Module 10: IoT Monitoring System with TimescaleDB & Grafana

Build a complete real-time IoT monitoring system. You'll create simulated sensors, store data in TimescaleDB, and visualize everything in Grafana with live dashboards and alerts.

What You'll Build

- **IoT sensors** sending temperature data every 2 seconds
- **TimescaleDB** storing time-series data efficiently
- **Grafana dashboard** with live charts and alerts
- **Alert system** for temperature thresholds

Prerequisites

- Docker and Docker Compose installed
- Python 3.8+ with pip
- Web browser

Step 1: Project Setup

Create Project Structure

```
# Create project directory
mkdir iot-monitoring
cd iot-monitoring

# Create subdirectories
mkdir sensors grafana-config init-scripts
```

Create Docker Configuration

Create `docker-compose.yml`:

```
services:
  timescaledb:
    image: timescale/timescaledb:latest-pg16
    container_name: iot-timescaledb
    environment:
      - POSTGRES_USER=admin
      - POSTGRES_PASSWORD=password123
      - POSTGRES_DB=iot_monitoring
    ports:
      - "5432:5432"
    volumes:
      - ./init-scripts:/docker-entrypoint-initdb.d
      - timescale_data:/var/lib/postgresql/data

  grafana:
    image: grafana/grafana:latest
    container_name: iot-grafana
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin123
    ports:
      - "3000:3000"
    volumes:
      - ./grafana-config:/etc/grafana/provisioning
      - grafana_data:/var/lib/grafana
    depends_on:
      - timescaledb

volumes:
  timescale_data:
  grafana_data:
```

Step 2: Database Setup

Create Database Schema

Create `init-scripts/01-setup.sql`:

```
-- Enable TimescaleDB extension
CREATE EXTENSION IF NOT EXISTS timescaledb;

-- Sensor data table
CREATE TABLE sensor_readings (
    time TIMESTAMPTZ NOT NULL,
    device_id TEXT NOT NULL,
    location TEXT NOT NULL,
    temperature DOUBLE PRECISION NOT NULL,
    humidity INTEGER,
    battery_level INTEGER DEFAULT 100
);

-- Convert to hypertable for time-series optimization
SELECT create_hypertable('sensor_readings', 'time');

-- Create index for better query performance
CREATE INDEX idx_sensor_device_time ON sensor_readings (device_id, time DESC);

-- Alert table
CREATE TABLE temperature_alerts (
   id SERIAL PRIMARY KEY,
   device_id TEXT NOT NULL,
   temperature DOUBLE PRECISION NOT NULL,
   threshold_exceeded DOUBLE PRECISION NOT NULL,
   alert_time TIMESTAMPTZ DEFAULT NOW(),
   message TEXT
);

-- Sample devices
INSERT INTO sensor_readings (time, device_id, location, temperature, humidity) VALUES
   (NOW() - INTERVAL '1 hour', 'SENSOR_001', 'Office', 22.5, 45),
   (NOW() - INTERVAL '1 hour', 'SENSOR_002', 'Server Room', 24.8, 35),
   (NOW() - INTERVAL '1 hour', 'SENSOR_003', 'Warehouse', 19.2, 60);

-- Function to check temperature alerts
CREATE OR REPLACE FUNCTION check_temperature_alert()
RETURNS TRIGGER AS $$
BEGIN
   -- Alert if temperature > 30°C
   IF NEW.temperature > 30.0 THEN
      INSERT INTO temperature_alerts (device_id, temperature, threshold_exceeded, message)
      VALUES (NEW.device_id, NEW.temperature, 30.0,
            format('High temperature alert: %.1f°C at %s', NEW.temperature, NEW.location));
   END IF;

   -- Alert if temperature < 15°C
   IF NEW.temperature < 15.0 THEN
      INSERT INTO temperature_alerts (device_id, temperature, threshold_exceeded, message)
      VALUES (NEW.device_id, NEW.temperature, 15.0,
            format('Low temperature alert: %.1f°C at %s', NEW.temperature, NEW.location));
   END IF;

   RETURN NEW;
END;
```

```
$$ LANGUAGE plpgsql;

-- Create trigger for automatic alerts
CREATE TRIGGER temperature_alert_trigger
    AFTER INSERT ON sensor_readings
    FOR EACH ROW
    EXECUTE FUNCTION check_temperature_alert();

-- Create continuous aggregate for hourly averages
CREATE MATERIALIZED VIEW hourly_averages
WITH (timescaledb.continuous) AS
SELECT
    time_bucket('1 hour', time) AS bucket,
    device_id,
    location,
    AVG(temperature) as avg_temperature,
    MIN(temperature) as min_temperature,
    MAX(temperature) as max_temperature,
    COUNT(*) as reading_count
FROM sensor_readings
GROUP BY bucket, device_id, location;

-- Simple auto-refresh policy (refresh every 30 minutes)
SELECT add_continuous_aggregate_policy('hourly_averages',
    start_offset => INTERVAL '1 day',
    end_offset => INTERVAL '1 hour',
    schedule_interval => INTERVAL '30 minutes');
```

Step 3: Start Database

```
# Start services
docker compose up -d

# Wait for initialization
sleep 20

# Verify services are running
docker compose ps

# Test database connection
docker exec iot-timescaledb psql -U admin -d iot_monitoring -c "SELECT COUNT(*) FROM
sensor_readings;"
```

Expected output: Should show "3" (sample records).

Step 4: IoT Sensor Simulation

Install Python Dependencies

```
# Create virtual environment
python3 -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install requirements
pip install psycopg2-binary requests
```

Create IoT Simulator

Create `sensors/iot_simulator.py`:

```
#!/usr/bin/env python3
import psycopg2
import time
import random
from datetime import datetime
import threading

class IoTSensor:
    def __init__(self, device_id, location, base_temp=22.0):
        self.device_id = device_id
        self.location = location
        self.base_temp = base_temp
        self.current_temp = base_temp
        self.battery = random.randint(80, 100)
        self.running = True

    def generate_temperature(self):
        # Realistic temperature changes
        change = random.uniform(-0.5, 0.5)
        self.current_temp += change

        # Keep within reasonable bounds
        self.current_temp = max(10, min(40, self.current_temp))

        # Occasional battery drain
        if random.random() < 0.01:  # 1% chance
```

```
            self.battery = max(0, self.battery - 1)

        return round(self.current_temp, 1)

    def get_reading(self):
        return {
            'device_id': self.device_id,
            'location': self.location,
            'temperature': self.generate_temperature(),
            'humidity': random.randint(30, 70),
            'battery_level': self.battery,
            'timestamp': datetime.now()
        }

class IoTSimulator:
    def __init__(self):
        self.db_config = {
            'host': 'localhost',
            'port': 5432,
            'database': 'iot_monitoring',
            'user': 'admin',
            'password': 'password123'
        }

        # Create sensors
        self.sensors = [
            IoTSensor('SENSOR_001', 'Office', 22.0),
            IoTSensor('SENSOR_002', 'Server Room', 26.0),
            IoTSensor('SENSOR_003', 'Warehouse', 18.0),
            IoTSensor('SENSOR_004', 'Kitchen', 24.0),
        ]

        self.running = True
        self.total_readings = 0

    def connect_db(self):
        return psycopg2.connect(**self.db_config)

    def insert_reading(self, reading):
        try:
            conn = self.connect_db()
            cursor = conn.cursor()

            cursor.execute("""
                INSERT INTO sensor_readings (time, device_id, location, temperature, humidity,
battery_level)
                VALUES (%s, %s, %s, %s, %s, %s)
            """, (
                reading['timestamp'],
                reading['device_id'],
                reading['location'],
                reading['temperature'],
                reading['humidity'],
                reading['battery_level']
            ))

            conn.commit()
            cursor.close()
            conn.close()

            self.total_readings += 1
            return True

        except Exception as e:
            print(f"Database error: {e}")
            return False

    def simulate_sensor(self, sensor):
        while self.running:
            reading = sensor.get_reading()
            success = self.insert_reading(reading)

            if success:
                print(f"{sensor.device_id}: {reading['temperature']}°C at {sensor.location}")

            time.sleep(2)  # Send data every 2 seconds

    def start_simulation(self):
        print("Starting IoT Sensor Simulation")
        print(f"Monitoring {len(self.sensors)} sensors...")

        # Start threads for each sensor
        threads = []
        for sensor in self.sensors:
            thread = threading.Thread(target=self.simulate_sensor, args=(sensor,))
            thread.daemon = True
            thread.start()
            threads.append(thread)
        try:
            while True:
                time.sleep(10)
                print(f"Total readings sent: {self.total_readings}")
        except KeyboardInterrupt:
            print("\nStopping simulation...")
            self.running = False

if __name__ == "__main__":
    simulator = IoTSimulator()
    simulator.start_simulation()
```

Test the Simulator

```
# Run the simulator (let it run in background)
python sensors/iot_simulator.py
```

You should see output like:

```
Starting IoT Sensor Simulation
Monitoring 4 sensors...
SENSOR_001: 22.3°C at Office
SENSOR_002: 26.8°C at Server Room
SENSOR_003: 17.5°C at Warehouse
SENSOR_004: 24.1°C at Kitchen
```

Keep this running and open a new terminal for the next steps.

Step 5: Access Grafana

Open Grafana Interface

1. Open browser: `http://localhost:3000`
2. Login: username `admin`, password `admin123`

Create Datasource

The auto-provisioned datasource might not work correctly. Let's create a new one:

1. Go to **Connections** → **Data sources** (in left sidebar)

2. Click **"Add data source"**
3. Select **"PostgreSQL"**
4. Fill in these settings:
   - **Name**: `TimescaleDB-IoT`
   - **Host**: `timescaledb:5432` (important: use container name, not localhost)
   - **Database**: `iot_monitoring`
   - **User**: `admin`
   - **Password**: `password123`
   - **SSL Mode**: `disable`
5. Click **"Save & test"** - you should see "Database Connection OK"

Step 6: Create Live Dashboard

Create New Dashboard

1. Click **Dashboards** → **New Dashboard**
2. Click **Add visualization**

Panel 1: Live Temperature Chart

1. Select **TimescaleDB-IoT** datasource (the one we just created)
2. Make sure **Time series** is selected as visualization type
3. Switch to **Code** mode (if not already)
4. Enter this query:

```
SELECT
  time as "time",
  temperature,
  device_id || ' (' || location || ')' as metric
FROM sensor_readings
WHERE $__timeFilter(time)
ORDER BY time
```

5. Click **"Run query"** - you should see temperature lines appear
6. Set **Panel title**: "Live Temperature Readings"
7. **Important**: Change time range to "Last 5 minutes" for recent data
8. Click **"Save"** (top right) and name the dashboard "IoT Monitoring"

Panel 2: Device Status

1. Click **Add** → **Visualization** (or the "+" icon)
2. Select **Table** visualization

3. Select **TimescaleDB-IoT** datasource
4. Enter this query:

```
SELECT
  device_id as "Device",
  location as "Location",
  temperature as "Temperature (°C)",
  humidity as "Humidity (%)",
  battery_level as "Battery (%)",
  time as "Last Updated"
FROM sensor_readings s1
WHERE s1.time = (
  SELECT MAX(time)
  FROM sensor_readings s2
  WHERE s2.device_id = s1.device_id
)
ORDER BY device_id
```

5. Click **"Run query"**
6. Set **Panel title**: "Current Device Status"
7. Click **"Save"**

Panel 3: Temperature Alerts

1. Click **Add** → **Visualization**
2. Select **Table** visualization
3. Select **TimescaleDB-IoT** datasource
4. Enter this query:

```
SELECT
  device_id as "Device",
  temperature as "Temperature (°C)",
  threshold_exceeded as "Threshold",
  message as "Alert Message",
  alert_time as "Alert Time"
FROM temperature_alerts
WHERE alert_time >= NOW() - INTERVAL '1 hour'
ORDER BY alert_time DESC
LIMIT 10
```

5. Click **"Run query"**
6. Set **Panel title**: "Recent Alerts"
7. Click **"Save"**

Configure Dashboard Settings

1. Click the **Settings** gear icon (top right)
2. Set **Auto-refresh**: 5s (updates every 5 seconds)
3. Set **Time range**: Last 30 minutes
4. Click **"Save"**

Step 7: Test Alerts

Run this SQL command to fix the trigger function:

```
docker exec iot-timescaledb psql -U admin -d iot_monitoring -c "
DROP FUNCTION IF EXISTS check_temperature_alert() CASCADE;

CREATE OR REPLACE FUNCTION check_temperature_alert()
RETURNS TRIGGER AS \$\$
BEGIN
  -- Alert if temperature > 30°C
  IF NEW.temperature > 30.0 THEN
    INSERT INTO temperature_alerts (device_id, temperature, threshold_exceeded, message)
    VALUES (NEW.device_id, NEW.temperature, 30.0,
        'High temperature alert: ' || NEW.temperature || '°C at ' || NEW.location);
  END IF;

  -- Alert if temperature < 15°C
  IF NEW.temperature < 15.0 THEN
    INSERT INTO temperature_alerts (device_id, temperature, threshold_exceeded, message)
    VALUES (NEW.device_id, NEW.temperature, 15.0,
        'Low temperature alert: ' || NEW.temperature || '°C at ' || NEW.location);
  END IF;

  RETURN NEW;
END;
\$\$ LANGUAGE plpgsql;

CREATE TRIGGER temperature_alert_trigger
  AFTER INSERT ON sensor_readings
  FOR EACH ROW
  EXECUTE FUNCTION check_temperature_alert();
"
```

Trigger Test Alerts

Create `sensors/trigger_alerts.py`:

```
#!/usr/bin/env python3
import psycopg2
from datetime import datetime


def trigger_alerts():
    db_config = {
        "host": "localhost",
        "port": 5432,
        "database": "iot_monitoring",
        "user": "admin",
        "password": "password123",
    }

    conn = psycopg2.connect(**db_config)
    cursor = conn.cursor()

    print("Triggering test alerts...")

    # Insert high temperature reading (should trigger alert > 30°C)
    cursor.execute(
        """
        INSERT INTO sensor_readings (time, device_id, location, temperature, humidity,
battery_level)
        VALUES (%s, 'SENSOR_001', 'Office', %s, 65, 85)
    """,
        (datetime.now(), 35.5),
    )  # Above 30°C threshold

    # Insert low temperature reading (should trigger alert < 15°C)
    cursor.execute(
        """
        INSERT INTO sensor_readings (time, device_id, location, temperature, humidity,
battery_level)
        VALUES (%s, 'SENSOR_003', 'Warehouse', %s, 45, 75)
    """,
        (datetime.now(), 12.8),
    )  # Below 15°C threshold

    conn.commit()
    # Check if alerts were created
    cursor.execute(
        "SELECT COUNT(*) FROM temperature_alerts WHERE alert_time >= NOW() - INTERVAL
'1 minute'"
    )
    alert_count = cursor.fetchone()[0]

    print(f"✅ Inserted extreme temperature readings")
    print(f"✅ {alert_count} alerts should be generated")

    # Display the alerts
    cursor.execute(
        """
        SELECT device_id, temperature, threshold_exceeded, message, alert_time
        FROM temperature_alerts
        WHERE alert_time >= NOW() - INTERVAL '1 minute'
        ORDER BY alert_time DESC
    """
    )

    alerts = cursor.fetchall()
    if alerts:
        print("\n📢 Generated alerts:")
        for alert in alerts:
            device, temp, threshold, message, time = alert
            print(f"  {device}: {message}")

    cursor.close()
    conn.close()


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

Run Alert Test

```
python sensors/trigger_alerts.py
```

Verify in Grafana

1. Go back to your Grafana dashboard
2. Check the "Recent Alerts" panel

3. You should see 2 new alerts

Step 8: Final Testing

Dashboard Settings

1. Set **Auto-refresh**: 5s (updates every 5 seconds)
2. Set **Time range**: Last 1 hour
3. Click **Save**

Verify Everything Works

Your dashboard should now show:

- **Live temperature chart** updating every 5 seconds
- **Device status table** with current readings
- **Recent alerts** from the test
- All data refreshing automatically

Cleanup

When finished:

```
# Stop IoT simulator
# Press Ctrl+C in the simulator terminal

# Deactivate virtual environment
deactivate

# Remove everything (optional)
docker compose down -v

# Check that containers are stopped
docker compose ps

# Remove Docker images to free up space
docker rmi timescale/timescaledb:latest-pg16
docker rmi grafana/grafana:latest

# Remove project directory
cd ..
rm -rf iot-monitoring
```

```
# Clean up any unused Docker resources
docker system prune -f
```

Congratulations!

You've built a complete IoT monitoring system with:

- **Real-time data collection** from 4 simulated sensors
- **TimescaleDB** storing time-series data efficiently
- **Live Grafana dashboard** with auto-refreshing charts
- **Automatic alerts** for temperature thresholds
- **Production-ready architecture**

What you learned:

- TimescaleDB hypertables for time-series data
- Real-time IoT data ingestion with Python
- Grafana dashboard creation and configuration
- Database triggers for automatic alerting
- Docker container orchestration

This system can easily be extended with real IoT devices, additional sensors, more complex alerts, and advanced analytics.