

# System ostrzegania przed przymrozkami

---

## Etap 2: Implementacja rozwiązania

Michał Kołodziejczyk, Sebastian Chmielewski, Bartłomiej Gromulski

Opiekun projektu: dr inż. Wojciech Kazubski

15 lutego 2026

### Spis treści

<b>1</b>	<b>Telegraf</b>	<b>2</b>
1.1	Rola . . . . .	2
1.2	Konfiguracja . . . . .	2
1.3	opis działania przepływu danych . . . . .	3
1.4	Monitorowanie statusu . . . . .	4
<b>2</b>	<b>MQTT</b>	<b>4</b>
2.1	Broker . . . . .	5
2.2	Struktura danych i tematy . . . . .	5
<b>3</b>	<b>InfluxDBv2</b>	<b>6</b>
3.1	Wdrożenie . . . . .	6
3.2	Konfiguracja struktury danych (v2) . . . . .	6
3.3	Model danych . . . . .	6
<b>4</b>	<b>Grafana</b>	<b>7</b>
4.1	Konfiguracja . . . . .	7
4.2	Dashboard . . . . .	7
<b>5</b>	<b>Strona internetowa</b>	<b>9</b>
<b>6</b>	<b>Stacja pomiarowa</b>	<b>10</b>
6.1	Komponenty stacji pomiarowej . . . . .	10
6.2	Działanie stacji pomiarowej . . . . .	10
6.3	Parametry Lory . . . . .	11
6.4	Przesyłana ramka Lory . . . . .	11
6.5	Stacja odbierająca dane . . . . .	12

# 1 Telegraf

## 1.1 Rola

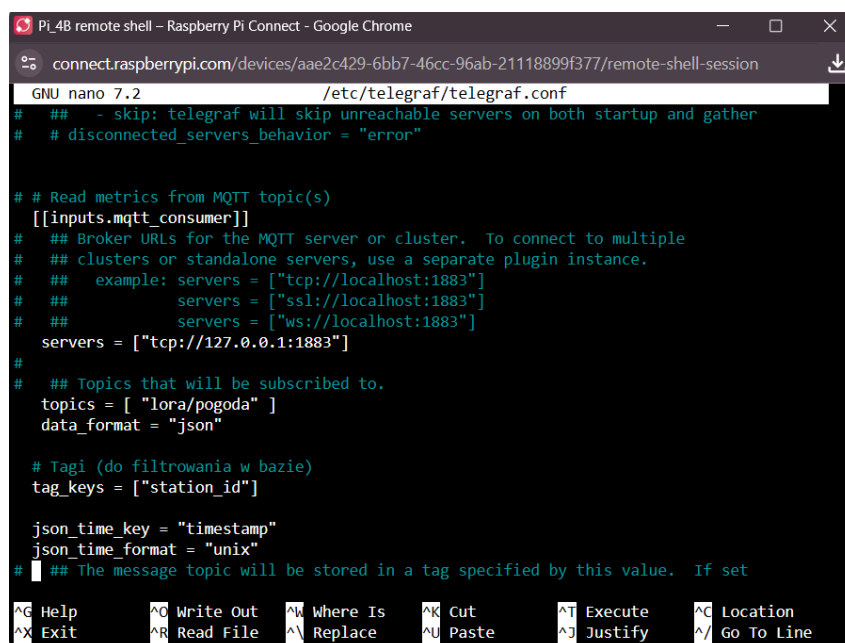
W tej fazie projektu wykorzystano narzędzie Telegraf jako pośrednika w warstwie transportu i przetwarzania danych. Jego głównym zadaniem jest nasłuchiwanie komunikatów przychodzących z brokera MQTT, parsowanie ich formatu oraz przesyłanie przetworzonych danych do bazy InfluxDB.

Użycie Telegrafa jest optymalnym rozwiązaniem, pozwoliło na wyeliminowanie konieczności pisania własnych skryptów (np. w Pythonie) do obsługi zapisu do bazy danych, co zwiększa stabilność i skalowalność systemu. Telegraf automatycznie buforuje dane w przypadku chwilowej niedostępności bazy.

## 1.2 Konfiguracja

Telegraf został skonfigurowany poprzez edycję głównego pliku konfiguracyjnego znajdującego się w ścieżce `/etc/telegraf/telegraf.conf`. W ramach konfiguracji aktywowano dwie kluczowe wtyczki (plugins):

- Input Plugin (Wejście): `mqtt_consumer` – odpowiedzialny za odbiór danych.
- Output Plugin (Wyjście): `influxdb_v2` – odpowiedzialny za zapis danych.



```
PI_4B remote shell - Raspberry Pi Connect - Google Chrome
connect.raspberrypi.com/devices/aae2c429-6bb7-46cc-96ab-21118899f377/remote-shell-session

GNU nano 7.2 /etc/telegraf/telegraf.conf
# ## - skip: telegraf will skip unreachable servers on both startup and gather
# ## disconnected_servers_behavior = "error"

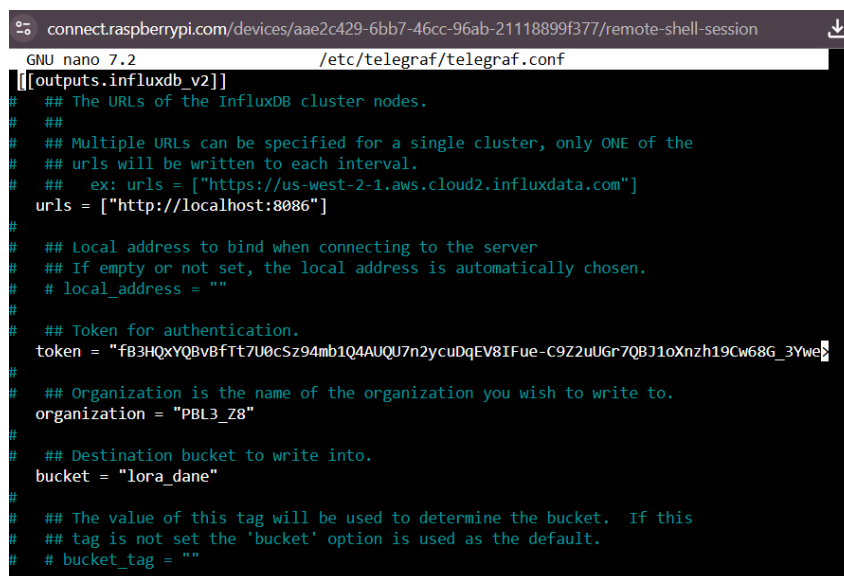
# ## Read metrics from MQTT topic(s)
[[inputs.mqtt_consumer]]
# ## Broker URLs for the MQTT server or cluster. To connect to multiple
# ## clusters or standalone servers, use a separate plugin instance.
# ## example: servers = ["tcp://localhost:1883"]
# ##          servers = ["ssl://localhost:1883"]
# ##          servers = ["ws://localhost:1883"]
# ##          servers = ["tcp://127.0.0.1:1883"]
# ##
# ## Topics that will be subscribed to.
# topics = [ "lora/pogoda" ]
# data_format = "json"

# Tagi (do filtrowania w bazie)
tag_keys = ["station_id"]

# json_time_key = "timestamp"
# json_time_format = "unix"
# ## The message topic will be stored in a tag specified by this value. If set
```

Rysunek 1: mqtt consumer

W tej sekcji zdefiniowano adres brokera, temat nasłuchiwania oraz format danych. Kluczowym ustawieniem jest `data_format = "json"`, który umożliwia automatyczne mapowanie kluczy z obiektu JSON.



```
connect.raspberrypi.com/devices/aae2c429-6bb7-46cc-96ab-21118899f377/remote-shell-session
GNU nano 7.2 /etc/telegraf/telegraf.conf
[[outputs.influxdb_v2]]
# The URLs of the InfluxDB cluster nodes.
#
# Multiple URLs can be specified for a single cluster, only ONE of the
# urls will be written to each interval.
# ex: urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
urls = ["http://localhost:8086"]
#
# Local address to bind when connecting to the server
# If empty or not set, the local address is automatically chosen.
# local_address = ""
#
# Token for authentication.
token = "fb3HQxYQBvBftt7U0cSz94mb1Q4AUQU7n2ycuDqEV8IFue-C9Z2uUGr7QBJ1oXnzH19Cw68G_3Ywe"
#
# Organization is the name of the organization you wish to write to.
organization = "PBL3_Z8"
#
# Destination bucket to write into.
bucket = "lora_dane"
#
# The value of this tag will be used to determine the bucket. If this
# tag is not set the 'bucket' option is used as the default.
# bucket_tag = ""
```

Rysunek 2: outputs dla influxdb2

W tej sekcji skonfigurowano połączenie z bazą danych, wykorzystując uwierzytelnianie oparte na tokenach (Token-based authentication), co jest standardem w InfluxDB v2.

### 1.3 opis działania przepływu danych

Po uruchomieniu usługi (`sudo systemctl start telegraf`) agent wykonuje następujące operacje w pętli:

1. Nawiązuje stałe połączenie TCP z lokalnym brokerem Mosquitto na porcie 1883.
2. Subskrybuje temat `lora_dane/sensors/#`.
3. Po odebraniu wiadomości, parsuje ją, wyodrębniając wartości liczbowe.
4. Dodaje do danych znaczniki systemowe (np. `host`) oraz precyzyjny znacznik czasu (`timestamp`).
5. Konwertuje dane do formatu **Influx Line Protocol**.
6. Wysyła dane do API InfluxDB (port 8086) w paczkach, co optymalizuje obciążenie sieci i dysku.

Poprawność działania konfiguracji została zweryfikowana poprzez analizę logów systemowych (`journalctl -u telegraf -e`), gdzie potwierdzono nawiązanie połączenia z oboma usługami.

```
pi@raspberrypi:~$ journalctl -u telegraf -e
--
--
--
Dec 19 21:14:33 raspberrypi systemd[1]: Starting telegraf.service - Telegraf...
Dec 19 21:14:40 raspberrypi systemd[1]: Started telegraf.service - Telegraf.
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z W! Strict environment variable handling will be the new default starting with v1.38.0! If your c
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z I! Loading config: /etc/telegraf/telegraf.conf
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z I! Starting Telegraf 1.37.0 brought to you by InfluxData the makers of InfluxDB
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z I! Available plugins: 243 inputs, 9 aggregators, 35 processors, 26 parsers, 67 outputs, 8 secret
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z I! Loaded inputs: cpu disk diskio kernel mem mqtt_consumer processes swap system
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z I! Loaded aggregators:
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z I! Loaded processors:
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z I! Loaded secretstores:
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z I! Loaded outputs: influxdb_v2
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z I! Tags enabled: host=rpi4-lora
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z I! [agent] Config: Interval:10s, Quiet:false, Hostname:"rpi4-lora", Flush Interval:10s
Dec 19 21:14:43 raspberrypi telegraf[1289]: 2025-12-19T20:14:43Z W! [agent] The default value of 'skip_processors_after_aggregators' will change to 'true' with T
```

Rysunek 3: dalej logi dla telegrafu

## 1.4 Monitorowanie statusu

Do sprawdzenia statusu użyto polecenia: `sudo systemctl status telegraf`

```
pi@raspberrypi:~$ sudo systemctl status telegraf
● telegraf.service - Telegraf
   Loaded: loaded (/lib/systemd/system/telegraf.service; enabled; preset: enabled)
   Active: active (running) since Thu 2025-12-18 09:30:01 CET; 1 day 11h ago
     Docs: https://github.com/influxdata/telegraf
   Main PID: 1254 (telegraf)
      Tasks: 13 (limit: 1573)
         CPU: 18min 20.946s
    CGroup: /system.slice/telegraf.service
            └─1254 /usr/bin/telegraf -config /etc/telegraf/telegraf.conf -config-directory /etc/telegraf/telegraf.d
              └─1612 /usr/bin/dbus-daemon --syslog --fork --print-pid 4 --print-address 6 --session

Dec 18 22:13:35 raspberrypi telegraf[1254]: 2025-12-18T21:13:35Z E! [agent] Error writing to outputs.influxdb_v2: Post "http://localhost:
Dec 18 22:13:48 raspberrypi telegraf[1254]: 2025-12-18T21:13:45Z E! [outputs.influxdb_v2] Post "http://localhost:8086/api/v2/write?bucket=
Dec 18 22:13:48 raspberrypi telegraf[1254]: 2025-12-18T21:13:45Z E! [outputs.influxdb_v2] When writing to [http://localhost:8086/api/v2/write?bucket=
Dec 18 22:13:48 raspberrypi telegraf[1254]: 2025-12-18T21:13:45Z E! [agent] Error writing to outputs.influxdb_v2: Post "http://localhost:8086/api/v2/write?bucket=
Dec 19 20:43:33 raspberrypi telegraf[1254]: 2025-12-19T19:43:33Z E! [outputs.influxdb_v2] Post "http://localhost:8086/api/v2/write?bucket=
Dec 19 20:43:33 raspberrypi telegraf[1254]: 2025-12-19T19:43:33Z E! [outputs.influxdb_v2] When writing to [http://localhost:8086/api/v2/write?bucket=
Dec 19 20:44:12 raspberrypi telegraf[1254]: 2025-12-19T19:44:12Z E! [agent] Error writing to outputs.influxdb_v2: Post "http://localhost:8086/api/v2/write?bucket=
Dec 19 20:44:12 raspberrypi telegraf[1254]: 2025-12-19T19:44:12Z E! [outputs.influxdb_v2] Post "http://localhost:8086/api/v2/write?bucket=
Dec 19 20:44:12 raspberrypi telegraf[1254]: 2025-12-19T19:44:12Z E! [outputs.influxdb_v2] When writing to [http://localhost:8086/api/v2/write?bucket=
Dec 19 20:44:12 raspberrypi telegraf[1254]: 2025-12-19T19:44:12Z E! [agent] Error writing to outputs.influxdb_v2: Post "http://localhost:8086/api/v2/write?bucket=
lines 1-21/21 (END)
```

Rysunek 4: dalej logi dla telegrafu

Analiza logów (Journal)

W tej części zrzutu widoczne są ostatnie logi systemowe.

- Komunikaty `Error writing to outputs.influxdb_v2` wskazują na momenty, w których nie mógł on nawiązać połączenia z bazą danych InfluxDB (np. 19 grudnia o godz. 20:44).
- Jest to zachowanie oczekiwane w fazie konfiguracji systemu – Telegraf posiada wbudowany mechanizm buforowania. Gdy baza danych jest tymczasowo niedostępna (np. podczas restartu lub prac konserwacyjnych), Telegraf przechowuje dane w pamięci RAM i ponawia próby wysyłki, nie przerywając własnego działania.

## 2 MQTT

Jako medium komunikacyjne w projekcie zastosowano protokół MQTT (Message Queuing Telemetry Transport). Jest to lekki protokół transmisji danych działający w modelu Publikuj/Subskrybuj (Publish/Subscribe)

## 2.1 Broker

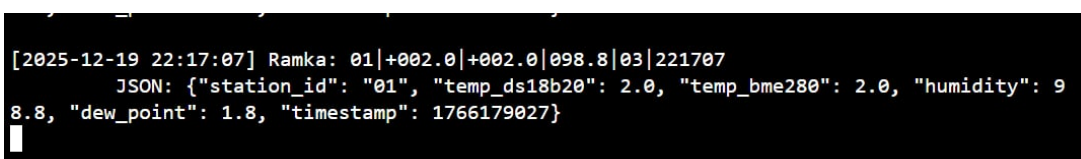
Jako broker wybrano Eclipse Mosquitto ze względu na jego stabilność, otwartość (Open Source) oraz niskie wymagania sprzętowe, co pozwala na uruchomienie go na platformie Raspberry Pi 4.

W celu umożliwienia dostępu do brokera z poziomu usługi Telegram oraz urządzeń zewnętrznych w sieci lokalnej, zmodyfikowano domyślną konfigurację w pliku konfiguracyjnym dodając zezwolenie na połączenie bez uwierzytelnienia oraz otwarcie portu na 1883 dla połączeń sieciowych.

## 2.2 Struktura danych i tematy

W projekcie przyjęto hierarchiczną strukturę tematów (topics), co pozwala na bardzo łatwe filtrowanie co będzie przydatne na przykład w grafanie.

- Temat główny: `lora/sensors/#` (znak # oznacza subskrypcję wszystkich podtematów).
- Przykładowy temat urządzenia: `lora/sensors/rpi4-lora/`.
- Payload: Dane przesyłane są w formacie JSON. Umożliwia to przesyłanie wielu parametrów w jednej wiadomości oraz jest natywnie obsługiwane przez parser Telegrama.



```
[2025-12-19 22:17:07] Ramka: 01|+002.0|+002.0|098.8|03|221707
      JSON: {"station_id": "01", "temp_ds18b20": 2.0, "temp_bme280": 2.0, "humidity": 98.8, "dew_point": 1.8, "timestamp": 1766179027}
```

Rysunek 5: Ramka danych

Na Rysunku przedstawiono log z aplikacji bramki, ukazujący moment przetworzenia surowej ramki danych na obiekt JSON gotowy do wysłania do brokera MQTT.

Interpretacja danych:

1. **station\_id:**  
Unikalny identyfikator stacji pomiarowej (węzła LoRa). Pozwala to na rozróżnienie w bazie danych, z którego urządzenia pochodzi pomiar, co umożliwia łatwe skalowanie systemu o kolejne czujniki.
2. **temp\_ds18b20:**  
Wartość temperatury [°C] odczytana z cyfrowego czujnika DS18B20 (sonda wodoodporna/zewnętrzna).
3. **temp\_bme280:**  
Wartość temperatury [°C] odczytana z czujnika środowiskowego BME280. Zdublowanie pomiaru temperatury pozwala na weryfikację poprawności wskazań (redundancja).
4. **humidity:**  
Wilgotność względna powietrza [%] odczytana z czujnika BME280.
5. **dew\_point:**  
Punkt rosy [°C] – parametr wyliczony programowo na podstawie temperatury i

wilgotności przed wysłaniem danych. Przesyłanie wartości przeliczonej zmniejsza obciążenie obliczeniowe po stronie bazy danych.

6. **timestamp:**

Znacznik czasu w formacie Unix Epoch (liczba sekund od 1 stycznia 1970). Przesyłanie czasu wygenerowanego przez stację nadawczą (lub bramkę w momencie odbioru) pozwala zachować precyzyjną chronologię zdarzeń.

## 3 InfluxDBv2

magazyn danych jaki wykorzystano do systemu jest InfluxDB w wersji 2.x. Jest to baza danych typu TSDB (Time Series Database), zaprojektowana specjalnie do obsługi szeregów czasowych – danych, w których kluczową rolę odgrywa moment wystąpienia zdarzenia, idealne dla naszego projektu.

### 3.1 Wdrożenie

Po zainstalowaniu pakietu influxdb2, usługę uruchomiono jako demona systemowego, baza danych nasłuchuje domyślnie na porcie 8086 i udostępnia interfejs graficzny dostępny przez przeglądarkę internetową. Aktualnie wszystko jest zawarte na localhost.

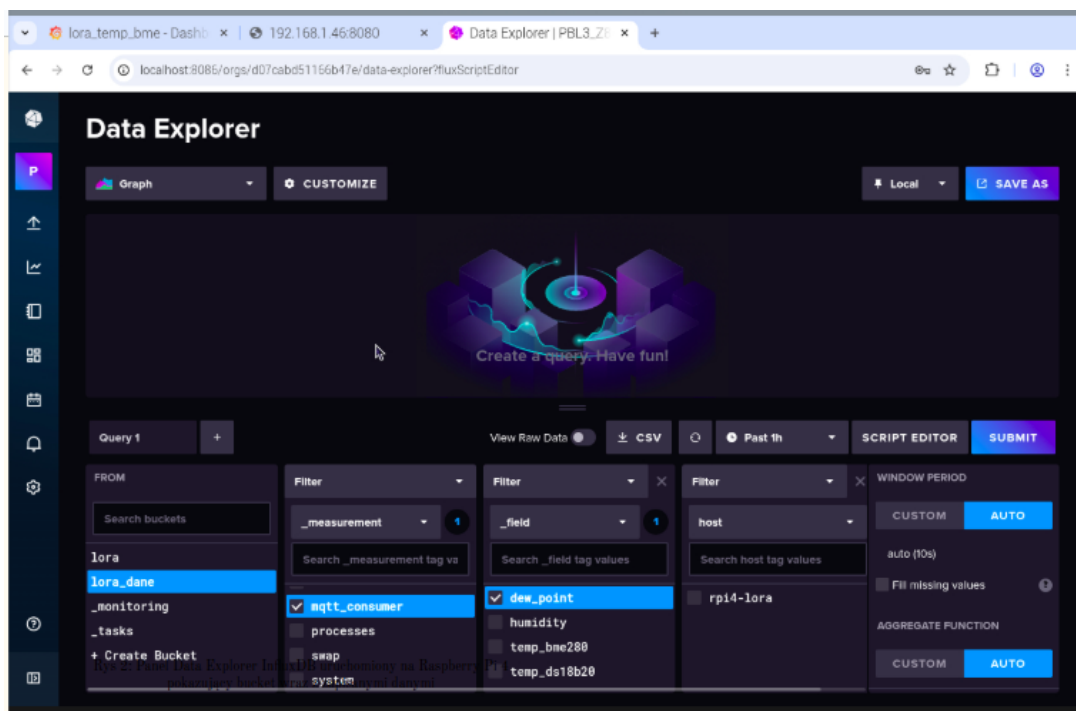
### 3.2 Konfiguracja struktury danych (v2)

- **Organizacja:** PBL3\_Z8
- **Bucket:** lora\_dane – kontener
- **Uwierzytelnianie:** Wygenerowano API Token: *fB3HQxYQBvBfTt7U0cSz94mb1Q4AUQU7n2ycuDqEV8IFue-C9Z2uUGr7QBJ1oXnzh19Cw68G3YweXrsDfogrA* == z uprawnieniami do zapisu i odczytu, który został następnie zaimplementowany w konfiguracji telegrafa.

### 3.3 Model danych

Dane trafiające do bazy za pośrednictwem Telegrafa są automatycznie indeksowane według następującego schematu:

- **\_measurement:** mqtt\_consumer (kategoria pomiaru).
- **\_field:** np temperature, humidity (mierzone wartości liczbowe).
- **\_tag:** host, topic (metadane tekstowe pozwalające na filtrowanie, np. po nazwie konkretnego czujnika).



Rysunek 6: wizualna prezentacja strony na rpi4

## 4 Grafana

W projekcie wykorzystano język zapytań Flux nie tylko do pobierania surowych danych, ale także do wykonywania obliczeń pochodnych w czasie rzeczywistym. Przykładem jest wyznaczanie Punktu Rosy (Dew Point) na podstawie temperatury i wilgotności, co odciąża aplikację wizualizacyjną.

W architekturze Grafana nie przechowuje danych, lecz pobiera je dynamicznie z bazy InfluxDB, renderując je w formie interaktywnych wykresów, wskaźników i tabel itd.

### 4.1 Konfiguracja

W celu integracji warstwy wizualizacyjnej z warstwą przechowywania, w systemie Grafana zdefiniowano nowe źródło danych typu InfluxDB. Ze względu na zastosowanie bazy danych w wersji 2.x, jako język zapytań wybrano Flux, co umożliwia pełne wykorzystanie możliwości analitycznych silnika bazy. Połączenie realizowane jest z lokalnym serwerem pod adresem <http://localhost:8086>, Grafana jest na <http://localhost:3000>.

### 4.2 Dashboard

każdy dashboard filtruje dane i prezentuje na wykresie, każdy ma także osobne zapytanie w języku Flux.

#### 1. Wykresy czasowe (Time Series):

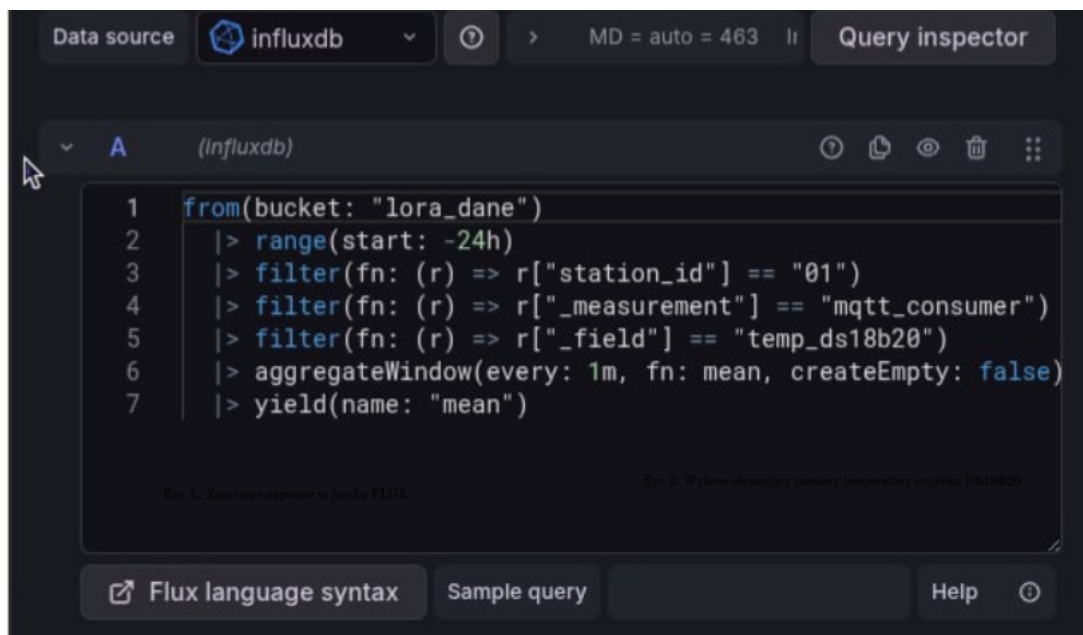
Służą do analizy trendów i zmian parametrów w czasie (np. przebieg temperatury z ostatnich 24 godzin). Pozwalają na identyfikację anomalii oraz cyklicznych zmian dobowych. Na jednym panelu nałożono dwie serie danych (`temp_ds18b20`, `temp_bme280` oraz `humidity`) w celu porównania odczytów z dwóch różnych czujników.

## 2. Wskaźniki statystyczne (Stat):

Prezentują ostatnią znaną wartość. Zastosowano je dla następujących parametrów:

- Aktualny punkt rosy (`dew_point`).

przykładowe zapytanie oraz wykres:



Rysunek 7: zapytanie dla temperatury z czujnika ds18b20



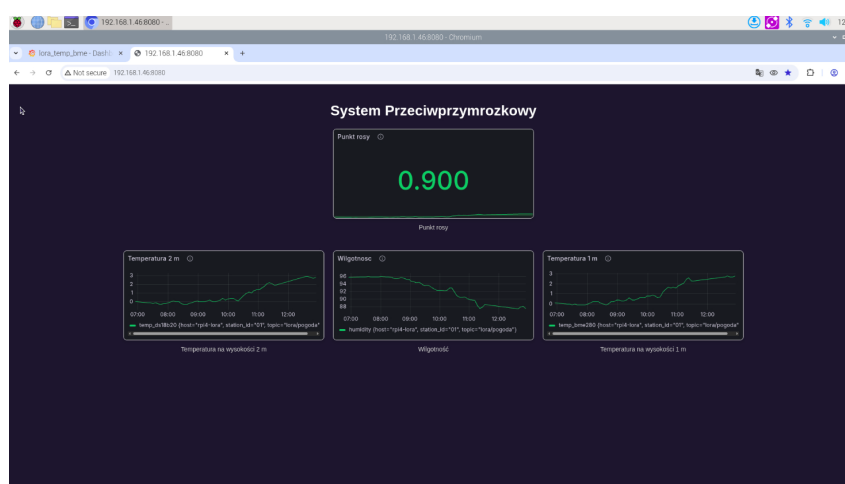


Rysunek 8: wykres tej temperatury naglej zmianę ukazuje różnice kiedy czujnik był w pomieszczeniu a kiedy na dworze.

Zastosowanie funkcji `aggregateWindow` jest kluczowe dla wydajności – przy wyświetlaniu wykresu z długiego okresu (np. 30 dni), Grafana nie pobiera milionów punktów, lecz jedynie uśrednione wartości, co znacząco przyspiesza renderowanie.

## 5 Strona internetowa

Wykorzystaliśmy nginx jako serwer HTTP. W obecnym kształcie strona przedstawia wyniki pomiarów (dwa pomiary temperatury, pomiar wilgotności i obliczony punkt rosy) jako wykresy z Grafany. W końcowej konfiguracji możliwe będzie podejście danych dla wielu stacji oraz wyświetlenie komunikatów o zagrożeniu przymrozkami.

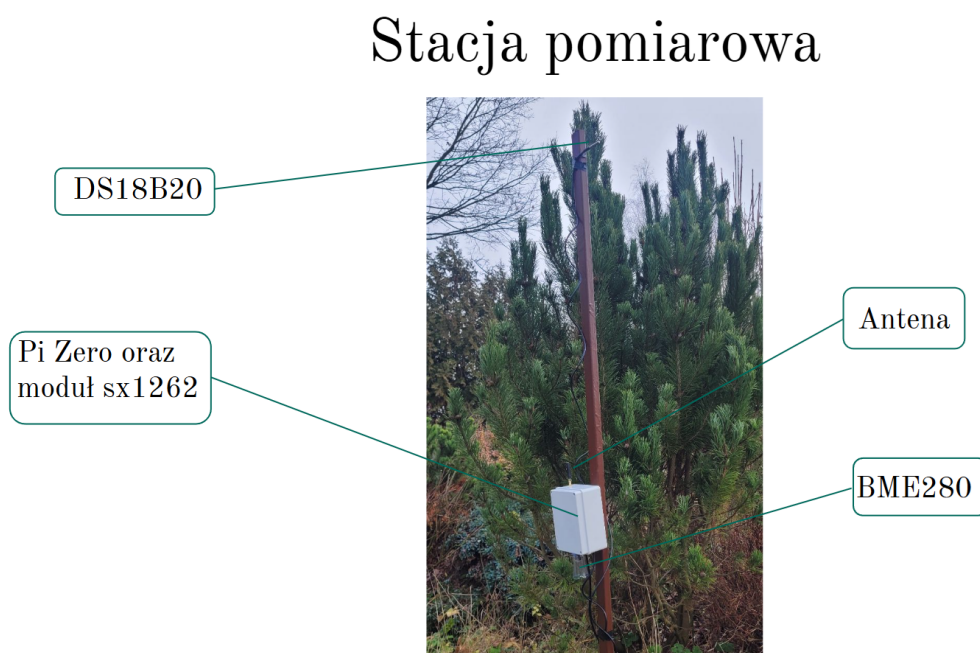


Rysunek 9: Strona przedstawiająca wyniki pomiarów w postaci wykresów z Grafany

## 6 Stacja pomiarowa

### 6.1 Komponenty stacji pomiarowej

W etapie drugim zbudowaliśmy stację zbierającą pomiary i wysyłającą dane za pomocą Lory do odbiornika. Stacja składa się z Raspberry Pi Zero, modułu SX1262 Lora, czujnika BME280, czujnika DS18B20 oraz anteny. Stacja wykonuje pomiary co 30 sekund i po 5 minutach liczy średnią, która potem jest wysyłana w ramce. Wysyłanie średniej co 5 minut pozwala zmniejszyć czas transmisji jednej stacji, co w przyszłości pozwoli na rozszerzenie systemu o dodatkowe stacje pomiarowe. Mierzone parametry to: wilgotność na wysokości 1 m (czujnik BME280), temperatura na wysokości 1 m (czujnik BME280), temperatura na wysokości 2 m (czujnik DS18B20). Raspberry Pi Zero oraz moduł SX1262 zostały umieszczone wewnątrz hermetycznej puszk instalacyjnej o wymiarach 150x110x70 mm, na górnej ścianie puszk wywiercony został otwór przez który przepuszczono przejściówkę do anteny. Antenę zamocowano na stałe w pozycji pionowej. Od spodu puszk wywiercony został otwór przez który przepuszczono kable do czujnika BME280, sam czujnik został osłonięty plastikową obudową, chroniącą przed bezpośrednim działaniem deszczu lub śniegu. U spodu puszk przewiercono również otwór dla czujnika DS18B20, który jest przymocowany u góry słupka na wysokości ok. 200 cm. Spód puszk znajduje się na wysokości ok. 105 cm, a czujnik BME280 na wysokości ok. 100 cm nad powierzchnią gruntu.



Rysunek 10: Opis komponentów w stacji pomiarowej

### 6.2 Działanie stacji pomiarowej

Po zweryfikowaniu działania wszystkich czujników i poprawnej komunikacji przy pomocy modułu Lora, stacja została umieszczona na dworze i w momencie pisania sprawozdania działa nieprzerwanie od ponad 70 godzin.

```
[20:38:52] OK | 01|+002.7|+002.7|098.9|10|203852
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[20:44:01] OK | 01|+002.6|+002.6|098.8|10|204401
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[20:49:10] OK | 01|+002.5|+002.6|098.8|10|204910
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[20:54:19] OK | 01|+002.5|+002.5|098.8|10|205419
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[20:59:28] OK | 01|+002.4|+002.4|098.8|10|205928
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[21:04:37] OK | 01|+002.3|+002.3|098.8|10|210437
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[21:09:46] OK | 01|+002.3|+002.3|098.8|10|210946
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[21:14:55] OK | 01|+002.2|+002.2|098.8|10|211455
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[21:20:05] OK | 01|+002.2|+002.2|098.8|10|212005
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[21:25:14] OK | 01|+002.2|+002.2|098.8|10|212514
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[21:30:23] OK | 01|+002.1|+002.2|098.8|10|213023
[SPI DEBUG] _writeBytes opCode=0x83, busyCheck=False
[SPI DEBUG] Wysłano: ['0x83', '0x0', '0x0', '0x0'], Odebrano: ['0xa2', '0xa2', '0xa2', '0xa2']
[21:35:32] OK | 01|+002.1|+002.1|098.7|10|213532
```

Rysunek 11: Logi przedstawiające ramki wysyłane przez Pi Zero

### 6.3 Parametry Lory

Użyte parametry to SF = 7, Bandwidth = 500 kHz, CR = 5, Moc nadawania 14 dBm. Parametr SF został ustalony, tak aby pasował do warunków bliskiej odległości nadajnika i odbiornika, wysokie BW pozwala zmniejszyć czas transmisji wiadomości, niski CR również pozwala ograniczyć czas transmisji. Przy niższej mocy występują problemy z odbiorem ramek, jednak może zostać to poprawione poprzez korzystniejsze ustawienie anteny odbiorczej.

### 6.4 Przesyłana ramka Lory

Obecnie przesyłamy w ramce długości 32 bajtów id stacji (w wypadku naszej stacji 01), średnią temperaturę za okres 5 minut dla czujnika DS18B20, średnią temperaturę dla BME280, średnią wilgotność dla BME280 oraz wartość najmniejszą dla liczby wykonanych pomiarów (tj. gdy BME280 w czasie 5 minut wykona tylko 2 pomiary, a DS18B20 10, zostanie zwrócona wartość 2 - jest to informacja diagnostyczna pozwalająca zauważyć nieprawidłowości w działaniu czujników). Obecna ramka jest tylko tymczasowa, gdyż później zostanie również dodany pomiar prędkości wiatru i pomiary wewnątrz puszk instalacyjnej (wilgotność i temperatura - może być przesyłana wartość 0 lub 1 gdy np. zostanie przekroczony stan alarmowy).

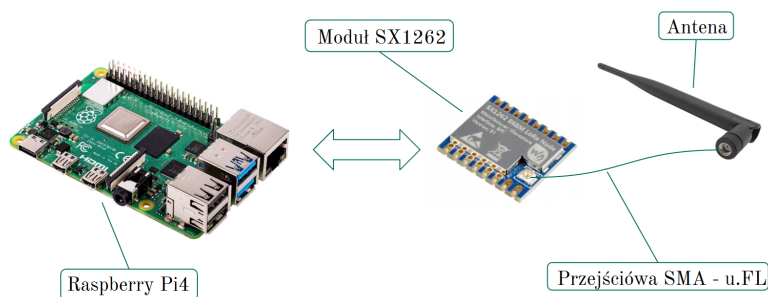
```
01|+001.9|+002.0|098.7|10|231222
```

Rysunek 12: Przesyłana ramka lora

## 6.5 Stacja odbierająca dane

Stacja ta składa się z Raspberry Pi 4B, modułu SX1262 oraz anteny. Pi odbiera dane za pomocą modułu Lora, parsuje je i wysyła w formacie JSON do brokera MQTT, dodając do odebranych danych punkt rosy obliczony z danych zebranych przez BME280.

### Stacja odbierająca dane



Rysunek 13: Schemat stacji odbiorczej