System monitoringu temperatury i wilgotności w LoRaWAN

Internet rzeczy i systemy autonomiczne

Autor: Dawid Pawłowski, Piotr Szczypior

Data: 30 stycznia 2026
Prowadzący:

1 Wstęp

Celem systemu jest zdalny pomiar temperatury i wilgotności oraz ich wizualizacja i nadzór progowy.
Komunikacja z urządzeniem realizowana jest w sieci LoRaWAN z wykorzystaniem TTN, a dane trafiają do bazy InfluxDB i są prezentowane w Grafanie.
Zaprojektowano ścieżkę zwrotną (downlink) do sygnalizacji alarmu na urządzeniu końcowym.
Całość uruchamiana jest lokalnie w kontenerach, z konfiguracją przekazywaną przez zmienne środowiskowe.

1.1 Zakres i cele

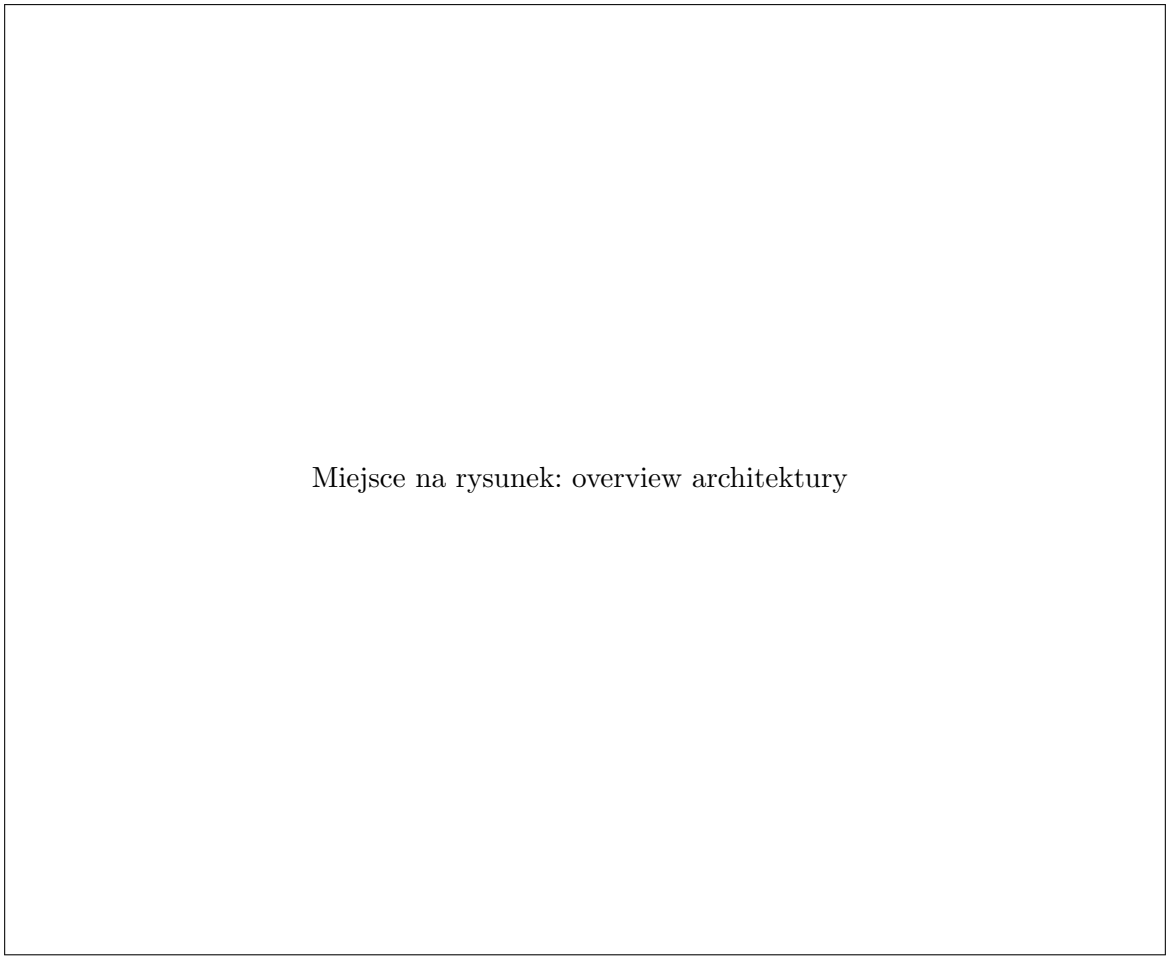
Zakres obejmuje firmware węzła LoRaWAN, odbiór uplinku po MQTT, zapis do bazy czasowej oraz wizualizację i alarmowanie.
System zapewnia jednolitą ścieżkę danych dla wielu pomiarów i umożliwia reakcję na przekroczenia progów.

1.2 Zastosowane protokoły

LoRaWAN.
MQTT (TTN).
HTTP/HTTPS (API InfluxDB oraz webhook Grafany).
TLS (szyfrowanie połączeń MQTT).

2 Architektura systemu – przegląd

Poniżej przewidziano miejsce na schemat ogólny architektury obejmujący węzeł LoRaWAN, TTN, warstwę MQTT, zapis do InfluxDB, wizualizację w Grafanie i ścieżkę downlink.



Rysunek 1: Schemat ogólny architektury systemu

Schemat powinien uwzględniać kierunki przepływu danych oraz punkty integracji pomiędzy usługami.

3 Architektura i przepływ danych

Węzeł pomiarowy (DHT11) wysyła ramki LoRaWAN metodą OTAA, a payload ma postać tekstu `T:x,H:y`.
Serwer odbiorczy subskrybuje uplink z TTN przez MQTT, parsuje dane i zapisuje je do InfluxDB przez API HTTP v2.
Grafana odczytuje dane z InfluxDB i generuje alerty, które są przekazywane webhookiem do aplikacji Flask.

3.1 Węzeł LoRaWAN

Pomiar wykonywany jest cyklicznie, a dane są kodowane do bufora tekstowego i wysyłane jako payload uplinku.

```
int len = snprintf(mydata, PAYLOAD_SIZE, "T:%d,H:%d", DHT.temperature, DHT.humidity);
LMIC_setTxData2(1, (uint8_t*)mydata, len, 1);
```

Użyte biblioteki firmware: `lmic`, `hal/hal`, `SPI`, `DFRobot_DHT11`.
Najważniejsze parametry: `TX_INTERVAL` (interwał wysyłki), `PAYLOAD_SIZE` (bufor payloadu), `DHT11_PIN` i `ALARM_LED_PIN` (piny sprzętowe), `CMD_LED_ON/OFF` (kody downlink), mapowanie `lmic_pins`.

```
#define DHT11_PIN 7
#define ALARM_LED_PIN 8
#define CMD_LED_OFF 0x00
#define CMD_LED_ON  0x01
const unsigned TX_INTERVAL = 5;
const lmic_pinmap lmic_pins = { .nss = 10, .rxtx = LMIC_UNUSED_PIN, .rst = LMIC_UNUSED_PIN,
```

Obsługa downlink realizuje proste sterowanie diodą alarmową.

```
uint8_t cmd = data[0];
if (cmd == CMD_LED_ON) {
    digitalWrite(ALARM_LED_PIN, HIGH);
} else if (cmd == CMD_LED_OFF) {
    digitalWrite(ALARM_LED_PIN, LOW);
}
```

3.2 Uplink i zapis do bazy

Po odebraniu wiadomości z TTN następuje parsowanie i zapis do InfluxDB w formacie Line Protocol.

```
payload_from_device = data["uplink_message"]["decoded_payload"]["text"]
temp, hum = parse_data(payload_from_device)
line_protocol = f"measurements_ttn,device={device_id} T={temp},H={hum}"
requests.post(url, data=line_protocol, headers=headers)
```

Warstwa zapisu korzysta z tokenu administracyjnego oraz bucketu wskazanego w zmiennych środowiskowych.

```
INFLUX_HOST = os.getenv("INFLUX_HOST", "http://localhost:8086")
INFLUX_TOKEN = os.getenv("INFLUXDB_INIT_ADMIN_TOKEN", "")
INFLUX_ORG = os.getenv("INFLUXDB_INIT_ORG", "pwr")
INFLUX_BUCKET = os.getenv("INFLUXDB_INIT_BUCKET", "pwr")
```

3.3 Alert i downlink

Alert Grafany trafia do webhooka, a system publikuje komendę downlink w TTN.

```
if status == "firing":
    send_downlink(CMD_LED_ON)
elif status == "resolved":
    send_downlink(CMD_LED_OFF)
```

Publikacja downlink obejmuje kodowanie payloadu do base64 oraz wysyłkę na temat TTN.

```
topic = f"v3/{APP_ID}@ttn/devices/{DEV_EUI}/down/push"
payload_bytes = bytes([command])
payload_b64 = base64.b64encode(payload_bytes).decode("ascii")
client.publish(topic, json.dumps(downlink_msg))
```

3.4 Przepływ danych – kroki

1. Odczyt DHT11 i zbudowanie payloadu.
2. Uplink przez LoRaWAN do TTN.
3. Odbiór MQTT i parsowanie przez serwer aplikacyjny.
4. Zapis punktu do InfluxDB.
5. Wizualizacja w Grafanie i generacja alertu.
6. Wywołanie webhooka i publikacja downlink do urządzenia.

4 Implementacja i konfiguracja

Moduł `main.py` inicjalizuje klienta MQTT, uruchamia pętlę subskrypcji uplinku oraz równolegle serwer Flask dla webhooków.
Moduł `uplink.py` dekoduje payload, wylicza temperaturę i wilgotność oraz zapisuje pomiar w linii `measurements_ttn,device=... T=...,H=...`.
Moduł `webhook.py` reaguje na alerty Grafany i steruje downlinkiem, a `downlink.py` publikuje komendy LED do TTN.
Środowisko uruchomieniowe oparto o `docker-compose` z usługami InfluxDB i Grafana oraz konfiguracją przez zmienne środowiskowe.

4.1 Inicjalizacja klienta MQTT

Konfiguracja połączenia z TTN wykonywana jest na starcie aplikacji.

```
mqtt_client = mqtt.Client(callback_api_version=CallbackAPIVersion.VERSION2)
mqtt_client.username_pw_set(USERNAME, KEY)
mqtt_client.tls_set()
mqtt_client.on_message = on_message
mqtt_client.connect(HOST, PORT, keepalive=60)
```

Wątek serwera Flask uruchamiany jest niezależnie od pętli MQTT.

```
flask_thread = threading.Thread(target=run_flask)
flask_thread.start()
initialize_mqtt_client()
```

4.2 Subskrypcja uplinku

Subskrypcja ogranicza się do urządzenia o wskazanym `DEV_EUI`.

```
sub = f"v3/{APP_ID}@ttn/devices/{DEV_EUI}/up"
mqtt_client.subscribe(sub)
```

4.3 Publikacja downlink

Komenda LED kodowana jest do base64 i wysyłana na temat TTN.

```
payload_bytes = bytes([command])
payload_b64 = base64.b64encode(payload_bytes).decode("ascii")
downlink_msg = {
    "downlinks": [
        {
            "f_port": 1,
            "frm_payload": payload_b64,
            "priority": "NORMAL"
        }
    ]
}
client.publish(topic, json.dumps(downlink_msg))
```

4.4 Usługi w kontenerach

InfluxDB i Grafana uruchamiane są jako osobne usługi, a konfiguracja jest przekazywana z pliku środowiskowego.

```
services:
  influxdb:
    image: influxdb:2.7
  grafana:
    image: grafana/grafana:latest
```

Istotne zmienne środowiskowe są wymagane do poprawnej konfiguracji usług i dostępu do TTN.

```
INFLUX_HOST                  - adres URL InfluxDB
INFLUXDB_INIT_ORG            - organizacja w InfluxDB
INFLUXDB_INIT_BUCKET         - bucket dla pomiarów
INFLUXDB_INIT_ADMIN_TOKEN    - token do autoryzacji zapisu
MQTT_HOST                    - adres brokera TTN
MQTT_PORT                    - port połączenia MQTT
MQTT_USERNAME                - nazwa użytkownika TTN
MQTT_KEY                     - klucz API TTN
MQTT_APP_ID                  - identyfikator aplikacji TTN
MQTT_DEV_EUI                 - identyfikator urządzenia
```

4.5 Struktura modułów

Podział na moduły upraszcza utrzymanie i wydziela odpowiedzialności.

```
main.py      - start aplikacji, MQTT, Flask
uplink.py    - odbiór i zapis do InfluxDB
downlink.py  - budowanie i wysyłka downlink
webhook.py   - obsługa alertów Grafany
main.ino     - firmware węzła LoRaWAN
```

5 Grafana – wizualizacja i alarmy

Grafana odczytuje pomiary z InfluxDB i prezentuje je na panelach czasowych.
Alerty progowe są definiowane po stronie Grafany i wysyłane webhookiem do aplikacji Flask.
Miejsce na zrzuty: dashboard, panel alertu, konfiguracja kontaktu webhook.

5.1 Webhook do aplikacji

Integracja webhook realizowana jest przez endpoint aplikacji i statusy alertu.

```python
@app.route("/webhook/alert", methods=["POST"])
def handle_grafana_alert():
    status = request.json.get("status")
    if status == "firing":
        handle_alarm_on()
    elif status == "resolved":
        handle_alarm_off()
```

6 Wnioski i materiały graficzne

Zaimplementowano pełny łańcuch od pomiaru do wizualizacji i reakcji zwrotnej poprzez downlink.
Konfiguracja systemu opiera się na TTN, MQTT, InfluxDB i Grafanie, co umożliwia szybkie uruchomienie w środowisku kontenerowym.

Struktura aplikacji rozdziela logikę odbioru uplinku, zapisu do bazy oraz obsługę webhooków, co upraszcza testowanie i utrzymanie.
Miejsca na ilustracje do uzupełnienia:
TTN – zrzut panelu urządzenia i ruchu uplink/downlink.
Grafana – panel wykresów i konfiguracja alertu.
InfluxDB – widok bucketu lub zapytania z danymi.

6.1 Dodatkowe miejsca na grafiki

Urządzenie końcowe – zdjęcie sprzętu i podłączenia czujnika.
Przepływ danych – schemat kroków od pomiaru do wizualizacji.
Logi aplikacji – przykładowe wpisy z odbioru i zapisu.

6.2 Uwagi końcowe

Rozwiązanie jest gotowe do rozbudowy o kolejne urządzenia i dodatkowe metryki, bez zmiany architektury logicznej.
Wprowadzanie progów i reguł alarmowania odbywa się po stronie Grafany, bez ingerencji w firmware.