Full Stack Engineering Challenge: O240

Music Festival Lineup

Background

An event planning company is hired to plan a music festival, and they need insights from their data that are not easily accessible from a CSV. So they want to be able to upload their CSV somewhere and have the service populate the data in a database, so they can easily query logistics about the performances.

Table Structure:

Performer	Stage	Start	End	Date
Megan Thee Stallion	Main Stage	8:00	10:00	2025-07-12
Olivia Rodrigo	Side Stage	7:00	9:00	2025-07-12
Rene Rapp	Outdoor Stage	4:30	5:30	2025-07-13

Example CSV: Music Festival Example Data

Note: *Your service does not validate the data (i.e., if artists' times overlap on the same stage), it is just there to populate the information given so that it can be queried later.*

Objective

Design and implement a service to parse the CSV (once it is uploaded) and populate the database. This database will enable insights by efficiently retrieving performance information by stage, performer, and time range.

Requirements

- **Services:** Utilize AWS S3 event notifications, AWS Lambda, AWS DynamoDB, AWS SNS
- **Programming Language:** Use Python 3.10 (vanilla). The only permitted external library is the AWS SDK for Python (Boto3).
- **Data Model:** Your model should enable efficient querying by Stage, Performer and Timeslot.
- **Production Readiness**: Implement a solution that is secure, scalable, and cost-efficient.
- **Cost and Scalability Analysis:** Provide an analysis document estimating infrastructure costs for handling 1,000, 10,000 and 100,000 daily records. Discuss scalability and cost-efficient strategies. File Size: Assume that every 100 lines of CSV data is approximately 5kB.

Implementation Instructions

- **File Upload:** Use AWS S3 event notifications to trigger the Lambda when a file is uploaded to an S3 bucket.
- **Lambda Function Implementation:**
  - The function should efficiently parse the data and upload it to a DynamoDB table.
  - After execution, the Lambda function should trigger an SNS notification to send an email about the success or failure of the operation.
- **DynamoDB Design Schema:** Create a DynamoDB table(s) schema optimized for querying performance data. Justify your design schema choices.
  - Your schema should efficiently support queries such as:
    - Retrieve all performances by a specific performer.
    - List all performances occurring within a given time range.
    - Fetch details for a specific performance given a stage and time.
- **Security:** Ensure all the resources have only the required permissions using AWS IAM. Follow the principle of least privilege. Justify your role and permission choices.
- **Documentation:** Prepare a README.md file including:
  1. An overview of the DynamoDB design schema and the rationale behind your choices.
  2. A cost and scalability analysis.

Evaluation Criteria

- **Functionality:** The Lambda function triggers only when a file is uploaded to S3 and sends an SNS email notification upon success.

- **Production Readiness:** The service is secure, scalable and cost-effective.
- **Database Design Efficiency:** Demonstrates an understanding of DynamoDB best practices and efficiently handles the required queries.
- **Documentation:** Clear, concise, and informative.

Bonus

Utilize AWS CloudFormation to define and deploy all required infrastructure components (S3 bucket, Lambda, DynamoDB, SNS) in a YAML template.

Submission Instructions

- The Lambda function code.
- The README.md documentation.
- CloudFormation YAML file for infrastructure.

Be prepared to showcase your solution and speak to your design decisions.