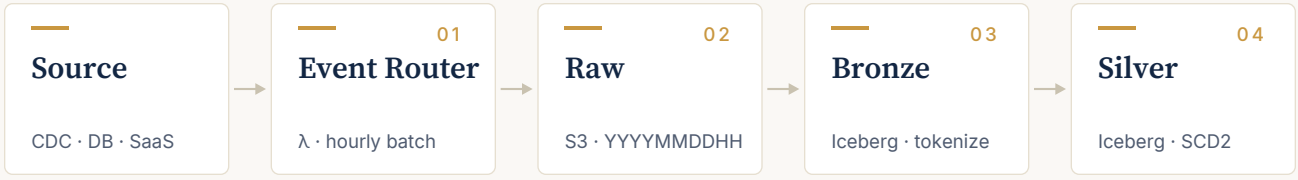
INTELEDYNE · FIELD BRIEF

A field guide to the lean data lake.

Assembling **S3**, **Lambda**, **EMR**, and **Iceberg** into a production-grade lake — and the four things you have to get right.

Many teams reach for a managed lakehouse because they assume the alternative is operationally too heavy. In our experience running production lakes at scale, the opposite is true. A small set of primitives, assembled with the right discipline, produces a leaner, cheaper, more debuggable lake than any managed platform — and it leaves every AI exit ramp wide open. The rest of this brief is a quick tour of what we have learned makes that work.

THE PIPELINE, AT A GLANCE



The **YYYYMMDDHH** partition stamped at stage **02** flows through every downstream table. Every row, in every layer, knows the hour it was born — so lineage, replay, and audit are free.

Primitives over platforms

Every layer is a service you already pay for. No proprietary runtime.

Lineage as a column

An hourly partition key carried Bronze → Silver beats any catalog tool.

AI on top, not bolted on

Bedrock agents read the same manifest, metrics, and catalog operators use.

Robin Tanner · Fractional Solution Architect · Inteledyne LLC robin.tanner@inteledyne.com · inteledyne.com

WHAT WE HAVE SEEN WORK

Four moves that make a lake actually work.

Every production lake we have built or rescued has the same four jobs to do well. Most teams get one or two right. The combination of all four is what separates a reliable platform from one that pages you on Sundays.

01

Slow the firehose before Spark touches it.

CDC streams arrive in the wrong shape for Spark — small, frequent, unpartitioned. A tiny staging Lambda (the 'event router') micro-batches arrivals into hourly windows and lands them at `s3://.../raw/{table}/{YYYYMMDDHH}/`. The partition ID is the lineage key for everything downstream — every Bronze and Silver row carries the hour it was born. Auditors love it. Pipeline operators love it more.

02

One Spark application per list of tables.

The most expensive Spark mistake is granularity. A separate application per table burns JVM startup cost on every run and turns a 30-table backfill into a six-hour parade. The fix is mundane: a coordinator that chunks tables (~25 per chunk works well), and a single Spark driver that iterates the list. For backfills, batch many partitions inside one session — we have measured **11× speed-ups** (54 min → 4.9). Serverless or EMR-on-EC2: the lesson is the same.

03

Gate concurrency on real cluster capacity.

Two `spark-submits` launched at the same time will starve each other on an undersized cluster. The fix is not 'add more nodes.' It is: poll YARN's `/ws/v1/cluster/metrics` for free memory and vCPU before launching the next chunk — and wait with a real timeout (eight hours and a failure email beats a silent stuck job every time). This single rule turns flaky pipelines into reliable ones.

04

Chain Bronze → Silver as the next step, not the next platform.

Silver is not a new product. It is the next state in the Step Function. Same cluster, same coordinator, same partition key flowing through. SCD Type 2 merges run as PySpark `MERGE INTO` against Iceberg, keyed on the source's monotonic change sequence — use the database's SCN or LSN, never the CDC tool's wall-clock timestamp (it is not unique and will trip a `MERGE_CARDINALITY_VIOLATION`). One platform, one governance model, one mental model.

ICEBERG IN PRACTICE

Three details that earn their keep.

Iceberg gives you ACID transactions, schema evolution, time travel, and hidden partitioning. Most teams use one. The other three only pay off when you wire them into the jobs themselves. Three patterns we recommend on every build:

01

Bake schema-drift handling into the jobs.

Source schemas change — someone adds a column, renames one, or quietly changes a type, and forgets to tell you. The robust response: a Bronze job that diffs the incoming schema against the current Iceberg table on every run, then applies the safe additive changes (`ALTER TABLE…ADD COLUMN`) before it writes. Silver propagates the same way. Schema evolution in Iceberg is a metadata operation — no data rewrite — so it costs almost nothing, and your pipeline survives a quiet upstream `notes_v2` add.

02

Tokenize PII inside the Bronze job — not ‘later.’

SSNs, payment data, government IDs: if it is PII, it should never land in Bronze in cleartext. The pattern is a tokenization step inside the same Spark application that writes Bronze — sensitive columns are tokenized with a deterministic, vault-backed scheme (so analytics still join on the tokenized value), only the tokenized form lands in Iceberg, and the cleartext source file is removed from the raw bucket in the same workflow. Cleartext lives in S3 for minutes, not weeks.

03

Tune Iceberg for your workload. Do not take the defaults.

The Iceberg defaults are optimized for something — probably not your shape of work. Three knobs pay back fast. **Merge-on-read vs. copy-on-write:** MoR is far cheaper on writes for high-volume CDC (defer compaction off-hours); CoW gives cleaner files for read-heavy gold tables. **Partitioning:** the `event_hour` ingest partition is right for landing, but queryable tables should also carry a business-time partition (date, region, customer cohort) — Iceberg’s hidden partitioning lets you keep both. **Compaction:** weekly `rewrite_data_files` at a ~256 MB target, monthly `expire_snapshots` with 14-day retention. Without these, you will have a million tiny files in six months — and very slow queries.

Each of these is a one-pass decision at build time. Skip them, and you will be revisiting Bronze in eighteen months — under deadline.

WHERE AI ACTUALLY PAYS OFF

Agents are most valuable when the lake is legible.

A lake built on the patterns from the previous page is unusually well-prepared for agentic workflows. Every dataset has a stable partition key, a manifest of its runs, and an audit trail — the substrate Bedrock-grounded agents need to do real work, not demos. Four high-leverage applications:

Catalog-grounded data discovery

A Bedrock Knowledge Base over Glue / Lake Formation metadata, plus dataset descriptions and recent profile statistics. Users ask in natural language; the agent returns governed SQL and the answer. Row- and column-level policy is enforced at the gateway, not in the model — so the same controls that protect a BI tool protect the agent.

Pipeline-operations agent

A failed Step Function used to mean a page. Now an agent can read YARN metrics, the DynamoDB manifest, the Step Function execution graph, and the last 24 hours of similar runs — then propose a fix, restart with corrected parameters, or file a ticket with context attached. Same coordinator, smarter inputs.

Self-documenting datasets

Manifest entries, partition stats, and Iceberg snapshot history are enough for an agent to write — and keep current — a plain-English narrative of every dataset: where it comes from, when it last refreshed, what changed, where it is used. Refreshed on every Silver merge. Your catalog becomes self-documenting.

Semantic data quality

Some defects do not show up in SQL: a free-text column drifting into the wrong language, an ‘instructor comments’ field slowly leaking PII, a notes column going off-topic. An LLM-backed quality check over narrative columns catches them — and writes findings to the same quality table that holds your numeric checks.

The same engineering that makes pipelines reliable is what makes them legible to a model.

A 60-minute architecture read

If any of this lands close to home —

A focused review for teams mid-build or mid-migration: one hour, your team’s questions, no deck, no upsell. You leave with a concrete sequence — what to keep, what to retire, what to add.

robin.tanner@inteledyne.com inteledyne.com