# AWS Integrated Assignment – Data Engineering Pipeline (S3 + Lambda + RDS + EC2)

# AWS Integrated Assignment – Data Engineering Pipeline (S3 + Lambda + RDS + EC2)

## Scenario
Traya is collecting order data from multiple sales channels. The raw data is uploaded daily to an S3 bucket by sales teams.
Your task is to build an **end-to-end AWS pipeline** that processes this data, stores it in RDS, and runs analytics jobs from an EC2 instance.

## Tasks

### 1. S3 Setup and Data Upload
- Create an S3 bucket named `traya-orders-data`.
- Upload the provided CSV file (`orders_detailed.csv`) into a prefix `raw/`.
- Configure bucket lifecycle rules to move files older than 15 days to Glacier.

### 2. Lambda Transformation Function
- Create a **Lambda function** triggered by new uploads to `raw/`.
- The function should:
1. Read the CSV from S3.
2. Drop any records with blank `customer_name` or negative/zero `price`.
3. Add a new calculated column `total_value = quantity * price`.
4. Normalize city names (title case) and trim spaces.
5. Write the cleaned data to S3 under `processed/cleaned_.csv`.

- Use **pandas** inside Lambda. This requires packaging it as a **Lambda Layer**.

### 3. RDS Integration
- Create an **RDS MySQL instance** (db.t3.micro, free-tier eligible).
- Define a table `orders_cleaned` with matching schema to the cleaned CSV.
- Lambda (or EC2) should write the cleaned records into RDS using `pymysql` or `sqlalchemy`.

### 4. EC2 Analytics Script
- Launch an EC2 instance (Amazon Linux 2023).
- Install Python, `boto3`, and MySQL client libraries.
- Write a Python script to:
1. Query `orders_cleaned` from RDS.
2. Compute:
- Top 5 cities by total sales.
- Average order value per channel.
- Discount usage rate per product.
3. Save the results as a JSON summary file.
4. Upload this file to `s3://traya-orders-data/analytics/summary_.json`.

- Schedule the script using cron to run every 12 hours.

### 5. (Bonus – Advanced)
- Add an SNS topic to send an alert email if any day's total sales exceed ■50,000.
- Secure RDS access via Security Group allowing only EC2 private IP access.

## Deliverables
1. Architecture diagram (hand-drawn or Lucidchart).
2. S3 bucket screenshot showing raw, processed, and analytics folders.
3. Lambda code (with pandas layer setup steps).
4. RDS schema and sample query output.
5. EC2 cron job snippet and S3 uploaded summary JSON.
6. (Bonus) SNS alert email screenshot.

## Evaluation Criteria
- 20 pts: S3 & Lifecycle Policy setup
- 25 pts: Lambda Transformation & Layer Packaging
- 25 pts: RDS Integration & Correct Table Schema
- 20 pts: EC2 Automation & JSON Upload Logic
- 10 pts: Code readability, comments, and error handling
- **+10 Bonus** for SNS alert setup or CloudWatch metric trigger

---

## Hints
- Use `boto3.client('s3')` in Lambda for S3 operations.
- Use `pymysql.connect()` to connect to RDS in Lambda or EC2.
- Ensure IAM roles grant only the needed permissions (least privilege).
- Debug locally before deploying Lambda (use test event JSONs).

## Objective
This assignment will test your end-to-end cloud engineering ability — combining storage (S3), compute (Lambda, EC2), and database (RDS) into a production-like data flow.