How To Monitor My Agent 🔍

You will build a platform for one of the important pillars of building AI agents: monitoring and observability.

Your task is to build a simplified version of how a monitoring service tool would work by ‘mocking’ events using an LLM (in the real world, these would be conversations), and creating a simple API that will ingest those events and create reports.

Time estimate: 2-3 hours. Quality is more important but bonus points for being quick.

Time limit: 24 hours from the moment you receive this guide.

Overview

You will build and deploy two Dockerized TypeScript services on any cloud provider of your choice with bonus points for deploying on AWS. Please include an explanation of your chosen cloud architecture and how the services integrate regardless of the platform you select.

1. **Service A (Ingest & Metrics)**

- Accepts simulated LLM-agent events, maintains rolling aggregates (last 5 minutes or 30 events), and exposes Prometheus-style and JSON metrics.

2. **Service B (Simulator)**

- Every minute calls an LLM API to generate one plausible event and POST(s) it to Service A may optionally have robust error catching and latency optimizations.

Both services must be publicly accessible so we can verify end-to-end functionality.

Service A: Ingest & Metrics

Endpoints

1. **POST /ingest**

Accepts one event JSON:

```
{
   "timestamp": "2025-06-10T12:34:56.789Z",
   "sessionId": "abc123",
   "intent": "BookFlight",
   "latencyMs": 120,
   "success": true,
   "confidence": 0.87
}
```

- Validate fields; return 400 on invalid payload.
- Store in a rolling window of the last 5 minutes **or** the most recent 30 events (whichever yields fewer).

2. **GET /metrics**

Returns Prometheus-like (can be different) plaintext metrics (e.g. runtime, latency, cost metrics):

```
voice_events_total 42
voice_latency_ms_avg 135.4
voice_error_rate 0.095
voice_confidence_avg 0.88
```

2. **GET /report**

Returns JSON summary:

```
{
   "totalEvents": 42,
   "avgLatencyMs": 135.4,
   "errorRate": 0.095,
   "avgConfidence": 0.88
}
```

Rolling Aggregates

Maintain in memory (e.g. deque or ring buffer) constrained by:

1. **Time window:** events within the last 5 minutes
2. **Count window:** up to the most recent 30 events

Use whichever filter results in fewer events for your calculations.

Metrics to compute:

```
totalEvents
avgLatencyMs
errorRate = 1 − (successCount / totalEvents)
avgConfidence
```

Packaging & Deployment

1. Node.js + TypeScript
2. Docker image listens on port 8080
3. Deploy it to the cloud, preferably to AWS
4. Provide public URLs for /ingest, /metrics, /report

Service B: Event Simulator

Behavior

1. Every minute, call an LLM through an API to generate one JSON event matching Service A's schema.
    a. We recommend you use Mistral's or Grok's API, as they each provide free tokens, but you may choose any provider you know best, keeping in mind the optional optimization challenge. Example:

```
curl https://api.groq.com/openai/v1/chat/completions -s \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <>" \
-d '{
```

```
"model": "meta-llama/llama-4-scout-17b-16e-instruct",
"messages": [{
   "role": "user",
   "content": "Explain the importance of fast language models"
}]
}'
```

2. POST the generated event to Service A's /ingest URL.

Bonus: Optimize for latency/error handling Service's B response.

Packaging & Deployment

1. Node.js + TypeScript
2. Config via environment variables:

```
LLM_OF_CHOICE_KEY
TARGET_URL (Service A's /ingest URL)
```

3. Docker image that launches the simulation on container start
4. Deploy to Cloud Run with a public endpoint

README Requirements

Clearly specify:

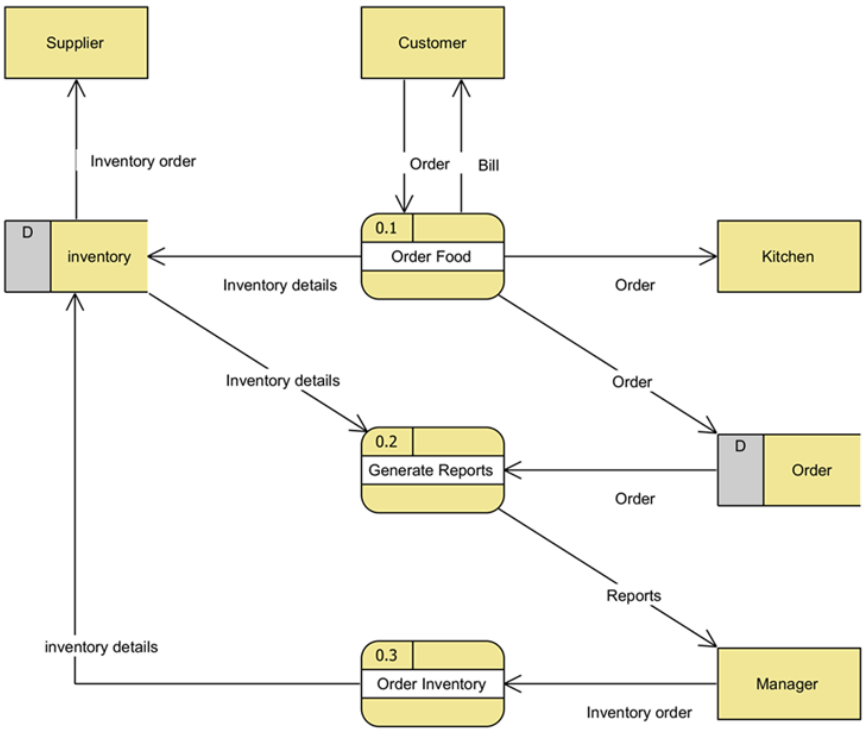
1. How to call and test each endpoint (/ingest, /metrics, /report) using Cloud Run URLs
2. Anything else you would like us to know about your implementation/challenges and how you approached them.
3. Any optimizations or thoughtful error handling you implemented.

Submission

1. Create a **private GitHub repository** and grant collaborator access to:

   `kaitakami` `SomeoneElseSt` `donadol`

2. Include all source code, Dockerfiles, and README.
3. Provide the public Cloud Run URLs for both services.
4. Indicate approximate time spent (this helps us understand how long candidates typically take, though it won't influence our decision)
5. Loom video no longer than five minutes with you covering:
    a. A flowchart diagram of how data flows through your system and high-level explanation of how it works. If you did not use AWS you must also explain how the system you used integrates into the services. Example:



    b. You must compare what you are showing in the diagram with actual snippets of the code you made, connecting code specifics with how they work as part of an overarching system.

Send all submissions material through the same medium where you received this guide.

Good luck!