

# Why Data Lakehouses Are Becoming the Default Architecture for Data Engineers

Data Science Collective



Advice, insights, and ideas from the Medium data science community

# Why Data Lakehouses Are Becoming the Default Architecture for Data Engineers



[Source](#)

I've worked with data warehouses, data lakes, and hybrid "duct-taped" analytical ecosystems that nobody wants to admit exist. And the truth is: **data complexity has exploded faster than the architectures we originally used to store and process it.**

Traditional data warehouses were optimized for structured data and BI dashboards.

Data lakes emerged to store *everything* cheaply, but with *very few guardrails*.

The **data lakehouse** is the architectural answer to this imbalance.

It gives us **the performance and transactional guarantees of a warehouse**, and **the flexibility and scalability of a lake**, all over *open formats* like Parquet.

And if your company works with datasets like the **Books** dataset below (streaming new editions, adding metadata, merging publisher records), you will feel *exactly* why this matters.

## Dataset Used in Examples (Conceptual Only)

```
books.csv
-----
book_id, title, author, genre, price, publication_date, rating
1, "The Silent Forest", "Marin Hale", "Fantasy", 18.99, 2022-01-14, 4.5
2, "Learning Python Fast", "J. O'Reilly", "Technology", 32.00, 2023-06-02,
4.2
3, "Cooking for Busy Minds", "A. Flores", "Cooking", 25.50, 2021-09-12, 4.7
...
```

This dataset comes in CSV today, JSON tomorrow, XML from a publisher next week, and logs from an reviews API every hour.

**That's why lakehouses exist.**

## 1. Why Choose Lakehouses Over Data Lakes?

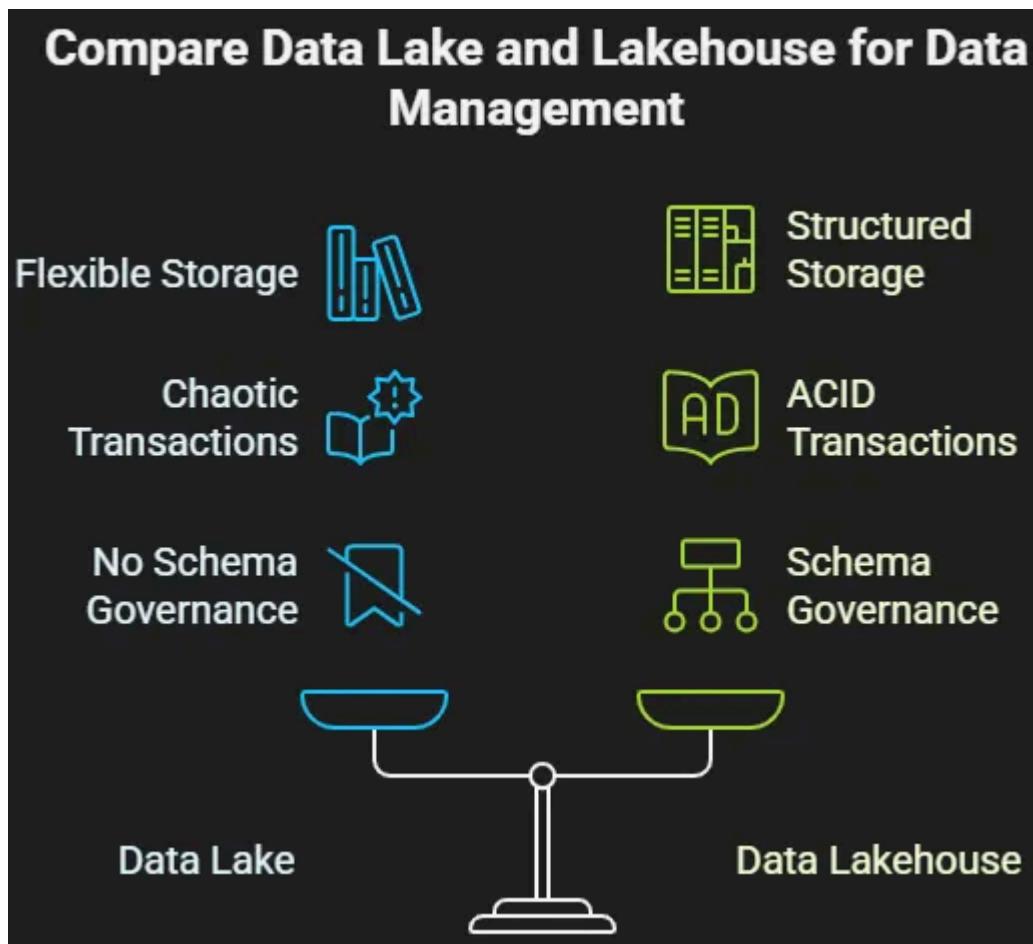
A **data lake** lets you store raw data in any format. That's helpful, until multiple teams try to read and write to the same data and **step on each other**. There are **no built-in transactions**, no guaranteed consistency, and no schema governance by default.

A **data lakehouse**, on the other hand:

- **Supports ACID transactions** (Atomicity, Consistency, Isolation, Durability)
  - If two teams update the books table at the same time, nobody corrupts data.
- **Maintains schema and version control**
  - The "Books" table *cannot* silently break dashboards.
- **Allows BI tools and ML pipelines to query from the same tables**
  - No more copies, extracts, re-exports, renamed exports, or "final\_v5\_really\_final.parquet".

In short:

**Data lakes store everything. Lakehouses make everything usable.**



A side-by-side comparison diagram of Data Lake vs Data Lakehouse. Source

## 2. What a Data Lakehouse Actually Is

A **data lakehouse** combines the **flexibility of data lakes** with the **reliability and performance of data warehouses**, all on top of **open table formats** like Delta Lake, Iceberg, or Hudi.

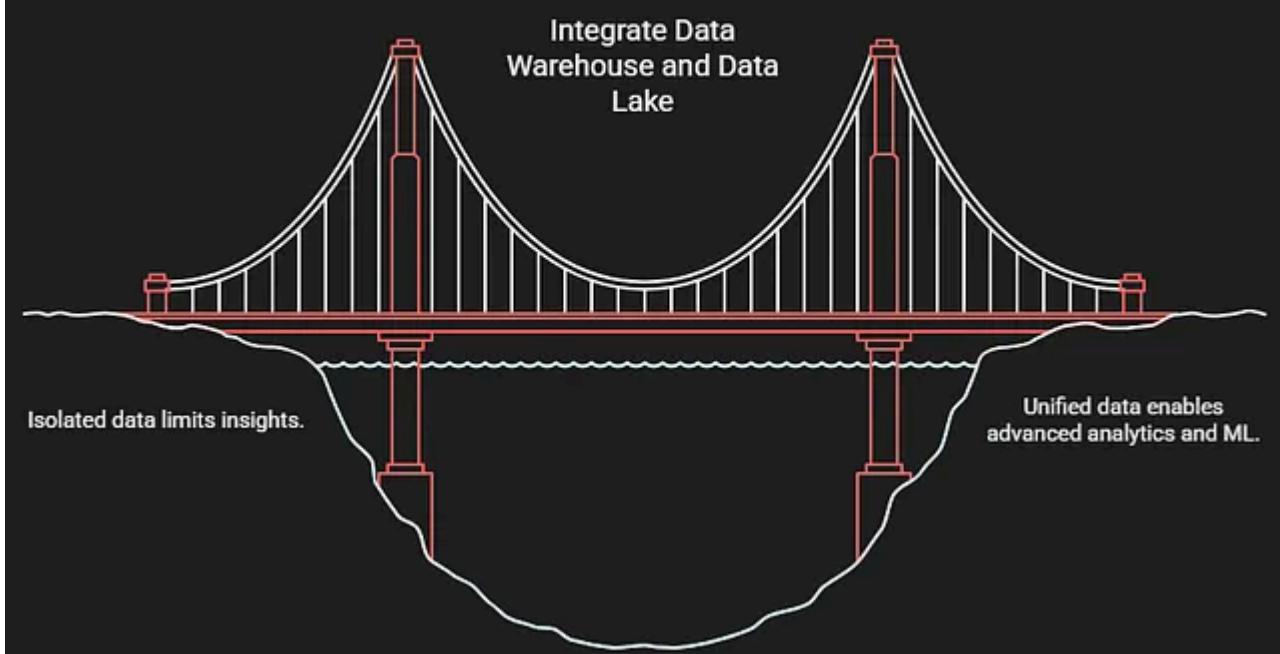
Meaning:

- You can **store data as Parquet** (cheap + efficient)
- But also **run analytics directly** with SQL, Pandas, Spark, or BI tools
- And **have consistent transactions**, even if multiple pipelines write simultaneously

If tomorrow a streaming API starts sending book ratings every minute, you simply **append to the same Delta/Iceberg table**.

No new warehouse staging schema needed.

## Data Lakehouse integrates Data Warehouse and Data Lake for comprehensive data management.



Layered diagram showing Data Warehouse features and Data Lake features combining into Data Lakehouse architecture. Source

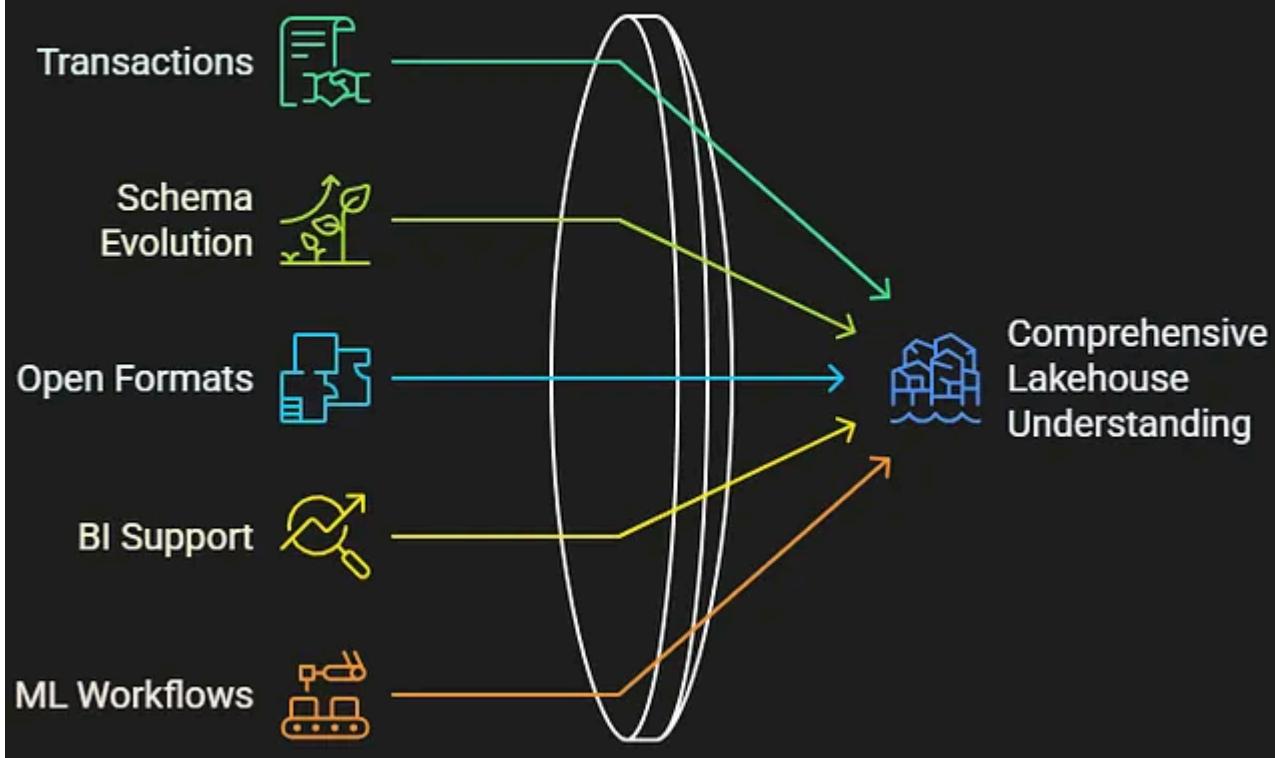
### 3. Key Characteristics of Lakehouses

Characteristic	Why It Matters (Books Example)
<b>ACID Transactions</b>	Two pipelines updating price and rating won't corrupt the table.
<b>Schema Governance</b>	If publisher sends wrong data, the write fails safely.
<b>Open Formats (Parquet, Delta, Iceberg)</b>	No vendor lock-in; data is portable.
<b>Concurrent Workloads</b>	BI dashboards, ML training, ingestion jobs can run simultaneously.
<b>Performance &amp; Scalability</b>	Data grows, compute scales independently.
<b>Supports Both BI &amp; ML</b>	Data scientists, analysts, and engineers share the same truth source.

Own table.

This is **the first architecture that does not force trade-offs**.

## Building the Lakehouse Foundation



Infographic listing the key characteristics of lakehouses with icons for transactions, schema evolution, open formats, BI support, and ML workflows. Source

## 4. The Medallion Architecture Explained

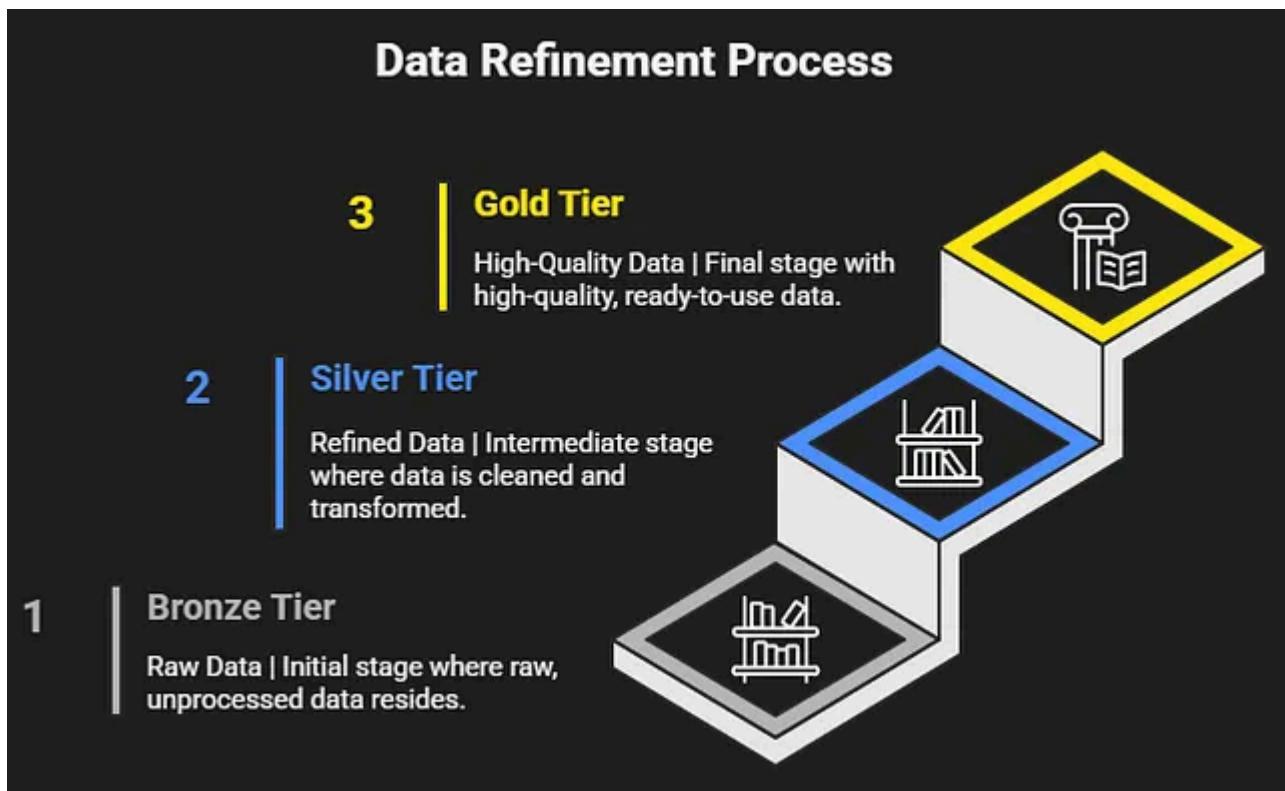
This is the common structure used inside lakehouses:

Layer	Purpose	Example with Books
<b>Bronze (Raw)</b>	Data as ingested, minimal checks.	Books from publishers in CSV, JSON, XML.
<b>Silver (Cleaned)</b>	Standardized + validated schema.	Unified books table with consistent fields and types.
<b>Gold (Curated)</b>	Business-ready model.	Star schema for reporting sales, ratings, trends.

Own table.

It is **not about renaming files**.

It is **a conceptual contract that creates trust**.



3-tier data flow diagram labeled Bronze → Silver → Gold (Medallion Architecture). Source

## 5. Zone-by-Zone Explanation

### Bronze Layer (Raw Ingestion)

- Raw files stored **as-is**
- Minimal validation (file readable? correct columns?)
- Often ingested using **Databricks Auto Loader / Spark Streaming**
- **Checkpoints** track progress to avoid duplicates

### Silver Layer (Clean & Standardize)

- Apply schema, normalize formats (dates, numbers)
- Remove corrupted or duplicated records
- No business logic

### Gold Layer (Analytics & BI)

- Build **dimensions and fact tables**
- Star or snowflake schema

- Used in dashboards, search, recommendations, forecasting



Bronze, Silver, and Gold lakehouse zones. Source

## 6. Challenges Still Exist

Challenge	Example
<b>Small file problem</b>	Streaming book ratings produces thousands of tiny files → need compaction.
<b>Low-latency access</b>	Real-time dashboard queries require optimized tables and caching.
<b>Random updates</b>	Book price corrections require MERGE support (Delta/Iceberg/Hudi solve this).

Own table.

A lakehouse **reduces** complexity, but it doesn't **eliminate** operational responsibility.

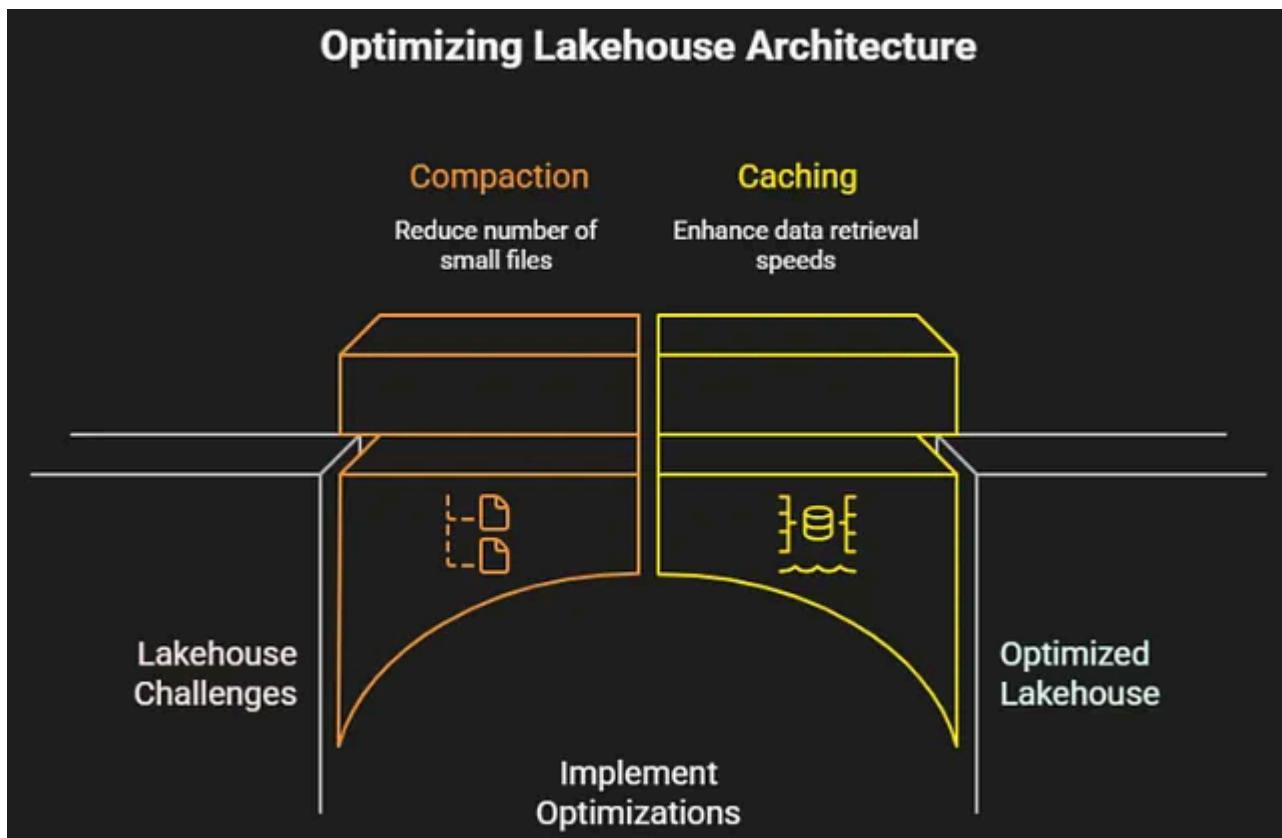


Diagram illustrating common lakehouse challenges, such as small files and latency, and solutions like compaction and caching. Source

## Closing Thought

The lakehouse is not a tool. It is a **commitment to treating data as an evolving asset**, not a liability.

If you work with datasets like *Books*, where format, volume, and speed of change vary every day, **you need guardrails, not just storage**.

And the lakehouse gives you exactly that.

## Want to Keep Improving in Data?

If you're into writing cleaner, more efficient Python, here are a few other stories I've written that readers have found helpful:  
 ## [97% of Python Devs Still Use pip + venv — uv Makes That Obsolete](#)

How this Rust-powered tool is redefining modern Python workflows.

[medium.com](https://medium.com)

[View original](#) ## [How I Used the Mann-Whitney U Test to Group Bank Cards by Behavior](#)

Discovering patterns in financial data with statistical tools

[ai.gopubby.com](http://ai.gopubby.com)

[View original](#) ## Still Using if-elif Chains? Let Me Show You a Better Way.

Why I now use pattern matching in every Python project.

[python.plainenglish.io](#)

[View original](#) ## 97% of Python Projects Could Be Cleaner With partial()

How one built-in tool can save you lines of code and mental energy.

[python.plainenglish.io](#)

[View original](#) ## How to Use Wikipedia API in Your Next Data Science Project

Using Wikipedia API to Add Context to Electric Vehicle Prediction Models

[medium.com](#)

[View original](#)