upsolver

Technical Whitepaper

A Roadmap to Self-service Data Lakes in the Cloud

Unlocking the value of streaming data by simplifying big data discovery, processing and management in cloud data lakes.

Written by Yoni Iny, Co-founder & CTO, Upsolver

Table of Contents

Table of Contents

Executive Summary

While the transformative nature of big data is commonly celebrated, its organizational applications are often less glamorous: Gartner analyst Nick Heudecker has recently estimated that 85% of big data projects fail, largely due to misaligned expectations, management resistance, lack of skills, and data governance challenges. From a technology perspective, failure takes the form of an inability to deliver tangible business benefits from big data initiatives within actionable time frames, despite significant resources and engineering hours being sunk into them.

Technical departments struggle to maintain the complex software and hardware stacks needed to effectively work with big data, and encounter difficulties in hiring and retaining skilled personnel across the analytical life-cycle - from data scientists who translate swathes of unstructured data into relevant insights, to data engineers who build the infrastructure that enables data exploration and experimentation.

In this paper we will present the infrastructural challenges of working with big data streams; we will then proceed to introduce our solution: Upsolver, a Streaming Data Platform that provides data management, processing and delivery as services, within a data lake architecture that utilizes the scalability of object storage such as Amazon S3. We will show how Upsolver simplifies the process of transforming streaming data into workable data, dramatically shortening time-to-value and increasing success rates for streaming data projects.

Drivers of Streaming Data Complexity

Streaming data is not merely big data - it is also fundamentally different data, produced by different sources and requiring its own set of tools to handle. Accordingly, the solution is rarely as simple as getting a bigger database.

Traditional organizational data originates from various operational systems such as ERP, CRM, finance and HR systems-of-record; streaming data is typically produced by sources such as industrial sensors, clickstreams, servers and user app activity. This creates several core differences between streaming and traditional data:

Non-static data

A traditional dataset typically captures a clearly delineated time and space, e.g.: number of items currently in stock, new employees hired in the past year; whereas big data is generated constantly, delivered in a continuous stream of small files, capturing event-based interactions on a second-by-second basis, e.g.: servers relaying their current state of operation, or a log of user activity on a mobile app. Querying this data means creating an arbitrary start and end point and creates challenges around data freshness, missing records and synchronization.

Unstructured versus tabular

Due to the high velocity in which event-based data is produced, and its highly disparate nature, it will be stored as objects (e.g. JSON files) rather than tables. The relational databases that power almost every enterprise and every application are built on tabular storage; using them to store unstructured data requires lengthy cleansing and transformation processes.

Experimental versus standardized use cases

Common data sources lend themselves to tried-and-tested forms of analysis and reporting; the potential of streaming data is unlocked through exploratory techniques, predictive modeling and machine learning. Applying these analyses requires broader and more flexible access to data, and can be hindered by the need to structure data according to existing enterprise data warehouse specifications.

Storage is a major cost factor

Databases can usually get away with using expensive storage with multiple redundancies, since the size of the data is relatively small and the compute and licensing costs outweigh the cost of storage. When dealing with big data, storage costs can easily dwarf all other project costs.

Rise of the Data Lake

The unique characteristics of streaming data lead organizations to accept the limitations of relational databases for storing and analyzing streams; instead, recent years have seen a growing prominence of data lake architectures.

In this framework, the organization foregoes the attempt to 'tame' big data as it is being ingested, instead adopting the principle of store now, analyze later: records are stored in their raw, unstructured and schema-less form in a distributed file system such as HDFS, or in cloud object storage such as Amazon S3. The complicated process of transforming it into workable form is postponed until the time the need arises to do so, in order to satisfy a new business application or answer an analytical query.

Data lakes present several advantage for streaming data architectures:

- **Easy to ingest data:** the data lake approach removes the need for the complex and expensive ETL processes that are prerequisite to storing streaming data in a relational database; instead, a relevant subset of the data can be transformed and wrangled as needed to address the particulars of the use case that is being developed. This reduces much of the upfront costs of working with big data.

- **Ability to support a large range of use cases:** Data lakes provide vastly greater flexibility for developing new data-driven applications, especially for innovative and exploratory analyses that rely on access to data in its original form. Since invariably transformations into a rigid schema cause loss of connections within the data, exploratory analysis often finds itself facing the brick wall of not having retained the necessary information for that specific question; the data lake approach sidesteps this issue by storing all the the raw data in its original form, making it easier to answer questions based on historical data.
- **Reduced storage costs**: In data lakes, storage is decoupled from compute and can rely on inexpensive object storage in the cloud or on-premises. Since compute resources are dramatically more costly than storage, data lakes can significantly reduce infrastructure costs when working with large amounts of data compared to storing the same amount of data in a database. This comes in addition to the savings associated with removing the need for data transformation and wrangling upon ingest.

Challenges and entry barriers:

Nevertheless, data lakes are not a silver bullet; while they serve to reduce complexity in storing data, they also introduce new challenges around managing, accessing and analyzing data, often becoming massive engineering

undertakings without clear results for the business. Typical stumbling blocks include:

- **Governance:** data lakes are at constant risks of becoming data swamps as new data is 'dumped' into them without a clear process of cataloging, managing and visualizing existing data assets. This creates difficulty in extracting relevant datasets and can pose governance and security challenges around access permissions.
- **Performance and latency:** unstructured data must be structured before it can be queried, resulting in lengthy and code-intensive ETL jobs. Batch processing can produce partial, inaccurate results (limited to the data in the batch), while stream processing introduces its own set of technical challenges. Additional latencies and performance issues manifest due to query engines needing to process millions of small files in order to return query results; optimizing the storage layer through compression, compaction and columnar file formats will help, but it is in itself a complex process that requires specialized knowledge.
- **Complexity and time to value:** addressing the previous challenges poses a major engineering challenge. DevOps and data engineering teams need to invest massive amounts of resources and time in hardware and software. Managed cloud services can address some of the physical infrastructure (at a cost), but the latter still requires a broad set of open-source and licensed

software tools which vary based on the particular data lake implementation, as well as significant customization and manual coding in Python and Scala to manage, process and structure streaming data. Skilled big data engineers are needed to maintain this infrastructure - and demand for these professionals greatly exceeds supply, making them to difficult to hire and retain.

Reference architecture for cloud data lakes (without Upsolver)

As we can see, this type of architecture relies on a lot of moving parts and a lot of data engineering. For the data lake to provide value, every piece of the puzzle is properly optimized and working as expected – which requires hundreds of big data engineering hours.

The end result: Projects that drag for months and years with no definite endpoint, exceeding budgets and failing to deliver the promised business results - leading decision makers to conclude that the streaming data initiative has failed, abandoning it altogether and potentially stifling innovation.

Upsolver: Reducing Friction and Complexity With a Streaming Data Platformr and potentially stifling innovation.

Design Principles:

In the previous chapter we highlighted the problem with streaming data; traditional, data warehouse-based approaches are inadequately suited for storing streams, while data lakes create significant complexity and technical overhead when data needs to be effectively analyzed.

Upsolver modernizes data lakes by applying the design principles of SaaS and full-stack analytics platforms to streaming data integration, processing and analysis. This means giving data consumers an end-to-end platform for transforming streaming data into usable data, with minimal IT or data engineering overhead.

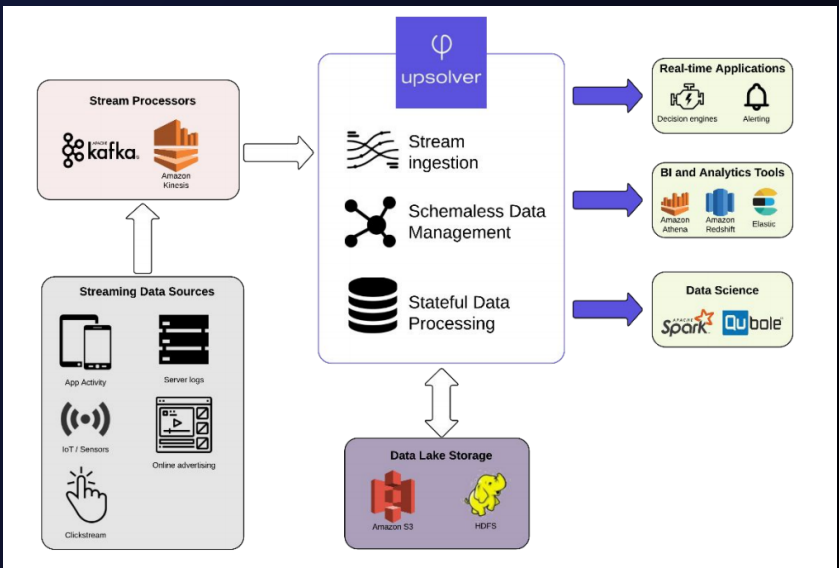
A visual self-service solution for governing data and building ETL flows: typical data lake architectures rely on code-intensive frameworks for schema

discovery, metadata management and data modeling; Upsolver provides a WYSIWYG user interface for visual data preparation and management.

A stateful ETL engine for streaming use cases: Upsolver uses homegrown in-memory processing and compression algorithms to deliver performance that far exceeds open source stream processing frameworks such as Apache Spark, out of the box and without the need for months of manual performance tuning.

Automating data engineering best practices: the need to transform thousands of unstructured small files into structured tables typically requires months of coding to build ETL flows and data pipelines according to a convoluted list of necessary best practices; Upsolver handles this process behind the scenes in a manner that is almost entirely invisible to the end user, and automatically transforms raw data streams into a high-performance data lake.

Reference architecture:



In the next sections we will dive deeper into the Upsolver data lake architecture, and explain how it addresses current challenges in the data lake world and enables data consumers to rapidly unlock the value of their big data streams.

The Platform: Core Components

1. Data Management and Governance

Data lakes are inherently messy as they ingest data as close as possible to its raw form. However, optimizing raw data storage for analytical performance can have tremendous impact on the ability to utilize the data further down the road, while schema discovery and metadata management are crucial for governing and controlling your data lake and preventing it from 'swampifying'.

Storage

In order to be able to work with large quantities of data, efficient data loading is essential. Data loaded into the data lake should be standardized and compressed to make it easier for data consumers to work with it without the end users needing to intimately understand the underlying storage layer.

Upsolver automatically optimizes the storage layer, using a hierarchical folder structure with a lexicographic date part to facilitate data discovery. Files created are either Avro or Parquet and rely on either Snappy or Gzip compression to provide optimal performance when accessed by common query engines such as Athena, Presto or Hive. Let's look at some of the details behind each process.

File Formats

Avro and Parquet are complementary formats, where Avro is a self describing hierarchical serialization format, and Parquet is a columnar file format that uses Avro as the in memory representation. Data can easily be converted between the two formats, and Avro is a very flexible and fast format. Upsolver ingests data in Avro while creating columnar Parquet files to facilitate analytical querying.

Some limitations exist in the Avro and Parquet specifications, for example field names must be alphanumeric. Since real data can have any format and losing information on the original data is undesirable, Upsolver uses an encoding within the Avro/Parquet format in order to extend it to all field names.

Compression

Upsolver gives users the option to choose between Snappy and Gzip for all data storage. Snappy is a fast and standard compression algorithm that gives reasonable compression rates. Gzip results in smaller files and is also standard, but has a much higher CPU overhead, which can be undesirable when processing many terabytes of data. Compression is one of the most important tools in reducing the size and cost of a data lake; data should never be stored uncompressed!

Schema Discovery and Metadata Management

Metadata management is required since the data lake is opaque without it. Unlike databases with built in querying and management tools, a data lake without metadata requires expensive and time consuming queries with external tools in order to figure out what it contains. Without a central repository of such information, users will be stuck doing these actions over and over each time they want to inspect their data.

An often-repeating pattern is a user creating an ETL flow based on where data should be and how they expect the data to look, only to discover after a costly ETL process that it isn't what they expected. We call this "blind ETL" and it is incredibly pointless and wasteful. A user doing a transformation of data should have full visibility into the actual data at their fingertips, so that they can make informed decisions and correct data issues before running an expensive batch job.

In Upsolver, rich metadata on all fields in the data is automatically extracted when data is loaded, and stored in an Upsolver Materialized View (more on that later). This allows consumers of the data lake to operate within a framework that gives visibility into the data at every step, preventing costly mistakes in ETL definitions and helping with data governance.

2. Stateful Stream Processing, ETL and Data Transformation

Why managing state is a challenge in data lake ETLs

Data lakes are highly valuable to enable agile data science, as storing the data in its original form gives us the ability to reproduce an accurate picture of a historical state of affairs [link to event sourcing article somewhere]. However, unstructured object storage is not built for analytic querying: Unstructured streams are impossible to analyze so ETLs (many many ETLs) are required to structure the data so it could be queried.

Application state, is what makes the ETL challenge much much harder with data lakes. Data applications usually require an aggregate view of the data over time (state). For example: predicting the next best offer for a user requires a dataset with historic user behaviour, tracking funnel KPIs like CTR requires a join between a stream of views and a stream of clicks.

Unlike data lakes, relational databases are built for joining data since they have indices. With data lakes, there are 2 options: Limit the use case to recent data (streaming analytics systems like Spark Streaming store raw data in RAM for joins) or spin-up an external NoSQL database (Redis and Cassandra are popular options) to manage state over a long period of time. The first option is limiting functionality and the second option creates a huge overhead.

In order to o accomplish the vision of data lakes - agile analytics and data science over un/semi structured data, a powerful ETL engine is required.

Stateful ETLs With Upsolver

Upsolver uses a built-in indexing system called Materialized Views (see below) to manage state. Materialized Views store the results of a Group-By query, running continuously on data streams, as a time-series. The relevant results can be fetched using a key and a snapshot time. For example: A data scientist could retrieve results from any snapshot in time and avoid the impact of Data Leakage in Models.

Materialized Views are persisted to an object storage, like AWS S3 and HDFS, and cached in RAM for fast retrieval (under 1 millisecond). Materialized Views work at a low RAM footprint (benchmark is 10% comparing to Cassandra) so it's possible to load a lot of data into RAM to drive stateful use cases like machine learning.

By utilizing object storage and avoiding local disk storage, Materialized Views dramatically reduce the cost and effort relative to using an external database, especially when it comes to scaling, replication and recovery.

Lastly, Materialized Views are built retroactively (Replay / Re-stream) using data already stored in an object storage, saving precious time spent waiting for data to accumulate.

Tools for workflow management, data transformation and movement

Data in a data lake is only as useful as your ability to consume it, which inevitably requires ETL. For example, join is a very slow operation in a data lake (no native indexing) so joins are usually performed in the ETL stage and not in the query stage.

The current go-to tool for big data transformation is Spark. Spark is an excellent tool that gives incredible flexibility in transformations and SQL support over the data lake, and has a vibrant ecosystem that is continually evolving. However, it is not without its drawbacks.

Firstly, Spark works best for ad-hoc work and large batch processes, whereas scheduling Spark jobs or managing them over a streaming data source requires external tools, and can be quite challenging. Second, while getting started with Spark for ad-hoc querying is very easy and does not require any special knowledge of big data, the expertise required to use Spark correctly in production systems is rare and expensive to obtain.

Upsolver offers an alternative approach by providing a robust WYSIWYG data transformation tool built in that leverages Materialized Views to allow users without big data expertise to define complex transformations over streaming data. Scheduling and cluster/task management are handled implicitly, so users only need to define the actual transformations without needing to have an in-depth understanding of the underlying operations or infrastructure.

Support for nested data structures

Discussions around data often assume that data is flat, as can be represented by a CSV file, or in a database table. There are many rows in the table, and each has a single value for several typed columns. Unfortunately, real data is usually much more complicated than that. A better data representation is what can be represented in a JSON file; multiple records, each containing multiple named fields, which in turn can be either records themselves, or arrays of values or records.

This nested hierarchy helps represent meaningful connections between different types of data, but does not lend itself well to simple query languages. Traditional solutions would either flatten data in advance, losing connection information and creating duplicate records; or else require code to be written in order to perform the transformations within the nested structure, often introducing bugs and performance issues.

Upsolver aims to give users maximum flexibility with real data structures. For that reason, data enrichments in Upsolver always write to a location within the original data hierarchy. How the data is processed depends on the relationship between the input fields and the target field of the enrichment, as well as the function used.

3. Real-time and Ad-hoc Data Consumption

Finally, for a data lake to provide actual return on investment, it needs to address an actual need within the business such as improving operational processes or enabling better data-driven decision making. Upsolver operationalizes organizational data lakes by delivering clean, structured data to various outputs and enabling high-throughput, low-latency access to data for real-time use cases.

A data project will usually have one of two distinct methods for consuming the data in its final state. The first is SQL, which is useful for analytics workloads and ad-hoc querying for research and reporting. The second is using an operational database for real-time serving, in order to power user interfaces and internal systems. In order for the data lake to be successful as the central repository of all data, it needs to be able to fill both of these roles.

Ad-hoc Analytics using SQL

SQL is the de-facto data querying language. Almost all consumers of data, from data analysts to data scientists, are happy to use SQL where possible, since it is expressive and easy to use.

Common use cases for SQL-based querying would be:

- **Business intelligence and analytics**: for example, a data analyst in a large web company wants to create a daily report that contains the conversion rate of all the different web pages of the organization.
- **Data science:** for example, a data scientist in an online retail company wants to find products to offer to users based on products that they already viewed/purchased. They will want to find all the products that the user viewed, and find similar users' purchases.

Integration with SQL Engines (Athena, Presto, Impala)

There are quite a few SQL over big data systems, including Apache Presto, Apache Impala, Apache Drill and Amazon Athena (to name a few). What all these systems have in common is that they do not manage the data itself, but only provide a SQL layer over existing data. In order for them to work well, the data needs to be stored in a way that keeps the amount of data scanned per query to a minimum.

These "best practices" of data storage are quite challenging to implement for the uninitiated. This will often cause teams to reach the mistaken conclusion that the query engine itself isn't performant enough, instead of realizing that the data storage format itself is the problem. Upsolver has a deep integration with SQL engines, which allows a user to treat their data lake as they would a database. In the same way that databases have opaque storage structures

that “just work”, Upsolver abstracts the storage implementation details away from the end user. This means that you will get the best performance possible for your data from the get go.

Many data use cases only require aggregated data, or a small subset of the data. Since data lakes store raw data, often data lake users are turned off to the idea of scanning all their data, when the actual interesting data set is a tiny fraction of that.

Upsolver helps with this use case by creating pre-aggregated and filtered outputs of the data lake. This allows users to have multiple data tables in their SQL engine, each looking at a different aspect of their data. Querying is made much easier, faster and inexpensive when done only on the relevant subset of the data.

Enabling Real-time Applications

The second data lake use case is where queries of the data lake are run in order to make real time decisions, or display many data points to a user in an interactive interface. These queries require very low latencies, and will often use an operational database like Cassandra in order to fulfil the performance requirements. Integrating such a database with data from a data lake can be quite challenging, which is why Upsolver encapsulates a native operational database within the data lake itself. Users can use Upsolver Materialized Views to create pre-defined aggregations over their data lake, which can be served in real time for operational use cases.

A few examples of operational workloads:

- **Real-time reporting:** A website wants to have interactive elements, such as showing how many other active users are currently viewing the same page from the same country.
- **Anomaly detection:** An industrial monitoring system measures many metrics such as noise levels and air content at points around a machine. This data is compared to historical values in real time in order to alert on anomalous readings, which allows technicians to shut down the machine before any permanent damage can occur.

Materialized Views

Materialized Views in Upsolver are key-value stores that are defined as aggregations over a stream of data. For example, “The number of distinct user ids per country in the last month” or “The first time and the last time we’ve seen each user id each calendar month in the last year” are both definitions of materialized views. In the first case, querying by country will return the number of distinct users, and in the second case querying by user id will return the first and last times we saw that user in each calendar month.

Materialized Views live on top of a stream, and all the data files are stored in cloud storage. The data contained in them changes every minute by adding the latest minute and removing the minute from the beginning of the time window.

They are built within an Upsolver Compute Cluster, and served from Upsolver Query Clusters, which load the data files from cloud storage directly. Data files are created every minute, and merged files are created in increasing time intervals to support access to large time windows with logarithmic complexity.

Materialized Views are stored as sorted immutable hash tables, with the data stored using a hybrid row/columnar format. Many databases use columnar formats in order to greatly improve compression and query times. Columnar formats also support run length encoding, which is a very efficient way to store sparse values.

Since Materialized Views are used for random access “get by key” requests, storing data in a pure columnar format introduces quite a lot of overhead - the value from each column would need to be found individually, and lots of random memory access slows things down considerably.

Instead, Upsolver uses columnar compression in a row based format. In order to achieve the same compression as a columnar format would using run length encoding, Arithmetic Coding is used as the entropy encoder. This allows for optimal compression in exchange for performance, but our implementation of the arithmetic decoder performs 16 million symbol decodes per second per CPU (on AWS m5 instances), which prevents it from being a performance bottleneck.

Use Case Examples and Reference Architectures

Sisense Drives New Analytical Insights from S3

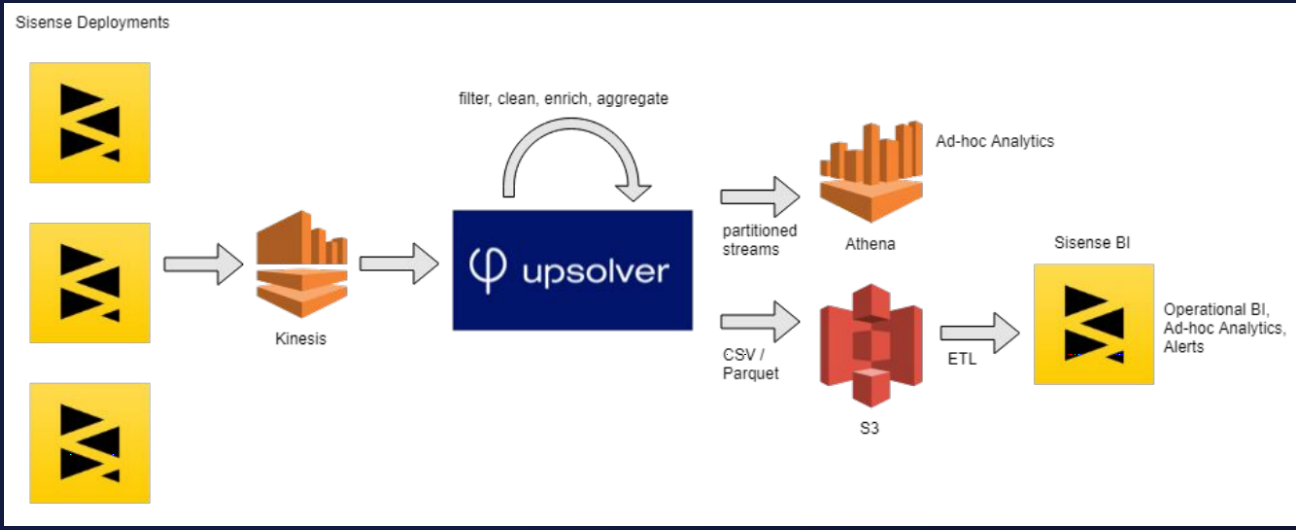
Sisense is one of the leading software providers in the highly competitive business intelligence and analytics space, and has been named a Visionary by Gartner for the past two years. Headquartered in New York and serving a global market, the company provides an end-to-end BI platform that enables users to reveal business insights from complex data. Sisense counts Nasdaq, Airbus, Philips and the Salvation Army among its customers.

Seeking to expand the scope of its internal analytics, Sisense set out to build a data lake in the AWS cloud in order to more effectively store and analyze product usage data - and following a recommendation from the AWS team, began looking into Upsolver's streaming data platform.

The Requirements

- Transform data streams into structured fact tables that could then be sent to Sisense's BI software
- Ability to quickly iterate and answer new business questions as they arise
- Self-service solution that would not require a dedicated data engineering team

The Solution



Business Benefits

- Ability to run new analytical queries on streaming data in Sisense
- Much-improved visibility into internal data
- Additional use cases around machine learning being rolled out

Engineering Benefits

- Agile infrastructure that enables new queries and tables to be generated on the fly
- Data lake project could be handled internally and without the need to devote a team of engineers.

To read the full case study, visit: https://www.upsolver.com/case-studies/sisense-s3-data-lake

Bigabid Builds a Real-time Architecture to Supercharge Mobile User Acquisition

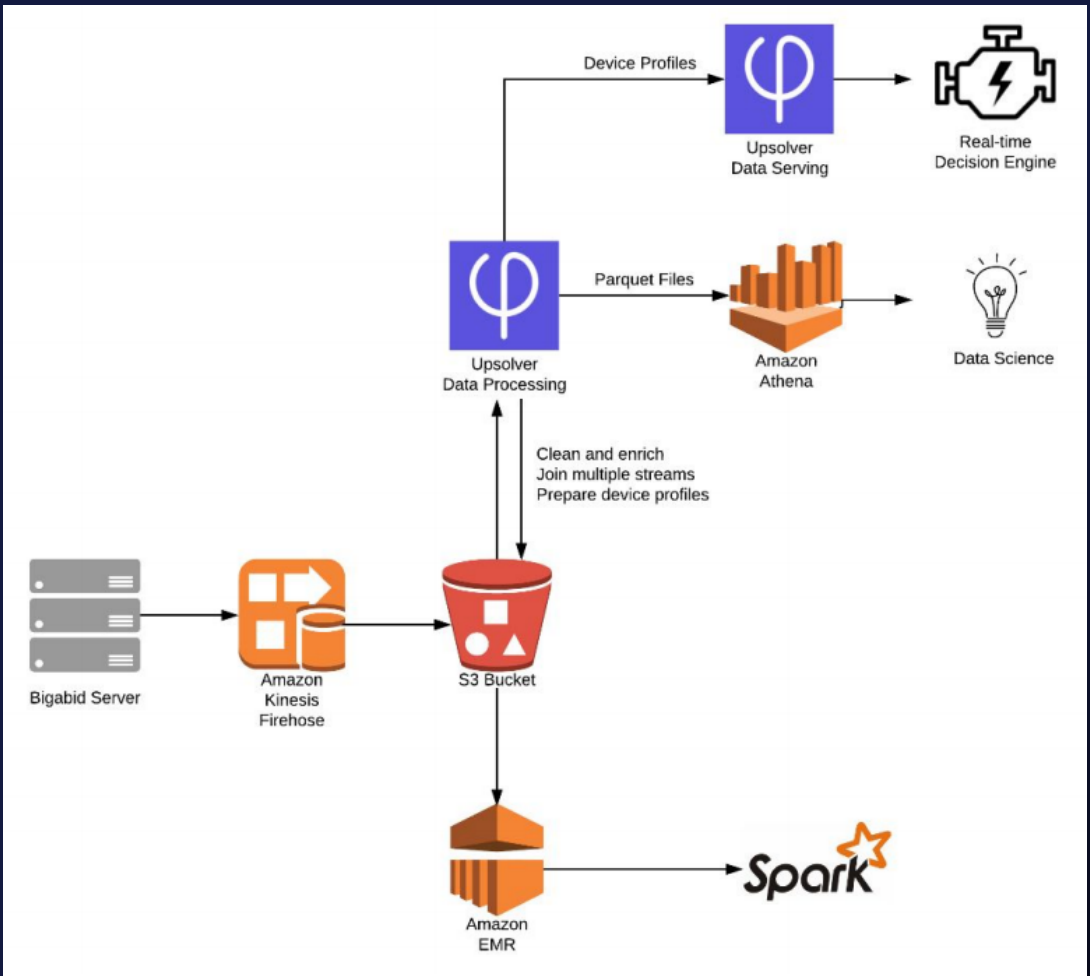
Bigabid is an innovative mobile marketing and real-time bidding company that empowers its partners to scale their mobile user acquisition efforts while lowering advertising costs through CPA and AI-based optimization.

In order to maintain a high level of performance in the competitive app advertising market, Bigabid needed to introduce real-time user profiling to its algorithmic decision engine. This case study will demonstrate how the company built a high-performance real-time architecture with minimal data engineering using Upsolver, S3 and Amazon Athena.

The Requirements

- Create a real-time aggregate view of users which would include data from the last 180 days, and use it to deliver more accurate bidding decisions.
- Clean, enrich and join data from several streams: bids, impressions, clicks, and several third-party data providers.
- Support for advanced features such as Count of Sessions per User, which would be difficult to achieve in a traditional database architecture.
- Improve data freshness from 8 hours to less than one minute.

The Solution:



The Results

Working with fresher, more accurate data has played a major role in Bigabid's ability to continuously provide exceptional performance for its customers. The real-time architecture, powered by Upsolver, is now an essential component in Bigabid's platform.

To read the full case study, visit: https://www.upsolver.com/case-studies/bigabid-real-time-architecture

Next Steps

- Learn more about Upsolver or schedule a live demo: https://www.upsolver.com/
- Read more Upsolver case studies: https://www.upsolver.com/customers
- Start a fully functional 14 day free trial: https://app.upsolver.com/signup