Termometry w praktyce: wybór, montaż i własny czujnik (DIY)

◫◫ Poziom: średnio

◫◫ Dotyczy: ESP32, ESPHome 2026.x, DS18B20, Raspberry Pi OS, Zabbix 7.0 LTS

▯ Czas: ~90 min

Budujemy własny zestaw czujników temperatury do szafy serwerowej: ESP32 z czterema sondami DS18B20 na jednej magistrali i alarmami w Zabbixie na `mon01`, a krócej wariant na Raspberry Pi. Wracamy tu przy montażu nowego zestawu, wymianie sondy i szukaniu przyczyny braku danych. Komendy ESPHome wykonujemy na stacji administratora z Pythonem 3.12 albo nowszym i dostępem do sieci serwerowni, a komendy drugiego wariantu na samej malince jako użytkownik z uprawnieniami sudo.

◫◫ Skróty użyte na tej stronie (24)

Skrót	Co znaczy
1-Wire	Magistrala firmy Dallas, w której czujniki rozmawiają z mikrokontrolerem po jednym przewodzie danych, a każdy czujnik ma własny adres
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
ASHRAE	American Society of Heating, Refrigerating and Air-Conditioning Engineers: amerykańskie stowarzyszenie inżynierów, którego zalecenia opisują temperaturę i wilgotność w serwerowniach
CE	Community Edition: darmowa, społecznościowa wersja Dockera, w odróżnieniu od płatnej wersji Enterprise
DIY	Do It Yourself: zrób to sam, czyli własnoręcznie zbudowany czujnik zamiast gotowego urządzenia
DS18B20	Cyfrowy czujnik temperatury z magistralą 1-Wire, z fabrycznym 64-bitowym adresem, popularny w układach domowych i małych serwerowniach
ESP32	Tani mikrokontroler z Wi-Fi i Bluetooth firmy Espressif: malutki komputer, z którego budujemy własne czujniki sieciowe

Skrót	Co znaczy
ESPHome	Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć
GPIO	General Purpose Input Output: uniwersalny pin mikrokontrolera, który można ustawić jako wejście albo wyjście
HTTP	Hypertext Transfer Protocol: protokół stron WWW
IT	Information Technology: dział albo praca związana z komputerami i infrastrukturą firmy
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
JSONPath	Zapis ścieżki, która wskazuje jedną wartość w dokumencie JSON, np. $.value to pole value na samym wierzchu
LTS	Long Term Support: wersja z długim wsparciem poprawkami
MQTT	Message Queuing Telemetry Transport: lekki protokół, w którym urządzenia wysyłają krótkie wiadomości na nazwane tematy przez serwer pośredniczący
OS	Operating System: system operacyjny
PoE	Power over Ethernet: zasilanie urządzenia tym samym kablem sieciowym, którym płyną dane
ppoż.	Przeciwpożarowy: dotyczący ochrony przed pożarem, na przykład centrala ppoż.
REST	sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu
SMS	Short Message Service: krótka wiadomość tekstowa na telefon, która dociera także bez internetu
URL	Uniform Resource Locator: adres strony lub zasobu
USB	Universal Serial Bus: standard portów i urządzeń podłączanych do komputera, tu pendrive'y i inne nośniki wymienne
Wi-Fi	Sieć bezprzewodowa: wygodna w biurze, ale w serwerowni dokłada kolejną możliwą przyczynę awarii
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Fiszka

□□ ZMIENIA SYSTEM

Instalacja ESPHome: pipx install esphome

```
pipx install esphome
```

ESPHome to narzędzie, które z jednego pliku YAML buduje program dla płytki ESP32 i wgrywa go przez USB albo przez sieć. Wersja 2026.9.1 wymaga Pythona 3.12 albo nowszego [1].

Przykład:

```
  installed package esphome 2026.9.1, installed using Python 3.13.5
  These apps are now globally available
    - esphome
done!
```

ESPHome = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **YAML** = YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach · **ESP32** = Tani mikrokontroler z Wi-Fi i Bluetooth firmy Espressif: malutki komputer, z którego budujemy własne czujniki sieciowe · **USB** = Universal Serial Bus: standard portów i urządzeń podłączanych do komputera, tu pendrive'y i inne nośniki wymienne · **pipx** = instaluje narzedzia napisane w Pythonie, kazde we wlasnym, odizolowanym srodowisku, bez mieszania zaleznosci miedzy programami

TYLKO ODCZYT

Wygenerowanie klucza API: openssl rand -base64 32

```
openssl rand -base64 32
```

Klucz szyfrujący natywne API ESPHome to 32 losowe bajty zapisane w base64. Wynik wpisujemy w sekcji `api` pliku konfiguracyjnego [2].

> W serwerowni Zielonej płytki to `esp01` i `esp02`: ESP32 z Ethernetem i zasilaniem PoE, a nie z Wi-Fi. Kabel nie zależy od zasięgu w szafie, a PoE zasila płytkę z przełącznika, więc nie ma osobnego zasilacza. Plik poniżej opisuje płytkę `esp02` w wariancie z Wi-Fi, który zostaje dla domu i do prób na biurku. W serwerowni zamiast bloku `wifi` wpisujemy blok `ethernet` jak na stronie ESPHome i urządzenia: adopcja esp01 i esp02, API albo MQTT, nazwy encji, a w sekcji `esp32` ustawiamy `board: esp32-poe`:

```
ethernet:
  type: LAN8720
  mdc_pin: GPIO23
  mdio_pin: GPIO18
  clk:
    pin: GPIO17
```

```
    mode: CLK_OUT
  phy_addr: 0
  power_pin: GPIO12
  manual_ip:
    static_ip: 10.20.10.71
    gateway: 10.20.10.1
    subnet: 255.255.255.0
    dns1: 10.20.10.11
```

Blok pochodzi z przykładu w dokumentacji ESPHome dla płytki Olimex ESP32-POE, a numery pinów sprawdzamy w opisie własnej płytki. Ethernet zajmuje część pinów, więc inne magistrale ustawiamy na wolnych. Magistrala 1-Wire na GPIO4 z pliku poniżej mieści się bez zmian.

API = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **ESPHome** = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **ESP32** = Tani mikrokontroler z Wi-Fi i Bluetooth firmy Espressif: malutki komputer, z którego budujemy własne czujniki sieciowe · **PoE** = Power over Ethernet: zasilanie urządzenia tym samym kablem sieciowym, którym płyną dane · **Wi-Fi** = Sieć bezprzewodowa: wygodna w biurze, ale w serwerowni dokłada kolejną możliwą przyczynę awarii · **MQTT** = Message Queuing Telemetry Transport: lekki protokół, w którym urządzenia wysyłają krótkie wiadomości na nazwane tematy przez serwer pośredniczący · **1-Wire** = Magistrala firmy Dallas, w której czujniki rozmawiają z mikrokontrolerem po jednym przewodzie danych, a każdy czujnik ma własny adres · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

PLIK KONFIGURACYJNY

Plik konfiguracyjny ESPHome: esp02.yaml

```
esphome:
  name: esp02
  friendly_name: esp02

esp32:
  board: esp32dev
  framework:
    type: esp-idf

logger:

wifi:
```

```
  ssid: "NazwaSieciWifi"
  password: "HasloSieciWifi"
  ap:
    ssid: "esp02"
    password: "HasloAwaryjne"

# Bez klienta API (Home Assistant) płytka restartowałaby się co 15 minut,
# a u nas dane pobiera Zabbix przez web_server, więc wyłączamy ten restart.
api:
  encryption:
    # Przykładowy klucz (32 bajty w base64). Własny generujemy poleceniem openssl rand -
base64 32.
    key: "0690dTduZJXaRAhYiVWcgHzWAV4sXEigdsftww8qW+4="
  reboot_timeout: 0s

ota:
  - platform: esphome
    password: "HasloOta"

web_server:
  port: 80
  auth:
    username: "nazwa_usera"
    password: "HasloWebServera"

one_wire:
  - platform: gpio
    pin: GPIO4

sensor:
  # Adresy poniżej są przykładowe: podmieniamy je na adresy z dziennika (patrz niżej).
  - platform: dallas_temp
    name: "wlot_dol"
    address: 0x8c0000046e9b4f28
    resolution: 12
    update_interval: 30s
    filters:
      - offset: 0.0
  - platform: dallas_temp
```

```
    name: "wlot_srodek"
    address: 0x3d0000046e9c1a28
    resolution: 12
    update_interval: 30s
    filters:
      - offset: 0.0
  - platform: dallas_temp
    name: "wlot_gora"
    address: 0xe20000046e9d7b28
    resolution: 12
    update_interval: 30s
    filters:
      - offset: 0.0
  - platform: dallas_temp
    name: "korytarz_goracy"
    address: 0x510000046e9e2c28
    resolution: 12
    update_interval: 30s
    filters:
      - offset: 0.0
  - platform: wifi_signal
    name: "wifi_sygnal"
    update_interval: 60s
```

Plik opisuje całą płytkę: sieć Wi-Fi, szyfrowane API, aktualizacje przez sieć, serwer HTTP z prostym interfejsem REST, magistralę 1-Wire na pinie GPIO4 i cztery czujniki. Składnia `one_wire` z `platform: gpio` oraz `dallas_temp` jest aktualna w dokumentacji ESPHome [3][4]. Nazwy czujników celowo nie mają spacji ani polskich liter, bo wchodzą do adresu URL.

W prawdziwym pliku hasła trzymamy w `secrets.yaml` i zapisujemy je jako `!secret nazwa`. Tutaj stoją placeholdery, żeby plik był jednym blokiem do skopiowania.

Przykład wpisów w pliku `secrets.yaml` i odwołań do nich:

```
# secrets.yaml
wifi_haslo: "HasloSieciWifi"
api_klucz: "0690dTduZJXaRAhYiVWcgHzWAV4sXEigdsftww8qW+4="   # przykład, własny klucz
generujemy poleceniem openssl rand -base64 32
ota_haslo: "HasloOta"
```

```
# w esp02.yaml
wifi:
  password: !secret wifi_haslo
api:
  encryption:
    key: !secret api_klucz
ota:
  - platform: esphome
    password: !secret ota_haslo
```

ESPHome = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **Wi-Fi** = Sieć bezprzewodowa: wygodna w biurze, ale w serwerowni dokłada kolejną możliwą przyczynę awarii · **API** = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **HTTP** = Hypertext Transfer Protocol: protokół stron WWW · **REST** = sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu · **1-Wire** = Magistrala firmy Dallas, w której czujniki rozmawiają z mikrokontrolerem po jednym przewodzie danych, a każdy czujnik ma własny adres · **URL** = Uniform Resource Locator: adres strony lub zasobu

PLIK KONFIGURACYJNY

Zamiana API na MQTT: sekcja mqtt

```
mqtt:
  broker: "NazwaBrokeraMqtt"
  username: "nazwa_usera"
  password: "HasloMqtt"
  topic_prefix: "serwerownia/esp02"
```

Ten blok wpisujemy w miejsce sekcji `api`, jeśli dane mają iść przez brokera MQTT, a nie przez Home Assistanta. Jeśli zostawimy oba, a nie ma klienta API, trzeba ustawić `reboot_timeout: 0s` w sekcji `api` [5].

API = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **MQTT** = Message Queuing Telemetry Transport: lekki protokół, w którym urządzenia wysyłają krótkie wiadomości na nazwane tematy przez serwer pośredniczący

TYLKO ODCZYT

Sprawdzenie pliku: esphome config esp02.yaml

```
esphome config esp02.yaml
```

Polecenie wczytuje plik, sprawdza składnię i parametry i wypisuje je w pełnej postaci, niczego nie wgrywając. Kończy się komunikatem `Configuration is valid!`.

esphome = ESPHome: buduje i wgrywa oprogramowanie na ESP32 z pliku YAML, pokazuje logi urządzenia

TYLKO ODCZYT

Wgranie programu przez USB: esphome run esp02.yaml

```
esphome run esp02.yaml
```

Przy pierwszym wgraniu płytkę podłączamy kablem USB do stacji administratora. Kolejne aktualizacje idą przez sieć, według ustawień z sekcji `ota`.

USB = Universal Serial Bus: standard portów i urządzeń podłączanych do komputera, tu pendrive'y i inne nośniki wymienne · **esphome** = ESPHome: buduje i wgrywa oprogramowanie na ESP32 z pliku YAML, pokazuje logi urządzenia

TYLKO ODCZYT

Odczyt adresów czujników: esphome logs esp02.yaml

```
esphome logs esp02.yaml
```

Po starcie ESPHome skanuje magistralę i wypisuje w dzienniku adresy wszystkich znalezionych urządzeń, nawet jeśli żaden czujnik nie jest jeszcze opisany w pliku [3]. Skanowanie widać także wtedy, gdy plik ma tylko sekcję `one_wire`.

Przykład (skrócony, bez znaczników czasu i nazw modułów):

```
Found devices:
  0x8c0000046e9b4f28 (DS18B20)
  0x3d0000046e9c1a28 (DS18B20)
  0xe20000046e9d7b28 (DS18B20)
  0x510000046e9e2c28 (DS18B20)
```

ESPHome = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **esphome** = ESPHome: buduje i wgrywa oprogramowanie na ESP32 z pliku YAML, pokazuje logi urządzenia

TYLKO ODCZYT

Odczyt czujnika przez REST: curl

```
curl --user nazwa_usera http://NazwaCzujnika/sensor/wlot_gora
```

Przy takim zapisie `curl` pyta o hasło, więc nie zostaje ono w historii poleceń. Serwer HTTP płytki odpowiada obiektem JSON z polem `value`, które Zabbix wyciąga w następnym kroku [6].

Przykład:

```
{"id":"sensor/wlot_gora","state":"26.1 °C","value":26.1}
```

REST = sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu · **HTTP** = Hypertext Transfer Protocol: protokół stron WWW · **JSON** = JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

TYLKO ODCZYT

Edycja pliku konfiguracyjnego Raspberry Pi: sudoedit /boot/firmware/config.txt

```
sudoedit /boot/firmware/config.txt
```

Na Raspberry Pi OS plik `config.txt` leży w partycji startowej pod `/boot/firmware/` [7]. Na końcu pliku dopisujemy linię z następnej fiszki.

OS = Operating System: system operacyjny · **sudoedit** = sudo edit: bezpieczne otwarcie pliku z uprawnieniami administratora we własnym edytorze, z zapisem pliku dopiero po wyjściu

PLIK KONFIGURACYJNY

Włączenie magistrali 1-Wire: dtoverlay=w1-gpio

```
dtoverlay=w1-gpio
```

Nakładka `w1-gpio` uruchamia sterownik 1-Wire na pinie GPIO4, a pin można zmienić parametrem `gpiopin` [8]. Zmiana wchodzi w życie po ponownym uruchomieniu.

1-Wire = Magistrala firmy Dallas, w której czujniki rozmawiają z mikrokontrolerem po jednym przewodzie danych, a każdy czujnik ma własny adres

TYLKO ODCZYT

Ponowne uruchomienie systemu: sudo reboot

```
sudo reboot
```

reboot = restart systemu: zamyka usługi i uruchamia komputer od nowa

TYLKO ODCZYT

Lista czujników 1-Wire: ls /sys/bus/w1/devices/

```
ls /sys/bus/w1/devices/
```

Każdy czujnik DS18B20 ma katalog zaczynający się od `28-`, a dalej jego numer fabryczny. Prefiks `28` to kod rodziny układu [9].

Przykład:

```
28-3c01d6075a21  28-3c01d6075b32  28-3c01d6075c43  w1_bus_master1
```

1-Wire = Magistrala firmy Dallas, w której czujniki rozmawiają z mikrokontrolerem po jednym przewodzie danych, a każdy czujnik ma własny adres · **DS18B20** = Cyfrowy czujnik temperatury z magistralą 1-Wire, z fabrycznym 64-bitowym adresem, popularny w układach domowych i małych serwerowniach · **ls** = list: wypisz zawartość katalogu

TYLKO ODCZYT

Odczyt jednego czujnika: cat w1_slave

```
cat /sys/bus/w1/devices/28-NumerCzujnika/w1_slave
```

(gdzie `28-NumerCzujnika` to nazwa katalogu czujnika z poprzedniej fiszki)

Plik ma dwie linie. Pierwsza kończy się słowem `YES`, gdy suma kontrolna się zgadza, a druga zawiera temperaturę w tysięcznych stopnia po `t=`.

Przykład:

```
4b 01 4b 46 7f ff 05 10 d8 : crc=d8 YES
4b 01 4b 46 7f ff 05 10 d8 t=20687
```

cat = concatenate: sklej i wypisz pliki

□□ ZMIENIA SYSTEM

Instalacja programu zabbix_sender: sudo apt install zabbix-sender

```
sudo apt install zabbix-sender
```

`zabbix_sender` to mały program, który wysyła pojedyncze wartości do serwera Zabbix, bez agenta na maszynie [10].

□□ **apt** = Advanced Package Tool: menedżer pakietów Debiana i Ubuntu

□□ TYLKO ODCZYT

Wysyłka testowa wartości: zabbix_sender

```
zabbix_sender -z NazwaSerweraZabbix -s NazwaHosta -k KluczElementu -o Wartosc -vv
```

(gdzie `NazwaHosta` to nazwa hosta dokładnie taka, jak w Zabbixie, z rozróżnieniem wielkości liter, a `KluczElementu` to klucz elementu typu trapper)

Przykład:

```
info from server: "processed: 1; failed: 0; total: 1; seconds spent: 0.000052"
sent: 1; skipped: 0; total: 1
```

□□ **zabbix_sender** = Zabbix sender: wysyła pojedynczą wartość do elementu typu trapper w Zabbixie

□□ TYLKO ODCZYT

Skrypt Pythona: w1_do_zabbixa.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""Odczyt czujników DS18B20 z magistrali 1-Wire i wysyłka do Zabbixa przez
zabbix_sender."""
import os
import subprocess
import sys

SERWER_ZABBIXA = "mon01"
```

```
NAZWA_HOSTA = "czujnik-rpi01"
KATALOG_W1 = "/sys/bus/w1/devices"

# numer czujnika z katalogu 28-... -> klucz elementu typu trapper w Zabbixie
CZUJNIKI = {
    "28-3c01d6075a21": "temp.wlot.dol",
    "28-3c01d6075b32": "temp.wlot.srodek",
    "28-3c01d6075c43": "temp.wlot.gora",
}


def odczytaj(numer, katalog=KATALOG_W1):
    """Zwraca temperaturę w °C albo None, gdy odczyt jest niepewny."""
    sciezka = os.path.join(katalog, numer, "w1_slave")
    try:
        with open(sciezka, encoding="ascii") as plik:
            linie = plik.read().splitlines()
    except OSError as blad:
        print("Brak odczytu %s: %s" % (numer, blad), file=sys.stderr)
        return None
    if len(linie) < 2 or not linie[0].endswith("YES"):
        print("Błędna suma kontrolna: %s" % numer, file=sys.stderr)
        return None
    pozycja = linie[1].find("t=")
    if pozycja < 0:
        return None
    mc = int(linie[1][pozycja + 2:])
    if mc == 85000:
        # 85,0 °C to wartość startowa układu, nie pomiar
        print("Wartość startowa 85 °C: %s" % numer, file=sys.stderr)
        return None
    return mc / 1000.0


def wyslij(wartosci):
    """Wysyła pary (klucz, wartość) jednym wywołaniem zabbix_sender."""
    wejscie = "".join(
        "%s %s %.3f\n" % (NAZWA_HOSTA, klucz, temp) for klucz, temp in wartosci
    )
```

```
    wynik = subprocess.run(
        ["zabbix_sender", "-z", SERWER_ZABBIXA, "-i", "-"],
        input=wejscie, text=True, capture_output=True, timeout=30,
    )
    print(wynik.stdout.strip())
    return wynik.returncode


def main():
    wartosci = []
    for numer, klucz in CZUJNIKI.items():
        temp = odczytaj(numer)
        if temp is not None:
            wartosci.append((klucz, temp))
    if not wartosci:
        print("Żaden czujnik nie dał poprawnego odczytu", file=sys.stderr)
        return 1
    return wyslij(wartosci)


if __name__ == "__main__":
    sys.exit(main())
```

Skrypt czyta pliki `w1_slave`, odrzuca odczyty z błędną sumą kontrolną i wartość startową 85 °C, a resztę wysyła jednym wywołaniem `zabbix_sender` na standardowe wejście programu. Numery w słowniku `CZUJNIKI` podmieniamy na numery z fiszki z listą czujników. Skrypt kompiluje się poleceniem `python3 -m py_compile`.

DS18B20 = Cyfrowy czujnik temperatury z magistralą 1-Wire, z fabrycznym 64-bitowym adresem, popularny w układach domowych i małych serwerowniach

ZMIENIA SYSTEM

Instalacja skryptu: sudo install

```
sudo install -m 0755 w1_do_zabbixa.py /usr/local/bin/w1_do_zabbixa.py
```

install = install: kopiuje pliki i tworzy katalogi z jednoczesnym ustawieniem uprawnień

TYLKO ODCZYT

Uruchomienie skryptu: python3 w1_do_zabbixa.py

```
python3 /usr/local/bin/w1_do_zabbixa.py
```

Przykład:

```
info from server: "processed: 3; failed: 0; total: 3; seconds spent: 0.000081"
sent: 3; skipped: 0; total: 3
```

python3 = uruchom interpreter Pythona w wersji 3 albo wskazany skrypt

ZMIENIA SYSTEM

Zadanie cykliczne: /etc/cron.d/w1-zabbix

```
* * * * * nazwa_usera /usr/bin/python3 /usr/local/bin/w1_do_zabbixa.py 2>&1 | logger -t
w1-zabbix
```

Wpis w katalogu `/etc/cron.d/` uruchamia skrypt co minutę jako wskazany użytkownik, a wynik trafia do dziennika systemowego pod znacznikiem `w1-zabbix`.

Z życia wzięte

Z ŻYCIA WZIĘTE

Poniedziałek, 7:35. Tomek wchodzi do serwerowni w Pamiątkowie i czuje, że jest ciepło, a wyświetlacz klimatyzatora pokazuje błąd po sobotnim zaniku zasilania. W Zabbixie nie ma żadnego alarmu, bo `hv02` i `hv01` zgłaszały ostrzeżenia temperatury tylko w swoich własnych logach, których nikt nie czytał. Serwery przetrwały dzięki zapasowi w ustawieniach wentylatorów, ale zespół uznał, że o przegrzaniu ma dowiadywać się z alarmu, a nie z zapachu. Tego samego dnia Tomek zamawia cztery sondy i płytkę, a do wieczora z Olą opisują, gdzie je zamontują.

Projekt DIY krok po kroku

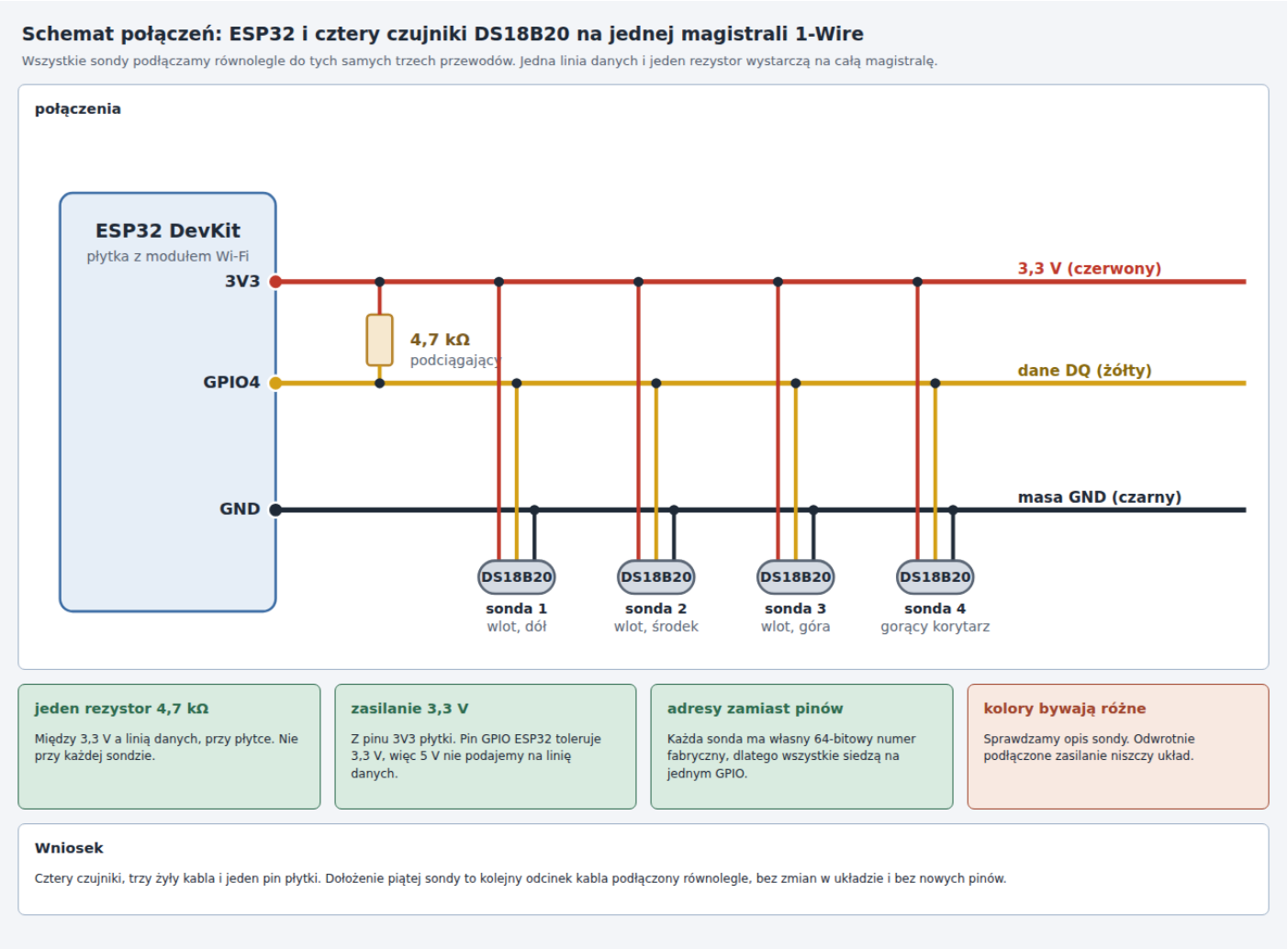
Zestaw mierzy temperaturę w czterech miejscach: trzy sondy na wlocie szafy (dół, środek, góra) i jedna w gorącym korytarzu. Dlaczego akurat tam, wyjaśnia strona Termometry: rodzaje, zasada

działania i zastosowania. Pierwsze trzy punkty to wartości, na których opiera się alarm, a czwarty służy do trendów i oceny różnicy temperatur.

Lista części

Część	Po co	Uwagi
Płytka ESP32 DevKit z modułem ESP32	procesor z Wi-Fi, który odczytuje czujniki i udostępnia je w sieci	płytka z gniazdem USB i wyprowadzonym pinem 3V3
Cztery sondy DS18B20 w stalowej gilzie na kablu	czujniki temperatury	wodoodporne sondy mają trzy przewody, a ich kolory różnią się między producentami
Rezystor 4,7 kΩ	podciągnięcie linii danych do 3,3 V	jeden na całą magistralę
Kabel trzyżyłowy albo skrętka	przedłużenie sond do miejsc w szafie	trzy żyły: zasilanie, masa i dane
Listwa zaciskowa albo lutowane połączenia z koszulkami	równoległe połączenie sond	sprawdzone, pewne styki
Zasilacz USB 5 V z certyfikatem CE	zasilanie płytki	z listwy w szafie, najlepiej z gniazda objętego zasilaniem awaryjnym
Obudowa na płytkę	ochrona przed kurzem i przypadkowym zwarciem	otwory na kable i dostęp do portu USB
Opaski, uchwyty i taśma dwustronna	mocowanie sond	z tworzywa, żeby sonda nie dotykała metalowej szafy
Termometr wzorcowy z certyfikatem wzorcowania	sprawdzenie sond	albo zlecenie wzorcowania w laboratorium
Lód i woda	punkt odniesienia 0 °C	w naczyniu o wysokich ściankach

Schemat połączeń



Rysunek 6. Schemat połączeń: wszystkie sondy podłączamy równolegle do zasilania 3,3 V, masy i jednej linii danych na pinie GPIO4. Jeden rezystor 4,7 kΩ między 3,3 V a linią danych obsługuje całą magistralę (kliknięcie otwiera pełny rozmiar)

Wszystkie cztery sondy łączymy równolegle, bo każda ma własny 64-bitowy adres i magistrala 1-Wire rozróżnia je po nim [9]. Zasilamy je z pinu 3V3 płytki, a rezystor 4,7 kΩ łączymy między 3V3 i linią danych. Dokumentacja ESPHome dopuszcza wartości w granicach ±1 kΩ, o ile kable nie są niezwykle długie [3].

Czujnik DS18B20 od SparkFun. Takie sondy łączymy równolegle do jednej linii danych, a kolory prze

Czujnik DS18B20 od SparkFun. Takie sondy łączymy równolegle do jednej linii danych, a kolory przewodów sprawdzamy w opisie konkretnej sondy. Źródło: Wikimedia Commons (autor i wolna licencja podane na stronie pliku).

Układ DS18B20 można też zasilać z linii danych (zasilanie pasożytnicze), ale w serwerowni wybieramy osobny przewód zasilający, bo daje stabilniejsze odczyty na dłuższym kablu [9]. Odwrotnie podłączone zasilanie nagrzewa i niszczy układ, więc kolory przewodów sprawdzamy w opisie sondy.

Pierwsze uruchomienie i adresy czujników

Najpierw wgrywamy plik tylko z sekcjami `wifi`, `api`, `ota` i `one_wire`, bez sekcji `sensor`. Po starcie dziennik (fiszka Odczyt adresów czujników) pokazuje cztery adresy, bo skanowanie magistrali działa także bez opisanych czujników [3].

Żeby poznać, który adres należy do której sondy, ogrzewamy ją w dłoni i obserwujemy, która wartość rośnie. Potem uzupełniamy plik o sekcję `sensor` z właściwymi adresami i nazwami `wlot_dol`, `wlot_srodek`, `wlot_gora` i `korytarz_goracy`. Adresów nie zastępujemy numerami kolejnymi (`index`), bo kolejność zmienia się po dodaniu albo wymianie sondy [4].

Montaż w szafie

Trzy sondy mocujemy na wlocie najwyższego, środkowego i najniższego serwera, około 5 cm przed obudową, tak jak opisuje ASHRAE przy sprawdzaniu instalacji [11]. Czwartą sondę zawieszamy za szafą, w gorącym korytarzu, mniej więcej na środku wysokości. Kable prowadzimy wzdłuż prowadnic szafy, a nie przez wlot powietrza, i nie zasłaniamy wentylatorów.

◫ ZRZUT EKRANU DO UZUPEŁNIENIA

Zdjęcie do zrobienia po montażu: szafa 1 z frontu, trzy sondy na wlocie na trzech wysokościach i opis każdej sondy na kablu.

Płytkę ESP32 montujemy poza strumieniem gorącego powietrza, najlepiej z boku szafy od strony zimnego korytarza. Sondy opisujemy na obu końcach kabla, a adresy zapisujemy w tabeli w NetBoxie przy szafie.

Kalibracja i porównanie z wzorcem

> **Pojęcie: wzorcowanie.** Porównanie wskazań przyrządu z wzorcem o znanej, wyższej dokładności i zapisanie różnicy. Oddzielnym krokiem jest regulacja, czyli poprawienie wskazań, na przykład przez dopisanie stałej poprawki w pliku konfiguracyjnym.

Punkt 0 °C przygotowujemy w kąpieli lodowej. Wysokie naczynie napełniamy po brzegi pokruszonym lodem, dolewamy wody tak, żeby sięgała pół centymetra poniżej brzegu, i czekamy minutę lub dwie [12]. Sondy zanurzamy w środku mieszaniny i delikatnie mieszamy, żeby nie dotykały ścianek ani dna, bo wtedy wynik jest fałszywy [12].

Dobrze przygotowana kąpiel ma 0 °C z dokładnością do ułamków stopnia, a źle przygotowana, na przykład zwykła szklanka z kostkami lodu, potrafi pokazać kilkanaście stopni za dużo [12].

DS18B20 w tym zakresie ma gwarantowane ±0,5 °C [9]. Odczyt zapisujemy dla każdej sondy osobno.

Drugi punkt wybieramy blisko temperatury pracy, w pokoju około 22 do 25 °C. Wszystkie sondy leżą obok siebie z wzorcowym termometrem w tym samym miejscu przez co najmniej kilkanaście minut, a potem porównujemy odczyty. Różnice poniżej ±0,5 °C przyjmujemy bez zmian, a większe korygujemy poprawką `offset` w sekcji `filters` danej sondy.

Obudowa i miejsce montażu

Obudowa płytki ma dostęp powietrza, bo ESP32 się grzeje i nie powinien podgrzewać sond. Sondy wychodzą kablem na zewnątrz i nie wiszą przy samej obudowie płytki. Płytkę zasilamy zasilaczem USB z listwy w szafie, a kabel USB prowadzimy tak, żeby nikt nie zahaczył o niego przy pracach w szafie.

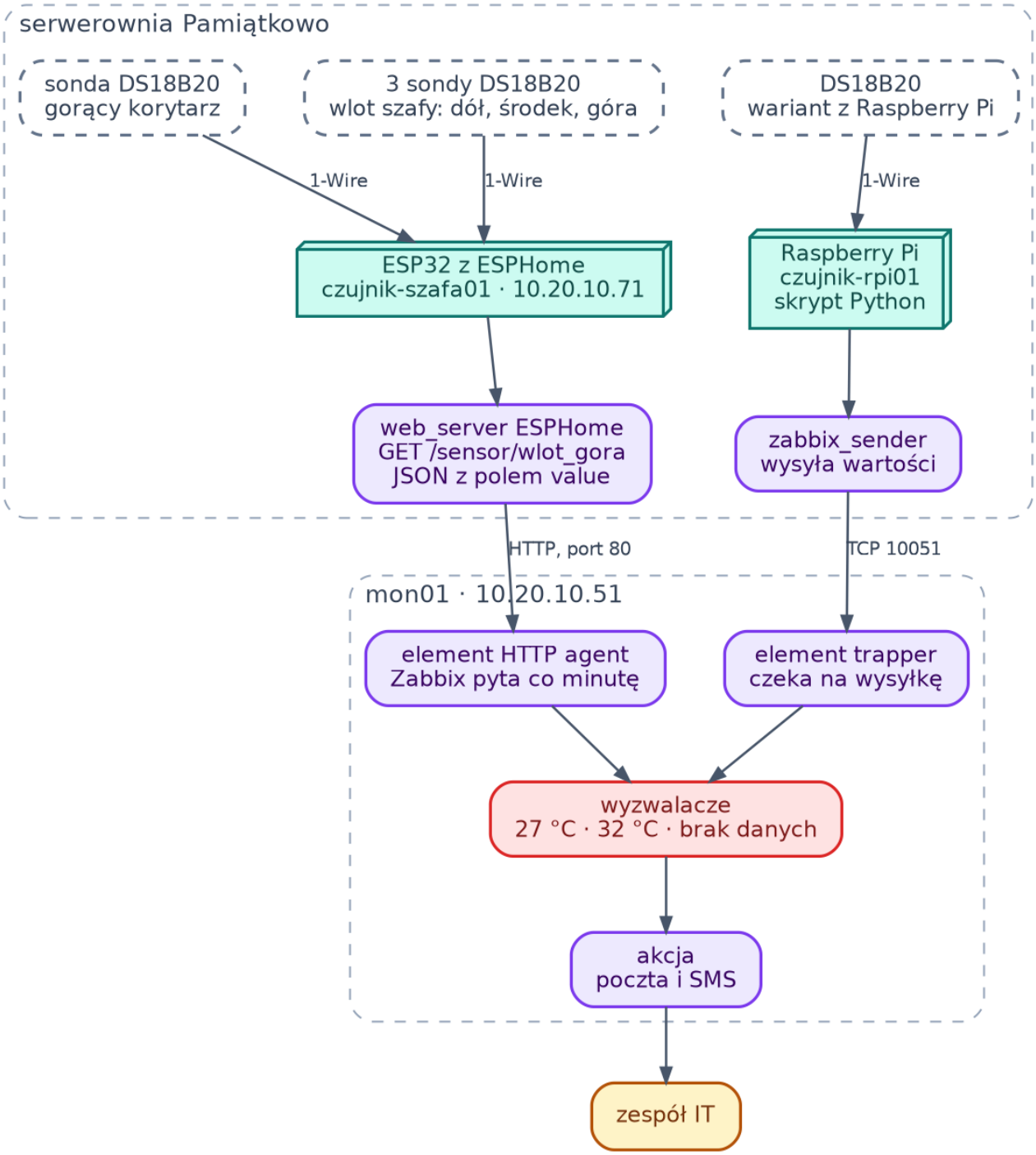
Drugi wariant: Raspberry Pi

Ten sam czujnik DS18B20 podłączamy do Raspberry Pi. Fizycznie łączymy pin 1 (3,3 V), pin 7 (GPIO4) i pin 9 (masa) z trzema żyłami sond, a rezystor 4,7 kΩ stawiamy między pinami 1 i 7. Schemat jest ten sam co na Rysunku 6, tylko zamiast płytki ESP32 stoi złącze malinki.

Włączamy nakładkę 1-Wire fiszkami Edycja pliku konfiguracyjnego, Włączenie magistrali i Ponowne uruchomienie. Nakładka `w1-gpio` ma też wariant `w1-gpio-pi5`, opisany jako wersja dla Raspberry Pi 5 [8]. Po restarcie czujniki widać w katalogu `/sys/bus/w1/devices/`.

Dane trafiają do Zabbixa przez element typu trapper, a nie przez odpytywanie. Skrypt z fiszki Skrypt Pythona czyta czujniki i co minutę wysyła wartości do `mon01` programem `zabbix_sender` [10]. Host w Zabbixie nazywa się tak samo jak `NAZWA_HOSTA` w skrypcie, czyli `czujnik-rpi01`.

Integracja z monitoringiem



Rysunek 7. Droga danych do Zabbixa: Zabbix sam pyta płytkę ESP32 przez HTTP, a malinka wysyła wartości przez zabbix_sender. Oba źródła spotykają się w wyzwalaczach na mon01 (kliknięcie otwiera pełny rozmiar)

Dla ESP32 wybieramy odpytywanie. Zabbix co minutę pyta płytkę o wartości przez serwer HTTP ESPHome, a odpowiedź JSON ma pole `value`, które element wyciąga ścieżką JSONPath [6]. Dzięki temu na płytce nie trzeba żadnych dodatków, a gdy płytka milknie, Zabbix to widzi.

Host i makra w Zabbixie

> **Pojęcie: element HTTP agent.** Element danych, który sam wysyła zapytanie HTTP pod wskazany adres i przetwarza odpowiedź. Typ elementu opisuje strona o pojęciach Zabbixa, a w wersji 7.0 pozwala na uwierzytelnianie podstawowe i przetwarzanie wstępne JSONPath [13].

Ścieżka w interfejsie, w której tworzymy host:

```
Data collection → Hosts → Create host
```

Host nazywamy `esp02`, dodajemy go do nowej grupy `Czujniki środowiska` i nie dodajemy żadnego interfejsu agenta, bo na płytce nie działa agent Zabbixa. Na hoście ustawiamy makra z tabeli.

Makro	Wartość	Po co
`{$CZUJNIK.ADRES}`	`10.20.10.71`	adres płytki ESP32
`{$CZUJNIK.UZYTKOWNIK}`	nazwa użytkownika z sekcji `web_server`	uwierzytelnianie do serwera HTTP
`{$CZUJNIK.HASLO}`	hasło z sekcji `web_server`, typ Secret text	hasło nie jest widoczne w interfejsie
`{$TEMP.WLOT.OSTRZEZENIE}`	`27`	górna granica zalecana (°C)
`{$TEMP.WLOT.KRYTYCZNY}`	`32`	górna granica dopuszczalna klasy A1 (°C)

Progi wynikają z zaleceń ASHRAE: 27 °C to górna granica zalecanego zakresu wlotu 18 do 27 °C, a 32 °C to górna granica zakresu dopuszczalnego klasy A1 [11][14]. Jeśli karta katalogowa serwera przewiduje klasę A2 albo szerszą, próg krytyczny podnosimy zgodnie z nią.

Elementy danych

Dla każdej sondy tworzymy jeden element. Zasady są takie same, zmienia się tylko nazwa i końcówka adresu.

Nazwa	Typ	Klucz	URL	Przetwarzanie wstępne
Wlot, dół	HTTP agent	`temp.wlot.dol`	`http://{$CZUJNIK.ADRES}/sensor/wlot_dol`	JSONPath `$.value`
Wlot, środek	HTTP agent	`temp.wlot.srodek`	`http://{$CZUJNIK.ADRES}/sensor/wlot_srodek`	JSONPath `$.value`
Wlot, góra	HTTP agent	`temp.wlot.gora`	`http://{$CZUJNIK.ADRES}/sensor/wlot_gora`	JSONPath `$.value`

Nazwa	Typ	Klucz	URL	Przetwarzanie wstępne
Gorący korytarz	HTTP agent	`temp.korytarz.goracy`	`http://{$CZUJNIK.ADRES}/sensor/korytarz_goracy`	JSONPath `$.value`

W każdym elemencie ustawiamy: typ informacji liczba zmiennoprzecinkowa, jednostka `°C`, interwał 1m, uwierzytelnianie podstawowe z makrami użytkownika i hasła, a historię i trendy według zasad bazy. Wartość `NaN` z uszkodzonej sondy nie przejdzie przez konwersję na liczbę, więc element stanie się niewspierany, a wtedy zadziała alarm braku danych.

Dla wariantu z malinką tworzymy host `czujnik-rpi01` z trzema elementami typu Zabbix trapper o kluczach `temp.wlot.dol`, `temp.wlot.srodek` i `temp.wlot.gora`. W polu Allowed hosts wpisujemy adres Raspberry Pi. Wartości wysyła skrypt, a nazwa hosta w wywołaniu musi być identyczna z nazwą w Zabbixie [10].

Wyzwalacze

Wyrażenia zapisujemy w składni Zabbixa 7.0, w której nazwę hosta i klucz podajemy w ukośnikach [15]. Próg ostrzegawczy sprawdzamy na wartości minimalnej z pięciu minut, czyli alarm zgłaszamy dopiero wtedy, gdy każdy odczyt w tym czasie jest powyżej progu.

Ostrzeżenie o wysokiej temperaturze na wlocie (poziom Warning):

```
min(/esp02/temp.wlot.gora,5m)>{$TEMP.WLOT.OSTRZEZENIE}
or min(/esp02/temp.wlot.srodek,5m)>{$TEMP.WLOT.OSTRZEZENIE}
or min(/esp02/temp.wlot.dol,5m)>{$TEMP.WLOT.OSTRZEZENIE}
```

Wyrażenie powrotu, które daje histerezę jednego stopnia, żeby alarm nie migał na granicy:

```
max(/esp02/temp.wlot.gora,5m)<{$TEMP.WLOT.OSTRZEZENIE}-1
and max(/esp02/temp.wlot.srodek,5m)<{$TEMP.WLOT.OSTRZEZENIE}-1
and max(/esp02/temp.wlot.dol,5m)<{$TEMP.WLOT.OSTRZEZENIE}-1
```

Alarm krytyczny, wyższy próg i krótszy czas (poziom Disaster):

```
min(/esp02/temp.wlot.gora,3m)>{$TEMP.WLOT.KRYTYCZNY}
or min(/esp02/temp.wlot.srodek,3m)>{$TEMP.WLOT.KRYTYCZNY}
or min(/esp02/temp.wlot.dol,3m)>{$TEMP.WLOT.KRYTYCZNY}
```

Brak danych z któregokolwiek czujnika (poziom High):

```
nodata(/esp02/temp.wlot.gora,5m)=1
or nodata(/esp02/temp.wlot.srodek,5m)=1
or nodata(/esp02/temp.wlot.dol,5m)=1
or nodata(/esp02/temp.korytarz.goracy,5m)=1
```

Funkcje `min`, `max` i `nodata` oraz wyrażenie powrotu to standardowe elementy składni wyzwalaczy Zabbixa [15]. Sondy w gorącym korytarzu nie obejmujemy alarmem temperatury, bo wylot jest z natury cieplejszy. Zbieramy z niej historię, żeby widzieć różnicę temperatur między korytarzami.

Akcja: poczta i SMS

Ścieżka w interfejsie, w której tworzymy akcję:

```
Alerts → Actions → Trigger actions → Create action
```

Dla poziomów od Warning w górę wysyłamy wiadomość pocztą do grupy użytkowników IT. Dla poziomu Disaster dodajemy drugą operację, czyli SMS przez typ nośnika SMS z modemem albo przez webhook do bramki SMS. Zasady łączenia zdarzenia z reakcją opisuje strona o pojęciach Zabbixa.

Test progów robimy bez grzania szafy: tymczasowo obniżamy makra `{$TEMP.WLOT.OSTRZEZENIE}` i `{$TEMP.WLOT.KRYTYCZNY}` do kilku stopni powyżej bieżącej temperatury, a sondę ogrzewamy w dłoni. Po sprawdzeniu, że alarmy i wiadomości dochodzą, przywracamy 27 i 32.

Bezpieczeństwo i ograniczenia

Własny czujnik nie ma świadectwa wzorcowania ani deklaracji zgodności, a jego dokładność wynika wyłącznie z karty katalogowej DS18B20 i z naszej kalibracji w kąpieli lodowej. Dlatego traktujemy go jako uzupełnienie istniejącego monitoringu, a nie jego zastąpienie.

- Gwarancja i ubezpieczenie: urządzenie własnej roboty w szafie z serwerami może być sprzeczne z warunkami gwarancji sprzętu albo polisy. Przed montażem uzgadniamy je z zarządcą serwerowni i ubezpieczycielem, a prace wykonujemy tylko na niskim napięciu i z gotowym zasilaczem z oznaczeniem CE.
- Chłodnia w Swarzędzu: dla żywności głęboko mrożonej wymagane są rejestratory zgodne z EN 12830, EN 13485 i EN 13486, a zapisy przechowujemy co najmniej rok, więc zestaw DIY ich nie zastępuje [16][17].
- Sygnalizacja pożarowa: czujki temperatury w systemie ppoż. podlegają serii norm EN 54 i odbiorowi, a nasz zestaw w żaden sposób ich nie zastępuje [18].
- Czujnik przy wylocie zasilacza serwera pokazuje temperaturę wylotu, nie wlotu. Montujemy sondy tylko na froncie szafy.

- Samonagrzewanie: sam DS18B20 grzeje się pomijalnie, ale płytka ESP32 i zasilacz USB nie, więc sondy prowadzimy kablem z dala od nich.
- Sieć: serwer HTTP ESPHome domyślnie używa uwierzytelniania podstawowego, a dokumentacja zapowiada zmianę domyślnego schematu na digest w wersji 2027.1.0 [6]. Na `fw01` ograniczamy dostęp do portu 80 płytki do adresu `mon01` i nie wystawiamy go poza serwery.
- Wi-Fi w serwerowni bywa niestabilne za metalowymi szafami. Jeśli odczyty wypadają, przechodzimy na wersję z przewodową siecią Ethernet, którą ESPHome również obsługuje.
- Aktualizacje: płytkę aktualizujemy przez sieć według sekcji `ota`, a przed aktualizacją sprawdzamy, że nie wypada okno zmian w listopadzie i styczniu.

Częste błędy

Objaw	Przyczyna	Co robimy
W dzienniku `Found no devices!`	brak rezystora podciągającego, zły pin albo odwrócone zasilanie	sprawdzamy 4,7 kΩ między 3V3 i linią danych, pin GPIO4 i kolory przewodów [3]
Sonda mocno się nagrzewa, a odczytu brak	odwrócone zasilanie	odłączamy zasilanie i wymieniamy sondę
Odczyt zawsze 85,0 °C	wartość startowa układu albo spadek napięcia	sprawdzamy zasilanie, dodajemy odrzucanie 85,0, jak w skrypcie [9]
Odczyty skaczą albo część brakuje	zbyt długi kabel, słabe styki	skracamy kabel, lutujemy zamiast skręcać, sprawdzamy rezystor [3]
Płytka restartuje się co 15 minut	sekcja `api` bez klienta i bez `reboot_timeout: 0s`	dopisujemy `reboot_timeout: 0s` albo przechodzimy na MQTT [5]
Element w Zabbixie niewspierany	błędny URL, hasło albo brak JSONPath	sprawdzamy `curl` z tego samego adresu i przetwarzanie wstępne `$.value`
`zabbix_sender` zgłasza `failed: 1`	nazwa hosta inna niż w Zabbixie, brak elementu trapper albo zły Allowed hosts	porównujemy nazwę hosta, klucz i adres nadawcy [10]
Odczyt w kąpieli lodowej 1 do 2 °C zamiast 0	lód bez wody, sonda przy ściance albo dnie	robimy kąpiel od nowa: pokruszony lód, woda, mieszanie, sonda w środku [12]
Czujnik w szafie pokazuje o kilka stopni za dużo	sonda przy wylocie zasilacza albo dotyka metalu	przenosimy na front i mocujemy w uchwycie z tworzywa

□□ Ćwiczenia

1. Ile rezystorów 4,7 kΩ potrzeba przy czterech sondach DS18B20 na jednej magistrali?

Odpowiedź

Jeden, na całą magistralę. Łączymy go między zasilaniem 3,3 V a linią danych, przy płytce [3].

1. Dlaczego sondy zasilamy z pinu 3V3, a nie z 5 V?

Odpowiedź

Piny GPIO płytki ESP32 pracują na 3,3 V, a linia danych jest podciągana do napięcia zasilania sond. DS18B20 wystarcza zasilanie od 3,0 do 5,5 V, więc 3,3 V jest bezpieczne dla obu stron [9].

1. Jakie wyrażenie wyzwalacza zgłosi brak danych z sondy `temp.wlot.gora` po pięciu minutach?

Odpowiedź

`nodata(/esp02/temp.wlot.gora,5m)=1`. Wyzwalacz przejdzie w stan problemu, gdy przez pięć minut nie przyjdzie żadna wartość, także wtedy, gdy element stał się niewspierany [15].

1. Sonda w kąpieli lodowej pokazuje 0,4 °C. Wymieniamy ją czy korygujemy?

Odpowiedź

Nie wymieniamy. Producent gwarantuje ±0,5 °C w tym zakresie, więc 0,4 °C mieści się w tolerancji [9]. Jeśli chcemy większej zgodności, dopisujemy poprawkę `offset: -0.4` w sekcji `filters` tej sondy, ale tylko po sprawdzeniu drugiego punktu z termometrem wzorcowym.

1. Dlaczego czwarta sonda w gorącym korytarzu nie ma własnego alarmu temperatury?

Odpowiedź

Gorący korytarz jest z natury cieplejszy, bo zbiera powietrze po przejściu przez serwery. Zalecenia ASHRAE dotyczą wlotu, więc alarmy ustawiamy na trzech sondach przy froncie, a czwartą zbieramy tylko do historii i trendów [11][14].

Źródła:

1. PyPI, esphome (wersja 2026.9.1, wymagany Python od 3.12)
2. ESPHome, Native API Component

3. ESPHome, 1-Wire Bus oraz 1-Wire przez GPIO
4. ESPHome, Dallas Temperature Sensor
5. ESPHome, MQTT Client Component
6. ESPHome, Web Server Component oraz Web Server API
7. Raspberry Pi, config.txt
8. Raspberry Pi, opis nakładek (overlays README)
9. Analog Devices, karta katalogowa DS18B20
10. Zabbix 7.0, zabbix_sender oraz strona podręcznika zabbix_sender
11. Upsite, Data Center Temperature Sensor Location
12. ThermoWorks, How to Create a Properly Made Ice Bath
13. Zabbix 7.0, element HTTP agent
14. ASHRAE, Thermal Guidelines, 5th edition, karta referencyjna
15. Zabbix 7.0, wyrażenia wyzwalaczy
16. Rozporządzenie Komisji (WE) nr 37/2005
17. EVS, EN 12830:2018
18. Wikipedia, EN 54

Revision #19
Created 2026-10-03 08:37:47 UTC by Tomasz Konieczny
Updated 2026-10-26 14:18:40 UTC by Tomasz Konieczny