Google Air Quality Monitoring Across GCC Cities with Queue

Overview

You are tasked with designing and implementing a **real-time event-driven system** using **NestJS** that monitors air quality in **Riyadh, Dubai, Doha, and Muscat**.

The system should:

1. Poll the **Google Maps Air Quality API** for the four cities. (Developer should use their own api key)
2. Detect unhealthy air-quality levels based on AQI or pollutant concentration.
3. Publish alerts as events to **RabbitMQ or Similar services**.
4. Consume and log those alerts using another NestJS micro-service.
5. Expose a REST endpoint (/alerts) to retrieve recent alerts.
6. Run the entire system locally using **Docker and docker-compose** (mandatory).

Optional : Be deployed to **AWS EC2** via a **CDK stack** that auto-configures Docker and runs the system using `docker-compose`.

*(**The CDK does not have to be production-ready; a clear template configuration is sufficient.**)*

Monitored Cities

City	Latitude	Longitude	Country Code
Riyadh	24.7136	46.6753	SA
Dubai	25.2048	55.2708	AE
Doha	25.2854	51.5310	QA
Muscat	23.5880	58.3829	OM

Each city must be polled separately in a concurrent manner.

Mandatory Tasks (≈ 8 hours)

Component 1: Data Collector Service

Purpose:

Continuously poll the **Google Maps Air Quality API** for the four cities and trigger critical air-quality alerts.

Polling Endpoint:

```
None
POST:<https://airquality.googleapis.com/v1/currentConditions:lookup?key=YOUR_API_KEY>
```

Example Request Body:

```
JSON
{
  "location": { "latitude": 25.2048, "longitude": 55.2708 },
 }
```

Key Extracted Fields:

- `indexes[].aqi` → Universal AQI (UAQI)
- `indexes[].category` → AQI Category
- `indexes[].color` → RGB color (AQI severity)
- `indexes[].dominantPollutant`

Critical Thresholds:

Parameter	Critical Level	Description
PM2.5	> 100 µg/m³	Fine particulate matter
PM10	> 150 µg/m³	Coarse dust particles
UAQI	> 100	“Unhealthy” AQI or worse

Behavior:

- Poll each city every 10 seconds.
- If any pollutant or AQI exceeds the threshold, publish a message to RabbitMQ.

Example Published Event

JSON


```
{
  "dateTime": "2025-11-02T08:00:00Z",
  "regionCode": "ae",
  "indexes": [
    {
      "code": "uaqi",
      "displayName": "Universal AQI",
      "aqi": 45,
      "aqiDisplay": "45",
      "color": {
        "red": 1,
        "green": 0.8862745
      },
      "category": "Moderate air quality",
      "dominantPollutant": "pm25"
    }
  ]
}
```


Component 2: Alert Processor Service

Purpose:

Consume air-quality alerts and process them for logging, analysis, or display.

Responsibilities:

- Subscribe to RabbitMQ queue.
- Parse and validate message payloads.
- Log the alerts for further analysis.
- Persist the data in DB (Postgres, MongoDB…)

Example Log Output:

```
None
[ALERT] CRITICAL AIR QUALITY DETECTED
City: Dubai | Region: AE
AQI: 180 | Category: Unhealthy
PM2.5: 172.4 µg/m³ | PM10: 205.8 µg/m³
Dominant: pm25 | Color: #FF0000
```

Component 3: Alert Endpoint

Provide a REST endpoint `/alerts` to deliver real-time alerts to clients. It should return the most recent alerts (max 20 alerts).

```
JSON
[
      {
            "city": "Doha",
            "aqi": 176,
            "category": "Unhealthy",
            "timestamp": "2025-11-02T10:30:00Z"
      },
      {
            "city": "Dubai",
            "aqi": 136,
            "category": "Unhealthy",
            "timestamp": "2025-11-03T10:30:00Z"
      }
 ]
```

Component 4: Docker Setup

Containerize all components to ensure a consistent, runnable local environment.

Deliverables:

- Individual Dockerfiles for both NestJS services.
- A `docker-compose.yml` orchestrating both services and RabbitMQ & DB.

Bonus Tasks (Optional – 2 to 4 Hours)

If time allows, you can extend the system with any of the following:

1. Health Checks

Implement `/health` endpoints in both microservices to verify readiness and liveness status.

2. WebSocket Real-Time Streaming

Create a web socket to stream the alerts to connected client so that can get the live update. Event can be of same response as of alert endpoint

3. AWS CDK

Design a **non-functional CDK template** (no actual deployment required) that outlines how this system could be deployed on AWS EC2.

Evaluation Criteria

Category	Description	Weight
Correctness	End-to-end working event flow	35%
Architecture	Modular, scalable NestJS structure	25%
Code Quality	Clean TypeScript, DTOs, async handling	20%
Docker Composition	Docker files for micro-services	10%
Enhancements	AQI enrichment, dashboards, history	10%

Expected Outcome

1. The Data Collector polls AQI for Riyadh, Dubai, Doha, and Muscat.
2. Whenever AQI > 100 or PM values are high, a critical event is sent to RabbitMQ.
3. The Alert Processor logs real-time alerts for those cities.
4. Alert endpoint should return the latest alerts
5. RabbitMQ confirms message flow.
6. Entire system should run locally via Docker Compose
7. (Bonus) Alerts can be transmitted by web-socket
8. (Bonus) A CDK template demonstrates how it runs in AWS EC2
9. (Bonus) Health check for service readiness