The Low Code Factory

ENGINEERING WHITE PAPER

Migrating enterprise content without losing a record.

How a checkpointed, resumable pipeline moved three terabytes into OpenText Content Server, and why losing a document and duplicating one had to be made unreachable rather than unlikely.

THELOWCODEFACTORY.COM HELLO@THELOWCODEFACTORY.COM JULY 2026 · V2.0

Abstract

An education group had accumulated more than a million files and folders across five separate repositories: local directories, Windows network shares, an Arabdox archive, a Constellio export and a database-backed document management system. All of it had to reach OpenText Content Server with folder structure, version history and source metadata intact.

The difficulty in moving three terabytes is not bandwidth. It is that a job of that length will be interrupted, and an interrupted migration has two ways to fail — losing a document, or writing it twice — both of which are unacceptable and neither of which announces itself.

This paper describes the engine built for that migration: a Scanner and a Worker that never share a fate, a database that is the only thing permitted to know what has happened, identity derived from the source rather than from local bookkeeping, and a verification pass that reads every document back out of the target before calling it done.

3 TB
of content consolidated

1,000,000+
files and folders, without loss or duplication

5
legacy source systems retired into one

SECTION 01

The estate

Content had been accumulating for years, and it had accumulated in five different places. Some sat in ordinary directories on local disks. Some lived on Windows network shares that had been added department by department, each with its own conventions. The rest was held in three document management systems, every one with its own storage model, its own metadata scheme, and its own idea of what a version is.

SOURCE	CHARACTER
Local directories	Unstructured; conventions vary by department
SMB / CIFS shares	Windows network shares, added ad hoc over years
Arabdox	Archive export directories
Constellio	XML export with separate blob storage
Database-backed DMS	Content and metadata in relational tables

Together they held more than a million files and folders, roughly three terabytes in total. The material mattered: student and staff records, correspondence, and scanned documents that an auditor can ask to see years after the person who filed them has left the organization.

SECTION 02

Why migrations of this size fail

A migration running for days will be interrupted. This is not a hypothetical to be defended against; it is the normal operating condition. A network share disappears for ninety seconds. A disk fills. A server reboots overnight for patching. An operator stops for a maintenance window and resumes in the morning.

When that happens, the two available failure modes are both severe.

- **Silent loss.** A document is never transferred, and nobody discovers it until it is requested — often years later, by which time the source has been decommissioned.
- **Duplication.** A document is written to the target twice. This is the more expensive of the two, because de-duplicating a populated content repository is harder than the original migration was to run, and the duplicates are frequently indistinguishable.

The approach most teams reach for makes both outcomes likely rather than unlikely. A script that scans and uploads in a single pass holds its progress in memory. A crash at hour sixty therefore leaves starting over as the safest available option — and starting over on a partially completed migration is precisely how duplicates are created.

> THE DESIGN CONSTRAINT
>
> Neither failure may be reachable. Not unlikely if everyone is careful — **unreachable**, as a property of the system. Everything in this paper follows from that requirement.

SECTION 03

Architecture

Scanning and transfer never share a fate

The pipeline is deliberately decoupled. A **Scanner** walks the source system and writes an inventory of what it finds into PostgreSQL. It is read-only on the source and it never touches the target. A separate **Worker** reads the records marked ready from that inventory and uploads them to OpenText. It never goes back to the source.

Between them sits the database, and the database is the only thing that knows what has happened. Nothing of consequence is held in an in-memory queue: every discovered node, every version status and every checkpoint is committed before the next step begins. A hard kill costs the request that was in flight and nothing else.

The separation also isolates failure. A broken connection to OpenText cannot corrupt the source inventory, and a source that goes offline cannot disturb an upload already under way.

Uploads are idempotent

Every document carries a stable identity derived from the source: a source UID combined with a checksum of the content. Before uploading, the Worker checks that identity against the target rather than trusting a local flag claiming the work was already done.

This is the distinction that matters. A local flag records what the engine *believes* it did. An identity check records what the target *actually holds*. Only the second survives a crash between the upload and the bookkeeping — which is exactly the window in which duplicates are born.

Re-running a completed batch is therefore safe by construction, and an operator who is unsure whether a job finished can simply run it again. That property is worth more in practice than it sounds: it converts every operational uncertainty from a risk into a non-event.

Uploads are verified

A successful HTTP response is not evidence that a document arrived. After upload, the engine reads the document back out of OpenText and confirms it is present and intact before marking the record complete.

This roughly doubles the number of calls the engine makes to the target, and it is the only reason anybody can state, as a fact rather than a hope, that nothing was lost. The migration is finished when the target says so, not when the network does.

SECTION 04

Preserving what the target cannot hold

Moving bytes is the straightforward part. Preserving structure turned out to be most of the work.

Versions

Multi-version documents keep their full history. The most recent version becomes the canonical document in Content Server, and earlier versions are archived alongside it rather than discarded. A migration that silently flattens a document to its latest revision has destroyed the record it was asked to preserve.

Structure

Folder hierarchies are mirrored exactly, so the shape an archivist remembers still holds after the move. This matters more than it appears to: for the people who use the archive, the folder path *is* the metadata, and it is frequently the only metadata they have.

Metadata the target has no field for

The legacy systems recorded categories, authors and dates for which Content Server has no native equivalent. There are three possible responses: drop them, force them into an unrelated field, or preserve them alongside the document. The first is silent loss and the second is worse, because it corrupts a field somebody will later trust.

The engine writes such metadata into a **JSON sidecar** attached to the document. Nothing is lost, nothing is misfiled, and the material remains available to a future system that can model it properly.

Filenames

Filenames are sanitised for characters the operating system will not accept, with collision-safe renaming. This is the kind of detail that costs almost nothing to handle deliberately and derails an unprepared migration at two in the morning, several hundred thousand files in.

SECTION 05

The control room

The engine is operated through a browser interface rather than a configuration file and a log tail. This was a requirement rather than a convenience: the people responsible for the content are analysts, not engineers, and a migration they cannot run themselves is a migration that leaves them dependent on a supplier for every subsequent source, re-run and correction.

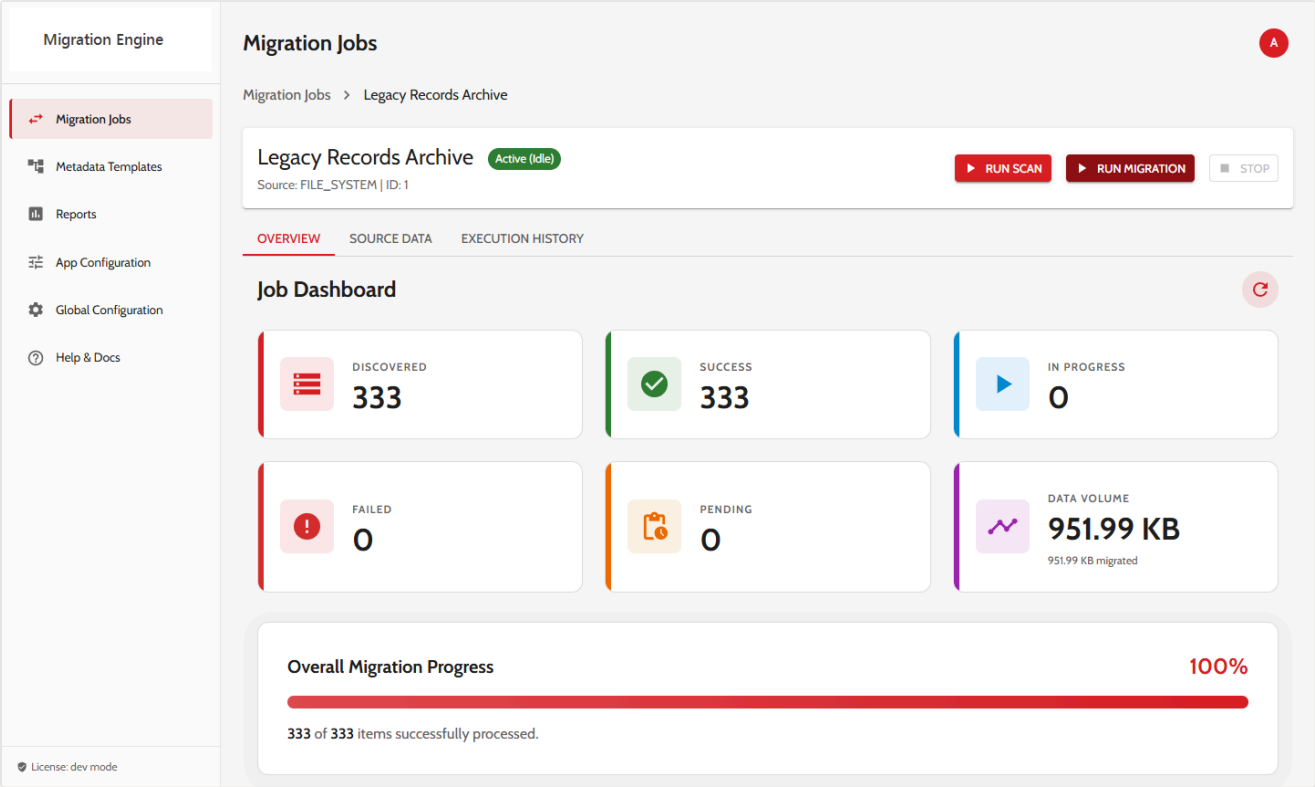

Figure 1. The job dashboard. Discovered, succeeded, in-progress, failed and pending counts with data volume and overall progress, refreshed live while the scan and the transfer run. Screenshot from an instance running against a sample document tree; no customer data is shown.

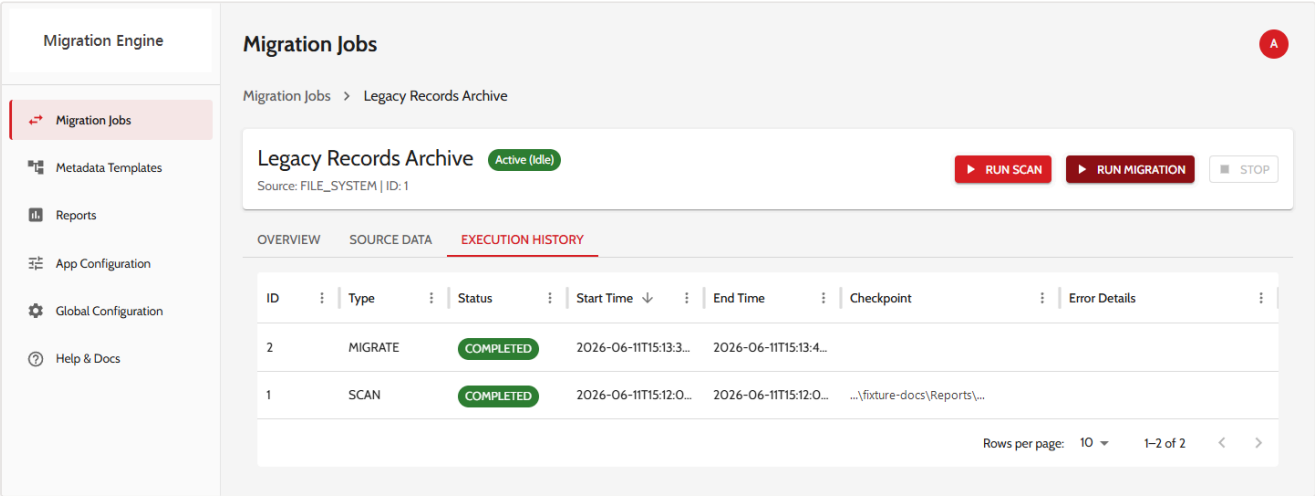

Figure 2. Execution history — every scan and transfer run with its status, its timing and its checkpoint. This is the audit trail that answers the question which always arrives halfway through a migration: what ran, when, and how far did it get?

SECTION 06

Validation before production

A migration that has been performed once against a harmless target is a materially different risk from one that has only been reasoned about.

The engine therefore ships with a **filesystem target connector** alongside the OpenText one. It runs the entire pipeline against local disk instead of Content Server: the same scanning, the same checkpointing, the same idempotency, the same reports. The team rehearsed a complete migration end to end, inspected exactly what would land where, and only then pointed the same engine at the real target.

The control room enforces the same discipline in normal operation. An analyst defines a job, runs the scan, and browses the discovered inventory record by record before anything moves. Nothing is transferred until a person has looked at what was found and agreed that it is right.

Delivery

Changes were branch-gated through feature, development and integration environments before reaching the main line, with a scripted harness that validated connectivity against each source and ran the suite before any promotion. A migration engine is a piece of software that will be trusted with irreplaceable material exactly once; the delivery discipline was set accordingly.

SECTION 07

Deployment

The Spring Boot backend, the React interface and an embedded PostgreSQL are packaged into a single Windows executable with the Java runtime bundled inside it.

This is not a stylistic choice. The engine runs inside the customer's network, on machines administered by people who should not have to install and version-manage a Java runtime in order to move their own documents. There are no prerequisites to satisfy on the host and no build server to depend on. Double-click, and the browser opens to the control room.

Backend	Spring Boot
Interface	React
State	PostgreSQL, embedded
Target	OpenText Content Server REST API
Packaging	Single Windows executable, JRE bundled

SECTION 08

Results

Five legacy sources are connected and retiring. Roughly three terabytes and more than a million files and folders have been consolidated into OpenText Content Server, with folder structure, version history and source metadata preserved.

No document has been lost, and none has been written to the target twice. That is not the result of careful supervision. It is a consequence of the design: identity is derived from the source, every upload is checked against the target before it is attempted, and every write is verified by reading it

back.

The engine is live at the customer's site and their own analysts operate it. The code, the database and the control room belong to them, running in their environment, with no dependency on the team that built it.

SECTION 09

Limitations

- **Verification doubles the target calls.** Reading every document back after writing it is not free. On a slow or rate-limited target this is the dominant cost, and it is a deliberate trade against certainty.
- **The sidecar is a preservation mechanism, not a modelling one.** Metadata written to a JSON sidecar is retained but is not natively queryable in Content Server. It is a guarantee that nothing was lost, not a promise that everything is indexed.
- **A new source is an adapter, not a configuration change.** Each source implements one narrow interface. Adding a sixth system means writing that adapter — a contained piece of work, but not a no-op.
- **Throughput is bounded by the target.** The pipeline is designed for correctness under interruption rather than for maximum speed, and it will not outrun what the destination will accept.

SECTION 10

Conclusion

The engineering that mattered on this project was not sophisticated. It was a refusal to hold important state anywhere except a database, and a willingness to pay the cost of that discipline on every single record. That decision is invisible in a demonstration and decisive at hour sixty.

The second lesson concerns who operates the system. It would have been faster to build a command-line tool and run the migration ourselves. It would also have left the customer dependent on us for every subsequent source and every correction. Building the control room cost weeks and bought them the ability to retire their next legacy system without a supplier — which is the outcome the engagement was actually for.

The Low Code Factory

An enterprise solutions company. We design, build and modernize business-critical applications, workflows, integrations and internal platforms.

thelowcodefactory.com · hello@thelowcodefactory.com · linkedin.com/company/tlcfworks