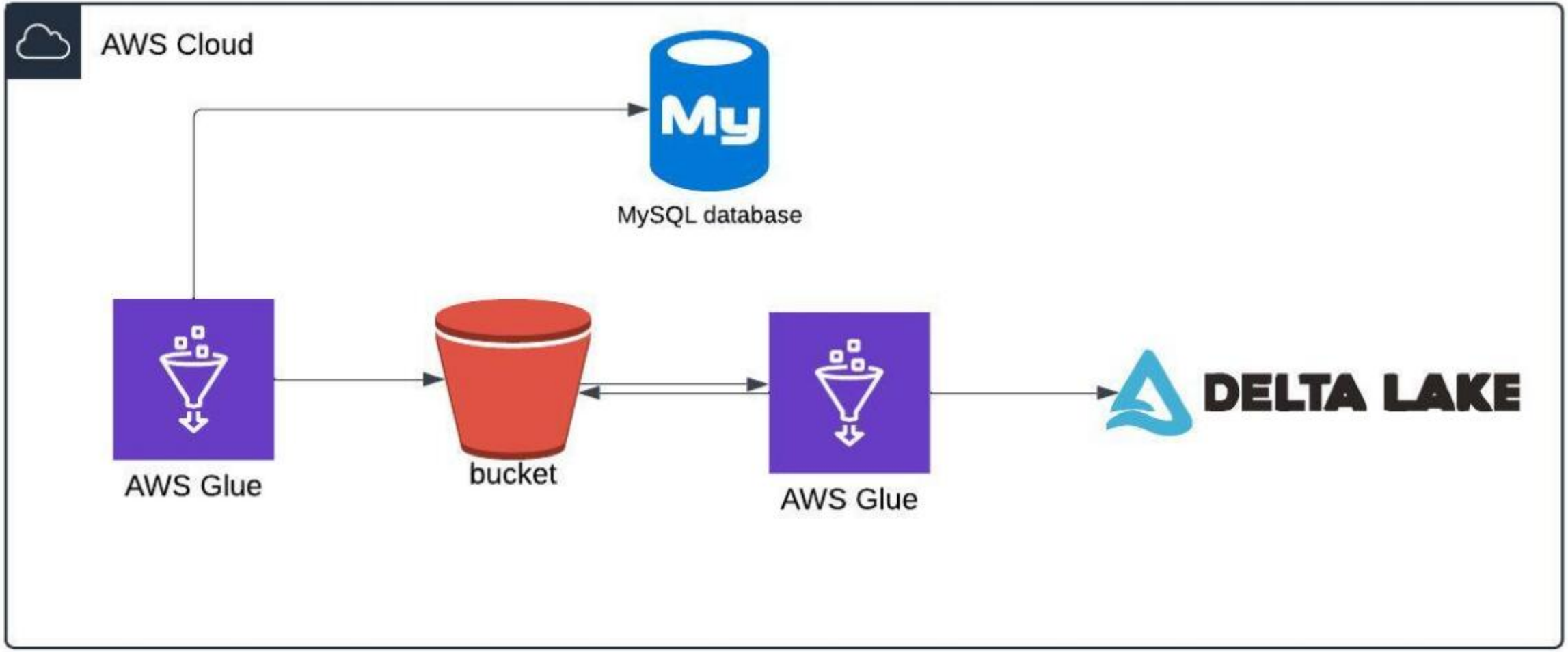
MySQL to Delta Lake Pipeline Using AWS Glue and PySpark

Objective

- Extract: Use an AWS Glue job with runtime arguments to dynamically extract data from a MySQL database and store it in Amazon S3.
- Transform: Leverage a PySpark-based AWS Glue job to process the extracted data and save it into a Delta Lake format in S3 for optimized storage and querying.

Tools Used

- MySQL: As the source database for extracting raw data.
- AWS Glue: For performing ETL operations, including data extraction and transformation.
- PySpark: For writing custom transformation logic in the Glue job to process data.
- Amazon S3: As the storage layer for both extracted data and Delta Lake files.
- Delta Lake: For storing processed data in a format optimized for data lakes and analytics.
- AWS Glue Runtime Arguments: For passing dynamic parameters like table names, partitions, or filters during the Glue job execution.
- Python: For scripting transformation logic in PySpark and configuring Glue jobs.
- CloudWatch: For monitoring Glue job execution and troubleshooting issues.


AWS Cloud
MySQL database
AWS Glue
bucket
AWS Glue
DELTA LAKE

```
    ->     departure_airport VARCHAR(10),
    ->     arrival_airport VARCHAR(10),
    ->     scheduled_departure_time DATETIME,
    ->     actual_departure_time DATETIME,
    ->     scheduled_arrival_time DATETIME,
    ->     actual_arrival_time DATETIME,
    ->     gate VARCHAR(10),
    ->     terminal VARCHAR(5),
    ->     created_at DATETIME
    -> );
Query OK, 0 rows affected (0.26 sec)

mysql> LOAD DATA LOCAL INFILE "flights-data/flight_logistics.csv" INTO TABLE flights_db.flight_logistics FIELDS TERMINATED BY ',' ENCLOSED BY
'"' LINES TERMINATED BY '\n' IGNORE 1 ROWS;
Query OK, 24966 rows affected, 25029 warnings (4.57 sec)
Records: 25000  Deleted: 0  Skipped: 34  Warnings: 25029

mysql> LOAD DATA LOCAL INFILE "flights-data/passenger_experience.csv" INTO TABLE flights_db.passenger_experience FIELDS TERMINATED BY ',' ENCL
OSED BY '"' LINES TERMINATED BY '\n' IGNORE 1 ROWS;
Query OK, 24966 rows affected, 34 warnings (2.87 sec)
Records: 25000  Deleted: 0  Skipped: 34  Warnings: 34

mysql> LOAD DATA LOCAL INFILE "flights-data/flights.csv" INTO TABLE flights_db.flights FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINAT
ED BY '\n' IGNORE 1 ROWS;
Query OK, 24966 rows affected, 34 warnings (4.60 sec)
Records: 25000  Deleted: 0  Skipped: 34  Warnings: 34

mysql> SHOW tables;
+----------------------+
| Tables_in_flights_db |
+----------------------+
| flight_logistics     |
| flights              |
| passenger_experience |
+----------------------+
3 rows in set (6.91 sec)

mysql>
```

```python
import boto3, sys, csv, pymysql, io, json, logging
from datetime import date
from botocore.exceptions import ClientError
from awsglue.utils import getResolvedOptions

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger()

args = getResolvedOptions(sys.argv, ["table_name","delta_value"])
table_name = args["table_name"]
delta_value = args["delta_value"]

def get_rds_credentials(secret_name, region_name):
    session = boto3.session.Session()
    client = session.client(service_name='secretsmanager', region_name=region_name)
    try:
        get_secret_value_response = client.get_secret_value(SecretId=secret_name)
    except ClientError as e:
        logger.error(f"Unable to retrieve secret: {e}")
        return None

    if 'SecretString' not in get_secret_value_response:
        logger.error("Secret does not contain a string.")
        return None

    secret = get_secret_value_response['SecretString']
    credentials = json.loads(secret)
    return credentials

credentials = get_rds_credentials("flights_db", "us-east-1")

user_name = credentials['username']
```

```python
credentials = get_rds_credentials("flights_db", "us-east-1")

user_name = credentials['username']
host = credentials['host']
password = credentials['password']
db_name = "flights_db"

s3_bucket = "gd-aws-de-labs"
s3_key = f'raw_landing_zone/{db_name}/{table_name}/data.csv'

def main():
    try:
        connection = pymysql.connect(
            host=host,
            user=user_name,
            password=password,
            database=db_name,
            cursorclass=pymysql.cursors.DictCursor
        )

        if table_name=='flights':
            sql = f"SELECT * FROM {table_name} where date(created_at)='{delta_value}'"

        else:
            sql = f"SELECT * FROM {table_name}"

        with connection.cursor() as cursor:

            cursor.execute(sql)
            result = cursor.fetchall()

        csv_data = convert_to_csv(result)
```

```python
def main():

        with connection.cursor() as cursor:

            cursor.execute(sql)
            result = cursor.fetchall()

        csv_data = convert_to_csv(result)

        s3 = boto3.client('s3')
        s3.put_object(Body=csv_data, Bucket=s3_bucket, Key=s3_key)
        logger.info('Data extracted and written to S3 successfully')

    except Exception as e:
        logger.error(f'Error: {str(e)}')
        sys.exit(1)

    finally:
        connection.close()

def convert_to_csv(data):
    if not data:
        return ""
    csv_file = io.StringIO()
    fieldnames = data[0].keys()
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
    writer.writeheader()
    for row in data:
        writer.writerow(row)
    return csv_file.getvalue()
```

```python
def main():
        logger.error(f'Error: {str(e)}')
        sys.exit(1)

    finally:
        connection.close()

def convert_to_csv(data):
    if not data:
        return ""
    csv_file = io.StringIO()
    fieldnames = data[0].keys()
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
    writer.writeheader()
    for row in data:
        writer.writerow(row)
    return csv_file.getvalue()

if __name__ == '__main__':
    main()
```

my-sql-extraction-job

Last modified on 06/01/2025, 14:41:06 Actions Save Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Script Info

```python
import boto3, sys, csv, pymysql, io, json, logging
from datetime import date
from botocore.exceptions import ClientError
from awsglue.utils import getResolvedOptions

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger()

args = getResolvedOptions(sys.argv, ["table_name","delta_value"])
table_name = args["table_name"]
delta_value = args["delta_value"]

def get_rds_credentials(secret_name, region_name):
    session = boto3.session.Session()
    client = session.client(service_name='secretsmanager', region_name=region_name)
```

Python Ln 1, Col 1 Errors: 0 Warnings: 0

my-sql-extraction-job

Last modified on 06/01/2025, 14:41:06 | Actions | Save | Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Job parameters Info

Key	Value - *optional*	
--table_name	flights	Remove
--delta_value	2024-05-29	Remove

Add new parameter

You can add 48 more parameters.

Tags

No tags associated with the resource.

Add new tag

You can add 50 more tags.

my-sql-extraction-job

Last modified on 06/01/2025, 14:41:06

Actions ▼ | Save | Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Job parameters Info

Key	Value - *optional*	
--table_name	passenger_experience	Remove
--delta_value	null	Remove

Add new parameter

You can add 48 more parameters.

Tags

No tags associated with the resource.

Add new tag

You can add 50 more tags.

my-sql-extraction-job

Last modified on 06/01/2025, 14:41:06 | Actions | Save | Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Job parameters Info

Key	Value - *optional*	
--table_name	flight_logistics	Remove
--delta_value	null	Remove

Add new parameter

You can add 48 more parameters.

Tags

No tags associated with the resource.

Add new tag

You can add 50 more tags.

Amazon S3

General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3

Block Public Access settings for this account

Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings

flights_db/

Copy S3 URI

Objects | Properties

Objects (3) Info

Copy S3 URI | Copy URL | Download | Open | Delete | Actions

Create folder | Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

1

Name	Type	Last modified	Size	Storage class
flight_logistics/	Folder	-	-	-
flights/	Folder	-	-	-
passenger_experience/	Folder	-	-	-

glue-spark > spark-transactional-delta-lake.py > ...

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date
from delta.tables import DeltaTable
import logging
import boto3
import sys
from awsglue.utils import getResolvedOptions

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

args = getResolvedOptions(sys.argv, ["table_name"])
table_name = args["table_name"]

spark = SparkSession.builder \
    .appName("Delta Lake Upsert Data Aggregations") \
    .getOrCreate()

s3_bucket_path = "s3://gd-aws-de-labs/"
input_path = f"{s3_bucket_path}/raw_landing_zone/flights_db/{table_name}/"
output_path = f"{s3_bucket_path}/lakehouse-dwh/{table_name}"

primary_key = "flight_id"

df = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load(input_path)

if table_name=='flights':
    df = df.withColumn("flight_date", to_date(col("created_at")))

logger.info(f"DataFrame schema for table {table_name}: {df.dtypes}")

delta_table_exists = DeltaTable.isDeltaTable(spark, output_path)
```

```python
df = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load(input_path)

if table_name=='flights':
    df = df.withColumn("flight_date", to_date(col("created_at")))

logger.info(f"DataFrame schema for table {table_name}: {df.dtypes}")

delta_table_exists = DeltaTable.isDeltaTable(spark, output_path)

if delta_table_exists:
    delta_table = DeltaTable.forPath(spark, output_path)

    logger.info(f"Delta table schema: {delta_table.toDF().dtypes}")

    merge_condition = " AND ".join([f"target.{key} = source.{key}" for key in primary_key])
    delta_table.alias("target").merge(
        df.alias("source"),
        merge_condition
    ).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
else:
    if table_name=='flights':
        df.write.format("delta").partitionBy("flight_date").mode("overwrite").save(output_path)
    else:
        df.write.format("delta").mode("overwrite").save(output_path)

s3 = boto3.client('s3')
bucket_name = s3_bucket_path.split('/')[2]
input_prefix = f"raw_landing_zone/{table_name}/"
archive_prefix = f"archived/{table_name}/"

try:
```

```python
        df.write.format("delta").partitionBy("flight_date").mode("overwrite").save(output_path)
    else:
        df.write.format("delta").mode("overwrite").save(output_path)

s3 = boto3.client('s3')
bucket_name = s3_bucket_path.split('/')[2]
input_prefix = f"raw_landing_zone/{table_name}/"
archive_prefix = f"archived/{table_name}/"

try:
    objects = s3.list_objects_v2(Bucket=bucket_name, Prefix=input_prefix).get('Contents', [])
    for obj in objects:
        source_key = obj['Key']
        destination_key = source_key.replace(input_prefix, archive_prefix)
        copy_source = {'Bucket': bucket_name, 'Key': source_key}
        s3.copy_object(CopySource=copy_source, Bucket=bucket_name, Key=destination_key)
        s3.delete_object(Bucket=bucket_name, Key=source_key)
    logger.info("Files moved to archive successfully.")
except Exception as e:
    logger.error(f"Error moving files to archive: {e}")


spark.stop()
```

testing

Last modified on 06/01/2025, 14:15:40 Actions Save Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Script Info

```python
import sys
import logging
import boto3
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsgluedq.transforms import EvaluateDataQuality
from pyspark.sql.functions import col, to_date

# Logging setup
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

Python Ln 1, Col 1 Errors: 0 Warnings: 0

testing

Last modified on 06/01/2025, 14:15:40 Actions Save Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Job parameters Info

Key	Value - *optional*	
--table_name	flights	Remove

Add new parameter

You can add 49 more parameters.

Tags

No tags associated with the resource.

Add new tag

You can add 50 more tags.

Successfully started job
Successfully started job testing. Navigate to Run details for more details.

testing

Last modified on 06/01/2025, 15:02:46 Actions Save Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Job parameters Info

Key	Value - *optional*	
--table_name	flight_logistics	Remove

Add new parameter

You can add 49 more parameters.

Tags

No tags associated with the resource.

testing

Last modified on 06/01/2025, 15:03:12 Actions Save Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Job parameters Info

Key	Value - *optional*	
--table_name	passenger_experience	Remove

Add new parameter

You can add 49 more parameters.

Tags

No tags associated with the resource.

Add new tag

You can add 50 more tags.

Amazon S3

General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3

Block Public Access settings for this account

Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings

Objects (6) Info

Copy S3 URI | Copy URL | Download | Open | Delete | Actions

Create folder | Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

1

Name	Type	Last modified	Size	Storage class
archived/	Folder	-	-	-
glue-scripts/	Folder	-	-	-
lakehouse-dwh/	Folder	-	-	-
raw_landing_zone/	Folder	-	-	-
spotify_data/	Folder	-	-	-
streams/	Folder	-	-	-

Amazon S3

General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3

Block Public Access settings for this account

Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings

Objects | Properties

Objects (3) Info

Copy S3 URI | Copy URL | Download | Open | Delete | Actions

Create folder | Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

1

Name	Type	Last modified	Size	Storage class
flight_logistics/	Folder	-	-	-
flights/	Folder	-	-	-
passenger_experience/	Folder	-	-	-

Data Catalog
Databases
Tables
Stream schema registries
Schemas
Connections
Crawlers
Classifiers
Catalog settings
Data Integration and ETL
Zero-ETL integrations New
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Interactive Sessions
Data classification tools
Sensitive data detection
Record Matching
Triggers
Workflows (orchestration)
Blueprints
Security configurations

Add data source

Data source
Choose the source of data to be crawled.

Delta Lake

Connection - *optional*
Select a connection to access the data sources below.

Clear selection | Add new connection

Include delta lake table paths
Browse for or enter an existing S3 path.

s3://gd-aws-de-labs/lakehouse-dwh/flights/ | Remove

Add new delta table path

Create tables for querying

- Create Native tables
  Allow integration with query engines that support querying of the Delta transaction log directly.
- Create Symlink tables
  Create a symlink manifest folder with manifest files partitioned by the partition keys, based on specified configuration parameters.

Cancel | Add a Delta Lake data source

Data Catalog
Databases
Tables
Stream schema registries
Schemas
Connections
Crawlers
Classifiers
Catalog settings
Data Integration and ETL
Zero-ETL integrations New
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Interactive Sessions
Data classification tools
Sensitive data detection
Record Matching
Triggers
Workflows (orchestration)
Blueprints
Security configurations

Crawler successfully starting
The following crawler is now starting: "testing"

Crawlers

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

Crawlers (1) Info

Last updated (UTC)
January 6, 2025 at 09:41:54

Action | Run | Create crawler

View and manage all available crawlers.

Filter crawlers

1

Name	State	Schedule	Last run	Last run t...	Log	Table chan...
testing	Running		-	-	-	-

Zero-ETL integrations New

Data Catalog

Databases
Tables
Stream schema registries
Schemas
Connections
Crawlers
Classifiers
Catalog settings

Data Integration and ETL

Zero-ETL integrations New
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Interactive Sessions
Data classification tools
Sensitive data detection
Record Matching
Triggers
Workflows (orchestration)

Crawlers

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

Crawlers (1) Info

Last updated (UTC)
January 6, 2025 at 09:51:13

Action | Run | Create crawler

View and manage all available crawlers.

Filter crawlers

1

Name	State	Schedule	Last run	Last run t...	Log	Table chan...
testing	Ready		Succeeded	January 6, 2...	View log	2 created, ...

Zero-ETL integrations New

Data Catalog

Databases

Tables

Stream schema registries

Schemas

Connections

Crawlers

Classifiers

Catalog settings

Data Integration and ETL

Zero-ETL integrations New

ETL jobs

Visual ETL

Notebooks

Job run monitoring

Interactive Sessions

Data classification tools

Sensitive data detection

Record Matching

Triggers

Workflows (orchestration)

Databases (2)

Last updated (UTC)
January 6, 2025 at 09:35:49

Edit | Delete | Add database

A database is a set of associated table definitions, organized into a logical group.

Filter databases

1

Name	Description	Location URI	Created on (UTC)
flights_data	-	-	January 6, 2025 at 09:35:47
mobile_network_aggregations	-	-	December 29, 2024 at 07:34:24

Zero-ETL integrations New

Data Catalog

Databases
Tables
Stream schema registries
Schemas
Connections
Crawlers
Classifiers
Catalog settings

Data Integration and ETL

Zero-ETL integrations New
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Interactive Sessions
Data classification tools
Sensitive data detection
Record Matching
Triggers
Workflows (orchestration)

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

flights_data

Last updated (UTC)
January 6, 2025 at 09:49:55

Edit | Delete

Database properties

Name	Description	Location	Created on (UTC)
flights_data	-	-	January 6, 2025 at 09:35:47

Tables (3)

Last updated (UTC)
January 6, 2025 at 09:55:08

Delete | Add tables using crawler | Add table

View and manage all available tables.

Filter tables

1

Name	Database	Location	Classific...	Depreca...	View data	Data quality	Column st
flight_logistics	flights_data	s3://gd-aws-de-	Parquet	-	Table data	View data qualit	View statis
flights	flights_data	s3://gd-aws-de-	delta	-	Table data	View data qualit	View statis
passenger_expe	flights_data	s3://gd-aws-de-	Parquet	-	Table data	View data qualit	View statis

Amazon Athena > Query editor tabs

Editor | Recent queries | Saved queries | Settings

Workgroup: primary

Data

Data source: AwsDataCatalog

catalogue: None

Database: flights_data

Tables and views

Create

Filter tables and views

Tables (3) 1

- flight_logistics
- flights

Query 1 | Query 2 | Query 3 | Query 4 | Query 5

```
SELECT * FROM "flights_data"."flight_logistics" limit 10;
```

SQL Ln 1, Col 1

Run again | Explain | Cancel | Clear | Create

Reuse query results up to 60 minutes ago

passenger_experience

Views (0) 1

Query results | Query status

Completed Time in queue: 87 ms Run time: 467 ms Data scanned: 479.20 KB

Results (10)

Copy Download results

Search rows 1

#	flight_id	fuel_usage	maintenance_check	baggage_errors	safety_incidents
1	FL1000-676	14032.87942444	0	5	0
2	FL1001-278	17636.57421086	0	2	2
3	FL1001-382	12101.33745187	0	5	0
4	FL1003-509	8001.70091412	0	2	1
5	FL1004-893	18434.16753342	0	5	1
6	FL1005-220	13874.24824317	1	4	1
7	FL1005-904	7375.96111243	1	3	0
8	FL1006-188	9460.68282960	1	3	1