CLOUD & INFRASTRUCTURE

Data Lakes on AWS

Designing a cost-effective lakehouse with S3, Glue and Athena

Houssam Kodad

PDF · DATAFORGE BOOKS

Questions about this sample or the full edition: **support@dataforgebooks.com**

CONTENTS

Table of Contents

CHAPTER 01

A Lakehouse on AWS

AWS gives you a hundred ways to build a data platform and very little opinion about which to choose. That freedom is both a gift and a trap: it is entirely possible to assemble a lake that is slow, expensive and ungoverned without ever straying from well-documented, officially-blessed services. This book takes a clear, opinionated position so that you end up with a lakehouse that stays fast and cheap rather than one that quietly becomes a swamp.

This chapter defines the destination. We explain the lakehouse pattern on AWS, introduce the S3-Glue-Athena core that does most of the work, place Redshift in context so you know when to reach for it, and preview the design decisions — storage layout, file formats, partitioning, governance — that determine whether the platform delights or disappoints. It is the conceptual foundation the hands-on chapters build on.

The recurring theme is that on AWS, cost and performance are mostly decided by physical design choices you make early and live with for a long time. How you lay out S3, what file format you pick, how you partition — these are not details, because Athena bills by the byte scanned and a careless layout can make every query ten times more expensive than it needs to be. We treat those choices with the seriousness they deserve.

Lake, Warehouse, Lakehouse

Three architectures compete for your data, and understanding their trade-offs is the foundation of every later decision. A data lake stores raw files cheaply and flexibly but offers little structure or guarantees, so it easily decays into a swamp. A data warehouse offers structure, performance and transactions but at higher cost and with data locked inside its engine. The lakehouse aims to combine the best of both: warehouse-like reliability and performance directly on cheap, open files in object storage.

On AWS, the lakehouse means keeping your data as open columnar files in S3 while layering table semantics and SQL on top. You get the warehouse's convenience without surrendering the lake's cost profile and openness — provided you make the right design choices. This book is largely a guide to making them, because the lakehouse rewards good design and punishes neglect more sharply than either of the architectures it replaces.

The S3-Glue-Athena Core

Three services form the backbone of a serverless lakehouse on AWS, and they work together so cleanly that they deserve to be understood as a unit. S3 holds the data as partitioned columnar files. The Glue Data Catalog records what tables exist, their schemas, and where their files live. Athena runs SQL over those files on demand, with no cluster to manage, billing only for the data each query scans.

The beauty of this core is that it is fully serverless: there is nothing to provision, patch or keep running, and it scales from a gigabyte to a petabyte without a change in approach. The constraint that shapes everything is Athena's pricing model — you pay for bytes scanned — which means the physical layout of your data in S3 is not a tuning afterthought but the primary determinant of your bill. We design around that fact from the first chapter.

```
-- Partition projection lets Athena prune by date with no metastore lookup.
SELECT user_id, COUNT(*) AS events
FROM   events
WHERE  dt BETWEEN DATE '2026-01-01' AND DATE '2026-01-31'
GROUP  BY user_id;
-- Scans one month of partitions, not the whole history — pennies, not euros.
```

Designing S3 Storage

How you organise S3 is the single most consequential design decision in an AWS lakehouse, because it determines how much data every query must read. A good layout uses a clear prefix structure, partitions data by the columns most often filtered on — usually date — and keeps the door open for lifecycle rules that move old data to cheaper storage tiers automatically.

The partitioning strategy in particular repays careful thought. Partition by a column that queries actually filter on, and Athena reads only the relevant slices; partition by the wrong column, or too finely, and you either scan everything or drown in millions of tiny files. We work through the trade-offs in detail, because partitioning is where most AWS cost savings — and most self-inflicted cost disasters — originate.

File Formats and the Small-File Problem

The file format you choose has an outsized effect on cost and speed. Columnar formats like Parquet store each column together and compressed, so a query that touches three columns of a hundred-column table reads roughly three percent of the data. Choosing Parquet over raw CSV or JSON is often the single biggest, easiest win available in a lakehouse, frequently cutting both scan cost and query time by an order of magnitude.

The counterpart problem is file sizing. Thousands of tiny files cripple query performance, because the engine spends its time opening files rather than reading data, while a few enormous files limit parallelism. The sweet spot is files of roughly a hundred megabytes to a gigabyte, which means you must actively compact small files as they accumulate. We treat compaction as a routine maintenance task, not an emergency.

The Glue Data Catalog

The Glue Data Catalog is the metadata layer that turns a pile of files in S3 into queryable tables. It records each table's schema, partitions and location, and it is shared across AWS analytics services, so a table defined once is visible to Athena, Spark on EMR, and Redshift Spectrum alike. It is the connective tissue of the whole platform.

There is a recurring choice between letting Glue crawlers infer schemas automatically and defining schemas explicitly. Crawlers are convenient for exploration but can guess wrong and drift; explicit definitions are more work but more reliable and reviewable for production tables. We come down on the side of explicit control where it matters, and explain how to manage schema evolution so a new column does not break every downstream query.

Open Table Formats

Plain files in S3 lack the transactions, updates and deletes that real workloads eventually need — you cannot easily correct a record or comply with a deletion request across millions of immutable files. Open table formats like Apache Iceberg add these capabilities on top of object storage, bringing reliable updates, deletes, schema evolution and even time travel to the lakehouse without locking you into a proprietary engine.

Iceberg and its peers are increasingly the default rather than an advanced option, because they solve problems that every maturing lakehouse hits. We cover when adopting a table format is worth the added complexity, how it changes your ingestion and maintenance, and how it interacts with the S3 layout and Glue catalog decisions made earlier — so you can adopt it deliberately rather than cargo-culting it.

Governance and Cost Control

A lakehouse that anyone can read is a liability, and Lake Formation provides the fine-grained access control — down to rows and columns — that a serious platform requires, layered over the IAM permissions that gate the underlying S3 and Glue resources. Designing this access model thoughtfully

is what lets you share data widely without sharing it indiscriminately.

Cost control runs through everything on AWS, and the levers are now familiar: scan less by partitioning and choosing columnar formats, store cheaper by tiering old data, and monitor spend so surprises are caught early. The chapters ahead take each piece introduced here — storage, formats, catalog, table formats, governance — and build it into a complete, cost-aware lakehouse you can operate with confidence.

This is a free sample

You've reached the end of the sample chapter.

Get the complete book — every chapter, fully worked — at **dataforgebooks.com**.

FULL EDITION · 256 PAGES · PDF

Read the full title at **dataforgebooks.com**

Questions? support@dataforgebooks.com