
PLURALSIGHT

ETL Pipeline

Week 5

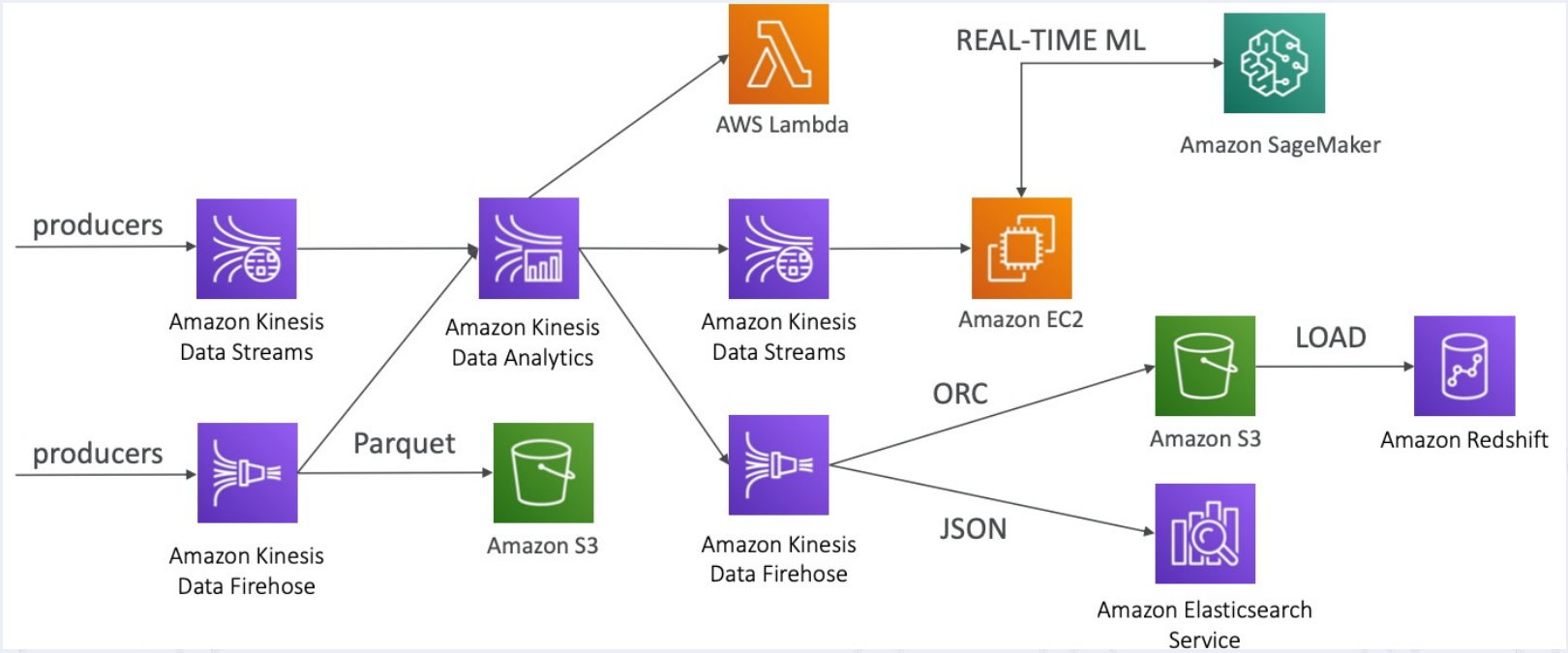
Snowflake Topics

HIGH LEVEL TOPICS FOR THIS WEEK

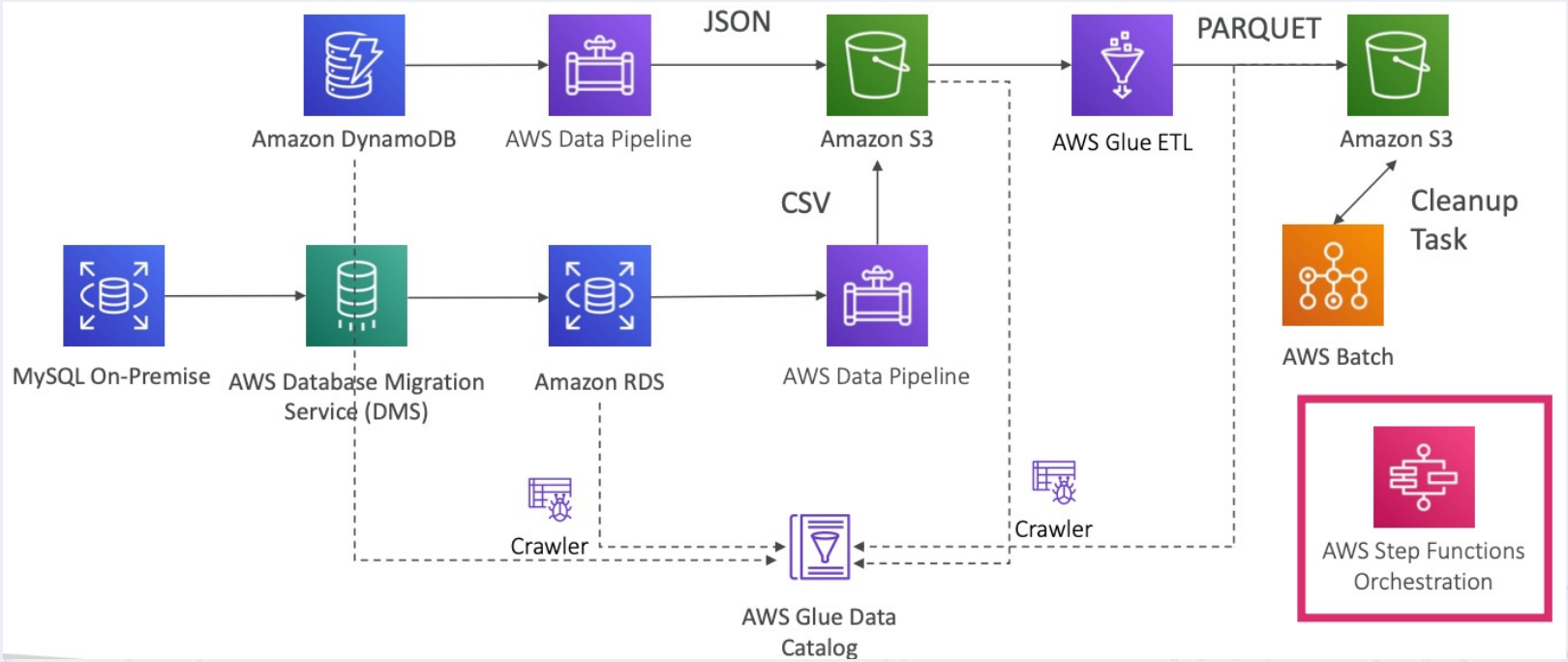
- Data Modeling Concepts
- Data Lakes, Lakehouse, and Delta Lakes
- Databricks Intro (Spark/PySpark/SQL)
- Quick intro to other Data Engineering Tools: Apache Nifi, Informatica, dbt, and AirFlow
- Amazon EMR Intro
- More ETL practice
- More Git Practice
- More Terminal Practice

Example Architectures

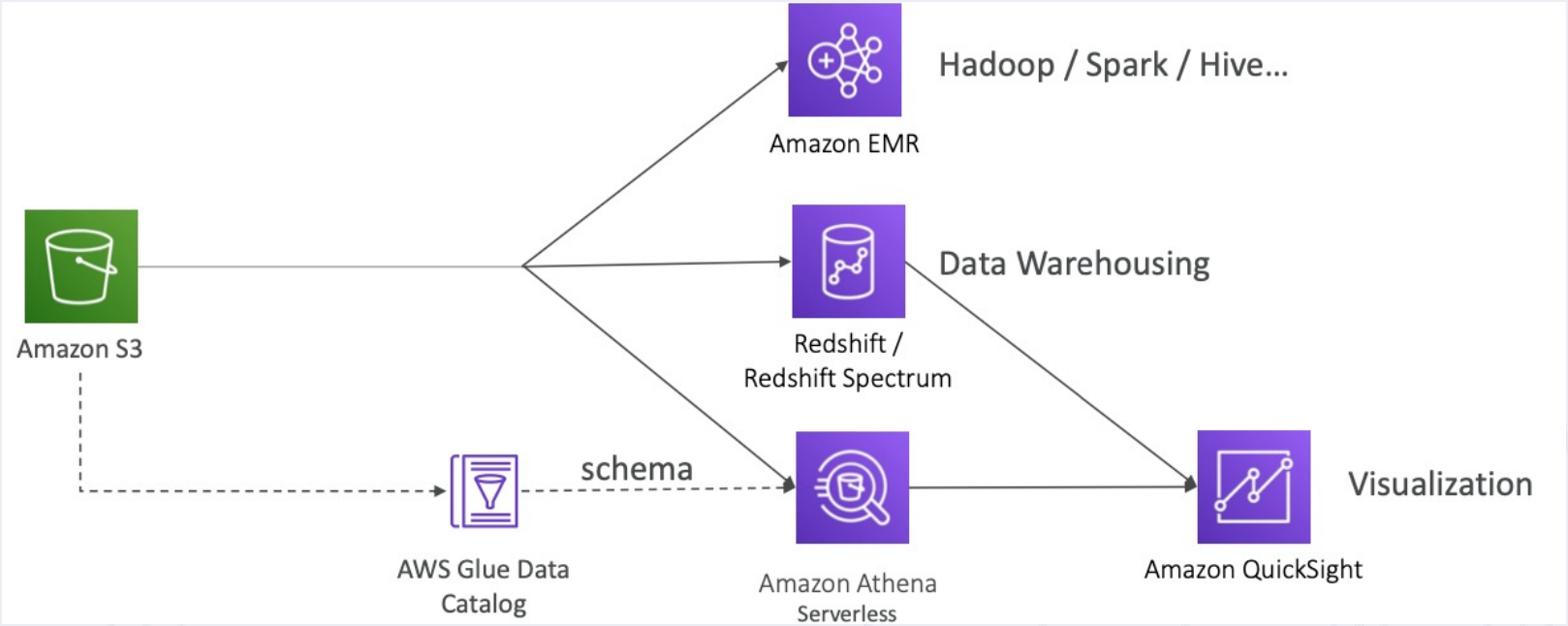
Data Engineering Pipeline (Real Time Layer)



Data Engineering Pipeline (Batch Layer)

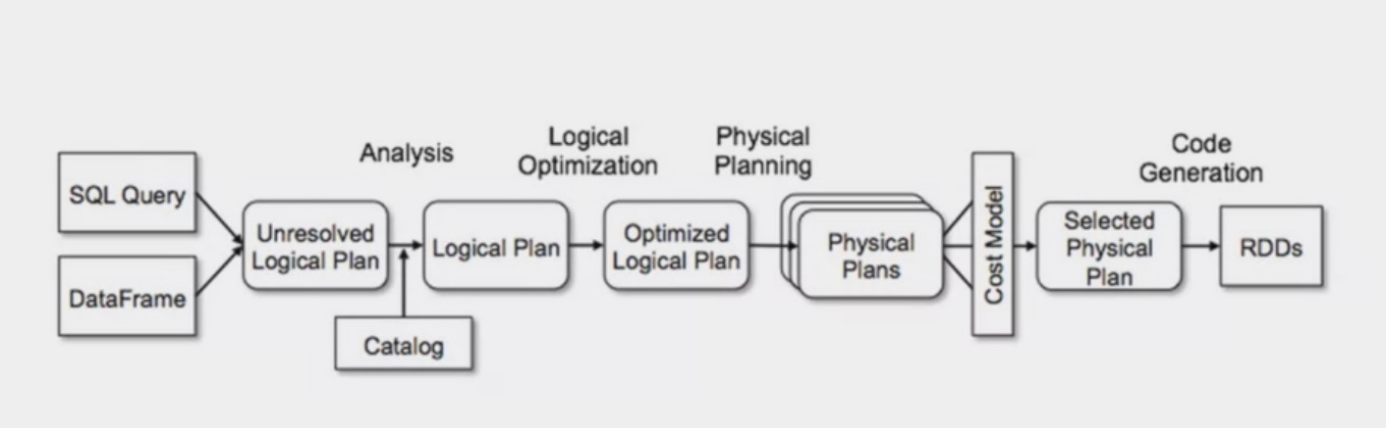


Data Engineering Pipeline (Analytics Layer)

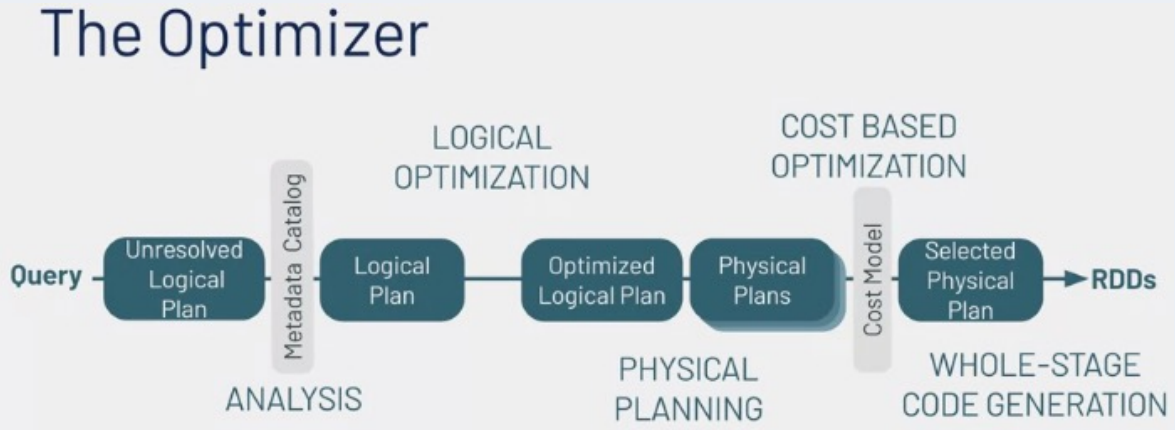


Analytics Services (Overview)

Spark SQL

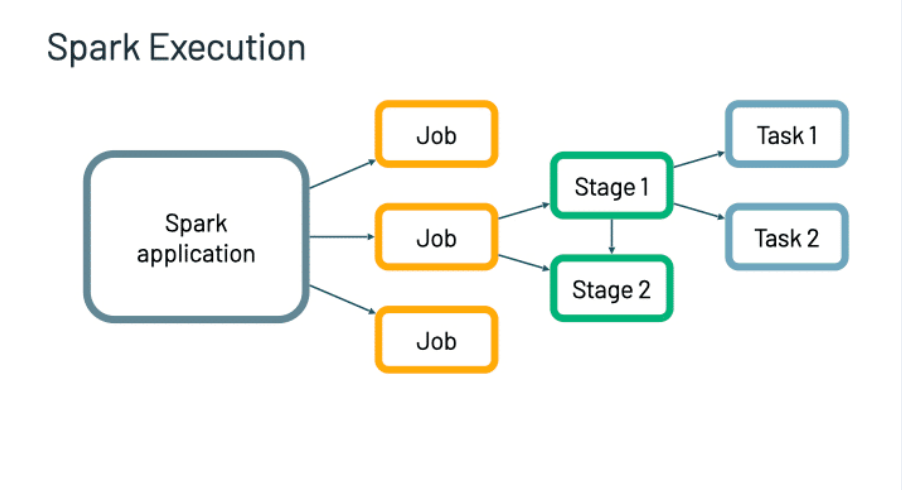


Spark SQL – Catalyst Optimizer



Spark SQL – Spark Execution

1. The secret to Spark's performances is **parallelism**. Each parallelized action is referred to as a **job**. Each job is broken down into **stages**.
2. Each job is broken down into stages, which is a **set of ordered steps** that, together, accomplish a job.
3. Tasks are created by the driver and assigned a partition of data to process. These are the smallest unit of work.



Spark SQL – Spark Execution

Spark SQL is a module used for structured data processing.

- One way to use Spark SQL is executing **SQL queries**
- We can also interact with Spark SQL by using the DataFrame API, available in Scala, Java, Python, and R

The same Spark SQL query can be expressed with **SQL** and the **DataFrame API**

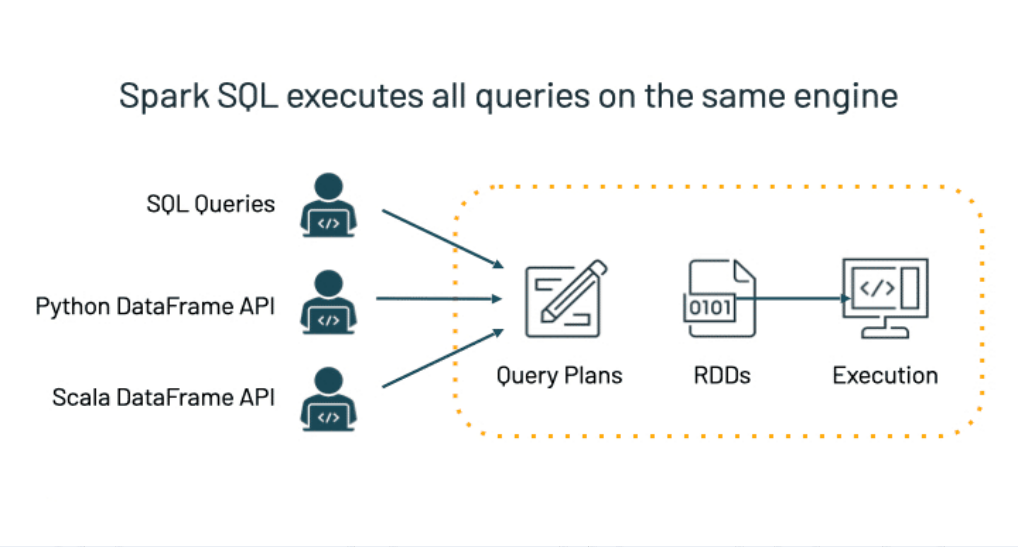
```
SELECT id, result
FROM exams
WHERE result > 70
ORDER BY result
```

```
spark.table("exams")
  .select("id", "result")
  .where("result > 70")
  .orderBy("result")
```

Spark SQL – Spark Execution

You can express a query with any interface, and the Spark SQL engine will generate the same query plan to execute on your Spark cluster.

With Spark SQL, you can easily mix SQL queries inside Spark programs and use a common API across languages.



Spark DataFrame

A **DataFrame** is a **distributed** collection of data grouped into named columns.

- A **Schema** defines the columns names and types of the DataFrame
- A DataFrame **Transformations** are methods that **return a new DataFrame** and are lazily evaluated.
- Select, where, and orderBy are examples of transformations. Because each of these return a DataFrame, we are able to chain these methods together to build new DataFrames.
- **Lazy Evaluation**: DataFrame transformations are not evaluated until an action is called.
- DataFrame **Actions** are methods that trigger computation e.g. df.count(), df.collect(), and df.show().

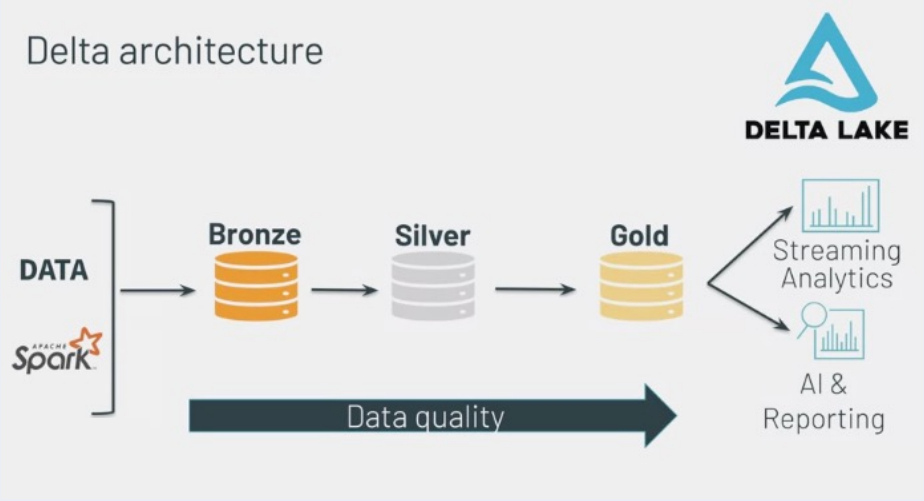
Delta Lake

What is Delta Lake?

- Core component of a data lakehouse
- Offers guaranteed consistency because it's ACID compliant
- Robust data store
- Designed to work with Apache Spark

- Delta Architecture
- Delta Storage Layer
- Delta Engine
- Delta Table

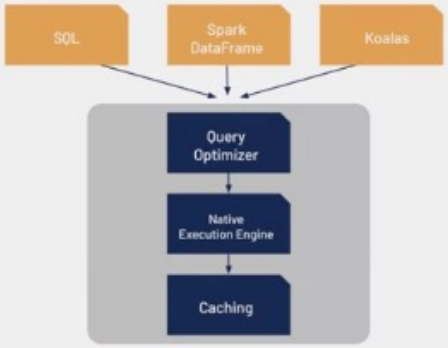
Delta Lake - Medallion architecture



Delta Lake – Delta Engine

Delta Engine

- File management optimizations
- Performance optimization with Delta Caching
- Dynamic File Pruning
- Adaptive Query Execution



DELTA LAKE

Delta Lake – Delta Tables

Delta tables

- Creates Delta files
- Registers the table in the Metastore
- Starts a transaction log

Instead of this:

```
CREATE TABLE events
USING json
AS SELECT *
FROM json.`/data/events/`
```

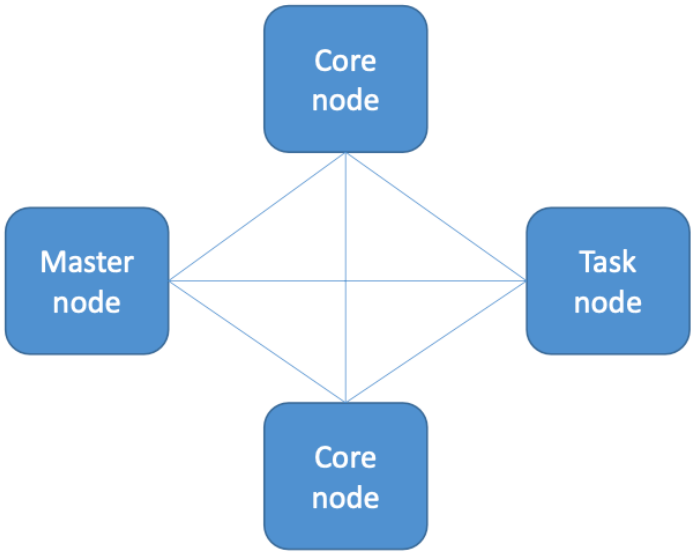
Write this:

```
CREATE TABLE events
USING delta
AS SELECT *
FROM json.`/data/events/`
```

Amazon EMR

- Elastic MapReduce
- Managed Hadoop framework on EC2 instances
- Includes **Spark**, **HBase**, **Presto**, **Flink**, **Hive** & more
- **EMR Notebooks**
- Several integration points with AWS
- Frameworks and applications are specified at cluster launch
- Connect directly to master to run jobs directly
- Or, submit ordered steps via the console
  - Process data in S3 or HDFS
  - Output data to S3 or somewhere
  - Once defined, steps can be invoked via the console

EMR Cluster



- **Master node**: manages the cluster
  - Tracks status of tasks, monitors cluster health
  - Single EC2 instance (it can be a single node cluster even)
  - AKA "leader node"
- **Core node**: Hosts HDFS data and runs tasks
  - Can be scaled up & down, but with some risk
  - Multi-node clusters have at least one
- **Task node**: Runs tasks, does not host data
  - Optional
  - No risk of data loss when removing
  - Good use of **spot instances**

Apache Components in Data Engineering

Distributed Computing and Batch Processing

- Apache Spark
- Apache Flink
- Apache Hadoop
- Apache Tez

Data Query and Management

- Apache Hive
- Apache Pig
- Apache Drill
- Apache Phoenix
- Apache Presto
- Apache Trino

Data Ingestion and Integration

- Apache Flume
- (Apache Kafka, although not listed, fits here as well)

User Interfaces and Visualization

- Apache Hue
- Apache Zeppelin

AWS Glue

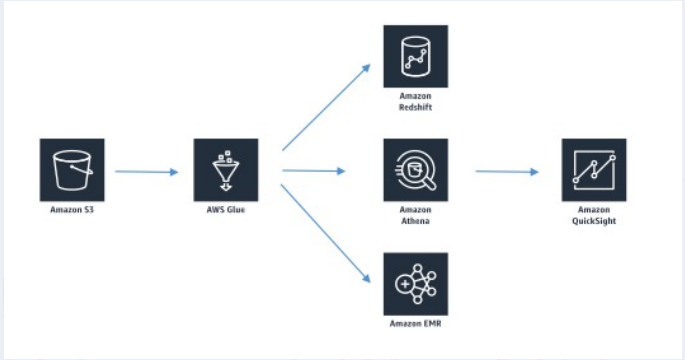
Glue ETL is based on **Spark**

If you want to use other engines (Hive, Pig, etc) Data Pipeline EMR would be a better fit.

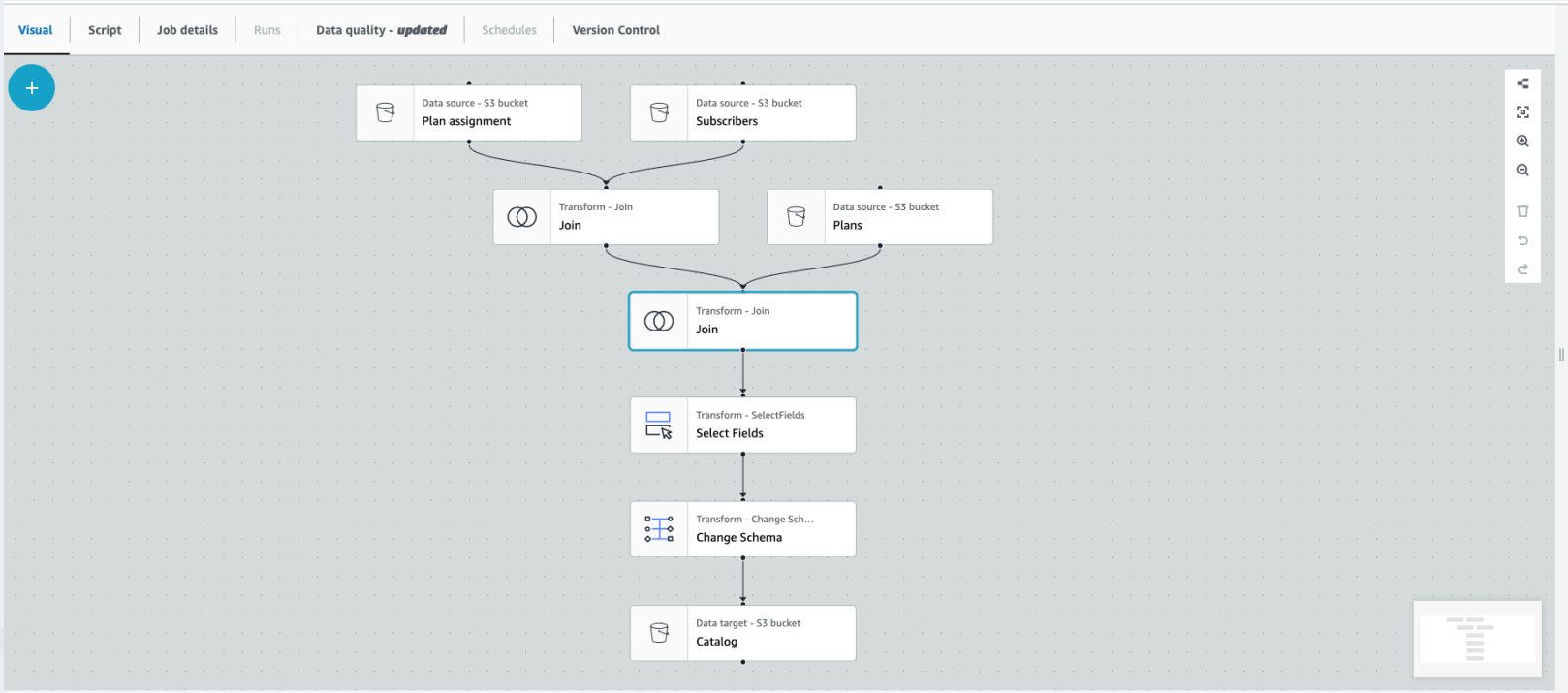
AWS Glue

Glue Crawler / Data Catalog

- Glue crawler scans data in S3, creates schema
- Can run periodically
- Populates the Glue Data Catalog
  - Stores only table definition
  - Original data stays in S3
- Once cataloged, you can treat your unstructured data like it's structured
  - Redshift Spectrum
  - Athena
  - EMR
  - Quicksight



AWS Glue Studio (Visual ETL)



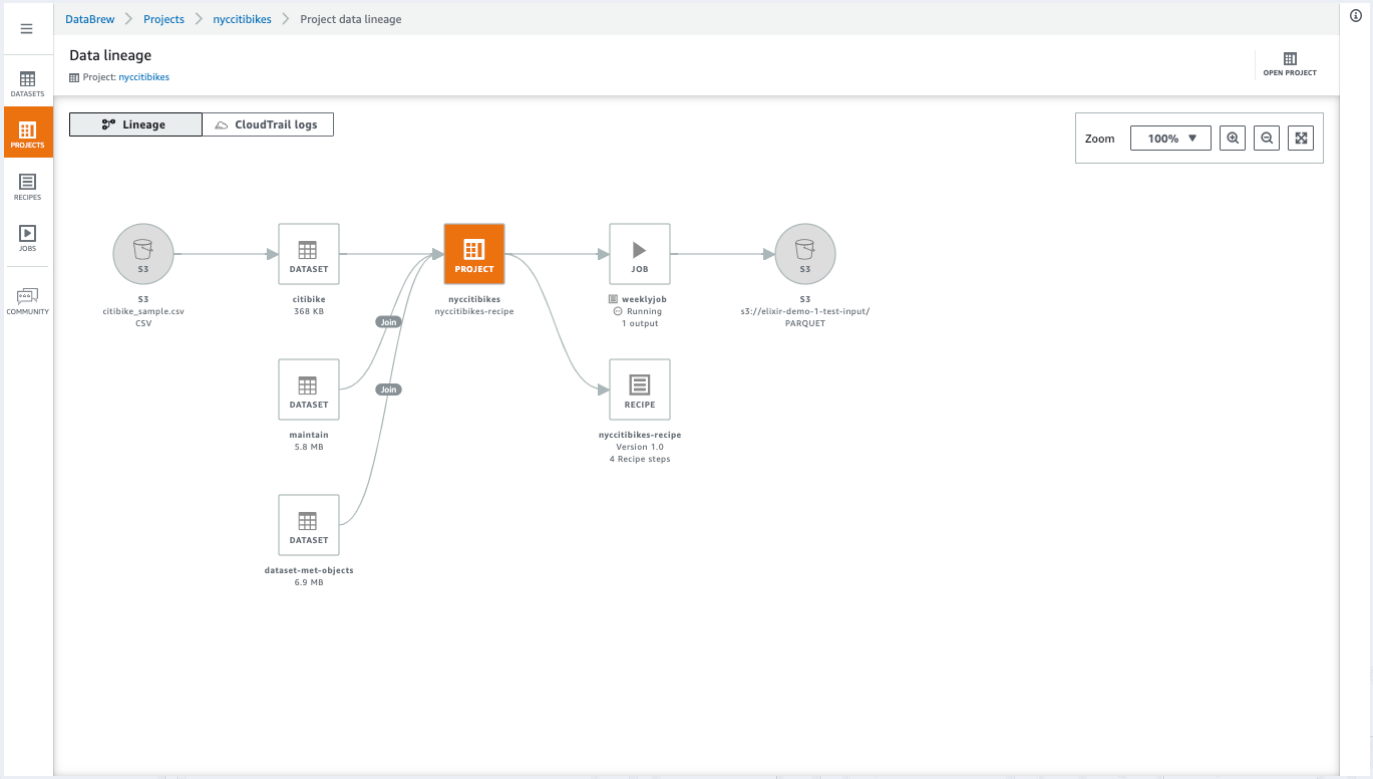
AWS Glue Studio

AWS Glue Studio
Use when: You need to create, run, and monitor ETL jobs visually and want a centralized view of the overall ETL workflow.

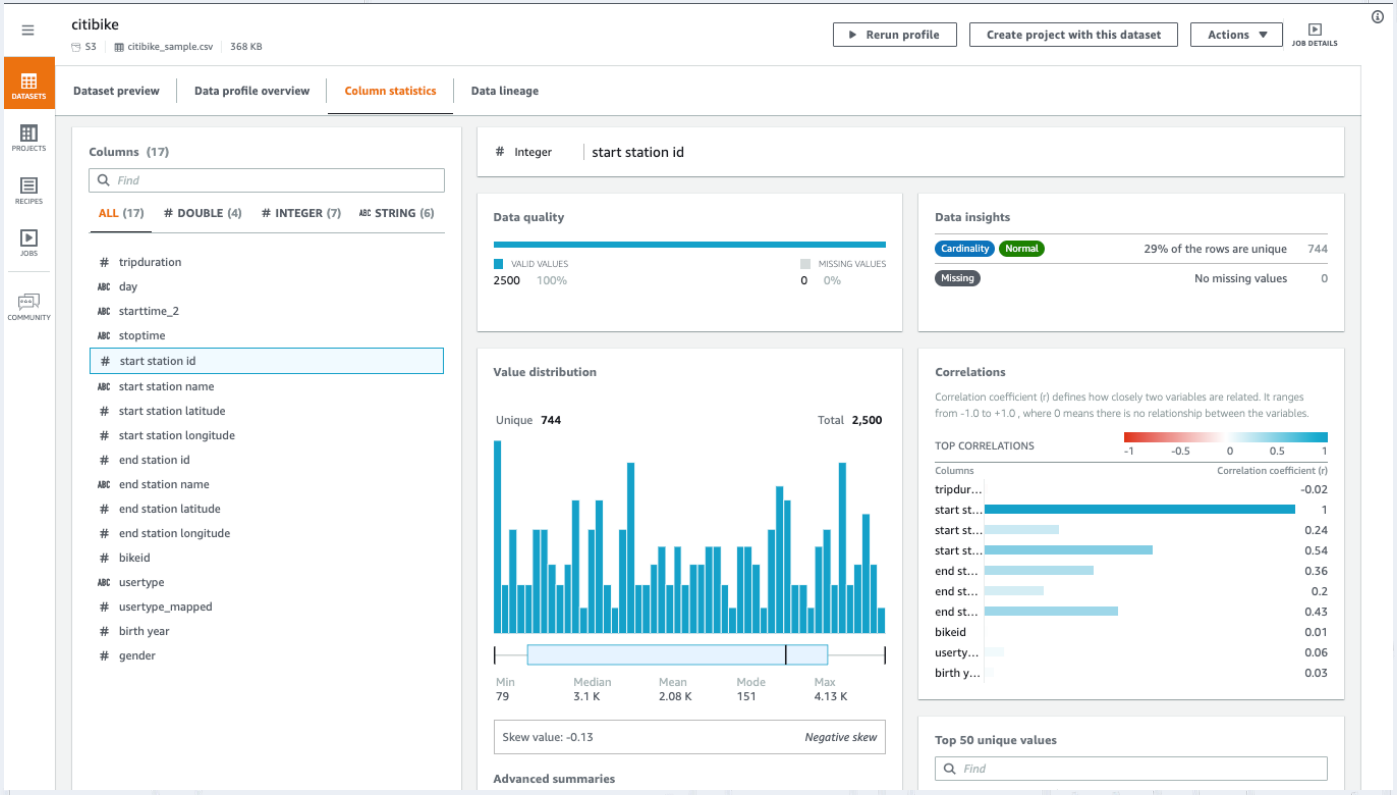
Key features:

- Visual interface for creating and managing ETL jobs.
- Provides a single pane view for monitoring and managing jobs.
- Supports data transformation and data profiling.
- Integrated with AWS Glue DataBrew for advanced data preparation.

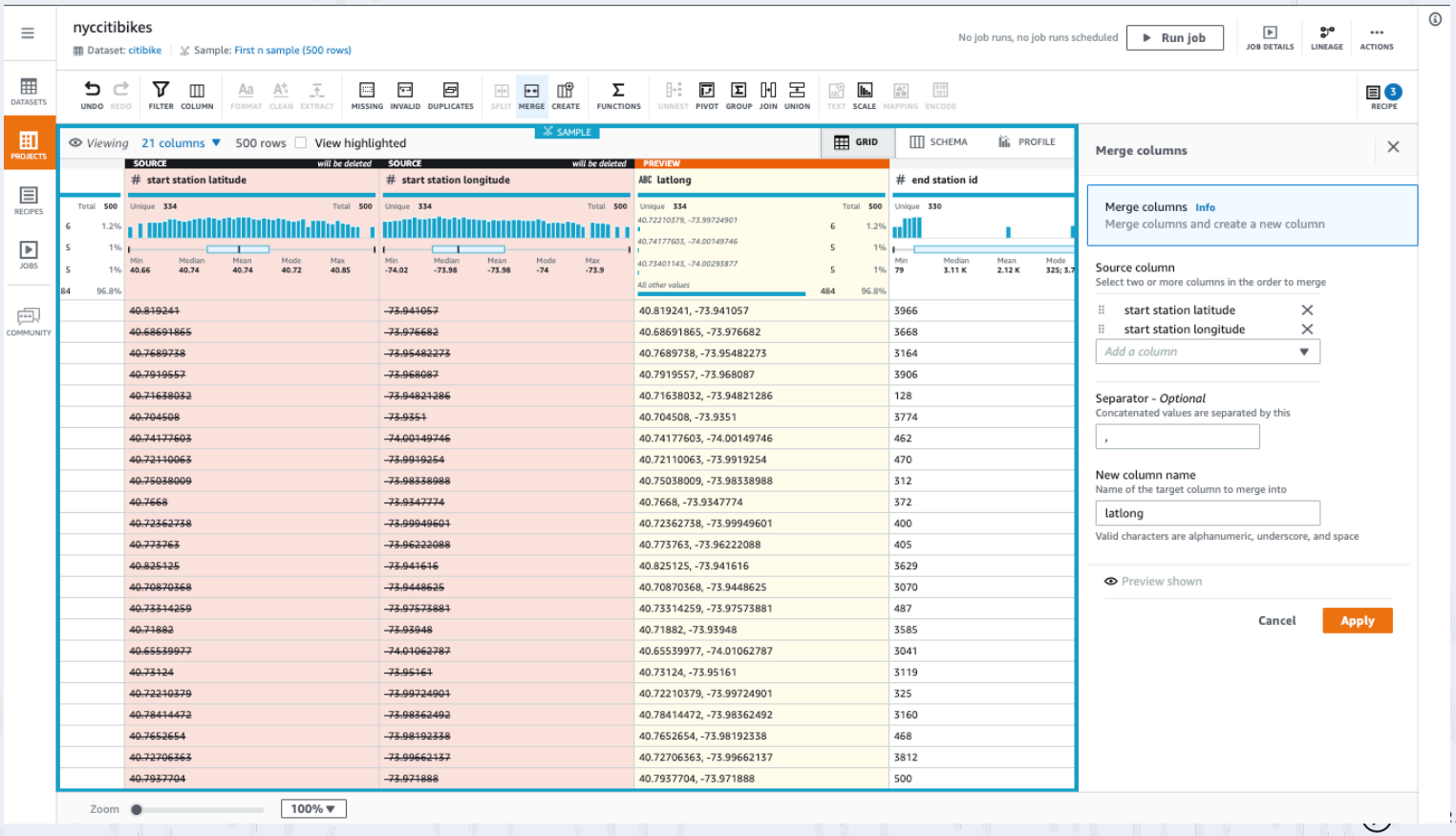
AWS Glue DataBrew



AWS Glue DataBrew



AWS Glue DataBrew



AWS Glue Studio

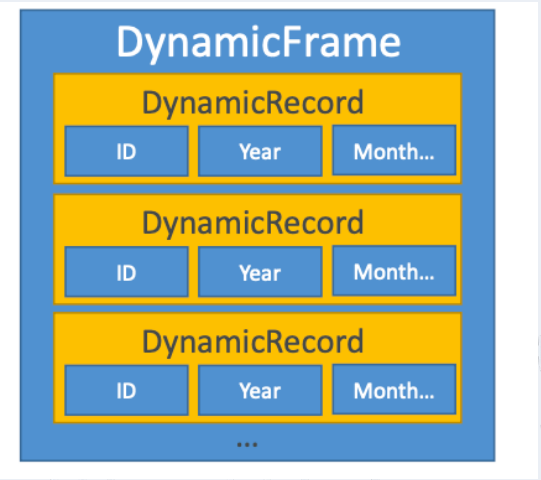
AWS Glue DataBrew
Use when: You need to prepare and transform data for analysis or machine learning purposes without writing code.

Key features:
- Visual data preparation tool for cleaning and normalizing data.
- Provides over 200 transformations without requiring code.
- Supports data profiling and data quality checks.
- Integrated with AWS Glue Studio for advanced ETL workflows.

AWS Glue

DynamicFrame is a collection of **DynamicRecords**

- **DynamicRecords** are self-describing and have a schema
- Very similar to Spark DataFrame but with more ETL stuff
- Scala and Python APIs



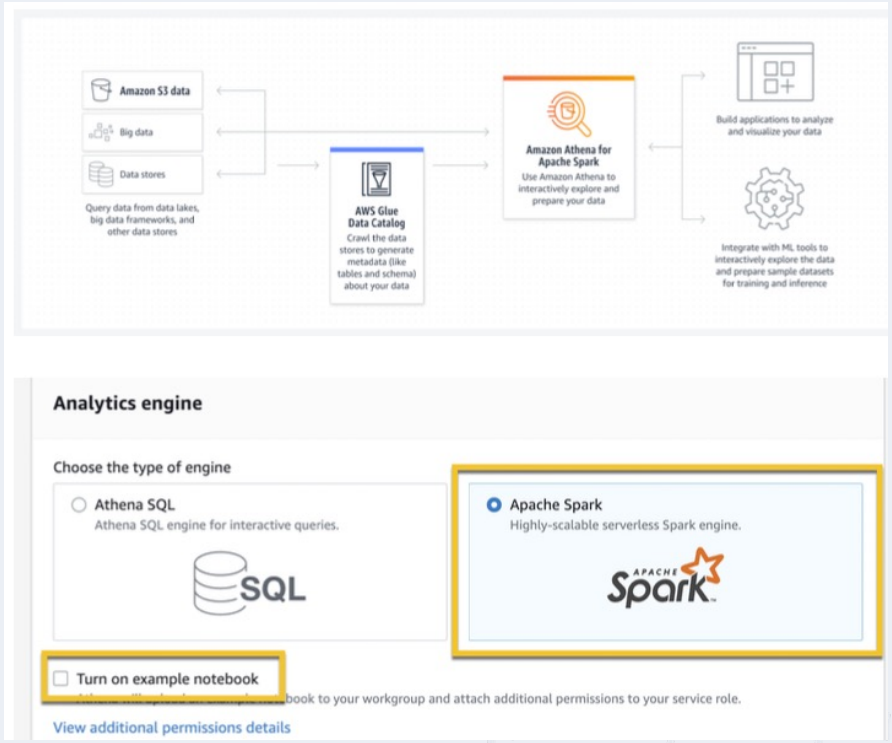
Athena ACID Transactions

- Powered by **Apache Iceberg**. Just add 'table_type' = 'ICEBERG' in your **CREATE TABLE** command
- Concurrent users can safely make row-level modifications
- Compatible with **EMR**, **Spark**, anything that supports Iceberg table format.
- Removes need for custom record locking
- Time travel operations: Recover data recently deleted with a SELECT
- Creates a new table from query results (CTAS)

```
CREATE TABLE new_table
WITH (
    format = 'Parquet',
    write_compression = 'SNAPPY')
AS SELECT *
FROM old_table;
```

```
CREATE TABLE my_orc_ctas_table
WITH (
    external_location =
's3://my_athena_results/my_orc_stas_table/',
    format = 'ORC')
AS SELECT *
FROM old_table;
```

Athena for Apache Spark



Spark Streaming

Spark Streaming

Spark Streaming + Kinesis

Spark Structured Streaming

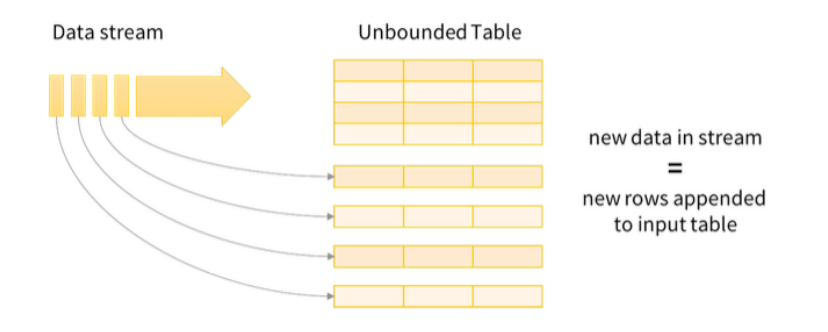
Spark Structured Streaming is **a high-level API** for **stream processing** that allows users to process streaming data using the same structured APIs (DataFrames and Datasets) as Spark's batch processing.

This approach enables developers to express streaming computations in a similar way to batch computations, leveraging the optimized Spark SQL engine for incremental and continuous processing of data streams.

Spark Structured Streaming

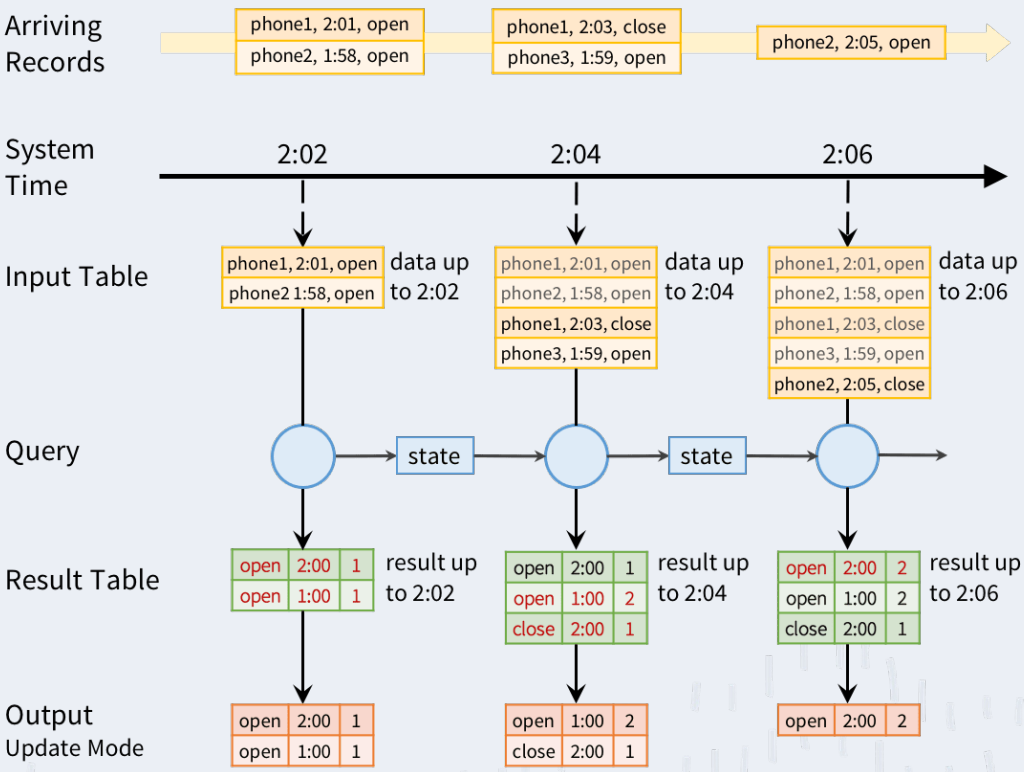
Structured Streaming treats all the data arriving as an unbounded input table. Each new item in the stream is like a row appended to the input table

The developer then defines a **query** on this input table, as if it were a **static table**, to compute a final result table that will be **written** to an output sink. Spark automatically converts this batch-like query to a streaming execution plan. This is called **incrementalization**: Spark figures out what state needs to be maintained to update the result each time a record arrives.



Data stream as an unbounded Input Table

Spark Structured Streaming



Spark Streaming

Streaming ETL

- Continuous extraction, transformation, and loading of data from various sources to data stores.

Data Enrichment

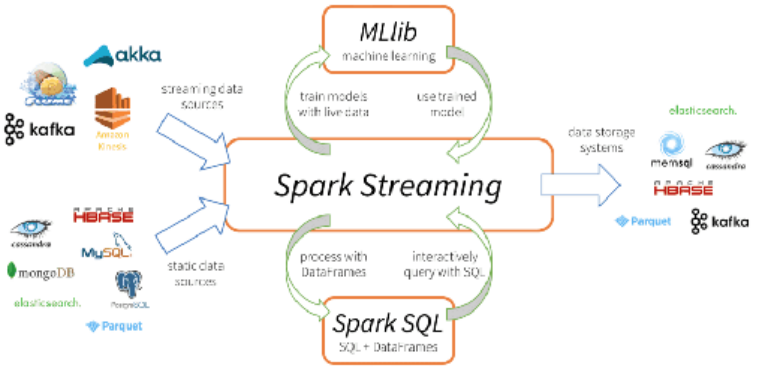
- Merging real-time data with other datasets to enable comprehensive, real-time analysis.

Triggers

- Detecting abnormal behavior in real-time and triggering downstream actions.
- Example: Responding to unusual sensor device behavior, such as thumbprint recognition anomalies.

Complex Sessions and Continuous Learning

- Updating machine learning models based on live session information.
- Example: Adapting models after user login on a website or application.

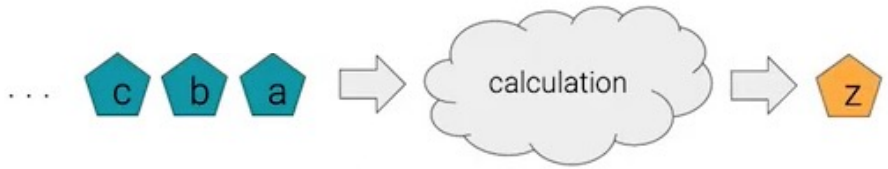


Stream Processing

What is Stream Processing?

A **stream** is a potentially unbounded sequence of data

Stream Processing is the act of performing continual calculations on a stream

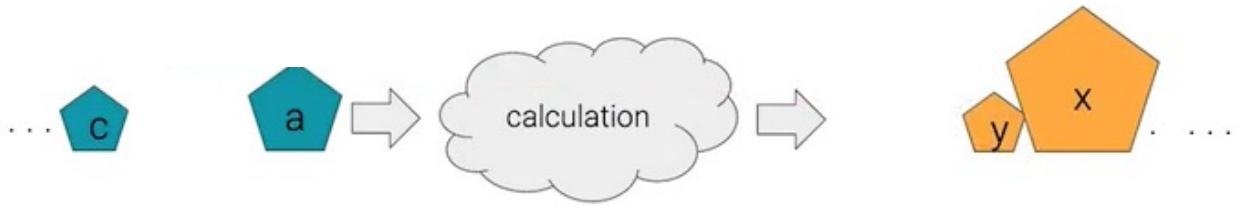


Stream Processing

What is Stream Processing?

Data may be produced at an **even** or **constant** rate

Data may also be produced **unevenly** and in **different shapes and sizes**



Stream Processing

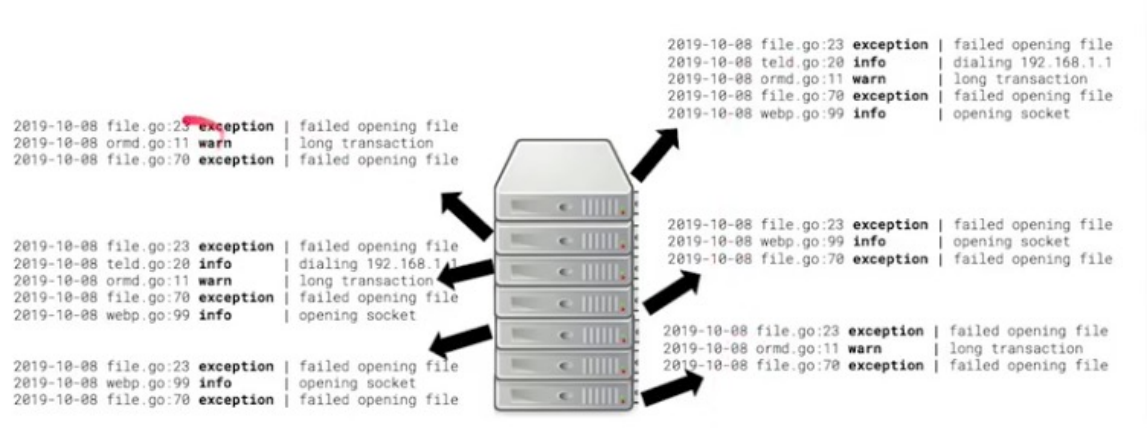
- Data streams are made up of **immutable** data. Data cannot be changed once in the stream
- Data records in streams is typically small, usually less than 1 MB
- Data throughout may range from one record a second up to many thousands per second
- You can receive thousands of streams per seconds, so these streaming systems need a high throughout even though those stream are small in size

What is an Event

- An event is an **immutable fact** regarding something that occurred within a system
- **Event** systems **react** to the facts communicated to them. The communication is **indirect** and the events they utilize are usually not **specifically targeted** to any one system
- The events emitted do not target a specific downstream system

Example of Stream Processing

Log Analysis: Logs are hard to process in batch systems due to the speed and size of data



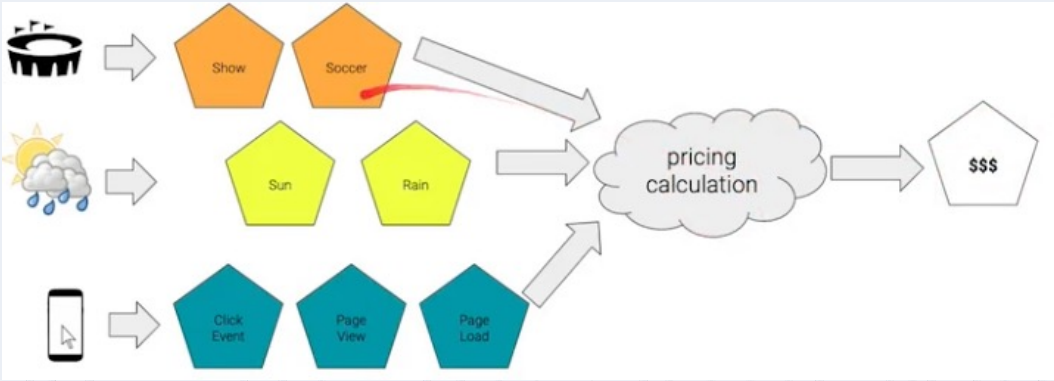
- Logs are being created at a very fast rate
- Companies push **log data** as **Events** into a **data stream** to perform stream processing, perform analytics to find insights from the data
- As log happens they are being pushed as streams

Example of Stream Processing

Web Analytics: Modern web applications track user actions such as clicks and page views

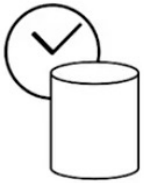
Stream processing allows companies to process data as it's generated and not ours after the fact as is common with batch processing

Real-Time Pricing: Real time pricing adjust to environmental and instantaneous demand



Batch vs Stream Processing

Batch Processing



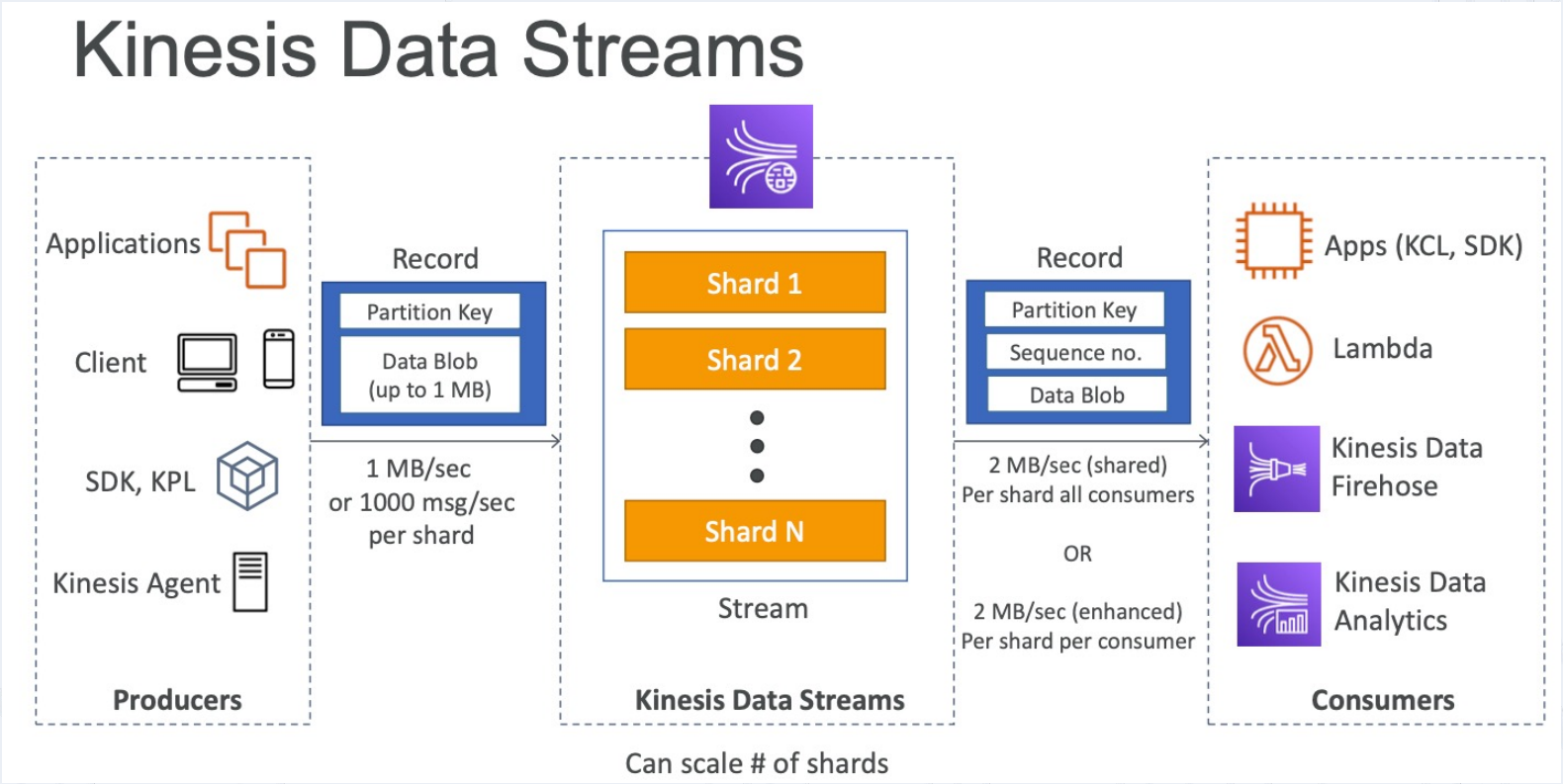
- Scheduled analysis of related groups of data
- As up to date as the last scheduled run
- May run for long periods of time
- Often involve mutable data stores
- May access all historical data

Stream Processing

- Real-time analysis of data as it is produced
- Runs as soon as event is produced
- As up to date as the last event generated
- Often involves immutable data stores
- Often uses recently produced, windowed data

- The differences are **generalizations, not hard rules**
- Most data engineering teams use both **batch** and **stream processing together** to achieve their goals.

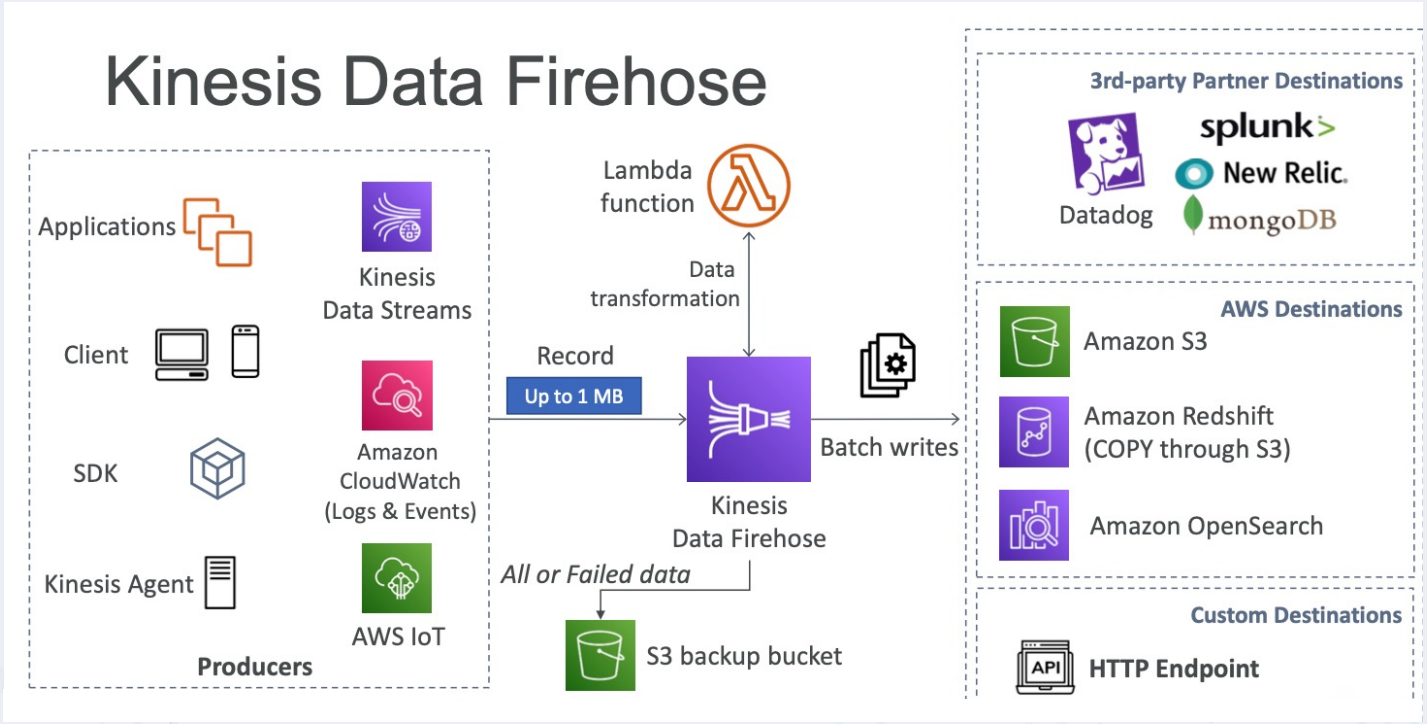
Kinesis Data Streams



Kinesis Data Streams

- Retention between 1 day to 365 days
- Ability to reprocess (replay) data
- Once data is inserted in Kinesis, it can't be deleted (immutability)
- Data that shares the same partition goes to the same shard (ordering)
- **Producers**:
  AWS SDK, Kinesis Producer Library (KPL), Kinesis Agent
- **Consumers**:
  Write your own: Kinesis Client Library (KCL), AWS SDK
  Managed: AWS Lambda, Kinesis Data Firehose, Kinesis Data Analytics

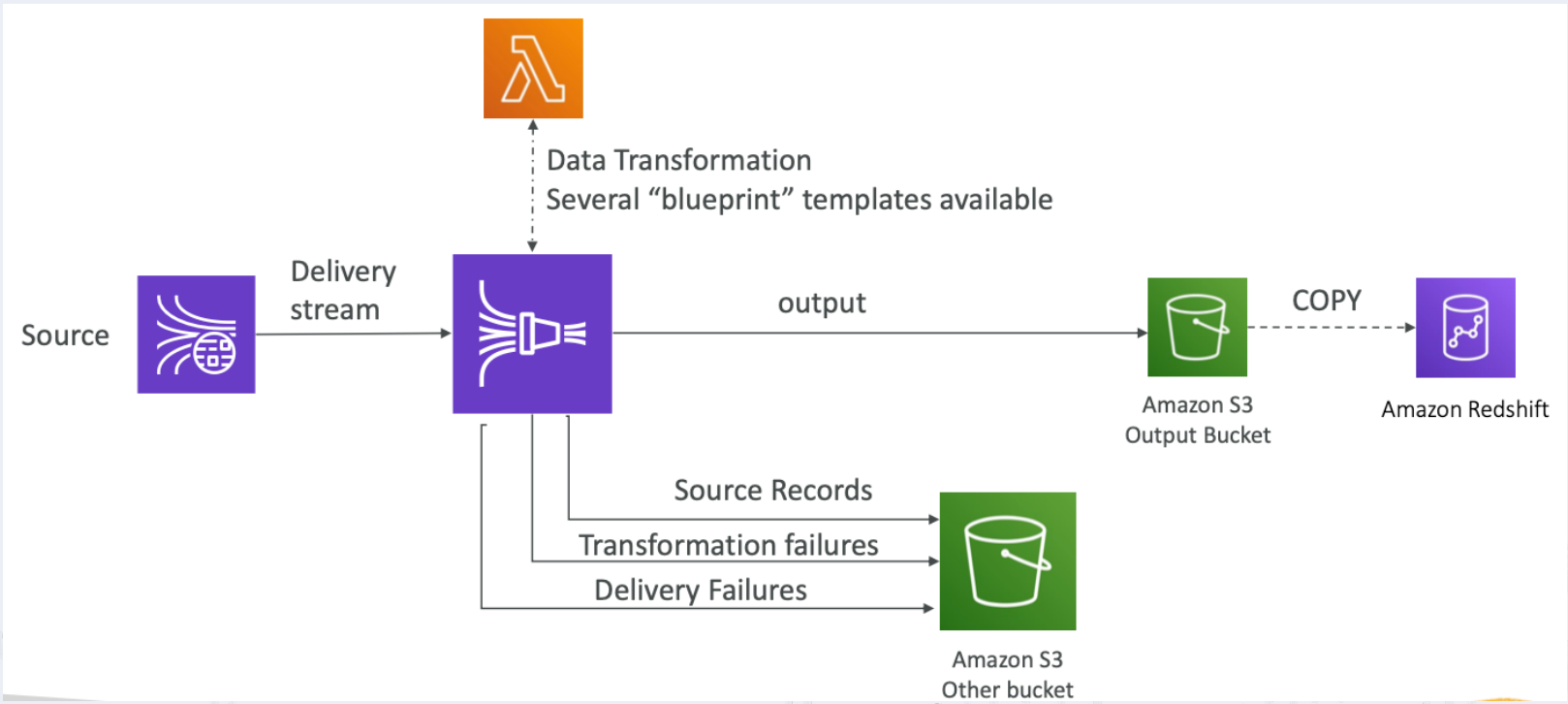
Kinesis Data Firehose



Kinesis Data Firehose

- Fully Managed Service, no administration
- **Near Real Time** (**Buffer based on time and size, optionally can be disabled**)
- Load data into Redshift / Amazon S3 / OpenSearch / Splunk
- Automatic scaling
- Supports many data formats
- Data Conversions from JSON to Parquet / ORC (only for S3)
- Data Transformation through AWS Lambda (ex: CSV => JSON)
- Supports **compression** when target is Amazon S3 (GZIP, ZIP, and SNAPPY)
- Only GZIP is the data is further loaded into Redshift
- Pay for the amount of data going through Firehose

Kinesis Data Firehose Delivery



Kinesis Data Firehose Buffer Sizing

- Firehose accumulates records in a buffer
- The buffer is flushed based on time and size rules
- Buffer Size (ex: 32MB): if that buffer size is reached, it's flushed
- Buffer Time (ex: 2 minutes): if that time is reached, it's flushed
- Firehose can automatically increase the buffer size to increase throughput

  High throughput => Buffer Size will be hit
  Low throughput => Buffer Time will be hit

Kinesis Firehose vs Kinesis Data Streams

Streams

- Going to write custom code (producer / consumer)
- Real time (~200 ms latency for classic, ~70 ms latency for enhanced fan out)
- Must manage scaling (shard splitting / merging)
- Data Storage for 1 to 365 days, replay capability, multi consumers
- Use with Lambda to insert data in real-time to OpenSearch (for example)

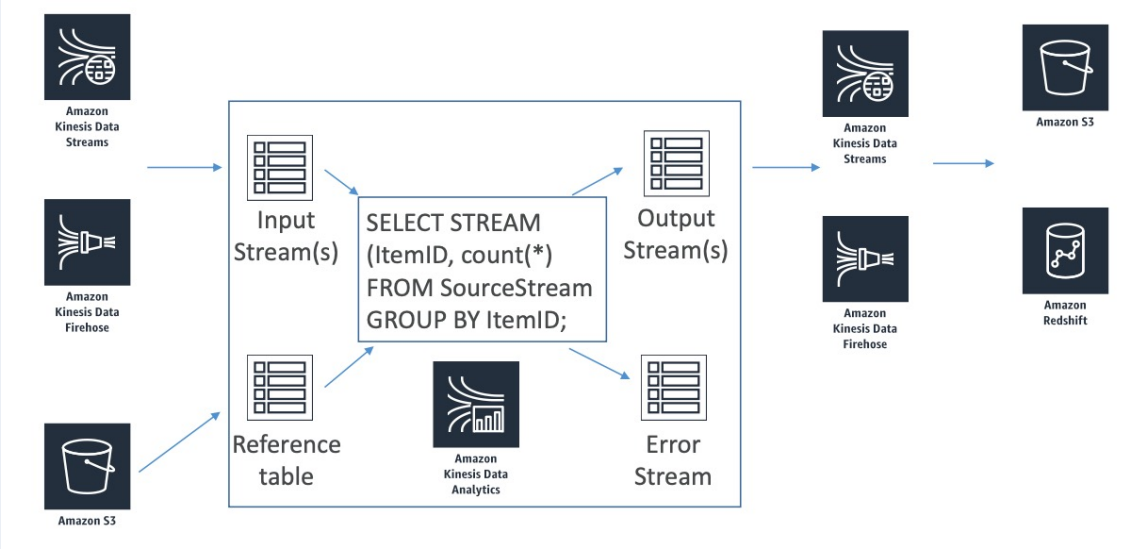
Firehose

- Fully managed, send to S3, Splunk, Redshift, OpenSearch
- Serverless data transformations with Lambda
- Near real time
- Automated Scaling
- No data storage

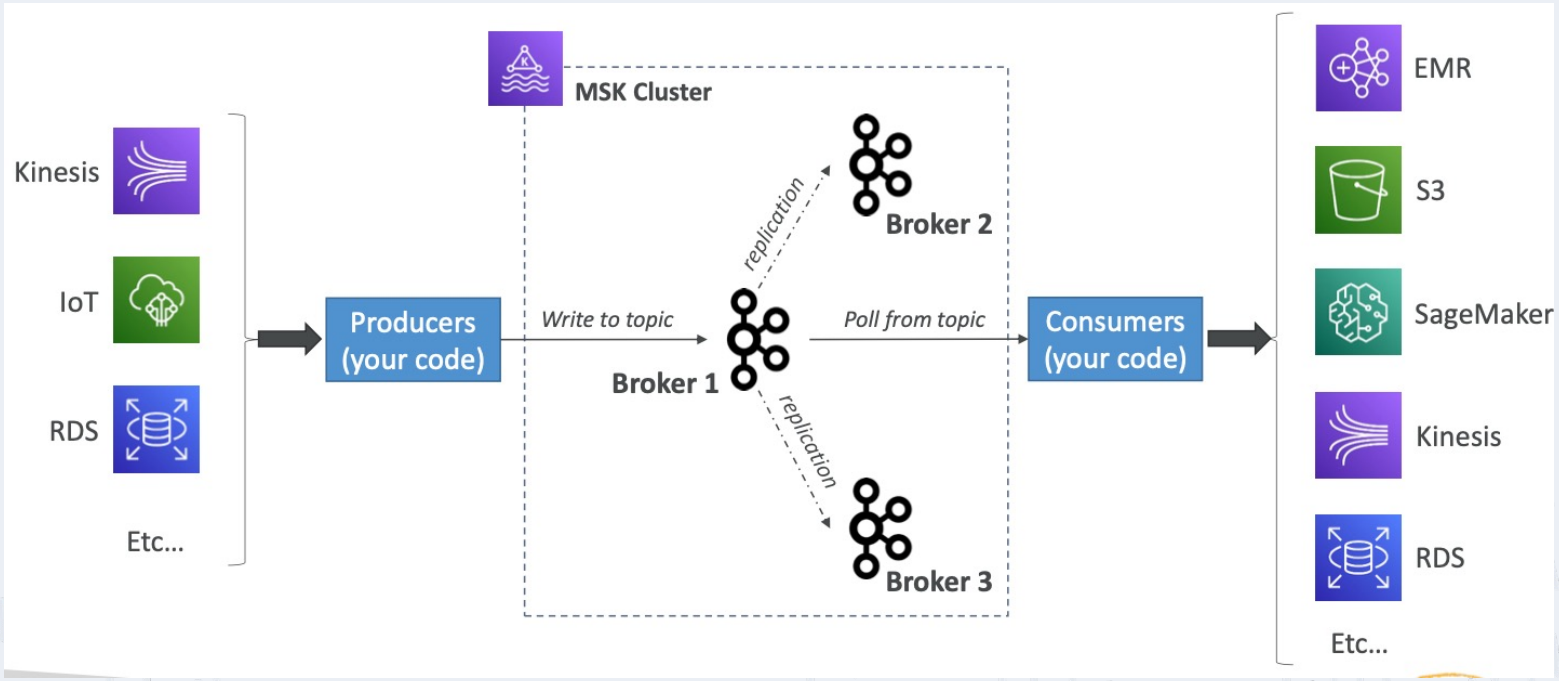
Kinesis Data Analytics (Apache Flink)

Common Use Cases:

- Streaming ETL
- Continuous metric generation
- Responsive analytics

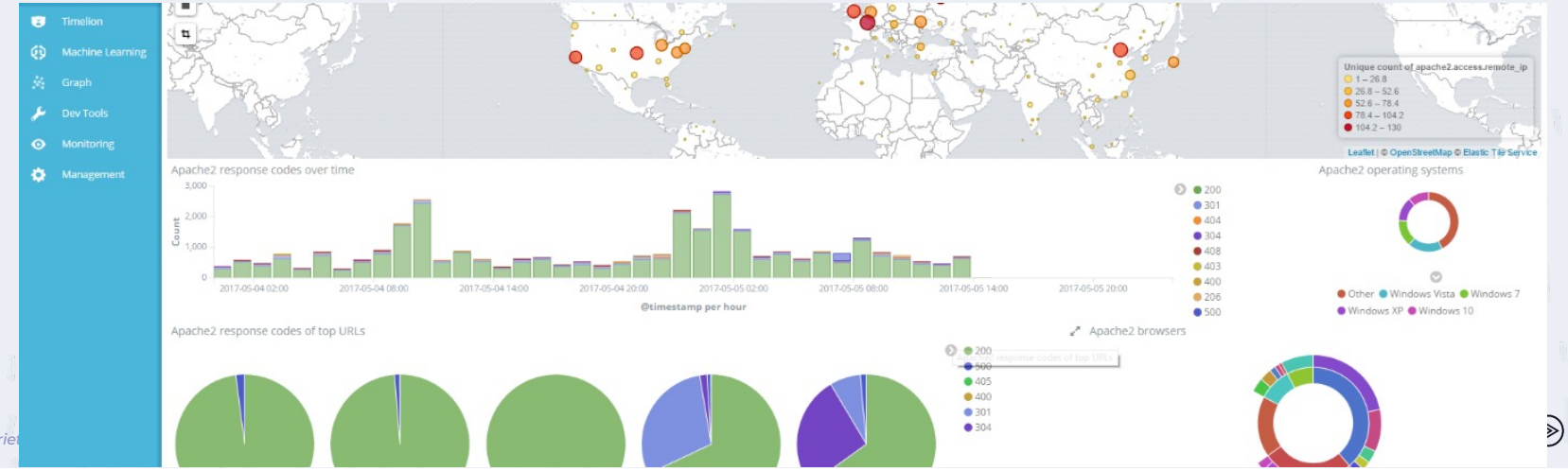


Apache Kafka



Amazon OpenSearch (formerly Elasticsearch)

- A fork of Elasticsearch and Kibana
- A search engine
- An analysis tool
- A visualization tool (Dashboards = Kibana)
- A data pipeline
- Kinesis replaces Beats & LogStash
- Horizontally scalable



Amazon OpenSearch (formerly Elasticsearch)

- Log files
- Messages
- Metrics
- Config info
- Documents and lists

Input

Capture, process, and load data into Amazon OpenSearch Service

Amazon OpenSearch Service

Includes built-in OpenSearch Dashboards and Kibana (1.5 to 7.10 versions)

- Application & infrastructure monitoring
- Security info & event management (SIEM)
- Search

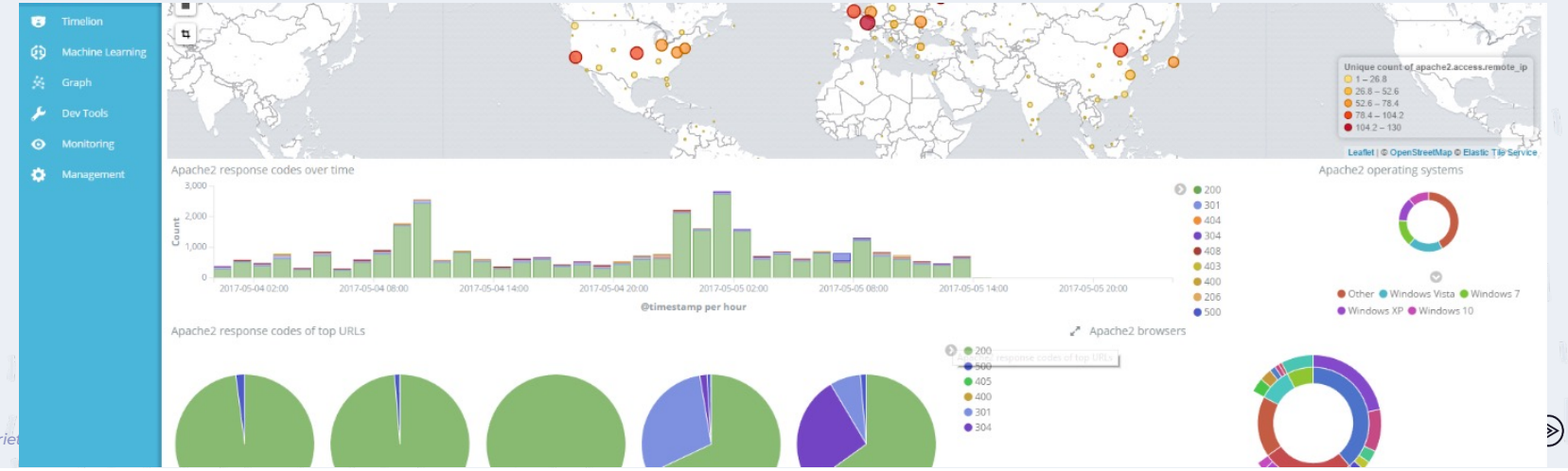
Output

Search, analyze, and visualize logs to get real-time insights

Amazon OpenSearch (formerly Elasticsearch)

Use Cases:

- Full-text search
- Log analytics
- Application monitoring
- Security analytics
- Clickstream analytics

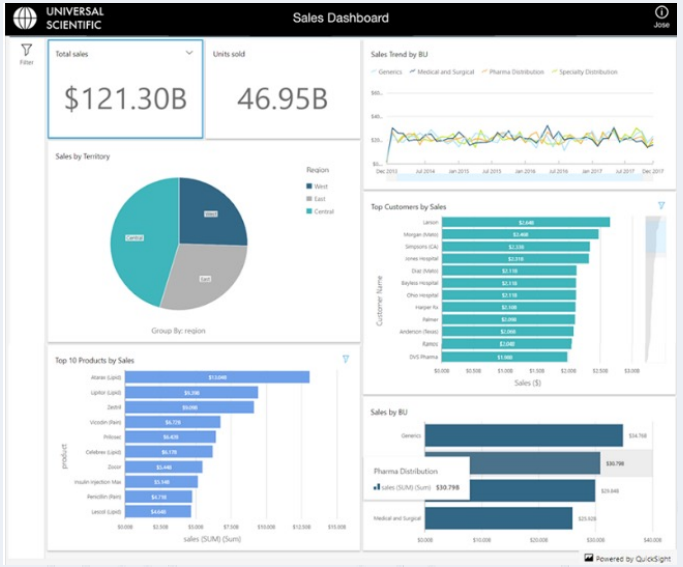


Amazon QuickSight

Fast, easy, cloud-powered business analytics service

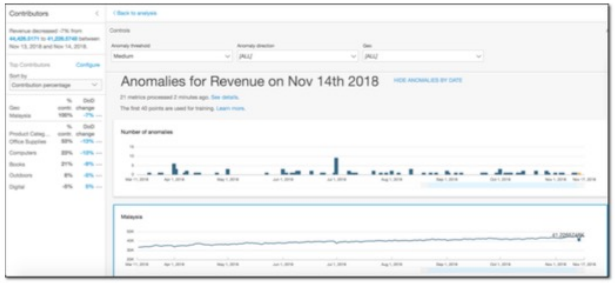
Allows all employees in an organization to:

- Build visualizations
- Build paginated reports
- Perform ad-hoc analysis
- Get alerts on detected anomalies
- Quickly get business insights from data
- Anytime, on any device (browsers, mobile)
- Serverless



Amazon QuickSight Machine Learning Insights

- ML-powered anomaly detection
  - Uses Random Cut Forest
  - Identify top contributors to significant changes in metrics
- ML-powered forecasting
  - Also uses Random Cut Forest
  - Detects seasonality and trends
  - Excludes outliers and imputes missing values
- Autonarratives
  - Adds “story of your data” to your dashboards
- Suggested Insights
  - “Insights” tab displays read-to-use suggested insights

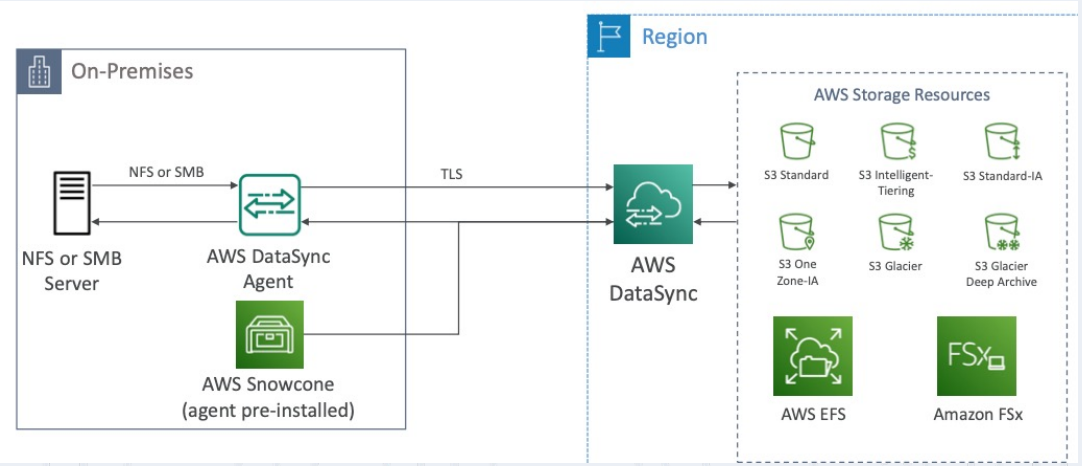


Total Revenue for Nov 17, 2018 **decreased by 2.15% (-131,031.34720000066)** from 6,100,697.1616 to 5,969,665.8144. Compounded growth rate for the last 4 days is -0.26% **worse than expected.**

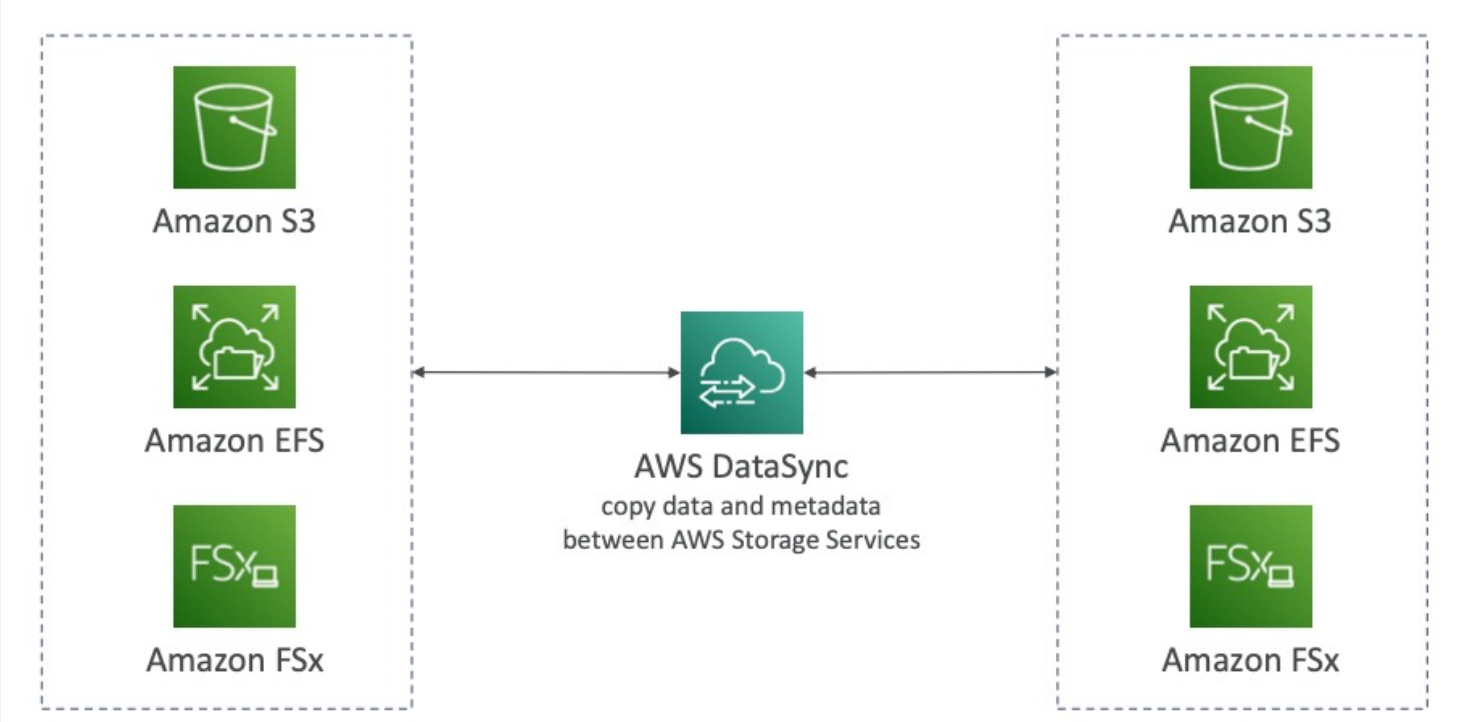
AWS DataSync

AWS DataSync

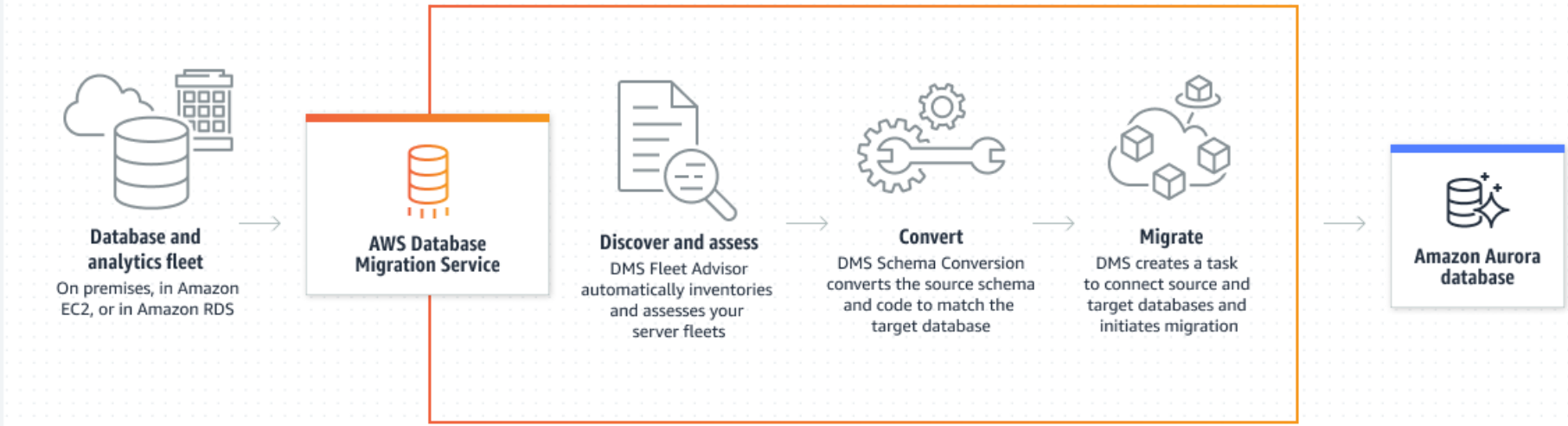
- Move large amount of data to and from
  - On-premises / other cloud to AWS (NFS, SMB, HDFS, S3 API...) **– needs agent**
  - AWS to AWS (different storage services) – no agent needed
- Can synchronize to Amazon S3 (any storage classes – including Glacier)
- Replication tasks can be scheduled hourly, daily, weekly
- **File permissions and metadata are preserved** (NFS POSIX, SMB...)
- One agent task can use 10 Gbps, can setup a bandwidth limit



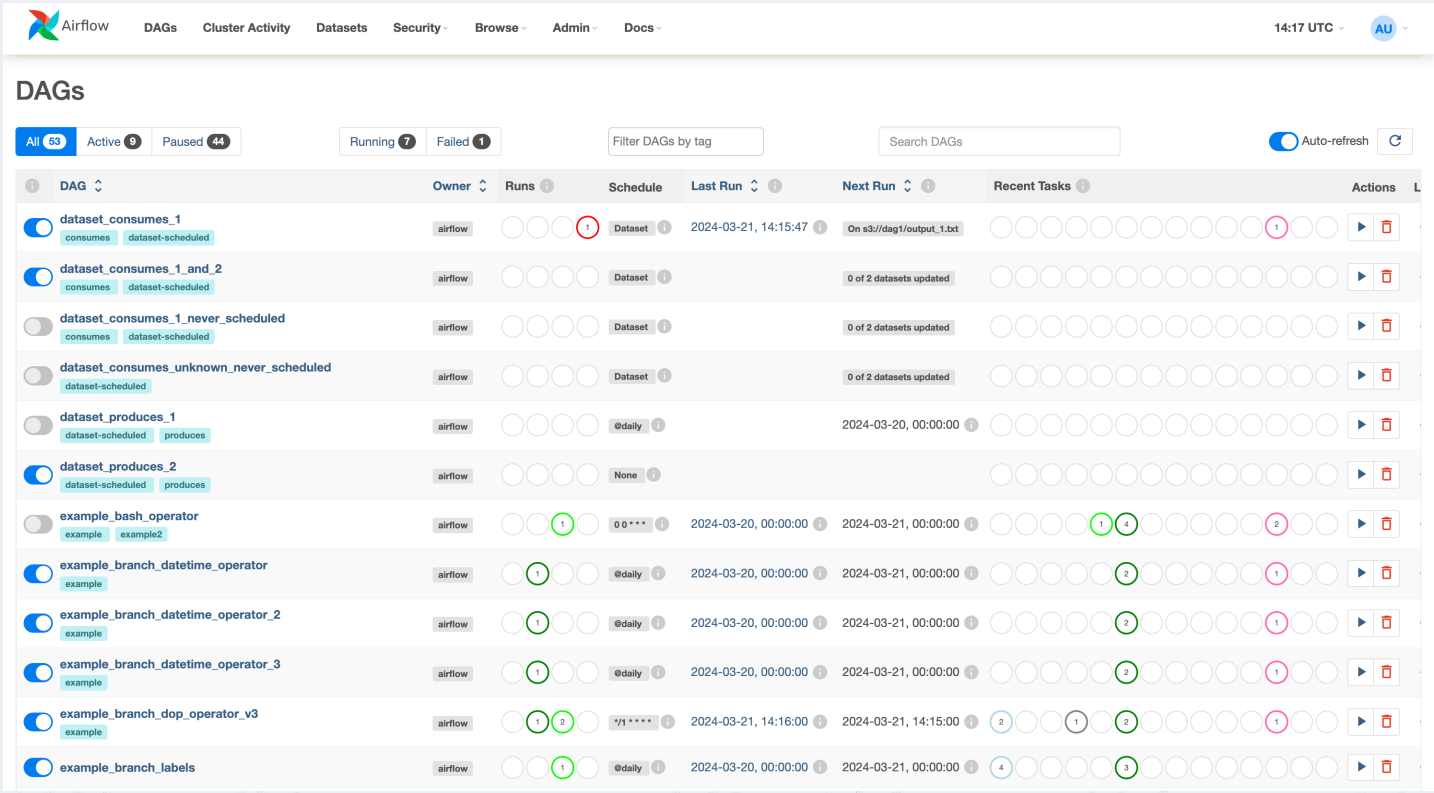
AWS DataSync – Transfer between AWS Storage Services



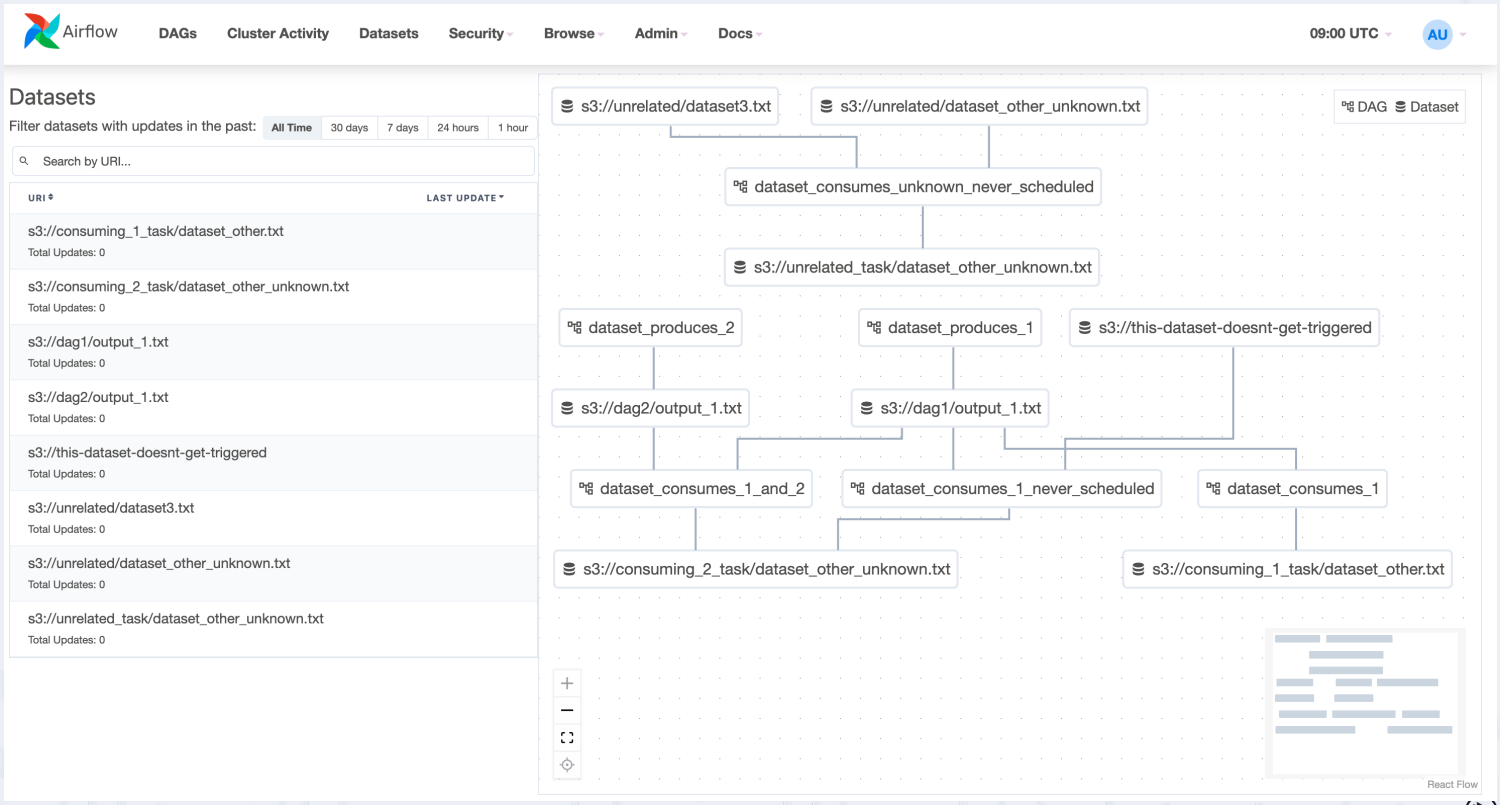
AWS DMS (Database Migration Services)



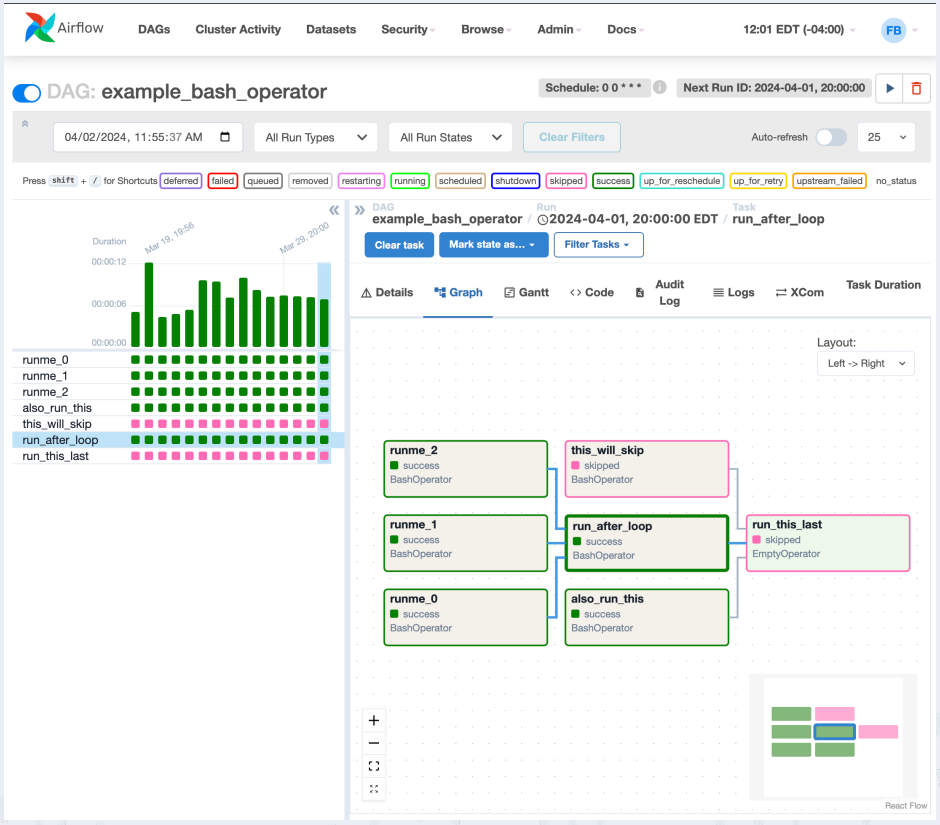
Apache Airflow



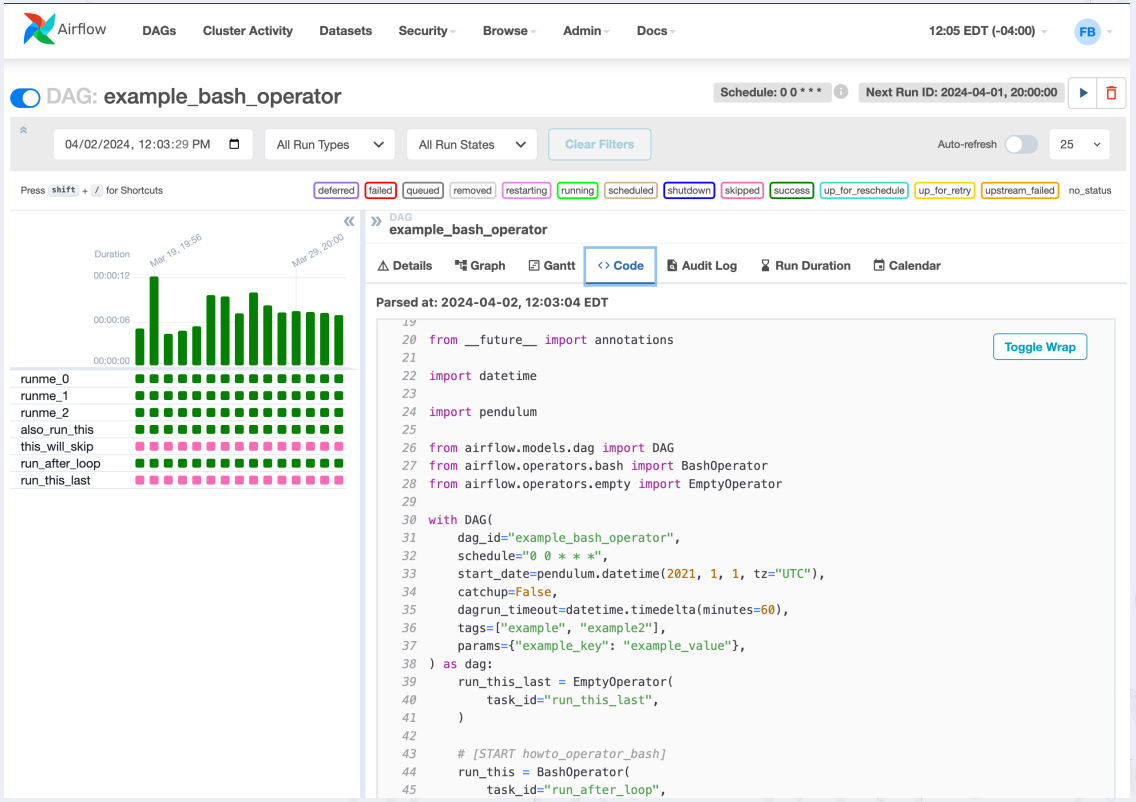
Apache Airflow



Apache Airflow

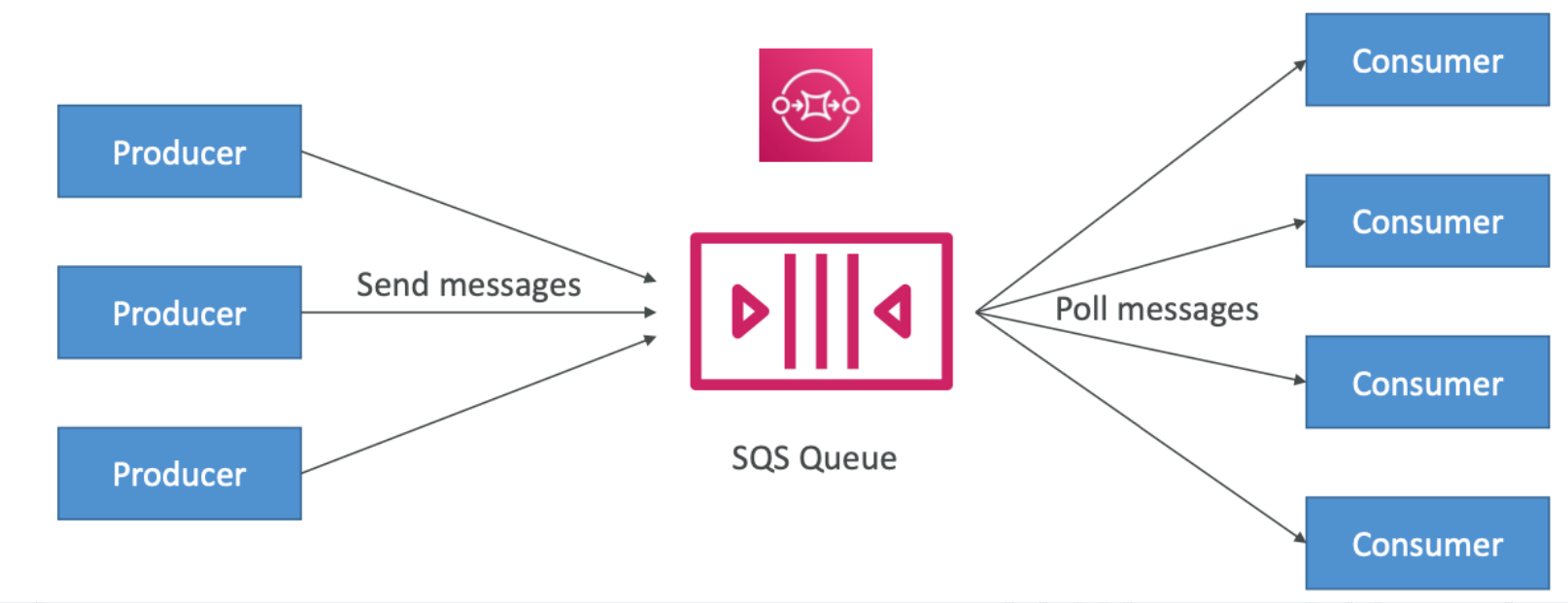


Apache Airflow



Application Integration

AWS SQS

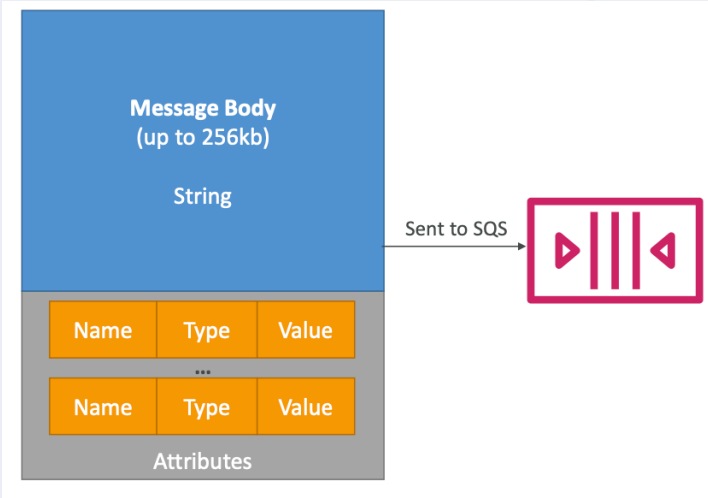


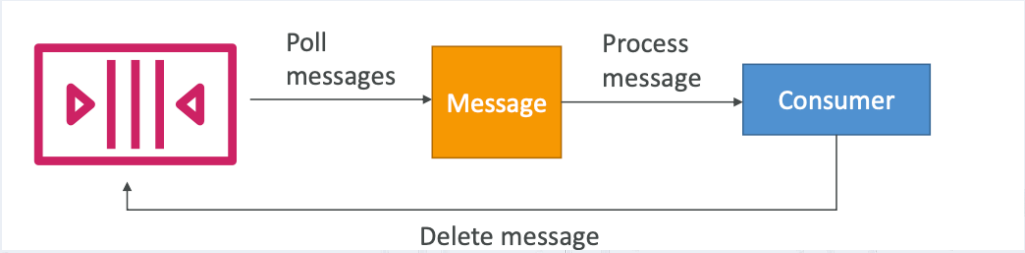
AWS SQS

- Oldest offering (over 10 years old)
- Fully managed
- Scales from 1 message per second to 15,000s per second
- Default retention of messages: 4 days, maximum of 14 days
- No limit to how many messages can be in the queue
- Low latency (<10 ms on publish and receive)
- Horizontal scaling in terms of number of consumers
- Can have duplicate messages (at least once delivery, occasionally)
- Can have out of order messages (best effort ordering)
- Limitation of 256KB per message sent

SQS Messages

- Define Body
- Add message attributes (Metadata optional)
- Provide Delay Deliver (optional)
- Get Back
  - Message Identifier
  - MD5 hash of the body





SQS Use Cases

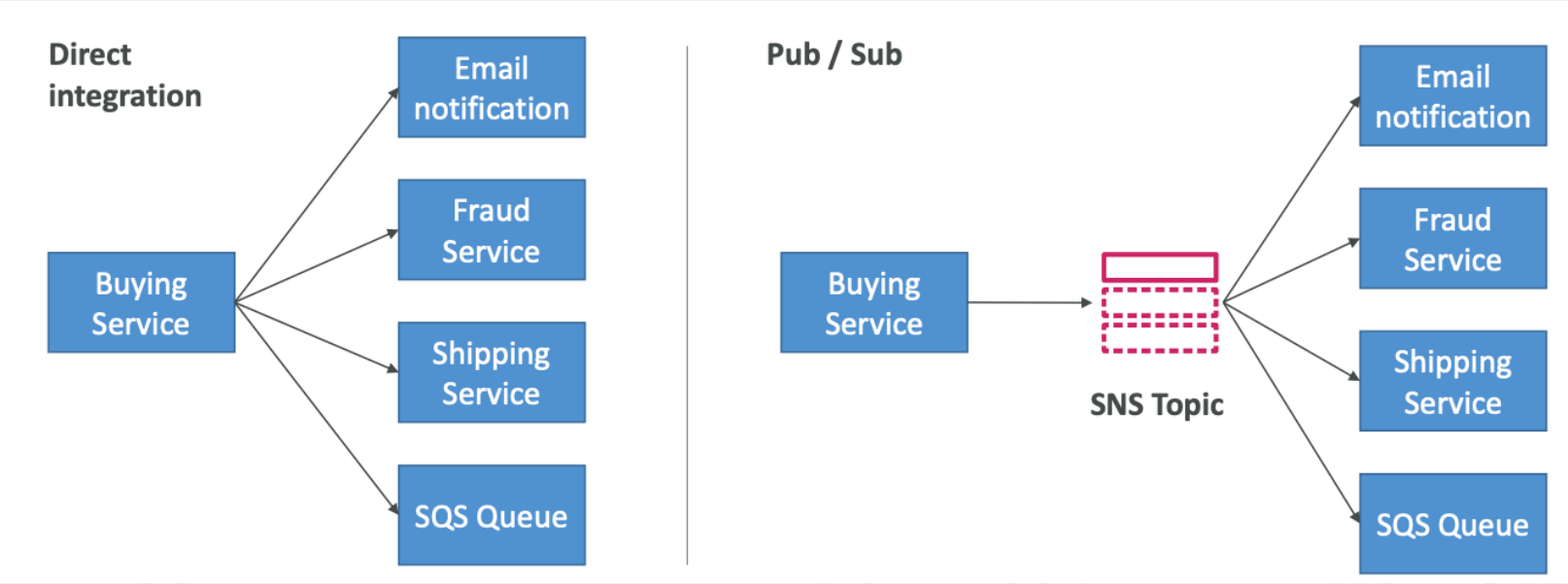
- **Decouple applications** (for example to handle payments asynchronously)
- **Buffer writes to a database** (for example a voting application)
- Handle large loads of messages coming in (for example an email sender)
- SQS can be integrated with Auto Scaling through CloudWatch!

SQS vs Kinesis Data Stream Use Cases

- **SQS use cases:**
  - Order processing
  - Image Processing
  - Auto scaling queues according to messages.
  - Buffer and Batch messages for future processing.
  - Request Offloading

- **Kinesis Data Streams use cases:**
  - Fast log and event data collection and processing
  - Real Time metrics and reports
  - Mobile data capture
  - Real Time data analytics
  - Gaming data feed
  - Complex Stream Processing
  - Data Feed from “Internet of Things”

Amazon SNS



Amazon AppFlow