CASE STUDY · PLATFORM

A data pipeline that gave an airline's teams numbers they could finally trust

A European low-cost airline — Software Development Engineer (AWS)

The airline's downstream teams were making calls on operational data that silently dropped records. Our CTO built a resilient, event-driven pipeline that stopped the data loss and scaled cost with load, giving operations, analytics and reporting numbers they could rely on.

0	38%	3	99.9%
RECORDS LOST (WAS ~2%)	LOWER PIPELINE COST	DOWNSTREAM TEAMS SERVED	ON-TIME DATA DELIVERY

THE CHALLENGE

The airline runs one of the busiest flight operations in Europe, generating a constant, high-volume stream of data. The teams that depend on it, from operations to analytics to reporting, can only be as good as the data they receive, and the business problem was trust: records were being silently dropped when something upstream misbehaved, so the numbers people planned on couldn't be relied on. The need was a pipeline that landed data where it should, on time, with failures handled rather than lost.

WHAT I DID

- Event-driven, serverless ingestion. Data arrived and was processed through a serverless pipeline, so throughput scaled with load and cost tracked usage rather than idle capacity.
- Orchestration with Step Functions. Multi-stage flows were modelled as state machines so each step was observable, retryable and recoverable, and a failure in one stage didn't take down the run or lose data.
- Durable staging on S3. S3 as the durable backbone for raw and transformed data, with clear zones so reprocessing was always possible from a known-good source.
- Operational visibility. Metrics, logging and alarms so the team saw the pipeline's health and got paged on real problems, not noise.

OUTCOME

The pipeline delivered operational data reliably at airline scale, so downstream teams finally had numbers they could trust. Records that once vanished were now quarantined and recoverable rather than silently dropped. Its serverless, event-driven design absorbed variable load without over-provisioning, so infrastructure cost tracked demand instead of paying for idle servers in the quiet hours.

STACK