Technical Assessment: The "Firehose" Collector

Time Limit: 3-5 Hours

Context

We are designing a new ingestion service for a high-traffic analytics platform. This service will receive "clickstream" events from millions of client devices.

The Challenge

Build a Python-based HTTP service capable of ingesting **5,000 requests per second** on a standard cloud instance without dropping data or crashing.

Requirements

1. The API Endpoint

Create a single endpoint: POST /event

- **Payload:** A JSON object containing user_id (int), timestamp (iso-string), and metadata (arbitrary nested JSON).
- **Response:** Must return HTTP 202 (Accepted) immediately.

2. Architecture & Persistence

- **Non-Blocking:** The API must not wait for the database write to complete before responding to the client.
- **Batching/Buffering:** You must implement a strategy to group insertions. Writing to the database on every single HTTP request is forbidden.
- **Storage:** Data must eventually be stored in a SQL database (SQLite is fine for this demo, pretend it is PostgreSQL).
- **Safety:** The metadata field is unstructured. Ensure your SQL storage strategy handles arbitrary JSON safely without injection risks.

3. Resilience

- Simulate a database outage or slowdown (e.g., lock the database for 5 seconds). Your API should continue accepting HTTP requests during this window and sync the data once the database recovers.

Deliverables

1. **Source Code:** Your Python application.

2. **Architecture Diagram/Description:** A text file or image explaining how you handled the buffering (e.g., "I used an in-memory deque," or "I used Redis").
3. **Load Test Script:** A simple script (using locust, wrk, or Python asyncio) that demonstrates your server handling at least 1,000 concurrent requests.

Grading Criteria

- **Throughput:** Does the server lock up under load?
- **Security:** How do you handle the arbitrary JSON metadata in a SQL context?
- **Code Quality:** Clean separation of concerns between the API layer and the Storage layer.