ACCELERATE DATA

CATEGORY REFERENCE · WHITEPAPER

Markers of an AI-Agent-Native Data Platform

The complete capability set a data platform needs before agents can be trusted with production data engineering — and how Vibedata delivers it. We call a platform's completeness against this set its **Autonomous Data Engineering Readiness (ADER)**.

INTRODUCTION

A coding agent specialized for data engineering

Vibedata is a coding agent specialized for data engineering — and the world around it. That world is a **harness**: it gives the agents the four things a general coding agent lacks for data work — **isolation**, so being wrong is survivable; **guardrails** scoped to the data platform rather than to a generic external API; **context** that is data engineering rather than application code; and **cross-platform** reach. Architecturally the harness is the agentic coordination layer above the data platform: it coordinates above the platform rather than replacing it.

MISSION

Vibedata is a coding agent specialized for data engineering workloads. Three agents carry the lifecycle inside a harness that gives them the four things a general coding agent lacks: **isolation**, so being wrong is survivable; **guardrails** scoped to the data platform rather than to a generic external API; **context** that is data engineering rather than application code; and **cross-platform** reach. That is what lets them build and maintain data products safely on whatever platform the team already runs.

VISION

Every data team ships with the production discipline that today takes deep engineering expertise to sustain — specs, tests, isolation, and CI on every change — at vibe-coding speed.

Agents can already write SQL. What they cannot yet be trusted to do is change production data safely. A wrong code change throws in seconds; a wrong data change fails silently — a bad join drops rows and the dashboard still renders green. Generic coding agents have no way to prove a change is correct against real data, so the last mile of data engineering stays manual.

Vibedata closes that gap. It runs the full lifecycle from a business request through production operations, on the customer's own platform and inside the customer's own perimeter. The model underneath is a commodity dependency; the harness around it is the product.

This whitepaper defines what a platform must provide before agents can be trusted with that work. We call the complete set of capabilities the **markers** of an AI-agent-native data platform, and a platform's completeness against them **Autonomous Data Engineering Readiness (ADER)**. The pages that follow name each marker, say why an agent needs it, show where it exists today, and describe how Vibedata delivers it.

THE INDEX

What ADER measures

A data platform needs a complete set of capabilities — its **markers** — to be AI-agent-native. We call a platform's completeness against them its **Autonomous Data Engineering Readiness (ADER)**, scored 0 to 1.0. **ADER 1.0** is the full set.

This document names the markers, says why an AI agent needs each one, points to where each is done well today (if anywhere), and describes how Vibedata provides it. It is the platform counterpart to the AI-Native Data Engineer persona — the practitioner who directs the work. The markers describe what that practitioner's platform must do for agents to be trusted with the work.

The thesis

A data platform is AI-agent-native only when it provides every marker in this framework — as a combination, for data, on one primitive. That single claim has four parts:

01 **ADER 1.0 is the full set.** Most platforms provide a strong subset, and many individual markers are now available in pieces. The argument is the assembly, not any single part.

02 **Mature platforms fall short for the same two reasons everywhere** — no deterministic quality gates bound to the agent's loop (M7), and no single primitive that runs build and operate under one native agent (M4). They stall not on capability but on priority: their business is platform consumption, so the coordination layer is a feature that drives usage, never the whole product, and never cross-platform.

03 **Generic coding agents and meta-harnesses cover a different subset** — code and compute isolation, spec-first planning, and gated CI when paired with dbt or SQLMesh. They lack the data-specific combination: per-agent data-platform isolation, verification bound to the loop, knowledge that compounds from incidents, and coverage of both build and operate.

04 **Vibedata designs for ADER 1.0 — one platform at a time, on native tooling.** Coordination, skills, and knowledge are portable across platforms; the isolation, compute, and verification primitives are native per Adaptor — Fabric ephemeral workspaces and DuckLake today, and on the roadmap Snowflake, then the Postgres Adaptor, Databricks, and BigQuery. Each platform's best version of every marker, not a watered-down common abstraction.

STRUCTURE

The four pillars

The ten markers group under four pillars. We borrow the vocabulary Databricks uses for agentic-era governance — **Control · Context · Choice** — and sharpen it. Where that framing asks what an agent may access, we frame each pillar as what an agent does; and we add a fourth, **Cost**, that an access-governance framing leaves out. Each marker belongs to exactly one pillar.

Context · M1–M4

Agents are only as good as their context — not a metadata layer served on request, but context that persists and compounds: a durable spec (M1), knowledge that accrues from every incident (M2), observability, lineage and documentation of the current state (M3), and one Intent that carries across build and operate (M4).

Why data is different: a coding agent grounds on the codebase; a data agent must ground on the data's meaning — grain, semantics, and source quirks that live in tribal knowledge, not the code. Syntactically perfect SQL can still be wrong because of what the data actually contains.

Control · M5–M8

Govern what agents do, not just what they can reach: disposable three-level isolation (M5), verification against real data (M6), in-loop gates and a deploy lock the agent cannot talk around (M7), and governed access that keeps data inside the perimeter (M8). The gates verify by independent execution — tests run, data diffs computed on real data — and the gate's result is what decides the work is done.

Why data is different: in software a wrong change throws in seconds; in data it fails silently — a bad join drops 10% of rows and the dashboard still renders green. "It ran" ≠ "it's correct", and you cannot `git revert` two billion overwritten rows.

Choice · M9

Choice not bounded by one vendor's platform: any model, any cloud, native across platforms, reachable from any channel, with one accountable driver at a time (M9). Where an incumbent's "choice" stops at the edge of its own platform, ours is cross-platform by construction.

Why data is different: code is portable; data has gravity — trapped in a warehouse with its own SQL dialect, governance model, and bill, exactly where lock-in lives. The coordination layer must ride above each platform's native primitives so skills and knowledge survive a move.

Cost · M10

Spend made visible and capped: per-run attribution down to the turn, budgets and caps at the policy point, and ~zero marginal cost per pipeline (M10). The pillar an access-governance framing omits, and the one agents make unavoidable.

Why data is different: a coding agent's runaway loop burns tokens; a data agent's re-scans a 2 TB table on a metered warehouse every scheduled run — a five-figure surprise by Monday. Compute is the dominant, variable, unattended cost.

How to read the marker tables

Each marker breaks into sub-markers. Every row has four fields: the **Marker** (the capability at the leaf level); **Why AI agents need it** (the agent-specific reason — agents execute literally, at machine speed and scale); **State of the art** (who does this well today, named honestly, if anyone — many markers exist in pieces elsewhere, the argument is the combination); and **How Vibedata does it** (the conceptual approach — design intent, not implementation detail).

> **Scope note.** These markers describe batch data engineering. ML model operations — drift detection, retraining, eval-gated traffic ramps — are out of scope. ML serving and streaming are explicit exclusions.

Context

M1 — Durable requirements and spec-first workflow

> An agent forms and verifies a hypothesis against a durable specification. Without one it executes literally and delivers the wrong outcome.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M1.1 Durable requirements capture	The business goal must survive across agent sessions, not live in a transient chat the next session cannot see.	Common in coding agents — durable project context via `CLAUDE.md`/`AGENTS.md` and memory files, plus bidirectional Linear/Jira through MCP — but these hold instructions and standards, not structured data requirements.	The Intent is the durable unit of work; it persists across every agent session, while sessions are transient.
M1.2 Spec before code	A spec the agent reasons against is what lets it verify its own output against intended outcomes.	Standard in coding agents: AWS Kiro (requirements → design → tasks with approval gates), GitHub Spec Kit, and Claude Code plan mode produce a reviewable spec before code. The open gap is a data-specific spec the agent verifies output against.	Each Intent carries a `design.md` — source mapping, model architecture, materialisation, validation — that the agent authors and reasons against before generating code.
M1.3 End-to-end traceability	Every action — requirement → code → test → PR → incident — must be attributable, so an agent or a reviewer can reconstruct why a change exists.	No generic harness closes a single requirement-to-incident chain. Spec Kit and Kiro chain requirement → design → task → PR, and Linear/MCP links issues to code, but the chain stops before production incidents.	Intent, branch, PR, and issue bind to one durable record, auditable end to end.
M1.4 Platform-mechanics grounding	Before authoring a spec or code, the agent must know the warehouse dialect (Snowflake SQL ≠ BigQuery SQL), the exact build and test commands, and the naming conventions — or it produces valid-looking code that will not run or conform.	Coding agents capture this in `CLAUDE.md`/`AGENTS.md`; the data-specific version (dialect, dbt/SQLMesh commands, layer naming) is what Upriver's "Agent Configuration & Context" dimension scores. No platform supplies it as native, per-platform grounding.	Each Adaptor supplies its platform's mechanics — dialect, native CI and test commands, naming conventions — as durable context the agent grounds against, so it is correct per platform without a hand-written rules file.

M2 — Accumulating, portable knowledge

> Source quirks, house rules, and incident fixes must compound and outlive any session — and stay with the team across model and platform changes.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M2.1 Build loop	Source quirks and validation patterns captured once as durable skills, versioned for every future agent run.	Partial and mostly manual: dbt docs, the semantic layer, and data contracts persist domain knowledge, but nothing open captures source quirks as reusable agent skills automatically.	Source quirks captured as durable skills in the Domain's knowledge layer.
M2.2 Deploy loop	A best practice codified into a rule the CI gates apply automatically on every PR.	No tool turns a one-off into an automatically-applied PR-time rule learned from data work; CI checks must be hand-added.	Best practices become house rules the CI gates apply automatically at PR time.
M2.3 Operate loop	A production incident converted into a new test or guardrail the next agent inherits.	Closest today: Monte Carlo's Operations and Monitoring Agents recommend monitors from incidents and deploy them one-click. Snowflake/CoCo skills are intentionally static and version-controlled — they do not self-learn from runs.	Incidents are converted into new tests and guardrails the next agent on that pipeline inherits.

M3 — Observability, lineage, and documentation

> An agent needs to see what it did, what the data did, what is connected, what the existing models mean, and how the code is structured — to understand the current state before it changes it or diagnoses anything.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M3.1 Platform / agent stack	Traces of what the agent did — planning, tool calls, SQL — so a failed run is debuggable.	Now native on mature platforms: Snowflake Cortex AI Observability (GA) logs full agent traces with LLM-as-judge evals; Databricks traces agent work similarly. No longer a gap on the incumbents.	OpenTelemetry and OpenInference traces keyed to Domain, Intent, Session, and Turn; paired with build-time runtime-audit records keyed to the `git_sha` that produced each table, so the operate loop can trace a symptom back to the exact code that built it.
M3.2 Data stack	Metrics of what the data did — silent failures show up here before they throw errors.	Native on Databricks (Unity Catalog data-quality monitoring) and on Snowflake (DMFs, Cortex anomaly detection); the independent layer is Elementary, Monte Carlo, Anomalo, plus dbt tests.	Elementary surfaces data-quality signals from dbt runs.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M3.3 Dependency lineage	The dependency graph for root-cause and impact analysis — "trace a failure to a schema change three tables upstream."	Databricks Unity Catalog is the strongest example — automatic table- and column-level lineage across the metastore under one permission model; a genuine incumbent advantage.	Consumes the platform's native lineage rather than rebuilding it.
M3.4 Documentation of current state	An agent must understand what the existing models mean and how they are shaped — grain, semantics, business definitions — before it can safely change them; a description that restates the name tells it nothing.	dbt docs and the dbt Semantic Layer (MetricFlow) are the open standard; Databricks ships AI-generated comments, Unity Catalog Semantics / Metric Views, and Genie Ontology; Snowflake has semantic models.	dbt docs, a ubiquitous-terms glossary, and design docs maintained in the Domain's repo and kept current by the build and operate loops; the M7 gates reject descriptions that merely restate the name.
M3.5 Legible code and pipeline structure	An agent navigates by layer and name; a flat directory of long, multi-purpose models makes it guess where logic lives and duplicate it. The platform must make models locatable from layer and name alone.	The open convention is dbt's layered project structure (staging → intermediate → marts), `stg_`/`int_`/`fct_`/`dim_` naming, one-transformation-per-model, and named CTEs. Nothing enforces it for data automatically.	The build loop generates and maintains layered, single-purpose models with conventional names and modular per-domain pipelines; the M7.1 hygiene gate rejects structural drift, so the structure holds whether authored by the team or the agent.

M4 — Full lifecycle on one primitive

> The same durable unit of work must drive both the build loop and the operate loop — build optimized for comprehensiveness, operate for speed. An agent that only builds, or only operates, leaves half the lifecycle uncoordinated.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M4.1 Build	Optimized for comprehensiveness: handle every scenario in the calculations and the tests. This is where it is worth spending the time to build it right.	Copilot in Fabric builds notebook code, queries, and reports; coding agents build but without data-platform context. Fabric Data Agents are read-only query. MotherDuck's Flights pair an external agent with its MCP control plane — but the agent is bring-your-own, not hosted. None pair a hosted build agent with the operate loop on the same primitive.	The Build agent drives the build loop in Studio, from a business request to a tested pipeline.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M4.2 Operate	Optimized for speed: restore service as fast as possible, even with a narrow fix, then hand the comprehensive follow-up back to build.	Operate agents shipped widely in mid-2026 (Databricks Genie ZeroOps, Fabric Operations agent GA, Snowflake CoCo Automations, AWS SageMaker Data Agent, BigQuery's Data Engineering Agent); each builds or operates, but **not both on one durable primitive under one native agent**.	The Fix agent drives one loop with three depths — explain → investigate → remediate — to a fast, bounded fix, then hands the comprehensive follow-up back to the build loop; triage and dispatch sit upstream (the Detect agent plus the platform). Both halves on the same Intent primitive.

Both loops also run unattended: **Automation**-mode invocation fires build and operate work on a schedule, and **Sense**-mode on a detected system signal, with no human present — under the governed identity of M8.3–M8.4 and the triggering model of M9.3. Automation and Sense are invocation modes; the surfaces are Studio and other channels.

Control

M5 — Three-level isolation

> A wrong agent hypothesis must always be disposable. If it is not isolated at every level, a wrong action corrupts shared code, shared compute, or the shared data platform.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M5.1 Code isolation	Each agent needs its own branch and filesystem so discarding a failed attempt is one command.	Coding agents do this well — Claude Code has first-class git-worktree isolation per session and per subagent; Codex isolates likewise.	A per-Intent git worktree and filesystem for each agent.
M5.2 Compute isolation	Parallel agents must not interfere with each other or with production while they run.	Done well: Claude Code and Codex enforce OS-level sandboxes; Modal and Daytona host hermetic cloud sandboxes. MotherDuck's hypertenancy gives each user and service account its own isolated Duckling.	Sandboxed containers, one per agent session.
M5.3 Data-platform isolation	An agent must run, validate, and self-correct against real data without touching the live platform.	Databricks does this well: Delta zero-copy shallow clone, per-environment catalogs, and Asset Bundles; Genie ZeroOps auto-orchestrates clone + scoped permissions + network isolation. MotherDuck offers zero-copy clones and Differential Storage snapshots — but no merge step, and the clone is a manual call, not auto-provisioned and bound to the agent's loop.	A per-Intent ephemeral lakehouse workspace — a zero-copy clone in concept, native to the Adaptor in mechanism (an ephemeral Fabric workspace, a DuckLake database) — torn down on merge or close.

M6 — Verification against real data

> Silent data failures do not throw errors. An agent must prove a fix against real data in isolation, not against tidy unit tests. The proof is the run itself — tests and data diffs executed on the isolated copy decide that the work is done.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M6.1 Agent self-verification on real data	The agent's inner loop must close against real data, or "the code runs" is mistaken for "the data is correct."	Strong across the field: Databricks Genie ZeroOps validates against a clone of production data; the open stack uses dbt tests, dbt Cloud "compare changes", SQLMesh table diff, and Datafold Data Diff — now exposed to agents over MCP. MotherDuck adds `try_bind()` for millisecond SQL validation.	Tests and data diff run on the ephemeral dev workspace inside the agent's inner loop.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M6.2 Human verification before production	A human must be able to sign off on output before a production merge.	A gated human step exists in the open stack — SQLMesh requires explicit approval before blue-green promotion; dbt Cloud CI gates merge on a reviewed PR with diffs attached — but it is not bound to an agent's verification loop.	User validation testing (UVT) on an ephemeral stage workspace; a production merge requires it.

M7 — In-loop quality gates and deploy lock

> Quality must be enforced on every change before production — automatically, as deterministic gates the agent cannot skip, not as prompt requests it may ignore.

Vibedata's gates follow the **AWAP** cycle (Audit → Write → Audit → Publish). The point is enforcement bound to the agent's loop: the harness refuses to publish when a gate fails, even when the model would prefer to proceed.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M7.1 Hygiene gate	Catches structural and standards violations before they compound across an agent's many fast iterations.	Gated data CI is standard: dbt Cloud CI, SQLMesh's CI/CD bot, `dbt-checkpoint`, and recce all gate PRs. The open gap is the audit-before-and-after-write (AWAP) cycle bound to the agent's loop.	The AWAP audit step blocks structural and standards violations before write.
M7.2 Quality / test gate	Tests and coverage must pass on every change; at agent scale, manual review is the bottleneck.	dbt Cloud CI runs `dbt build` + tests and blocks merge on failure; SQLMesh auto-runs unit tests in a PR environment. Snowflake WAF is advisory; enforcement lives in skills' validation gates, not the framework.	Test and coverage gates run on every PR via the platform's native CI.
M7.3 Verification gate	Data diff and spec conformance must block a merge that drifts from the spec.	dbt Cloud Advanced CI / "compare changes" adds data-level diffs to the PR; SQLMesh diffs across environments. Gated verification exists; binding it to the agent's spec is the gap.	Data diff and spec-conformance checks gate the merge.
M7.4 Deploy lock	Gates must be enforced deterministically, with one accountable party per unit of work, so the loop is debuggable and the lock cannot be talked around.	Omnigent enforces generic policies "not via prompts", but has no data gates and deliberately favors agent composition over a single accountable identity.	Gates run as hooks that refuse to publish on failure; one agent identity per Intent keeps the loop accountable. Gate crossings are recorded in the Intent's `design.md` Gate Ledger — an append-only, resumable trail of what passed and when.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M7.5 Immutable baseline tests	The cheapest path to a passing gate is to relax or delete the failing test rather than fix the code — and a weakened data check fails silently. Baseline tests must sit outside the agent's write scope.	Upriver names "protected baseline tests" as a readiness criterion; dbt and SQLMesh have no built-in immutability — protection is convention or CODEOWNERS. Binding it to the agent's loop is the gap.	Baseline tests are protected from agent modification; the agent may add tests but cannot weaken or remove existing ones, enforced by the same hooks as the M7.4 deploy lock.
M7.6 Task-adherence / scope lock	An agent left to run drifts past the request — it refactors an unrelated model or expands a narrow fix into a rewrite. At machine speed and unattended, scope drift is how a bounded change becomes an unbounded one.	Coding agents keep a session to-do / plan (Claude Code plan mode, Kiro tasks), but nothing binds the plan to a durable data unit of work as an enforced scope boundary — it is a prompt aid, not a gate.	The session to-do list holds the agent to the Intent's plan, so work stays bounded. Paired with the M7.4 deploy lock, this is the second of two guardrail layers: the deploy lock enforces CI adherence, the to-do list enforces task adherence.

M8 — Governed access; data stays in the perimeter

> The agent must act under scoped, short-lived, audience-bound credentials, and governed raw data must never be copied to an outside tool.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M8.1 Scoped credential brokering	The agent should never hold a raw, long-lived secret; credentials are short-lived, audience-bound, and revealed only at the approved egress.	Omnigent's OS sandbox injects secrets at the egress proxy, so the agent never sees the token. MotherDuck brokers access tokens with a configurable TTL plus read-only tokens, created and revoked programmatically over REST.	An intent-scoped, short-lived access token is brokered per session; the agent never holds a raw secret.
M8.2 Data never leaves the perimeter	Governed production data cannot be freely copied, shared, or handed to an outside agent.	Done well, but only natively: Databricks Genie runs under Unity Catalog permissions; Snowflake Cortex agents inherit the user's role and RBAC on every query, with masking and row-level policies applied.	The coordination layer runs inside the customer's perimeter and reads metadata and code, never rows.
M8.3 Non-human execution identity	A scheduled or system-triggered run has no human present, so the agent cannot borrow a live user's credentials; it must act under a scoped machine identity that cannot escalate.	Service principals and accounts are universal — Databricks service principals, Snowflake `TYPE=SERVICE` users, MotherDuck service accounts, cloud workload-identity federation — and AWS Bedrock AgentCore adds just-in-time token vending. None binds these to a turnkey native data-engineering agent's unattended run.	A bound Service Principal with a frozen role and immutable scope; bundled jobs use registry-owned Service Principals provisioned by reconciliation, and revoked or insufficient credentials fail closed.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M8.4 Deferred, fail-closed authority	A scheduled “run as me” must not execute on authority that lapsed after it was scheduled. Authority must be re-checked at fire time, not definition time.	Nobody binds a re-verified, just-in-time credential to the agent’s fire event: scheduled agent paths reuse stored bindings without a fail-closed re-check, and the JIT-vending primitives (Bedrock AgentCore, GCP Agent Identity) are not wired to a native data-engineering agent.	Run-as-self mints a fresh short-lived One-Time Token only after the schedule fires and live authority re-passes; if authority has lapsed the run fails closed and records the reason.

Choice

M9 — Multi-channel, multi-user coordination

> Agents must be reachable and steerable from where the work happens, by one accountable driver at a time, with safe handoff — and the agent's run must survive any single surface disconnecting.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M9.1 Multi-channel reach	Triggers are the work events themselves (Slack, Teams, webapp, GitHub Issues). Losing a surface must not stop the agent.	Omnigent exposes one agent over web, mobile, native app, and API; MotherDuck's remote MCP server reaches the platform from many agent clients — Claude, Cursor, ChatGPT, Teams/Copilot — plus SQL and REST.	A backend-owned event spine fans one agent run out to every channel; a dropped tab never stops the run.
M9.2 Single-driver with safe handoff	One accountable operator, keyed to the user (not the tab or channel), so two actors never corrupt one run.	Omnigent shares generic sessions for live co-review and steering, but has no single-driver data-safety invariant and no data-platform binding. None in data.	An edit lease keyed to the user enforces one driver, with cooperative handoff and a break-glass revoke.
M9.3 Scheduled and unattended triggering	Agents must fire on time-based schedules and system events with no human present — and a scheduled run must behave identically to a run-now, or the two paths diverge and the audit gaps.	Scheduled plumbing is commodity (Snowflake Tasks, Databricks Jobs, Airflow, MotherDuck Flights cron), but agent-backed scheduled runs with full conversation capture on one shared model are thin, and the native data-engineering agents that exist are human-in-the-loop.	Scheduled and run-now execution share one execution path and run record, distinguished only by trigger source; the full agent conversation is captured for every agent-backed run.

M9.1 covers human-initiated triggers across surfaces and surviving a dropped surface; M9.3 covers human-absent, time- and system-initiated triggers — together they cover both present-human and no-human execution.

Cost

M10 — Per-run cost attribution and control

> Agents scale unpredictably. Cost must be attributable and controllable before the bill surprises the team — a structural agent-native requirement, not an afterthought.

Marker	Why AI agents need it	State of the art	How Vibedata does it
M10.1 Cost attribution	Spend attributed to the specific agent run, so a runaway loop is visible and attributable.	Snowflake WAF names a Cost pillar and flags agent cost risk explicitly; Databricks attributes per-job and per-run cost via system tables and budgets. MotherDuck meters per-second Compute Units and breaks spend down per tenant on the invoice. None offers a per-agent-conversation cost line yet.	Cost attributed per Domain, Intent, Session, and Turn through OpenInference and Langfuse.
M10.2 Cost control	Budgets, caps, and pause-on-spend that stop an agent before it overruns.	Omnigent tracks per-session cost dynamically and can pause for approval after every $100 spent.	The LLM proxy is the policy point where budgets, caps, and the per-stage model policy apply — frontier models where judgment carries the load (specification, design, review), cheaper tiers for well-specified work. Structural verification is what makes the cheap tiers safe: the gates, not the model, decide what ships.

ANALYSIS

Why platforms stall short of ADER 1.0

The mature platforms have real markers. They stall short of ADER 1.0 for structural reasons, not capability gaps:

- **Business model.** Platforms that monetize compute and storage build an agent layer to drive consumption of their own platform, so it will never go cross-platform — lock-in is the model. That is the exact lowest-common-denominator trap Vibedata avoids by going native, one platform at a time.
- **Bounded scope.** A strong operate agent is still bounded to what increases platform stickiness — not the single practitioner's full requirements → build → deploy → operate lifecycle.
- **Gates without the loop.** Even after 2026's wave of runtime gates, enforcement is permission and approval — may this agent call this tool, did a human approve — not spec-first, data-quality gates bound to the agent's build loop. No incumbent refuses to publish on a failed data check the way an in-loop gate must. M4 and M7 together are where trust is decided: one primitive spanning build and operate, with gates that verify every change by executing it, positions hallucination structurally out of the picture — the gate's verdict, computed from the run, decides what ships.
- **Composition over accountability (a choice, not a gap).** Meta-harnesses headline composition — combining many agents. Vibedata deliberately does the opposite: single-agent-per-intent and single-driver. Composition trades away the accountability and debuggability that data work requires. We chose the trade the other way on purpose.

The honest read: most individual markers now exist somewhere — spec-first in coding agents, gated CI in dbt and SQLMesh, isolation and lineage in Databricks, governance and observability in Snowflake. What none of them assemble is the **combination**, for data, on one durable primitive that spans build and operate. That assembly is ADER 1.0.

TIMING

Why now: the category is validating this thesis

The markers are not a forecast. Within a single fortnight in June 2026 — Snowflake Summit and Microsoft Build around 2 June, then Databricks Data + AI Summit, Google's data agents, AWS Summit NYC, and MotherDuck between 16 and 18 June — the incumbents independently asserted Vibedata's core belief: that the harness above the model is the product, and the model is a commodity.

- **Databricks** shipped Omnigent, a "meta-harness" above coding agents ("the layer you work at shouldn't have to change every time the model does"), and Genie ZeroOps, an autonomous operations agent whose own pitch is that a generic coding agent structurally **cannot** run data operations — for the same reasons listed in M5.3, M6, and M8.
- **Snowflake** shipped a Well-Architected Framework "for the AI era" built on five pillars (Security & Governance, Operational Excellence, Reliability, Performance, Cost) and framed agent skills as "the SOPs that make AI production-ready."
- **Microsoft** ran its Build session "The AI-Native Data Engineer."

The category is consolidating around one thesis. The markers are how Vibedata stays ahead of it: not arguing that the harness matters — the incumbents now argue that for us — but defining the complete set a harness must provide, and building the last stretch to ADER 1.0 the platform vendors will not.

REFERENCES

Sources

State-of-the-art claims are grounded in the following. This whitepaper was last refreshed in July 2026; all vendor announcements and product docs were accessed as of that date.

Coding agents and meta-harness (M1, M5, M7.4, M8.1, M9, M10.2)

- Claude Code — project memory · plan mode · git worktrees · sandboxing
- OpenAI Codex — sandboxing
- AWS Kiro — specs (requirements → design → tasks) · GitHub Spec Kit · Linear MCP server
- Databricks Omnigent (meta-harness, cost policies, egress secret injection, session sharing)

Agent-readiness conventions (M1.4, M3.5, M7.5)

- Upriver — agent-ready data platform 5-dimension framework
- `AGENTS.md` open convention · dbt — "How we structure our dbt projects" · GitHub — CODEOWNERS + branch protection

Databricks (M3.3, M3.4, M4.2, M5.3, M6.1, M8.2, M10.1)

- Genie ZeroOps (detect → assess → remediate → verify; zero-copy clone sandbox) · ZeroOps Day-2 framing
- Delta / shallow clone · Asset Bundles · Unity Catalog lineage · AI-generated comments · semantics / metric views · Genie One / Ontology / Agents · job cost attribution · data-quality monitoring
- Unity AI Gateway Budgets (Public Preview) · Lakebase copy-on-write branches · Unity Catalog @ Data+AI Summit 2026 · Genie Code (instructions, skills, memory)

Snowflake (M1.2, M2.3, M3.1, M7.2, M8.2, M10.1)

- Well-Architected Framework "for the AI era" · Agent Skills as SOPs · Cortex agent skills
- Cortex AI Observability (GA) · access control (RBAC per query) · Horizon governance · data quality
- Cortex Code governed agent (Plan Mode, tiered tool-approval) · Budgets for AI features GA · Cortex Agent evaluations GA · intent to acquire Natoma (MCP identity gateway) · Snowflake Intelligence (channels) · Cortex agent resource budgets

Open data-engineering stack (M2, M3.2, M3.4, M4.1, M6, M7.1–7.3)

- dbt Cloud — Advanced CI / compare changes · documentation · Semantic Layer (MetricFlow)
- SQLMesh — CI/CD bot · table diff · isolated / blue-green systems
- Datafold Data Diff (and MCP) · Elementary — dbt-native observability
- Monte Carlo — Observability Agents · Operations Agent (GA) · Microsoft Fabric — Data Agent (read-only query, limits)

Unattended execution and workload identity (M8.3, M8.4, M9.3)

- Databricks — OAuth federation / service principals · Snowflake — tasks (`EXECUTE AS USER`, `TYPE=SERVICE`), `DATA_AGENT_RUN` (public preview)
- Google Cloud Agent Identity (SPIFFE, mTLS) · AWS Bedrock AgentCore Identity (JIT token vending) · Azure AI Foundry Routines (preview)
- IETF WIMSE workload identity practices · IETF agent audit trail (conversation capture, hash-chained records)

Platform capability sources

- Microsoft Fabric — branched workspace · governance & lineage (Purview) · chargeback / cost app · Operations agent GA · transparency note · Osmos acquisition · Fabric IQ (shared context layer)
- BigQuery / Google Cloud — Data Engineering Agent · pipelines (real-data profiling + human-approved plan) · table clones · Knowledge Catalog (ex-Dataplex) · Dataplex lineage · Gemini security & privacy · Semantic Governance Policies, Agent Gateway GA · new data agents across the agentic data cloud

- AWS — SageMaker Data Agent + serverless notebooks · EMR Spark Upgrade Agent (plan → fix → validate-vs-real-data) · Bedrock AgentCore Policy + Guardrails GA (Cedar deny-by-default) · granular Bedrock cost attribution · SageMaker Unified Studio lineage · Lake Formation access filters · Redshift data sharing · Bedrock AgentCore harness
- MotherDuck — Flights (agent-native pipeline runtime) · agent-native ingest · launch coverage · MCP server · building analytics agents (`try_bind`) · Agent Skills · Differential Storage · "Git for data" (no branching/merge) · hypertenancy · service accounts · Flights scheduling & runs · per-second billing · managed DuckLake preview

> **Note on authorship.** This whitepaper was produced by Accelerate Data's AI agents, working from the founding team's strategy inputs and internal source documents. The agents assembled the framework, drafted the prose, and checked every claim against those sources; the direction and judgment are Accelerate Data's.