Urządzenia, automatyzacje i panele

ESPHome, alarmy bez dublowania Zabbixa i panele.

- ESPHome i urządzenia: adopcja esp01 i esp02, API albo MQTT, nazwy encji
- Automatyzacje i powiadomienia: alarm temperatury w serwerowni bez dublowania Zabbixa
- Panele w Home Assistant: karty, panel serwerowni, panel domowy i uprawnienia

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].

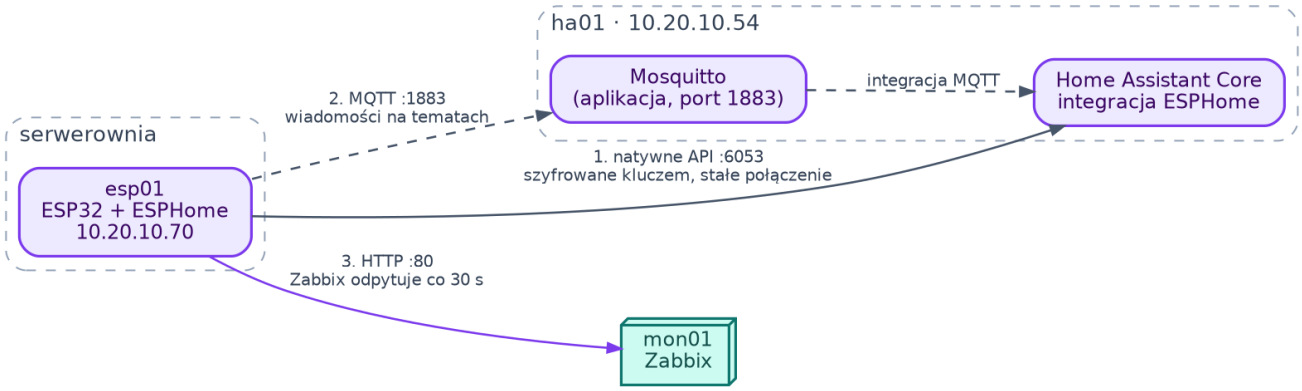
> **Pojęcie: MQTT i broker.** Protokół, w którym urządzenia publikują krótkie wiadomości na nazwanych tematach, a serwer pośredniczący, broker (u nas Mosquitto), przekazuje je subskrybentom. Płytka i Home Assistant nie znają się wzajemnie, znają tylko broker.

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

Automatyzacje i powiadomienia: alarm temperatury w serwerowni bez dublowania Zabbixa

◫◫ Poziom: średnio

◫◫ Dotyczy: Home Assistant 2026.9, ESPHome 2026.9.1, Zabbix na mon01, poczta MailCow na docker01

▯ Czas: ~75 min

Pokazujemy, jak z czujników `esp01` i `esp02` budujemy w Home Assistant automatyzacje i powiadomienia, a jednocześnie nie dublujemy alarmów, które należą do Zabbixa. Przykłady to alarm temperatury w serwerowni (firma) i zalanie w pralni (dom). Wracamy tu przy pisaniu nowej automatyzacji, przy podłączaniu poczty albo telefonu i przy pytaniu, kto w Zielonej odpowiada za który alarm.

◫◫ Skróty użyte na tej stronie (19)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
BMS	Building Management System: system zarządzania budynkiem, który zbiera odczyty z czujników (temperatura, wilgotność) i steruje klimatyzacją oraz wentylacją
DCIM	Data Center Infrastructure Management: dokumentacja tego, co fizycznie stoi w serwerowni, czyli szafy, urządzenia i okablowanie
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
GSM	Global System for Mobile Communications: standard sieci komórkowej, z którego korzysta modem wysyłający SMS

Skrót	Co znaczy
HTTP	Hypertext Transfer Protocol: protokół stron WWW
ICMP	Internet Control Message Protocol: protokół diagnostyczny sieci, którym działa między innymi ping
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
OS	Operating System: system operacyjny
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
SMS	Short Message Service: krótka wiadomość tekstowa na telefon, która dociera także bez internetu
SMTP	Simple Mail Transfer Protocol: protokół wysyłania poczty
STARTTLS	włączenie szyfrowania w trakcie już nawiązanej, nieszyfrowanej sesji pocztowej
UPS	Uninterruptible Power Supply: zasilacz awaryjny, podtrzymuje zasilanie urządzeń na czas zaniku napięcia z sieci
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Fiszka

□□ TYLKO ODCZYT

Włączenie pakietów w configuration.yaml: packages

```
homeassistant:
  packages: !include_dir_named packages
```

Pakiet to plik YAML w katalogu `packages`, który zbiera w jednym miejscu wszystko, co dotyczy jednego tematu: pomocniki, czujniki pochodne, automatyzacje i polecenia [1]. Plik `serwerownia.yaml` trzyma wszystko o serwerowni, a plik `dom.yaml` wszystko o domu. Blok `homeassistant:` jest w `configuration.yaml` jeden, więc dopisujemy do niego tylko klucz `packages`.

□□ **YAML** = YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

PLIK KONFIGURACYJNY

Pomocnik i czujnik pochodny: pakiet serwerownia.yaml

```
input_boolean:
  serwerownia_tryb_serwisowy:
    name: "Serwerownia: tryb serwisowy"
    icon: mdi:wrench

template:
  - sensor:
      - name: "Serwerownia wlot max"
        unique_id: serwerownia_wlot_max
        unit_of_measurement: "°C"
        device_class: temperature
        state_class: measurement
        state: >-
          {{ [
               states('sensor.esp01_wlot_r1_dol') | float,
               states('sensor.esp01_wlot_r1_srodek') | float,
               states('sensor.esp01_wlot_r1_gora') | float
             ] | max }}
        availability: >-
          {{ has_value('sensor.esp01_wlot_r1_dol')
             and has_value('sensor.esp01_wlot_r1_srodek')
             and has_value('sensor.esp01_wlot_r1_gora') }}
```

Przełącznik `serwerownia_tryb_serwisowy` wyłącza reakcje lokalne na czas prac w szafach. Czujnik `serwerownia_wlot_max` bierze najwyższy z trzech wlotów szafy R1. Gdy którakolwiek sonda jest niedostępna, blok `availability` robi z czujnika `unavailable`, zamiast liczyć maksimum z dwóch sond i udawać, że wszystko jest w porządku [2].

KLIKAMY W INTERFEJSIE

Sprawdzenie szablonu na żywo: Tools, Template

```
Settings → Tools → Template
```

W polu szablonu wpisujemy na przykład `{{ states('sensor.serwerownia_wlot_max') }}` oraz `{{ has_value('sensor.esp01_wlot_r1_gora') }}` i od razu widzimy wynik z prawdziwych stanów.

Szablon, który pokazuje dobrą liczbę w tym oknie, wkleja się do pliku bez poprawek. W wersjach starszych niż 2026.8 ta sama zakładka nazywała się Developer tools [17].

ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: okno Settings → Tools → Template w `ha01`, w lewym polu szablon z `states('sensor.serwerownia_wlot_max')`, w prawym wynik z temperaturą.

PLIK KONFIGURACYJNY

Polecenie wysyłania SMS przez bramkę: rest_command

```
rest_command:
  sms_bramka:
    url: "https://sms.zielona.example/api/wyslij"
    method: POST
    content_type: "application/json; charset=utf-8"
    headers:
      Authorization: "Bearer TokenBramkiSms"
    payload: '{"numer": "{{ numer }}", "tekst": "{{ tekst }}"}'
```

`rest_command` tworzy akcję `rest_command.sms_bramka`, którą wywołujemy z automatyzacji, podając `numer` i `tekst` [4]. Adres i format zapytania bramki SMS są przykładowe i zależą od dostawcy, więc dopasowujemy je do dokumentacji naszej bramki. Token wpisujemy przez `!secret`, a w repozytorium zostaje `secrets.yaml.przyklad`.

Strona integracji `sms` dla modemów GSM ma w dokumentacji status integracji usuniętej [18], więc SMS wysyłamy przez bramkę z interfejsem HTTP albo przez dostawcę z własną integracją. Listę integracji powiadomień sprawdzamy w dokumentacji naszej wersji [5].

SMS = Short Message Service: krótka wiadomość tekstowa na telefon, która dociera także bez internetu · **GSM** = Global System for Mobile Communications: standard sieci komórkowej, z którego korzysta modem wysyłający SMS · **HTTP** = Hypertext Transfer Protocol: protokół stron WWW

PLIK KONFIGURACYJNY

Automatyzacja: sygnalizator lokalny przy wysokiej temperaturze wlotu

```
automation:
  - id: serwerownia_sygnalizator_wlot
    alias: "Serwerownia: sygnalizator lokalny przy wysokiej temperaturze wlotu"
    description: "Reakcja lokalna. Alarm dla zespołu IT wysyła Zabbix, ta automatyzacja
nie wysyła poczty ani SMS."
    mode: restart
    triggers:
      - trigger: numeric_state
        entity_id: sensor.serwerownia_wlot_max
        above: 27
        for: "00:05:00"
        id: wysoka
      - trigger: numeric_state
        entity_id: sensor.serwerownia_wlot_max
        below: 25
        for: "00:05:00"
        id: norma
    conditions:
      - condition: state
        entity_id: input_boolean.serwerownia_tryb_serwisowy
        state: "off"
    actions:
      - choose:
          - conditions:
              - condition: trigger
                id: wysoka
            sequence:
              - action: switch.turn_on
                target:
                  entity_id: switch.serwerownia_sygnalizator
              - action: persistent_notification.create
                data:
                  notification_id: serwerownia_wlot_wysoki
                  title: "Serwerownia: wysoka temperatura wlotu"
                  message: >-
                    Najwyższy wlot: {{ states('sensor.serwerownia_wlot_max') }} °C
                    od 5 minut. Alarm i eskalacja są w Zabbixie na mon01.
          - conditions:
              - condition: trigger
```

```
      id: norma
    sequence:
      - action: switch.turn_off
        target:
          entity_id: switch.serwerownia_sygnalizator
      - action: persistent_notification.dismiss
        data:
          notification_id: serwerownia_wlot_wysoki
```

Wyzwalacz `numeric_state` uruchamia się w chwili przekroczenia progu, a nie przez cały czas, gdy wartość jest powyżej niego [6]. Opcja `for` wymaga, żeby stan trwał pięć minut, ale po restarcie Home Assistant albo przeładowaniu automatyzacji licznik zaczyna od zera [6]. Wyższy próg 27 °C i niższy 25 °C dają dwustopniową histerezę (zdjęcie alarmu dopiero 2 °C niżej), taką samą jak w Zabbixie.

Zasady dla każdej automatyzacji w pakiecie: stałe pole `id`, opis mówiący, co automatyzacja robi i czego nie robi, oraz identyfikatory wyzwalaczy (`wysoka`, `norma`), do których odwołuje się warunek `trigger` [7]. Przełącznik `switch.serwerownia_sygnalizator` to lampa przy drzwiach, podłączona do płytki ESPHome przekaźnikiem, i nie opisujemy tu jej budowy.

ESPHome = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **IT** = Information Technology: dział albo praca związana z komputerami i infrastrukturą firmy · **SMS** = Short Message Service: krótka wiadomość tekstowa na telefon, która dociera także bez internetu

PLIK KONFIGURACYJNY

Automatyzacja: zapasowy alarm, gdy Zabbix milczy

```
- id: serwerownia_zapasowy_alarm_gdy_zabbix_milczy
  alias: "Serwerownia: zapasowy alarm, gdy Zabbix milczy"
  description: "Działa tylko wtedy, gdy mon01 nie odpowiada na ping od 10 minut."
  mode: single
  triggers:
    - trigger: numeric_state
      entity_id: sensor.serwerownia_wlot_max
      above: 32
      for: "00:03:00"
  conditions:
    - condition: state
      entity_id: binary_sensor.mon01_ping
```

```
        state: "off"
        for: "00:10:00"
    actions:
      - action: notify.dyzur_it
        data:
          title: "Serwerownia: wlot powyżej 32 °C"
          message: "Zabbix na mon01 milczy, a temperatura wlotu to {{
states('sensor.serwerownia_wlot_max') }} °C."
          data:
            tag: serwerownia-wlot-krytyczny
            group: serwerownia
            push:
              interruption-level: time-sensitive
      - action: rest_command.sms_bramka
        data:
          numer: "+48500000000"
          tekst: "Zielona: serwerownia, wlot {{ states('sensor.serwerownia_wlot_max') }}
C, Zabbix milczy."
```

Ten fragment dopisujemy w pliku zaraz pod pierwszą automatyzacją, jako kolejny element listy pod tym samym kluczem `automation:`. Numer telefonu i adres bramki są przykładowe. Warunek `for: "00:10:00"` na stanie ping sprawia, że krótka przerwa na restart `mon01` nie wywołuje zapasowego alarmu.

□□ TYLKO ODCZYT

Czujnik dostępności mon01: integracja Ping

```
Settings → Devices & services → Add integration → Ping (ICMP)
```

W polu Host wpisujemy 10.20.10.51, a liczbę zapytań zostawiamy domyślną. Integracja tworzy czujnik binarny pokazujący, czy host odpowiada, a domyślnie pyta co 30 sekund [8]. Identyfikator encji zmieniamy ręcznie na `binary_sensor.mon01_ping`, żeby pasował do automatyzacji.

□□ PLIK KONFIGURACYJNY

Grupa telefonów dyżurnych: notify, platform group

```
notify:
  - platform: group
    name: "Dyzur IT"
    services:
      - action: mobile_app_telefon_dyzurny
      - action: mobile_app_telefon_drugiego_admina
```

Grupa powiadomień daje jedną akcję, `notify.dyzur_it`, która wysyła wiadomość na kilka telefonów. Zmianę osoby dyżurnej robimy w jednym miejscu, a nie w każdej automatyzacji [9]. Nazwy `mobile_app_...` zależą od nazw telefonów zarejestrowanych w aplikacji i sprawdzamy je w Tools → Actions.

IT = Information Technology: dział albo praca związana z komputerami i infrastrukturą firmy

TYLKO ODCZYT

Podłączenie poczty z MailCow: integracja SMTP

```
Settings → Devices & services → Add integration → SMTP
```

W kreatorze wpisujemy adres nadawcy `homeassistant@zielona.example`, hosta `poczta.zielona.example`, port 587, zabezpieczenie połączenia STARTTLS oraz nazwę użytkownika i hasło skrzynki [10]. Skrzynkę `homeassistant@zielona.example` zakładamy w MailCow, jak opisuje strona MailCow: codzienna praca z domenami, skrzynkami i spamem. W kreatorze dodajemy też pierwszego odbiorcę (`it@zielona.example`), a każdy odbiorca dostaje własną encję powiadomień [10].

ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: ekran integracji SMTP w `ha01` z dodanym odbiorcą `it@zielona.example` i encją powiadomień, której identyfikator wpisujemy potem w automatyzacjach.

SMTP = Simple Mail Transfer Protocol: protokół wysyłania poczty · **STARTTLS** = włączenie szyfrowania w trakcie już nawiązanej, nieszyfrowanej sesji pocztowej

PLIK KONFIGURACYJNY

Automatyzacja domowa: zalanie w pralni (pakiet dom.yaml)

```
automation:
  - id: dom_zalanie_pralnia
    alias: "Dom: zalanie w pralni"
    description: "Czujnik wody pod pralką: powiadomienie w aplikacji i poczta."
    mode: single
    triggers:
      - trigger: state
        entity_id: binary_sensor.esp_pralnia_zalanie
        to: "on"
        for: "00:00:10"
    actions:
      - action: notify.mobile_app_telefon_tomka
        data:
          title: "Dom: woda w pralni"
          message: "Czujnik pod pralką wykrył wodę o {{ now().strftime('%H:%M') }}."
          data:
            tag: dom-zalanie-pralnia
            push:
              interruption-level: time-sensitive
      - action: notify.send_message
        target:
          entity_id: notify.poczta_dom
        data:
          title: "Dom: woda w pralni"
          message: "Czujnik pod pralką wykrył wodę o {{ now().strftime('%H:%M') }}."
```

W domu Home Assistant jest jedynym systemem, więc sam wysyła alarm i nie ma tu drugiego właściciela. Opcja `for: "00:00:10"` odsiewa chwilowe zwarcia styku, a `tag` powoduje, że kolejne powiadomienie o tym samym zdarzeniu zastępuje poprzednie, zamiast się dokładać [11]. Poziom `time-sensitive` na iPhonie przebija tryb skupienia, a poziom `critical` ma osobną dokumentację, którą sprawdzamy przed użyciem [11].

TYLKO ODCZYT

Odczyt stanu encji przez REST API: curl

```
curl --silent --header "Authorization: Bearer $(cat PlikZTokenem)"
http://AdresHomeAssistant/api/states/NazwaEncji
```

Odpowiedź to dokument JSON z polami `entity_id`, `state`, `last_changed`, `last_updated` i `attributes` [12]. Token długoterminowy tworzymy na stronie profilu użytkownika, w sekcji Security, i widzimy go tylko raz. W Zielonej zakładamy do tego osobne konto `zabbix` bez uprawnień administratora i trzymamy token w Vaultwardenie oraz w makrze sekretnym Zabbixa. Konto bez uprawnień administratora nie dociera do ustawień, ale nie zakładamy, że token jest tylko do odczytu, więc traktujemy go jak hasło.

REST = sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu · **API** = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **JSON** = JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

KLIKAMY W INTERFEJSIE

Element HTTP agent w Zabbixie czytający Home Assistant: Items

```
Data collection → Hosts → ha01 → Items → Create item
```

Pola elementu zapisujemy w osobnym bloku:

```
Name: Wlot max z Home Assistant
Type: HTTP agent
Key: ha.serwerownia.wlot_max
URL: http://10.20.10.54/api/states/sensor.serwerownia_wlot_max
Request type: GET
Headers: Authorization: Bearer {$HA.TOKEN}
Type of information: Numeric (float)
Update interval: 1m
Preprocessing: JSONPath, parametr $.state
```

Makro `{$HA.TOKEN}` ma typ Secret text, a makra, także sekretne, można wpisywać w nagłówkach elementu [13]. Gdy encja jest `unavailable`, wartość nie przechodzi na liczbę i element staje się nieobsługiwany, co zgłasza wyzwalacz oparty na `nodata`.

HTTP = Hypertext Transfer Protocol: protokół stron WWW · **HA** = High Availability: wysoka dostępność, czyli działanie mimo awarii jednego elementu

TYLKO ODCZYT

Pilnowanie ha01 przez Zabbixa: ping i port

```
Key: icmpping[10.20.10.54]
Key: net.tcp.service[http,10.20.10.54,80]
```

Oba elementy dodajemy na hoście `ha01` jako proste sprawdzenia (Simple check). Pierwszy mówi, czy maszyna odpowiada, drugi, czy interfejs WWW nasłuchuje. Port 80 dotyczy nowej instalacji Home Assistant OS, a dla kontenera na `docker01` wpisujemy 8123 i adres 10.20.10.42.

WWW = World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS · **OS** = Operating System: system operacyjny

TYLKO ODCZYT

Sprawdzenie i przeładowanie: ha core check

```
ha core check
```

W kontenerze używamy `docker exec homeassistant python -m homeassistant --script check_config --config /config`. Po poprawnym wyniku przeładowujemy zmiany w Settings → Tools → YAML. Zmiany w sekcjach `http` i `recorder` wymagają restartu, a dla pozostałych sprawdzamy w tym oknie, które mają przycisk przeładowania.

YAML = YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 2:40 w nocy. Tomek dostaje dwa powiadomienia o tym samym: SMS z bramki, że wlot w szafie R1 ma 32,4 °C, i sygnał z aplikacji Home Assistant o tej samej temperaturze. Dzwoni do Oli, a ona odbiera tak samo zdezorientowana, bo jej telefon pokazał trzecie, w innym brzmieniu. Rano ustalają zasadę: alarm ma jednego właściciela, a drugi system tylko sygnalizuje na miejscu i wchodzi do akcji, gdy ten pierwszy milczy.

Jak to działa

Rozwiń: jak to działa pod spodem

Anatomia automatyzacji

> **Pojęcie: automatyzacja.** Zapisana reguła, która na zdarzenie (wyzwalacz) sprawdza, czy zachodzi dodatkowy warunek, i jeśli tak, wykonuje akcje. Home Assistant wykonuje ją u siebie, bez chmury i bez udziału człowieka.

Część	Klucz w YAML	Pytanie, na które odpowiada	Przykład z Zielonej
Wyzwalacz	`triggers`	co uruchamia automatyzację	wlot powyżej 27 °C przez 5 minut
Warunek	`conditions`	czy wolno jej zadziałać teraz	tryb serwisowy wyłączony
Akcja	`actions`	co robimy	włączamy lampę, tworzymy powiadomienie
Tryb	`mode`	co się dzieje, gdy wyzwalacz wróci w trakcie	`restart` dla lampy, `single` dla alarmu

Od wersji 2026.7 edytor oferuje wyzwalacze i warunki opisane celem, na przykład „gdy w sypialni spadnie poniżej 18 °C”, a istniejące automatyzacje w YAML działają bez zmian [3]. Dokumentacja zaznacza jednak, że część kluczy nowych wyzwalaczy zmieniła nazwy, więc automatyzacje używające starych kluczy poprawiamy ręcznie po aktualizacji [3].

Edytor czy YAML

W edytorze klikamy automatyzację i zapisujemy ją do pliku `automations.yaml`. W pakiecie piszemy ją w YAML i trzymamy w repozytorium Git razem z resztą konfiguracji. W Zielonej przyjmujemy zasadę: automatyzacje związane z serwerownią żyją w pakiecie `serwerownia.yaml`, a domowe i próbne w edytorze, żeby każda miała jedno miejsce życia.

Pakiety mają swoje reguły łączenia. Pomocniki wymagają unikalnych kluczy w całej konfiguracji, a automatyzacje w pakiecie zapisujemy pod jednym kluczem `automation:` i łączymy listy [1]. Ustawienie `auth_providers` musi zostać w głównym `configuration.yaml` [1].

Szablony

> **Pojęcie: szablon.** Fragment tekstu w podwójnych nawiasach klamrowych, który Home Assistant wylicza z bieżących stanów, na przykład `{{ states('sensor.esp01_wlot_r1_gora') | float }}`. Szablony budują czujniki pochodne i treść wiadomości.

Stan encji jest tekstem, więc liczbę zawsze zamieniamy filtrem `float`. Gdy czujnik jest niedostępny, `states()` zwraca tekst `unavailable`, który `float` bez wartości domyślnej zamienia na błąd, a z wartością domyślną na liczbę udającą pomiar [2]. Dlatego w czujniku `serwerownia_wlot_max` stan wylicza szablon, a o dostępności decyduje osobny szablon z `has_value`, który zwraca fałsz dla `unavailable` i `unknown` [2].

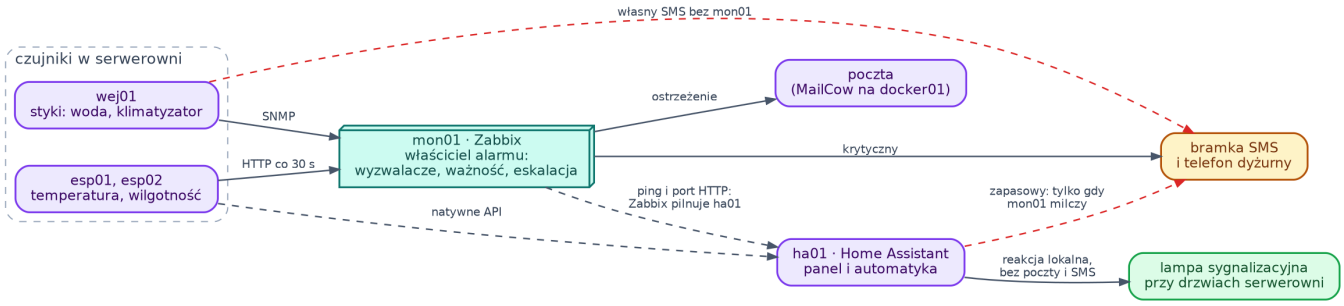
Kanały powiadomień

Kanał	Jak działa	Zalety	Ograniczenia
Panel w interfejsie (`persistent_notification`)	wpis w panelu powiadomień Home Assistant	działa bez internetu	widać go tylko po otwarciu interfejsu
Aplikacja mobilna (`notify.mobile_app_...`)	wiadomość przez usługę push Google albo Apple	dociera poza biurem, ma priorytety	do 500 powiadomień dziennie na urządzenie; treść nie jest szyfrowana w usłudze Firebase [14]
Push lokalny	wiadomość przez połączenie WebSocket z Home Assistant	nie przechodzi przez chmurę, nie liczy się do limitu [15]	działa, gdy telefon ma bezpośrednie połączenie z serwerem
Poczta (integracja SMTP, MailCow)	wiadomość z naszej poczty	ślad w skrzynce, dobre na ostrzeżenia	wolniejsza od push; zależy od MailCow na `docker01`
SMS przez bramkę (`rest_command`)	zapytanie HTTP do bramki	dociera bez internetu w telefonie	wymaga bramki, koszt wiadomości, własny format zapytania

W Zielonej poczta i SMS to kanały Zabbixa. Home Assistant używa ich tylko w zapasowym alarmie, a codzienny sposób dawania znać to panel, lampa i aplikacja na telefonie dyżurnym. Wiadomości z treścią wrażliwą, na przykład hasłami, nie wysyłamy w ogóle przez usługi push.

Kto alarmuje: Zabbix czy Home Assistant

Temperaturę wlotu widzą jednocześnie dwa systemy, więc pierwsza zasada brzmi: jeden alarm ma jednego właściciela. W Zielonej tym właścicielem jest Zabbix na `mon01`, bo ma wyzwalacze z ważnością, eskalację do drugiego administratora i prezesa oraz historię potrzebną do przeglądów [16].



Rysunek 1. Kto alarmuje: czujniki z serwerowni idą do Zabbixa na mon01, który wysyła pocztę (ostrzeżenie) i SMS (krytyczny), i równolegle do Home Assistant na ha01, który zapala lampę przy drzwiach. Moduł wejść wej01 ma własny SMS bez mon01, a Home Assistant ma zapasowy SMS tylko wtedy, gdy mon01 milczy (przerywane czerwone linie). (kliknięcie otwiera pełny rozmiar)

Zdarzenie	Kto alarmuje	Co robi Home Assistant
Wlot powyżej 27 °C (ostrzeżenie)	Zabbix, poczta do zespołu IT	zapala lampę przy drzwiach i zostawia wpis w panelu
Wlot powyżej 32 °C (krytyczny)	Zabbix, SMS i telefon dyżurny	to samo, a przy milczącym `mon01` powiadomienie i SMS jako zapasowy alarm
Woda pod klimatyzacją	`wej01` własnym SMS i Zabbix	tylko pokazuje stan na panelu
Pożar	centrala pożarowa	nic: pożar nigdy nie zależy od Home Assistant
Awaria `ha01`	Zabbix, sprawdzenie ping i portu	nic, bo nie działa
Dom	Home Assistant	wszystko, bo nie ma innego systemu

Zapasowy alarm to zgodna z mapą serwerowni druga droga: gdy poczta i `mon01` stoją na klastrze, który właśnie padł od przegrzania, ktoś musi o tym powiedzieć [16]. Home Assistant jest w tej roli dobrym kandydatem, bo działa na osobnej maszynie `ha01` albo na mini PC i ma własny kontakt z telefonem. Warunek ping `mon01` w stanie `off` od 10 minut sprawia, że w zwykłych dniach zapasowy alarm milczy, więc nie dubluje Zabbixa.

Dane z Home Assistant do Zabbixa

Zabbix czyta czujniki bezpośrednio z płytek, więc dla `esp01` i `esp02` nie potrzebuje Home Assistant. Zostają trzy drogi dla danych, które istnieją tylko w Home Assistant.

Droga	Jak	Kiedy
A: bez pośrednika	Zabbix czyta płytkę przez HTTP, jak opisują strony DIY	czujniki ESPHome, nasz wybór
B: REST API Home Assistant	element HTTP agent z tokenem konta `zabbix`	encje, które są tylko w Home Assistant: czujnik pochodny, urządzenie Zigbee, integracja chmurowa
C: Home Assistant wysyła do Zabbixa	skrypt wysyłający wartości do trappera Zabbixa	nie opisujemy, bo wymaga dodatkowego skryptu i jego utrzymania

Droga B ma jedną wadę: Zabbix zależy wtedy od Home Assistant, więc awaria `ha01` wygląda jak awaria czujnika. Dlatego dla krytycznych wartości zostajemy przy drodze A, a drogą B czytamy tylko to, co jest wygodne i niekrytyczne.

Pilnowanie samego Home Assistant

Panel i automatyka, którym nikt nie patrzy na ręce, psują się po cichu, więc `ha01` jest hostem w Zabbixie z dwoma elementami: ping i port WWW. Wyzwalacz na brak odpowiedzi ma ważność ostrzeżenia, bo `ha01` nie jest drogą alarmu. Zabbix pilnuje też płytek: wyzwalacz `nodata` na każdym czujniku, jak w mapie serwerowni.

W domu

W domu nie mamy Zabbixa, więc kanał jest jeden: Home Assistant. Zalanie w pralni, otwarte okno przy włączonym ogrzewaniu albo lodówka, której drzwi stoją otwarte, powiadamiają telefon w aplikacji, a ważniejsze dodatkowo pocztą. Gdy czujnik wody wysyła alarm, a telefon jest wyłączony, wiadomość pocztowa i tak czeka w skrzynce.

Ta sama płytka ESP32 z innym plikiem ESPHome obsługuje więc dwa światy. W firmie jej odczyt czyta Zabbix i Home Assistant, w domu tylko Home Assistant. Pakiety `serwerownia.yaml` i `dom.yaml` trzymamy w osobnych instalacjach, a nie w jednej, żeby awaria domowego eksperymentu nie wyłączyła panelu serwerowni.

> □□ Więcej możliwości albo gotowe rozwiązanie od ręki (out of the box): gotowa lampa lub syrena sygnalizacyjna z wejściem na styk i zasilaniem z sieci PoE, albo sterownik GSM z wejściami alarmowymi i własnym modemem, który wysyła SMS bez żadnego serwera. W dużych centrach danych stosuje się systemy BMS i DCIM, które zbierają alarmy z klimatyzacji, UPS i czujników

> w jednym miejscu, mają zdefiniowaną eskalację i dwa niezależne tory powiadamiania.

Częste błędy

Objaw	Przyczyna	Co robimy
Ten sam alarm dostajemy dwa razy	Zabbix i Home Assistant mają to samo powiadomienie	zostawiamy właściciela alarmu (Zabbix), a Home Assistant sygnalizuje lokalnie i alarmuje tylko przy milczącym `mon01`
Automatyzacja nie reaguje na wysoką temperaturę, choć jest powyżej progu	wyzwalacz `numeric_state` działa przy przekroczeniu progu, a temperatura już była powyżej po restarcie	sprawdzamy w historii, kiedy wartość przekroczyła próg, i pamiętamy, że `for` po restarcie liczy od zera [6]
Czujnik pochodny pokazuje 0 albo nieprawdopodobną liczbę	`float` bez wartości domyślnej albo z zerem przy niedostępnej sondzie	używamy `availability` z `has_value`, żeby stan był `unavailable`, a nie zero [2]
Powiadomienie push nie przychodzi	telefon bez internetu, wyłączone powiadomienia aplikacji albo przekroczone 500 dziennie	sprawdzamy ustawienia aplikacji, a dla ważnych zdarzeń dodajemy poczty i SMS [14]
`check_config` zgłasza błąd na `automation:`	w pakiecie jest drugi klucz `automation:` w tym samym pliku	łączymy automatyzacje w jedną listę pod jednym kluczem [1]
Poczta z Home Assistant nie dochodzi	brak STARTTLS, zły port albo hasło skrzynki	sprawdzamy dziennik Home Assistant i dziennik MailCow na `docker01` [10]
Zabbix nie czyta encji z Home Assistant	zły token, konto bez uprawnień albo encja `unavailable`	testujemy ten sam adres poleceniem `curl` z `mon01` i sprawdzamy kod odpowiedzi [12]

□□ Ćwiczenia

1. Dlaczego automatyzacja „sygnalizator lokalny” nie wysyła poczty ani SMS, choć technicznie mogłaby?

Odpowiedź

Bo alarm w serwerowni ma jednego właściciela, Zabbixa. Gdyby Home Assistant też wysyłał pocztę, zespół dostawałby ten sam alarm dwa razy w różnym brzmieniu, a przy awarii trudno

byłoby ustalić, który system działa [16].

1. Do czego służy warunek `binary_sensor.mon01_ping` w stanie `off` przez 10 minut w zapasowym alarmie?

Odpowiedź

Zapasowy alarm ma działać tylko wtedy, gdy Zabbix nie może powiedzieć o przegrzaniu. Dziesięć minut ciszy odróżnia awarię od chwilowego restartu `mon01`, więc w zwykłych dniach alarm zapasowy milczy.

1. Sonda `esp01_wlot_r1_gora` przestała odpowiadać, a czujnik `serwerownia_wlot_max` pokazuje 0. Co poprawiamy w szablonie?

Odpowiedź

Dodajemy blok `availability` z warunkiem `has_value` dla wszystkich trzech sond. Wtedy czujnik ma stan `unavailable`, automatyzacje na `numeric_state` nie reagują na zero, a Zabbix i panel widzą, że dane są niedostępne [2].

1. Czemu w domowej automatyzacji o zalaniu nie ma Zabbixa ani zapasowego alarmu?

Odpowiedź

W domu nie ma drugiego systemu, który mógłby być właścicielem alarmu, więc Home Assistant jest jedynym. Dla ważnych zdarzeń dokładamy tylko drugi kanał, na przykład pocztę obok powiadomienia w aplikacji.

Powiązane: Instalacja Home Assistant: system na maszynie wirtualnej, mini PC albo w kontenerze · ESPHome i urządzenia: adopcja esp01 i esp02, API albo MQTT, nazwy encji · Serwerownia Zielonej: gdzie stoją czujniki i dokąd płyną alarmy · Monitorowanie typowych usług · MailCow: codzienna praca z domenami, skrzynkami i spamem

Źródła:

1. Home Assistant, pakiety (`packages`, reguły łączenia, `auth_providers`): home-assistant.io/docs/configuration/packages

2. Home Assistant, funkcje szablonów: `has_value`, `states`, `float`: home-assistant.io/template-functions
3. Home Assistant 2026.7, notatki wydania (wyzwalacze i warunki opisane celem, YAML bez zmian): home-assistant.io/blog/2026/07/01/release-20267
4. Home Assistant, integracja `rest_command`: home-assistant.io/integrations/rest_command
5. Home Assistant, integracja `notify`: home-assistant.io/integrations/notify
6. Home Assistant, wyzwalacz `numeric_state`: home-assistant.io/triggers/numeric_state
7. Home Assistant, wyzwalacze i identyfikatory wyzwalaczy: home-assistant.io/docs/automation/trigger
8. Home Assistant, integracja Ping (ICMP): home-assistant.io/integrations/ping
9. Home Assistant, grupy powiadomień (`platform: group`): home-assistant.io/integrations/group
10. Home Assistant, integracja SMTP (kreator w interfejsie, odbiorcy jako encje): home-assistant.io/integrations/smtp
11. Home Assistant Companion, podstawy powiadomień (`tag`, `group`, `interruption-level`): companion.home-assistant.io/docs/notifications/notifications-basic
12. Home Assistant, REST API (token, `/api/states/<entity_id>`): developers.home-assistant.io/docs/api/rest
13. Zabbix, element typu HTTP agent (nagłówki, makra, JSONPath): zabbix.com/documentation/current/en/manual/config/items/itemtypes/http
14. Home Assistant Companion, prywatność i limity powiadomień (500 dziennie na urządzenie): companion.home-assistant.io/docs/notifications/notification-details
15. Home Assistant Companion, push lokalny: companion.home-assistant.io/docs/notifications/notification-local
16. Serwerownia Zielonej: gdzie stoją czujniki i dokąd płyną alarmy, w tym rozdział „Gdy mon01 milczy” (w tej bazie): f0-02-serwerownia-zielonej-mapa-czujnikow.md
17. Home Assistant 2026.8, notatki wydania (zakładka Tools zamiast Developer tools): home-assistant.io/blog/2026/08/05/release-20268
18. Home Assistant, integracja SMS (strona usuniętej integracji): home-assistant.io/integrations/sms

Panele w Home Assistant: karty, panel serwerowni, panel domowy i uprawnienia

Poziom: średnio

Dotyczy: Home Assistant 2026.9, panele w układzie sekcji (sections), konta użytkowników

Czas: ~60 min

Pokazujemy, jak z encji `esp01` i `esp02` budujemy panel serwerowni, jak ten sam układ kart działa w domu i komu który panel pokazujemy. Wracamy tu przy dodawaniu karty, przy zakładaniu konta dla nowej osoby i przy pytaniu, czy użytkownik widzi więcej, niż powinien. Panel jest miejscem do patrzenia, a nie do alarmowania: alarm w serwerowni zostaje w Zabbixie, jak opisuje strona Automatyzacje i powiadomienia.

Skróty użyte na tej stronie (8)

Skrót	Co znaczy
BMS	Building Management System: system zarządzania budynkiem, który zbiera odczyty z czujników (temperatura, wilgotność) i steruje klimatyzacją oraz wentylacją
DCIM	Data Center Infrastructure Management: dokumentacja tego, co fizycznie stoi w serwerowni, czyli szafy, urządzenia i okablowanie
ESPHome	Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć
ICMP	Internet Control Message Protocol: protokół diagnostyczny sieci, którym działa między innymi ping
PoE	Power over Ethernet: zasilanie urządzenia tym samym kablem sieciowym, którym płyną dane
SMS	Short Message Service: krótka wiadomość tekstowa na telefon, która dociera także bez internetu
VPN	Virtual Private Network: szyfrowany tunel do sieci firmowej
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Fiszka

□□ KLIKAMY W INTERFEJSIE

Dodanie panelu w interfejsie: Settings, Dashboards

```
Settings → Dashboards → Add dashboard
```

Kreator pyta o tytuł, ikonę i o to, czy panel ma być widoczny w pasku bocznym. Przełącznik Admin only ogranicza panel do administratorów [1]. Panel zrobiony w interfejsie przechowuje Home Assistant, a panel w trybie YAML zapisujemy w pliku, który trzymamy w Git razem z resztą konfiguracji.

□□ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: ekran Settings → Dashboards w `ha01` z dwoma panelami, Serwerownia z ikoną serwera i przełącznikiem Admin only oraz Dom.

□□ **YAML** = YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

□□ PLIK KONFIGURACYJNY

Plik panelu serwerowni: dashboards/serwerownia.yaml

```yaml
title: Serwerownia Zielonej
views:
  - title: Serwerownia
    path: serwerownia
    icon: mdi:server
    type: sections
    max_columns: 3
    sections:
      - type: grid
        cards:
          - type: heading
            heading: Wloty szaf, góra
            icon: mdi:thermometer
          - type: tile
            entity: sensor.esp01_wlot_r1_gora
```

```
      name: Wlot R1
    - type: tile
      entity: sensor.esp02_wlot_r2_gora
      name: Wlot R2
    - type: tile
      entity: sensor.esp02_wlot_r3_gora
      name: Wlot R3
    - type: tile
      entity: sensor.esp02_wlot_r4_gora
      name: Wlot R4
- type: grid
  cards:
    - type: heading
      heading: Najwyższy wlot
      icon: mdi:gauge
    - type: gauge
      entity: sensor.serwerownia_wlot_max
      name: Wlot max
      min: 10
      max: 40
      severity:
        green: 18
        yellow: 27
        red: 32
    - type: history-graph
      title: Wlot max, ostatnie 24 godziny
      hours_to_show: 24
      entities:
        - entity: sensor.serwerownia_wlot_max
          name: Wlot max
- type: grid
  cards:
    - type: heading
      heading: Stan i sygnalizacja
      icon: mdi:alarm-light
    - type: tile
      entity: sensor.esp01_zimny_korytarz_wilgotnosc
      name: Wilgotność H1
    - type: tile
```

```
      entity: binary_sensor.esp01_ping
      name: esp01
    - type: tile
      entity: binary_sensor.esp02_ping
      name: esp02
    - type: tile
      entity: switch.serwerownia_sygnalizator
      name: Lampa przy drzwiach
    - type: tile
      entity: input_boolean.serwerownia_tryb_serwisowy
      name: Tryb serwisowy
    - type: markdown
      content: >
        Alarmy obsługuje Zabbix. Listę problemów otwieramy
        pod adresem [zabbix.zielona.example](https://zabbix.zielona.example).
```

Układ składa się z trzech sekcji (grid). Pierwsza ma cztery kafelki wlotów, druga wskaźnik i wykres najwyższego wlotu, trzecia stan płytek, lampy i tryb serwisowy oraz odnośnik do Zabbixa [2]. Karta `heading` otwiera sekcję, karta `tile` pokazuje jedną encję, `gauge` wskaźnik, `history-graph` wykres, a `markdown` tekst z odnośnikiem [3][4][5][6].

Progi wskaźnika `severity` kolorują zakres od zielonego od 18 °C, przez żółty od 27 °C, do czerwonego od 32 °C, czyli według progów ostrzeżenia i krytycznego z mapy serwerowni [3]. Dolny próg 15 °C nie ma osobnego koloru, a jego alarm obsługuje Zabbix. Identyfikatory encji dopasowujemy do nazw z plików `esp01.yaml` i `esp02.yaml`: tu `esp02` jest przykładowym właścicielem wlotów R2 do R4.

□□ PLIK KONFIGURACYJNY

Plik panelu domowego: dashboards/dom.yaml

```
title: Dom
views:
  - title: Dom
    path: dom
    icon: mdi:home
    type: sections
    max_columns: 3
    sections:
      - type: grid
```

```
      cards:
        - type: heading
          heading: Salon i kuchnia
          icon: mdi:sofa
        - type: tile
          entity: sensor.esp_salon_temperatura
          name: Temperatura w salonie
        - type: tile
          entity: sensor.esp_salon_wilgotnosc
          name: Wilgotność w salonie
        - type: tile
          entity: light.kuchnia
          name: Światło w kuchni
    - type: grid
      cards:
        - type: heading
          heading: Dom
          icon: mdi:door
        - type: tile
          entity: binary_sensor.drzwi_wejsciowe
          name: Drzwi wejściowe
        - type: tile
          entity: sensor.pralka_pozostaly_czas
          name: Pralka
        - type: tile
          entity: sensor.dom_moc_teraz
          name: Prąd teraz
    - type: grid
      cards:
        - type: heading
          heading: Sceny
          icon: mdi:play-circle
        - type: tile
          entity: scene.wychodze
          name: Wychodzę
```

Ten sam układ trzech sekcji, ale z innymi encjami: czujnik w salonie z ESPHome, światło w kuchni, drzwi, pralka, pomiar mocy i scena „Wychodzę”. Encje `light.kuchnia`, `binary_sensor.drzwi_wejsciowe`, `sensor.pralka_pozostaly_czas`, `sensor.dom_moc_teraz` i `scene.wychodze`

są przykładowe i pochodzą z integracji, które każdy ma inne. Pomiar mocy może pochodzić z płytki z pomiarem energii, jak na stronie Pomiar energii na ESP32 (DIY).

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 · **DIY** = Do It Yourself: zrób to sam, czyli własnoręcznie zbudowany czujnik zamiast gotowego urządzenia

PLIK KONFIGURACYJNY

Rejestracja paneli w configuration.yaml: lovelace

```
lovelace:
  dashboards:
    dash-serwerownia:
      mode: yaml
      filename: dashboards/serwerownia.yaml
      title: Serwerownia
      icon: mdi:server
      show_in_sidebar: true
      require_admin: true
    dash-dom:
      mode: yaml
      filename: dashboards/dom.yaml
      title: Dom
      icon: mdi:home
      show_in_sidebar: true
```

Klucz panelu musi zawierać myślnik, stąd `dash-serwerownia` i `dash-dom`, a `mode: yaml` jest wymagany [1]. Opcja `require_admin: true` ogranicza panel do administratorów, a `filename` wskazuje plik względem katalogu konfiguracji. Pliki kładziemy w `/config/dashboards/`.

TYLKO ODCZYT

Czujniki dostępności płytek: integracja Ping

```
Settings → Devices & services → Add integration → Ping (ICMP)
```

Dodajemy dwa hosty, 10.20.10.70 i 10.20.10.71, a identyfikatory encji zmieniamy ręcznie na `binary_sensor.esp01_ping` i `binary_sensor.esp02_ping` [7]. Integracja pyta co 30 sekund i

tworzy czujnik binarny, który mówi, czy host odpowiada [7]. Adresy płytek są takie jak na stronie o ESPHome, a ich nazwy do potwierdzenia przez autora.

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

KLIKAMY W INTERFEJSIE

Założenie konta zwykłego użytkownika: Settings, People

```
Settings → People → Add person → Allow login
```

W oknie konta wybieramy, czy osoba ma być administratorem i czy ma mieć dostęp tylko lokalny (Local access only) [8]. Konto bez uprawnień administratora widzi swoje panele, ale nie dociera do ustawień [9]. Hasło nadajemy osobiście i zapisujemy w Vaultwardenie, a po pierwszym logowaniu włączamy uwierzytelnianie dwuskładnikowe, jak na stronie Kopia, aktualizacja i dostęp zdalny.

KLIKAMY W INTERFEJSIE

Panel domyślny dla użytkownika: Profile

```
Profil użytkownika → General → Dashboard
```

Każdy użytkownik wybiera w swoim profilu panel, który otwiera się po wejściu [1]. Konto prezesa ustawiamy na panel serwerowni po jego pierwszym logowaniu.

TYLKO ODCZYT

Sprawdzenie konfiguracji: ha core check

```
ha core check
```

W kontenerze używamy `docker exec homeassistant python -m homeassistant --script check_config --config /config`. Polecenie sprawdza także blok `lovelace` w `configuration.yaml`. Plik panelu w YAML sprawdza dopiero przeglądarka, więc po zmianie odświeżamy stronę i szukamy komunikatu o błędzie konfiguracji nad kartą.

YAML = YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Z życia wzięte

Z ŻYCIA WZIĘTE

Czwartek, 11:00. Pan Zbigniew pyta, czy mógłby widzieć w telefonie, że „z serwerownią wszystko w porządku”, bo kiedyś dowiedział się o awarii klimatyzacji od ochroniarza. Tomek chce od razu założyć mu konto administratora, „żeby widział wszystko”, ale Ola przypomina, że administrator może też zmienić ustawienia i restartować system. Zakładają zwykłe konto, ustawiają mu panel serwerowni jako domyślny i zostawiają dostęp zdalny przez VPN.

Jak to działa

Rozwiń: jak to działa pod spodem

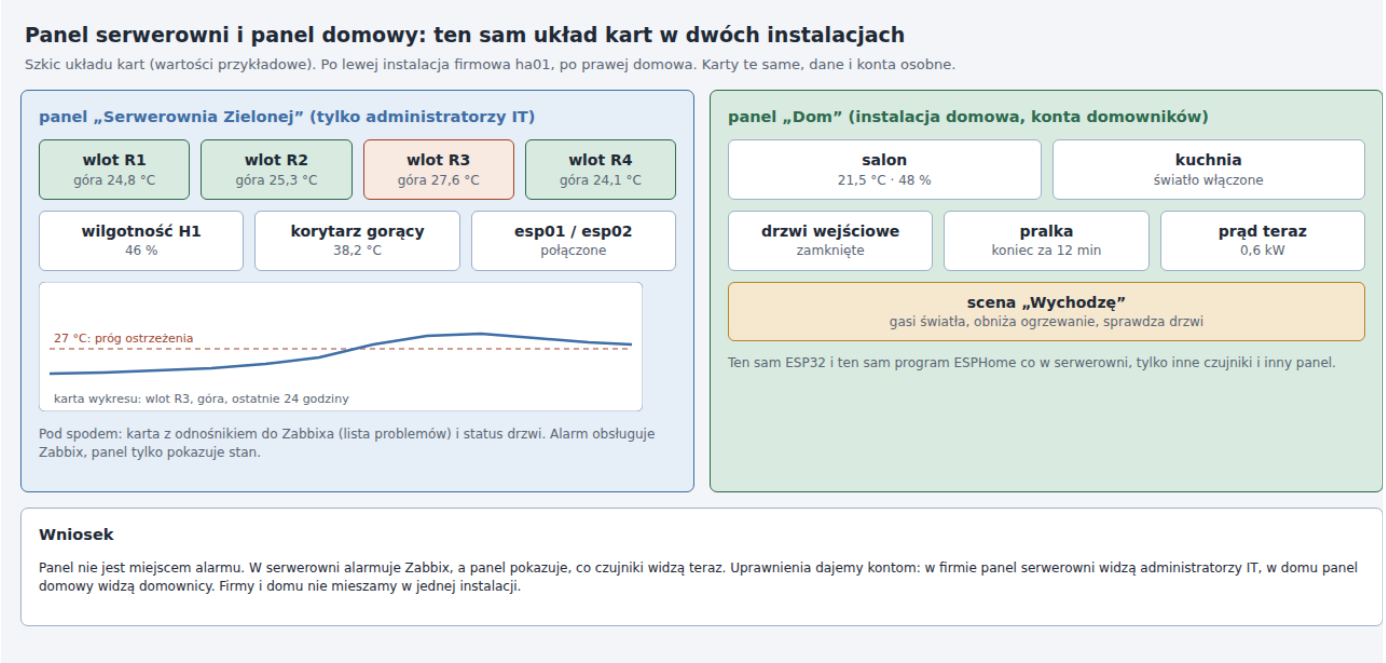
Panel, widok, sekcja, karta

Pojęcie: panel. Zbiór widoków pokazywany pod jedną pozycją w pasku bocznym, na przykład Serwerownia albo Dom. Panel ma tytuł, ikonę i listę widoków.

Pojęcie: karta. Pojedynczy element widoku, który pokazuje encję lub tekst: kafelek, wskaźnik, wykres, tekst. Karty układamy w sekcjach, a sekcje w widoku typu `sections`, który jest układem domyślnym [10].

Karta	Do czego	Przykład w panelu serwerowni
`heading`	tytuł sekcji, opcjonalnie z ikoną i plakietkami	„Wloty szaf, góra” [6]
`tile`	jedna encja i jej stan, z możliwością dotknięcia	wlot R1, lampa, tryb serwisowy [4]
`gauge`	wartość na skali z kolorami progów	najwyższy wlot, 10 do 40 °C [3]
`history-graph`	wykres z ostatnich godzin	wlot max z 24 godzin [5]
`markdown`	tekst, odnośniki, szablony	odnośnik do Zabbixa [11]

Panel serwerowni i panel domowy



Rysunek 1. Ten sam układ kart w dwóch instalacjach: po lewej panel serwerowni na ha01 z kafelkami wlotów, wykresem z progiem 27 °C i odnośnikiem do Zabbixa, po prawej panel domowy z salonem, drzwiami, pralką, prądem i sceną „Wychodzę”. Warto zauważyć, że dane i konta są osobne, a układ ten sam. (kliknięcie otwiera pełny rozmiar)

Panel serwerowni pokazuje, co czujniki widzą teraz. Alarmu nie robimy w panelu: kolor kafelka zmienia się przy 27 °C, ale mail i SMS wysyła Zabbix. Czerwony kafelek jest więc podpowiedzią dla osoby, która właśnie patrzy na ekran, a nie powiadomieniem.

W domu karty są te same, a encje inne. Z jednego panelu sterujemy też światłem i scenami, bo w domu panel jest narzędziem pracy, a nie tylko podglądem. Panel domowy trzymamy w innej instalacji niż panel serwerowni.

Uprawnienia: kto co widzi

Dokumentacja opisuje trzy poziomy kont: właściciela, administratorów i użytkowników bez uprawnień administratora [9]. Właściciel i administratorzy konfigurują integracje, urządzenia i konta, a zwykły użytkownik korzysta z Home Assistant i widzi swoje panele, ale nie dociera do ustawień [9].

Konto	Rola	Panele	Uwagi

Konta zespołu IT (dwa)	właściciel i drugi administrator	wszystkie	dwa konta, żeby nie zablokować się przy utracie jednego
Prezes	zwykły użytkownik	Serwerownia jako domyślny	dostęp zdalny tylko przez VPN
Dom: domownicy	zwykły użytkownik albo administrator	Dom	osobna instalacja i osobne hasła
`zabbix`	zwykły użytkownik, tylko token	żaden	token do odczytu przez REST, bez logowania do interfejsu

Opcja Local access only ogranicza konto do połączeń z sieci lokalnej [8]. Dokumentacja ról nie opisuje, co dokładnie liczy się za sieć lokalną, gdy Home Assistant stoi za serwerem pośredniczącym, więc po wystawieniu przez nginx sprawdzamy to na koncie próbnym, zanim na nim poleganiemy.

> **Pojęcie: Admin only.** Przełącznik panelu (w YAML opcja `require_admin`), który ogranicza panel do administratorów [1]. To jest prawdziwe ograniczenie, w przeciwieństwie do opcji `visible` w widoku.

Opcja `visible` w widoku ukrywa tylko zakładkę widoku, a adres widoku nadal otwiera się dla każdego, kto go zna [10]. Dokumentacja wprost mówi, że to nie jest funkcja bezpieczeństwa [10]. Ukrycie widoku przed kontem traktujemy więc jako porządek na ekranie, a nie jako zabezpieczenie.

Dokumentacja ról opisuje tylko poziomy właściciela, administratora i zwykłego użytkownika. Nie zakładamy więc, że zwykły użytkownik jest kontem tylko do odczytu, i nie dajemy mu panelu z przełącznikami, których nie powinien przełączać. To samo dotyczy tokenu konta `zabbix`: traktujemy go jak hasło.

Dom a firma

Panel domowy i panel serwerowni to dwie osobne instalacje, ponieważ mają różne konta, różne dane i różne ryzyko. Dzielone są plik ESPHome z innymi czujnikami i układ kart z innymi encjami. Gdy chcemy w domu panel „na ścianie”, zakładamy konto domowe z panelem domyślnym i dostępem lokalnym, a tablet zostaje w sieci domowej.

> “ □□ Więcej możliwości albo gotowe rozwiązanie od ręki (out of the box): gotowy tablet ścienny z uchwytem i zasilaniem PoE, który otwiera panel

> Home Assistant w trybie kiosku, albo wyświetlacz stanu serwerowni przy drzwiach ze wskaźnikiem w trzech kolorach. W dużych centrach danych stosuje się systemy DCIM i BMS z panelami wielu poziomów, od hali po pojedynczą szafę, oraz osobne ekrany w centrum operacyjnym.

Częste błędy

Objaw	Przyczyna	Co robimy
Karta pokazuje „Entity not available” albo pusty kafelek	zła nazwa encji w pliku panelu	kopiujemy identyfikator z Settings → Devices & services → Entities
Kolor wskaźnika nie zmienia się przy 27 °C	próg `severity` ustawiony na inną wartość albo wskaźnik ma inny `min` i `max`	sprawdzamy `green`, `yellow`, `red` oraz zakres skali [3]
Panel w pliku nie pokazuje się w pasku bocznym	klucz bez myślnika albo `show_in_sidebar: false`	klucz `dash-nazwa` i `show_in_sidebar: true` [1]
Zwykły użytkownik widzi panel serwerowni	brak `require_admin`	ustawiamy `require_admin: true` albo przełącznik Admin only [1]
Ukryty widok nadal otwiera się z adresu	`visible` ukrywa tylko zakładkę	do ograniczenia dostępu używamy osobnego panelu z Admin only [10]
Po zmianie pliku panelu karta się nie zmieniła	przeglądarka trzyma starą wersję	odświeżamy stronę z pominięciem pamięci podręcznej
Kafelek `esp01` jest „off”, choć płytka działa	host nie odpowiada na ping z `ha01` (zapora, VLAN)	sprawdzamy ping z `ha01` i zasady zapory na `fw01`

Ćwiczenia

1. Pan Zbigniew chce w telefonie widzieć stan serwerowni i nic więcej. Jakie konto mu zakładamy i co ustawiamy?

Odpowiedź

Zwykłe konto bez uprawnień administratora, z panelem serwerowni jako domyślnym w profilu [8][9]. Dostęp zdalny dajemy przez VPN, a w panelu zostawiamy tylko karty, których przełączanie nie szkodzi.

1. Ola ukryła widok „Admin” opcją `visible` dla konta prezesa. Czy konto prezesa na pewno go nie otworzy?

Odpowiedź

Nie. Opcja `visible` ukrywa tylko zakładkę, a adres widoku nadal działa dla każdego, kto go zna [10]. Do ograniczenia dostępu potrzebujemy osobnego panelu z opcją `require_admin` albo Admin only [1].

1. Wskaźnik pokazuje 28 °C na żółto. Czy to znaczy, że Zabbix wysłał już alarm?

Odpowiedź

Nie musi. Kolor wskaźnika to tylko podpowiedź na panelu. Zabbix wysyła ostrzeżenie dopiero po spełnieniu swojego wyzwalacza, czyli po 5 minutach powyżej 27 °C, i to on odpowiada za pocztę i SMS.

1. Dlaczego panelu domowego nie dodajemy do instalacji `ha01`, skoro karty są te same?

Odpowiedź

Bo firma i dom mają inne konta, inne dane i inny poziom ryzyka. Awaria domowego eksperymentu nie może wyłączyć panelu serwerowni, a domownicy nie powinni mieć dostępu do serwera firmowego.

Powiązane: Instalacja Home Assistant: system na maszynie wirtualnej, mini PC albo w kontenerze · ESPHome i urządzenia: adopcja esp01 i esp02, API albo MQTT, nazwy encji · Automatyzacje i powiadomienia: alarm temperatury w serwerowni bez dublowania Zabbixa · Serwerownia Zielonej: gdzie stoją czujniki i dokąd płyną alarmy

Źródła:

1. Home Assistant, panele (YAML, `lovelace`, `require_admin`, Admin only, panel domyślny): home-assistant.io/dashboards/dashboards
2. Home Assistant, widok typu sections: home-assistant.io/dashboards/sections
3. Home Assistant, karta gauge (`severity`, `min`, `max`): home-assistant.io/dashboards/gauge
4. Home Assistant, karta tile: home-assistant.io/dashboards/tile
5. Home Assistant, karta history graph (`hours_to_show`): home-assistant.io/dashboards/history-graph

6. Home Assistant, karta heading: home-assistant.io/dashboards/heading
7. Home Assistant, integracja Ping (ICMP): home-assistant.io/integrations/ping
8. Home Assistant, integracja Person (Allow login, Local access only, Administrator): home-assistant.io/integrations/person
9. Home Assistant, konfiguracja użytkowników i role: home-assistant.io/docs/configuration/user-configuration
10. Home Assistant, widoki (typy, opcja `visible`, „to nie jest funkcja bezpieczeństwa”): home-assistant.io/dashboards/views
11. Home Assistant, karta Markdown: home-assistant.io/dashboards/markdown