upsolver

VS

APACHE Spark™

Building a Cloud Data Lake

Table of Contents

Apache Spark promised a bright future for data lakes. Has it lived up to expectations?

Since the middle of the last decade, Apache Spark has become the de-facto standard for large-scale distributed data processing. This open-source framework leveraged in-memory MapReduce and promised to simplify and accelerate data science projects, which had previously been mired by the difficulties of the Hadoop framework from which Spark evolved. In a 2016 Stack Overflow survey, Spark was rated as the highest paying and 2nd most trending technology, indicating the promise that the development community saw in it.

However, Apache Spark is now over a decade old and no longer the new kid on the block, but an established part of the big data development toolkit - either in its original open-source format or in the various proprietary tools it has progenated - including Databricks, Elastic MapReduce, and Google Cloud's Dataproc.

Has Spark lived up to its promise? Is it delivering value to organizations in the age of cloud computing? Does it serve data engineers in building data lake infrastructure, as well as data scientists using it for ad-hoc data exploration?

In this ebook we will highlight the ways in which Spark and Spark-based managed services fit into the modern data engineering landscape, and see how they compare to Upsolver, a cloud-native data processing solution built for data lakes.

The Basics

	upsolver	Apache Spark
Data transformations	**UI / SQL**	**Scala / Python**
Raw data schema detection	**Upsolver**	**User**
Job orchestration	**Upsolver**	**User**
State management	**Upsolver**	**User**
Exactly-once processing	**Upsolver**	**User**
File-system optimization	**Upsolver**	**User**
Healing scaling upgrades	**Upsolver**	**User / Platform**
Time-to-Production	**Days**	**Months**
Headcount	**1 Engineer / DBA / Analyst**	**3-4 Big Data Engineers**

The Data Lake Dilemma

Data lakes provide unlimited and affordable raw data storage, but have proven difficult to operationalize due to costly and complex data processing required to make that data consumable. To understand the role data processing tools play in modern data lake implementations, we first need to understand what a data lake is and why organizations choose to implement data lakes in the first place.

What is a data lake?

The data lake is an approach to big data architecture that premised on storing all the dat an organization generates in its original form, in a single repository that serves multiple analytic use cases or services.

Data lakes center around decoupled and virtually limitless object storage in which to preserve all raw data in a “store now, analyze later” paradigm. In the cloud, the storage layer leverages inexpensive object storage such as Amazon S3, Azure Blob Storage, or Google Cloud Storage; and the data lake typically also consists of a variable range of tools and technologies that move, structure, and catalog the data to make it queryable – and therefore valuable.

Why do organizations use data lakes?

1. Storage costs: Decoupling cloud storage from compute allows organizations to store more data without worrying about ballooning costs.

2. **Reliability:** Cloud object storage is the most reliable means of storage, with tools such as Amazon S3 promising 99.999999999% object durability.
3. **Fault tolerance:** Since all data is stored in raw form, it can easily be replayed to recreate a historical state and for error recovery.
4. **Openness:** Using cloud object storage prevents lock-in to a specific database or data warehouse vendor, allowing companies can employ a best-of-breed tooling strategy.

The attractiveness of the data lake is that when data is generated, it is often unclear what parts of the data will be used for what purpose. A cheap raw storage layer keeps your options open.

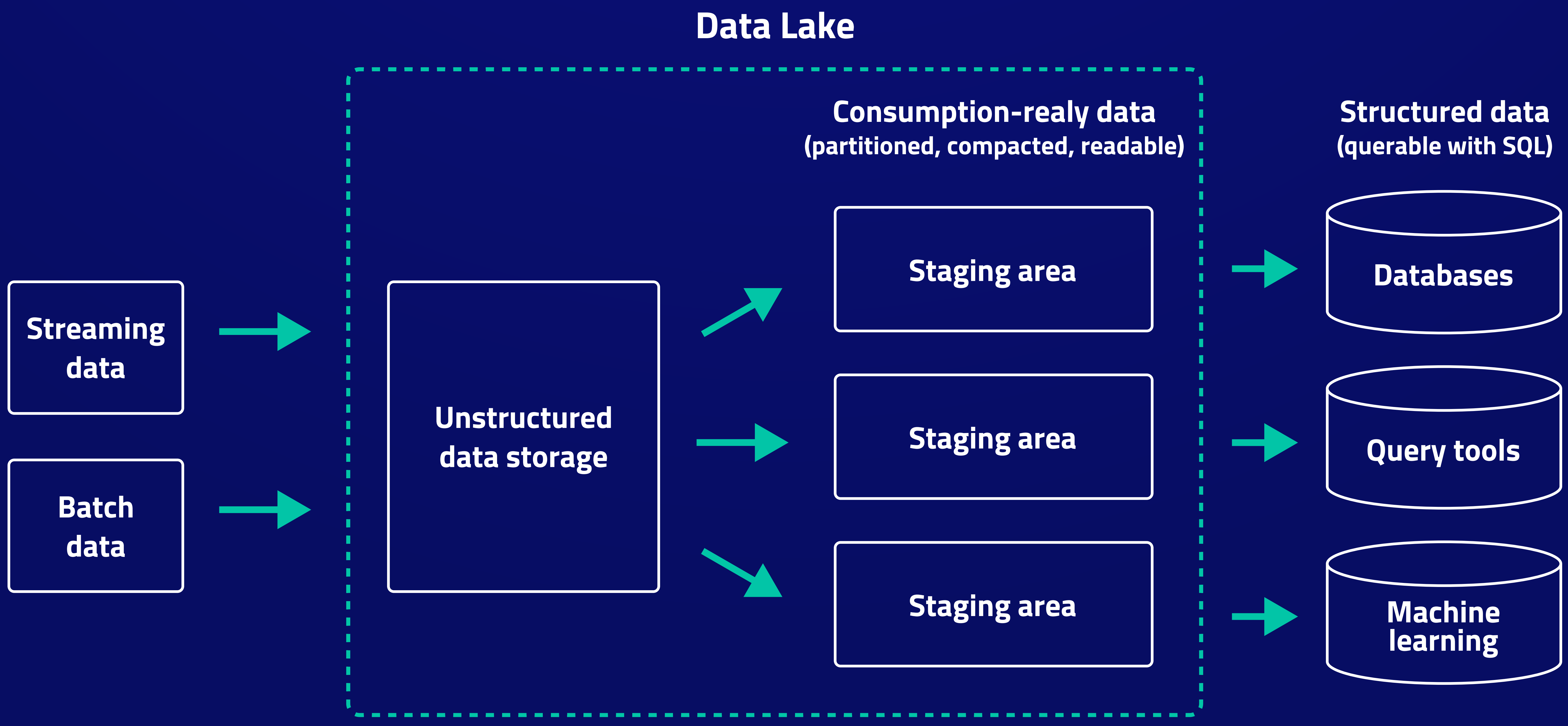
Data processing: a common roadblock

While the advantages of data lakes make them indispensable in a modern big data architecture, many organizations struggle to realize value from data lake projects, and instead find themselves mired in lengthy and complex implementations. The reason for this is that while data lakes make data very easy to **store**, when it comes to operationalizing the data for analysis, things get tricky.

Data needs to be processed before it can be put to use

Unlike a data warehouse, where data is preprocessed and structured before it is written into database tables, data lake storage consists of objects - often millions of them - stored in folder-like structure in a cloud repository such as Amazon S3. Data will often originate from multiple sources and land in structured, semi-structured and unstructured formats, and even within the same source, schema and fields change over time. Data being 'poured' into the lake in this fashion often leads to the infamous problem of data swamps - with the lake becoming a practically unusable mess of unstructured, uncataloged and unidentifiable data.

Eventually, we do not wish to store data merely for the sake of storing it; data is meant to serve the business through some kind of production use case - analytics, machine learning or powering data-driven applications. Supporting these varied use cases means transforming the raw data into structured and optimized datasets that can be made available to different consumers. This is where data processing tools come in.



To make raw data consumable you must process it first so that it is structured and optimized for use by data consumers.

Why we need to process the data

There are multiple data processing challenges that need to be addressed in order to make data lake storage analytics-ready:

- Optimizing the object store for querying: Query engines struggle to handle large volumes of small files.

  This problem becomes acute when dealing with streaming sources such as application logs or IoT devices, which can generate thousands of event logs per second, each stored in a separate JSON, XML or CSV file. These files need to be converted into a file system which is optimized for fast analytical queries, built on columnar formats such as Apache Parquet or ORC.

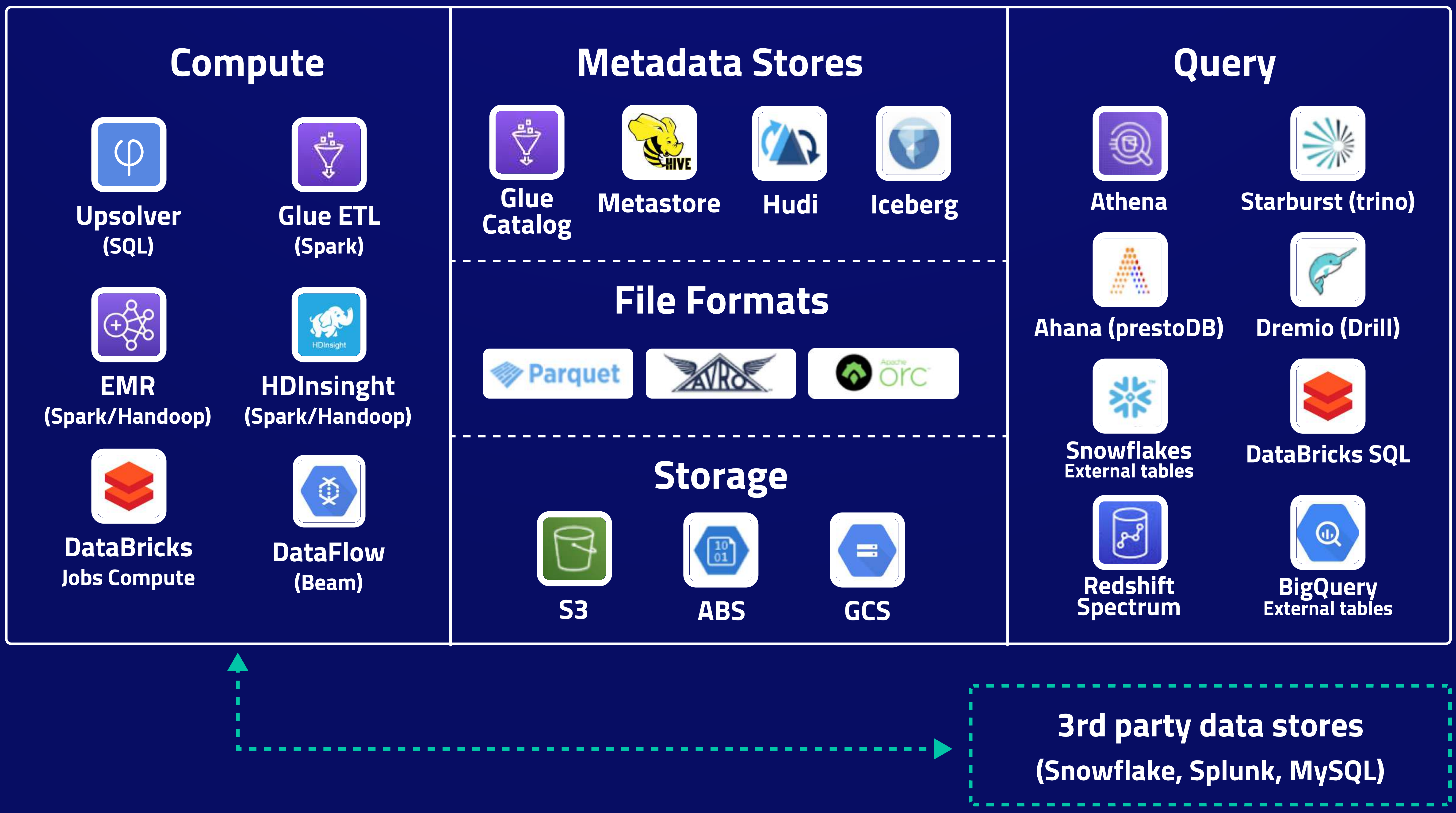
- Perform stateful transformations: Since we want the data to be queryable from the lake by different consumers, we would like to be able to unify data from multiple sources. In order to do so, compute-intensive operations such as joins and aggregations need to be performed in-memory by a separate compute engine.

- Manage the metadata layer: Metadata is stored in a separate catalog service such as a Apache Hive Metastore or AWS Glue Catalog (a managed version of a Hive metastore), which needs to be properly partitioned and kept in-sync with the physical file locations on disk.

- Perform Upserts: Performing operations like upserts (inserts and updates) and deletes on a data lake file system, which isn’t indexed and is built for append-only use cases, will often require separate data engineering effort using tools such as Apache Hudi or Iceberg.

Data Processing Alternatives

Cloud data lake analytics landscape



The open source approach - Spark and Managed Spark Platforms

Spark, first introduced in 2009 and released under the open-source Apache license 2013, offered a modern alternative to Hadoop MapReduce. Spark offers a **flexible real-time compute engine** that supports complex transformations, and its relative popularity ensures there is a **large open source community** that continues to support it. Due to its performance and support for popular programming languages like Java, Scala and Python, Spark has been a popular environment for creating data lake processing jobs.

Why many companies struggle with Spark

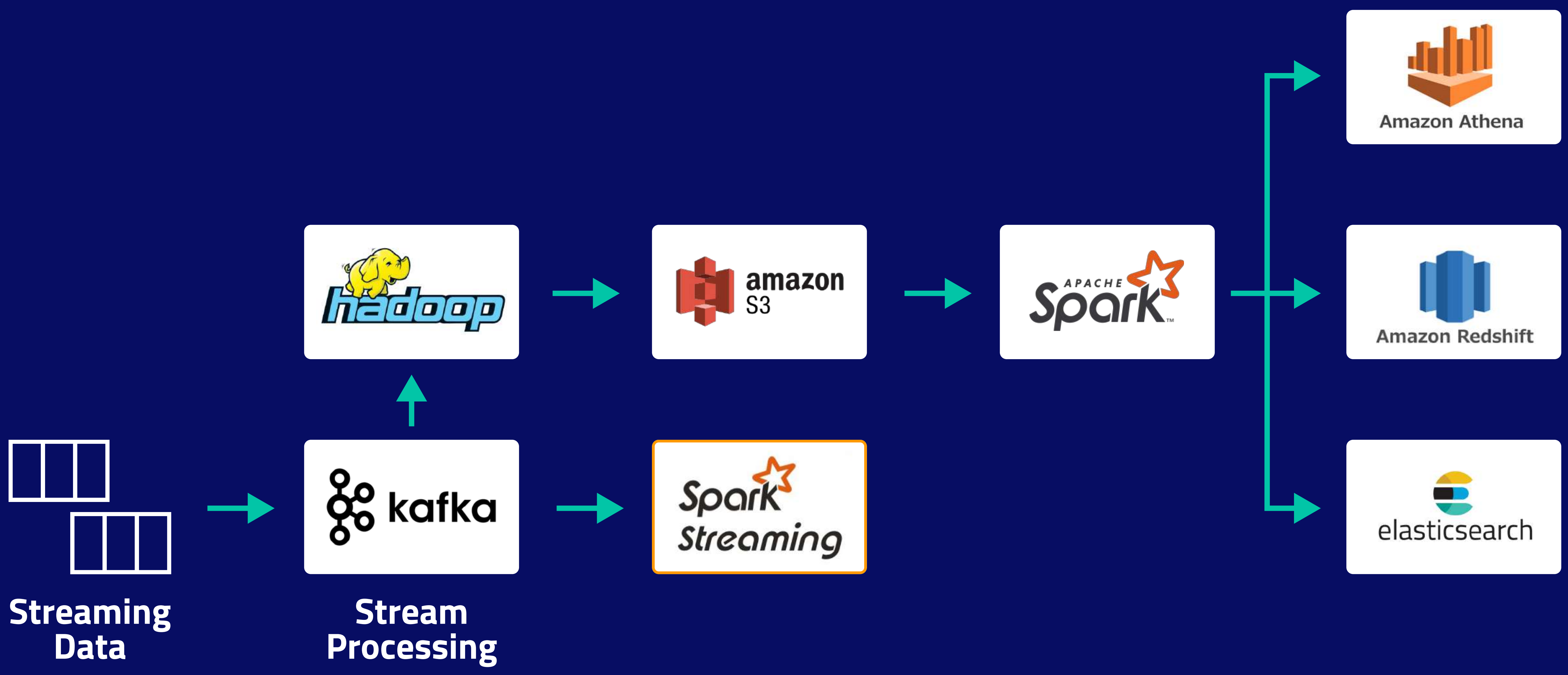
Many organizations struggle to see get to production with Spark as it has a very high technical entry barrier and requires extensive dedicated engineering resources.

Spark inherited the complexity of Hadoop and requires specialized engineers that understand distributed systems, coding in Scala/Java, workflow orchestration and analytics best practices. This combination of skills isn’t immediately available to most engineering teams - and when it is, it’s often better utilized on higher-value work such as application development or machine learning.

There is no real self-service version of Spark - Spark jobs are written in low-level code (Scala/Java) and there are a lot of “levers” to pull in order to tune the data lake for performance. Programming transformations, state management and file system management is time intensive, prone to errors and beyond the technical scope of 95% of traditional database practitioners.

Cloud managed Spark

While Spark's roots are rooted firmly in open-source and on-premises development, the move towards cloud computing sprung several proprietary products that built on Spark's core features while adding security, reliability and managed cloud infrastructure. Notable products in this space include Databricks, Amazon EMR, and Google Dataproc. These tools add managed scale using the elasticity of the cloud, but do not address the limitations noted above.



ETL tools won't save the day

Data warehouses have a rich ecosystem of ETL and ELT tools that help ingest application data and orchestrate workflows. Some popular choices include DBT open source, FiveTran, and Matillion. These tools can be used in 2 ways: 1) read data from sources, perform minimal transformations and load into tables (Extract-Transform-Load) or 2) run transformations jobs on the database compute engine after landing the data (Extract-Load-Transform).

However, this type of ETL flow relies on having a database to do the heavy lifting. If we want to keep our data in usable formats on the lake, avoid vendor lock-in and support the open architecture that the data lake is intended for, we need to transform and integrate data without relying on database compute resources.

The Upsolver approach

Upsolver was founded by database engineers who wanted to build a modern cloud data lake without the complexity of Spark and Hadoop. Upsolver takes the traditional database approach (configuration, not code) and applies it to data lake processing by applying the following principles:

- Give end users access to 100% of their data: Upsolver's data processing is governed by a visual, SQL-based interface which can be used by all data practitioners including DBAs, data architects, analysts, data scientists, product managers and big data engineers.

- Hide database complexity to allow for focus on the data: users want to focus on writing business logic (transformations); the subsequent engineering work is abstracted away and automated. Data consistency (no loss, no duplications), ETL jobs orchestration, ETL state management and file system management are all handled automatically according to best practices.

- Have it your way: no-code, low-code or high-code: Upsolver's data transformation language is based on SQL and is extensible with Python. Any use case that can be expressed with SQL, can be executed on Upsolver so the user doesn't need to compromise between flexibility and ease-of-use.

- Elastic scaling: Upsolver never stores data on local server storage so processing can elastically scale according to the workload. The infinite throughput of cloud object storage is key to ensuring this.
- No infrastructure to manage: Upsolver runs on fully managed clusters (dedicated per customer) that auto-scale on-demand. There is no need to backup Upsolver instances since all data is stored on cloud storage. If an instance crashes, a new one will spin up and continue the processing work.

- Streaming first: Upsolver processes all data as a stream, ensuring extremely low latency even when working with data that hits all the 'V's of big data - volume, velocity, variety, and veracity.
- Low cost: Upsolver utilizes cloud object storage and low cost compute instances for processing, creating the lowest possible footprint of cloud provider costs.

Comparing Upsolver vs Spark - Feature by Feature

Cost

Cost	Upsolver	Apache Spark	Managed Spark tools
Storage	Cloud object storage		
Compute Type	On-demand / Reserved / Spots		
Compute RAM Footprint	Low Streaming Compressed indexes	High Batches Uncompressed RDDs, DataFrames	
Manual effort	Near zero	Data engineering team	DevOps + Data engineering team

Development Effort

	Upsolver	Apache Spark	Managed Spark tools
Manage infrastructure	Platform	Platform	User
Ingestion	Plug-n-play connectors	Code	
Schema discovery	Auto-generated schema-on-read Field level statistics	Code	
Transformations	SQL / UI	Code	
High cardinality joins	Included	3rd party key-value store	

Workflow Orchestration	Automatic	Manual
Compaction	Automatic	Manual
Integration to metastores	Automatic	Manual
Performance tuning	Automatic	Manual

Maintenance

	Upsolver	Apache Spark	Managed Spark tools
Instance management	Automatic	Manual	Automatic
Version updates	Automatic	Manual	Automatic
Auto-scaling	Automatic	Manual	Automatic
Schema evolution	3 clicks	Code	
Reprocessing / Replay	3 clicks	Code	

User profile

	Upsolver	Apache Spark	Managed Spark tools
Big data engineers	Yes		
Data Scientists	Yes	No	Sometimes
Data Analysts	Yes	No	No
RDBMS DBAs	Yes	No	No
Product Managers	Yes	No	No
Developers	Yes	No	No

Summary

	Upsolver	Apache Spark	Managed Spark tools
Dev-time	Minutes to hours (similar to databases)	Months to years (extensive development project)	Weeks to months (writing code)
Maintenance overhead	Near zero	Very high	High
Stateful transformations	Yes. Using SQL (Extensible with Python)	Yes. Using code	Yes. Using code
Scaling	Elastic	Manual	Elastic
Total cost of ownership	Low	High	High

Start for free with Upsolver on the AWS Marketplace.

Upsolver is an Amazon Web Services Advanced Technology Partner and a trusted AWS Athena partner. Data-driven companies such as ironSource, Wix, and The Meet Group use Upsolver to power data lakes and deliver batch and stream processing at unparalleled ease.

Ready to get started?

Get access to all functionality for free by deploying Upsolver on the AWS marketplace.

GET STARTED

aws marketplace