The Low Code Factory

ENGINEERING WHITE PAPER

A durable outbox for attendance data.

How every attendance event reaches SAP exactly once, unattended, across five remote sites — and why the record has to be written down before the network is ever touched.

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

Abstract

An energy and mining group captures workforce attendance on biometric access-control devices at five project sites. Every event has to reach SAP, because SAP is the system of record that drives payroll.

That data reached SAP through four scheduled jobs owned by the access-control vendor, running on that vendor's servers at each location. When the business decided to decommission the vendor, every piece of SAP connectivity was going to disappear with them — at every site, silently, with no single person obviously responsible for noticing.

This paper describes the replacement: a self-installing Windows service built around a durable outbox, in which every event is committed to a local ledger before any network call is attempted. It covers the outbox model, the retry and dead-letter design, the distinction between an accidental retry and a deliberate re-run, and how undocumented business rules were recovered by running the old and new systems in parallel.

3,000+
employees covered across five project sites

4
vendor-owned SAP jobs replaced by one artifact

100%
of SAP calls stored with request, response and timing

SECTION 01

A payroll pipeline with a retirement date

Across five project sites, more than three thousand employees badge in and out on biometric access-control devices. Each generates a couple of events a day, and every one has to reach SAP.

Attendance was reaching SAP through four scheduled jobs owned by the access-control vendor: a punch push, an employee pull, a shift pull, and a device health check. When a newer device fleet arrived, an in-house shim copied its events into the old vendor's database so those jobs could carry on feeding SAP.

The business then decided to decommission the vendor entirely, for sound commercial reasons. That decision would have taken every piece of SAP connectivity down with it.

Why the old pipeline could not simply be rebuilt as it was

Reproducing the retiring jobs would have carried four properties forward that were already causing quiet damage.

- **Failures were silent.** A failed push into SAP — a network glitch, an SAP outage, an expired credential — meant those records were simply never sent. No retry queue, no failure list, no alert.
- **Nothing tracked the state of a record.** No individual event was ever marked delivered, so when a job did not run, nobody found out until payroll noticed a discrepancy at the end of the cycle.
- **Configuration lived in code.** Device identifiers, endpoints, filters and credentials were compiled in and redeployed by hand at every location.

- **The business rules were undocumented.** Night-shift attribution and entry-exit classification existed only inside legacy scripts, understood by nobody currently employed.

The cost of all this was not primarily engineering time. It was that attendance disputes had no authoritative record to appeal to. When an employee contested a figure, there was no way to reconstruct what the system had done, or why.

SECTION 02

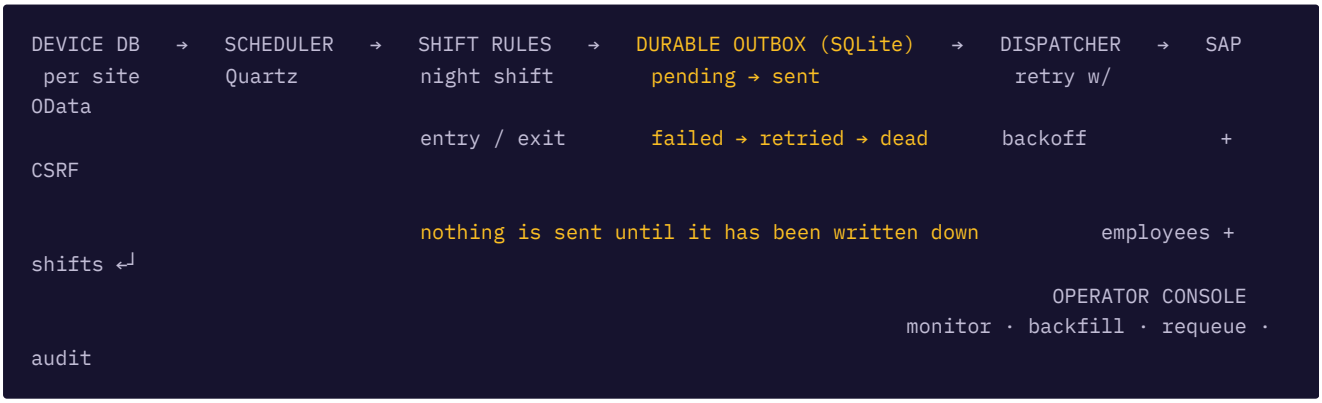
Constraints

- **Payroll could not be interrupted.** The changeover had to happen while the business ran, against a decommissioning date it did not control.
- **The devices could not change.** Whatever was built had to read the existing device databases exactly as they were, with no modification at the edge.
- **Sites are remote; connectivity is not guaranteed.** Power interruptions and network partitions at a project site are ordinary events, not exceptional ones.
- **SAP is the system of record and is not negotiable.** Delivery goes through its OData interface, with its authentication and CSRF requirements, on its terms.
- **The HR-operations team would run it.** Not engineers. Monitoring, backfill and correction had to be usable by the people who own the payroll process.

SECTION 03

Architecture: write it down before you send it

The replacement is a single Kotlin service that installs itself as a Windows service, with the Java runtime bundled, and takes over every SAP-facing responsibility on the day it goes live. The devices carry on capturing events exactly as before. Nothing changes at the edge.



The outbox comes before the network

Every event the service reads is written into a local SQLite ledger *before* any attempt is made to send it anywhere. Dispatch is a separate step that reads from that ledger, marks the state of each record individually, and never re-sends anything SAP has already acknowledged.

This single property is what makes the rest of the system tractable. A power cut at a remote site, an SAP outage in the middle of a batch, or a service restart during patching costs nothing, because the record was durable before the risky part began. The next run resumes exactly where the last one stopped.

SQLite was chosen deliberately over a server database. The outbox has to survive a power cut at a remote site with no database administrator present — a requirement an embedded, file-backed store meets and a networked one does not.

Retries, and then a dead-letter queue

Transient failures retry automatically with exponential backoff, which absorbs the overwhelming majority of what goes wrong in practice. A failure that is genuinely permanent lands in a dead-letter state rather than being retried indefinitely or dropped, and an operator can requeue it from the console once the underlying cause is fixed.

The distinction matters. A queue that retries forever hides a permanent fault behind an endless stream of noise. A queue that drops permanent failures loses records. A dead-letter state does neither: the record is safe, the failure is visible, and a human decides.

A deliberate re-run is not an accidental retry

Every event is keyed so that a repeat delivery is recognised and discarded. Taken alone, that would make the system safe and useless — because there are legitimate reasons to send a record again. A shift schedule is corrected retroactively. A classification rule changes. A backfill is needed after an outage.

So the service distinguishes between the two. An **accidental retry** is deduplicated away. A **deliberate re-run**, initiated by an operator, is honoured — and audited, with who asked for it, when, and against which records.

Systems that do not draw this distinction end up with an unofficial procedure involving direct database edits, which is precisely the thing an audit trail exists to prevent.

Master data flows back from SAP

Employees and shift schedules are pulled from SAP on their own schedule rather than maintained separately. This removes an entire category of reconciliation problem: the service works from the same definition of an employee and a shift that payroll does.

SECTION 04

Recovering the business rules

The rules buried in the legacy scripts were the real risk in this project, and they could not be recovered by reading the old code. Code is a statement of what happens, not of what was intended, and a faithful reimplementation of a misunderstanding is still a misunderstanding.

They were instead reconstructed and reimplemented test-first, and then validated by running the two systems side by side against live data — treating every disagreement as a defect in our understanding rather than as noise. Most of them turned out to be exactly that.

RULE	BEHAVIOUR
Night-shift attribution	An event at 02:00 belongs to the previous day's shift, not the one that has just begun
Duplicate-scan suppression	Consecutive scans within a short window are one event, not several
Entry / exit classification	Entries, exits and breaks are classified, not recorded as undifferentiated timestamps
Location allowlist	Each site may report only the locations it is permitted to

None of this is glamorous, and all of it is the reason the figures reaching payroll are correct.

SECTION 05

The operator console

The old pipeline's only interface was a log file nobody read. The service ships with a console built for the HR-operations team rather than for engineers — because the people who field attendance disputes are the people who must be able to answer them.

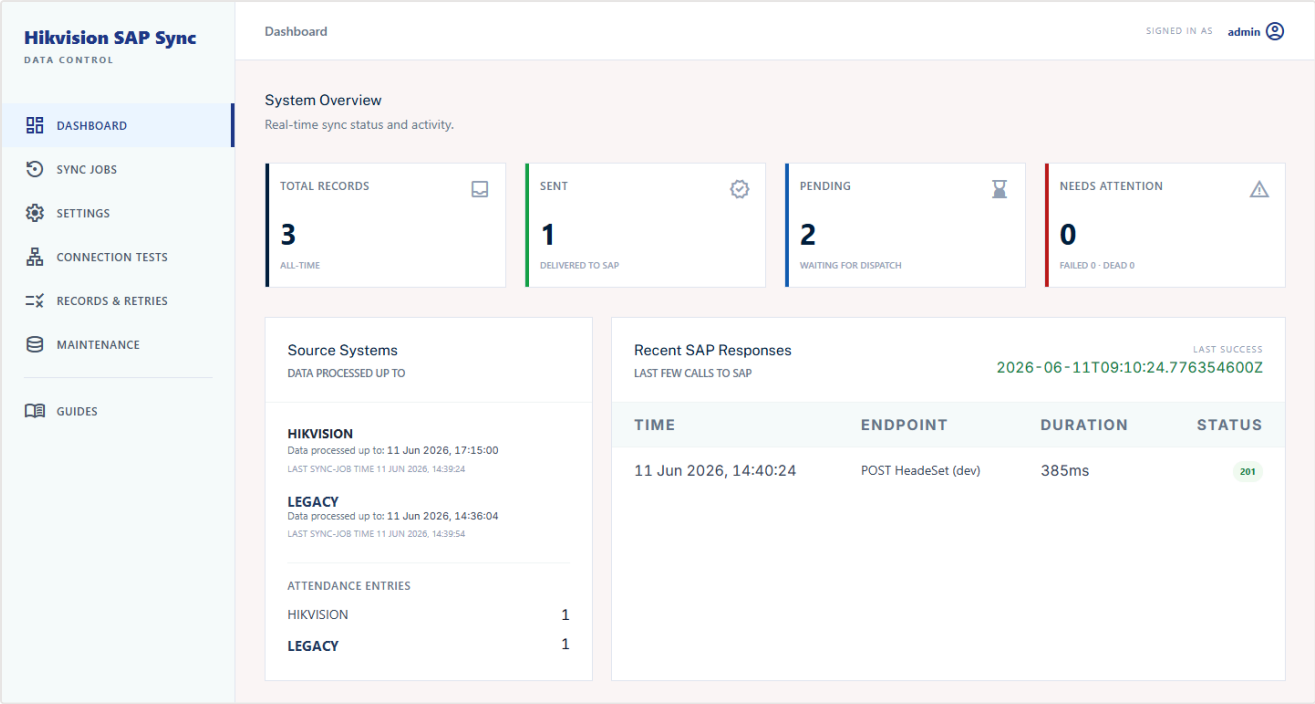


Figure 1. System overview — live counts of records ingested, delivered, pending and needing attention; per-source sync watermarks; and the most recent SAP responses with status and duration. Screenshot from an instance running against test databases; no customer data is shown.

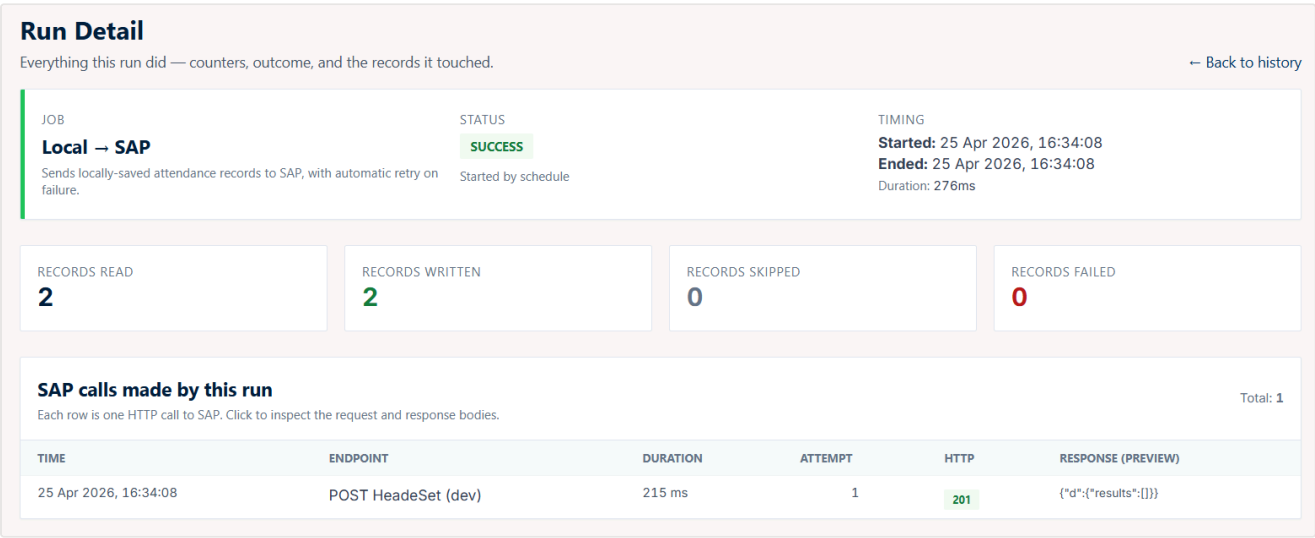


Figure 2. Run detail — every scheduled run records what it read, wrote, skipped and failed, and every SAP call it made, down to request duration and response body. A failure is reproducible from the record alone.

Beyond monitoring, the console gives operators the ability to rewind a source's sync watermark, backfill a date range after an outage, and browse ingested data read-only. Every one of those actions is audited. This is what makes intervention safe enough to permit — and an organization that cannot audit its interventions is eventually forced to forbid them, which leaves the operations team powerless in exactly the situations where they have the most context.

SECTION 06

Rollout: parallel run, then cut over on evidence

A replacement for a payroll pipeline cannot be validated by testing alone. It has to be proven against the system it replaces, on live data, before anybody is asked to trust it.

The new service was therefore brought up alongside the legacy jobs and run in parallel, reading the same devices and producing the same deliveries, with the two sets of results compared. That period is what recovered the undocumented business rules accurately, because the legacy system was still present to disagree with us.

Only once the two agreed was the legacy pipeline switched off — site by site rather than all at once. The parallel run continues at the customer's discretion, which costs almost nothing and means the decision to fully retire the old jobs is theirs to take on evidence rather than on a date.

The console was built early rather than last, because the HR-operations team needed to see what the new service was doing during the parallel run in order to develop any confidence in it at all.

SECTION 07

Configuration instead of redeployment

Schedules, endpoints, credentials, batch sizes and location codes live in typed configuration rather than in the source. One artifact is deployed to all five sites, and each is configured rather than rebuilt.

The same design means the group's planned consolidation onto a single central deployment — replacing the five distributed ones — is a configuration change rather than a second project.

Service	Kotlin, Spring Boot, self-installing Windows service, JRE bundled
Scheduling	Quartz
Outbox	SQLite, embedded and file-backed
Target	SAP OData, with CSRF
Console	HTMX

SECTION 08

Results

The service runs unattended across five project sites, covering more than three thousand employees, and has replaced four vendor-owned SAP jobs with a single deployable artifact. The vendor was decommissioned on schedule without payroll losing its data supply.

Nothing SAP has acknowledged is delivered twice, and nothing captured is lost, because both properties are structural rather than operational. Every call into SAP is stored with its request, its response and its duration.

The change that the customer notices most is the one that is hardest to put in a metric: attendance disputes now have an authoritative record to appeal to. When payroll asks why a figure looks the way it does, the answer is produced from the console rather than reconstructed from memory.

SECTION 09

Limitations

- **The outbox is per-site.** Each deployment holds its own ledger. This is correct for the distributed phase and is the thing the planned central deployment changes.
- **Delivery is at-least-once with deduplication at the target key, not distributed-transaction exactly-once.** The guarantee is that SAP ends up holding each event once; it is not a two-phase commit, and it does not pretend to be.
- **Master-data freshness is bounded by its pull schedule.** An employee added to SAP minutes ago is not yet known to the service.
- **The device databases are read as they are.** If a device fails to record an event, no downstream system can recover it. The service guarantees delivery of what was captured, not the completeness of the capture.

SECTION 10

Conclusion

The most valuable engineering decision on this project was also the least sophisticated: write the record down before attempting the network call. Everything else — the retries, the dead-letter queue, the backfills, the audit trail, the ability to run in parallel with the system it replaced — is only possible because the durable record exists first.

The second lesson concerns undocumented business rules, which are the real risk in any replacement of a legacy integration. They are not recovered by reading the old code. They are recovered by running the two systems side by side and investigating every disagreement. That period is not a delay in the project. In a replacement of any consequential system, it *is* the project.

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