Abstact

This report documents the engineering decisions and implementation steps behind a real-time Indoor-Air-Quality (IAQ) dashboard built as a graduate capstone. The solution continuously ingests telemetry (CO_2, TVOC, temperature, relative humidity), computes a composite IAQ score, stores the data in a time-series database, and exposes it to a Grafana front-end for visualization and alerting. Key artifacts, including Docker Compose files, SQL schema, and Python ingestion scripts, are provided to ensure full reproducibility.

Introduction

Poor indoor air quality is associated with decreased cognitive performance and adverse health outcomes. Real-time dashboards help facilities staff intervene before thresholds are crossed. The project goal was to deploy an end-to-end pipeline, from sensor API to dashboard, using open-source tooling that can scale from a single device to an enterprise installation.

Method

Overall Architecture

The architecture consists of six integrated layers that communicate over RESTful interfaces or container networks:

1. **IoT Device**
    - Equipped with sensors for CO_2, Temperature, Humidity, and TVOC
    - Collects real-time environmental data every 20–30 seconds

2. **ThingsBoard (IoT Gateway)**
    - Manages authentication and device telemetry
    - Exposes time-series API endpoints over HTTP
    - Acts as a gateway for remote access to sensor data

3. **Python Data Fetcher**
    - Periodically fetches data from ThingsBoard API using a secure token
    - Extracts and transforms sensor data
    - Calculates an IAQ Score using a weighted model
    - Batches and inserts records into the database

4. **TimescaleDB (Docker-Containerized)**
    - Extension of PostgreSQL optimized for time-series data
    - Enables high-performance inserts and queries
    - Stores each sensor reading along with the IAQ score

5. **Grafana (Docker-Containerized)**

- Connects to TimescaleDB as a data source
- Provides live and historical visualizations
- Supports panel creation, filtering, and alert configuration

6. **Email (SMTP Integration)**
   - Configured in Grafana alert settings
   - Automatically sends alerts when IAQ scores fall below thresholds (e.g., IAQ < 60)

Containerized Environment

A two-service stack was orchestrated with Docker Compose. The **TimescaleDB-HA** image bundles PostgreSQL 17 and the Timescale extension; **Grafana** provides visualization and alerting.

docker-compose.yml

```
version: "3.9"

services:
 timescaledb:
   image: timescale/timescaledb-ha:pg17      # PG 17 + TimescaleDB-HA
   container_name: timescaledb
   restart: unless-stopped
   ports:
     - "5432:5432"
   environment:
     # --- minimum viable env vars ---
     POSTGRES_PASSWORD: password            # change in prod
     POSTGRES_USER: postgres                # default anyway
     POSTGRES_DB: iaq                       # creates a starter DB
     # --- HA image tuning (optional) ---
     # TIMESCALEDB_TELEMETRY: "off"
     # PGDATA: /var/lib/postgresql/data/pgdata
   volumes:
     - timescale-data:/var/lib/postgresql/data

 grafana:
   image: grafana/grafana:latest
   ports: ["3000:3000"]

volumes:
 timescale-data:
```

After `docker compose up -d`, the stack is reachable at `localhost:5432` (database) and `localhost:3000` (Grafana).

Database Schema

Once the containers were running, a database and schema were created to store all the data.

```
# create database
docker exec -it timescaledb psql -U postgres -c "CREATE DATABASE
iaq_demo;"

# open a psql shell inside that DB
docker exec -it timescaledb psql -U postgres -d iaq_demo
```

The given code started psql shell where a hypertable was created. A hypertable behaves like a regular PostgreSQL table, but is optimized for time-series workloads by automatically partitioning data across time and space (like by device ID or location).

```
CREATE EXTENSION IF NOT EXISTS timescaledb;

CREATE TABLE iaq_measurements (
  time         TIMESTAMPTZ       NOT NULL,
  device_id    TEXT              NOT NULL DEFAULT 'offline_csv',
  temp_c       DOUBLE PRECISION,
  rh_pct       DOUBLE PRECISION,
  co2_ppm      DOUBLE PRECISION,
  tvoc_ppb     DOUBLE PRECISION,
  iaq_score    INTEGER,
  PRIMARY KEY (time, device_id)
);

SELECT create_hypertable('iaq_measurements', 'time');
```

The old data were downloaded as a CSV file and uploaded into the Docker container, allowing us to mass-load it into the database using the following command.

```
\copy
iaq_measurements(time,temp_c,rh_pct,co2_ppm,tvoc_ppb,iaq_score)
FROM '/tmp/iaq.csv'
WITH (FORMAT csv, HEADER true);
```

Python Fetcher

The Python scripts consisted of three parts: fetching the data, calculating the IAQ index, and inserting the results into the database.

The Python agent uses a .env-based config and continuously polls the API using the following steps:

1. Authenticate via /api/auth/login
2. Pull the latest timeseries data for CO2, Temperature, Humidity, and TVOC
3. Compute IAQ score
4. Insert to TimescaleDB using psycopg2 and execute_values()

5. Sleep and repeat

See Appendix A for full code.

Insert block:

```
sql = """
        INSERT INTO iaq_raw
          (time, device_id, co2_ppm, temp_c, rh_pct,
tvoc_ppb, iaq_score)
        VALUES %s
        ON CONFLICT DO NOTHING
    """
```

IAQ Scoring Function

The IAQ score is computed using a blend of sensor values:

- CO_2 and TVOC sub-indices are banded on defined ranges
- A Thermal Comfort Index (THI) is computed from temperature and humidity via bilinear interpolation
- Weighted formula: IAQ = 0.4*CO2_score + 0.3*TVOC_score + 0.3*THI_score
- Final score clipped to [0, 100] range

See Appendix B for full code.

Grafana Integration

Step 1: Add a new PostgreSQL data source and provide the configuration information.

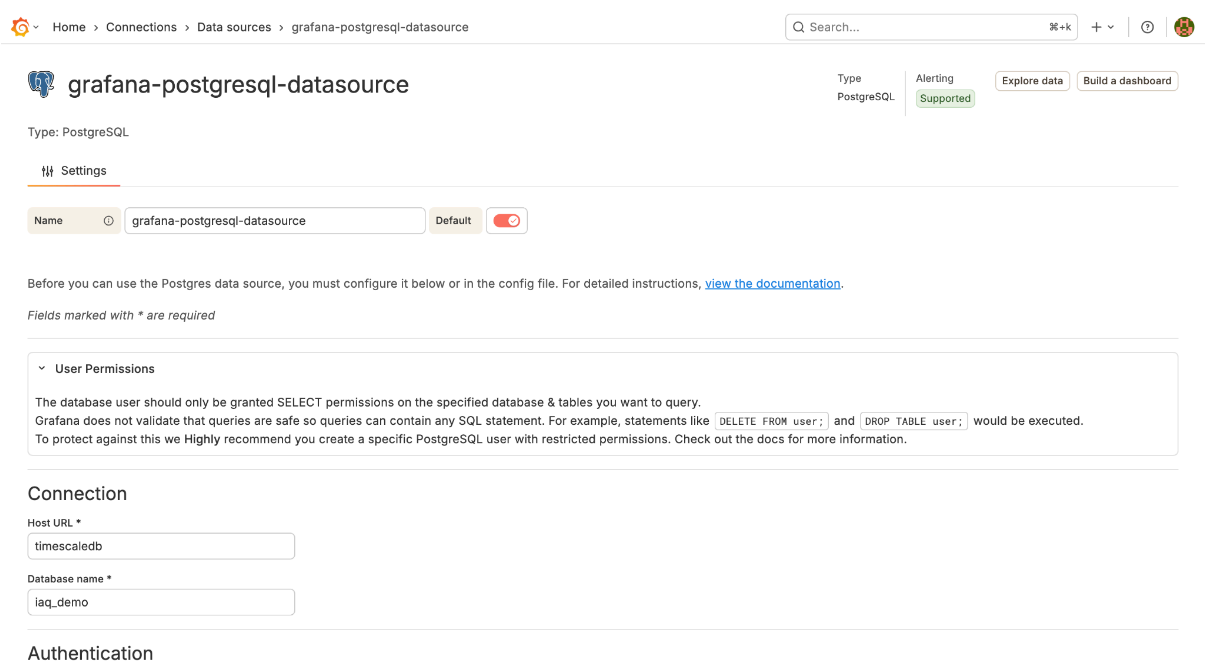

Figure 1: Data Source Connection- Part 1

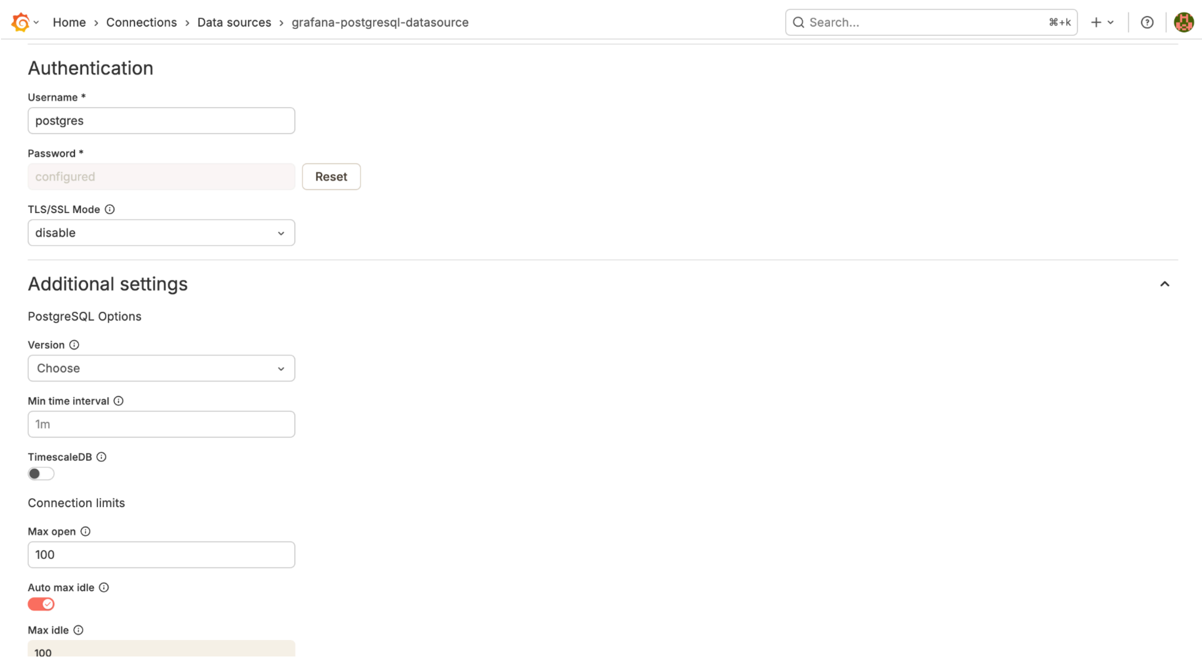

Figure 2: Data Source Connection- Part 1

Step 2: Once the connection is established, configure the alert system. Open the Docker container files -> etc -> grafana -> grafana.ini. Change the SMTP configurations.

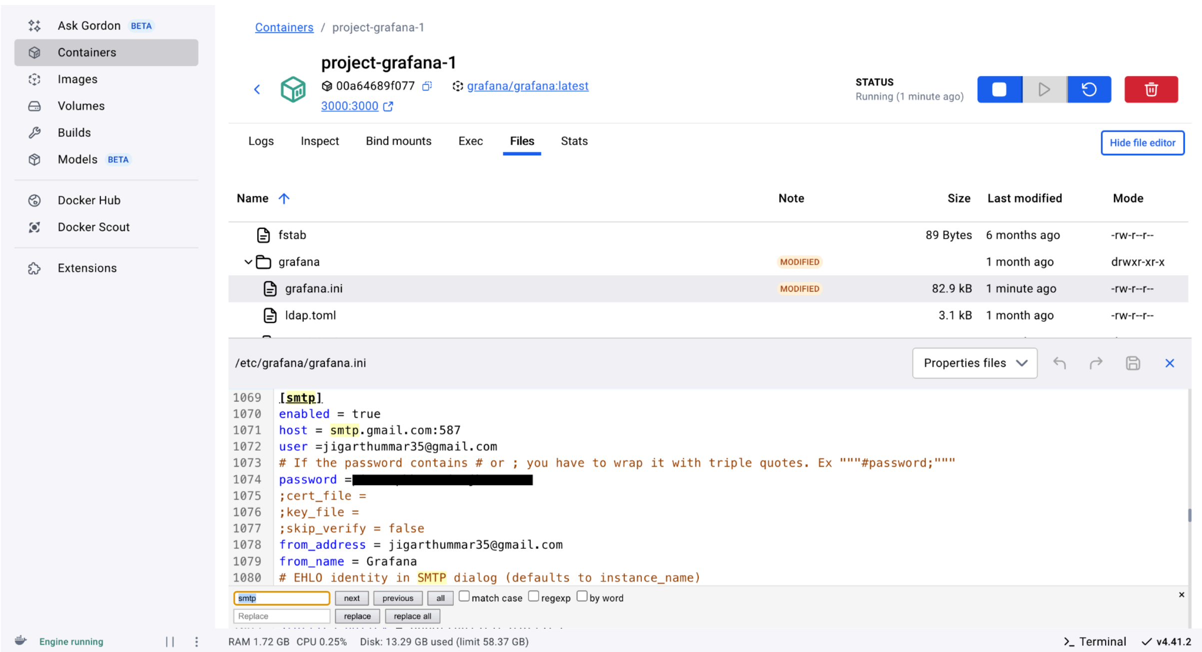

Figure 3: SMTP Configuration

After configuring SMTP, add contact points.

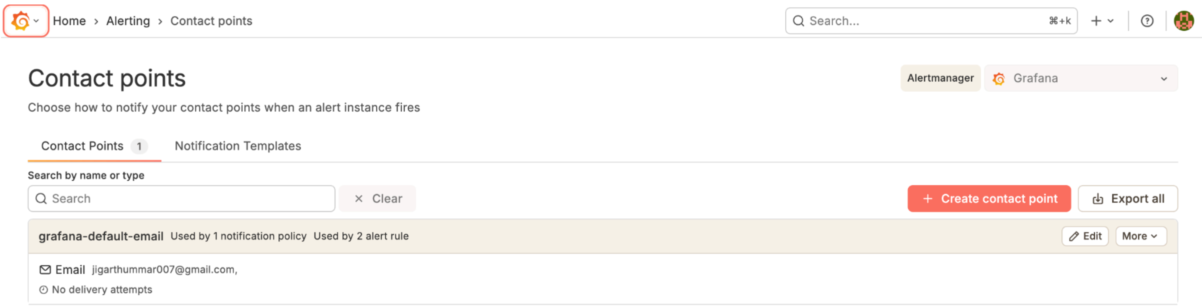

Figure 4: New Contact Point

Create a Dashboard

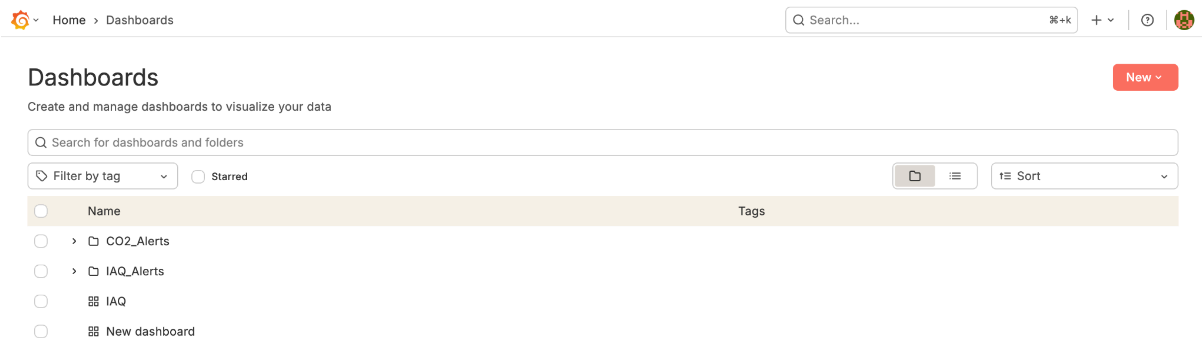

Figure 5: Create A New Dashboard

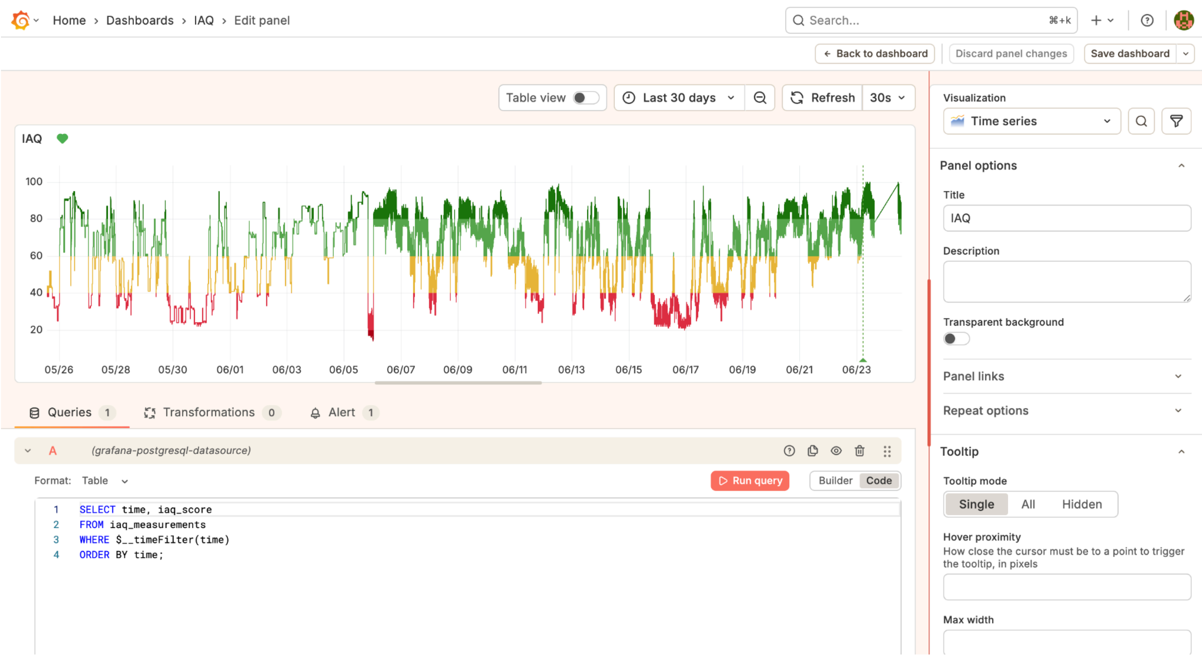

Figure 6: Create A Graph

To create a graph, we switched to Code from Builder and wrote an SQL query to get the data from the database. We adjusted the settings to change the visualization. We followed the same process for all the other graphs.

Setting up the Alert System

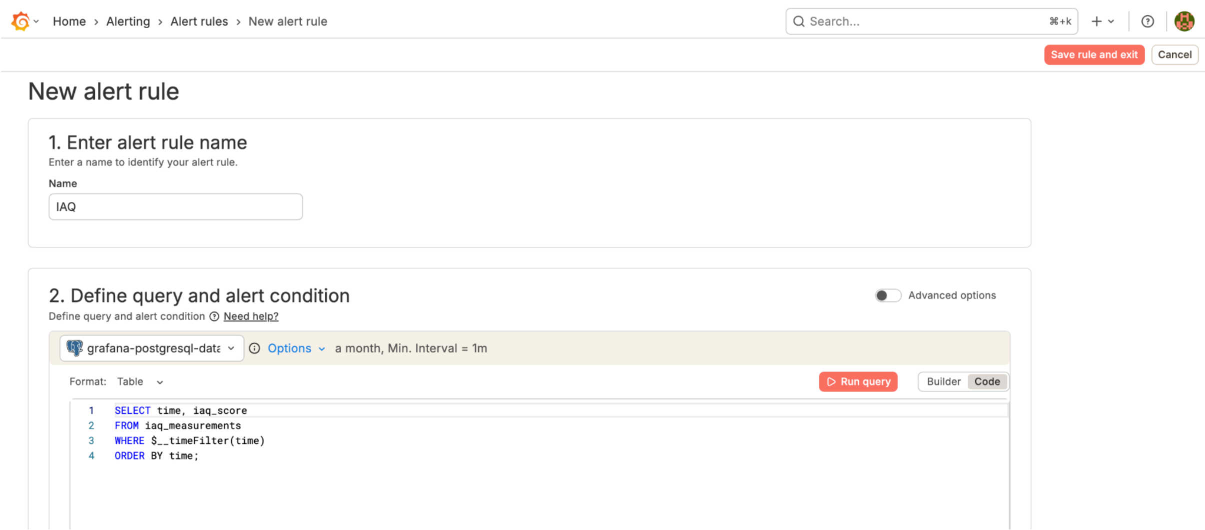

Figure 7: Create A New Alert

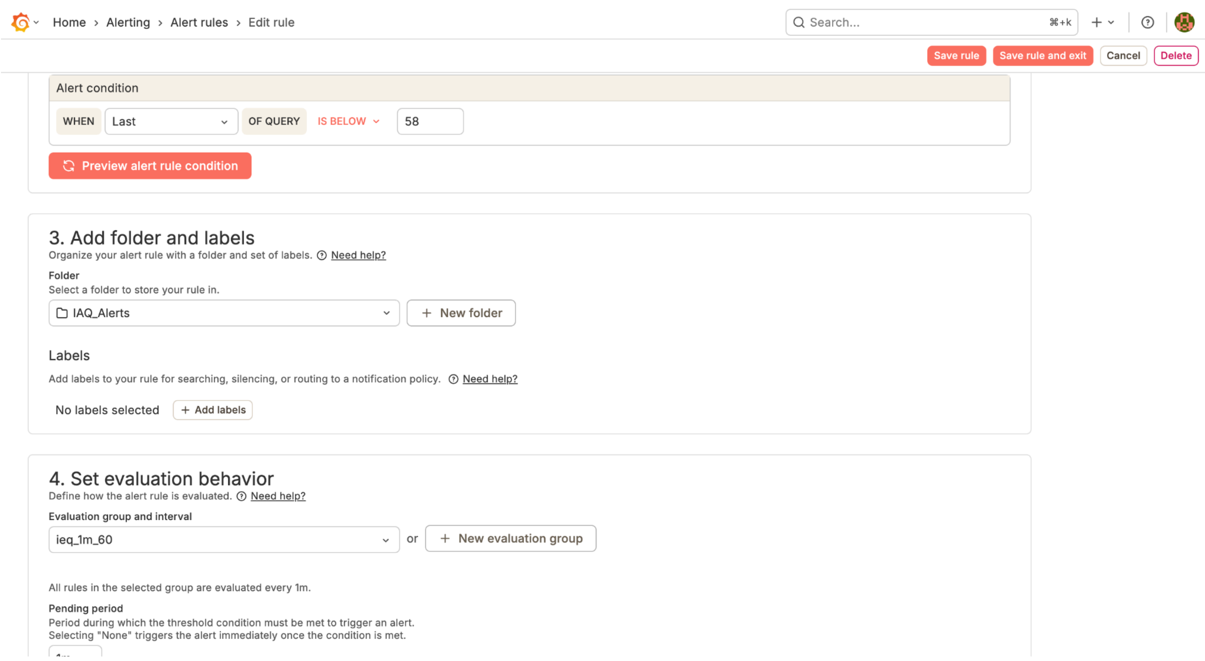

Figure 8: Set up The Alert Rules and Folder to Store Alerts

Results

- The end-to-end pipeline was fully operational under a local containerized environment
- Grafana correctly visualized all metrics and IAQ scores

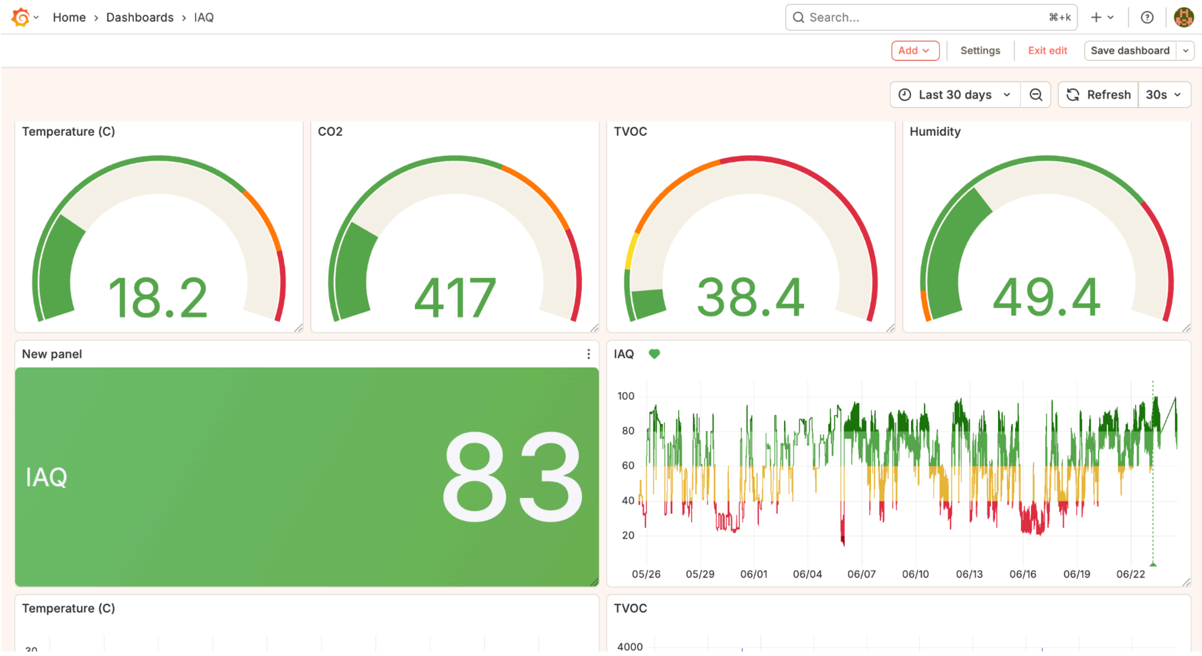

Figure 9: Final Dashboard

- Alert conditions (IAQ < 60 for 5 mins) triggered as expected via email

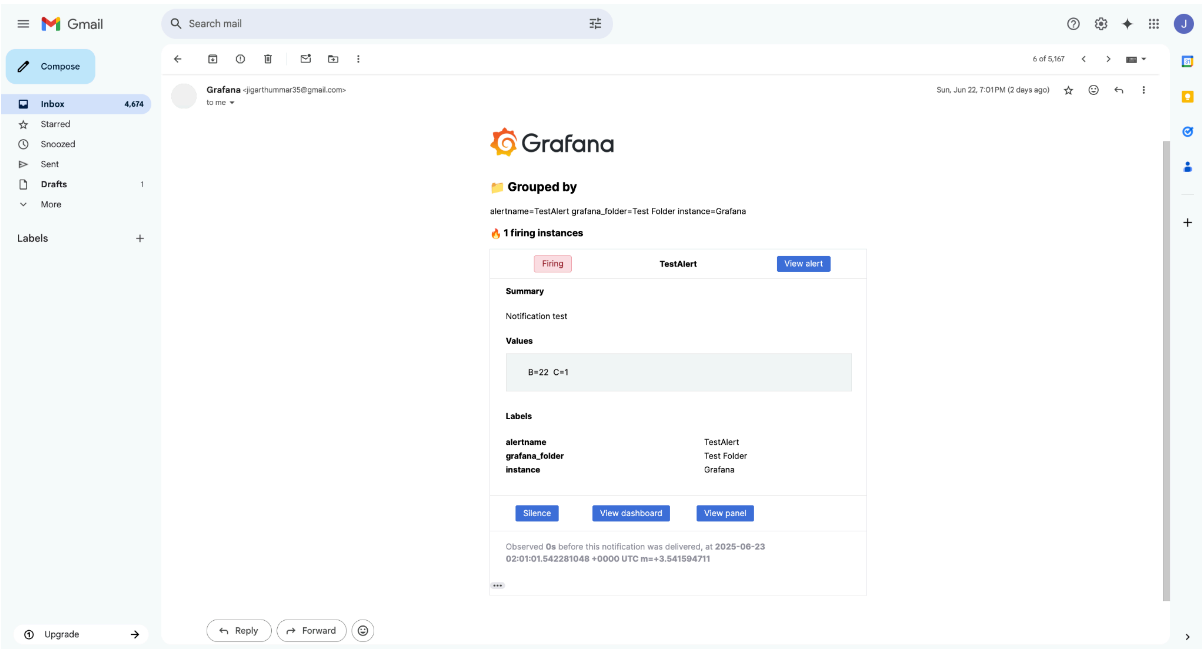

Figure 10: Email Alerts

Conclusion

This IAQ monitoring dashboard demonstrates a working proof-of-concept for capturing, scoring, storing, and visualizing indoor environmental metrics. Using modern cloud-native tools like TimescaleDB and Grafana ensures the system is extensible, performant, and ready for real-world deployment in smart buildings and classrooms.

References

Grafana Labs. (n.d.). *Grafana open source analytics & monitoring solution*. Grafana. https://grafana.com/oss/

TigerData. (n.d.). *Install TimescaleDB on Docker*. In *Self-hosted TimescaleDB documentation*. https://docs.tigerdata.com/self-hosted/latest/install/installation-docker/

Breeze Technologies. (2019). *Calculating an actionable indoor air quality index*. Breeze Technologies Blog. https://www.breeze-technologies.de/blog/calculating-an-actionable-indoor-air-quality-index/

Appendix

Appendix A: Python Code

```
# /// script
# dependencies = [
#   "requests",
#   "dotenv",
#   "psycopg2",
#   "datetime",
# ]
# ///

import os
import time
import datetime as dt
from datetime import timezone
import requests
import psycopg2
from psycopg2.extras import execute_values
from dotenv import load_dotenv
from iaq_formula import iaq_score         # ← NEW: import your IAQ
function

# ── config ─────────────────────────────────────────────────
DEVICE_ID   = os.getenv("DEVICE_ID",
"e8ff8480-eccb-11ee-a39c-0f270afb2199")
API_HOST    = os.getenv("API_HOST",   "http://52.184.14.252:8080")
POLL_SEC    = int(os.getenv("POLL_SEC", 5))
BATCH_SIZE  = int(os.getenv("BATCH_SIZE", 1))

DB_DSN = {
   "host":     os.getenv("PG_HOST", "localhost"),
   "port":     int(os.getenv("PG_PORT", 5432)),
```

```
   "dbname":   os.getenv("PG_DB",   "iaq_demo"),   # ← CHANGED
default DB
   "user":     os.getenv("PG_USER", "postgres"),
   "password": os.getenv("PG_PW",   "password"),
}

# ── helpers ──────────────────────────────────────────────
def tb_login() -> str:
   body = {"username": os.getenv("VIZHUB_LOGIN"),
           "password": os.getenv("VIZHUB_PW")}
   r = requests.post(f"{API_HOST}/api/auth/login", json=body,
timeout=10)
   r.raise_for_status()
   return r.json()["token"]

def fetch_reading(token: str) -> dict:
   url =
f"{API_HOST}/api/plugins/telemetry/DEVICE/{DEVICE_ID}/values/times
eries"
   params  = {"keys": "CO2,Temperature,Humidity,TVOC", "limit": 1}
   headers = {"X-Authorization": f"Bearer {token}"}
   r = requests.get(url, params=params, headers=headers,
timeout=10)
   r.raise_for_status()
   j = r.json()
   ts_ms = next(iter(j.values()))[0]["ts"]

   co2  = float(j["CO2"][0]["value"])
   temp = float(j["Temperature"][0]["value"])
   rh   = float(j["Humidity"][0]["value"])
   tvoc = float(j["TVOC"][0]["value"])
   iaq  = iaq_score(temp, rh, co2, tvoc)[0]        # ← NEW:
compute IAQ
   return {
       "ts":   dt.datetime.utcfromtimestamp(ts_ms / 1000),
       "device": DEVICE_ID,
       "co2":   co2,
       "temp":  temp,
       "rh":    rh,
       "tvoc":  tvoc,
       "iaq":   iaq,                               # ← include IAQ
field
   }

def insert_rows(conn, rows):
   sql = """
       INSERT INTO iaq_raw
```

```
          (time, device_id, co2_ppm, temp_c, rh_pct, tvoc_ppb,
iaq_score)
        VALUES %s
        ON CONFLICT DO NOTHING
    """
    tuples = [
        (r["ts"], r["device"], r["co2"], r["temp"],
         r["rh"], r["tvoc"], r["iaq"])              # ← IAQ value
inserted
        for r in rows
    ]
    with conn.cursor() as cur:
        execute_values(cur, sql, tuples)
    conn.commit()

# ── main loop ───────────────────────────────────────────────
def main():
    load_dotenv()
    token      = tb_login()
    token_time = time.time()

    conn = psycopg2.connect(**DB_DSN)
    buffer = []

    while True:
        try:
            if time.time() - token_time > 3600:
                token, token_time = tb_login(), time.time()

            reading = fetch_reading(token)
            buffer.append(reading)
            print(f"[{reading['ts']:%H:%M:%S}] "
                  f"IAQ={reading['iaq']} | CO₂={reading['co2']} ppm
| "
                  f"T={reading['temp']}°C | RH={reading['rh']} % |
"
                  f"TVOC={reading['tvoc']} ppb")

            if len(buffer) >= BATCH_SIZE:
                insert_rows(conn, buffer)
                buffer.clear()

        except Exception as exc:
            print("⚠️ ", exc)
            time.sleep(10)

        time.sleep(POLL_SEC)
```

```
if __name__ == "__main__":
   main()
```

Appendix B: IAQ Calculation Code

```
def iaq_score(temp_c, rh_pct, co2_ppm, tvoc_ppb):
   # ------------ 1. CO₂ & TVOC sub-index ----------------
   def band(value, limits):                # limits = [upper1,
upper2, …]
       for i, lim in enumerate(limits, start=1):
           if value <= lim:
               return i
       return len(limits) + 1              # worst case

   co2_idx  = band(co2_ppm,  [600, 1000, 1500, 2000, 5000])
   tvoc_idx = band(tvoc_ppb, [50, 100, 150, 200, 300])

   # ------------ 2. Temp/Humidity sub-index --------------
   HUM = [10,20,30,40,50,60,70,80,90]
   TMP = [16,17,18,19,20,21,22,23,24,25,26,27,28]
   GRID = [  # 9 × 13 matrix (rows = HUM, cols = TMP)
       [6,6,6,6,6,6,6,6,6,6,6,6,6],  # 10 %
       [6,5,5,5,5,5,5,5,5,5,5,5,6],  # 20 %
       [6,5,5,4,4,4,4,4,4,4,4,5,6],  # 30 %
       [6,5,5,4,3,3,3,3,2,2,3,5,6],  # 40 %
       [5,5,4,3,2,1,1,1,1,1,1,5,6],  # 50 %
       [5,4,3,2,1,1,1,1,1,2,3,5,6],  # 60 %
       [5,4,3,2,1,1,1,1,2,3,4,5,6],  # 70 %
       [5,4,2,2,2,2,2,3,4,5,5,5,6],  # 80 %
       [6,5,4,3,3,3,3,4,5,5,6,6,6],  # 90 %
   ]

   # locate the surrounding box
   import bisect, numpy as np
   r = min(len(HUM)-2, bisect.bisect_left(HUM, rh_pct)-1)
   c = min(len(TMP)-2, bisect.bisect_left(TMP, temp_c)-1)

   # relative positions 0-1
   t = (rh_pct - HUM[r]) / (HUM[r+1]-HUM[r])
   u = (temp_c - TMP[c]) / (TMP[c+1]-TMP[c])

   # bilinear interpolation
   th_idx = (
       (1-t)*(1-u)*GRID[r][c]      + (1-t)*u*GRID[r][c+1] +
          t *(1-u)*GRID[r+1][c]    +  t *u*GRID[r+1][c+1]
   )

   # ----------- 3. Sub-scores & final score ------------
```

```
def to_subscore(idx): return 120 - 20*idx

sub = {
    'CO2' : to_subscore(co2_idx),
    'TVOC': to_subscore(tvoc_idx),
    'THI' : to_subscore(th_idx)
}

iaq = round(0.4*sub['CO2'] + 0.3*sub['TVOC'] + 0.3*sub['THI'])
iaq = max(0, min(100, iaq))
return iaq, sub
```