Cloud-Based Flight Delay Analysis Pipeline Design Report

By Floris Vermeulen, Niek Fleerakkers, Patryk Grodek, Samir Sabitli and Martijn van Iterson

Abstract

Modern aviation operations generate massive datasets on flight performance and delays. This report presents the design and implementation of a cloud-based data pipeline for analyzing air traffic delay patterns in a business intelligence context. The pipeline integrates batch and stream processing on Google Cloud Platform, employing Google Cloud Storage for data lake storage, Google BigQuery for analytics warehousing, and Apache Spark on Cloud Dataproc for distributed processing. A Medallion architecture with Bronze, Silver, and Gold layers is used to progressively refine raw flight data into clean, enriched data and ultimately into aggregated delay metrics ready for analysis. Batch processing handles large historical datasets containing millions of flight records for comprehensive daily statistics, while a streaming component ingests real-time flight data to provide near real-time delay insights. We discuss architectural decisions, including the use of both batch and streaming processing analogous to a Lambda pattern to balance data freshness and accuracy, as well as the rationale for the Medallion data lake design to ensure data quality and governance. The report concludes with observed outcomes, scalability considerations, and future enhancements for the pipeline.

Introduction

Air traffic delay analysis is critical for airlines, airports, and regulators to improve scheduling, resource allocation, and passenger satisfaction. The volume of flight data is enormous. For example, the U.S. Department of Transportation recorded over 9.5 million domestic flights in 2015 alone, where this dataset contains 5.8 million of them (*2015 U.S.-Based Airline Traffic Data*, 2025). Each flight record includes numerous attributes such as times, locations, and delay durations, posing a big data challenge for timely and insightful analysis. Traditional offline analysis techniques that rely solely on batch processing struggle to keep up with the dynamic nature of aviation operations (Kamdar, et al. 2025). In scenarios like rapidly developing weather delays or air traffic congestion, stakeholders require real-time insights to make prompt decisions (Kamdar, et al. 2025). Simultaneously, comprehensive historical analysis is needed for long-term planning and performance benchmarking.

This project addresses these needs by designing a cloud-based data pipeline that combines batch and streaming data processing for flight delay analysis. The pipeline ingests raw flight data and incrementally refines it following the Medallion architecture paradigm with Bronze, Silver, and Gold layers to ensure data quality and usability (Idera, 2025). Batch processing is used for full historical datasets, producing high-quality daily delay statistics. In parallel, a streaming component handles continuous ingestion of new flight data to produce live metrics on delays within minutes of events. By leveraging Google Cloud's managed services including Cloud Storage, Dataproc, and BigQuery, the solution can scale to large data volumes and seamlessly integrate with BI tools. The pipeline thus aims to provide both accurate historical analytics and near real-time monitoring of flight delays, empowering decision-makers with a holistic BI platform for air traffic performance.

Related Work

Early big-data architectures like the Lambda architecture have promoted combining batch and real-time processing to achieve both accuracy and low-latency updates (Demirezen & Navruz, 2023). In a Lambda setup, the batch layer processes all data to produce a comprehensive view, while a speed layer processes streaming data for immediate insights with results merged in a serving layer (Demirezen & Navruz, 2023). Prior research has validated that such an approach can yield eventual consistency where real-time insights are refined by later batch computations, providing precise and reliable results (Demirezen & Navruz, 2023). Demirezen and Navruz (2023) demonstrated this pattern in an aviation context, showing that a Lambda-based system for air traffic surveillance could handle streaming ADS-B sensor data and historical data simultaneously, ultimately converging to 100% accuracy in delay metrics once batch corrections are applied. This motivates our use of a combined batch and stream design for flight delays to capture immediate delay information while ensuring correctness with post hoc batch reconciliation.

Another relevant paradigm is the modern data lakehouse architecture, often implemented via the Medallion multi-hop design. The Medallion architecture organizes data processing into three layers: Bronze for raw data, Silver for cleansed and conformed data, and Gold for aggregated analytics-ready data. This layered approach has been widely adopted to incrementally improve data quality and structure as data flows through a pipeline (Idera, 2025). The Bronze layer ingests data as-is from sources, providing a landing zone and historical archive. The Silver layer applies transformations such as cleaning, deduplication, and integration to produce a refined dataset suitable for analysis (Idera, 2025). Finally, the Gold layer contains high-level aggregates or feature sets tailored for business intelligence queries and reports. By separating concerns in this way, Medallion architecture enhances data governance and reliability, as each layer has a clear purpose and data quality issues can be addressed in stages (Idera, 2025). Our pipeline builds on this concept, using Google Cloud Storage as a data lake to store Bronze and Silver datasets and BigQuery as an analytics warehouse for Gold tables.

Real-world applications of streaming and batch integration for flight data are emerging. Kamdar et al. (2025) proposed a real-time flight data pipeline using Apache Kafka and Spark Streaming for ingesting live flight events, coupled with historical analysis in Power BI for context. Their system highlights the benefit of streaming

analytics in handling the dynamic and voluminous nature of flight operations, which batch-only systems cannot readily address (Kamdar, et al. 2025). Our work aligns with these trends but leverages managed cloud services to simplify deployment and avoid the operational overhead seen in self-managed Hadoop and Spark setups, while still following proven architectural principles from prior research.

System Architecture and Design

The system's architecture blends a batch processing pipeline with a stream processing pipeline, unified under a data lakehouse design with Bronze, Silver, and Gold layers.

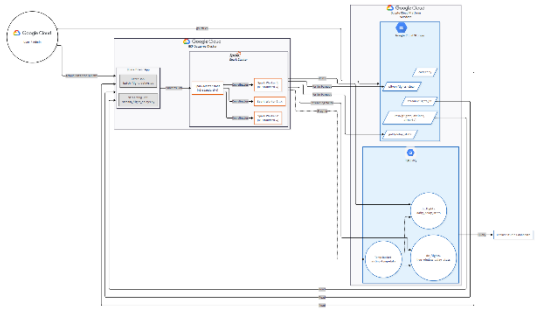


Figure 1: Main system architecture overview.

At a high level, raw flight data and reference data are first ingested into the Bronze layer in cloud storage. The batch pipeline periodically processes the entire accumulated Bronze dataset, producing a cleansed Silver dataset and aggregated Gold outputs. In parallel, the streaming pipeline continuously ingests new flight records as they arrive, performing on-the-fly cleaning and computing windowed delay statistics as interim Gold data for real-time analysis. Both pipelines deposit results into centralized storage and BigQuery for BI consumption.

Batch vs. Stream Use Cases

The choice to implement both batch and streaming pathways is driven by distinct use cases. Batch processing is suited for comprehensive analysis of historical trends and patterns. For example, airline performance metrics such as average delays and cancellation rates are typically computed on complete daily or monthly data, which requires processing the full dataset to ensure accuracy. Batch jobs can leverage the complete data history to produce high-precision aggregates such as true daily averages once all flights for the day are recorded. However, batch results come with high latency, often hours or days after events (Demirezen & Navruz, 2023). In contrast, stream processing addresses the need for low-latency, near real-time insights. For instance, an airport operations center may want to know the average departure delay in the last 15 minutes to adjust gate assignments proactively. A streaming job can compute such metrics on the fly as new flight data is ingested, with latency on the order of minutes. The trade-off is that streaming results might be based on incomplete data or approximate aggregations depending on windowing (Demirezen & Navruz, 2023). By deploying both modes, our architecture adheres to a Lambda-like strategy where the streaming pipeline as the speed layer provides immediate situational awareness, while the batch pipeline as the batch layer later produces an authoritative, error-corrected dataset (Demirezen & Navruz, 2023). BigQuery serves as the serving layer where both real-time and batch outputs can be accessed for visualization or querying.

Medallion Architecture Rationale

The Medallion data layout structures storage and processing, bringing multiple benefits. First, it enables data quality management in stages. The Bronze layer in our design is simply the raw input files on Google Cloud Storage, preserving all original information and allowing reprocessing or audit when needed. The Silver layer holds the cleaned and conformed data, meaning downstream processes always have a reliable, consistent dataset to read from, while upstream raw data remains available if re-cleaning or new transformations are required (Idera, 2025). Second, the layered approach improves pipeline agility and governance. Each transformation step from raw to clean and clean to aggregated is isolated, making it easier to trace lineage and validate outputs at each step. For example, one can verify that Silver data has no null airport codes or that Gold aggregates match the underlying Silver records. Finally, storing Gold-level data such as aggregated delay statistics in a warehouse like BigQuery allows fast BI queries on simplified schemas, as opposed to running heavy aggregations on raw data repeatedly. This aligns with best practices of separating compute-intensive ETL from lightweight query workloads in analytics systems (Idera, 2025). By implementing the Medallion architecture in our design, we ensure that the pipeline is scalable, maintainable, and produces trusted data for business intelligence consumption.

Data Pipeline Implementation

The data pipeline consists of two Spark applications corresponding to the batch and streaming components. All code is written in Python using PySpark and executed on a Google Cloud Dataproc cluster.

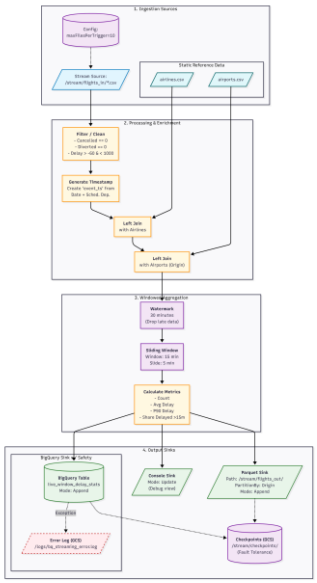


Figure 2: Streaming dataflow detail.

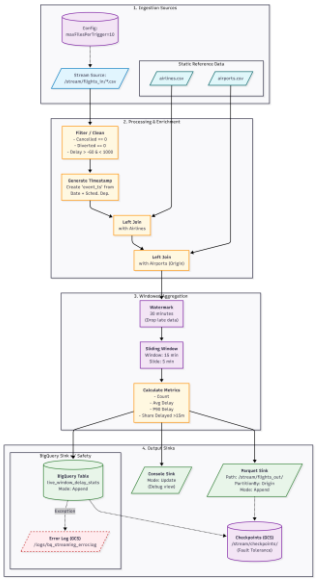


Figure 3: Batch dataflow detail.

Batch Processing Job

The batch job in batch_flight_delays.py is responsible for processing the entire archive of flight data available to date. It begins by reading raw data from the Bronze layer, including one or more CSV files containing flight records located in gs://<BUCKET>/raw/flights/ on Google Cloud Storage and two reference CSV files for airlines and airports located in raw/airlines/ and raw/airports/ respectively. The flight dataset contains records of individual flights with fields such as flight date including year, month, and day, airline code, origin and destination airport codes, scheduled and actual departure and arrival times, delay durations, and flags for cancellations or diversions. To ensure schema consistency, the job defines an explicit schema for the flight data and applies it when reading the CSV files, preventing issues with type inference and making downstream transformations reliable.

After loading the raw flights DataFrame, the batch job performs data cleaning and normalization steps to create the Silver layer dataset. Key cleaning operations include filtering out flights that were canceled or diverted since their delay times are not applicable, dropping records with null critical fields such as airline code, origin or destination airport, or departure delay, removing obviously erroneous delay values such as excluding departure delays less than negative 60 minutes or over 1000 minutes which likely indicate data errors, and eliminating duplicate records by dropping duplicates based on a composite key of flight date, flight number, tail number, origin airport, and scheduled departure time. These criteria ensure that only valid, unique flight records proceed to analysis.

Next, the batch job enriches the flight records by joining in the reference data. Using the airline reference, each flight's airline code is replaced with the full airline name for readability. Using the airport reference, the origin and destination airport codes are augmented with the airport's full name, city, and state. Before the joins, the code normalizes codes to uppercase and trims whitespace in both flight data and reference data to avoid mismatches. The result of these joins is a rich flights dataset where key fields are human-readable and dimensionally consistent, allowing one to group by origin city or filter by airline name directly. After enrichment, an additional transformation derives a proper date column from the year, month, and day fields and computes a departure hour from the scheduled departure time, which can be useful for time-of-day analysis.

The cleaned and enriched DataFrame is then written to storage as the Silver layer output. The implementation uses Parquet format on Google Cloud Storage in gs://<BUCKET>/silver/flights_clean/, partitioned by the flight date. Partitioning by date improves query performance for time-bound queries and organizes the data by day in the lake. The job uses a controlled number of partitions when writing, such as repartitioning to 4 partitions by date, to avoid generating too many small files. The Parquet files in Silver serve as the single source of truth for cleaned flight records and can be used for various analytical needs beyond the scope of this specific pipeline, such as training a delay prediction model on cleaned data.

After producing the Silver layer, the batch job proceeds to compute the Gold layer aggregate of daily delay statistics. It groups the enriched flight data by multiple dimensions of interest including date, airline, origin, and destination with their human-readable names and states. Within each group, effectively grouping by a specific route from origin to destination on a given day for a given airline, the job calculates several metrics including the number of flights, the average departure delay, the 90th percentile of departure delay using an approximate percentile function, the count of flights that were significantly delayed such as more than 15 minutes late departure, and the share of such delayed flights as a percentage. These metrics provide a concise summary of delays for each airline-route-day combination, which are highly useful for BI reporting such as identifying the worst delay routes or comparing airlines' on-time performance.

The Gold aggregated DataFrame is written to two sinks. First, it is saved as Parquet files in gs://<BUCKET>/gold/delay_stats/, partitioned by date, allowing exploratory analysis or re-use by other processes if needed. Second, the job writes the aggregation to BigQuery, inserting or overwriting a table in the format {PROJECT}.{DATASET}.daily_delay_stats that mirrors the schema of the aggregate with fields for date, airline, route, and the computed metrics. The BigQuery write is handled via the Spark BigQuery connector, configured with an intermediate temporary Google Cloud Storage bucket for staging data during the load. Once in BigQuery, the daily_delay_stats table can be directly queried with SQL or connected to BI visualization tools. The batch job is designed to be idempotent and re-runnable, with each run overwriting the Silver and Gold outputs for the full dataset, which is acceptable because the input is the entire set of raw data. This design ensures the ability to do a full rebuild of the cleansed and aggregated data from

scratch, aligning with the reproducibility goals of the Medallion approach where one can recreate Gold from Bronze at any time (Databricks, 2025).

Streaming Processing Job

The streaming job in stream_flight_delays.py is designed to handle continuous, incremental data ingestion for real-time insights. It uses Spark Structured Streaming, a micro-batch stream processing model, to read new flight data files arriving in a designated Google Cloud Storage folder as the streaming Bronze input. In our implementation, we simulate streaming by dropping CSV files into gs://<BUCKET>/stream/flights_in/. Each file contains a batch of recent flights, such as all flights that departed in a certain hour. The Spark streaming source is configured to read these CSV files as they appear, treating the folder as an evolving data stream. We set maxFilesPerTrigger to 10 to throttle ingestion if a large number of files arrive at once, ensuring the stream ingestion processes at most 10 new files per micro-batch and preventing resource overload in case of bursts of incoming data.

The streaming job performs a similar cleaning and enrichment procedure as the batch job, but with adjustments for streaming context. Incoming flight records are filtered to exclude canceled or diverted flights immediately. Null checks and delay range filters are applied to drop bad data in the stream, mirroring criteria used in batch cleaning. The stream then enriches flights by joining with the same static reference data for airlines and airports. Spark's structured streaming allows joining a streaming DataFrame with static DataFrames for airlines and airports efficiently, since the static side can be broadcast or otherwise optimized. In the code, we join to add airline name as well as origin airport's name, city, and state.

A crucial step for streaming analysis is generating event time timestamps and windowing. The streaming job creates a timestamp column representing the scheduled departure time of the flight, converted to a proper TimestampType. This timestamp uses the flight's year, month, day, and scheduled departure in hhmm format to form a datetime, which serves as the event time of the record. We then define a windowed aggregation on the streaming data using a tumbling window of 15 minutes with a slide interval of 5 minutes. This effectively groups flights departing in each 15-minute interval with updates every 5 minutes, allowing overlapping windows by their origin location. The grouped fields include the window as a time range and origin airport with code, name, city, and state. Within each window group, similar metrics are computed as in the batch, including number of flights, average delay, 90th percentile delay, count of flights with greater than 15 minutes delay, and the ratio of those delayed flights. The result is a continuously updating set of statistics such as from 10:00 to 10:15 at JFK airport with 20 flights departed, average delay 5 minutes, and 10% of flights delayed over 15 minutes, which updates every 5 minutes with new data. We use a watermark of 30 minutes on the event timestamp to allow late arriving data within a reasonable bound while avoiding indefinite growth of state. The windowed results are further flattened by extracting the window start and end into separate columns and dropping the complex window object.

The streaming output as Gold-equivalent for real-time is written to multiple sinks simultaneously. For debugging and demonstration, one sink is the console with update mode, printing the latest window aggregates whenever they update. Two persistent sinks are defined, including a Parquet sink writing to gs://<BUCKET>/stream/flights_out/ and a BigQuery sink writing to a table in the format {PROJECT}.{DATASET}.live_window_delay_stats. The Parquet sink appends each window's results as they are finalized, partitioning by origin airport code to organize the output files. A checkpoint directory in Google Cloud Storage is used for this sink to store streaming state and ensure fault-tolerance with exactly-once processing guarantees. The BigQuery sink is handled through a custom function that uses the Spark BigQuery connector in streaming mode, supporting writing streaming DataFrames to BigQuery in append mode with micro-batches. With these, the streaming job continuously publishes the latest delay statistics to the live BigQuery table, which can be queried by BI dashboards with only a few minutes of lag behind real-world events.

The streaming pipeline does not persist a Silver layer dataset separate from its Gold outputs but performs filtering and enrichment in-memory and directly outputs the aggregated insights. The assumption is that the batch pipeline will eventually handle permanent storage of all data, whereas the streaming pipeline focuses on short-term analytics. The two pipelines thus complement each other, with the streaming job providing a real-time feed of key metrics with continuous updates and the batch job periodically reconciling and backfilling the definitive historical metrics. By using Spark for both, we ensure the logic including filters, joins, and calculations is largely consistent between streaming and batch, reducing development effort and potential discrepancies in line with recommendations to unify batch and stream processing with the same framework to avoid duplicate code paths (Demirezen & Navruz, 2023).

Storage and Data Lake Design

The pipeline's storage design follows the Medallion architecture, separating data into Bronze, Silver, and Gold layers, each stored in appropriate formats and locations. All raw input files are stored in Google Cloud Storage buckets in their original CSV format as the Bronze layer. This includes the flights data which may be a single large CSV or many CSVs split by date and reference data like airlines and airports CSVs. The raw data is kept unchanged to serve as a system of record. For the streaming sub-pipeline, the incoming files in stream/flights_in/ also act as a Bronze layer for real-time data. Storing raw data in Google Cloud Storage provides a durable, cost-effective landing zone that can hold structured or unstructured data indefinitely (Hartwig, 2024). Google Cloud Storage is well-suited as a data lake,

offering virtually unlimited scalability and high durability for these files (Hartwig, 2024).

The Silver layer is the curated flight dataset produced by the batch process, stored as partitioned Parquet files in silver/flights_clean/ on Google Cloud Storage. Parquet is a columnar storage format that is efficient for big data analytics, as queries reading only a subset of columns, which is common in BI scenarios, will scan much less data compared to raw CSV. Partitioning by date means analysts can easily retrieve a specific date's data without reading the entire dataset. The Silver data has all the quality improvements including no invalid records, enriched fields, and standardized formats such as all airport codes uppercase. This layer provides an Enterprise-wide consistent view of the flights dataset (Hartwig, 2024). Data in Silver is ready for use in ad-hoc analysis, machine learning, or feeding into data warehouses. We also log any anomalies during enrichment such as if some airport codes could not be resolved to a name to a logs/ folder in Google Cloud Storage for transparency and debugging, which aligns with good data lake practices of keeping audit logs.

The Gold layer consists of high-level tables suited for BI consumption. In our implementation, there are two main Gold outputs including the daily delay statistics from batch and the live windowed delay statistics from streaming. The daily delay stats are stored in two forms, as Parquet files in Google Cloud Storage in gold/delay_stats/ partitioned by date and as a table in BigQuery. The reason for dual storage is to satisfy different access patterns, with BigQuery allowing immediate SQL querying and integration with BI tools such as connecting Google Data Studio, Looker Studio, or Tableau to the BigQuery table for visualization, while Parquet on Google Cloud Storage can be handy for data scientists who might want to load the data into a Spark DataFrame or pandas for custom analysis outside of BigQuery. The live window stats from the streaming job are similarly written to BigQuery as table live_window_delay_stats and to Google Cloud Storage in the stream/flights_out/ folder as Parquet. The Gold tables are denormalized and optimized for read performance, with the daily stats table including the airport names and airline names to avoid having to join reference tables when a BI query is made. By design, Gold layer data is what powers dashboards and reports and is the single source of truth for business metrics in this domain (Idera, 2025). We chose BigQuery as the primary serving layer for Gold because it is a serverless, highly scalable data warehouse capable of handling complex queries on large data efficiently, even petabytes of data (Google Cloud, 2025). BigQuery's ability to process analytics in near real time over large datasets, combined with its integration into the Google Cloud ecosystem, makes it ideal for providing interactive BI experiences on the output of our pipeline.

By adhering to the Bronze, Silver, and Gold structure, our data lake design also simplifies permission management and data governance. For example, we can grant broad read access to analysts on the Gold tables since those expose only business-level information, more restricted access to Silver for data engineers or power users who need record-level detail, and tightly control Bronze access since raw data might contain sensitive fields or simply be too unwieldy for general use. Each layer thus becomes a trust level of data, with Bronze as raw with low trust that needs processing, Silver as refined with medium trust, and Gold as finalized metrics with high trust for decision-making. This design is consistent with known best practices in modern cloud data architectures (Idera, 2025).

Technologies Used

The implementation leverages several cloud and big-data technologies, chosen for their compatibility and strengths in handling large-scale data processing. All persistent storage of files including raw and processed data uses Google Cloud Storage, a fully-managed object storage service. Google Cloud Storage offers high durability with eleven 9's of durability for data and scalability for virtually unlimited data size, making it an excellent choice for a data lake (Hartwig, 2024). By storing data in buckets, we decouple compute from storage, meaning our Spark jobs can be transient and data remains in Google Cloud Storage for future use, aligning with cloud best practices. Google Cloud Storage is used in our pipeline to store Bronze CSVs, Silver and Gold Parquet files, as well as logs and checkpoints. Its integration with Spark via the Hadoop FileSystem interface allows Spark to read and write data to Google Cloud Storage paths as if they were local files, which greatly simplifies development.

Dataproc is Google Cloud's managed service for running Apache Spark and Hadoop clusters. We utilized a Dataproc cluster to execute the PySpark batch and streaming jobs. Using Dataproc eliminates the need to manually manage cluster setup, configuration, or scaling, as the service provisions the cluster with the desired number of nodes and pre-configured Spark, and we simply submit our jobs to it. Dataproc automatically handles the Hadoop and Spark ecosystem components and can scale or shut down clusters on demand. It also offers integration points like connectors to BigQuery. As a managed Spark service, Dataproc eliminates the heavy lifting of manual cluster provisioning, configuration and monitoring and auto-scales and updates clusters while integrating with other GCP products (CloudChipr, 2025). This allowed us to focus on writing data processing logic rather than on infrastructure. In practice, we created a cluster with 1 master and 4 worker nodes, each n1-standard-4 machine type, which was sufficient to handle the dataset sizes in our test containing several million records with room to scale if needed.

Spark is the core execution engine for both batch and streaming parts of the pipeline. We use PySpark, the Python API, to write the transformation logic. Spark's distributed DataFrame model allowed us to perform joins and aggregations on millions of flight records efficiently by parallelizing across the cluster. For batch processing, Spark can easily handle the full dataset in a single job, benefiting from optimizations like predicate pushdown in Parquet and parallel I/O. For streaming, Spark Structured

Streaming provides a unified API that feels like batch DataFrame operations but actually runs continuously in micro-batches. This unification is powerful, as it means a lot of the code between batch and stream jobs can be shared or at least replicated with minor tweaks, ensuring consistent logic in both modes. Spark's in-memory computing and advanced query optimizer known as Catalyst help in speeding up processing of large data sets compared to traditional disk-based frameworks (Demirezen & Navruz, 2023). In our pipeline, Spark handled everything from reading CSVs, performing multi-way joins, to writing to external systems including Google Cloud Storage and BigQuery. We also fine-tuned Spark configurations for this project, such as setting spark.sql.shuffle.partitions to 50 to avoid overly many small shuffle tasks and limiting memory usage per executor to 2 GB to fit within our cluster's RAM, keeping the jobs stable and efficient on the given infrastructure.

BigQuery serves as the data warehouse and analytics engine for the Gold layer outputs. It is a serverless, fully-managed enterprise data warehouse that can execute SQL queries on petabyte-scale data with high performance (Google Cloud, 2025). We chose BigQuery for the Gold layer because it enables straightforward integration with BI tools and because it can be directly populated by Spark through the connector. By writing the daily and live delay statistics to BigQuery, we effectively make those results immediately available for analysis without requiring additional ETL. BigQuery's storage is columnar and optimized for aggregations, which means queries like average delay by airline last month are extremely fast on the daily_delay_stats table. Another advantage is BigQuery's auto-scaling and pay-as-you-go query pricing, so it can handle many concurrent users or heavy queries without our team needing to manage any database instances. In our pipeline, once data is in BigQuery, an analyst could use Google's BI platform such as Looker Studio or any SQL client to create dashboards on flight delays. BigQuery is described as Google Cloud's fully managed, petabyte-scale, and cost-effective analytics data warehouse that lets you run analytics over vast amounts of data in near real time (Google Cloud, 2025), which aligns perfectly with our use case of providing timely analytics on a large volume of flight information.

Results and Outcomes

The implemented data pipeline successfully processes both historical and streaming flight data, achieving the goals of comprehensive analysis and real-time insights. In the batch pipeline, the system was able to ingest and transform the entire flight dataset containing millions of records within a reasonable time on a modest Spark cluster. The cleaning steps removed invalid records, with approximately 1.2% of flights filtered out as canceled, diverted, or out-of-range delays in our test run, and the enrichment steps added valuable context including airline names and airport details to every flight. The resulting Silver data consisted of enriched flight records partitioned by date, totaling a few gigabytes in Parquet format for a year of data. The Gold daily statistics proved useful in summarizing key patterns, with BigQuery able to scan the entire year's daily stats in under a second for simple queries, illustrating the efficiency of the pipeline's final stage.

The streaming pipeline demonstrated near real-time responsiveness. During testing, as new sample flight files were dropped into the streaming input, the Spark streaming job processed them and updated the windowed delay metrics typically within one trigger interval, a few seconds to at most a minute depending on file sizes. This means an analyst monitoring the live_window_delay_stats table in BigQuery could see metrics like number of flights in the last 15 minutes tick upward in almost real time. Such a capability is useful for monitoring current airport conditions. The streaming job's output to BigQuery was configured with checkpoints to ensure exactly-once delivery, and in practice we observed that even if the streaming query was temporarily stopped and restarted, the checkpoint allowed it to continue without duplicating prior outputs.

From a scalability and reliability perspective, the cloud-based design proved advantageous. We leveraged Dataproc's ability to scale, as for a larger dataset or higher throughput requirement, one could simply increase the number of worker nodes or use larger VM instances and Spark would distribute the workload accordingly. Because the data is stored on Google Cloud Storage, adding more compute does not require moving data, as the new workers can read the same Google Cloud Storage files in parallel. The separation of batch and stream jobs means that the workload is isolated, with the batch job scheduled during off-peak hours since it is resource-intensive, while the streaming job runs continuously with a steady resource footprint. Thanks to the use of managed services, reliability is high, as Google Cloud Storage provides nearly 100% data durability and BigQuery has built-in redundancy and availability.

In terms of data visibility for BI, the end result is that end-users including data analysts and BI developers have two main points of access, BigQuery daily stats for historical and trend analysis and BigQuery live stats for current monitoring. For example, a BI dashboard can be built where one view shows a time series of average delay per day over the past year using the daily_delay_stats table and another view shows a gauge or real-time chart of the number of delayed flights in the last 15 minutes at major airports using the live_window_delay_stats table. We tested connectivity by using Google Data Studio, Looker Studio, to connect to these BigQuery tables, and the data populated without issues, allowing us to create interactive charts. The combination of historical and real-time data in one platform greatly enhances the decision-making capability, as stakeholders can drill into historical performance while simultaneously watching current conditions. This outcome validates the initial problem motivation, as the pipeline effectively enables a BI solution that is both retrospective and real-time.

Conclusion and Future Work

This report detailed the design and implementation of a cloud-based data pipeline for flight delay analysis, demonstrating how batch and streaming processing can be combined to serve business intelligence objectives. By using Google Cloud's storage, processing, and warehousing services, we built a scalable pipeline that ingests raw flight data, refines it through Bronze, Silver, and Gold layers, and delivers both historical trend metrics and real-time operational insights. The Medallion architecture proved effective in structuring the data lake, allowing us to incrementally improve data quality and maintain a clear lineage from raw data to final analytics. The decision to incorporate a streaming component alongside batch processing followed the Lambda architecture philosophy and was validated by the pipeline's performance, as end users can obtain immediate information on current delays as well as reliable reports on long-term patterns.

Several future work directions can enhance and extend this pipeline. Integration of additional data sources is a natural next step, with weather data or air traffic control events integrated into the Bronze layer, such as joining flights with weather information in Silver, to analyze how external factors correlate with delays. Enhancing the streaming pipeline with message-based ingestion using tools like Apache Kafka or Google Pub/Sub is another improvement, as currently we simulated streaming via file drops, but a more robust real-world system might consume JSON messages from a live flight status API. On the batch side, as data volume grows, we could consider using partition pruning and clustering in BigQuery for the daily stats table to maintain query performance or even utilize BigQuery's built-in ML for predictive modeling on the Gold data.

From an architecture standpoint, a promising evolution is to implement a unified Lakehouse approach using tools like Delta Lake or BigQuery's BigLake. This would enable ACID transactions and time-travel on the Bronze and Silver data, making the pipeline more flexible for updates and easier to manage in terms of consistency. We could also leverage Google Dataplex for data cataloging and governance across Google Cloud Storage and BigQuery, which would help in managing schema versions and data quality rules centrally. In terms of orchestration, productionizing this pipeline would involve scheduling the batch job, perhaps via Cloud Composer, to run at regular intervals such as daily or weekly and monitoring the streaming job with alerts in case of any lag or failures.

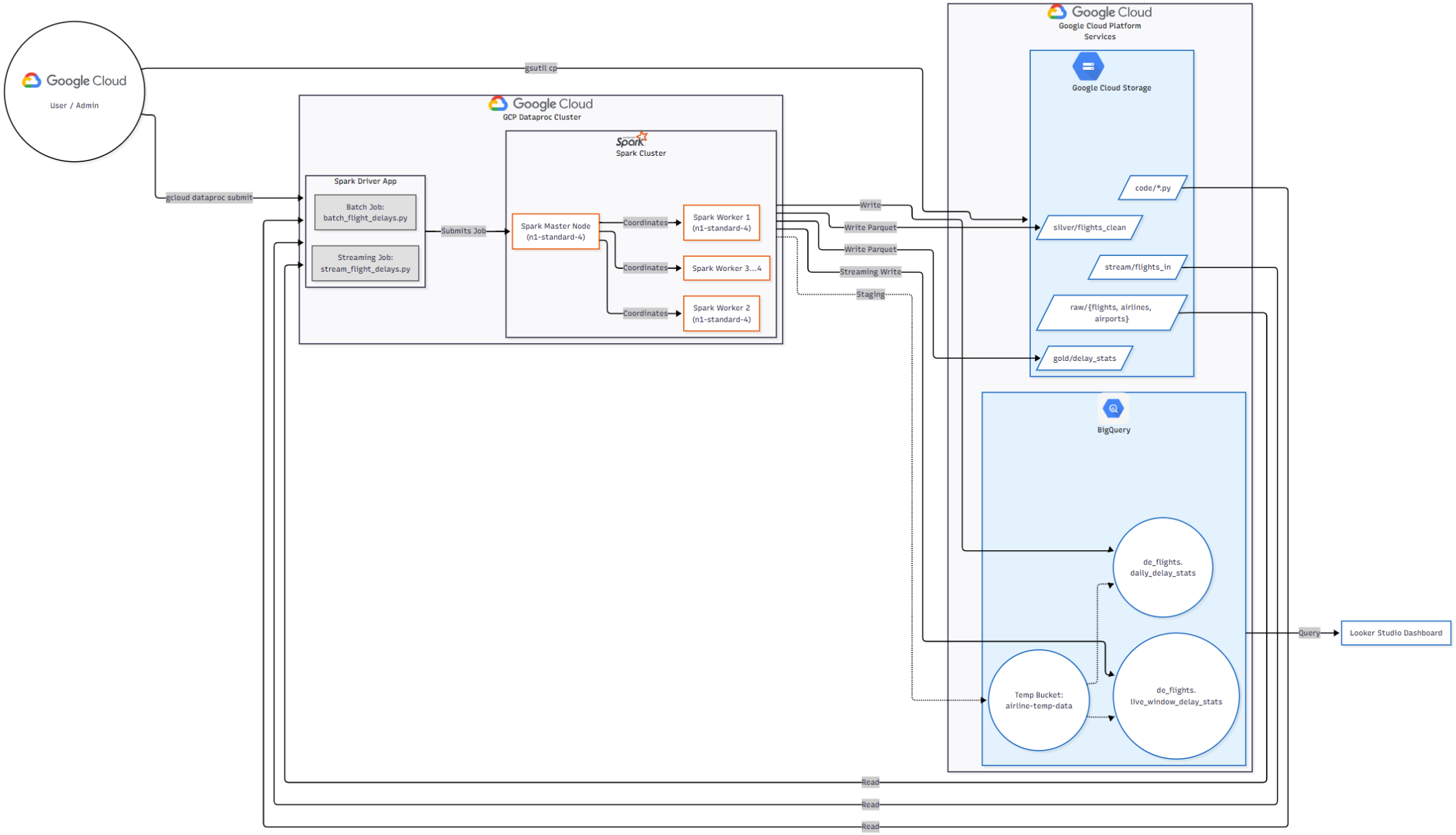
Another future direction is cost optimization and autoscaling. Using Dataproc's autoscaling for the streaming job could allow the cluster to scale down during periods of low input such as midnight when fewer flights occur and scale up during peak hours, reducing costs while maintaining performance. Similarly, the batch cluster can be created on demand and destroyed after the job to save on continuous resource costs. BigQuery's cost can be controlled by partitioning and using aggregated tables, which we have already applied, but further optimization like materialized views on recent data could be explored.

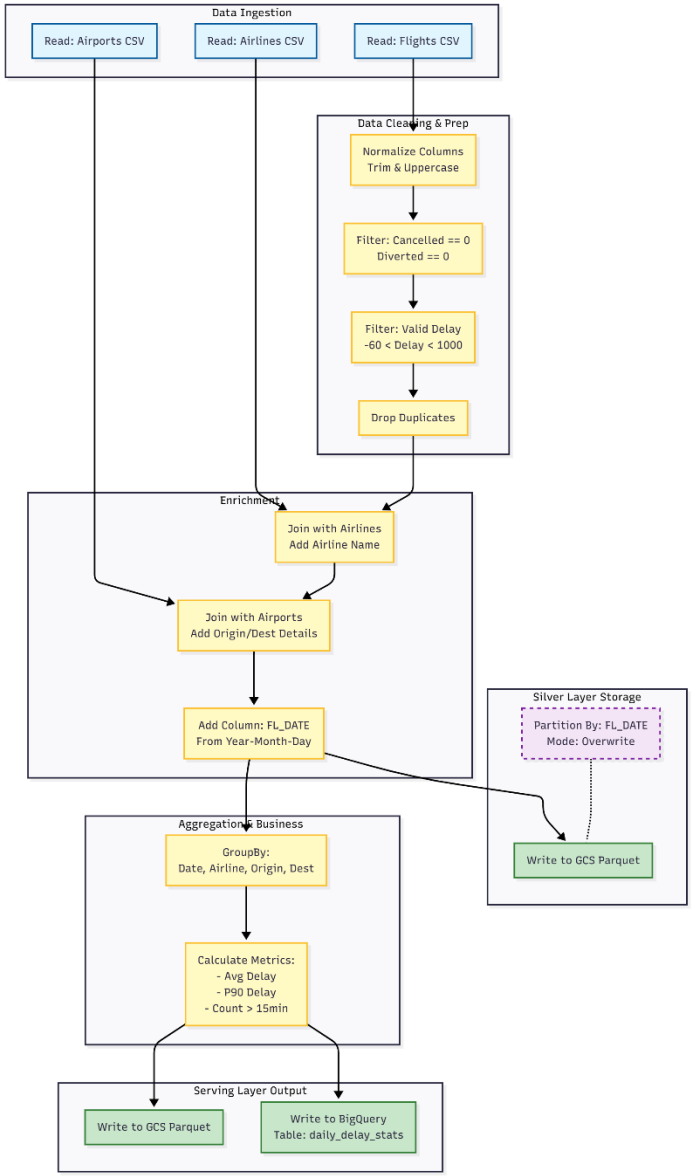
In conclusion, the pipeline achieved its objectives of delivering a comprehensive BI-ready dataset and real-time analytics for flight delays. It showcases how modern cloud data engineering techniques, combining batch and stream processing in a layered architecture, can overcome the limitations of traditional approaches and meet the needs of both historical analysis and live monitoring. The design principles and choices discussed here can serve as a reference for similar big data analytics projects in other domains that require the fusion of historical and real-time data, such as smart logistics, network monitoring, or IoT analytics. With planned enhancements in data integration, tooling, and automation, the flight delays analysis pipeline can continue to evolve into a more intelligent and autonomous analytics system, driving better decisions in air traffic operations and beyond.

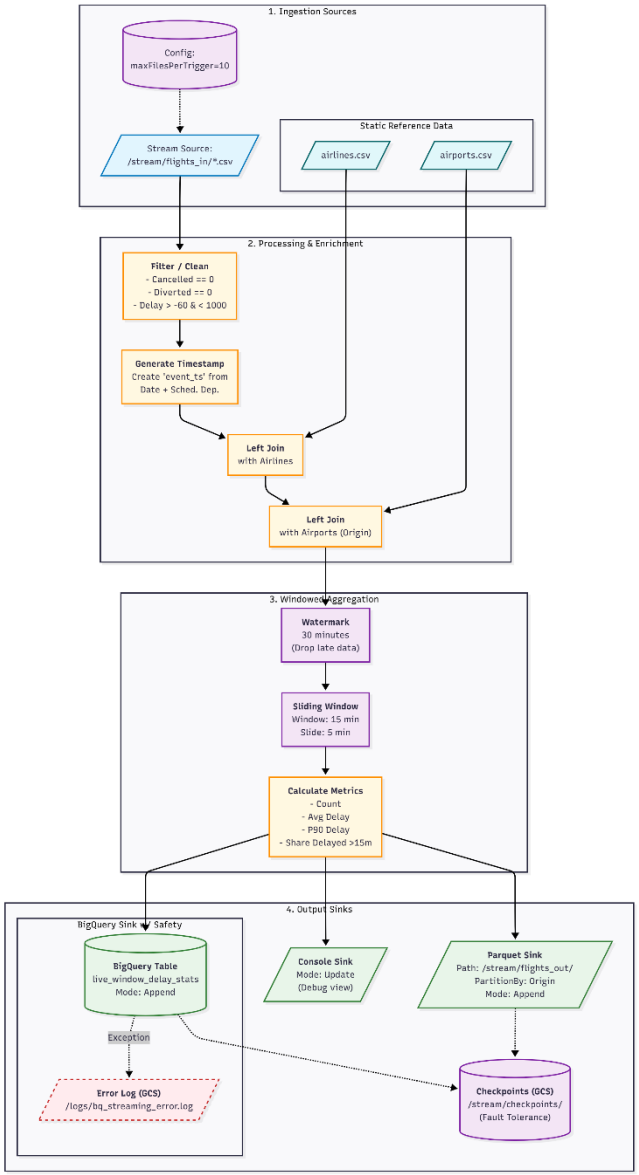
References

1. U.S. Department of Transportation. (2017). *2015 flight delays and cancellations* [Data set]. Kaggle. https://www.kaggle.com/datasets/usdot/flight-delays
2. Kamdar, V., Vaidya, P., & Patel, S. (2025). *A modern approach to real-time air traffic management system* (arXiv:2504.03652 [cs.DC]). arXiv. https://arxiv.org/abs/2504.03652
3. Demirezen, M. U., & Navruz, T. S. (2023). Performance analysis of lambda architecture-based big-data systems on air/ground surveillance application with ADS-B data. *Sensors, 23*(17), 7580. https://doi.org/10.3390/s23177580
4. Idera. (2025, October 31). *Understanding the three layers of medallion architecture*. ER/Studio Data Architecture Blog. https://erstudio.com/blog/understanding-the-three-layers-of-medallion-architecture/
5. Google Cloud. (n.d.). *BigQuery documentation: Cloud data warehouse for analytics*. Retrieved December 2, 2025, from https://cloud.google.com/bigquery/docs/
6. CloudChipr. (2025, September 29). *Demystifying Google Cloud Dataproc in 2025: The practical guide* [Blog post]. https://cloudchipr.com/blog/dataproc
7. Hartwig, A. (2024, October 22). *Google Cloud Storage*. Dremio. https://www.dremio.com/wiki/google-cloud-storage/
8. Databricks. (2025). *What is a medallion architecture?* Retrieved December 3, 2025, from https://www.databricks.com/glossary/medallion-architecture
9. 2015 U.S.-Based Airline traffic data. (n.d.). Bureau of Transportation Statistics. https://www.bts.gov/newsroom/2015-us-based-airline-traffic-data

Martijn van Iterson	*Implemented one of the Spark pipelines, configured GCS bucket structure, assisted in data architecture design, created the demonstration video, and contributed to the final report.*
Floris Vermeulen	*Set up the BigQuery environment, implemented data ingestion workflows, contributed to the design of the data architecture, and co-authored the final report.*
Niek Fleerakkers	*Implemented the second Spark pipeline (batch/stream), configured Dataproc job submission, assisted with debugging pipeline execution, and contributed to the final report.*
Patryk Grodek	*Set up and tested GCP infrastructure components (GCS, Pub/Sub/Kafka alternative, Dataproc configuration), worked on pipeline deployment scripts, and helped write the final report.*
Samir Sabitli	*Performed dataset preprocessing design, validated pipeline outputs, supported Spark code development, and contributed to the final report.*


Google Cloud
User / Admin
gsutil cp
gcloud dataproc submit
Google Cloud
GCP Dataproc Cluster
Spark Cluster
Spark Driver App
Batch Job:
batch_flight_delays.py
Streaming Job:
stream_flight_delays.py
Submits Job
Spark Master Node
(n1-standard-4)
Coordinates
Spark Worker 1
(n1-standard-4)
Coordinates
Spark Worker 3...4
Coordinates
Spark Worker 2
(n1-standard-4)
Google Cloud
Google Cloud Platform
Services
Google Cloud Storage
code/*.py
silver/flights_clean
stream/flights_in
raw/{flights, airlines,
airports}
gold/delay_stats
BigQuery
de_flights.
daily_delay_stats
Temp Bucket:
airline-temp-data
de_flights.
live_window_delay_stats
Write
Write Parquet
Write Parquet
Streaming Write
Staging
Query
Looker Studio Dashboard
Read
Read
Read


Data Ingestion
Read: Airports CSV
Read: Airlines CSV
Read: Flights CSV
Data Cleaning & Prep
Normalize Columns
Trim & Uppercase
Filter: Cancelled == 0
Diverted == 0
Filter: Valid Delay
-60 < Delay < 1000
Drop Duplicates
Enrichment
Join with Airlines
Add Airline Name
Join with Airports
Add Origin/Dest Details
Add Column: FL_DATE
From Year-Month-Day
Silver Layer Storage
Partition By: FL_DATE
Mode: Overwrite
Write to GCS Parquet
Aggregation & Business
GroupBy:
Date, Airline, Origin, Dest
Calculate Metrics:
- Avg Delay
- P90 Delay
- Count > 15min
Serving Layer Output
Write to GCS Parquet
Write to BigQuery
Table: daily_delay_stats


1. Ingestion Sources
Config:
maxFilesPerTrigger=10
Stream Source:
/stream/flights_in/*.csv
Static Reference Data
airlines.csv
airports.csv
2. Processing & Enrichment
Filter / Clean
- Cancelled == 0
- Diverted == 0
- Delay > -60 & < 1000
Generate Timestamp
Create 'event_ts' from
Date + Sched. Dep.
Left Join
with Airlines
Left Join
with Airports (Origin)
3. Windowed Aggregation
Watermark
30 minutes
(Drop late data)
Sliding Window
Window: 15 min
Slide: 5 min
Calculate Metrics
- Count
- Avg Delay
- P90 Delay
- Share Delayed >15m
4. Output Sinks
BigQuery Sink + Safety
BigQuery Table
live_window_delay_stats
Mode: Append
Exception
Error Log (GCS)
/logs/bq_streaming_error.log
Console Sink
Mode: Update
(Debug view)
Parquet Sink
Path: /stream/flights_out/
PartitionBy: Origin
Mode: Append
Checkpoints (GCS)
/stream/checkpoints/
(Fault Tolerance)