EPJ Web of Conferences **251**, 02041 (2021) https://doi.org/10.1051/epjconf/202125102041
CHEP 2021

Enabling interoperable data and application services in a federated ScienceMesh

Ishank Arora[1,*], *Samuel* Alfageme Sainz[1], *Pedro* Ferreira[1], *Hugo* Gonzalez Labrador[1], and *Jakub* Moscicki[1]

[1]CERN, 1 Esplanade des Particules, Meyrin, Switzerland

Abstract. In recent years, cloud sync & share storage services, provided by academic and research institutions, have become a daily workplace environment for many local user groups in the High Energy Physics (HEP) community. These, however, are primarily disconnected and deployed in isolation from one another, even though new technologies have been developed and integrated to further increase the value of data. The EU-funded CS3MESH4EOSC project is connecting locally and individually provided sync and share services, and scaling them up to the European level and beyond. It aims to deliver the ScienceMesh service, an interoperable platform to easily sync and share data across institutions and extend functionalities by connecting to other research services using streamlined sets of interoperable protocols, APIs and deployment methodologies. This supports multiple distributed application workflows: data science environments, collaborative editing and data transfer services.
In this paper, we present the architecture of ScienceMesh and the technical design of its reference implementation, a platform that allows organizations to join the federated service infrastructure easily and to access application services out-of-the-box. We discuss the challenges faced during the process, which include diversity of sync & share platforms (Nextcloud, Owncloud, Seafile and others), absence of global user identities and user discovery, lack of interoperable protocols and APIs, and access control and protection of data endpoints. We present the rationale for the design decisions adopted to tackle these challenges and describe our deployment architecture based on Kubernetes, which enabled us to utilize monitoring and tracing functionalities. We conclude by reporting on the early user experience with ScienceMesh.

1 Introduction

Enterprise file synchronization and sharing (EFSS) providers in the European landscape, such as Nextcloud, Owncloud, and Seafile, have become indispensable to research groups, scientists and engineers, owing to the ease of sharing, transferring and synchronizing petabytes of data and deploying applications to work on it. Especially during the pandemic, there has been a significant increase in the use and implementation of small local ecosystems of tools for research collaboration. Typical deployments, however, suffer from the problem of service fragmentation, as they consist mostly of ad-hoc binding implementations of multiple user-facing applications and storage backends. While this approach works well in the short

*e-mail: ishank.arora@cern.ch

term by reducing the time taken to take the system to production, it introduces multiple problems [1]:

1. Integrating new application services developed at other sites is a cumbersome process as the implementations cater to specific infrastructures and use-cases and are, therefore, independent from each other. This makes portability across EFSSs extremely difficult.

2. Users don't have the freedom to work with an application of their choice as the system only provides bindings with a single option. This issue has been plaguing a widespread migration to cloud-based services.

3. Collaboration is restricted to just the local consumers, imposing barriers which limit it to the institutional level.

To address these issues and to enable interactive and agile sharing across deployments, the CS3MESH4EOSC [2] project aims to combine various systems and services to boost open-science by developing a joint, coordinated service for users in the research and education space at a pan-European level. The goal of the project, funded by the European Union, is to develop the ScienceMesh service, a federated mesh which would allow users to sync and share data, and consume multiple applications across deployments all over Europe.

1.1 Use Cases

ScienceMesh aims to go above and beyond the functionalities provided by the commonly used cloud sync platforms, providing tools and applications focused on improving the degree of collaboration in the research ecosystem, especially the HEP community, by leaps and bounds. Initially, focus has been directed to the following data services and efforts for their development are in full swing:

- Federated, secure sharing and synchronization of data across different sync-and-share providers.
- Real-time collaborative editing through open-source office suites such as Collabora and Codimd, data archiving and publishing through platforms such as Zenodo.
- Data science environments, such as SWAN [3], for data processing and analysis in JupyterLab notebooks.
- On-demand data transfers, moving large datasets across countries using tools such as Rclone and Rucio [4].

The rest of the paper is organized as follows: we start by presenting the architecture and workflows involved in the ScienceMesh service in Section 2 and the technical aspects behind this architecture in the reference implementation, which serves as the backend for the aforementioned use cases, in Section 3. We describe our Kubernetes-based deployment infrastructure in Section 4 and conclude by describing the initial user testing with the platform and the next steps in the project in Section 5.

2 CS3MESH4EOSC: Design and Workflow

To facilitate more EFSS providers to join the federated mesh, we utilize existing standard protocols wherever appropriate and work on maturing and extending protocols already in use by the CS3 community [5]. As a design principle, industry standard transport protocols (for example, REST and gRPC) are used to interconnect server-side components and to achieve

maximal interoperability. The choice of the transport protocol is based on the following criteria: quality of implementation, wide adoption in the industry and in open-source projects, support for multiple programming languages (to increase integration with all systems based on different technologies) and enterprise-grade tooling (debuggers, testing frameworks).

2.1 Open Cloud Mesh (OCM)

Open Cloud Mesh (OCM) [6, 7] aims to develop a web-based protocol for universal file access beyond individual cloud deployments that would support a globally interconnected mesh of such deployments without sacrificing any of the innumerable advantages in privacy, control and security that an on-premise cloud provides. OCM defines a vendor-neutral, common file access layer across an organization and/or across globally interconnected organizations, regardless of the user data locations and choice of clouds.

To keep the API specification minimal, only the endpoints to enable receiving and listing shares from other mesh providers have been added [8]; the initiation of share creation by the users has been left to the internal mechanism adopted by the individual EFSS providers. When a user initiates an OCM share, their EFSS provider sends a POST request to an endpoint exposed by the recipient's provider. These endpoints can be retrieved either from the local database or a central registry to which all providers must register. These remain unprotected as the original user's provider won't have any information about the authentication mechanisms followed on the others. While no authentication happens between any two providers, there are, however, access control mechanisms in place to prevent DOS attacks and bogus requests, which are discussed in Section 3. A share is created only when the calls to both mesh providers complete successfully, ensuring data consistency.

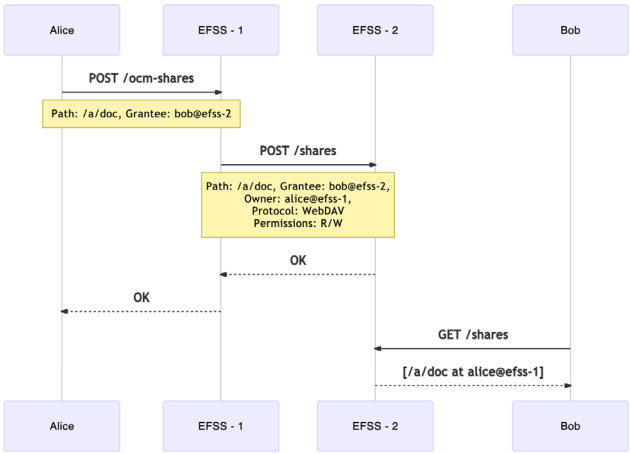

Figure 1. OCM Share workflow across providers in ScienceMesh

2.2 User discovery

The existing OCM protocol implements sharing that requires the "origin" user to know the exact federated identity of the party in the remote system, as is evident in Figure 1. While this requirement seems natural from the point of view of the design of an API which is meant to

be simple and platform-agnostic, from a user experience point of view such knowledge cannot be reasonably expected, since identity management strategies will vary across providers and often involve seemingly random unique identifiers which would be almost always impossible to predict by the sharing party. Multiple approaches were considered and evaluated to introduce a discovery mechanism to OCM:

- Having a central registry which enables searching for users and groups. While this is the most straightforward approach, it would introduce multiple constraints and require additional infrastructure, not to mention scalability issues. It would involve exposing information about local legacy users to other deployments, to which not every user might consent. Furthermore, the same user may be registered across multiple providers with the same identifiers, which would significantly degrade the user experience associated with the search process.

- Having a workflow which allows users to be "invited" in order to start collaborating. Through invites shared via any third-party communication channel, the recipient would be redirected to a service which would allow them to "reveal" and prove their identity via their cloud provider. This would solve the issue of consent and users having multiple accounts, but would have the downside of introducing a dependency on an additional central service.

The approach adopted for ScienceMesh starts from the second option described above and tries to improve on the scalability aspect by removing the additional dependency on the central server. As an alternative, each provider itself exposes a service which allows the recipient to choose the system through which they want to collaborate, in case they have accounts on more than one. Once the recipient is authenticated to their system, this is communicated to the host provider, and the invitation workflow is complete.

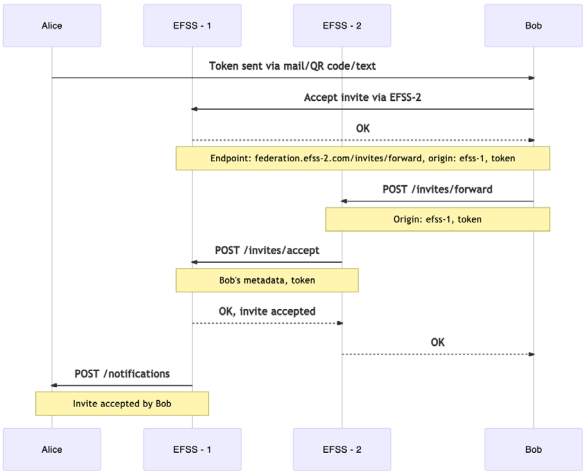

Figure 2. Creating and accepting invites to initiate OCM sharing

2.3 File manipulation over WebDAV

The aforementioned invitation and OCM share workflows enable the required metadata exchange, keeping in mind the need to share as minimal information as possible to preserve user privacy. These involve sending POST requests to the other mesh providers and thus won't create any significant bottlenecks as far as performance is concerned. The actual file access and manipulation, however, can prove to be extremely resource-heavy and thus needed time-tested protocols to be implemented. ScienceMesh currently serves these over the WebDAV [9] protocol. WebDAV (Web Distributed Authoring and Versioning) extends the Hypertext Transfer Protocol (HTTP) by adding functionalities which allow clients to author remote Web content. These functionalities include maintenance of metadata such as authors, modification date and time, namespace management, collections, and permissions enforcement.

The Interoperability Platform (Section 3) implements a custom WebDAV service which abstracts the underlying details from the end users. When the originating system creates a share on the recipient's provider, a JWT [10] token is transported which contains encrypted information about the resource path, namespace, permissions and the endpoints for accessing it. When the recipient wants to access the shared resource, this request forwards this token back to the host system, where it is translated by a storage registry (Section 3.2), and the resources are served accordingly.

3 Implementation of the Interoperability Platform

We now present a reference implementation for the complete architecture, developed with the aim to facilitate joining the federated service infrastructure by interested organizations and to encourage wider adoption of the OCM protocol. The implementation is based on the vendor-neutral Cloud Sync and Share Services APIs (CS3APIs) [1, 11] interfaces and is present in the Interoperability Platform (IOP) called Reva [12].

3.1 CS3APIs

CS3APIs provide programming interfaces to define and connect various services to ensure interoperability in a sync & share system. These connect various microservices, from storage backends and authentication mechanisms to front-end applications and sharing drivers, in a vendor-neutral format over a well-defined metadata control plane. Thus, these enable porting applications adhering to certain specifications across platforms, allowing administrators to significantly widen the scope of services being offered by integrating new applications and data services seamlessly.

CS3APIs messages are defined using Protocol Buffers, which allows encoding structured data in extremely efficient formats, and removes the limitations associated with the choice of technology stacks by providing language interoperability. gRPC is the chosen transport protocol for inter-component communication.

In addition to the already existing interfaces spanning the breadth of sync and share system requirements, we introduce new definitions for OCM-specific endpoints which would allow multiple drivers to be easily plugged in. These include:

- **OCM Invite API:** Provides endpoints to generate invite tokens, forward recipient metadata to other mesh providers and complete the invitation workflow.
- **OCM Share API:** These are endpoints which are utilized by EFSSs to create internal OCM shares. These are also responsible for triggering actions on the remote providers to ensure consistency across providers. These also store share references which are translated to WebDAV endpoints through which the shared resources can be manipulated.

- **OCM Core API:** These unprotected endpoints (Section 2.1) are exposed by all providers registered in the mesh, using which other trusted mesh providers can create remote shares on their site.
- **OCM Provider API:** These are utilized to retrieve metadata exposed by the mesh providers and verify incoming requests, based on methods described in 3.3.

3.2 Reva

Reva is a microservice-based implementation of CS3APIs which has been powering the backend at CERNBox [13] since 2018, a sync and share platform which supports 37,000 users, accessing over 12 PB of data. It also serves as the storage and applications component for the new flagship ownCloud platform, oCIS [14]. It serves as the entry point multiple driver implementations of each of the CS3APIs interfaces which can be served simultaneously (Figure 3). This allows EFSSs to connect directly to it via CS3APIs and get access to a rich ecosystem of applications without worrying about their inner workings. Reva operates using dynamic rule-based registries which redirect the interface requests to the corresponding drivers based on an administrator-set configuration, client headers or user agents. Service discovery is enabled via configuration files but a dedicated microservice which would allow arbitrary services to register themselves is also planned.

The OCM protocol has been integrated with Reva, which allows reusing state of the art software, and access to functionalities not specified in the OCM specification. These added functionalities can vary across the various providers registered in the mesh, and discussing those is out of the scope of this paper. The existing application bindings are also available out-of-the-box, which significantly expands the breadth of use cases fulfilled in ScienceMesh.

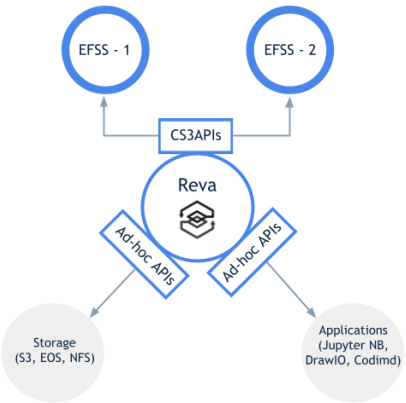

Figure 3. Reva: interoperability across interfaces

3.3 Access Control over Unprotected Endpoints

As described in Section 2, both the OCM share and invitation APIs need to expose unprotected endpoints which are accessed by other providers in the federated mesh. This lack of authentication introduces multiple security concerns that can be exploited by attackers. We employ the following measures to counter these:

- Rate limiting the number of requests to the metadata endpoints. Since shares and invites are initiated manually, an increase in the access rate to the corresponding endpoints may suggest a DOS attack. This limit has to be applied to the unprotected endpoints, while data access through WebDAV is protected by bearer tokens and can't be placed under this restriction as multiple applications might require read or write access at high frequencies.
- Shared API keys between the different providers can be used to establish trust between pairs of providers. As mentioned in Section 2.1, these keys are part of provider metadata which can either be retrieved locally or polled from a central service. While the former approach is bound to fail if there's a change in the keys or if a new provider registers in the mesh, the latter places a lot of trust on the central server, which might become a single point of failure. The keys might also be leaked in cases of man-in-the-middle (MITM) attacks, which can then be utilized to propagate bogus data.
- Another access control mechanism is based on reverse lookups of the incoming request's host. This would require knowledge of the domains of all the registered providers and making sure that the request initiator can be resolved to these. This approach takes care of the invulnerability to MITM attacks.

4 Deployment Reference

A very relevant aspect to consider while designing complex software systems is how they are deployed and operated over time. As pointed in previous sections, we have relied on well-established open-source projects as the mesh building blocks, and when it came to deploying those blocks, we chose to be consistent with that same approach. With this in mind, we focused on two mature technologies that have gained popularity and adoption in the last decade:

- **Application containers:** While container runtimes are not a new concept, the Docker project put them back in the spotlight by providing a simple but powerful packaging and a distribution model for the generated artifacts.
- **Container orchestration:** It offers a set of primitives that can be perceived to provide operating-system-like APIs and features (scheduler, jobs, etc.) to manage container workloads distributed across a cluster. It reduces the toil of managing and scaling complex service topologies and provides built-in extensibility and policy-enforcement mechanisms.

4.1 Challenges of Cloud-Native Adoption

To embrace the technologies just described, we evaluated Kubernetes as the ideal platform to deploy the sites participating in the CS3MESH4EOSC testbed. Maintaining comprehensive documentation [15] to use these abstractions is also crucial for this assumption to work. While there are several reasons to back up this decision, it is also relevant to describe some of the challenges that it poses. Being such a powerful but recent and evolving technology, Kubernetes still presents a steep learning curve, with most organizations still evaluating it prior to adoption. Also, some aspects related to networking and storage are still not as mature as they should be out-of-the-box, requiring additional components to overcome those.

4.2 Packaging and Distribution

Given the substantial challenge of deploying the project on such a spectre of organizations, we tried to lower the entry barrier for adoption as much as possible while providing simple but powerful configuration options.

The Helm project defines itself as a 'Kubernetes package manager' and was selected for this task primarily for its simplicity and composability, as well as the broad community adoption and learning resources. The following taxonomy is a bottom-up summary of where each artifact fits in this strategy.

1. **Container images:** The atomic functional units. These are publicly available [16] and can be deployed in a cluster using pods.

2. **Helm charts:** Use templates to instantiate and configure application manifests that consume the container images based on the user-provided values.

3. ***"Umbrella"* charts:** Meta-packages, collections of charts that can be used to:

   - Reuse existing, official or community-contributed packages to create semantically-related collections of third-party software that share a context.
   - Hide the fine-grained dependency details.

Initial versions of the charts deployed a single daemon architecture with minimal configuration options and appropriate defaults, backed by ephemeral storage. Persistent storage through independent storage providers supported by either native Kubernetes entities [17] or by network file systems, and configured with separated concerns was subsequently introduced as the project grew. These charts reduce the complexity of traditional configuration management systems to a set of YAML files, ensuring that users do not require prior knowledge of the platform to deploy the applications.

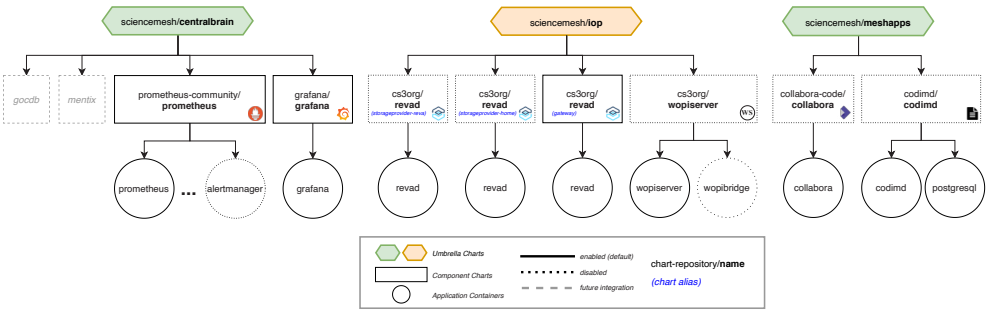

Figure 4. ScienceMesh project chart topology.

Using the IOP umbrella chart [18], a site can be deployed, upgraded, or configured on an existing cluster, extended and integrated with supported applications, all with a single command. This principle also applies to the central monitoring component, deployed through the 'Central Brain' chart. In figure 4, we describe some of the relationships between the project charts and their components.

4.3 Observability and Telemetry

To monitor user interaction and activity, we need techniques that capture and describe the internal state of such a complex system. The 'observability' term has been broadly adopted recently to describe such needs. It seats on three pillars: metrics, logs and traces. While the gathering of the first two is well-established in IT operations teams, the complexity introduced

by distributed architectures has also created the need for more powerful tools, capable of preserving the relevant context of a distributed transaction across the system.

To achieve the desired level of visibility for these sites, we devised a "central brain" component, composed of a subset of projects from the rich Prometheus ecosystem. This involved the integration of probes guided by alarming thresholds to determine the behaviour of the platform from an admin perspective. Following the umbrella-chart model, a similar package to simplify the deployment of this monitoring infrastructure was published.

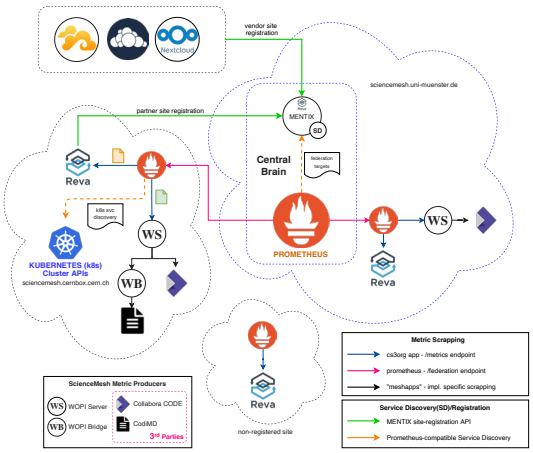

Figure 5. ScienceMesh Federated Monitoring Architecture

Once the sites were complemented with increasing applications and components, the choice of Prometheus proved its value by being aligned with the building principles of ScienceMesh: scalable and decentralised. A lightweight Prometheus server instance on each site aggregates disparate metrics for a short retention period and is the entry point for the central server, fine-tuned for increased retention and integrated with alarming and visualization tools, such as Grafana. A different approach is applied for logging, where the centralization of the logs though projects like Logstash [19] or Grafana Loki [20], should be made on a per-site basis to prevent the disclosure of user-sensitive information. Lastly, we also treat the tracing instrumentation using the OpenTelemetry [21] and Jaeger projects as first-class citizens throughout the project, acknowledging the added value that the correlation of events can provide to detect and troubleshoot issues.

5 Preliminary Tests and Conclusion

Implementation of the OCM protocols interacting over CS3APIs in the IOP has enabled us to work towards making these use cases available across the 8 early adopter sites, which are based on different EFSS platforms - CERNBox, SURFdrive, PSNCBox, CloudSTOR, Sciebo, owncloud@CESNET, SWITCHdrive and ScienceData, while other partners are about to join. Initial automated as well as manual rounds of tests across these sites covered many scenarios, from validating the registration of new sites in the mesh, to issuing OCM share requests and manipulating resources over WebDAV. Various bugs and missing functionalities were reported during these tests and fixed, making the platform more robust. Collaborative editing using the Collabora CODE [22] application was tested across multiple sites as well, a small demonstration of one of the many the use cases that ScienceMesh can enable.

Efforts into developing bindings for more research-oriented applications and attracting members from the HEP community to join the mesh will continue over the next few years. These developments would enable us to scale our platform to a much larger audience, enabling true collaboration among researchers all over Europe.

Acknowledgements

CS3MESH4EOSC has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 863353.

References

[1] H.G. Labrador, J.T. Mościcki, M. Lamanna, A. Pace, *Increasing interoperability for research clouds: CS3APIs for connecting sync&share storage, applications and science environments*, in EPJ Web of Conferences (EDP Sciences, 2020), **Vol. 245**, p. 07041, https://doi.org/10.1051/epjconf/202024507041.

[2] CS3MESH4EOSC, https://cs3mesh4eosc.eu/, accessed: 2021-02-25

[3] D. Piparo, E. Tejedor, P. Mato, L. Mascetti, J.T. Mościcki, M. Lamanna, *SWAN: A service for interactive analysis in the cloud*, in Future Generation Computer Systems, **Volume 78**, Part 3, 2018, Pages 1071-1078, https://doi.org/10.1016/j.future.2016.11.035.

[4] M. Barisits, T. Beermann, F. Berghaus, B. Bockelman, J. Bogado, D. Cameron, D. Christidis, D. Ciangottini, G. Dimitrov, M. Elsing et al., Computing and Software for Big Science **3**, 11 (2019)

[5] CS3 Community, https://www.cs3community.org/, accessed: 2021-02-25

[6] OCM, https://wiki.geant.org/display/OCM/Open+Cloud+Mesh, accessed: 2021-02-25

[7] P. Szegedi, OCM, https://indico.cern.ch/event/565381/contributions/2401967/, CS3 Workshop, 2017

[8] CS3MESH4EOSC: Initial definition of protocols and APIs, https://www.cs3mesh4eosc.eu/sites/default/files/2020-10/d3.1_cs3mesh4eosc.pdf, accessed: 2021-02-25

[9] WebDAV, https://tools.ietf.org/html/rfc4918, accessed: 2021-02-25

[10] JSON Web Token, https://tools.ietf.org/html/rfc7519, accessed: 2021-02-25

[11] CS3APIs, https://cs3org.github.io/cs3apis/, accessed: 2021-02-25

[12] Reva, https://cs3org-reva.netlify.app/, accessed: 2021-02-25

[13] L. Mascetti, H.G. Labrador, M. Lamanna, J.T. Mościcki, A. Peters, *CERNBox + EOS: end-user storage for science*, in Journal of Physics: Conference Series (IOP Publishing, 2015), **Vol. 664**, p. 062037, https://doi.org/10.1088/1742-6596/664/6/062037.

[14] oCIS, https://owncloud.dev/ocis/, accessed: 2021-02-26

[15] Sciencemesh: Kubernetes Deployment, https://developer.sciencemesh.io/docs/iop/deployment/kubernetes/, accessed: 2021-02-26

[16] CS3 Org docker images, https://hub.docker.com/u/cs3org, accessed: 2021-05-06

[17] Kubernetes Persistent Volumes, https://kubernetes.io/docs/concepts/storage/persistent-volumes/, accessed: 2021-05-06

[18] Artifact Hub: Sciencemesh organization, https://artifacthub.io/packages/search?org=sciencemesh, accessed: 2021-02-26

[19] Elasticsearch Logstash, https://www.elastic.co/logstash, accessed: 2021-05-06

[20] Grafana Loki, https://grafana.com/oss/loki/, accessed: 2021-05-06

[21] OpenTelemetry, https://opentelemetry.io/, accessed: 2021-02-26

[22] Collabora, https://www.collaboraoffice.com/, accessed: 2021-02-25