Data engineering report

Stock Market Analytics Pipeline

An end-to-end cloud pipeline: six months of daily prices for five large-cap stocks ingested to S3, summarized by a serverless function calling a foundation model on AWS Bedrock, stored in a Delta Lake table on Databricks, and reported with Databricks SQL.

Prepared by	Leah Hall, Leah Kat Insights
Date	April 2026
Contact	leah@leahkatinsights.com, leahkatinsights.com

Table of Contents

Introduction . . . 3

1 Executive summary . . . 3

Key points . . . 3

2 Background and objective . . . 3

3 Architecture and method . . . 4

Pipeline . . . 4

Data flow . . . 4

Design decisions . . . 4

4 Results . . . 5

5 Limitations . . . 5

6 Next steps . . . 6

A Technical stack and repository . . . 6

B Glossary . . . 7

About the author . . . 7

Introduction

This report documents a cloud data pipeline built as a portfolio project: what it does, how it is designed, and what the reporting layer produces. It is written for a technical reader who wants to evaluate the architecture, not for an investor.

All data is public market data from Yahoo Finance. Nothing in this report is investment advice.

The code, including the Lambda function, the Delta load, and the SQL, is public on GitHub.

1. Executive summary

Stock analytics is a common first portfolio project. This version runs on real cloud infrastructure end to end, uses a language model as the analysis layer, and stores the data in a lakehouse with ACID transactions and time travel instead of a local file.

Key points

4	**Four stages, each on managed infrastructure.** Python ingests to S3, an S3 event triggers a Lambda that calls Bedrock, Databricks loads a Delta table, and Databricks SQL reports on it.
ACID	**Versioned, transactional storage.** Every load into Delta Lake is a version with schema enforcement, so the table can be queried or restored at any prior state.
1	**Model choice is configuration.** The Lambda uses the Bedrock Converse API, which is model-agnostic. The model ID is an environment variable, not code.
5	**Five analytical queries.** Latest close, six-month range, monthly trend, best and worst trading day per ticker, and month-over-month change, all expressed with SQL window functions.

2. Background and objective

Most introductory stock projects download a CSV, plot it in a notebook, and stop. The goal here was to take the same simple domain and build it the way a data engineering team would: automated ingestion to object storage, event-driven processing, a lakehouse table with transactional guarantees, and a SQL reporting layer that other people can query.

The language-model step is deliberately small. A serverless function sends each new file to a foundation model and writes the returned summary back to storage. The point is to show how an LLM slots into a pipeline as one more processing stage, with the model itself swappable.

3. Architecture and method

Pipeline

Table 1. Pipeline stages

Stage	What happens	Output
Ingest	Python pulls six months of daily open, high, low, close, and volume for the five tickers from the Yahoo Finance API, normalizes column names, and uploads one CSV to S3.	s3://bucket/data/stocks_6mo.csv
Analyze	An S3 ObjectCreated event triggers the Lambda for any CSV under data/. It sends the first 50 rows to a Bedrock model through the Converse API and writes the summary back.	s3://bucket/analysis/*_analysis.txt
Store	Databricks reads the CSV from a volume or external location and writes a Delta Lake table in Unity Catalog with schema enforcement and time travel.	Delta table
Report	Databricks SQL computes trends, ranges, best and worst days, and month-over-month change, rendered as dashboards and the chart in Section 4.	SQL results, chart

Data flow

```
Yahoo Finance
  -> ingest/pull_stock_data.py        -> s3://bucket/data/stocks_6mo.csv
  -> S3 event -> lambda_function.py   -> Bedrock Converse API
                                      -> s3://bucket/analysis/*_analysis.txt
  -> databricks/01_load_to_delta.py   -> Delta table (Unity Catalog, time travel)
  -> databricks/02_sql_analytics.sql  -> trends, ranges, window functions
  -> charts/stock_chart.py            -> exhibits/stock-chart.png
```

Design decisions

- **Event-driven analysis.** The Lambda is triggered by the object write rather than a schedule, so any new file is processed without an orchestrator.
- **Converse API over a model-specific API.** The request shape is the same for every model on Bedrock, so switching models is a configuration change.
- **Delta over CSV for the serving layer.** ACID writes, schema enforcement, and time travel make each load reproducible and auditable.
- **Local fallback.** With no bucket configured, the ingest script writes locally so the pipeline can be run and tested without AWS credentials.

4. Results

The reporting layer answers the questions an analyst asks first of a price series. Table 2 lists the queries and the SQL features they use; Figure 1 is the monthly trend query rendered as a chart.

Table 2. Reporting queries

Query	Output	SQL feature
Latest close	Most recent closing price per ticker	Ordering and limit per partition
Six-month range	High and low close per ticker over the period	Aggregation
Monthly trend	Average closing price per ticker per month	Date truncation, grouping
Best and worst day	Largest gain and loss per ticker	RANK() OVER a ticker partition
Month-over-month	Change in monthly average versus the prior month	LAG() OVER

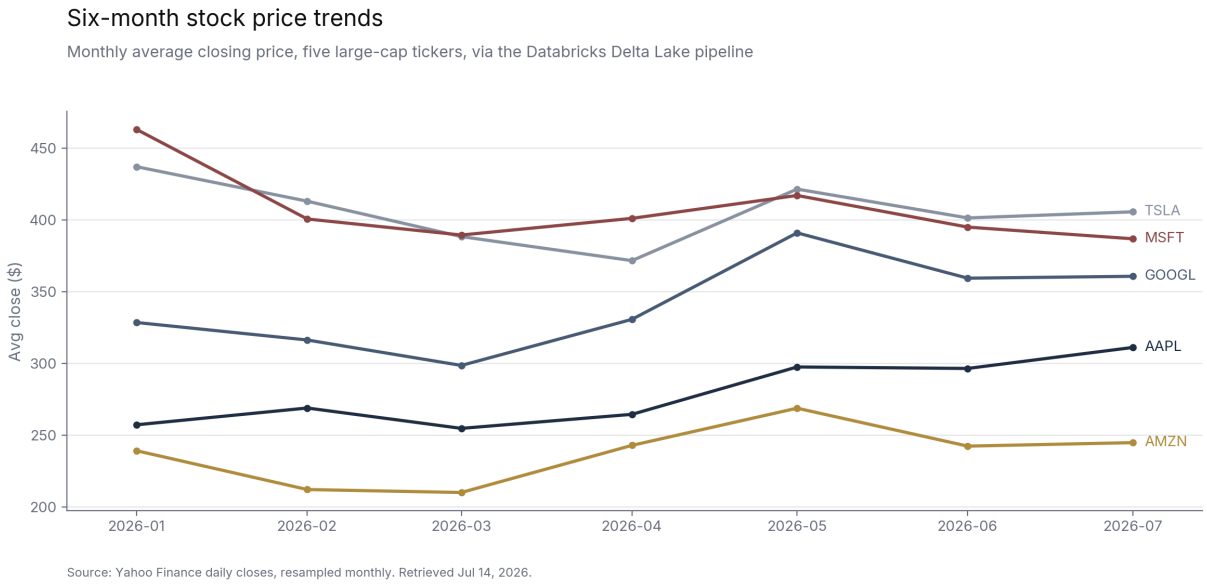


Figure 1. Monthly average closing price, five tickers, six months

Source: Yahoo Finance daily closes, resampled monthly in Databricks SQL.

5. Limitations

- **Short window, few tickers.** Six months of five large caps is enough to exercise the pipeline, not to support any market conclusion.
- **The model summary is unvalidated.** The Bedrock output is stored as text for reference. It is not scored, and nothing downstream depends on it.

- **Sample sent to the model.** The Lambda sends the first 50 rows of each file, a deliberate limit on cost and payload size.
- **Not investment advice.** The project demonstrates infrastructure; it makes no claim about future prices.

6. Next steps

1. Schedule the ingest script on EventBridge so the pipeline refreshes daily without manual runs.
2. Load incrementally into Delta with MERGE so each day appends rather than reloads.
3. Add a Databricks job that runs the load and the SQL on a schedule and publishes the dashboard.
4. Widen coverage to a sector or index once the incremental load is in place.

Appendix A. Technical stack and repository

Table 3. Stack

Layer	Tool
Source	Yahoo Finance via yfinance
Storage	AWS S3; Delta Lake on Databricks
Compute	AWS Lambda; AWS Bedrock (Converse API)
Analytics	Databricks SQL
Charts	matplotlib
Tests	pytest, Lambda helper tests

Repository layout

```
ingest/       pull_stock_data.py
lambda/       lambda_function.py (S3 trigger, Bedrock Converse, write-back)
databricks/   01 Delta load, 02 SQL analytics
charts/       stock_chart.py and the shared exhibit style
exhibits/     banner and chart
tests/        Lambda helper tests
```

Appendix B. Glossary

ACID	Atomic, consistent, isolated, durable: the transactional guarantees Delta Lake adds to files in object storage.
Bedrock	AWS managed service for calling foundation models through one API.
Converse API	Bedrock's model-agnostic request format for chat-style calls.
Delta Lake	An open table format on top of Parquet that adds transactions, schema enforcement, and versioning.
Lakehouse	A data platform that applies warehouse-style tables and governance to data lake storage.
Time travel	Querying or restoring a Delta table as it existed at an earlier version.
Window function	A SQL function such as RANK() or LAG() that computes over a set of rows related to the current row.

About the author

Leah Hall is a data and analytics professional based in Seal Beach, California. She builds data pipelines, forecasting models, and financial analyses for ecommerce, finance, and hospitality, and publishes the work at leahkatinsights.com.

Project page	leahkatinsights.com/projects.html#project-01
Source code	github.com/leahkatinsights/stock-analytics-lakehouse
Email	leah@leahkatinsights.com
Website	leahkatinsights.com
LinkedIn	linkedin.com/in/leah-hall
GitHub	github.com/leahkatinsights

Portfolio project on public market data. Not investment advice. Code is on GitHub.