Integrating FTS in the Fenix HPC Infrastructure

Shiting Long[1,*], *Dirk* Pleiter[2], *Mihai* Patrascoiu[3], *Cristiano* Padrin[4], *Michele* Carpene[4], *Sergi* More[5], and *Miguel* Carpio[5]

[1]Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany
[2]KTH Royal Institute of Technology, 10044 Stockholm, Sweden
[3]CERN, 1211 Geneva 23, Switzerland
[4]CINECA, 40033 Casalecchio di Reno, Italy
[5]Barcelona Supercomputing Center, 08034 Barcelona, Spain

Abstract. As compute requirements in experimental high-energy physics are expected to significantly increase, there is a need for leveraging high-performance computing (HPC) resources. However, HPC systems are currently organised and operated in a way that this is not easily possible. Here we will focus on a specific e-infrastructure that incorporates HPC resources, namely Fenix, which is based on a consortium of 6 leading European supercomputing centres. Fenix was initiated through the Human Brain Project (HBP) but also provides resources to other research communities in Europe. The Fenix sites are integrated into a common AAI and provide a so-called Archival Data Repository that can be accessed through a Swift API.

In this paper, we report on our efforts to realise a data transfer service that allow to exchange data with the Fenix e-infrastructure. This has been enabled by implementing support of Swift in FTS3 and related software components. We will, in particular, discuss how FTS3 has been integrated into the Fenix AAI, which largely follows the architectural principles of the European Open Science Cloud (EOSC). Furthermore, we show how end-users can use this service through a WebFTS service that has been integrated into the science gateway of the HBP, which is also known as the HBP Collaboratory. Finally, we discuss how transfer commands can be automatically distributed over several FTS3 instances to optimise transfer between different Fenix sites.

1 Introduction

Over the past decades, the performance of HPC systems has increased exponentially. This remarkable progress has enabled scientific research in multiple disciplines that were and are in need of immense computing power. During this period, it was typically sufficient to run workflows within a data centre. As the range of HPC applications continues to expand, it is unrealistic for HPC centres to fulfil the entirety of computational requirements, therefore the workflows extend beyond one single data centre. This is in part driven by the need for enabling international collaborative research. The European brain research community organised in the Human Brain Project (HBP) is one example. Another strong driver is the growing trend towards a data-driven paradigm in scientific discovery, where overwhelming

*e-mail: s.long@fz-juelich.de

sets of data are to be generated, curated, analysed and visualised [1]. This applies to various science instruments that are in need of HPC for processing the data. This approach has been pioneered by the radio astronomy community (see, e.g., [2]). In the upcoming High Luminosity phase of the Large Hadron Collider (HL-LHC), the compute requirements of experiments are expected to significantly increase such that HPC resources will be needed (see, e.g., estimates from the Atlas experiment [3]). Serving distributed research communities with diverse and data-intensive applications leads to the need for geographically dispersed resources that are federated.

Fenix [4, 5] combines the services of 6 leading European supercomputing centres and offers unified access for research communities. It is a federated e-infrastructure that provides both computing and data services based on resources available at the involved European supercomputing centres. One step towards enabling the processing of experimental data within Fenix is the realisation of a data transfer service that allows moving experimental data to the relevant supercomputing centres and vice versa. More specifically, we consider the case where the experimental data is stored in the ESCAPE Data Lake [6]. For this purpose, we enhanced the File Transfer Service (FTS), specifically its current iteration FTS3 [7] to support Fenix storage services.

We start by providing some background in section 2 before explaining the details of our implementation in section 3. In section 4 we report on a proof-of-concept setup. Next, we discuss possible future deployments of FTS3 services within Fenix in section 5, before presenting a summary in section 6.

2 Background

Fenix is a federated computing and data infrastructure based on a memorandum of understanding among BSC (Spain), CEA (France), CINECA (Italy), CSC (Finland), CSCS (Switzerland), and JSC (Germany). Its creation is fueled by an increasing need for extreme-scale computing power and storage requirements. The design process is catalysed by international collaborations in science communities. Specifically, Fenix was initiated by the Human Brain Project (HBP), a pan-European project aiming to 1) gain an in-depth understanding of the complex structure and function of the human brain, and 2) develop a research platform for brain science that is called EBRAINS.[1]

A key element of federation is a common Authentication and Authorisation Infrastructure (AAI). The Fenix AAI follows the Authentication and Authorisation for Research and Collaboration (AARC) [8] blueprint architecture that is also the basis for EOSC. It is based on an Identity Provider (IdP) Proxy that can propagate authentication requests to the supported IdPs, validate authenticity against these IdPs, and eventually authorise access to Fenix services. This essentially means that users can authenticate to Fenix AAI by authenticating to any of its registered site IdPs. Each of the supercomputing centres provides an IdP service and assigns virtual identities with the highest level of assurance. The EBRAINS IdP is also supported as a community AAI, but the level of assurance is treated as low.

With the paradigm of cloud computing becoming more popular, numerous research endeavours have arisen to investigate the performance of HPC applications running on clouds and have obtained promising results [9]. It is inevitable that HPC centres adopt the idea of cloud abstraction and offer convenient on-demand services. However, coupling HPC and cloud workloads under the same management for HPC centres may result in security challenges, particularly when granting unrestricted access to external networks. Thus, Fenix divides computing and data services into two separate environments: HPC and Cloud.

[1] https://ebrains.eu/

In practice, Fenix abstracts its services on each site as follows:

1. **Scalable Computing Services (SCC)**, which are massively parallel HPC systems for scalable and/or compute-heavy applications;
2. **Interactive Computing Services (IAC)**, which provide quick access to single computer servers to analyse and visualise data interactively;
3. **Virtual Machine (VM) Services**, which deploy virtual machines via the OpenStack platform;
4. **Active Data Repositories (ACD)**, which are site-local data repositories close to the HPC resources, and are typically POSIX parallel file systems based on Lustre or GPFS;
5. **Archival Data Repositories (ARD)**, which are cloud object stores, for which currently an OpenStack Swift interface is prescribed.

The SCC, IAC and ACD belong to the HPC environment, which is connected to a local site network and requires SSH or similar secured access. The VM services belong to the cloud environment, which is connected to the external network. The ARD can be accessed from both HPC and cloud environments, making it ideal as a bridge for intra- and inter-site communications.

Fenix addresses the challenge of sharing access to large-scale HPC systems by 1) separating data storage into two layers (ACD and ARD), and 2) establishing ACD-to-ARD connections within a site and ARD-to-ARD connections between different sites. A generic Fenix use case is shown in Figure 1. The arrows in the data workflow depict data transferability, whereas the arrows in the processing workflow indicate job execution and results retrieval. The lines bridging the two workflows indicate the high I/O bandwidth connection between ACD and HPC systems. Essentially, Fenix users may make use of available HPC resources located at different sites in one scientific workflow with the help of the Fenix data services.

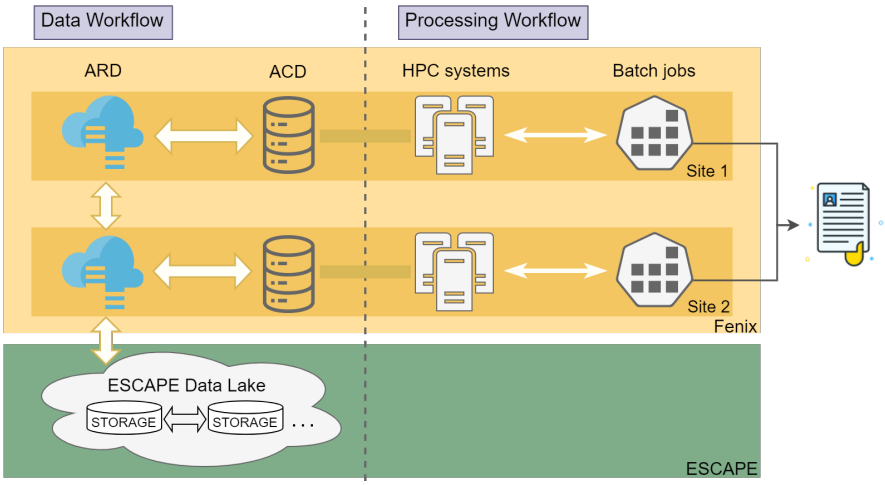

Figure 1: Fenix example use case

The Fenix data services are realised using two tools. One is the Fenix *Data Mover*, which is based on a software named Nodeum developed by MT-C.[2] It is designed with the goal of

[2]https://www.mt-c.com/

offering programmable, high-speed, scalable and secure data movement between ACD and ARD. The other one is the Fenix *Data Transfer Service*, which orchestrates data transfers between ARDs. We chose FTS, developed by CERN and well-recognised in the high-energy physics communities, as the orchestrator. FTS has proven robust over the years and is compatible with Fenix after some enhancements. Another benefit is that FTS is adopted worldwide in various data centres, presenting collaboration opportunities with Fenix. We show such potential in Figure 1 as extended data transferability to the ESCAPE Data Lake [6].

3 FTS enhancements

FTS is designed for distributing data across the WLCG infrastructure and it is a critical element for data management [7]. FTS comprises multiple components including 1) the FTS server that schedules, optimises, and executes transfer jobs, and 2) the FTS REST server that offers a RESTful interface for the submission and retrieval of transfer jobs. We refer to FTS as the entire collection of its components if not specifying it as a server. FTS supports multiple protocols thanks to the GFAL2[3] library. The FTS team has also released a Web interface named WebFTS[4] (currently maintained by EGI) for usage via browsers instead of a CLI.

FTS was developed for traditional grid infrastructures when X.509 certificate and private key authentication were mainstream, whereas the trend of cloud computing has brought HTTP token-based authentication to the stage centre. Therefore, FTS recently adopted the support for OpenID Connect (OIDC) protocol [10], presenting a chance for Fenix AAI to be integrated. Since cloud resources become increasingly attractive to computing facilities, cloud object stores have become an interesting option for data management within WLCG [11]. Efforts have been made for the integration of object stores within high-energy physics workflows. This functionality is largely enabled by the HTTP library DaviX.[5] It is used as a plugin for GFAL2 and hence can be exploited by FTS. However, DaviX supported only Amazon S3 and Microsoft Azure. Supporting access to Fenix Archival Data Repositories required adding Swift API support in DaviX.

3.1 Authentication and authorisation

The authentication and authorisation process to transfer data with FTS in Fenix consists of two parts: one with FTS and one with ARD, i.e., Swift object store. FTS leaves the user authentication to other platforms, namely, it does not redirect the user to log in and eventually grants access. Instead, FTS takes an OIDC access token, introspects it and accepts the transfer job associated with it. When executing the job, FTS uses the access token to exchange for a different access token which implies FTS is accessing the storage elements on behalf of the actual user. Steps 2-4 in Figure 2 show this part of the authentication flow.

We register Fenix AAI with our FTS deployment such that the service can be accessed by Fenix users and retrieve access tokens from the Fenix AAI (see Step 1 in Figure 2). For strict CLI access, Device Code Flow [12] is the recommended authorisation grant to obtain OIDC tokens. A common tool *oidc-agent* [13] (version $\geq$ 4.4.1) can be used in this case. BSC has also implemented a simple website that displays an access token granted to the user after they log in through a browser. With WebFTS supporting OIDC and Authorization Code Flow [14], the complexities with tokens are hidden from the user and a standard log-in-to-use model is applicable. WebFTS redirects the user to log in via Fenix AAI, receives an OIDC access token from Fenix AAI and passes the token to FTS upon job submission.

[3] https://github.com/cern-fts/gfal2
[4] https://github.com/EGI-Federation/webfts
[5] https://github.com/cern-fts/davix

In a typical OpenStack deployment, Keystone is used as a site-local identity service. It acts as a token server and issues a token upon successful user authentication. Based on that token, the Swift object store may authorise access. This token is a Fernet token, which in the following we call *OS token*.

There is a gap between an OIDC access token and an OS token. To enhance user experience, we do not require users to submit both OIDC tokens for authentication to FTS and OS tokens for authentication to Swift. Therefore, we made use of the newly introduced OIDC module for Keystone and implemented a method in FTS REST that exchanges an OIDC access token for an OS token automatically, shown in Steps 5-6 in Figure 2. Lastly, FTS can use the OS token for file management in Swift.

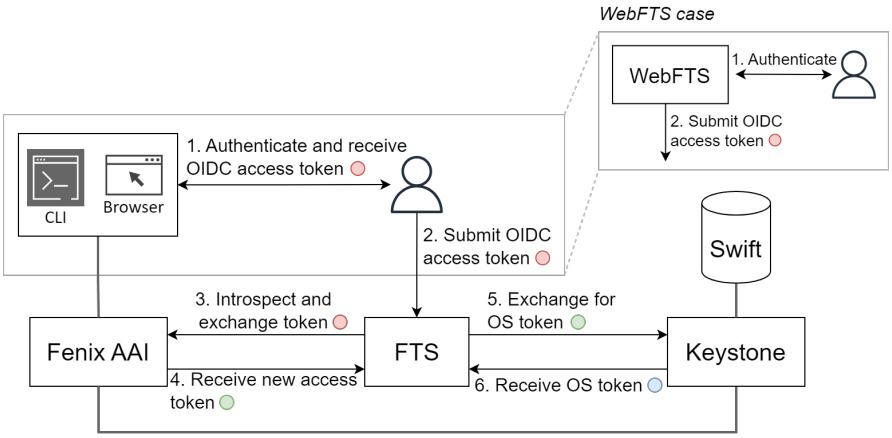

Figure 2: Token workflow of obtaining access to Swift object store with FTS in Fenix

3.2 File management with DaviX

We implemented file management methods with Swift API in DaviX. Basic functions such as uploading, downloading and deleting files (objects in cloud terminologies) as well as advanced requests with checksum, stat and partial read are supported. We also enabled the Swift Static Large Object (SLO)[6] support for DaviX, namely, large files are uploaded in segments to the Swift object store automatically to provide efficiency.

Similar to the cases with other cloud storage, specific parameters are to be given for Swift when executing commands with DaviX. One requires an OS token for authentication as specified in Section 3.1 and an OS project ID which indicates the storage space where the file is. An example command to upload a file to a Swift object store would be:

```
davix-put --ostoken xxx --osprojectid yyy swifts://<path_to_file>
```

3.3 Data transfer with FTS/WebFTS

The Swift-specific parameters are likewise handled in GFAL2 and FTS. The `ostoken` (optional in FTS because of Steps 5-6 in Figure 2) and `osprojectid` can be given in the commands. Swift can be treated as a "protocol" for file management with the prefix `swift`. These configurations can be inherited during a transfer job, starting from FTS and flowing through GFAL2 until reaching DaviX, where the actual data is read and written.

[6]https://docs.openstack.org/swift/latest/overview_large_objects.html

WebFTS exploits the FTS REST API and can be seen as a front end of FTS. To enable Swift support in WebFTS as well, we added Swift-specific parameters management in WebFTS and implemented additional APIs in FTS REST for Swift. The current capabilities of WebFTS encompass a range of fundamental tasks including submitting, listing, and cancelling transfer jobs. Additionally, a hierarchical presentation of files within the Swift object store is available, facilitating the (multi-)selection of files during the job submission process.

4 Proof-of-concept setup

Fenix aims to implement a ground-level infrastructure for computing and storage by aggregating the facilities and systems of the partner sites. It is a model that can be extended with external platforms, providing a more powerful framework. Our attention is directed towards the ESCAPE project, with a specific focus on enabling data transfer between the ESCAPE Data Lake [6] and a Fenix ARD.

A proof-of-concept is accordingly proposed to bridge ESCAPE and Fenix. The use case is defined as an ESCAPE user transferring data from an ESCAPE WebDAV server to a Fenix ARD, and vice versa. With the help of Swift support and pre-existing WebDAV support in FTS, we have successfully tested this functionality and we are federating the AAI services to provide ESCAPE-Fenix interoperability.

ESCAPE AAI (a community AAI) and Fenix AAI (an infrastructure AAI) are both involved in this use case. However, ESCAPE AAI is not fully integrated into Fenix AAI, i.e., both ESCAPE and Fenix resources are protected by their own AAIs. We drafted the proof-of-concept minimising the alterations of the current ESCAPE AAI settings while enforcing Fenix AAI to complete a federation between the two AAIs. In this case, users do not need to provide credentials for both ESCAPE and Fenix when initiating data transfers.

This requires a more complex authentication flow than discussed in Section 3.1 as shown in Figure 3. We use the term *token* when referring to *OIDC access tokens*. Furthermore, we added the name of the AAI that issued the token. The key changes are the following:

- The ESCAPE AAI registers a client for Fenix AAI, which can introspect ESCAPE tokens and perform token exchange requests.
- The Fenix AAI supports token exchange with ESCAPE AAI on the user's behalf, i.e., a token exchange request can be performed with Fenix AAI using an ESCAPE token.

Figure 3 shows the ESCAPE-Fenix token workflow, in which the renewed ESCAPE token can be used to access the WebDAV server in ESCAPE and the OS token can be used to access the Swift object store in Fenix. We omit the process of the user obtaining an ESCAPE token, as it can be done similarly as in the case of Fenix.

5 Future deployment strategy

The current deployment of the Data Transfer Service in Fenix involves an FTS, an FTS REST and a WebFTS server at BSC, all of which have adopted the enhancements discussed in Section 3. This service supports Swift endpoints at BSC, CEA, CSCS and JSC. The ARD at CINECA is expected to have an S3 interface, which is also supported by FTS. CSC joined Fenix in 2021, later to the initial five partner sites. Its storage services are not integrated yet and hence they are not within the scope of this paper.

Since Swift like other cloud object stores does not support third-party copy, deploying only an FTS server at a single site would result in all data being transferred via this site instead of taking a possibly more direct route from source to destination. Therefore, we

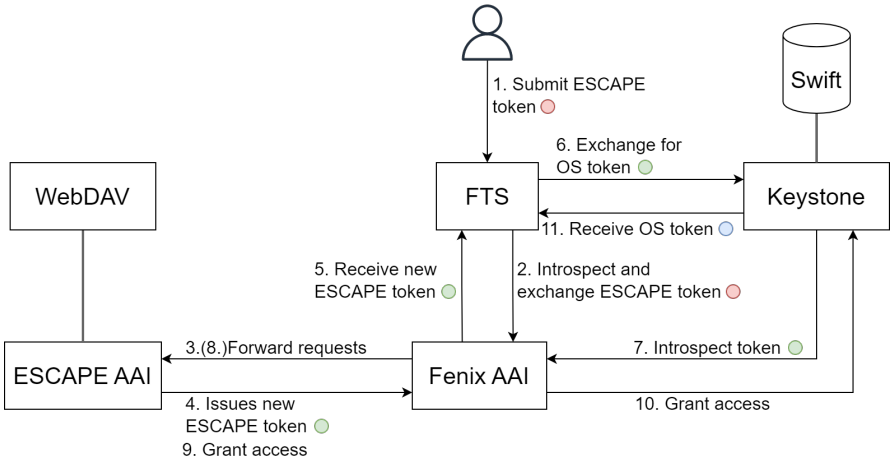

Figure 3: Token workflow of obtaining access to both ESCAPE and Fenix

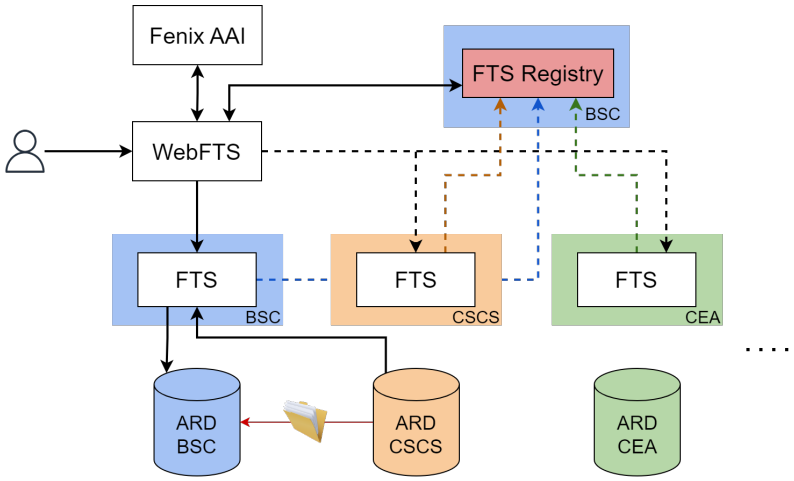

Figure 4: Multi-site FTS deployment strategy

propose a setup that foresees FTS deployments at multiple sites as illustrated in Figure 4. The information on the available FTS instances would have to be kept in an FTS registry, which still needs to be realised. An enhanced version of WebFTS can fetch this information and decide which FTS to use for a transfer job. This selection would primarily be based on the source and destination of the transfer job such that inter-site network communication is minimised. For instance, a transfer from CSCS to BSC in Figure 4 could either use the FTS service at BSC or CSCS. The difference between the two FTS is trivial unless one of them has a high load, in such case the vacant one should be selected. In the extreme cases where both FTS are overloaded, FTS at another site may be chosen.

6 Summary and conclusions

The Data Transfer Service adopting the FTS stack has reached production in the Fenix research infrastructure. It has successfully enabled data transfer between Swift object stores

located at various partner sites. It has ensured connectivity to multiple HPC systems together with the Data Mover service.

In this paper, we showcase the enhancements made to the FTS stack to facilitate seamless integration with Fenix. Additionally, we provide a comprehensive overview of the potential extension of Fenix data services through a proof-of-concept. The future work includes the proof-of-concept, realising the multi-site FTS deployment strategy and evaluating the overall data transfer performance. We envision our services unlocking their full potential for diverse scientific communities, particularly those reliant on HPC capabilities.

Acknowledgements

Funding for the work is received from the European Commission H2020 program under Specific Grant Agreement No. 800858 (ICEI) and No. 945539 (HBP SGA3).

References

[1] A.J. Hey, S. Tansley, K.M. Tolle et al., *The fourth paradigm: data-intensive scientific discovery*, Vol. 1 (Microsoft research Redmond, WA, 2009)

[2] K. Van Der Schaaf, C. Broekema, G. Van Diepen, E. Van Meijeren, Experimental Astronomy **17**, 43 (2004)

[3] A. Collaboration, Tech. rep., CERN, Geneva (2022), `https://cds.cern.ch/record/2802918`

[4] S.R. Alam, J. Bartolome, M. Carpene, K. Happonen, J.C. Lafoucriere, D. Pleiter, Commun. ACM **65**, 46–47 (2022)

[5] S. Alam, J. Bartolome, S. Bassini, M. Carpene, M. Cestari, F. Combeau, S. Girona, S. Gorini, G. Fiameni, B. Hagemeier et al., *Fenix: Distributed e-Infrastructure Services for EBRAINS*, in *Brain-Inspired Computing*, edited by K. Amunts, L. Grandinetti, T. Lippert, N. Petkov (Springer International Publishing, Cham, 2021), pp. 81–89, ISBN 978-3-030-82427-3

[6] R. Dona, R. Di Maria, EPJ Web Conf. **251**, 02060 (2021)

[7] A. Ayllon, M. Salichos, M. Simon, O. Keeble, *FTS3: new data movement service for WLCG*, in *Journal of Physics: Conference Series* (IOP Publishing, 2014), Vol. 513, p. 032081

[8] A. Biancini, L. Florio, M. Haase, M. Hardt, M. Jankowski, J. Jensen, C. Kanellopoulos, N. Liampotis, S. Licehammer, S. Memon et al., arXiv preprint arXiv:1611.07832 (2016)

[9] M.A. Netto, R.N. Calheiros, E.R. Rodrigues, R.L. Cunha, R. Buyya, ACM Computing Surveys (CSUR) **51**, 1 (2018)

[10] E. Karavakis, A. Manzi, M.A. Rios, O. Keeble, C.G. Cabot, M. Simon, M. Patrascoiu, A. Angelogiannopoulos, *FTS improvements for LHC Run-3 and beyond*, in *EPJ Web of Conferences* (EDP Sciences, 2020), Vol. 245, p. 04016

[11] A.A. Ayllon, M.A. Rios, G. Bitzes, F. Furano, O. Keeble, A. Manzi, *Making the most of cloud storage-a toolkit for exploitation by WLCG experiments*, in *Journal of Physics: Conference Series* (IOP Publishing, 2017), Vol. 898, p. 062027

[12] W. Denniss, J. Bradley, M.B. Jones, H. Tschofenig, *OAuth 2.0 Device Authorization Grant*, RFC 8628 (2019), `https://www.rfc-editor.org/info/rfc8628`

[13] G. Zachmann, SKILL 2018-Studierendenkonferenz Informatik (2018)

[14] D. Hardt, *The OAuth 2.0 Authorization Framework*, RFC 6749 (2012), `https://www.rfc-editor.org/info/rfc6749`