Predicting Sales Conversion Likelihood

Complete ML Pipeline Implementation Technical Guide

Table

1. Project Summary

This project delivers an end-to-end MLOps pipeline for predicting customer lead conversion likelihood using a modern, scalable AWS cloud-native architecture. It automates the full machine learning lifecycle from data ingestion to prediction delivery, ensuring speed, accuracy, and reliability. The result is a robust lead scoring system that enables the sales team to make data-driven decisions, ultimately improving operational efficiency and increasing conversion rates.

Business Impact and Value Realization

Increased Efficiency
The solution automatically identifies high-potential leads, significantly reducing the time and effort spent on low-probability prospects. This prioritization enhances sales team productivity and accelerates the sales cycle.

Improved Conversion Rates
With accurate lead scoring, the sales team can focus their efforts on leads with the highest likelihood of conversion. This results in higher conversion rates and a better return on marketing and sales investments.

Scalability and Reliability
The architecture is built entirely on AWS managed services, making it highly scalable and fault-tolerant. It can handle growing volumes of data and users without manual intervention, ensuring consistent system availability and performance.

Key Architectural Achievements

Fully Automated Pipeline
The pipeline integrates services such as S3, AWS Glue, Amazon Redshift, SageMaker, MLflow, and Airflow (MWAA) to provide a fully automated workflow for data processing, model training, deployment, and monitoring. Airflow DAGs orchestrate the entire pipeline.

Robust Experiment Tracking with MLflow
All model experiments, including hyperparameters, performance metrics, and artifacts, are logged to MLflow Tracking Server hosted on SageMaker Studio. This enables experiment reproducibility, model comparison, and full traceability of model artifacts.

Proactive Monitoring with Evidently AI
The pipeline includes automated drift detection between training data and new inference data using Evidently AI. If drift is detected, alerts are sent through Amazon SNS, and retraining workflows can be triggered using Airflow.

Modular and Decoupled Architecture
The system architecture is modular and scalable. Services like Glue (ETL), Redshift (data

warehousing), SageMaker (model training), and Flask (API endpoint) are loosely coupled, making the pipeline easy to maintain and scale.

Technical Innovation and MLOps Maturity

This project demonstrates advanced MLOps capabilities by moving beyond one-time model training to a fully operationalized system with automated monitoring, retraining, and governance. It reflects MLOps maturity level 3, characterized by:

- CI/CD pipelines for machine learning using Airflow
- Model versioning, reproducibility, and governance with MLflow
- Real-time data drift detection with Evidently AI
- Automated retraining workflows

The result is a production-grade machine learning solution embedded in core business processes.

2. Business Problem

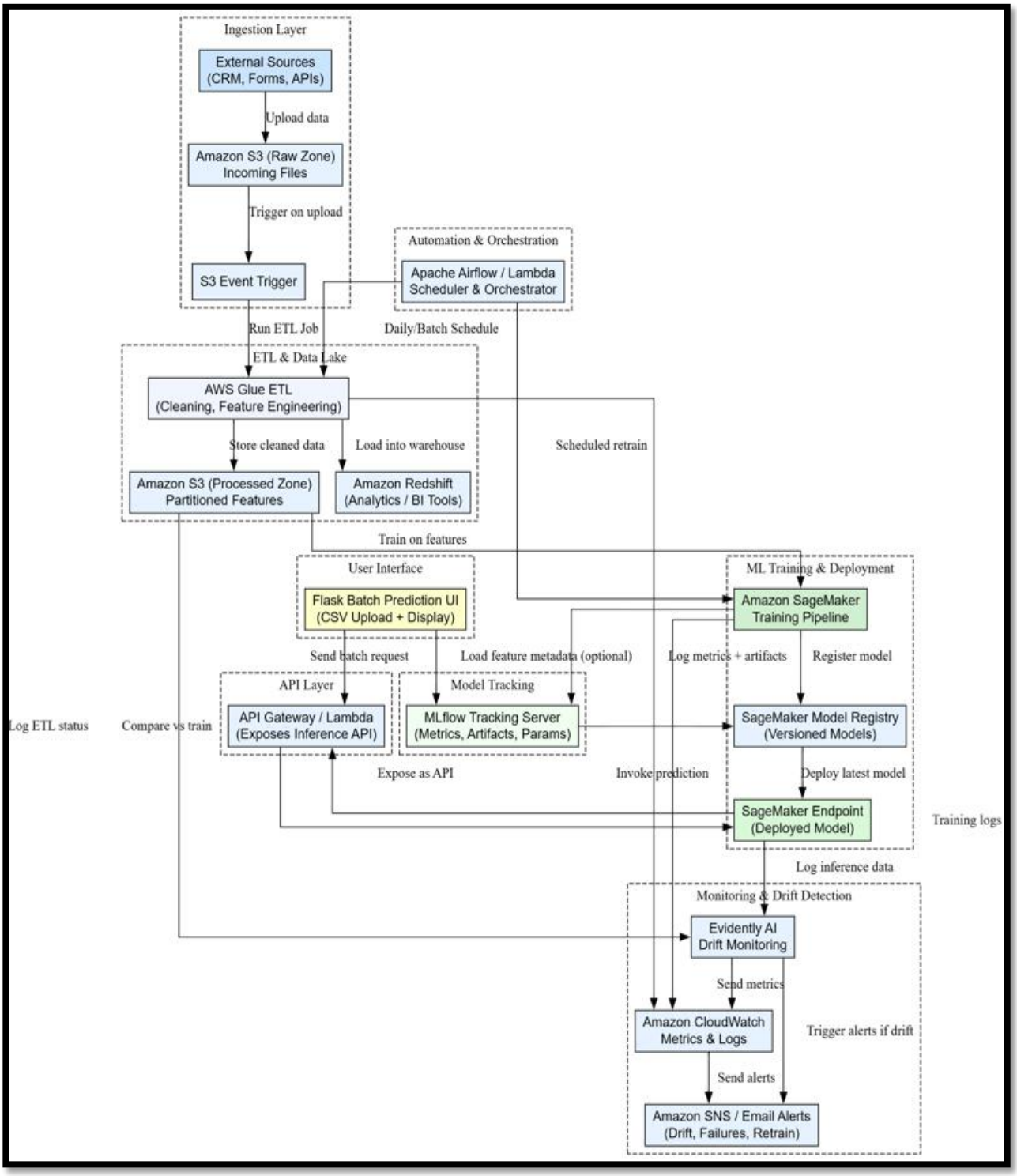
Problem Statement

Sales teams often deal with a large volume of incoming leads but lack reliable mechanisms to identify which leads are worth pursuing. Manual lead scoring is time-consuming, inconsistent, and prone to bias. This project solves the problem by building a machine learning-powered lead scoring system that uses historical behavioral and demographic data to predict the likelihood of conversion. The system supports both real-time and batch scoring and is designed to adapt as new data becomes available.

Key Requirements

- Scalable ingestion and transformation of raw lead data
- Scheduled model training and automated retraining
- Drift detection and model monitoring
- Accessible API interface for scoring leads in real time or batch

2. Architecture Diagram



2.1. Technology Stack

Layer	Service/Tool	Purpose
Storage	Amazon S3	Raw and processed data, models, reports
ETL	AWS Glue	Data cleaning, schema alignment
Data Warehouse	Amazon Redshift	Stores processed tabular data
ML Training	SageMaker Studio Notebook	Preprocessing, modeling, evaluation
Tracking	MLflow	Experiment tracking, model registry
Model Deployment	Flask + ngrok	Interactive batch UI
Orchestration	Amazon MWAA (Airflow)	Scheduled DAGs for ingestion, drift, retrain
Monitoring	Evidently AI	Data drift and model monitoring

2.2 Common Technical Prerequisites & Tips

Area	Tip/Precaution
AWS IAM	Ensure you attach correct roles to SageMaker and Glue for S3/Redshift access
MLflow Access	If using MLflow in SageMaker Studio, attach a custom policy (e.g. SageMakerMLflowAccessPolicy)
S3 Buckets	Use different folders: /raw, /processed, /models, /reports
ngrok	Use a registered ngrok account for HTTPS public URL
MWAA	Ensure your DAGs are in the /dags folder inside the Airflow S3 bucket

3. Data Ingestion Using AWS S3 → Glue → Redshift

Architecture Overview

This document outlines the implementation of a robust data ingestion pipeline that processes lead data from multiple sources through the following flow:

Raw Data (CSV) → Amazon S3 → AWS Glue → Amazon Redshift → Analytics/ML

Key Components:

- **Amazon S3**: Raw and processed data storage
- **AWS Glue Crawler**: Schema discovery and cataloging
- **AWS Glue ETL Jobs**: Data transformation and loading
- **Amazon Redshift**: Data warehouse for analytics

Prerequisites

Required AWS Services

- Amazon S3
- AWS Glue (with Visual ETL Editor)
- Amazon Redshift
- AWS IAM

Required Permissions

- Administrative access to AWS Console
- Ability to create IAM roles and policies
- VPC and Security Group management permissions

Tools Needed

- AWS CLI (optional but recommended)

CSV data files for ingestion

3.1 S3 Bucket Setup

Step 1: Create S3 Bucket

Via AWS Console:

1. Navigate to **S3 Console** → **Create bucket**
2. **Bucket name**: leaddata23
3. **Region**: Select your preferred region (e.g., us-east-1)
4. **Block Public Access**: Keep all options checked
5. **Bucket Versioning**: Enable (recommended)
6. **Default Encryption**: Enable with SSE-S3
7. Click **Create bucket**

[Screenshot placeholder: S3 bucket creation screen]

Via AWS CLI:

bash

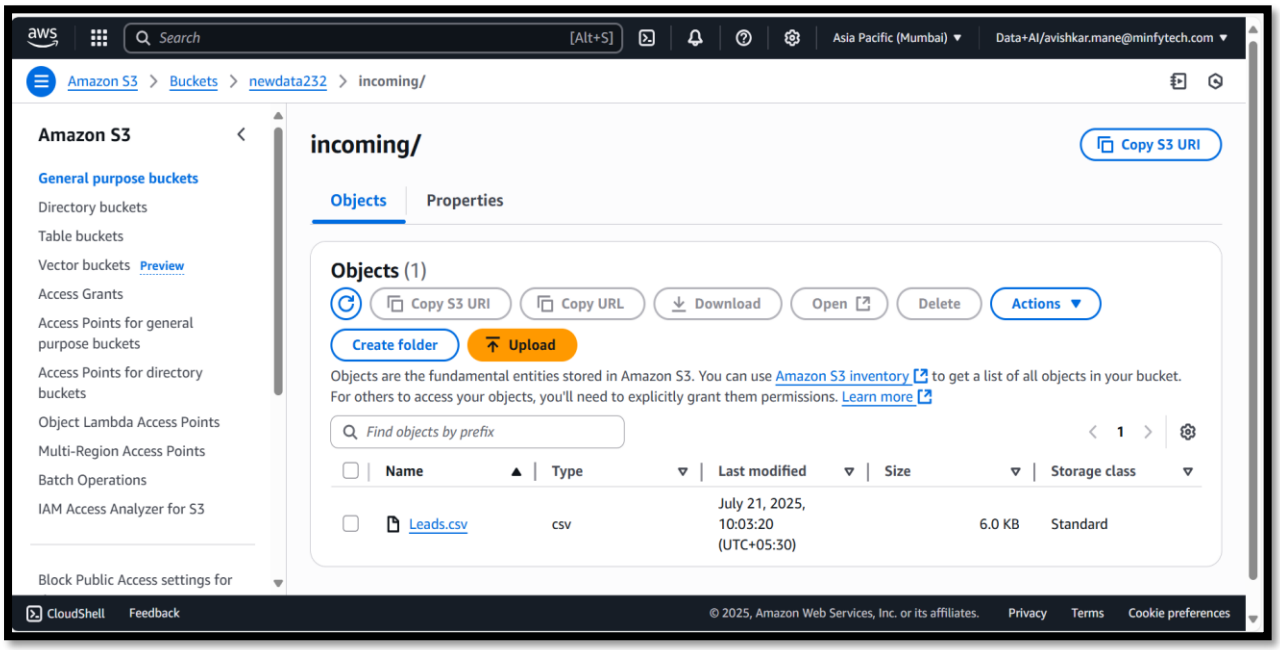
```
# Create the bucket
aws s3 mb s3://leaddata23 --region us-east-1


# Enable versioning
aws s3api put-bucket-versioning \
  --bucket leaddata23 \
  --versioning-configuration Status=Enabled


# Enable default encryption
aws s3api put-bucket-encryption \
  --bucket leaddata23 \
  --server-side-encryption-configuration '{
    "Rules": [
      {
        "ApplyServerSideEncryptionByDefault": {
          "SSEAlgorithm": "AES256"
        }
```

```
        }
    ]
}'
```

Step 2: Create Folder Structure

Via AWS Console:

1. Go to **S3 Console** → **leaddata23** bucket
2. Click **Create folder**
3. Create the following folders:
    - raw/
    - processed/
    - logs/
    - scripts/



Via AWS CLI:

bash

```
# Create folder structure (folders are created when objects are uploaded)
aws s3api put-object --bucket leaddata23 --key raw/.keep
aws s3api put-object --bucket leaddata23 --key processed/.keep
aws s3api put-object --bucket leaddata23 --key logs/.keep
```

aws s3api put-object --bucket leaddata23 --key scripts/.keep

Best Practice: Use consistent file naming like Leads_Month.csv to help automate with Glue bookmarks later.

3.2 Load Processed Data into Amazon Redshift via AWS Glue

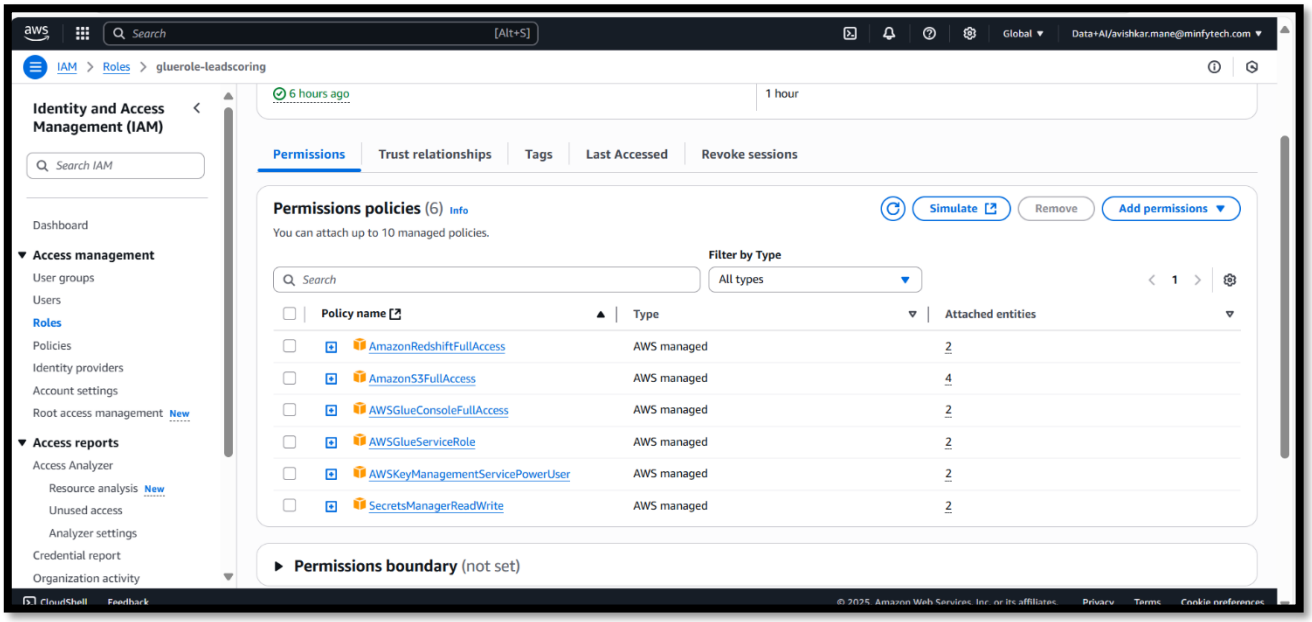
IAM Roles and Policies

Glue Service Role

Step 1: Create Glue Service Role

Via AWS Console:

1. Navigate to **IAM Console** → **Roles** → **Create role**
2. **Trusted entity type**: AWS service
3. **Use case**: Glue
4. **Role name**: AWSGlueServiceRole-leadETL



Step 2: Attach Required Policies

Managed Policies to Attach:

- AWSGlueServiceRole

- AmazonS3FullAccess
- AmazonRedshiftFullAccess

Custom Inline Policy for Enhanced Security:

json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3BucketAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::leaddata23",
        "arn:aws:s3:::leaddata23/*"
      ]
    },
    {
      "Sid": "GlueCatalogAccess",
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase",
        "glue:GetDatabases",
```

```
            "glue:CreateDatabase",
            "glue:GetTable",
            "glue:GetTables",
            "glue:CreateTable",
            "glue:UpdateTable",
            "glue:GetPartition",
            "glue:GetPartitions",
            "glue:CreatePartition",
            "glue:UpdatePartition",
            "glue:BatchCreatePartition",
            "glue:BatchUpdatePartition"
        ],
        "Resource": "*"
    },
    {
        "Sid": "RedshiftAccess",
        "Effect": "Allow",
        "Action": [
            "redshift:DescribeClusters",
            "redshift:GetClusterCredentials"
        ],
        "Resource": "*"
    },
    {
        "Sid": "VPCAccess",
        "Effect": "Allow",
        "Action": [
            "ec2:CreateNetworkInterface",
```

```
            "ec2:DeleteNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups",
            "ec2:DescribeVpcs"
        ],
        "Resource": "*"
    }
  ]
}
```

Step 3: Trust Policy for Glue Role

json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "glue.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Objective:

To create a robust, production-ready data pipeline that loads cleaned lead data into Amazon Redshift. This section outlines the steps to:

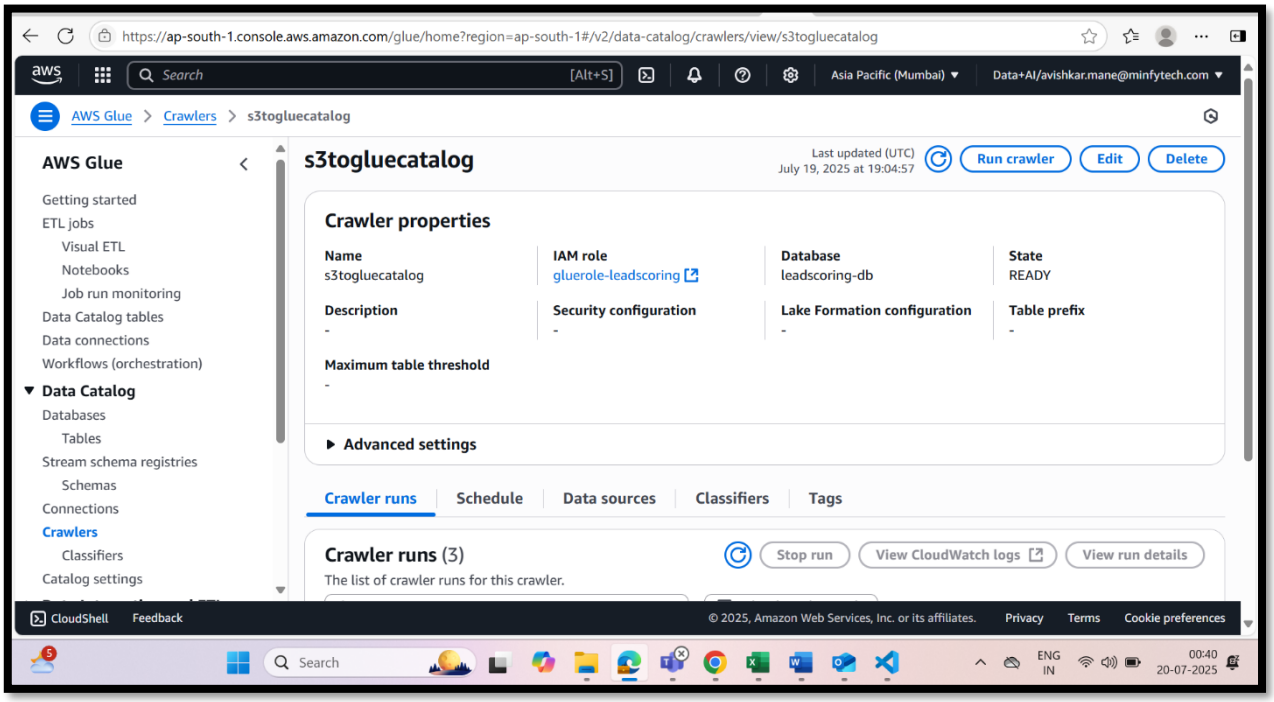
- Create a Glue Crawler to infer schema from processed Parquet data
- Configure an ETL job that reads from the Data Catalog
- Transform the data to match a unified schema
- Load it into Redshift using Glue + Redshift Connector

3.3 Create AWS Glue Crawler

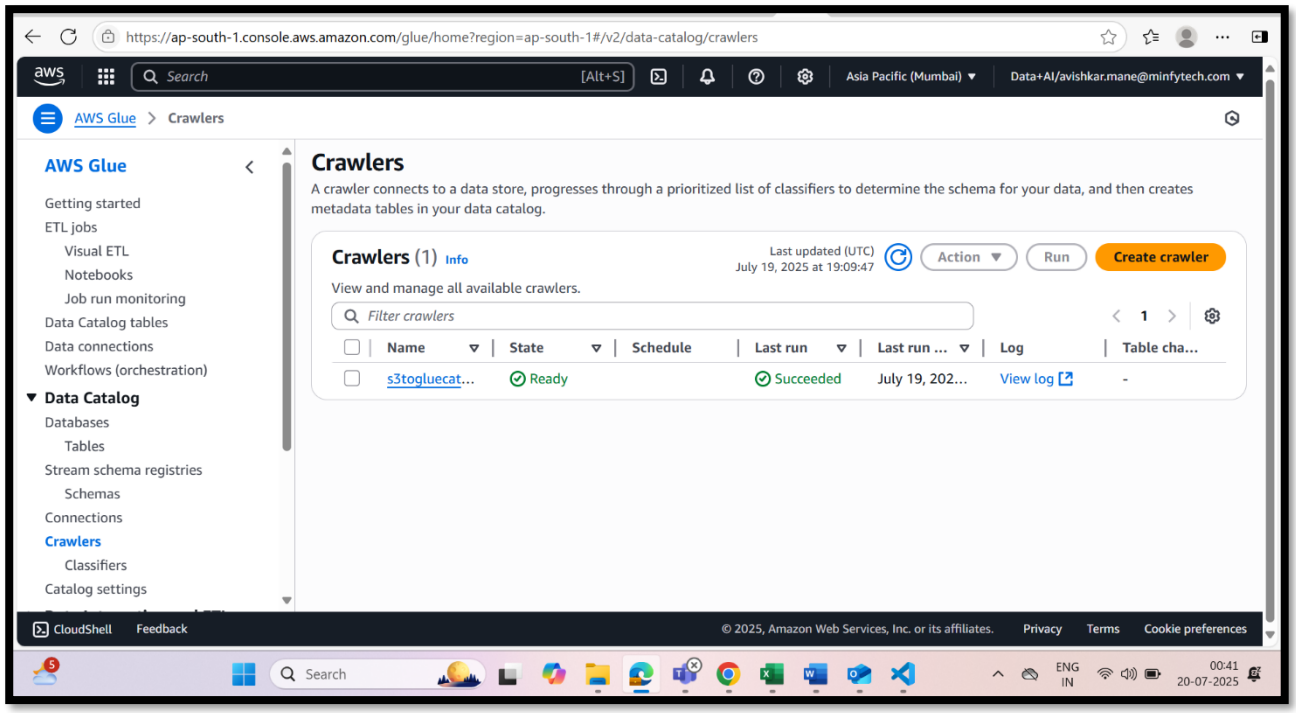
1. Navigate to AWS Glue Console → Crawlers → Add Crawler
2. Name: lead-crawler
3. Source Type: Data stores → Amazon S3
4. S3 Path: s3://leaddata23/processed/
5. Choose a Crawler IAM Role: Use or create a role with S3 and Glue access (e.g., AWSGlueServiceRole-leadETL)
6. Output Database: Create or choose an existing Glue Database (e.g., leaddb)
7. Crawler Schedule: Select "Run on demand"
8. Run the crawler

Result: A new table (e.g., leads_cleaned) will appear in the Glue Data Catalog under the specified database with inferred schema.

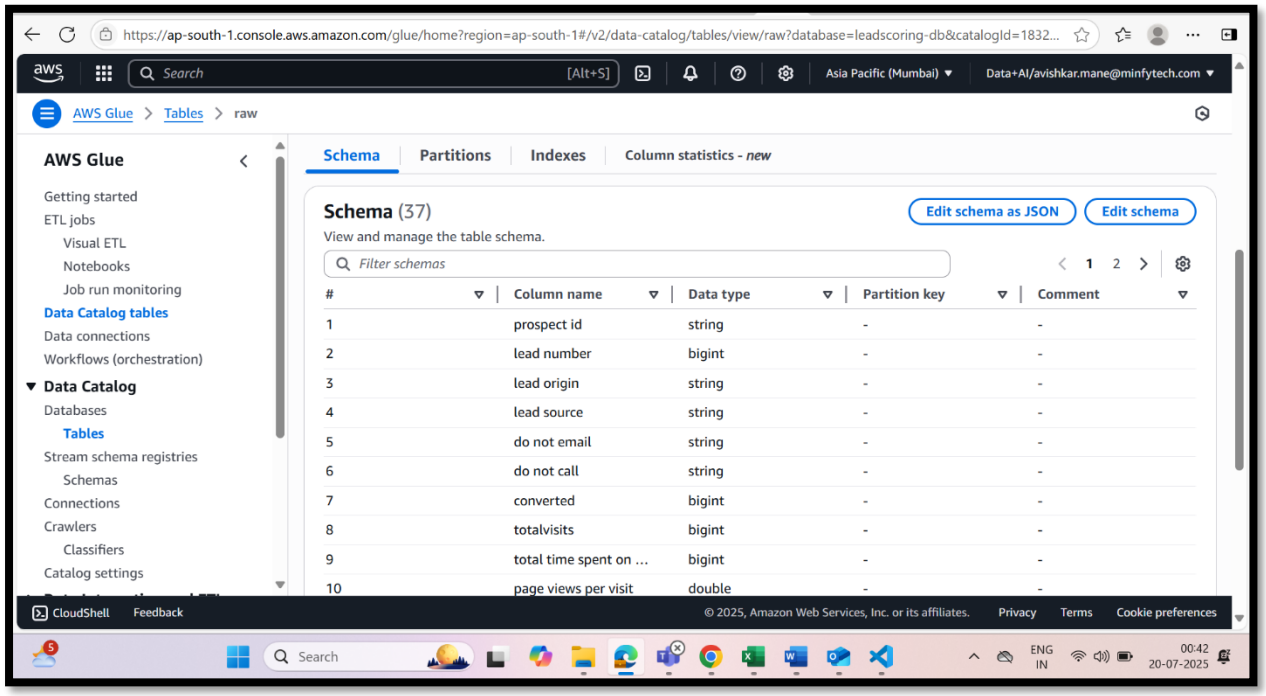
Glue Crawler config summary



- **Crawler run completion screen**



- **Table schema in the Glue Data Catalog**



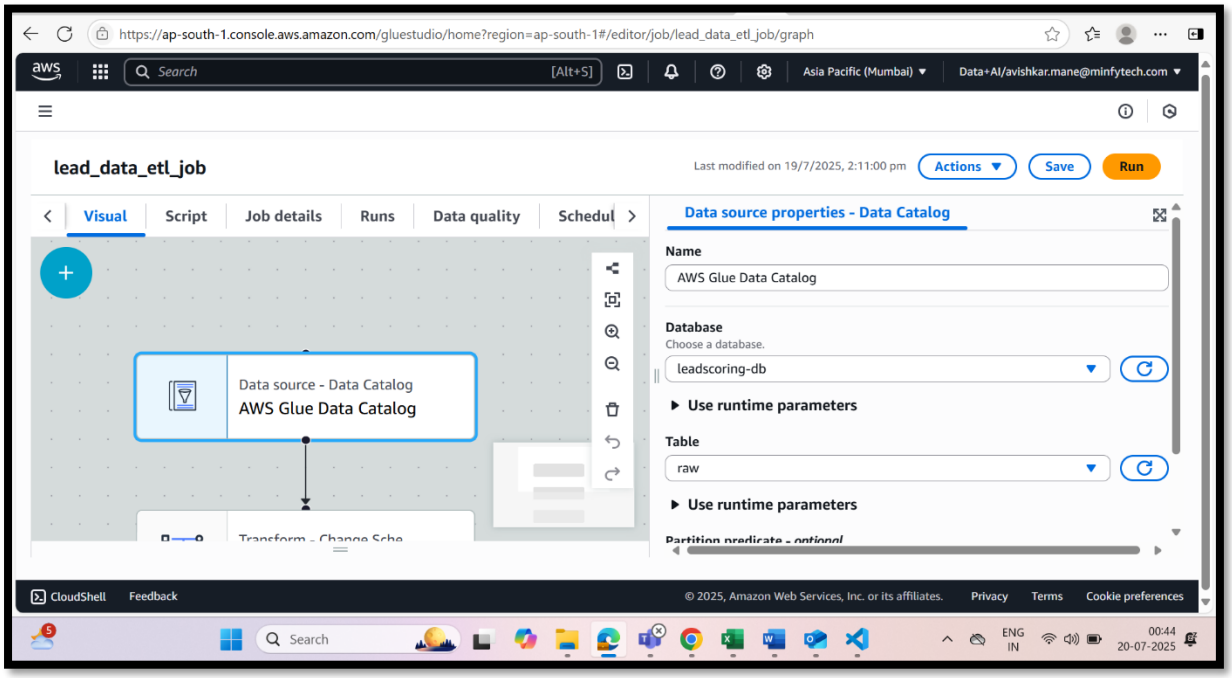
3.4 Configure AWS Glue Visual ETL Job

Once the schema is available in the Data Catalog, we use that as our source to build a visual Glue job.

1. Open AWS Glue → Jobs → Create Job (Visual with a source and target)

2. Set Source:

- Source type: AWS Glue Data Catalog

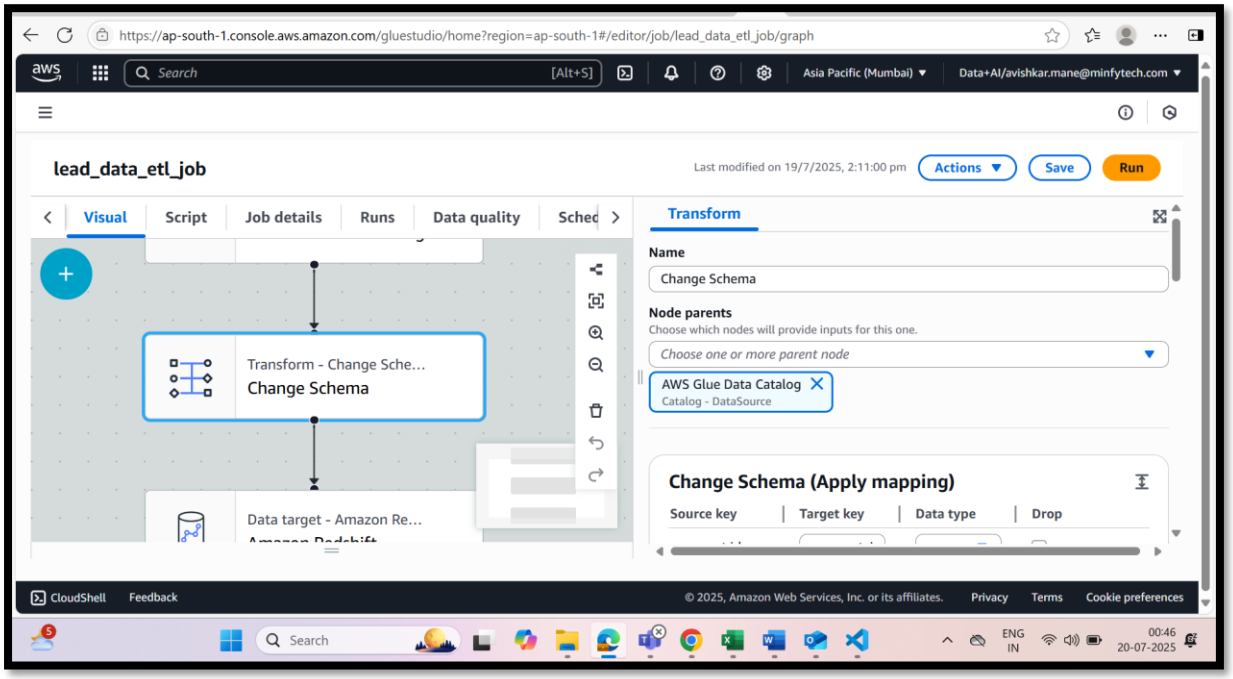


- Database: leaddb
- Table: leads_cleaned

This ensures that we ingest already-schema-inferred data.

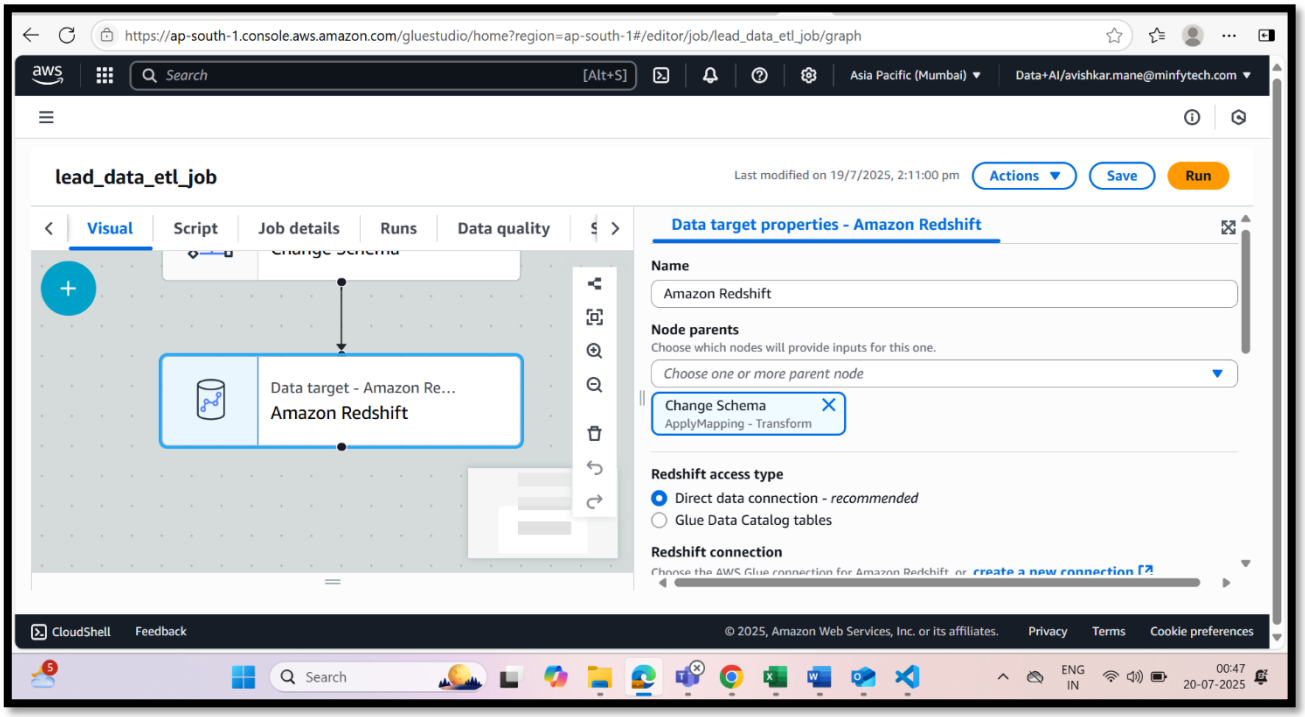
Apply Transformations:

- Drag in a Change Schema or ApplyMapping node.
- Select "Choose another schema" option to map current schema to an ideal unified schema (since raw data might come from different sources).
- Rename or drop any inconsistent/unwanted fields.



Configure Target

1. Add **Target** node
2. **Target type**: Amazon Redshift
3. **Connection**: redshift-glue-conn
4. **Schema**: leads
5. **Table**: leads_final
6. **Write mode**: Append (or Overwrite for full refresh)



3.5 Configure Redshift Connection for Glue

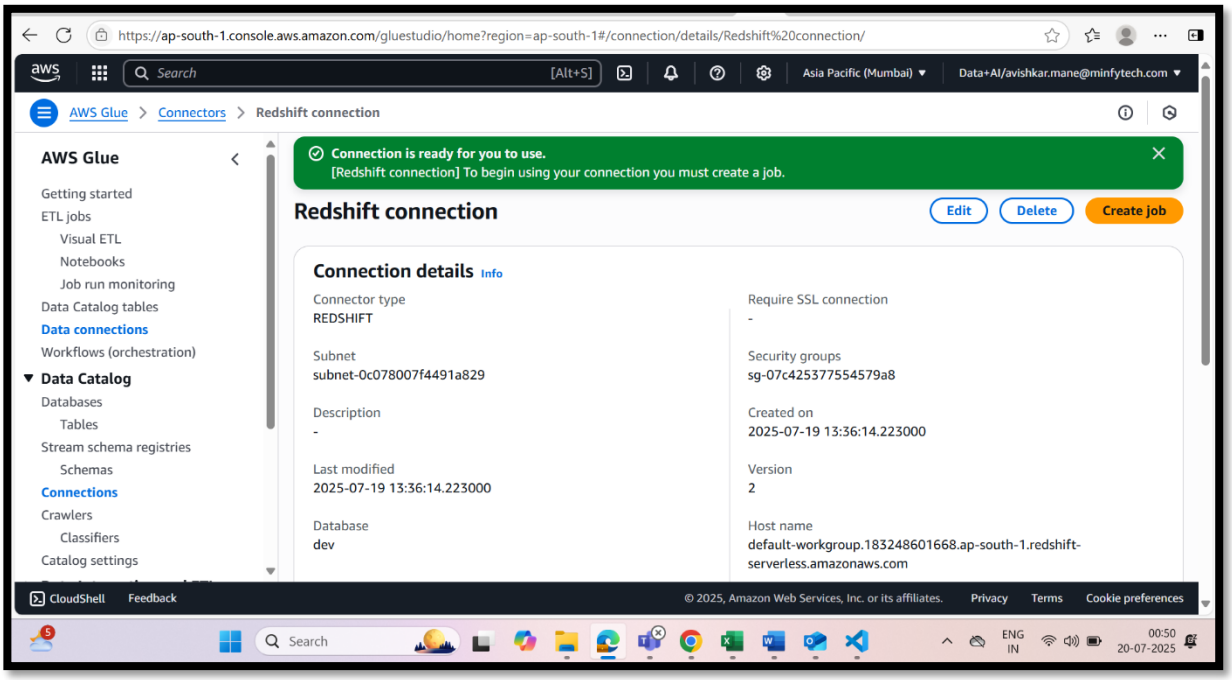
Before running the job, establish a secure connection between Glue and Redshift.

1. Go to AWS Glue → Connections → Create connection

- Name: redshift-glue-conn
- Type: Amazon Redshift
- Connection Method: Redshift
- Username/Password: Use Redshift credentials

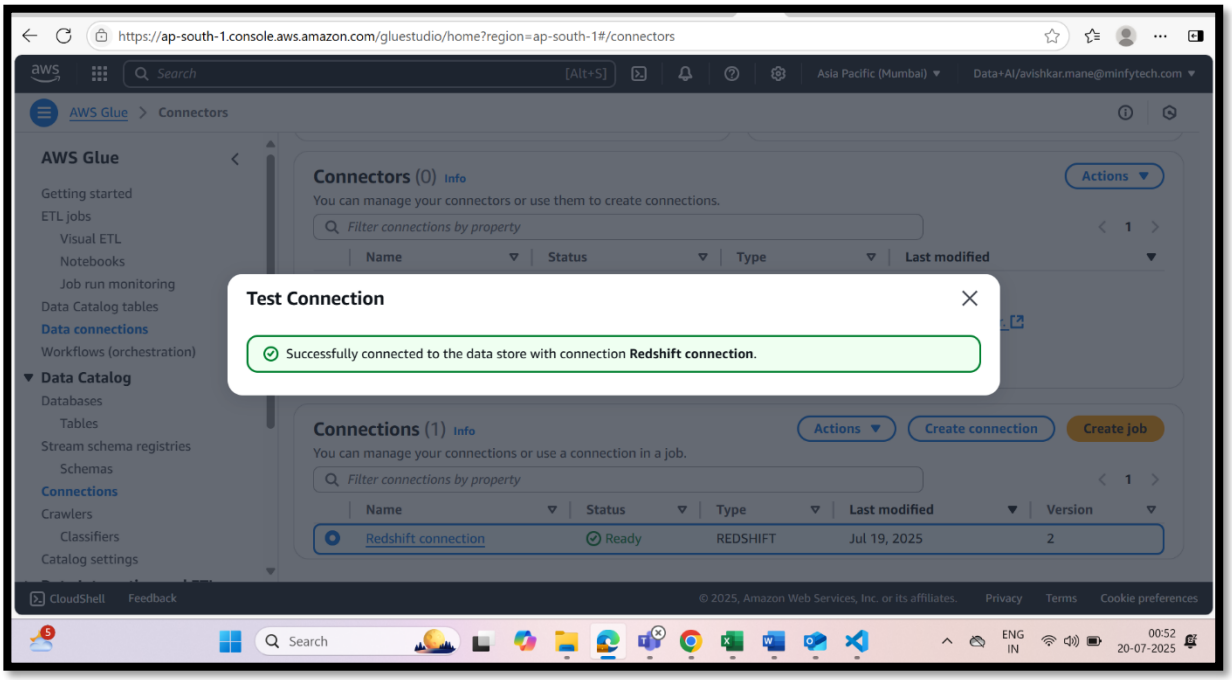
- VPC and Subnet: Choose the same VPC/subnet as your Redshift cluster
- Security Group: Allow inbound access from Glue (port 5439)

Screenshots:



- Security group rules (port 5439 open for Glue)

Check Connection Before running the ETL Job by selecting Connection and after in action there is option test connection



D. Configure Job Details

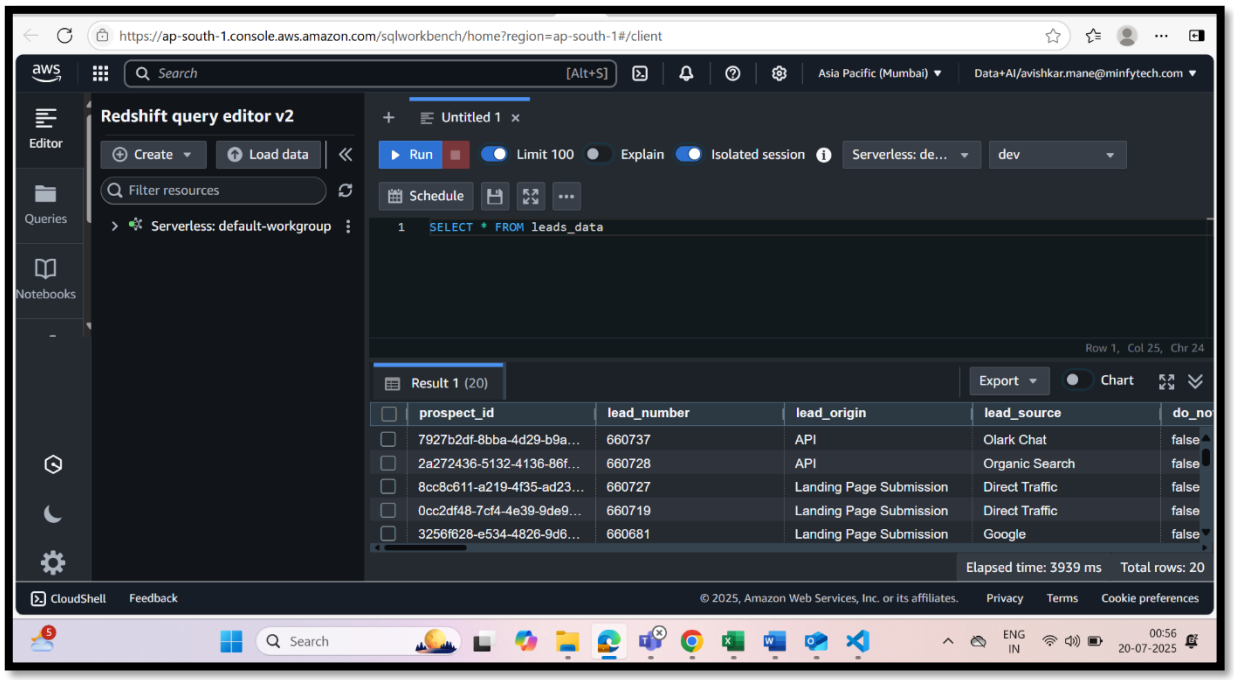
1. Job Name: glue-redshift-etl-job
2. IAM Role: AWSGlueServiceRole-leadETL
3. Connection: Select redshift-glue-conn from dropdown
4. Enable Job Bookmarking: Yes (to skip reprocessing)
5. Script Location (Optional): Store the job script in S3 for version control

E. Run the Glue ETL Job

1. Click Run
2. Monitor job progress from the Jobs tab
3. On success, navigate to Amazon Redshift Query Editor v2 to validate

SELECT * FROM leads_final;

Expected Output: Fully transformed and cleaned lead data in tabular form in your Redshift table (leads_final)



Required IAM Policies:

For Redshift IAM Role (used in external schema)

- AmazonS3ReadOnlyAccess
- AWSGlueConsoleFullAccess

3.6 Redshift Service Role

Step 1: Create Redshift Service Role

Role name: RedshiftServiceRole-leadETL

Managed Policies:

- AmazonS3ReadOnlyAccess

Custom Inline Policy:

json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3AccessForRedshift",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::leaddata23",
        "arn:aws:s3:::leaddata23/*"
      ]
    }
```

```
  ]
}
```

Step 2: Trust Policy for Redshift Role

json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "redshift.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3.7 Troubleshooting:

3.7.1 Common Issues and Solutions

Connection Timeout Errors

Error: Connection timeout to Redshift cluster

Solutions:

- Verify security group allows port 5439 from Glue
- Check VPC routing and NAT gateway configuration
- Ensure Glue job runs in same VPC as Redshift

bash

```
# Check security group rules
aws ec2 describe-security-groups --group-ids sg-12345678
```

Schema Mismatch Errors

Error: Column 'xyz' does not exist in target table

Solutions:

- Re-run crawler after data schema changes
- Update ApplyMapping transform in Glue job
- Verify Redshift table schema matches expected output

sql

-- Check Redshift table schema

SELECT column_name, data_type, is_nullable

FROM information_schema.columns

WHERE table_schema = 'leads'

AND table_name = 'leads_final'

ORDER BY ordinal_position;

Job Fails with Out of Memory

Error: Container killed due to exceeding memory limits

Solutions:

- Increase worker type (G.1X → G.2X)
- Increase number of workers
- Enable dynamic allocation
- Partition large datasets

Duplicate Records in Target

Solutions:

- Enable job bookmarks
- Use MERGE/UPSERT logic instead of INSERT
- Add timestamp-based filters

sql

-- Create UPSERT procedure

CREATE OR REPLACE PROCEDURE leads.upsert_leads(

```
    p_lead_id VARCHAR(50),
    p_email VARCHAR(255),
    -- other parameters
)
AS $$
BEGIN
    UPDATE leads.leads_final
    SET email = p_email, updated_date = CURRENT_TIMESTAMP
    WHERE lead_id = p_lead_id;

    IF NOT FOUND THEN
        INSERT INTO leads.leads_final (lead_id, email, created_date)
        VALUES (p_lead_id, p_email, CURRENT_TIMESTAMP);
    END IF;
END;
$$ LANGUAGE plpgsql;
```

3.7.2 Monitoring and Alerting Setup

CloudWatch Alarms:

```
bash
# Job failure alarm
aws cloudwatch put-metric-alarm \
    --alarm-name "GlueJobFailure-LeadETL" \
    --alarm-description "Alert when Glue ETL job fails" \
    --metric-name glue.driver.aggregate.numFailedTasks \
    --namespace AWS/Glue \
    --statistic Sum \
    --period 300 \
    --threshold 1 \
```

```
  --comparison-operator GreaterThanOrEqualToThreshold \
  --evaluation-periods 1

# Long running job alarm
aws cloudwatch put-metric-alarm \
  --alarm-name "GlueJobLongRunning-LeadETL" \
  --alarm-description "Alert when job runs longer than expected" \
  --metric-name glue.driver.ExecutorAllocationManager.executors.numberAllExecutors \
  --namespace AWS/Glue \
  --statistic Average \
  --period 3600 \
  --threshold 0 \
  --comparison-operator GreaterThanThreshold \
  --evaluation-periods 2
```

3.8 Security Best Practices

1. Data Encryption

- Enable S3 bucket encryption (SSE-S3 or SSE-KMS)
- Use encrypted connections to Redshift
- Enable Redshift encryption at rest

2. Access Control

- Follow principle of least privilege for IAM roles
- Use VPC endpoints for S3 access
- Restrict network access using security groups

3. Data Privacy

- Implement PII masking in Glue transforms
- Set up data retention policies

- Enable CloudTrail logging for audit

4. Monitoring

- Enable detailed CloudWatch logging
- Set up SNS notifications for job failures
- Implement custom metrics for data quality

3.8.1 Cost Optimization

1. Glue Job Optimization

- Use appropriate worker types for workload
- Enable auto scaling for Glue jobs
- Implement job bookmarking to avoid reprocessing

2. Redshift Optimization

- Use appropriate node types and sizing
- Enable automatic WLM
- Implement table maintenance (VACUUM, ANALYZE)

3. S3 Storage Optimization

- Use S3 Intelligent Tiering
- Implement lifecycle policies
- Compress files before upload

3.8.2 Maintenance and Operations

Weekly Tasks:

- Review job execution metrics
- Check data quality reports
- Monitor storage usage

Monthly Tasks:

- Review and optimize query performance
- Update IAM policies if needed

- Backup Glue job scripts

Quarterly Tasks:

- Review architecture for cost optimization
- Update security configurations

Disaster recovery testing

Final Outcome:

- Glue Crawler stores inferred schema in Data Catalog
- ETL Job reads from catalog and applies unified schema
- Job connects securely to Redshift via VPC
- Transformed data is inserted into leads_final table in Redshift

4. **ML Model Training & Experiment Tracking: SageMaker + MLflow**

This part of document outlines the implementation of a complete ML pipeline using Amazon SageMaker for model training and MLflow for experiment tracking.

Core Components:

- **Amazon SageMaker Studio**: ML development environment with Jupyter notebooks
- **MLflow Tracking Server**: Experiment tracking and model registry
- **Amazon S3**: Data storage and model artifacts
- **Flask Application**: Model serving interface
- **Evidently AI**: Data drift detection and monitoring

Data Flow: Lead Data (Redshift) → SageMaker Studio → Model Training → MLflow Registry → Flask API → Predictions

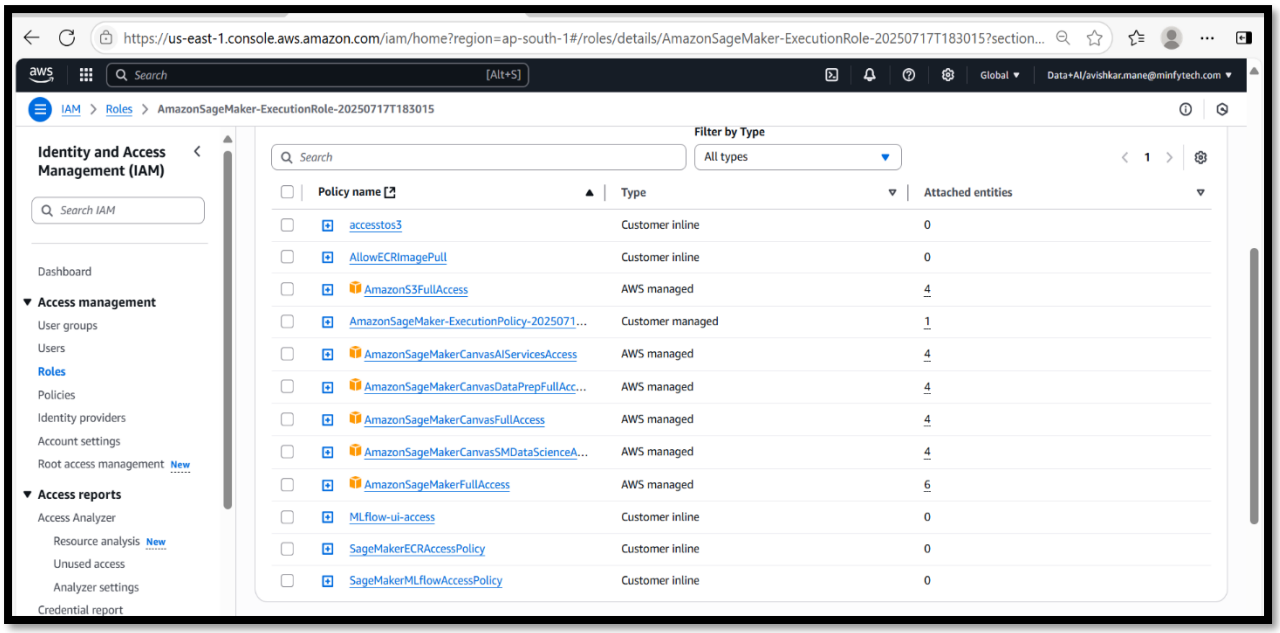
4.1 IAM Roles and Policies

SageMaker Execution Role

Step 1: Create SageMaker Execution Role

Via AWS Console:

1. Navigate to **IAM Console** → **Roles** → **Create role**
2. **Trusted entity**: AWS service
3. **Service**: SageMaker
4. **Use case**: SageMaker - Execution
5. **Role name**: SageMakerExecutionRole-MLPipeline



Step 2: Attach Required Managed Policies

Required Managed Policies:

- AmazonSageMakerFullAccess
- AmazonS3FullAccess
- AmazonRedshiftFullAccess
- AmazonEC2ContainerRegistryFullAccess

Step 3: Custom Inline Policy for MLflow and Enhanced Access

json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MLflowArtifactsAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
```

```
        "s3:ListBucket",
        "s3:GetBucketLocation",
        "s3:GetBucketVersioning"
      ],
      "Resource": [
        "arn:aws:s3:::leaddata23",
        "arn:aws:s3:::leaddata23/*",
        "arn:aws:s3:::mlflow-artifacts-*",
        "arn:aws:s3:::mlflow-artifacts-*/*"
      ]
    },
    {
      "Sid": "MLflowModelRegistryAccess",
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateModelPackage",
        "sagemaker:CreateModelPackageGroup",
        "sagemaker:DescribeModelPackage",
        "sagemaker:DescribeModelPackageGroup",
        "sagemaker:ListModelPackages",
        "sagemaker:UpdateModelPackage",
        "sagemaker:CreateModel",
        "sagemaker:CreateEndpointConfig",
        "sagemaker:CreateEndpoint"
      ],
      "Resource": "*"
    },
    {
```

```
      "Sid": "VPCAndNetworkAccess",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeRouteTables",
        "ec2:CreateVpcEndpoint",
        "ec2:DescribeVpcEndpoints"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EFSAccess",
      "Effect": "Allow",
      "Action": [
        "elasticfilesystem:CreateAccessPoint",
        "elasticfilesystem:DescribeAccessPoints",
        "elasticfilesystem:DescribeFileSystems",
        "elasticfilesystem:DescribeMountTargets"
      ],
      "Resource": "*"
    }
  ]
}
```

4.2 SageMaker Domain User Role

Step 1: Create Domain User Role

Role name: SageMakerDomainUserRole-MLPipeline

Managed Policies:

- AmazonSageMakerCanvasFullAccess
- AmazonSageMakerFeatureStoreAccess

Step 2: Custom Policy for Domain User

json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "StudioAppPermissions",
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateApp",
        "sagemaker:DeleteApp",
        "sagemaker:DescribeApp",
        "sagemaker:ListApps"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "sagemaker:OwnerUserProfileArn": "false"
        }
      }
    },
    {
```

```
        "Sid": "MLflowExperimentAccess",
        "Effect": "Allow",
        "Action": [
          "sagemaker:Search",
          "sagemaker:DescribeModelPackage",
          "sagemaker:ListModelPackages",
          "sagemaker:DescribeExperiment",
          "sagemaker:DescribeTrial",
          "sagemaker:DescribeTrialComponent"
        ],
        "Resource": "*"
      }
    ]
}
```

5. **Setup SageMaker Domain & User Profile**

Objective:

To launch Amazon SageMaker Studio for a specific user within your AWS environment and enable ML development in an isolated, scalable environment.

5.1 SageMaker Domain Setup

Step 1: Create SageMaker Domain

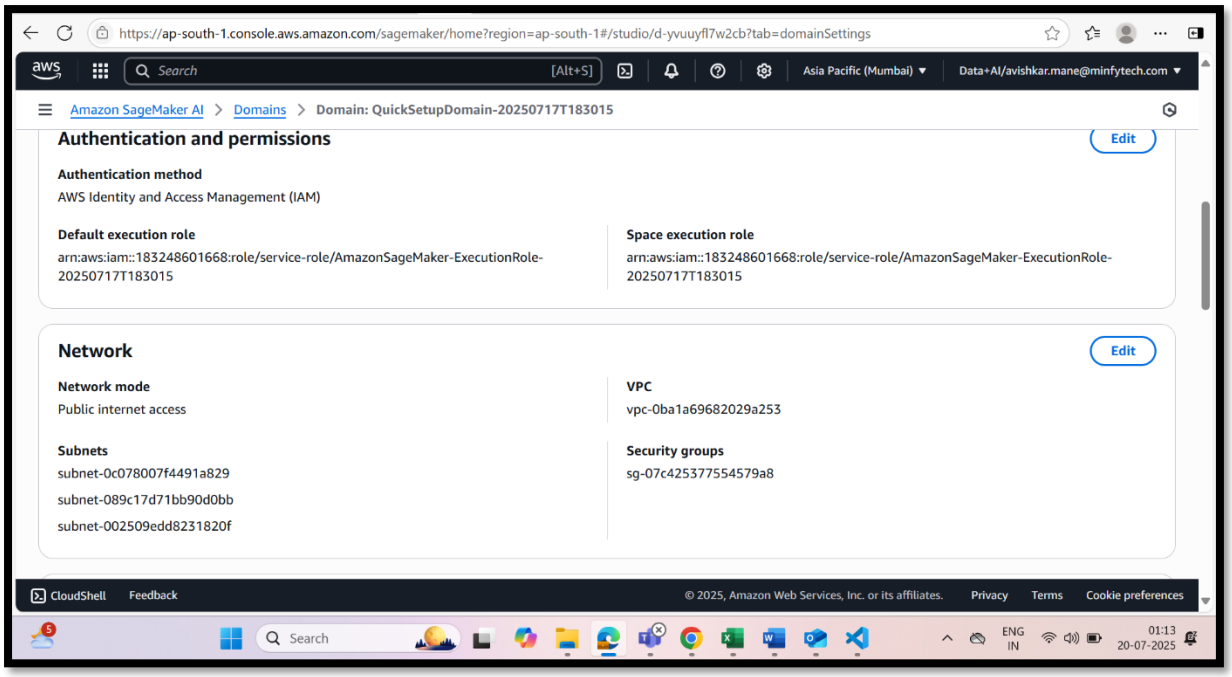
Via Console:

1. Navigate to **Amazon SageMaker Console** → **Studio** → **Domains**
2. Click **Create domain**
3. Choose **Quick setup** for simplified configuration

[Screenshot placeholder: SageMaker domain creation options]

Quick Setup Configuration:

4. **Domain name**: mlstudio-domain



5. **Default execution role**:
    - Select **Create a new role**
    - **S3 buckets**: Specific buckets → Add leaddata23 and mlflow-artifacts-leadscoring-[unique-id]
    - Or select **Use existing role** → SageMakerExecutionRole-MLPipeline

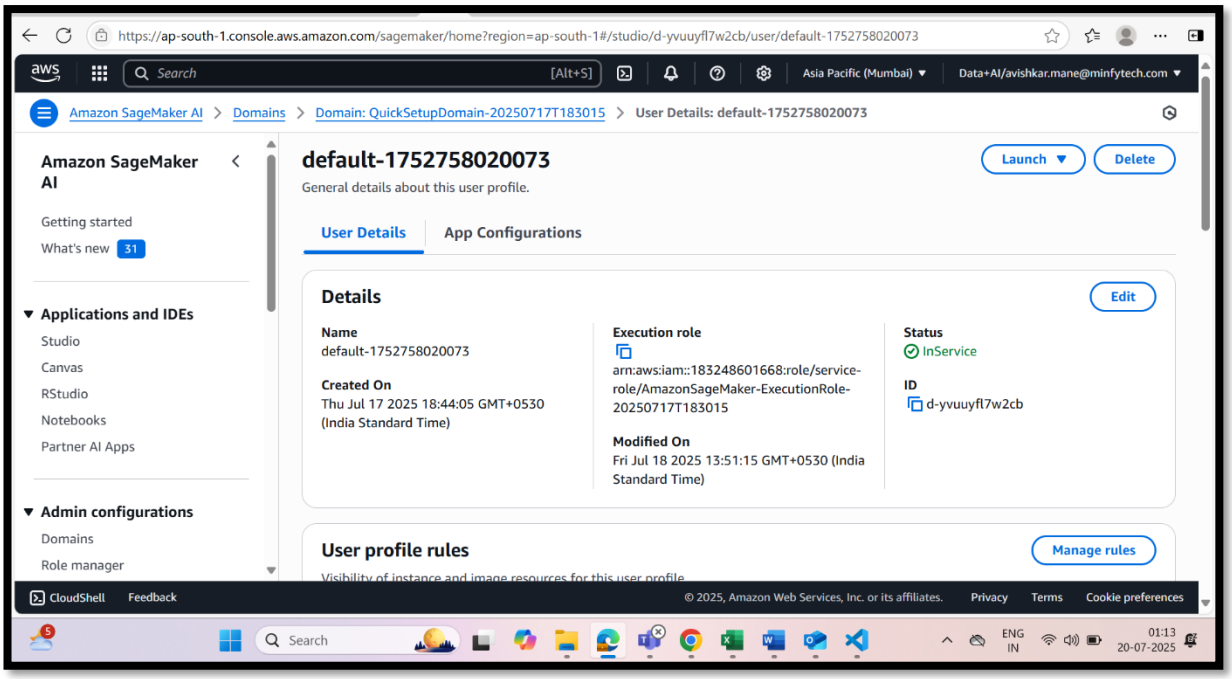
Additional Settings:

6. **Enable SageMaker Projects**: Yes
7. **Enable SageMaker JumpStart**: Yes
8. **Enable SageMaker Canvas**: Yes (for business users)

Step 2: Wait for Domain Creation

Status Monitoring:

- Domain creation takes 10-15 minutes
- Status should show **InService** when ready
- Monitor progress in the Domains dashboard

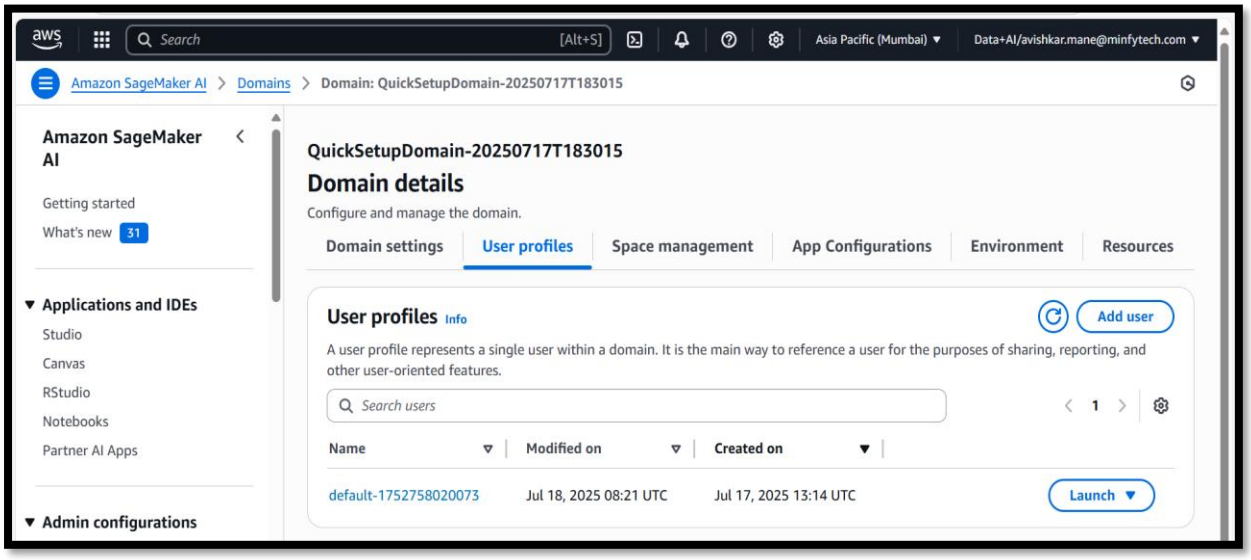


5.2 SageMaker Studio Configuration

Step 1: Add User Profile

Via Console:

1. Navigate to **SageMaker Console** → **Studio** → **mlstudio-domain**
2. Click **Add user** in the User profiles section
3. **User profile name**: mlflow-user
4. **Default execution role**: SageMakerExecutionRole-MLPipeline



IAM Role

- Create or choose an IAM role that has the following policies attached:
    - AmazonS3FullAccess
    - AmazonSageMakerFullAccess
    - AmazonEC2ContainerRegistryFullAccess
    - AWSGlueConsoleFullAccess
    - Custom policy (if using MLflow): access to mlflow-artifacts S3 bucket

5.3 Launch SageMaker Studio

Access Studio Environment:

1. Go to **SageMaker Console** → **Studio** → **mlstudio-domain**
2. Find user **mlflow-user**
3. Click **Open Studio**

[Screenshot placeholder: Studio launch interface]

First Launch Configuration:

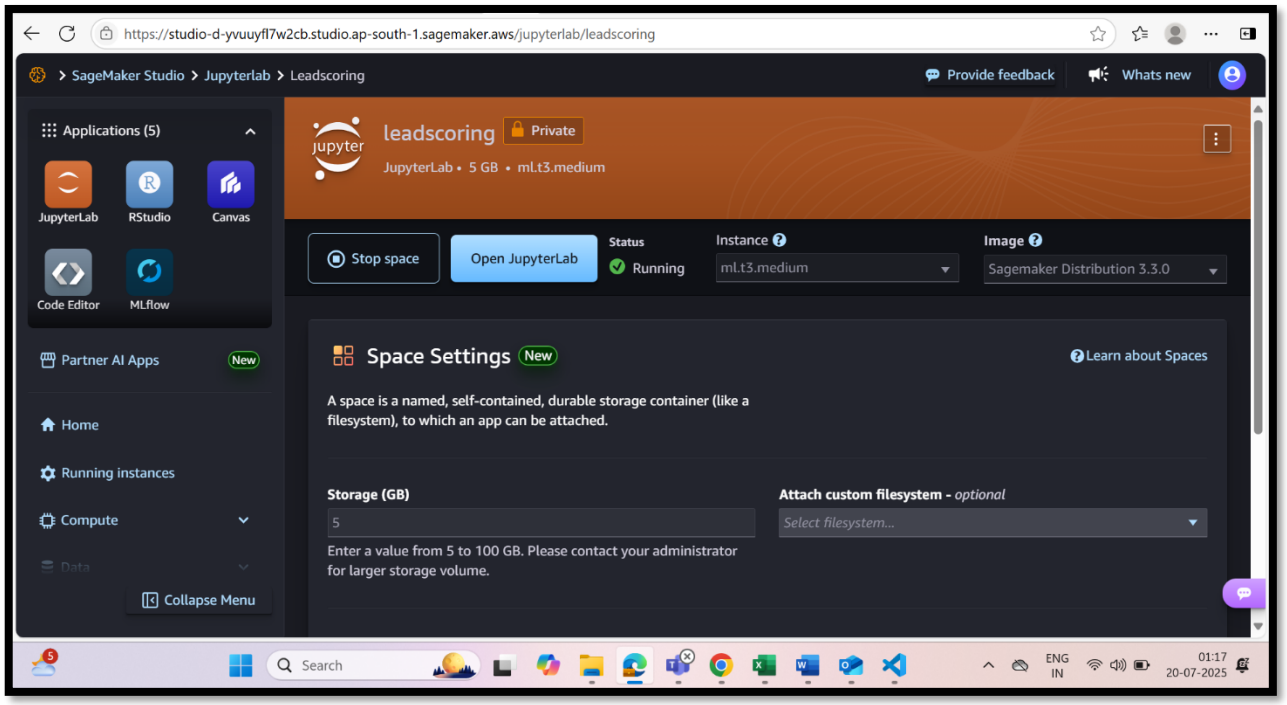
4. **Select image**: Data Science 3.0 (includes ML libraries)
5. **Select instance type**: ml.t3.medium
6. **Launch JupyterServer**

[Screenshot placeholder: Studio environment launch]

Step 3: Environment Verification

Create Test Notebook:

1. In Studio, click **File** → **New** → **Notebook**
2. **Kernel**: Python 3 (Data Science 3.0)
3. **Instance type**: ml.t3.medium

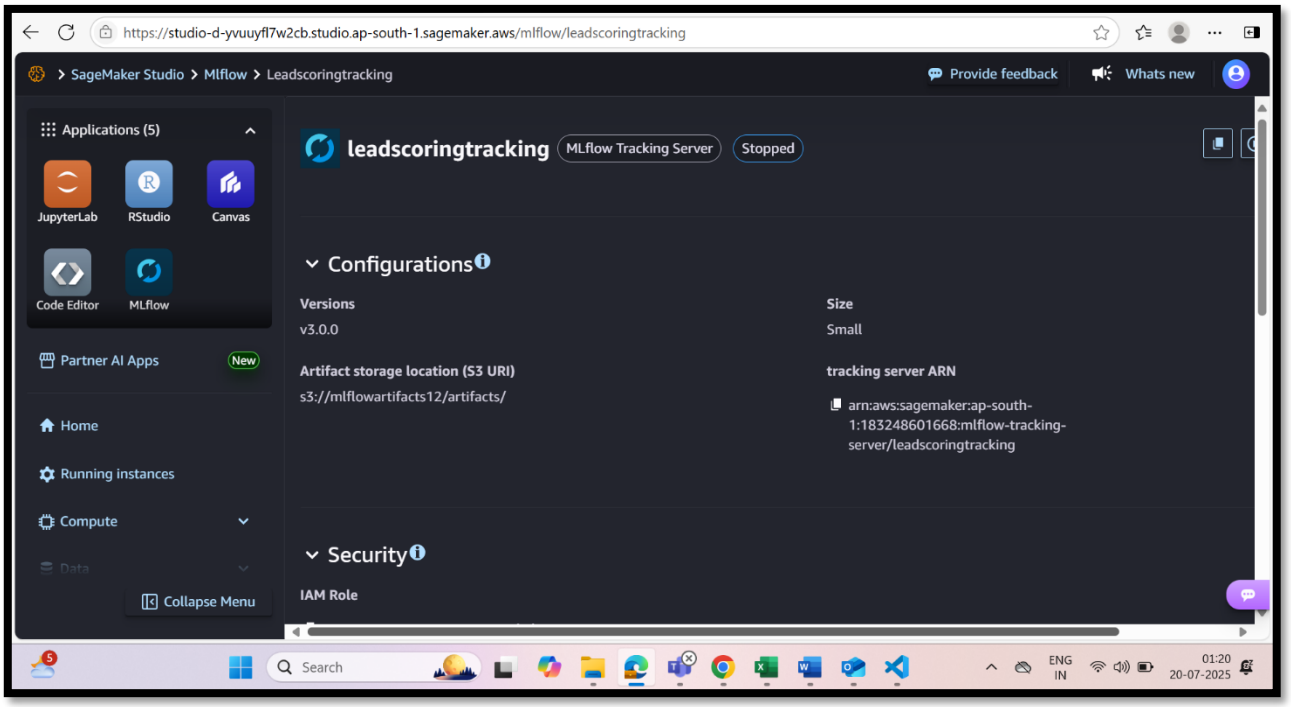


6. MLflow Tracking Server Setup

Step 1: Create MLflow Tracking Server

Via SageMaker Console:

1. Navigate to **SageMaker Console** → **MLflow** → **Tracking servers**
2. Click **Create tracking server**



3. **Tracking server name**: leadscoring-mlflow-server

6.1 Configuration Details

4. **Artifact store URI**: s3://mlflow-artifacts-leadscoring-[unique-id]/artifacts
5. **Automatic creation of default bucket**: Disable (using existing bucket)
6. **Database**: Automatic (managed by SageMaker)

Advanced Configuration:

7. **Weekly maintenance window**: Sunday 03:00-04:00 UTC
8. **Automatic minor version upgrades**: Enable

Step 2: Configure Network Settings

VPC Configuration:

1. **VPC**: Select same VPC as SageMaker domain
2. **Subnets**: Select private subnets with internet access
3. **Security groups**: Create new security group

Security Group Rules:

- **Type**: HTTPS
- **Port**: 443
- **Source**: SageMaker domain security group

Step 3: Access MLflow UI

Get Tracking Server URL:

1. Go to **SageMaker Console** → **MLflow** → **Tracking servers**
2. Click on **leadscoring-mlflow-server**
3. Copy **Tracking server URL**

Create a new Jupyter Notebook or a Python file and add:

python

```
import os


# Set MLflow Tracking URI
```

```
os.environ['MLFLOW_TRACKING_URI'] = "https://<your-mlflow-tracking-server-endpoint>"

# If authentication is required
os.environ['MLFLOW_SAGEMAKER_USER_ARN'] = "<your-user-arn>"

# (Optional) Disable SSL cert warnings if needed
os.environ['MLFLOW_TRACKING_INSECURE_TLS'] = "true"
```

6.2 IAM Policies to Attach

Attach the following to the **SageMaker execution role**:

- AmazonS3FullAccess *(for data reading/writing to S3)*
- AmazonRedshiftFullAccess
- AmazonSageMakerFullAccess
- Custom Policy for MLflow if needed (if not working via console)

6.3 Troubleshooting

Issue	Solution
MLflow logs not visible	Check tracking URI and experiment name
Redshift connection error	Ensure VPC/Subnet access + correct credentials
xgboost import error	Use !pip install xgboost
Data shape mismatch	Ensure consistent feature engineering during prediction

7. Model Training in Amazon SageMaker

This section outlines how the **Lead Scoring model is trained using a Scikit-learn pipeline in SageMaker Studio**, with experiment tracking through **MLflow**, and saving the best model to Amazon S3 for downstream usage (Flask API or batch prediction).

Step-by-Step Process

Step 1: Launch SageMaker Studio

1. Go to AWS Console → SageMaker → **Studio**.
2. Choose the domain you created and click **"Open Studio"**.
3. Launch a **System Image with sklearn and pandas** pre-installed (or use a custom image).

Step 2: Upload Your Code to SageMaker Studio

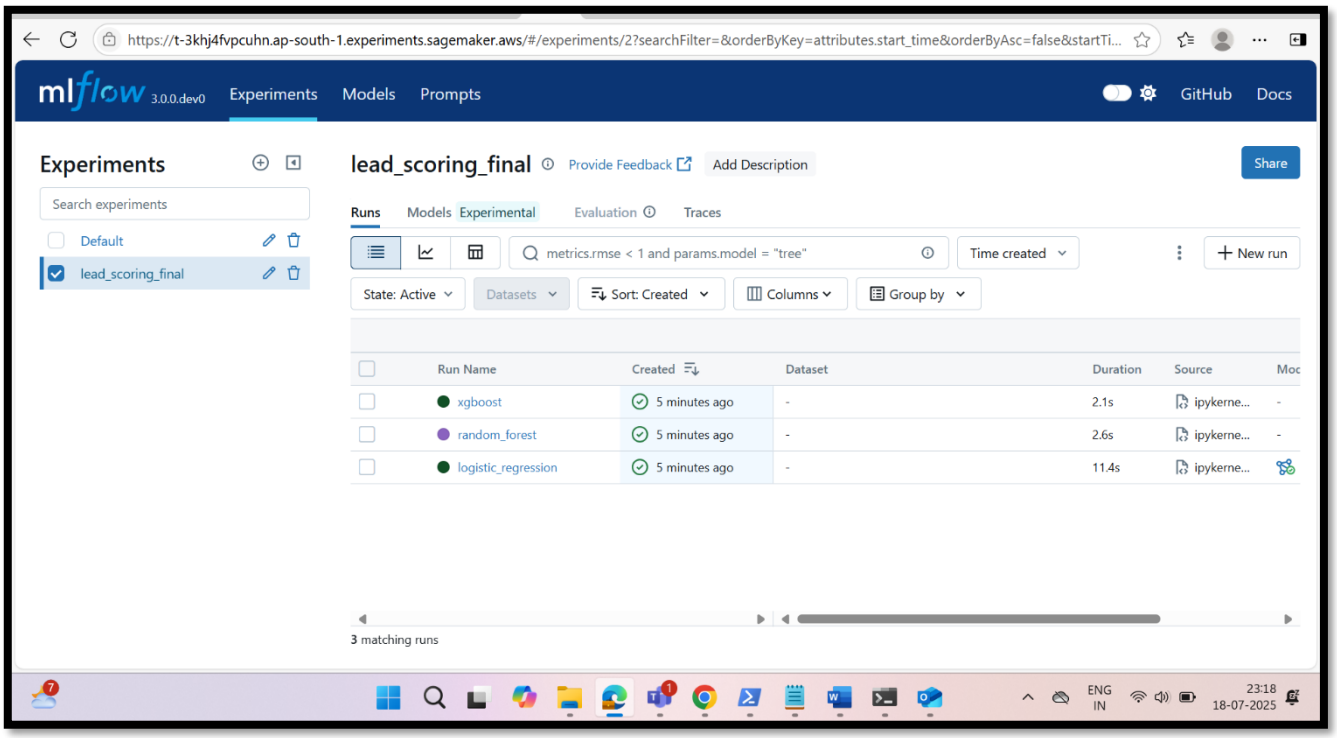
Upload your modular project directory to Studio. The main files include:

File	Purpose
train_model.py	Main script for model training
data_preprocessing.py	Preprocessing logic
feature_engineering.py	Feature generation
model_utils.py	Utility functions to compare and log models
config.yaml	S3 paths and model parameters

- **Step 4: : Monitor Training Progress**

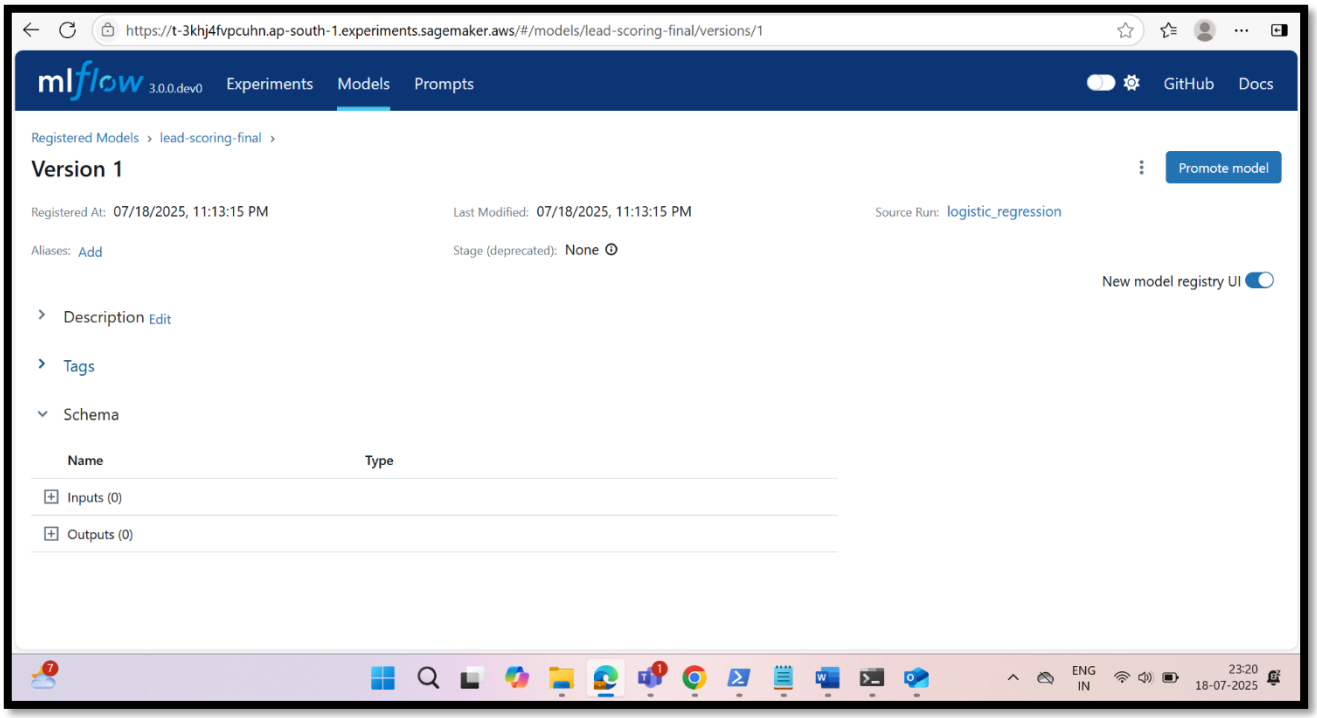
7.1 MLflow UI

1. Access MLflow tracking server URL
2. Navigate to **lead-scoring-experiments**
3. Monitor experiment runs and metrics



View Model Registry:

1. Click **Models** in MLflow UI
2. Check **LeadScoringModel** registry
3. Review model versions and performance



Step 5: Run the Training Script

1. Open train_model.py inside SageMaker Studio.
2. Click **"Run All"** or run cells line by line.
3. Ensure you see:
    - Model training logs
    - Upload to S3 confirmation
    - Metrics logged to MLflow

Notes:

- This training is manually triggered via SageMaker Studio.
- For automation, this can be triggered via an **Airflow DAG** in your MWAA environment.
- The model saved here is used in:
    - Flask API (/predict)
    - Batch inference scripts
    - Drift detection comparison
    -

8. Data Drift Detection with Evidently AI

This part describes how we detect **data drift** between:

- The original **training data** vs **new incoming data** (typically last week's data),
- Or **test data vs historical data**.

This is important to ensure the model continues to perform well in production.

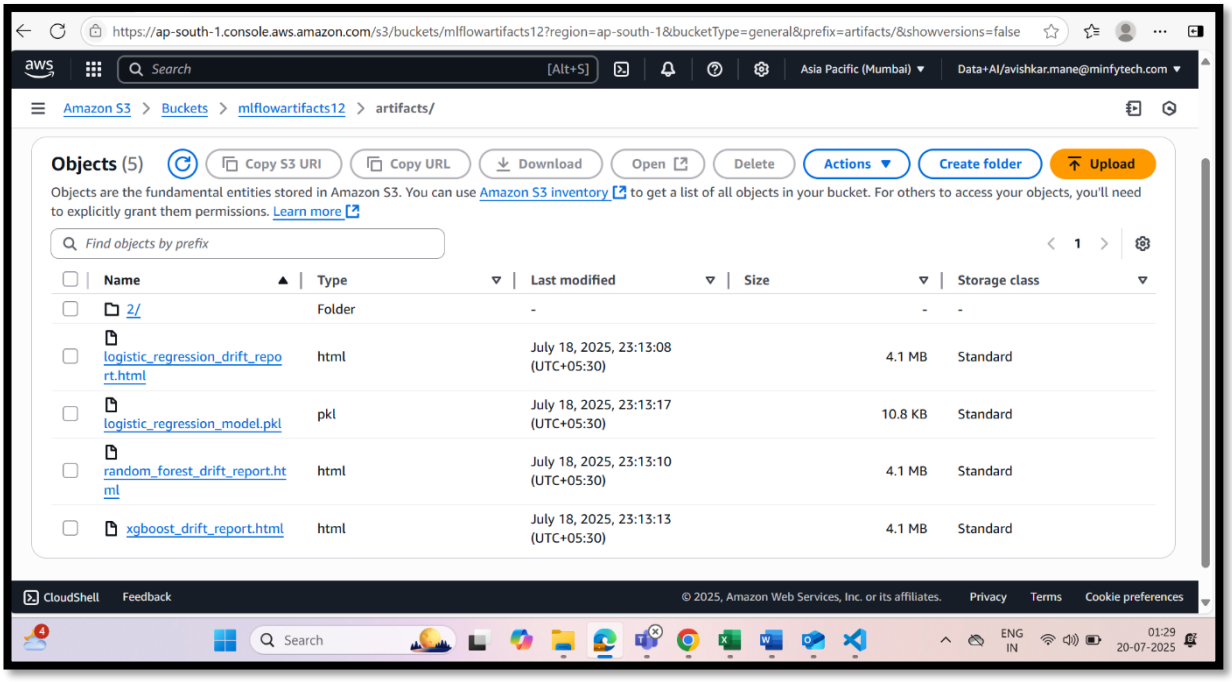
8.1 Drift Detection Flow

Step 1: Setup – Save Processed Train Data to S3

Before we can compare old vs new data, we need the original **preprocessed training data** saved in S3:

```
# Inside train_model.py or data_preprocessing.py

df_processed.to_csv("final_train.csv", index=False)
```

```
# Upload to S3 (done in SageMaker)

import boto3

s3 = boto3.client("s3")

s3.upload_file("final_train.csv", "leaddata23", "drift/train_data.csv")
```



Saved to: s3://leaddata23/drift/train_data.csv

🔹 **Step 2: New Data Upload (Example: Weekly)**

Your new lead data is expected to land in:

s3://leaddata23/raw/new_leads_<date>.csv

For this example, let's assume Airflow already ingested this file and saved a processed version to:

s3://leaddata23/processed/new_data.csv

9. Flask UI with Dynamic Model Loading from MLflow

Goal

Create an HTML-based Flask app that:

- Loads the best ML model from MLflow Model Registry (hosted in SageMaker)
- Accepts raw lead input via a form or CSV upload
- Preprocesses input using the same pipeline
- Returns predictions: **0 and 1** conversion potential

9.1 **Folder Structure**

lead_scoring_flask_app/

```
│
├── app.py
├── mlflow_model_loader.py
├── templates/
│   ├── index.html
│   └── result.html
├── static/
│   └── style.css
├── requirements.txt
```

9.2 Commands to Run the App

1. Install dependencies:

pip install flask pandas mlflow scikit-learn

2. Run:

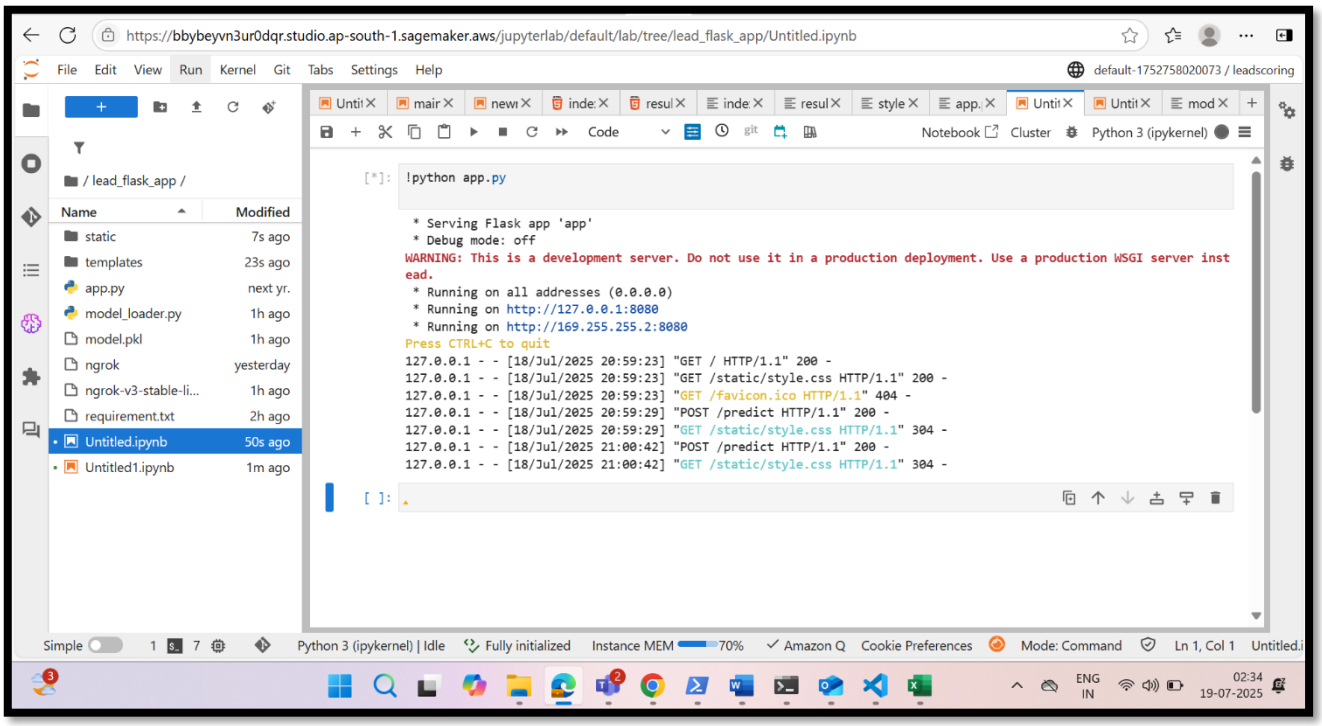
python app.py

3. Open in browser:

http://localhost:5000/

Notes

- Only selected, required features are shown in the UI (not all 100+).
- Model includes preprocessing pipeline, so raw input is accepted.
- mlflow_model_loader.py dynamically loads from MLflow Registry.
- SHAP/Explainability or Probabilities can be added to result table.
- Works for both single entry and batch CSV uploads.



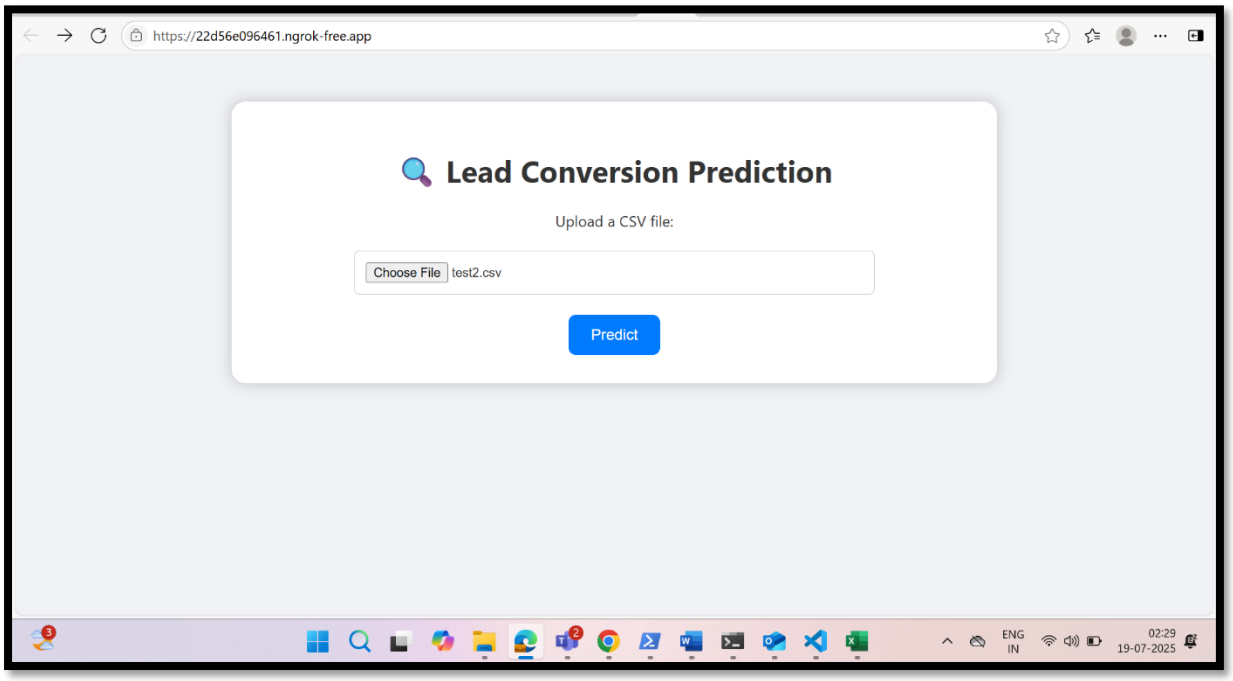
9.3 Web Interface Configuration

Features:

- **Single Prediction**: Form input for individual leads
- **Batch Prediction**: CSV file upload
- **Model Information**: Display current model version
- **Prediction Confidence**: Show probability scores

UI Components:

- **Input Form**: Lead characteristics (source, score, etc.)
- **Results Table**: Predictions with confidence scores
- **Model Metrics**: Current model performance
- **Upload Interface**: Batch prediction via CSV



10. MWAA Environment Setup and DAG Orchestration Guide

10.1 Objective

To configure and deploy Apache Airflow on Amazon Managed Workflows for Apache Airflow (MWAA) to automate the orchestration of a complete MLOps pipeline for lead scoring, including:

- Automated ingestion of new data from PostgreSQL/S3
- Data drift detection and alerting
- Conditional model retraining
- End-to-end orchestration via DAGs with dependencies

10.2 MWAA Environment Setup

Step 1: Create an S3 Bucket for MWAA Assets

Command:

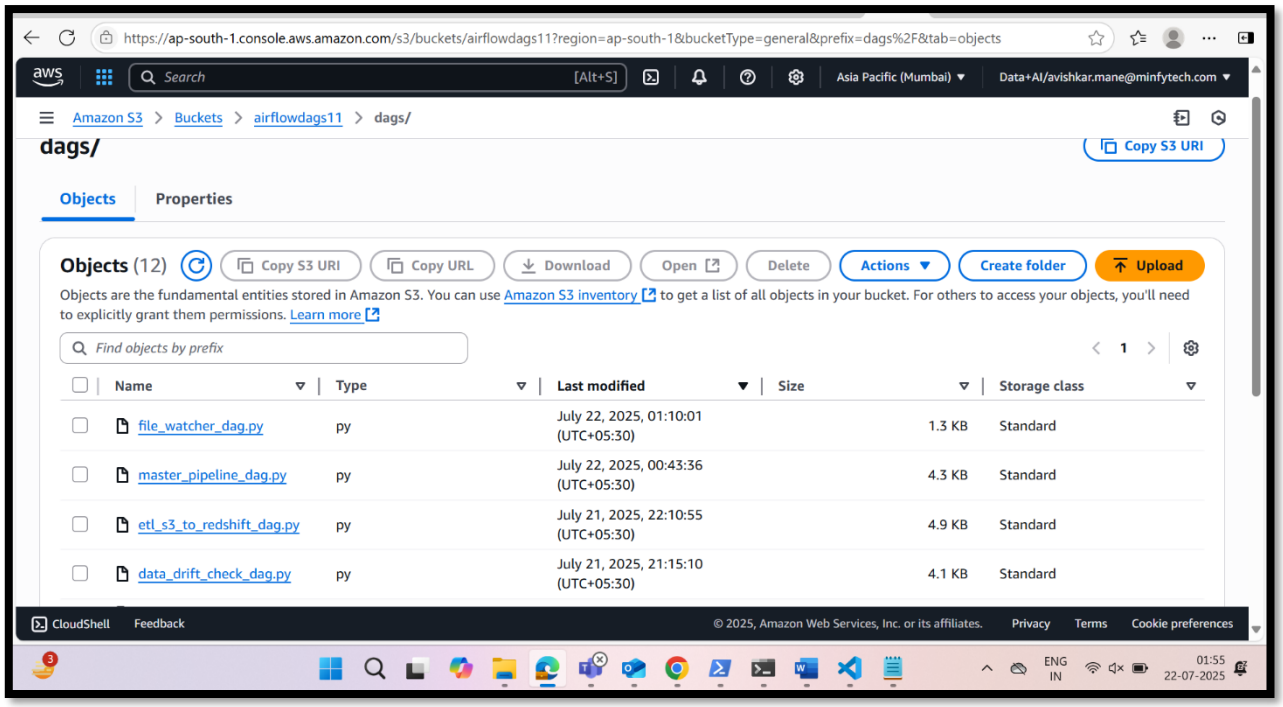
aws s3api create-bucket --bucket airflow-pipeline-leads --region <your-region>

Folder Structure:

s3://airflow-pipeline-leads/

```
├── dags/
│   ├── watcher_dag.py
│   ├── master_dag.py
│   ├── etl_job_dag.py
│   ├── data_drift_dag.py
│   └── model_retraining_dag.py
├── plugins/
├── requirements.txt
```

Note: Ensure this bucket is created in the same AWS region as the MWAA environment.



Step 2: Prepare requirements.txt

Content:

boto3

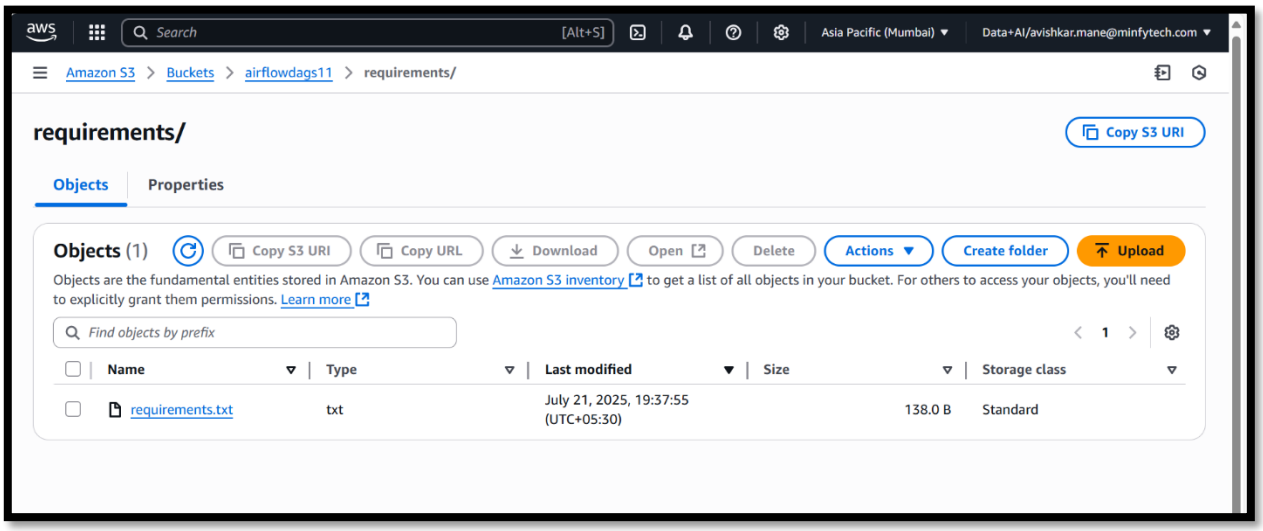
pandas

sagemaker

evidently

Upload to S3:

aws s3 cp requirements.txt s3://airflow-pipeline-leads/requirements.txt

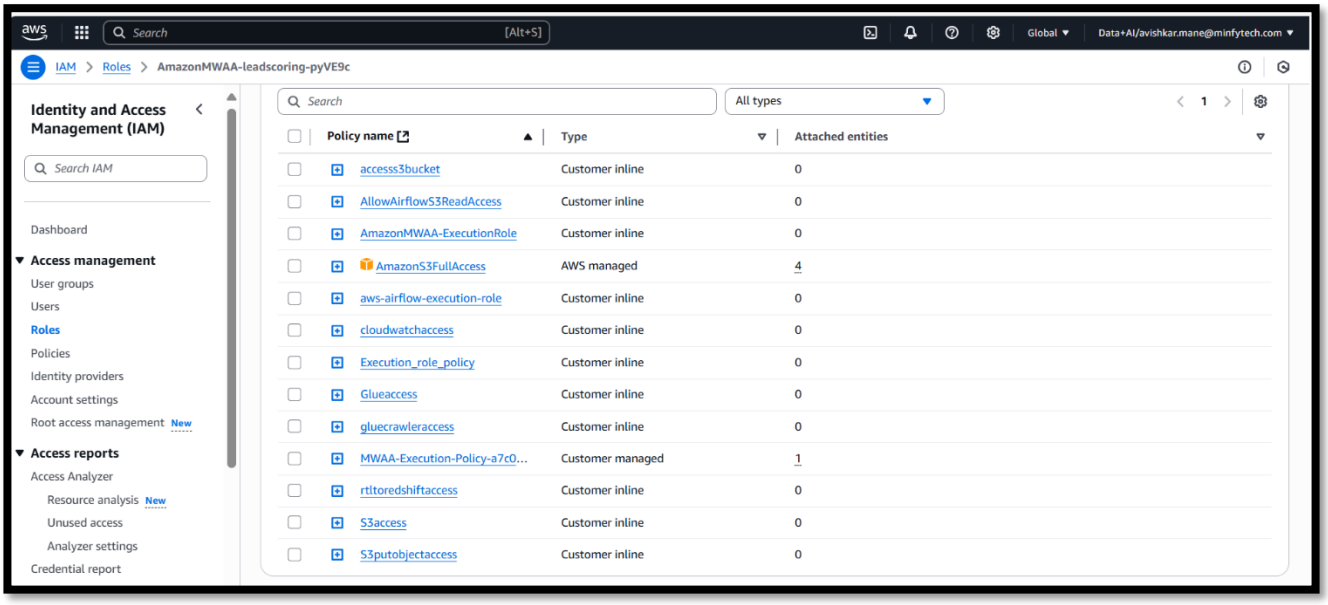


Step 3: Create IAM Execution Role for MWAA

Create an IAM role with the following policies:

- AmazonMWAAWebServerAccess
- AmazonS3FullAccess (or restricted to the DAG bucket)
- AmazonSageMakerFullAccess
- AWSGlueConsoleFullAccess
- AmazonRedshiftFullAccess
- AmazonSNSFullAccess
- CloudWatchLogsFullAccess

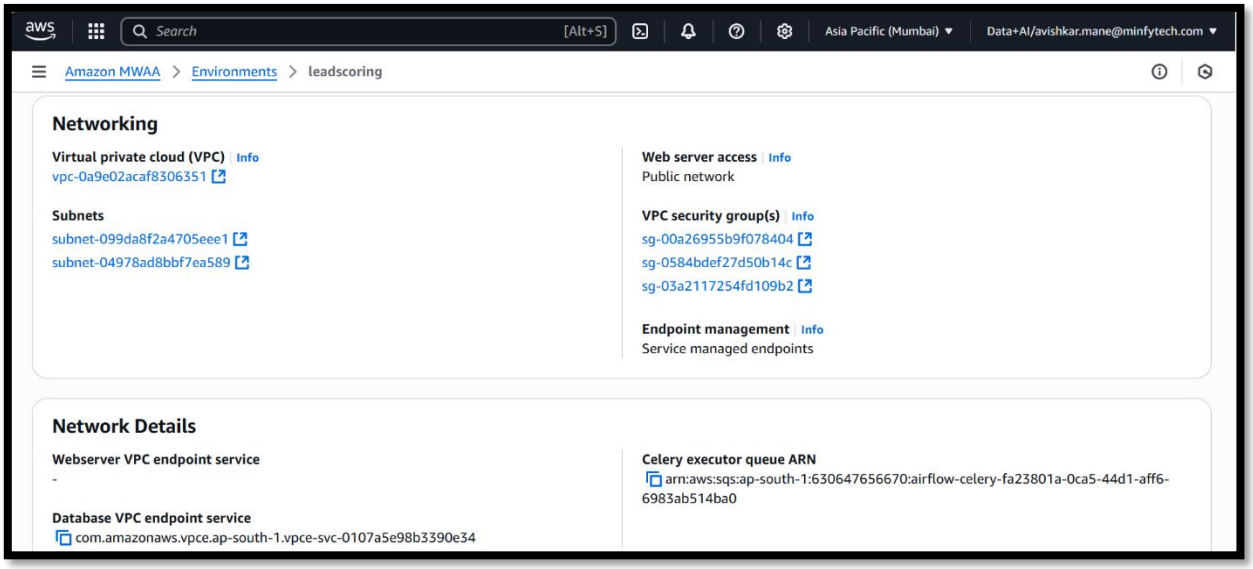
Command: Use AWS IAM console or CLI to attach these policies to the new role (e.g., MWAAExecutionRole).

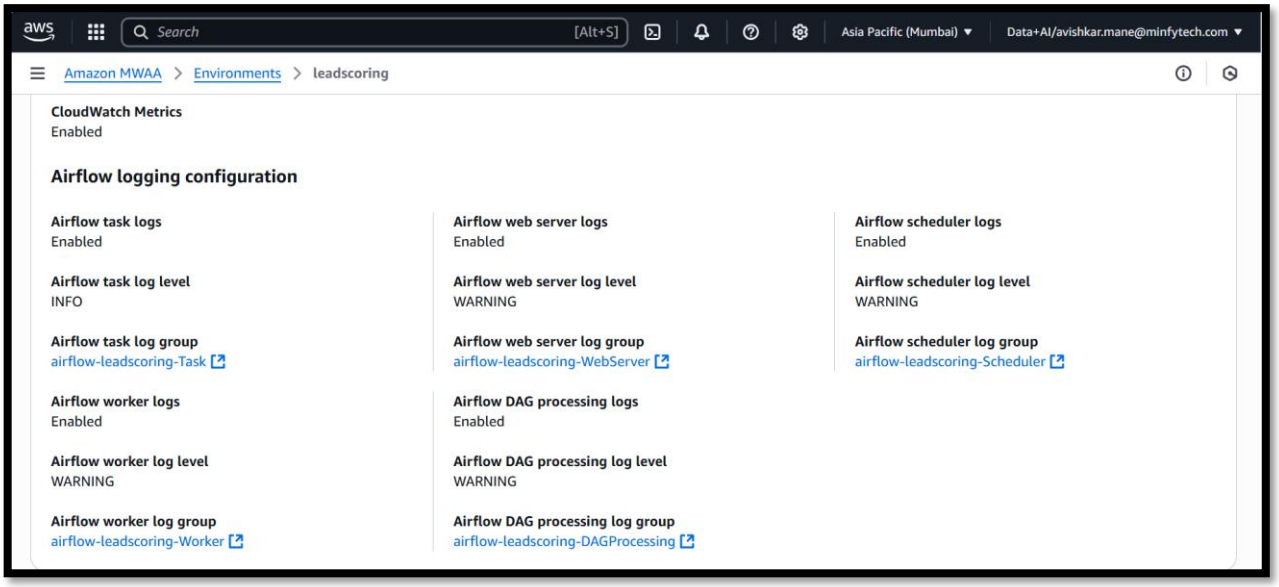


Step 4: Launch the MWAA Environment

1. Go to the Amazon MWAA Console → Create Environment
2. Fill in the following:
    - Environment name: leadscore-mwaa
    - Execution Role: Select MWAAExecutionRole
    - DAGs S3 path: s3://airflow-pipeline-leads/dags
    - Plugins path: leave empty (unless needed)
    - Requirements.txt path: s3://airflow-pipeline-leads/requirements.txt

3. Enable CloudWatch logging as required
4. Click **Create Environment** (Provisioning takes 20–25 minutes)

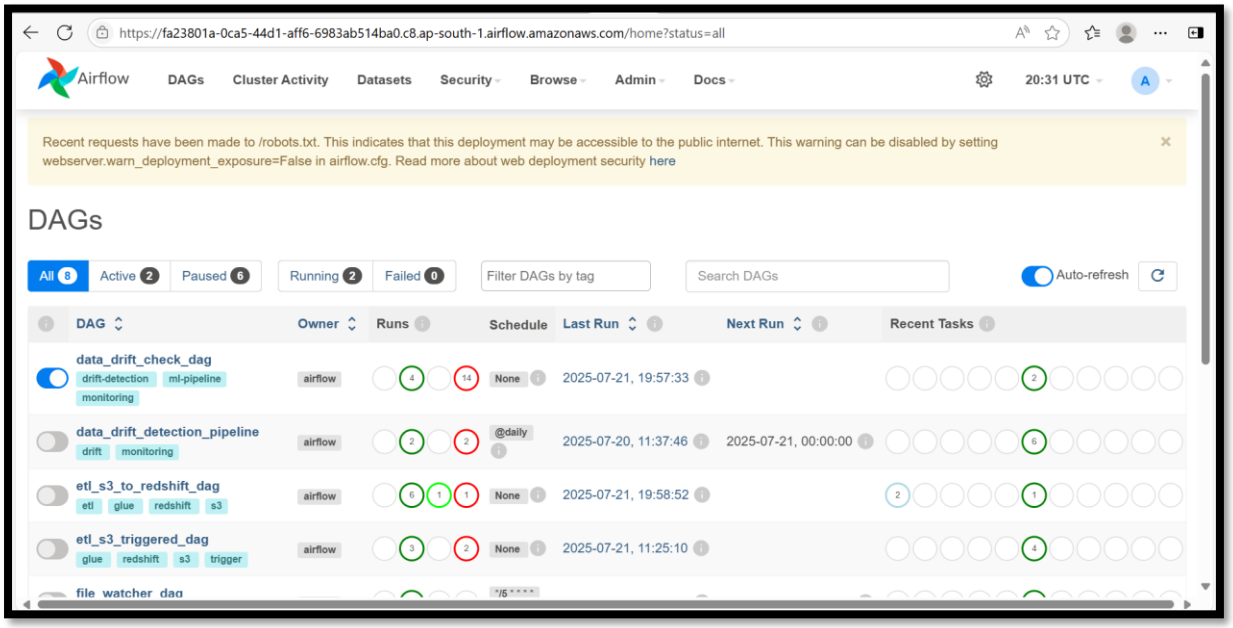




Step 5: Access the Airflow UI

Once the environment is in Available state:

- Click on the Airflow UI link
- You should see your DAGs once Airflow syncs with the S3 path



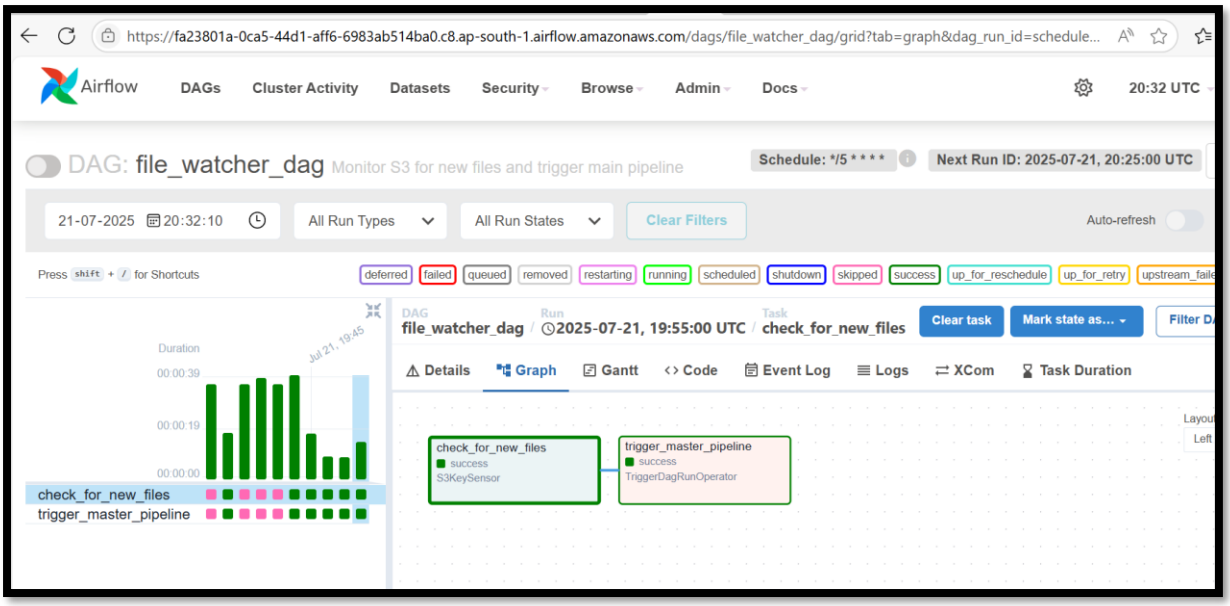
11. DAG Orchestration Overview

1. watcher_dag.py

Purpose: Continuously monitors a source S3 folder. If a new file is detected, it moves it to a processing folder and triggers master_dag.py.

Tasks:

- S3KeySensor to detect incoming files
- PythonOperator to move the file
- TriggerDagRunOperator to trigger master_dag.py

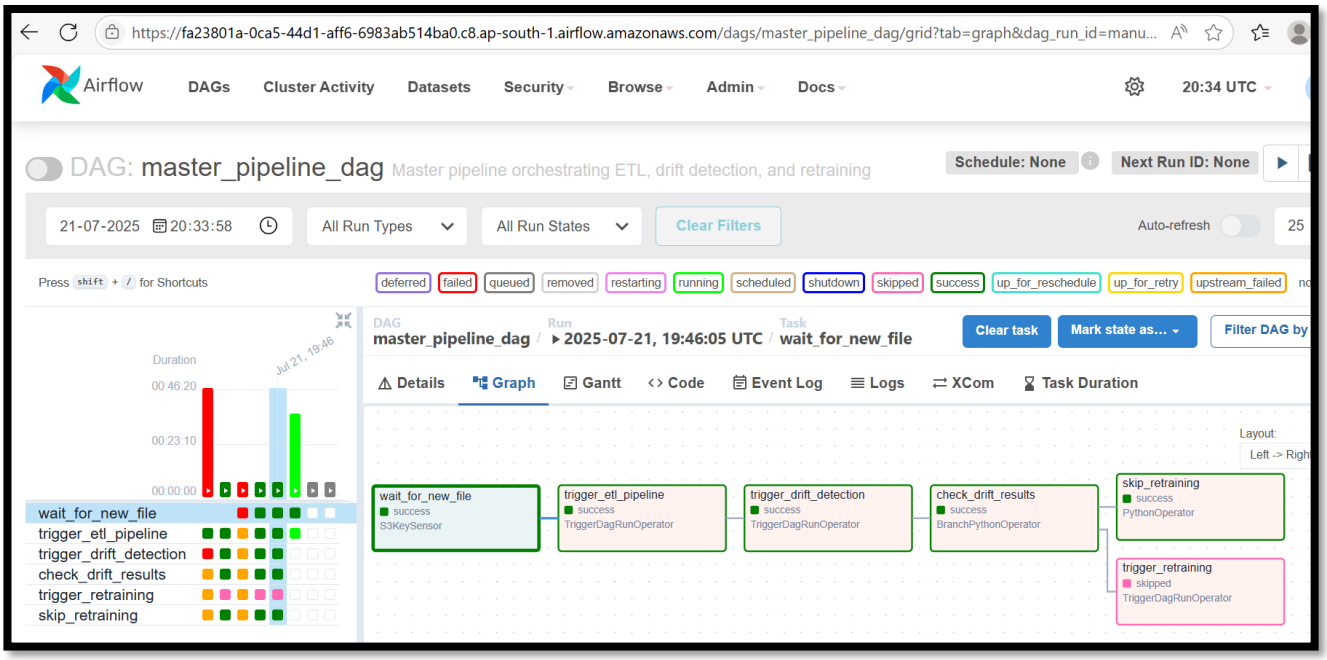


2. master_dag.py

Purpose: Acts as the central controller DAG that sequences the full pipeline.

Tasks:

- Trigger etl_job_dag.py
- Wait for ETL completion
- Trigger data_drift_dag.py
- Based on drift results, trigger model_retraining_dag.py or skip it

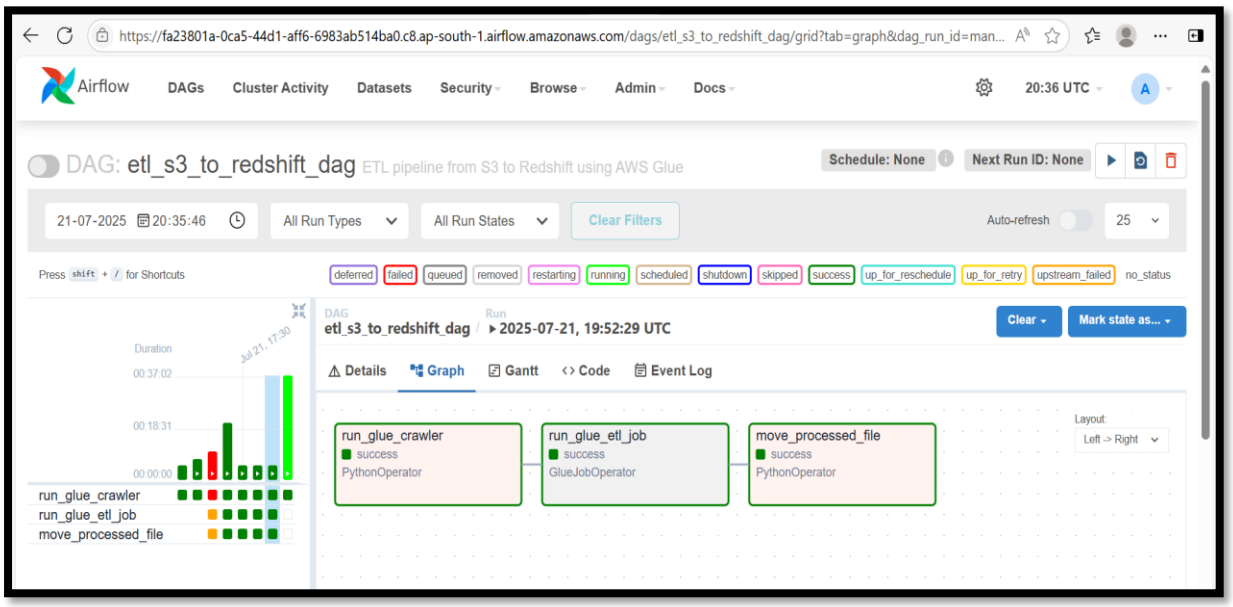


3. etl_job_dag.py

Purpose: Runs an AWS Glue job to clean and transform raw lead data and load it into Amazon Redshift.

Tasks:

- GlueJobOperator to execute the ETL job



4. data_drift_dag.py

Purpose: Compares recent data with training data using Evidently AI. Uploads an HTML report and sends an alert if drift is detected.

Tasks:

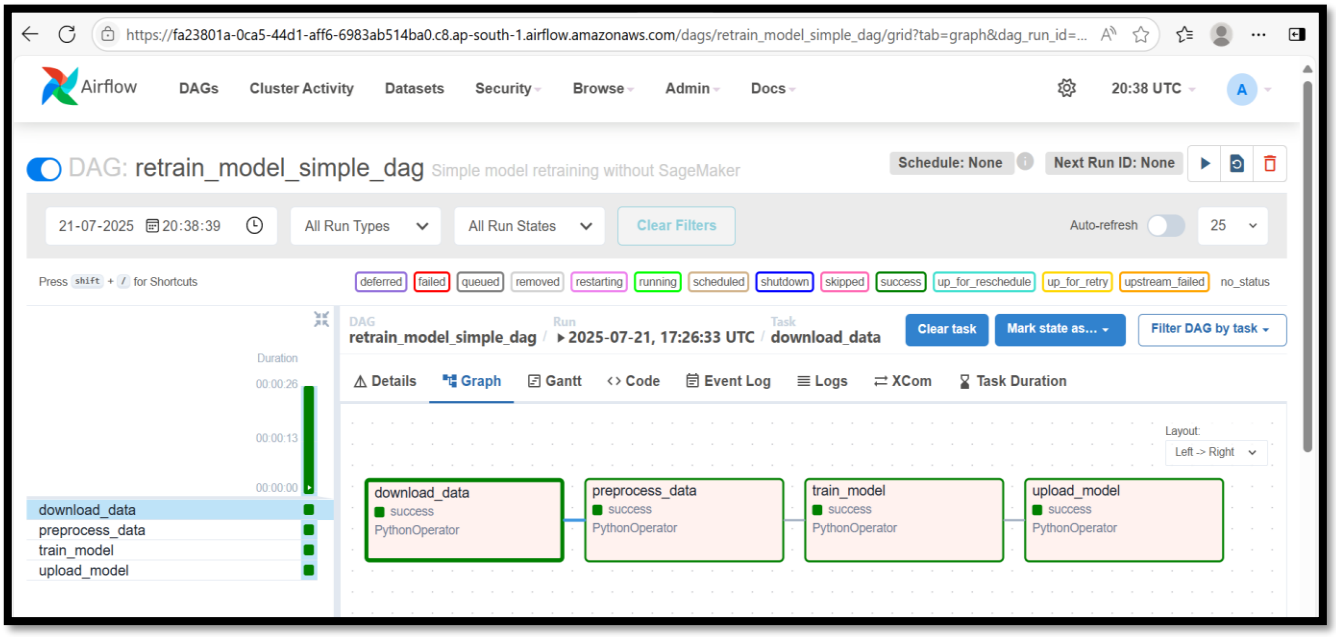
- PythonOperator to fetch datasets from S3
- Generate drift report with Evidently
- Upload report to S3
- Publish alert via SNS if drift score crosses threshold

5. model_retraining_dag.py

Purpose: Retrains the model only if data drift is detected.

Tasks:

- Load updated training data
- Launch SageMaker training job
- Evaluate model AUC or performance
- Save model to S3 or MLflow registry if better



11.1 DAG Scheduling

DAG Name	Schedule	Trigger Type
watcher_dag	Continuous	S3 Sensor Trigger
master_dag	Triggered	watcher_dag
etl_job_dag	Daily @ 1:00 AM	master_dag
data_drift_dag	Daily @ 3:00 AM	master_dag
model_retraining_dag	Monthly / Drift	Conditional (drift)

11.2 Security Best Practices

- Limit IAM role to least privileges (avoid full access unless needed)
- Use S3 bucket policies to restrict external write access
- Store credentials (e.g., database passwords) in AWS Secrets Manager
- Enable encryption at rest for S3, Redshift, and SageMaker outputs
- Monitor CloudWatch logs for failed DAG runs