ORCHESTRA WHITEPAPERS

7 CRITICAL BARRIERS TO SELF-HEALING ARCHITECTURE

WHAT STANDS BETWEEN DATA TEAMS AND TRULY AUTONOMOUS DATA PIPELINES

2025

THE DREAM OF SELF-HEALING DATA

For many data engineers, AI in data engineering revolves around one thing: fixing pipelines. An engineer opens Claude Code, pastes some logs, and a pull request appears.

But semantics matter. When people say **"self-healing"** what they really mean is **"self-managing"**. The measure of success in AI is not the quality of manual intervention — it is the *absence* of it. The dream for data teams is a system where pipelines and workflows simply succeed, without any human in the loop at all.

The largest companies in the world are already building agents toward this autonomous architecture — Databricks' Genie Zero Ops being the worst-kept secret in the category. It is a genuinely exciting piece of kit: always running, aware of everything in its catalog, able to test and hotfix data in sandboxes and wait for a human in the loop. Yet it is wanting in many ways — unclear credential management, limited control, and crucially no context of anything outside its own walled garden, and no proper event-based orchestrator.

Three things stand in the way. **Agents require context** — the know-how to tell a transient error from an upstream schema change from a dropped table. **A mindset shift is needed** — "new branch, merge, re-run" is slow and decidedly un-agentic. And **data does not branch well** — "Git for data" was promised for years and is still not mainstream.

In this paper we cover the **seven barriers** between the typical data stack of today and the nirvana of autonomous, self-healing pipelines.

1 Context and failure recall
2 Elastic infrastructure
3 Operational agents & quality data
4 Git for data
5 Pervasion through the industry
6 Agent sandboxes & new orchestrators
7 Standards for proxy servers

THE THESIS

Collectively, solving these seven barriers lets data teams build a **single pane of glass for AI** — where agents execute when they need to, with the context, infrastructure and guardrails to do it safely.

CONTEXT & FAILURE RECALL

Being able to fix pipelines at all is the baseline requirement for any AI system. And fixing them requires knowledge of the system.

Pipelines fail for a plethora of reasons, which fall into four broad types: **infrastructure**, **code**, **data**, and **transient / third-party** issues. The catch is that the knowledge to resolve each often lives in a single person's head.

Acme's Kubernetes cluster may only be reachable by Bob, the one person with the special access key hidden in Secrets Manager under a non-standard header. Analyst Sophie may know to quietly gloss over the fact that sales are reported in multiple currencies. Someone may know the flaky internal API only succeeds if retried between 2:47am and 3:12am. Ridiculous examples — but they make the point.

Gathering lineage, logs, code and documentation is imperative — and AI is genuinely good at working things out from them. But metadata context alone is not enough. As data folks, we've all thought: *"how on earth could I have known that?"* At the end of the day, only humans know where the bodies are buried.

HOW ORCHESTRA HELPS

Orchestra is the **single pane of glass** over the entire data estate. Every run carries its **lineage, logs, code, parameters and full failure history** in one place — the metadata context an agent needs to diagnose a failure, plus the human-authored runbooks and annotations that capture the knowledge metadata can't.

ELASTIC INFRASTRUCTURE

Considering infrastructure failures specifically, AI needs more than scale — it needs an API to manage that scale. We call this **“elastic” infrastructure**.

Elastic infrastructure does two things: it scales, *and* it exposes an interface to be managed programmatically. A lone EC2 instance is not elastic — it does not scale beyond a point. A Kubernetes cluster on a locked-down machine is not elastic with respect to the cloud — there is no API to manage it.

The reason this matters is simple: **to recover a failure, AI must be able to act on the infrastructure that produced it.** If the platform offers no programmatic surface, the agent is left looking at a wall.

SaaS providers should relish this. By taking the management burden of infrastructure away from data teams — for a fee — they are inherently AI-friendly. (Though this collides with Barrier 6, which we’ll come to.)

SCALES ON DEMAND

Capacity flexes with the workload, not the other way around.

API TO MANAGE

A programmatic surface an agent can actually act on.

RECOVERABLE

Failures can be replayed and resolved automatically.

HOW ORCHESTRA HELPS

Orchestra is a **serverless control plane** that triggers and monitors elastic services through their APIs — ingestion, transformation, warehouses and AI jobs alike. It gives agents a single, governed surface to **re-run, re-parameterise and recover** work across the stack.

OPERATIONAL AGENTS & QUALITY DATA

Pete in Finance has overwritten the US planning sheet again. There are zero rows in `us_forecast_dec_v1` and the aggregate is stale. AI reports the connectors are fine — there was simply no data. It can do nothing.

So what is the solution? A quick quiz — pick the right answer:

1. Let AI hallucinate the forecasts.
2. Let AI hallucinate them in the warehouse, then re-run the sheet pipeline later.
3. AI tells Pete to upload the damn forecasts.
4. A warm pool of rented humans is dispatched to bother Pete in person until he fixes it himself.

Of course, there is no right answer — the options range from bad to ludicrous (and option 4 isn't AI at all, it's teamwork). **Quality data is, as ever, the most important thing for a data engineer.** Teams should ask it in interviews more often: "how good is your data?"

That said, operational agents do have a place. A genuine fat-finger error — a $10m deal that should read $1m — could be corrected by an agent with a Salesforce API key, which then restarts the pipeline.

HOW ORCHESTRA HELPS

Orchestra pairs **data-quality monitoring** and **alerting** with the ability to **run agents**. It catches the empty table and the stale aggregate, routes the human-shaped problems to Slack, Teams or email, and lets governed operational agents fix the genuinely fixable — then re-run the affected assets.

GIT FOR DATA

The fat-finger example raises a deeper question: **should AI agents edit production?**

If you've juggled multiple Salesforce environments, you feel the pain — but you see the point. Suppose the AE really did land a $10m whale. Far better for the agent to edit a **staging** instance, verify the pipeline against it, and switch the corrected data in — than to write straight to production.

An AI agent branches the pipeline, clones the data zero-copy, runs and overrides values, then merges and clones back — replacing the slow manual PR process.

In every case, AI needs a new branch to work on. That branch needs **zero-copy clones** of the data, a **Git-for-data** approach, and an efficient way to "switch in" the result at the end. Without it, trusting AI to fix things reliably becomes a governance nightmare of production write access.

Snowflake is well-positioned here, having supported zero-copy cloning for years; MotherDuck too. But the clearest winner is **Iceberg** — time travel, rollback and branching built in.

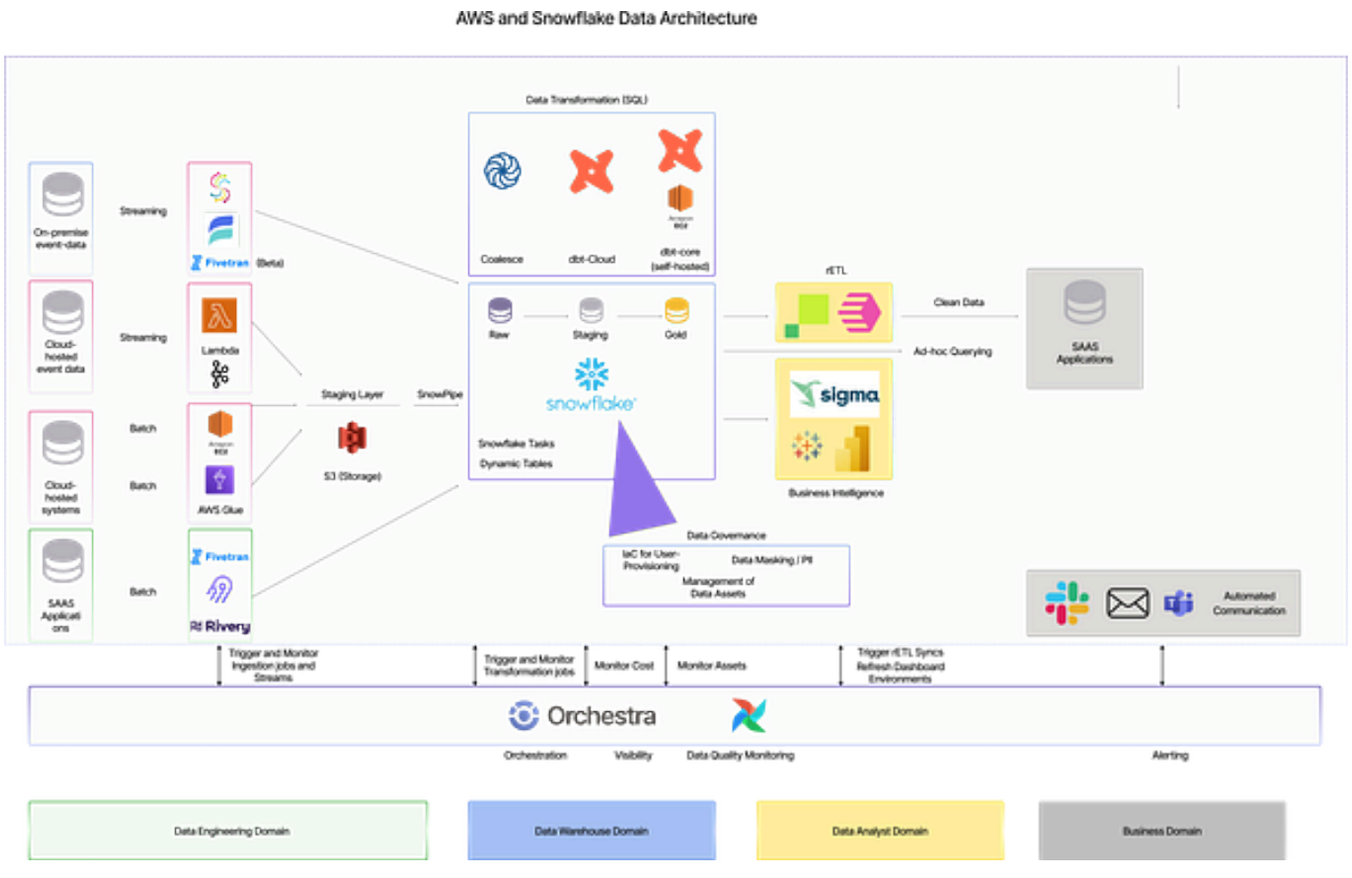
HOW ORCHESTRA HELPS

Orchestra **orchestrates the branch → clone → run → verify → switch-in workflow** end-to-end, on top of the zero-copy primitives in Snowflake, MotherDuck and Iceberg — so an agent can safely fix data on a branch and you can trust what lands in production.

PERVASION THROUGH THE INDUSTRY

Self-healing architecture hits a wall the moment we talk about interoperability.

Fivetran and dbt made a big noise about **open data infrastructure** in 2025. It is not the same as open-*source* infrastructure — it describes what is better called the **modular data architecture**, where different functions get different best-of-breed tools.



A modular data architecture — ingestion, transformation, warehouse, BI and reverse-ETL — with Orchestra as the orchestration, visibility and quality layer beneath it all.

There is no point in a self-healing architecture if the underlying components don't support it. Suppose an ELT provider has a silent failure: the columns and types hold, but the *values* drift — now reporting in Yen as well as USD. The right fix is to amend the ELT job in staging, verify downstream, switch the corrected data in, then cut over the bad job. **Many ELT tools simply don't expose the APIs to do this** — pressuring today's players to change, or die.

HOW ORCHESTRA HELPS

Orchestra is the **orchestration layer across the modular stack** — Fivetran, dbt, Snowflake, Sigma and more — triggering and monitoring each component from one place. It is the interoperability glue that lets self-healing patterns span tools instead of stopping at a vendor boundary.

AGENT SANDBOXES & NEW ORCHESTRATORS

The logical place to run agents that fix things is **inside an orchestration tool** — because the orchestrator already has what the agent needs.

- The ability to run any code, and to replay any DAG with arbitrary parameters.
- Connections to every part of the system — an orchestrator orchestrates, so it has access.
- Alerting, monitoring, recovery and scalable infrastructure, built in.

But there is one enormous problem: **security.** Companies like Cloudflare have built dedicated agent sandboxes precisely because capable models can break out — especially under prompt-injection attack. Running an agent in the same infrastructure as your legacy orchestrator is dangerous, and AI workloads could trample data ones.

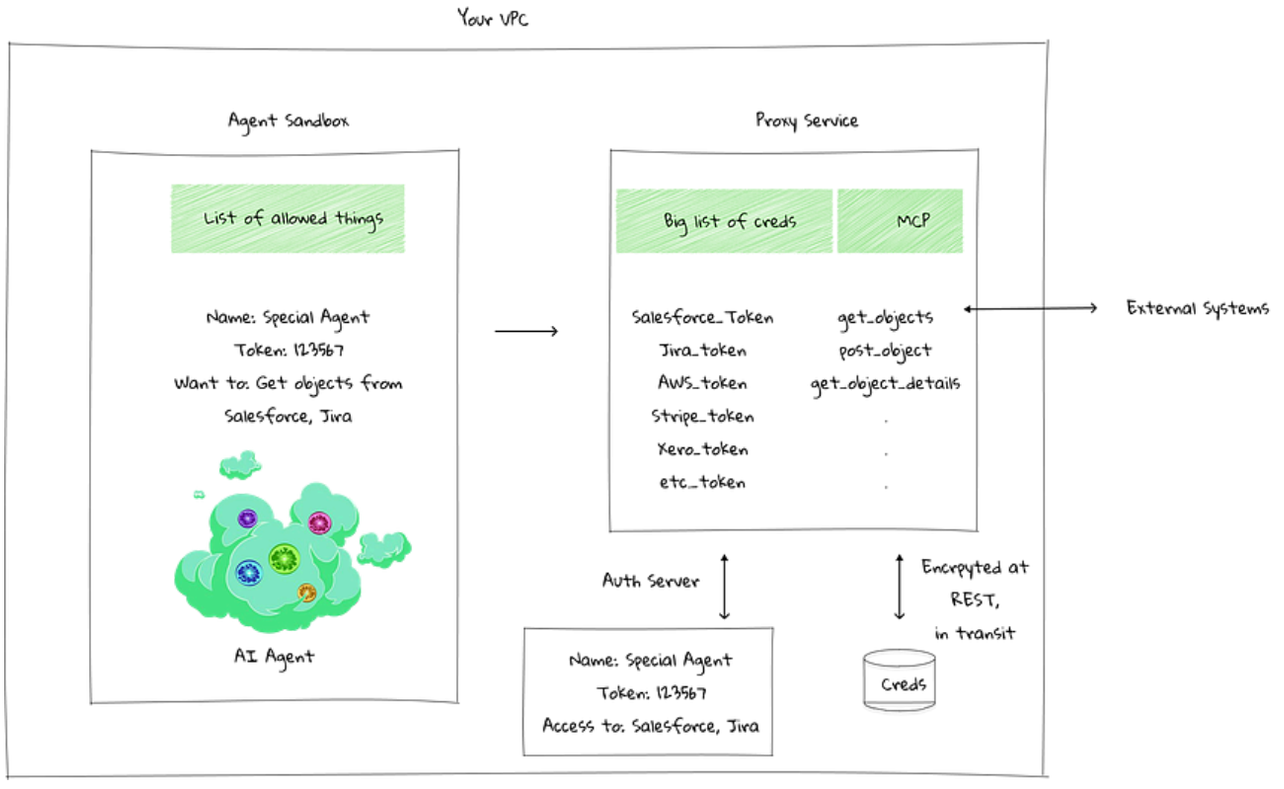
Legacy orchestration tools were simply not built to host agents this way. It is clear agents will need access to orchestration frameworks — whether that means the model providers ship an orchestrator, new-age orchestrators add agent sandboxes, or the two interoperate. Something has to give. Because: security.

HOW ORCHESTRA HELPS

Orchestra is the **new-age orchestrator** — built to run agents in **isolated sandboxes**, separate from your data workloads, with the connections, replay and alerting an agent needs but without the blast radius of bolting AI onto a legacy scheduler.

STANDARDS FOR PROXY SERVERS & AGENT DEFINITION

One answer to security is a **proxy service** for agents. Rather than installing secrets in the sandbox, the agent is given access to a defined set of tools and MCPs.



A basic proxy service: the agent declares what it wants; an auth server brokers access; the proxy holds the credentials and is the only thing that touches external systems.

The proxy becomes the only component with access to external systems. So even if the agent falls victim to prompt injection, the damage is bounded by the endpoints in the MCPs it was granted — nothing more.

What this proxy should look like is not obvious. MCP is big; Cloudflare released Code Mode; configuring servers across many endpoints is genuinely hard. **Open standards should prevail** — any agent talking to multiple systems benefits, security-wise, from a proxy. They exist today, but mostly inside private SaaS. Frameworks for *designing* agents must emerge too: a single agent wired to hundreds of MCPs may simply carry too much context to be feasible.

HOW ORCHESTRA HELPS

Orchestra keeps **secrets in the control plane**, not the sandbox — agents reach external systems only through governed, auditable endpoints. It is the proxy-and-credential model these barriers demand, built on open standards rather than a walled garden.

A SINGLE PANE OF GLASS FOR AI

Solve the seven barriers together and data teams can build a single pane of glass for AI — where agents execute when they need to, with the context and guardrails to do it safely.

1 **Context** — the information to solve any problem.

2 **Elastic infrastructure** — the foundation for fixing pipelines.

3 **Quality data** — eradicates the human side of inputs.

4 **Git for data** — creates reliability and trust.

5 **Mass adoption** — prevents industry collapse.

6 **Sandboxes & new orchestrators** — remove legacy architecture.

7 **Proxy servers** — do their best to guarantee security.

Core data primitives — Git for data, elastic infrastructure, support throughout the ecosystem — turn this from a theoretical idea into a practical reality. Teams pursuing autonomous architecture will impose real pressure on existing vendors to support interoperability, accelerating consolidation as the walled gardens of Salesforce, SAP and ServiceNow roll out their own agentic studios.