EPJ Web of Conferences **245**, 07041 (2020) https://doi.org/10.1051/epjconf/202024507041
CHEP 2019

Increasing interoperability for research clouds: CS3APIs for connecting sync&share storage, applications and science environments

Hugo González Labrador[1,*], *Jakub T.* Mościcki[1], *Massimo* Lamanna[1], *Alberto* Pace[1],

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

Abstract. Cloud Services for Synchronization and Sharing (CS3) [14] have become increasing popular in the European Education and Research landscape in the last years. Services such as CERNBox, SWITCHdrive, SURFdrive, PSNCBox, Sciebo, CloudStor and many more have become indispensable in everyday work for scientists, engineers, educators and other users in public research and education sector.
CS3 services are currently too fragmented and lack interoperability. To fix this problem and interconnect storage-, application- and research services a set of interoperable interfaces, CS3APIs [10], has been developed.
CS3APIs enable creation of easily-accessible and integrated science environments, facilitating cross-institutional research activities and avoiding fragmented silos based on ad-hoc solutions.
In this paper we introduce the CS3APIs and its reference implementation, Reva [16].

1 Introduction

Cloud Services for Synchronization and Sharing (CS3) [14] have become a commonplace in the European Education and Research landscape in the last years. CS3 services represent an important part of the EFSS market segment (Enterprise File Sync and Share). According to the report at the CS3 2019 conference in Rome[2], 25 sites provide a service to the total of 395 thousand researchers and educators around the globe (in Europe and Australia, China, US, Brazil, South Africa and Russia) serving 2.7 billion files (corresponding to 11.5 PB of storage). CS3 provides easily accessible, sync&share services with intuitive and responsive user interfaces.

CS3 services are based on software provided by several EU-based companies such as ownCloud[3] (DE), cite[5] (DE), PyDio[7] (FR), Powerfolder[4] (DE) and China-based Seafile[6]. All are based on the open source, or open core, software model and are available for deployment at the customer's site(s).

Although these services are becoming popular because of their intuitive interface for sharing and synchronization of data, availability on all platforms (mobile, desktop and web) and capabilities to adapt to different user scenarios such as offline work, the commercially developed sync&share platforms are not at all integrated with research services, tools and applications. This lack of integration currently forms a major bottleneck for European collaborative

*e-mail: hugo.gonzalez.labrador@cern.ch

research communities. In addition, services as operated by several European providers who are in CS3, are currently too fragmented. This poses the following problems:

1. researchers face isolated service islands which are not able to interconnect and extend beyond the institutional boundaries; hence the researchers are not able to use services that may be available at another site, even if they do collaborate scientifically with researchers at these sites;

2. sites and e-Infrastructure providers are not able to easily integrate new application services, developed at other sites which use different technology solutions and thus they risk of providing inadequate service or drastically increasing the service costs;

3. research institutions still run the risk at becoming locked-in to a single provider;

4. the application developers are not able to benefit from the economy of scale and need to re-develop applications for multiple solutions, duplicating the investment and effort;

5. the technology companies are not benefiting from the contributions and know-how from the research community to enrich their service offering and possibly translate them into commercial and business applications because of lack of documented and standard APIs and lack of inter-operability.

6. the freedom of the users to choose a specific application to work with was lost with the migration to cloud-based services.

The problem of isolated service islands (1) is addressed by the Open Cloud Mesh initiative[17], a joint project under the umbrella of GEANT organization.

In the following sections we elaborate on the CS3APIs which address the problems 2-5; and, in section 5, on the "Bring Your Own Application" concept to address problem 6.

2 Background

The CS3APIs have their roots in the research project ClawIO[18], a cloud synchronization benchmarking framework. The concepts and technologies introduced in ClawIO alongside the operational experience of running a multi-petabyte cloud collaborative hub at CERN, CERNBox [19, 20, 23], led to the creation of the CS3APIs. The CERN implementation of CS3APIs is called Reva [16].

The CERNBox production service is powered by the CS3APIs since end of 2018 for streamlining operational procedures and reducing integration costs with other services. In addition CERNBox integrates end-user applications in a flexible way using application proxy services defined by the CS3APIs. For example, an application provider component hosting the OnlyOffice service backend for document editing is connected to the CERNBox service and the underlying Owncloud platform. CS3APIs allow to maximize portability of such integrations across different application providers and different platforms.

3 Interoperable platform APIs

CS3APIs provide an interoperable connection among sync and share platforms, storage providers and application providers. This allows to decrease the burden of porting applications developed for different sync and share platforms. This also allows connect the sync and share platforms for existing or new storage repositories over a well-defined metadata control plane.

For site administrators of sync and share services, CS3APIs allow to reuse existing connectors to other research services (such as digital repositories) independently of the technology stack. Furthermore, CS3APIs allow to easily integrate the "Bring Your Own Application" concept with existing sync and share services and present the users with a more productive environment. Finally, interoperability decreases the risk of vendor lock-in and ease migration to other CS3APIs-compatible software stacks.

4 API Specification

This specification is designed for use with gRPC[11] as transport protocol and Protocol Buffers[13] as the message definition language. The use of the CS3APIs over any protocol other than gRPC and Protocol Buffers is out of scope of this document.

The APIs are structured into various business domains that map one to one to gRPC service definitions. Figure 1 shows the logical layout for the APIs.

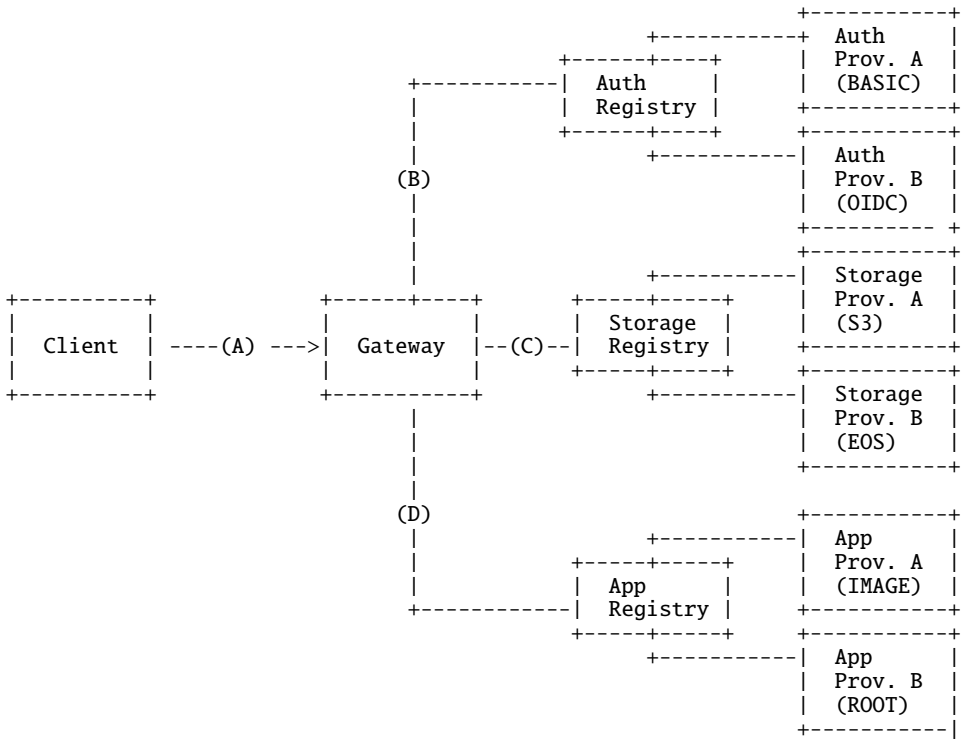

Figure 1: API logical domains

(A) A client contacts the Gateway API for any interaction with CS3 components. The Gateway API facade abstracts complex logic (storage re-directions, retries, ...) from the end client.

(B) When the client authenticates the Gateway asks the Auth Registry for a suitable Auth Provider handling the authentication strategy specified by the client (such as basic au-

thentication or Open ID Connect). Once the client is authenticated it may now perform actions against the Gateway.

(C) The client issues a storage operation such as "list my /home/Docs folder". The Gateway then asks the storage registry for the Storage Provider handling the "/home" namespace and the operation will continue with that Storage Provider.

(D) The client asks the Gateway for a suitable application to edit a particular file type. The Gateway asks the Application Registry for a suitable Application Provider providing an editor/viewer for such file type.

4.1 Storage Registry API

The Storage Registry API is responsible for the discovery and addressing of the different storage providers of the system. A client can use this API to list of all available storage providers and their capabilities, allowing the client to perform better decisions on data placement. The storage registry can be implemented in such a way to allow the auto-discovery of nodes using well-known technologies in the industry such as Consul[8] and etcd[9].

4.2 Storage Provider API

The Storage API is responsible for the manipulation of an underlying storage system (Amazon S3, EOS, local storage, ...) for all metadata-related operations. Data transfers are fulfilled by an out-of-band mechanism that allows to use any data transfer protocol. This mechanism works as follows: when a client wishes to initiate a data transfer, it will contact the storage provider and this will return to the client an URI[27] that contains the details of the transfer, including the protocol to use, like HTTP[22] or XROOTD[21] among others. The collection of storage providers creates a distributed unique namespace that clients can discover using the Storage Registry API. Every storage provider is exposed under a unique path in the distributed setup, for example, one storage provider may expose a root path named "/home" and other storage provider may export a root path named "/scratch". A client can ask a storage registry for the list of storage providers and decide where to place the data based on the capabilities and functionalities offered by the storage providers (latency, durability, replication,...).

4.3 Authentication Registry API

This API is for discovering the different authentication strategies that the client could use to authenticate against the cluster.

4.4 Authentication Provider API

The authentication provider service is responsible for the authentication of clients by following any authentication strategy (basic authentication, OIDC, ...). The service will mint an access token to use for subsequent requests.

4.5 Application Registry API

This API is responsible for the discovery and addressing of the different application providers of the system. When a client (usually a browser) wants to open or edit a specific file type it will ask the application registry for an application provider that handles the desired file type. For example, a user clicks a file named "Revenues.md" in their cloud storage platform and a request is made to the registry asking for an application that can handle this type of file. The registry finds a suitable application and returns to the caller the location of the app, so the browser can load it for the user to use it.

4.6 Application Provider API

The application provider API is responsible for handling the creation of an HTML application for viewing and editing files of a specific mime type or file extension. The provider will render the necessary HTML so the caller can display the application in the browser.

The combination of the Application Provider and the Application Registry allows to bring any application closer to the user. Applications available in a cloud storage are usually configured system-wise by an administrator, however, these APIs allow to dynamically add applications to the user environment. We coin this concept "Bring Your Own Application" and is further explained in the next section.

5 Bring Your Own Application

CS3APIs enable end-users to bring their own applications to the cloud.

In a non-cloud environment, such as a workstation or a personal laptop, end-users have the freedom to choose the application they wish to manipulate a particular file. For example, some users use Photoshop and others use Pixelmator to edit images, but is up to them to decide what to use. This freedom was lost when moving users to cloud-based services; it is the administrator of the cloud that decides for the users what applications are available.

This lack of freedom creates two problems. Firstly it forces the users to learn new tools despite that the may already be productive with other tools. Secondly, the provided application may not satisfy the expectations of the users. This may then lead to an inefficient workflow: a user needs to first download a file from the cloud to the local computer, manipulate the file locally and upload the file again to the cloud.

In the closed-source software environments, companies such as Google or Dropbox are already offering some degree of freedom on this capability with their services. They provide an API to connect applications to their storage services to allow creating new workflows on top of their platforms. However, the applications created for these platforms are locked-in to a specific vendor, therefore, re-using one application from one cloud to another is not possible. Thus an opportunity to benefit from an open source community and sharing of application components is lost, unless all users agree to use only one platform.

"Bring Your Own Application" provides a complementary functionality for open-source sync and share platforms and for the existing ecosystem of applications. For example, a CS3 community could offer an image editing service that can be used by end-users from their respective clouds, without having to ask the cloud administrator to install a specific application and compatible with all existing sync and share technology stacks (e.g. Owncloud, Nextcloud, Seafile).

Figure 2 shows the flow of the Bring Your Own Application.

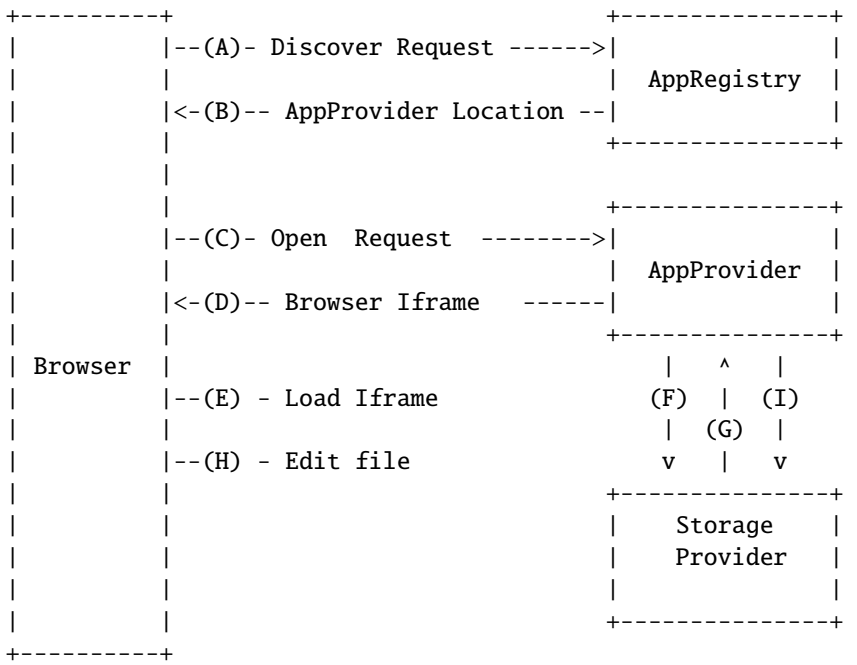

Figure 2: Bring Your Own Application flow

(A) The browser asks the Application Registry component to find an application provider that can handle a specific mime type. For example, where is the provider to edit a file with mimetype application/text?

(B) The Application Registry answers with the address of an application provider that will handle this mimetype.

(C) The Browser opens a request to the Application Provider. The open request contains the location of the storage provider storing the resource.

(D) The Application Provider answers with an HTML IFrame to be rendered into the browser to view or edit the file.

(E) The browser renders the IFrame

(F) The Application Provider will ask the Storage Provider for the file.

(G) The Storage Provider returns the file contents and the IFrame renders the file contents.

(H) The user will manipulate the file in the browser through the IFrame

(I) The Application Provider will save the new data to the Storage Provider.

The concept of Bring Your Own Application is closely related to unhosted web apps[12] and the Solid[24] project from Tim Berners-Lee. The unhosted web apps, also known as “serverless”, “client-side”, or “static” web apps, allow to choose a storage server provider independent of which applications users choose to use. Solid aims to allow users to choose

where their data resides and who is allowed to access it by decoupling content from the application itself. Vendor-locking is avoided and users are able to switch between applications and data storage servers seamlessly.

6 Technology Stack

6.1 Protocol Buffers and gRPC

The APIs use the Protocol Buffer language definition for defining the message payload consumed by all of the API methods. The use of Protocol Buffers over other message encoding mechanisms like JSON or XML was chosen for the following reasons:

1. Protocol Buffers encode structured data in an efficient yet extensible format. The data is described using a simple language to model messages in a namespace called a package. Encoding the semantics of the business in proto files will ensure that the logical boundaries created are enforced.

2. Protocol Buffers allow for a high degree of backwards compatibility. The fields in a proto file are numbered, thus avoiding version checks, which is one of the explicitly stated motivations for the design and implementation of Protocol Buffers. Therefore, new fields are easily introduced, as intermediate servers pass them through without needing to know about all the fields.

3. Protocol Buffers reduce boilerplate code. The usage of other encodings like JSON or XML oftentimes require the development of boilerplate code for enconding and decoding message from these encodings into data structures of the desired programming language. The generated code from Protocol Buffers will almost never be modified, thus allowing to focus on the business rather than writing boilerplate code.

4. Protocol Buffers provide language interoperability. Protocol Buffers are implemented in a variety of languages, they allow to create a polyglot mesh of services that can interact between each other. The usage of other encodings across languages requires often boilerplate code to map language types from one language to another.

The APIs use gRPC as transfer protocol. gRPC is a RPC platform originally developed by Google (under the Cloud Native Computing Foundation since 2017) which was announced and made open source in late Feb 2015. We chose gRPC over plain HTTP for the following reasons:

1. If gRPC is used with Protocol Buffers as the IDL (Interface Description Language), gRPC services are automatically generated for various programming languages. As the CS3APIs are defined in Protocol Buffers, using the same proto files to also define the service interactions increases the maintainability of the system.

2. gRPC is based on the HTTP/2 [25] industry standard. The fact of being based on HTTP/2, which is a binary transfer protocol rather than text based, allows more efficient communications over the wire. Another benefits are the built-in capabilities like headers compression, persistent TPC connections, cancellation and timeout contracts between clients and servers and built-in flow controls on top of HTTP/2 data frames.

3. gRPC supports unary and streaming communication flows. Unary communications are synchronous request-response style interactions. Streaming communications are powerful and gRPC supports these configurations: client-side streaming (client pushes messages to the server), server-side streaming (server pushes messages to the client) and bi-directional streaming (client and server push messages to each other). The streaming capabilities of gRPC are a good solution for some of the challenges faced many times on big storage deployments where a client may ask for a big list of resources.

   Consider the example of a client listing the contents of a folder containing millions of entries. In a plain HTTP connection, the response needs to be paginated, bringing more complexity to the client as it needs to perform multiple requests to obtain a full response. By using gRPC server streaming, all the files can be streamed to the client without performing additional network round trips, reducing latency and complexity.

   On the other side, sometimes a client wants to send various resources to the server. With protocols that do not support client-side streaming like HTTP, a frequent solution is to bundle requests into one, increasing the complexity both client and server side. Using gRPC client-side streaming there is no need for bundling as the underlying connection is persistent and messages can be streamed to the server as soon as they are ready on the client side.

4. gRPC has built-in support for client-side load balancing and request retry strategies with configurable back offs. These capabilities reduce the boilerplate code needed when instrumenting communications to other services. As timeouts often happen in the network, offering transparent retries for requests allows the service to hide transient errors to clients.

5. Fallback to plain HTTP and JSON: despite not being part of the gRPC core, the gRPC community has developed a fallback mechanism to plain HTTP and JSON[26]. This allows clients not supporting gRPC to be still be able to consume the same service from the same Protocol Buffers definitions. gRPC can also be consumed in the last mile of computing by browsers thanks to gRPC Web[30].

6.2 Relationship to other standards

The CS3APIs only define the control plane for connecting storage, sync and share platforms and application providers. Data transfer protocols are not part of the specification, rather the specification defines a flexible scheme to use arbitrary data transfer protocol like HTTP, FTP or any other transfer protocol. CS3APIs also allow the protocol-switch where appropriate, for example to the existing collaborative protocols such as WOPI[31] and to federated sharing protocols such as OCM[17].

7 Microservices architecture

The CS3APIs specification does not enforce any type of implementation architecture, however, it is recommended to follow a microservices[28, 29] architecture, with one dedicated microservice for every API service.

Microservices are an architectural and organizational approach for software development where software is composed of small independent services that communicate over well-defined APIs. These services are highly maintainable and testable, loosely coupled, independently deployable and organized around business-logic capabilities. They are owned by

small and self-contained teams and enable organizations to evolve its technology stack in a fewer steps, allowing on-boarding new technologies at a fairly low cost.

The different logical domains explained in Section 4 may be implemented in different microservices, allowing to be developed, deployed, operated, and scaled without affecting the functioning of other services. As the APIs are defined in Protocol Buffers and gRPC, the polyglot services do not need to share their code or to be implemented with other services; any communication between individual components happens via these well-defined APIs. Therefore, each service is designed for a set of capabilities and focuses on solving a specific problem (the storage provider service manipulates storage, the share provider manipulates shares, etc ...). If developers contribute more code to a service over time and the service becomes complex, it may be broken into smaller services.

The architectural choice of microservices allows the deployment of these services in cloud native platforms such as Kubernetes with minimal friction, bringing the benefits of dynamic scaling, continuous deployment and advanced distributed setups across multiple geographical areas.

8 Conclusions

CS3APIs have been chosen as interoperability API in the CS3MESH4EOSC [32], a European project that aims at creating a federation of data and higher-level services across the private clouds of research centers, science laboratories, and National Research & Education Networks.

Further development of CS3APIs will be done within the CS3MESH4EOSC project and in collaboration with the larger CS3 community, driven by the demands from education and research communities with the goal of maximizing the portability of the applications and service extensions.

9 Licensing

To promote free and unrestricted adoption of CS3APIs [10] and the reference implementation Reva [16]) by all developers and solution providers, both community and commercial, Open Source and Open Core, CERN released the source code repositories under the Apache 2.0 license.

References

[1] ownCloud, https://owncloud.com (access time: 13/03/2020)
[2] CS3 community debriefing, (access time: 30/06/2020)
[3] ownCloud, https://owncloud.com (access time: 13/03/2020)
[4] PowerFolder, https://powerfolder.com (access time: 30/06/2020)
[5] Nextcloud, https://owncloud.com (access time: 30/06/2020)
[6] Seafile, https://seafile.com (access time: 30/06/2020)
[7] Pydio, https://pydio.com (access time: 30/06/2020)
[8] Consul, https://consul.io (access time: 30/06/2020)
[9] etcd, https://etcd.io (access time: 30/06/2020)
[10] CS3APIS, https://cs3org.github.io/cs3apis/ (access time: 11/02/2020)
[11] GRPC, https://grpc.io/ (access time: 11/02/2020)
[12] https://unhosted.org/ (access time: 11/03/2020)

[13] Protocol Buffers, https://developers.google.com/protocol-buffers (access time: 11/02/2020)

[14] CS3 Community, https://cs3community.org (access time: 11/02/2020)

[15] CS3 Rome Conference, https://indico.cern.ch/event/726040/timetable/ (access time: 11/02/2020)

[16] Reva: Inter-operability Platform, https://reva.link/ (access time: 11/02/2020)

[17] Open Cloud Mesh https://wiki.geant.org/display/OCM/Open+Cloud+Mesh (access time: 11/02/2020)

[18] Hugo Gonzalez Labrador, Arno Formella, Jakub T. Moscicki, Zenodo, *ClawIO: Cloud Synchronisation Benchmarking Framework* (2016)

[19] Luca Mascetti, Hugo G. Labrador, Massimo Lamanna, Jackub Mościcki and Andreas Joachim Peters, Journal of Physics: Conference Series **664**, 062037, *CERNBox + EOS: end-user storage for science* (2015)

[20] H. G. Labrador, *CERNBox: Petabyte-Scale Cloud Synchronisation and Sharing Platform* (University of Vigo, Ourense, 2015) EI15/16-02

[21] Alvise Dorigo et al., *XROOTD-A Highly scalable architecture for data access*, WSEAS Transactions on Computers 1.4.3 (2005)

[22] Roy Fielding et al., *Hypertext transfer protocol–HTTP/1.1."* (1999)

[23] H. G. Labrador et al., EPJ Web of Conferences **214**, 04038, *CERNBox: the CERN storage hub* (2019)

[24] Andrei Vlad Sambra et al., Technical report, MIT CSAIL & Qatar Computing Research Institute, *Solid: a platform for decentralized social applications based on linked data* (2016)

[25] M. Belshe, R. Peon, R & M. Thomson *Hypertext transfer protocol version 2* (2015)

[26] D. Crockford, RFC 4627, *The application/json media type for javascript object notation (json)* (2006)

[27] Tim Berners-Lee, Roy Fielding, and Larry Masinter. *Uniform Resource Identifier (URI): Generic Syntax (RFC 3986)*, Network Working Group (2005).

[28] Johannes Thönes, IEEE software 32.1 *Microservices*, (2015)

[29] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, D. Montesi, R. Mustafin, & L. Safina, Springer, Cham, *Microservices: yesterday, today, and tomorrow. In Present and ulterior software engineering* (2017)

[30] GRPC-Web, https://github.com/grpc/grpc-web (access time: 11/02/2020)

[31] WOPI, https://wopi.readthedocs.io/en/latest/ (access time: 11/02/2020)

[32] Science Mesh : Interactive and agile/resposnsive sharing mesh of storage, data, & applications for the European Open Science Cloud (CS3MESH4EOSC), https://cordis.europa.eu/project/id/863353