IoT Data Monitoring System Documentation

Aneeq Hassan

Project Purpose:

- My goal is to build a small application that simulates monitoring temperature and humidity data from multiple IoT sensors in real time.
- This project will give me experience with TimescaleDB, which is known to be the stronger postgres. It will store time-series data and I'll see how it's better than vanilla.
- I'll be using a typescript backend for the API, and React (D3.js or chart.js) for frontend.

Key Features

1. Sensor data ingestion: This project simulates IoT sensors by sending temperature and humidity data with their timestamps to the backend
2. Efficient time-series storage: TimescaleDB will allow me to store and query time-series data efficiently via hypertables (storing chunks of tables), compression (to save storage space) and continuous aggregates.
3. Querying Historical Data: The API will fetch both aggregated data (e.g. average temp over a time frame) and anomalies (e.g. a spike or missing data)
4. TypeScipt-Postgres compatibility: Zod will be used for runtime schema validation and pgtyped for build-time schema compatibility.

```
WITH per_hour AS (
SELECT
time,
value
FROM kwh_hour_by_hour
WHERE "time" at time zone 'Europe/Berlin' > date_trunc('month', time) - interval '1 year'
ORDER BY 1
), hourly AS (
 SELECT
      extract(HOUR FROM time) * interval '1 hour' as hour,
      value
 FROM per_hour
)
SELECT
    hour,
    approx_percentile(0.50, percentile_agg(value)) as median,
    max(value) as maximum
FROM hourly
GROUP BY 1
ORDER BY 1;
```

SQL Table

```
CREATE TABLE sensor_data (
   time TIMESTAMPTZ NOT NULL,
   sensor_id INT NOT NULL,
   temperature DOUBLE PRECISION,
   humidity DOUBLE PRECISION
);
SELECT create_hypertable('sensor_data', by_range('time'))
```

Queries:

Look up counts of all sensor ID’s:

```
SELECT sensor_id,  COUNT(*) FROM sensor_data GROUP BY sensor_id ORDER BY
sensor_id ASC;
```

Average Temperature by Hour:

```
SELECT time_bucket('1 hour', time) AS bucket,
      AVG(temperature) AS avg_temp
FROM sensor_data
GROUP BY bucket
ORDER BY bucket;
```

Average Temperature by Day:

```
SELECT time_bucket('1 day, time) AS bucket,
      AVG(temperature) AS avg_temp
FROM sensor_data
GROUP BY bucket
ORDER BY bucket;
```

Sensor activity over time:

```
SELECT time_bucket('1 day', time) AS day,
      sensor_id,
      COUNT(*) AS readings
FROM sensor_data
GROUP BY day, sensor_id
ORDER BY day, sensor_id;
```

Min/Max Temp Over time:

```
SELECT time_bucket('1 hour', time) AS bucket,
      MIN(temperature) AS min_temp,
      MAX(temperature) AS max_temp
FROM sensor_data
GROUP BY bucket
ORDER BY bucket;
```

Continuous Aggregates

Hourly Average Temps:

```
CREATE MATERIALIZED VIEW avg_temperature_hourly
WITH (timescaledb.continuous) AS
SELECT time_bucket('1 hour', time) AS bucket,
       sensor_id,
       AVG(temperature) AS avg_temp
FROM sensor_data
GROUP BY bucket, sensor_id;
```

Get general average across all sensors per hour:

```
SELECT bucket,
      AVG(avg_temp) AS overall_avg_temp
FROM avg_temperature_hourly
WHERE bucket >= '2024-01-01' AND bucket <= '2024-01-02'
GROUP BY bucket
ORDER BY bucket ASC;
```

Sensor Readings Hourly:

```
CREATE MATERIALIZED VIEW sensor_readings_hourly
WITH (timescaledb.continuous) AS
SELECT
  time_bucket('1 hour', time) as bucket,
  sensor_id,
  AVG(temperature) as avg_temperature,
  AVG(humidity) as avg_humidity
FROM sensor_data
GROUP BY bucket, sensor_id;
```

CURL Commands for Endpoints

Hourly Averages for All Sensors

Endpoint:

```
GET /api/hourly-averages?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00
```

cURL Command:

```
curl -X GET
"http://localhost:3000/api/hourly-averages?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00"
```

Hourly Average for a specific Sensor

Endpoint:

```
GET
/api/hourly-averages?sensor_id=1&start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00
```

cURL Command:

```
curl -X GET
"http://localhost:3000/api/hourly-averages?sensor_id=1&start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00"
```

Sensor Data for Specific Sensor

Endpoint:

```
GET /api/sensors/1?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00
```

cURL Command:

```
curl -X GET
"http://localhost:3000/api/sensors/1?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00"
```

Anomalies

Endpoint:

```
GET /api/sensors/1?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00
```

cURL Command:

```
curl -X GET
"http://localhost:3000/api/sensors/1?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00"
```

General Hourly Averages Across All Sensors (Continuous Aggregate)

Endpoint:

```
GET
/api/aggregates/hourly-average?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00
```

cURL Command:

```
curl -X GET
"http://localhost:3000/api/aggregates/hourly-average?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00"
```

Hourly Average for Each Sensor (Continuous Aggregate)

Endpoint:

```
GET
/api/aggregates/hourly-average/sensors?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00
```

cURL Command:

```
curl -X GET
"http://localhost:3000/api/aggregates/hourly-average/sensors?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00"
```

Hourly Sensor Readings (Continuous Aggregate)

Endpoint:

```
GET
/api/aggregates/sensor-readings/hourly?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00
```

cURL Command:

```
curl -X GET
"http://localhost:3000/api/aggregates/sensor-readings/hourly?start_time=2024-01-01T00:00:00&end_time=2024-01-02T00:00:00"
```

Policies (Automating the Continuous Aggregrates):

Hourly Average Temps:

```
SELECT add_continuous_aggregate_policy('avg_temperature_hourly',
  start_offset => NULL,
  end_offset => INTERVAL '1 hour',
  schedule_interval => INTERVAL '1 hour');
```

Hourly Sensor Readings:

```
SELECT add_continuous_aggregate_policy('sensor_readings_hourly',
  start_offset => INTERVAL '1 day',
  end_offset => INTERVAL '1 hour',
  schedule_interval => INTERVAL '1 hour');
```