

Tape REST API, Tokens & FTS: A Unified Proposal

Mihai Patrascoiu (*CERN-IT Storage*)

Objective

Extend the WLCG Tape REST API with support for token-based AuthNZ

- A pragmatic approach is desired, with minimal number of modifications
- The modifications must align with Experiment Frameworks, FTS and StorageEndpoint systems and should not lead to huge development tasks
- Extensions are agreed by Experiment, FTS & StorageEndpoint representatives
- Extensions are later presented to DOMA community and accepted as the way forward
- Extensions will enable Tape operations to be done fully with tokens

Recap from OTF#1 ([Indico](#))



Recap: Disk-to-Disk Transfers

- Token-based disk-to-disk transfers demonstrated during DC'24, implementation perfected since

(FTS) Approach: require 2 tokens (source & destination) for **each** transfer in submission

FTS in charge of token-lifecycle (via token-exchange & token-refresh flows)

Token deduplication performed FTS-side

- Offers experiments the greatest flexibility, including workflows with per-file scoped tokens
- Opened the door to long-lived, per-file, *unmanaged* token experimentation
(*unmanaged*: FTS won't exchange/refresh the client tokens)
- Well-established in production systems by now

Recap: Disk-to-Disk Transfers

- Token-based disk-to-disk transfers demonstrated during DC'24, implementation perfected since

(FTS) Approach: require 2 tokens (source & destination) for **each** transfer in submission

FTS in charge of token-lifecycle (via token-exchange & token-refresh flows)


Token deduplication performed FTS-side

- Offers experiments the greatest flexibility, including workflows with per-file scoped tokens
- Opened the door to long-lived, per-file, *unmanaged* token experimentation
(*unmanaged*: FTS won't exchange/refresh the client tokens)
- Well-established in production systems by now


Use per-transfer token submission
for tape operations?


Recap: Tape REST API

Staging


HTTP POST /stage
 x509 → <payload>
← <response>

HTTP GET /stage/<req-id>
 x509 ← <response>

HTTP POST /stage/<req-id>/cancel
 x509 → <payload>
← <response>

HTTP POST /release/<req-id>
 x509 → <payload>
← <response>

Archiving

HTTP POST /archiveinfo
 x509 → <payload>
← <response>

Recap: Tape REST API

Staging

HTTP POST /stage
x509 → <payload>
← <response>

HTTP GET /stage/<req-id>
x509 ← <response>

HTTP POST /stage/<req-id>/cancel
x509 → <payload>
← <response>

HTTP POST /release/<req-id>
x509 → <payload>
← <response>

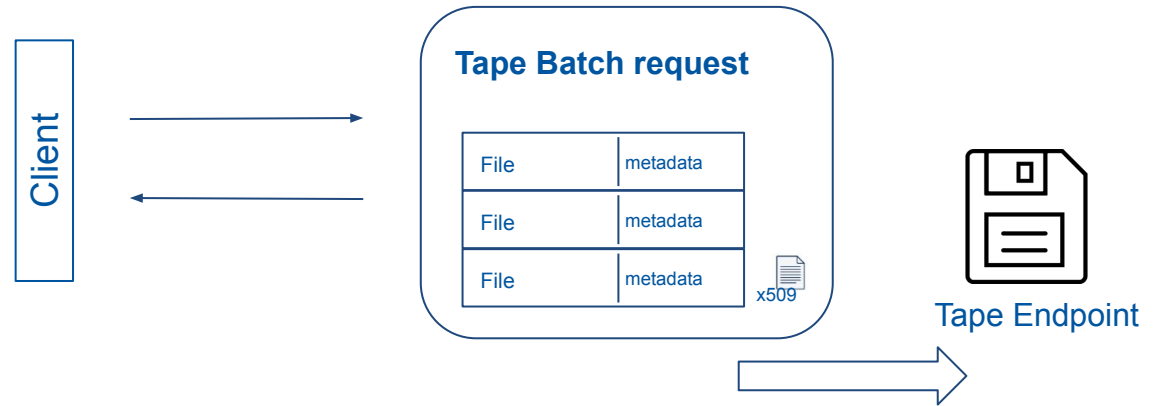
Archiving

HTTP POST /archiveinfo
x509 → <payload>
← <response>

Long-running polling operations (hours to days)

Recap: Tape Storage Endpoints

- Tape storage endpoints deal in “batch” requests, not individual files
- A tape batch: (1 authentication credential, N files)
- The client must construct the batch requests



Recap: Clients, FTS & Tape Batching

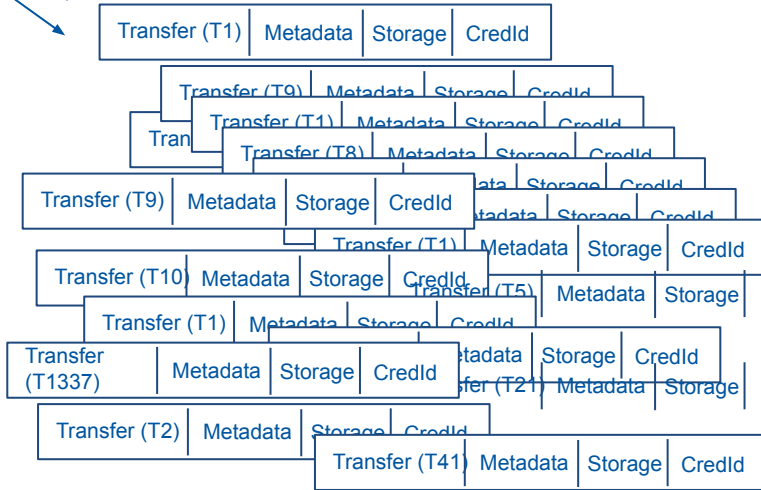
- Experiment frameworks (clients) submit “jobs” to FTS
- FTS jobs are simply a collection of transfers. No common property required by FTS (e.g.: an FTS job can have transfers from/to multiple sources/destinations)
 - Experiment Frameworks & FTS work on a **per-transfer** basis
- Tape Storages require batches for operations → **per-batch** basis

FTS is the bridge between **per-transfer** to **per-batch**

Recap: Clients, FTS & Tape Batching



FTS Submissions



Groups by
Storage & Credential
(configurable size limit)

Tape Batch #1
(Storage: CTA-CERN)



Transfer (T1)	file_metadata
Transfer (T3)	file_metadata
Transfer (T4)	file_metadata



Tape Batch #2
(Storage: dCache-Desy)



Transfer (T2)	file_metadata
Transfer (T5)	file_metadata



FTS, Tape Batching & Tokens

- No direct correlation between experiment framework submission and tape batches
- FTS takes care of the tape batch formation
- **Important:** batches share the same (`storage, credential ID`)
 - With certificates, the credential ID is easy
 - What about with tokens?

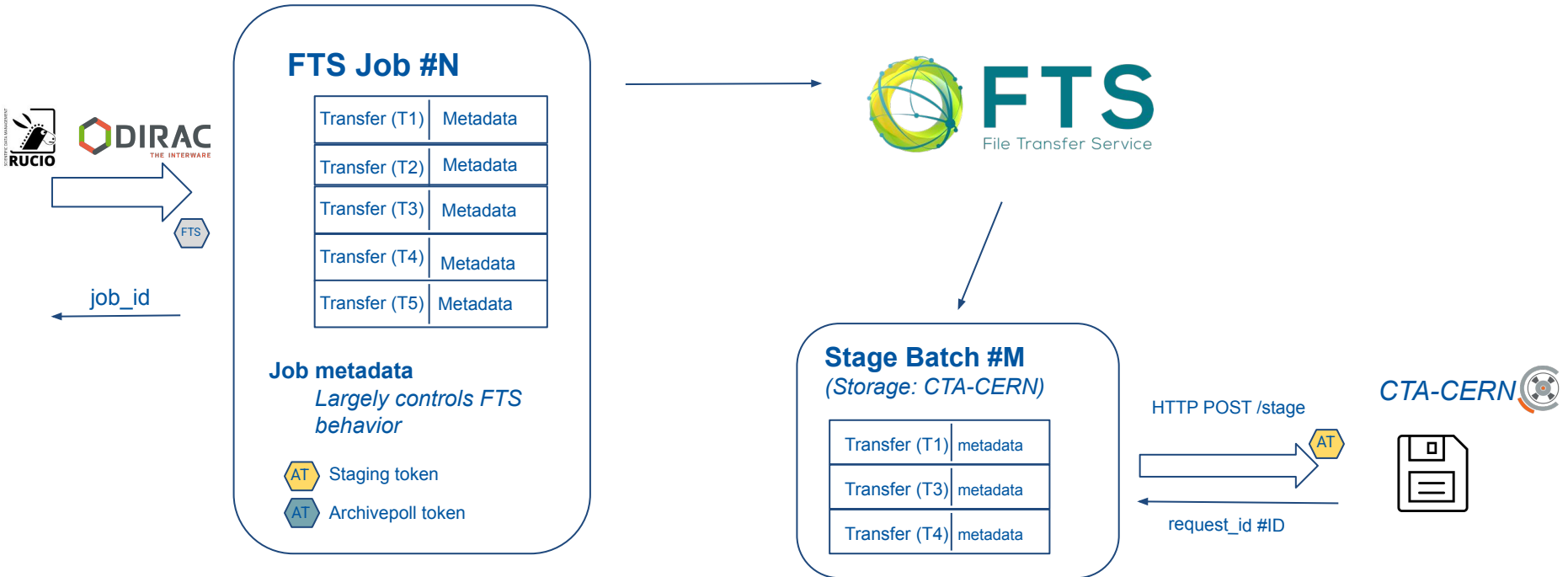
Difficult: ~~tokens are per file (requires large Tape REST API + Storage changes)~~

Pragmatic: make tape tokens per FTS job submission

The Unified “Tape + Tokens” Proposal

- **The TAPE REST API** is modified to accept bearer token credential instead of x509 certificate
 - x509 proxy certificate → “Authorization: Bearer <token>” header
- **Storages** implement AuthZ of tape requests also for bearer tokens
- **FTS** to require a **single staging & archivepoll** token per FTS job submission
 - The tape token (staging/archivepoll) applies to all N transfers within the FTS job submission
 - Grouping into tape batches is done by FTS based on the tape token
- **Experiment frameworks** modify FTS submission interface for tape + tokens

Example: FTS Tape + Tokens Submission



The Unified Proposal: Considerations

- Experiment frameworks continue to submit per-file transfers to FTS
- Experiments now need to ensure the tape token fits the entire job submission !
 - Similar to current x509 situation (if x509 certs could be made fine-grained)
- FTS remains in charge of creating the batches
 - FTS can batch without having to look at the “insides” of the tape token !
- **Minimal changes to the Tape REST API !!**
 - Relatively small changes to FTS & experiment frameworks
 - Design congruent with how Tape Storages currently do authentication (single auth credential for the whole batch)

The Unified Proposal: Considerations

Staging token – needed for stage creation, stage polling & stage abort

- Requires a special **storage.stage:/<path>** scope
- In practice, scope should be large
 - Large scope (storage.stage:/) → similar batching behavior as x509 auth
 - Restricted scope (storage.stage:/<path>) → more batches compared with x509 auth
 - **Risk:** clients submit fine-grained staging tokens → creates many granular stage requests
 - **Mitigation:** FTS already limits the # of staging requests / site

Archivepoll token – needed for archive polling

- Should create a special scope for polling (**storage.stat ?**)
- Similar scope, risk and mitigation as for staging

The Unified Proposal: Considerations

Token lifecycle: FTS will exchange and refresh the staging & archivepoll tokens

- At best, 1 staging / archivepoll token for the grid
- At worst, similar to disk-to-disk per-file transfers
- Pragmatically, 1 staging / archivepoll token will cover $O(\text{thousands})$ files
 - Performance/Scalability, this scenario doesn't raise additional concerns

TokenProvider availability: downtime can disable a large amount of tape polling progress
(continues to raise concerns)

- **Mitigation:** Don't care about polling errors → resubmit → FTS resumes polling
(tape storages don't reset their progress, despite FTS not polling)
- **Mitigation:** Submit long-lived, *unmanaged* tokens
- **Mitigation:** Explore polling tokens offered by the storages (similar to macarons for HTTP-TPC)

The Unified Proposal: Status

- Current proposal was reached after (several) consultations with the involved actors (Tape Storage developers, FTS, Experiment representatives)
- Proposal presented in the WLCG Token Taskforce (29th April 2025):
 - Presentation: [WLCG TokenTF_FTS_Tape_REST_API_Unified_Proposal](#)
 - Proposal up for review & further refinement
- A new scope to be defined (i.e. **storage.stat**) → Extension via WLCG Common JWT Profile
- Implementation timelines to be established (hopeful for late 2025 / early 2026 demonstrations)
 - Track progress via DOMA-BDT Working Group

Need: Run a “Token DataChallenge”, focused on functionality / token-readiness

Conclusions

- A proposal for Tape Operations + Tokens involving:
 - Minimal changes to the TAPE REST API
 - In line with current experiment frameworks - FTS interaction, Tape Storage authentication & FTS tape batching
- Selecting the right staging/archivepoll token falls to the experiment framework
- Improperly scoped tokens can lead to many, small-sized requests
 - mitigated by FTS stage requests limits per site
- TokenProvider availability remains a concern
 - several mitigation options available

Backup

I. Stage

```
HTTP POST https://example.se/api/v1/stage
→ <x509 cert>
→ {"files":
  [
    {"path": "<file-path>", ... },
    ... ]
}

← {"requestId": "<stage-request-id>" }
```

```
HTTP POST https://example.se/api/v1/stage
→ Authorization: Bearer <token>
→ {"files":
  [
    {"path": "<file-path>", ... }
    ... ]
}

← {"requestId": "<stage-request-id>" }
```

- X509 cert replaced with “Authorization” header
- Token must present sufficient **storage.stage:/** scope for the files within
- Minimal changes to v1

II. Stage Poll

```
HTTP GET
```

```
https://example.se/api/v1/stage/<stage-req-id>  
→ <x509 cert>
```

```
← {"files":  
  [  
    { "path": "<path>",  
      "onDisk": <true|false>,  
      "onTape": <true|false>, ... },  
    ... ], ...  
}
```

```
HTTP GET
```

```
https://example.se/api/v1/stage/<stage-req-id>  
→ Authorization: Bearer <token>
```

```
← {"files":  
  [  
    { "path": "<path>",  
      "onDisk": <true|false>,  
      "onTape": <true|false>, ... },  
    ... ], ...  
}
```

- X509 cert replaced with “Authorization” header
- Same **storage.stage:/** token will be used (scopes guaranteed to be valid at this point)
- Minimal changes to v1

III. Stage Abort

```
HTTP POST https://example.se/api/v1/  
        stage/<stage-req-id>/cancel  
→ <x509 cert>  
→ {"paths":  
    [ "<path1>", "<path2>", ... ]  
}
```

```
HTTP POST https://example.se/api/v1/  
        stage/<stage-req-id>/cancel  
→ Authorization: Bearer <token>  
→ {"paths":  
    [ "<path1>", "<path2>", ... ]  
}
```

- X509 cert replaced with “Authorization” header
- Same **storage.stage:/** token will be used (scopes guaranteed to be valid at this point)
- Minimal changes to v1

IV. Stage File Release

```
HTTP POST https://example.se/api/v1/  
        release/<stage-req-id>  
→ <x509 cert>  
→ {"paths":  
    [ "<path1>", "<path2>", ... ]  
}
```

```
HTTP POST https://example.se/api/v1/  
        release/<stage-req-id>  
→ Authorization: Bearer <token>  
→ {"paths":  
    [ "<path1>", "<path2>", ... ]  
}
```

- X509 cert replaced with “Authorization” header
- Same **storage.stage:/** token will be used (scopes guaranteed to be valid at this point)
- Minimal changes to v1

V. Archive Poll

```
HTTP POST
https://example.se/api/v1/archiveinfo
  → <x509 cert>
  → {"paths":
      [ "<path1>", "<path2>", ... ]
    }
← {"files":
   [
     { "path": "<path>",
       "onDisk": <true|false>,
       "onTape": <true|false>, ... },
     ... ]
}
```

```
HTTP POST https://example.se/api/v1/archiveinfo
  → Authorization: Bearer <token>
  → {"paths":
      [ "<path1>", "<path2>", ... ]
    }
← {"files":
   [
     { "path": "<path>",
       "onDisk": <true|false>,
       "onTape": <true|false>, ... },
     ... ]
   }
```

- X509 cert replaced with "Authorization" header
- Token must present sufficient **storage.stat** scope for the files within
- Minimal changes to v1