FirecREST: a RESTful API to HPC systems

Felipe A. Cruz*, Alejandro J. Dabin†, Juan Pablo Dorsch*, Eirini Koutsaniti*, Nelson F. Lezcano†,
Maxime Martinasso* and Dario Petrusic*
*Swiss National Supercomputing Centre, ETH Zurich, Lugano, Switzerland
†Centro Internacional de Metodos Computacionales en Ingeniería (CIMEC), CONICET-UNL, Santa Fe, Argentina

***Abstract*—As science gateways are becoming an increasingly popular digital interface for scientific communities, it is important for High-Performance Computing centers to provide a modern and externally accessible interface such as Web-enabled APIs. Such an interface provides access to HPC center resources to allow scientific web portals to submit a job or move data in and out the HPC center. This work presents the FirecREST API, a RESTful Web API infrastructure that allows scientific communities to access the various integrated resources and services available on HPC systems. The capabilities of FirecREST have been defined based on the use case requirements described in this work.**

I. Introduction

Web Application Programming Interfaces [1] (Web APIs) provide third-party developers with frameworks for building HTTP-based services that can be accessed by software applications over a variety of platforms. In this way, developers can envision and implement new business processes, build new client workflows that simplify the user experience, or enable them to develop completely new platforms and services.

The core of the current Web API development gravitates towards Representational State Transfer (REST) [2] [3]. RESTful API is a software design pattern that specifies a uniform and predefined collection of stateless operations. The REST software architecture is well suited for enabling services that work over the Web as this design pattern introduces several desirable properties for web services, such as performance, scalability, and flexibility. In essence, RESTful Web APIs provide interoperability between systems and applications. It has become a building block of web software development. It simplifies the software development of web-enabled applications and portals and it improves integration across multiple services and organizations.

Over the past years at the Swiss National Supercomputing Centre (CSCS), we have developed FirecREST, a RESTful Web API infrastructure. FirecREST is a generic interface that can be used on any HPC system. It interfaces primarily to the batch scheduler and HPC storage technology. CSCS is using FirecREST to interface with its flagship system Piz Daint, a Cray XC system.

FirecREST web services allow science gateway [4] developers to integrate their platforms with HPC resources. In practice, the FirecREST API gives access to two core HPC services: submitting and monitoring jobs, and moving data across multiple filesystems. Moreover, it does so while enforcing integration with the authorization and authentication infrastructure (AAI) of an HPC center.

In this work, we present the architecture and capabilities of the FirecREST API. Prior to describing the API, we introduce use cases that have driven its requirements. The key contributions of our work are:

- To present concrete use cases that require the capability for an HPC center to provide resource access through a Web interface;
- To describe the architecture and capability of the FirecREST API.

II. Use cases

FirecREST improves the accessibility of HPC resources to scientific communities by enabling them to build platforms that target specific scientific goals. For an HPC center, providing a Web API to HPC resources expands the center user base as new scientific platforms develop. Platform developers benefit from a standard interface, whereas the HPC center provides a single technology to satisfy multiple user communities. In this section, we present three use cases for which FirecREST provides key programmable functionalities. Even if those use cases are targeting CSCS services, their requirements remain sufficiently generic to be valuable for any HPC center.

A. Materials Cloud

The Materials Cloud [5] is a web platform to share resources in computational materials science. One feature of the Materials Cloud is to run computational intensive jobs of well-known HPC-aware scientific applications for discovering new properties of materials.

The Materials Cloud has been released and is online. It it is currently running on a cloud system at CSCS. Computational intensive jobs are submitted to Piz Daint. To enable reproducible workflows and job submissions, the Materials Cloud uses AiiDA [6]. On the technical side, AiiDA interacts with Piz Daint by executing SSH commands. AiiDA will greatly benefit from a RESTful API. It will ensure a better security, a lower effort of maintenance (as it is CSCS responsibility to provide a working API service), and a simplification of its internal mechanisms to access Piz Daint. It will also allow AiiDA to seamlessly interface to other HPC centers using FirecREST.

B. Interactive computing

The second use case drives FirecREST requirements for an internal CSCS service. CSCS offers an interactive service based on Jupyter notebooks [7]. Jupyter notebooks have

978-1-6654-2272-7/20/$31.00 ©2020 IEEE
DOI 10.1109/SuperCompCloud51944.2020.00009

become a preferred interface for scientists and many HPC centers provide a similar interactive service. Moreover, science gateways are using customized Jupyter interfaces as a user-facing interface. For instance, the previous use case is running its web interface through a JupyterHub service.

The current CSCS interactive service uses a standard Jupyter notebook spawner connected to the batch scheduler used to submit jobs on Piz Daint. In the future, this spawner will be modified to use a RESTful API. One of the main advantages of using a standard API technology is to target multiple infrastructures for executing notebooks keeping a unique interface and code base. Each infrastructure has a cost model associated to and it becomes possible for the end users to target either an HPC system for dedicated resources and a higher cost or a cloud system for virtualized resources and a lower cost.

C. Super facility

The last use case describes the coupling of two remote facilities connected over Internet. The Paul Scherrer Institute (PSI) provides experimental devices to explore and visualize the structure of tiny objects. Such devices include a synchrotron. PSI and CSCS are developing a workflow to stream the output data of the synchrotron to CSCS for being processed on Piz Daint. Within this workflow, PSI users can interactively visualize the inside of the objects with a very high resolution thanks to the scalable computational capability of Piz Daint. For such a use case, a web interface is primordial to connect both facilities. FirecREST is used by the workflow to submit jobs and move data. Moreover, to guarantee the computation resources during the experiment, a reservation service of compute nodes has been developed and interfaced with FirecREST.

D. Summary and requirements

From the above use cases, we can identify three major requirements:

- The necessity of the API to integrate with various identity providers external to the center. Apart from internal services such as interactive computing, users do not necessarily have accounts at the targeted HPC center. This feature depends on the global Identity Access Management policy of the center, and the use of standard authentication protocols;
- The ability to execute workloads on HPC systems;
- The capability to do external transfer of data to/from the centre filesystems attached to the HPC system.

We expect that FirecREST will enable new use cases and further increase the reach of HPC to scientific communities by: enabling the development of more modern and comfortable web interfaces to HPC, and by providing an interface to the centre resources that are common, stable, secure, and maintainable, thus avoiding scientific community platforms to implement their custom interfaces and integration with infrastructure.

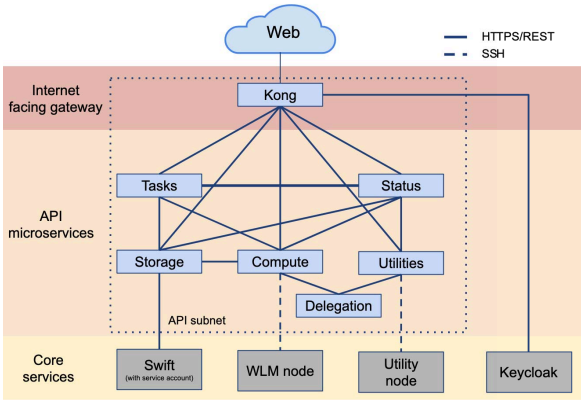


Fig. 1. FirecREST microservice architecture.

III. FirecREST microservice architecture

FirecREST provides to developers a web-enabled API to an HPC system that is stable, secure, and maintainable. Internally, FirecREST translates every HTTP request into its appropriate operations on the supercomputer, such as: enforcing authentication and authorization, job management, data mover, and other operations. The operations to perform are loosely coupled and involve different resources. Thus, to improve maintainability, test-ability, and to match CSCS organization, we followed a microservice architecture built using open source tools such as Keycloak [8], Kong [9], Flask [10], Paramiko [11], OpenSSH [12], OpenAPI [13], and Redis [14]. Figure 1 describes the FirecREST microservice architecture diagram. In the following subsections, we present the core components and microservices that are part of FirecREST. Identity Access Management, API gateway, compute, storage, utilities, asynchronous task execution, delegation, and status.

A. Identity Access Management

The Identity and Access Management (IAM) infrastructure ensures that users and web applications have the appropriate permissions to access resources at the center by using a secure protocol. For the whole of the IAM infrastructure at CSCS, we will only discuss Keycloak, the Identity and Access Management solution deployed at the center. Keycloak allows to secure applications and services by providing a mechanism for the authentication and authorization of CSCS users, CSCS services, and third party applications. Among the many features of Keycloak, we highlight the following:

- Single sign-on solution
- Integration with Kerberos authentication service
- Fine-grained authorization control for services
- Client registration and authorization
- Support of OpenID Connect (OIDC) protocol [15]

The integration of FirecREST with Keycloak has been achieved through the use of the OIDC protocol. OIDC is an authentication protocol that extends the OAuth 2.0 specification. OAuth 2.0 is an industry-standard protocol for token-based authorization that is commonly used as a mechanism for

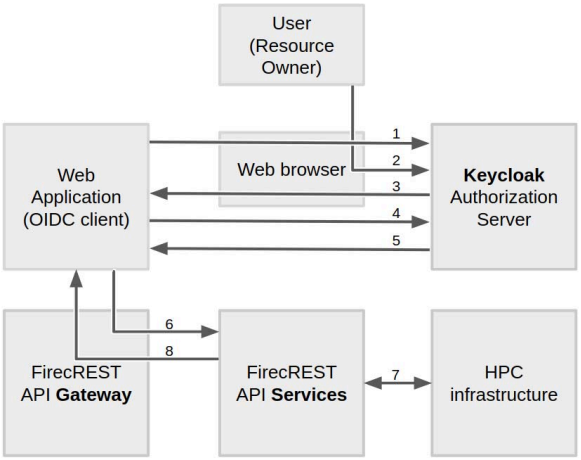


Fig. 2. FirecREST authentication and authorization workflow: 1) Client performs a user authentication and access permission request; 2) User authenticates himself and authorizes client application; 3) Keycloak responds to the application with an authorization code; 4) Application uses the authorization code to request for an Access Token; 5) Keycloak grants the Access Token to the application over a secure backchannel; 6) Application request access to user-owned resources via the FirecREST API Gateway, enforcing the Access token permissions and redirecting to the correct service API endpoint; 7) FirecREST services translate the web request into actions on the HPC infrastructure; 8) FirecREST responds to the web application request.

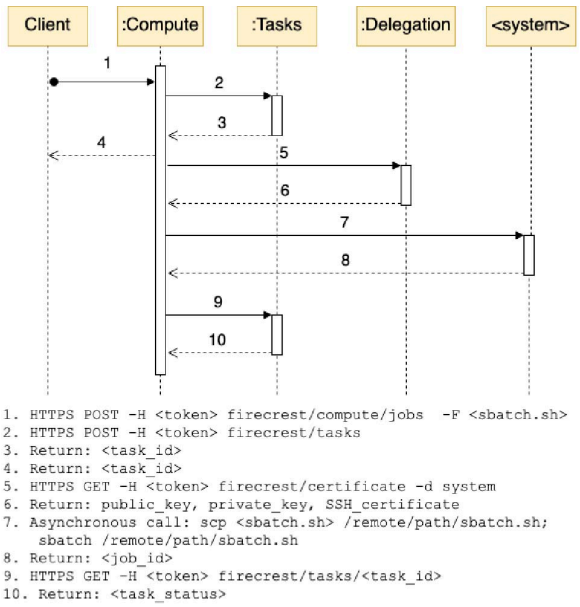


Fig. 3. Sequence diagram of the job submission workflow. Please note that for conciseness we have skipped the API gateway step from the description, however, all interactions with the FirecREST API are passed and validated first by the gateway.

users to grant access permission to a web application to access user-owned resources and services. The extensions provided by OIDC to the OAuth 2.0 protocol add a user authentication layer, providing a mechanism that enables single sign-on to users.

FirecREST leverages on Keycloak and OIDC for authentication and authorization of web applications, enabling the following capabilities:

- Enforce that all requests are authenticated
- Applications never manipulate user credentials
- Only allow requests from registered applications
- User-managed access permissions per application
- Stateless security model by use of tokens
- Short lifespan for sensitive access tokens
- Extend client sessions by the mean of refresh tokens

In a nutshell, FirecREST OIDC-based IAM enables the user to login to a registered web application using their CSCS credentials and grant a web application with access to user-owned resources at the center. Moreover, it does so without the user ever sharing their credentials with the web application. Figure 2 presents a complete description of the OIDC Authorization Code Flow used by FirecREST.

B. API Gateway

The API gateway provides an interface to publish, maintain, monitor, and secure all FirecREST API endpoints. As shown in Figure 1, the gateway is hosted on a machine that is facing the internet. All interactions with the FirecREST API are first passed and validated before being redirected to any other microservice.

In this way, every request made to the FirecREST API arrives first at the gateway, which will proxy the request towards the requested microservice endpoint. However, before the request is passed on to the microservice, the gateway will enforce that the requests are correctly authenticated and authorized by requiring and validating the Access Token that must accompany each API request.

The current implementation of the gateway service is based on the Kong API gateway. Kong is a widely used open source microservice API gateway that implements functionalities such as a variety of authentication and authorization mechanisms, support for OIDC, IP filtering, access control lists, analytics, and rate limiting that have allowed us to configure the gateway to our requirements.

C. Compute

The compute microservice implements the interface to the workload manager, thus allowing applications to submit, stop, and query the status of jobs by using non-blocking asynchronous API calls. This service depends on: the ”tasks” microservice (see section III-G) that provides a temporary resource that tracks the state of each call; and the delegation microservice (see section III-E) that issues a restricted SSH certificate that allows the execution of operations on behalf of the user. We now describe the integration with the SLURM [16].

For conciseness, in this section we will only describe the *job submission* workflow, as other operations follow similar or simpler workflows. Let us consider now the *job submission* workflow shown in the sequence diagram in Figure 3. As

it can be observed, the job submission starts with the client calling the API endpoint `firecrest/compute/jobs` with a `POST` operation, passing the following parameters: the access token which identifies the user and authorizes the call; the system where the job is submitted to; and, a file part that contains the job definition written in SLURM's sbatch format. Upon receiving a request, the compute microservice checks the validity of the parameters passed with the request and then calls the tasks microservice, creating a new task which will track the progress of the operation returning a task resource as an immediate response to the client request. The client will use the task microservice to access the task resource (identified by its `task id`) and use it to track the status of the request in an asynchronous way, meanwhile the compute microservice continuously updates the task resource as the job request progresses.

The compute microservice now requests to the delegation microservice a SSH certificate, passing the access token as a parameter. The delegation microservice will respond by retrieving the username from the access token and generating an SSH certificate that will be signed with the *Certificate Authority key* (see section III-E for details on this). Thus, the delegation microservice responds to the compute microservice request with a valid SSH certificate and the related public and private keys used in the process.

Next, the compute microservice makes use of the Paramiko library to establish a SSH session using the certificates and keys obtained from the delegation microservice. Over this SSH session, a unique temporary folder in the user's `$HOME` directory is created, at this location the job information will be stored. The compute microservice then copies the sbatch script into the newly created temporary directory. Finally, the microservice runs the script using an sbatch command over the SSH session and captures its output, updating the task identified by the initial `task id` accordingly. Information in the task resource, such as `SLURM's job id field` can be used by the client to query for the state of the scheduled job.

D. Data mover

This microservice enables users to upload and download large files to/from CSCS, while also enabling the movement of data within the different filesystems available on Piz Daint. It uses non-blocking calls to high-performance storage services which immediately respond with a reference to a resource that tracks the state of the request (see section III-G). A full description of the upload and download workflow is presented in Figure 4 and Figure 5, respectively.

E. Delegation

The delegation microservice is a FirecREST internal service that is not exposed to the user. This service takes a valid JWT access token as input and creates a short-lived SSH certificate to be used for user authentication.

OpenSSH user certificates are formed by: a public key; user identity information; and a set of constraints that limits

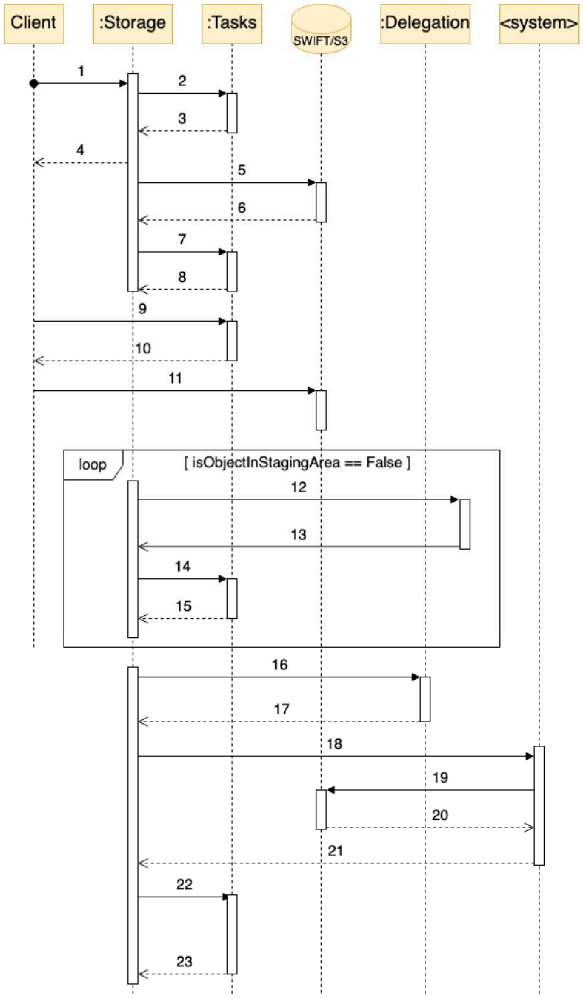
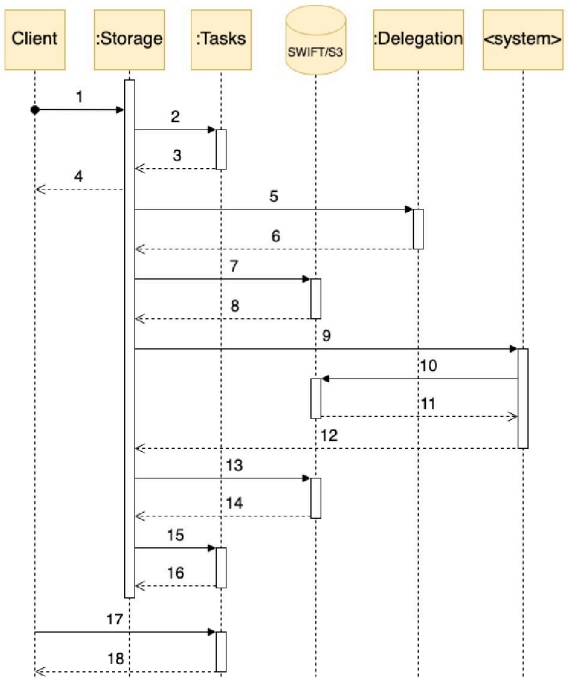


Fig. 4. Sequence diagram of the data upload workflow for an asynchronous upload of a local file into CSCS infrastructure. The client calls the storage microservice giving parameters such as system, filesystem path and local file path. The storage microservice then uses a service account in an object storage solution to generate an HTTP form for temporary file upload. The client uses the temporary upload URL to upload the file into an staging area. FirecREST regularly checks for the object to be uploaded successfully, when this happens, the storage microservice starts downloading object from SWIFT server into CSCS filesystem on behalf of the user.




```
1. HTTPS POST -H <token> firecrest/storage/xfer-external
/download -F <sourcePath>
2. HTTPS POST -H <token> firecrest/tasks
3. Return: <task_id>
4. Return: <task_id>
5. HTTPS GET -H <token> firecrest/certificate -d system
6. Return: public_key, private_key, SSH_certificate
7. SWIFT/S3 API: create temporary upload for <sourcePath>
8. Return: temporary upload URL
9. Asynchronous call: curl <temporary_upload_form> <sourcePath>
10. upload <sourcePath> to staging area
11. Return: upload complete
12. Return: file uploaded
13. SWIFT/S3 API: create temporary download URL for <sourcePath>
14. Return: temporary upload URL
15.  HTTPS PUT -H <token> firecrest/tasks/<task_id>
16. Return: task update status
17. HTTPS GET -H <token> firecrest/tasks/<task_id>
18. Return: <task_status: download URL>
```

Fig. 5. Sequence diagram of the data download workflow. The workflow is initiated by the client requesting a file from one of the HPC filesystem. An SSH user-certificate is created by the delegation microservice. The SSH certificate allows the storage microservice to upload the file into the FirecREST service account in SWIFT that is used as an staging area. Upon completion of the upload into the staging area, the storage microservice uses SWIFT API to create a temporary download URL. The temporary download URL is created by SWIFT with an unique hash containing that expiration time, object name and a secret, thus circumventing the need for the client to authenticate in order to start the remote download from SWIFT. Finally, temporary URL is returned to client.

the certificate validity. SSH certificates are signed using the `ssh-keygen` tool of `OpenSSH` with the `Certificate authority key` of the delegation microservice, also a standard SSH key. To enable SSH servers to accept certificates for user authentication, the sshd server must also be configured to trust the microservice CA public key.

Once a valid certificate is generated by the delegation microservice, other FirecREST microservices can use the certificate to perform remote command execution on behalf of the user. As such, this microservice enables FirecREST to perform delegation by means of secure system access using SSH certificates.

F. Status

This microservice provides information on the availability and state of services and relevant infrastructure that is accessible through FirecREST.

G. Tasks

The task microservice responds to the need of managing the state of requests that are being resolved asynchronously. One clear example for the need of this microservice can be observed in the data transfer operations handled by the storage microservice (see section III-D), as otherwise some of those workflows would not be possible. As such, FirecREST microservices during an asynchronous request can rapidly create and respond with a new task resource. The operational result of the request is then tracked as the originating microservice continuously updates the task as progress is being made. Thus, task resources allow a client to perform other activities while a FirecREST asynchronous tasks is completed.

IV. FirecREST API specification

The FirecREST API has been described using `OpenAPI` specification [13] in `YAML` format. `OpenAPI` is a language-agnostic standard for describing all aspects of a REST API: endpoints, endpoint operations, operation input parameters, operation output, and authentication methods. Moreover, the FirecREST project can leverage on open-source tools [17] built to support OpenAPI that simplify the reading and writing the API, API documentation, and automatic library generation.

Each REST calls have a set of error responses that are part of the API specification. Such error responses are forwarded to the HTTP 4XX message and provide insight of the occurring error.

A. Development effort

FirecREST API has been developed by a team of around five engineers over two years. The code is mainly developed in Python and represents about 20,000 lines of code. FirecREST is open source and available at the following address: https://github.com/eth-cscs/firecrest.

V. Related work

The European project UNICORE [18] [19] aims to develop a general-purpose federation software suite by following standard Grid and Web services. The Uniform Interface to Computing Resources UNICORE framework can federate in a single view different systems ranging from high-end HPC systems to single Linux servers. UNICORE development follows a project-based funding which constraint the project to a discontinuous pace development focusing on adding new features. For instance, key features such as using modern protocols for authentication and authorization or using an API specification like OpenAPI [13] are not yet integrated. Our

work offers a simpler approach by using a standard API which reduces its development cost compared to UNICORE.

NEWT [20] [21] aims to make HPC resources easily accessible to scientists by using Web applications. NEWT provides a RESTful API and we investigated its feature set prior to starting this work. We found that some key aspects of our requirements were not integrated inside NEWT. For instance, NEWT has a monolithic architecture with one point of failure, authentication and authorization should be ported to modern protocols to integrate with third party clients and delegation needs to be implemented. We concluded that NEWT has been developed and tailored for NERSC ecosystem requirements, and porting it to CSCS environment requires an equivalent development effort.

Recently, Slurm proposed a REST API by introducing a new daemon that captures REST calls. Such API is meant to enable clients in the same infrastructure to control Slurm, it is not designed to face directly calls from the web. For instance, it does not provide an authorization mechanism to enable secure interfacing with the web. Attempts to provide a web-facing interface to Slurm exists [22], however, such attempts are limited to query Slurm about job status to provide a dashboard and do not include job submission or data transfer.

Globus Transfer API [23] is a REST API that allows to transfer files together with a set of utilities such as monitoring the transfer progress. Globus provides its own authentication mechanism which limits its portability. Our contribution differs from Globus as we use object storage supporting both S3 and SWIFT interfaces instead of a single file transfer service.

VI. Conclusions and future work

In this work, we present FirecREST, a RESTful Web API infrastructure that scientific gateways utilize to integrate with the High-Performance resources and services available from the Cray XC system at CSCS. We intend to use FirecREST with the use cases presented in this paper.

As new use cases will emerge, new requirements will be requested for FirecREST. As a concrete example, CSCS and the Paul Scherrer Institute (PSI) are collaborating to couple PSI scientific devices with CSCS compute capability. FirecREST is a key component to enable this connection, and it will be extended with a reservation service of compute nodes and a configurable data transformation service [24].

References

[1] M. Masse, *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc., 2011.

[2] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000, vol. 7.

[3] L. Richardson and S. Ruby, *RESTful web services*. " O'Reilly Media, Inc.", 2008.

[4] K. A. Lawrence, M. Zentner, N. Wilkins-Diehr, J. A. Wernert, M. Pierce, S. Marru, and S. Michael, "Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4252–4268, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3526

[5] L. Talirz, S. Kumbhar, E. Passaro, A. V. Yakutovich, V. Granata, F. Gargiulo, M. Borelli, M. Uhrin, S. P. Huber, S. Zoupanos *et al.*, "Materials cloud, a platform for open computational science," *arXiv preprint arXiv:2003.12510*, 2020.

[6] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, and B. Kozinsky, "AiiDA: automated interactive infrastructure and database for computational science," *Computational Materials Science*, vol. 111, pp. 218 – 230, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0927025615005820

[7] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter Notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90.

[8] Keycloak Team. Open source identity and access management. [Online]. Available: https://www.keycloak.org/

[9] The Cloud Connectivity Company. Kong gateway. [Online]. Available: https://konghq.com/kong

[10] The Pallets organization. Flask. [Online]. Available: https://flask.palletsprojects.com/

[11] Paramiko Team. Paramiko, a python implementation of SSHv2. [Online]. Available: http://www.paramiko.org/

[12] OpenBSD Project. OpenSSH. [Online]. Available: https://www.openssh.com/

[13] OpenAPI Initiative. The OpenAPI Specification: a broadly adopted industry standard for describing modern APIs. [Online]. Available: https://www.openapis.org/

[14] Redislabs. Redis. [Online]. Available: https://redis.io/

[15] OpenID Foundation. OpenID connect. [Online]. Available: https://openid.net/connect/

[16] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

[17] Smartbear. Swagger, API Development for Everyone. [Online]. Available: https://swagger.io/

[18] D. W. Erwin and D. F. Snelling, "UNICORE: A grid computing environment," in *Euro-Par 2001 Parallel Processing*, R. Sakellariou, J. Gurd, L. Freeman, and J. Keane, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 825–834.

[19] M. Romberg, "UNICORE: Beyond web-based job-submission," in *Proceedings of the 42nd Cray User Group Conference*, 2000, pp. 22–26.

[20] S. Cholia, D. Skinner, and J. Boverhof, "NEWT: A RESTful service for building high performance computing web applications," in *2010 Gateway Computing Environments Workshop (GCE)*. IEEE, 2010, pp. 1–11.

[21] S. Cholia and T. Sun, "The NEWT platform: an extensible plugin framework for creating ReSTful HPC APIs," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4304–4317, 2015.

[22] EDF. Slurm-web. [Online]. Available: https://github.com/edf-hpc/slurm-web

[23] The University of Chicago. Globus, transfer API documentation. [Online]. Available: https://docs.globus.org/api/transfer/

[24] S. Leong, H.-C. Stadler, M. Chang, J. Dorsch, T. Aliaga, and A. Ashton, "SELVEDAS: A data and compute as a service workflow demonstrator targeting supercomputing ecosystems," in *SuperCompCloud: 3rd International Workshop on Interoperability of Supercomputing and Cloud Technologies*. IEEE, Nov 2020, accepted.