Backend Engineering Task: Robust Data Processor

The Zoom-Out

Your task is to build a scalable, robust API backend. This system must be capable of ingesting massive streams of unstructured logs from diverse sources, run a simulated time consuming worker on them, and store the files securely.

We are not evaluating your frontend skills. You don't need to build a frontend at all. We will test by simply making calls to the endpoint of your backend.

We are looking for a **Backend Engineer** who can architect a **Multi-Tenant, Event-Driven Pipeline** on **Google Cloud (GCP) or AWS** that normalizes chaotic data inputs into a clean, resilient stream.

Feel free to use any AI-assisted coding tools like Copilot, Claude, or Cursor.

1. The Core Challenge

Build a Unified Ingestion Gateway that handles multiple data formats and isolates data by tenant.

The API will have a simple `/ingest` endpoint. That's all. Deploy this API such that it:

1. **Normalizes Data:** Ingests both **JSON** and **TXT** payloads, normalizing them into a single internal, flat **txt** format.
2. **Ensures Isolation:** Processes data asynchronously and saves it to a database structure that strictly separates "User A" from "User B".
3. **Survives Chaos:** Handles heavy traffic loads and recovers gracefully if the worker crashes mid-process. We will hit the endpoint with a high RPM (Requests Per Minute).

2. The Architecture

You have freedom to choose the compute services (Cloud Run, Lambda, etc.) that best fit the problem. Some guidelines:

- **Orchestration:** Use a managed message broker (GCP Pub/Sub or AWS SQS/SNS).
- **Compute:** Serverless (Cloud Run, Functions, or Lambda).
- **Storage:** NoSQL DB (Firestore or DynamoDB).

The Data Flow: `[Source: JSON/TXT] -> (API Endpoint) -> [Message Broker] -> (Worker) -> [NoSQL DB]`

3. Functional Requirements

A. The Unified Ingestion API (Component A)

Your API must expose one single endpoint: `POST /ingest`. It must be **non-blocking** (Async) and capable of handling at least **1,000 RPM** (Requests Per Minute).

Scenario 1: Structured Data (JSON)

- **Header:** `Content-Type: application/json`
- **Payload:** `{"tenant_id": "acme", "log_id": "123", "text": "User 555-0199..."}`
- **Action:** Validate, serialize to internal txt format, and publish to the Broker.

Scenario 2: Unstructured Data (Raw Text)

- **Header:** `Content-Type: text/plain`
- **Header:** `X-Tenant-ID: acme` (You must extract the tenant from here).
- **Payload:** A raw text string (e.g., a log file dump).
- **Action:** Publish to the *same* Broker topic used in Scenario 1.

B. The Worker (Component B)

This service is triggered via the Message Broker. It pulls the normalized messages and performs the work.This is a dummy CPU bound process which takes time.

1. **Simulate "Heavy" Processing:**
    - Sleep for `0.05s` per character of text (e.g., 100 chars = 5s sleep).

C. Storage (Multi-Tenancy)

You must save the processed metadata to a NoSQL DB (Firestore/DynamoDB)

Critical Requirement: You must strictly isolate tenants using **Sub-collections** or **Partition Keys**.

- **Schema Structure (NoSQL):** `tenants/{tenant_id}/processed_logs/{log_id}`

Example NoSQL Document:

```json
{
  "source": "text_upload",
  "original_text": "User 555-0199 accessed...",
  "modified_data": "User [REDACTED] accessed...",
```

```
  "processed_at": "2023-10-27T10:00:00Z"
}
```

4. The Constraints

- **Deployment:** Must be live on the public internet. Use **GCP Free Tier** or **AWS Free Tier**.
- **Auth:** No authentication required for the API **(keep it public so we can test it).**
- **Scale to Zero:** Use Serverless. **DO NOT** use "always on" VMs (EC2/Compute Engine).
- **Infrastructure as Code:** Optional. You may use the Cloud Console GUI to set this up, or Terraform if you prefer.

5. How We Will Test It (The Rubric)

We will run a "Chaos Script" against your live URL.

1. **The Flood:** We will fire **1,000 requests per minute**. Mixed JSON and TXT.
    - *Pass:* API responds instantly (`202 Accepted`).
    - *Fail:* API hangs or times out.
2. **The Isolation Check:** We will inspect your Database.
    - *Pass:* `acme_corp` data is physically separated from `beta_inc` data in the path/collection structure.
    - *Fail:* All logs are dumped into one flat `all_logs` table.

6. Deliverables

1. **The Live URL:** We will run a `curl` request to your API.
2. **Video Walkthrough (3-5 mins):**
    - Walk us through your Cloud Console (show the Queue and Database).
    - Show a request hitting the API and appearing in the DB.
    - Explain your Multi-Tenant architecture choice.
3. **Code + README:**
    - Source code.
    - Architecture Diagram (Show how TXT and JSON paths merge).
    - Short explanation of how you handled the "Crash Simulation."