SILECTIS · MAGPIE

Platform Architecture

Technical overview of the platform structure. August 2026.

Magpie is built to accelerate time-to-value on enterprise data projects. It does that by integrating three things most stacks treat as separate products: the runtime that executes the work, the catalog that describes it, and the AI assistants that act on both. One platform, one metadata model underneath. The foundations are open source, so existing skills, tools, and cloud investments carry over.

THE FOUR ARCHITECTURAL CHOICES

The catalog is a byproduct of execution.

Schema, lineage, ownership, and activity are captured as work happens. There is no parallel registration step, no scheduled crawl, no eventually-consistent shadow catalog. A `CREATE TABLE` in MagpieScript creates the table and records its metadata in the same operation. The catalog is the trace of the work, not a model of it.

The AI reads the platform's own metadata.

There is no second catalog built for the assistants. The Catalog Assistant traverses the same lineage graph the runtime captured. The Data Exploration Assistant validates queries against the same schema execution will use. Traceable rather than approximate: the assistants read the source of truth, not an embedding-store summary of it.

Source data stays in place.

Magpie reads from customer-controlled stores using credentials the customer issues and revokes. There is no data copy held inside the platform independent of customer-granted access. When upstream access changes, the change applies immediately. Customer IAM is the gate.

Open at the substrate.

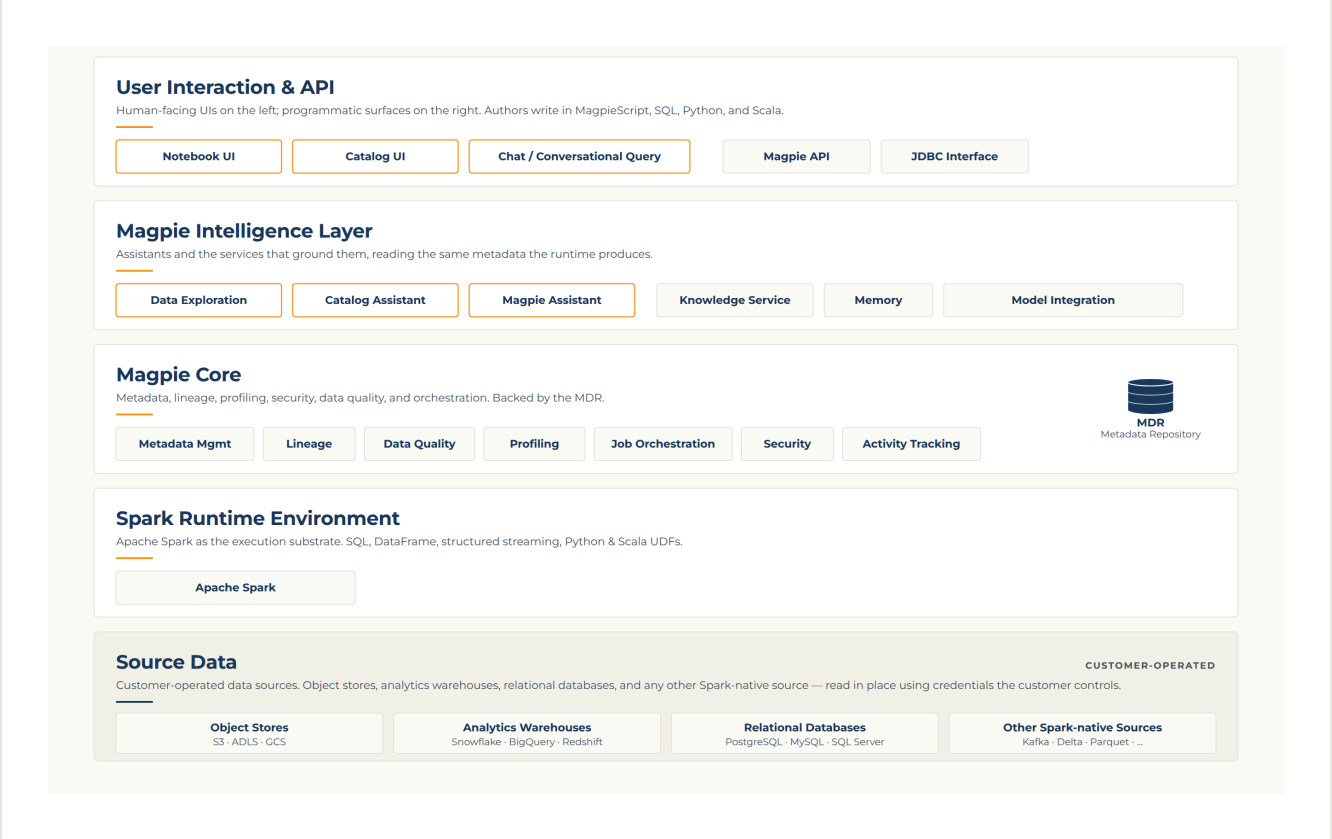
No proprietary connector, driver, file format, or model surface is required. Apache Spark is the runtime; existing Spark jobs run unmodified. JDBC is the access protocol, so existing BI tools connect natively. The knowledge corpus is plain markdown. Model providers are pluggable across OpenAI, Anthropic Claude, AWS Bedrock, and Azure Foundry.

The sections that follow describe each layer. The choices above are what hold the platform together as one thing rather than four.

THE STACK

Five layers. One metadata model.

Magpie is structured as five layers. Each layer has a single responsibility. The layers connect through well-defined interfaces, and the metadata captured at each level is the metadata every other level reads.



Magpie's five layers. Each has a single responsibility; the layers connect through well-defined interfaces.

01 LAYER ONE

User Interaction & API

The top layer is where humans and other systems interact with Magpie: three human-facing surfaces, two programmatic ones.

Notebook HUMAN-FACING	Multi-language interactive computing with real-time collaboration and the Notebook Assistant inline.
Catalog UI HUMAN-FACING	A first-class interface for browsing schemas, projects, sources, repositories, lineage graphs, and field-level history. The Catalog Assistant lives inside it.
Data Exploration HUMAN-FACING	Natural-language data exploration through conversation. Each response includes the SQL it ran and the catalog objects it touched, so the work is reviewable, saveable, and shareable.
Magpie API PROGRAMMATIC	A REST interface to the platform's capabilities: catalog operations, query generation, conversational queries, notebook execution, saved questions, and more. Customer applications, internal automation, and integration partners build against it directly.
JDBC Interface PROGRAMMATIC	Exposes Magpie to existing BI tools (Tableau, Looker, Power BI) and any other JDBC-aware client via the standard protocol. No platform-specific connector required.

FOUR FIRST-CLASS LANGUAGES

MagpieScript / SQL / Python / Scala

All four are first-class in the notebook and at execution.

02 LAYER TWO

Magpie Intelligence Layer

The Intelligence Layer hosts the assistants and the infrastructure that grounds them. There is no second catalog for the assistants; each one reads the same metadata the runtime captured.

Data Exploration Assistant ASSISTANT	Translates natural-language questions into validated SQL and presents the results. Queries are checked against the same schema execution will use. Every answer carries its own provenance: the SQL it ran and the catalog objects it touched. Results can be refined across turns, pivoted into charts, saved, and shared.
Catalog Assistant ASSISTANT	An action-planning agent. It traverses lineage, runs validating queries, and proposes catalog updates. Every action is grounded in the platform's existing metadata; every change is reviewable before it applies.
Notebook Assistant ASSISTANT	Works alongside the user in the notebook. It reads prior paragraphs and the working catalog for context, generates MagpieScript, SQL, Python, and Scala inline, and proposes every change as a diff the user approves or rejects. Three interaction modes: Build (write inline), Plan (design multi-turn), Refine (iterate).
Knowledge Service SUPPORTING SERVICE	Retrieves from a hierarchical documentation corpus organized by topic. Retrieval is by explicit topic ID, not embedding similarity. The result is more predictable assistant behavior and a simpler audit path when something needs to be diagnosed.
Memory SUPPORTING SERVICE	Maintains durable assistant context across conversations through observation logging and background compression.
Model Integration SUPPORTING SERVICE	Treats AI model providers as pluggable. OpenAI, Anthropic Claude, AWS Bedrock, and Azure Foundry are all supported. The assistants don't change behavior based on which model is wired in.

The Intelligence Layer composes prompts and context; the data itself is processed within the platform.

03 LAYER THREE

Magpie Core

Magpie Core is the engine. It owns metadata management, lineage, profiling, security, data quality, job orchestration, and activity tracking.

All of it is backed by the **Metadata Repository (MDR)**, the persistent store that records everything the platform knows about the data it operates on. Schema and lineage are captured automatically on every data load. The catalog is a first-class output of execution, not a downstream observation.

Multi-tenant identity, role-based access control, and KMS-encrypted secrets are enforced at the Core. Every action produces an audit record.

04 LAYER FOUR

Spark Runtime Environment

The runtime is **Apache Spark**. Magpie extends it with command processors and lineage capture, but the Spark API itself is unchanged. Existing Spark jobs run on Magpie unmodified.

SQL, DataFrame, structured streaming, and UDFs in Python and Scala all work as expected. Customers' Spark, Python, Scala, and SQL skills transfer.

Continuous streaming ingestion sits alongside scheduled batch loads: same platform, same catalog, same lineage. Real-time and historical data join in the same pipeline.

The catalog is a first-class output of execution, not a downstream observation.

05 LAYER FIVE

Source Data

The base layer is the customer's source data. Magpie reads in place, across object stores, analytics warehouses, conventional relational databases, and any other source the Spark runtime can connect to.

Object stores
AWS S3, Azure ADLS, Google Cloud Storage

Analytics warehouses
Snowflake, BigQuery, Amazon Redshift

Relational databases
PostgreSQL, MySQL, SQL Server, and other JDBC-compatible engines

Other Spark-native sources
Kafka, Delta Lake, Parquet, Avro, ORC, and the broader Spark connector ecosystem

The customer issues credentials with the access scope they want Magpie to have and revokes them through their cloud's standard IAM or database authentication. There is no data copy held inside the platform independent of customer-granted access. When access changes upstream, the change applies immediately.

Data sources can live in any cloud account or on-prem environment the runtime can reach.

DEPLOYMENT

Customer data. Customer IAM. Customer audit logs.

Magpie's compute (Spark runtime, Magpie Core, the Intelligence Layer, and the user-facing services) runs in Silectis-operated cloud accounts on AWS, GCP, or Azure. The customer operates the cloud accounts that hold source data. Magpie reads from those accounts using credentials the customer issues.

THE COMMAND DSL

MagpieScript

MagpieScript is the platform's command DSL: the language used to describe data structures, move data, run pipelines, schedule work, and administer the platform, alongside SQL, Python, and Scala in the same notebook.

It is not a SQL alternative. SQL is for querying; MagpieScript is for everything that happens around the query: telling the platform what data it should know about, where the data comes from, when work should run, and who can see what.

Structure
`CREATE`, `ALTER`, `DROP`, `DESCRIBE` for schemas, tables, streams, fields, and data sources

Data movement
`COPY`, `EXTRACT`, `EXPORT`, `SAVE`, `EXECUTE SQL SCRIPT`

Orchestration
`CREATE PROJECT`, `CREATE JOB`, `CREATE TASK`, schedules, cluster lifecycle

Context
`USE` to navigate organization, repository, schema, project, and cluster

Security
`GRANT`, `REVOKE`, role and access-token management

Quality
`PROFILE` and `VALIDATE` for schemas and tables

Every command produces a metadata side-effect. `CREATE TABLE` defines the table and records it in the catalog in the same step. `CREATE JOB` registers a runnable unit that is immediately visible to the Catalog UI and addressable through the API. There is no second mechanism for telling the catalog about the work; the work and the catalog entry are the same operation.

Because every platform action is expressible as code, AI agents work in the same medium the customer's team works in. The Catalog Assistant proposes catalog updates as MagpieScript. The Notebook Assistant generates platform operations the same way it generates queries. Every change is reviewable as a diff before anything ships. The feedback loop runs both ways: work the platform does gets recorded as metadata, metadata grounds the AI, and the AI proposes more code the team reviews and approves.

The runtime parses MagpieScript and routes each command to the appropriate processor: Spark for query and transformation, save processors for table writes, data-source processors for connections, validation processors for data quality. The processor surface is extensible; the grammar is the contract.

THE METADATA MODEL

More than 40 object types, audited.

The MDR records what the platform knows about the data it operates on. Every object is a first-class entry with a stable identity, an owner, and an audit trail. The objects fall into four groups.

Tenancy and identity

Organization — top-level tenant; the boundary for users, repositories, and clusters

User — individual identity

Role — named permission set; global or organization-scoped

Data structures

Repository — namespace containing schemas, data sources, and projects

Data source — connection to an external system

Schema — logical namespace for tables and streams

Table / Stream / Field / Sink — the data structures themselves

Orchestration

Project — container for related jobs

Job / Task — executable units and their steps

Job schedule — when a job runs; subscriptions notify on events

Cluster — Spark compute target with its own schedule

Capability and security

Validation suite / Validator — named data-quality rules

Secret — encrypted credential or configuration value

File — blob storage for scripts and artifacts

Identity provider / Access token — SAML/OAuth and programmatic credentials

The hierarchy is multi-tenant by construction. Organizations contain repositories; repositories contain schemas; schemas contain tables and streams. Lineage and activity history are not separate objects: they are derived from operations on the objects above. The metadata captured during execution is the metadata the Catalog UI and the assistants use in their processing.

SECURITY

SOC 2 Type II audited annually.

The audit covers internal security practices, including software development, access control, data handling, and platform operations. Within the platform: multi-tenant identity, role-based access control, KMS-backed encryption for secrets and field-level data, and an audit record for every action. Source data stays in your cloud accounts, read under credentials you issue and revoke.

See it run on your data · demetri@silect.is · silect.is/demo