# Event Logger Service - Technical Case

## Project Overview

You are tasked with building a lightweight Event Logger Service that accepts application events via an API, queues them using a message broker (like Kafka or RabbitMQ), and stores them in a database asynchronously. This simulates a real-world architecture for logging, analytics, or telemetry systems.

## Goals

- Accept event logs through a REST API.
- Push event logs to a message broker.
- Consume events from the broker and persist them in a database.
- Dockerize the entire solution using docker-compose.

## Architecture

[Client / App]

|

v

[REST API Service]

|

v

[Message Queue (Kafka or RabbitMQ)]

|

v

[Consumer Service]

|

v

[MongoDB or PostgreSQL]

## Requirements

### Backend Language (pick one):

- Go

- Python (preferred)

- Node.js (Express)

### Tools:

- Message Broker: Kafka or RabbitMQ

- Database: MongoDB or PostgreSQL

- Containerization: Docker & Docker Compose

## API Endpoints

### POST /events

- Description: Accepts an event and pushes it to the message queue.

- Request Body:

```
{
"source": "auth-service",
"type": "user_login",
"payload": {
"userId": "abc-123",
"ip": "192.168.0.10"
},
"timestamp": "2025-08-22T14:00:00Z"
}
```

- Response:

```
{
"status": "queued",
"eventId": "generated-uuid"
}
```

## Consumer Responsibilities

- Listen to the queue topic.

- Validate incoming event structure.

- Store the event in the database.

### Event Schema (Database)

```
{
"eventId": "uuid",
"source": "string",
"type": "string",
"payload": { "object" },
"timestamp": "ISO8601 datetime",
"receivedAt": "system timestamp"
}
```

## Docker Setup

### docker-compose.yml should include:

- Backend API container

- Message queue container (Kafka or RabbitMQ)

- Consumer service container

- MongoDB or PostgreSQL container

## Evaluation Criteria

Category	Description
Correctness	API functions correctly, message flow works end-to-end
Clean Code	Follows good practices, modular and well-documented
Dockerization	docker-compose up brings up all services
Resilience	Handles malformed input and queue failures

| Bonus | Monitoring endpoint or dashboard (e.g., /health, /stats) |

## Bonus Challenges (Optional)

- Add a GET /events endpoint to view stored events.

- Add filtering by source or type.

- Add Grafana + Prometheus to monitor queue/consumer.

- Add basic authentication to the API.

## Submission Guidelines

- Upload your code to GitHub.

- Include a README.md explaining:

- How to run the project locally

- API usage with curl/Postman examples

- Technologies used and design decisions

- Deadline: [define based on your schedule]