Technical Task: Next.js + Redis Queue + Scheduled Weather Worker Dockerized System (TypeScript)

Objective

Build a small full-stack system that demonstrates:

- Redis-based job queuing (producer–consumer model)
- Automated background scheduling
- Integration with a real external weather API
- Data persistence in PostgreSQL
- Type-safe backend and worker services using TypeScript
- **Next.js frontend + API routes** for UI and backend logic
- Full containerization using Docker Compose

The entire system must run together with:

docker compose up --build

System Architecture

1. Next.js App (Frontend + API)

The Next.js app should include:

- **Page 1 – Dashboard**
  A page with a button labeled “Fetch Weather Now”.
  - Clicking the button triggers a call to `POST /api/job` to enqueue a new weather-fetching job.
  - Display a table with the recent job history either it happened manually or through the producer (schedular).
- **Page 2 – Weather Data**
  A page accessible at `/weather` that displays a table with the latest data for four standard cities:
  - London
  - New York
  - Tokyo
  - Cairo

- The table should include:
    - City name
    - Temperature
    - Wind speed
    - Last updated timestamp
- Below the table, display “Last sync at [timestamp]”, representing when data was last updated in the database.

API Routes

- `POST /api/job` → Adds a weather-fetching job to Redis
- `GET /api/weather` → Returns all stored weather data from PostgreSQL for display on the `/weather` page

---

2. Background Producer (Scheduler)

A TypeScript-based Node.js service running in its own container.
It should automatically enqueue a new job every 60 seconds containing the list of the four standard cities above.

The producer must define clear TypeScript interfaces for:

- Job payload
- City metadata (name, latitude, longitude)
- Environment configuration

---

3. Worker Service (Consumer)

A TypeScript Node.js service that consumes jobs from Redis.
For each job:

Fetch weather data for each city using the **Open-Meteo API**.
Example endpoint for testing (London):
https://api.open-meteo.com/v1/forecast?latitude=51.5072&longitude=-0.1276&current_weather=true

1. Parse the response and extract relevant data such as temperature, wind speed, and timestamp.
2. Upsert each city’s latest weather record in PostgreSQL.

The worker should implement basic error handling, type-safe interfaces for API responses and database rows, and structured logs for debugging.

---

4. Redis

Used as the central job queue connecting:

- The Next.js app (producer via API)
- The background scheduler (automated producer)
- The worker (consumer)

Each service should connect via environment variables and communicate through Docker’s shared network.

---

5. PostgreSQL

Stores the processed weather data for each city.
The schema should support one record per city, always containing the most recent temperature, wind speed, and timestamp.

Developers are expected to design the schema and upsert logic themselves.

Requirements

Stack

- Next.js for frontend + API
- Node.js (TypeScript) for worker and producer
- Redis
- PostgreSQL

Environment Variables
Each container must use environment variables for Redis and PostgreSQL connections.

Docker Compose Setup

All services must be orchestrated using Docker Compose.
The setup should include:

- `app` – Next.js frontend + API (port 3000)
- `worker` – TypeScript worker consuming Redis jobs
- `producer` – TypeScript scheduler adding jobs every minute
- `redis` – Redis server
- `postgres` – PostgreSQL database

The system must start and function correctly using only:

docker compose up --build

Example Flow

1. The user clicks **“Fetch Weather Now”** on the dashboard.
    - This triggers `POST /api/job`, enqueuing a job in Redis.
2. The **worker** consumes the job, fetches weather data for the four standard cities, and stores the results in PostgreSQL.
3. The **/weather** page retrieves the data via `GET /api/weather` and displays the table with temperature, wind speed, and last updated time.
4. The **producer** automatically adds a new job every 60 seconds to keep data refreshed.

Deliverables

A GitHub repository containing:

- `/app` → Next.js frontend + API
- `/worker` → TypeScript consumer service
- `/producer` → TypeScript scheduler service
- `/docker-compose.yml` → Container orchestration
- `/README.md` → Setup and usage instructions

Running the system locally must require only:

docker compose up --build