loading bronze silver golde layer in medallion architecture using databricks The Medallion Architecture in Databricks organizes data into three layers—Bronze, Silver, and Gold—to progressively improve data quality and structure in a lakehouse. Below is a guide to loading data into these layers using Databricks, with practical steps and best practices based on the provided context.

Overview of Medallion Architecture Layers

Bronze Layer: Stores raw, unprocessed data as ingested from source systems, preserving its original format for auditing, lineage, and reprocessing. Silver Layer: Cleansed, validated, and standardized data from the Bronze layer, suitable for analytics and machine learning. Gold Layer: Aggregated, enriched, and optimized data for business intelligence, reporting, and advanced analytics.

Steps to Load Data into Bronze, Silver, and Gold Layers 1. Loading the Bronze Layer (Raw Data Ingestion) The Bronze layer captures raw data from external sources with minimal transformation to maintain fidelity. Steps:

Ingest Data: Use Databricks tools like Autoloader for streaming or batch ingestion from sources such as cloud storage (e.g., S3, ADLS, GCS), Kafka, or databases. Preserve Original Format: Store data in Delta Lake format to leverage ACID transactions and schema evolution. Keep data types as-is (often as strings) to avoid data loss due to schema changes. Add Metadata: Include columns like ingestion timestamp, source file name, or process ID for traceability. Best Practices:

Use Delta tables for versioning and reprocessing capabilities. Avoid heavy transformations; focus on quick Change Data Capture (CDC). Store data in a dedicated catalog (e.g., bronze\_catalog.raw\_data) for governance.

Example (PySpark): pythonfrom pyspark.sql.functions import current\_timestamp

# Read raw JSON data from cloud storage df\_raw = spark.read.json("/mnt/data/raw\_data.json")

# Add ingestion timestamp df\_bronze = df\_raw.withColumn("ingest\_time", current\_timestamp())

# Write to Bronze layer as Delta table df\_bronze.write.format("delta") \ .mode("append") \ .saveAsTable("bronze\_catalog.raw\_data.raw\_table") Notes:

Use Autoloader for streaming ingestion to handle schema evolution and rescue mode. For batch processing, read CSV/JSON/Parquet files and store as Parquet or Delta.

2. Loading the Silver Layer (Cleansed and Validated Data) The Silver layer transforms Bronze data by cleaning, deduplicating, and standardizing it for analytics. Steps:

Read from Bronze: Load data from Bronze tables using spark.read or streaming reads for append-only sources. Transform Data:

Enforce schema and handle nulls. Deduplicate records (e.g., based on primary keys). Normalize or flatten nested structures. Apply minimal transformations (ELT approach) to maintain agility.

Write to Silver: Save as Delta tables in a separate catalog (e.g., silver\_catalog.processed\_data). Best Practices:

Avoid direct ingestion to Silver to prevent schema change issues. Use Delta Live Tables (DLT) with @dlt.expect\_all for data quality checks. Perform incremental processing with merges to handle updates efficiently.

Example (PySpark): pythonfrom delta.tables import DeltaTable

# Read from Bronze df\_bronze = spark.table("bronze\_catalog.raw\_data.raw\_table")

# Clean and transform (e.g., filter, deduplicate, rename columns) df\_silver = (df\_bronze .filter("status = 'active'") .dropDuplicates(["id"]) .withColumnRenamed("old\_column", "new\_column"))

# Write to Silver layer df\_silver.write.format("delta") \ .mode("overwrite") \ .saveAsTable("silver\_catalog.processed\_data.silver\_table")

# Incremental merge for updates silver\_table = DeltaTable.forName(spark, "silver\_catalog.processed\_data.silver\_table") bronze\_updates = spark.read.format("delta").load("/mnt/data/new\_raw\_data") silver\_table.alias("silver").merge( bronze\_updates.alias("bronze"), "silver.id = bronze.id" ).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute() Notes:

Use streaming reads for append-only Bronze tables to reduce latency. Apply 3rd-Normal Form or Data Vault-like models for write performance.

3. Loading the Gold Layer (Business-Ready Data) The Gold layer aggregates and optimizes Silver data for reporting and analytics, often using dimensional models. Steps:

Read from Silver: Load cleansed data from Silver tables. Transform and Aggregate:

Perform joins, aggregations (e.g., sums, averages), and business logic. Create fact and dimension tables for star schemas or data marts.

Write to Gold: Save as Delta tables in a catalog (e.g., gold\_catalog.analytics) optimized for read performance. Best Practices:

Use Z-Ordering to optimize query performance on frequently filtered columns. Create materialized views for common aggregates to reduce query costs. Tailor data to business needs, collaborating with stakeholders to validate metrics.

Example (PySpark): python# Read from Silver df\_silver = spark.table("silver\_catalog.processed\_data.silver\_table")

# Aggregate data (e.g., total sales per category) df\_gold = (df\_silver .groupBy("category") .agg({"sales": "sum"}) .withColumnRenamed("sum(sales)", "total\_sales"))

# Write to Gold layer df\_gold.write.format("delta") \ .mode("overwrite") \ .saveAsTable("gold\_catalog.analytics.gold\_table")

# Optimize with Z-Ordering (SQL) spark.sql("OPTIMIZE gold\_catalog.analytics.gold\_table ZORDER BY (category)") Notes:

Use Delta Live Tables (DLT) to automate pipeline updates and ensure dependencies. Schedule updates with Databricks jobs for regular refreshes.

Additional Best Practices

Catalog Separation: Use separate catalogs (bronze\_catalog, silver\_catalog, gold\_catalog) for governance and access control. Incremental Processing: Implement merges to process only new or updated data, reducing costs. Data Quality: Use DLT expectations to enforce quality checks in Silver and Gold layers. Monitoring: Leverage Databricks tools to monitor pipeline performance and data quality. Naming Conventions: Use clear schema names (e.g., raw\_data, processed\_data, analytics) to distinguish layers, especially for managed tables.

Example End-to-End Pipeline Here’s a simplified workflow using Databricks notebooks:

Bronze Notebook (Batch Processing - Bronze.py):

Read CSV/JSON from cloud storage. Save as Delta table in bronze\_catalog.

Silver Notebook (Batch Processing - Silver.py):

Read from Bronze, clean, and deduplicate. Save to silver\_catalog with incremental merges.

Gold Notebook (Batch Processing - Gold.py):

Read from Silver, aggregate, and optimize. Save to gold\_catalog for reporting.

Workflow Setup:

Use Databricks Workflows to orchestrate notebooks. Schedule jobs for periodic updates or use continuous streaming with Autoloader.

Tools and Features in Databricks

Autoloader: Handles streaming ingestion with schema evolution. Delta Live Tables (DLT): Automates pipeline creation and data quality enforcement. Delta Lake: Ensures ACID transactions, versioning, and performance optimizations. Unity Catalog: Manages governance and access control across layers.

Considerations

Cost vs. Latency: Choose ingestion frequency (continuous, triggered, or batch) based on cost and latency needs. Schema Evolution: Handle schema changes in Bronze using Autoloader’s rescue mode. Reprocessing: Bronze layer’s raw data enables reprocessing without hitting source systems.

For further details, explore Databricks documentation or community resources:

Databricks Medallion Architecture Guide: https://www.databricks.com/glossary/medallion-architecture Pricing for SuperGrok or API services: https://x.ai/grok or https://x.ai/api

This approach ensures a scalable, reliable, and governed data pipeline in Databricks using the Medallion Architecture.27 𝕏 posts20 web pages2 / 24.5sloading bronze silver golde layer in medallion architecture using databricks with cstomers data fileThe Medallion Architecture in Databricks organizes data into Bronze, Silver, and Gold layers to process and refine customer data efficiently in a lakehouse setup. Below is a step-by-step guide to load a customers data file (e.g., CSV, JSON, or Parquet) into these layers using Databricks, including code examples and best practices tailored to customer data.

Assumptions

Data Source: A customer data file (e.g., customers.csv) stored in cloud storage (e.g., S3, ADLS, GCS). File Structure: Contains columns like customer\_id, name, email, address, signup\_date, status. Objective: Ingest raw customer data into Bronze, clean and standardize it in Silver, and create aggregated views in Gold for analytics (e.g., customer segmentation or activity metrics).

Overview of Medallion Architecture

Bronze Layer: Raw, unprocessed customer data as ingested, stored in Delta Lake for lineage and reprocessing. Silver Layer: Cleaned, validated, and standardized customer data, ready for analytics. Gold Layer: Aggregated and enriched customer data for business insights, such as reports or dashboards.

Steps to Load Customers Data into Bronze, Silver, and Gold Layers 1. Loading the Bronze Layer (Raw Customer Data) The Bronze layer captures the raw customer data file with minimal processing to preserve its original state. Steps:

Ingest the File: Use Databricks Autoloader for streaming ingestion or spark.read for batch processing from cloud storage. Add Metadata: Include ingestion timestamp and source file name for traceability. Save as Delta Table: Store in bronze\_catalog to leverage Delta Lake’s ACID transactions and schema evolution.

Best Practices:

Use Delta Lake for versioning and scalability. Avoid transformations; keep data as-is for auditability. Use Autoloader for incremental ingestion and schema changes.

Example (PySpark): pythonfrom pyspark.sql.functions import current\_timestamp, input\_file\_name

# Configure cloud storage path input\_path = "/mnt/data/customers.csv"

# Read raw CSV file df\_raw = spark.read.format("csv") \ .option("header", "true") \ .option("inferSchema", "true") \ .load(input\_path)

# Add ingestion metadata df\_bronze = df\_raw.withColumn("ingest\_time", current\_timestamp()) \ .withColumn("source\_file", input\_file\_name())

# Write to Bronze layer as Delta table df\_bronze.write.format("delta") \ .mode("append") \ .saveAsTable("bronze\_catalog.customers.raw\_customers")

# For streaming ingestion with Autoloader df\_stream = spark.readStream.format("cloudFiles") \ .option("cloudFiles.format", "csv") \ .option("header", "true") \ .option("inferSchema", "true") \ .load(input\_path) df\_stream = df\_stream.withColumn("ingest\_time", current\_timestamp()) df\_stream.writeStream.format("delta") \ .outputMode("append") \ .start("bronze\_catalog.customers.raw\_customers") Notes:

Replace /mnt/data/customers.csv with your storage path (e.g., s3://bucket/customers/). Autoloader handles schema evolution and new files automatically.

2. Loading the Silver Layer (Cleaned and Standardized Customer Data) The Silver layer transforms raw customer data by cleaning, deduplicating, and standardizing it for analytics. Steps:

Read from Bronze: Load data from the Bronze Delta table. Transform Data:

Handle missing or invalid values (e.g., null emails). Deduplicate based on customer\_id. Standardize formats (e.g., lowercase emails, parse dates).

Save to Silver: Store as a Delta table in silver\_catalog.

Best Practices:

Use Delta Live Tables (DLT) for automated data quality checks. Implement incremental merges for updates. Enforce schema to ensure consistency.

Example (PySpark): pythonfrom pyspark.sql.functions import col, lower, to\_date from delta.tables import DeltaTable

# Read from Bronze df\_bronze = spark.table("bronze\_catalog.customers.raw\_customers")

# Clean and standardize df\_silver = (df\_bronze .filter(col("customer\_id").isNotNull()) # Remove null IDs .dropDuplicates(["customer\_id"]) # Deduplicate .withColumn("email", lower(col("email"))) # Standardize email .withColumn("signup\_date", to\_date(col("signup\_date"), "yyyy-MM-dd")) # Parse dates .filter(col("status").isin(["active", "inactive"]))) # Valid statuses

# Write to Silver layer df\_silver.write.format("delta") \ .mode("overwrite") \ .saveAsTable("silver\_catalog.customers.processed\_customers")

# Incremental merge for updates silver\_table = DeltaTable.forName(spark, "silver\_catalog.customers.processed\_customers") bronze\_updates = spark.read.format("delta").load("bronze\_catalog.customers.raw\_customers") silver\_table.alias("silver").merge( bronze\_updates.alias("bronze"), "silver.customer\_id = bronze.customer\_id" ).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute() DLT Example (Optional): pythonimport dlt from pyspark.sql.functions import col

@dlt.table(name="processed\_customers", table\_properties={"quality": "silver"}) @dlt.expect\_all({ "valid\_id": "customer\_id IS NOT NULL", "valid\_email": "email IS NOT NULL AND email LIKE '%@%.%'" }) def silver\_customers(): df = dlt.read("raw\_customers") return (df .filter(col("customer\_id").isNotNull()) .dropDuplicates(["customer\_id"]) .withColumn("email", lower(col("email"))) .withColumn("signup\_date", to\_date(col("signup\_date"), "yyyy-MM-dd"))) Notes:

Adjust transformations based on your data (e.g., clean addresses or validate formats). Use DLT for automated pipelines and quality checks.

3. Loading the Gold Layer (Business-Ready Customer Data) The Gold layer aggregates and enriches Silver customer data for reporting, such as customer counts by status or signup trends. Steps:

Read from Silver: Load cleaned data from the Silver Delta table. Transform and Aggregate:

Create metrics like customer counts by status or signup month. Build fact/dimension tables for analytics (e.g., star schema).

Save to Gold: Store as Delta tables in gold\_catalog with optimizations for queries.

Best Practices:

Use Z-Ordering for query performance on common filters (e.g., signup\_date). Create materialized views for frequent queries. Validate metrics with business stakeholders.

Example (PySpark): pythonfrom pyspark.sql.functions import year, month, count

# Read from Silver df\_silver = spark.table("silver\_catalog.customers.processed\_customers")

# Aggregate: Customer counts by status and signup year df\_gold = (df\_silver .groupBy("status", year("signup\_date").alias("signup\_year")) .agg(count("customer\_id").alias("customer\_count")))

# Write to Gold layer df\_gold.write.format("delta") \ .mode("overwrite") \ .saveAsTable("gold\_catalog.customers.customer\_metrics")

# Optimize with Z-Ordering spark.sql("OPTIMIZE gold\_catalog.customers.customer\_metrics ZORDER BY (signup\_year)") Chart Example (Customer Counts by Status): If you want a chart of customer counts by status, confirm and I’ll generate one. For now, here’s how the data might look: python# Sample aggregation for charting df\_chart = spark.table("gold\_catalog.customers.customer\_metrics").groupBy("status").agg(count("customer\_id").alias("count")) df\_chart.show() # Example output: # +--------+-----+ # | status |count| # +--------+-----+ # | active | 500 | # |inactive| 200 | # +--------+-----+

End-to-End Workflow

Bronze Notebook (Ingest\_Customers\_Bronze.py):

Read customers.csv using Autoloader or batch. Save to bronze\_catalog.customers.raw\_customers.

Silver Notebook (Process\_Customers\_Silver.py):

Read from Bronze, clean, and deduplicate. Save to silver\_catalog.customers.processed\_customers.

Gold Notebook (Aggregate\_Customers\_Gold.py):

Read from Silver, aggregate (e.g., by status or signup year). Save to gold\_catalog.customers.customer\_metrics.

Orchestration:

Use Databricks Workflows to chain notebooks. Schedule daily/weekly updates or run continuously with streaming.

Additional Best Practices

Catalog Management: Use Unity Catalog to separate bronze\_catalog, silver\_catalog, and gold\_catalog for governance. Incremental Processing: Use Delta merges for efficiency with large customer datasets. Data Quality: Apply DLT expectations (e.g., valid emails, non-null IDs) in Silver. Performance: Optimize Gold tables with ZORDER or partitioning by signup\_year. Monitoring: Track pipeline health using Databricks job metrics.

Sample Customers Data Pipeline Input File (customers.csv): csvcustomer\_id,name,email,address,signup\_date,status 1,John Doe,john.doe@email.com,123 Main St,2023-01-15,active 2,Jane Smith,JANE@EMAIL.COM,456 Oak Ave,2023-02-10,inactive 3,John Doe,john.doe@email.com,789 Pine Rd,2023-03-05,active Bronze Output:

Delta table with raw data + ingest\_time, source\_file.

Silver Output:

Deduplicated (remove duplicate customer\_id=1), standardized emails (lowercase), parsed dates.

Gold Output:

Aggregated table, e.g., status=active, signup\_year=2023, customer\_count=2.

Tools in Databricks

Autoloader: For incremental file ingestion. Delta Live Tables (DLT): Automates pipelines and quality checks. Delta Lake: Ensures reliability and performance. Unity Catalog: Governs data access across layers.