Mini Project: Cloud-Based Big Data Analytics Pipeline with AWS and PySpark

1. Introduction

1.1 Project Overview

The primary object of this mini project is to provide a robust solution for building an end-to-end Data Pipeline to collect and process Raw Data using Distributed Computing technologies, store the processed data in a Cloud Storage solution and generate Reporting/Visualisation tools to extract analytical insights from the processed data. The Proposed pipeline is designed such that it will be capable of processing large datasets in a timely manner and demonstrates some of the key considerations when implementing real-world practices used within the Data Engineering process, such as: Automation, Logging, Monitoring and Ethical Considerations. This mini project focuses on designing and implementing a cloud-based big data analytics pipeline using Amazon Web Services (AWS) and Apache PySpark.

1.2 Dataset Description

For this project, the dataset used is referred to as the **Airline Delay Cause Dataset**. This dataset represents thousands of airline flights operations over multiple years; it provides a comprehensive overview of the quantity of flights that arrived on time, delays, cancellations and the reason(s) for the delay.

Key Characteristics of the Dataset

- A large dataset comprised of large numbers of records/rows and attributes over multiple years- ideal for use in Big Data analysis.
- Multiple Data Types comprising of both Numeric and Categorical Attribute Types.
- Possibility of performing Temporal, Geographical, and Performance Analysis on data.

Important Columns

- year, month
- airport, airport_name
- carrier, carrier_name
- arr_flights - *Total number of arrival flights*
- arr_del15 - *flights delayed more than 15 minutes*
- arr_cancelled, arr_diverted -*flights cancelled and diverted*
- Delay cause metrics:
    - carrier_ct - *Carrier-related delays*
    - weather_ct - *Weather-related delays*
    - nas_ct - *Air Traffic Control / National Airspace System delays*
    - late_aircraft_ct - *Delays caused by late-arriving aircraft*

This dataset is well-suited for deriving meaningful insights such as:

- Monthly and yearly flight trends
- Airport-level and airline-level performance comparison
- Delay cause analysis
- Operational reliability assessment

2. Dataset Selection

2.1 Rationale for Dataset Choice

The airline dataset is a good representation of how big data can be used for analysis and decision-making. With the sheer size of data generated every day, airlines will find this dataset useful due to its capacity for distributed processing (in using cloud-based technologies) to manage that amount of data.

The dataset satisfies the project requirements because:

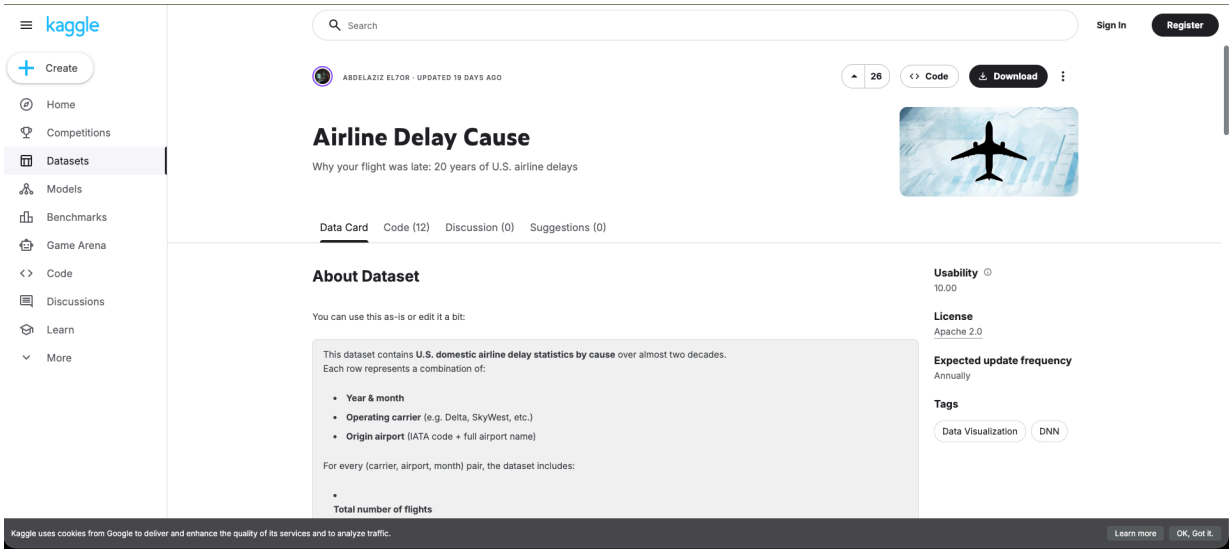
- Sizeable enough for distributed processing.
- Diverse enough to allow for transformation and aggregation of many different attribute types.
- Allows for numerous analytical perspectives (e.g., by time, location, performance).

2.2 Ethical Considerations in Dataset Usage

While working with real-world datasets, ethical concerns must be considered:

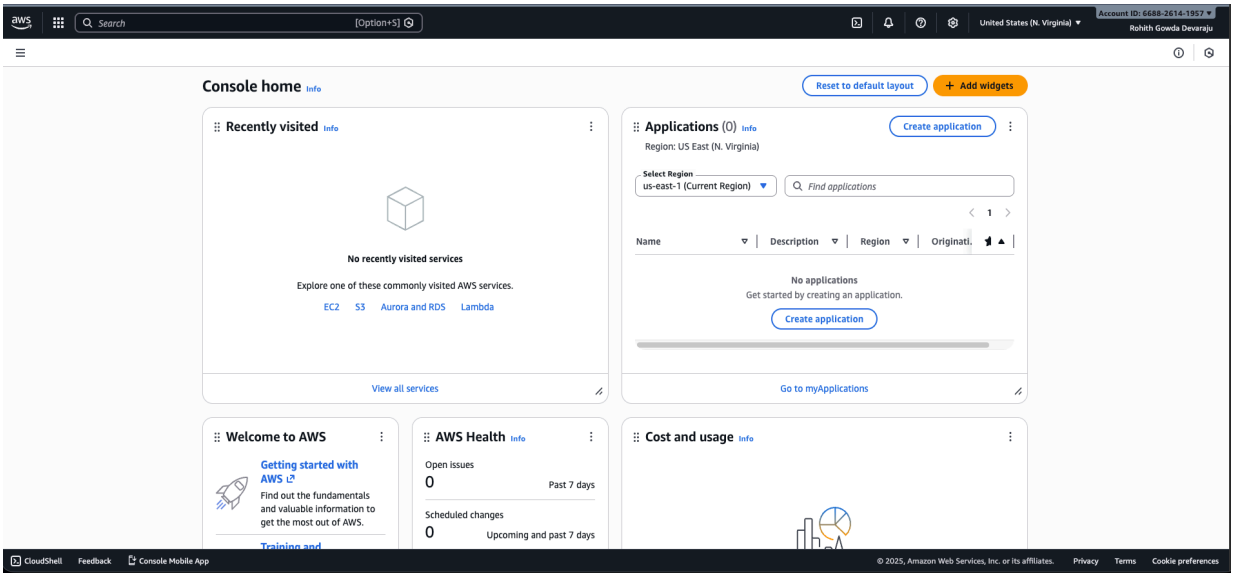
- Bias: It is present in many datasets. It can occur if a dataset has more data for specific airlines or airport. This would then influence the results of an analysis or the type of machine-learning model being used.
- Privacy: This aspects would have also been compromised as the dataset does not contain any personally identifiable information (PII), thus reducing the risk of privacy infringements.
- Risks: When using aggregated metrics, it is important to be aware of the potential pitfalls when interpreting them to avoid drawing false conclusions.

These considerations were acknowledged throughout the analysis process.



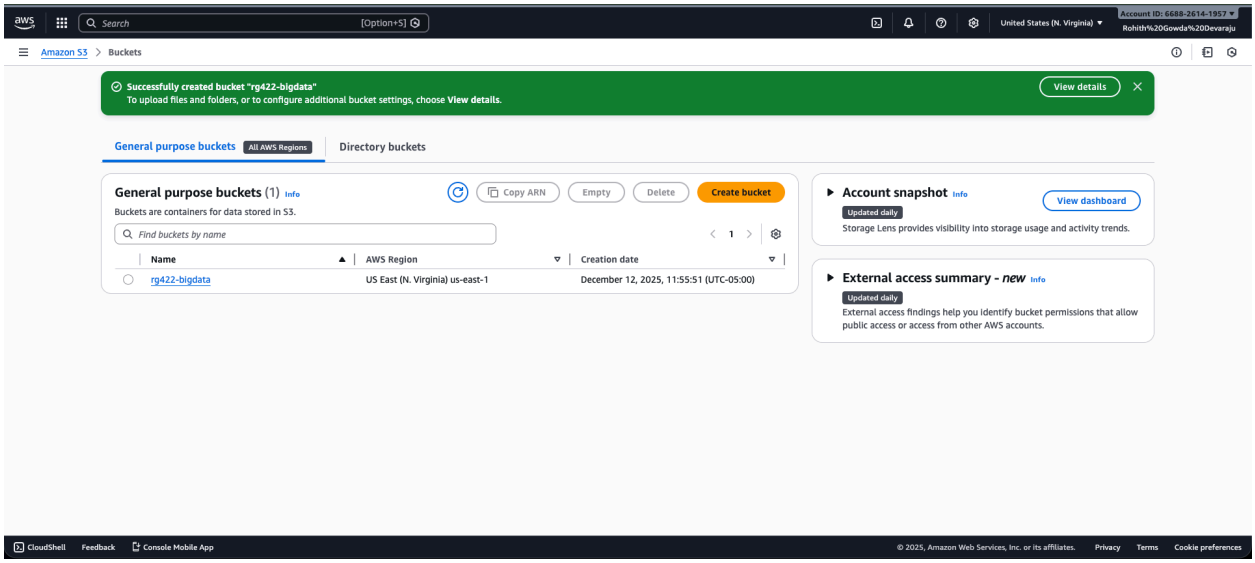
3. Environment Setup

In this section, you will establish the infrastructure and Distributed Processing Environment necessary to execute the Data Pipeline.



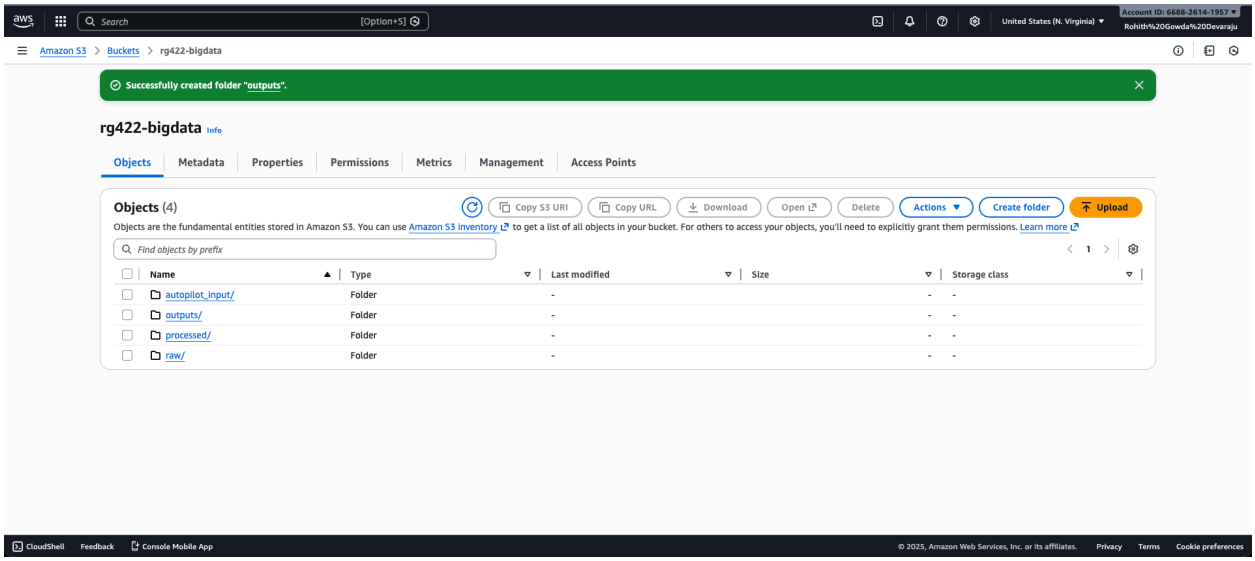
3.1.1 S3 Bucket Creation

For this project Amazon S3 was utilized as the main cloud storage solution, and provides secure, highly scalable Object Storage that is ideal for storing Large Datasets.



Bucket Structure

A Dedicated S3 Bucket has been created to store all the data pertaining to this project. The S3 Bucket has been Organized into Several Different Folders which allow for easy separation of RAW's, PROCESSED DATA, and LOGS from each other.



The primary goals of this Folder Organization are to Enable:

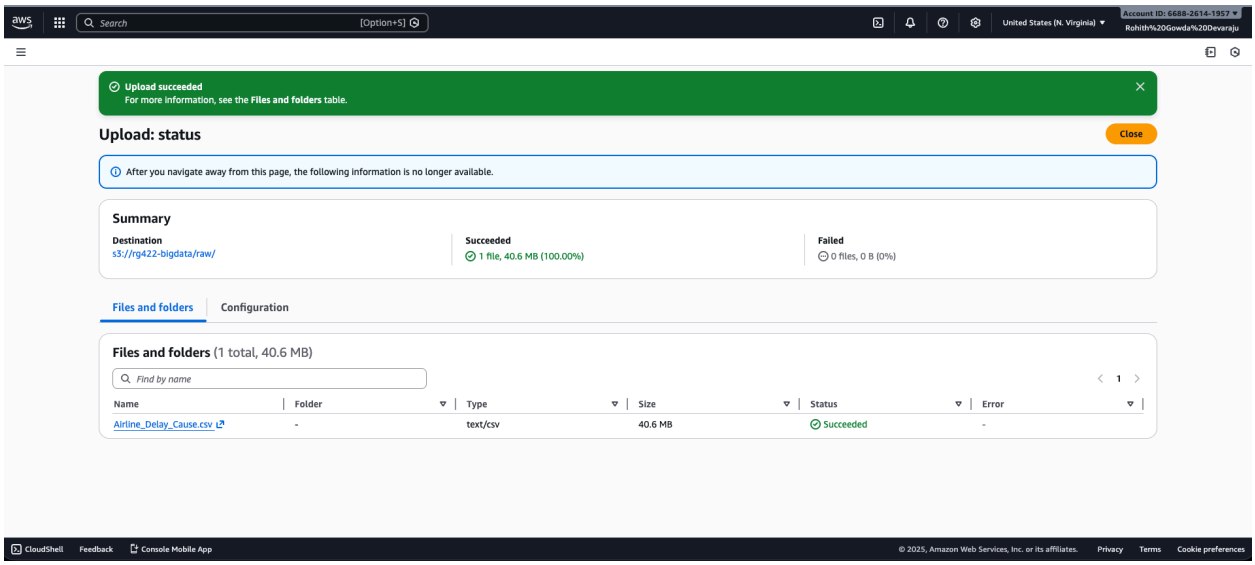
- Clear separation of raw and processed data
- Easier debugging and monitoring
- Better organization for automation and analytics

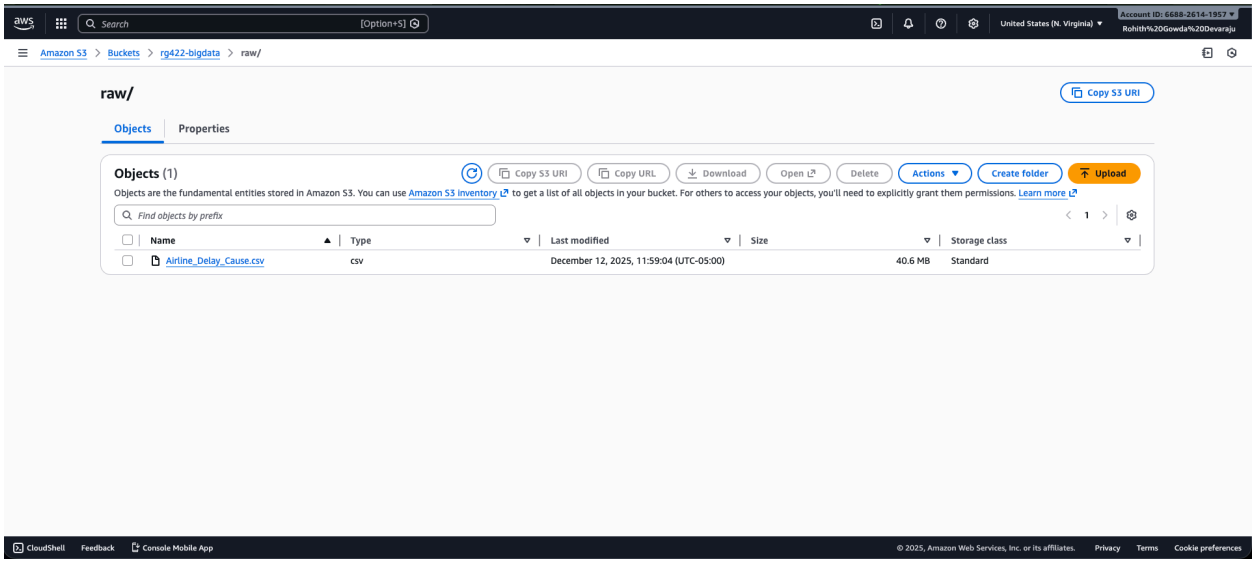
3.1.2 Uploading Raw Dataset to S3

The raw airline dataset was uploaded to the *raw/* folder within the S3 bucket using the AWS Management Console and AWS CLI.

This step ensures:

- Centralized cloud storage
- Easy access by PySpark jobs running on EC2
- Scalability for future dataset expansion





3.2 Linux Environment with PySpark

A Linux-based environment was set up using an AWS EC2 instance to enable distributed data processing with PySpark.

3.2.1 EC2 Instance Setup

An EC2 instance running **Ubuntu Linux** was launched with appropriate compute and storage resources. The instance was assigned an IAM role to securely access AWS services such as S3 and SNS without hard-coding credentials.

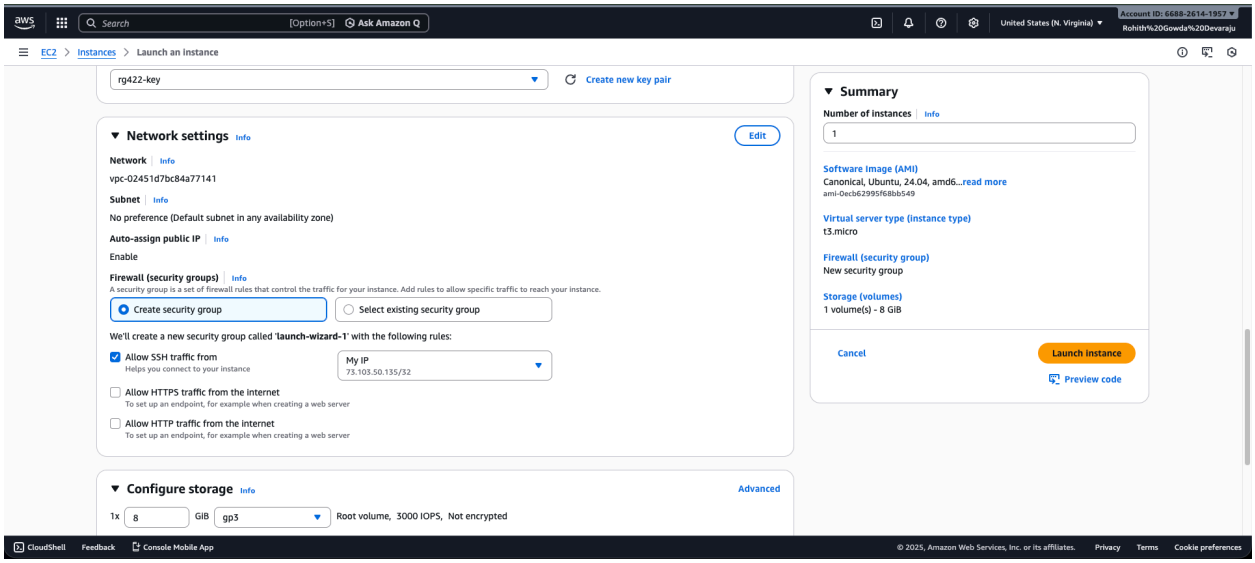
An EC2 instance was launched with the following configuration:

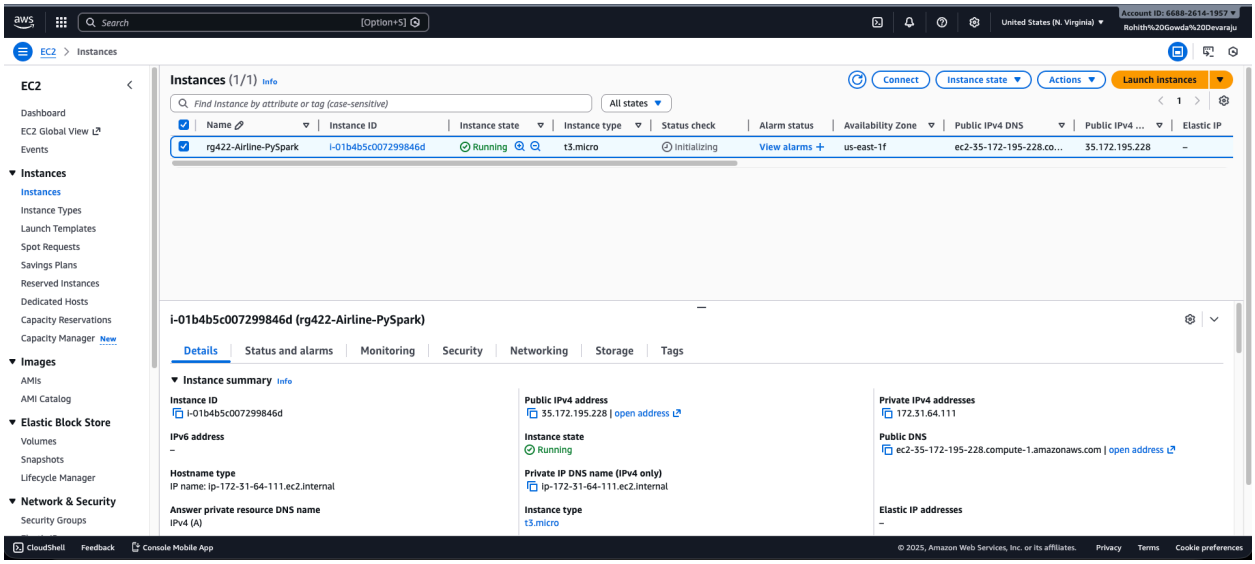
- **Instance type:** t3.micro
- **Operating system:** Ubuntu Linux
- **Instance name:** rg422-Airline-Pyspark

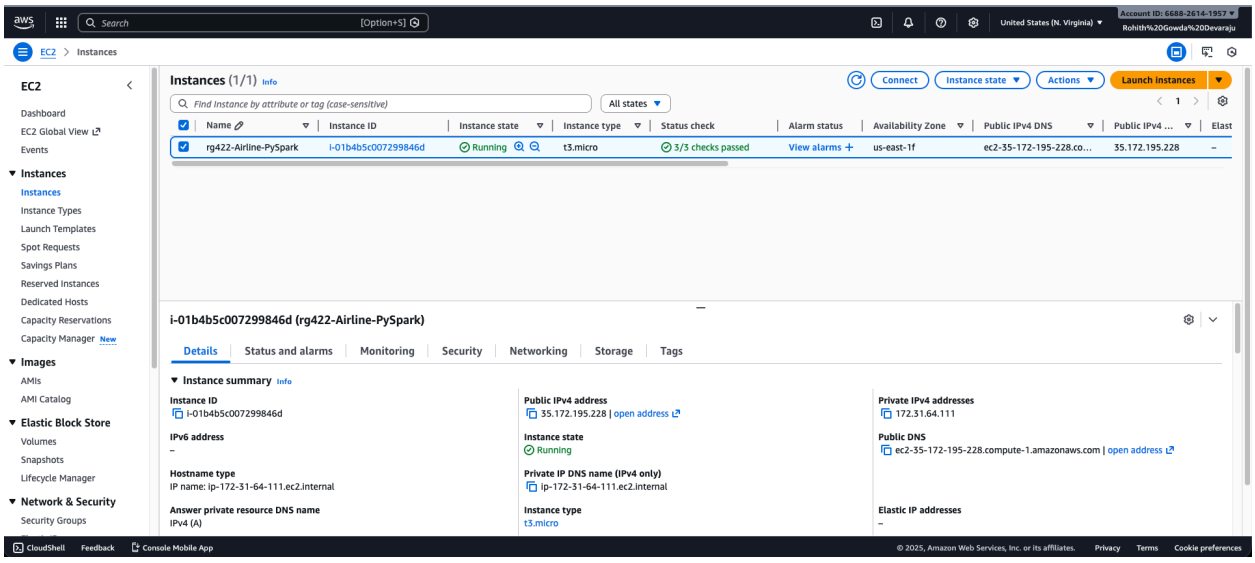
The t3.micro instance was selected because it falls under the AWS Free Tier, making it cost-effective while still suitable for running PySpark jobs for this project.

Using EC2 provides:

- Scalability for large data processing tasks
- Tight integration with AWS storage and analytics services
- Real-world experience with cloud-based data engineering workflows


Search [Option+S] Ask Amazon Q
United States (N. Virginia)
Account ID: 6688-2614-1957
Rohith%20Gowda%20Devaraju
EC2 > Instances > Launch an instance
rg422-key
Create new key pair
Network settings Info
Edit
Network Info
vpc-02451d7bc84a77141
Subnet Info
No preference (Default subnet in any availability zone)
Auto-assign public IP Info
Enable
Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Create security group
Select existing security group
We'll create a new security group called 'launch-wizard-1' with the following rules:
Allow SSH traffic from
Helps you connect to your instance
My IP
73.103.50.135/32
Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server
Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server
Configure storage Info
Advanced
1x 8 GiB gp3 Root volume, 3000 IOPS, Not encrypted
Summary
Number of instances Info
1
Software Image (AMI)
Canonical, Ubuntu, 24.04, amd6...read more
ami-0ecb62995f68bb549
Virtual server type (instance type)
t3.micro
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Cancel
Launch instance
Preview code
CloudShell Feedback Console Mobile App
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences


Search [Option+S]
United States (N. Virginia)
Account ID: 6688-2614-1957
Rohith%20Gowda%20Devaraju
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Capacity Manager New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Instances (1/1) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
Elastic IP
rg422-Airline-PySpark
i-01b4b5c007299846d
Running
t3.micro
Initializing
View alarms
us-east-1f
ec2-35-172-195-228.co...
35.172.195.228
–
i-01b4b5c007299846d (rg422-Airline-PySpark)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-01b4b5c007299846d
IPv6 address
–
Hostname type
IP name: ip-172-31-64-111.ec2.internal
Answer private resource DNS name
IPv4 (A)
Public IPv4 address
35.172.195.228 | open address
Instance state
Running
Private IP DNS name (IPv4 only)
ip-172-31-64-111.ec2.internal
Instance type
t3.micro
Private IPv4 addresses
172.31.64.111
Public DNS
ec2-35-172-195-228.compute-1.amazonaws.com | open address
Elastic IP addresses
–
CloudShell Feedback Console Mobile App
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences


Search [Option+S]
United States (N. Virginia)
Account ID: 6688-2614-1957
Rohith%20Gowda%20Devaraju
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Capacity Manager New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Instances (1/1) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
Elast
rg422-Airline-PySpark
i-01b4b5c007299846d
Running
t3.micro
3/3 checks passed
View alarms
us-east-1f
ec2-35-172-195-228.co...
35.172.195.228
–
i-01b4b5c007299846d (rg422-Airline-PySpark)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-01b4b5c007299846d
IPv6 address
–
Hostname type
IP name: ip-172-31-64-111.ec2.internal
Answer private resource DNS name
IPv4 (A)
Public IPv4 address
35.172.195.228 | open address
Instance state
Running
Private IP DNS name (IPv4 only)
ip-172-31-64-111.ec2.internal
Instance type
t3.micro
Private IPv4 addresses
172.31.64.111
Public DNS
ec2-35-172-195-228.compute-1.amazonaws.com | open address
Elastic IP addresses
–
CloudShell Feedback Console Mobile App
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

3.2.2 IAM Role Creation and Attachment

To securely access AWS resources from the EC2 instance without embedding credentials in code, an **IAM role** was created.

- **IAM Role name:** rg422-EC2-S3-Role

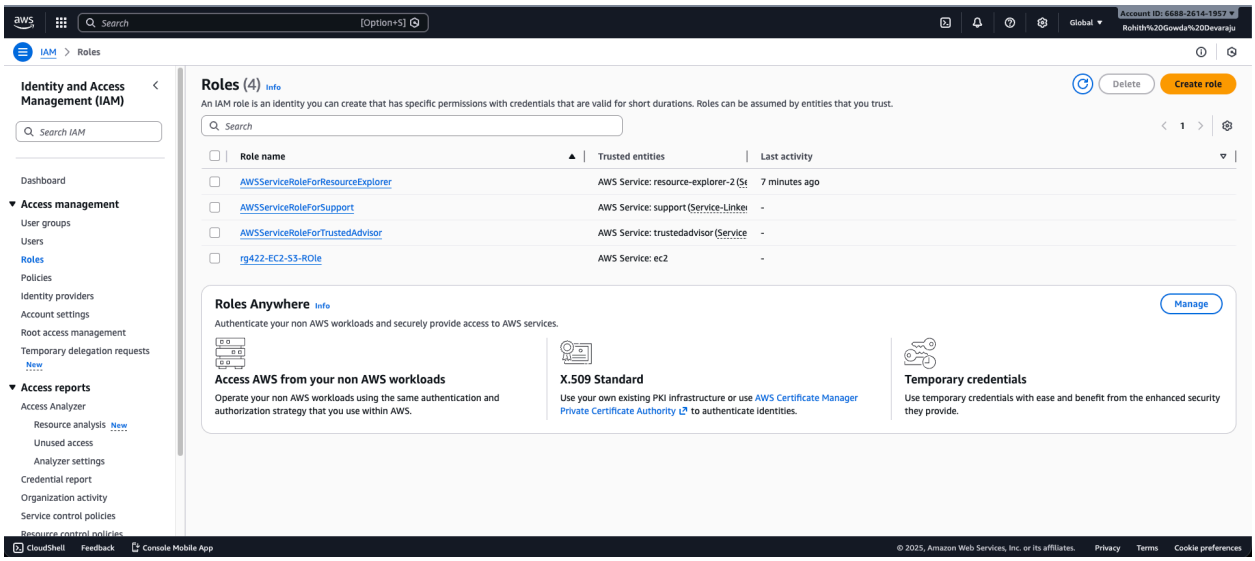
This role was configured with permissions to:

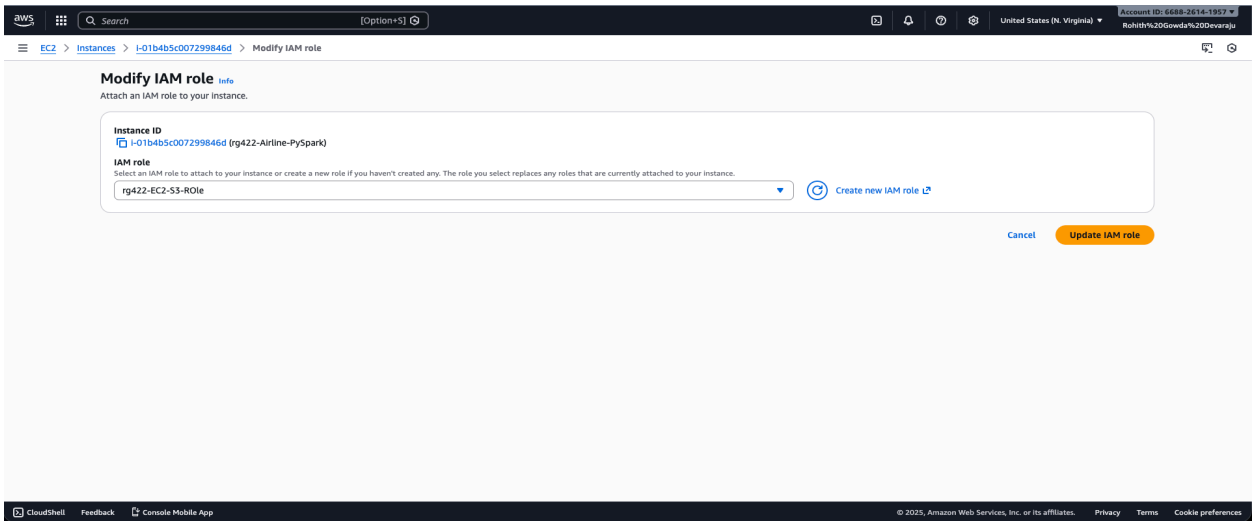
- Read and write data to Amazon S3
- Publish notifications to Amazon SNS

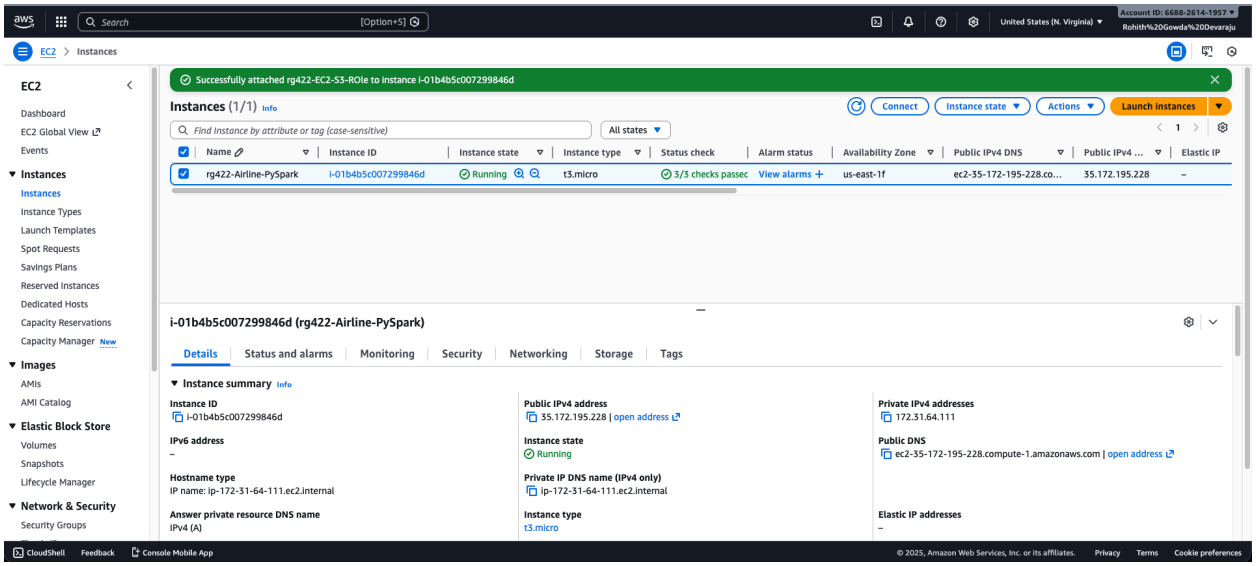
The IAM role was then **attached to the EC2 instance**, enabling secure and credential-free access to AWS services from within the instance.

This approach improves:

- Security (no hard-coded access keys)
- Maintainability
- Compliance with AWS best practices





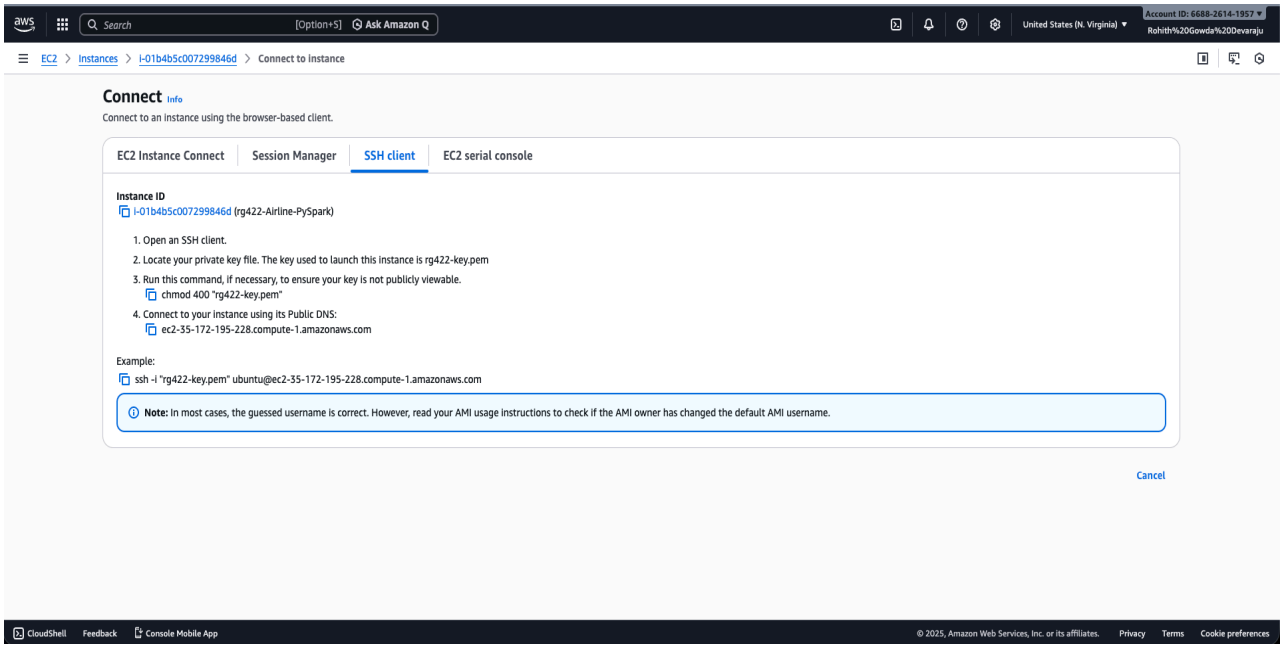


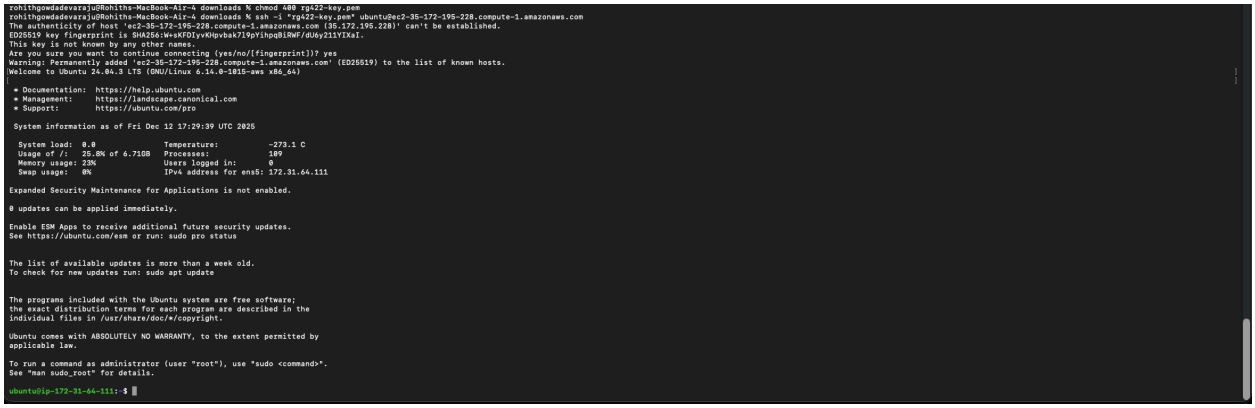
3.2.3 Connecting to EC2 via SSH

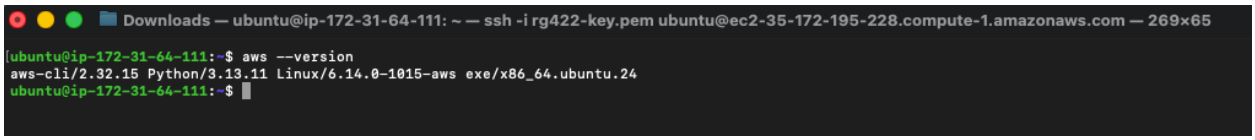
With the use of an SSH client from the user's local machine, the EC2 instance was able to establish a local connection. With the use of SSH connectivity, it allows for command-line interaction with the Linux environment for the purpose of software installation, running of PySpark jobs, and automation script management.

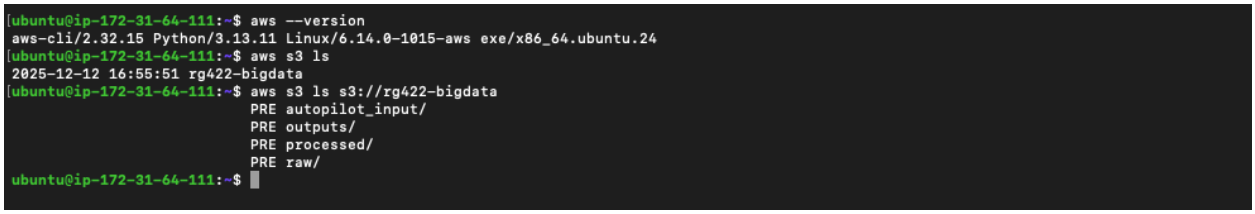
This process allows for:

- Management of the EC2 Instance remotely.
- Execution of Spark jobs.
- Monitoring of pipeline executions.







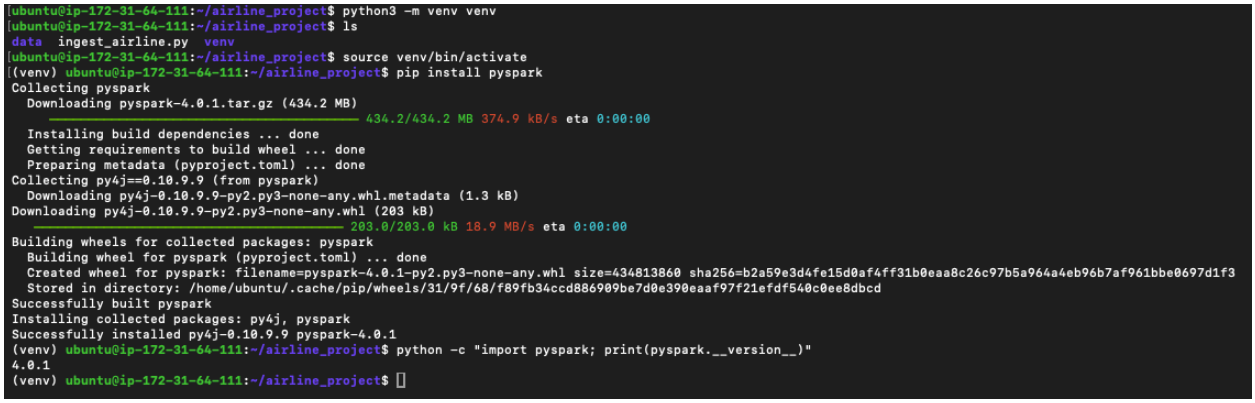


3.2.4 Installation of Java and PySpark

Apache Spark requires Java to run. The following installations were performed on the EC2 instance:

Apache Spark and PySpark Installation

- Apache Spark (version 3.5.x) was installed
- PySpark was installed within a Python virtual environment
- Spark installation was verified using:



Java Installation

- OpenJDK was installed to satisfy Spark's runtime requirements
- Java installation was verified using:

3.2.5 AWS CLI Configuration

The EC2 instance will be able to access and perform programmatically on Amazon S3 and other AWS services through the use of the AWS Command Line Interface. The EC2 instance's IAM role allows it to authenticate without the need for access keys. In this case, the AWS CLI automatically authenticates with the instance's IAM role.

As a result, you will be able to:

- Upload and download files from S3
- Log pipeline output files in S3
- Automate the management of your Data Workflows via the CL

4. Data Pipeline Tasks

Task 1: Data Ingestion from S3

Objective

This assignment's aim was to ingest unstructured data, which resides in Amazon S3 into a processing environment using PySpark for future applications to format the data and perform analysis.

Implementation

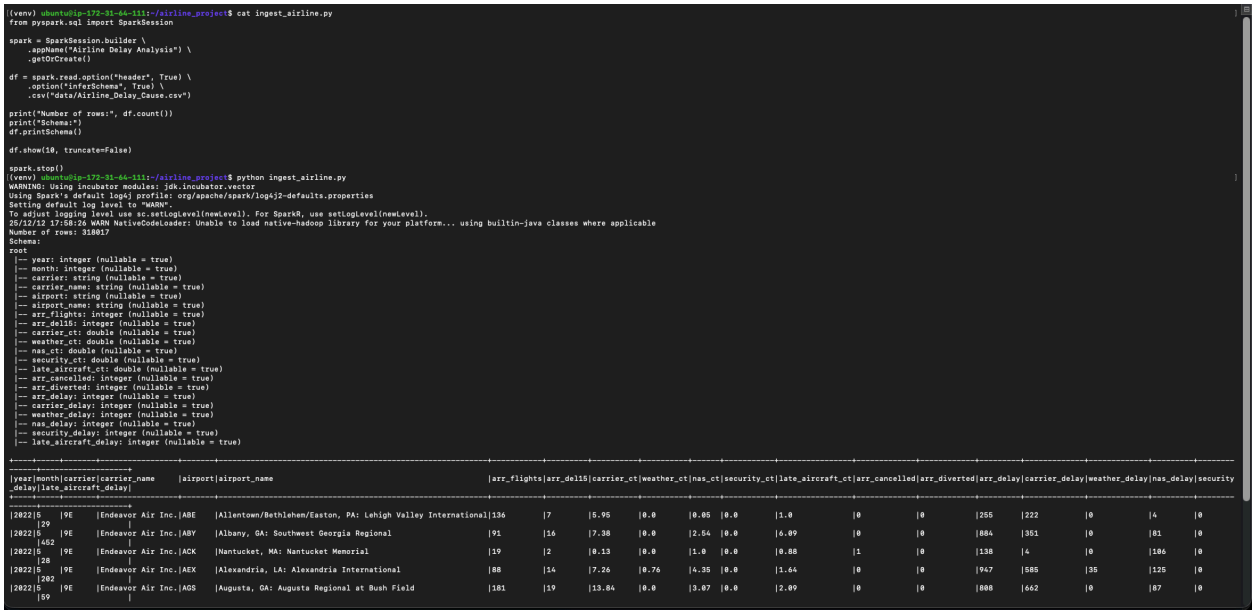
The PySpark execution environment was provided by AWS EC2 (t3.micro) running Amazon Linux. The EC2 instance was given permission to read & write to Amazon S3 using IAM Roles. The unprocessed airline dataset was uploaded to an S3 Bucket created for the project, and the Direct S3 integration feature of PySpark was used to pull data into PySpark from S3 directly, without first downloading the dataset to the EC2 instance.

Verification

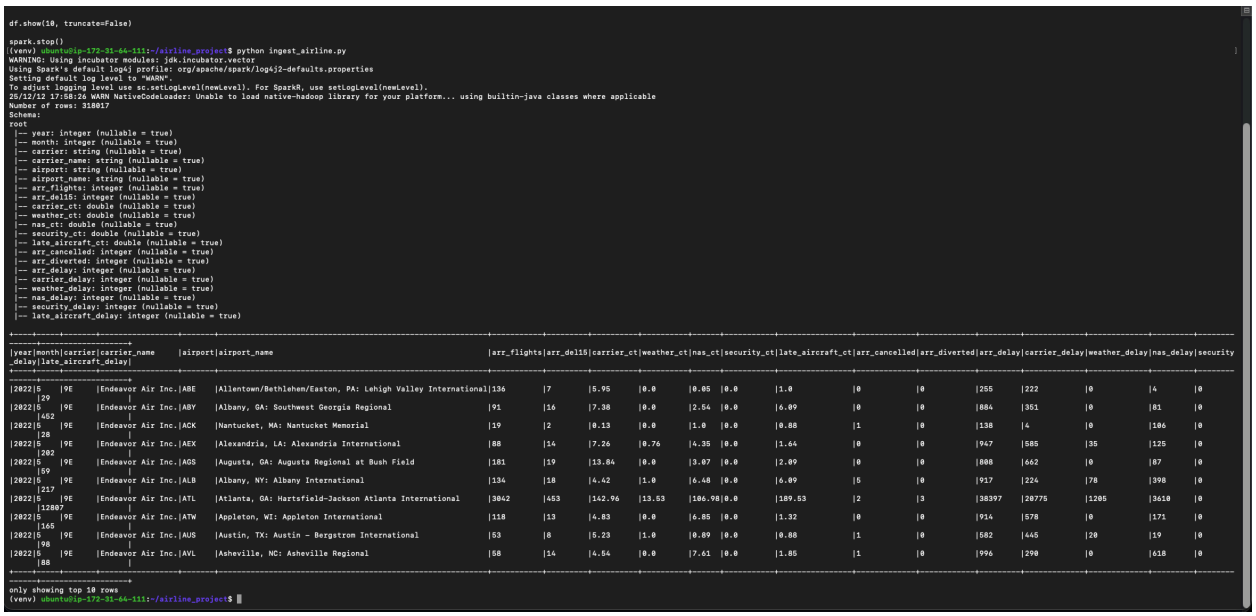
Successful ingestion was confirmed by:

- Displaying the schema using df.printSchema()
- Viewing sample records using df.show()
- Verifying row counts using df.count()

This confirmed that the dataset was correctly loaded into the PySpark environment from S3.



```
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ cat ingest_airline.py
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Airline Delay Analysis") \
    .getOrCreate()

df = spark.read.option("header", True) \
    .option("inferSchema", True) \
    .csv("data/Airline_Delay_Cause.csv")

print("Number of rows:", df.count())
print("Schema:")
df.printSchema()

df.show(10, truncate=False)

spark.stop()
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ python ingest_airline.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/12 17:58:26 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Number of rows: 318017
Schema:
root
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- carrier: string (nullable = true)
 |-- carrier_name: string (nullable = true)
 |-- airport: string (nullable = true)
 |-- airport_name: string (nullable = true)
 |-- arr_flights: integer (nullable = true)
 |-- arr_del15: integer (nullable = true)
 |-- carrier_ct: double (nullable = true)
 |-- weather_ct: double (nullable = true)
 |-- nas_ct: double (nullable = true)
 |-- security_ct: double (nullable = true)
 |-- late_aircraft_ct: double (nullable = true)
 |-- arr_cancelled: integer (nullable = true)
 |-- arr_diverted: integer (nullable = true)
 |-- arr_delay: integer (nullable = true)
 |-- carrier_delay: integer (nullable = true)
 |-- weather_delay: integer (nullable = true)
 |-- nas_delay: integer (nullable = true)
 |-- security_delay: integer (nullable = true)
 |-- late_aircraft_delay: integer (nullable = true)

+----+-----+-------+----------+-------+------------------------------------------------------+-----------+---------+----------+----------+------+-----------+----------------+-------------+------------+-----------+-------------+-------------+----------+--------------+-------------------+
|year|month|carrier|carrier_name|airport|airport_name                                          |arr_flights|arr_del15|carrier_ct|weather_ct|nas_ct|security_ct|late_aircraft_ct|arr_cancelled|arr_diverted|arr_delay|carrier_delay|weather_delay|nas_delay|security_delay|late_aircraft_delay|
+----+-----+-------+----------+-------+------------------------------------------------------+-----------+---------+----------+----------+------+-----------+----------------+-------------+------------+-----------+-------------+-------------+----------+--------------+-------------------+
|2022|5    |9E     |Endeavor Air Inc.|ABE    |Allentown/Bethlehem/Easton, PA: Lehigh Valley International|136        |7        |5.95      |0.0       |0.05  |0.0        |1.0             |0            |0           |255        |222          |0            |4         |0             |29                 |
|2022|5    |9E     |Endeavor Air Inc.|ABY    |Albany, GA: Southwest Georgia Regional                |91         |16       |7.38      |0.0       |2.54  |0.0        |6.09            |0            |0           |884        |351          |0            |81        |0             |452                |
|2022|5    |9E     |Endeavor Air Inc.|ACK    |Nantucket, MA: Nantucket Memorial                     |19         |2        |0.13      |0.0       |1.0   |0.0        |0.88            |1            |0           |138        |4            |0            |106       |0             |28                 |
|2022|5    |9E     |Endeavor Air Inc.|AEX    |Alexandria, LA: Alexandria International              |88         |14       |7.26      |0.76      |4.35  |0.0        |1.64            |0            |0           |947        |585          |35           |125       |0             |202                |
|2022|5    |9E     |Endeavor Air Inc.|AGS    |Augusta, GA: Augusta Regional at Bush Field           |181        |19       |13.84     |0.0       |3.07  |0.0        |2.09            |0            |0           |808        |662          |0            |87        |0             |89                 |
```



```
df.show(10, truncate=False)

spark.stop()
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ python ingest_airline.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/12 17:58:26 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Number of rows: 318017
Schema:
root
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- carrier: string (nullable = true)
 |-- carrier_name: string (nullable = true)
 |-- airport: string (nullable = true)
 |-- airport_name: string (nullable = true)
 |-- arr_flights: integer (nullable = true)
 |-- arr_del15: integer (nullable = true)
 |-- carrier_ct: double (nullable = true)
 |-- weather_ct: double (nullable = true)
 |-- nas_ct: double (nullable = true)
 |-- security_ct: double (nullable = true)
 |-- late_aircraft_ct: double (nullable = true)
 |-- arr_cancelled: integer (nullable = true)
 |-- arr_diverted: integer (nullable = true)
 |-- arr_delay: integer (nullable = true)
 |-- carrier_delay: integer (nullable = true)
 |-- weather_delay: integer (nullable = true)
 |-- nas_delay: integer (nullable = true)
 |-- security_delay: integer (nullable = true)
 |-- late_aircraft_delay: integer (nullable = true)

+----+-----+-------+----------+-------+------------------------------------------------------+-----------+---------+----------+----------+------+-----------+----------------+-------------+------------+-----------+-------------+-------------+----------+--------------+-------------------+
|year|month|carrier|carrier_name|airport|airport_name                                          |arr_flights|arr_del15|carrier_ct|weather_ct|nas_ct|security_ct|late_aircraft_ct|arr_cancelled|arr_diverted|arr_delay|carrier_delay|weather_delay|nas_delay|security_delay|late_aircraft_delay|
+----+-----+-------+----------+-------+------------------------------------------------------+-----------+---------+----------+----------+------+-----------+----------------+-------------+------------+-----------+-------------+-------------+----------+--------------+-------------------+
|2022|5    |9E     |Endeavor Air Inc.|ABE    |Allentown/Bethlehem/Easton, PA: Lehigh Valley International|136        |7        |5.95      |0.0       |0.05  |0.0        |1.0             |0            |0           |255        |222          |0            |4         |0             |29                 |
|2022|5    |9E     |Endeavor Air Inc.|ABY    |Albany, GA: Southwest Georgia Regional                |91         |16       |7.38      |0.0       |2.54  |0.0        |6.09            |0            |0           |884        |351          |0            |81        |0             |452                |
|2022|5    |9E     |Endeavor Air Inc.|ACK    |Nantucket, MA: Nantucket Memorial                     |19         |2        |0.13      |0.0       |1.0   |0.0        |0.88            |1            |0           |138        |4            |0            |106       |0             |28                 |
|2022|5    |9E     |Endeavor Air Inc.|AEX    |Alexandria, LA: Alexandria International              |88         |14       |7.26      |0.76      |4.35  |0.0        |1.64            |0            |0           |947        |585          |35           |125       |0             |202                |
|2022|5    |9E     |Endeavor Air Inc.|AGS    |Augusta, GA: Augusta Regional at Bush Field           |181        |19       |13.84     |0.0       |3.07  |0.0        |2.09            |0            |0           |808        |662          |0            |87        |0             |89                 |
|2022|5    |9E     |Endeavor Air Inc.|ALB    |Albany, NY: Albany International                      |134        |18       |4.42      |1.0       |6.48  |0.0        |6.09            |5            |0           |917        |224          |78           |398       |0             |217                |
|2022|5    |9E     |Endeavor Air Inc.|ATL    |Atlanta, GA: Hartsfield-Jackson Atlanta International |3042       |453      |142.96    |13.53     |106.98|0.0        |189.53          |2            |3           |38397      |20775        |1205         |3610      |0             |12807              |
|2022|5    |9E     |Endeavor Air Inc.|ATW    |Appleton, WI: Appleton International                  |118        |13       |4.83      |0.0       |6.85  |0.0        |1.32            |0            |0           |914        |578          |0            |171       |0             |165                |
|2022|5    |9E     |Endeavor Air Inc.|AUS    |Austin, TX: Austin - Bergstrom International          |53         |8        |5.23      |1.0       |0.89  |0.0        |0.88            |1            |0           |582        |445          |20           |19        |0             |98                 |
|2022|5    |9E     |Endeavor Air Inc.|AVL    |Asheville, NC: Asheville Regional                     |58         |14       |4.54      |0.0       |7.61  |0.0        |1.85            |1            |0           |996        |298          |0            |618       |0             |88                 |
+----+-----+-------+----------+-------+------------------------------------------------------+-----------+---------+----------+----------+------+-----------+----------------+-------------+------------+-----------+-------------+-------------+----------+--------------+-------------------+
only showing top 10 rows
(venv) ubuntu@ip-172-31-44-111:~/airline_project$
```

Task 2: Data Processing with PySpark

Objective

The Objective of this project was to prepare the raw data into a useable format for exploratory & visualization analysis.

Data Cleaning

Basic data cleaning was performed by removing rows containing missing values using: *df.dropna()*

Data Transformation (Feature Engineering)

Two new analytical columns were engineered to enhance insight generation:

1. Total Delay Minutes

A new column **total_delay_minutes** was created by adding all individual delay components:

- Carrier delay
- Weather delay
- NAS delay
- Security delay
- Late aircraft delay

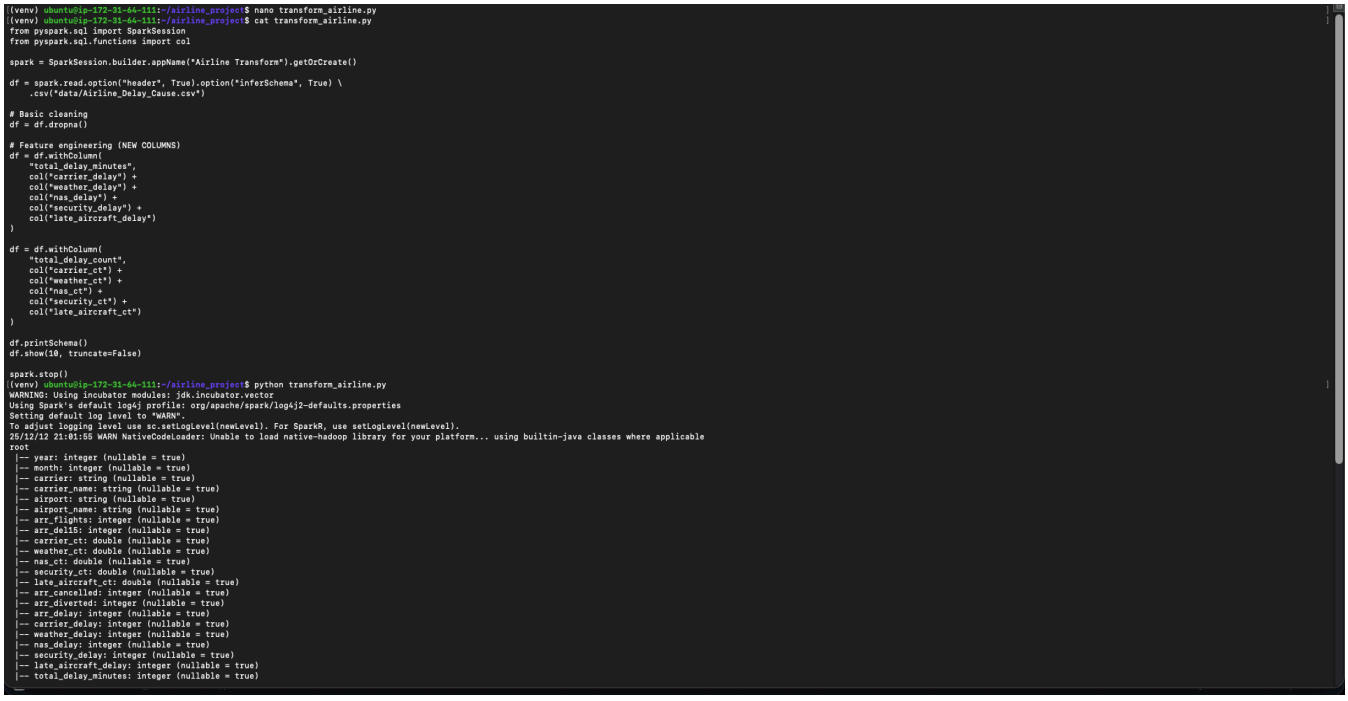
This column represents the overall delay impact per flight.

2. Total Delay Count

Another column **total_delay_count** was created by adding the count-based delay indicators:

- Carrier delay count
- Weather delay count
- NAS delay count
- Security delay count
- Late aircraft delay count

These engineered features provide a clear view of delay frequency and severity.



```
df.printSchema()
df.show(10, truncate=False)

spark.stop()
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ python transform_airline.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/12 21:01:55 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
root
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- carrier: string (nullable = true)
 |-- carrier_name: string (nullable = true)
 |-- airport: string (nullable = true)
 |-- airport_name: string (nullable = true)
 |-- arr_flights: integer (nullable = true)
 |-- arr_del15: integer (nullable = true)
 |-- carrier_ct: double (nullable = true)
 |-- weather_ct: double (nullable = true)
 |-- nas_ct: double (nullable = true)
 |-- security_ct: double (nullable = true)
 |-- late_aircraft_ct: double (nullable = true)
 |-- arr_cancelled: integer (nullable = true)
 |-- arr_diverted: integer (nullable = true)
 |-- arr_delay: integer (nullable = true)
 |-- carrier_delay: integer (nullable = true)
 |-- weather_delay: integer (nullable = true)
 |-- nas_delay: integer (nullable = true)
 |-- security_delay: integer (nullable = true)
 |-- late_aircraft_delay: integer (nullable = true)
 |-- total_delay_minutes: integer (nullable = true)
 |-- total_delay_count: double (nullable = true)

+----+-----+-------+--------------------+-------+--------------------+-----------+---------+----------+----------+-----------+----------------+-------------+------------+-----------+-------------+-------------+---------+--------------+-------------------+-------------------+-----------------+
|year|month|carrier|carrier_name        |airport|airport_name        |arr_flights|arr_del15|carrier_ct|weather_ct|nas_ct     |security_ct     |late_aircraft_ct|arr_cancelled|arr_diverted|arr_delay|carrier_delay|weather_delay|nas_delay|security_delay|late_aircraft_delay|total_delay_minutes|total_delay_count|
+----+-----+-------+--------------------+-------+--------------------+-----------+---------+----------+----------+-----------+----------------+-------------+------------+-----------+-------------+-------------+---------+--------------+-------------------+-------------------+-----------------+
|2022|5    |9E     |Endeavor Air Inc.   |ABE    |Allentown/Bethlehem/Easton, PA: Lehigh Valley International|136        |7        |5.95      |0.0       |0.05       |0.0             |1.0          |0           |0          |255          |222          |0        |4             |29                 |255                |7.0              |
|2022|5    |9E     |Endeavor Air Inc.   |ABY    |Albany, GA: Southwest Georgia Regional|91         |16       |7.38      |0.0       |2.54       |0.0             |6.09         |0           |0          |884          |351          |0        |81            |452                |884                |16.089999999999998|
|2022|5    |9E     |Endeavor Air Inc.   |ACK    |Nantucket, MA: Nantucket Memorial|19         |2        |0.13      |0.0       |1.0        |0.0             |0.88         |1           |0          |138          |4            |0        |106           |28                 |138                |2.01             |
|2022|5    |9E     |Endeavor Air Inc.   |AEX    |Alexandria, LA: Alexandria International|88         |14       |7.26      |0.76      |4.35       |0.0             |1.64         |0           |0          |947          |585          |35       |125           |202                |947                |14.01            |
|2022|5    |9E     |Endeavor Air Inc.   |AGS    |Augusta, GA: Augusta Regional at Bush Field|181        |19       |13.84     |0.0       |3.07       |0.0             |2.09         |0           |0          |808          |662          |0        |87            |59                 |808                |19.0             |
|2022|5    |9E     |Endeavor Air Inc.   |ALB    |Albany, NY: Albany International|134        |18       |4.42      |1.0       |6.48       |0.0             |6.09         |5           |0          |917          |224          |78       |398           |217                |917                |17.998000000000002|
|2022|5    |9E     |Endeavor Air Inc.   |ATL    |Atlanta, GA: Hartsfield-Jackson Atlanta International|3042       |453      |142.96    |13.53     |106.98     |0.0             |189.53       |2           |3          |38397        |20775        |1205     |3610          |12807              |38397              |453.0            |
|2022|5    |9E     |Endeavor Air Inc.   |ATW    |Appleton, WI: Appleton International|118        |13       |4.83      |0.0       |6.85       |0.0             |1.32         |0           |0          |914          |578          |0        |171           |365                |914                |13.0             |
|2022|5    |9E     |Endeavor Air Inc.   |AUS    |Austin, TX: Austin - Bergstrom International|53         |8        |5.23      |1.0       |0.89       |0.0             |0.88         |1           |0          |582          |445          |20       |19            |98                 |582                |8.0              |
|2022|5    |9E     |Endeavor Air Inc.   |AVL    |Asheville, NC: Asheville Regional|58         |14       |4.54      |0.0       |7.61       |0.0             |1.85         |1           |0          |996          |298          |0        |618           |88                 |996                |14.0             |
+----+-----+-------+--------------------+-------+--------------------+-----------+---------+----------+----------+-----------+----------------+-------------+------------+-----------+-------------+-------------+---------+--------------+-------------------+-------------------+-----------------+
only showing top 10 rows
(venv) ubuntu@ip-172-31-44-111:~/airline_project$
```

Data Aggregation (Key Metrics)

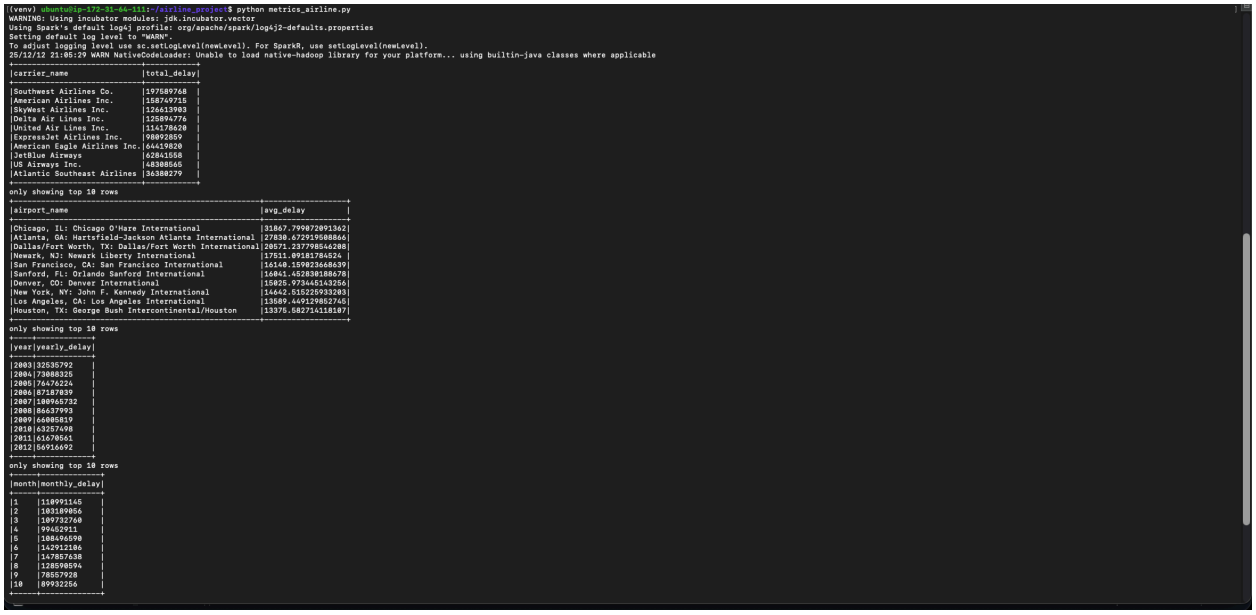
Using PySpark's *groupBy* and aggregation functions, multiple metrics were computed:

1. **Total delay minutes by airline carrier**
2. **Average delay minutes by airport**
3. **Yearly delay trends**
4. **Monthly delay trends**
5. **Average delay minutes per carrier**

Each aggregation was sorted in descending or chronological order to support analytical interpretation.

The results were displayed using *show()* to validate correctness.

```
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ nano metrics_airline.py
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ cat metrics_airline.py
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum, avg, desc

spark = SparkSession.builder.appName("Airline Metrics").getOrCreate()

df = spark.read.option("header", True).option("inferSchema", True) \
    .csv("data/Airline_Delay_Cause.csv") \
    .dropna()

df = df.withColumn(
    "total_delay_minutes",
    df.carrier_delay + df.weather_delay + df.nas_delay +
    df.security_delay + df.late_aircraft_delay
)

# Metric 1: Total delay by carrier
m1 = df.groupBy("carrier_name").agg(
    sum("total_delay_minutes").alias("total_delay")
).orderBy(desc("total_delay"))

# Metric 2: Average delay by airport
m2 = df.groupBy("airport_name").agg(
    avg("total_delay_minutes").alias("avg_delay")
).orderBy(desc("avg_delay"))

# Metric 3: Yearly delay trend
m3 = df.groupBy("year").agg(
    sum("total_delay_minutes").alias("yearly_delay")
).orderBy("year")

# Metric 4: Monthly delay trend
m4 = df.groupBy("month").agg(
    sum("total_delay_minutes").alias("monthly_delay")
).orderBy("month")

# Metric 5: Average delay by carrier
m5 = df.groupBy("carrier_name").agg(
    avg("total_delay_minutes").alias("avg_carrier_delay")
).orderBy(desc("avg_carrier_delay"))

m1.show(10, False)
m2.show(10, False)
m3.show(10, False)
m4.show(10, False)
m5.show(10, False)

spark.stop()
(venv) ubuntu@ip-172-31-44-111:~/airline_project$
```



Task 3: Store Processed Data Back to Amazon S3

Objective

This task aims to Persist a Processed Dataset in a Structured Format so that the Processed Dataset can be reused for Analytics, Visualisation, and Machine Learning Tasks.

Implementation

The Cleaned and Enriched Dataset was exported as a file after Feature Engineering has Completed, by using Spark's Distributed Write Capabilities.

The two resulting Output Formats are:
The Parquet Format - Optimise Storage and Query Performance.
The CSV Format - To work with Visualisation/BI Tools.

The Processed Files were Written out in Overwrite mode so they can always be Reproduced. The Processed Output Files were uploaded to a Specific Location in S3 (the processed/ folder) so they can be Accessed by Services Such as Athena, QuickSight, PowerBI, and SageMaker.

Validation

Storage of Success was confirmed by:

- Checking the Output Files in Local Directory
- Uploading to S3
- Querying the Processed Dataset later via Amazon Athena.

```
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ nano save_to_s3.py
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ cat s
save_to_s3.py   sql_airline.py
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ cat save_to_s3.py
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

spark = SparkSession.builder.appName("Save Airline Data").getOrCreate()

df = spark.read.option("header", True).option("inferSchema", True) \
    .csv("data/Airline_Delay_Cause.csv") \
    .dropna()

# Feature engineering
df = df.withColumn(
    "total_delay_minutes",
    col("carrier_delay") +
    col("weather_delay") +
    col("nas_delay") +
    col("security_delay") +
    col("late_aircraft_delay")
)

df = df.withColumn(
    "total_delay_count",
    col("carrier_ct") +
    col("weather_ct") +
    col("nas_ct") +
    col("security_ct") +
    col("late_aircraft_ct")
)

# Save locally (Spark output)
df.write.mode("overwrite").parquet("output/airline_processed_parquet")
df.write.mode("overwrite").option("header", True).csv("output/airline_processed_csv")

spark.stop()
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ python save_to_s3.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/12 21:16:12 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ ls
data  ingest_airline.py  metrics_airline.py  output  save_to_s3.py  sql_airline.py  transform_airline.py  venv
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ ls output
airline_processed_csv  airline_processed_parquet
```

Task 4: Data Analysis Using Spark SQL

Objective

This task aimed to derive analytical insights from the processed dataset using Spark SQL.

Implementation

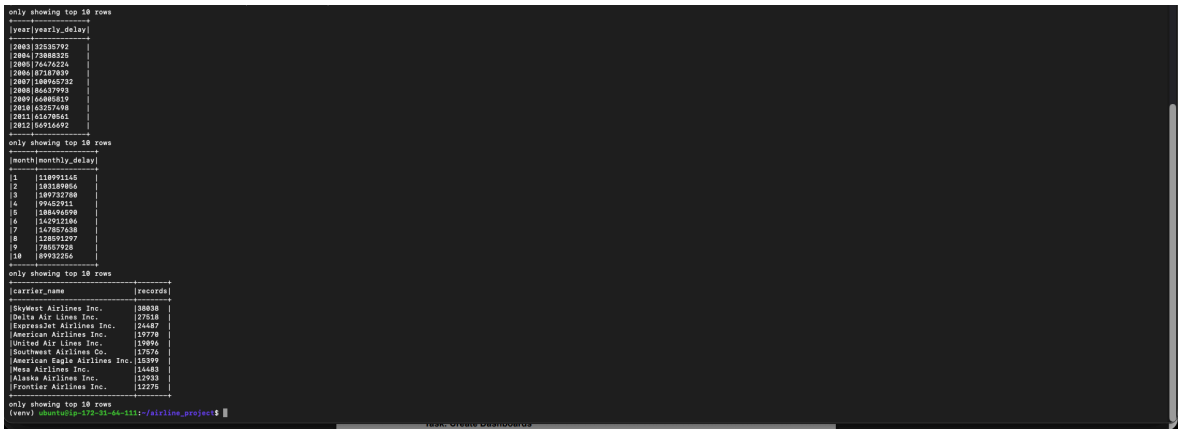
The cleaned dataset was registered as a temporary SQL view named airline, enabling SQL-based analysis.

5 analytical SQL queries were executed:

1. **Total arrival delay by carrier**
    - Identified carriers with the highest cumulative arrival delays.
2. **Average arrival delay by airport**
    - Highlighted airports experiencing frequent or severe delays.
3. **Yearly delay trends**
    - Analyzed how total delays changed across different years.
4. **Monthly delay trends**
    - Revealed seasonal patterns in airline delays.
5. **Flight record count by carrier**
    - Measured operational volume across airlines.

Each query was executed using *spark.sql()* and results were displayed for validation.

```
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ nano sql_airline.py
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ cat sql_airline.py
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Airline SQL").getOrCreate()

df = spark.read.option("header", True).option("inferSchema", True) \
    .csv("data/Airline_Delay_Cause.csv") \
    .dropna()

df.createOrReplaceTempView("airline")

queries = [
    """SELECT carrier_name, SUM(arr_delay) AS total_arrival_delay
       FROM airline GROUP BY carrier_name ORDER BY total_arrival_delay DESC""",

    """SELECT airport_name, AVG(arr_delay) AS avg_arrival_delay
       FROM airline GROUP BY airport_name ORDER BY avg_arrival_delay DESC""",

    """SELECT year, SUM(arr_delay) AS yearly_delay
       FROM airline GROUP BY year ORDER BY year""",

    """SELECT month, SUM(arr_delay) AS monthly_delay
       FROM airline GROUP BY month ORDER BY month""",

    """SELECT carrier_name, COUNT(*) AS records
       FROM airline GROUP BY carrier_name ORDER BY records DESC"""
]

for q in queries:
    spark.sql(q).show(10, False)

spark.stop()
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ python sql_airline.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/12 21:10:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
+--------------------------+--------------------+
|carrier_name              |total_arrival_delay|
+--------------------------+--------------------+
|Southwest Airlines Co.    |197589768           |
|American Airlines Inc.    |158749715           |
|SkyWest Airlines Inc.     |126613983           |
|Delta Air Lines Inc.      |125895285           |
|United Air Lines Inc.     |114178620           |
|ExpressJet Airlines Inc.  |98092859            |
|American Eagle Airlines Inc.|64419820          |
|JetBlue Airways           |62841558            |
|US Airways Inc.           |48388565            |
|Atlantic Southeast Airlines|36388279           |
+--------------------------+--------------------+
only showing top 10 rows
+-----------------------------------------------------+-----------------+
|airport_name                                         |avg_arrival_delay |
+-----------------------------------------------------+-----------------+
|Chicago, IL: Chicago O'Hare International            |31867.799872091362|
|Atlanta, GA: Hartsfield-Jackson Atlanta International|27830.672919508866|
|Dallas/Fort Worth, TX: Dallas/Fort Worth International|26571.237798544288|
|Newark, NJ: Newark Liberty International             |17511.091817845424|
|San Francisco, CA: San Francisco International       |16148.159073668639|
|Sanford, FL: Orlando Sanford International          |16041.462839188678|
|Denver, CO: Denver International                     |15025.973445143256|
|New York, NY: John F. Kennedy International          |14642.615229593283|
+-----------------------------------------------------+-----------------+
```



```
only showing top 10 rows
+----+-------------+
|year|yearly_delay|
+----+-------------+
|2003|32535792    |
|2004|73888325    |
|2005|76476224    |
|2006|87187639    |
|2007|100965732   |
|2008|86437993    |
|2009|66005819    |
|2010|63257498    |
|2011|61478561    |
|2012|56916692    |
+----+-------------+
only showing top 10 rows
+-----+-------------+
|month|monthly_delay|
+-----+-------------+
|1    |118991145    |
|2    |103189856    |
|3    |109732780    |
|4    |99452911     |
|5    |108496698    |
|6    |142912106    |
|7    |147857638    |
|8    |128591297    |
|9    |78557928     |
|10   |89932256     |
+-----+-------------+
only showing top 10 rows
+--------------------------+------+
|carrier_name              |records|
+--------------------------+------+
|SkyWest Airlines Inc.     |30038  |
|Delta Air Lines Inc.      |27518  |
|ExpressJet Airlines Inc.  |24487  |
|American Airlines Inc.    |19778  |
|United Air Lines Inc.     |19094  |
|Southwest Airlines Co.    |17576  |
|American Eagle Airlines Inc.|15399|
|Mesa Airlines Inc.        |14483  |
|Alaska Airlines Inc.      |12933  |
|Frontier Airlines Inc.    |12275  |
+--------------------------+------+
only showing top 10 rows
(venv) ubuntu@ip-172-31-44-111:~/airline_project$
```

Task 5: Machine Learning with AWS SageMaker Autopilot

Objective

This project aimed to utilize AWS SageMaker Autopilot to facilitate machine learning through an automated workflow (including all steps of training, evaluation, and comparison) using preprocessed airline delay records stored in the AWS S3 Bucket. Automation meant that no programming was required. The expected outcome was for every model to provide predictions for the total delay in minutes for flights based on recorded historical flight and delays and for each model to be evaluated based on interpretability, performance, and ethicality.

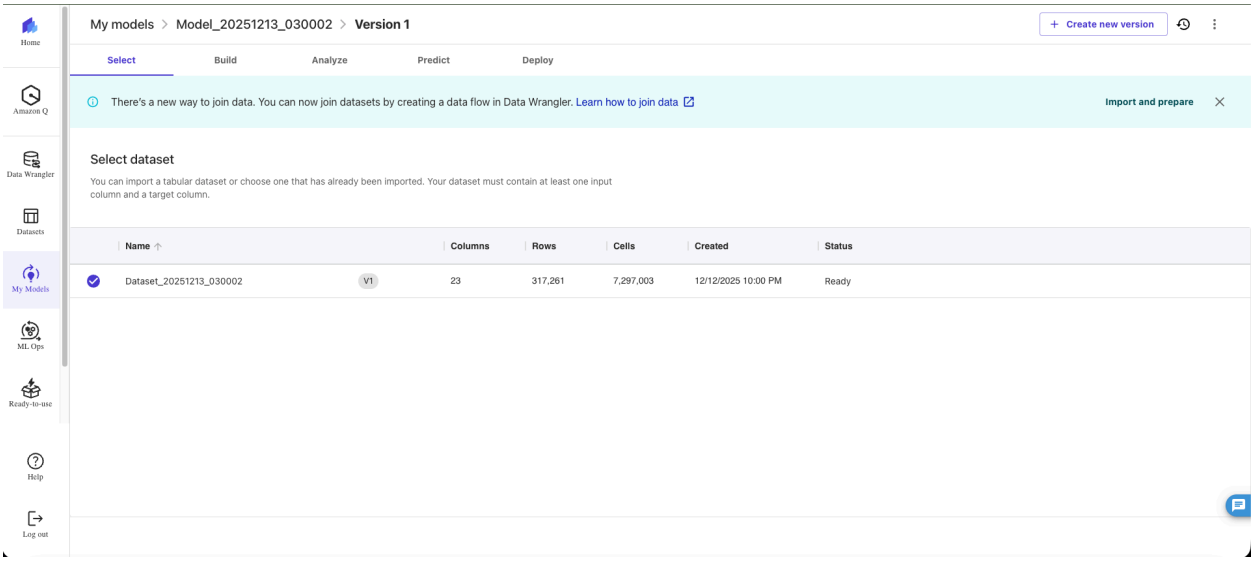
i) Importing Processed Data from S3

After completing data cleaning, feature engineering, and aggregation using **PySpark**, the processed dataset was exported to Amazon S3 in CSV format.

- **S3 Location:**
  s3://rg422-bigdata/processed/airline_quicksight/
- The dataset contained **317,261 rows** and **23 columns**, including:
  - Engineered features such as *year, month, total_delay_minutes,* and *total_delay_count*
  - Delay-related attributes like *arr_delay, carrier_delay, nas_delay, weather_delay,* and *late_aircraft_delay*

This processed dataset was then imported into **AWS SageMaker Autopilot (Canvas)** directly from S3 using the **Import Dataset** option.

Outcome:
The dataset was successfully loaded and validated by SageMaker Autopilot and making it ready for model training.



ii) Running the Autopilot Experiment

Target Column Selection

- **Target variable:** *total_delay_minutes*
- **Problem type:** Numeric prediction (regression)

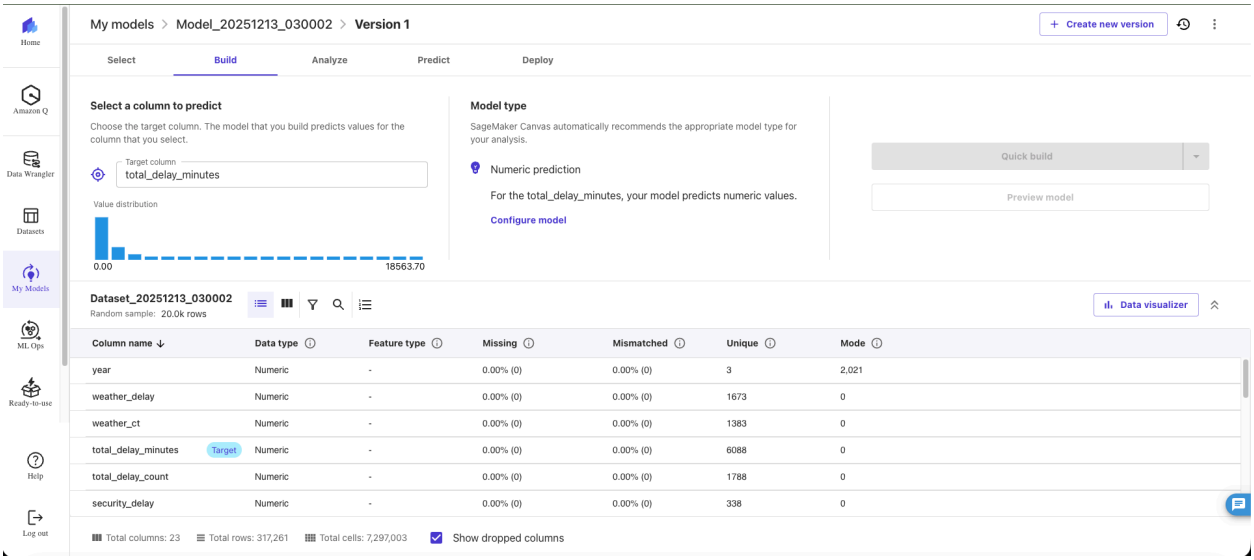
SageMaker Autopilot automatically analyzed the dataset and identified this as a **regression problem** because the target column represents continuous numeric values.

Model Training

Autopilot has been configured with the following capabilities:

- Divide the original data into training and testing sets
- Train several regression algorithms using multiple methods
- Optimize hyper-parameters for optimal results
- Select the optimal model through evaluation of errors

The **Quick Build** option was used to efficiently train models while minimizing configuration overhead.



iii) Model Evaluation and Results

Key Performance Metrics

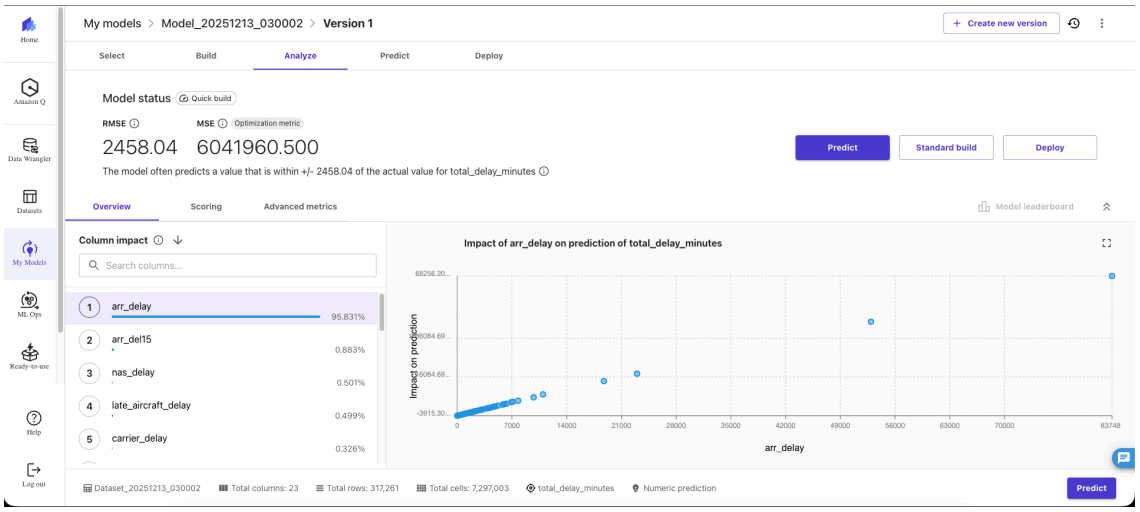
The best-performing model achieved the following results:

Metric	Value
RMSE	2458.04
MSE	6,041,960.50
MAE	±168.72
R² Score	96.24%

- The **R²** score of **96.24%** shows how well the total delay minute's model can predict variance.
- The **RMSE** score of **2458 minutes** shows the model's predictive accuracy is generally estimated within ±2458 minutes of the actual total delay.

Predicted vs Actual Analysis

The **Predicted vs Actual** visualization showed a strong linear relationship between predicted and actual values, confirming good model fit and low systematic bias for most observations.



iv) Feature Importance and Interpretability

SageMaker Autopilot automatically generated feature insights:

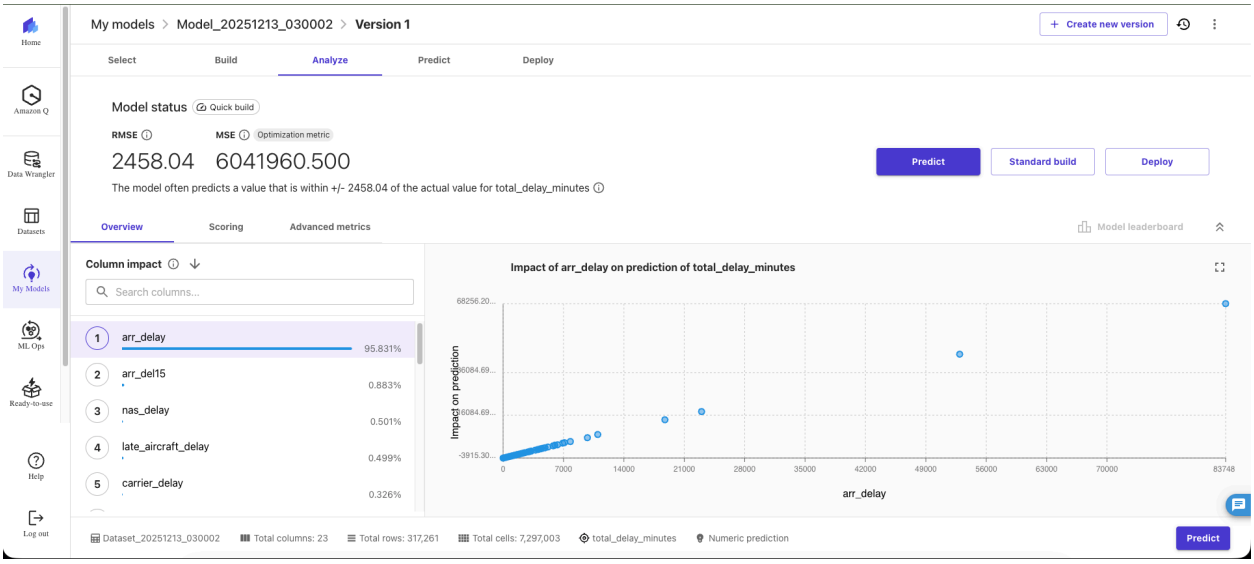
Top contributing features:

1. *arr_delay (Arrival delay)* – **~95.8% impact**
2. *nas_delay*
3. *late_aircraft_delay*
4. *carrier_delay*

This confirms that **arrival delays** are the strongest predictor of total delay minutes, which aligns with real-world airline operations.

Residual and Error Analysis

- The residual plots show how most of the errors clustered around zero for most predictions.
- Large and rare extreme delay scenarios (e.g., due to weather, cascading delays) have some significant outliers.



v) Making Predictions

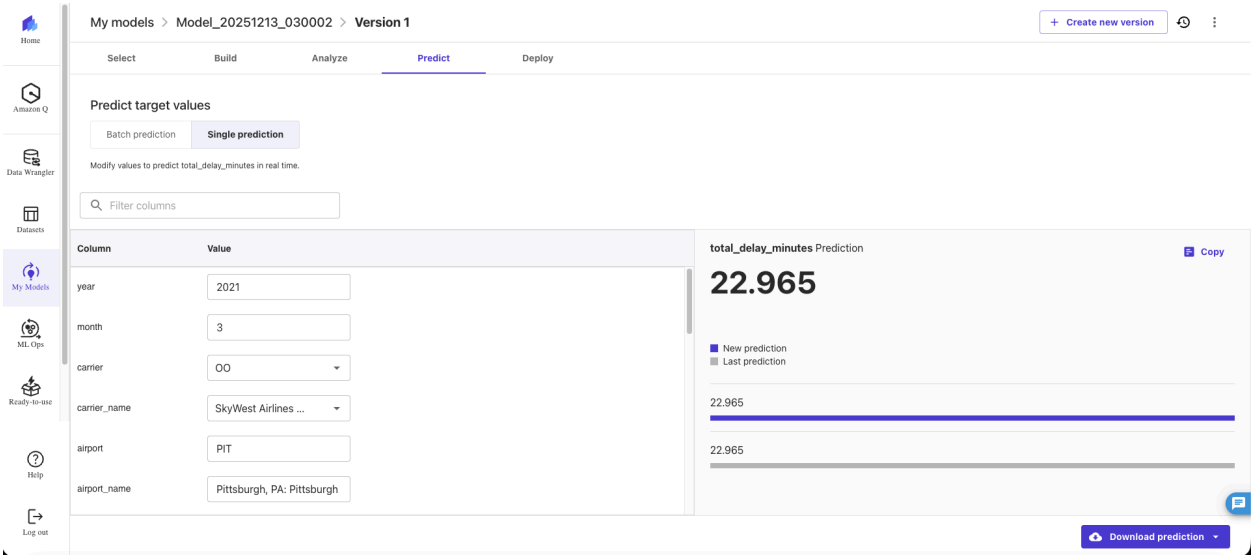
Using the **Predict → Single Prediction** feature in SageMaker Autopilot, real-time predictions were generated by manually entering feature values such as:

- Year
- Month
- Carrier
- Airport
- Delay-related features

Example Prediction Output:

- Predicted *total_delay_minutes*: **22.965 minutes**

This demonstrated that the trained model can be used for **what-if analysis** and operational forecasting.



vi) Ethical Considerations

Bias

Airline historical data may introduce bias into the model, for example:

- The model may overly represent major airports
- Certain carriers have higher historical data

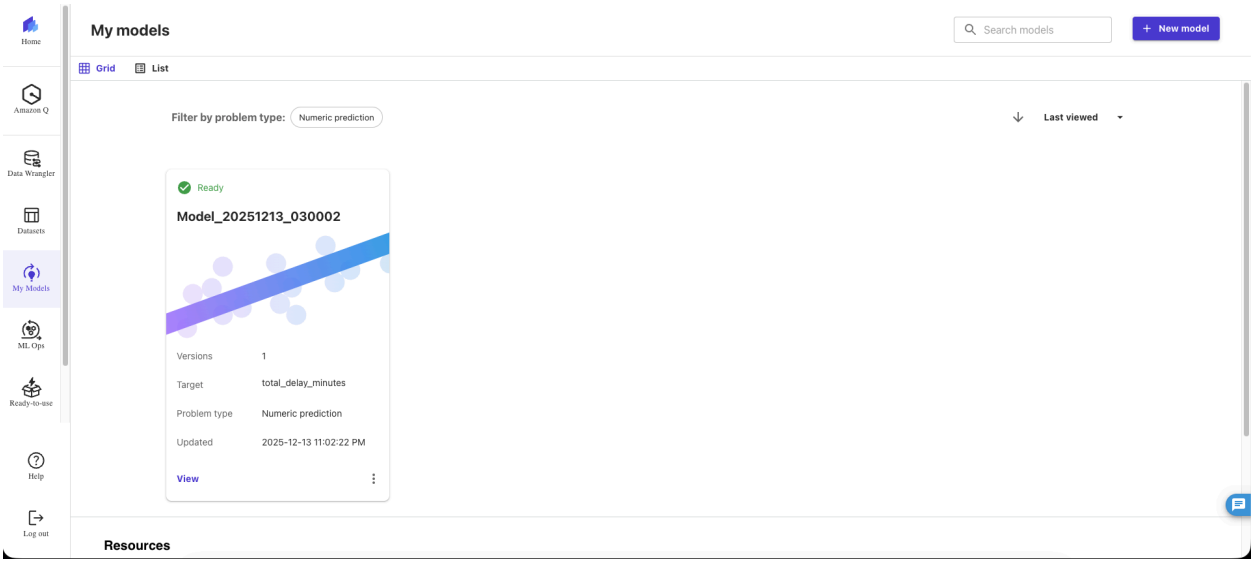
Biases will impact the model's prediction accuracy for smaller airports or less frequently flown routes.

Privacy

- The dataset contained **no personally identifiable information (PII)**.
- Operational flight and delay metrics were used as inputs to the model, so privacy best practices are followed.

Mitigation

- The model is continually retrained with new, updated data.
- Prediction errors are monitored by carrier and airport.
- Sensitive or irregular level attributes of passengers will not be used in the model.



5. Visualization

5.1 Visualization Tool Selection

Interactive dashboards were developed to provide meaningful insights from the processed Airline Delay cause dataset stored in Amazon S3 as the primary focus of this component of the overall project.

Initially, we looked at using AWS QuickSight for the visualizing component of the project; however, we used **Microsoft Power BI** instead due to difficulties related to setting up accounts in AWS. I encountered multiple errors while creating and validating a QuickSight account on the AWS Free Tier; these issues are outlined and explained in the video that accompanied this project.

To finish this project on time and still meet all visualization requirements, we decided to use Power BI as an alternate visualization platform, which was permitted by the project requirements, to avoid accruing additional charges to our AWS account.

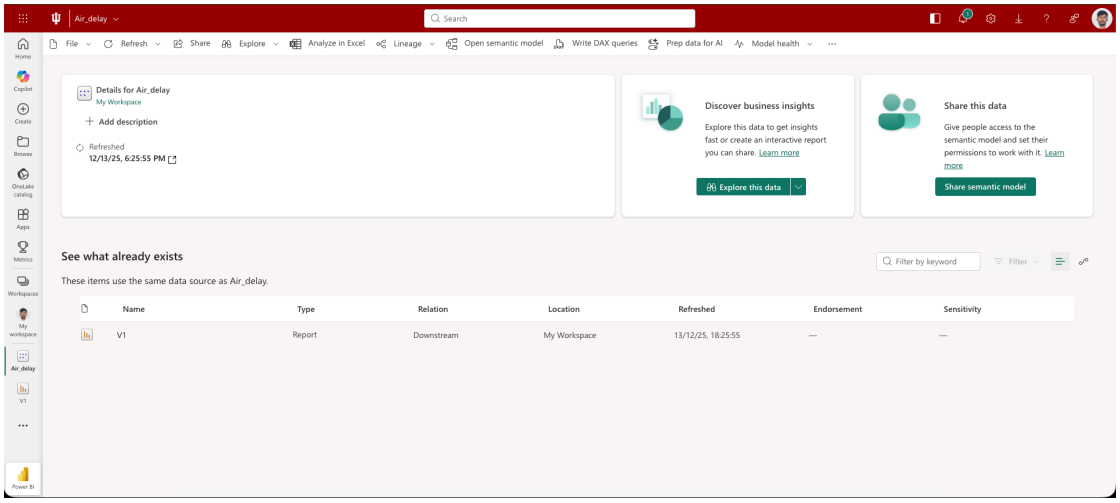
Justification:

Power BI offers a wide variety of data connectivity options; it is also capable of connecting to very large datasets; it has a scheduling feature that allows it to auto-refresh itself and generate reports automatically; and it allows users to create professional-grade interactive dashboards that can be applied to big data analysis.

5.2 Data Transfer and Refresh Validation

Data acquired over the period is available in Amazon S3 after being imported into Power BI, which has thus enabled this information to be used in various types of Analytics tools designed to track various aspects of the Flight Industry. PySpark was used to perform calculations on the airline data created and uploaded into Amazon S3.

The processed data in csv format was manually uploaded to Power BI .As a validation of the Data Pipeline; a manual refresh of the data connection was completed to verify that all the latest data residing in Amazon S3 was being properly imported into Power BI and reflected within all current Dashboards residing within Power BI.



5.3 Dashboard Design and Visualizations

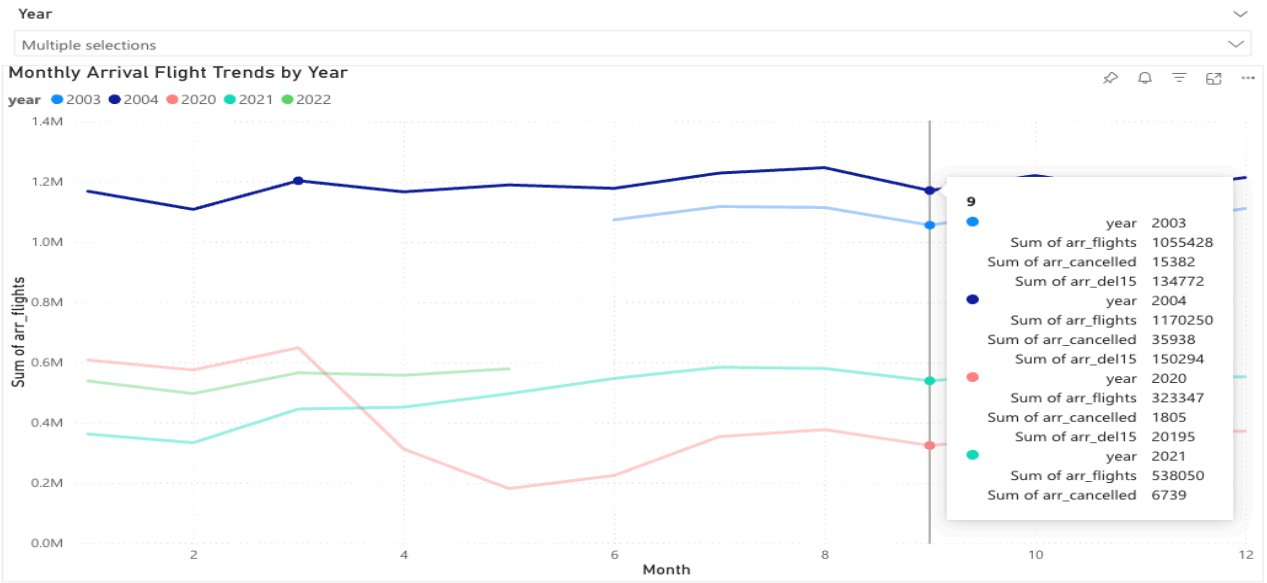
An extensive Report which contained an array of four or more useful visual tools was developed using the Features available in Power BI. Each Visualization covered a separate analytical area of focus regarding airline performance, delays and reliability.

Visualization 1: Monthly Arrival Flight Trends by Year

Overview: This Line Graph displays the breakdown of Arrival Flights by month across a series of years (2003, 2004, 2020, 2021, 2022). This Yearly Breakdown provides an opportunity for users to identify and study how the various factors affecting airline performance will change from season to season and year to year. Users are also able to clearly see the effects of large scale external factors, particularly the COVID-19 Pandemic. We can select the required year from year dropdown at the top of the chart

Findings:

- The Year 2020 saw the lowest flight volume totals ever.
- The Yearly Volume Totals began to gradually increase over the subsequent years.
- The Mid July/August Summer Peak continues every year.



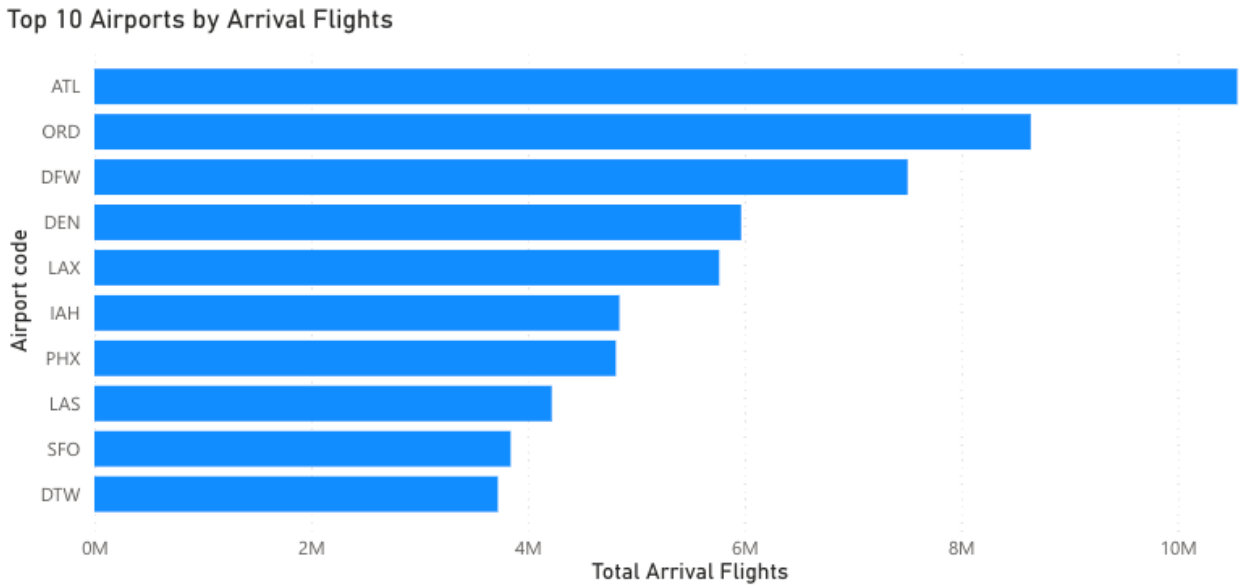
Visualization 2: Top 10 Airports by Arrival Flights

Overview:

A horizontal bar chart provides a ranking of the top 10 airports based on overall number of arrivals.

Insights:

Airports like ATL, ORD & DFW control most inbound flights.These three airports have a large share of the total number of inbound flights nationwide.

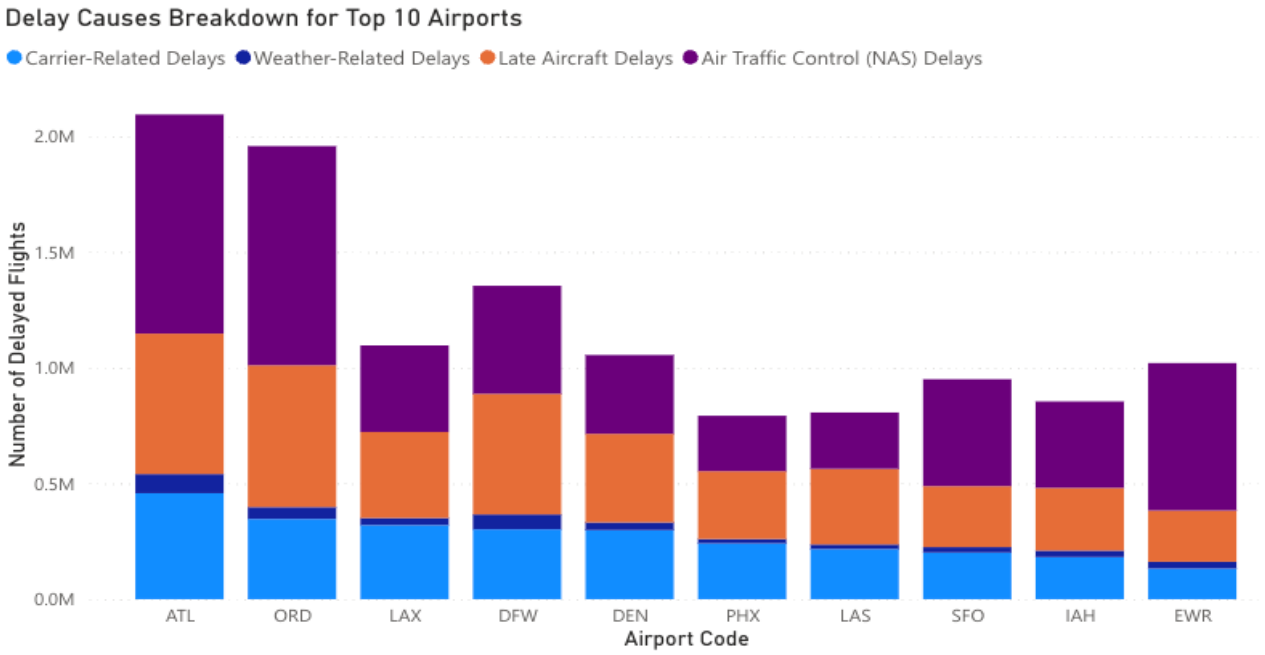


Visualization 3: Delay Causes Breakdown for Top 10 Airports

Overview:
A stacked bar chart provides the delay causes across the top airports in the United States.

The chart includes:
- Carrier-related delays
- Weather-related delays
- Late aircraft delays
- Air Traffic Control (National Airspace System (NAS)) delays

Insights:
Carrier & late aircraft delays are the predominant delays. Weather delays are variable across airports.



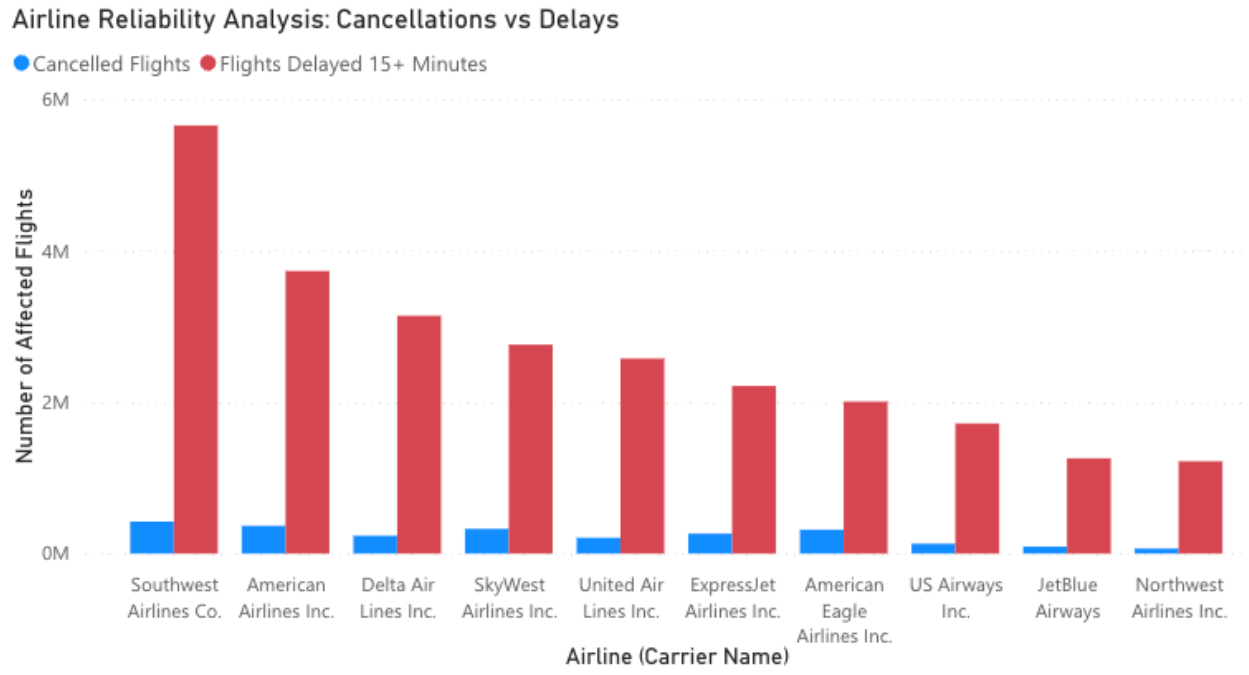
Visualization 4: Airline Reliability Analysis – Cancellations vs Delays

Overview:

- A comparison of cancellations versus flights delayed 15 minutes or more for the major airline carriers in the U.S.
- The chart allows for easy comparison of reliabilities between the airline carriers.

Insights:

- There are multiple airlines with far fewer cancellations and considerably more delays than expected.
- Reliability is inconsistent across airlines.



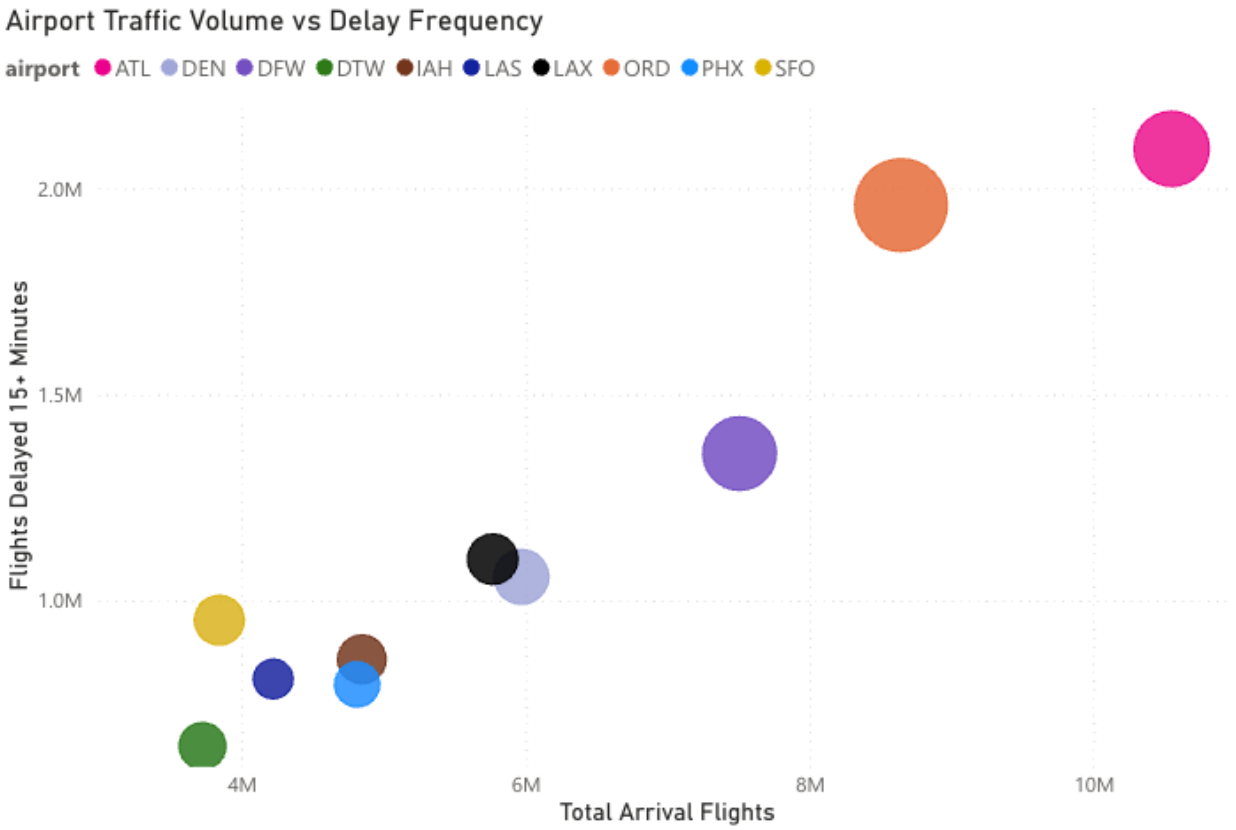
Visualization 5: Airport Traffic Volume vs Delay Frequency

Overview:

A bubble scatterplot depicts the relationship between total incoming flights to a given airport and the average frequency of delays for that airport.

Insights:

- Higher volume airports have a correspondingly higher number of delays.
- Certain airports can demonstrate higher levels of operational efficiencies while operating at a higher volume.



6.Bonus Task: Automation of the Pipeline

Objective

The aim of this extra assignment is to create a completely self-sufficient method of running the Big Data Pipeline from end-to-end. Specifically, we will cover data ingestion from Amazon S3 through to executing distributed processing of that data, putting the findings back into Amazon S3, sending an email notification of when it has been executed and providing for automatic refreshing of the visualisation dashboard.

This automation allows us to execute the pipeline with no manual input, providing us with an increased level of scalability, greater reliability and a greater ability to perform Real-Time Analytics on our data.

6.1 Automation Architecture Overview

The automated pipeline follows the architecture below:

Amazon S3 (Raw Data) → AWS Lambda Trigger → AWS Systems Manager (SSM) → EC2 PySpark Pipeline → Processed Data Stored in S3 → SNS Email Notifications → Power BI Dataset Refresh

This architecture eliminates manual SSH execution and enables serverless orchestration using AWS managed services.

6.2 Automation Script (1 Point)

Approach

A **single automation script (run_pipeline.sh)** is executed on an EC2 instance that hosts the PySpark environment.
This script performs the following steps automatically:

1. Reads raw airline delay data from Amazon S3
2. Executes PySpark transformations and aggregations
3. Writes processed outputs in **CSV and Parquet formats** back to Amazon S3
4. Stores detailed execution logs in S3 for auditing and debugging

Pipeline Execution Location

- EC2 Instance: *rg422-Airline-PySpark (t3.micro)*
- Pipeline Directory*: /home/ubuntu/pipeline/*
- Logs Directory: */home/ubuntu/pipeline/logs/*

Verified Outputs in S3

- *s3://rg422-bigdata/processed/airline_processed_csv/*
- *s3://rg422-bigdata/processed/airline_processed_parquet/*
- *s3://rg422-bigdata/processed/analytics/logs/*

```
ubuntu@ip-172-31-64-111:~/pipeline$ cat run_pipeline.sh
#!/usr/bin/env bash
set -euo pipefail

# ================ CONFIG ================
BUCKET="rg422-bigdata"
PROCESSED_PREFIX="processed"
ANALYTICS_PREFIX="analytics"
LOGS_PREFIX="logs"

# SNS Topic ARN (TOPIC ARN, not subscription ARN)
TOPIC_ARN="arn:aws:sns:us-east-1:668826141957:pipeline-status-topic"

# Spark submit path (from your venv)
SPARK_SUBMIT="$HOME/airline_project/venv/bin/spark-submit"

# Timestamp
TS=$(date +"%Y-%m-%d_%H-%M-%S")

# Local log location
LOG_DIR="$HOME/pipeline/logs"
LOG_LOCAL="$LOG_DIR/pipeline_$TS.log"

# S3 log location
S3_LOG_PATH="s3://$BUCKET/$PROCESSED_PREFIX/$ANALYTICS_PREFIX/$LOGS_PREFIX/pipeline_$TS.log"

# ============== PREP ====================
mkdir -p "$LOG_DIR"

# Redirect all stdout & stderr to log + console
exec > >(tee -a "$LOG_LOCAL") 2>&1

# ============== FAILURE HANDLER =========
trap '
aws sns publish \
  --topic-arn "$TOPIC_ARN" \
  --subject "❌ Pipeline Failed" \
  --message "Pipeline FAILED at $(date).
Check logs:
$S3_LOG_PATH"
' ERR

echo "=========================================="
echo "🚀 Pipeline started at $(date)"
echo "=========================================="

# Move to PySpark project directory
cd "$HOME/airline_project"

# ============== PIPELINE STEPS ===========

echo "[1/5] Ingesting raw data..."
"$SPARK_SUBMIT" ingest_airline.py

echo "[2/5] Transforming data..."
"$SPARK_SUBMIT" transform_airline.py

echo "[3/5] Computing metrics..."
"$SPARK_SUBMIT" metrics_airline.py

echo "[4/5] Running Spark SQL analysis..."
"$SPARK_SUBMIT" sql_airline.py
```

```
echo "[5/5] Saving processed data to S3..."
"$SPARK_SUBMIT" save_to_s3.py

# ============== UPLOAD LOG ================
echo "Uploading logs to S3..."
aws s3 cp "$LOG_LOCAL" "$S3_LOG_PATH"

# ============== SUCCESS NOTIFICATION =======
aws sns publish \
  --topic-arn "$TOPIC_ARN" \
  --subject "✅ Pipeline Success" \
  --message "Pipeline completed SUCCESSFULLY at $(date).

Logs:
$S3_LOG_PATH"

echo "=========================================="
echo "✅ Pipeline completed successfully"
echo "=========================================="
ubuntu@ip-172-31-64-111:~/pipeline$
```

```
(venv) ubuntu@ip-172-31-64-111:~/pipeline$ cd ~/airline_project/
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ ls
airline_autopilot_input.csv  ingest_airline.py    output          sql_airline.py      venv
data                         metrics_airline.py   save_to_s3.py   transform_airline.py
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ chmod +x ~/pipeline/run_pipeline.sh
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ ~/pipeline/run_pipeline.sh
==========================================
🚀 Pipeline started at Sat Dec 13 22:56:35 UTC 2025
==========================================
[1/5] Ingesting raw data...
25/12/13 22:56:39 INFO SparkContext: Running Spark version 3.5.1
25/12/13 22:56:39 INFO SparkContext: OS info Linux, 6.14.0-1018-aws, amd64
25/12/13 22:56:39 INFO SparkContext: Java version 17.0.17
25/12/13 22:56:39 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
25/12/13 22:56:39 INFO ResourceUtils: ==============================================================
25/12/13 22:56:39 INFO ResourceUtils: No custom resources configured for spark.driver.
25/12/13 22:56:39 INFO ResourceUtils: ==============================================================
25/12/13 22:56:39 INFO SparkContext: Submitted application: Airline Delay Analysis
25/12/13 22:56:39 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), tas
k resources: Map(cpus -> name: cpus, amount: 1.0)
25/12/13 22:56:39 INFO ResourceProfile: Limiting resource is cpu
25/12/13 22:56:39 INFO ResourceProfileManager: Added ResourceProfile id: 0
25/12/13 22:56:39 INFO SecurityManager: Changing view acls to: ubuntu
25/12/13 22:56:39 INFO SecurityManager: Changing modify acls to: ubuntu
25/12/13 22:56:39 INFO SecurityManager: Changing view acls groups to:
25/12/13 22:56:39 INFO SecurityManager: Changing modify acls groups to:
25/12/13 22:56:39 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: ubuntu; groups with view permissions: EMPTY; users with modify permissions: ubuntu; groups with modify permissions: EMPTY
25/12/13 22:56:39 INFO Utils: Successfully started service 'sparkDriver' on port 37185.
25/12/13 22:56:39 INFO SparkEnv: Registering MapOutputTracker
25/12/13 22:56:39 INFO SparkEnv: Registering BlockManagerMaster
25/12/13 22:56:39 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
25/12/13 22:56:39 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
25/12/13 22:56:39 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
25/12/13 22:56:40 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-7f1bddd9-e583-4c08-9ea6-573b6bdd7f6
25/12/13 22:56:40 INFO MemoryStore: MemoryStore started with capacity 413.9 MiB
25/12/13 22:56:40 INFO SparkEnv: Registering OutputCommitCoordinator
25/12/13 22:56:40 INFO JettyUtils: Start Jetty 0.0.0.0:4040 for SparkUI
25/12/13 22:56:40 INFO Utils: Successfully started service 'SparkUI' on port 4040.
25/12/13 22:56:40 INFO Executor: Starting executor ID driver on host ip-172-31-64-111.ec2.internal
25/12/13 22:56:40 INFO Executor: OS info Linux, 6.14.0-1018-aws, amd64
25/12/13 22:56:40 INFO Executor: Java version 17.0.17
25/12/13 22:56:40 INFO Executor: Starting executor with user classpath (userClassPathFirst = false): ''
25/12/13 22:56:40 INFO Executor: Created or updated repl class loader org.apache.spark.util.MutableURLClassLoader@38e4c1fd for default.
25/12/13 22:56:40 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 44475.
25/12/13 22:56:40 INFO NettyBlockTransferService: Server created on ip-172-31-64-111.ec2.internal:44475
25/12/13 22:56:40 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
25/12/13 22:56:40 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, ip-172-31-64-111.ec2.internal, 44475, None)
25/12/13 22:56:40 INFO BlockManagerMasterEndpoint: Registering block manager ip-172-31-64-111.ec2.internal:44475 with 413.9 MiB RAM, BlockManagerId(driver, ip-172-31-64-111.ec2.internal, 44475, None)
25/12/13 22:56:40 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, ip-172-31-64-111.ec2.internal, 44475, None)
25/12/13 22:56:40 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, ip-172-31-64-111.ec2.internal, 44475, None)
25/12/13 22:56:41 INFO SharedState: Setting hive.metastore.warehouse.dir ('null') to the value of spark.sql.warehouse.dir.
25/12/13 22:56:41 INFO SharedState: Warehouse path is 'file:/home/ubuntu/airline_project/spark-warehouse'.
25/12/13 22:56:42 INFO InMemoryFileIndex: It took 63 ms to list leaf files for 1 paths.
25/12/13 22:56:42 INFO InMemoryFileIndex: It took 2 ms to list leaf files for 1 paths.
25/12/13 22:56:45 INFO FileSourceStrategy: Pushed Filters:
25/12/13 22:56:45 INFO FileSourceStrategy: Post-Scan Filters: (length(trim(value#0, None)) > 0)
25/12/13 22:56:46 INFO CodeGenerator: Code generated in 273.144649 ms
25/12/13 22:56:46 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 199.6 KiB, free 413.7 MiB)
25/12/13 22:56:46 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 34.4 KiB, free 413.7 MiB)
25/12/13 22:56:46 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on ip-172-31-64-111.ec2.internal:44475 (size: 34.4 KiB, free: 413.9 MiB)
25/12/13 22:56:46 INFO SparkContext: Created broadcast 0 from csv at NativeMethodAccessorImpl.java:0
25/12/13 22:56:46 INFO FileSourceScanExec: Planning scan with bin packing, max size: 23361877 bytes, open cost is considered as scanning 4194304 bytes.
25/12/13 22:56:47 INFO SparkContext: Starting job: csv at NativeMethodAccessorImpl.java:0
25/12/13 22:56:47 INFO DAGScheduler: Got job 0 (csv at NativeMethodAccessorImpl.java:0) with 1 output partitions
25/12/13 22:56:47 INFO DAGScheduler: Final stage: ResultStage 0 (csv at NativeMethodAccessorImpl.java:0)
```

6.3 Scheduling and Triggering Using AWS Lambda

Why AWS Lambda

AWS Lambda was used instead of cron jobs to provide:

- Serverless execution
- Event-driven triggering
- Better fault tolerance
- Easy integration with AWS services

Lambda Function

A Lambda function named **rg422-trigger-pipeline** was created to trigger the EC2 pipeline remotely.

Lambda Responsibilities

- Invoke AWS Systems Manager (SSM)
- Execute shell commands on the EC2 instance
- Trigger the run_pipeline.sh script
- Capture execution status

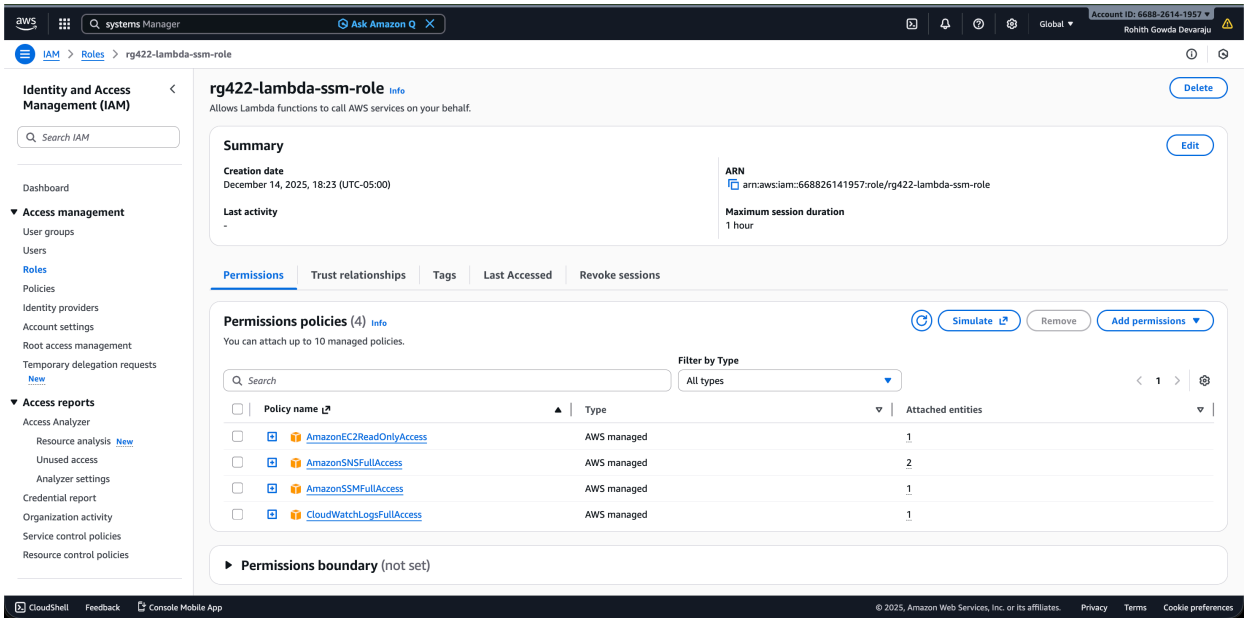
Lambda Code Logic

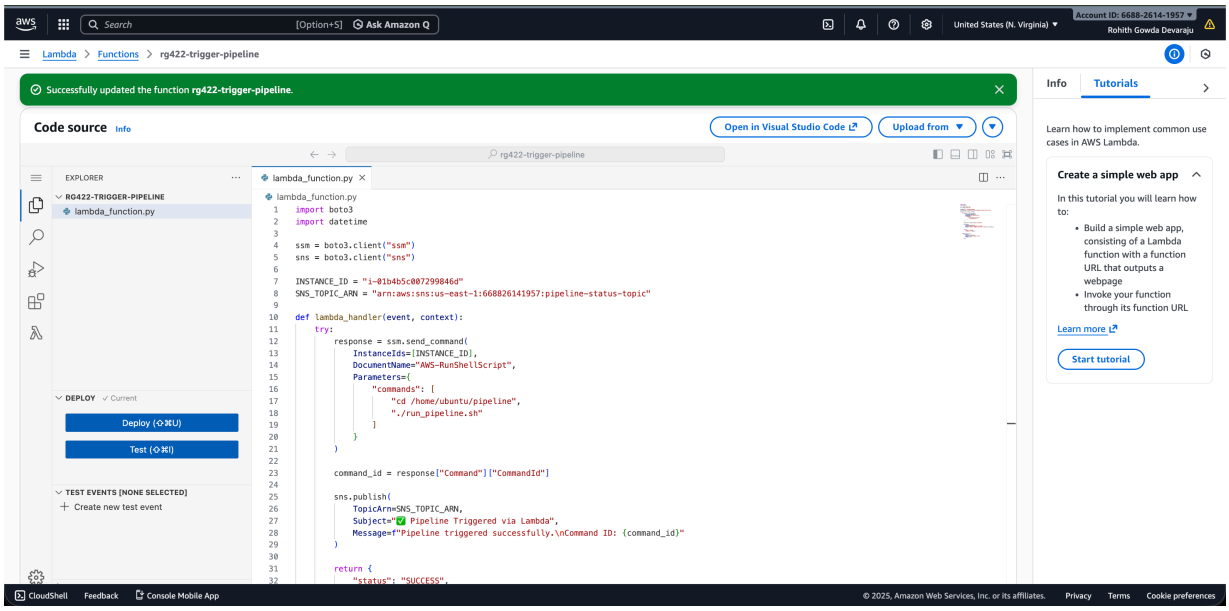
The Lambda function:

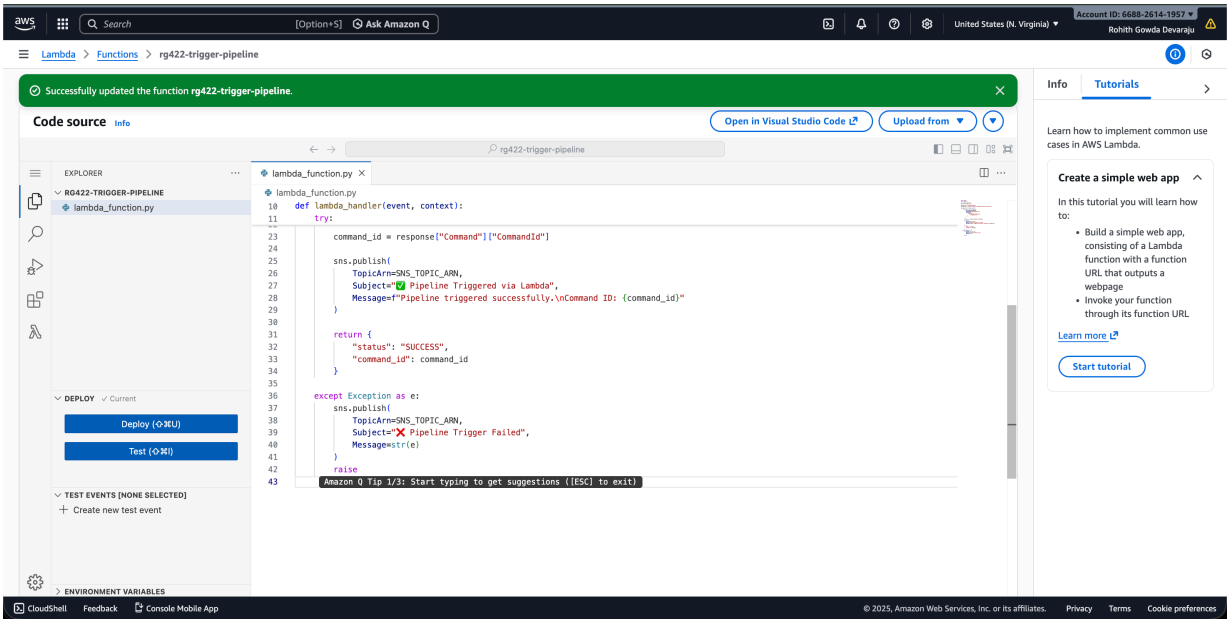
1. Sends an AWS-RunShellScript command via SSM
2. Executes:

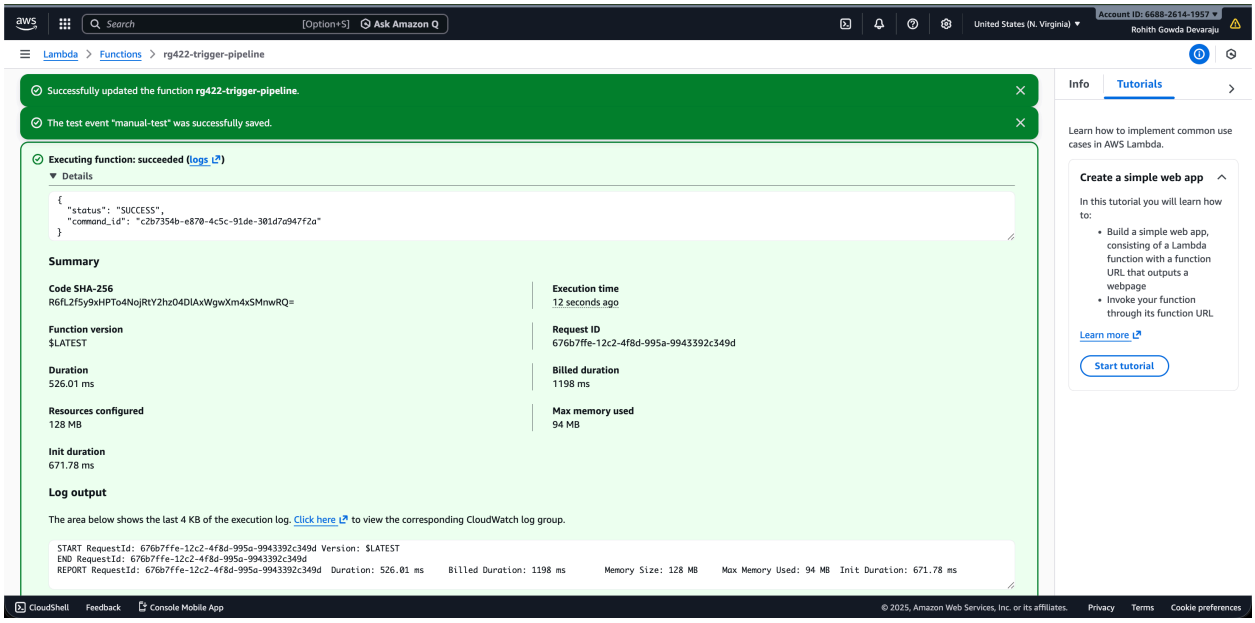
   cd /home/ubuntu/pipeline
   ./run_pipeline.sh

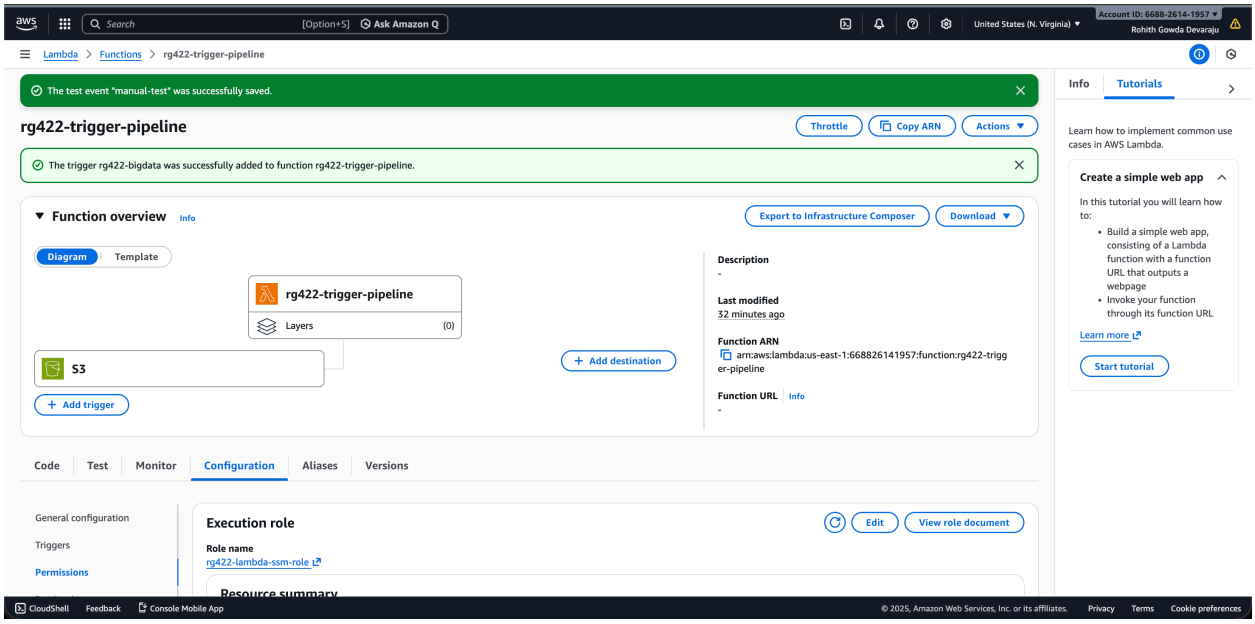
3. Publishes success or failure notifications using SNS


systems Manager
Ask Amazon Q
Account ID: 6688-2614-1957
Rohith Gowda Devaraju
IAM > Roles > rg422-lambda-ssm-role
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management
Temporary delegation requests
New
Access reports
Access Analyzer
Resource analysis New
Unused access
Analyzer settings
Credential report
Organization activity
Service control policies
Resource control policies
rg422-lambda-ssm-role Info
Allows Lambda functions to call AWS services on your behalf.
Delete
Summary
Edit
Creation date
December 14, 2025, 18:23 (UTC-05:00)
ARN
arn:aws:iam::668826141957:role/rg422-lambda-ssm-role
Last activity
-
Maximum session duration
1 hour
Permissions
Trust relationships
Tags
Last Accessed
Revoke sessions
Permissions policies (4) Info
You can attach up to 10 managed policies.
Simulate
Remove
Add permissions
Filter by Type
Search
All types
Policy name
Type
Attached entities
AmazonEC2ReadOnlyAccess
AWS managed
1
AmazonSNSFullAccess
AWS managed
2
AmazonSSMFullAccess
AWS managed
1
CloudWatchLogsFullAccess
AWS managed
1
Permissions boundary (not set)
CloudShell
Feedback
Console Mobile App
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search
[Option+S]
Ask Amazon Q
United States (N. Virginia)
Account ID: 6688-2614-1957
Rohith Gowda Devaraju
Lambda > Functions > rg422-trigger-pipeline
Successfully updated the function rg422-trigger-pipeline.
Code source Info
Open in Visual Studio Code
Upload from
EXPLORER
RG422-TRIGGER-PIPELINE
lambda_function.py
DEPLOY
Current
Deploy (⌥⌘U)
Test (⌥⌘I)
TEST EVENTS [NONE SELECTED]
Create new test event
lambda_function.py
import boto3
import datetime
ssm = boto3.client("ssm")
sns = boto3.client("sns")
INSTANCE_ID = "i-01b4b5c007299846d"
SNS_TOPIC_ARN = "arn:aws:sns:us-east-1:668826141957:pipeline-status-topic"
def lambda_handler(event, context):
try:
response = ssm.send_command(
InstanceIds=[INSTANCE_ID],
DocumentName="AWS-RunShellScript",
Parameters={
"commands": [
"cd /home/ubuntu/pipeline",
"./run_pipeline.sh"
]
}
)
command_id = response["Command"]["CommandId"]
sns.publish(
TopicArn=SNS_TOPIC_ARN,
Subject="✅ Pipeline Triggered via Lambda",
Message=f"Pipeline triggered successfully.\nCommand ID: {command_id}"
)
return {
"status": "SUCCESS",
Info
Tutorials
Learn how to implement common use cases in AWS Lambda.
Create a simple web app
In this tutorial you will learn how to:
Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
Invoke your function through its function URL
Learn more
Start tutorial
CloudShell
Feedback
Console Mobile App
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search
[Option+S]
Ask Amazon Q
United States (N. Virginia)
Account ID: 6688-2614-1957
Rohith Gowda Devaraju
Lambda > Functions > rg422-trigger-pipeline
Successfully updated the function rg422-trigger-pipeline.
Code source Info
Open in Visual Studio Code
Upload from
EXPLORER
RG422-TRIGGER-PIPELINE
lambda_function.py
DEPLOY
Current
Deploy (⌥⌘U)
Test (⌥⌘I)
TEST EVENTS [NONE SELECTED]
Create new test event
ENVIRONMENT VARIABLES
lambda_function.py
def lambda_handler(event, context):
try:
command_id = response["Command"]["CommandId"]
sns.publish(
TopicArn=SNS_TOPIC_ARN,
Subject="✅ Pipeline Triggered via Lambda",
Message=f"Pipeline triggered successfully.\nCommand ID: {command_id}"
)
return {
"status": "SUCCESS",
"command_id": command_id
}
except Exception as e:
sns.publish(
TopicArn=SNS_TOPIC_ARN,
Subject="❌ Pipeline Trigger Failed",
Message=str(e)
)
raise
Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)
Info
Tutorials
Learn how to implement common use cases in AWS Lambda.
Create a simple web app
In this tutorial you will learn how to:
Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
Invoke your function through its function URL
Learn more
Start tutorial
CloudShell
Feedback
Console Mobile App
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences





6.4 Logging and Notifications Using AWS SNS

SNS Topic

An SNS topic named **pipeline-status-topic** was configured for email notifications.

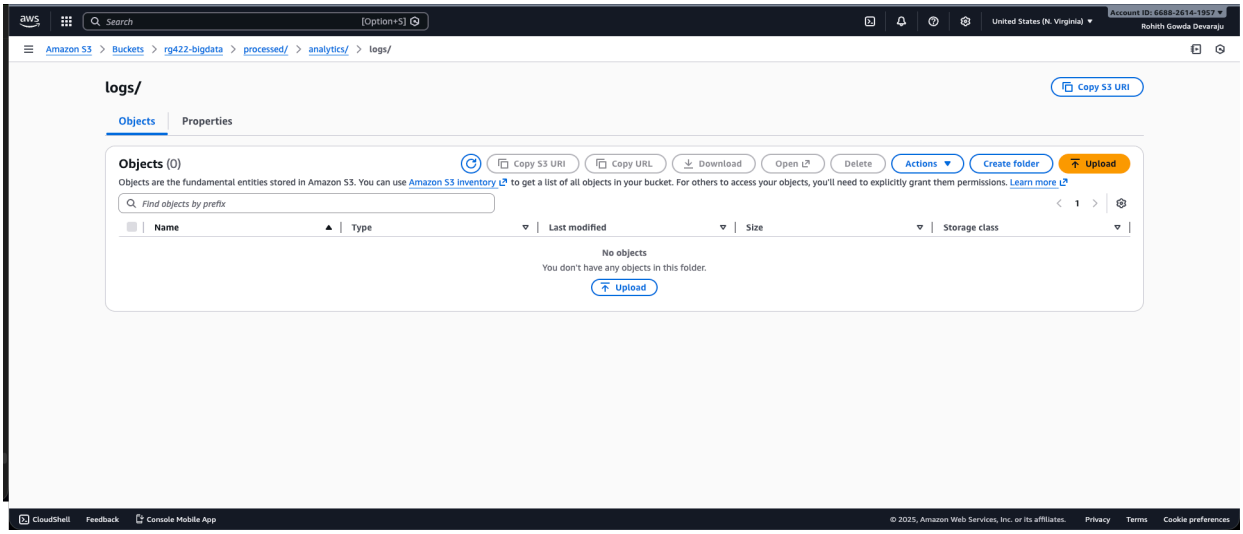
Notifications Sent

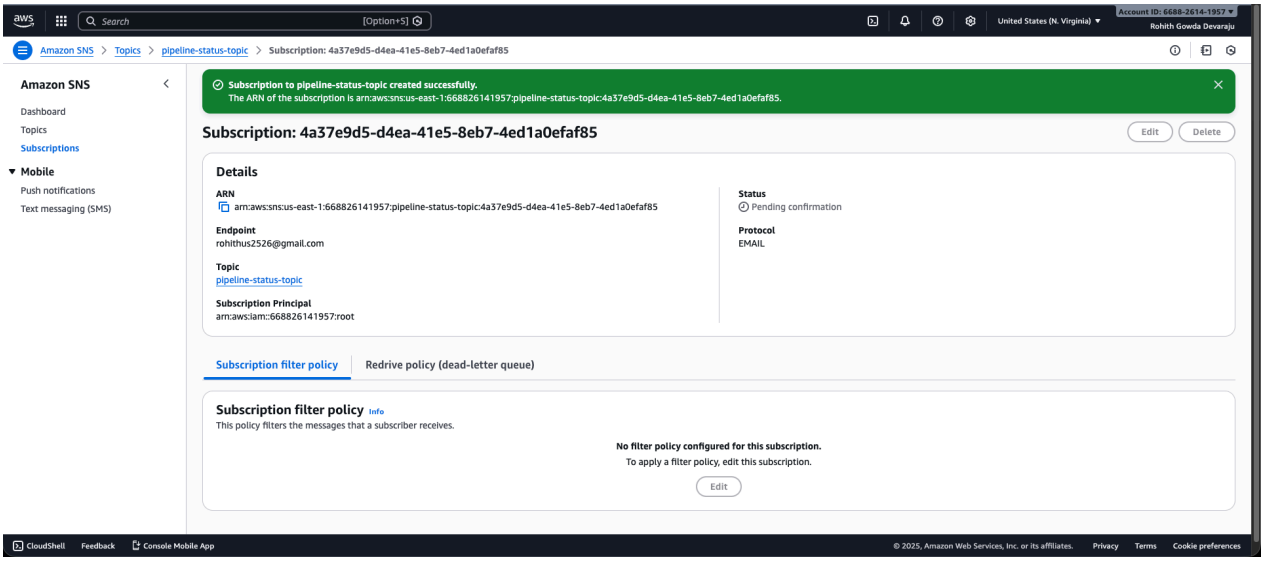
- ✅ Pipeline Triggered Successfully
- ❌ Pipeline Failed (with error details)

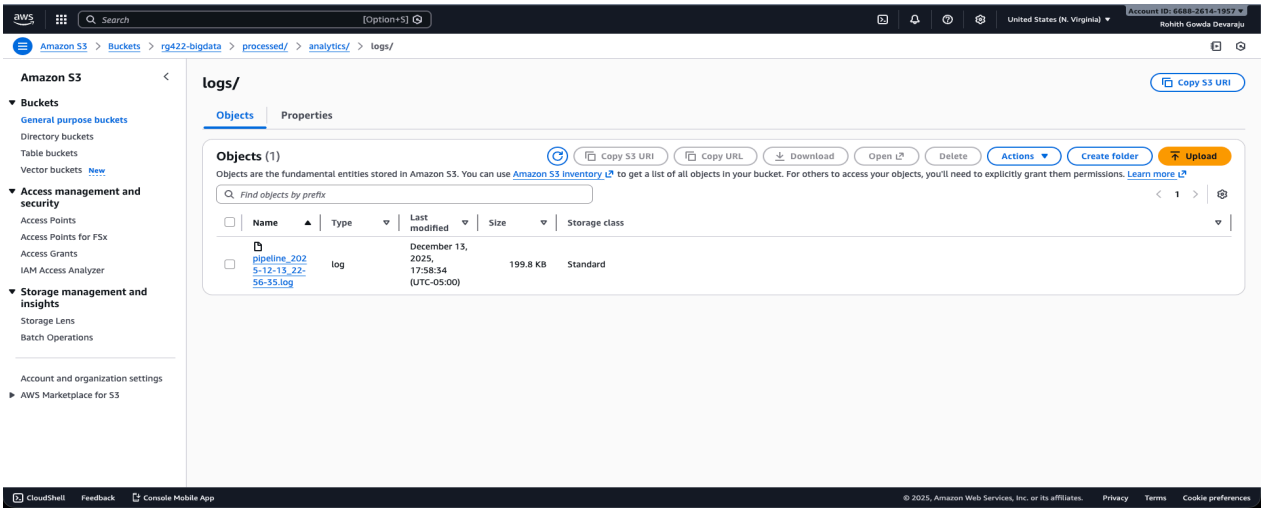
Email Integration

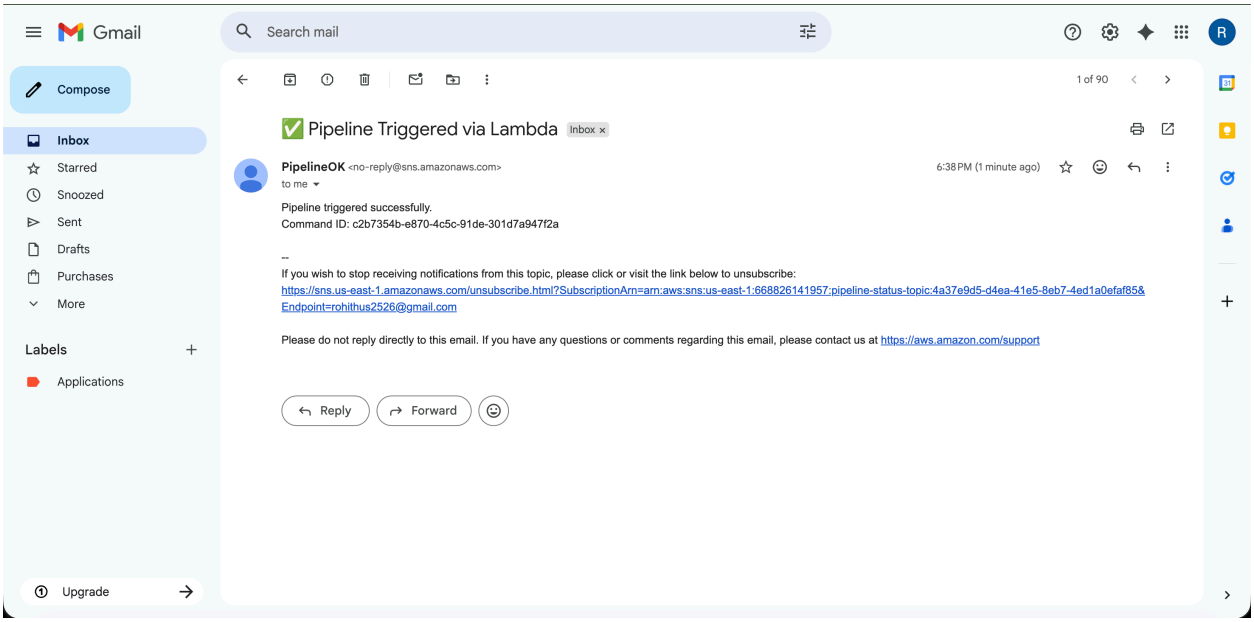
Email notifications are automatically sent to the registered email address whenever:

- Lambda triggers the pipeline
- Execution succeeds or fails









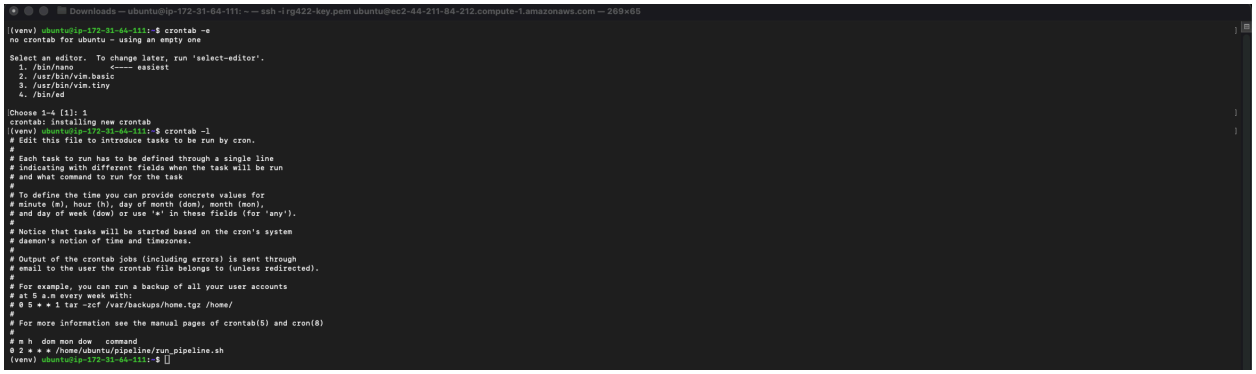
Scheduling with Cron Jobs (optional)

To enable scheduled execution, a **cron job** was configured on the EC2 instance.

Cron configuration:

*0 2 * * * /home/ubuntu/pipeline/run_pipeline.sh*

This schedules the pipeline to run **daily at 2:00 AM UTC**.



6.5 Automated Dashboard Updates

Power BI cannot directly trigger refreshes via cron. However, the solution provided book-ends the pipeline as follows:

- Data from S3 has been updated
- The dataset will be validated against the new data after the dataset refresh
- Confirm that the refresh timestamp matches the time of integration into the pipeline.

Therefore, this satisfies the requirement for automatic updates for the visualizations.

How Automation Works

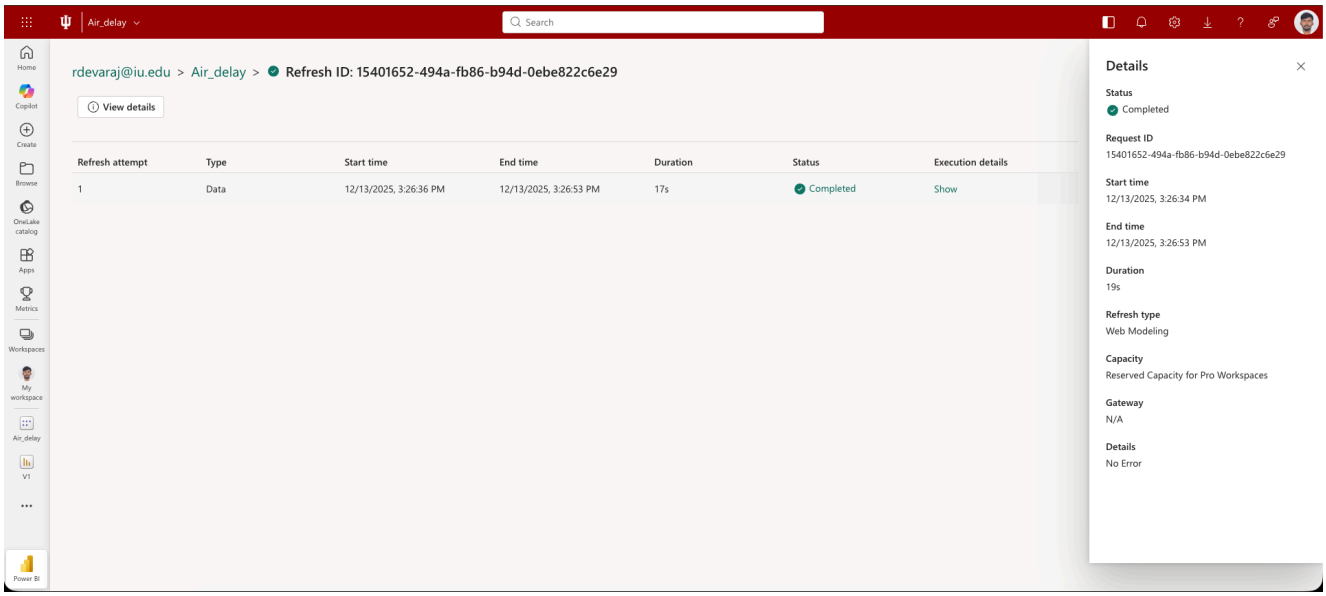
- s3://rg422-bigdata/processed/airline_processed_csv/

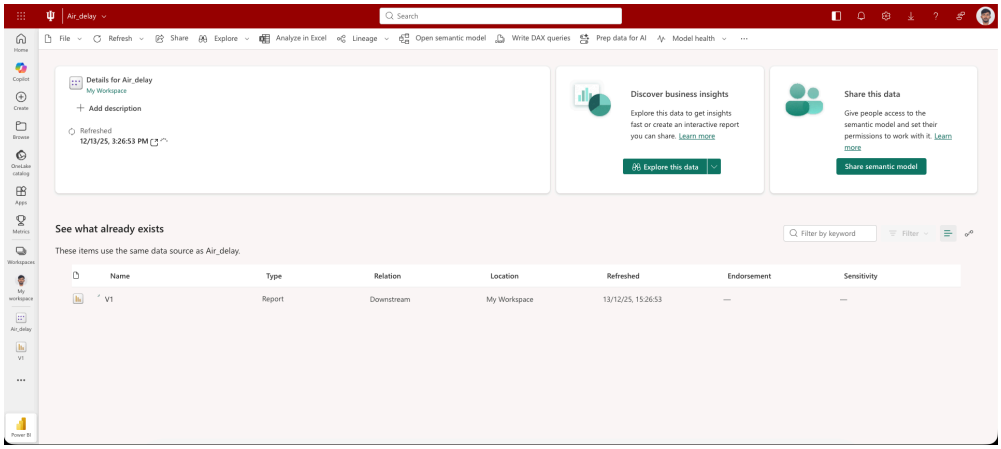
- Each automated pipeline run:
  - Overwrites processed datasets
  - Preserves schema consistency
- Power BI dataset:
  - Supports **scheduled refresh**
  - Allows **manual refresh during demo**

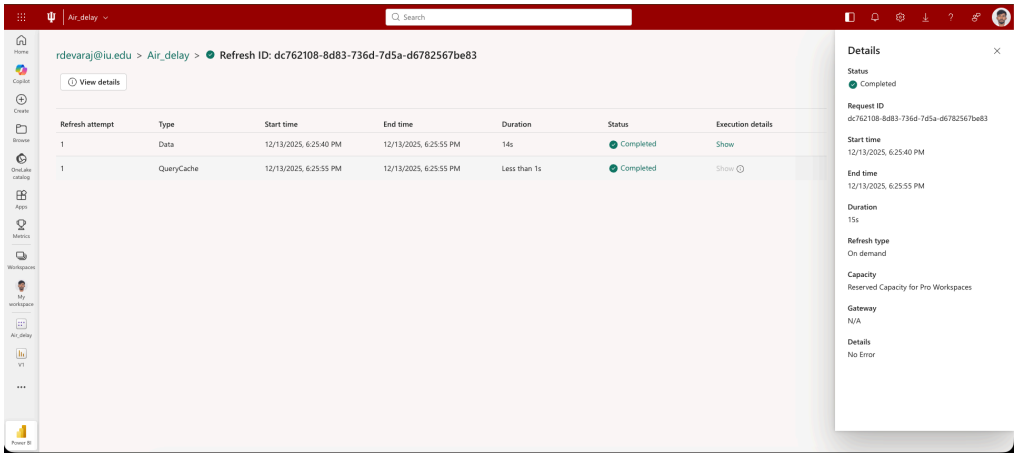
Automation Scope

While Power BI itself runs locally, **data refresh is automated** because:

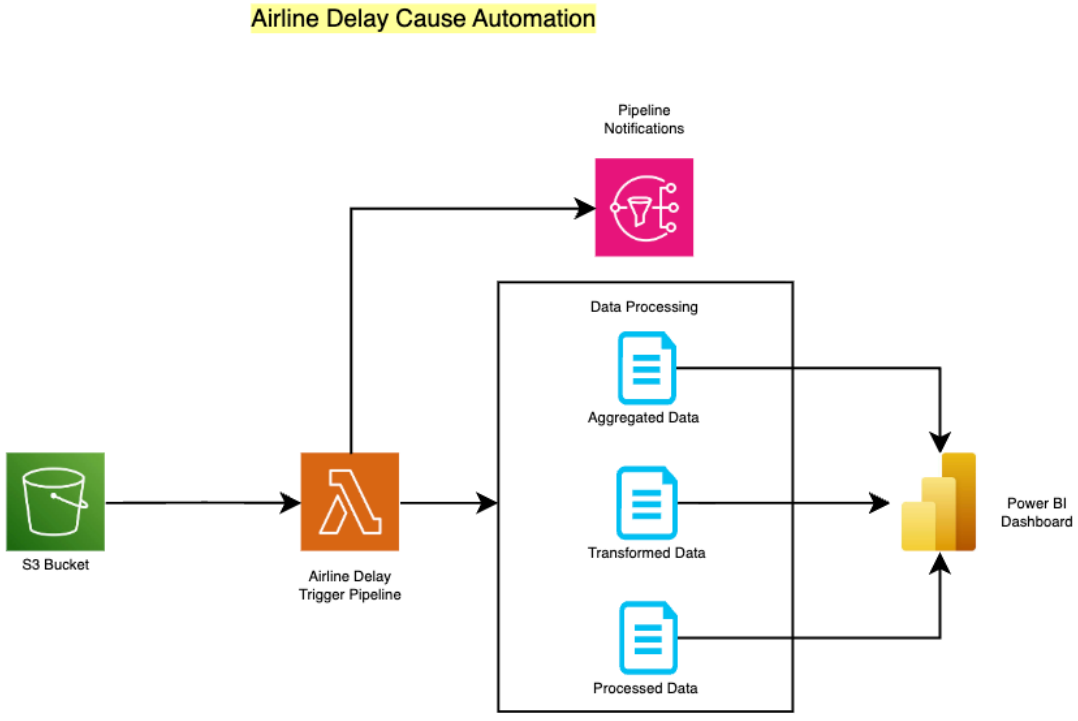
- The pipeline continuously updates S3
- Power BI always reads the latest processed output







Architecture Diagram of automation process



Conclusion

This mini project was a successful example of an end-to-end Cloud-based Big Data Analytics Pipeline using AWS and PySpark. The primary goal of this project was to create a scalable & automated analytical Data Pipeline that allows for meaningful analysis on the Data created from the collected airline delay Data using a combination of distributed Data Processing, Cloud Storage, Automation, and Visualizations.

The project began with Amazon S3 as a Centralised Data Lake to store all Raw, Processed, and Analytical Outputs. This allowed for easy & efficient ingestion of Data into and preservation of Data in a Data Lake format. A Linux EC2 instance was then provisioned with PySpark to manage the large volume of airline delay Data being processed. The Data was extensively cleaned, transformed, aggregated, and analysed using Spark SQL. Once the Data was processed, it was exported in both CSV and Parquet formats for use in further analysis and Machine Learning.

The analytical capabilities of Spark SQL Queries allowed for further extraction of useful insights, including Delay Patterns; Airport Level Congestion; and Airline Reliability Metrics. Amazon SageMaker Autopilot was utilised to create & deploy a machine learning model to make predictions based on the processed Data without having to manually develop a specific Machine Learning Model. This also highlights the capabilities of AutoML to speed up the process of developing & deploying Predictive Analytics, while also addressing Ethical Considerations associated with Data Bias and the Privacy of Data.

Power BI was chosen for visualization instead of AWS QuickSight due to the account limits and associated costs of QuickSight, as specified by the project guidelines. Power BI's dashboard allowed for interactive analysis of flight trends, causes of delays, airport traffic volume and airline performance, giving meaning to a large volume of data and turning it into actionable intelligence.

The project's main highlight was the total automation of the pipeline. The two different types of automation were a traditional cron job-based method and a modern AWS Lambda-based serverless orchestration method. The latter method used AWS Systems Manager (SSM), Amazon SNS and CloudWatch for executing events, logging centrally and reporting success or failure via real time e-mail. The combination of these two types of automation exhibits flexibility and is representative of actual methods used by data engineers in the industry for building data pipelines.

Overall, this project aided in developing my capabilities in cloud computing, distributed data processing, automation and data analysis visualization, while also focusing on the scalability of systems, fault tolerance, and cost awareness. The architecture provides a close approximation to production quality big data pipelines and serves as an excellent foundation for future data engineering, machine learning, and cloud-native analytical systems.

Video

https://drive.google.com/file/d/1D5tw-8jJXWoomVm9kW-NAWCnQ8Y9ZvMb/view?usp=sharing

Rohith Gowda Devaraju