Integration of iRODS data workflows in an extensible HTTP REST API framework

Mattia D'Antonio
CINECA - Interuniversity Consortium
Casalecchio di Reno (BO) - Italy
m.dantonio@cineca.it

Claudio Cacciari
CINECA - Interuniversity Consortium
Casalecchio di Reno (BO) – Italy
c.cacciari@cineca.it

Giuseppa Muscianisi
CINECA - Interuniversity Consortium
Casalecchio di Reno (BO) – Italy
g.muscianisi@cineca.it

Michele Carpenè
CINECA - Interuniversity Consortium
Casalecchio di Reno (BO) - Italy
m.carpen@cineca.it

Giuseppe Fiameni
CINECA - Interuniversity Consortium
Casalecchio di Reno (BO) – Italy
g.fiameni@cineca.it

ABSTRACT

We developed a set of HTTP REST APIs on top of iRODS to support users of different communities to automate both ingestion and retrieval data workflows. We built a common REST APIs layer by implementing basic functionalities, including the interaction with iRODS, within an extensible framework (RAPyDO: Rest Apis with Python on Docker) that we developed and adopted to build communities-specific REST APIs. More in details, we are collaborating with the EUropean DATa infrastructure EUDAT Collaborative Data Infrastructure (CDI); European projects like EOSC-hub and SeaDataCloud; national initiatives in collaboration with Telethon Foundation (a non-profit organization for genetic diseases research) and SIGU (Italian Society for Human Genomics). All endpoints are written by using the Python language on the Flask framework. APIs are served through an uWSGI web server deployed within a Docker container. We created a wrapper of the python irods client (PRC) to let both the core framework and communities specific APIs for easily interact with iRODS by supporting all main authentication protocols like native passwords, Pluggable Authentication Modules (PAM), Grid Security Infrastructure (GSI). To be able to support all required authentication methods we contributed to the PRC development by implementing authentication modules for both GSI and PAM. Most of iRODS-based functions that we developed can be mapped against corresponding icommands like ils, iget, iput, imv, icp, imeta, irule, iticket but also more complex functionalities have been realized, for instance streamed read/write operations from/to network sockets. To be able to execute data intensive and complex workflows, we also introduced an asynchronous layer implemented on Celery, a task management queue based on distributed message passing.

Keywords

HTTP API, Python, data management, iRODS, REST API, authentication, PAM, GSI

iRODS UGM 2019, June 25-28, 2019, Utrecht, Netherlands

INTRODUCTION

Data management is becoming one of the most challenging topic in computer science, since requirements for data production and manipulation often overcame whom for data analysis. We at CINECA [1], the Italian Interuniversity Consortium, are involved in many European Projects and National Initiatives strongly based on the requirement of efficient and flexible data workflows. Every project has its own very specific set of constraints and requirements, but by comparing all of them, it is possible to identify several common needs. By working on these requirements, a shared data management layer can be implemented and used as a base for every data oriented project. In this paper, we will describe our current solution, based on a data management layer built over iRODS. A quick overview of the main projects that involve our group can highlight the common needs we are working on.

EUDAT Collaborative Data Infrastructure (CDI)

The EUDAT CDI [2] is a European e-infrastructure of integrated data services and resources to support research. This infrastructure and its services have been developed in close collaboration with over 50 research communities spanning across many different scientific disciplines and involved at all stages of the design process. The EUDAT CDI network provides a range of services for data upload, retrieval, identification, description, replication through a series of core services like B2SAFE [3] (data storage), B2STAGE [4] (data transfer) and many other (like B2HANDLE, B2SHARE, B2FIND, B2NOTE).

B2SAFE is a robust, safe and highly available service, which allows community and departmental repositories to implement data management policies on their research data across different geographical and administrative domains in a trustworthy manner. Currently iRODS is used at the EUDAT sites for the federation of the data nodes and the node-wise policy enforcement.

To offer functionalities for data transfer between EUDAT resources and external computational facilities we built the B2STAGE service. B2STAGE supports different functionalities allowing users to either stage data outside EUDAT or ingest computational results while maintaining the coherency of associated Persistent Identifiers (PIDs). This service offers two interfaces, one based on the GridFTP protocol [5] and one based RESTful HTTP endpoints. The main requirements of this service are the ease of use, the interoperability with other EUDAT services and the support for the automation of ingestion and retrieval workflows. The HTTP API interface of B2STAGE is built by adopting the common strategies discussed in this paper on top of B2SAFE and iRODS. Furthermore, B2STAGE HTTP-API is in turn the base to build other services (like SeaDataCloud HTTP API, see paragraph below) and is part of the EOSC-HUB.

EOSC-hub

The European Open Science Cloud (EOSC) [6] aims to offer open and seamless services for archiving, management, analysis and re-use of research data, across different scientific disciplines. Within this project the Hub, a single contact point for European researchers, represents a particular role and innovators to discover, access, use and reuse a broad spectrum of resources for advanced data-driven research. EOSC-hub brings together multiple service providers from the EGI Federation [7], EUDAT CDI, INDIGO-DataCloud [8] and other major European research infrastructures to deliver a common catalogue of research data, services and software for research. Through EUDAT CDI, B2STAGE is part of the EOSC-hub network, bringing to an increased level of flexibility and interoperability required from this service.

SeaDataNet - SeaDataCloud

SeaDataNet [9] is a standardized infrastructure for managing the large and diverse data sets collected by the oceanographic fleets and the automatic observation systems. This infrastructure aims to aggregate and standardize the highly fragmented marine observing system based on more than 600 scientific data collecting laboratories (from governmental organizations to private industries) in the countries bordering the European seas. Data are collected by using various sensors on board of research vessels, submarines, fixed and drifting platforms, airplanes and satellites, to measure physical, geophysical, geological, biological and chemical parameters, biological species etc. SeaDataNet infrastructure was implemented during the SeaDataNet project (2006-2011), grant agreement 026212, EU Sixth Framework Programme and consolidated in the second phase, SeaDataNet 2 project (2011-2015), grant agreement 283607, EU Seventh Framework Programme. SeaDataCloud project (2016-2020), grant agreement 730960, EU H2020 programme, is the third (and current) phase and it aims at considerably advancing SeaDataNet Services and increasing their usage, adopting cloud and High Performance Computing technology for better performance. This background highlights the main requirements that these services will have to satisfy: efficient and simple ingestion interfaces; support for data workflows for data manipulation, format conversion, quality verification; facilities for data searching and retrieval. SeaDataCloud (SDC) CDI HTTP API are built over B2STAGE HTTP API by extending the EUDAT services with the inclusion of custom endpoints. In particular, SDC introduced the integration with Rancher [10] for the execution of quality check workflows based on docker containers [11] and greatly enhanced the use of asynchronous endpoints (implemented by means of the Celery [12] framework).

The Genomic Repository

The Genomic Repository is a data and metadata archive born to meet the requirements from two different partners and leading to the definition of a single solution adopted for both the platforms. The Telethon Foundation [13], a non-profit organization for genetic diseases research, is involved in the Undiagnosed Diseases Program (UDP) [14] pursuing the goal of providing a diagnosis to pediatric patients with a genetic disease without a name. To match this goal all clinical and genomic data have to be stored into a common platform able to compare such information with similar databases across the world, looking for similarities and hopefully the identification of *second cases*. Genomic data, obtained from massive next-generation sequencing (NGS) platforms, require computationally intensive analyses available on HPC computational centers.

The second use case is from the NIG (Network of Italian Genome) project of the Italian Society for Human Genetics (SIGU) [14]. The main purpose of this project is the definition of an Italian Reference Genome for the identification of genes responsible for genetic diseases and susceptibility genes for complex diseases in both basic and translational researches; genetic variants responsible for inter-individual differences in drug response in the Italian population; new targets for both diagnosis and treatment of genetic diseases. This project required the creation of a shared database containing data from nucleic acids sequencing of hundreds to thousands of Italian subjects. Sequencing data are then analyzed by executing HPC bioinformatics workflows and results are stored into the database and linked to phenotypic information. A final procedure collects all data to produce aggregated results such as distinctive Italian variants, variant frequencies, variant geographical distribution, and genotyping distribution.

Both the projects required to access to information in an automatable way to be able to interact with external initiatives and the opportunity to share data on HPC clusters in a secure way. These requirements have been achieved with the adoption of HTTP APIs and iRODS. In particular, iRODS has been identified as an important technology to make data available on computing nodes by preserving the high security levels needed to treat with human clinical data.

THE RAPYDO FRAMEWORK

To share technological solutions in very different contexts, from oceanography to genomics, we implemented a common framework adopted by all these projects and named RAPyDo [15] (acronym for Rest Api with Python on Docker). RAPyDo is a wrapper for docker-compose [16], a tool for defining and running multi-container Docker [17] applications. Docker is a tool implementing high-level APIs to provide lightweight containers that run processes in isolated environments by packaging an application and its dependencies in a virtual container. RAPyDO is able to merge several levels of configurations (base configurations, project-level configurations and deployment-level configurations) to create dynamic docker-compose definitions allowing easy deployment on every platform (from its own laptop for development purpose to production servers). Furthermore, RAPyDO is an extensible and modular framework and it implements basic functionalities and services integrations that can be used or extended by projects. RAPyDo supports iRODS as a base technology for data management and implements a wrapper client based on the python-irods-client [18]. RAPyDO projects can be administrated and deployed through a controller script implementing all the logics over docker-compose and able to merge configurations (e.g. to enabled and disabled supported services) and functionalities (e.g. base and custom endpoints). An overview of the main components is described below.

RAPyDo Architecture

RAPyDO is mostly implemented in Python programming language, in particular the controller and the http-api components. An optional Web interface is implemented with Angular.

HTTP-APIs are powered by python Flask micro-framework and provides access to set of RESTful HTTP API connected to all the other backend services (e.g. database, tasks queue, data storage, etc). When development mode is enabled, REST endpoints are directly served by Flask through uWSGI, in production mode an nginx reverse proxy is deployed ahead of the backend to enhance security and introduce SSL certificates management. Supported services, when enabled, are automatically deployed as docker containers but can be configured as external resources with independent deployment. An overview of the main components is show in Figure 1

Supported services

Databases

RAPyDo supports several databases and in particular relational databases (MySQL, MariaDB, PostgreSQL, SQLite), graph databases (Neo4j) and no-sql databases (MongoDB). All these databases can be enable to support the authentication functionalities to manage usernames, password, groups and sessions (based on JWT tokens)

Celery

HTTP-APIs endpoints typically implement synchronous operations by directly providing to the client the result of the requested resources. This basic schema implies fast responses (no more than few seconds) to avoid connection timeouts but this not always can be achieved, in particular when handling with huge amount of data. Asynchronous endpoints provide to the client an operation identifiers than can be used to track the progress of the request until it completes. RAPyDo adopted the Celery framwork to introduce asynchronous operations. Celery is a task management queue based on distributed message passing and built on top of RabbitMQ.

iRODS

RAPyDo adopted iRODS as base layer for data management, since this technology offers several valuable advantages. iRODS implements data virtualization, allowing access to distributed storage assets under a unified name-space and allows to access from multiple clients: command line (icommands), web interface (webdav), ftp

transfer (grid-ftp). Furthermore, it integrates certificate-based authentication with the GSI plugin, irules to trigger actions based on events at collection or data objects level, access control list (ACL) mechanisms to allow for the implementation of complex access rights policies preserved at every level, regardless of the access method.

The different access methods and the support for ACLs allow to successfully integrating the software stack with the use High Performance Clusters (HPC) for analysis purpose. In particular all data stored into iRODS can be shared on the different components by preserving all the security policies.

Rancher

The use of High Performance Clusters is not always possible or flexible enough. In specific contexts, Docker can be used to execute analyses, quality checks, data validation tasks and other project-specific software. Rancher is an open source multi cluster management platform used to deploy and manage docker containers (through the Cattle engine up to version 1.6 and through Kubernetes from version 2.0). RAPyDO HTTP APIs are able to communicate with Rancher APIs (version 1.6) to automatically deploy and interact with docker containers executed in a distributed environments.

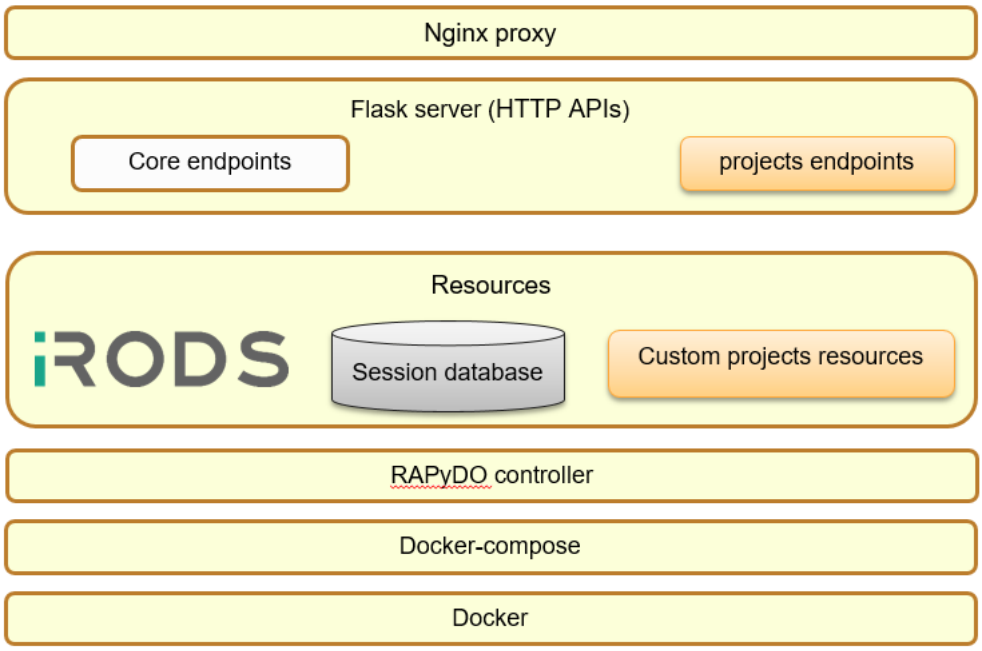

Figure 1. Architecture of the main components of the RAPyDo framwork

IRODS CLIENT

iRODS officially supports Python through the python-irods-client [18] (PRC), an open source project released on Github. RAPyDO HTTP APIs integrates PRC to implement all iRODS-based operations on both Flask (HTTP sync endpoints) and Celery (asynchronous tasks). We created a wrapper client based on the python-irods-client to implement utility operations used from both endpoints and tasks. Most of the methods implemented by this client can be divided in three main categories:

- Methods that can be strictly mapped on icommands, e.g. list(), mkdir(), copy(), put(), get(), move(), remove(), set_permissions(), get_metadata(), ticket() and others. These methods can be respectively mapped on ils, imkdir, icopy, iput, iget, imv, irm, ichmod, imeta ls, iticket.

- Simple utilities methods without a corresponding icommand, e.g. exists(), is_collection(), is_dataobject() and others
- Method to perform more complex operations, e.g.
    - write_file_content(path, string)
    - get_file_content(path) returning a string
    - read_in_chunks(path, chunk_size) returning an iterator
    - read_in_streaming(path) directly streaming data object content as Flask stream
    - write_in_streaming(path) directly writing data object from Flask streams

This list not exhaustive since it only include general purposes methods, took as examples.

Authentication

RAPyDO HTTP APIs support all main iRODS authentication protocols and in particular native credentials, Grid Security Infrastructure (GSI) and Pluggable authentication modules (PAM).

Native credentials (usernames and passwords defined into the iRODS database) is the default authentication protocol and the python-irods-client natively supports it. In a production environment, iRODS is often secured by means of GSI certificates or by using heterogenes methods (e.g. LDAP) supported by PAM protocol. Since PRC was lacking the support for GSI and PAM authentication, we contributed to the development by implementing the specific missing modules. The GSI integration is a completed task and the contribution is merged into the main branch since around January 2017. The PAM integration is completed respect to some use cases but requires improvements to achieve a larger audience. The current PAM module is merged into the main branch since December 2018.

To implement the Pluggable authentication modules (PAM) support we introduced a PluginAuthMessage to encapsulate PAM requests

```
class PluginAuthMessage(Message):

    _name = 'authPlugReqInp_PI'

    auth_scheme_ = StringProperty()

    context_ = StringProperty()
```

The same class is also used to perform GSI requests.

PAM requests are propagated to the iRODS server by sending ah authorization request with type RODS_API_REQ and apiNumber 1201, equivalent to AUTH_PLUG_REQ_AN. The body of the message is a PluginAuthMessages containing a context based on user, pam password and ttl.

```
ctx_user = '%s=%s' % (AUTH_USER_KEY, self.account.client_user)

ctx_pwd = '%s=%s' % (AUTH_PWD_KEY, self.account.password)

ctx_ttl = '%s=%s' % (AUTH_TTL_KEY, "60")

ctx = ";".join([ctx_user, ctx_pwd, ctx_ttl])

message_body = PluginAuthMessage(

    auth_scheme_=PAM_AUTH_SCHEME,

    context_=ctx

)
```

```
auth_req = iRODSMessage(
    msg_type='RODS_API_REQ',
    msg=message_body,
    int_info=1201
)
```

The server response in stored in a AuthPluginOut:

```
class AuthPluginOut(Message):
    _name = 'authPlugReqOut_PI'
    result_ = StringProperty()
```

Containing a new temporary password with validity equivalent to the negotiated ttl and used to initialize a native password connection

```
auth_out = output_message.get_main_message(AuthPluginOut)
self._login_native(password=auth_out.result_)
```

To implement the Grid Security Infrastructure (GSI) a more complex workflow has been introduced. As high-level overview, the GSI protocol can be summarized in three main steps:

1) send to iRODS server a message to request GSI authentication. This step is similar to the PAM use case but providing a different auth_scheme (GSI_AUTH_PLUGIN) and a different context (AUTH_USER_KEY=self.account.client_user)
2) create a context handshaking GSI credentials by creating a GSI SecurityContext to send the GSI client token and to receive corresponding token from the server
3) Complete the protocol by sending the username through a requesto to the iRODS server with an apiNumber 704, equivalent to AUTH_RESPONSE_AN

CONCLUSIONS

In this paper, we presented a modular and extensible framework (RAPyDo) that we developed to share common strategies among European and National projects whom we are involved into. Our main intention is to extract from each project all requirements that can be satisfied by applying previously identified and implemented solutions. New requirements are analyzed to identify candidates for generalizable solutions to be included in the framework and to be reused in further projects. In this way the framework can growth and enhance itself at each application. RAPyDo supports several services (databases, task queues, interfaces) and in particular adopted iRODS as basic data management technology. The HTTP APIs module included in RAPyDo also introduces a python client based on the official python-irods-client (PRC) implementing utility functions and authentication protocols like GSI and PAM. We already adopted this framework in several European and National projects like the EUDAT Collaborative Data Infrastructure (CDI), the Hub of European Open Science Cloud (EOSC-hub), SeaDataCloud HTTP APIs and a Genomic Repository built in collaboration with Telethon Foundation and Italian Society for Human Genetics (SIGU). All these projects are data repository based on iRODS technology. Furthermore, the framework is also

adopted onto European Projects not based on iRODS, like I-Media Cities (grant agreement N° 693559E from the uropean Union's Horizon 2020 research and innovation programme) and Meteo Italian Supercomputing Portal MISTRAL (funded under the Connecting Europe Facility (CEF) – Telecommunication Sector Programme of the European Union). As future perspectives, we plan to enhance the current implementation by expanding the number of supported projects. Furthermore, we need to fight against the natural obsolescence of adopted solutions by continuously improve the framework by investigating state-of-the art technologies candidate to be included in the supported stack.

ACKNOWLEDGMENTS

The implementation of the GSI and PAM modules started by forking the code of the python irods client [18] and PRC is also a base component of our APIs.

For the implementation of the GSI plugin a special mention is for Roberto Mucci and Paolo D'Onorio De Meo, not included as authors of this paper. Paolo D'Onorio De Meo is also one of the main developers of the RAPyDo framework.

EUDAT is funded by from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

EOSC-hub is funded by from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 777536

SeaDataCloud is funded by from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 730960

REFERENCES

[1] CINECA - Italian Interuniversity Consortium, https://www.cineca.it/en

[2] EUDAT CDI, https://eudat.eu/eudat-cdi

[3] B2SAFE, https://www.eudat.eu/b2safe

[4] B2STAGE, https://www.eudat.eu/b2stage

[5] GridFTP - Globus Toolkit, http://toolkit.globus.org/toolkit/docs/latest-stable/gridftp/

[6] EOSC-hub, https://www.eosc-hub.eu

[7] EGI Federation, https://www.egi.eu/federation/

[8] INDIGO DataCloud, https://www.indigo-datacloud.eu/

[9] SeaDataNet, https://www.seadatanet.org/

[10] Rancher: Container Orchestration, https://rancher.com/

[11] Docker Container Platform, https://www.docker.com/

[12] Celery: Distributed Task Queue, http://www.celeryproject.org/

[13] Telethon Foundation, http://www.telethon.it/en

[14] SIGU - Italian Society for Human Genetics, https://www.sigu.net/

[15] RAPyDo Framework, https://github.com/rapydo

[16] Docker Compose, https://docs.docker.com/compose/

[17] Docker Container Platform, https://www.docker.com/

[18] Python iRODS Client (PRC), https://github.com/irods/python-irodsclient