VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

REPORT

BULDING STREAMING DATA PIPELINE FOR FLIGHTS DATA

SEMESTER 231 2023-2024

Advisor: Prof. Le Hong Trang

Students: Nguyen Phuoc Nguyen Phuc 2053342
Nguyen Hong Quan 2052228

HO CHI MINH CITY, December 2023

Member list & Workload

No.	Fullname	Student ID	Task
1	Nguyen Phuoc Nguyen Phuc (Leader)	2053342	- Section 1, Section 2, Section 4
2	Nguyen Hong Quan	2052228	- Section 3

Contents

List of Figures

1 Introduction

1.1 Problem Statement

Airlines and other travel stakeholders are constantly seeking ways to optimize their pricing strategies and maximize revenue. One approach is to implement a dynamic pricing model, which adjusts ticket prices based on real-time demand and supply factors. To effectively implement such a model, airlines require a reliable and up-to-date dataset of flight information, including historical prices, booking trends, and current market conditions.

Traditionally, airlines have relied on internal data sources, such as their reservation systems, to gather flight information. However, this approach is often limited in scope and may not provide a comprehensive understanding of the broader market landscape. Additionally, internal data sources may not be updated in real time, making it difficult to respond quickly to changing market conditions.

To address these limitations, airlines are increasingly turning to external data providers that offer comprehensive and up-to-date flight information. These providers aggregate data from various sources, including airline websites, travel agencies, and online booking platforms. By leveraging external data sources, airlines can gain a more holistic understanding of the market and make more informed pricing decisions.

However, the process of collecting, processing, and analyzing large volumes of external data can be challenging. Airlines need to develop a robust data pipeline that can efficiently handle the data ingestion, transformation, and storage processes. Additionally, they need to implement data quality checks and ensure that the data is reliable and consistent.

Once the data pipeline is in place, airlines can use the data to train machine learning models that predict flight demand and optimal ticket prices. These models can be integrated into the airline's reservation system to automatically adjust prices in real time. By implementing a dynamic pricing model based on real-time data, airlines can maximize revenue and improve profitability.

Figure 1: An busy airport

1.2 Applications

A Big Data pipeline for flights streaming can be used to collect, process, and analyze real-time flight data from a variety of sources, such as airline websites, flight tracking systems, and social media. This data can be used to provide a variety of benefits to airlines, airports, and passengers, including:

Real-time flight tracking: Airlines can use real-time flight data to track the progress of their flights and provide accurate arrival and departure times to passengers. This can help to reduce delays and improve the overall passenger experience.

Predictive analytics: Airlines can use machine learning algorithms to analyze real-time flight data and predict future events, such as delays, cancellations, and diversions. This information can be used to proactively alert passengers and rebook flights if necessary.

Price optimization: Airlines can use real-time flight data to optimize their pricing strategies and maximize revenue. This can be done by adjusting ticket prices based on real-time demand and supply factors.

Operational efficiency: Airports can use real-time flight data to improve their operational efficiency by optimizing resource allocation and planning for arrivals and departures. This can help to reduce congestion and improve the overall airport experience.

Customer service: Airlines and airports can use real-time flight data to provide better customer service by proactively communicating with passengers about delays, cancellations, and other disruptions. This can help to reduce passenger frustration and improve customer satisfaction.

Personalized travel recommendations: Airlines can use real-time flight data to provide personalized travel recommendations to passengers based on their past travel history and preferences.

Real-time flight disruption alerts: Passengers can be alerted to real-time flight disruptions, such as delays, cancellations, and diversions, via their mobile devices or social media feeds.

Interactive flight maps: Passengers can view interactive flight maps that show the real-time position of their flight, as well as the weather conditions and traffic at their destination.

As the volume and variety of flight data continues to grow, Big Data pipelines will become increasingly important for airlines, airports, and passengers. By adopting Big Data technologies, these organizations can gain a deeper understanding of the flight data and use it to improve their operations, provide better customer service, and develop new and innovative applications.

1.3 How the problem has been solving

The problem of collecting, processing, and analyzing large volumes of flight data has been addressed through the development and implementation of Big Data technologies. Big Data technologies provide a framework for handling the challenges associated with large-scale data management, including:

Data Ingestion: Big Data pipelines enable the efficient ingestion of data from various sources, including airline websites, flight tracking systems, and social media platforms.

Data Processing: Big Data tools like Apache Spark and Apache Hadoop facilitate the processing and transformation of large datasets, enabling the extraction of meaningful insights from raw data.

Data Storage: Distributed storage solutions like Apache HBase and Amazon S3 provide scalable and cost-effective storage for massive amounts of flight data.

Real-time Analytics: Technologies like Apache Kafka and Apache Flink enable real-time analysis of streaming flight data, providing airlines with immediate insights into flight status, demand patterns, and pricing opportunities.

Machine Learning Integration: Machine learning algorithms can be integrated into Big Data pipelines to analyze historical and real-time flight data, enabling airlines to predict flight delays, cancellations, and demand trends.

1.4 Our method

Our method will consider the method by [3]. Furthermore, we will propose one more approaches, which is more convenient for build data pipeline.

2 Methods

2.1 About flight data

As you can see in the images, the raw flight data, which are collected from online api, have JSON format, nested and many ”null” rows. Therefore we need to processing this data to transform it into suitable form. Specifically, we will flatten it out, so it can be able to transformed into CSV format, then clean null values.

(a) (b)

Figure 2: Raw flight data

2.2 Pipeline for Streaming data

In this part, we will describe our proposed stream data pipeline, begin from the raw data from api to end with the output to users. The detail is as follow:

1. **Get data from api/source** For streaming data: we will request to get data from api source continuously (example: every 5 mins). Of course, the data format is still JSON.

2. **Store all raw data in a storage - rawzone[1]** Raw data need to be stored in a storage, we must set up a data zone only for raw data. Storing raw data in a dedicated raw data storage area provides benefits in terms of integrity, flexibility for future use cases, compliance with data regulations, and efficient data processing support. Furthermore, these data need to be arranged into suitable order for further retrieval, structuring files into folders and subfolders based on categories, projects, dates or criteria. This hierarchical arrangement helps maintain order and makes it easier to locate specific files / queries.

3. **Process the data and store in another data storage - golden zone** As mentioned earlier, we need to transform our raw data. Therefore in this zone, data will be transformed into suitable form (flatten, add timestamp, combine,..). If in raw zone data is in JSON format, in Golden zone will store CSV format. Furthermore, we will apply Slowly Dimension Change (SCD) [2], which is beneficial for managing different versions of data .

4. **Create table from processed data for querying purposes** The ”active” field is created to track the version of data which has the same key We should create table from the CSV format for further steps such as querying. To optimize data structure, customize data view, performance Enhancement, simplified reporting and analytics. Tables also play a role in the space between the Golden zone and the Insight zone - which will be mentioned in the next step. Having an intermediary helps to limit

customer_ID	Information	Time arrive	active ?
123	BIIHUHUNO	10:30 a.m	0
123	OOUHIYG*YG	11 a.m	1

Figure 3: Simple example of SCD type 1. Field ”Active” is created to track the version of data. ”1” mean this is the latest version and ”0” is the old one.

errors when ETLing data. For example, if data is transferred directly from Golden to Insight and something changes in Golden, Insight will be affected.

5. **Store valuable data - Insight zone** Actually data from Golden Zone still not enough to make decisions. Because each task only requires some specific data (for example: making decision may require more fields than Visualizing in the ”FLight Schedule” at the airport,...). Therefore, Insight Zone will be the storage to store essential data for each specific task, which means that we may have many Insight Zones.

6. **Visualize data** When we have Insight Zone, next step will be visualize our data, not for making decision because this is for streaming. For example: show data on the ”Flight Schedule” at the Airport

7. **Make the flow run automatically (orchestration)** Finally, we will make our flow run automatically. Each step will trigger the next step on some conditions (schedule, finishing,..)

3 Experiments

3.1 Implementation Description

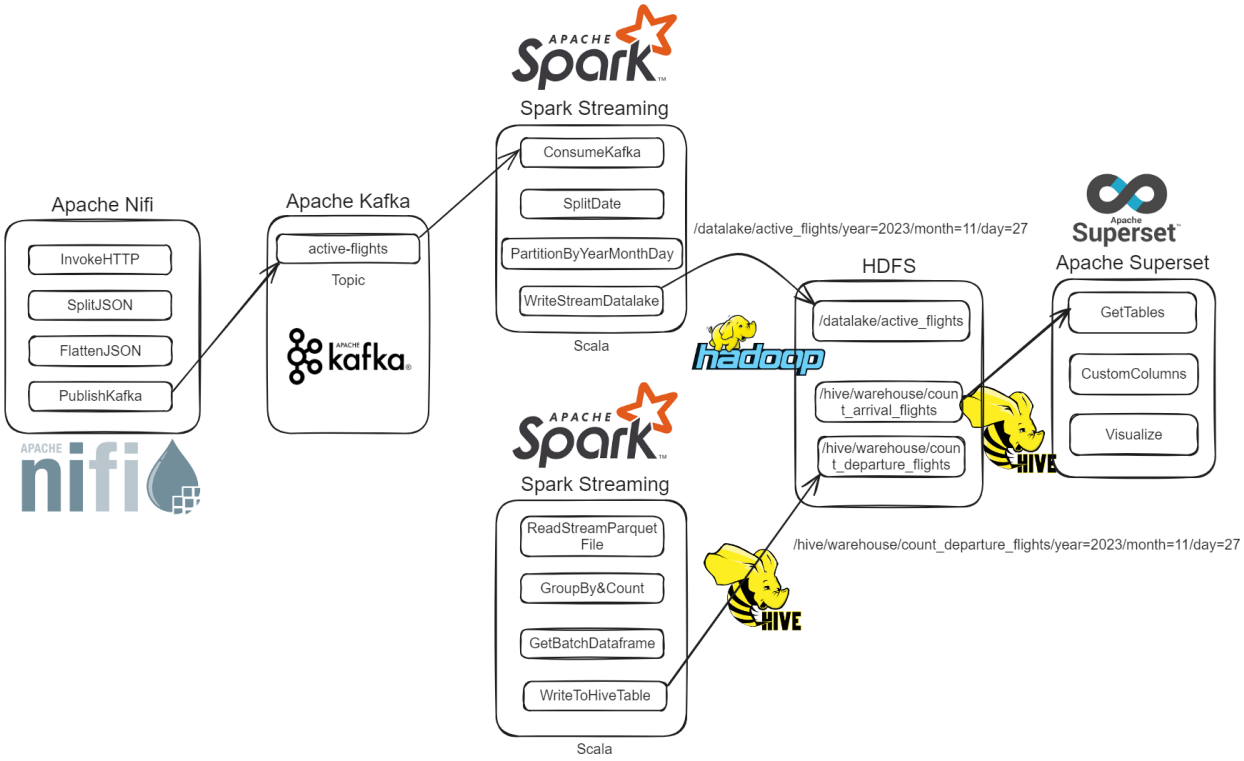

Figure 4: Data Pipeline Architecture

Our Data Pipeline Architecture involves a series of steps to move, transform, and process the data from source to its destination in real-time manner.

Data Ingestion: Apache NiFi serves as the primary tool for gathering real-time flight information from aviationstack.com by InvokeHTTP processor triggering at regular 5-minute intervals. The step continues to split the initial data into smaller JSON parts, with each JSON representing information for a single active flight. The data is then flattened to extract crucial fields. Subsequently, the processed data is published to Kafka, our chosen message queue tier for streaming purposes.

Data Transformation and Data Storage Optimization: To handle the streaming aspect, Spark streaming scripts have been developed in Scala. These scripts are responsible for consuming the data from Kafka, and then splitting the 'Date' column into three distinct columns (year, month, day) to enhance the efficiency of date-based queries and better organization of data. After transforming the data, we partition it and store as parquet files in our data lake, which we utilize HDFS as the storage solution.

Data Processing: We retrieve a streaming dataset from our data lake corresponding to the current datetime. Subsequently, we execute group-by operations on specified columns and calculate the count for the departure and arrival flights. To persist the results, we iterate through each batch of the resulting dataframes and write the information to a Hive table, which serves as our data warehouse.

Data Visualization: For the data visualization stage, Apache Superset is employed. We create a dataset connected to the external Hive table of the processed flight data, constructed on top of HDFS storage, offering an abstraction layer for the underlying storage system and providing a schema-on-read approach. Subsequently, we perform HiveSQL to customize the charts to meet the specific requirements of the visualization.

Listing 1: Data Ingestion

```
import org.apache.log4j.{Logger, Level}
import org.apache.spark.sql.{SparkSession}
import org.apache.spark.sql.types.{StructType, StructField}
import org.apache.spark.sql.types.DataTypes._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.SaveMode
import java.time.LocalDate
import org.apache.spark.sql.streaming.Trigger

object Main {
  def main(args: Array[String]): Unit = {
    val logger = Logger.getLogger("org")
    logger.setLevel(Level.ERROR)

    val HDFS_STORE = "hdfs://localhost:9000"

    val spark = SparkSession
      .builder()
      .appName("Streaming Active Flights")
      .master("local[*]")
      .config("spark.streaming.stopGracefullyOnShutdown", "true")
      .getOrCreate()

    val schema = StructType(
      Seq(
        StructField("flight_date", StringType),
        StructField("flight_status", StringType),
        StructField("dept_airport", StringType),
        StructField("dept_scheduled_time", TimestampType),
        StructField("dept_icao", StringType),
        StructField("arr_airport", StringType),
        StructField("arr_scheduled_time", TimestampType),
        StructField("arr_icao", StringType),
        StructField("airline", StringType),
        StructField("airline_icao", StringType),
        StructField("flight_number", StringType),
        StructField("flight_icao", StringType)
      )
    )

    import spark.implicits._

    val df = spark.readStream
      .format("kafka")
      .option("kafka.bootstrap.servers", "172.21.63.44:9092")
      .option("subscribe", "active-flights")
      .load()

    val raw_flights_df = df
      .select(
        from_json(col("value").cast(StringType), schema).alias("FLIGHT")
      )

    // TRANSFORMATION

    val split_date = split(col("FLIGHT.flight_date"), "-", 3)
    val flight_df = raw_flights_df
      .withColumn("year", split_date.getItem(0))
```

```
      .withColumn("month", split_date.getItem(1).cast("int"))
      .withColumn("day", split_date.getItem(2).cast("int"))

    val target_df = flight_df
      .select(
        "year",
        "month",
        "day",
        "FLIGHT.dept_airport",
        "FLIGHT.dept_scheduled_time",
        "FLIGHT.dept_icao",
        "FLIGHT.arr_airport",
        "FLIGHT.arr_scheduled_time",
        "FLIGHT.arr_icao",
        "FLIGHT.airline",
        "FLIGHT.airline_icao",
        "FLIGHT.flight_number",
        "FLIGHT.flight_icao"
      )
      .filter($"dept_airport".isNotNull)
      .filter($"dept_icao".isNotNull)
      .filter($"arr_airport".isNotNull)
      .filter($"arr_icao".isNotNull)


    val datalake_writer_query = target_df.writeStream
      .format("parquet")
      .queryName("Ingest to data lake")
      .option("mode", SaveMode.Append.toString())
      .partitionBy("year", "month", "day")
      .option("path", s"$HDFS_STORE/datalake/active_flights")
      .option("checkpointLocation", "chk-point-dir/datalake")
      .start()

    datalake_writer_query.awaitTermination()
    spark.stop()
  }
}
```

Listing 2: Data Processing

```
import org.apache.log4j.{Logger, Level}
import org.apache.spark.sql.{SparkSession}
import org.apache.spark.sql.types.{StructType, StructField}
import org.apache.spark.sql.types.DataTypes._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.SaveMode
import java.time.LocalDate
import org.apache.spark.sql.streaming.Trigger
import org.apache.spark.sql.DataFrame

object Main {
  def main(args: Array[String]): Unit = {
    val logger = Logger.getLogger("org")
    logger.setLevel(Level.ERROR)

    val HDFS_STORE = "hdfs://localhost:9000"
    val DATALAKE = "datalake/active_flights"
    val HIVE_WAREHOUSE = "/user/hive/warehouse"

    val spark = SparkSession
      .builder()
      .appName("Streaming processed active flights")
      .master("local[*]")
      .config("hive.metastore.uris", "thrift://localhost:9083")
      .config("hive.exec.dynamic.partition", "true")
      .config("hive.exec.dynamic.partition.mode", "nonstrict")
      .enableHiveSupport()
      .config("spark.streaming.stopGracefullyOnShutdown", "true")
      .getOrCreate()

    import spark.implicits._

    val schema = StructType(
      Seq(
        StructField("dept_airport", StringType),
        StructField("dept_scheduled_time", TimestampType),
        StructField("dept_icao", StringType),
        StructField("arr_airport", StringType),
        StructField("arr_scheduled_time", TimestampType),
        StructField("arr_icao", StringType),
        StructField("airline", StringType),
        StructField("airline_icao", StringType),
        StructField("flight_number", StringType),
        StructField("flight_icao", StringType)
      )
    )
    val currentDate = LocalDate.now()
    val currentYear = currentDate.getYear
    val currentMonth = currentDate.getMonthValue
    val currentDay = currentDate.getDayOfMonth

    val df = spark.readStream
      .schema(schema)
      .format("parquet")
      .option("path",
        s"$HDFS_STORE/$DATALAKE/year=$currentYear/month=$currentMonth/day=$currentDay")
      .load()

    val countDeptFlightsDF = df
      .groupBy("dept_icao", "dept_airport")
```

```
      .count()
      .orderBy($"count".desc)
      .withColumn(
        "total_departure_flights",
        col("count").as("total_departure_flights")
      )
      .withColumn("year", lit(currentYear))
      .withColumn("month", lit(currentMonth))
      .withColumn("day", lit(currentDay))
      .drop(col("count"))
      .select("*")

    val countArrFlightsDF = df
      .groupBy("arr_icao", "arr_airport")
      .count()
      .orderBy($"count".desc)
      .withColumn(
        "total_arrival_flights",
        col("count").as("total_arrival_flights")
      )
      .withColumn("year", lit(currentYear))
      .withColumn("month", lit(currentMonth))
      .withColumn("day", lit(currentDay))
      .drop(col("count"))
      .select("*")

    val streamingQueryDept = countDeptFlightsDF.writeStream
      .outputMode("complete")
      .queryName("Write countDept to Hive table")
      .foreachBatch { (batchDF: DataFrame, batchId: Long) =>
        batchDF.write
          .format("hive")
          .partitionBy(
            "year",
            "month",
            "day"
          )
          .option("checkpointLocation", "chk-point-dir/departure")
          .mode("overwrite")
          .saveAsTable("count_departure_flights")
      }
      .start()

    val streamingQueryArr = countArrFlightsDF.writeStream
      .outputMode("complete")
      .queryName("Write countArr to Hive table")
      .foreachBatch { (batchDF: DataFrame, batchId: Long) =>
        batchDF.write
          .format("hive")
          .partitionBy(
            "year",
            "month",
            "day"
          )
          .option("checkpointLocation", "chk-point-dir/arrival")
          .mode("overwrite")
          .saveAsTable("count_arrival_flights")
      }
      .start()

    spark.streams.awaitAnyTermination()
    spark.stop()
```

```
122     }
123 }
```

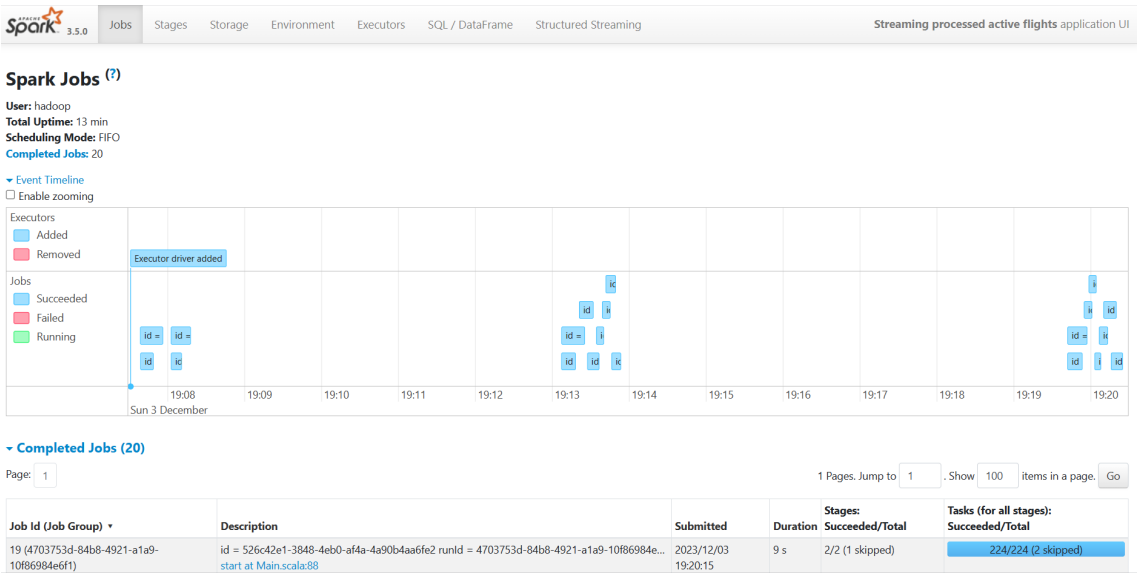

Figure 5: Event Timeline of Spark Jobs for Data Ingestion

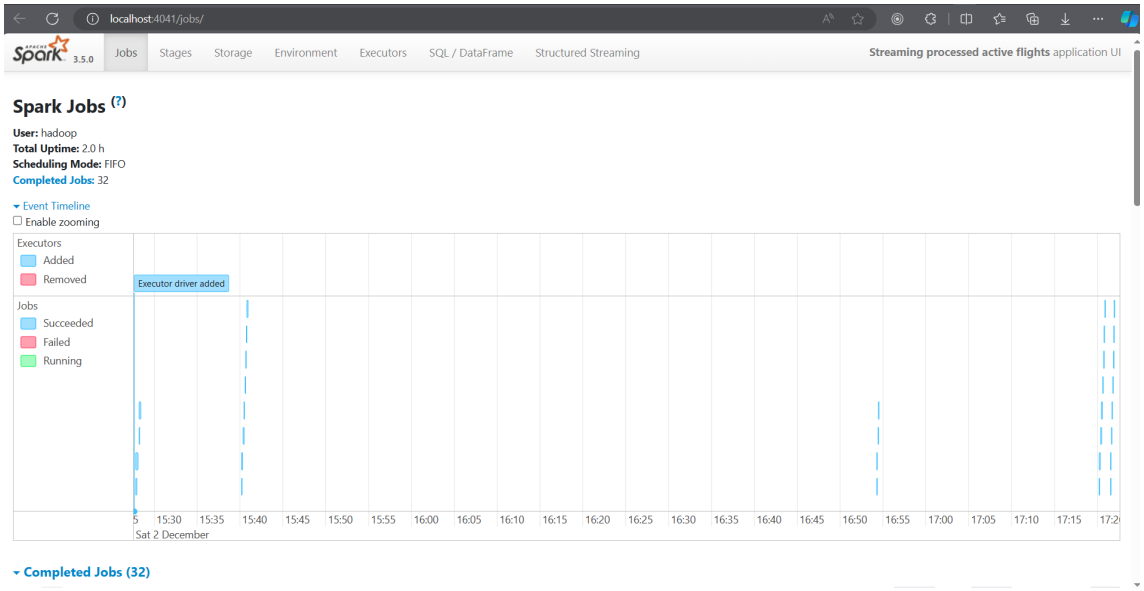

Figure 6: Event Timeline of Spark Jobs for Data Processing

3.2 Visualization

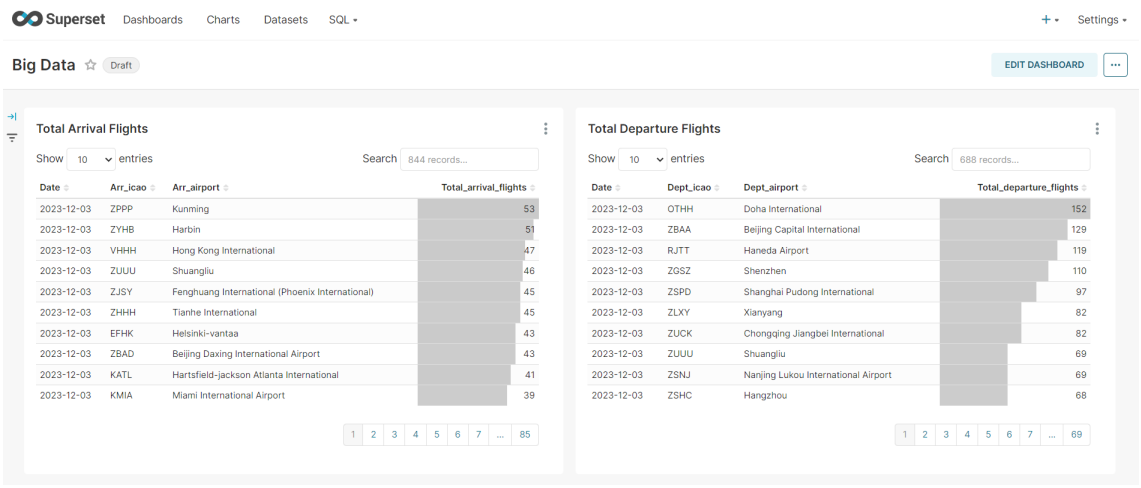

Figure 7: Apache Superset Visualization Dashboard

4 Application Propose

In this section, we will propose the way to build data pipeline using serverless application from Amazon Web Server (AWS).

4.1 What is Amazon Web Server (AWS)

AWS is a comprehensive suite of cloud computing services offered by Amazon.com. It provides a broad range of solutions, from basic compute power and storage to sophisticated artificial intelligence and machine learning tools. Think of it as a virtual data center in the sky, accessible from anywhere in the world with an internet connection.

What does AWS offer?

Figure 8: AWS Services.

The breadth of AWS offerings is staggering. Here are just a few of the key services:

- Compute: Access a variety of virtual servers (EC2 instances) with different configurations to run your applications.
- Storage: Store your data securely and reliably with options like S3 object storage and EBS block storage.
- Databases: Choose from a range of managed database services like Amazon Aurora for relational databases and DynamoDB for NoSQL databases.
- Networking: Connect your AWS resources to the internet and build secure, scalable networks.
- Analytics: Analyze your data with powerful tools like Amazon Redshift and Amazon QuickSight.
- Machine Learning: Leverage the power of artificial intelligence with services like Amazon SageMaker and Amazon Rekognition.
- Management Tools: Manage your AWS resources efficiently with tools like AWS CloudFormation and AWS CloudTrail.

Who uses AWS?

AWS caters to a diverse range of users, from individual developers and startups to large enterprises and government agencies. Some of the popular use cases include:

- Building and deploying web applications: AWS provides the infrastructure and tools to launch and scale your web apps quickly and easily.
- Developing mobile apps: AWS offers services like Amazon Cognito for user authentication and Amazon DynamoDB for NoSQL data storage, ideal for mobile app development.
- Big data analytics: AWS has powerful tools for analyzing large datasets and gaining insights from your data.

- Disaster recovery: Back up your data and applications to AWS for secure and reliable disaster recovery.

Benefits of using AWS

Choosing AWS comes with a multitude of benefits:

- Scalability: Easily scale your resources up or down based on your needs, paying only for what you use.
- Flexibility: Choose from a wide range of services to build the perfect solution for your specific needs.
- Reliability: AWS offers a highly reliable and secure infrastructure with global reach.
- Cost-efficiency: Pay only for the resources you use, eliminating the need for upfront investments in hardware and software.
- Innovation: AWS constantly innovates, introducing new services and features to help you stay ahead of the curve.

Getting started with AWS

AWS offers a free tier for new users, allowing you to experiment with its services without any cost. With its intuitive interface and extensive documentation, getting started with AWS is easier than ever.

Conclusion

AWS is not just a cloud computing platform; it's a powerful tool that can transform the way you do business. By leveraging its vast resources and innovative services, you can build and scale your applications, analyze your data, and gain valuable insights, all while optimizing your costs and staying ahead of the competition.

4.2 Apply AWS to build Streaming Data Pipeline

The detail architecture for building data pipeline in AWS is as follow:

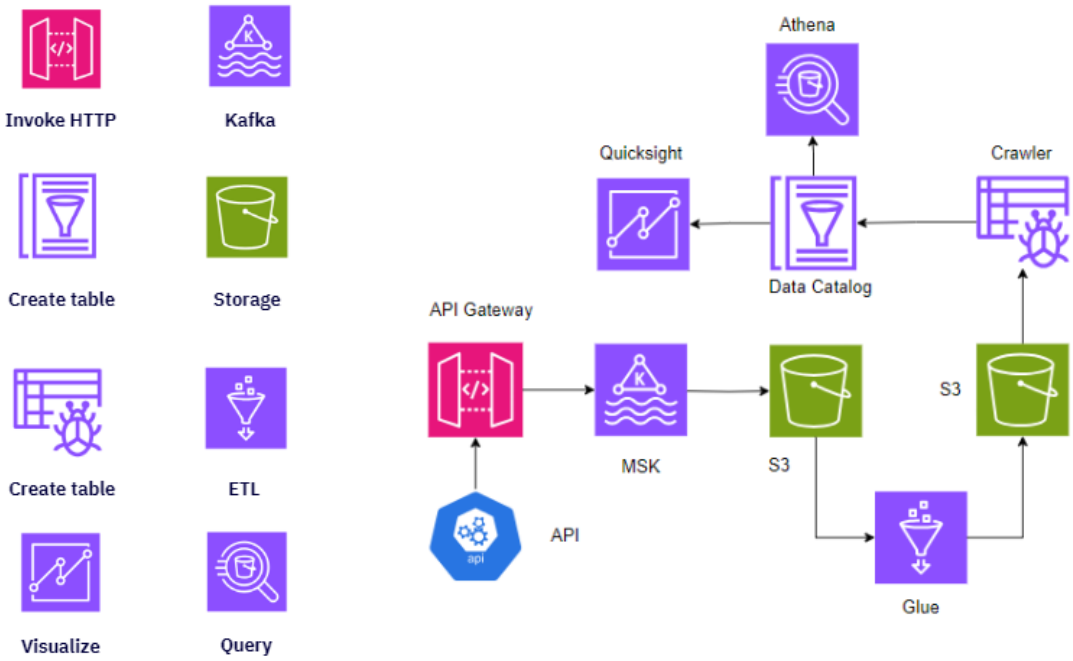

Figure 9: Architecture and Service use

1. **API Gateway** This service is used to Invoke HTTP and request data from API
2. **Manage Streaming for Apache Kafka** This service is simply like Apache Kafka.

3. **S3** This data storage provided by AWS. This storage can perform folder - file and many format of data, which is suitable for Data Zones as defined in section 2
4. **Glue** Glue handles Extract - Transform - Load (ETL) Task. As you can see in figure 4, it helps move data from zone to zone, including any Transformation. if needed.
5. **Crawler and Data Catalog** These 2 services are actually belong to Glue. Crawler will crawl data from storage and save them as table in Data Catalog
6. **Athena** This is a query tool provided by AWS
7. **Quicksight** Quicksight provide powerful ability for Data Visualization.

Every service that we mentioned above is low-code setting. Which mean that you don't have to code to define them, they already define them and build a friendly UI for the users to custom them. For further understanding, we recommend you explore AWS yourself, Follow the link here. Make sure to read the documentation part before using.

References

[1] Cody Slingerland, ”Data Lake Architecture: An Intro Guide To Using Data Lakes”, https://www.cloudzero.com/blog/data-lake-architecture/, June 07, 2021.

[2] Madison Schott, Analytics Engineer and Blogger, ”Slowly Changing Dimensions: What they are and why they matter”, Published May 26, 2023,https://www.thoughtspot.com/data-trends/data-modeling/slowly-changing-dimensions-in-data-warehouse.

[3] XSiddhSaraf, ”Covid-19-Data-Pipeline-Based-On-Messaging-and-Analysis”, https://github.com/XSiddhSaraf/Covid-19-Data-Pipeline-Based-On-Messaging-and-Analysis