Last updated on ***Sep 4, 2025***

What is the medallion lakehouse architecture?

The medallion architecture describes a series of data layers that denote the quality of data stored in the lakehouse. Databricks recommends taking a multi-layered approach to building a single source of truth for enterprise data products.

This architecture guarantees atomicity, consistency, isolation, and durability as data passes through multiple layers of validations and transformations before being stored in a layout optimized for efficient analytics. The terms bronze (raw), silver (validated), and gold (enriched) describe the quality of the data in each of these layers.

Medallion architecture as a data design pattern

A medallion architecture is a data design pattern used to organize data logically. Its goal is to incrementally and progressively improve the structure and quality of data as it flows through each layer of the architecture (from Bronze ⇒ Silver ⇒ Gold layer tables). Medallion architectures are sometimes also referred to as *multi-hop architectures*.

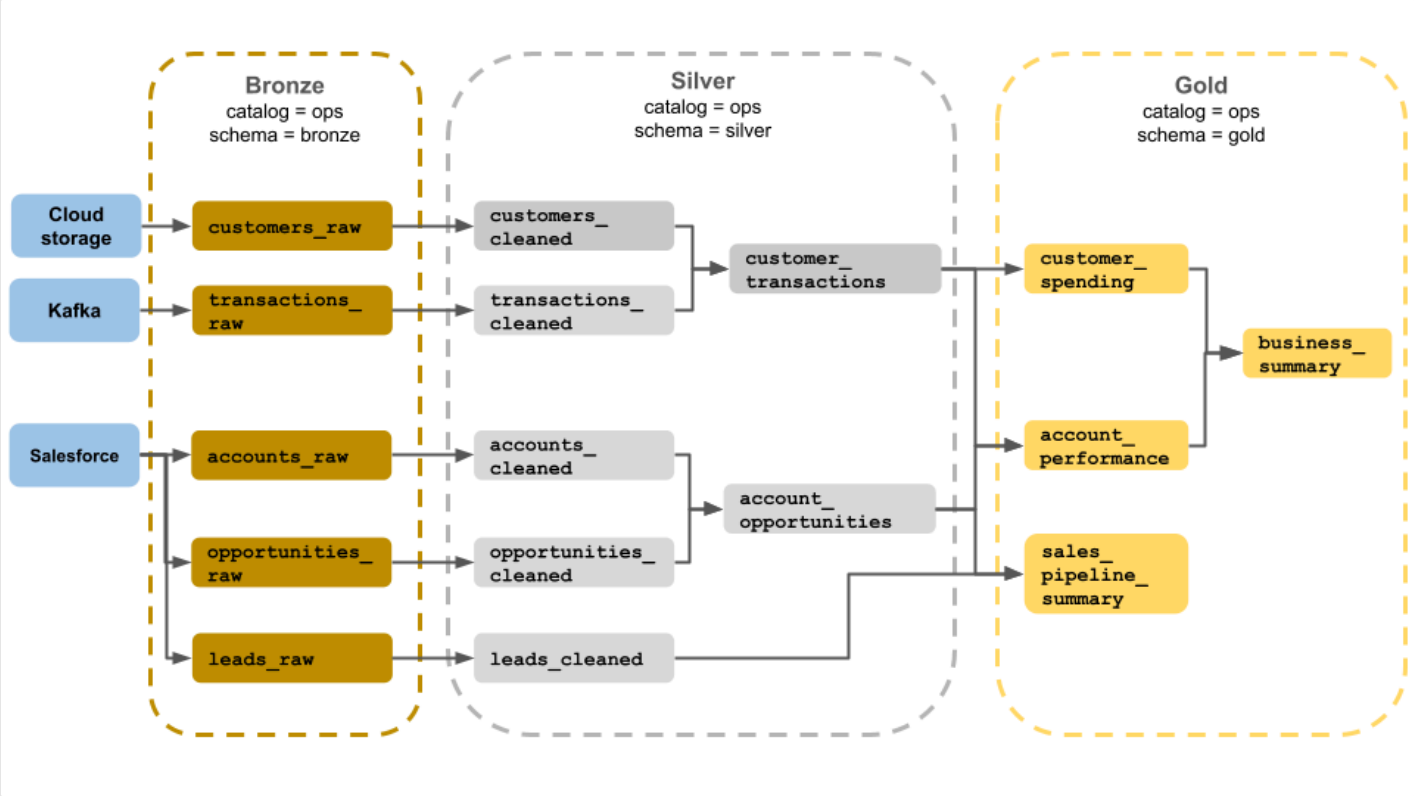
By progressing data through these layers, organizations can incrementally improve data quality and reliability, making it more suitable for business intelligence and machine learning applications.

Following the medallion architecture is a recommended best practice but not a requirement.

Question	Bronze	Silver	Gold
What happens in this layer?	Raw data ingestion	Data cleaning and validation	Dimensional modeling and aggregation
Who is the intended user?	• Data engineers • Data operations • Compliance and audit teams	• Data engineers • Data analysts (use the Silver layer for a more refined dataset that still retains detailed information necessary for in-depth analysis) • Data scientists (build models and perform advanced analytics)	• Business analysts and BI developers • Data scientists and machine learning (ML) engineers • Executives and decision makers • Operational teams

Example medallion architecture

This example of a medallion architecture shows bronze, silver, and gold layers for use by a business operations team. Each layer is stored in a different schema of the ops catalog.



- **Bronze layer** (`ops.bronze`): Ingests raw data from cloud storage, Kafka, and Salesforce. No data cleanup or validation is performed here.
- **Silver layer** (`ops.silver`): Data cleanup and validation are performed in this layer.
  - Data about customers and transactions is cleaned by dropping nulls and quarantining invalid records. These datasets are joined into a new dataset called `customer_transactions`. Data scientists can use this dataset for predictive analytics.
  - Similarly, accounts and opportunity datasets from Salesforce are joined to create `account_opportunities`, which is enhanced with account information.
  - The `leads_raw` data is cleaned in a dataset called `leads_cleaned`.
- **Gold layer** (`ops.gold`): This layer is designed for business users. It contains fewer datasets than silver and bronze.
  - `customer_spending`: Average and total spend for each customer.
  - `account_performance`: Daily performance for each account.
  - `sales_pipeline_summary`: Information about the end-to-end sales pipeline.
  - `business_summary`: Highly aggregated information for the executive staff.

Ingest raw data to the bronze layer

The bronze layer contains raw, unvalidated data. Data ingested in the bronze layer typically has the following characteristics:

- Contains and maintains the raw state of the data source in its original formats.
- Is appended incrementally and grows over time.
- Is intended for consumption by workloads that enrich data for silver tables, not for access by analysts and data scientists.
- Serves as the single source of truth, preserving the data's fidelity.
- Enables reprocessing and auditing by retaining all historical data.
- Can be any combination of streaming and batch transactions from sources, including cloud object storage (for example, S3, GCS, ADLS), message buses (for example, Kafka, Kinesis, etc.), and federated systems (for example, Lakehouse Federation).

Limit data cleanup or validation

Minimal data validation is performed in the bronze layer. To ensure against dropped data, Databricks recommends storing most fields as string, VARIANT, or binary to protect against unexpected schema changes. Metadata columns might be added, such as the provenance or source of the data (for example, `_metadata.file_name`).

Validate and deduplicate data in the silver layer

Data cleanup and validation are performed in silver layer.

Build silver tables from the bronze layer

To build the silver layer, read data from one or more bronze or silver tables, and write data to silver tables.

Databricks does not recommend writing to silver tables directly from ingestion. If you write directly from ingestion, you'll introduce failures due to schema changes or corrupt records in data sources. Assuming all sources are append-only, configure most reads from bronze as

streaming reads. Batch reads should be reserved for small datasets (for example, small dimensional tables).

The silver layer represents validated, cleaned, and enriched versions of the data. The silver layer:

- Should always include at least one validated, non-aggregated representation of each record. If aggregate representations drive many downstream workloads, those representations might be in the silver layer, but typically they are in the gold layer.
- Is where you perform data cleansing, deduplication, and normalization.
- Enhances data quality by correcting errors and inconsistencies.
- Structures data into a more consumable format for downstream processing.

Enforce data quality

The following operations are performed in silver tables:

- Schema enforcement
- Handling of null and missing values
- Data deduplication
- Resolution of out-of-order and late-arriving data issues
- Data quality checks and enforcement
- Schema evolution
- Type casting
- Joins

Start modeling data

It is common to start performing data modeling in the silver layer, including choosing how to represent heavily nested or semi-structured data:

- Use `VARIANT` data type.
- Use `JSON` strings.
- Create structs, maps, and arrays.
- Flatten schema or normalize data into multiple tables.

Power analytics with the gold layer

The gold layer represents highly refined views of the data that drive downstream analytics, dashboards, ML, and applications. Gold layer data is often highly aggregated and filtered for specific time periods or geographic regions. It contains semantically meaningful datasets that map to business functions and needs.

The gold layer:

- Consists of aggregated data tailored for analytics and reporting.
- Aligns with business logic and requirements.
- Is optimized for performance in queries and dashboards.

Align with business logic and requirements

The gold layer is where you'll model your data for reporting and analytics using a dimensional model by establishing relationships and defining measures. Analysts with access to data in gold should be able to find domain-specific data and answer questions.

Because the gold layer models a business domain, some customers create multiple gold layers to meet different business needs, such as HR, finance, and IT.

Create aggregates tailored for analytics and reporting

Organizations often need to create aggregate functions for measures like averages, counts, maximums, and minimums. For example, if your business needs to answer questions about total weekly sales, you could create a materialized view called `weekly_sales` that preaggregates this data so analysts and others don't need to recreate frequently used materialized views.

```
CREATE OR REPLACE MATERIALIZED VIEW weekly_sales AS
SELECT week,
       prod_id,
       region,
       SUM(units) AS total_units,
       SUM(units * rate) AS total_sales
```

```
FROM orders
GROUP BY week, prod_id, region
```

Optimize for performance in queries and dashboards

Optimizing gold-layer tables for performance is a best practice because these datasets are frequently queried. Large amounts of historical data are typically accessed in the sliver layer and not materialized in the gold layer.

Control costs by adjusting the frequency of data ingestion

Control costs by determining how frequently to ingest data.

Data ingestion frequency	**Cost**	**Latency**	**Declarative examples**
Continuous incremental ingestion	Higher	Lower	• Streaming Table using `spark.readStream` to ingest from cloud storage or message bus. • The pipeline that updates this streaming table runs continuously. • Structured Streaming code using `spark.readStream` in a notebook to ingest from cloud storage or message bus into a Delta table. • The notebook is orchestrated using a Databricks job with a continuous job trigger.
Triggered incremental	Lower	Higher	• Streaming Table ingesting from cloud storage or message bus using

Data ingestion frequency	Cost	Latency	Declarative examples
ingestion			`spark.readStream`. • The pipeline that updates this streaming table is triggered by the job's scheduled trigger or a file arrival trigger. • Structured Streaming code in a notebook with a `Trigger.Available` trigger. • This notebook is triggered by the job's scheduled trigger or a file arrival trigger.
Batch ingestion with manual incremental ingestion	Lower	Highest, because of infrequent runs.	• Streaming Table ingest from cloud storage using `spark.read`. • Does not use Structured Streaming. Instead, use primitives like partition overwrite to update an entire partition at one time. • Requires extensive upstream architecture to set up the incremental processing, which allows for a cost similar to Structured Streaming reads/writes. • Also requires partitioning source data by a `datetime` field and then processing all records from that partition into the target.