Rucio Project Talk

Dimitrios Christidis (CERN)
for the Rucio team and its communities
16 September 2026

Migration to FTS4

- **The upstream Rucio project is not yet actively testing FTS4**
  - Our CI/CD still uses FTS3
- **We have relied on ATLAS and CMS to conduct their own independent tests and report any Rucio-related issues back to us**
- **All existing functionality works without any need for adaptation, with one exception**
  - Given an FTS endpoint and a job ID, the mechanism to construct the URL to the FTS web monitoring is as follows:
    1. Replace the port with `8449`
    2. Append a hardcoded *path*
    3. Append the job ID
  - In FTS3 the *path* started with `/fts3/ftsmon`; in FTS4 it was replaced with `/fts/webui`
  - We assume our communities may need two operate both versions for a period of time
  - Consequently, we need forward compatibility in FTS3 and backward compatibility in FTS4

Scaling up a Rucio instance for a Data Challenge

- **There is no scientific method to calculate how many daemon instances are needed to handle a target load**
- **A lot of anecdotal operational experience (mostly reactionary)**
  - *'If a backlog is being formed in that plot, double the amount of instances'*
  - Rucio is designed to scale horizontally
  - Autoscaling in Kubernetes was considered, but eventually not pursued
- **Let's look closer at the interactions between Rucio and FTS**

Transfer submission

- **The process looks as follows (heavily simplified):**
  1. **The queued requests (pending transfers) are partitioned by their internal ID**
  2. **Each thread fetches some requests from its partition**
  3. **Then prepares them, groups them, and eventually submits them to FTS**
- **Grouping first happens at the level of FTS job parameters (e.g. need for bring-online), then using a policy**
  - Not equivalently efficient (potential fragmentation)
    - `rule`, `dest`, `src_dest`, `rule_src_dest`, `activity_dest`, `activity_src_dest`
  - Multi-source and multi-hop transfers are not grouped
- **A backlog of queued requests is unwanted**
- **The default maximum amount of transfers in an FTS job is 200**
  - What would happen if we increased it?

Transfer finalisation

- **Two different mechanisms (both should be running on a Rucio instance)**
- **By listening to messages on a bus (ActiveMQ)**
  - Low waste, low latency, but detrimental if a message is somehow lost
  - This is the preferred way
- **By periodically querying FTS (polling)**
  - Becomes even less efficient as the number of submitted requests increases
  - This should be used as a safeguard instead

Transfer cancellation

- **Rucio cannot cancel as quickly as it submitted (room for improvement)**
  - Can we cancel multiple jobs with a single request?
- **Grouping of unrelated transfers may result in unintended cancellation**
  - We may want to investigate selective cancellation (specific files within a job)
  - But this, too, may affect overall cancellation performance

Grouping and tape transfers with tokens

- **Some of the grouping policies are incompatible with the sharing of tape-related tokens among transfers inside the same job**
- **We need a bit more time to assess how to implement this optimally**
  - To not delay testing, we may need to mandate using either `src_dest`, `rule_src_dest`, or `activity_src_dest`

The problem with tape-to-tape transfers

- **The FTS REST API was evolved to require a staging access token and/or a polling access token**
  - Based on whether the bring-on-line and/or archive timeouts are set
- **What if the source is tape, the destination is tape, and archive monitoring is enabled?**
  - This is not a theoretical scenario (e.g. recovery after data loss on tape)
- **The polling token will need to be useable at both ends**
  - This means adding both domains to the Audience claim
  - Not an attractive implementation for us

Migration away from GFAL

- **GFAL has served us well over the years**
- **We appreciate that it will receive rudimentary support for EL10**
  - Mainly, we like having official packages, ideally in EPEL
  - Note that Rucio 41 (July 2026) has removed support for SRM and GridFTP
- **The Rucio community will need to find replacement implementations for the WebDAV and XRoot protocols**

XRoot implementation

- **Current implementation built using the XRootD client CLI is not fit for purpose**
- **There is some effort underway to replace it with the XRootD Python bindings**
- **Our requirements:**
  - An intuitive programming interface, matching high-level operations and requiring minimal boilerplate
  - Safe to use in a multi-threaded environment (e.g. passing tokens in isolated contexts)
  - Enforcing high-level operation timeouts

WebDAV implementation

- **Current implementation built using the `requests` library is not fit for purpose**
- **Yet there is something attractive about having a basic installation of a Rucio client running exclusively with Python**
  - We aim to pursue this and see how far we can go
- ***How hard can it be?*™**
  - Most operations should be a single request
  - The exception is the upload operation
  - Discrepancies between storages should be brought to DOMA and standardised

Additional Resources

Website	https://rucio.cern.ch/
Documentation	https://rucio.cern.ch/documentation
Repository	https://github.com/rucio/
Containers	https://hub.docker.com/r/rucio/
Online Support	https://rucio.cern.ch/documentation/join_rucio_mattermost/
Contact	rucio-contact@cern.ch
Journal Article	https://doi.org/10.1007/s41781-019-0026-3
Social	https://fosstodon.org/@rucio

home.cern