Coding Challenge: Football Match Data Processor

Please complete this coding challenge and submit your solution within **7 days** of receiving it. If you need additional time or have any questions, feel free to reach out to us.

Objective:
To assess your expertise in AWS CDK (TypeScript), serverless architecture, and application development using TypeScript or Python, we've designed a football-themed coding challenge. You will build a serverless application that processes and analyzes football match data in real-time.

Challenge Overview

You will create a serverless application on AWS that:

1. Ingests live football match event data (e.g., goals, passes, fouls) via an API.
2. Processes and enriches the data in real-time.
3. Stores the processed data in a DynamoDB table.
4. Provides a REST API to query match statistics (e.g., total goals, total passes).

The infrastructure will be defined using **AWS CDK in TypeScript**, and the application logic will be implemented in **TypeScript or Python**.

Requirements

1. Infrastructure as Code (AWS CDK)

- Use **AWS CDK in TypeScript** to define the infrastructure.
- Deploy the following resources:
  - **API Gateway**: To expose a REST API for querying match statistics.
  - **Lambda Functions**: For data ingestion, processing, and querying.
  - **DynamoDB Table**: To store processed match event data.
  - **EventBridge**: To trigger real-time processing of match events.
  - **S3 Bucket**: To store raw match event data (optional).
  - **IAM Roles**: With least-privilege permissions for all resources.

2. Application Development

- **Data Ingestion**:
  - Create a Lambda function that simulates receiving live match event data (e.g., JSON payloads) via an API Gateway endpoint.
  - Example payload:

```
{
  "match_id": "000001",
  "event_type": "goal",
  "team": "Team A",
  "player": "Player 1",
  "timestamp": "2023-10-15T14:30:00Z"
}
```

    - Publish the event to an **EventBridge** bus.
- **Data Processing**:
    - Create a Lambda function triggered by EventBridge to process and enrich the event data.
    - Enrichment example: Add a `season` field based on the timestamp (e.g., "2025-2026").
    - Store the enriched data in a **DynamoDB table**.
- **Query API**:
    - Create a Lambda function to query match statistics from DynamoDB.
    - Expose the following endpoints via API Gateway:
        - `GET /matches/{match_id}/goals`: Returns the total number of goals for a match.
        - `GET /matches/{match_id}/passes`: Returns the total number of passes for a match.

3. Bonus (Optional)

- Use **Amazon MSK (Kafka)** to stream match events instead of EventBridge.
- Add a **Step Function** to orchestrate the data processing workflow.
- Implement unit tests for your Lambda functions.

Deliverables

1. **AWS CDK Code**: A TypeScript project that defines the infrastructure.
2. **Application Code**: TypeScript or Python code for the Lambda functions.
3. **README File**: Instructions on how to deploy and test the application.
4. **Postman Collection**: To test the REST API endpoints.

Evaluation Criteria

1. **Infrastructure as Code**:
    - Correct use of AWS CDK to define resources.
    - Adherence to best practices (e.g., least-privilege IAM roles).
2. **Application Logic**:
    - Correct implementation of data ingestion, processing, and querying.
    - Code readability, modularity, and error handling.
3. **Scalability and Performance**:
    - Use of serverless services to ensure scalability.
    - Efficient DynamoDB table design (e.g., partition keys, indexes).
4. **Bonus Points**:
    - Implementation of optional features (e.g., MSK, Step Functions).
    - Unit tests and documentation.

Example Workflow

1. A match event (e.g., goal) is sent to the API Gateway endpoint.
2. The ingestion Lambda function publishes the event to EventBridge.
3. The processing Lambda function enriches the event and stores it in DynamoDB.
4. A user queries the total goals for a match via the REST API.
5. The query Lambda function retrieves the data from DynamoDB and returns the result.

Submission

- Share your code via a **Git repository** (e.g., Bitbucket, GitHub, GitLab).
- Include a **README** with deployment instructions and sample test data.

This challenge is designed to test your ability to design and implement a serverless, event-driven application on AWS while working with football-related data. Good luck, and we look forward to seeing your solution!