

Recovery Without Duplicate Work

Checkpoints, retries, and identity reconciliation in a batch client

ENGINEERING DEEP DIVE

AMY V

14 OCTOBER 2026

A lost response, a repeated cursor, and a missing content body are different failures. A reliable explanation gives each one a specific meaning and a next action. This article examines the policies implemented in a small, inspectable Python reference.

CREATE ONCE

Preserve acceptance

OBSERVE

Bound the reads

RECONCILE

Check identities

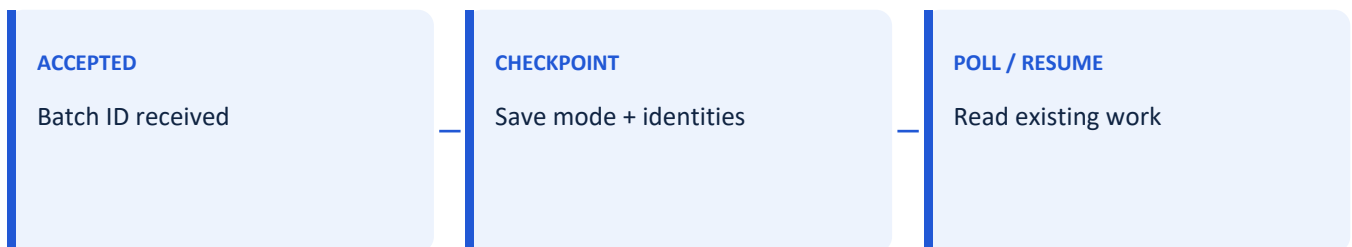
Preserve what the client knows. Do not turn a missing answer into a second operation.

Independent portfolio writing prepared for Hamza and the Olostep team.
Not commissioned by, affiliated with, or endorsed by Olostep.

The response can disappear after acceptance.

Consider a client that sends a batch-creation request and then receives a network timeout. Two histories fit the same observation: the service never accepted the request, or it accepted the request and the response was lost. The client cannot distinguish those histories from the timeout alone. A general-purpose retry decorator hides that distinction at exactly the point where it matters most.

The reference runner attempts creation once. Network uncertainty and server-side 5xx responses during creation become `creation_unknown_do_not_resubmit`. The result is incomplete, with an instruction to reconcile before trying again. This is a conservative client policy; it does not claim that Olostep lacks an idempotency feature or that every failed request created work. It simply does not assume a guarantee it has not established. [1]



A checkpoint is a local recovery promise

When a valid batch ID arrives, the CLI saves a small manifest before its first poll: version, mode, batch_id, and expected_ids. URLs, content, and credentials are excluded. The implementation writes a temporary file, flushes and fsyncs it, then replaces the destination atomically. A conflicting existing manifest is rejected. [2]

This reduces the chance of readers observing a half-written file. It does not make provider acceptance and local persistence one transaction. If the process disappears in that interval, reconciliation is still necessary. Nor is this file format a concurrent, multi-process checkpoint database. Calling it a checkpoint rather than an exactly-once mechanism keeps the promise precise.

A saved batch ID answers which operation to inspect. A saved manifest also answers which results belong to it.

A bare resume ID is supported, but the report labels its weaker provider-count-only verification. Keeping the original custom IDs lets recovery check identity, not merely the size of the returned collection.

Retry reads within one shared budget.

Once the operation is known, observation has a different risk profile from creation. The runner retries reads for network failures and selected HTTP responses: 429, 500, 502, 503, and 504. The default is three total attempts for a read, not three extra retries. Authentication failures stop instead of entering that loop. These are implementation choices in this client, not a provider service-level agreement. [1]

Policy element	Implemented behavior
Delay	The larger of exponential delay and valid Retry-After, plus bounded live jitter.
Retry-After	Supports seconds and HTTP-date forms.
Deadline	Polling, retries, enumeration, and retrieval share the same run budget.
Request timeout	Capped at ten seconds and the remaining run budget.
Insufficient time	Stop instead of issuing an early retry that violates the wait.

This budget prevents a sequence of individually reasonable retry loops from consuming unbounded time. It is still a best-effort application bound: DNS resolution and operating-system networking can exceed a socket timeout. The guide should promise a bounded policy, not hard real-time scheduling.

A cursor is part of the correctness contract

The batch-item API separates item listing from content retrieval. The reference enumerates completed and failed inventories, follows cursor pages, and then retrieves content using each retrieval ID. It rejects repeated cursors, invalid cursor types, duplicate item identities, and excessive page counts. The local cap is 100 pages per category. [1, 3]

A terminated loop alone does not prove a complete inventory. The runner compares the observed identities with the original manifest and checks the provider total when available. If three records come back but all three IDs are unexpected, equal counts are not success. If no manifest or usable provider count exists, completeness remains unknown.

A bounded live run supplied an integer cursor on its first page and omitted the cursor on its last. That observation supports this adapter path. Repeated-cursor failures remain fixture tests, not observed incidents in the live service. [4]

Make the recovery path executable.

The following sequence runs from a fresh checkout with Python 3.11 or later. It uses synthetic fixtures only: no API key, live request, or credit-consuming creation is required. The commands below were rerun for this article. Their purpose is to teach the distinction between a partial result and an interrupted workflow. [5]

```
python -m unittest discover -s tests -v
python -m lab --scenario partial --output output/partial
python -m lab --scenario timeout --budget 3
python -m lab --checkpoint output/checkpoints/batch_fixture.json --scenario success
```

Each command occupies one line and works in PowerShell or a POSIX shell. Run the sequence from a fresh checkout so an unrelated local checkpoint does not conflict with the fixture manifest. The final report is written to output/latest; the partial example is retained separately.

Step	Expected evidence
Tests	42 tests pass. The suite is local and offline.
Partial fixture	Two completed items, one failed item; report status is partial.
Timeout fixture	Incomplete report with deadline_exceeded; CLI exit code 2.
Checkpoint resume	Reads existing fixture batch; three completed items; no create event.

Exit code zero means the workflow finished, not that every source succeeded. A caller must inspect the report status to distinguish complete from partial. The timeout command deliberately returns a nonzero code; it is an expected demonstration result, not a failed installation.

Turn the explanation into a regression check

The suite asserts that a checkpoint exists before the first poll, that recovery does not create another batch, and that a Retry-After exceeding the available budget prevents another request. It also rejects wrong identities even when counts match. These tests protect the claims in the tutorial. A test count alone would be much less informative than this connection between behavior and explanation.

The record of a successful resume should retain the original verification basis. Losing the manifest and silently falling back to count equality would make the tutorial look resilient while weakening the very assertion it teaches.

Keep the evidence stronger than the claim.

Retrieval introduces another boundary. A completed item may have no retrieval ID, an empty Markdown body, or a hosted-content condition that this client does not follow. The runner records such items as unresolved. Its nonempty-string check is a transport-level usefulness test, not a content-quality score. Separate evaluation is needed before using the material for research, decisions, or model inputs. [1, 6]

Notifications prompt investigation

The companion SQLite inbox is an isolated notification simulator. It stores an event ID and a digest of the semantic payload. Identical redelivery is ignored across restarts; changed data under the same event ID is rejected. Delivery-attempt metadata is excluded from the digest so ordinary redelivery metadata does not become a false conflict. [7]

That does not authenticate a public sender or deliver exactly-once business effects. Olostep documents completion notifications and retries; a production consumer still needs a verified authentication design, durable scheduling, and crash recovery around its own actions. In this reference, a notification is a reason to reconcile through authenticated reads, not a trusted replacement for the underlying inventory. [8]

A narrow live check is valuable precisely because it is narrow

The recorded two-URL run completed creation, polling, pagination, identity reconciliation, and Markdown retrieval. It required no adapter changes for that lifecycle. The response order differed from the submission order and usage remained null. The run did not test real failed-item responses, concurrent clients, public webhook delivery, billing, or general extraction accuracy. [4]

The practical result is a reviewable contract between the explanation and the implementation: one creation attempt, bounded observation, retained identities, explicit unresolved content, and a report that keeps unknowns visible. Those decisions can be taught, tested, and revised independently of the provider implementation.

SOURCES & REVIEW PATH

1. [Runner implementation and policy.](#)
2. [Checkpoint persistence and validation.](#)
3. [Olostep Batch Items reference.](#)
4. [Recorded live check and its limits.](#)
5. [Quickstart, commands, and offline tests.](#)
6. [Olostep Retrieve Content reference.](#)
7. [Isolated SQLite notification inbox.](#)
8. [Olostep webhook reference.](#)