USER GROUP MEETING
2022 PROCEEDINGS

PUBLISHED BY THE iRODS CONSORTIUM

iRODS
User Group Meeting 2022
Proceedings

14TH ANNUAL CONFERENCE SUMMARY

The iRODS User Group Meeting of 2022 gathered together iRODS users, Consortium members, and staff to discuss iRODS-enabled applications and discoveries, technologies developed around iRODS, and future development and sustainability of iRODS and the iRODS Consortium.

The in-person and virtual four-day event was held from July 5th to 8th, hosted by KU Leuven in Leuven, Belgium, with 103 people attending from 10 countries. Attendees and presenters represented over 80 academic, governmental, and commercial institutions.

TALKS AND PAPERS

LIGHTNING TALKS

Customizable metadata @ the Maastricht Data Repository

Daniel Theunissen – Maastricht University

Using Virtual Research Environment (VRE) desktop to sync iRODS data

Ton Smeele – Utrecht University

Upcoming 4.3.? GenQuery

Kory Draughn – iRODS Consortium

Upcoming Hackfest-GA4GH Data Repository Service

Mike Conway – NIH / NIEHS

Planned integration between RSpace and iRODS

Rory Macneil – Research Space

A selection of iRODS prototypes & more

Christine Staiger – Wageningen University

Minimal iRODS Testing Environment Demo

Alan King – iRODS Consortium

Best practices in iRODS System Administration - to Kickstarter!

John Constable – Wellcome Sanger Institute

Dockerized iRODS Server

Kaivan Kamali – Penn State University

Sustainable and FAIR Data Ecosystem, supporting new insights in Life Sciences

Berenice Wulbrecht, Carl Latham, Valerie Morel
ONTOFORCE
carl.latham@ontoforce.com

ABSTRACT

The last decade Data has become the new oil, and as a key asset in all industries. However, to leverage the power of data, it needs to be refined and distributed. This transformation has largely affected the Life Science field from academy to industry, who has adhered largely to the FAIR principles of findability, accessibility, interoperability, and reusability. Data is not consumables from an experiment anymore, it is now set to be re-used, re-interpreted… Some challenges remain like creating a consolidated view of disparate and siloed data or setting the infrastructure to store, search, retrieve and analyze data.

iRODS stands for 'Integrated Rule-Oriented Data System'. It is open-source data management software that links unstructured data to metadata and is used for distributed storage and data management automation.

The knowledge platform, DISQOVER, enables data-driven decisions and accelerates research by unlocking insights from siloed data. The platform is integrating and harmonizing data silos across internal, public, and third-party sources into an integrated knowledge graph. DISQOVER helps you do your research in one place to answer complex questions and solve problems. One consistent and easy-to-use interface democratizes access to data through self-service knowledge discovery, allowing each scientist to access and explore data and generate insights.

We proposed here the integration of iRODS and Disqover, to offers a sustainable data infrastructure to store and search for data, knowledge, and insights. The proposal highlights the FAIR data principles. As data and meta-data captured in various source systems are centralized in iRODS. Data can eventually be processed with an entity extraction service to further enrich meta-data. Relevant data and meta-data re loaded on the Disqover platform. The data are integrated and harmonized using various ontologies and reference datasets. Researchers can now very easily search and explore available data on Disqover and redirect their requests to iRODS to access original data. The integration of Disqover and iRODS platform provide a self-service data access for research and a sustainable data ecosystem. Such integration has broad applicability supporting research and development in Life Sciences.

From SRB to iRODS: 20 years of data management at the petabyte scale

Jean-Yves Nief, Yonny Cardenas
CC-IN2P3
nief@cc.in2p3.fr

ABSTRACT

CC-IN2P3 has been using SRB and then iRODS in a wide variety of projects and use cases for the last 20 years.

CC-IN2P3 is a data center hosting services such as computing and data storage for international projects mainly in the fields of subatomic physics and astrophysics. Data management has always been a key activity for a data center such as CC-IN2P3, due to the ever growing size of the projects, their international dimension.

This talk will emphasize on the evolution of the data management needs, the pitfalls, the endless migration cycle (both hardware and software) over the years.

It will also focus on the ongoing prospects, especially the long term data preservation needs and open science.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

MrData: An iRODS Based Human Research Data Management System

Blake G. Fitch
Max Planck Institute for Biological Cybernetics[1]
blake.fitch@tuebingen.mpg.de

Sebastian Müller*
Max Planck Institute for Biological Cybernetics[1]
Eberhard Karls University Tuebingen[2]
sebastian.mueller@tuebingen.mpg.de

Dario Bosch
Max Planck Institute for Biological Cybernetics[1]
Eberhard Karls University Tuebingen[2]
dario.bosch@tuebingen.mpg.de

ABSTRACT

MrData is an iRODS based archival system for human subject research producing medical image data. MrData was designed to automate collection and archival of data flowing from a Siemens 9.4 Tesla MRI system. Of particular importance to this project was managing metadata related to human subject recruiting in a GDPR compliant manner. We chose Castellum, a Max Planck developed open source system specifically designed for managing human subject data securely, and we worked with that team to integrate it with the MrData system. An additional requirement for us was "mixed use" metadata, that is information necessary for both subject recruiting and scientific processing. Mixed use metadata, such as handedness, is managed by Castellum but also passed to MrData for scientific and archival purposes securely, and without manual transcription. Our system never records any personally identifying information at the MRI scanner, so the resulting image files are never contaminated with a subject name, date of birth, etc. MrData is based on the iRODS ecosystem, GitLab, Flask, and Python processes, and deployed as a set of Docker micro-services. We will present an overview of this project, including current production status and future directions. We welcome feedback on whether some or all of this system would be usefully open-sourced.

Keywords

iRODS, MRI, medical imaging, data management, Python, GDPR, DICOM, Flask, Castellum.

INTRODUCTION

The MrData system was created to automate, and make GDPR (General Data Protection Regulation)[1] compliant, the handling of human subject medical image data for research at the Max Planck Institute for Biological Cybernetics. The initial imaging system we focused on is a Siemens 9.4 Tesla MRI system. MrData is built based on the iRODS[2] ecosystem.

GDPR compliance informed several aspects of the MrData system architecture. Early in the project, we made a decision to use the Castellum[3] system for human subject recruiting and personal information data management. The alternative, a system where Personal Health Information (PHI)[4] and Personal Identifiable Information (PII) are managed by the same system that manages the scientific data archive, has been done in earlier work. Our objective was a separation of concerns. PHI and PII would be kept in one security domain managed by Castellum, and scientific information would be kept in another security domain managed by MrData. A challenge was ensuring that mixed use metadata, needed in both domains, would have a single root source in Castellum but be available for scientific data archival structure and search.

The MrData system is implemented as a set of micro-services deployed in Docker[5] containers. We use Ansible[6][7] to build and deploy these containers into production on a bare metal Linux server, as well as into Linux virtual machines for testing. The containers are a mix of services we developed in-house using Python, and services such as

iRODS UGM 2022 July 5-8, 2022, Leuven, Belgium

[1]High-field Magnetic Resonance Center, Max Planck Institute for Biological Cybernetics, Tuebingen, Germany
[2]Department of Biomedical Magnetic Resonance, Eberhard Karls University Tuebingen, Tuebingen, Germany

iRODS, Davrods[8], Metalnx[9], and Nginx[10], developed externally. This mix of services is integrated using Docker infrastructure, managed by Ansible, yielding a single cohesive application deployed on a single server.

The remainder of this paper will be structured as follows: we describe our use of Castellum, management of mixed use metadata, the workflow a scientist experiences using MrData, the infrastructure environment, review each micro service component, compare our solution with other options, and conclude with future research directions.

CASTELLUM AND MIXED USE METADATA

Castellum provides a web interface for investigators and administrators to recruit human research subjects. It manages all relevant personal health information and contact information for subjects in a secure, private fashion. Castellum also manages additional configurable subject attributes which may be required for a recruiting search. A user of Castellum first defines a study and then uses Castellum to recruit subjects for that study. A key feature of Castellum is that each subject in a study is given a unique, randomized pseudonym which is used to refer to that subject in all research documents and data. Subjects are only referred to using pseudonyms in the MrData archival system.

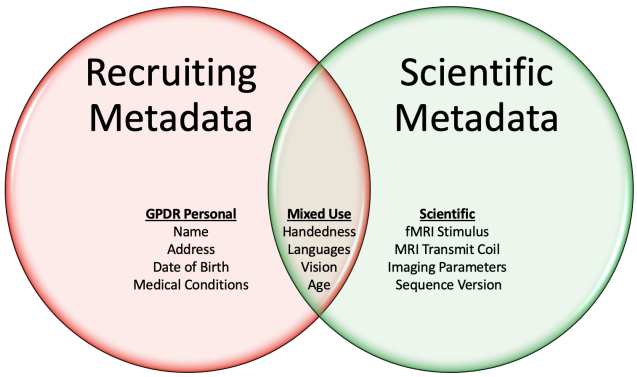

Figure 1. Mixed use metadata.

A challenge to using separate systems for subject recruiting and scientific data archival is mixed use metadata. Mixed use metadata is comprised of subject attributes that are required for both recruiting and for scientific data processing. For MRI related neuroscience, examples of mixed use metadata attributes include handedness, languages spoken, year of birth. These attributes may be used in Castellum to select subjects for a study and also for using, or reusing, scientific data. An important requirement is that any given piece of metadata have a single source and that metadata not be manually transferred between systems. For this reason we worked with the Castellum team to create a one-way API by which the MrData system can extract mixed use metadata to use in archiving and searching scientific data.

Figure 2. Mixed use metadata moves one-way from Castellum to MrData.

The mixed use metadata must be explicitly authorized for export on an attribute by attribute basis. Only attributes that are approved by our data security coordinator can be authorized, and only by an administrative action. For

these attributes, the MrData system keeps a full history, even if a subject is deleted from Castellum or a study is deactivated. This is done to maintain the ability to access the archived scientific data which we are obligated to do for a minimum of ten years. It is however possible to remove a given subject from search or user data access as required.

TERMS and DEFINITIONS

This section will review terms and definitions useful in understanding the MrData system.

Study and StudyID

Generally, a study is a project where the scientific investigator will work on a specific research question by running experiments on a group of subjects. Studies may also be defined for calibration or other projects where we wish to archive the resulting data but will not have actual human subjects. A study is defined in Castellum where it is given an StudyID, and may also be given a text descriptor. Castellum can then be used to recruit subjects who will participate in the study.

Subject and Pseudonym

A subject is a person who is recruited to participate in a study using Castellum. Once a subject is recruited for a study, that subject will be given a study-specific pseudonym by Castellum. If the subject participates in multiple studies, the subject will be assigned a different pseudonym in each study. The pseudonym is a randomized, alphanumeric character sequence, and is used to represent the subject everywhere outside of Castellum. We do not record the subject's name, which is PII, at the MRI scanner as would be traditional in a healthcare context.

Experiment and ExperimentID

An experiment is a single data collection session with a subject who undergoes the protocols of the study. In the case of the 9.4T MRI scanner, the subject would meet the MRI operator who would perform a scan that might last a few hours. A scan may produce several types of data that need to be archived. A single subject can participate in several experiments, in one or more studies.

An experiment is identified by an alphanumeric ExperimentID. This is done using MrData Experiment Registration web graphical user interface (GUI). At the web page, a StudyID, subject Pseudonym, investigator userid, and type of experiment, must be entered to acquire an ExperimentID. The ExperimentID is recorded by the MrData system and provided to the investigator who will need to enter it at the MRI scanner console in the "Patient Name" field.

Expanded Data

The data types collected by MrData can be expanded beyond the file types produced directly by the scanner. In the context of an experiment, the scientific investigator can collect arbitrary additional data. Examples are subject questionnaires, experiment parameters, MRI scanner log files, fMRI stimulus parameters, source code, and freely formulated experiment descriptions. MrData provides a directory where investigators can deposit any relevant files. This directory, including its structure, will be archived together with the recorded MRI data to help provide context for later data processing and analysis. The Expanded Data directory also provides a convenient way to preserve digital artifacts needed to facilitate reproducibility.

SCIENTIFIC WORKFLOW

The MrData project's highest objective, after complying with privacy and security requirements, is making the scientific investigator's workflow as convenient, automated, and understandable as possible. Although greater automation may be available for a given use-case, we view it as important that the minimal use-case be as simple as possible. Once the investigator realizes benefit from the minimal use-case, we hope to encourage greater use of coded pipelines, for example via NextFlow[11], in support of reproducible research.

The following is a list of the steps an investigator will follow from defining a study through accessing and processing archived data.

- Use the Castellum web GUI to define a Study and acquire a StudyID.
- Use the Castellum web GUI to recruit Subjects to participate in a Study.
- Use the Castellum web GUI to acquire a Pseudonym for an individual Subject in a Study.
- Use the MrData web GUI to Register an Experiment using the Pseudonym and StudyID, get an ExperimentID.
- At the MRI scanner console, the operator enters their userid and an ExperimentID in the Patient Name field.
- The operator performs the experiment and image data automatically streams into MrData/iRODS.
- The MRI experiment data is then accessed via iRODS (as a network share or via Python API, etc.).

MRDATA SYSTEM

This section will describe the technical details of the MrData system. This includes describing the surrounding infrastructure that enables MrData, as well as the implementation of core MrData application.

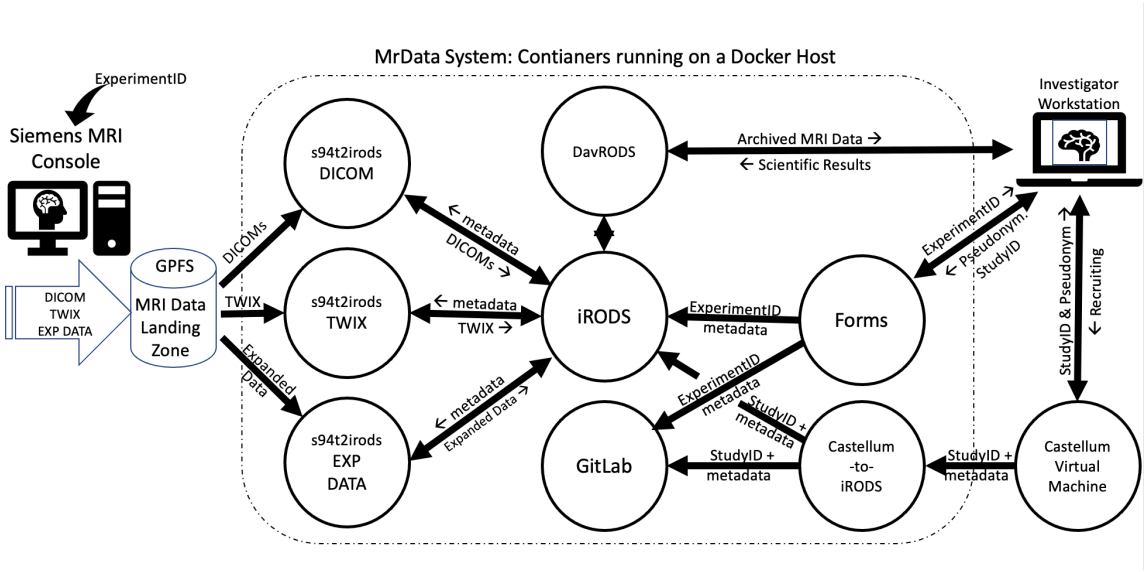

Figure 3. Overview of the MrData environment with core MrData application containers outlined.

Infrastructure Enabling MrData

This section reviews the major pieces of hardware and software infrastructure that enable the MrData system but which are not part of the core application.

Castellum Deployment

Castellum is deployed in a virtual machine maintained by a system administrator. IT staff maintains the highest level of administrative control over Castellum, however scientific investigators have several other, more limited administrative and functional roles. Castellum is the root source of information for study definition, subject assignment to studies, and all mixed use metadata. This information is made available to the investigators using the Castellum web interfaces and to the MrData system via a secure REST[12] API.

Docker Host

The Docker Host is the server on which all the MrData micro-services are run. The server is a bare-metal install of Rocky Linux[13] version 8 with Docker and little else installed. This is a single purpose server and only administrator login is permitted. It mounts an external storage system and has a local 100 terabyte (TB) disk array with a ZFS[14] filesystem. Users access this system via iRODS and HTTPS network protocols. The server is located in the same data center as our compute cluster and storage systems. It is connected to a 25 gigabit Ethernet network via a two port ether-bond. The Docker Host server has the following specs:

```
Memory:              380GB
Disk:                100TB
lModel name:         Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz
Socket(s):           2
Core(s) per socket:  16
Thread(s) per core:  2
L1d cache:           32K
L1i cache:           32K
L2 cache:            1024K
L3 cache:            22528K
```

Siemens 9.4 Tesla MRI System

We have focused on a Siemens Magnetom Plus 9.4 Tesla Magnetic Resonance Imaging scanner (Siemens Healthineers, Erlangen, Germany) running Syngo VE12U software. The 9.4T system produces research data of several types such as anatomical MRI, functional MRI, and raw MRI data that has not been processed into an image. The system can produce hundreds of gigabytes in a single session of a few hours.

The 9.4T MRI scanner produces data stored in two file formats, DICOM[15] format for processed images and "TWIX", which is a Siemens proprietary format for "raw", unprocessed, k-space MRI data.

The DICOM file format is a standard image data format produced by MRI and other medical imagining systems. The Siemens 9.4T operating software allows for setting up an automated DICOM export to an SMB network path, where files will be written to directly after creation. These files are automatically picked up and archived as they are completed.

TWIX files are not normally accessed in a healthcare environment but are required for some MRI research projects. Exporting TWIX files from the scanner interferes with any running scan, so automated exports will only be triggered in the night. However, scanner operators can choose to start an export manually at any time, which can take from minutes up to an hour, depending on the amount of data recorded. Exporting raw data is performed by using the RDS tool provided by the Yarra[16] project. The tool automatically determines which files have yet to be exported and writes them onto a preconfigured network drive. In our case, this is the MrData landing zone defined below.

MrData Landing Zone

The storage where data is stored for import to an archival system is often called a "landing zone", or LZ. MrData uses storage hosted by an IBM Spectrum Scale[17] scalable filesystem, commonly called "GPFS", exported via both Server Message Block (SMB) and Network File System (NFS) protocols. This network file system is used as an LZ for data written from the Siemens console and from the Yarra RDS software and read by MrData. The Siemens console is a Windows system which is allowed a narrow network interface to mount the LZ as a SMB share. The Docker Host mounts the LZ as an NFS share and that mounted filesystem is then made available inside relevant containers as a Docker volume.

As DICOM, TWIX, and Expanded Data files land in the LZ, they are processed by the 9.4T "uploaders" implemented

as Docker containers, one uploader for each type of data. The uploaders locate the ExperimentID in the files, extract metadata, index archival information, and put the data into the iRODS archive.

MrData Micro-services

In this section we describe each of the micro-services that make up the MrData system in detail. Each service is deployed as a Docker container. All containers run on a single Docker host.

As mentioned above, we use the Ansible Docker Module to build and start these containers. The reasoning behind this decision warrants elaboration. We first implemented all MrData services as individual virtual machines (VMs), built using Ansible. However, we could not realize enough I/O bandwidth using VMs and tests indicated Docker containers would solve this problem. Since we were targeting a single server environment, Docker Compose[18] seemed like the right container orchestrator. However, as the Docker Compose version came together, we ran into several challenges. First, Docker Compose isn't as flexible as Ansible in configuring the Docker Host, for example ensuring an NFS filesystem is mounted before starting a container that requires it. Second, Docker Compose needs to run on the Docker Host or, at least, with remote access the Docker Host's Docker instance. Ansible more naturally controls a remote host, Docker or otherwise, with ssh and no remote agent. In summary, Ansible enabled using a single "infrastructure as code" tool to configure, build, and deploy the MrData production system on a bare-metal server, while enabling the same capability for test systems in VMs by changing a single file. Ultimately, we are targeting a fully automated continuous integration environment driven by GitLab[19].

Snapshot of the MrData Docker containers running the Docker Host as a production system.

```
IMAGE                  COMMAND                  CREATED        STATUS       NAMES
s94t2irods_image_prod  "/home/mradmin/mr2ir..."  3 weeks ago   Up 3 weeks   s94t2irods_EXP_DATA_prod
s94t2irods_image_prod  "/home/mradmin/mr2ir..."  3 weeks ago   Up 3 weeks   s94t2irods_TWIX_prod
s94t2irods_image_prod  "/home/mradmin/mr2ir..."  3 weeks ago   Up 3 weeks   s94t2irods_DICOM_prod
irods_image_prod       "/docker-entrypoint...."  2 months ago  Up 2 months  irods_prod
cast2irods_image_prod  "/home/mradmin/cast2..."  3 months ago  Up 3 weeks   cast2irods_prod
forms_image_prod       "/home/mradmin/mrfor..."  3 months ago  Up 3 months  forms_prod
nginx:latest           "/docker-entrypoint...."  4 months ago  Up 4 months  nginx
davrods_image_prod     "/bin/sh -c 'dockeri..."  4 months ago  Up 4 months  davrods_prod
```

Castellum to iRODS

The cast2irods service is responsible for fetching exportable Study and Subject metadata from Castellum and making it available in iRODS for the MrData system. This is required to structure the archive and for scientific data search and processing. The service also records a history of the exportable metadata in a private, local GitLab repository.

cast2irods periodically polls a Castellum REST API, retrieving the exportable metadata for all subjects in each active study. This metadata is then organized into a canonical directory structure. The metadata is stored in this directory as YAML[20] files with key value pairs in sorted order. This directory is git diff'd with a reference repository cloned from our local GitLab. The resulting diffs represent changes in the metadata in Castellum relative to what MrData has recorded already. The diffs are merged into the cloned git repository with the exception that we do not delete files so basic archival information about deactivated studies remains available. This allows the git repository to track all exportable metadata and it's history. The modified version of the repository is then committed and pushed to GitLab. These diffs are also used to store the same information in iRODS for use by MrData services such as those that upload and archive data. When a study becomes inactive, its data is no longer exported by Castellum and there will be no further modifications to the corresponding files in GitLab or iRODS.

MrData Forms

The MrData forms service is a small Python Flask[21] application responsible for presenting a web GUI to scientific staff enabling them to register experiments. An experiment is a single session on a scanner with a single subject and

may last from tens of minutes to a few hours. A registered experiment is given an ExperimentID.

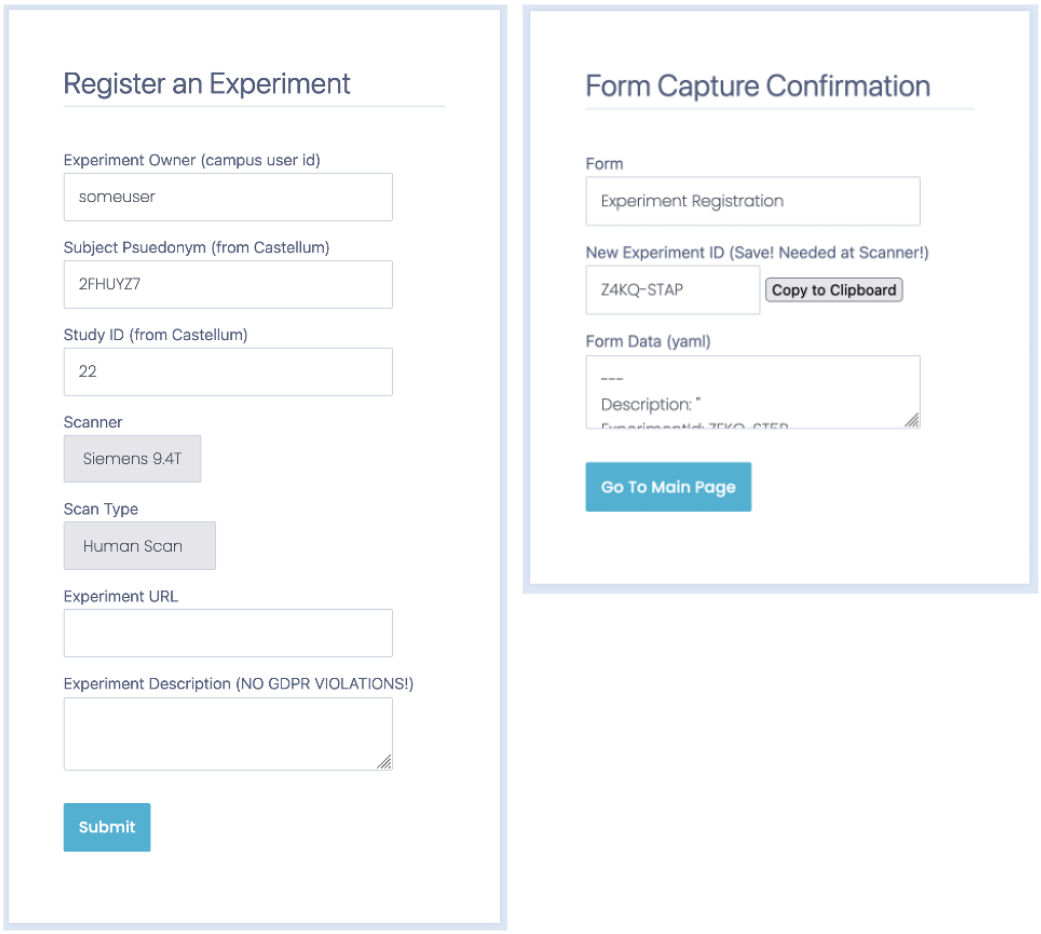

Figure 4. MrData Experiment Registration web page (left) and Confirmation web page (right).

To register an experiment and acquire an ExperimentID, one enters a Castellum StudyID, a Castellum Subject Pseudonym, and a userid into the web GUI. There are drop down tabs to select what kind of experiment will be done, which scanner will be used, etc. Once this input is validated, the Flask application returns a response page with an alphanumeric ExperimentID of 9 characters. The ExperimentID is encoded to enable detection of transcription errors. The ExperimentID with its metadata is committed to GitLab as a YAML file and also stored in iRODS for use by the upload services. The ExperimentID will later be entered into the Patient Name field on the MRI scanner user interface by the operator.

iRODS

iRODS is deployed as a Docker container which extends an existing, published PostgreSQL container. Backing store for the iRODS vault is currently provided by a 100TB ZFS raid array directly connected to the Docker Host. The iRODS container performs regular, incremental backups of the PostgreSQL database to the container backing store, which is in turn regularly backed up to tape. We have configured the iRODS deployment for Transport Layer Security (TLS)[22] only access.

TLS-only iRODS does result in some extra encryption/decryption processing for data exchanges on the Docker internal network, where TLS is arguably not required. Our current understanding is that iRODS cannot force use of encryption on one network interface but allow lack of encryption on another. This ability would be required to enable intra-container communications on the Docker Host to avoid encrypting traffic to iRODS.

Davrods

We deploy the Davrods software from Utrecht University as our main path to providing iRODS data to end-users. We have selected rclone[23] as our primary high bandwidth, data download tool which accesses iRODS through Davrods. Various other WebDAV[25] clients may be used for browsing archived data, including those that present a shared filesystem on the user's workstation.

Metalnx

Metalnx will be added as a browsing method in the near future.

NGINX Reverse Proxy

Nginx is used as a reverse proxy. All external access to MrData web based interfaces, in particular to MrData forms, Davrods, and Metalnx, go through Nginx via HTTPS. Nginx forwards traffic to the individual services on an internal Docker network using mere HTTP.

Siemens 9.4T Data Uploaders

We deploy three containers for uploading data from the Siemens 9.4T MRI scanner. They each handle one of the following categories of data:

- DICOM data – Siemens provided DICOM image files – large number of small files
- TWIX data – Siemens proprietary raw data format files – small number of large files
- Expanded Data – Files not produced directly by the MRI scanner – a user defined file tree replicated to iRODS

Each of the upload services is implemented as a Python program using the Python iRODS Client[24] library. Each service polls a particular area of the MrData LZ filesystem for data uploaded from the MRI scanner. As files flow in to the LZ areas, the processes extract metadata in a content dependent way and recover an ExperimentID. The ExperimentID is used to look up the experiment metadata in iRODS, in particular locating the StudyID. Using the ExperimentID and the StudyID, an iRODS archive path for the files to upload is determined. The headers of the DICOM and TWIX files are also processed to extract other metadata useful for searching and processing the archived data. The data and metadata are then placed in the iRODS archive as collections and objects, with metadata in Attribute/Value/Units (AVUs). The Expanded Data area is uploaded as an unmodified directory hierarchy to iRODS. In the future, we will add the ability to codify metadata to be attached to objects in the Expanded Data.

An MrData experiment data archive iRODS path is formed on the following template:

```
/MRDataZone/home/mrdata/echtdata/studies/<studyID>/experiments/<experimentID>/<DataType>/
```

End-to-end MRI data path through MrData

MrData automates the flow of data from an MRI scanner to the computational processes of the scientific investigators. Here, we follow the path of data from the MRI scanner, through the uploaders, and into the archive.

- The MRI scanner operator enters their userid and the ExperimentID in the Patient Name field on the console.

- The operator proceeds to scan the subject, potentially using stimulus required by the experimental protocol.
- During the scan, data flows from the MRI scanner to the MrData LZ mounted as a SMB network filesystem.
- The uploader processes poll the LZ mounted as an NFS filesystem, and begin processing the data as it arrives.
- The ExperimentID is extracted from each DICOM and/or TWIX file, and used to find the StudyID.
- The StudyID, ExperimentID, the file type, and metadata from file headers are used to form an iRODS path.
- The iRODS archive path for the ExperimentID is created and the DICOM and/or TWIX files put into iRODS.
- Additional metadata from the MRI file headers is added to the iRODS collections and objects as they are stored.
- The investigator (or automation) can then access the iRODS archive data for the experiment in soft real time.
- Viewing the image data may be done while the subject is still in the scanner so the experiment can be adjusted.
- Experiment Expanded Data in the LZ is copied to iRODS after 24 hours, or when the area is marked "finished".
- The experiment will be marked as finalized after given amount of time has passed or based on user input.
- Manual and automated process can now search and access all experiment data using normal iRODS methods.
- Investigators will be encouraged to store their computational results into iRODS under their own userid.

RELATED WORK

There are several systems available for medical image data handling which we explored before building MrData. We explored using XNAT [26] and Loris [27] in depth. These are both excellent systems but in each case we found ourselves extending them with significant amounts of local scripting. Further, we were not using a good deal of the available features since they overlapped with Castellum, which satisfied our high level objective of separating the human subject recruiting system from the data archival system. Finally, the modularity of the MrData system and the underlying iRODS ecosystem will ease adding additional capabilities for future data management projects.

CONCLUSION AND FUTURE WORK

We have implemented MrData, an iRODS based data management system that automates the archival of data streaming from a research MRI system. We carefully integrated with a GDPR compliant human subject recruiting system, Castellum. To avoid manual transcription of subject mixed use metadata, we worked with the Castellum team to establish a REST API to access this data via automation. The MrData system is deployed using Ansible and Docker containers in a micro-services architecture making it extensible as well as testable.

Future work includes:

- Collecting DICOM files sequence groups and archiving a tar and NiFTI[28] file for each sequence group.
- Extending MrData automated archival to additional MRI systems at our institute and beyond.
- Full automation of the continuous integration pipeline using GitLab, Ansible, and Virtual Machine targets.
- Where it makes sense, explore creating open source repositories for the MrData project.
- Trigger automatic workflow processing on the acquired data.

ACKNOWLEDGEMENTS

The authors want to thank Dr. Timo Göttel, from the Max Planck Institute for Human Development in Berlin, and the Castellum team for supporting the integration of Castellum in this project. We further want to thank Dr. Jonas Bause, from the Max Planck Institute for Biological Cybernetics in Tübingen, for his contribution to the design of the MrData project. Funding by the Deutsche Forschungsgemeinschaft (DFG German Research Foundation) under the Reinhart Koselleck Programme (DFG SCHE 658/12) and by the European Research Council (ERC Advanced Grant No 834940, SpreadMRI) is gratefully acknowledged.

References

[1] General Data Protection Regulation (GDPR). 2018. General Data Protection Regulation (GDPR) – Final text neatly arranged. [online] Available at: <https://gdpr-info.eu/> [Accessed 29 June 2022].

[2] RODS: Open Source Data Management Software. `https://irods.org` [Accessed 2 July 2022]

[3] Castellum A Privacy-Compliant Subject Management for Scientific Research `https://www.mpib-berlin.mpg.de/research-data/castellum` [Accessed 1 July 2022]

[4] Protected Health Information vs Personal Identifiable Information `https://www.accountablehq.com/post/pii-vs-phi` [Accessed 2 July 2022]

[5] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux Journal ()2014)

[6] Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks. `https://docs.ansible.com/ansible/latest/index.html` [Accessed 2 July 2022]

[7] Ansible Community Modules and plugins for working with Docker `https://galaxy.ansible.com/community/docker` [Accessed 2 July 2022]

[8] Davrods - An Apache WebDAV interface to iRODS `https://github.com/UtrechtUniversity/davrods` [Accessed 1 July 2022]

[9] Metalnx is a web application designed to work alongside iRODS. It is a graphical user interface and serves as a client that authenticates to an existing iRODS Zone. `https://github.com/irods-contrib/metalnx-web` [Accessed 3 July 2022]

[10] Nginx `https://nginx.org/en/` [Accessed 1 July 2022]

[11] Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., Notredame, C.. Nextflow enables reproducible computational workflows. Nature biotechnology (2017)

[12] REST is an acronym for REpresentational State Transfer and an architectural style for distributed hypermedia systems `https://restfulapi.net/` [Accessed 2 July 2022]

[13] Rocky Linux is an open-source enterprise operating system designed to be 100% bug-for-bug compatible with Red Hat Enterprise Linux® `https://rockylinux.org/` [Accessed 2 July 2022]

[14] OpenZFS is an open-source storage platform. `https://openzfs.org/wiki/Main_Page` [Accessed 13 July 2022]

[15] DICOM® — Digital Imaging and Communications in Medicine — is the international standard for medical images and related information. `https://www.dicomstandard.org` [Accessed 1 July 2022]

[16] Wiggins,R., ,Block, K. T. Yarra Framework – Open-Source Toolkit for Clinical-Translational MRI Research. [online] `https://s3.amazonaws.com/download.yarraframework.com/doc/ISMRM19_Whitepaper.pdf` [Accessed 30 June 2022]

[17] IBM Spectrum Scale `https://www.ibm.com/products/spectrum-scale` [Accessed 2 July 2022]

[18] Compose is a tool for defining and running multi-container Docker applications. https://docs.docker.com/compose [Accessed 2 July 2022]

[19] DevOps platform build around git https://about.gitlab.com/ [Accessed 2 July 2022]

[20] YAML Ain't Markup Language™ https://yaml.org/ [Accessed 2 July 2022]

[21] Flask is a lightweight WSGI web application framework. https://flask.palletsprojects.com/en/2.1.x/ [Accessed 2 July 2022]

[22] The Transport Layer Security (TLS) Protocol Version 1.3 https://datatracker.ietf.org/doc/html/rfc8446 [Accessed 2 July 2022]

[23] Rclone syncs your files to cloud storage https://rclone.org/ [Accessed 2 July 2022]

[24] Python iRODS Client (PRC) https://github.com/irods/python-irodsclient [Accessed July 1 2022]

[25] WebDAV Protocol (rfc4918). https://datatracker.ietf.org/doc/html/rfc4918

[26] Marcus, D. S., Olsen, T. R., Ramaratnam, M., Buckner, R. L. The Extensible Neuroimaging Archive Toolkit: an informatics platform for managing, exploring, and sharing neuroimaging data. Neuroinformatics (2007)

[27] Das, S., Zijdenbos, A. P., Harlap, J., Vins, D., Evans, A. C. LORIS: a web-based data management system for multi-center studies. Frontiers in neuroinformatics (2012)

[28] Neuroimaging Informatics Technology Initiative https://nifti.nimh.nih.gov/ [Accessed 13 July 2022]

Programmable authentication workflows in iRODS

Stefan Wolfsheimer
SURF
Utrecht, The Netherlands
stefan.wolfsheimer@surf.nl

Claudio Cacciari SURF
Utrecht, The Netherlands
claudio.cacciari@surf.nl

Harry Kodden SURF
Utrecht, The Netherlands
harry.kodden@surf.nl

ABSTRACT

iRODS (Integrated Rule-Oriented Data System) [1] supports various authentication methods such as native authentication (username and password), GSI, Kerberos, and OpenID. New authentication methods are implemented as shared libraries that need to be installed on client and server sides. Client libraries such as python-irodsclient may need to be patched to support any new authentication protocol.

A universal implementation that supports all authentication flows is clearly favored over managing combinations of client and server libraries and flows. The PAM (Pluggable Authentication Module) [2] mechanism is a way to implement and customize authentication flows on the server without needing to adjust the software that uses this mechanism. Existing PAM libraries may be combined to implement flows featuring branches, multiple-factor authentication, and much more. The PAM mechanism is already supported by iRODS but the current version of the plugin is restricted to the standard flow only (username and password). We have implemented an authentication plugin for iRODS 4.3.0 "`pam_interactive`" that enables the flexibility of fully-fledged PAM authentication flows beyond the standard case.

SURF, the Dutch cooperative association of educational and research institutions, will use that implementation to offer new features to iRODS users. Two scenarios are especially relevant: the support of the SURF Access Management Provider (SRAM), which allows multiple Identity Providers to authenticate a user with iRODS, and the support of Multi-Factor Authentication (MFA) directly at iRODS level, which is often required for sensitive data management.

Keywords

PAM stack, authentication, plugin, OIDC.

INTRODUCTION

Linux-PAM [2] is a mechanism that aims at standardizing user authentication workflows. The mechanism is flexible such that it is possible to support a number of different authentication methods and combinations of them. PAM supports four management groups: account management, authentication, password management, and session management. The scope of this paper and the implemented iRODS plugin is the authentication flow only. The present paper is a follow-up of the work described in a paper presented at UGM 2019 [6], where a similar approach was adopted. In the previous implementation, the complexity of the flow was encapsulated by an additional web component required in front of the iRODS Catalog Provider which increased the overhead and limits the flexibility.

System administrators can mix and match from a variety of *PAM-modules* to implement authentication flows of arbitrary complexity [3]. PAM-modules are layered on a stack which is processed from top to bottom. Finally, a status is returned indicating the success or failure of the authentication flow. Each module itself returns a status code. A control value, which is assigned to each layer, indicates a criterion of how status codes are to be handled. For

iRODS UGM 2022 July 5-8, 2022, Leuven, Belgium

example, if a module with control value `sufficient` returns status code `success`, the stack terminates with success (state "authenticated"). While a failed module assigned `required` causes the whole flow to fail.

PAM-Modules are shared libraries that bridge the communication between directory services, user databases, flat files, etc. with the PAM framework [4].

In order to enable applications with PAM, application developers need to implement the user-facing parts of the authentication flow (e.g., retrieving login information from the user) and delegate the flow control to the PAM library [5]. This approach is described below in the PAM flows sections. In the section Usage, we discuss a few configuration patterns and examples using the flexible `pam_python` module.

IMPLEMENTATION

PAM flows

The authentication procedure of a PAM-enabled application is controlled by the PAM library. The process can be seen as a state machine defined by the PAM configuration. The user interaction is realized by callback functions that are passed from the application to the PAM library. The function is called on each transition that requires user interaction (e.g. querying users' credentials, printing a message on the screen, etc.). The PAM architecture with each component is shown in Figure 1.

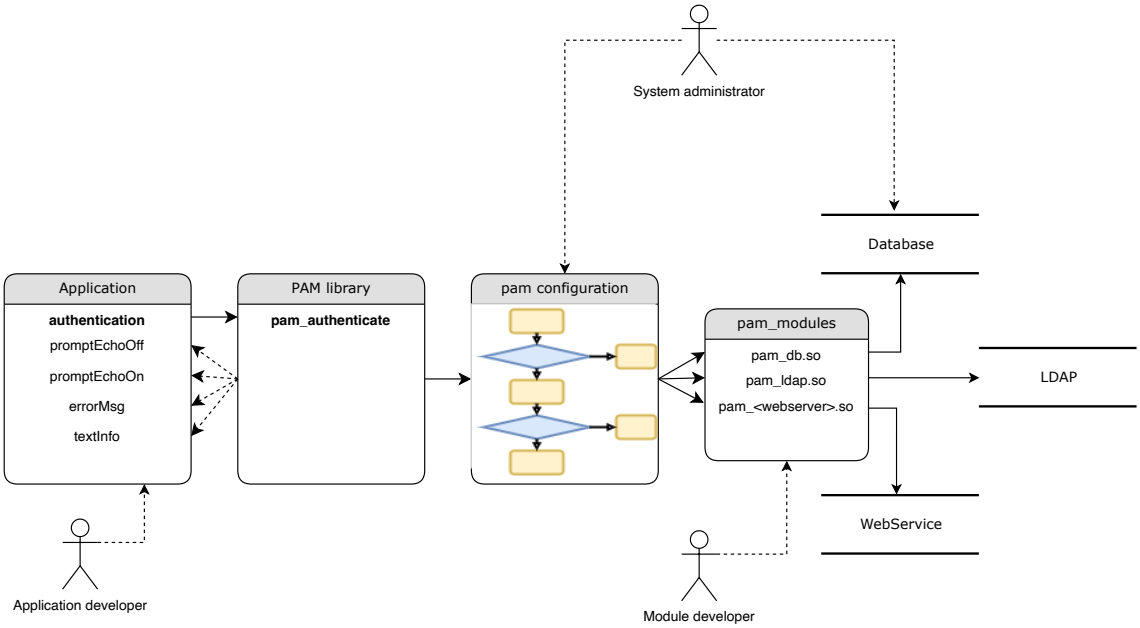

Figure 1. PAM configuration: PAM enabled application, PAM library, PAM stack configuration, PAM modules, and services

PAM flows over the network

Enabling applications with PAM is straightforward when all components, the PAM library, the PAM configuration, and the application are installed on the same host. The situation is more complicated for client-server systems such as iRODS. In this case, the PAM library and the configuration are installed on the server, while the user interaction is realized on the client-side. This implies that the callback functions invoked by the PAM library need to wait for user

input on another host during their lifetimes. On the other hand, the iRODS API is implemented as a request-response model, where the client drives the communication between the components by requesting resources from the server. A callback to the client from the server is not directly supported in such protocols.

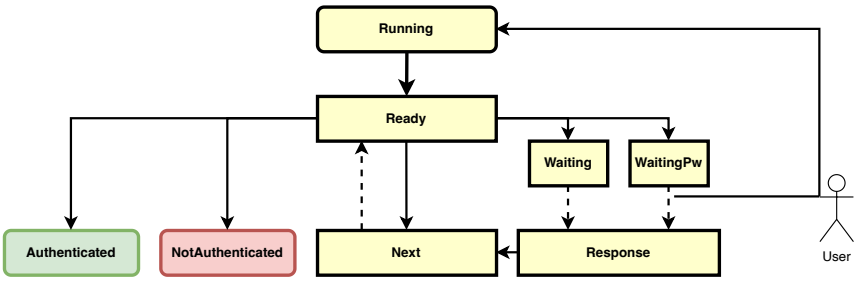

Figure 2. State diagram of the PAM flow

To overcome this limitation we have designed the PAM workflow as a state machine (see Figure 2). The conversation is triggered by a user wanting to login to iRODS (e.g. using `iinit`). The initial state is **Running** which is immendiatly turned to **Ready**. A transition from **Ready** to one of the states **Waiting**, **WaitingPw**, **Next**, **Authenticated** or **NotAuthenticated** is driven by the PAM configuration. As suggested by the names, the states **Authenticated** and **NotAuthenticated** refer to the final states of successful and unsuccessful authentication, respectively. The transition from **Ready** to next **Next** is silent or accompanied by a message that is printed on the screen of the client. After the transition from **Ready** to **Waiting** or **WaitingPw** a message is printed and the user is expected to type a response. The transition from **Waiting** and **WaitingPw** to **Response** is triggered by the client after the user has provided the response. The transition from **Next** to **Ready** is triggered by the client to indicate the readiness for another iteration. The transitions triggered by the client are indicated as dashed lines. Since those transitions depend on user input, the callback functions cannot simply return a value back to the PAM library. Instead, instances of the functions remain idle waiting for a response from the client. Technically, this behavior is implemented by a condition variable that is active during the lifetime of the callback function.

The sequence diagram in Figure 3 illustrates a simple PAM conversation over the network. There are four components involved:

- The iRODS client (icommands)
- The iRODS server
- The `pam_interactive` plugin
- The PAM library.

The blue boxes indicate the lifetimes of the callbacks. Notice that the transition from **Waiting** to **Response** will wake up a condition variable.

USAGE

State persistency

After the user has successfully authenticated using the PAM stack, a temporary password is generated which is valid for one hour by default. This password is used by `icommands` to authenticate against the server. After the expiration period of the password, the PAM authentication is again executed whenever a user invokes a `icommands`. The responses of the last conversation are locally cached and repeated. Below, we describe alternative flows.

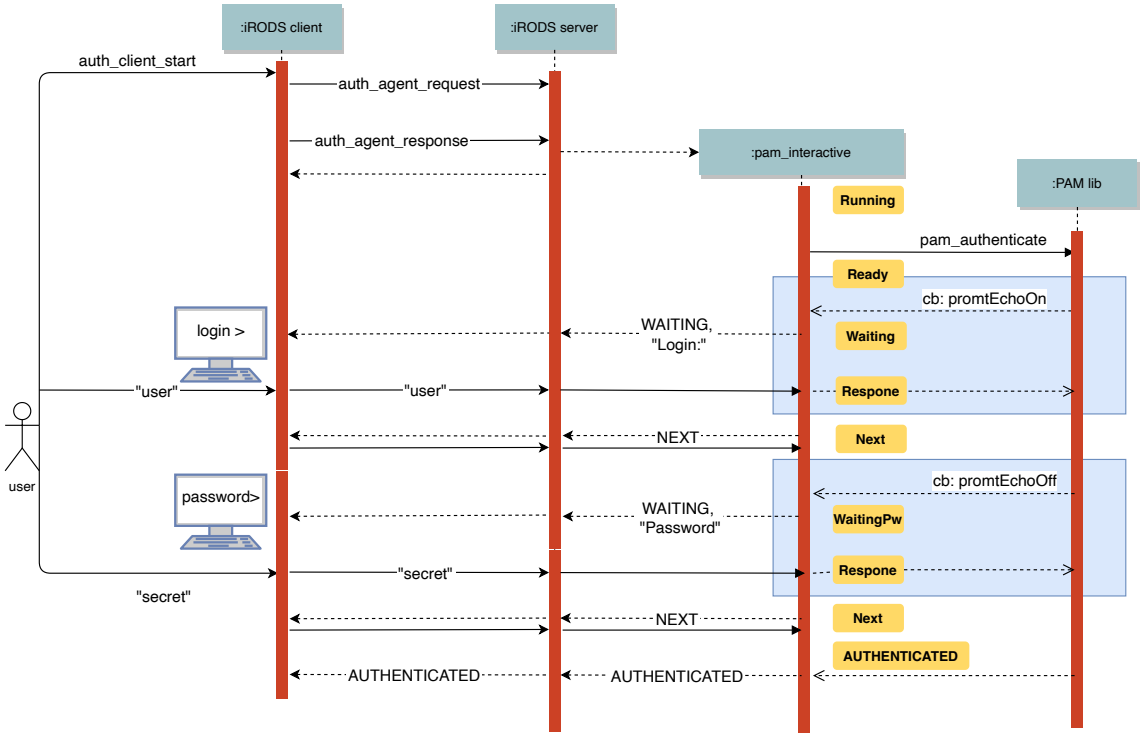

Figure 3. State diagram of the PAM flow

The expiration time of the password can be overridden by the user via the time-to-live option (`-ttl`, in hours) of `iinit`. The iRODS server administrator can set the range of valid values for the TTL value in the server configuration file (`/etc/irods/server_config.json`):

```
"plugin_configuration": {
  "authentication": {
    "pam_interactive" : {
      "password_min_time": 3600,
      "password_max_time": 7200
    }
  }
}
```

Notice that in the current implementation of iRODS, the smallest granularity of the TTL is in hours. The values in the server configuration are given in seconds to support future versions with smaller granularity.

Prototyping with `pam_python`

In this section, we discuss the ability of the `pam_interactive` plugin from the perspective of a PAM module developer and a PAM system administrator. In order to keep the discussion illustrative and generic, we use the `pam_python` [7] module. In contrast to many other modules, this module does not rely on specific backends or user databases. `pam_python` is convenient for

- implementing prototypes of PAM modules for novel backends
- illustrating PAM flows and
- implementing regression tests

However, the pace of the development is relatively slow. Thus, there is no guarantee that the software will be supported in the future.

Consider, for example, the following PAM stack (`/etc/pam.d/irods`), which uses the `pam_python.so` module with the *required* control variable:

```
auth required  pam_python.so /etc/pam.d/simple.py
```

The user will be successfully authenticated when the `pam_sm_authenticate` function defined in the python module returns `PAM_SUCCESS`. The following implementation mimics the authentication against a simple user database.

```
USERS_DB={
    'ayub': 'pw',
    'mara': 'ACtoRPHI',
    'noah': 'NgPOWArs'
}
def pam_sm_authenticate(pamh, flags, argv):
    msg = pamh.conversation(pamh.Message(pamh.PAM_PROMPT_ECHO_ON, "login:"))
    pwd_msg = pamh.conversation(pamh.Message(pamh.PAM_PROMPT_ECHO_OFF, "password:"))
    login = msg.resp
    password = pwd_msg.resp
    if login in USERS_DB and password == USERS_DB[login]:
        return pamh.PAM_SUCCESS
    return pamh.PAM_AUTH_ERR
```

In a real-life application, the dictionary would be replaced by a user directory, such as LDAP or another database.

Next, we illustrate how one would enable a second factor required to successfully log in. This can be realized by adding a second required layer on the pam stack

```
auth required  pam_python.so /etc/pam.d/simple.py
auth required  pam_python.so /etc/pam.d/2fa.py
```

Now, both layers are required. The user needs to enter their regular credentials before being asked for a one-time PIN generated by a key generator. The module of the second layer can be implemented as follows:

```
def pam_sm_authenticate(pamh, flags, argv):
    msg = pamh.conversation(pamh.Message(pamh.PAM_PROMPT_ECHO_ON, "pin:"))
    pin = msg.resp
    if pin == "1234":
        return pamh.PAM_SUCCESS
    else:
        return pamh.PAM_AUTH_ERR
```

According to the implementation, the correct PIN is the fixed value 1234. In real-life applications, this stub should obviously be replaced by a real validation. Notice that the logic is entirely driven by the backend and controlled by the systems administrator. In contrast to other iRODS authentication methods, new policies (e.g. enabling a second factor) can be rolled out without changing the local client configurations and actively supporting users.

Persistent client information

The user responses to the conversation are stored locally in a JSON document next to the scrambled password. They can be reused as default values when the user logs in again. However, in some cases, this behavior is not desirable. For example, it does not make sense to store and reuse the values of one-time passwords for second-factor authentications. On the other hand, some workflows may require storing and retrieving data without user interaction. In order to address these use cases, we have extended the standard protocol. The server can either send simple messages (as in the example above), or JSON payloads describing a set of operations. The message has the form of a JSON object with the following (optional) keys:

- `prompt`: a message to be printed on the screen,
- `default_path`: the JSON path to the default value,
- `patch`: a list of patches to be applied to the locally stored JSON document (The patches are implemented according to the specification RFC6902 [8, 9]. and
- `retrieve`: a JSON path to the locally stored JSON node to be sent back to the server

Example 1: `prompt` *and* `patch`

The following example, a modification of the `2fa.py` script from the previous section, illustrates the use of the `prompt` and `patch` fields. The PIN is returned but not stored locally because of the absence of the `patch` field. After successful authentication, a token is sent to the client and stored locally:

```
import json
import uuid

def pam_sm_authenticate(pamh, flags, argv):
    # just prompt, don't save the pin locally
    pin = {"prompt": "pin:"}
    msg = pamh.conversation(pamh.Message(pamh.PAM_PROMPT_ECHO_ON, json.dumps(pin)))
    pin = msg.resp

    if pin == "1234":
        token = str(uuid.uuid4())
        # save token on client, no prompt
        patch = {"patch": [{"op": "add",
                            "path": "/token",
                            "value": token}]}
        msg = pamh.conversation(pamh.Message(pamh.PAM_TEXT_INFO,
                                             json.dumps(patch)))

        return pamh.PAM_SUCCESS
    else:
        return pamh.PAM_AUTH_ERR
```

Example 2: `default_path`

Notice that it is also possible to query a message from the user and save it locally under a given path. For example, the following message queries a pin and stores it under the path `/pin` in the local JSON document. The next time the PIN is queried, the default value is take from the JSON document under the path `/pin`.

```
# prompt and save the pin locally
pin = {"prompt": "enter pin:",
       "default_path": "/pin",
       "patch": [{"op": "add",
                  "path": "/pin"}]}
```

Example 3: `retrieve`

Suppose we have stored the token under the path `/token`. Then the data can be retrieved with the payload `{"retrieve": "/token"}`. The following `pam_python` module requests the locally stored token from the client and returns the result

```
import json

def pam_sm_authenticate(pamh, flags, argv):
    payload = {"retrieve": "/token"}
    msg = pamh.conversation(pamh.Message(pamh.PAM_PROMPT_ECHO_ON, json.dumps(payload)))
    token = msg.resp
    payload_resp = {"prompt": "token={}".format(token)}
    pamh.conversation(pamh.Message(pamh.PAM_TEXT_INFO, json.dumps(payload_resp)))
    return pamh.PAM_SUCCESS
```

This example concludes the fundamental operations that can be applied to the local JSON document. PAM module developers and/or administrators can make use of them to implement complex flows.

In the following section, we turn to an application that has motivated the need of a flexible authentication flow.

OpenID Connect

Our need to support multi-factor authentication and federated identity management led us to choose the OpenID Connect protocol, which adds, on top of the OAuth2 authorization protocol, an authentication token that includes some basic user profile information. The main use case for the OIDC protocol is the authentication of a web application against an Identity Provider (IdP). But our users want to log in iRODS via the command line. In order to do that we have adapted one of the OIDC `flow`, the Authorization Code Flow (defined in OAuth 2.0 RFC 6749, section 4.1, [10]), as shown in Figure 4.

Clearly, it would not have been possible to implement that flow without the new authentication plugin. In fact, the user is presented with a challenge (the log in URL) and the server waits for a response. Beyond the Authorization Code Flow, we have added steps 13-15 to map the identity of the user to an iRODS account, using one of the available attributes, like, for example, the email address. In terms of the PAM python module, the function `pam_sm_authenticate` could be written in the way given in the APPENDIX.

When the token is not valid, the authentication simply fails, but it would be possible to use a refresh token to automatically renew the expired one.

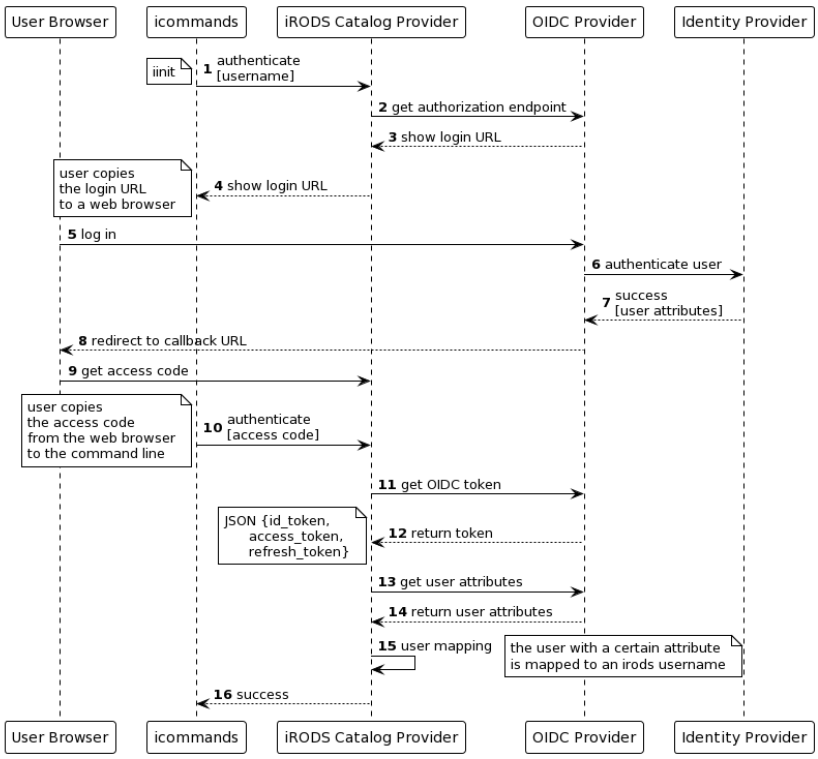

Figure 4. OIDC Authorization Code Flow: adapted for the interaction via command line

CONCLUSION

`pam_interactive` provides a backend-driven programmable and flexible authentication mechanism for iRODS. By simplified examples, we have illustrated solution patterns for programing multifactor authentication flows and token management.

The plugin supports a large range of authentication methods and customized flows because the conversation is not restricted to simple login-password credentials. The capability of storing information locally can be extended in future releases. One can think of JSON Web Token as a common technique to implement single signed-on in web-based applications. Adopting this technology to iRODS would improve the interoperability with other systems.

ACKNOWLEDGMENTS

We thank the members of the iRODS Authentication Working Group for the organization of regular meetings and open discussions. We also thank our colleague Maithili Kalamkar-Stam for critical proofreading on very short notice.

APPENDIX

OIDC authentication example code

```
def pam_sm_authenticate(pamh, flags, argv):
    try:
        user = pamh.get_user(None)
    except pamh.exception, e:
```

```
        user = None
        pamh.conversation(pamh.Message(pamh.PAM_ERROR_MSG, str(e.pam_result)))
    if user == None:
        return pamh.PAM_USER_UNKNOWN

    no_token = True
    # Check if the user has a token
    payload = json.dumps({ "retrieve": "/oauth2_access_token"})

    token_msg = pamh.conversation(pamh.Message(pamh.PAM_PROMPT_ECHO_ON, payload))
    if token_msg is not None and len(token_msg.resp.strip()) > 0:

        # Validate the token
        result = validate_token(user, token_msg.resp, INTROSPECT_URL, OIDC_USER_MAP)
        pamh.conversation(pamh.Message(pamh.PAM_TEXT_INFO, "Authentication: {}".format(result)))
        if (result.strip() == "Success"):
            no_token = False
            return pamh.PAM_SUCCESS
        else:
            pamh.conversation(pamh.Message(pamh.PAM_TEXT_INFO, "Invalid token: {}".format(result)))

    if no_token:

        state = uuid.uuid4()
        # Get the login URL
        params = {"response_type": "code",
                  "client_id": CLIENT_ID,
                  "redirect_uri": REDIRECT_URI,
                  "scope": "openid offline_access email eduperson_principal_name"
                  "state": state}
        loginURL = AUTHORIZATION_EP + '?' + urlencode(params)

        # Copy it to the browser
        payload = {"prompt": "Copy the following URL to your web browser:"}
        pamh.conversation(pamh.Message(pamh.PAM_TEXT_INFO, json.dumps(payload)))

        payload = {"prompt": loginURL}
        pamh.conversation(pamh.Message(pamh.PAM_TEXT_INFO, json.dumps(payload)))

        # Copy back the callback string
        # request without saving answer locally
        payload = {"prompt": "Copy the callback string from your web browser here:"}
        callback_msg = pamh.conversation(pamh.Message(pamh.PAM_PROMPT_ECHO_ON, json.dumps(payload)))

        # Get a token
        token = oidc_get_token(callback_msg.resp, REDIRECT_URI, TOKEN_EP, BASE64CREDS)

        # Validate the token
        result = validate_token(user, token, INTROSPECT_URL, OIDC_USER_MAP)
        pamh.conversation(pamh.Message(pamh.PAM_TEXT_INFO, "Authentication: {}".format(result)))
```

```
if (result.strip() == "Success"):
    # save a simple cookie on the client
    # display an optional message
    payload = {"prompt": "the cookie 'oauth2_access_token' has been updated",
               "patch": [{"op": "add",
                          "path": "/oauth2_access_token",
                          "value": token}]}
    pamh.conversation(pamh.Message(pamh.PAM_TEXT_INFO, json.dumps(payload)))
    return pamh.PAM_SUCCESS
else:
    return pamh.PAM_AUTH_ERR
```

REFERENCES

[1] Integrated Rule-Oriented Data System (iRODS) https://irods.org/

[2] Linux-PAM. http://www.linux-pam.org/ Visited last on 06.24.2022.

[3] Morgan, A.G., Kukuk, T.: The Linux-PAM System Administrators' Guide, Version 1.1.2, 31. (2010) http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html

[4] Morgan, A.G., Kukuk, T.: The Linux-PAM Module Writers' Guide http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_MWG.html Version 1.1.2, 31. (2010)

[5] Morgan, A.G., Kukuk, T.: The Linux-PAM Application Developers' Guide http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_ADG.html Version 1.1.2, 31. (2010)

[6] Cacciari, C., Muscianisi G., Carpené, M., D'Antonio, M. and Fiameni G. An authentication solution for iRODS based on the OpenID Connect protocol, iRODS User Group Meeting Proceedings (2019)

[7] Stuart, R.: pam_python: Write PAM modules in Python http://pam-python.sourceforge.net/ Version 1.0.8-1. (2020)

[8] Lohmann, N. et. al.: JSON for Modern C++ https://json.nlohmann.me/ Version 3.7.3 (2022)

[9] JavaScript Object Notation (JSON) Patch, RFC 6902 https://www.rfc-editor.org/info/rfc6902 (2013)

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

iRODS as a data backend for the LEXIS workflow orchestration platform

Mohamad Hayek, Stephan Hachinger
Leibniz Supercomputing Centre
mohamad.hayek@lrz.de

Martin Golasowski, Jan Martinovič
IT4Innovations, VŠB – Technical University of Ostrava
martin.golasowski@vsb.cz

ABSTRACT

In this contribution, we present the current status of the iRODS federation used as a part of the Distributed Data Infrastructure in the LEXIS platform. This backend has been built in the European Horizon 2020 project "Large-Scale EXecution for Industry and Society" (LEXIS, H 2020 GA 825532) and was verified against a wide range of use cases from industry and society. We report on our experience in maintaining and extending the iRODS federation with a focus on the current challenges. Afterwards, we lay out our experience with enabling OpenID authentication for Keycloak integration and methods used to ensure synchronized fine-grained access control between iRODS and Keycloak. We then discuss our strategy to enable data staging between iRODS and various Cloud and HPC systems within the LEXIS platform via a REST API. Furthermore, we address the periodic testing of different aspects of the federation and the alerting system put in place to react to any irregularities in the tests. Finally, we present the results of speed tests done between the different nodes of the federation and we give an outlook on future work that might be interesting for the iRODS community.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

Managing high-throughput sequencing and other -omics data with RODEOS and rodeos-ingest

Clemens Messerschmidt, Marten Jäger, Mathias Kuhring, Dieter Beule and Manuel Holtgrewe
Berlin Institute of Health at Charité – Universitätsmedizin Berlin, Core Unit Bioinformatics
clemens.messerschmidt@bih-charite.de

ABSTRACT

Omics data are generated by high-throughput biochemical assays that simultaneously quantify and/or characterize molecules of the same type in biological samples. In biomedical research, omics data acquisition is often performed in specialized technology units referred to as core facilities. Using complex (and often expensive) devices such as sequencers and mass-spectrometers, these units produce a wealth of different high-volume datasets that need to be organized, stored, quality checked, pre-processed or transformed and eventually delivered to clients, archived or deleted.

To streamline and automate the data management and handling processes while supporting the diversity of projects and clients present in the research organization, we introduce RODEOS (Raw Omics Data accEss and Organization System). The system is based on iRODS and rodeos-ingest, a custom event handler that extends the iRODS automated ingest framework. The automatic ingest enables an easier control of data through its life cycle from generation to delivery and deletion by unlocking iRODS' advantages like data discovery, connecting workflows based on the rule engine, as well as secure collaboration.

To enrich metadata beyond simple file attributes, rodeos-ingest extracts additional technology-specific parameters from files generated by the omics units' devices when processing samples. We provide examples for widely used Illumina sequencers and demonstrate how the extracted metadata could be used to support demultiplexing and data QC workflows. Furthermore we integrated Metalnx as an additional user interface to RODEOS. This allows the wet-lab staff to easily add further iRODS metadata, e.g. for choosing data delivery paths and also empowers clients to view their data and track progress. This reduces the complexity of operations for everyone involved, especially when used in cross-institutional settings if coupled to (possibly multiple) Active Directory services for user authentication.

RODEOS is in active use at the integrated sequencing unit of the Max Delbrück Center for Molecular Medicine (basic research) and the Berlin Institute of Health at Charité (university hospital). Additional rodeos-ingest modules are planned to support more facilities and technologies, e.g. mass spectronomy for metabolomics or proteomics.

Rodeos-ingest is MIT-licensed and available at https://github.com/bihealth/rodeos-ingest

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

Can Blockchain Technology Play a Role in iRODS?

Arcot Rajasekar
University of North Carolina at Chapel Hill
sekar@renci.org

ABSTRACT

Blockchain technology has matured and is increasingly applied in a diversity of applications. Some of its intrinsic properties, such as secure database, distributed ledger, provenance tracking, integrity checking and trust worthiness, consensus maintenance and data sharing, and crypto-security, are values that are also central to iRODS. One view of blockchain is a Distributed, Immutable Ledger (DIL) that facilitates recording information about assets. One view that is interesting to the iRODS community is that of a Blockchain Storage (BCS) can be used to save data files (sharded as blocks) in a decentralized network as opposed to storing files in a centralized cloud storage. This approach provides all the advantages of the blockchain technology but uses enormous amount of storage. An alternate is to store just the hash of the data (but store data elsewhere) in the blockchain. One can also attach minimal useable metadata (MUM) to the hash and provide access to that in a private or public network. Blockchains also support rule-based actions, called smart contracts. Smart contracts are digital 'contracts' stored on the blockchain that are automatically executed when predetermined terms and conditions are met. One can notice the similarities to the iRODS rule system. Blockchains also support the concepts of private, public, permissioned exchanges of information. With such close functional similarity, taking advantage of synergetic properties will enhance the applications of iRODS in a diversity of applications including supply chain, health informatics, government, retail, etc. where transactional properties with large datasets dominate. In this paper, we look at various ways one can enhance iRODS with blockchain technology.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

Data Management Environment at the National Cancer Institute

Sunita Menon
Cancer Data Science Initiatives, Frederick National Laboratory for Cancer Research, Frederick, MD
sunita.menon@nih.gov

Eran Rosenberg
Cancer Data Science Initiatives, Frederick National Laboratory for Cancer Research, Frederick, MD
eran.rosenberg@nih.gov

Yuri Dinh
Cancer Data Science Initiatives, Frederick National Laboratory for Cancer Research, Frederick, MD
yuri.dinh@nih.gov

Dr. Zhengwu Lu
Bioinformatics and Computational Science, Frederick National Laboratory for Cancer Research, Frederick, MD
zhengwu.lu@nih.gov

Prasad Konka
Cancer Data Science Initiatives, Frederick National Laboratory for Cancer Research, Frederick, MD
prasad.konka@nih.gov

Dr. George Zaki
Cancer Data Science Initiatives, Frederick National Laboratory for Cancer Research, Frederick, MD
george.zaki@nih.gov

Udit Sehgal
Cancer Data Science Initiatives, Frederick National Laboratory for Cancer Research, Frederick, MD
udit.sehgal@nih.gov

Sarada Chintala
Cancer Data Science Initiatives, Frederick National Laboratory for Cancer Research, Frederick, MD
sarada.chintala@nih.gov

Ruth Frost
Bioinformatics and Computational Science, Frederick National Laboratory for Cancer Research, Frederick, MD
ruth.frost@nih.gov

Dr. Eric Stahlberg
Cancer Data Science Initiatives, Frederick National Laboratory for Cancer Research, Frederick, MD
eric.stahlberg@nih.gov

ABSTRACT

An efficient and cost-effective mechanism is required to store and manage the large heterogeneous datasets generated by high throughput technologies such as Next Generation Sequencing, Cryo-Electron Microscopy, and High Content Imaging. High-performance tier 1 storage is expensive, and the affordable tier 2 devices used standalone do not lend themselves well to discovering and disseminating datasets. The Data Management Environment (DME), a data management platform for storing and managing high-value scientific datasets, was developed at the National Cancer Institute to close this gap. DME addresses the long-term data management needs of research labs and cores at NCI per the FAIR [1] (Findable, Accessible, Interoperable, and Reusable) guiding principles for data management. It supports S3 compatible object store, as well as file system storage. DME uses iRODS [2] as the metadata management layer enabling virtualization of backend storage, replacement of storage providers with zero impact on users, and transparent migration of data across providers. The granular permissions scheme provided by iRODS coupled with DME's authentication and authorization mechanism enables researchers to share data with collaborators securely. This paper will provide an overview of the capabilities and architecture of the Data Management Environment and discuss how DME has leveraged iRODS to deliver enhanced data management and storage management capabilities.

Keywords

DME, iRODS, data management, scientific datasets, metadata, virtualization, data migration, object store

INTRODUCTION

Research labs and cores utilizing high throughput instruments regularly generate data at terabyte and petabyte-scale. The data collected from the instrument is moved to a local drive or network-attached storage, from where it makes its way to one or more computational servers and analysis workstations. Copies of the raw and analysis data are made to secure them, resulting in the generation of multiple redundant copies along the processing path. Often, researchers share the data with one or more collaborators, who make more copies along new processing paths. New files are often added to these directories, or the original files are reorganized to align with the analysis performed. After a while, the provenance information of the initial dataset is no longer available, making storage space recovery extremely challenging. Staff turnover only adds to the problem. The data stays forever in the expensive tier 1 storage devices. Some of it is moved later to the infrequently accessed tier 2 devices like tape storage, which is much cheaper but needs heavy investment in time and effort to retrieve the data. Since this data is not annotated, further effort is required to search and locate what is needed. These limitations also prevent the sharing and processing of data in integration and analysis platforms for further study and research.

To solve this problem, we need to store the data in cost-effective, reliable storage (Figure 1) from where it is directly and easily accessible for reuse. The data needs to be secured from unauthorized access while at the same time being shareable with collaborators whenever required. It should be annotated with the appropriate provenance and domain metadata, easily searchable and downloadable. It should be migratable to other storage devices when the lifecycle of the current device has ended. The migration should be transparent to users so that there is no burden on learning to use a new interface or technology to retrieve the data and no impact on the bioinformatics pipelines and data analysis platforms that access it programmatically. It should be easy to tier the data to cold storage or dispose of it when the predetermined lifecycle of the data has ended.

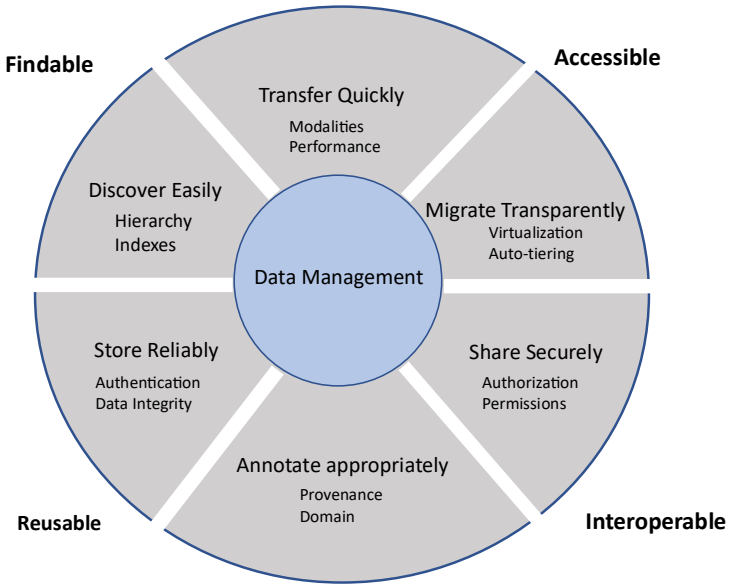

Figure 1. Data Management System Requirements

The Data Management Environment (DME) was developed to address these needs. DME is a metadata-based data management platform that provides secure, virtualized storage for high-value scientific datasets generated at NCI. Its reliable storage mechanism and the ease of accessing and sharing large datasets eliminate the need for users to maintain copies of datasets in their local or network-attached storage.

SYSTEM OVERVIEW

DME was designed to archive and share large, heterogeneous datasets. DME archives data to S3-based object stores that presently include Cleversafe [3], Cloudian [4], and Amazon S3 Glacier Deep Archive [5]. Support is also available for archiving to network file systems.

Data in DME is associated with provenance and domain metadata to enable the targeted discovery of datasets. DME performs all data management functions are performed through iRODS. These functions include the management of collections, data objects, user accounts, user groups, metadata, and permissions. Using iRODS for data management functions has enabled DME to perform storage virtualization, data migration, and data tiering transparently. These critical capabilities have provided users with a seamless data management experience, enabled secure data sharing, and significantly eased the IT functions required to manage large data volumes. It has also enabled DME to be domain agnostic, facilitating its broad adoption across NCI.

Interfaces

DME provides the following interfaces (Figure 2) for users to interact with the system:

- The DME web application enables users to easily browse, store, search and download data through an intuitive user interface. Transfer status screens enable detailed tracking of ongoing asynchronous bulk transfers. Users can create 'bookmarks' to enable quick access to desired collections. Other capabilities include user account management, group management, and reporting.
- The command-line utilities (CLU) provide shell commands for programmatic access from bioinformatics pipelines and workflows. CLUs can be used to store, search and download data. User account management and bookmark creation are also supported.
- The representational state transfer (REST) API suite provides fine-grained control of DME services and enables programmatic integration with data analysis platforms and external third-party applications.
- The DME Archival workflow supports users requiring regular bulk uploads. It enables fully automated archival of datasets. The workflow scans the source directories specified by the user at pre-configured intervals to locate the files to be archived. It then extracts the metadata from metadata input files based on the rules configured in a custom user module and uploads the annotations and the corresponding dataset to DME. Supported metadata file formats are JSON, XML, and Excel.

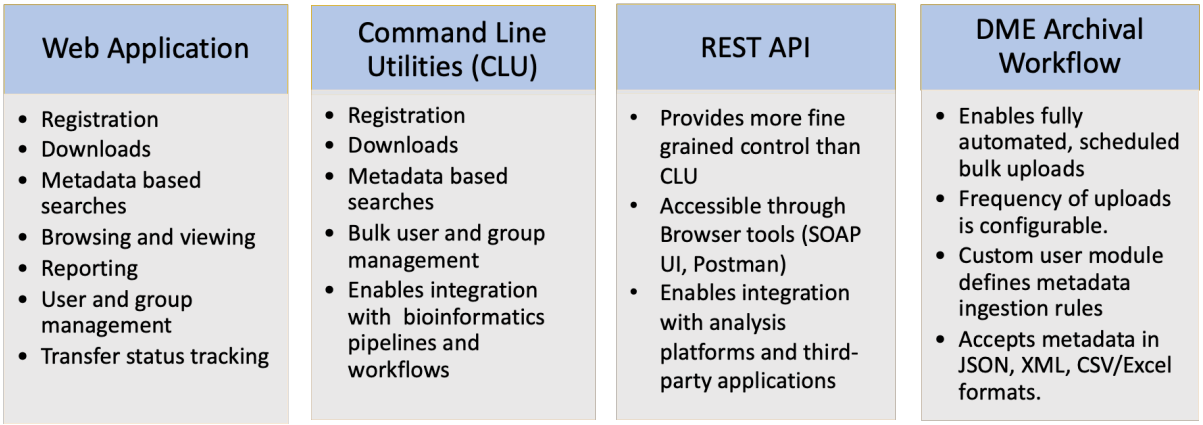

Figure 2. DME Interfaces

Authentication and Authorization

A DME user account is required to obtain access to the system. A user account can be created for a user only if the requested user identifier is present in the NIH Active Directory system. However, a user with an active DME account cannot access files or collections unless explicitly permissioned by the data generator.

To prevent the password and username from being sent for each CLU or REST API call, an access token is issued when the user successfully authenticates with the system using the token generator CLU or the authenticate API. The returned token can then be used for subsequent calls until it expires. The expiration period is configurable for a specific installation.

Transfer Modalities

Data can presently be uploaded from or downloaded to five endpoint types - Amazon S3 [6], Google Cloud [7], Google Drive [8], Globus endpoint [9], and the user's local file system. We continually evaluate and add new transfer modalities to make the system more broadly useable based on user feedback and NCI needs.

DME ARCHITECTURE

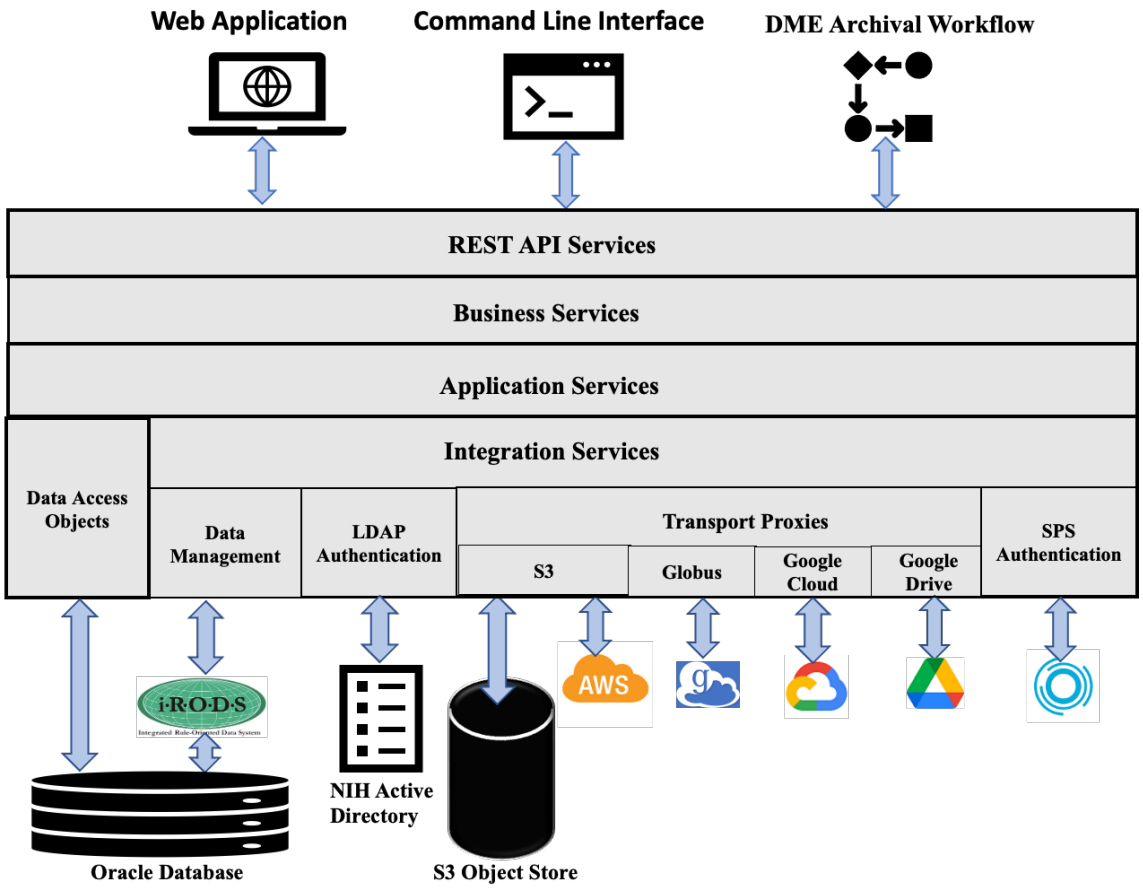

Figure 3. DME Logical Architecture

DME consists of the API server providing the platform core services, the DME web application that provides the graphical user interface, and the command-line interface (CLI) that is fronted by the command line utilities (CLU). The web application and the CLI/CLU communicate with the API server through the DME REST API. The API server includes 'schedulers' that perform various tasks in the background at separately configured intervals.

The DME production infrastructure consists of the following components:

- Tomcat 8 [10] server hosting the DME web application
- 6 API servers running on Apache ServiceMix [11]
- iRODS 4.2.9 server
- Oracle [12] 19c database server hosting the iRODS metadata database
- On-premises Cleversafe and Cloudian vaults
- Amazon S3 and S3 Glacier Deep Archive

All servers run on CentOS 7 machines with the default Java 8 installation. The scheduler is packaged as a separate ServiceMix feature enabling it to be deployed separately from the API server. The schedulers are distributed across the 6 API servers, enabling dynamic, horizontal scaling of services in response to user load.

We implemented the platform core services in a modular, layered architecture (Figure 3) to provide clean interfaces and separation of concerns, making it easy to maintain and scale the system. The services are implemented in Java using the Spring Framework. Each horizontal layer exposes its services through the layer's exported API, thus hiding the details of its implementation.

The REST API layer invokes the APIs exposed by the business services layer, which orchestrates one or more application services to deliver the requested service.

The data access objects (DAOs) connect to the Oracle database to write and read data to and from the DME tables and materialized views that have been set up to support reporting and other business requirements.

The integration services interfaces with external subsystems, enabling easy replacement of these subsystems without impacting the higher layers. It contains the following modules:

- LDAP authenticator authenticates the user credentials with the NIH Active directory.
- The Data Management module communicates with the iRODS server through the Jargon API [13].
- The Data Transfer Proxies manage the transfer of data to and from physical storage and the migration and tiering of data across S3 storage providers. The proxies include modules for interfacing with S3 providers, Globus, Google Cloud, Google Drive, and SPS [14]. This decoupling of data transfer functions enables easy replacement of the backend storage without impacting the data management functions.
- SPS authorization module provides the token received from third-party applications to the NIH authorization web service for verification.

DATA MANAGEMENT

Metadata is associated with both collections and data objects in DME. DME categorizes metadata as system and user metadata. System metadata is automatically captured in DME when a data object is created and cannot be added or modified by the user. It includes the file size, file archive location, checksum, data transfer type, data transfer status, and transfer date. The user metadata is provided by the user and consists of provenance and domain metadata. Provenance metadata is the same for all the user groups in DME and is collected for administrative and maintenance purposes, including determining the end of the data lifecycle. Domain metadata forms the backbone of efficient data discovery and is defined by the user depending on their data management workflow and the granularity of the searches required. The user metadata may be configured as mandatory or optional. The mandatory metadata is supplied during object registration and is validated during that time. The optional metadata can be added anytime during the data lifecycle and is not subject to validation. Most of the provenance metadata is mandatory.

DME provides flexibility to each Division/Office/Center (DOC) to define their data hierarchy (virtual folder organization) and metadata structure (attributes defined for each level in the hierarchy) in DME. These together constitute the metadata model, which consists of three JSON policy files structured as follows:

- Data hierarchy file: Specifies each collection type, whether it is container collection, and the parent of the collection

- Collection metadata validation rules: Specifies the attributes of a collection. The name, a brief description, the parent collection type, and the attribute type (whether mandatory or optional) are provided for each attribute.
- Data object validation rules: Specifies the attributes of a data object. The name, a brief description, the parent collection type, and the attribute type (whether mandatory or optional) are provided for each attribute.

A DOC can have multiple metadata models, one for each sub-group.

User and Group Management

DME has implemented REST APIs and command-line utilities to manage users and groups using IRODs. Users and groups can also be managed through the DME web application.

Group administrators and system administrators can create or delete users. They can retrieve all the users present in a group or all users having a specific role. A new user account can be created in DME only if that user has an active NIH Active Directory account. For creating the new account, only the NIH user identifier for that user is required. DME automatically populates the last name and first name from the NIH LDAP. Email notification is optionally sent to the user when the account is successfully created in DME. While adding a bookmark for a user, the user account is automatically created if it does not exist, and permissions are set on a file or collection for that user.

Group administrators and system administrators can create or delete groups. They can add or remove users to/from a group. They can search for groups and retrieve all the groups to which a user belongs. If metadata containing encrypted PII information is present, DME decrypts it for a user only if that user belongs to the DOC's DME security group.

Roles and Permissions

DME provides the ability to set the permission on all datasets, ensuring that only authorized users of the system can access the data. IRODS fine-grained permissions scheme, coupled with the authentication mechanism implemented by DME, has enabled highly secure data sharing with NIH collaborators.

The iRODS permissions scheme is applied to DME as follows:

- OWN permissions: The data generator uploading data to DME automatically gets OWN permissions for the file or collection. In the case of core facilities uploading data for clients, the data generator or group administrator grants OWN permissions to the data owner or the designee of the data owner.
- WRITE permissions: The data generator or data owner grants WRITE permissions to users who need to modify the metadata associated with the data asset.
- READ permissions: The data generator or the data owner grants READ permissions to researchers and collaborators who need to browse and download the data. Data can be shared with a larger audience by setting permissions for a group rather than individual users.

Permissions can be set on multiple files and collections simultaneously for a user. Additionally, multiple users can be permissioned to a file or collection.

DME uses the iRODS roles to manage the activities a user is permitted to perform. A role can perform all the tasks of the next lower role, in addition to the tasks described below:

- System Administrator (*rodsadmin*): This role is granted only to DME administrators. It enables them to create the metadata model for new onboarding DOCs and monitor the status of the data transfers initiated by users. System administrator privileges are required for data migration and tiering in DME. On request from the DOC, System Administrators may perform user or data management functions.
- Group Administrator (*groupadmin*): This role is granted to the data generators, lab managers, or bioinformatics analysts in a research lab or core. Group administrators perform data archival and user and group management functions for their DOC. They also set permissions on the collections and files for the researchers and collaborators requiring access.

- User (*rodsuser*): This is the default role assigned to researchers and collaborators working on a project. It enables them to view, search and download the files and collections they are permissioned to see. If they are provided OWN permissions to a collection, they can also register new sub-collections or files.

STORAGE VIRTUALIZATION

All users access data in DME through the logical path presented by iRODS. Users view the data through the data hierarchy they have defined. This hierarchy translates to a non-hierarchical prefix-based structure for the S3 bucket, with the logical path mapped to the key of the S3 object.

The physical path is stored as a metadata attribute of the data object, and the physical location and organization of data are transparent to the users. The metadata in DME is decoupled from the storage, enabling easy replacement of storage modalities. The storage provider URL is stored as a database configuration, and switchover of the S3 storage provider only involves changing this configuration.

Since all references to the data are made through the iRODS collections and data objects, changes to the backend storage have no impact on the systems and pipelines that are integrated with DME, enabling significant changes to the storage infrastructure with just a few minutes of system downtime.

DATA MIGRATION AND TIERING

As the data progresses through its lifecycle and gets used less frequently, moving it to a slower, cheaper storage is more cost-effective. Since the metadata is not attached to the data, this transition only involves setting up the lifecycle rules and storage class and invoking the appropriate S3 API to facilitate the transfer. Tiering REST APIs have been added in DME to enable tiering from Cleversafe and Cloudian to Glacier and Glacier Deep Archive. DME performs data retrieval from Glacier in two steps – it is first restored from Glacier to AWS S3 and then downloaded from AWS S3. An email notification is sent to the requesting user when the data has been restored to AWS S3.

Storage systems need to be replaced when they reach end-of-life or end of support, which requires the migration of all data to a new system. Using iRODs to manage the metadata separately has enabled seamless data migration from one storage provider to another. The virtual path presented by the data management layer and the metadata associated with a file or collection remains the same. Only the configuration parameters representing the file's physical location are modified. The Migration REST API enables the transfer of data from and to Cleversafe, Cloudian, or AWS S3. Further archiving to Glacier or Glacier Deep Archive can be performed by setting the appropriate lifecycle policies on the S3 buckets. All calls to retrieve the data after the migration will automatically fetch it from the new storage device, making the migration fully transparent to the user.

CONCLUSION

DME presently hosts over 4 Petabytes of data from 23 labs and cores across NCI, and the number is multiplying rapidly. The infrastructure has been scaled significantly over the past year to accommodate the growing demand. As the data continues to grow and DME gets leveraged by more and more stakeholders for data sharing and archival, the focus is on implementing innovative approaches to improve the performance further while adding new capabilities and enabling integrations with external platforms.

ACKNOWLEDGEMENTS

The authors would like to thank their colleagues at the Enterprise Information Technology group of the Frederick National Laboratories for Cancer Research for setting up the storage and infrastructure for this platform.

We are grateful to the various research labs and core facilities across NCI who have onboarded to DME and provided valuable feedback to improve the system.

This work has been funded in whole or in part with Federal funds from the National Cancer Institute, National Institutes of Health, under Contract No. 75N91019D00024. The content of this publication does not necessarily reflect the views or policies of the Department of Health and Human Services, nor does the mention of trade names, commercial products, or organizations imply endorsement by the U.S government.

REFERENCES

[1] FAIR Principles, **https://www.go-fair.org/fair-principles**
[2] iRODS, **https://irods.org**
[3] IBM Cloud Object Storage, **https://www.ibm.com/cloud/object-storage**
[4] Cloudian, **https://cloudian.com/**
[5] Amazon S3 Glacier Storage Classes, **https://aws.amazon.com/s3/storage-classes/glacier/**
[6] Amazon S3, **https://aws.amazon.com/s3/**
[7] Google Cloud Storage, **https://cloud.google.com/storage**
[8] Google Drive, **https://www.google.com/drive/**
[9] Globus, **https://www.globus.org/**
[10] Apache Tomcat, https://tomcat.apache.org/
[11] Apache ServiceMix, **https://servicemix.apache.org/**
[12] Oracle Database 19c, **https://docs.oracle.com/en/database/oracle/oracle-database/19/index.html**
[13] Jargon core libraries, **https://github.com/DICE-UNC/jargon**
[14] Lightweight Directory Access Protocol, **https://ldap.com/**
[15] One Identity – Safeguard for Privileged Sessions, **https://www.oneidentity.com/products/one-identity-safeguard-for-privileged-sessions/**

iRODS as an Object Store for the Galaxy Platform

Kaivan Kamali, Nate Coraor, John Chilton, Anton Nekrutenko
Penn State University
kxk302@gmail.com

Marius van den Beek
Galaxy Project

ABSTRACT

Galaxy platform (https://galaxyproject.org) is a computational workbench used by thousands of scientists across the world to analyze large heterogeneous datasets (e.g., biomedical, genomics, and climate). Galaxy supports data imports from the user's computer, by URL, and directly from many online resources, and supports a range of widely used data formats, and translation between those formats. The Galaxy sites provide substantial CPU and disk space, making it possible to analyze large datasets -- On usegalaxy.org, the median size of the datasets created by all users per day is 8.12 TB. Galaxy enables scientists with no programming or system administration experience to perform complex analysis. Galaxy workflows let users capture all the steps in an analysis, and their order, allowing the analysis to be reproduced. Galaxy workflows and datasets can be shared, enabling transparent research. Finally, Galaxy Training Network (GTN) offers hundreds of online tutorials provided by the Galaxy community.

Galaxy's ObjectStore is its data virtualization layer. It abstracts Galaxy's business logic for data persistence technology. In other words, the ObjectStore makes it possible to store data on a wide-variety of persistence media spanning from local storage to cloud-based solutions. Galaxy's ObjectStore currently supports disk, Network Attached Storage (NAS), and various cloud-based backends, such as S3. In this work, we are extending Galaxy's ObjectStore to add support for iRODS. We discuss the challenges we faced while implementing this feature and how we addressed those challenges and our plans for the future.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

iRODS speaks SFTP: More ways to securely transfer your data

Illyoung Choi, Edwin Skidmore, Nirav Merchant
CyVerse / University of Arizona
iychoi@arizona.edu

ABSTRACT

Secure File Transfer Protocol (SFTP) is a widely utilized and supported protocol for securely transferring data. There are multiple client options that are open source and cross platform which include both command line and desktop GUI's (Graphical User Interface).

The need for compliance and data encryption during transfer is a strict requirement for many science domains that are working with confidential data e.g. public health records, the use of SFTP based transfer and clients is well known and validated, thus meeting multiple compliance needs.

Realizing this unmet need for secure and encrypted transfers for CyVerse users, our team decided to implement SFTP access to iRODS. This approach complements the existing secure data transfer and authentication method currently provided in iRODS via SSL and PAM authentication, which however are challenging to integrate into existing services or research workflows for multiple reasons: requiring changes on iRODS server, firewall configurations, and training users for complex client side installations of icommands.

In this talk, we introduce our work on adding iRODS as a backend storage option for SFTPGo (https://github.com/drakkan/sftpgo) utilizing the Go iRODS library developed at CyVerse (https://github.com/cyverse/go-irodsclient). We also redesigned its public-key authentication on top of iRODS Proxy Authentication, thus avoiding users embedding passwords in their scripts and automation and relying on key based authentication. The system is easy to deploy and has been validated with popular desktop SFTP clients such as FileZilla, Cyberduck etc. Our deployment of the system showed 58.0 MB/s for uploading and 15.0 MB/s for downloading when transferring a 1GB file. Compared with SFTPGo's local storage, the iRODS integration to SFTPGo showed reduced I/O performance due to remote data access – SFTPGo's local storage showed 77.0 MB/s for uploading and 64.0 MB/s for downloading. We plan to optimize the code to improve the I/O performance.

We expect the new system, SFTPGo for iRODS will allow researchers working with confidential data to readily integrate this capability into their research workflows alongside familiar client tools while meeting some of their compliance requirements.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

iRODS Delay Server Migration

Terrell Russell
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

Kory Draughn
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
korydraughn@renci.org

ABSTRACT

The iRODS Delay Server can now be safely moved from one iRODS server to another without requiring a restart. This paper describes the requirements, the design goals, the algorithm, the implementation, and the effects of this new functionality.

Keywords

iRODS, delay rules, delay server, migration

DELAY QUEUE

The iRODS platform provides powerful data management capabilities through the combination of storage technology abstraction (via resource plugins), data discovery (via metadata), and policy enforcement (via the rule engine framework). The rules allow taking action on data, based on the metadata (and any other available inputs). iRODS policies (rules) can be fired in one of three ways:

1. now (upon request)
2. now (upon action)
3. later (via delay())

The iRODS Delay Server (`irodsDelayServer`), formerly known as the Rule Execution Server (`irodsReServer`), is the mechanism provided by iRODS to run rules at a later time than when they are enqueued (option 3 above). iRODS rules can be enqueued via `delay()` to execute at a later time. Any rules that are queued persist in the iCAT database and are processed by the `irodsDelayServer` in a priority order. The `irodsDelayServer` sleeps most of the time, but spawns an `irodsAgent` every 30 seconds (by default) to check the Delay Queue for any delayed rules that need to be run.

This paper will discuss the architecture of the iRODS Delay Server in iRODS 4.3.0 and the process by which the iRODS Consortium arrived at this design and implementation.

DELAY SERVER ARCHITECTURE

The iRODS 4.3.0 Delay Server Architecture is the culmination of design, preparation, and incorporation of smaller features for more than four years. 4.2.4 was released in 2018 while the first changes discussed here were included in 4.2.5 in 2019.

iRODS UGM 2022 July 5-8, 2022, Leuven, Belgium

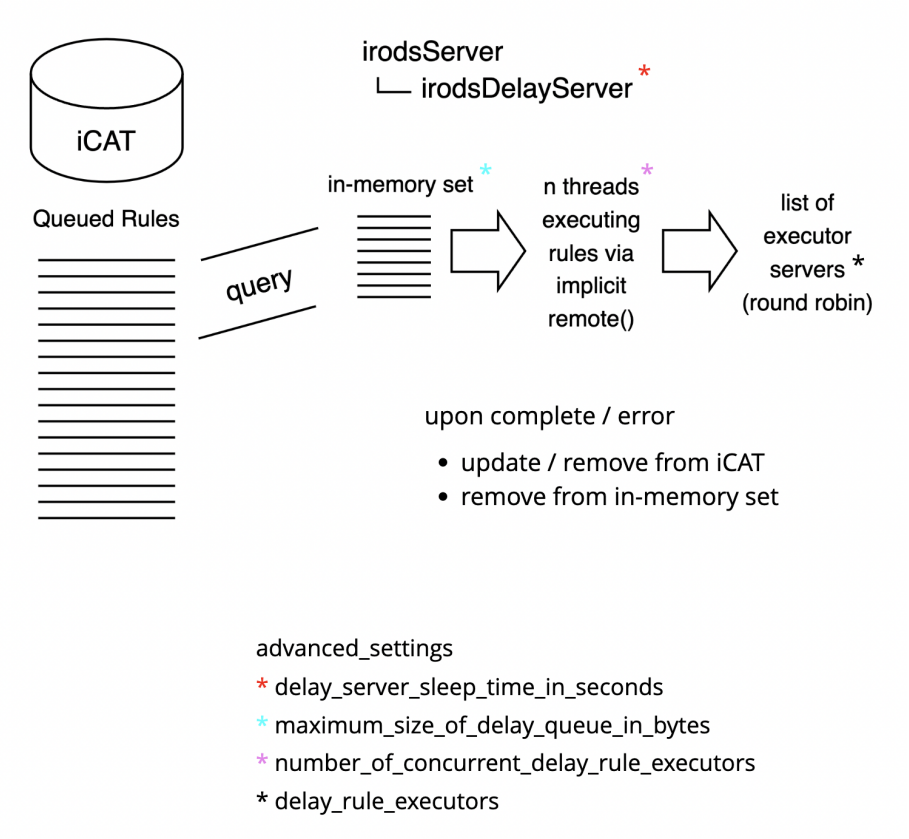

Figure 1. iRODS 4.3.0 Delay Server Architecture, incorporating updates from 4.2.5 to 4.3.0

4.2.5

iRODS 4.2.5 included updates that fixed the delay queue from being blocked by long-running rules (issue 4250 [1]). It also moved from processing rules directly from the catalog to holding an in-memory set of rules that had been fetched via query and then processed with fewer round trips to the database. This allowed a more efficient use of cores within a single machine to process the queue of delayed rules. In addition, the entire delay server itself was refactored to use threads instead of individual processes (issue 4251 [2]), reducing the memory footprint (issue 3782 [3] and issue 4266 [4]) and allowing reuse of some data structures.

The addition of the in-memory set required a new advanced setting named `maximum_size_of_delay_queue_in_bytes`. This gave the administrator control over how much memory is allocated to running the delay server. As subsequent queries are executed to pull rules from the catalog, if they do not fit into this in-memory set, then they are discarded until enough rules have been executed and there is room for additional rules to be added.

4.2.8

iRODS 4.2.8 included a refactoring of the delay server binary itself to use the newly available query processor (issue 4430 [5]). This reduced the length, redundancy, and complexity of the delay server by calling purpose-built library code. The delay server had been a full copy of the iRODS server source, which was wasteful of memory and required any bugfixes to be applied in two locations.

4.2.9

iRODS 4.2.9 included a change in location where the delay rule contexts were stored (issue 3049 [6]). Prior to 4.2.9, delay rule context was stored on the disk of the machine running the delay server at the time of enqueuing. A new text-like column was added to the `R_RULE_EXEC` table, namely `exe_context` of type `text` (PostgreSQL), `longtext` (MySQL), or `clob` (Oracle). By moving the context into the database, it separated the enqueuing machine from the execution machine. This allowed the delay server to be a different machine, if desired, at a later time.

4.3.0

iRODS 4.3.0 included two additional changes that allowed the delay server to be moved from one machine to another within a zone.

As part of the algorithm (discussed in a later section), the iRODS server must now regularly check to see if it is designated to run the delay server. There were other items that required regular work in the server, so it was decided to implement a cron-like facility within the iRODS server for these tasks.

One of the use-cases that was presented during discussion with the community included the ability to run a large number of concurrent delay rules at the same time, potentially on different hosts. To solve this need to scale up the processing of delay rules, we added an admin-defined list of eligible executors (servers) which could receive delay rules and context and run them in a distributed manner. To get this behavior by default, we also implemented an implicit `remote()` call (issue 4429 [7]) when processing the in-memory set which would send rules to a selected member of the list of executors (for now, in a round robin, or sequential, manner). The default and upgrade behavior is to have an empty list of executors which will use the local server as the only executor.

DESIGN GOALS

The sections above describe the work that was necessary to prepare for a future with a delay server migration algorithm running. The following design goals were defined through community discussion and brainstorming. There were four main goals.

1. No `irodsServer` restarts required. There are very large deployments under continuous load and a restart would be very disruptive, and probably require a maintenance window. This is unacceptable if a single delay server unexpectedly going away could cause the same disruption.
2. No double spends. Only one `irodsDelayServer` can be running in a particular Zone at any moment as it is the one pulling enqueued rules from the catalog, executing them, and then removing them from the catalog. This is paramount for ensuring organizational policy is followed, as a delay rule shall not be executed more than once.
3. Hands-free migration in case of disaster. If the `irodsDelayServer` is not coming back (the machine it is running on is dead or has been disconnected), then another machine in the Zone should be promoted to run the `irodsDelayServer` without any new action from the administrator.
4. Visibility. The delay server migration process should have as few moving parts and controls as possible, making it easy to interrogate and debug for the administrator.

These four elements led to the following approach.

APPROACH

The following three operational decisions were made to satisfy the design goals.

1. Use the transactional database to store zone-wide information. This provides a single source of truth and allows for confidence that all servers can act on the same information.

2. Split roles of leader and successor. To allow for the asynchronous nature of multiple servers agreeing on their duties as assigned, splitting the leader from the successor allows each server to run appropriate code and make correct decisions.

3. Run an identical algorithm on all iRODS servers. Each server is responsible for their own behavior but together, they provide a predictable, consistent response during a migration event.

The roles are designated as zone-wide settings in the catalog in the `R_GRID_CONFIGURATION` table.

namespace	option_name	option_value
delay_server	leader	<hostname>
delay_server	successor	<hostname>

Table 1. New configuration options

DEMONSTRATION

The following sequence of administrator shell commands shows the interrogation of the delay server status from the catalog, the setting of the new delay server, and then watching as the values change as the servers execute the algorithm and perform a clean delay server migration.

```
$ hostname
05f4be918c0f

$ iadmin get_delay_server_info
{
    "leader": "other.server.example.org",
    "successor": ""
}

$ iadmin set_delay_server $(hostname)

$ iadmin get_delay_server_info
{
    "leader": "other.server.example.org",
    "successor": "05f4be918c0f"
}

$ iadmin get_delay_server_info
{
    "leader": "05f4be918c0f",
    "successor": ""
}
```

Demonstrated are the two new `iadmin` subcommands, `get_delay_server_info` and `set_delay_server`.

The help text for these two subcommands are as follows:

```
$ iadmin h get_delay_server_info
```

```
get_delay_server_info

Prints information about the delay server as JSON.

This command allows administrators to identify which server is running the
delay server and if the delay server is being migrated.

This information is retrieved from the R_GRID_CONFIGURATION database table.

Example Output:

    {
        "leader": "consumer-1.irods.org",
        "successor": ""
    }

$ iadmin h set_delay_server
set_delay_server HOSTNAME

Set the delay server for the local zone in R_GRID_CONFIGURATION.

The hostname entered will be saved as the 'successor'.

Each iRODS server will periodically check the catalog to determine
if it should promote itself to be the delay server for the local zone.

This mechanism allows for graceful delay server migration without downtime.
```

ALGORITHM

This algorithm is designed to run regularly on all servers in a zone. It is in charge of starting and then stopping a local instance of the `irodsDelayServer`.

```
if self == leader
    if successor defined and not self
        gracefully finish and exit
    else
        if necessary, start irodsDelayServer
else if self == successor
    run health check on leader
    if leader is not running
        promote self to leader in iCAT
    else
        save health stats
else
    if necessary, gracefully finish and exit
```

There are three roles a particular server can find itself in, leader, successor, or neither.

1. Leader.

If the server running the algorithm checks the catalog and determines that the designated leader hostname matches its own hostname, then the first stanza is executed. If there is a successor designated in the catalog as well, and it is not also the same as this server's hostname, then a migration has been requested by the administrator and it is time for this server to request the local `irodsDelayServer` complete any delay tasks that are already being executed and then gracefully exit.

Otherwise, if the server is not currently running an `irodsDelayServer`, then it should start a local instance.

2. Successor.

   If the check of the catalog produces a hostname for successor that matches the local hostname, then it becomes the job of this server to monitor the leader for signs of health. If the leader is still running its own `irodsDelayServer` (meaning that it has not yet gracefully exited), then the successor should save into the catalog that it has checked once and plans to try again on the next check. If the leader is not running (or if the health check is deemed to have failed because of repeated non-response), then the successor promotes itself to leader within the catalog and returns. The next time through the algorithm this server will find it matching the 'leader' condition (above).

3. Neither.

   If the check of the catalog produces no match for either leader or successor with the local hostname, then this server should tell any local `irodsDelayServer` to complete any work it has and then gracefully exit.

The two-phase promotion of a server to 'leader' in this algorithm should behave in the following scenarios:

1. Servers behaving as expected.

   Only the leader will ever start an `irodsDelayServer`. If requested, the leader will let go as soon as it has completed its current work. All other servers will do the same if not designated as the leader. The successor will not promote itself until the leader has let go.

2. The current leader is non-responsive.

   If the current leader has suffered an unrecoverable error or has been disconnected, then the successor performing a series of health checks can promote itself to leader. Only the successor can promote itself to leader.

3. The current leader has long-running jobs.

   The health check by the successor is implemented by asking for the PID of the `irodsDelayServer` on the leader. If the leader continues to answer that it is healthy, and just continuing to process its current rules, then the successor will simply wait and check again.

4. Fast switching of the successor by the administrator.

   If an administrator has changed the successor after the migration has begun, but before the entire algorithm has settled on a new leader and started the new `irodsDelayServer`, then as long as the checks on each machine are running in a well-spaced manner, then two servers should never decide they are both the leader at the same time. For additional confidence, a 'leader' confirmation may be performed prior to each time an `irodsDelayServer` retrieves new delay rules to execute.

LEARNED ALONG THE WAY

There were a few things that proved themselves tricky along the way to arriving at an implementation that satisfied the vision.

1. Database credentials.

   The decision to have the main `irodsServer`, via the cron-like facility, reach out and contact the database directly to gather the information about the leader and successor meant that every server has to have database credentials in their `server_config.json` file. This will be remedied in later releases.

2. Control plane as process / Blocking ourselves.

   The health check required by the successor to ask whether the current leader is still running an `irodsDelayServer` meant that a network request was sent but it could not be answered due to the main loop of the `irodsServer` waiting to fork an Agent to answer the request. This required moving the cron-like facility into its own thread, similar to the control plane, and unblocking the main server from answering incoming requests.

3. Who is the parent process?

   In learning about the control plane above, we learned to write down a PID-file to help other processes be able to identify the parent process and answer whether the `irodsDelayServer` was still running.

FUTURE WORK

This work is complete and functional and included as part of iRODS 4.3.0. We expect to learn a few things once deployments are in the real world, however, the following items are optimizations we have already identified and plan to include in future releases.

1. Remove database credentials requirement

   This requirement is a short-term limitation and will be removed relatively soon.

2. Detect and skip a redundant implicit `remote()`

   The current implementation is wrapping all calls in the `remote()` function. We expect that a simple check to compare the selected remote host against the current hostname will allow skipping a redundant call to `remote()`.

3. Advanced setting for sleep time between migration algorithm runs

   As shipped in 4.3.0, the delay server migration algorithm is set to execute every five seconds. We were trying to make sure that this new feature would be responsive enough to experimenting administrators before providing a configuration. We plan to introduce a new advanced setting for this value, including the possibility of configuring the server to never run the algorithm.

CONCLUSION

This paper discussed the design, implementation, and future work around the new delay server migration algorithm in iRODS 4.3.0. We expect this work to feature prominently in production workloads as many asynchronous tasks are being enqueued for later execution.

REFERENCES

[1] iRODS GitHub Issue 4250. "irodsReServer blocks on in-progress jobs"
`https://github.com/irods/irods/issues/4250`

[2] iRODS GitHub Issue 4251. "Use thread pool in rule execution server"
`https://github.com/irods/irods/issues/4251`

[3] iRODS GitHub Issue 3782. "delayed rule queue processing slows down as the queue length grows"
`https://github.com/irods/irods/issues/3782`

[4] iRODS GitHub Issue 4266. "Having at least 256 rules in the queue prevents new rules from being processed"
`https://github.com/irods/irods/issues/4266`

[5] iRODS GitHub Issue 4430. "Refactor delay server as an irods::query_processor"
`https://github.com/irods/irods/issues/4430`

[6] iRODS GitHub Issue 3049. "Move packedReis to db, add delay server boolean to server_config.json"
`https://github.com/irods/irods/issues/3049`

[7] iRODS GitHub Issue 4429. "Add implicit remote() to delayed rule execution"
`https://github.com/irods/irods/issues/4429`

Towards the FAIRification of lab-data

Martin Schobben
Utrecht University
schobbenmartin@gmail.com

ABSTRACT

Data management and subsequent downstream recycling of data is a primer for future innovation. Solutions for better data management infrastructures, such as formalized in the FAIR data guiding principles, are not yet implemented in most academic laboratories populated by a range of analytical instruments. Each of these machines often has their own vendor supplied software suite for data processing and diagnostics, and thus prevents transparency of these workflows. This so-called "vendor lock-in" further results in various data models which are not easily integrated.

In this talk I want to share some visions and perspectives on a strategy that could aid data collection and harmonization in laboratories. The ultimate aim of the project is to develop an universal tool that can be easily integrated in a day-to-day workflow of a scientist (using Python or R) as well as operating as a sub-system of iRODS for storage and downstream recycling of lab-data.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

iRODS S3 Resource Plugin: Glacier Support

Justin James
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
jjames@renci.org

Terrell Russell
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

ABSTRACT

The iRODS S3 Resource Plugin has been extended to honor the Glacier semantics of an S3 storage system including reacting appropriately to responses that indicate the data requested will be available later. This paper describes the implementation details and future work.

Keywords

iRODS, S3, glacier, storage

INTRODUCTION

The iRODS S3 Resource Plugin has been steadily improving over the years and growing new features, including cacheless behavior (removing the need to have it be configured as a child of a compound resource) [1], detached mode (which removes the need to redirect any S3 request to a particular iRODS server to service that request)[1], and direct streaming (allowing multipart put and get directly into the S3 layer) [2]. This year's update includes new support for Glacier semantics, similar to Amazon's own storage class behavior, but available in multiple other vendor's S3-compatible storage offerings.

OVERVIEW

The S3 storage service provided by Amazon offers multiple different storage classes [3]. The Glacier storage classes are the archive tiers for S3. While these classes and this interface and behavior are defined by the Amazon offering, other vendors have implemented the same. This new iRODS solution has been tested against Amazon and Fujifilm's Object Archive [4] product at this time.

The behavior is transparent (as compared to a regular iRODS transfer) except for a flag that is needed to define the storage class. This flag is part of the request and is required to signal the intent of the caller.

When downloading data, the behavior is asynchronous, except for storage classes that are deemed "instant retrieval". This is a vendor-specific definition and should be investigated and documented for any particular product and deployment. The iRODS S3 storage resource does not know what this definition may mean for any particular S3-compatible storage that may be configured for it to use.

GLACIER SUPPORT ON OBJECT READ

Adding Glacier support was relatively straightforward. Prior to reading an object from the iRODS namespace stored in an S3 resource, a `HeadObject` operation must be called to ascertain if the object is currently in archive within the S3 service. If the object is determined to be in the archive, then the object is requested and restoration will be scheduled. Otherwise, the object is returned immediately.

iRODS UGM 2022 July 5-8, 2022, Leuven, Belgium

Determining the status of the object is through the inspection of the `x-amz-storage-class` header:

- If it exists and is either `GLACIER` or `DEEP_ARCHIVE`, inspect the `x-amz-restore` header:
  - If `x-amz-restore` has `ongoing-request=true`, then a restore has already been scheduled. Return `REPLICA_IS_BEING_STAGED` error with message indicating the object is in process of being restored.
  - If `x-amz-restore` has `ongoing-request=false`, then the object has already been restored. Proceed as normal.
  - If `x-amz-restore` does not exist, the object is in archive. Call `RestoreObject` and return `REPLICA_IS_BEING_STAGED` with message indicating the object is being queued for restoration.
- If `x-amz-storage-class` header does not exist or is not `GLACIER` or `DEEP_ARCHIVE`, the object can be immediately retrieved. Proceed as normal.

RESTORING AN OBJECT FROM ARCHIVE

When it is determined that an object be restored from the archive, the `RestoreObject` operation is requested. To support this operation, two new resource context configuration settings have been introduced to the iRODS S3 resource plugin, `S3_RESTORATION_TIER` and `S3_RESTORATION_DAYS`. The use of these two configuration settings give the administrator full control over the behavior of the object restoration.

`S3_RESTORATION_TIER` - This is the value sent in the `<tier>` tag when `RestoreObject` is called. The values are not case sensitive. Valid values are 'Standard', 'Bulk', and 'Expedited'. The restoration tier, in combination with the storage class, defines the length of time needed to complete the restoration. The following are the restoration times for Amazon's S3 service:

	Glacier	Deep Archive
Expedited	1-5 minutes	Not Allowed
Standard (default)	3-5 hours	Within 12 hours
Bulk	5-12 hours	Within 48 hours

Note: `RestoreObject` is neither necessary nor allowed for objects stored in `Glacier_IR`.

`S3_RESTORATION_DATS` - The number of days the object will be restored. The default in the S3 plugin is 7. (According to Amazon, this is overridden if you have the bucket set up with lifecycle configuration.)

GLACIER SUPPORT ON WRITE OR COPY

For writing into the S3 resource, a single new resource context setting named `S3_STORAGE_CLASS` is provided. It is used to define the destination storage class for uploaded data objects and one of four valid values must be set (these are not case-sensitive):

- `STANDARD` - default
- `GLACIER`
- `DEEP_ARCHIVE`
- `GLACIER_IR` - Glacier Instant Retrieval

If defined, this setting is sent in the `x-amz-storage-class` header for `PutObject` and `CopyObject`.

This header may also have the following values which are either not relevant for Glacier support or have not yet been implemented:

- `STANDARD_IA` - Standard Infrequent Access
- `ONEZONE_IA` - One Zone Infrequent Access
- `INTELLIGENT_TIERING`
- `OUTPOST`

CHANGES TO LIBS3

The `libs3` library did not have support for Glacier and Deep Archive at the beginning of this work. The iRODS fork of the `libs3` library has been updated and the following three changes have been incorporated [5]:

- Implemented the `RestoreObject` API
- Added the ability to set `x-amz-storage-class` header on `PutObject` and `CopyObject`
- Added the ability to read `x-amz-storage-class` and `x-amz-restore` headers from the `HeadObject` header

We are planning to open a pull request to the upstream `libs3` library with these changes.

EXAMPLE GLACIER SETUP AND FILE RETRIEVAL

The following is an example of the configuration necessary to use the new Glacier behavior.

This configuration creates a resource that places files in Glacier, performs expedited restorations, and restores for 1 day.

```
$ iadmin mkresc s3resc s3 ‘hostname‘:/justinkylejames-irods1/amazons3resc \
"S3_DEFAULT_HOSTNAME=s3.amazonaws.com;S3_AUTH_FILE=/var/lib/irods/amazon.keypair;
S3_REGIONNAME=us-east-1;S3_PROTO=HTTP;HOST_MODE=cacheless_attached;
S3_STORAGE_CLASS=Glacier;S3_RESTORATION_TIER=Expedited;S3_RESTORATION_DAYS=1"
Creating resource:
Name:           "s3resc"
Type:           "s3"
Host:           "ce61bbc3beec"
Path:           "/justinkylejames-irods1/amazons3resc"
Context:        "S3_DEFAULT_HOSTNAME=s3.amazonaws.com;S3_AUTH_FILE=/var/lib/irods/amazon.keypair;
S3_REGIONNAME=us-east-1;S3_PROTO=HTTP;HOST_MODE=cacheless_attached;S3_STORAGE_CLASS=Glacier;
S3_RESTORATION_TIER=Expedited;S3_RESTORATION_DAYS=1"
```

To begin, create a local file (`test.txt`) and then put it into iRODS onto the newly created and configured S3 storage resource.

```
$ echo test123 > test.txt
$ iput -R s3resc test.txt
```

Next, try to get the object. The error code and error message lets the user know that the object is in the Glacier archive, has been queued for restoration within the S3 fabric, and to come back again and try to retrieve the file later.

```
$ iget test.txt -
remote addresses: 172.17.0.2 ERROR: getUtil: get error for - status = -721000 REPLICA_IS_BEING_STAGED
Level 0: [-]    /github/irods_resource_plugin_s3/s3/s3_transport/src/s3_transport.cpp:208:irods::error
irods::experimental::io::s3_transport::restore_s3_object(const std::string &,
libs3_types::bucket_context &, const unsigned int, const std::string &, const std::string &) :
status [REPLICA_IS_BEING_STAGED]  errno [] --
message [Object is in GLACIER and has been queued for restoration.  Try again later.]
```

Then, try to get the object a second time. The error code returned is the same as before, since the object is still not yet available, but the error message reflects the slightly different situation within the S3 fabric.

```
$ iget test.txt -
remote addresses: 172.17.0.2 ERROR: getUtil: get error for - status = -721000 REPLICA_IS_BEING_STAGED
Level 0: [-]    /github/irods_resource_plugin_s3/s3/s3_transport/src/s3_transport.cpp:133:irods::error
irods::experimental::io::s3_transport::handle_glacier_status(const std::string &,
libs3_types::bucket_context &, const unsigned int, const std::string &,
irods::experimental::io::s3_transport::object_s3_status, const std::string &) :
status [REPLICA_IS_BEING_STAGED] errno [] --
message [Object is in GLACIER and is currently being restored.  Try again later.]
```

Finally, wait a few minutes and attempt to retrieve the object again. The file is returned cleanly without any errors.

```
$ iget test.txt -
test123
```

STATUS AND FUTURE WORK

The iRODS S3 Resource Plugin has learned the Glacier semantics and has been partially released. Restoration from Glacier was added in 4.2.11.0 and included in 4.3.0.0.

The next release (4.3.0.1) will include support for setting the storage class on `PutObject` and `CopyObject` which will work with 'Deep Archive' for the put, get, and copy operations within the iRODS namespace.

After that, we expect that support for the intelligent tiering storage class should be trivial but this has not yet been implemented or tested.

Additionally, we could write a server-side iRODS rule to read metadata on an atomic put and select the storage class dynamically for object level control.

REFERENCES

[1] James, J., Russell, T., Coposky, J.: iRODS S3 Resource Plugin: Cacheless and Detached Mode. pp65-71. 2019 iRODS User Group Meeting. (2019)

[2] James, J., Draughn, K., Coposky, J., Russell, T. : S3:TNG - iRODS S3 Resource Plugin with Direct Streaming. pp15-24. 2020 iRODS User Group Meeting. (2020)

[3] Amazon S3 Glacier storage classes. `https://aws.amazon.com/s3/storage-classes/glacier/`

[4] Fujifilm Object Archive. `https://datastorage-na.fujifilm.com/object-archive/`

[5] James, J.: libs3 - Add glacier support. `https://github.com/irods/libs3/issues/16`

iRODS and Globus Deployment at the VSC

Vas Vasiliadis
University of Chicago
vas@uchicago.edu

Ingrid Barcena Roig
KU Leuven
ingrid.barcenaroig@kuleuven.be

ABSTRACT

We will provide a brief overview of the Globus service and how it integrates with iRODS for secure, reliable file transfer and sharing from diverse storage systems. We will also describe how the VSC is planning to deploy and use the Globus for iRODS connector as part of the member institutions' data management infrastructure.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

An Update on SODAR: the iRODS-powered System for Omics Data Access and Retrieval

Mikko Nieminen, Manuel Holtgrewe, Mathias Kuhring, Oliver Stolpe, Dieter Beule
Berlin Institute of Health at Charité
mikko.nieminen@bih-charite.de

ABSTRACT

In life science research, an ever-growing number of high-throughput omics assays in the areas of genomics, proteomics, metabolomics and transcriptomics is creating challenges for data management. These challenges include handling large amounts of data, modeling complex experimental designs for studies, making data accessible and enabling collaboration between multiple institutes.

We present an update to SODAR (System for Omics Data Access and Retrieval), which is our effort to fulfill these requirements. SODAR is specialized software for combining the modelling of complex studies with storage of large bulk data. To facilitate data management workflows, SODAR provides project-based data encapsulation and access control, web-based graphical user interfaces, programmatic access via REST APIs as well as various tools for managing data in research projects.

SODAR is based on open-source solutions. The system uses iRODS for bulk data storage, with a transaction subsystem facilitating complex data transfer operations with validation and rollback capabilities. Davrods is used for web access to files and integrations with third party software. Graphical user interfaces and APIs are implemented in Python using the Django framework. The data model is based on the ISA-Tab standard, with a browser and editor component for ISA-Tab studies implemented in Vue.js. Core project management functionalities and related tools are available as a separate reusable library, which allows for creating other data management systems sharing common project access control structures.

SODAR was previously presented in the iRODS user group meeting in 2019. Since then, major development has been done regarding, e.g., metadata editing, iRODS file management and REST APIs. We will demonstrate a use case with an emphasis on these new features and updates.

SODAR is under continuous development and has been deployed in our institutes for several years. It is currently used in over 300 research projects and stores approximately 350 terabytes of data. The system is publicly available as open source with a permissive license.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

iRODS Python/PRC based portal and tools for active data support in research contexts

Paul Borgermans
KU Leuven
paul.borgermans@kuleuven.be

ABSTRACT

The main work is a modular web portal that can be tailored for various needs/use cases. The focus is on "active data" in research data management solutions using iRODS. For manual / ad hoc manipulation of metadata, (hierarchical) schemas can be defined which are rendered as user friendly forms/templates. Further integration of external tools to support specific workflows such as metadata discovery. The software stack is kept simple and uses the Flask web framework, bootstrap 5 UI elements and vanilla javascript.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

iRODS Development and Testing Environments (v8)

Alan King
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
alanking@renci.org

Terrell Russell
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

ABSTRACT

iRODS Build and Test continues to evolve. Testing a distributed system is hard and this paper describes the eighth generation of our efforts to do it well. This paper includes containers, Python, and no Groovy.

Keywords

iRODS, testing, development, framework, python

INTRODUCTION

The iRODS Build and Test infrastructure has now been around for more than a decade and continues to evolve as additional features and coverage are required. The iRODS Team has worked to easily and flexibly deploy different iRODS topologies and efficiently exercise the numerous scenarios that a robust iRODS installation can service.

In addition to writing and maintaining a growing lists of tests, seeing those tests pass builds confidence in the changes developers make to software, asserting the correctness of the changes and of the entire system.

Providing an easy and consistent framework in which to run these tests, and see their results, builds confidence for the entire community.

This eighth version (v8) is the best we've done so far.

HISTORY

The following listing is an overview of the history of the various Build-and-Test Systems for iRODS. Each shows a flow of technologies used to provide confidence in iRODS at the time. Over the years, this series of flows also became the gatekeeper for whether iRODS was ready for a new release.

v1 - July 2011: Python → Node.js → RabbitMQ → Celery → Eucalyptus [1]

v2 - October 2012: Python → Node.js → ssh → OpenStack

v3 - January 2013: Hudson → Python → OpenStack

v4 - October 2013: Hudson → Python → vSphere long-running VMs [2]

v5 - Spring 2015: Jenkins → Python → Ansible → zone_bundles → vSphere dynamic VMs [3]

v6 - Spring 2017: Jenkins → Python → vSphere dynamic VMs → build/test hooks [4]

iRODS UGM 2022 July 5-8, 2022, Leuven, Belgium

v7 - Summer 2019: Docker → Jenkins → Python → Docker → build/test hooks [5]

v8 - Summer 2022: Python → Docker → build/test hooks

LIMITATIONS OF THE PAST

The most recent change to the system is the removal of Jenkins. Jenkins [6], the successor to Hudson, is a java-based automation and continuous integration server that was very helpful in being the place where 'jobs' were saved and managed and run from 2013 to 2022. However, there were a few limitations of that approach that we hoped to overcome with v8.

First, since the move to a Docker-based, every-developer-runs-their-own-system approach with v7, the requirement that every developer must now also be an administrator of their own Jenkins became a bit heavyweight and onerous.

Second, the existing structure of Jenkins is pretty inflexible with its notion of jobs and servers and a history of each job over time. We found that we wanted more granular insight across types of inputs to those jobs, rather than just by job name itself.

Third, the combinatoric explosion of variables that we would like to test had become too difficult to maintain in a simple list of Jenkins jobs. If we realized we wanted to test an additional variable, our list of manually defined and curated jobs (in Groovy!) could multiply by the number of enumerated values of the new variable. This was unsustainable and we found ourselves considering writing Python wrappers to generate these Groovy jobs.

Additionally, the test results were captured in the Jenkins namespace and were hard to extract in a flexible manner. Any packages created needed to be extracted and kept in a parallel namespace taking up room in our mental model and additional disk space on individual developers' machines.

Lastly, also related to disk space, since every Jenkins job was building things from scratch, we were dealing with a Docker image explosion - there was one tag per test run in the system - leading to thousands and thousands of nearly identical images. There were a number of disk full events on the development team which were always surprisingly tricky and annoying to recover from.

Stepping back, we realized our needs both as developers and project maintainers were not being met satisfactorily and we needed something better.

iRODS BUILD-AND-TEST SYSTEM (V8)

After a few whiteboard sessions, we determined that we really wanted to separate the building of packages (solving our Docker image explosion problem) from the running of tests (solving our too-many-Jenkins-jobs problem). We needed a consistent flow from source code to built packages to test results (see Figure 1).

We also were looking to provide a more consistent learning and development environment for interns and new hires. This led us to a very straightforward replacement of the initial generic flow with two stages, development and testing. Each stage is now represented by its own standalone git repository with clean inputs and outputs as seen in Figure 2.

DEVELOPMENT ENVIRONMENT

The new iRODS development environment git repository [7] is designed to provide the machinery to easily build the iRODS server, its various plugins, and the required external dependencies (externals) for all supported operating systems as well as a selection of debugging tools. The abstraction layer to provide this on a single machine is through the use of container technology, currently handled by Docker.

The containers build from code that is local to the host machine and produce local packages on the host machine. All of the build tools and processes are run within the containers and do not otherwise affect the host environment

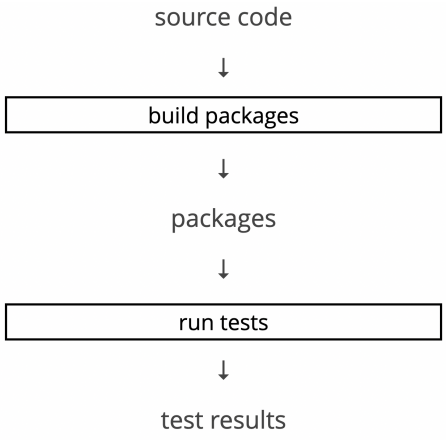

Figure 1. Generic Build-and-Test Workflow

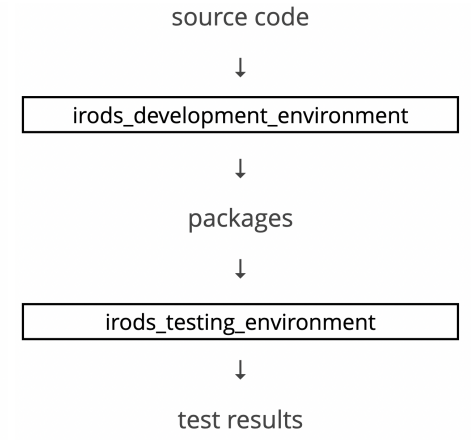

Figure 2. Workflow with Repositories

or filesystem.

The advantages to this approach are numerous when compared to VMs or multiple build machines. First, there is less network traffic to and from the code repositories (in our case, largely GitHub). Second, local source files allow the developer to use their preferred coding environment and tooling which increases speed and confidence. Third, the container-technology's build cache allows for faster iteration between build and test cycles. Lastly, having a consistent process means that the development efforts look very similar to the release process - the same machinery that builds our packages for development and testing is now used to build the packages that are released to everyone else.

An example usage of the iRODS development environment is as follows:

```
$ docker run --rm \
           -v ${irods_sourcedir}:/irods_source:ro \
           -v ${irods_builddir}:/irods_build \
           -v ${icommands_sourcedir}:/icommands_source:ro \
           -v ${icommands_builddir}:/icommands_build \
           -v ${irods_packagedir}:/irods_packages \
           -v ${externals_packagedir}:/irods_externals_packages:ro \
           irods-core-builder:${PLATFORM}-${VERSION}
```

The above example runs an `irods-core-builder` for a particular `${PLATFORM}` and `${VERSION}`. The three read-only (`ro`) volume mounts specify the location of the local source code for the iRODS server (`${irods_sourcedir}`), the iRODS iCommands (`${icommands_sourcedir}`), and the external dependency packages already built and gathered for this platform and version (`${externals_packagedir}`). The other three volume mounts specify the locations of two build directories for the build artifacts (`${irods_builddir}` and `${icommands_builddir}`) and the location where the newly built packages will be deposited (`${irods_packagedir}`).

TESTING ENVIRONMENT

The new iRODS testing environment git repository [8] is designed to provide the machinery to test various iRODS configurations. The python scripts currently leverage the functionality of Docker Compose to easily stand up one or more iRODS zones, configure them, federate them, and then run tests and gather the results. Again, through the use of container technology this happens on a single host machine.

These local scripts execute the issued commands in long-running containers. They install and configure local (newly built) or released packages and generate local test results. The scripts have options to skip the testing which allows a developer to quickly have access to a running zone from their latest built packages for manual inspection and testing.

The first advantage of this approach is precision control for running various tests in parallel since multiple independent containers can run independent zones to avoid any interaction or dependencies. A second related advantage is that this approach provides a convenient way to reproduce reported issues because of the consistent, reproducible configurations. This also provides a consistent process for both bench (manual) and automated testing.

The current list of scripts available in the testing repository:

- stand_it_up.py - stand up a zone with multiple servers
- federate.py - stand up and federate multiple zones
- run_core_tests.py - run iRODS server tests
- run_unit_tests.py - run unit tests for iRODS libraries
- run_topology_tests.py - run tests on a multi-server zone
- run_federation_tests.py - run tests in federated zones
- run_plugin_tests.py - run tests for iRODS plugins

An example usage of running the core tests is as follows:

```
$ python run_core_tests.py \
  --project-directory projects/ubuntu-20.04/ubuntu-20.04-postgres-10.12 \
  --irods-package-directory ~/hdd/builds/irods_packages/4-3-stable/ubuntu-20.04 \
  --concurrent-test-executor-count 4
```

The above example runs the core test suite with configuration details defined in the `--project-directory` located in the relative path of `projects/ubuntu-20.04/ubuntu-20.04-postgres-10.12` with the binary packages found in the `--irods-package-directory` of `~/hdd/builds/irods_packages/4-3-stable/ubuntu-20.04`. The `--concurrent-test-executor-count` of `4` instructs the script to stand up four concurrent identical zones, distribute the tests across those four zones, and run the tests in parallel. As the tests complete, the log files and the test results are copied back to the host machine. Once all tests are complete, the script stops the four zones and removes the running containers.

The results of the fourth zone can be seen here:

```
-----
results for [ubuntu-2004-postgres-1012_irods-catalog-provider_4]
        passed tests:
                [[  30.0808]s]  [test_collection_mtime]
                [[ 837.2263]s]  [test_iadmin]
                [[  59.8457]s]  [test_ichmod]
                [[  13.3225]s]  [test_ifsck]
                [[  58.9188]s]  [test_ils]
                [[   8.9275]s]  [test_imeta_help]
```

```
                [[  34.9601]s]  [test_imv]
<snip>
                [[  28.2190]s]  [test_quotas]
                [[1081.7692]s]  [test_resource_types.Test_Resource_CompoundWithUnivmss]
                [[ 808.2622]s]  [test_resource_types.Test_Resource_Passthru]
                [[2091.9287]s]  [test_resource_types.Test_Resource_Replication]
                [[ 857.6486]s]  [test_resource_types.Test_Resource_Unixfilesystem]
                [[ 917.5663]s]  [test_rulebase]
                [[  78.1721]s]  [test_symlink_operations]
        skipped tests:
        failed tests:
        return code:[0]
        time elapsed: [7.345e+03]seconds ([   2]hours [    2.424]minutes)
-----


All tests passed! :)
time elapsed: [10955.3559]seconds ([   3]hours [ 2.5893]minutes)
==== end of test run results ====
```

The logs for this test run can then be found in the reported location:

```
2022-07-04 20:57:00,726 INFO - collecting logs
[/tmp/ubuntu-2004-postgres-10123x3tjb2r/ubuntu-2004-postgres-1012_be703715-7901-4a34-affa-10ebea651ff4]
```

FUTURE WORK

The next few steps for the iRODS Build and Test infrastructure are incremental. The container-based approach will probably last for a while and progress will come from a few different areas, including automation, client testing, and environmental reproduction and orchestration.

Since moving away from Jenkins, the automation of testing every commit has fallen away. Working to reproduce the visibility of continuous integration is a near-term goal. Some progress has already been made, but it is not clear whether building this from scratch will be worth the effort.

Adding various iRODS clients to the testing infrastructure is an ongoing effort as well. Most clients do not already have their own test suites. Command line tools will be easy enough to write tests for, but GUIs will require additional work.

The original design goal for the iRODS Zone Report was to provide a serialization format for a zone's topology that could be generated from an existing deployment as well as be handed to a tool for automatic deployment for testing and issue reproduction. The format itself has proven useful but needs some modernization work for the 4.3 release series.

SUMMARY

The eighth generation of the iRODS Build and Test infrastructure provides a cleaner slate for the iRODS Consortium to build confidence and visibility around the core iRODS server and its plugins. It has already increased iteration speed for the development team and can guarantee any released binaries come from the same machinery that shows all the tests are passing.

REFERENCES

[1] Russell, T., Coposky, J., Brieger, L., Stealey, M.: Initial Enterprise iRODS Release. 2012 iRODS User Group Meeting (2012). https://irods.org/uploads/2012/03/Russell-RENCI-EiRODS.pdf

[2] Russell T.: iRODS 4.0 - Build and Test. 2014 iRODS User Group Meeting (2014). https://irods.org/uploads/2014/06/Terrell-iRODS-4.0-BuildAndTest.pdf

[3] Russell T., Keller, B.: iRODS Cloud Infrastructure and Testing Framework. 2015 iRODS User Group Meeting (2015). https://irods.org/uploads/2015/06/RussellKeller-TestingFramework.pdf

[4] Russell, T., Gill, J.: iRODS Build and Test (part of the iRODS Technology Update). 2018 iRODS User Group Meeting (2018). https://irods.org/uploads/2018/Russell-iRODS-Technology_Update-slides.pdf

[5] Russell, T., Gill, J.: iRODS Build and Test (part of the iRODS Technology Update). 2019 iRODS User Group Meeting (2019). https://irods.org/uploads/2019/Russell-iRODS-UGM2019_Technology_Update-slides.pdf

[6] Jenkins. https://www.jenkins.io

[7] iRODS Development Environment. https://github.com/irods/irods_development_environment

[8] iRODS Testing Environment. https://github.com/irods/irods_testing_environment

Data: the final frontier. These are the voyages of the Informatics Digital Solutions team at Sanger. Its five-year mission: to migrate old data. To seek out new features. To boldly go where no iRODS Zone has gone before!

John Constable
Wellcome Sanger Institute
jc18@sanger.ac.uk

ABSTRACT

John Constable from the Informatics Support Group, part of the Informatics Digital Solutions team at Wellcome Sanger Institute will talk about the past years work with iRODS, covering migrating 8PB of data, improving the searching of 400 million items of metadata, deploying NFSRODS, switching to PostgreSQL, and adding the usual few petabytes of storage.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

iRODS Client Library: Python iRODS Client 1.1.4

Daniel Moore
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
dmoore@renci.org

ABSTRACT

This talk will cover the new work since 1.0.0 last year. This includes fixes for the XML protocol, connection reuse, the anonymous user, ticket enhancements, and compatibility with iRODS talking directly to S3.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

iRODS Build and Packaging Update

Markus Kitsinger
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
kitsinger@renci.org

ABSTRACT

The release of iRODS 4.3.0 has freed the main branch to begin a new journey. This talk will explore the noble goal of making the iRODS source tree 'Normal and Boring' with regard to CMake modules, inclusion of dependencies, packaging across different operating systems, and general good hygiene.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium

Streamline-connecting data to interactive-apps in CyVerse Discovery Environment via iRODS CSI Driver

Illyoung Choi, Sarah Roberts, Edwin Skidmore, Nirav Merchant
CyVerse / University of Arizona
iychoi@arizona.edu

ABSTRACT

Container technologies such as Docker etc. have seen widespread adoption in many disciplines for building reproducible analysis workflows. The ephemeral nature of container-based workflows presents unique challenges for providing data visibility and access from external data repositories. The CyVerse Discovery Environment (DE) is a web workbench and a managed Kubernetes based container orchestration platform. DE allows researchers to readily build customized apps utilizing Docker containers to perform their custom analysis with data stored in the CyVerse Data Store (DS) which is based on iRODS.

DE stages data needed by the app to local storage where the container is running, and upon completion of the analysis, new data is copied back to DS. This usage pattern is not intuitive for users, as scientific data is becoming increasingly larger the local data staging method becomes more inefficient in terms of transfer time and local storage. Additionally interactive applications such as Jupyter notebooks and Rstudio that support exploratory data analysis (EDA), what data sets need to be staged are not known to the user while launching the container. To overcome many of these limitations we have developed a new method that transfers data on-demand within the DE using the Kubernetes-native storage interface, named the iRODS CSI Driver. This new method provides apps direct data access to the DS, eliminating the need to copy data on local storage. The new method has been deployed in production since January 2022.

In this talk, we demonstrate our work on integrating the iRODS CSI Driver to the DE and share how the CSI Driver is configured within the DE. We also demonstrate new capabilities in the CSI Driver that were added to optimize performance and ease integration. Lastly, we share issues encountered in production and how we fixed them.

The integration of iRODS CSI Driver to the DE enabled scientists to access the DS more conveniently without limitations. Although the current CSI Driver shows reduced I/O performance compared to the previous staging method, the benefits in user experience outweighed slight losses in data access performance.

iRODS UGM 2022, July 5-8, 2022, Leuven, Belgium