MEDINFO 2021: One World, One Health – Global Partnership for Digital Innovation
P. Otero et al. (Eds.)

doi:10.3233/SHTI220124

Designing a Cloud-Based System for Affordable Cyberinfrastructure to Support Software-Based Research

Blake Anderson[a], Jason Cameron[a], Urmeka Jefferson[a,b], Blaine Reeder[a,c,d]

[a] *Hekademeia Research Solutions, Columbia, MO, USA*
[b] *College of Nursing, Rush University, Chicago, IL, USA*
[c] *Sinclair School of Nursing, University of Missouri, Columbia, MO, USA*
[d] *MU Institute for Data Science and Informatics, University of Missouri, Columbia, Missouri, USA*

Abstract

Interest in cloud-based cyberinfrastructure among higher-education institutions is growing rapidly, driven by needs to realize cost savings and access enhanced computing resources. Through a nonprofit entity, we have created a platform that provides hosting and software support services enabling researchers to responsibly build on cloud technologies. However, there are technical, logistic, and administrative challenges if this platform is to support all types of research. Software-enhanced research is distinctly different from industry applications, typically characterized by needs for lower reduced availability, greater flexibility, and fewer resources for upkeep costs. We describe a swarm environment specifically designed for research in academic settings and our experience developing an operating model for sustainable cyberinfrastructure. We also present three case studies illustrating the types of applications supported by the cyberinfrastructure and explore techniques that address specific application needs. Our findings demonstrate safer, faster, cheaper cloud services by recognizing the intrinsic properties of academic research environments.

Keywords:
cyberinfrastructure, cloud-computing, multi-institutional

Introduction

The cost-savings and resources of cloud computing represent an attractive solution to increased productivity among academic institutions. However, there are barriers that have slowed adoption, including policy and politics, distrust of for-profit vendors, and complexity of use [1]. Nonetheless, cloud-based architectures facilitate new and important research and their virtues far outnumber their limitations. As an increasing amount of research is performed on the cloud, a key question becomes how to sustain such research over time, when funding lapses, vendors fall out of contract, or personnel turn over.

Most modern platforms and technologies are created or licensed at individual institutions operating as insulated silos. To collaborate with other institutions, a researcher must overcome substantial technical and bureaucratic overhead. Specifically, multi-institute research projects must achieve compliance between multiple organizations, establish appropriate procedures with each Institutional Review Board, and negotiate intellectual property rights. It can take months or years to establish multi-institutional agreements that outline the responsibilities and ownership of shared data, and collaborating with sensitive data elements is practically impossible. At the same time, there is a national and international awareness of the need to promote cyberinfrastructure for collaboration [2]. In addition, there is a need for affordable, sustainable architecture that supports equity in research that reaches and engages vulnerable and lower socioeconomic populations [3].

While interest in cloud-based cyberinfrastructure is growing, there is limited information on the methods and technologies being employed at higher education institutes [4]. Our own experience suggests that creating affordable cyberinfrastructure is a heavily administrative undertaking with a wide variety of stakeholders with different interests and levels of participation. While researchers create software to increase knowledge or efficacy, departments and institutions are obligated to consider the business case for maintaining these tools. After painstaking years spent in designing and creating innovative software through sponsored research, researchers commonly face project termination due to loss of funding, gaps in institutional support, and lack necessary skills to maintain operations.

To establish potential solutions, we conducted case studies to determine the specific needs of various domains of research and identify support gaps within the process of creating or using software in academic research environments. We found consistent patterns of frustration and confusion when it comes to developing software components, publishing these tools, and maintaining these solutions over time [5]. There are a number of products that provide long-term storage for research data, such as DataONE.org, Databrary.org, Datadryad.org, DataHub.io, and FAIRSharing.org. Some projects also find a long-term home on a code repository site, such as GitHub or GitLab. However, when it comes to client-server applications, which require operation and upkeep, there are few, if any, viable solutions.

Our approach to sustainable cyberinfrastructure was an independent, nonprofit entity, Hekademeia Research Solutions ("Hekademeia"), created to serve as an honest broker and technical services provider for research at academic institutions. The sustainable operating model for Hekademeia relies on project-based fees for hosting and services, however it is vitally important that projects outside of an active funding cycle remain online. Therefore, we invest time upfront to ensure that projects in our environment have minimal cost and complexity for hosting and support. It is also necessary to foster strong relationships to serve as a trusted collaborator and included in active award cycles. To survive this early growth period and support the necessary research and development toward a full-scale cyberinfrastructure system, we have established a cooperative agreement with the University of Missouri as a proof-of-concept to provide a source of software-based projects and pilot funding. This agreement has enabled us to stand-up an initial cyberinfrastructure and formatively evaluate our efforts to build an affordable, reliable system.

One major cost of providing long-term hosting for research-based software is computational resources, which can be measured in cycles, memory, and storage. Efficient operation is a careful balance between quality and cost. Underspending can result in poor performance, instability, weak security, or worse. The conventional wisdom in industry software projects is that traffic brings revenue, and this correlation covers the cost to scale. With the unlimited scalability of the cloud, this model is elegantly simple and easy to execute. However, research projects are rarely amenable to traffic-based revenue creation. By definition, a majority of these projects will serve limited or underprivileged audiences where no industry model would be considered viable. To feasibly host these projects and provide long-term benefits to the research community and the general public, it is necessary to reconsider the conventional wisdom and engineer truly sustainable cyberinfrastructure for academic research.

Methods

Architecture

Hekademeia provides a novel hosting approach designed to minimize costs with negligible impact on quality of service. The core of this approach is a Docker-based swarm [6], which allows us to commission a cluster of servers to share in the responsibility of hosting individual software services, called containers. The swarm's redundancy provides robustness in the event of hardware failure and allows us to scale computational resources as needed. Moreover, these containers logically act as independent servers, making it possible to give each project an isolated, secure, and fully-customizable environment, while leveraging the substantial cost-savings of shared hosting. With well-designed containers, it is possible to fit hundreds, or even thousands of projects on a small number of low-grade server nodes. By deriving new projects from a family of base container images, we are able to share many layers of software libraries and configuration, making individual containers safer and easier to use at a fraction of the size of a dedicated server environment. Finally, the static nature of containers practically eliminates the need for system maintenance over time. In fact, users in the Hekademeia environment have no system-level access to deployed containers. Containerized services can be fully designed and tested in a development environment. The combination of these techniques results in a highly-scalable, sustainable architecture at a much lower cost and lower risk than conventional hosting solutions.

A Docker container is based on a collection of read-only layers, which represent the result state of discrete instructions to copy files, install software libraries, or execute commands. Containers can be derived from other containers, such that they share any number of these layers. In fact, a Docker node only needs to retain a single instance of this layer, so sharing container layers can result in substantial file storage and bandwidth savings. To maximize this feature of containers, we require all containers within the proposed architecture to be derived from one of our base images. We also encourage our users to use scratch or alpine base images wherever possible, as these have a significantly reduced system footprint. This allows a majority of the system libraries and common frameworks to be reduced to a single copy on the swarm. In practice, a container for many projects can be a simple layer of the project files themselves, reducing a gigabyte system to a handful of megabytes.

Another key aspect of a Docker container is persistence. Since the container itself is read-only, operations to modify the file system occur within a container instance and, therefore, cease to exist when that container is destroyed. While this may seem like a limitation, container persistence is a valuable property for designing predictable, scalable systems. By limiting persistent storage within a container, it is possible to spawn multiple instances for load balancing or parallelization. This model works particularly well for applications using an independent database system. Docker also provides secrets, which are a secure way to inject sensitive data, such as settings, passwords, and encryption keys into a container. When it is necessary for a container to work with truly persistent storage (e.g., file uploads), a host directory can be explicitly designated as a mounted volume. For a swarm environment, it is helpful to use a Network File System for persistent storage that will be available from any node where the container might be instantiated. Once an application is adapted to a static environment, it can be "pre-baked", tested, and distributed to a container registry prior to deployment. For public images, DockerHub is a popular open registry while some code repositories, such as GitHub and GitLab, provide more private container registry services. Ultimately, limiting the dynamic components of the application to explicit volume mounts and independent database systems creates simplicity and reliability for the architecture.

Security is another major consideration for a shared environment, and it is important to be responsive to project needs. Each institution has a series of data classification levels, representing sensitivity and risk of breech. These classifications inform our planning and allocation of resources. For public data, the only consideration is preventing corruption or tampering. More sensitive data requires access controls and authorization protocols to ensure that only the appropriate users have access. Highly confidential information requires hardware and software-based encryption, end-to-end encryption, as well as code reviews and testing. The most restricted data, such as protected health information (PHI), has additional architectural considerations. In our case, we require that all access to such data come through a managed API with appropriate monitoring and audit controls.

These security methods are common and effective in most environments where software is developed by trusted in-house personnel with certifications and accountability. Unfortunately, in an academic environment, software is created by inexperienced students, third-party vendors, and even those with deliberately malicious or clandestine intent. Since it is infeasible to review, test, and monitor every piece of software within an academic architecture, special attention must be given to minimizing the impact of vulnerable, malicious, or poorly-designed code. It is necessary to assume that projects will have all of these qualities and plan accordingly.

We have found that the ubiquity of Docker and the static nature of containers make it relatively easy to develop and test container in a local environment prior to releasing a new version to a production system. Rather than making changes directly to a live application, a developer can deploy a container on a local machine, where it can communicate with localized editions of storage and settings. In fact, there is little reason to work with production data within a development environment. We found that developing on a set of pre-defined "unit data" leads to more consistent development and debugging. It is often possible to reproduce and patch a software bug with confidence by creating the appropriate data situation in development. For sensitive applications, this means that developers need never see or access to production data. We call this strategy sandboxing, and we have developed a number of advanced methods for creating sandbox environments with synthetic data sets.

Projects are rarely designed for this sandboxed method of development or even optimized for container-based hosting. Before we could test our cloud-based architecture, we had to migrate each project to a compatible format. It is crucial that this migration does not affect the function of the code or create undue burden on the developer. At the same time, a light touch is necessary to keep labor costs down and keep the project maintainable by its creators. There are a few basic techniques that we have found common for this migration process. The first step in the process is building a functioning container from one of our base images. This requires a basic understanding of the frameworks and languages used in the project to ensure that the correct dependencies are available in production. Next, it is necessary to leverage environment variables and localize configuration files. In the production environment, these containers will be given specific database credentials, file paths, and other settings and the application must be designed to handle these changes. We also recommend a schema management and migration tool to facilitate and automate schema changes on the production environment. The migration process can be time-consuming, and it often requires some level of training or discussion on the architectural approach. However, the migrated projects are often substantially smaller, faster, and easier to maintain. The revisions add little or no complexity and provide long-lasting benefits to most projects.

We have constructed a Docker swarm on Amazon Web Services (AWS) [7] to demonstrate the techniques and host a series of pre-existing research applications. Our swarm consists of four servers: a master node (t3.small), two worker nodes (t3.micro), and a dedicated server (t3.micro) for swarm access and specialized tasks. We have also incorporated two Amazon Relational Database Systems (RDS) for PostgreSQL (postgresql.org) and MySQL (mysql.com) applications. At the time of this publication, our swarm is hosting 14 stacks, with a total of 31 services. We use Traefik (traefik.io) as a load-balancer and proxy service to issue SSL certificates and direct traffic to the correct container(s) on the swarm. The hosted applications represent many varieties of application, from basic project website and mobile applications to scheduled data scrapers to general research tools.

Results

Over the past decade, we have hosted many of the same research applications under traditional hosting systems. This allows us to draw some quantitative comparisons for the new method. The immediate predecessor to this architecture was primarily PHP 7, Apache 2, Ubuntu 18, various HTML/JavaScript applications and HAProxy running on t2.micro and t2.smalls in a similar configuration. Most containers volume-mounted their project files from a network file system on the swarm. Previously, we decided to disable active monitoring, which brought our CPU usage down by about 4% to an average load of roughly 8%. This configuration was still under the baseline utilization for an AWS node, so things typically ran smoothly with a similar four-server setup. However, we chose to reconsider the architecture due to security considerations for third-party developers and an unscalable, manual process for renewing SSL certificates every 90 days (dockercloud/haproxy did not automate this process as expected). Many of the performance improvements were consequences of our focus on CPU, RAM, and disk usage during revision.

Most of our containers were migrated to a Nginx and PHP-FPM 7 stack derived from the dockage/alpine:3.11 Docker image. The immediate result of this change was a container size that dropped from 800MB+ to 21MB. Since the container layers are replicated to each swarm node that hosts the relevant container, this reduces bandwidth and file storage per node dramatically (although it is important to note that there is only a single copy of the base layers for any number of derived containers).

These containers were also prebuilt with localized production settings as environment variables and Docker secrets. This enhancement makes it possible to deploy services to the swarm without a copy of the code or other resources. This substantially simplified the software release process and makes it compatible with Continuous Integration (CI) deployment. For compiled applications, such as Webpack-based front-ends, the prebake process allows the CPU-intensive build process to be performed and the 500MB+ JavaScript libraries to remain on the development machine. After compilation, these containers are often 2MB in size, plus the resources for the site media. Importantly, this approach eliminates the need for developers to have access to the production systems or containers, as they have no "moving parts".

Our experience moving from HAProxy to Traefik was a simplified upkeep process, particularly around automated SSL certificate renewals, but also Traefik's label-based service orchestration. Traefik has more robust configuration options, such as the ability to redirect traffic based on regular expression matching. These redirects would otherwise be part of the server layer, complicating development sandboxing and production localization.

Finally, we have observed major performance and CPU usage improvements. As shown in Figure 1, the baseline CPU usage for most of our nodes hovers just above 0% (compared to the previous 8%). Other than the usage peaks associated with scrapes and scheduled processing jobs, we remain well below the AWS baseline usage mark of 10 or 20%. This means that where we were previously pushing the limit of what we could host on our swarm before increasing nodes, we can now support hundreds of projects, depending on their resource characteristics.

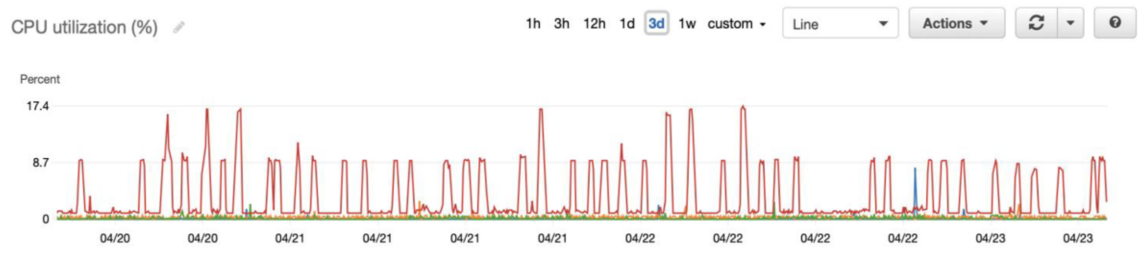

Figure 1– Representative 3-day report of CPU-usage from three AWS swarm nodes hosting multiple research software stacks.

Case Studies

The Hekademeia architecture is designed to support a variety of applications, and research software tends to involve eccentric combinations of tools and technologies. The following are specific examples of common application types and techniques used during the design phase to accommodate their research requirements.

One type of project that we have migrated to our swarm environment is a Python-based tool (python.org) for scraping Twitter geolocations. After minor adaptations to a containerized environment, the script itself functioned properly. However, the script was designed to run a single location in a wait-loop. Analysis showed that each instance of the script consumed 80MB of RAM, limiting the number of locations that could be scraped concurrently. We knew that a single instance of the script scraping multiple locations would save some overhead. This was a case where we had to weigh the labor cost of modifying the script against the long-term costs for hosting. Fortunately, we were able to use Python's pseudo-threading mechanism to parallelize the command with minimal code modification. The end result was a script that consumed 80MB of RAM, but only an additional 2MB per location, making it far more scalable in a production environment.

Another common application type we see are mobile applications. There are countless uses for mobile technology in research, and it is easy to see early success in prototypes. However, the long-term costs for upkeep of mobile applications can be prohibitive. The first problem with mobile applications are the rapidly changing operating system versions and features. It is fairly common for an application to stop working after a software update, particularly for Android devices. Even a minor content change to a mobile app, in iOS or Android, often requires adapting the app to new libraries and procedures. These changes must also undergo review by Apple or Google, often taking weeks to get an update to end users. To mitigate these issues and simplify the maintenance process, a researcher can rely on a web-based framework, such as Apache Cordova or Expo Mobile. These platforms can create up-to-date, native code for mobile devices, while using the same or similar code base for web-based instantiations of the software. However, the choice of framework can have a major impact on upkeep. For example, Expo relies on the application to be built from source libraries on the fly, making it difficult to containerize. Where an app could normally be pre-baked to a respectable 10MB or less in file size, these applications tend to be closer to 1GB, increasing the server load and required disk space.

A third type of application is the data-intensive project. For example, projects that collect or analyze sensor and wearables data can easily accumulate hundreds of thousands of records per month. While cloud storage is cheap, it is not unlimited, and these types of applications can lead to runaway storage costs. More importantly, an active project can induce long query times. With scalable containers, the bottleneck for performance is typically the database server, so these applications can have a noticeable system-wide impact. In these cases, it is important to consider the nature of the application to determine the appropriate long-term management plan. For applications that do not need granular data values, it can be beneficial to capture data less frequently. It is also important to assess the likely patterns of data use. If recent data is used differently than historical data, it can be worthwhile to create a rollover or staging mechanism to reduce the system burden for long-term sustainability.

Discussion

Using the methods above, it is possible to host hundreds or even thousands of applications on a small cluster of cloud servers. However, it is vital that applications be designed appropriately to gain the full benefits of the shared platform. For example, a typical project stack within our environment consumes approximately 20MB of RAM. By contrast, a single WordPress site can consume 180MB of RAM or more. The choice of technologies and framework can have a substantial effect on the overall scale (and recurring costs) of the platform. Therefore, it is worthwhile to dedicate considerable effort toward reducing the resource usage of an application prior to long-term hosting.

It is also prudent to consider the optimal reliability for a given project. In some cases, a study involves a pre-determined audience or limited period of performance. Other projects may be prone to bursts of heavy traffic or computing activity requiring increased availability. Our cyberinfrastructure can accommodate either of these situations, but it is not cost-effective to try to do both within the same swarm or node subset. One easy method of increasing reliability is to create additional master nodes, which can take over in the event of node failure. However, this increases the processing and node requirements, which may not be appropriate for prototype applications or low-fidelity studies. Matching the needs of a study to the appropriate level of reliability can dramatically impact overall costs.

Even with the proposed methods to reduce equipment costs, there is a sizeable labor component to software hosting. Servers need monitoring and maintenance; software requires bug fixes and feature changes. Man-hours are orders of magnitude more expensive than equipment, so it is imperative that the systems are designed for minimal upkeep and attention. Before a full-scale swarm can be populated with hundreds of applications, it will be necessary to design an appropriate project management system (PMS). The PMS needs to allow researchers to directly manage and deploy software solutions without the intervention of IT staff, while enforcing the access controls and permissions of users on the swarm. Continuous Integration technologies can provide users with the automated deployment options and have the benefit of enforcing compliance tests such as security or web accessibility prior to software publishing. These tools are an important next step in the development of a reliable and sustainable cyber infrastructure platform.

Conclusions

We have established a novel, cloud-based architecture to provide long-term, sustainable hosting and support for research software. We have demonstrated methods to lower costs and enhance the capabilities for many types of research. Further study is needed to determine whether computationally-intensive settings, such as machine learning or big data studies are appropriate for a shared, cloud-based architecture. While there are still many challenges ahead, we are optimistic that the use of such an architecture within a multi-institutional honest-broker organization, such as Hekademeia Research Solutions, can provide affordable options to create and maintain web and mobile applications, data collection tools, and other software platforms for research. Indeed, we are currently engaged with two other major research universities to host multiple software-based research projects.

Acknowledgements

This work is supported by the University of Missouri.

References

[1] A Shakeabubakor, E Sundararajan, A Hamdan. Cloud Computing Services and Applications to Improve Productivity of University Researchers. *3rd Int Conf on Electronics Engineering and Inform*, 2014.

[2] J Roskoski. Dear Colleague Letter: Cyberinfrastructure Framework for 21st Century Science and Engineering. *National Science Foundation*. NSF 10-015.

[3] C Harrington, B Jelen, A Lazar, et al. (2021) Taking Stock of the Present and Future of Smart Technologies for Older Adults and Caregivers. https://cra.org/ccc/resources/ccc-led-whitepapers/#2020-quadrennial-papers

[4] M Banu Ali, T Wood-Harper, M Mohamad. Benefits and Challenges of Cloud Computing Adoption and Usage in Higher Education: A Systematic Literature Review. *Int J Enterprise Inform Systems* 14, issue 4. Oct-Dec (2018).

[5] B Anderson, B Reeder, Hekademeia: Sustainable Research Technology Infrastructure for Persistent Project Access Across Funding Gaps. *AMIA Annual Symposium*, Chicago, IL, 2020.

[6] N Marathe, A Gandhi and J M Shah, "Docker Swarm and Kubernetes in Cloud Computing Environment," *2019 3rd Int Conf on Trends in Electronics and Inform*, 2019, pp. 179-184.

[7] S Mathew, and J Varia. "Overview of amazon web services." *Amazon Whitepapers*, 2014.

Address for correspondence

Blake Anderson, blake@hekademeia.org
Executive Director
Hekademeia Research Solutions
2601 Northridge Dr., Columbia, MO 65202, USA.