

Auraa and Genie Code

Better Together – An Agentic Data Platform Meets Databricks' AI Coding Agent

Auraa is Covasant's Databricks-native, agent-driven data platform that captures every data engineering decision as structured metadata. **Genie Code** is Databricks' autonomous AI coding agent that reasons about data, writes production-grade code, and automates multi-step workflows inside the Lakehouse. This whitepaper explains why the two are complementary, not competitive and how their combination produces an agentic data platform that neither could deliver alone.



Executive Summary

In March 2026, Databricks launched Genie Code: an autonomous AI agent that writes SQL and Python, builds Lakeflow pipelines, debugs failures, generates dashboards, and monitors production workloads, all from inside the Databricks workspace. On real-world data science tasks, Genie Code more than doubled the success rate of leading coding agents.

It is a genuine breakthrough for individual data practitioners.

But a coding agent, no matter how capable, does not constitute an enterprise data platform.

Genie Code writes excellent code. It does not manage multi-tenant isolation. It does not store data engineering decisions as queryable, versioned metadata. It does not create, reconcile, or enforce grant policies across a fleet of tenant catalogs. It does not guarantee that a pipeline executed today will produce the same result tomorrow.

These are organizational capabilities, and they are exactly what Auraa was built to provide.

Auraa was built for agents. Not adapted for them. Not retrofitted. Built, from its first line of code, with the expectation that AI agents, not humans, would be the primary operators of the platform. Every capability is exposed as a tool. Not some capabilities. All of them. When a human uses Auraa, they use it through the same tool interface that an agent would. The web application, the API, the MCP server, these are different surfaces over the same tool layer.

This is a deliberate architectural commitment. Auraa is a Databricks-native platform where every data engineering decision, from source connection parameters to quality rule definitions to grant policies, is captured as structured configuration stored in Delta Lake tables. It does not generate code. It generates and stores configuration: typed, versioned, auditable records that are deterministically executable by any agent on the platform.

This is the fundamental distinction. Genie Code produces code that must be read, reviewed, and maintained by humans. Auraa produces configuration that is queryable, diffable, and reproducible by machines, and by other agents.

The two platforms are complementary because they operate at different layers of the same problem. Genie Code needs structured context to be effective, its documented "curation tax" means that poorly curated Unity Catalogs produce poor results. Auraa produces exactly that structured context through metadata-driven provisioning. Auraa, in turn, benefits from an intelligent coding agent that can accelerate novel development work that falls outside its configuration-driven patterns.

What makes this complementarity actionable, not just theoretical, is the Model Context Protocol (MCP). Auraa exposes its full capability surface as an MCP server: tools for execution, prompts for workflow guidance, and skills for deep domain knowledge. Genie Code consumes MCP natively. The integration requires no custom code, no middleware, no API bridging.

Auraa registers as a Databricks App. Genie Code connects in Agent mode. The practitioner has access to 166 governed platform operations alongside Genie Code's native coding abilities.

And Auraa is not limited to serving Genie Code. The same MCP surface is accessible to Claude Code, to Databricks Agents, or to any MCP-compatible client. Auraa also operates as its own standalone Databricks App with a full web interface, and as an autonomous agent (AIVARA) capable of executing complex workflows independently.

Three modes, App, Agent, and MCP Server. Auraa meets organizations wherever they already work.

What Genie Code Brings to the Table

Genie Code is not a chatbot bolted onto a notebook. It is a purpose-built AI agent for data work, deeply integrated with the Databricks platform.

Genuine Coding Partner

In Agent mode, Genie Code operates autonomously: it plans multi-step solutions, retrieves relevant assets from the workspace, writes and validates production-grade code, and fixes errors without human intervention. It works across notebooks, the Lakeflow Pipelines Editor, dashboards, and MLflow. It monitors pipelines in the background, triages failures, and investigates anomalies proactively. Over time, it learns team coding preferences through persistent memory.

These are real capabilities. On internal benchmarks, Genie Code achieved a 77.1% success rate on real-world data science tasks, compared to 32.1% for leading general-purpose coding agents.

The performance gap reflects its deep integration with Unity Catalog. Genie Code understands business semantics, governance policies, and data lineage in ways that a general-purpose LLM cannot.

Extensibility via MCP

Genie Code connects to external systems through the Model Context Protocol (MCP), an open standard for connecting AI agents to tools, resources, and contextual information. Teams can deploy custom MCP servers as Databricks Apps, and Genie Code will discover and use their tools automatically in Agent mode.

Built-in connectors provide access to Google Drive, SharePoint, GitHub, Glean, and Atlassian.

There is one critical constraint: Genie Code's MCP integration is limited to 20 tools across all connected servers. This hard ceiling forces careful curation of which operations to expose, a design consideration that becomes central to how Auraa integrates with Genie Code.

The Gaps That Remain

For all its capability, Genie Code was designed as a coding agent, not a platform. Several organizational requirements fall outside its scope:

Multi-tenant isolation. Genie Code operates within a single user's workspace context. It has no concept of tenant routing, catalog-per-tenant isolation, or hierarchical settings that cascade from platform to client template to tenant to project.

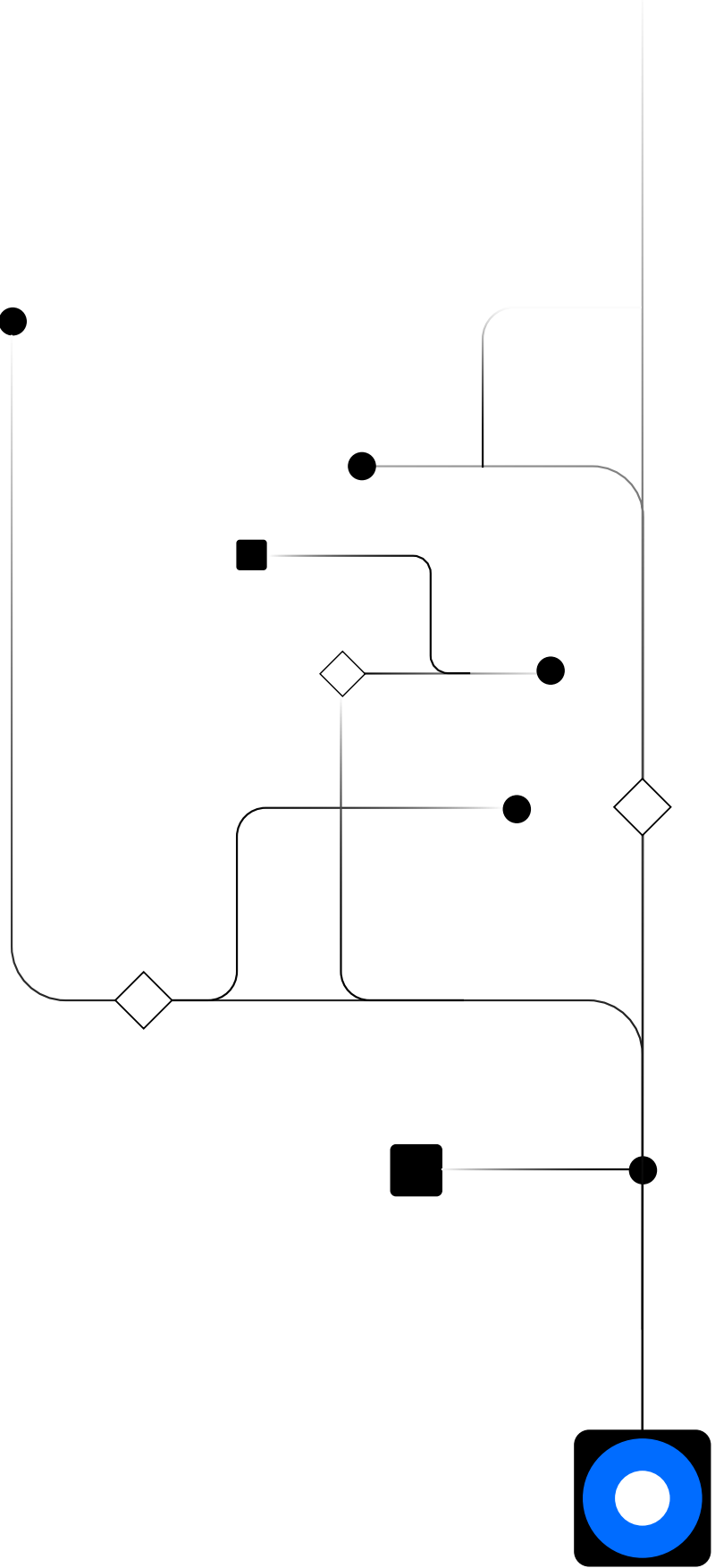
Configuration over code. Genie Code generates code, SQL, Python, pipeline definitions. Code is flexible but non-deterministic: the same natural language prompt may produce structurally different code on successive runs. For regulated industries that require audit trails and reproducible pipelines, this is a structural gap.

Governance automation. Genie Code reads Unity Catalog ACLs to enforce access control during its operations. But it does not create grant policies, reconcile them across tenant catalogs, detect permission drift, or apply hierarchical role-based templates. It respects governance; it does not produce governance.

Structured metadata. When Genie Code builds a pipeline, the result is a notebook or a pipeline definition, code in a workspace. The decisions embedded in that code, which tables to ingest, which quality rules to apply, which transformations to execute, live in procedural syntax, not in queryable tables. The next agent that encounters this pipeline must parse the code to understand what it does.

These are not weaknesses. They are boundaries, reasonable ones for a coding agent.

They also happen to describe, with precision, the capabilities that Auraa was built to provide.



What Auraa Provides

Most platforms that call themselves “AI-powered” started as human-first tools and added an AI layer afterward. The chatbot wraps the UI. The copilot suggests what the human could type faster. The agent is an assistant to the human operator.

Auraa inverts this entirely.

Auraa was built for agents. It is a tool-centric, AI-first, Databricks-native platform designed from its first commit with the expectation that AI agents, not applications, not dashboards, not humans clicking through forms, would be the primary operators. Humans provide intent and oversight. Agents do the work.

Every capability in Auraa is exposed as a tool. Every single one. Registering a data source is a tool. Running a quality check is a tool. Provisioning a tenant, applying a grant policy, detecting permission drift, generating an ingestion spec, all tools. There are no internal functions that only the UI can reach. There are no admin-only backdoors. There is one execution path, and everything, human-initiated or agent-initiated, flows through it.

This is not a philosophical statement. It is an architectural constraint enforced across 166 tools, 7 domain components, and every line of platform code. When a human clicks “Run Quality Check” in the web application, it invokes the same `aicura.execute_dqc.v1` tool that Genie Code would call via MCP. Same authorization check. Same audit trail entry. Same result.

The platform is built on a single organizing insight: data engineering decisions are not inherently code. They are metadata, structured, versionable, queryable records that describe what should happen, leaving the how to the platform. This insight, explored in depth in WP-001: Rethinking Data Engineering, produces the design principles that distinguish Auraa from both traditional data platforms and AI coding agents.

Configuration Generation, Not Code Generation

This is the most important distinction between Auraa and Genie Code.

When Genie Code builds an ingestion pipeline, it writes Python or SQL code in a notebook. When Auraa builds an ingestion pipeline, it creates a structured record in `metadata.ingestion_specs`, a Delta Lake table with typed fields for source, destination, column mappings, scheduling policy, and quality gate references.

The record is governed by Unity Catalog, versioned by Delta’s time-travel, and auditable through the bronze event log. Auraa stores all platform data in **Delta Lake tables** for analytical workloads and **Lakebase (PostgreSQL)** for sub-5ms OLTP reads, the same configuration record is written to both via GovernanceWriter, ensuring that governance lookups are fast and analytical queries are comprehensive.

A quality check in Genie Code is a SQL assertion embedded in a notebook cell. A quality check in Auraa is a record in `metadata.quality_rules` with a rule type, target column, threshold, and evaluation semantics. The same record can be executed by any tool that reads the quality rules table, today, next month, or after an agent proposes an improvement to the threshold.

This distinction matters for three reasons:

- **Reproducibility**, re-executing a metadata spec always produces the same result, regardless of which agent triggered it.
- **Auditability**, every configuration change flows through GovernanceWriter and is recorded in the bronze audit trail, an append-only Delta table that captures the who, what, when, and why of every mutation.
- **Composability**, metadata records can be queried, joined, and analyzed by any tool, including Genie Code itself. Code requires parsing.



Three Operating Modes

Auraa is not a single-purpose application. It operates in three modes, each serving a different integration pattern:

- **Databricks App.** A full web application deployed as a Databricks App with OAuth authentication, multi-tenant navigation, and agentic workflows powered by the Auraa Semantic Flow pattern. This is the primary interface for data teams who want a dedicated platform experience.
- **Autonomous Agent.** AIVARA, Auraa’s orchestrating agent, executes complex multi-step workflows from high-level intent. It discovers available tools at runtime, builds execution plans, and delegates to the appropriate components, all governed by tenant context and role-based access control.
- **MCP Tool Server.** All 166+ platform tools, 7 workflow prompts, and 7 domain skills are exposed via MCP. Any MCP-compatible agent, Genie Code, Claude Code, Databricks Agents, or a custom build, can connect and use Auraa’s capabilities as if they were native operations. This is the lowest-friction integration path and the one most relevant to Genie Code teams.

The three modes are not separate products. They share the same tool surface, the same authorization layer, and the same metadata substrate.

A quality rule created through the web App is immediately available to Genie Code via MCP. A tenant onboarded by AIVARA is visible to Claude Code in the next session. The modes differ in who initiates the work and how the results are presented, not in what operations are available.

The NEXARA Registry

Central to Auraa’s design is NEXARA, the platform’s registry for all machine-actionable capabilities. NEXARA maintains Delta Lake tables for tools, agents, agent-tool bindings, prompts, connectors, and affordances, each with versioning, component ownership, and tenant-scoped enablement policies.

- **Tools** are the atomic units of platform capability. Each of the 166 tools has a unique identifier, semantic version, typed input and output schemas, and component ownership. Tools are deterministic: given the same inputs, they produce the same outputs. The tool registry is not a static manifest, it is a runtime-queryable table that agents use to discover what operations are available for a given tenant, project, and role.
- **Agents** are registered with their configuration and tool bindings. The agent_tools_links table maps which agents can invoke which tools, with priority, weight, and context-filtering fields that allow the same agent to behave differently in different domains.
- **Prompts** are versioned templates with domain scoping and variable interpolation. They carry explicit version strings, enabling teams to pin workflows to specific prompt versions while newer versions are tested in staging.
- **Skills** are a distinct category. Unlike prompts, which are short parameterized templates, skills are deep, multi-page reference documents covering an entire platform domain, the kind of knowledge that would take a new team member weeks to absorb.

Skills are exposed as MCP resources via skills://aura/* URIs. They do not consume tool slots. An agent loads a skill into its context on demand, gaining comprehensive domain expertise instantly.

Critically, skills are not just instructions for using Auraa’s tools. They **embody enterprise-class data engineering best practices and patterns**, the accumulated knowledge of how to provision tenants safely, how to structure grant policies for least-privilege access, how to design ingestion pipelines that survive schema evolution, how to apply quality rules that catch real defects without generating noise.

When Genie Code loads an Auraa skill, it does not merely learn which tool to call. It learns how an experienced platform engineer thinks about the problem, the tradeoffs, the failure modes, the patterns that work at scale.

This is institutional knowledge made machine-readable.

Skill	Domain
skills://aura/foundra-platform	Platform initialization, tenant onboarding, lifecycle management
skills://aura/nexara-registry	Tool resolution, enablement policies, RBAC integration
skills://aura/infraia-infrastructure	Catalog provisioning, grants, secret scopes
skills://aura/compose-interaction	Adaptive transient forms via the Auraa Semantic Flow pattern
skills://aura/nexara-registry	Building Auraa Databricks Apps, identity/RBAC, multi-tenant context, AIGENE governance integration
skills://aura/aura-install	Platform installation, configuration, and validation
skills://aura/mcp-workflows	Multi-step workflow sequences for common operations

Every registry entry shares four properties:

- **Versioned**, prompts and tools carry semantic versions; affordances track version and component.
- **Auditable**, all writes flow through GovernanceWriter with bronze audit trail.
- **Tenant-scoped**, platform defaults can be overridden per tenant, project, or role.
- **Discoverable at runtime**, agents resolve available capabilities dynamically based on context.

The Complementarity

The previous sections describe what each platform does independently. This section explains why they are better together, why the combination produces capabilities that neither achieves alone.

Auraa Eliminates the Curation Tax

Genie Code's effectiveness is directly proportional to the quality of the Unity Catalog it operates against. This is well-documented: the "curation tax" means that poor naming conventions, empty column descriptions, and missing lineage metadata cause Genie Code to hallucinate at scale. The agent reflects organizational sloppiness back, amplified.

Most teams treat catalog curation as a manual discipline, a documentation task assigned to conscientious individuals. This approach does not scale, and it does not survive team turnover.

Auraa eliminates the curation tax structurally. When Auraa provisions a tenant, it creates catalogs, schemas, tables, and grants through a reproducible, auditable pipeline. Every catalog object is created with structured descriptions, column-level metadata, and lineage context, not because someone remembered to fill in a description field, but because the provisioning pipeline requires it.

Quality rules are stored as queryable data, not scattered SQL comments. Business context flows from the metadata spec that created the object, not from a separate documentation effort.

The result: when Genie Code operates against an Auraa-managed Lakehouse, it encounters a catalog that is uniformly curated, consistently structured, and richly annotated.

The curation tax is zero, not because someone paid it, but because it was never incurred.

Active Governance, Not Passive Enforcement

Genie Code reads Unity Catalog ACLs and respects them during execution. This is valuable, it prevents unauthorized data access. But reading permissions is not the same as managing them.

Enterprise data governance requires active operations: creating grant policies that define what permissions each role should have, applying those policies consistently across dozens of tenant catalogs, detecting when actual permissions drift from declared policies, and remediating drift when it occurs. These are continuous, cross-tenant operations that no coding agent is designed to perform.



Auraa's governance stack provides exactly this:

- **Twelve registry tables** of grant policies, role capabilities, role-capability mappings, group definitions, and secret scope ACL policies, the declarative source of truth for what permissions should exist.
- **GovernanceWriter**, a parallel dual-write engine that upserts governance changes to both Delta Lake and Lakebase (PostgreSQL) simultaneously, providing sub-5ms policy lookups for API authorization checks.
- **Tenant Reconciler**, a service that compares declared policies against actual Unity Catalog state and produces a drift report with one-click remediation.
- **Grant Templates**, versioned, hierarchical templates that cascade from platform defaults through client templates to tenant-specific overrides, ensuring that new tenants start with correct permissions and that policy updates propagate systematically.

When Auraa governs the Lakehouse, Genie Code operates in a richer context. Every catalog, schema, and table it encounters has proper ACLs, correct ownership, and auditable provenance. The agent can focus on the data work it was built for, confident that the governance substrate is sound.

Solving Non-Determinism with Configuration

Genie Code generates code. Code is flexible, expressive, and capable of handling novel problems, and it is non-deterministic. The same natural language request may produce structurally different SQL or Python on successive runs.

For creative, exploratory work this is acceptable. For production pipelines in regulated industries that require audit trails and reproducible execution, it is a structural gap.

Auraa takes a fundamentally different approach. It produces and stores **configuration**, not code.

An ingestion pipeline is not a Python notebook, it is a structured spec record with typed fields for source, destination, column mappings, and scheduling. A quality check is not an ad-hoc SQL assertion, it is a rule record with deterministic evaluation semantics. A grant policy is not a one-time SQL GRANT statement, it is a versioned record that can be re-applied, compared, and audited.

When Genie Code is connected to Auraa via MCP, the two approaches compose naturally:

- Genie Code can query existing configuration specs instead of generating code from scratch.
- It can propose changes to metadata records, changes that are reviewable, diffable, and auditable in the same bronze event log that tracks every other platform mutation.
- It can invoke Auraa tools that produce deterministic outcomes from configuration.

This division of labor plays to each platform's strength. Let Genie Code handle the creative work: novel analysis, complex transformations, bespoke dashboards. Let Auraa handle the repeatable work: provisioning, governance, quality enforcement, ingestion.



Multi-Tenant Isolation

Genie Code operates within a single user's workspace context. This is natural for a coding agent, the practitioner works in their notebook, against their data, with their permissions.

Enterprise data platforms operate at a different scale. Auraa manages multi-tenant environments where each tenant has isolated catalogs, scoped schemas, hierarchical settings, and role-based access control that cascades from platform defaults through four levels of configuration. Cross-tenant sharing agreements allow controlled collaboration between organizations.

This architecture is not just multi-tenant isolation, it is a data mesh topology enforced at the platform level. Each tenant catalog is an autonomous data domain node: independently governed, independently provisioned, independently queryable. Cross-tenant sharing agreements provide the controlled federation that data mesh requires without sacrificing isolation. Auraa operationalizes data mesh; Genie Code works within it.

Every tool invocation resolves tenant context from the authenticated principal's role bindings. The same operation behaves differently depending on which tenant the user belongs to.

This is invisible to Genie Code, and that is the point. When Genie Code connects to Auraa's MCP server, tenant isolation is enforced transparently. The practitioner sees only the tools and data their tenant and role allow.

Auraa handles the organizational complexity so that the coding agent operates in a clean, scoped context.

The 20-Tool Limit

Genie Code's MCP integration is limited to 20 tools across all connected servers. This is a hard constraint that forces careful curation. Most MCP servers burn through it quickly by exposing individual operations, a pattern that is generous with tool slots and stingy with capability.

Auraa solves this architecturally with a three-meta-tool pattern:

- **list_available_tools** returns the full catalog of 166+ tools available to the current tenant and project, filtered by component, role, and enablement policies. The agent uses this to browse what operations exist.
- **find_tool** searches for tools by keyword, domain, or capability. When an agent knows what it needs to do but not which tool does it, find_tool returns the best matches with descriptions and input schemas. This is targeted discovery, the agent does not need to load the full catalog to find the right operation.
- **invoke_tool** executes any tool from the catalog by its fully qualified identifier, with the provided arguments. The invocation routes through Auraa's full authorization, audit, and execution pipeline. Every call is tenant-scoped, role-checked, and recorded in the bronze audit trail.

These three tools consume 3 of the 20 available MCP slots. The remaining 17 are used for curated convenience tools, **list_tenants**, **onboard_tenant**, **run_quality_check**, **profile_table**, **get_platform_status**, and others, that provide optimized schemas for the most common operations. The convenience tools are accelerators, not requirements. The three meta-tools provide complete coverage of all 166 platform operations.

This design makes Auraa uniquely efficient as a Genie Code MCP server: maximum capability within the tool budget, with graceful discovery built in.



Guiding Genie Code via Prompts and Skills

MCP exposes three primitive types: tools (executable operations), resources (contextual data), and prompts (guided workflows). Tools count against Genie Code's 20-tool limit. Prompts and resources do not. They are free.

Auraa exploits this asymmetry. Seven MCP workflow prompts teach Genie Code how to use the platform effectively, step-by-step, in the right order, with the right tools and parameters:

Prompt	Workflow
onboard_tenant_workflow	Create tenant, assign admin, verify provisioning, run quality baselinet
quality_assessment_workflow	Profile table, propose rules, execute checks, review results
ingestion_pipeline_workflow	Discover source, generate spec, propose quality rules, execute
user_access_workflow	Register user, assign role, verify access, audit permissionst
grant_compliance_workflow	Detect drift, review policies, remediate, verify compliance
metadata_tagging_workflow	Discover tags, apply rules, run auto-tagger, review proposals
platform_health_workflow	Check status, review tenants, investigate issues

These prompts are powered by NEXARA's prompt registry, versioned, domain-scoped templates with variable interpolation. When Genie Code activates a prompt, it receives a structured multi-step plan that references specific Auraa tools by name. The agent does not need to reason about the workflow from first principles. Auraa teaches it.

Skills operate at a deeper level. Where prompts guide a single workflow, skills provide comprehensive domain knowledge, not just how to use Auraa's tools, but the enterprise data engineering best practices that inform when and why to use them.

A skill like skills://aura/foundra-platform covers the entire platform initialization lifecycle: what phases exist, what each one does, how they depend on each other, what to do when one fails, and the architectural reasoning behind the design. A skill like skills://aura/infraia-infrastructure teaches how to structure Unity Catalog grants for least-privilege access, how to manage secret scopes across environments, and how to design catalog hierarchies that scale with organizational growth.

Skills are loaded into the agent's context as MCP resources, on demand, at zero cost to the tool budget.

This is the difference between giving an agent a tool and giving it expertise.

Any MCP server can expose tools. Auraa's skills transfer the judgment and institutional knowledge that distinguish a competent pipeline from a production-grade one, the patterns that survive schema evolution, the quality strategies that catch real defects without generating noise, the governance structures that pass compliance audits.

When Genie Code loads an Auraa skill, it operates not just with more capability, but with better judgment.

The three layers compose into a guidance model:

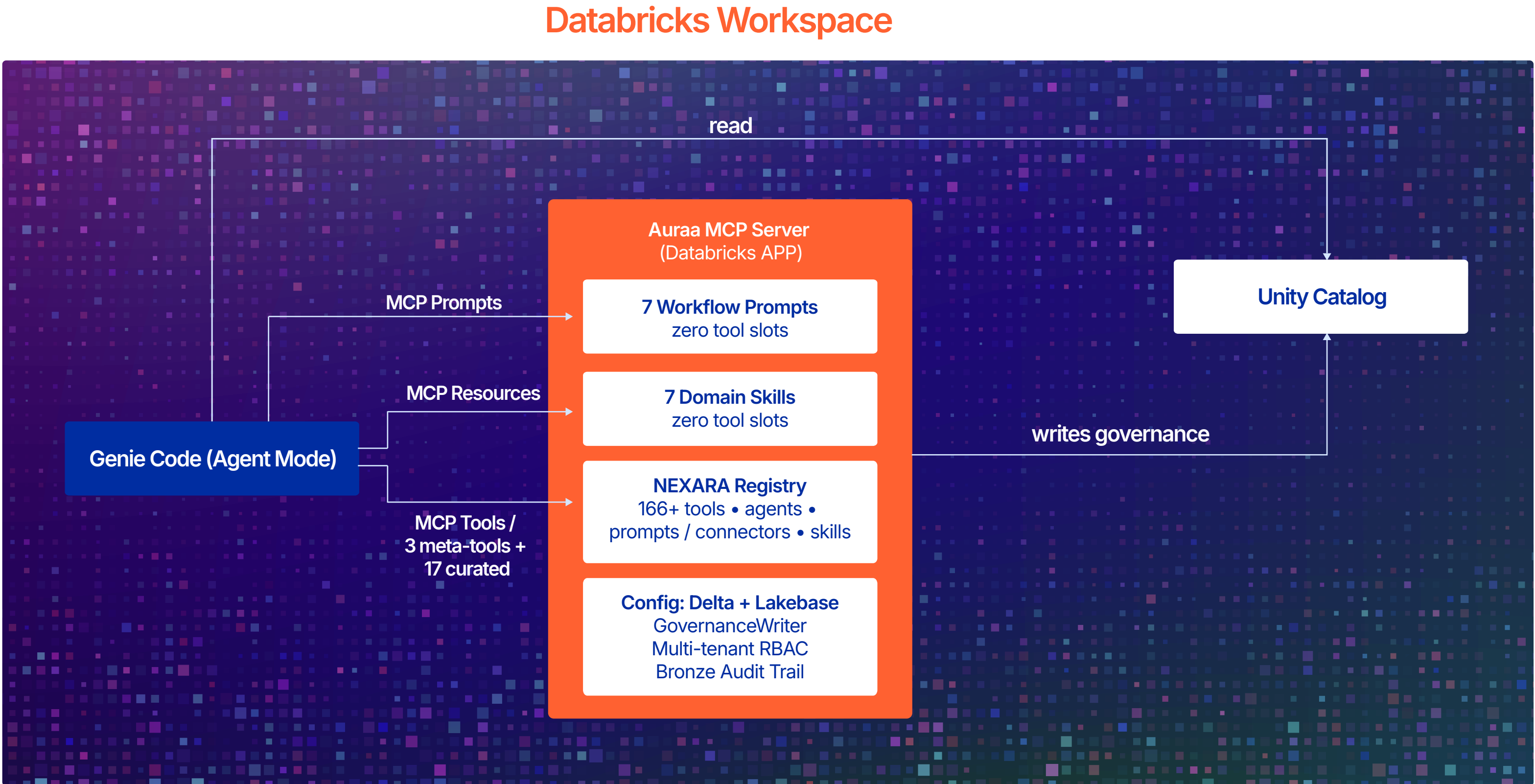
- **Skills** provide domain knowledge and best practices.
- **Prompts** provide structured workflow plans.
- **Tools** provide executable operations.

An agent that has loaded a skill, activated a prompt, and has access to the three meta-tools can perform complex platform operations with the precision of an experienced engineer, on its first interaction with the platform.

Integration Architecture

How the Pieces Connect

The integration between Genie Code and Auraa requires no custom code. Auraa is deployed as a Databricks App with an MCP endpoint. Genie Code connects to that endpoint in Agent mode. The result is a unified experience where the practitioner has access to both Genie Code's native coding abilities and Auraa's full platform surface.



Scenario: Building an Ingestion Pipeline

A data engineer opens Genie Code in a notebook and types: “Build an ingestion pipeline for the orders schema from our PostgreSQL source.”

Genie Code, connected to Auraa’s MCP server, executes:

1. **Discover**, invokes Auraa’s discover_metadata tool, which connects to the registered PostgreSQL source, probes schemas, tables, columns, and row counts, and returns structured metadata.
2. **Generate**, invokes generate_ingestion_spec, which creates a metadata-driven ingestion specification stored in Delta, not ad-hoc SQL, but a versioned configuration record.
3. **Propose quality rules**, invokes recommend_quality_rules, which analyzes column types and data patterns and proposes rule records stored in the quality rules table.
4. **Review**, the engineer reviews the spec and rules, either inline in Genie Code or through Auraa’s Semantic Flow UI.
5. **Execute**, invokes run_ingestion, which executes deterministically from the approved spec through the medallion pipeline.

Every step is audited in the bronze event log. The ingestion spec is version-tracked in Delta. The quality rules are queryable by any future agent. If the engineer wants to modify the pipeline six months later, the configuration is still there, a structured record, not a notebook that might have been edited, moved, or deleted.

Scenario: Governance Compliance Audit

A platform administrator asks Genie Code: “Which tenants have grant drift?”

1. Genie Code invokes Auraa’s check_grant_drift tool, which queries the grant policy registry against actual Unity Catalog state across all managed tenants.
2. The tool returns a structured drift report: which tenants are out of compliance, which grants are missing, and which unexpected grants exist.
3. The administrator reviews the report and approves remediation.
4. Genie Code invokes apply_grant_policies with dry_run=false, and Auraa’s Tenant Reconciler applies the corrections, with full audit trail.

This workflow crosses multiple tenants, compares declared state against observed state, and applies corrections through a governed pipeline. It is not something a coding agent can improvise from Unity Catalog’s SQL interface. It requires the structured policy registry, the reconciliation logic, and the audit infrastructure that Auraa maintains.

Auraa as a Universal MCP Server

Auraa’s MCP surface is not exclusive to Genie Code. The same tools, prompts, and skills are accessible to any MCP-compatible agent:

- **Claude Code**, developers working in VS Code or JetBrains connect to Auraa via MCP, gaining access to platform governance and metadata operations alongside their normal coding workflow.
- **Databricks Agents**, custom agents built on the Mosaic AI Agent Framework can consume Auraa’s tools for specialized workflows.
- **Custom integrations**, any system that speaks MCP can connect. A CI/CD pipeline, a Slack bot, a monitoring system, all can invoke Auraa tools through the same authenticated, audited interface.
- **AIVARA**, Auraa’s own orchestrating agent uses the same tool surface for fully autonomous operations.

This positions Auraa as the governance and metadata substrate for the entire agentic ecosystem. The choice of which agent to use is a user-experience decision, not an infrastructure one. The tools, the metadata, and the audit trail are shared.

Why Now

Databricks is actively promoting Genie Code as the future of data work. The timing is significant for three reasons.

Genie Code creates demand for what Auraa provides. As organizations adopt Genie Code and move past initial experimentation, they will encounter the curation tax, the governance gap, and the reproducibility challenge. These are not theoretical risks, they are documented limitations with real production consequences. The organizations that invest in structured metadata, active governance, and configuration-driven pipelines will extract the most value from Genie Code. Auraa is that investment, already built.

MCP creates a universal integration surface. The Model Context Protocol, adopted by Databricks, Anthropic, and the broader AI ecosystem, provides a standard way for agents to consume external capabilities. This eliminates the custom integration burden that historically made platform-to-platform connections expensive to build and fragile to maintain. Auraa’s MCP server is a Databricks App deployment, no middleware, no proxies, no custom protocol adapters.

The agentic ecosystem is fragmenting, and that is healthy. Genie Code is not the only agent. Claude Code, GitHub Copilot Workspace, and custom-built agents each serve different workflows and personas. Organizations need a governance and metadata layer that works across all of them, not one that is locked to a single agent. Auraa’s three operating modes (App, Agent, MCP Server) ensure that it serves as the common substrate regardless of which agent is the user-facing interface.

Conclusion

Genie Code is transformative for individual data practitioners. It makes them dramatically more productive at writing code, debugging pipelines, building dashboards, and analyzing data. It is, by Databricks' own benchmarks, the most capable data-focused coding agent available.

Auraa is transformative for data organizations. It makes the entire Lakehouse governed, reproducible, and configuration-driven. It manages multi-tenant isolation, enforces grant policies, tracks every mutation in an auditable event log, and stores every data engineering decision as structured metadata.

Together, the practitioner gets an AI coding agent that operates with full organizational context, governed access, deterministic configuration, versioned metadata, and enterprise-grade isolation.

The integration is elegant: three meta-tools give Genie Code access to 166 operations in 3 tool slots. Seven prompts guide workflows without consuming any slots. Seven skills provide deep domain knowledge on demand. NEXARA's versioned registry provides the auditable foundation.

And the flexibility is unique. Auraa adapts to the organization's adoption curve:

- Start by connecting Genie Code to Auraa's MCP tools, the lowest-friction path.
- Explore the standalone web App when the team is ready for a dedicated platform experience.
- Let AIVARA operate autonomously when trust in the governance substrate is established.
- Connect Claude Code for developers who live in their IDE.

Genie Code is the hands. Auraa is the brain and the rules.

Together, they are the agentic data platform.

Further Reading

- [Rethinking Data Engineering: An AI-First, Metadata-Driven Platform](#), The founding vision: why metadata-as-data changes everything.
- [Auraa Platform: An Agentic Data Activation Platform for Databricks](#), Detailed architecture of every Auraa component.
- [Auraa & Databricks: The Agentic Substrate for the Intelligent Lakehouse](#), How Auraa utilizes and extends the Databricks platform.
- [Databricks: Introducing Genie Code](#), Databricks' launch announcement and capabilities overview.
- [Databricks: Agentic Data Engineering with Genie Code and Lakeflow](#), Genie Code's data engineering workflows.
- [Databricks: Connect Genie Code to MCP Servers](#), MCP integration documentation and tool limits.





About Covasant

Covasant Technologies is an emerging global leader in delivering Agentic AI-led services that address complex, industry-specific challenges. Through its pioneering Services-as-Software model, Covasant brings together capabilities in data engineering, digital and cloud, AI engineering, and Enterprise Risk Management (ERM). Strategically located in innovation hubs including Hyderabad, Plano, New York, Los Angeles, London, and Dubai, Covasant leverages a premier talent ecosystem to deliver scalable, autonomous solutions. Our "Governance-Led" approach ensures that global enterprises can automate decision-making and orchestrate business processes with the reliability and speed required in today's market.



Plano, TX (USA) • London, UK • Hyderabad, India • Dubai, UAE



kathy.harnois@Covasant.com