ESPHome i urządzenia: adopcja esp01 i esp02, API albo MQTT, nazwy encji

◫◫ Poziom: średnio

◫◫ Dotyczy: ESPHome 2026.9.1, Home Assistant 2026.9, ESP32 z Ethernetem i PoE, Mosquitto

▯ Czas: ~60 min

Pokazujemy, jak płytki ESP32 `esp01` i `esp02` z serwerowni stają się urządzeniami w Home Assistant: skąd bierzemy ESPHome, czym różni się natywne API od MQTT i jak nazywamy encje, żeby po roku wiedzieć, co jest czym. Wracamy tu przy dodawaniu płytki, przy zmianie sposobu przekazywania danych i przy porządkowaniu nazw. Komendy ESPHome wykonujemy na stacji administratora z Pythonem 3.12 albo nowszym.

◫◫ Skróty użyte na tej stronie (22)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
ASCII	American Standard Code for Information Interchange: podstawowy zestaw znaków (litery, cyfry) używany w komputerach
BMS	Building Management System: system zarządzania budynkiem, który zbiera odczyty z czujników (temperatura, wilgotność) i steruje klimatyzacją oraz wentylacją
CFD	Computational Fluid Dynamics: komputerowe modelowanie przepływu powietrza, np. w serwerowni, żeby znaleźć gorące punkty
DCIM	Data Center Infrastructure Management: dokumentacja tego, co fizycznie stoi w serwerowni, czyli szafy, urządzenia i okablowanie
DIN	Deutsches Institut für Normung: niemiecki instytut normalizacyjny; szyna DIN to standardowa szyna montażowa aparatów w rozdzielnicy
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

Skrót	Co znaczy
ESP32	Tani mikrokontroler z Wi-Fi i Bluetooth firmy Espressif: malutki komputer, z którego budujemy własne czujniki sieciowe
ESPHome	Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć
HTTP	Hypertext Transfer Protocol: protokół stron WWW
I2C	Inter-Integrated Circuit: prosta dwuprzewodowa magistrala, którą mały układ (np. czujnik) rozmawia z mikrokontrolerem
mDNS	Multicast DNS: sposób, w którym urządzenia w jednej sieci lokalnej ogłaszają swoją nazwę (np. esp01.local) zapytaniami na adres multicast, bez serwera DNS; nie przechodzi między podsieciami
Mosquitto	popularny serwer pośredniczący (broker) MQTT, który przekazuje wiadomości między urządzeniami a Home Assistant
MQTT	Message Queuing Telemetry Transport: lekki protokół, w którym urządzenia wysyłają krótkie wiadomości na nazwane tematy przez serwer pośredniczący
PoE	Power over Ethernet: zasilanie urządzenia tym samym kablem sieciowym, którym płyną dane
REST	sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu
SHT3x	rodzina cyfrowych czujników temperatury i wilgotności na magistrali I2C, często stosowana z ESP32
SNMP	Simple Network Management Protocol: protokół odczytu stanu urządzeń sieciowych
USB	Universal Serial Bus: standard portów i urządzeń podłączanych do komputera, tu pendrive'y i inne nośniki wymienne
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach
Zigbee	Radiowy protokół sieci domowych o małym poborze prądu, używany w czujkach, czujnikach i przełącznikach na baterie

Fiszka

◫ ZMIENIA SYSTEM

Instalacja ESPHome na stacji administratora: pipx install

```
pipx install esphome
```

ESPHome buduje program dla płytki z pliku YAML. Wersja 2026.9.1 z 29 września 2026 wymaga Pythona 3.12 albo nowszego, ale poniżej 3.15 [1].

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 · **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 to 32 losowe bajty zapisane w base64 [2]. Każda płytka dostaje własny klucz, a wynik zapisujemy w Vaultwardenie na `hasla.zielona.example`.

API = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą ·**openssl** = zestaw narzędzi do certyfikatów i szyfrowania

PLIK KONFIGURACYJNY

Plik konfiguracyjny płytki: esp01.yaml

```
substitutions:
  nazwa: esp01
  nazwa_przyjazna: "esp01"
  adres: 10.20.10.70
  api_klucz: "emllbG9uYS1wcnp5a2xhZC1rbHVjei1hcGktMzJidCE="
  ota_haslo: "HasloOta"
  web_user: "nazwa_usera"
  web_haslo: "HasloWebServera"

esphome:
  name: ${nazwa}
  friendly_name: ${nazwa_przyjazna}
  area: Serwerownia

esp32:
  board: esp32-poe
  framework:
    type: esp-idf
```

```
logger:

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: ${adres}
    gateway: 10.20.10.1
    subnet: 255.255.255.0
    dns1: 10.20.10.11

# Home Assistant jest klientem API, ale Zabbix czyta płytkę przez web_server.
# reboot_timeout: 0s wyłącza restart co 15 minut, gdy chwilowo nie ma klienta API.
api:
  encryption:
    key: ${api_klucz}
  reboot_timeout: 0s

ota:
  - platform: esphome
    password: ${ota_haslo}

web_server:
  port: 80
  auth:
    username: ${web_user}
    password: ${web_haslo}

one_wire:
  - platform: gpio
    pin: GPIO4

i2c:
```

```
  sda: GPIO13
  scl: GPIO16
  scan: true

sensor:
  - platform: dallas_temp
    name: "Wlot R1 dol"
    address: 0x8c0000046e9b4f28
    resolution: 12
    update_interval: 30s
  - platform: dallas_temp
    name: "Wlot R1 srodek"
    address: 0x3d0000046e9c1a28
    resolution: 12
    update_interval: 30s
  - platform: dallas_temp
    name: "Wlot R1 gora"
    address: 0xe20000046e9d7b28
    resolution: 12
    update_interval: 30s
  - platform: sht3xd
    address: 0x44
    update_interval: 30s
    temperature:
      name: "Zimny korytarz temperatura"
    humidity:
      name: "Zimny korytarz wilgotnosc"
```

Plik opisuje płytkę z Ethernetem i zasilaniem PoE: blok `ethernet` jest przykładem z dokumentacji dla płytki Olimex ESP32-POE, a statyczny adres ustawia podblok `manual_ip` [3]. Numery pinów ustalamy według opisu własnej płytki, a adresy sond DS18B20 poznajemy z dziennika, jak na stronie Termometry w praktyce: wybór, montaż i własny czujnik (DIY). Adresy sond i klucz API w pliku są przykładowe: klucza z tej strony nie używamy w produkcji. Nazwy czujników są bez polskich liter i bez znaków specjalnych, bo wchodzą do adresów REST, z których korzysta Zabbix.

Gotowy plik sprawdzamy poleceniem z następnej fiszki. Ten plik przeszedł `esphome config` w wersji 2026.9.1. Hasła w prawdziwym pliku trzymamy w `secrets.yaml` i zapisujemy je jako `!secret nazwa`, a w repozytorium zostaje tylko `secrets.yaml.przyklad`.

PoE = Power over Ethernet: zasilanie urządzenia tym samym kablem sieciowym, którym płyną dane · **DS18B20** = Cyfrowy czujnik temperatury z magistralą 1-Wire, z fabrycznym 64-bitowym adresem, popularny w układach domowych i małych serwerowniach · **DIY** = Do It Yourself: zrób to sam, czyli własnoręcznie zbudowany czujnik zamiast gotowego urządzenia · **API** = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **REST** = sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu · **LAN8720** = układ scalony (PHY) obsługujący Ethernet 10/100 w płytkach ESP32 z kablem sieciowym, np. Olimex ESP32-POE; w ESPHome zapisujemy go jako type: LAN8720

TYLKO ODCZYT

Sprawdzenie pliku: esphome config

```
esphome config esp01.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

Pierwsze wgranie programu przez USB: esphome run

```
esphome run esp01.yaml
```

Pierwsze wgranie robimy kablem USB, a ESPHome pyta o port. Kolejne aktualizacje idą przez sieć, według sekcji `ota`.

USB = Universal Serial Bus: standard portów i urządzeń podłączanych do komputera, tu pendrive'y i inne nośniki wymienne · **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

Podgląd dziennika płytki przez sieć: esphome logs

```
esphome logs esp01.yaml
```

Dziennik pokazuje adresy znalezionych sond, połączenie Ethernet i klientów API.

API = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **esphome** = ESPHome: buduje i wgrywa oprogramowanie na ESP32 z pliku YAML, pokazuje logi urządzenia

TYLKO ODCZYT

Odczyt wartości przez REST: curl

```
curl --user nazwa_usera http://NazwaPlytki/sensor/wlot_r1_gora
```

Odpowiedź ma pole `value`, które Zabbix wyciąga w elemencie HTTP agent [4]. Identyfikatory encji sprawdzamy w strumieniu zdarzeń płytki, tak jak opisuje strona o higrometrach.

REST = sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu · **HTTP** = Hypertext Transfer Protocol: protokół stron WWW · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

TYLKO ODCZYT

Dodanie płytki w Home Assistant: Devices & services

```
Settings → Devices & services → ESPHome
```

Płytka z mDNS pojawia się jako Discovered. Gdy się nie pojawia, wybieramy dodanie ręczne i wpisujemy adres (dla `esp01` 10.20.10.70), port 6053 i klucz szyfrujący z pliku [5]. Opcję zezwalającą urządzeniu na wywoływanie akcji w Home Assistant zostawiamy wyłączoną: dokumentacja każe włączać ją tylko dla urządzeń, którym ufamy [5].

ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: ekran Settings → Devices & services → ESPHome w `ha01` po dodaniu `esp01`, widoczne urządzenie z obszarem Serwerownia i pięcioma encjami.

mDNS = Multicast DNS: sposób, w którym urządzenia w jednej sieci lokalnej ogłaszają swoją nazwę (np. esp01.local) zapytaniami na adres multicast, bez serwera DNS; nie przechodzi między podsieciami · **ESPHome** = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć

TYLKO ODCZYT

Przypisanie urządzenia do obszaru: Settings

```
Settings → Areas, labels & zones → Areas → Serwerownia
```

Obszar nadajemy raz, a encje urządzenia dziedziczą go. W pliku ESPHome pole `area` w sekcji `esphome` podpowiada obszar już przy dodaniu płytki.

ESPHome = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć

PLIK KONFIGURACYJNY

Zamiana API na MQTT: sekcja mqtt

```
mqtt:
  broker: 10.20.10.54
  username: "nazwa_usera"
  password: "HasloMqtt"
  topic_prefix: "zielona/serwerownia/esp01"
  discovery: false
```

Blok wpisujemy w miejsce sekcji `api`. Gdy zostawimy oba, a Home Assistant łączy się przez API, odkrywanie MQTT tworzy duplikaty encji, dlatego ustawiamy `discovery: false` [6]. Gdy usuwamy `api`, a zostaje samo MQTT, bez `reboot_timeout: 0s` płytka restartuje się co 15 minut [6].

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

Podsłuch tematów płytki: mosquitto_sub

```
mosquitto_sub -h AdresBrokera -u nazwa_usera -P HasloMqtt -t
'zielona/serwerownia/esp01/#' -v
```

Polecenie pokazuje wiadomości płytki na brokerze, w tym `online` i `offline` na temacie `status` [6].

mosquitto_sub = Mosquitto subscribe: subskrybuje tematy MQTT w brokerze i wypisuje przychodzące wiadomości

PLIK KONFIGURACYJNY

Plik Compose ESPHome jako kontener: /docker/compose/esphome/compose.yaml

```
services:
  esphome:
    container_name: esphome
    image: ghcr.io/esphome/esphome:2026.9.1
    restart: unless-stopped
    network_mode: host
    environment:
      TZ: Europe/Warsaw
    volumes:
      - /docker/apps/esphome:/config
```

Wariant dla Home Assistant w kontenerze, który nie ma aplikacji. Obraz `ghcr.io/esphome/esphome` uruchamia interfejs ESPHome Device Builder na porcie 6052, a tryb hosta pozwala poprawnie pokazywać stan urządzeń [7]. Numer wersji obrazu zgodny z pakietem 2026.9.1 sprawdzamy w rejestrze przed pierwszym użyciem, bo lista tagów jest tam zmienna.

> W naszym środowisku dane aplikacji trzymamy w `/docker/apps/<nazwa>`, a plik Compose w `/docker/compose/<nazwa>/compose.yaml`. To konwencja, nie wymóg Dockera: można ją zmienić, byle w całej firmie była jedna. Katalog `/docker` to osobny wolumen `vgDocker/lvDocker`, żeby pełny dysk aplikacji nie zatrzymał systemu.

ESPHome = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć

Z życia wzięte

Z ŻYCIA WZIĘTE

Poniedziałek, 9:15. Tomek dodaje `esp01` do Home Assistant i po chwili dodaje `esp02`, a na liście encji ma dwie pozycje „Temperatura”, dwie „Wilgotnosc” i jedną „Wlot R1 dol”, bez wskazania, która płytka jest która. Ola pyta, czy wie, który „Temperatura 2” stoi w zimnym korytarzu północnym, a Tomek przyznaje, że sprawdza to po wartościach. Ustalają wtedy

> zasadę nazw: urządzenie, miejsce, wielkość, wszystko ASCII, a po polsku tylko to, co widać na panelu.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: natywne API ESPHome.** Protokół, w którym Home Assistant otwiera i trzyma stałe połączenie z płytką, zwykle szyfrowane kluczem, i dostaje zmiany od razu, bez odpytywania [5][2]. Port domyślny to 6053, a płytka ESP32 obsługuje jednocześnie pięć połączeń [2].

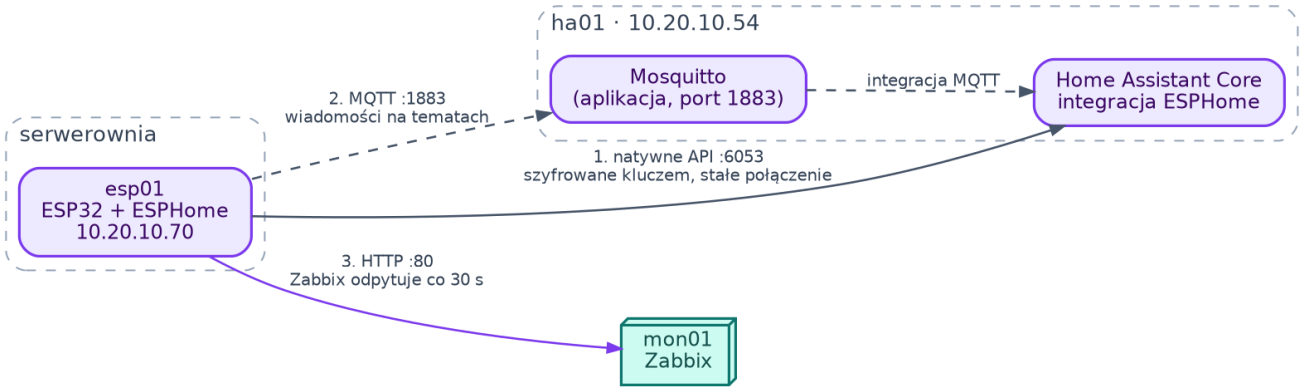
> **Pojęcie: MQTT i broker.** Protokół, w którym urządzenia publikują krótkie wiadomości na nazwanych tematach, a serwer pośredniczący, broker (u nas Mosquitto), przekazuje je subskrybentom. Płytka i Home Assistant nie znają się wzajemnie, znają tylko broker.

Gdzie uruchamiamy ESPHome

Miejsce	Kiedy	Uwagi
Stacja administratora (`pipx install esphome`)	pierwsze wgranie przez USB, drobne poprawki	pełny dostęp do portu USB, pliki w repozytorium Git
Aplikacja ESPHome Device Builder w Home Assistant OS	zespół pracuje z interfejsu, jedno miejsce na pliki	tylko Home Assistant OS, bo to aplikacja Supervisora
Osobny kontener na `docker01`	Home Assistant w kontenerze albo kompilacja poza `ha01`	kompilacja obciąża `docker01`, więc nie w godzinach pracy sklepu B2B

Natywne API kontra MQTT

Dla `esp01` i `esp02` wybieramy natywne API, a MQTT zostawiamy na szczególne przypadki. Zabbix czyta płytki osobno przez HTTP, więc ani API, ani MQTT nie są drogą alarmu.



Rysunek 1. Trzy drogi danych z płytki esp01: natywne API do Home Assistant (stałe połączenie, szyfrowane), MQTT przez Mosquitto (opcja, przerywana linia) i HTTP do Zabbixa na mon01. Warto zauważyć, że trzecia droga nie zależy od pozostałych dwóch. (kliknięcie otwiera pełny rozmiar)

Cecha	Natywne API	MQTT przez Mosquitto
Połączenie	stałe, Home Assistant jest klientem płytki	płytka i Home Assistant łączą się z brokerem
Szyfrowanie	klucz 32 bajtów w pliku	zależy od ustawień brokera (TLS, hasła)
Odkrywanie urządzeń	mDNS albo adres ręcznie	odkrywanie MQTT (tematy `homeassistant/...`) [8]
Dodatkowy element	brak	broker, który też trzeba utrzymać i monitorować
Ile systemów czyta jednocześnie	kilka (u ESP32 pięć połączeń) [2]	dowolnie wielu subskrybentów brokera
Kiedy wybieramy	Home Assistant jest głównym odbiorcą	wiele systemów czyta te same dane albo urządzenie nie ma API

Dokumentacja Home Assistant zaznacza, że płytki ESPHome zwykle używają natywnego API, a MQTT jest opcją [8]. Kto chce MQTT w Home Assistant, instaluje aplikację Mosquitto broker, która sama tworzy bezpieczne konto dla integracji [8].

Integracje, urządzenia, encje, obszary

> **Pojęcie: integracja, urządzenie, encja.** Integracja to połączenie Home Assistant z systemem (ESPHome, MQTT). Urządzenie to fizyczna rzecz, na przykład płytka `esp01`. Encja to jedna wielkość, którą urządzenie udostępnia: czujnik temperatury, wilgotności

albo przełącznik.

Pojęcie: obszar. Pomieszczenie lub miejsce, do którego przypisujemy urządzenia i encje, na przykład Serwerownia albo Salon. Obszary pozwalają zbierać karty na panelach i pisać automatyzacje dla całego pomieszczenia.

Każda encja ma identyfikator w postaci `domena.nazwa`, na przykład `sensor.esp01_wlot_r1_gora` [9]. Nazwa wyświetlana to druga rzecz: widzimy ją na panelu i może być po polsku. Od wydania 2026.8 identyfikatory encji można zmieniać ręcznie [12].

Konwencja nazw w Zielonej

Element	Zasada	Przykład
Urządzenie ESPHome	`espNN` dla płytek czujników, `wejNN` dla modułów wejść, małe litery	`esp01`, `esp02`
Identyfikator encji	`domena.urzadzenie_miejsce_wielkosc`, ASCII, podkreślniki, bez polskich liter	`sensor.esp01_wlot_r1_gora`
Nazwa wyświetlana	po polsku, krótka, bez numeru urządzenia	„Wlot R1, góra"
Obszar	nazwa pomieszczenia po polsku	Serwerownia, Biuro IT
Pomocnik (helper)	prefiks miejsca, ASCII	`input_boolean.serwerownia_tryb_serwisowy`
Czujnik pochodny (szablon)	prefiks miejsca i znaczenie	`sensor.serwerownia_wlot_max`

Nazwy w pliku ESPHome zapisujemy bez polskich liter, bo na nich opiera się adres REST dla Zabbixa. Nazwy po polsku nadajemy w Home Assistant, w polu nazwy encji. Po dodaniu płytki sprawdzamy w Settings → Devices & services → Entities, jakie identyfikatory powstały, bo skład identyfikatora zależy od nazwy urządzenia i encji, i w razie różnic poprawiamy je ręcznie.

Co zmienia się dla esp02

Plik `esp02.yaml` ma te same sekcje, a zmieniają się wartości.

Pole	esp01	esp02
`nazwa` i `nazwa_przyjazna`	`esp01`	`esp02`
`adres`	10.20.10.70	10.20.10.71

Pole	esp01	esp02
`api_klucz`	własny klucz	inny własny klucz
Sondy DS18B20	wloty szaf R1 (dół, środek, góra)	wloty szaf według planu czujników, a gorący korytarz w osobnej sondzie
Czujnik SHT3x	zimny korytarz północny, H1	zimny korytarz południowy, H2

Adresy 10.20.10.70 i .71 to płytki `esp01` i `esp02` z mapy serwerowni i ze stron DIY w tej bazie.

W domu

Ta sama płytka z innym plikiem służy w domu jako czujnik w salonie. Zamiast bloku `ethernet` wpisujemy `wifi`, zamiast sond DS18B20 jeden czujnik SHT3x na magistrali I2C, a obszar ustawiamy na Salon.

```
substitutions:
  nazwa: esp-salon
  nazwa_przyjazna: "esp-salon"
  api_klucz: "emllbG9uYS1wcnp5a2xhZC1rbHVjei1hcGktMzJidCE="
  ota_haslo: "HasloOta"

esphome:
  name: ${nazwa}
  friendly_name: ${nazwa_przyjazna}
  area: Salon

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

logger:

wifi:
  ssid: "NazwaSieciWifi"
  password: "HasloSieciWifi"
  ap:
    ssid: "esp-salon"
```

```
    password: "HasloAwaryjne"

api:
  encryption:
    key: ${api_klucz}

ota:
  - platform: esphome
    password: ${ota_haslo}

i2c:
  sda: GPIO21
  scl: GPIO22

sensor:
  - platform: sht3xd
    address: 0x44
    update_interval: 30s
    temperature:
      name: "Temperatura"
    humidity:
      name: "Wilgotnosc"
  - platform: wifi_signal
    name: "Sila sygnalu Wi-Fi"
    update_interval: 60s
    entity_category: diagnostic
```

W domu zwykle nie ma Zabbixa, więc `web_server` i `reboot_timeout: 0s` nie są potrzebne, a restart co 15 minut przy braku klienta jest tu pożądaną samonaprawą. Zasada nazw zostaje ta sama: identyfikator `sensor.esp_salon_temperatura`, nazwa wyświetlana „Temperatura w salonie". Gdy w domu chcemy Zigbee albo Z-Wave, potrzebujemy klucza USB w mini PC, a nie na maszynie wirtualnej [10].

> □□ Więcej możliwości albo gotowe rozwiązanie od ręki (out of the box): sieciowy moduł czujników temperatury i wilgotności z PoE i odczytem przez SNMP albo HTTP, wejścia styków bezpotencjałowych w obudowie na szynę DIN oraz gotowe czujniki Zigbee z baterią dla domu. W dużych centrach danych stosuje się czujniki w każdej szafie (po trzy na wlocie), integrację z

> systemem DCIM i BMS oraz modelowanie przepływu powietrza (CFD).

Częste błędy

Objaw	Przyczyna	Co robimy
Płytka restartuje się co 15 minut	sekcja `api` bez klienta API	`reboot_timeout: 0s` w `api` albo przejście na MQTT [2][6]
Każda encja występuje dwa razy	jednocześnie API i odkrywanie MQTT	`discovery: false` w sekcji `mqtt` [6]
Home Assistant pyta o klucz i go odrzuca	klucz inny niż w pliku albo spacje	kopiujemy klucz z `api.encryption.key` pliku, który jest wgrany na płytkę [5]
Płytka nie pojawia się jako Discovered	mDNS nie dociera (inny VLAN, wyłączony mDNS)	dodajemy ręcznie po adresie IP, port 6053 [5]
Encje mają nazwy w rodzaju „Temperatura 2"	dwie płytki z tym samym polem `name` bez przedrostka urządzenia	nazywamy encje według konwencji i poprawiamy identyfikatory ręcznie
Sonda pokazuje 85,0 °C	wartość startowa układu DS18B20	sprawdzamy zasilanie i dodajemy filtr odrzucający 85,0, jak na stronie o termometrach
`esphome run` nie widzi portu USB	brak sterownika, kabel tylko do ładowania	inny kabel, a na Linuksie dostęp do `/dev/ttyUSB0`

□□ Ćwiczenia

1. Płytka `esp01` ma w pliku sekcję `api`, ale Home Assistant jest wyłączony na noc. Dlaczego Zabbix czasem widzi brak danych i jak temu zapobiec?

Odpowiedź

Płytka bez klienta API restartuje się po 15 minutach, a w czasie restartu `web_server` nie odpowiada Zabbixowi. Dodajemy `reboot_timeout: 0s` w sekcji `api` [2].

1. Jak nazywamy encję temperatury z sondy w dolnej części wlotu szafy R2 na `esp01`, jeżeli nazwa w pliku to „Wlot R2 dol"?

Odpowiedź

Identyfikator to `sensor.esp01_wlot_r2_dol`, a nazwę wyświetlaną ustawiamy w Home Assistant na „Wlot R2, dół". Identyfikatora nie tłumaczymy na polski, żeby nie mieć polskich liter w adresach i automatyzacjach.

1. Kiedy wybieramy MQTT zamiast natywnego API?

Odpowiedź

Gdy te same dane ma czytać kilka systemów naraz albo urządzenie nie obsługuje natywnego API. W pozostałych przypadkach wybieramy API, bo nie wymaga brokera. Przy MQTT ustawiamy `discovery: false`, jeśli Home Assistant ma już płytkę przez API [6][8].

1. Dlaczego pliku ESPHome nie zapisujemy z kluczem API z tej strony?

Odpowiedź

Klucz z podręcznika jest publiczny, więc każdy, kto go zna, mógłby połączyć się z płytką. Własny klucz generujemy poleceniem `openssl rand -base64 32` i trzymamy w Vaultwardenie [2].

Powiązane: Termometry w praktyce: wybór, montaż i własny czujnik (DIY) · Higrometry w praktyce: wybór, montaż i własny czujnik (DIY) · Serwerownia Zielonej: gdzie stoją czujniki i dokąd płyną alarmy · Pomiar energii na ESP32 (DIY)

Źródła:

1. PyPI, pakiet esphome (wersja 2026.9.1, Python od 3.12): pypi.org/project/esphome
2. ESPHome, Native API Component (port 6053, klucz, `reboot_timeout`, liczba połączeń): esphome.io/components/api
3. ESPHome, Ethernet Component (Olimex ESP32-POE, `manual_ip`): esphome.io/components/ethernet
4. ESPHome, Web Server Component: esphome.io/components/web_server
5. Home Assistant, integracja ESPHome: home-assistant.io/integrations/esphome
6. ESPHome, MQTT Client Component (`discovery`, restart bez API): esphome.io/components/mqtt
7. ESPHome, uruchamianie w Dockerze: esphome.io/guides/getting_started_command_line
8. Home Assistant, integracja MQTT (aplikacja Mosquitto, odkrywanie): home-assistant.io/integrations/mqtt

9. Home Assistant, obiekt stanu i identyfikatory encji: home-assistant.io/docs/configuration/state_object
10. Home Assistant, instalacja w Hyper-V (brak USB): home-assistant.io/installation/windows
11. ESPHome, repozytorium projektu: github.com/esphome/esphome
12. Home Assistant 2026.8, notatki wydania (ręczna zmiana identyfikatorów encji): home-assistant.io/blog/2026/08/05/release-20268

Revision #14
Created 2026-10-03 12:08:32 UTC by Tomasz Konieczny
Updated 2026-10-26 14:20:14 UTC by Tomasz Konieczny