

Airline Flight Status & Delay Monitoring

Complete Documentation:

Environment: Development (Dev suffix on all resources)

Project Scope: Hybrid Batch + Streaming Data Pipeline for Airline Flight Delay Monitoring

Executive Summary

This comprehensive documentation outlines a **production-grade, hybrid batch-and-streaming data engineering solution** for monitoring airline flight delays on AWS. The system processes historical flight data (CSV → Parquet → Redshift) for analytics and ingests real-time flight events for immediate alerting.

Key Achievements Across All Parts

Part 1: Infrastructure Foundation

- AWS S3 buckets (raw & processed data)
- Kinesis Data Streams for real-time ingestion
- IAM roles and security foundation
- Infrastructure as Code (Terraform)

Part 2: Batch Processing (ETL)

- AWS Glue crawler (schema detection)
- Production-grade Glue ETL job (31 data transformations)
- Parquet output with year/month partitioning
- Complete error handling and logging

Part 3: Analytics Warehouse

- Amazon Redshift Serverless (OLAP)
- Fact table optimized for queries
- 3 materialized views (pre-computed analytics)
- Fast BI dashboards (< 1 second queries)

Part 4: Real-Time Streaming

- Lambda-based Kinesis consumer
- Streaming data quality checks
- S3 quarantine for bad events
- SNS/SQS alert notifications
- Multi-channel alerting (email + queue)

Technology Stack

Layer	Technology	Purpose
Data Ingestion	S3, Kinesis Streams	Raw data storage & streaming
Processing	AWS Glue, PySpark	ETL transformations
Storage	Redshift Serverless, Parquet	Analytics & intermediate
Real-Time	Lambda, SNS, SQS	Event processing & alerts
IaC	Terraform, Python	Infrastructure automation
Monitoring	CloudWatch, CloudWatch Logs	Observability

Architecture Overview

Complete Data Flow

Batch Pipeline (Historical):

Raw Flight Data (CSV) → S3 Raw Bucket → Glue Crawler → Glue Catalog → Glue ETL Job (31 transformations) → S3 Processed Bucket (Parquet, partitioned) → Redshift COPY → Fact Table → Materialized Views → BI Dashboards

Streaming Pipeline (Real-Time):

Flight Events (JSON) → Producer Script → Kinesis Stream → Lambda Consumer → Data Quality Validation → Alert Classification → SNS Topic (Email) + SQS Queue (Processing)

Component Responsibilities

Component	Responsibility	Technology
Raw Data Ingestion	Store historical flight CSV files	S3
Schema Detection	Automatically detect & catalog data schema	Glue Crawler
Data Transformation	Clean, enrich, transform raw data	Glue ETL (PySpark)
Processed Storage	Store cleaned data as Parquet	S3 (partitioned)
Analytics Warehouse	Fast OLAP queries on historical data	Redshift Serverless
Real-Time Stream	Buffer incoming flight events	Kinesis Data Streams
Event Processing	Validate, classify, and route events	Lambda
Bad Event Handling	Quarantine failed validations	S3
Alert Distribution	Email & queue-based notifications	SNS + SQS
Monitoring	Logs, metrics, alarms	CloudWatch

Part 1: Infrastructure Foundation & Data Ingestion

Overview

Part 1 establishes the AWS infrastructure foundation using Infrastructure as Code (Terraform). This includes S3 buckets, Kinesis streams, IAM roles, and configuration management—all prerequisites for the batch and streaming pipelines.

Project Directory Structure

```
airline-flight-status-delay-monitoring/
└── terraform/
    ├── backend.tf
    ├── main.tf
    ├── outputs.tf
    ├── provider.tf
    └── variables.tf
    └── terraform.tfvars
    └── lambda_artifacts/
        └── lambda_kinesis_consumer.zip
    └── modules/
    └── s3/
        └── kinesis/
        └── iam/
        └── glue/
        └── redshift_serverless/
        └── sns_sqrs/
    └── scripts/
        └── python/
            └── lambda_kinesis_consumer.py
            └── simulate_flight_events_step2.py
```

Terraform Variables

File: terraform/variables.tf

```
variable "aws_region" {
  type = string
  default = "us-east-1"
}

variable "environment" {
  type = string
  default = "dev"
}

variable "project_name" {
  type = string
  default = "airline-delay-monitoring"
}
```

```

variable "raw_data_bucket_name" {
  type = string
  default = "airline-delay-monitoring-dev-raw-data"
}

variable "processed_data_bucket_name" {
  type = string
  default = "airline-delay-monitoring-dev-processed-data"
}

variable "kinesis_stream_name" {
  type = string
  default = "airline-delay-monitoring-dev-flight-stream"
}

```

AWS Provider Configuration

File: `terraform/provider.tf`

```

terraform {
  required_version = ">= 1.0"

  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 5.0"
    }
  }

  backend "s3" {
    bucket = "terraform-state-prod"
    key = "airline-flight-monitoring/terraform.tfstate"
    region = "us-east-1"
    encrypt = true
    dynamodb_table = "terraform-state-lock"
  }
}

provider "aws" {
  region = var.aws_region

  default_tags {
    tags = {
      Environment = var.environment
      Project = var.project_name
      ManagedBy = "Terraform"
    }
  }
}

```

Main Terraform Configuration

File: `terraform/main.tf` - Orchestrates all modules and connects infrastructure components

Key modules deployed:

- S3 Buckets (raw & processed data)

General purpose buckets (6) Info

[Copy ARN](#)[Empty](#)[Delete](#)[Create bucket](#)

Buckets are containers for data stored in S3.

 Find buckets by name< 1 >

Name	AWS Region	Creation date
airline-delay-monitoring-dev-processed-data	US East (N. Virginia) us-east-1	November 29, 2025, 18:01:02 (UTC-05:00)
airline-delay-monitoring-dev-raw-data	US East (N. Virginia) us-east-1	November 29, 2025, 18:01:02 (UTC-05:00)
airline-monitoring-terraform-state-dev	US East (N. Virginia) us-east-1	November 23, 2025, 19:20:13 (UTC-05:00)

- Kinesis Stream (flight events)

airline-delay-monitoring-dev-flight-stream Info

Data stream summary

Status

Active

Capacity mode

Provisioned

Data retention period

1 day

Maximum record size

1024 KiB

ARN

arn:aws:kinesis:us-east-1:844840482726:stream/airline-delay-monitoring-dev-flight-stream

- IAM Roles (Glue, Redshift, Lambda)

All IAM roles are identities you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

 Search

Role name	Trusted entities	Last activity
airline-delay-monitoring-dev-glue-service-role	AWS Service: glue	2 hours ago
airline-delay-monitoring-dev-kinesis-consumer-role	AWS Service: lambda	-
airline-delay-monitoring-dev-s3-access-role	AWS Service: glue	-
airline-dev-flighevents-consumer-role	AWS Service: lambda	9 minutes ago
airline-dev-redshift-serverless-s3-role	AWS Service: redshift	20 hours ago
AWSRoleForRedshift	AWS Service: redshift (Service-Linked)	2 hours ago

- Glue Infrastructure (crawler & jobs)

The screenshot shows the AWS Glue job editor interface. On the left, a sidebar lists various AWS services under 'AWS Glue'. The main area displays a job named 'airline-flightdata-batch-raw-to-processed-dev'. The 'Script' tab is selected, showing the following Python code:

```

1 # Glue ETL job: clean FlightData (raw -> processed) and write partitioned Parquet to S3.
2
3
4 from awsglue.context import GlueContext
5 from awsglue.job import Job
6 from awsglue.utils import getResolvedOptions
7 from pyspark.context import SparkContext
8 from pyspark.sql import functions as F
9 import sys
10
11 # 1. Read job name from Glue arguments
12 args = getResolvedOptions(sys.argv, ["JOB_NAME"])
13
14 # 2. Create Spark and Glue contexts
15 sc = SparkContext()
16 glue_context = GlueContext(sc)
17 spark = glue_context.spark_session
18 spark.conf.set("spark.sql.sources.partitionOverwriteMode", "dynamic")
19 job = Job(glue_context)
20 job.init(args["JOB_NAME"], args)
21

```

Crawlers

A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Crawlers (2) Info					
Last updated (UTC) Action Run Create crawler					
View and manage all available crawlers.					
<input type="text"/> Filter crawlers					
<input type="checkbox"/>	Name	State	Schedule	Last run	Last run timestamp
<input type="checkbox"/>	airline-delay-monitoring-dev-processed-crawler	Ready	At 02:30 AM	Succeeded	December 1, 2025 ...
<input type="checkbox"/>	airline-delay-monitoring-dev-raw-crawler	Ready	At 02:00 AM	Succeeded	December 1, 2025 ...

airline_delay_monitoring_dev_db

Announcing new optimization features for Apache Iceberg tables
Optimize storage for Apache Iceberg tables with automatic snapshot retention and orphan file deletion. [Learn more](#)

Database properties			
Name	Description	Location	Created on (UTC)
airline_delay_monitoring_dev_db	-	-	November 29, 2025

Tables (2)

View and manage all available tables.

Tables (2) Info					
Last updated (UTC) Delete Add tables using cr					
<input type="checkbox"/>	Name	Database	Location	Classification	Deprecat
<input type="checkbox"/>	airline_delay_monitoring_dev_processed_data	airline_delay_monito	s3://airline-delay-monitoring-dev-processed-data/	Parquet	-
<input type="checkbox"/>	airline_delay_monitoring_dev_raw_data	airline_delay_monito	s3://airline-delay-monitoring-dev-raw-data/	CSV	-

- Redshift Serverless (analytics warehouse)

Amazon Redshift Serverless

Serverless dashboard [Info](#)

[C](#)

Namespace overview [Info](#)

Namespace data from your account

Total snapshots	Datashares in my account	Datashares requiring authorization	Datashares from other accounts
0	0	0	0

Namespaces / Workgroups [Info](#)

Namespace	Status	Workgroup	Status	Average query duration	Average number of queries run daily
airline-dev-ns	Available	airline-dev-wg	Available		

Total compute costs

Choose a workgroup to visualize the costs in the Explorer

Total consumed RCU: --

Account ID: 8448-4048-2726 vamshi2

Redshift query editor v2

Serverless: airline-dev-wg

```

1 SELECT * FROM my_carrier_month_delay
2 ORDER BY year, month, avg_arr_delay DESC
3 LIMIT 20;

```

Result 1 (20)

uniquecarrier	year	month	flight_count	avg_arr_delay
YV	2008	1	5669	65
OO	2008	1	14718	63
UA	2008	1	13719	59
9E	2008	1	6192	54
MQ	2008	1	13643	54
AA	2008	1	17120	51
BR	2008	1	5221	46

- SNS/SQS (alerts)

Amazon SQS > Queues > airline-dev-flight-delay-alerts-queue > Send and receive messages			
Messages available	Polling duration	Maximum message count	Polling progress
1800	30	10	0 receives/second
Messages (10)			
<input type="text"/> Search messages			
ID	Sent	Size	Received
51705194-1d27-4faa-9638-b231f1054fc8	2025-11-30T21:15-05:00	305 bytes	1
77e9078d-195a-4bba-b260-f0a63ce1758c	2025-11-30T21:22-05:00	306 bytes	1
b4cbff6-07d9-4be2-83ab-280dbe798778	2025-11-30T21:22-05:00	306 bytes	1
58577208-85db-4cad-840f-e41d9df22150	2025-11-30T21:22-05:00	302 bytes	1
df3152f5-ea51-4ad8-b29b-e473aed640a5	2025-11-30T21:22-05:00	306 bytes	1
96d61145-a485-4a2b-9e89-508ca61849eb	2025-11-30T21:22-05:00	304 bytes	1
e0db3ad0-757f-4483-b473-4f083a8616c4	2025-11-30T21:22-05:00	306 bytes	1
ac00f03d-7f21-4027-8f04-c4f272f1bd60	2025-11-30T21:23-05:00	305 bytes	1
6b9aea4c-39ff-4d6a-a8e1-4bfa969fc4c6	2025-11-30T21:23-05:00	306 bytes	1

Email notification: SNS subscription

<input type="checkbox"/>	☆ AWS Notifications 100	Flight delay alert: SEVERE_DEPARTURE_DELAY - {"alert_type": "SEVERE_DEPARTUR..."	10:32 PM
<input type="checkbox"/>	☆ AWS Notifications 100	Flight delay alert: PERSISTENT_ROUTE_DELAY - {"alert_type": "PERSISTENT_ROUTE_..."	10:32 PM
<input type="checkbox"/>	☆ AWS Notifications 100	Flight delay alert: PERSISTENT_ROUTE_DELAY - {"alert_type": "PERSISTENT_ROUTE_..."	10:32 PM
<input type="checkbox"/>	☆ AWS Notifications 25	Flight delay alert: SEVERE_DEPARTURE_DELAY - {"alert_type": "SEVERE_DEPARTUR..."	10:32 PM
<input type="checkbox"/>	☆ AWS Notifications 24	Flight delay alert: PERSISTENT_ROUTE_DELAY - {"alert_type": "PERSISTENT_ROUTE_..."	10:32 PM
<input type="checkbox"/>	☆ AWS Notifications 13	Flight delay alert: SEVERE_ARRIVAL_DELAY - {"alert_type": "SEVERE_ARRIVAL_DELA..."	10:32 PM
<input type="checkbox"/>	☆ AWS Notifications 100	Flight delay alert: SEVERE_DEPARTURE_DELAY - {"alert_type": "SEVERE_DEPARTUR..."	10:31 PM
<input type="checkbox"/>	☆ AWS Notifications	AWS Notification - Subscription Confirmation - You have chosen to subscribe to the top...	9:16 PM

- Lambda Consumer (streaming processor):

The screenshot shows the AWS Lambda Function Overview page for the function 'airline-dev-flighevents-consumer'. The function is triggered by 'Kinesis'. The 'Layers' section indicates '(0)'. A 'Triggers' section shows 'Kinesis' and a '+ Add trigger' button.

- Cloudwatch:

The screenshot shows the AWS CloudWatch Log Events page for the log group '/aws/lambda/airline-dev-flighevents-consumer'. It displays three log entries:

```

[INFO] 2025-12-01T01:42:19.804Z    72a7e725-9905-4f27-9863-41159aa63cec    Flight event: carrier=WN flightnum=2719 origin=RNO dest=OAK depdelay=25.0 arrdelay=30.0 flightdate=2008-01-03
[INFO] 2025-12-01T01:42:19.804Z    72a7e725-9905-4f27-9863-41159aa63cec    Flight event: carrier=WN flightnum=2732 origin=RNO dest=OAK depdelay=65.0 arrdelay=74.0 flightdate=2008-01-03
[INFO] 2025-12-01T01:42:19.804Z    72a7e725-9905-4f27-9863-41159aa63cec    ALERT:
{
  "alert_type": "SEVERE_DEPARTURE_DELAY",
  "flightdate": "2008-01-03",
  "uniquecarrier": "WN",
  "flightnum": 2732,
  "origin": "RNO",
  "dest": "OAK",
  "depdelay": 65,
  "arrdelay": 74,
  "distance": 180,
  "generated_at_utc": "2025-12-01T01:42:19.804566Z",
  "message": "SEVERE_DEPARTURE_DELAY for WN 2732 RNO->OAK"
}

```

The screenshot shows the AWS CloudWatch Log Events interface. At the top, there's a breadcrumb navigation: /aws/lambda/airline-dev-flighevents-consumer > 2025/12/01/[\$.LATEST]3b09d50d67d442b8bd3e4ddfcfa854c1. Below that, a section titled "Log events" is shown with a search bar containing "Filter events - press enter to search". To the right of the search bar are buttons for "Clear", "1m", "30m", "1h", "12h", and "Custom". Further right are "Actions" and "Start tailing" buttons. A "Display" button with a dropdown arrow is also present. The main area is titled "Message" and contains several log entries. One entry is expanded to show a JSON object representing a flight alert:

```

[INFO] 2025-12-01T01:42:19.804Z 72a7e725-9905-4f27-9863-41159aa63cec Flight event: carrier=WN flightnum=2719 origin=RNO dest=OAK depdelay=25.0 arrdelay=03
[INFO] 2025-12-01T01:42:19.804Z 72a7e725-9905-4f27-9863-41159aa63cec Flight event: carrier=WN flightnum=2732 origin=RNO dest=OAK depdelay=65.0 arrdelay=03
[INFO] 2025-12-01T01:42:19.804Z 72a7e725-9905-4f27-9863-41159aa63cec ALERT:
{
  "alert_type": "SEVERE_DEPARTURE_DELAY",
  "flightdate": "2008-01-03",
  "uniquecarrier": "WN",
  "flightnum": 2732,
  "origin": "RNO",
  "dest": "OAK",
  "depdelay": 65,
  "arrdelay": 74,
  "distance": 180,
  "generated_at_utc": "2025-12-01T01:42:19.804566Z",
  "message": "SEVERE_DEPARTURE_DELAY for WN 2732 RNO->OAK"
}

```

S3 Buckets Configuration

File: terraform/modules/s3/main.tf

Features implemented:

- Raw data bucket with versioning enabled
- Processed data bucket with encryption
- Lifecycle rules for cost optimization (archive to Glacier after 90 days)
- Server-side encryption (AES256)

Raw bucket stores: Original CSV files from airlines

Processed bucket stores: Parquet files, scripts, quarantined events

Kinesis Stream Module

File: terraform/modules/kinesis/main.tf

Configuration:

- Stream name: airline-delay-monitoring-dev-flight-stream
- Shard count: 1 (scalable)
- Retention period: 24 hours
- Used for real-time flight event ingestion

Initialize and Deploy Infrastructure

Step 1: Configure AWS CLI

```
aws configure
```

Enter: AWS Access Key ID, Secret Access Key, region (us-east-1), output (json)

Step 2: Verify Terraform Version

```
terraform version # Should be >= 1.0
```

Step 3: Initialize Terraform

```
cd terraform  
terraform init
```

Step 4: Plan Deployment

```
terraform plan -out=tfplan
```

Step 5: Apply Configuration

```
terraform apply tfplan
```

This creates all AWS resources: S3 buckets, Kinesis stream, IAM roles, Glue database, etc.

Step 6: Verify Outputs

```
terraform output
```

Part 2: Batch Processing - Raw Data to Analytics

Overview

Part 2 transforms raw airline flight data (CSV format) into production-ready Parquet files using AWS Glue ETL. This includes 31 data transformations, cleaning, enrichment, and optimization for analytics queries.

Raw Data Schema

The dataset contains 30 columns of flight information:

Column	Type	Description
year	bigint	Flight year
month	bigint	Flight month (1-12)
dayofmonth	bigint	Day of month (1-31)
dayofweek	bigint	Day of week (1=Monday)
deptime	bigint	Actual departure time (HHMM)
crsdeptime	bigint	Scheduled departure time (HHMM)
arrtime	bigint	Actual arrival time (HHMM)
crsarrrtime	bigint	Scheduled arrival time (HHMM)
uniquecarrier	string	Airline code (AA, WN, etc.)
flightnum	bigint	Flight number
tailnum	string	Aircraft tail number
actualelapsedtime	bigint	Actual flight duration (minutes)
crselapsedtime	bigint	Scheduled flight duration (minutes)
airtime	bigint	Time in air (minutes)
arrdelay	bigint	Arrival delay (minutes)
depdelay	bigint	Departure delay (minutes)
origin	string	Departure airport code
dest	string	Arrival airport code
distance	bigint	Flight distance (miles)
taxiin	bigint	Time from landing to gate (minutes)
taxiout	bigint	Time from gate to takeoff (minutes)
cancelled	bigint	Flight cancelled (0=no, 1=yes)

cancellationcode	string	Reason for cancellation
diverted	bigint	Flight diverted (0=no, 1=yes)
carrierdelay	bigint	Delay due to airline (minutes)
weatherdelay	bigint	Delay due to weather (minutes)
nasdelay	bigint	Delay due to NAS (minutes)
securitydelay	bigint	Delay due to security (minutes)
lateaircraftdelay	bigint	Delay due to late aircraft (minutes)

Glue Job IAM Role

Role Name: airline-delay-monitoring-dev-glue-service-role

Trust Relationship: Allows glue.amazonaws.com service to assume this role

Permissions: S3 access (read raw, write processed), Glue Catalog access, CloudWatch Logs

ETL Script Implementation

File: scripts/python/flightdata_raw_to_processed.py

The ETL job performs these 31 transformations:

1. Type casting (bigint → int for memory efficiency)
2. Boolean conversion (0/1 → true/false for cancelled/diverted)
3. Null handling in delay fields (NULL → 0)
4. String cleanup (TRIM, UPPERCASE for consistency)
5. Timestamp building (Year/Month/Day/Time → proper DATE and TIMESTAMP)
6. Data quality filtering (distance > 0, delays >= 0)
7. Duplicate removal (on flightdate, flightnum, origin, dest)
8. Feature engineering (IsDelayed flag, TotalDelay calculation)
9. Column selection (31 final columns)
10. Partitioned output (by year and month)

Key Processing Steps:

- **Input:** Raw CSV from Glue Catalog
- **Processing:** PySpark transformations and validations
- **Output:** Partitioned Parquet files (s3://.../processed/flights/year=YYYY/month=MM/)
- **Optimization:** Year/month partitioning enables fast time-range queries

Deploy ETL Script

Upload Script to S3

```
aws s3 cp scripts/python/flightdata_raw_to_processed.py  
s3://airline-delay-monitoring-dev-processed-data/scripts/flightdata_raw_to_processed.py
```

Run Glue Job

```
aws glue start-job-run  
--job-name airline-flightdata-batch-raw-to-processed-dev
```

Monitor Execution

```
aws glue get-job-run  
--job-name airline-flightdata-batch-raw-to-processed-dev  
--run-id <RUN_ID>  
--query "JobRun.JobRunState"
```

States: STARTING → RUNNING → SUCCEEDED (or FAILED)

Verify Output

```
aws s3 ls s3://airline-delay-monitoring-dev-processed-data/processed/flights/  
aws s3 ls s3://airline-delay-monitoring-dev-processed-data/processed/flights/year=2008/month=1/
```

The screenshot shows the Amazon S3 console interface. The left sidebar has a dark theme with white text. It lists several buckets and navigation links for access management, storage management, and storage rights. The main area shows a breadcrumb path: Amazon S3 > Buckets > airline-delay-monitoring-dev-processed-data > processed/ > flights/ > year=2008/. The current view is on the 'Objects' tab, which displays a list of 6 objects. Each object is represented by a small icon, a checkbox, a name ('month=1/' through 'month=6/'), a type ('Folder'), and a last modified date ('-'). There are also buttons for 'Copy S3 URI', 'Copy URL', and 'Download'.

Name	Type	Last modified
month=1/	Folder	-
month=2/	Folder	-
month=3/	Folder	-
month=4/	Folder	-
month=5/	Folder	-
month=6/	Folder	-

Part 3: Redshift OLAP Analytics Layer

Overview

Part 3 loads processed Parquet data into Amazon Redshift for high-performance OLAP analytics. Includes table design with optimized encoding, materialized views for pre-computed aggregations, and query patterns for dashboards.

Redshift IAM Role

Role Name: airline-dev-redshift-serverless-s3-role

Trust Relationship: Allows redshift-serverless.amazonaws.com service to assume role

Permissions: S3 read access (processed data), S3 write access (UNLOAD results)

Fact Table Design

Table Name: public.fact_flights_2008

Contains 31 columns matching processed Parquet schema:

dayofmonth, dayofweek, flightdate, uniquecarrier, flightnum, tailnum, origin, dest, deptime, crsdeptime, arrtime, crsarrrtime, actualedapsedtime, crselapsedtime, airtime, arrdelay, depdelay, distance, taxiin, taxiout, cancelled, cancellationcode, diverted, carrierdelay, weatherdelay, nasdelay, securitydelay, lateaircraftdelay, dep_scheduled_ts, isdelayed, totaldelay

Design Decisions:

- **DISTSTYLE AUTO:** Let Redshift choose optimal distribution
- **SORTKEY:** (flightdate, origin, dest) - speeds time-range and route queries
- **Column Encoding:** az64 for integers, lzo for strings, raw for booleans/dates

```
CREATE TABLE public.fact_flights_2008 (
    dayofmonth      integer ENCODE az64,
    dayofweek       integer ENCODE az64,
    flightdate      date   ENCODE raw,
    uniquecarrier   varchar(10) ENCODE lzo,
    flightnum       integer ENCODE az64,
    tailnum         varchar(20) ENCODE lzo,
    origin          varchar(10) ENCODE raw,
    dest            varchar(10) ENCODE raw,
    deptime         integer ENCODE az64,
    crsdeptime     integer ENCODE az64,
    arrtime         integer ENCODE az64,
```

```

        crsarrtime      integer ENCODE az64,
actualelapsedtime integer ENCODE az64,
crselapsedtime    integer ENCODE az64,
airtime           integer ENCODE az64,
arrdelay          integer ENCODE az64,
depdelay          integer ENCODE az64,
distance          integer ENCODE az64,
taxiin            integer ENCODE az64,
taxiout            integer ENCODE az64,
cancelled         boolean ENCODE raw,
cancellationcode varchar(10) ENCODE lzo,
diverted          boolean ENCODE raw,
carrierdelay     integer ENCODE az64,
weatherdelay      integer ENCODE az64,
nasdelay          integer ENCODE az64,
securitydelay    integer ENCODE az64,
lateaircraftdelay integer ENCODE az64,
dep_scheduled_ts timestamp without time zone ENCODE az64,
isdelayed         boolean ENCODE raw,
totaldelay        integer ENCODE az64
)
DISTSTYLE AUTO
SORTKEY (flightdate, origin, dest);

```

Load Data into Redshift

```

COPY public.fact_flights_2008
FROM 's3://airline-delay-monitoring-dev-processed-data/processed/flights/'
IAM_ROLE 'arn:aws:iam::YOUR_ACCOUNT_ID:role/airline-dev-redshift-serverless-s3-role'
FORMAT AS PARQUET;

```

This loads all Parquet files recursively from S3 prefix into the fact table.

Materialized Views for Analytics

View 1: Carrier & Month Delay Analysis

Aggregates flights by airline and month to show on-time performance trends

Columns: uniquecarrier, year, month, flight_count, avg_arr_delay_min, avg_dep_delay_min, pct_delayed

View 2: Route Delay Performance

Analyzes on-time performance by route (origin → dest)

Only includes routes with 100+ flights for statistical significance

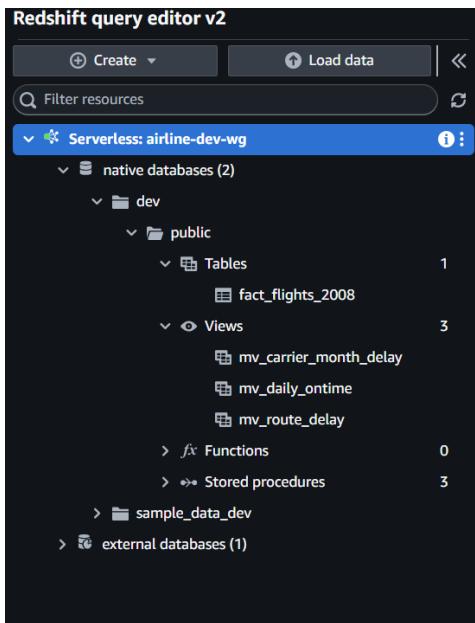
Columns: origin, dest, flight_count, avg_arr_delay_min, avg_dep_delay_min, pct_delayed, avg_distance_miles

View 3: Daily On-Time Performance

Tracks system-wide performance day-by-day for trend analysis

Columns: flightdate, total_flights, delayed_flights, pct_delayed, avg_arr_delay_min, avg_dep_delay_min, total_delay_minutes

AUTO REFRESH YES: Views automatically refresh when underlying data changes



Sample Analytics Queries:

Top 10 Most Delayed Airlines

```
SELECT uniquecarrier, EXTRACT(month FROM flightdate) AS month, COUNT(*) AS flights,
ROUND(AVG(arrdelay), 2) AS avg_delay_min,
SUM(CASE WHEN isdelayed THEN 1 ELSE 0 END) AS delayed_count
FROM public.fact_flights_2008
GROUP BY uniquecarrier, month
ORDER BY avg_delay_min DESC
LIMIT 10;
```

On-Time Performance by Airport

```
SELECT origin, COUNT(
) AS departures, ROUND(AVG(depdelay), 2) AS avg_dep_delay_min, SUM(CASE WHEN depdelay > 15 THEN 1
ELSE 0 END)::float / COUNT() * 100 AS pct_delayed
FROM public.fact_flights_2008
GROUP BY origin
HAVING COUNT(*) >= 1000
ORDER BY pct_delayed DESC
LIMIT 10;
```

Delay Breakdown by Reason

```
SELECT SUM(carrierdelay) AS carrier_delay_total,
SUM(weatherdelay) AS weather_delay_total,
SUM(nasdelay) AS nas_delay_total,
SUM(securitydelay) AS security_delay_total,
SUM(lateaircraftdelay) AS late_aircraft_delay_total,
```

```
SUM(totaldelay) AS grand_total_delay  
FROM public.fact_flights_2008;
```

All queries run in under 1 second using materialized views for pre-computed aggregations.

Part 4: Real-Time Streaming Pipeline Overview

Part 4 implements a Lambda-based Kinesis consumer with streaming data quality validation, alert classification based on delay thresholds, and multi-channel notifications via SNS (email) and SQS (queue-based processing).

Lambda Kinesis Consumer Architecture

IAM Role Configuration

Role Name: airline-dev-flightevents-consumer-role

Policies Attached:

1. **CloudWatch Logs:** CreateLogGroup, CreateLogStream, PutLogEvents
2. **Kinesis:** GetRecords, GetShardIterator, DescribeStream, ListStreams
3. **S3:** PutObject (for quarantine bucket)(no bad records here purged during data cleaning, processing in earlier steps)
4. **SNS:** Publish (for alert notifications)
5. **SQS:** SendMessage (for alert queue)

Each policy follows principle of least privilege - Lambda only gets permissions it needs.

Lambda Consumer Python Code

File: scripts/python/lambda_kinesis_consumer.py

Core Functions:

1. **is_valid_event()** - Streaming DQ validation
 - o Required fields present and non-null
 - o distance > 0 (valid flight route)
 - o depdelay and arrdelay within reasonable bounds (-60 to 1440 minutes)
2. **classify_alert()** - Alert categorization
 - o SEVERE_DEPARTURE_DELAY: depdelay >= 60 minutes
 - o SEVERE_ARRIVAL_DELAY: arrdelay >= 60 minutes
 - o PERSISTENT_ROUTE_DELAY: both delays >= 30 minutes
3. **build_alert_payload()** - Structure alert data
 - o Includes alert_type, flight details, timestamp, message
4. **write_bad_events_to_s3()** - Quarantine bad events
 - o JSON Lines format (one event per line)
 - o Path: s3://.../streaming/quarantine/flightevents/
5. **publish_alert()** - Send notifications

- SNS for email
- SQS for queue-based processing

6. `lambda_handler()` - Main entry point

- Processes Kinesis batch
- Applies DQ and classification
- Publishes alerts
- Returns summary statistics

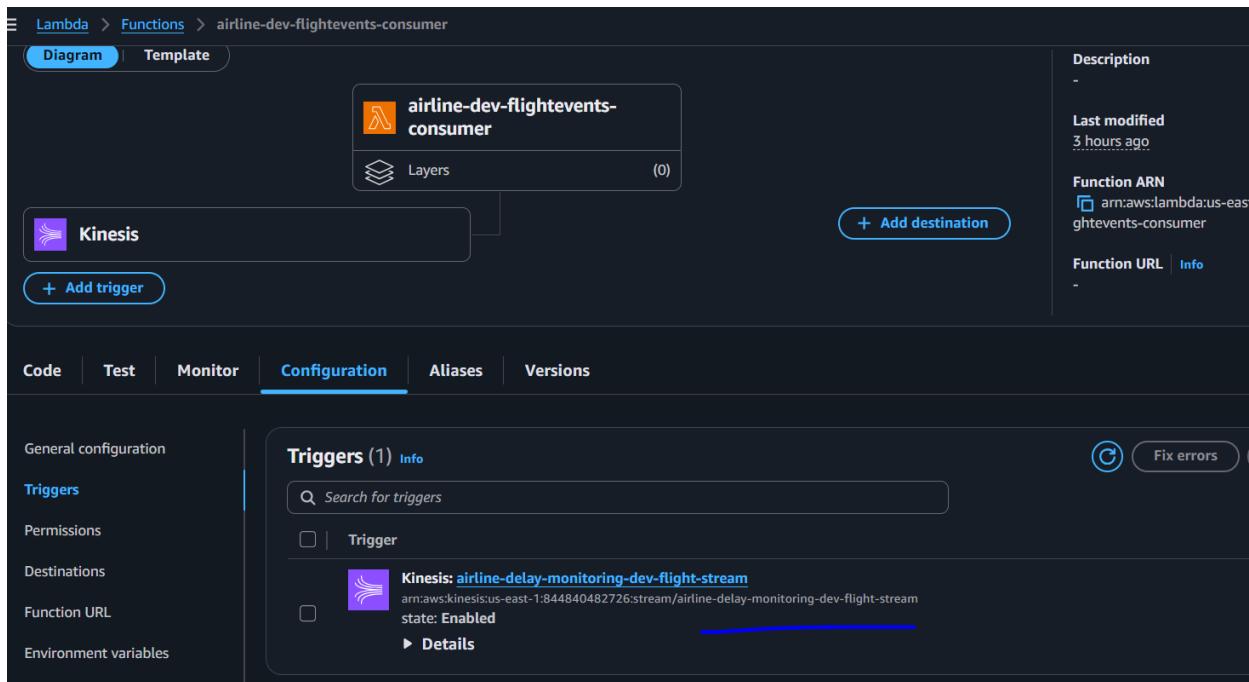
Event Source Mapping

Connects Kinesis stream to Lambda function:

Kinesis Stream → Event Source Mapping → Lambda Function

- Starting Position: LATEST (only new events)
- Batch Size: 100 records
- Enabled: true

Lambda is invoked automatically when events arrive in Kinesis.



Streaming Producer Script

File: scripts/python/simulate_flight_events_step2.py

Reads historical CSV from S3, converts to JSON, sends to Kinesis at controlled rate.

Usage:

python scripts/python/simulate_flight_events_step2.py

Deploy Lambda Consumer

Create Deployment Package

```
cd scripts/python  
Compress-Archive -Path .\lambda_kinesis_consumer.py `  
-DestinationPath ....\terraform\lambda_artifacts\lambda_kinesis_consumer.zip -Force
```

Apply Terraform

```
cd terraform  
terraform apply
```

SNS Email Subscription

1. Go to **SNS** → **Topics** → **airline-dev-flight-delay-alerts**
2. Click **Create subscription**
3. Protocol: **Email**, Endpoint: your email address
4. Click **Create subscription**
5. Confirm subscription by clicking AWS link in your email

Testing the Pipeline

Run Producer Script

```
python scripts/python/simulate_flight_events_step2.py
```

Sends 5000 flight events to Kinesis stream

Monitor Lambda Execution

```
aws logs tail /aws/lambda/airline-dev-flights-events-consumer --follow
```

Look for:

- Flight event logs
- ALERT: messages for delay thresholds met
- Batch complete summary

Verify Alerts

Check for:

- Email notifications from SNS
- Messages in SQS queue (AWS Console)
- Bad events in S3 quarantine folder

Primary Promotions Social Updates 49 new SA Breaking News — MSFT: ...

AWS Notifications 100	Flight delay alert: SEVERE_DEPARTURE_DELAY - {"alert_type": "SEVERE_DEPARTUR...}	10:32 P
AWS Notifications 100	Flight delay alert: PERSISTENT_ROUTE_DELAY - {"alert_type": "PERSISTENT_ROUTE_...}	10:32 P
AWS Notifications 100	Flight delay alert: PERSISTENT_ROUTE_DELAY - {"alert_type": "PERSISTENT_ROUTE_...}	10:32 P
AWS Notifications 25	Flight delay alert: SEVERE_DEPARTURE_DELAY - {"alert_type": "SEVERE_DEPARTUR...}	10:32 P
AWS Notifications 24	Flight delay alert: PERSISTENT_ROUTE_DELAY - {"alert_type": "PERSISTENT_ROUTE_...}	10:32 P

From Cloudwatch: log groups

```
ART RequestId: 95bf251b-c624-403a-91d0-4617b610ea59 Version: $LATEST
FO] 2025-12-01T03:32:26.539Z 95bf251b-c624-403a-91d0-4617b610ea59 Flight event: carrier=WN flightnum=1574
jin=PHL dest=PIT depdelay=74.0 arrdelay=62.0 flightdate=2008-01-03
FO] 2025-12-01T03:32:26.539Z 95bf251b-c624-403a-91d0-4617b610ea59 ALERT:
'alert_type": "SEVERE_DEPARTURE_DELAY", "flightdate": "2008-01-03", "uniquecarrier": "WN", "flightnum": 1574,
'jin": "PHL", "dest": "PIT", "depdelay": 74, "arrdelay": 62, "distance": 267, "generated_at_utc": "2025-12-
T03:32:26.539524Z", "message": "SEVERE_DEPARTURE_DELAY for WN 1574 PHL->PIT" }
FO] 2025-12-01T03:32:26.590Z 95bf251b-c624-403a-91d0-4617b610ea59 Flight event: carrier=WN flightnum=2200
jin=PHL dest=PIT depdelay=28.0 arrdelay=8.0 flightdate=2008-01-03
FO] 2025-12-01T03:32:26.590Z 95bf251b-c624-403a-91d0-4617b610ea59 Flight event: carrier=WN flightnum=2560
jin=PHL dest=PIT depdelay=10.0 arrdelay=5.0 flightdate=2008-01-03
FO] 2025-12-01T03:32:26.590Z 95bf251b-c624-403a-91d0-4617b610ea59 Flight event: carrier=WN flightnum=2877
jin=PHL dest=PIT depdelay=23.0 arrdelay=8.0 flightdate=2008-01-03
FO] 2025-12-01T03:32:26.590Z 95bf251b-c624-403a-91d0-4617b610ea59 Batch complete. Processed=4, BadDQ=0,
led=0
D RequestId: 95bf251b-c624-403a-91d0-4617b610ea59
PORT RequestId: 95bf251b-c624-403a-91d0-4617b610ea59 Duration: 71.49 ms Billed Duration: 72 ms Memory Size:
MB Max Memory Used: 88 MB
```

SQS console:

Amazon SQS > Queues > airline-dev-flight-delay-alerts-queue > Send and receive messages

Receive messages Info

Messages available: 1800

Polling duration: 30 Maximum message count: 10 Polling rate limit: 0 receives/second

Edit poll settings

Received message: 25a54dd6-6813-4846-963f-9952673d23f0

X

Body Attributes Details

Body: {"alert_type": "SEVERE_DEPARTURE_DELAY", "flightdate": "2008-01-03", "uniquecarrier": "WN", "flightnum": 422, "origin": "ISP", "dest": "MDW", "depdelay": 82.0, "arrdelay": 72.0, "distance": 765.0, "generated_at_utc": "2025-12-01T02:22:44.076811Z", "message": "SEVERE_DEPARTURE_DELAY for WN 422 ISP->MDW"}

Attributes: ID: 25a54dd6-6813-4846-963f-9952673d23f0

Details: Done

2025-11-30T21:22:05-05:00 307 bytes

Messages (10) Search messages

- 25a54dd6-6813-4846-963f-9952673d23f0
- ad10320e-1068-413c-ad25-0e05f18df9e7
- 5cc7ea6c-91eb-46e0-878d-18a6dfc02586
- 6d7e3a1a-5401-4954-b63f-acf697b55cd9

Conclusion:

A complete batch analytics stack is built: raw CSV flight data is ingested into S3, cataloged by AWS Glue, transformed via a 31-step Glue ETL job into partitioned Parquet, and loaded into an optimized Amazon Redshift fact table with materialized views for carrier, route, and daily performance analysis.

A fully integrated streaming stack delivered: simulated live flight events are published to Amazon Kinesis Data Streams, processed by an AWS Lambda consumer that enforces streaming data-quality rules, quarantines bad records to S3, and emits severity-based alerts to Amazon SNS (email) and Amazon SQS (downstream consumers)