Experiences in Managing CHEESEHub: A Cloud-based Cybersecurity Education Gateway

Sara Lambert
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Urbana, USA
lambert8@illinois.edu

Rajesh Kalyanam
Research Computing
Purdue University
West Lafayette, USA
rkalyana@purdue.edu

Baijian Yang
College of Information Technology
Purdue University
West Lafayette, USA
byang@purdue.edu

Rob Kooper
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Urbana, USA
kooper@illinois.edu

***Abstract*—CHEESEHub is a web-accessible, public science gateway that hosts containerized, hands-on demonstrations of cybersecurity concepts. There are now a plethora of services and tools designed to simplify modern gateway deployment and configuration such as commercial and academic composable cloud, the Terraform infrastructure as service tool, Kubernetes and Helm for container orchestration, as well as CILogon for simplified user authentication. Despite leveraging these tools, our day to day experience with deploying, upgrading, scaling, and extending CHEESEHub has not been entirely straightforward. We describe here some of the major challenges we have encountered in managing CHEESEHub and developing web-accessible demonstrations for the last five years. We hope this will help both new and seasoned gateway developers to effectively leverage these modern tools while avoiding these same pitfalls, while also providing starting points for discussions around gateway development and deployment best-practices.**

***Index Terms*—gateway, containers, deployment, experiences, cybersecurity**

I. Introduction

CHEESEHub [1] was developed in response to the growing need for cybersecurity expertise in the IT workforce and to narrow the skills gap between classroom instruction and hands-on practical training in the current cybersecurity curriculum. Consisting of containerized demonstrations of cybersecurity concepts (such as the ARP poisoning attack, HeartBleed bug, and SQL Injection attack, to name a few), and complementary learning and assessment materials; CHEESEHub is designed to complement classroom instruction by providing students with the means to reproduce these security attacks themselves and evaluate possible solutions.

To lower the barrier to entry and expand its users to include those who may be unfamiliar with system administration, CHEESEHub is designed to be web-accessible and uses containers to both package and isolate the insecure code or security exploits relevant to a cybersecurity demonstration. Once logged in, users can launch a particular demonstration with just a couple of button clicks on the CHEESEHub web gateway. Behind the scenes, one or more containers are launched and the user is presented with the web-accessible interface to these containers: often a Jupyter interactive computing environment or an entire Linux desktop via noVNC.

CHEESEHub is built on the National Data Service (NDS) Labs Workbench gateway framework [2] that can be deployed to a Kubernetes cluster on most commercial and campus cloud resources via publicly available Helm charts. Furthermore, Labs Workbench supports federated user authentication via CILogon, which allows for customization to either only allow users from one (or more) institutions, or more widely anyone with a Google account. Finally, Labs Workbench supports a customizable and extensible application catalog via a repository of JSON configuration files, which contain all of the details needed to launch an application from the catalog as a Docker container.

CHEESEHub has now been in production operations for five years and has been used to complement various undergraduate and graduate courses at Purdue University, and in both in-person and virtual hands-on workshops for the IT education community. While we heavily leverage modern cloud and container orchestration tools such as Terraform, Kubernetes, and Helm to simplify the management, porting, extension, and upgrade of CHEESEHub over these five years, there are also several challenges around deployment that have revealed themselves during this period. Just as importantly, we have also faced challenges in designing applications that are both easy to develop, but are also user friendly with a low barrier to access and use by users unfamiliar with server administration. In this paper we provide an account of such challenges that may not necessarily be apparent during the design or initial deployment phases. We should note that CHEESEHub presents some unique challenges around security, since we are after all deploying containers with security exploits and need

Funded by the National Science Foundation (NSF) awards: 1820573 and 1820608.

Presented at Gateways 2022, San Diego, USA, October 18–20, 2022. https://zenodo.org/communities/gateways2022/

to avoid malicious users adversely affecting the platform or other users' ability to use the platform. While we do not discuss these challenges here, interested readers may refer to the authors' prior publication [3] describing these security challenges and solution strategies.

II. Challenges in Managing and Developing for CHEESEHub

We divide the challenges into three broad categories: deployment and upkeep, user management, and application design.

A. Deployment and Upkeep

When deploying CHEESEHub over the past five years, we have had to not only keep up with technology changes but also adapt to new use cases, and infrastructure and/or software updates. Key among these challenges was designing a deployment solution for a variety of cloud environments and OpenStack flavors, learning new tools such as Terraform for maintaining the deployment process, and keeping up with updates to the Kubernetes API to ensure graceful redeployment after making updates to different parts of the platform.

1) Cloud Migration Challenges: One large problem that we have faced has been supporting the growing number of commercial cloud providers, who vary in their own support and deployment patterns. Our initial choice of using Kubernetes to run the underlying platform was very valuable toward this goal: we still have the option to deploy standard virtual machines in any shared network environment, install Kubernetes on them to join them into a cluster, and deploy CHEESEHub on the cluster. Due to the increasing native support for Kubernetes by commercial cloud providers, we are seeing more options for deploying it with native support within their respective ecosystems such as AWS Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), and Azure Kubernetes Service (AKS). The result of using these new native services is a cluster that is all-around easier to monitor, upgrade, and scale, but the testing and documentation of these deployment steps with slight differences on various clouds does generate additional overhead. Furthermore the deployment steps and cluster requirements can already seem very complicated to those unfamiliar with containers and orchestration, and adding branching notes about which steps are required or optional under particular providers can certainly make the documentation even harder to follow.

2) Variance in On-Premises Hardware Patterns: For those with access to on-premises hardware, our Terraform deployment process works with select versions of OpenStack to deploy a Kubernetes cluster with all of the required pieces. We also provide a Helm chart that will launch the application atop the cluster once deployment is finished. We rely mainly on Terraform's built-in support for various different OpenStack version. Although the API is generally compatible using this approach, in our experience every OpenStack deployment has unique quirks. For example, OpenStack's volume and network setups can vary between deployments; there are times when we need to specify the name of a network or pool, while other times we need to provide an ID instead.

One curious observation is that the pain points of supporting various different OpenStack clusters are indeed the same as those for differences in support between on-premise and various commercial cloud providers: differences in the native support and behavior of networking and volumes. These seem to be the most variable part of any deployment, as the security and data needs can vary uniquely from use case to use case.

3) Adding Nodes via Terraform: While adopting Terraform for deployment has indeed been very valuable, it is yet another complex deployment tool that has taken additional time to learn. It has been particularly troublesome attempting to use this pattern to maintain only one or two nodes, only to find that Terraform is now accidentally operating on the full cluster. For example, in attempting to scale up the cluster by adding a new node we found that the existing nodes were having their data volumes being re-mounted under a different incorrect path. While a restart would fix the problem and mount the data in the correct location, we were left wondering why was operating on the old nodes at all when it was only asked to create and join a new one. Ultimately, this was not a bug in Terraform, but due to how our deployment scripts weren't properly performing change checking for its steps. While rebooting the nodes would remount the data to the correct location, we later determined that we weren't providing Terraform with enough information about how to determine whether or not it had already run some of the deployment steps, namely the step that mount the data volumes, so the tool had determined that this step needed to be re-run every time.

4) Kubernetes API Challenges: Kubernetes is a wonderfully complex piece of software. Their API is very well thought-out, with regular new releases occurring every 3 months, and changes to the API are supported across multiple major versions. Despite the generous rolling window, keeping up with these changes has been a real challenge, particularly with the Golang Kubernetes client. Golang has gone through several iterations of dependency management technology since the original backend for Labs Workbench was implemented, turning simple package dependency upgrades into a non-trivial task. Compliance with this new pattern could not be guaranteed without significant changes to rewrite or modify the code to run as a "go module", so we are instead working to move the backend to Python where these common development patterns are more stable than what we've found with Golang.

B. User Management

While CHEESEHub is considerably less complex than other gateways that require distinct user roles and role-based resource authorization, we have tried to further simplify the adoption of CHEESEHub in educational and training activities by eschewing account creation before first use. Consequently, we employ CILogon for user authentication and authorization via institutional credentials. Nevertheless, this is not a panacea as we discovered when working with diverse institutions that look to utilize CHEESEHub in their training.

1) CILogon Integration and Identity Management: Every institution has a unique login system. Even though they all tend to use the same protocol, there are still specific parameters for each university that must be used for authentication. This is where a system like CILogon becomes very handy. CILogon is identity management software that does the work of keeping track of individual universities and supporting their login flows, which usually involves working with the target institution's IT staff to some degree. While CILogon does support a vast number of institutions and providers, it is still not a perfect solution to this problem. For example, there have been cases where support for lesser-used universities has lapsed over time. This is understandable from an outside perspective, as it must be very difficult to maintain support for every possible institution. We have found this to be especially true when dealing with smaller institutions that have limited IT resources. In these cases, we have been able to pass along the details of what would be needed on their end for CILogon to add them to the list of providers, but little more can be done from our end to ensure that their IT staff follows through on the instructions.

There have been cases where the steps needed are simply too much for them to guarantee at the given time, and so we have to find alternative login methods for the students in these cases. Our go-to fallback has been for people to use a Google account, as it is simple to register a throwaway account if they don't have a personal Google account or are not comfortable using it in this context. Unfortunately this makes it a bit too easy to register for an account, as even a malicious user can register for a Google account. Since we are providing access to running containers, this would be a problem if not for the time-limited usage of our use case in the classroom or workshop setting.

2) Various User Profiles: We have recently enhanced CHEESEHub by integrating with Keycloak, an open-source software that allows us to manage users. This has opened up a lot of possibilities for us. We can now easily assign roles to groups and users, add new authentication client options, and provide a mechanism for authenticating via JSON Web Token (JWT) [4] to provide all of this information, all without building any of that logic into our own application. Keycloak is very complex to configure, but it is certainly worth the time investment to understand it in order to save on extra development effort toward a hardened and extensible authentication mechanism. There are some slight gaps in their use cases that we have found when using CILogon, which allows people to authenticate using credentials from various organizations and institutions. For instance, there are cases where a user may have two different valid sets of credentials from two different institutions. This is a case that Keycloak does not support, as it attempts to do a database insert to assign the linked identity. This fails, as the CILogon identity is already linked to the account and we cannot link another one with the same provider. One way to solve this might be to upsert when linking a new identity to overwrite the existing link with a new one when using new credentials. Another might be to customize Keycloak's OIDC handling to allow for multi-providers to use a specified field to maintain their uniqueness by specifying a claim name to append. This might result in two linked identities (e.g. `cilogon-ncsa` and `cilogon-uiuc`).

C. Application Design

CHEESEHub is intentionally designed to be accessible via web browser and to be used on a variety of hardware such as tablets, chromebooks, etc. that are common in educational settings. Consequently, all of our applications have to be designed to be easy to interact with in these environments. At the same time, CHEESEHub seeks to allow the broader community to easily contribute demonstrations of cybersecurity concepts as reproducible containers with minimal overhead; hence we cannot expect highly interactive and user friendly web applications.

1) Use of Interactive Computing: To emulate a desktop environment in the cloud, there are several options available including noVNC and Xpra. These options are fairly limited in the operations that they support, with notable limitations that inhibit basic usage. For example, copying and pasting between the host and the emulated desktop has always been notoriously difficult for such a simple and common operation. However, certain demonstrations do require an entire desktop environment. For instance, the demonstration of HeartBleed on CHEESEHub illustrates how a malicious hacker can leverage the HeartBleed bug on unpatched servers to steal session cookies from authenticated web sessions and gain access to these sessions in their own browser. This demonstration requires the user to use both a terminal to run commands to attempt to steal the session cookie and a browser to evaluate whether the stolen cookie works in gaining access to authenticated sessions. The only option in this case is to embed documentation locally within the noVNC-based container to allow users to easily copy and paste between the included documentation and the terminal or browser, which is not very user friendly.

2) When a Terminal Will Do: In certain demonstrations, all that is needed is a terminal into the running container. For this reason, we use Websockets to facilitate the exchange of data behind the scenes to provide a web-based terminal within the user interface. This allows users to send commands and receive their output using a familiar interface. Due to the security implications of sending raw commands to the underlying container, however, it is necessary to protect access to this resource. As such, we have implemented a timeout that automatically closes the console after several minutes of inactivity. Since we are using Websockets to handle potentially long-running CLI processes, the definition of "inactivity" here can be subjective or situational. For example, say we are running a CLI command that takes several minutes to complete. If this process does not send any new output during our timeout period, the user likely will not be sending new input until the command has completed. At this point the connection could be classified as "inactive", as it is not seeing any output coming back from the command or any new input

coming through from the user. Given this possibility we would like to set the timeout sufficiently high to allow the user to finish their lesson, but not so long that it would pose a potential security risk, making this configuration something that would need periodic evaluation and revision.

3) Providing Instructions Alongside the Hands-on Activities: One significant challenge that we have encountered is how to provide users with step-by-step instructions to reproduce a demonstration. Our current solution is to use the Software Carpentries model to publish a separate web-accessible lesson plan, however this requires users to constantly switch between the lesson website and the web container environment where they need to type their commands. We have experimented with using Jupyter as a self-contained interactive environment with both step-by-step instructions (in Jupyter notebooks) and terminal access. This too can often be challenging when there are three separate "servers" to enter commands on; for instance, the demonstration of ARP poisoning attacks requires the user to repeatedly function as the "hacker", "server", or "hacked user", entering commands in the corresponding terminals. With no way to identify which is which, we often have to resort to providing the server or the terminal user with an identifiable name based on its role in the demonstration (i.e, hacker, server, or user).

III. Discussion

We note that while some of the aforementioned challenges are of broad scope and applicable to most modern gateways that leverage container orchestration on commercial and academic cloud, some others are specific to CHEESEHub and its user base. Specifically, gateways designed for educational activities may need to prioritize ease of following instructions more often than those designed for savvy researchers. Similarly, gateways that decide at the outset on a specific hosting platform and authentication solution to use, may not need to build in the generality, but only adapt to changing technologies and solution availability.

We conclude with a summary of questions that we wish we had asked ourselves earlier when designing our web-accessible gateway for cloud deployment:

1) Do you intend to host on a single cloud platform or provide solutions for various cloud providers?
2) Have you identified between these platforms in how they manage and reference deployed resources?
3) Is there any official deployment documentation from the target cloud platform(s) that could be leveraged to simplify your own?
4) Is there a sufficient dependency management pattern in your target language that will allow for easy upgrade of external API clients to newer versions?
5) Have you considered the various institutions that may either deploy, integrate with, or use your gateway and their computing and support resources?
6) Could user, group, and identity management be handled separately from the rest of the application?
7) Are there web-based alternatives available for containerized software products that typically run as graphical native desktop software?
8) Would a web terminal suffice for this lesson or demonstration? Are any necessary timeout parameters set high enough to avoid interrupting the user during the lesson?
9) Could lesson plans be more thoroughly integrated with the software to grant users a hands-on educational experience without forcing them to frequently switch contexts?

References

[1] R. Kalyanam, B. Yang, C. Willis, M. Lambert, and C. Kirkpatrick, "CHEESE: Cyber Human Ecosystem of Engaged Security Education," 2020 IEEE Frontiers in Education Conference (FIE), pp. 1–7, IEEE, October 2020.

[2] C. Willis, M. Lambert, K. McHenry, and C. Kirkpatrick, "Container-based analysis environments for low-barrier access to research data," Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact, pp. 1–4. 2017.

[3] M. Lambert, R. Kalyanam, R. Kooper, and B. Yang, "Securing CHEESEHub: A Cloud-based, Containerized Cybersecurity Education Platform," Practice and Experience in Advanced Research Computing. Association for Computing Machinery, New York, NY, USA, Article 43, pp. 1—4. 2021.

[4] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, ¡https://www.rfc-editor.org/info/rfc7519¿.