#FABCONSQLCON2026

FABCON Microsoft Fabric COMMUNITY CONFERENCE | SQLCON Microsoft SQL COMMUNITY CONFERENCE

ATLANTA MARCH 16 - 20, 2026

If You're Not Using Open Mirroring Yet, You're Doing It the Hard Way

Jon Christian Halvorsen

Track: Data Integration

Level 300

Who am I
(and why this talk exists)

Jon Christian Halvorsen
Data Platform Architect, Twoday

- Build Microsoft Fabric data platforms for SMB customers in Norway
- Focus: Ingestion, architecture, operational reliability
- The pattern comes from real pipelines, not labs

Most of this room is paying for orchestration, not for data

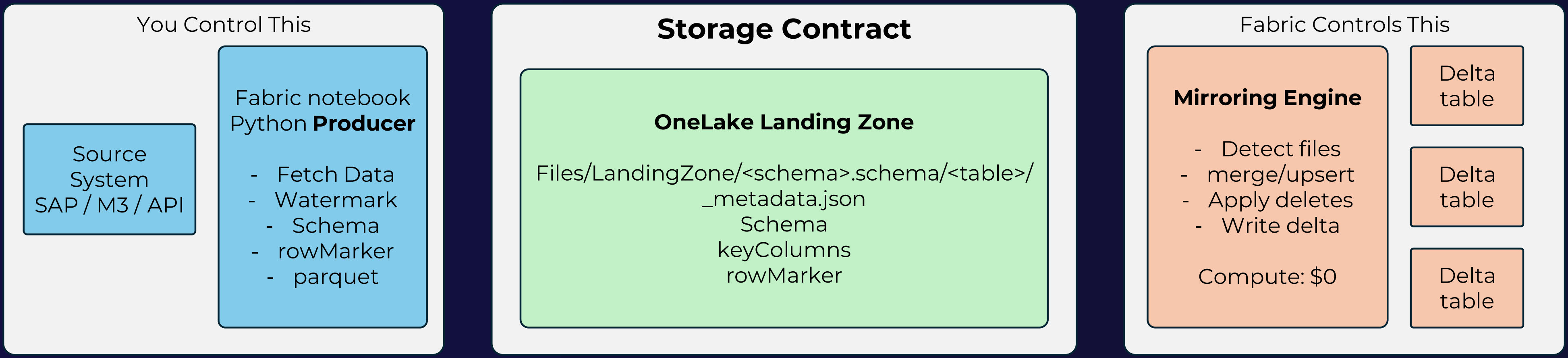
Typical nightly delta runs:

- ~1 minute file generation
  - As always, depends on source latency
- Additional time for mirroring engine runtime
- No manual merge logic
- Some sources: Auto discovery of schemas and keys
- **Ingestion compute cost: $0**

What surprised me building this

- Open mirroring is not a connector – it’s a storage protocol
- The contract is extremely strict
- Abstractions break when file lifecycle changes

Open Mirroring is a Storage Contract
You produce the files, fabric does the rest



This is scheduled batch, not CDC - and that is a deliberate choice

- **Most source systems are batch anyway**

ERP and LOB systems often change hourly or nightly, not every second.

- **CDC Requires always-on compute**

Streaming needs always-on infrastructure. Batch runs only when needed.

- **Open Mirroring makes batch cheap and fast**

You write files on schedule. The mirroring engine handles the merge at **$0 compute**.

Get these three things wrong before the first file and you never fully recover

1. Folder path

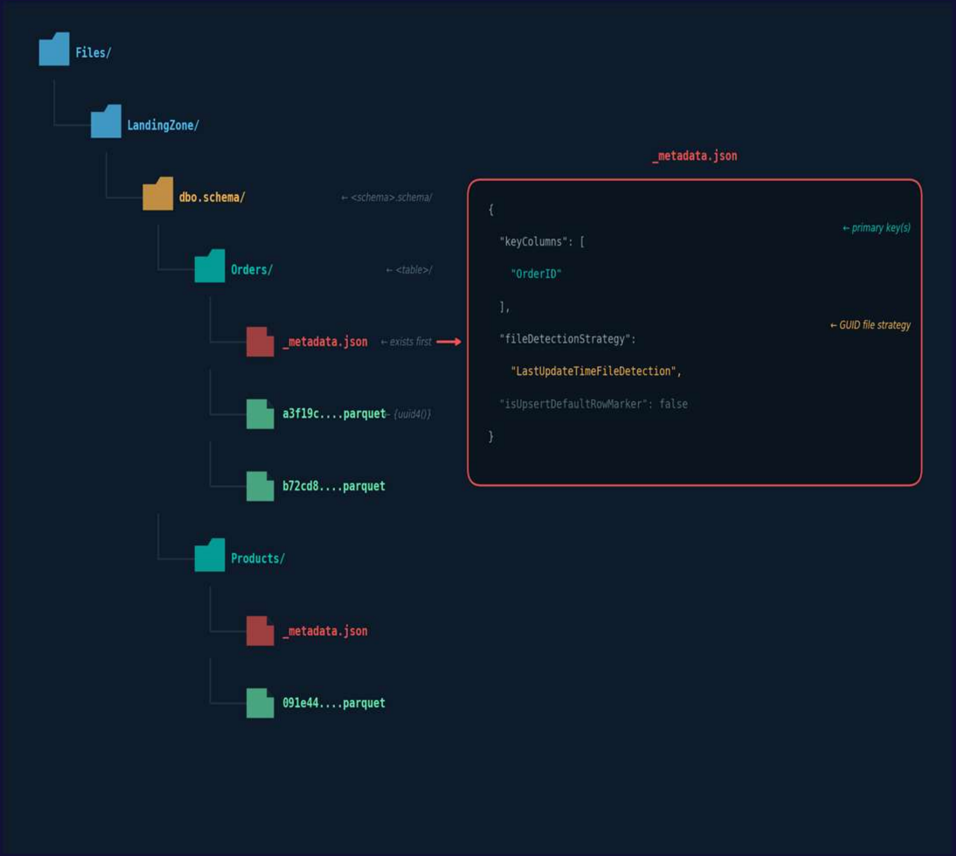
- Files/LandingZone/<schema>.schema/<table>/

2. Schema must never drift

- Column name + order + types must match

3. Key columns define merge behaviour

- Declared in _metadata.json



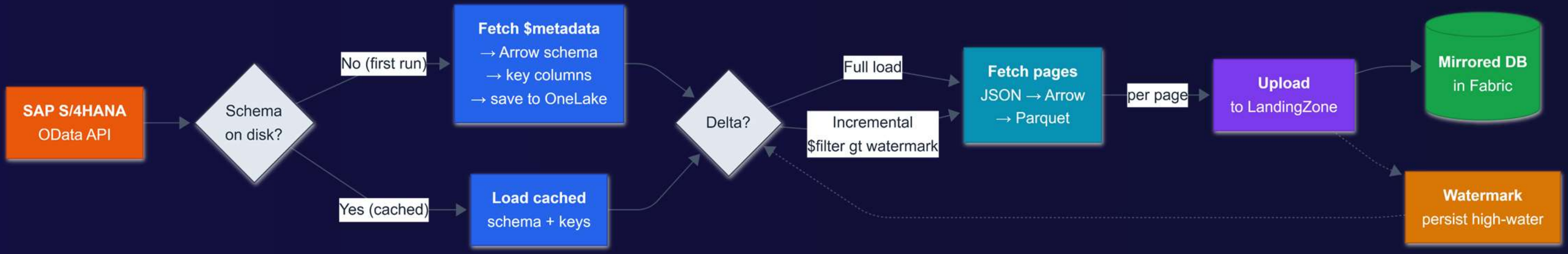
```
{
  "keyColumns": [
    "OrderID"
  ],
  "fileDetectionStrategy":
    "LastUpdateTimeFileDetection",
  "isUpsertDefaultRowMarker": false
}
```

Most sources clear four of five criteria
The one gap is usually deletes

Criteria	SAP OData S/4HANA	Infor M3 Compass	D365 F&O / BC (Odata)	File-based Blob / ADLS
Keys from metadata	✓ auto EntityType Key	✗ static config	✓ auto EntityType Key	✗ static config
Reliable watermark	✗ static Config per table	✓ server-side compasstimestamp	✓ always SystemModifiedAt	✓ blob LastModified
Delete tracking	✗ none poll only	✓ boolean flag in row	✗ none poll only	✗ none poll only
Paging built-in	✓ nextLink /$skip fallback	✓ async job CSV pages	✓ nextLink OData v4	✓ file-by-file LastModified
Schema auto-gen	✓ $metadata full EDM types	✓ type metadata in response	✓ $metadata full EDM types	✗ define once from first file

🟩 ✓ Clean fit   🟥 ✗ Limitation — manageable

When the API has a metadata endpoint, schema and keys come for free

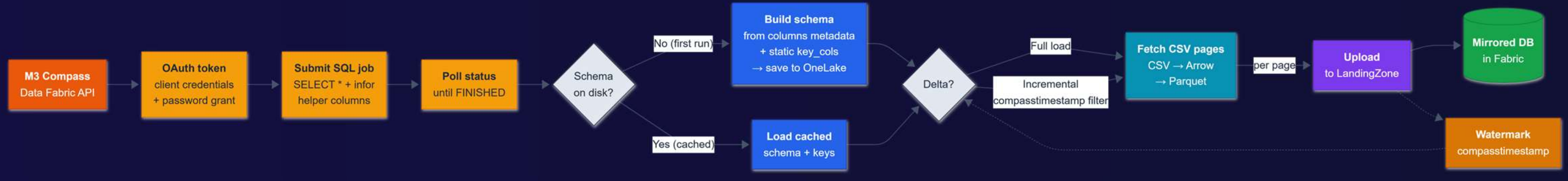


Examples: SAP S/4HANA OData v2/v4, Dynamics 365 Business Central, D365 F&O

The point: $metadata does the work

- One function call per entity — schema + create_table + page loop + watermark all inside
- First run fetches $metadata and caches to disk. Every run after loads from JSON — no HTTP call.
- delta_column in entity config triggers watermark delta loading automatically

When there is no metadata, you implement the same contract manually

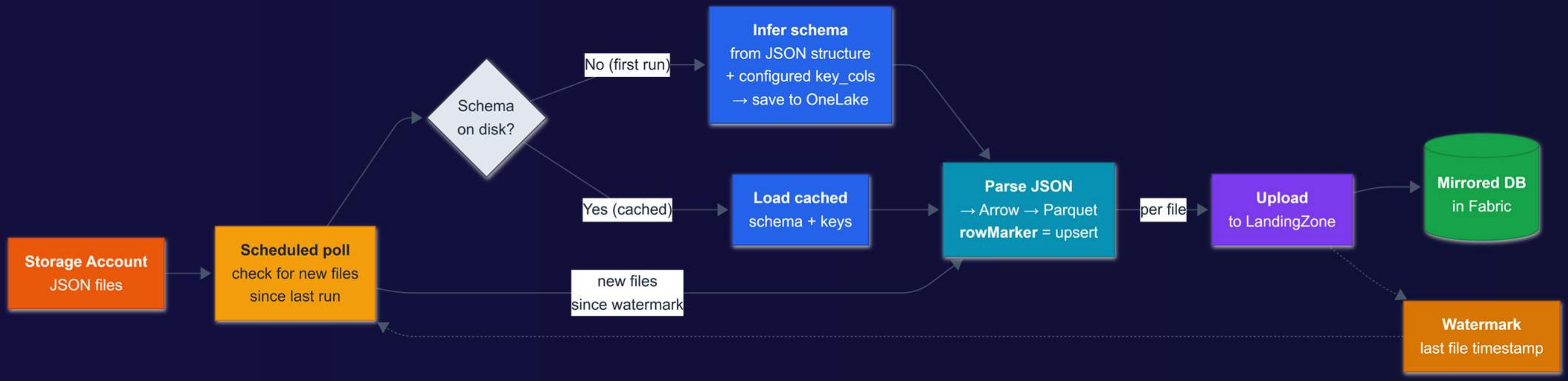


Examples: Infor M3 Compass, Salesforce Bulk API 2.0

The point: same outcome, you do more legwork — submit, poll, page

- M3: keys are static config, no metadata API - specify key_cols in TABLES list
- Watermark: max(infor_compasstimestamp) - same high-watermark pattern

When there is no API at all, blob LastModified is your watermark



Examples: Azure Blob, ADLS — utility meters, IoT exports, nightly file drops, legacy extracts

The point: source does not need an API at all

- **Late corrections handled for free:** meter estimated → actual arrives later → rowMarker=4 (Upsert) overwrites. Zero extra logic.
- Keys are static config — define once from the first file you parse
- Pattern applies to any system exporting to storage: healthcare, construction, energy, legacy mainframes

Demo Time

Three categories of failure — and none of them surface an error message

Nothing loads at all

- Wrong folder path — must be exactly LandingZone/<schema>.schema/<table>/
- Missing or malformed _metadata.json — table is invisible to the mirroring engine
- File named incorrectly — sequential name in a LastUpdateTime table (or vice versa)
- **Fix:** validate folder + _metadata.json exist before first upload; check Monitoring/tables.json

Data lands but looks wrong

- Schema mismatch: column name case, column order, or type differs from the stored schema
- **pandas in the pipeline:** None→NaN, int64→float64, bool→object — silent type corruption
- Decimal128, timestamp[us,UTC], bool — must be built explicitly in Arrow, not inferred
- **Fix:** Arrow all the way from source record to pq.write_table() — zero pandas anywhere in the chain

Permissions and identity

- FabricTokenCredential token expires mid-run — refresh every 55 min or token call will fail silently
- **Fix:** memoize with 5-min pre-expiry buffer (already in OpenMirroringClient.get_access_token)

Change data fails silently; scale constraints fail loudly but too late

Change data goes wrong

- Duplicates / unexpected upserts — keyColumns wrong or missing in _metadata.json
- Deletes don't delete — rowMarker=2 requires correct key match; wrong keys = silently ignored
- rowMarker=0 (Insert) on re-runs creates duplicates — always default to 4 (Upsert)
- **Fix:** verify keyColumns match actual primary key before first load; always ship rowMarker=4

Scale and operational drift

- Too many small files — ingestion overhead dominates; target ~10k rows per Parquet file
- Schema evolution: adding a column requires drop and recreate — no incremental alter
- SequentialFileName counter breaks if Fabric reorganises files — GUID files have no counter to break
- Throttling limits: ~1 TB/day change rate, 500 tables per mirrored database

The rule: treat Open Mirroring like an API contract. Write _metadata once, lock the schema, never manually touch the folder.

Real numbers from two live production pipelines

SAP S/4HANA — 25 OData entities

- Initial full load: 37 entities | ~3 M rows | 12 minutes | Schema auto-generated from $metadata
- Delta runs: 58 seconds for parquet file dumping, 2,5 minute for mirroring engine sync

Infor M3 — 3 largest tables via Compass async SQL API

- Initial full load: Not interesting, limited by Compass API
- Delta runs: 39 seconds for parquet file dumping, 2 minute for mirroring engine sync

No Spark. No Dataflow Gen2. No data pipelines.
No orchestration compute.
OneLake initial file landing operation is the only cost.
OneLake storage free (up to a cap)

Why we implemented a direct storage client

SDK Limitations

- Service principal only — incompatible with notebookutils.getToken(); cannot use Fabric workspace identity
- Sequential filenames only — the counter Fabric now corrupts during its own processed-file cleanup
- Re-initialises _metadata.json on every call — destructive in a running pipeline
- No token refresh — long loads fail silently when the token expires mid-run

What i learned from Raki Rahman's implementation

- Direct azure.storage.filedatalake calls — no SDK abstraction layer, no SDK bugs
- GUID filenames (LastUpdateTimeFileDetection) — no sequential counter, nothing to corrupt
- Token memoized with 5-min pre-expiry buffer — runs as long as your pipeline needs

What the engine can actually do: 1.2 billion rows/minute on F2, 30–60 sec lag to Delta

- Stress-tested by Raki Rahman (Microsoft SQL Server Telemetry Team) — empirical, not theoretical
- File size sweet spot: below 1.25M rows/file — latency degrades above this threshold
- Stress-test on appending data, not on merging, that's why we see longer lag locally.
- rakirahman.me/fabric-open-mirorring-stress

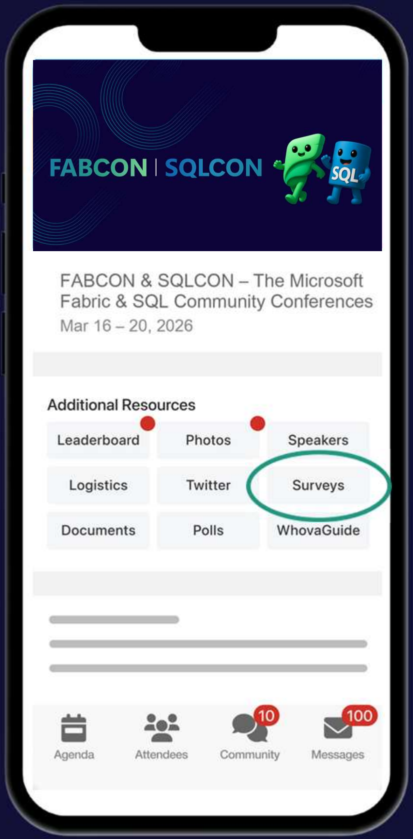
This pattern is ready to ship — here is how to start

- Default to Open Mirroring for new ingestion pipelines
- Use LastUpdateTimeFileDetection + GUID filenames — simpler, no counter to break
- Generate schema from source metadata on first run, cache to JSON forever after
- Add watermarks before you need them — retrofitting delta into a full-load pipeline is painful
- Separate source libs from the mirroring lib — keep the core clean, copy the source lib per source

Benchmark + stress test: **rakirahman.me/fabric-open-mirorring-stress**

How was the session?

Complete Session Surveys in Whova for your chance to WIN PRIZES!


FABCON | SQLCON
FABCON & SQLCON – The Microsoft Fabric & SQL Community Conferences
Mar 16 – 20, 2026
Additional Resources
Leaderboard
Photos
Speakers
Logistics
Twitter
Surveys
Documents
Polls
WhovaGuide
Agenda
Attendees
Community
Messages