Managing Privilege and Access on Federated Edge Platforms

Joe Breen
University of Utah
joe.breen@utah.edu

Lincoln Bryant
University of Chicago
lincolnb@uchicago.edu

Jiahui Chen
University of Utah
jiahui.chen@utah.edu

Emerson Ford
University of Utah
emerson.ford@utah.edu

Robert W. Gardner*
University of Chicago
rwg@uchicago.edu

Gage Glupker
University of Michigan
gglupker@umich.edu

Skyler Griffith
University of Utah
skyler.griffith@utah.edu

Ben Kulbertis
University of Utah
ben.kulbertis@utah.edu

Shawn McKee
University of Michigan
smckee@umich.edu

Rose Pierce
University of Chicago
rosepierce@uchicago.edu

Benedikt Riedel
University of Chicago
benedikt.riedel@icecube.wisc.edu

Mitchell Steinman
University of Utah
mitchell.steinman@utah.edu

Jason Stidd
University of Utah
jason.stidd@utah.edu

Luan Truong
University of Utah
luan.truong@utah.edu

Jeremy Van
University of Chicago
jeremyvan@uchicago.edu

Ilija Vukotic
University of Chicago
ivukotic@uchicago.edu

Christopher Weaver†
University of Chicago
cnweaver@uchicago.edu

ABSTRACT

The SLATE (Services Layer at the Edge) platform supports collaborative, multi-institution scientific computing through federation of containerized edge services. This paper considers issues of trust between resource providers and developers of orchestrated services which span administrative domains. The context for discussion is the SLATE federation architecture and application deployment pattern. The major features are a custom central API server which implements the federation, partitioning of user and group applications on edge clusters, and a curated catalog of applications which can be installed within the federation.

CCS CONCEPTS

• **Computer systems organization → Grid computing, Edge Computing**.

*Corresponding Author
†First Author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PEARC '19, July 28-August 1, 2019, Chicago, IL, USA

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-7227-5/19/07…$15.00
https://doi.org/10.1145/3332186.3332234

KEYWORDS

Distributed computing, Containerization, Edge computing

ACM Reference Format:
Joe Breen, Lincoln Bryant, Jiahui Chen, Emerson Ford, Robert W. Gardner, Gage Glupker, Skyler Griffith, Ben Kulbertis, Shawn McKee, Rose Pierce, Benedikt Riedel, Mitchell Steinman, Jason Stidd, Luan Truong, Jeremy Van, Ilija Vukotic, and Christopher Weaver. 2019. Managing Privilege and Access on Federated Edge Platforms. In *Practice and Experience in Advanced Research Computing (PEARC '19), July 28-August 1, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, **??** pages. https://doi.org/10.1145/3332186.3332234

1 INTRODUCTION

Many current science experiments are performed by collaborations which span multiple institutions and laboratories, and these collaborations often utilize computing grids such as the Open Science Grid [**?**] and the Worldwide LHC Computing Grid [**?**] to access computing resources at yet different locations. This paradigm of computing requires considerable supporting software infrastructure to support movement of data and computing jobs among sites at which computation occurs. Software is typically distributed using the CernVM File System [**?**] on individual compute nodes, with Frontier Squid [**?**] used as a cache to reduce redundant requests from the site as a whole, as well as caching of other HTTP requests from computing jobs [**?**]. As computing and storage cannot always be colocated, data transfer services, such as Globus [**?**], XRootD [**? ?**], Rucio [**?**], and iRODS[**?**] are often needed to handle the large volumes of data processed or produced. The 'compute element' components of a grid, such as the HTCondor CE [**?**] and Arc CE [**?**], must also be deployed in order to handle transferring users'

batch jobs from submit hosts to the computing facilities. In many cases, the expertise concerning which of these software services are needed and how they must be configured is held substantially by the science collaborations or the grid organizations, yet in order to function they must be deployed at the computing facilities, with the permission of those sites' administrators.

The SLATE platform [**?** **?**] supports this type of collaborative, multi-institution scientific computing by enabling distributed software infrastructure to deploy close to computing resources, at sites such as high performance computing centers and storage facilities. It is designed to provide the necessary trust guarantees enabling service administrators to deploy and manage their software infrastructure effectively, while simultaneously maintaining the security of the computing resource providers' software, storage, computing, and network resources. To do this, SLATE combines a federation architecture, which limits access required by the platform to the resources contributed by the site, and a curated catalog of software which the SLATE process audits for basic suitability. While SLATE uses Kubernetes [**?** **?**], a platform for deploying and managing containerized applications, as its underlying technology, the concept outlined here is largely independent of particular Kubernetes features, making it applicable to other implementations. While the SLATE platform is still quite new, several groups are investigating using it to deploy services. Using SLATE, the XCache variant distribution of XRootD has been successfully deployed at multiple sites, and is being tested for use as a component of the ATLAS data processing system. Typical components of Open Science Grid site deployments including Frontier Squid and the Stashcache version of XRootD [**?**] are already available from the SLATE catalog and are being tested prior to production use.

SLATE supports both edge [**?**] infrastructure services, such as caches and data transfer endpoints, and domain applications which perform particular scientific calculations. While there are important differences between infrastructure services and domain applications, for the context of this paper they can generally be treated together and will be referred to generically as 'applications'. For the purposes of this paper, we will draw focus to how edge infrastructure services provided by SLATE may augment the capabilities of HPC facilities and reduce overall operational burden.

2 SLATE ARCHITECTURE

SLATE uses a container-based approach to deploying applications to provide uniform environments and encourage scalable, stateless applications. In order to orchestrate these containers, the SLATE team uses Kubernetes, a container orchestration system with a robust open source community and used by a number of scientific computing providers. Many options are possible for federating multiple sites using Kubernetes, including "stretched" Kubernetes clusters with geographically distributed nodes, native Kubernetes federation [**?**], and add-on products such as Admiralty [**?**]. The approach taken by SLATE, as illustrated in Figure **??**, is instead to introduce a central, custom component, distinct from Kubernetes, to implement federation capabilities. SLATE's "lightweight" federation introduces a number of advantages: a) a tailoring of the federation to the exact needs of the SLATE platform, b) a partial decoupling of the development of the SLATE platform from changes in the Kubernetes project itself, and c) allowing sites to retain full administrative control of their own Kubernetes cluster.

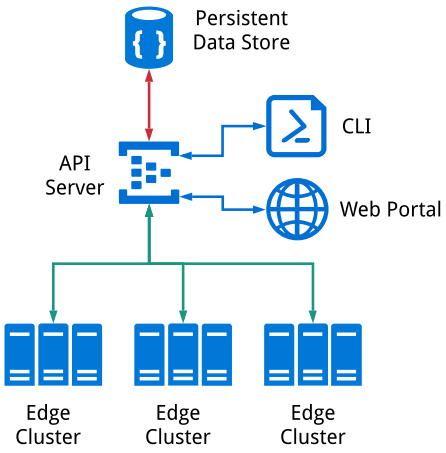

Figure 1: Structure of the SLATE federation. The central API server accepts user requests from the command line interface (CLI) and graphical interface (the web portal), looks up data as needed from the persistent store, generates commands to participating Kubernetes clusters, and sends back results.

The central SLATE component implements a REST API, accepts user requests, and forwards instructions to individual Kubernetes clusters as appropriate. Within the SLATE framework, a user or developer can typically generate requests through either a graphical web portal or a command line interface. The SLATE web portal and central services do not provide a custom authentication backend, and instead use federated identity as implemented by the Globus Research Data Portal [**?**]. The state of the SLATE platform is persisted in a database, currently DynamoDB [**?**], which provides scalability and independence from hardware at a single site. When determining whether to forward instructions to Kubernetes, the API server accepts the user requests, validates the user with federated identity access, and applies authorization rules to determine whether the user in question has sufficient privilege for the requested action. Groups defined in the SLATE platform determine the capabilities of users. These same groups also authorize the user to manage member clusters and application instances. Each resource provider which chooses to participate in the federation typically operates one Kubernetes cluster, on which multiple groups may deploy applications.

2.1 Application packaging

In order to deploy applications on the SLATE platform, developers containerize applications and package them through the use of Helm [**?**] charts and templates. Helm charts and templates are a simple format for providing some configuration values and for requesting Kubernetes object definitions. This approach makes application deployments simple and consistent. Charts are able to

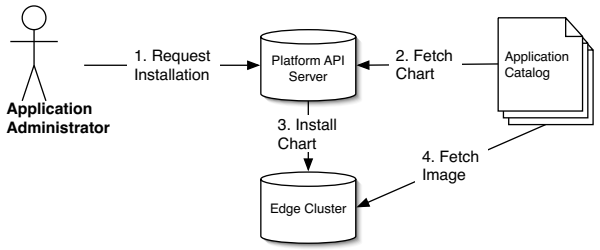

Figure 2: Application install process on the SLATE platform. Applications are made available in a standardized catalog, with the SLATE API server enforcing that applications are installed only from this source.

abstract application configuration settings out into a single document (in the form of a YAML [**?**] file). When the application is installed, the settings are substituted into the object definitions which are loaded into Kubernetes. Charts make usage simple for science users who are conversant with the configuration of the application (e.g. the amount of storage allocated for a data cache or the number of nodes to run in a distributed data analysis cluster) but may not be familiar with the syntax and structure of Kubernetes. Codifying applications into charts has the additional benefit that the chart represents a unified description of how the application will install. Another entity can then review and audit this unified description for several key security properties, such as code loaded and network ports utilized. The design and implementation of the SLATE platform will install only the charts which source from an application catalog maintained by the SLATE platform team (currently) to enforce a baseline level of trust in the software which it can run at participating sites, as shown in Figure **??**. Section **??** goes through the details of maintaining this catalog.

2.2 Cluster use permissions

When a Kubernetes cluster is first joined to the SLATE platform, by default only the administrating group who added it have authorization to deploy applications to it. These cluster administrators can then grant other groups access to the cluster, including having the option to open access to the cluster to all groups. Furthermore, cluster administrators may regulate which applications from the catalog each allowed group may install; by default each group which is allowed access may install any application, but the cluster administrators may choose to replace this universal permission with a specific whitelist of allowed applications on a per-group basis. These capabilities are intended to give the cluster administrators at each participating site the necessary oversight to enforce site-specific security policies beyond the basic trust level of SLATE users and applications. At this time the SLATE platform does not contain a specific, formal system for arbitrating these types of access being granted. Instead users within SLATE are given access to the contact information provided for other users and groups so that, for example, an application administrator who wants to deploy a application to a cluster in the federation can look up the group which administers the cluster and send a request for access, and the two parties can reach their own understanding of the scope and acceptability of the request.

2.3 Multi-tenancy

One of the key design goals of SLATE is to be able to support multi-tenancy; that is, multiple users of the SLATE platform deploying applications on the same participating Kubernetes cluster should not be able to interfere with each others' activities. Kubernetes provides two critical building blocks for protecting sensitive user data: namespaces [**?**] and secrets [**?**]. Namespaces provide scopes for organizing user objects. Secrets directly represent pieces of sensitive data (such as database passwords, certificates, etc.). Kubernetes guarantees that a secret in one namespace cannot be accessed by applications in another namespace, and SLATE therefore organizes applications run by different groups of users into separate namespaces. Additionally, sites wishing to participate in the SLATE federation may want to do so with general-purpose Kubernetes clusters which they already operate, and so they may wish to ensure that the SLATE platform itself cannot interfere with their other users. Normally, in order to create Kubernetes namespaces SLATE would require access to all namespaces on the cluster, meaning that a bug or security flaw in SLATE could lead to undesired access to or alteration of non-SLATE resources. This possible class of problems is mitigated through the use of an additional component, the NRP-controller, written by D. Mishin [**?**]. This controller supplies a pair of Custom Resource Definitions (CRDs) which extend the Kubernetes API to grant SLATE indirect access to create namespaces for its users, without granting it access to all namespaces. These particular implementation choices are specific to using Kubernetes as the underlying technology; with a different orchestration technology equivalent multi-tenancy isolation features would have to be used.

3 CATALOG CURATION

SLATE's design allows flexible deployment of new science services in close proximity to major computing resources, but this proximity brings additional security concerns. Because of the amount of computing and bandwidth available at such sites, administrators must be vigilant not only for malicious software directly abusing the resources, but also legitimate, but misconfigured, software creating openings for abuse. The fundamental concept of the SLATE application catalog is to limit applications to those which the platform operations team can certify meet security criteria required by resource providers. This concept requires application developers submitting their work for review by the platform team, and constructing mechanisms to ensure that only code which has passed the platform team's review can deploy. The review process must not be unduly burdensome to application developers, lest the overhead of working with the platform be greater than the value the platform can provide.

Deploying containerized applications to Kubernetes involves two levels of code which must be trusted, and therefore must be reviewed. The first is the actual executable code and its immediate configuration, which is embodied in a Docker [**?**] image, defined by a set of image sources (primarily a 'Dockerfile'). The second is the higher level configuration of the container within Kubernetes,

which (at present) SLATE treats in the form of a Helm chart. Helm charts offer the opportunity to separate the majority of application configuration from a reduced set of user adjustable parameters. While Helm has proven to be the most convenient mechanism presently available, SLATE's usage of it is fairly simple, and it is possible that it could be replaced with a different templating mechanism if a good alternative appears or SLATE's needs change. In order to ensure that an application is well behaved, the Helm chart must be reviewed to ensure that it provides a suitable configuration of the application, and exposes no adjustable parameters which would compromise that configuration. The Docker image sources must be likewise audited. Finally, steps must be taken to ensure that the Docker image which will be used in the future will correspond to the sources which are reviewed. This is required because images are typically fetched from central repositories (e.g. Docker Hub) when an application instance is deployed, and these repositories typically allow the image corresponding to a given identifier to be replaced.

To address all of these concerns, application image sources and Helm charts are maintained by the platform team in the application catalog in the form of a public repository (that is, readable by the public but modifiable only by the platform team), and the platform team publishes charts and images for use on the platform from this repository only. New applications (and changes to existing applications) are proposed by application developers, and reviewed and approved by members of the platform team. Typically, the repository would be hosted on a provider like GitHub [**?**], and proposals for additions and changes are made in the standard way supported by that provider, i.e. Pull Requests on GitHub. The SLATE project team provides and maintains a public repository of applications [**?**], though the pattern could be copied by an independent organization of federated sites to source specific applications from trusted sources. A complete workflow involving an Application Developer and a Platform Application Reviewer is as follows:

(1) Application Developer writes image sources, and tests locally until the image functions as intended
(2) Application Developer writes chart sources, tests locally until the chart installs cleanly
(3) Application Developer 'forks' the application catalog git repository
(4) Application Developer adds image and chart sources to his/her copy of the catalog repository (to the section for applications under consideration, the 'incubator' catalog)
(5) Application Developer submits a 'pull request' to the original catalog git repository to merge his/her additions
(6) Continuous integration system detects the request and performs a test build of the application image and chart, which includes simple automated checks (i.e. for valid syntax). The results of this test build are visible to the Platform Application Reviewer, and an indication of its success or failure to the public.
(7) Platform Application Reviewer examines the pull request. If all sources appear valid he or she merges the pull request
(8) Continuous integration system detects the change, publishes an updated version of the catalog, builds the image and publishes it to a Docker repository

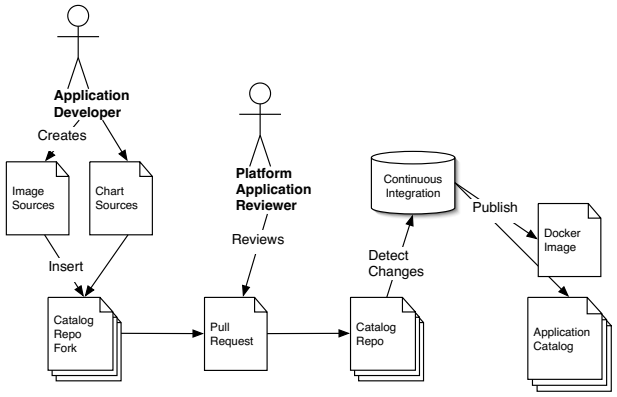

Figure 3: Application review process for the SLATE platform. An application developer submits suggested changes, including both sources for a container image and a Helm chart, a platform reviewer considers the changes, and after approval by the reviewer an updated version of the application catalog is automatically published.

(9) Platform Application Reviewer performs a review of the contributed chart and image sources to determine whether they meet all requirements for inclusion in the stable catalog
(10) Platform Application Reviewer moves the chart and image sources to the stable section of the catalog, if requirements are met.
(11) Continuous integration system detects the change, publishes an updated version of the catalog, and builds the image and publishes it to a Docker repository

A simplified diagram of this scheme is shown in Figure **??**. Essentially the same process is also relevant for changes to an existing application, with the science application developer preparing a set of changes and submitting a pull request for their inclusion, from which point the review steps would be the same except that basic information about the application would not generally need to be recollected.

When the Continuous Integration (CI) system performs its test build on submitted applications in Step **??** of the process, it must not be vulnerable to arbitrary code execution. As the build process is defined by configuration files which are themselves stored in the catalog repository, the CI system must be configured to ignore changes to these specific files, or to refuse to build proposed changes which include alterations to them. As long as the integrity of the build system itself is ensured, building the Helm charts executes no user defined code (barring vulnerabilities in Helm itself), and produces fixed output files which are known ahead of time. Building Docker images necessarily involves executing user-supplied code, however, this is contained within the scope of the container being created. Standard best practices should be followed to mitigate Docker container breakout exploits.

An important caveat is that most Docker images are based on other, more general images; for example, many application images are based on an operating system image. Inspecting the contents

of entire operating system distributions is clearly beyond the scope of the SLATE platform, and in addition it is already typical in scientific computing to trust these same distributions when they install directly on the hardware in a data center. Likewise, some applications which might be relevant for inclusion in the catalog are already major open-source projects with their own security and quality procedures, which are already widely trusted. Examples might include the Apache httpd [**?**] and NGINX [**?**] web servers. When such projects already prepare their own Docker container images which are well trusted by the computing community it is more desirable for the SLATE platform to make use of them than to make and attempt to maintain copies. Therefore, not all SLATE applications should be required to include image sources within the catalog if the images sources are maintained and the images published by a suitably reputable group. This is formalized by the platform reviewers maintaining a whitelist of trusted organizations, and application developers should be able to propose new additions to the whitelist. It is expected that when such an application is submitted to the catalog review process, the Application Developer making the submission would include a written explanation of why the images being used, and the organization which produces them should be added to the whitelist, and the Reviewer would take this into consideration.

Where possible it is desirable to automate investigation of container images to allow deeper insight than what the Reviewer can gain from a manual inspection, given limited review time per application. One class of tools available in this area are image vulnerability scanners [**? ? ?**]. These tools generally unpack images and compare the collection of software found to be installed against public databases of known software vulnerabilities, reporting any vulnerabilities known to be present in installed package versions. It is therefore useful to include such scanning in the set of automated checks performed by the application catalog's continuous integration system, but the output probably cannot be acted on in an automatic way. This is due to the fact that scanner tools are not able to usefully inspect all images (often due to the images being minimally constructed so that they do not include any package manager information about software versions) and images which can be scanned often produce large numbers of real but likely unimportant vulnerability warnings. As a result, the intention is to make the results of such scans available to Platform Application Reviewers and alongside applications which are accepted into the catalog, but it appears impractical to require that submitted images successfully pass a scan with zero known vulnerabilities. Instead, the Reviewer should consider whether detected vulnerabilities are sufficiently relevant that the image should be revised to eliminate them, and cluster administrators would also have the scan results available for when considering requests to run the application on their cluster. It would be natural to also provide in the same place any related information that participants in the platform may generate and contribute, such as results from audits performed on particular applications.

One possible area for the application contribution, review, and publishing process to be extended is verification of contributors' identities and authenticity of charts and images. By using private-key cryptography it could be possible for application developers to sign their contributions and then for the SLATE platform to verify that subsequent contributions are made by the owner of the same private key. When submissions are managed through a service provider like GitHub, which already carries out authentication, adding such steps would make little difference in security, and would add an additional burden on scientific application developers who do not typically employ cryptographic signing in their development workflows. However, they would be an important addition if such a service provider is not used. Likewise, it is possible for the SLATE platform to sign charts and images it publishes so that when used they can be verified not to have been tampered with after the platform's build process. It is difficult, though to provide any type of verification mechanism that the build artifacts were indeed created from the intended sources, particularly since in the case of the container images the 'sources' often implicitly include whatever package versions were most recently available from a particular Linux distribution package manager at the time the image was built. At this time SLATE does not employ cryptographic signing in either context, but this will be explored further in the future.

If problems do arise with an application after its publication in the catalog, such as security vulnerabilities or some type of malicious code being discovered, a few mitigations are possible. First, an application determined to be dangerous can be removed from the catalog until it is corrected, preventing it from being deployed any further. Existing deployments can be shut down, either by the service administrators who deployed them, or by the central administrators of the SLATE platform, depending on urgency. While SLATE cannot give particular insight into the behavior of each packaged application (such as interpreting its particular log files), it does record when any application instance is deployed or removed, which information could be made available to administrators of sites at which the suspect application had been running to assist them in investigating possible impacts.

4 SUMMARY

The SLATE platform facilitates efficient deployment of science services and applications by creating a trust relationship between application developers, application administrators, and computing resource providers. This trust is built from isolation of sensitive application inputs among user applications, providing controls to limit privileges on federation member clusters by groups, and management of a central catalog of reviewed applications. Not only is this type of federation capable of supporting both infrastructure services and domain applications, but multiple, independent federations can also be operated using the same tools, potentially with their own catalogs of applications. We are not aware of any other published solutions which address the same needs of building a trust relationship for multi-institution federation.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation Office of Advanced Cyberinfrastructure (OAC), grant number 1724821.

The authors would also like to thank T. Barton, A. Tiradani, B. Lin, B. Kalosinki, D. Mishin, J. Graham, K. Hurtado, M. Stanfield, K. Lannon, and R. Killen for insightful discussions on these topics which informed the designs outlined in this paper.