerge.

The Tapis API framework is an open-source software system enabling collaborative, automated, reproducible computational research [4]. A hosted, software-as-a-service platform, the main Tapis installation running at the Texas Advanced Computing Center at the University of Texas at Austin currently supports tens of thousands of users across 14 active projects. At the end of 2019, annual usage reached 30 million API requests. But Tapis's usage and adoption has grown substantially over a period of more than 10 years. Through a steady commitment to sustainable software practices, the Tapis project has evolved from its roots as the iPlant Collaborative's Foundation API, first launched in 2008 [5].

Clearly, sustaining software requires resources, and therefore, efforts to make software more sustainable can be organized into two broad categories: 1) efforts that reduce the resources required to sustain the software and 2) efforts that increase the resources available to perform sustainment tasks. Like many academic research and development projects, Tapis has operated on significantly less direct funding than its commercial counterparts. As a result, we have adopted a multi-faceted sustainment approach comprised of a number techniques and practices from both categories 1) and 2). In this paper we present our approach to sustaining the Tapis platform, including concrete examples of both technical and organizational strategies, with the hope of benefiting other projects and contributing to the identification of best practices in software sustainability.

We begin with a review of related efforts to study sustainable software practices in Section 2. Then, in Section 3, we provide details about the history and background of the Tapis project necessary for the rest of the paper. In Section 4 we describe strategies we have used to reduce the cost of sustaining Tapis while in Section 5, we discuss our approaches to growing the Tapis community and increasing the resources available for sustainment efforts. In section 6, we examine data associated with the evolution of Tapis which we argue suggests a temporal correlation between these practices and the overall increased sustainability of Tapis. Finally, in Section 8 we close by summarizing the primary ideas presented for improving sustainability of research software.

2. Related Work

A large and growing body of articles in the current literature study the use of software in research and the challenge of sustainability [6]. In this section, we review some of the work done to date.

2.1. Empirical Studies

A number of articles have taken an empirical approach to studying the properties of sustainable software used in research. For example, in a 2015 paper, the authors conducted a survey of cyberinfrastructure users in which respondents were asked to list projects they identified as successfully sustained [7]. The paper presents the results of in-depth interviews conducted with the project leads of 12 cyberinfrastructure projects. Some high-level themes common across the projects emerged, such as "the use of open-source licenses", "strong, committed, visionary leadership", and "domain experts engaged with developers" are presented.

Another paper from 2017 presents the results of a survey aimed at determining: 1) the extent to which different software quality requirements contribute to a perception of sustainability and 2) the dependencies between four software sustainability dimensions (social, technical, economic, and environmental) [8]. Software qualities such as modifiability, functional correctness, availability, interoperability, security, and freedom from risk, were among those identified as the most crucial. The paper also reported that the technical and economic dimensions are most closely related, followed by technical and environmental.

More recently, a study from 2019 conducted interviews with Research Software Engineers (RSEs) to determine if there is a common understanding of the meaning of sustainable software [9]. A thematic assessment of the data revealed two sustainability dimensions – *intrinsic* and *extrinsic* sustainability — instead of four, as well as a variety of views on how to obtain each.

2.2. Evaluation Frameworks

Another set of efforts involves defining formal metrics for measuring the sustainability of a software package. Nearly 20 years ago, it was argued that common quality measures at the time such as mean time to repair (MTTR) were inadequate for measuring software sustainability and the Weighted Modification Request Days (WMRD) and associated measures were proposed instead ([10]). A 2014 paper proposed that software sustainability measures could be defined analogous to measures previously defined for software dependability, suggesting the following primary measures for sustainability: extensibility, interoperability, maintainability, portability, reusability, scalability, and usability [11]. More recent papers build and expand upon these efforts (see for example, [12] and [13]).

Taken together, these articles and a number of others like them (see for example, [14], [15] and [16]), suggest a field very much in its infancy where the basic concepts are note agreed upon and best practices are still very much in flux. Indeed, it would seem "the concept of software sustainability is complex and multifaceted with any consensus towards a shared definition within the field of software engineering yet to be achieved," as noted in a paper that likened software sustainability to the Tower of Babel [17]. This also suggests that the qualities required to make a software project sustainable may vary substantially from project to project, as remarked in [7].

2.3. Case Studies

The final subclass of prior works, and the one most closely related to the present work, are in-depth studies of specific software systems. We find these papers particularly compelling for their potential to illuminate techniques and strategies that have actually worked in real software projects that have been sustained.

In [18], the authors present the common software architecture leveraged by several systems at the NASA Infrared Processing and Analysis Center (IPAC) at Caltech, and they examine the ways in which the architecture contributes to sustainability. Particular emphasis is placed on meeting the challenge of a rapidly growing collection of astronomical data. A key aspect of the design is a component-based architecture where more than 100 components, each performing one generic function, provide modularity and extensibility. A list of 15 best practices for software sustainability is provided. Even though the IPAC software and Tapis are very different systems from different times ([18] was published in 2011), it is interesting to note the commonalities between them.

Some previous work has considered the software-as-a-service (SaaS) model; in [19], it is argued that in general, the SaaS model has advantages over other models for sustainability. A case study of the Globus software suite also argued that subscriptions generated by the SaaS model have put Globus on a path to long-term sustainability. [20].

3. Tapis: Background and History

In this section we provide an overview of the Tapis project.

3.1. High Level Description and Features

Tapis is an open source, hosted API platform for distributed computation that enables researchers to manage data and execute codes on a wide range of remote systems, from high-speed storage and high-performance computing systems to commodity servers. Tapis's network of microservices interact with a vast array of physical resources on behalf of users: high performance and high throughput computing clusters, file servers and other storage systems, databases, bare metal and virtual servers, etc. The goal of Tapis is to provide a unified, simple to use API enabling teams to accomplish computational and data intensive computing in a secure, scalable, and reproducible way so that domain experts can focus on their research instead of the technology needed to accomplish it.

Chief among all capabilities in Tapis is the ability to execute user-defined codes ("apps") on remote execution resources. In Tapis, each invocation of an app is called a "job", and Tapis is capable of scheduling jobs on a variety of different computing resources, from traditional supercomputers using batch schedulers, to high-throughput compute clusters on cloud systems. Tapis provides APIs for managing data on remote storage servers including high-performance parallel file systems, physical servers and VMs and object stores such as AWS S3 [21], and . Underlying all of the APIs is the Tapis Security Kernel, providing user role and permission access controls along with secure secret storage.

3.2. History and Major Versions

The first major API platform for computational research developed at TACC was the "Foundation API". Built as part of the iPlant Collaborative project and originally released in 2008, Foundation API was leveraged by iPlant's Discovery Environment, a web-based graphical interface [5]. We retrospectively refer to this version of the platform as "v1". Agave was the second major API version (v2), starting in 2014. Agave added multi-tenancy to the Foundation API, as well as some other major features such as OAuth2 support. In 2019, the National Science Foundation funded work for a third major version of the platform called "Tapis". Tapis (v3) makes some fundamental changes to architecture and other aspects of v2, such as security, but it continues to support virtually all of the end-user functions such as data management and job execution. It also adds major new services such as the Streams API for real-time sensor data and it incorporates another TACC project, the Abaco platform, directly as a first-class API, which provides functions-as-a-service based on Docker containers and the Actor Model of Concurrent Computation.

3.3. Usage

Tapis has experienced tremendous growth and success since its initial incarnation as the "Foundation API" for iPlant. In the ensuing twelve years, more than 14 independently funded projects have leveraged the Tapis platform across a wide variety of domains of science and engineering, and many smaller projects interact with Tapis without a formal engagement. In addition to the two major version releases, scores of minor releases have been made over this period in response to community usage and feedback. Table 1 provides high-level usage numbers for the lifetime of the platform since 2015.

4. Reducing the Sustainability Cost

In this section, we describe specific decisions we have made and strategies we have used to reduce the cost of sustaining Tapis. We divide these efforts into three types: Architecture, Open Standards and Technologies,

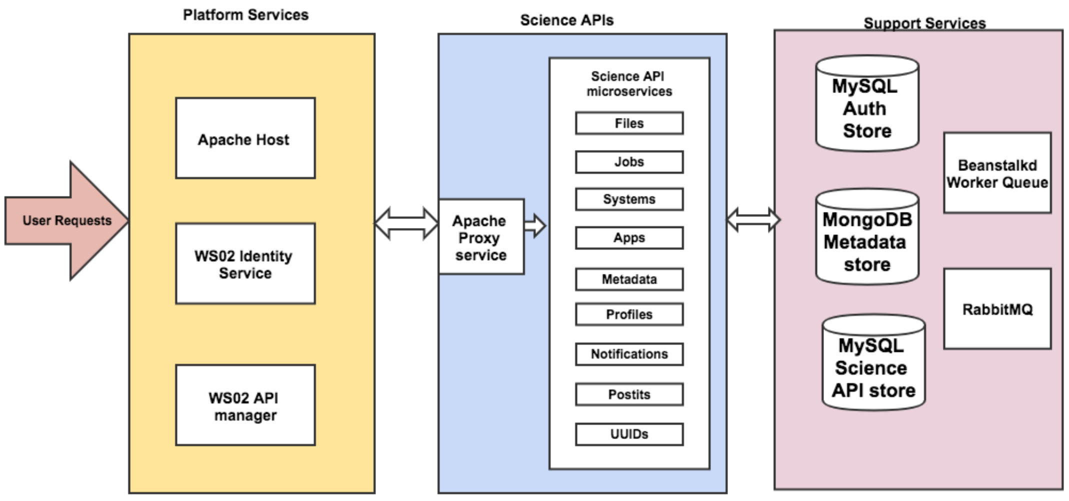

Figure 1. Tapis V2 Architecture
The primary components of the Tapis v2 system; rectangles represent independent processes and arrows indicate communication between processes. Arrow sizes and colors are for readability and carry no other implied meaning.

Metric	Value	Description
Official Installations	3	Total official, independent installations of the platform.
Official Projects	14	Independently funded projects leveraging the platform.
Systems	$8,700$	Number of systems registered.
Apps	$5,150$	Number of unique application codes registered.
Clients	$53,126$	Number of unique OAuth client applications registered registered.
Jobs	285K	Total number of batch jobs run.
Metadata items	442K	Total number of metadata item stored across all projects.
Postits	400K	Total number of API objects shared and retrieved with postits.
Data moved (files/size)	1.15B/3.5 PB	Total number of files transferred and total number of bytes transferred.

Table 1. Summary of aggregate Tapis usage.

though it should be noted that this categorization is more for organizational purposes — clearly, all three types are inherently connected.

4.1. Architecture

The Tapis architecture has evolved substantially over the past decade since the initial release of the Foundation API, with sustainability being one of the primary driving forces. Tapis uses a microservices architecture where independent components, or services, run as standalone programs. The evolution to microservices came gradually with each major version of the platform.

In version one, all primary services ran in a single Java Virtual Machine (JVM), including authentication services. One source code base and build produced a single Java artifact, which was deployed to a server or virtual machine (VM). A single developer contributed the vast majority of the code for version one.

For version two, the identity and authentication services were split from the "core" services and a more distributed architecture was used as shown in Figure 1. The core services, including the Systems, Files, Apps, Jobs and Meta APIs, each used a different build producing a different Java artifact, but dependencies existed between the builds. For example, one could not build the Files API artifact without first building the Systems API artifact, the Jobs API build depended on the builds of almost every other core API, etc. The reason for these build dependencies had to do with another architecture choice: even though the services were independent, deployable applications, communication between core services happened by way of direct library calls in Java code. Thus, if the Jobs service needed to invoke a Systems API function, it did so via library calls, not a HTTP request.

Four primary services comprised the identity and authentication services of version 2: an OAuth server, an API gateway, a Profiles API, and a reverse proxy and load balancer to handle TLS negotiation and primary request routing. The identity and authentication services leveraged a mix of monolithic and microservice architecture, with the most of the primary Oauth2 and API gateway functionality bundled as a single monolith and the other functions split across three independent applications. It's worth noting that the authentication services used different persistence (i.e., databases and message queues) than the core services.

These changes allowed more developers to contribute to the code bases of the version two services. At least six individuals (Cardone, Dooley, Kuritz, Padhy, Stubbs, Terry) made substantial code contributions including initial authoring or major refactoring/rewriting of one or more services. Several others made smaller contributions in the form of bug fixes or other improvements.

Version two of the platform supported multiple *tenants*, or logically isolated views of the API objects (e.g., systems, apps, jobs, etc.) for different projects. This architectural choice greatly expanded the platform's usability and has been essential to the growth over the last five to seven years. It also impacted the deployment architecture substantially. First, the core team deployed and managed a version of the authentication services for each tenant. Additionally, several tenants also ran isolated versions of the core services. In total, including the persistence layer of databases and message queues, version 2 ran across over 50 VMs.

The most recent Tapis v3 platform implements a true microservice architecture as shown in Figure 2 where each service: 1) has an independent code base, hosted in its own repository, 2) an independent build and deployment system, and 3) uses network calls over HTTP for communication with other services. As a result, in just nine months of development, we have already seen major development contributions to core components from 11 individuals across two institutions (Black, Cardone, Cleveland, Garcia, Jamthe, Looney, Meiring, Packard, Padhy, Stubbs, Terry).

4.2. Open Standards

Adoption of open standards contribute substantially to reducing sustainability efforts for Tapis. Open standards allow for the use of community-developed libraries, tools and frameworks, enabling core developers to focus on problems more unique to Tapis. In our experience, adoption of open standards involves more subtlety than may appear on the surface, with competing standards and different versions offering trade-offs between features and sustainability costs.

One class of open standards that have been adopted by Tapis are standards that have been in existence for over a decade; these include HTTP, JSON [22], and SSH [23] as well as the Representational State Transfer (REST) architectural style. On the surface, these choices may seem obvious, but each involved balancing more modern standards with sustainability. For example, HTTP 1.1 [24] was chosen over HTTP 2 [25], while REST and JSON were chosen over gRPC [26] and Protocol Buffers [27]. Even with SSH, which Tapis uses to connect to remote storage and computing resources, other options were considered, including a custom protocol over TCP. In all of these cases, sustainability drove the decision. Due to the maturity of these standards, high quality libraries and frameworks exist in virtually all modern programming languages for working with them. Moreover, developers tend to be more familiar with them, and thus, the barrier to contributing is lower.

By contrast, newer standards come with unknowns; e.g., what technologies should be used to persist long-lived HTTP 2 connections across components, and what client-side tooling is available for users? Typically, these unknowns translate into adoptions of more experimental architectures and technologies which in turn tend to be more costly to sustain.

Other standards adopted by Tapis could be seen as more risky. For example, the OAuth2 [28] specification arrived in October, 2012, less than two years before the version two platform initially went to production with its OAuth2 server. Adoption of the OpenAPI v3 [29] standard as a specification language, which first appeared in July of 2017, for defining the Tapis version three API contracts, provides another example. In these cases, core developers performed assessments, including background research and prototype development, to determine if adoption of the standard would lead to a net project benefit.

4.3. Technology

Much like architecture and open standards, technology selection plays a crucial role in the long-term cost of sustaining a software project. However, for an NSF-funded project like Tapis which must be innovative in addition to providing utility, technology selection, including the incorporation of

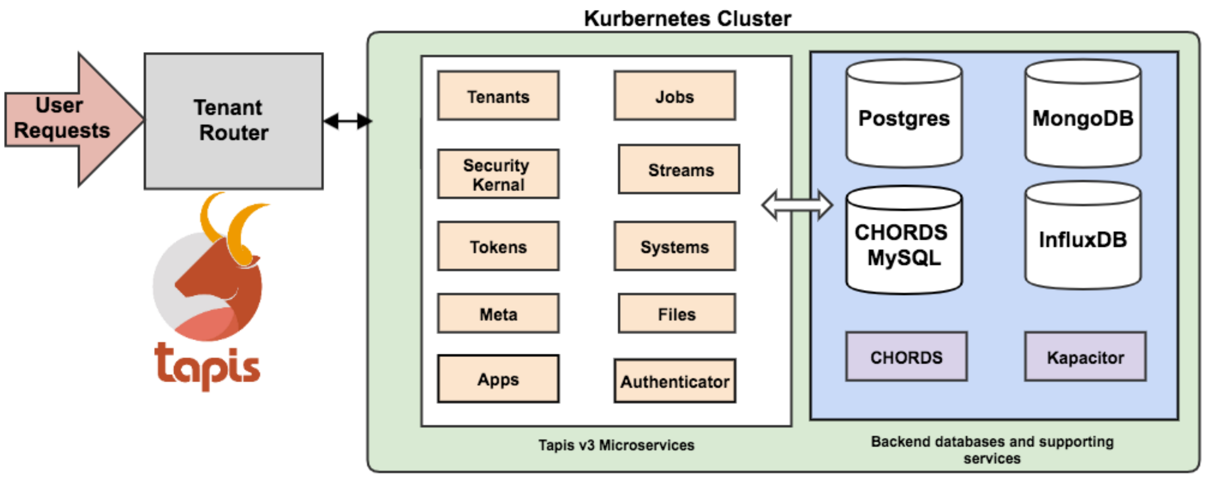

Figure 2. Tapis V3 Architecture
The primary components of the Tapis v3 system; rectangles represent independent processes and arrows indicate communication between processes. Arrow sizes and colors are for readability and carry no other implied meaning.

experimental and cutting-edge technologies, tends to form part of the foundation of the project's intellectual merit claims. Still, a number of technologies have reached a maturity level that the risk associated with adoption seems low compared to the net benefit.

A number of technologies adopted by Tapis clearly lowered the cost of sustainability. These include the primary programming languages (Python and Java), third-party libraries (e.g., Flask, Tomcat), databases and message queues (Postgres [30], MongoDB [31], RabbitMQ [32]), and a variety of open-source components (e.g., reverse proxies such as Apache HTTPd [33]). Sustainability considerations impacted these decisions: for example, the team chose not to write new services with the Golang programming language even though there was a perceived benefit. Separately, a significant effort was made to rewrite a service to replace its use of the Redis [34] database in order to reduce the number of database technologies in use.

Whenever possible, Tapis makes use of free services to minimize overall sustainment cost. Source code repositories are hosted for free on github [35], which also provides issue tracking and a space for documentation such as change logs for released versions, local development instructions, etc. We use github pages for hosting fully static applications, including our interactive Live Docs [36] application based on our OpenAPI definition files and a full-featured remote file browser application for demonstrating the utility of the Systems and Files APIs.

Tapis invests significantly in continuous integration, including an on-premise deployment of the open-source Jenkins server, automated build and deployment scripts written in Ansible [37], and a suite of integration tests for each API. While not without cost, continuous integration allows for faster development while enabling more individuals to contribute without risking quality of service, which is critical to long-term usability and sustainability.

High-quality documentation is essential for growing the user base of a project (see Section 5). Once again, Tapis uses a number of free services and open-source software to minimize the cost of maintaining documentation. We use the Sphinx [38] documentation engine which generates high-quality documentation products in multiple formats, including HTTP and pdf, from restructured-text (RST) source code. RST is well-documented and easy to write for developers as well as support staff. We host the RST source code for each documentation site on github and the Sphinx-generated documentation products are hosted (again for free) on readthedocs.org. In fact, commits to the master branch on the github repository automatically trigger a build of the documentation on readthdocs.org for most of our projects, meaning that the cost of maintaining our documentation sites is near zero beyond the cost to write the RST. Dozens of individuals from several institutions have contributed to our various documentation projects for Tapis.

The impact of other technology selections on Tapis sustainment are less clear. For example, Tapis adopted Docker containers for packaging and distributing its components in late 2014 when the technology was still young. In the ensuing five years, container technology in general, and Docker in particular, experienced substantial adoption and features, performance and stability improved dramatically. It could be argued that had Tapis delayed its adoption of container technology, the costs of sustaining the platform would have been reduced. On the other hand, the Tapis core team developed expertise in containers which led to other advantages. Similar lines of reasoning apply to the adoption of Kubernetes in Tapis version three. In the end, the benefits of adopting cutting-edge technologies must be balanced against the risks and potential sustainability costs. This balancing act involves a mix of careful analysis and old-fashioned luck, and no two projects are likely to make the same sets of choices.

4.4. Technical Debt

A critical aspect of software sustainability involves the proper management of *technical debt* — additional effort that will be required in the future resulting from choices made to simplify an implementation in the present. In research software, often times a project must try to build a new feature or use a new technology without knowing how well it will work, how many people will use it, etc. Thus, some technical debt, taken on at the right time, can be useful. However, technical debt makes it harder to add new features in the future, and too much technical debt can be insurmountable.

Beginning in late 2016, as usage started to increase, we began to see reliability and scalability issues with our Jobs service. We decided to pay down a significant technical debt by re-implementing the Jobs service with a new architecture. Though not a complete rewrite, major structural changes were made and the project took over two years to complete. Along the way, substantial updates to the Files API and other services were also made, including updating the Java runtime environment and many third-party libraries. The effort came at the expense of new features, but the improvements resulted in an immensely more serviceable and maintainable platform. Explosive growth followed, and in the end, the decision appears to have been the right one.

5. Increasing Sustainability Resources

Sustainment of Tapis would not be possible without the enormous contributions of our collaborators and the greater community. In many ways, the efforts to grow the Tapis community impact sustainability of the platform as much as any of the technical or architecture decisions described in the previous section.

5.1. Groups Building with Tapis

The Tapis project has invested in aligning itself closely with several other groups of scientists and developers at the Texas Advanced Computing Center. These groups include the Life Sciences Computing group, the Web and Mobile Applications group, the Data Intensive Computing group, and the Advanced Computing Systems group. These individuals bring use cases and requirements that shape and prioritize the next features and improvements to Tapis. They attend regular requirements sessions, participate in early/beta testing sessions, and even serve as Co-PIs or senior personnel on Tapis grants. In turn, Tapis core team members serve in advisory roles on projects led by these other groups.

The contributions from these groups make Tapis a substantially more compelling platform, and the projects they lead bring the majority of users to Tapis. For example, many users interact with Tapis through one of a number of science gateways developed by the Web and Mobile Applications group. The Life Sciences Computing Group has developed a large number of Tapis applications for bioinformatics, including a number of containerized applications, and has contributed substantially to Tapis developer tooling, including the command-line interface (CLI) [39], the Python Software Development Kit (SDK) [40], and much of the documentation. The Data Intensive Computing Group has collaborated on a number of projects involving large datasets, including helping to build and administer highly scalable databases and develop machine learning applications. The Advanced Computing Systems group advised Tapis core developers on architecture and access patterns for high performance computing and storage resources.

5.2. Partner Institutions Deploying Tapis

As the architecture of version two of the platform became significantly more complex and distributed, the project invested a substantial effort in automating the deployment and management of Tapis software components. Docker container images for packaging and distribution combined with Ansible scripts for server configuration and management were leveraged. While the project itself benefited directly from these efforts, an equally important outcome was that the University of Hawaii Manoa and Centers for Disease Control and Prevention deployed standalone Tapis instances using the deployment tools.

The deployments at each additional institution resulted in substantial improvements to Tapis that have reduced the long-term sustainment cost. First, both groups have made direct code contributions to the core services as well as the deployment tooling. Perhaps more importantly, their involvement in Tapis has shaped our approach to a number of changes to the platform. Knowing that non-Tapis experts would need to be able to deploy and operate the platform required the project team to develop mature practices with respect to documentation and distribution of releases, error messages and logging with better diagnostic information, and to prioritize repeatable maintenance procedures.

5.3. Students

For the past several years, core members of the Tapis project have supervised undergraduate students as part of software internships and research experiences for undergraduates. Former Tapis students have written documentation and have contributed code to the primary services. They have also conducted performance studies and other research that has resulted in publications.

In addition to these tangible contributions, students provide excellent benchmarks for determining how easy it is to contribute to a project. They often lack any significant experience with common engineering tools such as build systems and interactive development environments (IDEs). Every day counts when a student has less than three months over the course of the summer, working part-time, to make a significant contribution. A variety of Tapis processes and documentation — from developer on-boarding and quality quick-start guides, to reviewing PRs — have been streamlined and improved as a result of working with students. These outcomes directly impact long-term sustainability of the project.

5.4. Support and Outreach by Core Team

The core team performs a number of important activities that help grow the Tapis community. Tapis team members regularly present workshops and tutorials on the Tapis API at conferences such as Practice & Experience in Advanced Research Computing (PEARC) and Gateways, and at TACC events such as the annual User Meeting, TACCster, and the TACC Summer Institute series. The Tapis team has also been invited to present at various institutions such as the Hawaii Data Science Institute at the University of Hawaii and the Centers for Disease Control and Prevention. Additionally, individuals present webinars on platforms such as the Natural Hazards Engineering Research Infrastructure (NHERI) and the Science Gateways Community Institute (SGCI) Webinar series.

Finally, the Tapis user support system is crucial to community engagement and growth. Dedicated support staff monitor the project's Slack channels and triage user support tickets. Additionally, staff monitor a "failed jobs" report that runs hourly. The vast majority of jobs fail due to user error, but the report can help identify users who are struggling to get an application to run correctly. Support staff can then reach out to the users — on our Slack channel [41], for example, if they are members — and offer assistance. For more involved issues, such as questions about solutions architecture or bugs, support staff bring the issues to Tapis core engineers.

6. Analysis

In this section, we present data describing the evolution of Tapis with respect to: (i) total usage and cost and (ii) total resources for sustainability, and we compare these data to the dates of adoption of practices described in previous sections, that, in our opinion, were the most important for improving sustainability. While obviously not a formal, controlled experiment, we believe these data suggest a correlation between adoption of these practices and an overall improvement in the sustainability level of Tapis. For the sake of an equitable comparison and because historical data from before 2014 were not as abundantly available, we focus primarily on the years 2014 to present.

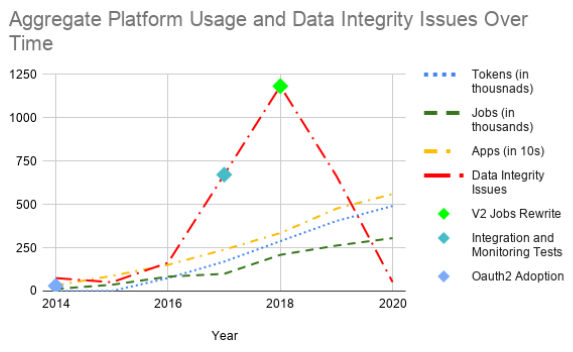

Figure 3. Aggregate Platform Usage and Data Integrity Issues over Time
The figure above shows aggregate Tapis usage has increased across multiple dimensions, but data integrity issues have been decreasing significantly since 2019.

In figure 3, we present the total aggregate OAuth2 tokens (in thousands), aggregate jobs (in thousands) and aggregate apps created in the platform as well as the total annual data integrity issues that required manual intervention over time. While platform usage consistently increased across all dimensions, data integrity issues decreased starting in 2019. Though many factors clearly contributed to this result, we generally attribute it to the increased use of integration and monitoring tests and our decisions to pay down technical debt — in particular, the effort to re-implement the Jobs service in a new architecture.

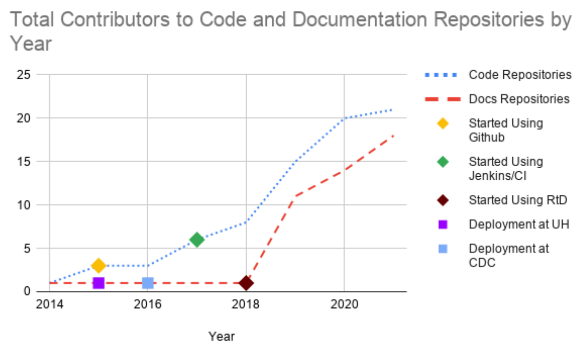

Figure 4. Contributors to Tapis Repositories over Time
The figure above charts the total number of individuals whose first commit to a Tapis repository was in a given year.

In figure 4, we show the total number of individuals who have contributed to the Tapis source code and documentation repositories over time. In general, the numbers of contributors has grown substantially over time, but clearly we begin to see significant gains in 2018. For documentation contributions, a major factor (if not the major factor) was moving away from a custom documentation site written in PHP to using Sphix and ReadTheDocs. Exactly one individual ever contributed to the custom PHP documentation site; however, by July of 2020 we have had nearly 20 contributors to our Sphix/ReadTheDocs documentation repositories. For code contributions, we feel our use of devops automation with Jenkins to provide continuous integration was a major factor. In both cases, additional deployments at UH and CDC were also important.

7. Future Work

In Tapis V3, we expect to improve deployment, monitoring and debugging by leveraging components in the Kubernetes ecosystem, such as Jaeger and ELK/EFK, that make it easier to adopt and operate a microservice based system. In addition, the new architecture allows associate sites to implement a subset of Tapis V3 locally for security or performance reasons. These associate sites still rely on a primary site to manage much of the infrastructure, thus making expansion to new sites more attractive.

8. Conclusion

Looking back on twelve years of Tapis evolution, we see that open source and free-license software and services form a strong foundation for low cost, sustainable systems. Most of the third party software we use, as well as Tapis itself, conforms to open standards that enable components to be swapped in and out with little disruption. We've exercised this flexibility numerous times, for example, by switching code repositories, SQL databases, REST frameworks, documentation generators and logging subsystems.

We continue to reduce sustainability costs by embracing devops, paying down technical debt, and placing bets on strategic technologies. As a small research group responsible for both the development and operation of Tapis, we try to optimize the whole software lifecycle, including design, coding, build, deployment and monitoring. Our build and deployment processes are approaching continuous integration with the use of Jenkins, Kubernetes and automated test suites. We can deploy a new Tapis V3 installation with a single command. We discussed how the V2 Jobs service was not able to keep up with demand. By rewriting it we achieved a level of reliability that significantly increases developer productivity going forward. Finally, as early adopters of container technology and now of Kubernetes, we reap the benefits of industry trends and take advantage of a huge inventory of new, innovative software.

On the other side of the ledger, we increase sustainability resources by fostering communication and by developing partnerships. When we introduced a dedicated support person we not only provided users with a single point of contact, but we drastically reduced the number of interruptions developers experienced. Dedicated support means that slack channels and email inquiries are constantly monitored, and that documentation and tutorials are kept up to date. We hold frequent live tutorials and hands-on training sessions, often in a conference settings where we also present our latest results and interact with our peers. These interactions have led to partnerships, both formal and informal, across institutional boundaries. Our partners have contributed significantly to our code base and to the diversity of application domains to which Tapis is applied.

9. Acknowledgement

This material is based upon work supported by the National Science Foundation Office of Advanced CyberInfrastructure [grant numbers 1931439 and

1931575].

References

[1] U. Nangia and D. K. Katz, "Track 1 paper: Surveying the U.S. National Postdoctoral Association regarding software use and training in research," Zenodo, June 2017. doi: 10.6084/m9.figshare.5328442.

[2] U. Nangia and D. S. Katz, "Understanding software in research: Initial results from examining Nature and a call for collaboration," in *2017 IEEE 13th International Conference on e-Science (e-Science)*, pp. 486–487, 2017.

[3] M. Bello and F. Galindo-Rueda, "Charting the digital transformation of science: Findings from the 2018 OECD International Survey of Scientific Authors (ISSA2)," *OECD Science, Technology and Industry Working Papers*, no. 2020/03, 2020. doi: 10.1787/1b06c47c-en.

[4] J. Stubbs, R. Cardone, M. Packard, A. Jamthe, S. Padhy, S. Terry, J. Looney, J. Meiring, S. Black, M. Dahan, S. Cleveland, and G. Jacobs, "Tapis: An api platform for reproducible, distributed computational research," 2020. submitted.

[5] R. Dooley *et al.*, "Software-as-a-Service: The iPlant Foundation API," IEEE, 5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS), 2012.

[6] C. Calero, M. F. Bertoa, and M. . Moraga, "A systematic literature review for software sustainability measures," in *2013 2nd International Workshop on Green and Sustainable Software (GREENS)*, pp. 46–53, 2013.

[7] C. Stewart, W. Barnett, E. Wernert, J. Wernert, V. Welch, and R. Knepper, "Sustained software for cyberinfrastructure," pp. 63–72, 06 2015. doi: 10.1145/2753524.2753533.

[8] N. Condori-Fernndez and P. Lago, "Characterizing the contribution of quality requirements to software sustainability," *Journal of Systems and Software*, vol. 137, 12 2017. doi: 10.1016/j.jss.2017.12.005.

[9] M. Rosado de Souza, R. Haines, M. Vigo, and C. Jay, "What makes research software sustainable? an interview study with research software engineers," in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pp. 135–138, 2019.

[10] R. C. Seacord, J. Elm, W. Goethert, G. A. Lewis, D. Plakosh, J. Robert, L. Wrage, and M. Lindvall, "Measuring software sustainability," in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, pp. 450–459, 2003.

[11] C. Venters, L. Lau, M. Griffiths, V. Holmes, R. Ward, C. Jay, C. Dibsdale, and J. Xu, "The blind men and the elephant: Towards an empirical evaluation framework for software sustainability," *Journal of Open Research Software*, vol. 2, 07 2014. doi: 10.5334/jors.ao.

[12] S. Oyedeji, M. Adisa, B. Penzenstadler, and A. Wolff, "Validation study of a framework for sustainable software system design and development," 06 2019.

[13] N. Condori-Fernndez, P. Lago, M. Luaces, and A. P. Saavedra, "An action research for improving the sustainability assessment framework instruments," *Sustainability*, vol. 12, p. 1682, 02 2020. doi: 10.3390/su12041682.

[14] N. Condori-Fernandez and P. Lago, "Characterizing the contribution of quality requirements to software sustainability," *Journal of Systems and Software*, vol. 137, pp. 289–305, 2018.

[15] R. H. A. Aldabjan and C. Jay, "How should we measure the relationship between code quality and software sustainability?," in *WSSSPE4*, 2016.

[16] I. Groher and R. Weinreich, "An interview study on sustainability concerns in software development projects," in *SEAA*, 2017. doi: 10.1109/SEAA.2017.70,.

[17] C. Venters, C. Jay, L. Lau, M. Griffiths, V. Holmes, R. Ward, J. Austin, C. Dibsdale, and J. Xu, "Software sustainability: The modern Tower of Babel," *CEUR Workshop Proceedings*, vol. 1216, pp. 7–12, 01 2014.

[18] G. Berriman, J. Good, E. Deelman, and A. Alexov, "Ten years of software sustainability at the infrared processing and analysis center," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 369, pp. 3384–97, 08 2011. doi: 10.1098/rsta.2011.0136.

[19] V. V. Foster I and T. S., "Software as a service as a path to software sustainability," 06 2013. Available from http://dx.doi.org/10.6084/m9.figshare.791604.

[20] B. Allen, R. Ananthakrishnan, K. Chard, I. Foster, R. Madduri, J. Pruyne, S. Rosen, and S. Tuecke, "Globus: A case study in software as a service for scientists," in *Proceedings of the 8th Workshop on Scientific Cloud Computing*, ScienceCloud 17, (New York, NY, USA), p. 2532, Association for Computing Machinery, 2017. doi: 10.1145/3086567.3086570.

[21] "AWS S3." Last access: 2020-07-14.

[22] "Javascript object notation." Last access: 2020-07-14.

[23] "Secured shell protocol." Last access: 2020-07-14.

[24] "HTTP 1.1," 2020. Last access: 2020-07-13.

[25] "HTTP 2," 2020. Last access: 2020-07-13.

[26] "gRPC." Last access: 2020-07-13.

[27] "Protocol Buffers." Last access: 2020-07-13.

[28] "OAuth 2." Last access: 2020-07-14.

[29] "Openapi specification 3.0," 2019. Last access: 2020-07-13.

[30] "Postgres." Last access: 2020-07-14.

[31] "MongoDB." Last access: 2020-07-14.

[32] "RabbitMQ." Last access: 2020-07-14.

[33] "Apache Httpd." Last access: 2020-07-14.

[34] "Redis." Last access: 2020-07-14.

[35] "Tapis project," 2019. Last access: 2020-07-14.

[36] "Tapis Live Docs," 2020. Last access: 2020-07-14.

[37] "Ansible," 2020. Last access: 2020-07-14.

[38] "Sphnix," 2020. Last access: 2020-07-14.

[39] "Tapis CLI," 2020. Last access: 2020-07-14.

[40] "Tapis Python SDK," 2020. Last access: 2020-07-14.

[41] "TACC CLOUD SLACK," 2020. Last access: 2020-07-14.