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_sqs/
├── 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

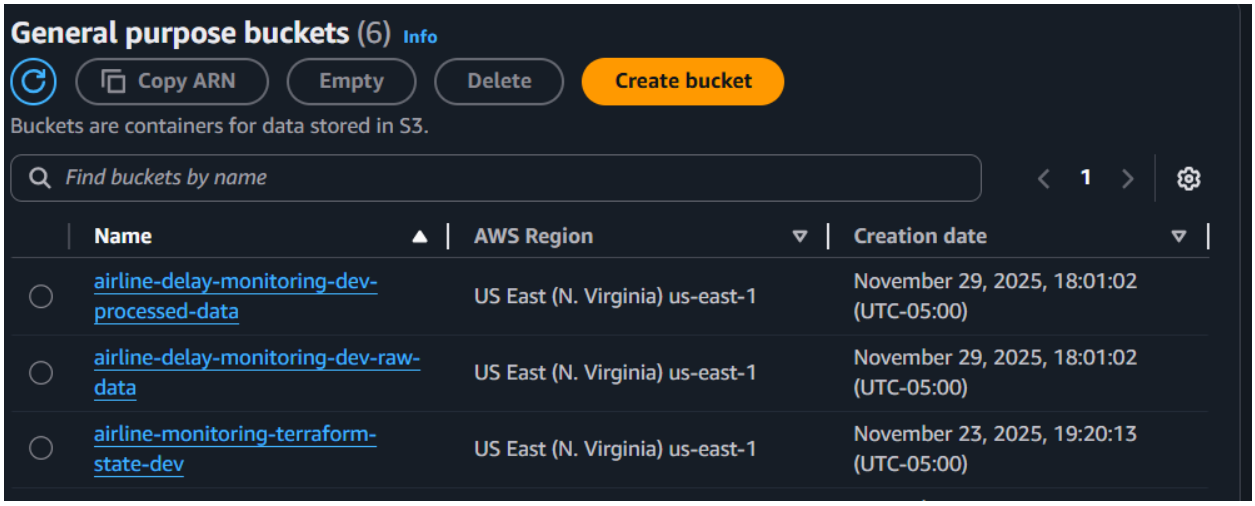
```
terraform {
required_version = ">= 1.0"

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

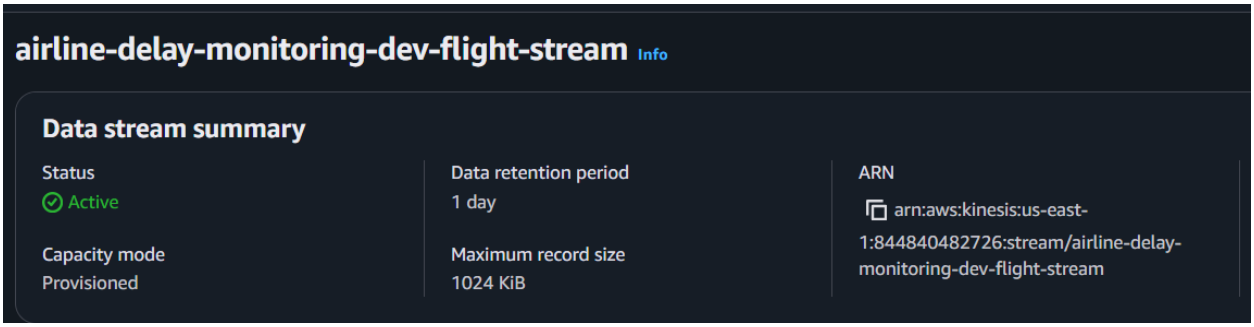
File: terraform/main.tf - Orchestrates all modules and connects infrastructure components

Key modules deployed:

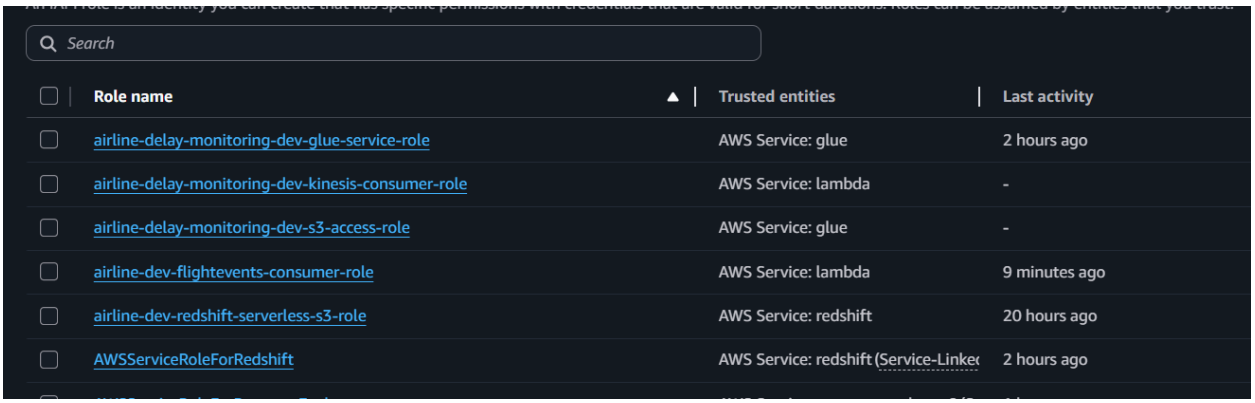
- S3 Buckets (raw & processed data)



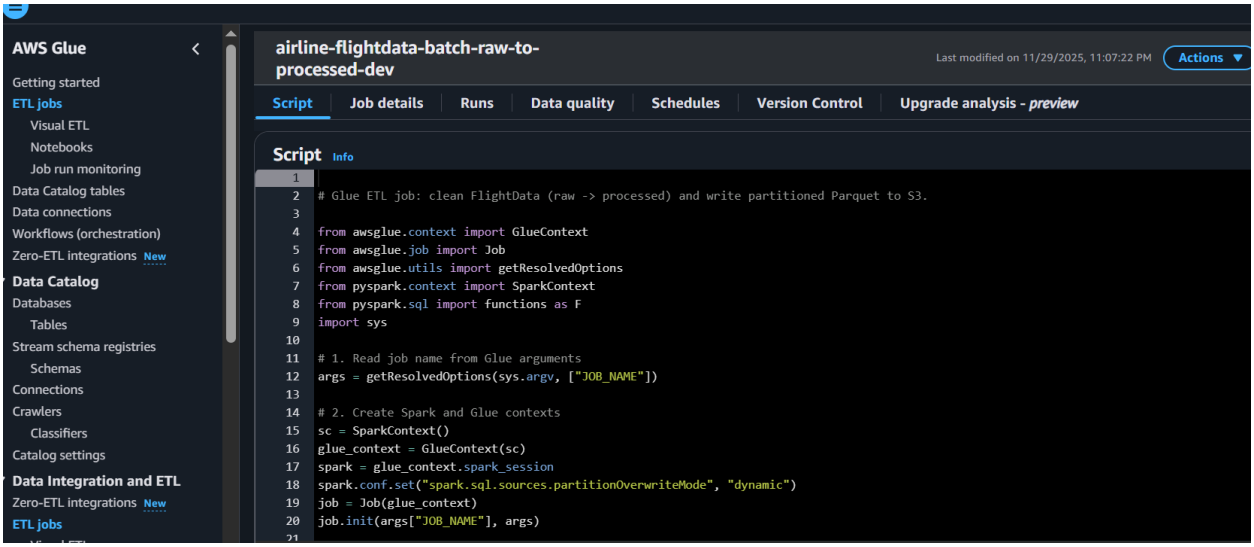
- Kinesis Stream (flight events)

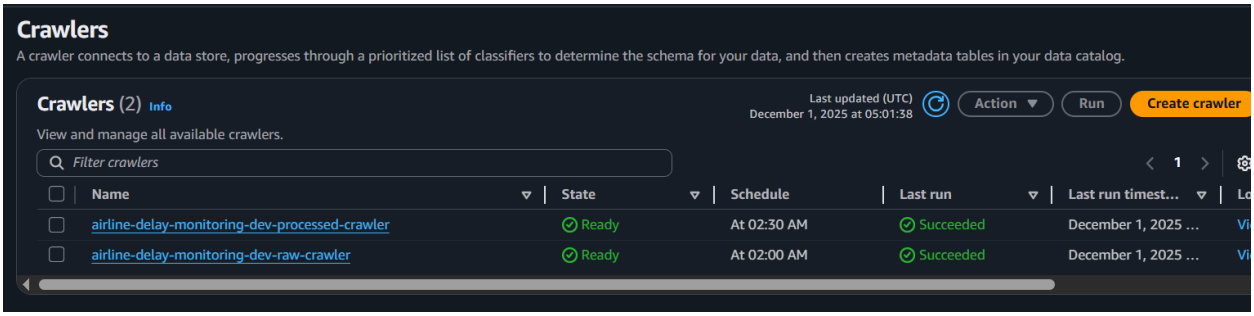


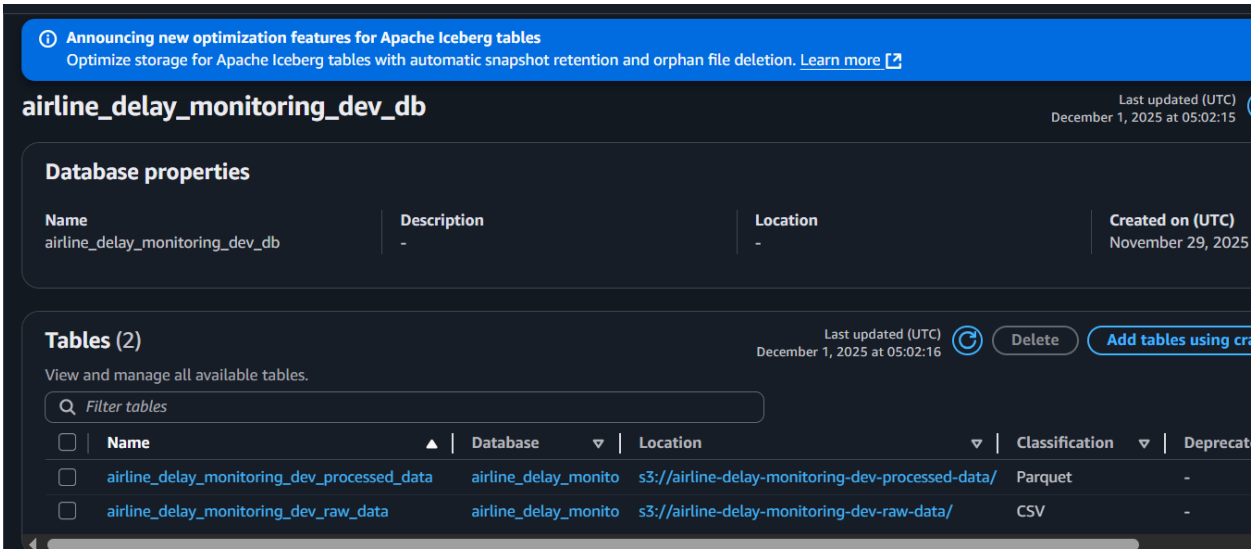
- IAM Roles (Glue, Redshift, Lambda)



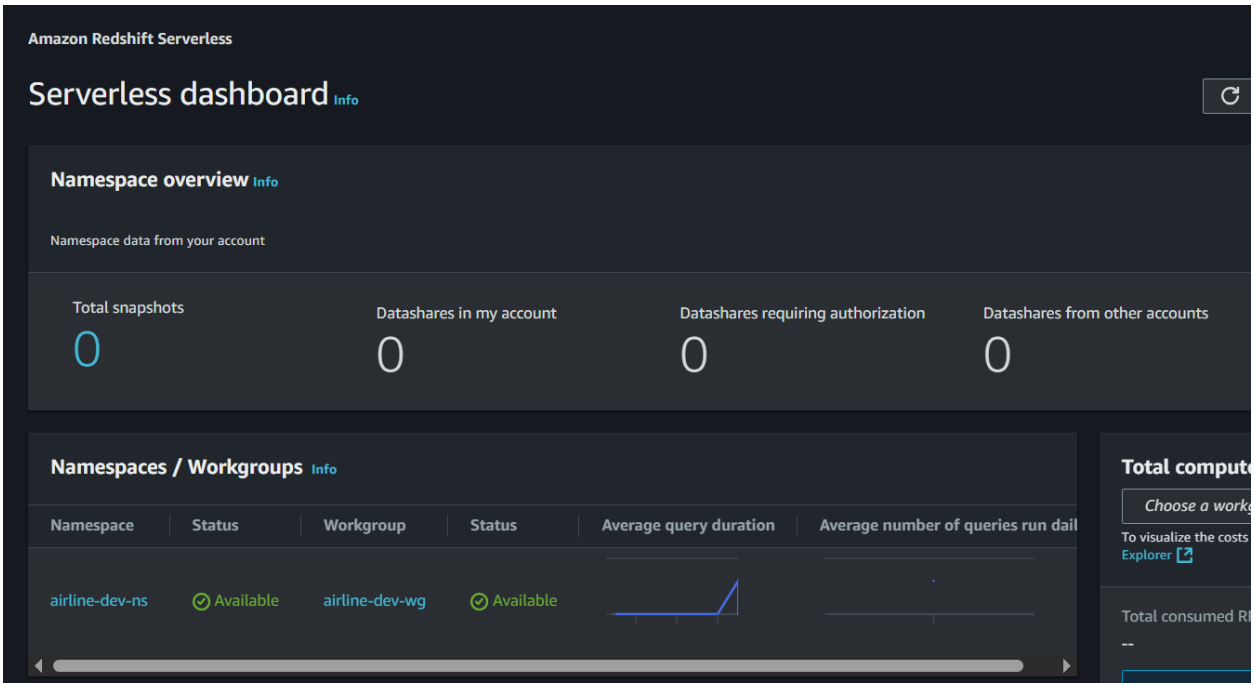
- Glue Infrastructure (crawler & jobs)

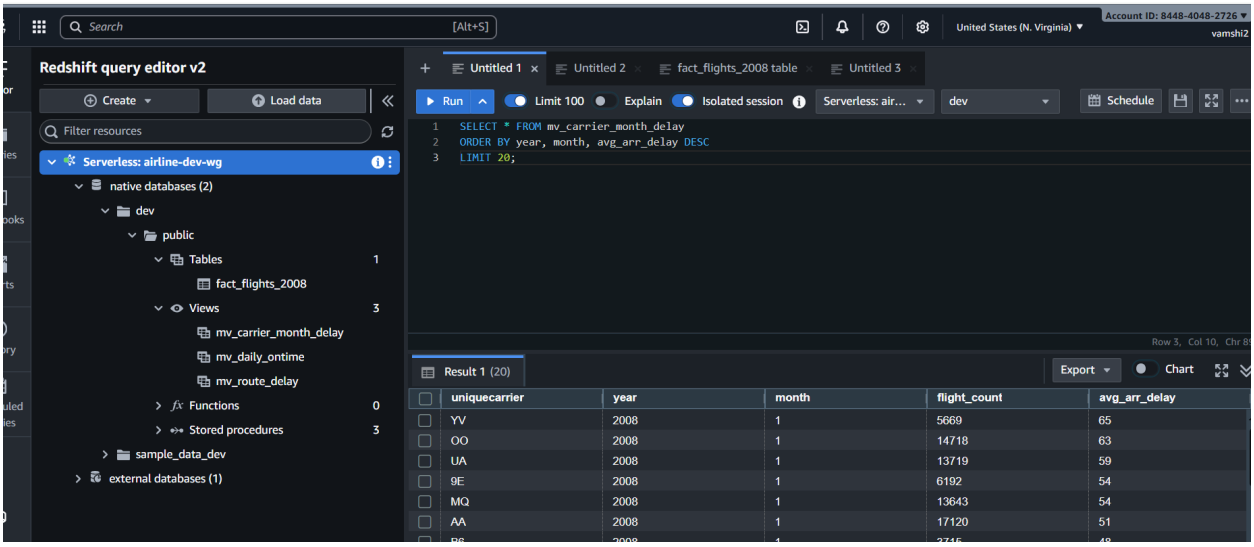




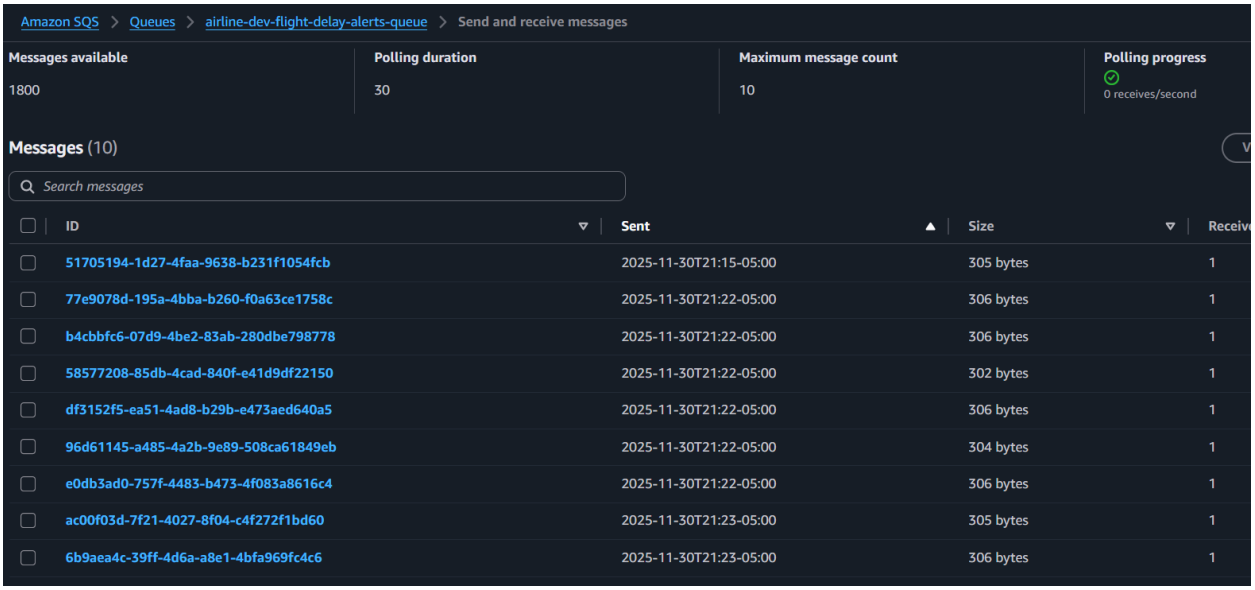


- Redshift Serverless (analytics warehouse)

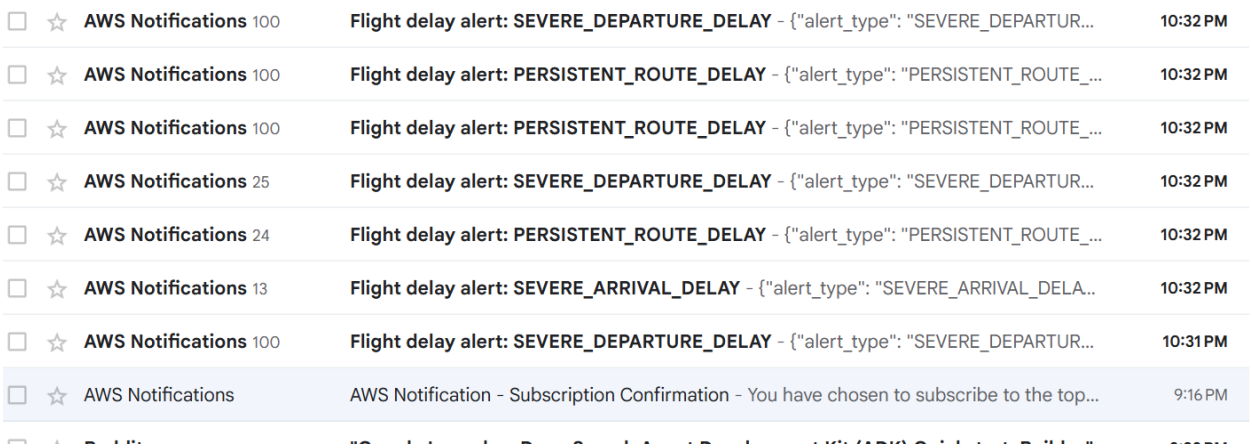




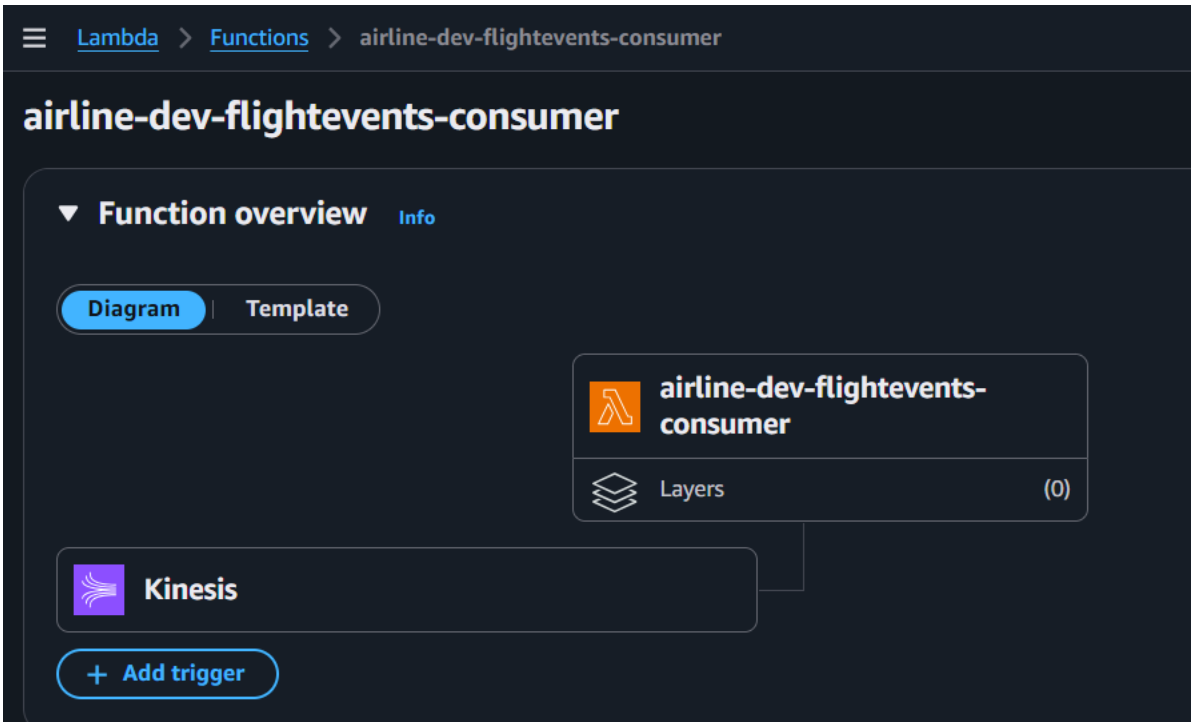
- SNS/SQS (alerts)



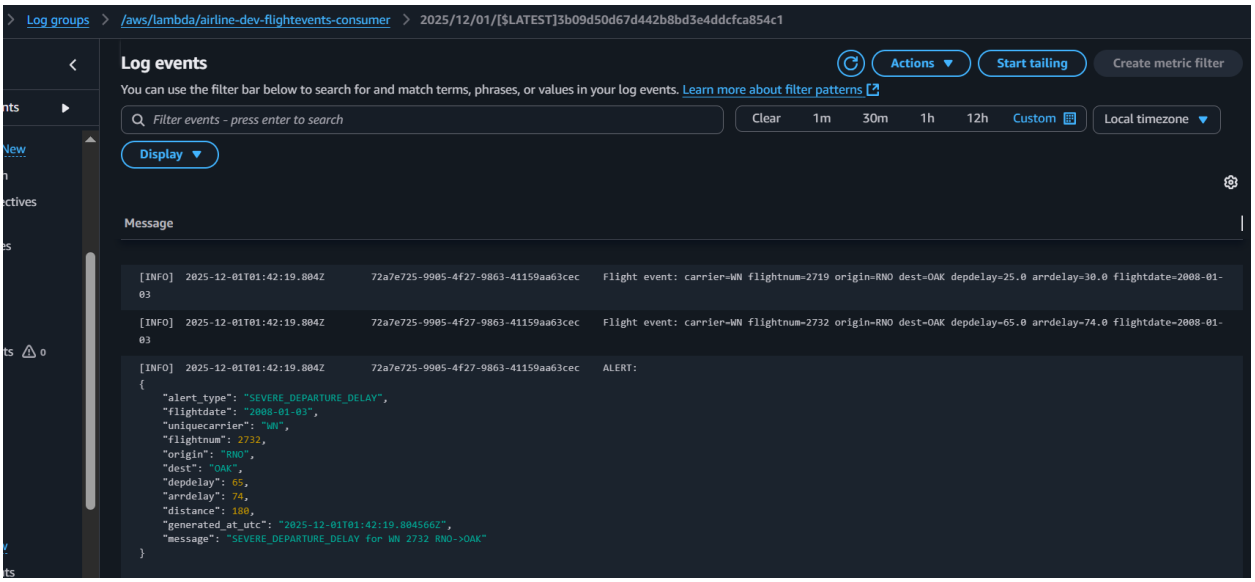
Email notification: SNS subscription

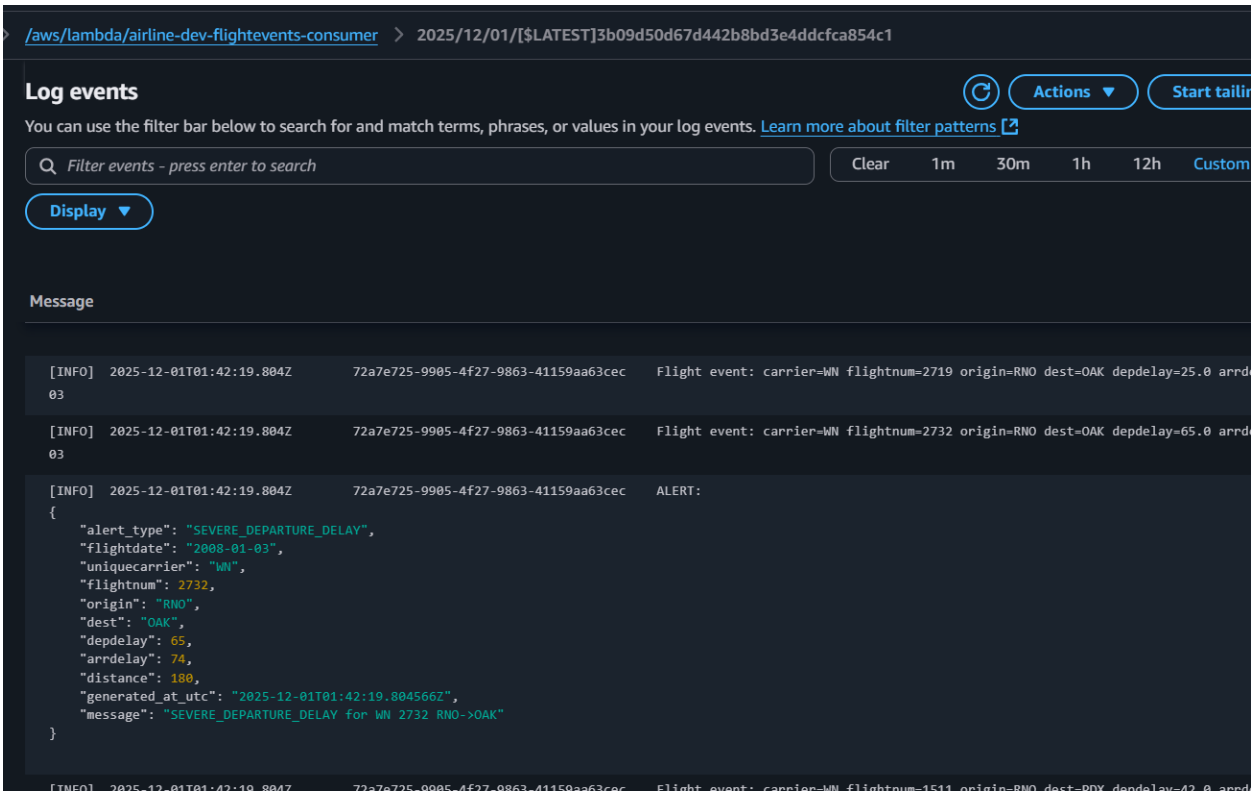


- Lambda Consumer (streaming processor):



- Cloudwatch:





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)
crsarrtime	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

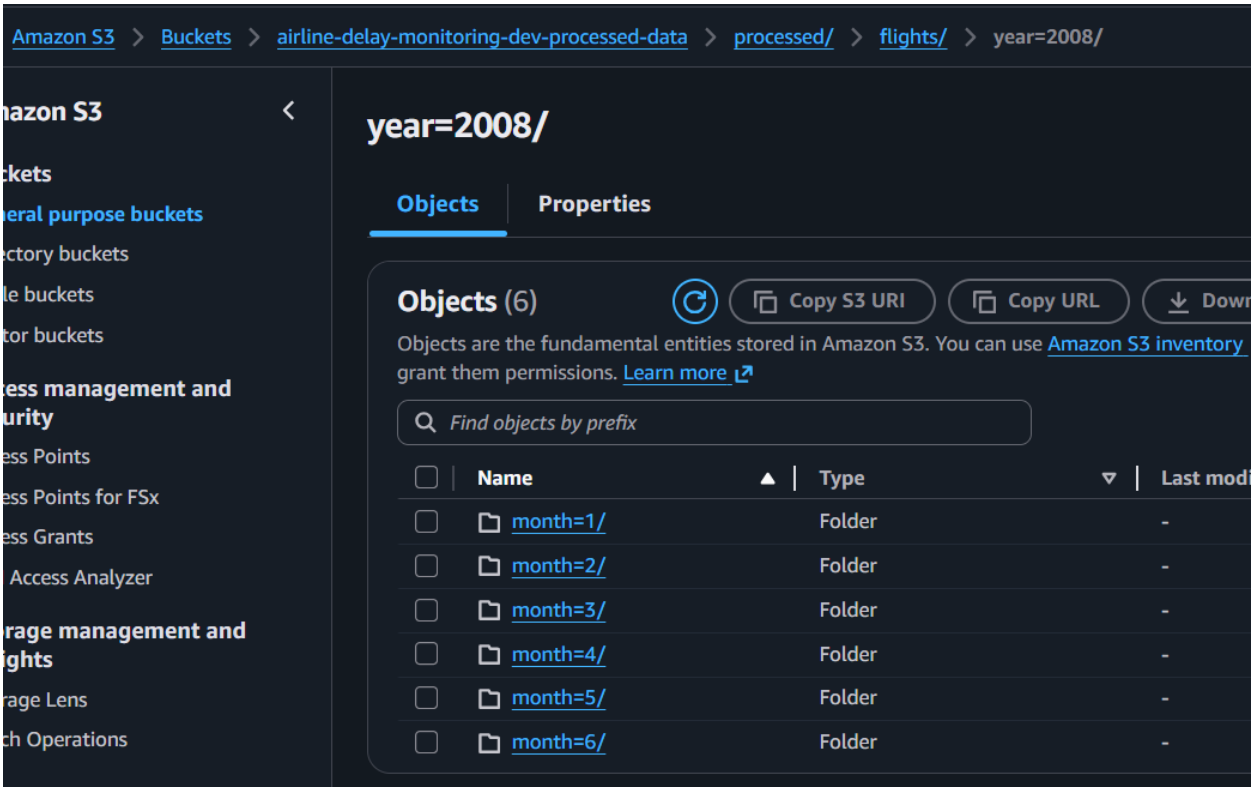
Monitor Execution

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/



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, crsarrtime, actualelapsedtime, 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

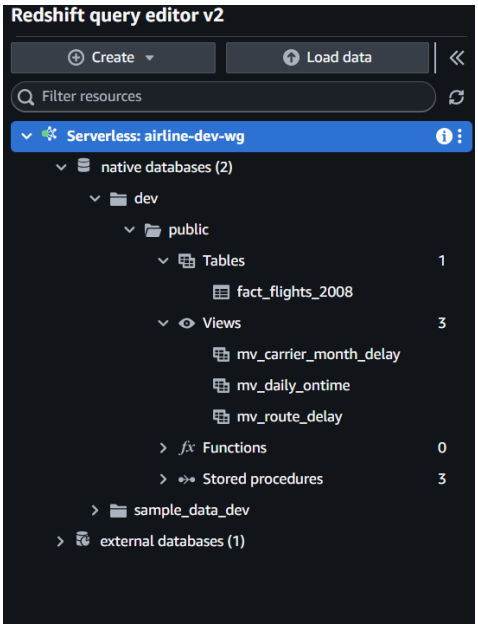
Columns: origin, dest, flight_count, avg_arr_delay_min, avg_dep_delay_min, pct_delayed, avg_distance_miles

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
    - Required fields present and non-null
    - distance > 0 (valid flight route)
    - depdelay and arrdelay within reasonable bounds (-60 to 1440 minutes)
2. **classify_alert()** - Alert categorization
    - SEVERE_DEPARTURE_DELAY: depdelay >= 60 minutes
    - SEVERE_ARRIVAL_DELAY: arrdelay >= 60 minutes
    - PERSISTENT_ROUTE_DELAY: both delays >= 30 minutes
3. **build_alert_payload()** - Structure alert data
    - Includes alert_type, flight details, timestamp, message
4. **write_bad_events_to_s3()** - Quarantine bad events
    - JSON Lines format (one event per line)
    - 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

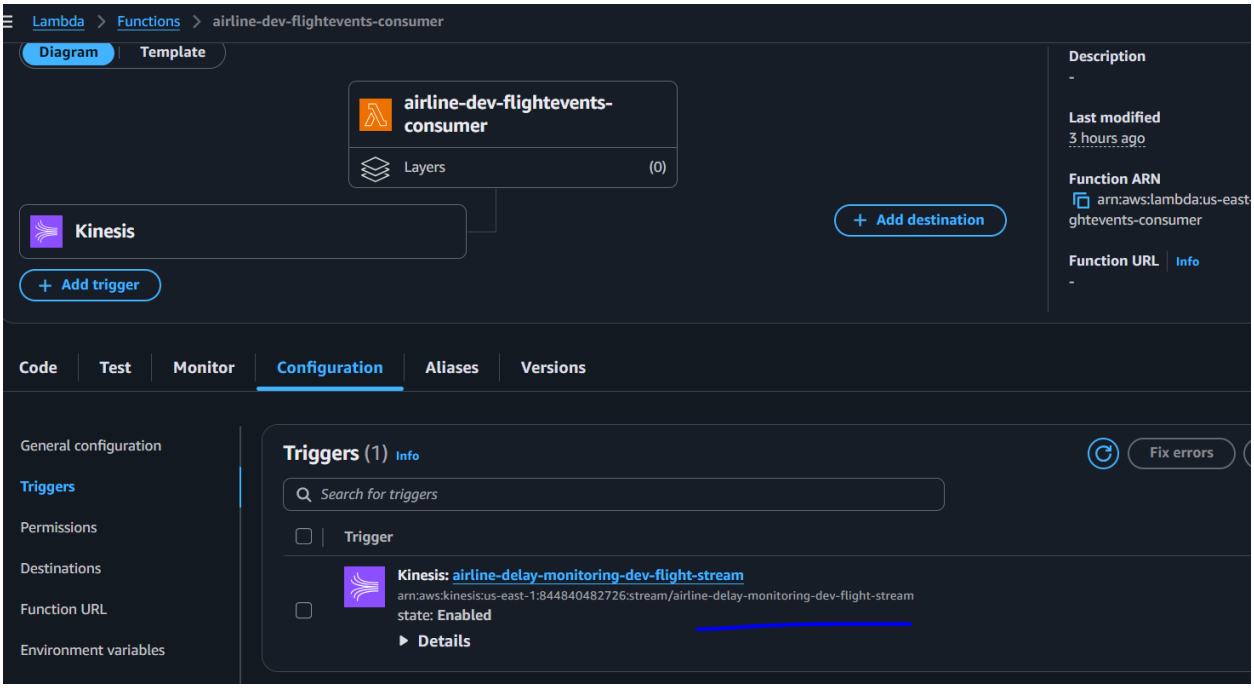
Event Source Mapping

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-flightevents-consumer --follow

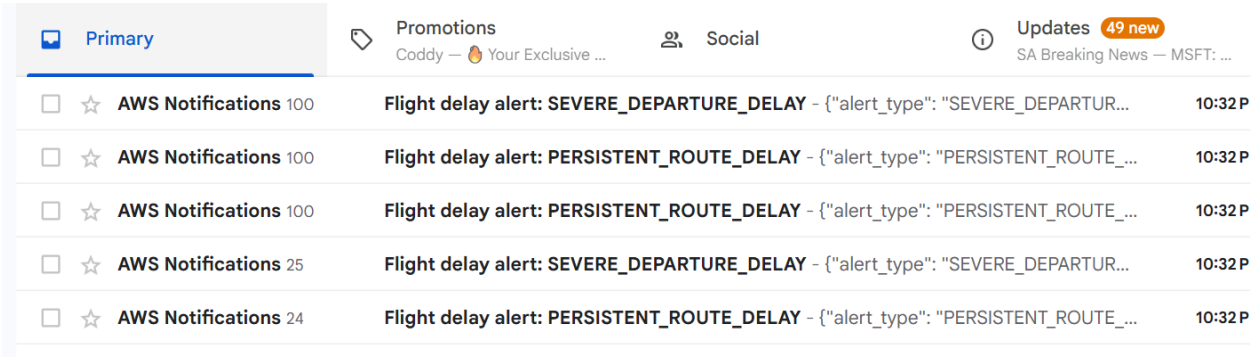
Look for:

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

Verify Alerts

Check for:

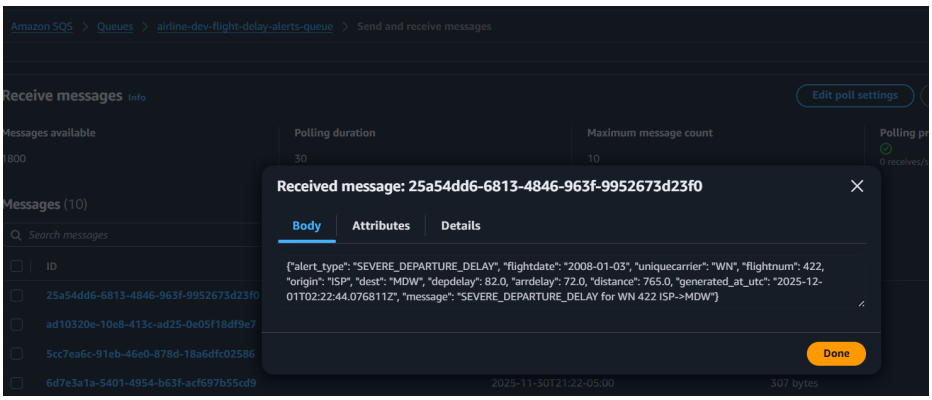
- Email notifications from SNS
- Messages in SQS queue (AWS Console)
- Bad events in S3 quarantine folder



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
gin=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,
rigin": "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
gin=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
gin=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
gin=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:



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)