Bron: Shutterstock

Cloud-gebaseerde energiemeter

met een ESP32-module en een PZEM-004T spanning/stroom-sensor

A project from Elettronica In
https://elettronicain.it

Emanuele Signoretta (Italië)

Met de alsmaar stijgende prijs van elektriciteit is verstandig gebruik en besparing een must geworden. Met een ESP32-module en een paar andere hardwarecomponenten is het mogelijk om een energiemeter te maken die onze energiegegevens via een verbinding met het WiFi-netwerk naar het InfluxDb Cloud-platform stuurt.

Italië is, net als veel andere landen, sterk afhankelijk van buitenlandse energiebronnen omdat de eigen duurzame energieproductie niet voldoende is om aan de binnenlandse vraag te voldoen. Dit wordt een nog groter probleem wanneer externe omstandigheden de beschikbaarheid van fossiele brandstoffen beperken, wat leidt tot hogere prijzen. Als gevolg daarvan moeten mensen in Italië, net als elders, bezuinigen op hun energieverbruik, bepaalde gemakken opofferen en hun energieverbruik nauwlettend in de gaten houden om onaangename verrassingen op hun stroom- en gasrekening te voorkomen. De belangrijkste drijfveer om het energieverbruik te verminderen is de angst om te veel uit te geven, en niet zozeer de bezorgdheid over de klimaatverandering.

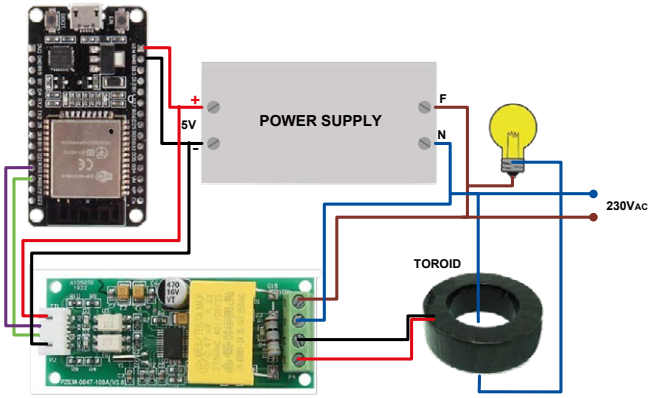
Om degenen te helpen die zich bewust zijn van hun elektriciteitsverbruik, introduceert dit artikel een energieverbruiksmeter die gebruik maakt van een Espressif ESP32-module en een PZEM-004T spanning/stroom-sensor. De meter verzamelt gegevens en stuurt deze naar de cloud, om precies te zijn naar de servers van de InfluxDB online-service.

De hardware

Voor de hardware van de energiemeter hebben we een ESP32 WiFi-module nodig (**figuur 1**) en een PZEM-004T sensor (ingekapseld te zien in zijn doorzichtige plastic behuizing in **figuur 2**). We hebben ook een schakelende voeding nodig met een 5V-uitgang (als je er een kiest met een microUSB-uitgang, kun je deze gebruiken om het ESP-board

Figuur 1. Het ESP32-board.

Figuur 2. De PZEM-004T-sensor met transformator.



Figuur 3. Het bedradingsschema van de schakeling.

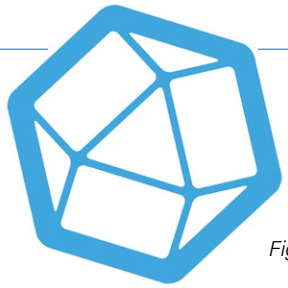
van voeding te voorzien en de vereiste voeding voor de PZME-004T via V_{IN}...), female-female jumpers voor de Arduino, aansluitklemmen, wat bedrading en een plastic behuizing [2]. De datasheet van de sensor is te vinden op [3], terwijl **figuur 3** de algemene bedrading van de schakeling laat zien – in de paragraaf **Installatie** verderop gaan we daar dieper op in.

De sensormodule in een kunststof behuizing bestaat uit een schakeling om stroom en spanning te meten met een open ringkern. Hij bevat weinig elektronica, is geïsoleerd door optocouplers en heeft een seriële interface. De module kan nauwkeurig spanningen meten van 80 V tot 260 V, met een resolutie van 0,1 V en een nauwkeurigheid van 0,5%. De sensor kan ook stromen meten van 0 A tot 100 A met een nauwkeurigheid van 0,5% en een resolutie van 0,001 A. Bovendien kan de sensor de fasehoek (arbeidsfactor) tussen de spannings- en stroomvectoren bepalen met een nauwkeurigheid van 1%, waardoor metingen van actief en reactief vermogen mogelijk zijn. Hiermee kunnen we de elektrische efficiëntie evalueren van het apparaat dat wordt gemeten. De keuze van de componenten voor de schakeling was gebaseerd op prijs/prestatieverhouding, gezien de aanhoudende crisis in de halfgeleiderindustrie, en de indrukwekkende specificaties van de ESP32.

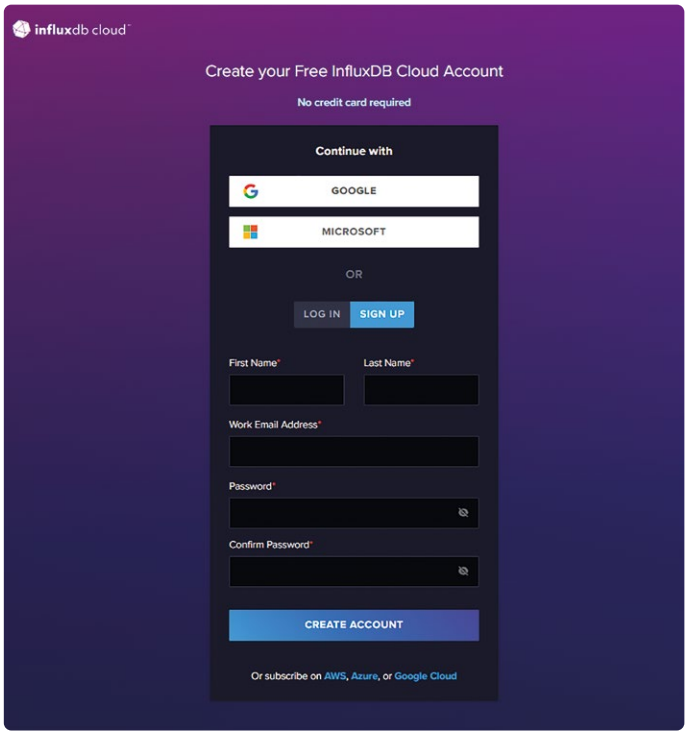
Configuratie van InfluxDB

Er zijn meerdere softwarepakketten beschikbaar voor het ontvangen en analyseren van gegevens die worden verzonden door IoT-apparaten. Bekende voorbeelden zijn Home Assistant, Grafana, Blynk, Thingspeak en andere. In dit geval gebruiken we InfluxDB, waarvan **figuur 4** het logo toont.

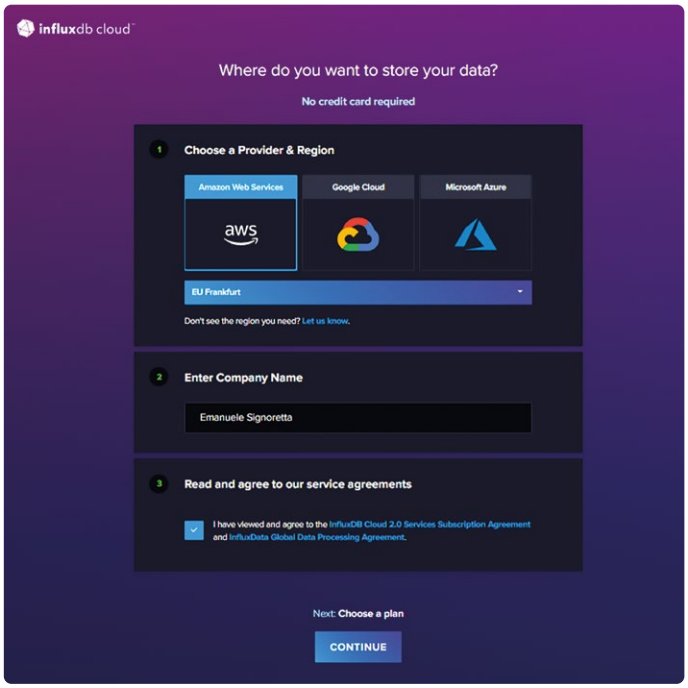
InfluxDB is veelzijdige software die functionaliteit biedt voor het maken van dashboards, het uitvoeren van queries en het versturen van meldingen. Het biedt meerdere mogelijkheden voor implementatie, waaronder uitvoerbare versies voor verschillende architecturen,



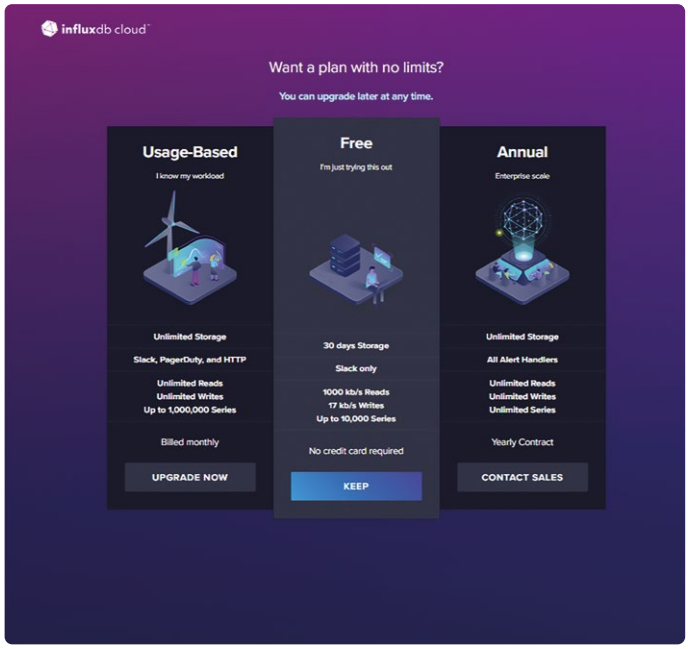
Figuur 4. Het InfluxDB-logo.



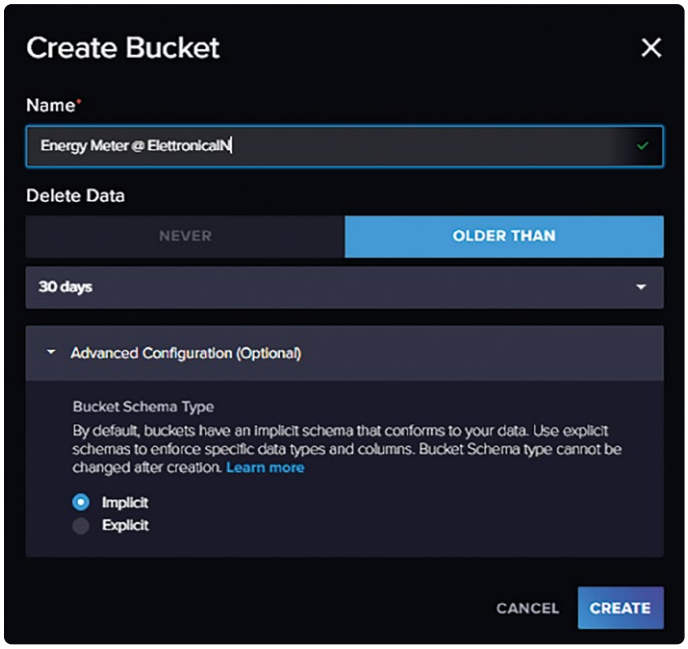
Figuur 5. De InfluxDB-registratiepagina.



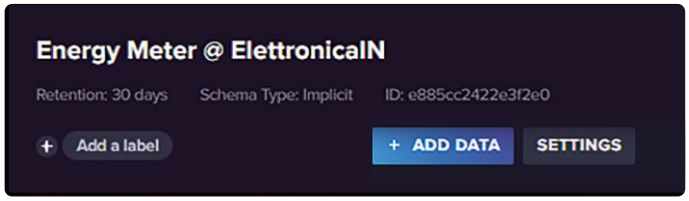
Figuur 6. Keuze van provider en aanvaarding van de servicevoorwaarden.



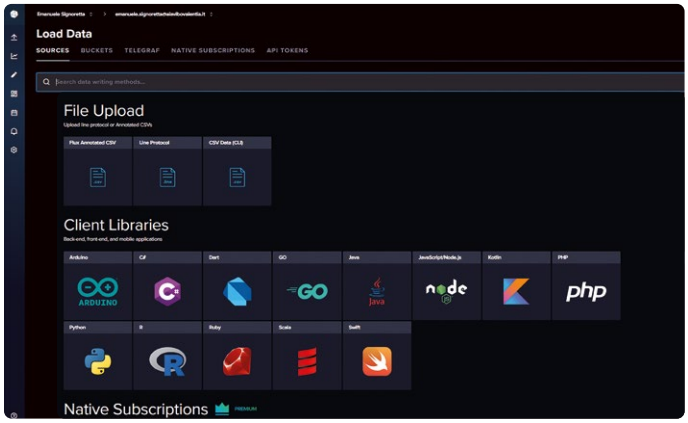
Figuur 7. Keuze van het abonnement.



Figuur 8. Aanmaken van een nieuwe bucket.



Figuur 9. De nieuwe bucket.



Figuur 10. Keuze van de data-uploadbron.

Figuur 11. Fragment van een demo-sketch.

Docker-containers en cloud-gebaseerde oplossingen. Aanvankelijk waren we van plan om het systeem op een Raspberry Pi te installeren vanwege het eigendomsrecht van de gegevens, maar de schaarste van componenten dwong ons om in plaats daarvan de InfluxDb Cloud v2-service te gebruiken.

Ga om te beginnen naar link [4] en registreer je eerst. Je kunt ervoor kiezen om je aan te melden met je Microsoft- of Google-inloggegevens of handmatig de verplichte velden in te vullen, zoals te zien is in **figuur 5**. Nadat je het registratieproces hebt voltooid, krijg je een scherm te zien dat lijkt op dat van **figuur 6**, waar je wordt gevraagd om verschillende details op te geven, waaronder de naam van de serviceprovider. Voor ons project hebben we Amazon Web Services (AWS) geselecteerd als provider.

Op het volgende scherm (te zien in **figuur 7**) moet je een abonnement kiezen. In ons geval kozen we voor een gratis abonnement, waarmee je 30 dagen lang gegevens kunt opslaan en meldingen via Slack kunt ontvangen.

Na het bevestigen van onze keuze verschijnt je persoonlijke dashboard. Om een bucket aan te maken, ga je naar *Load Data Buckets Create new bucket*. Hiermee open je een scherm dat lijkt op dat van **figuur 8**. Vul het veld *Name* in en klik op de knop *Create*. Zodra de bucket is aangemaakt, verschijnt deze bij de beschikbare gegevensbronnen, zoals te zien in **figuur 9**.

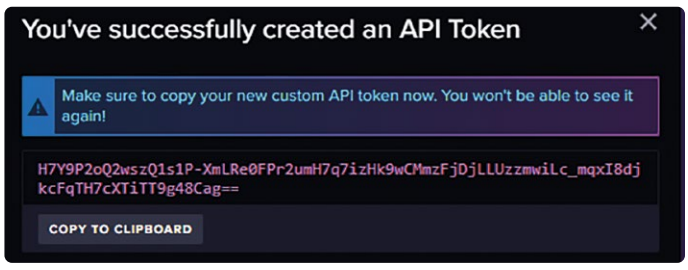
Klik vervolgens op *Add data Client library*. Er verschijnt een nieuw scherm met verschillende opties voor het uploaden van gegevens. Kies uit deze opties *Arduino* (**figuur 10**). In het volgende venster (**figuur 11**) zie je een reeks codefragmenten die samen een demosketch vormen. Kopieer de gegevens voor `INFLUXDB_URL`, `INFLUXDB_ORG`, en `INFLUXDB_BUCKET` uit het eerste fragment.

Om de installatie te voltooien, moeten we een toegangscode voor de bucket aanmaken. Ga hiervoor naar de linker zijbalk en ga naar *Load Data API Tokens*.

Klik in het venster dat dan verschijnt op *Generate API Token*. Nadat je de bucket hebt geselecteerd, schakel je zowel lees- als schrijfrechten in, zoals te zien in **figuur 12**. Zodra de code is aangemaakt, kopieer je deze en bewaar hem goed, want we hebben hem nodig in de volgende stappen.

De sketch

De sketch voor ons project is een combinatie van een aantal sketches van de bibliotheken die we hebben gebruikt. Je kunt de sketchbestanden downloaden van de GitHub-repository [5]. Laten we nu onze code analyseren, die is verdeeld in drie secties die we zullen behandelen met de bijbehorende listings.



Figuur 12. Aanmaken van een API-key.

Listing 1. De bibliotheken toevoegen.

```
#include <WiFiMulti.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <InfluxDbClient.h>
#include <InfluxDbCloud.h>
#include <PZEM004Tv30.h>
#include <Every.h>
```

Laten we het eerst hebben over het toevoegen van de benodigde bibliotheken en afhankelijkheden, zoals getoond in **listing 1**.

Hierbij is het belangrijk op te merken dat de *WiFiMulti*-bibliotheek wordt gebruikt voor het beheren van de WiFi-communicatie tussen de ESP32 en het access point. De *ESPmDNS*-, *WiFiUDP*- en *ArduinoOTA*-bibliotheken [6] worden gebruikt voor het OTA-uploaden (over-the-air) van sketches en het host-name beheer. Zorg ervoor dat Python versie 2.7.x geïnstalleerd is op je PC voor deze functionaliteit. De *InfluxDbClient*- en *InfluxDbCloud*-bibliotheken worden gebruikt voor het uploaden van gegevens naar de InfluxDB Cloud. De *PZEM004Tv30*-bibliotheek [7] zorgt voor seriële communicatie met de sensor. Tot slot maakt de *Every*-bibliotheek het mogelijk om codeblokken met regelmatige tussenpozen uit te voeren zonder afhankelijk te zijn van `delay()`.

Laten we nu de hieropvolgende delen van de code bekijken, te beginnen met **listing 2**, die het pre-processorgedeelte bevat. Met `#define REFRESH_TIME 5000` stellen we het interval (in milliseconden) in voor het uploaden van gegevens naar de cloud. Vervolgens definiëren we de naam van het apparaat, de seriële communicatiepinnen en de gewenste seriële poort. Objecten met betrekking tot het WiFi-netwerk en de sensor worden geconfigureerd. IP-adressen worden gedefinieerd en als je een verbinding met het toegangspunt met een statisch IP wilt configureren, kun je de parameters aanpassen aan het subnet van je netwerk. Tot slot definiëren we parameters voor de WiFi-verbinding en de InfluxDB Cloud-toegang. Vervang deze regels code door het gekopieerde fragment en voer de ontbrekende gegevens in. Met `#define WIFI_SSID` en `#define WIFI_PASSWORD` geven we respectievelijk de SSID en het wachtwoord van het WiFi-netwerk op.

Vervolgens stellen we de parameters in om verbinding te maken met InfluxDB Cloud. De enige ontbrekende parameter is `INFLUXDB_TOKEN`, die kan worden opgehaald uit de API-sleutel die we eerder hebben gemaakt. Zet die API-sleutel tussen de aanhalingstekens. Met `#define TZ_INFO "CET-1CEST,M3.5.0,M10.5.0/3"` specificeren we de Midden-Europese tijdzone voor tijdregistraties.

Gebruik de onderstaande code om het clientobject te maken dat verbinding maakt met de InfluxDB-server:

```
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG,
                      INFLUXDB_BUCKET, INFLUXDB_TOKEN,
InfluxDbCloud2CACert);
```

Listing 2. Definities.

```
#define REFRESH_TIME 5000 // Delay interval for data upload
#define DEVICE "ESP32"
#define PZEM_RX_PIN 27
#define PZEM_TX_PIN 26
#define PZEM_SERIAL Serial2

/*************************
ESP32 initialization
---------------------

   The ESP32 HW Serial interface can be routed to any GPIO pin
   Here we initialize the PZEM on Serial2 with RX/TX pins 26 and 27
*/
PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN);
WiFiMulti wifiMulti;
IPAddress local_IP(192, 168, 178, 154);
IPAddress gateway(192, 168, 178, 1);
IPAddress subnet(255, 255, 255, 0);
IPAddress primaryDNS(192, 168, 178, 1); //optional
IPAddress secondaryDNS(1, 1, 1, 1); //optional
// WiFi AP SSID
#define WIFI_SSID ""
// WiFi password
#define WIFI_PASSWORD ""
// InfluxDB v2 server url, e.g. https://eu-central-1-1.aws.cloud2.influxdata.com
 // (Use: InfluxDB UI -> Load Data -> Client Libraries)
#define INFLUXDB_URL "https://eu-central-1-1.aws.cloud2.influxdata.com"
// InfluxDB v2 server or cloud API token
 //   (Use: InfluxDB UI -> Data -> API Tokens -> Generate API Token)
#define INFLUXDB_TOKEN ""
// InfluxDB v2 organization id (Use: InfluxDB UI -> User -> About -> Common Ids )
#define INFLUXDB_ORG ""
// InfluxDB v2 bucket name (Use: InfluxDB UI ->  Data -> Buckets)
#define INFLUXDB_BUCKET "Energy Meter @ ElettronicaIN"
// Set timezone string according to
//    https://www.gnu.org/software/libc/manual/html_node/TZ-Variable.html
#define TZ_INFO "CET-1CEST,M3.5.0,M10.5.0/3"
// InfluxDB client instance with preconfigured InfluxCloud certificate
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG,
    INFLUXDB_BUCKET, INFLUXDB_TOKEN, InfluxDbCloud2CACert);
// Data point
Point sensor("EnergyMeter");
```

Listing 3. Setup.

```
void setup() {
  Serial.begin(115200);
  // Uncomment in order to reset the internal energy counter
  // pzem.resetEnergy()
  // Setup wifi
  WiFi.mode(WIFI_STA);
  //Comment to use DHCP instead of static IP
   if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
    Serial.println("STA Failed to configure");
  }
```

```
  wifiMulti.addAP(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to wifi");
  while (wifiMulti.run() != WL_CONNECTED) {
    //Serial.print(".");
    //delay(100);
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
  }
  Serial.println();
  // Port defaults to 3232
  // ArduinoOTA.setPort(3232);
  // Hostname defaults to esp3232-[MAC]
  ArduinoOTA.setHostname("ESP32-Energy Meter");
  // No authentication by default
  // ArduinoOTA.setPassword("admin");
  // Password can be set with it's md5 value as well
  // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
  // ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");
  ArduinoOTA
  .onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";
    // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  })
  .onEnd([]() {
    Serial.println("\nEnd");
  })
  .onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  })
  .onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
  });
  ArduinoOTA.begin();
  // Add tags
  sensor.addTag("Dispositivo", DEVICE);
  // Accurate time is necessary for certificate validation and writing in batches
  // For the fastest time sync find NTP servers in your area: https://www.pool.ntp.org/zone/
  // Syncing progress and the time will be printed to Serial.
  timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov");\
  // Check server connection
  if (client.validateConnection()) {
    Serial.print("Connected to InfluxDB: ");
    Serial.println(client.getServerUrl());
  } else {
    Serial.print("InfluxDB connection failed: ");
    Serial.println(client.getLastErrorMessage());
  }
}
```

Tenslotte wordt het sensorobject gemaakt met `Point sensor("EnergyMeter")`. Dit object, met de naam EnergyMeter, wordt geassocieerd met alle gegevens die door de sensor worden verzameld. De code in **listing 3** geeft de configuratie van het board weer en initialiseert de seriële poort op een baudrate van 115.200.
Het codeblok begint met `WiFi.mode(WIFI_STA);` dat WiFi in de station-modus initialiseert om verbinding te maken met een toegangspunt. Het volgende blok code:

```
if (!WiFi.config(local_IP, gateway, subnet,
  primaryDNS, secondaryDNS)) {
 Serial.println(“STA failed to configure”);
}
```

stelt een statisch IP in, in plaats van DHCP te gebruiken. Als er een fout optreedt, wordt er een waarschuwing weergegeven via de seriële poort. Als je DHCP wilt gebruiken, kun je dit deel van de code wegcommentariëren.
De functie-aanroep `wifiMulti.addAP(WIFI_SSID, WIFI_PASSWORD)` probeert verbinding te maken met het toegangspunt met het opgegeven SSID en wachtwoord.
Vervolgens wordt geprobeerd verbinding te maken met het access point; als dit mislukt wordt eerst een foutmelding weergegeven via de seriële poort en wordt het board na een wachttijd van vijf seconden opnieuw opgestart.
De hostnaam wordt ingesteld met `ArduinoOTA.setHostname("ESP32-Energy Meter")`, die zal worden verzonden via mDNS en weergegeven op de Arduino IDE voor het laden van de OTA-sketch. Er is becommentarieerde code om het laden van de OTA-sketch te beperken tot gebruikers met een gebruikersnaam en wachtwoord. Zet het commentaar van deze regels code uit om deze beveiligingsfunctie toe te voegen.
De instellingen voor de OTA-modus worden beheerd met het volgende codeblok:

```
ArduinoOTA.onStart([]() {
  String type;
  // …
  if (error == OTA_AUTH_ERROR) Serial.println("Auth
Failed");
  else if (error == OTA_BEGIN_ERROR) Serial.println("Begin
Failed");
   else if (error == OTA_CONNECT_ERROR) Serial.
println("Connect Failed");
   else if (error == OTA_RECEIVE_ERROR) Serial.
println("Receive Failed");
  else if (error == OTA_END_ERROR) Serial.println("End
Failed");
});
ArduinoOTA.begin();
```

Het codeblok behandelt OTA-start, -einde, sketch laden en fouten (in die volgorde). `ArduinoOTA.begin()` start het OTA-laadproces.
Met `sensor.addTag(“Device”, DEVICE);` wordt een *tag* ingesteld die in dit geval overeenkomt met de naam van het apparaat. Deze functie kan handig zijn om een of meer sensoren binnen een groep te onderscheiden.
Met `timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov");` maakt de ESP verbinding met de twee NTP-servers *pool.ntp.org* en *time.nis.gov* (network time protocol) om de correcte datum en tijd te verkrijgen.
Tenslotte probeert het codeblok verbinding te maken met de InfluxDB servers:

```
if (client.validateConnection()) {
  Serial.print(“Connected to InfluxDB: ”);
  Serial.println(client.getServerUrl());
}
else {
  Serial.print(“InfluxDB connection failed: ”);
  Serial.println(client.getLastErrorMessage());
}
```

Als de verbinding succesvol is, wordt de URL van de server naar de seriële poort verzonden. Zo niet, dan wordt in plaats daarvan de bijbehorende foutmelding geprint.
Laten we nu verder gaan met **listing 4**, die de `loop()`-code voor onze sketch bevat. Met `ArduinoOTA.handle()` wacht de ESP32 tot OTA gecompileerde sketches ontvangt.
Met `EVERY(REFRESH_TIME) { ... }` wordt de code binnen de haakjes uitgevoerd met regelmatige intervallen en zonder gebruik te maken van vertragingen. Het tijdsinterval is al eerder gedefinieerd. Binnen het `EVERY`-blok wordt eerst het adres van de PZEM gelezen en serieel weergegeven.
Vervolgens worden sensorvariabelen aangemaakt en geïnitialiseerd met functies uit de sensorbibliotheek. Deze variabelen zijn spanning, stroom, actief vermogen (in Wh), actieve energie (in kWh), frequentie en arbeidsfactor (een waarde tussen –1 en 1, die de verhouding tussen actief vermogen en schijnbaar vermogen weergeeft). Als een van de variabelen geen getal is, wordt een leesfout naar de seriële poort gestuurd. In alle andere gevallen worden de sensormetingen verzonden. Hierna worden de geïnitialiseerde velden en tijdsaanduidingen gewist en worden de variabelen voorbereid voor het uploaden naar de cloud. Tot slot worden de WiFi-verbinding en het succesvol uploaden van de gegevens gecontroleerd. Als het uploaden mislukt, wordt de laatste foutcode verzonden naar de seriële poort.

Installatie

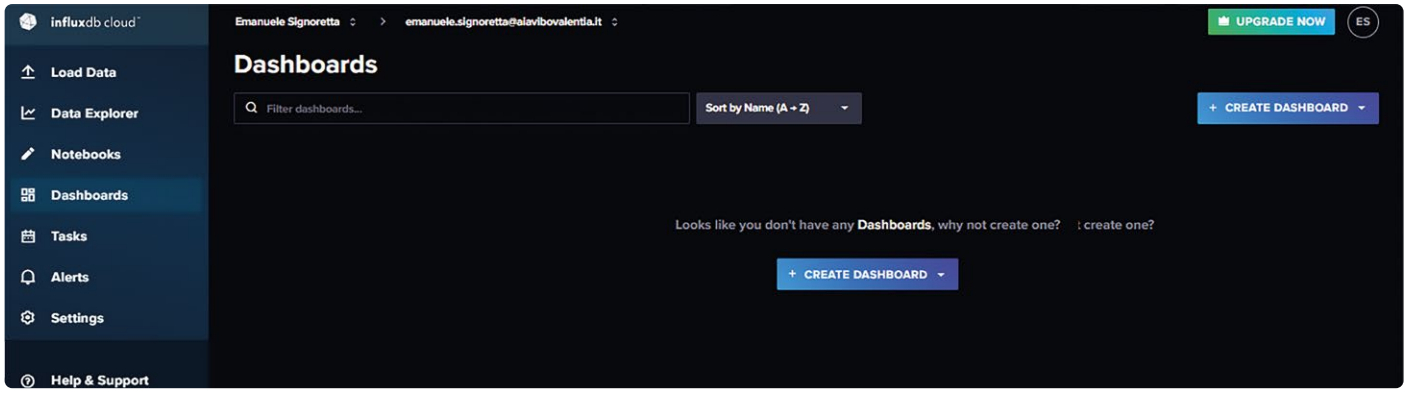
We waren van plan om de voeding, de ESP32 en de hoofd-sensormodule in een mooie plastic behuizing onder te brengen, met alleen de kabels voor de voeding en voor de ringkern van de stroomsensor naar buiten gevoerd. Helaas is zo'n behuizing niet bijzonder passend, omdat de 230V-printkroonstenen niet zijn afgeschermd tegen onbedoeld aanraken. De datasheet geeft ook geen informatie over de vraag of de voedingslijnen voor de torus volgens de regels galvanisch geïsoleerd zijn van de 'hoogspanning'; daar mag je niet blindelings op vertrouwen. Zorg er dus voor dat de behuizing met de voedingslijnen voldoet aan de regelgeving op het gebied van veiligheid! Als je het apparaat

Listing 4. Loop.

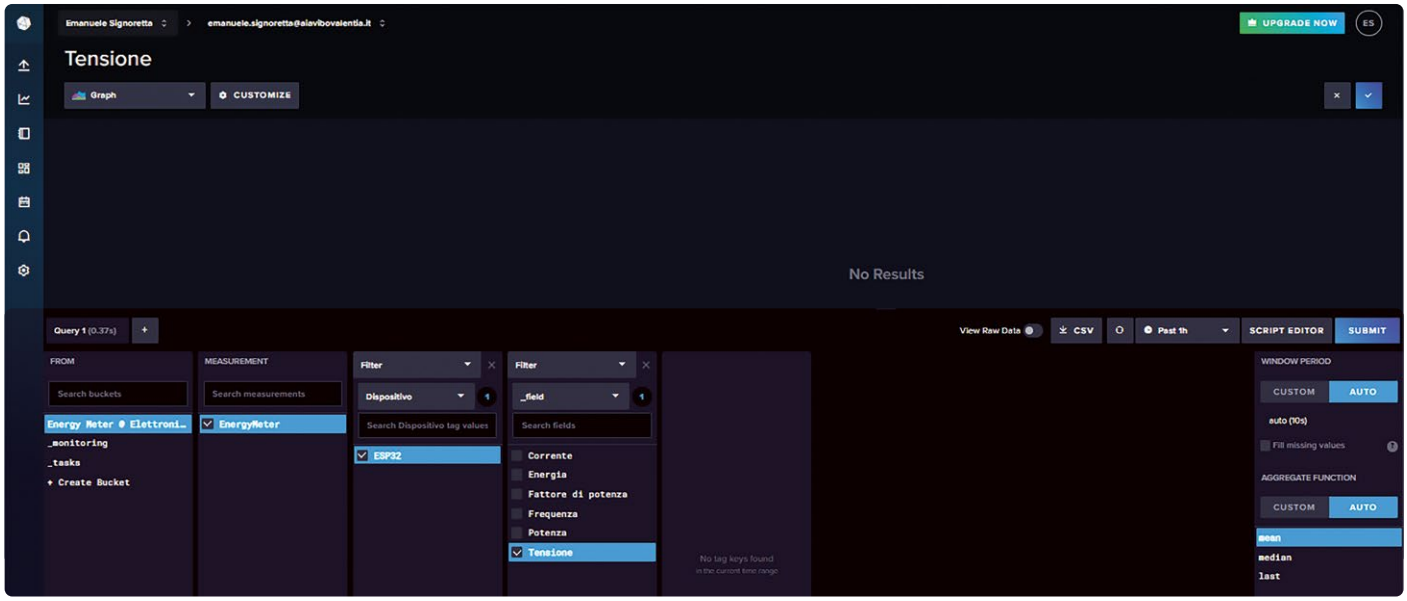
```
void loop() {

  ArduinoOTA.handle();

  EVERY(REFRESH_TIME) {
    // Print the custom address of the PZEM
    Serial.print("Custom Address:");
    Serial.println(pzem.readAddress(), HEX);
    // Read the data from the sensor
    float voltage = pzem.voltage();
    float current = pzem.current();
    float power = pzem.power();
    float energy = pzem.energy();
    float frequency = pzem.frequency();
    float pf = pzem.pf();
    // Check if the data is valid
    if (isnan(voltage)) {
      Serial.println("Error reading voltage");
    } else if (isnan(current)) {
      Serial.println("Error reading current");
    } else if (isnan(power)) {
      Serial.println("Error reading power");
    } else if (isnan(energy)) {
      Serial.println("Error reading energy");
    } else if (isnan(frequency)) {
      Serial.println("Error reading frequency");
    } else if (isnan(pf)) {
      Serial.println("Error reading power factor");
    } else {
      // Print the values to the Serial console
      Serial.print("Voltage: ");      Serial.print(voltage);      Serial.println("V");
      Serial.print("Current: ");      Serial.print(current);      Serial.println("A");
      Serial.print("Power: ");        Serial.print(power);        Serial.println("W");
      Serial.print("Energy: ");       Serial.print(energy, 3);     Serial.println("kWh");
      Serial.print("Frequency: ");    Serial.print(frequency, 1); Serial.println("Hz");
      Serial.print("PF: ");           Serial.println(pf);
      ////UPLOAD DATA
      // Clear fields for reusing the point. Tags will remain untouched
      sensor.clearFields();
      // Store measured value into point
      sensor.addField("Tensione", voltage);
      sensor.addField("Corrente", current);
      sensor.addField("Potenza", power);
      sensor.addField("Energia", energy);
      sensor.addField("Frequenza", frequency);
      sensor.addField("Fattore di potenza", pf);
      // Print what are we exactly writing
      Serial.print("Writing: ");
      Serial.println(sensor.toLineProtocol());
      // Check WiFi connection and reconnect if needed
      if (wifiMulti.run() != WL_CONNECTED) {
        Serial.println("Wifi connection lost");
      }
      // Write point
      if (!client.writePoint(sensor)) {
        Serial.print("InfluxDB write failed: ");
        Serial.println(client.getLastErrorMessage());
      }
    }
    Serial.println();
  }
}
```



Figuur 13. Het dashboard maken.



Figuur 14. Toevoegen van de dashboard-graphics.

wilt gebruiken om de energiehonger van een individueel apparaat te bepalen, raden we aan om de hele elektronica, inclusief alle kabels, in een randaarde-stekkerbehuizing te installeren, zodat er geen aanraakgevaar bestaat. In dat geval is er zelfs geen galvanisch gescheiden 5V-voeding nodig!

Neem de bedrading die al in figuur 3 is getoond als richtlijn, met links de 5V-bedrading en rechts in blauw en bruin de 230V-bedrading, parallel aan de ingang van de voedingseenheid en aan de klemmen 1/2 van de PZEM-004T print, die helaas niet duidelijk gemarkeerd zijn. De ringkern wordt aangesloten op de klemmen 3/4. Eén geleider (L of N) en de PE-beschermgeleider worden doorgelust in de connectorbehuizing en de andere wordt door de ringkern geleid.

Voordat je de afzonderlijke onderdelen van de energiemeter stevig vastzet in de behuizing, moet je de sketch in het ESP-board laden via microUSB (waarmeede module ook wordt gevoed). Daarna kun je alle onderdelen bedraden zoals beschreven en in de behuizing bevestigen. Na controle van de bedrading (zie ook **tabel 1**) kan de behuizing worden gesloten en het systeem worden opgestart. Op deze manier kom je nooit te dicht bij de gevaarlijke netspanning!

Opstarten en dashboard instellen

Zodra het board is opgestart, gaan we terug naar de InfluxDB-website en vanuit ons dashboard gaan we naar *Dashboards Create new dashboard*. Dan verschijnt een scherm zoals in **figuur 13**, waar we een naam toekennen aan ons dashboard en op *Add Cell* klikken. In het scherm dat verschijnt (**figuur 14**) gaan we selecteren welke parameters we willen opnemen in elk van de grafieken die deel uitmaken van het dashboard. We selecteren, in deze volgorde, de bucket, de metingen, het apparaat en tenslotte de waarden die moeten worden weergegeven. Je kunt kiezen welke grafiek je wilt gebruiken voor de waarden: heatmap, meter, eenvoudige grafiek, tabel enzovoort, en ook hoe je de waarden wilt schalen. We passen de cellen naar wens aan en herhalen de procedure voor elke waarde die we willen weergeven totdat we een resultaat krijgen zoals in **figuur 15**, die het uiteindelijke dashboard toont.

Tabel 1. Bedrading van de energiemeter en de ESP32-module.

PZEM-004 pin	ESP32 pin
VCC	V5
GND	GND
RX	26
TX	27

Figuur 15. Het uiteindelijke dashboard.

Conclusie

Tot zover de beschrijving van onze eenvoudige maar doeltreffende energiemeter. De flexibiliteit van het ESP32-board en de vele mogelijkheden van de ondersteunde netwerkprotocollen en platforms zorgen ervoor dat we het toepassingsgebied naar behoefte kunnen uitbreiden, bijvoorbeeld door de meter te integreren in een load management-systeem en toch zelfstandig te laten werken.

vertaling: Hans Adams — 230279-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de redactie van Elektor via redactie@elektor.com.

Over de auteur

Emanuele Signoretta is geboren in 2000 in Vibo Valentia, een kleine stad in het zuiden van Italië. Hij is een enthousiast ICT-er. Op de middelbare school ontdekte hij de Arduino en het eenvoudige programmeren ervan. Vanaf dat moment schreef hij code voor Arduino- en Fishino-boards. Nu is hij begonnen met STM32- en ESP32-gebaseerde boards. Hij gelooft in de open source-gedachte en gebruikt Linux-gebaseerde distro's. Emanuele studeert momenteel af aan de Politecnico di Torino in elektronica- en communicatietechniek. Verder werkt hij voor Rai, de Italiaanse nationale omroep.

Gerelateerde producten

- **ESP32-C3-DevKitM-1**
  www.elektor.nl/20324
- **Koen Vervloesem, *Getting Started with ESPHome*, Elektor 2021**
  www.elektor.nl/19738
- **Bundle: *Getting Started with ESPHome* + LILYGO TTGO T-Display ESP32 (16 MB)**
  www.elektor.nl/19896

WEBLINKS

[1] ESP32-board: https://futuranet.it/prodotto/esp32-scheda-di-sviluppo-32-gpio-con-wifi-e-bluetooth
[2] Plastic behuizing: https://futuranet.it/prodotto/contenitore-plastico-ermetico-546x784x1182-mm
[3] PZEM-004 datasheet op Github: https://bit.ly/3qTZdHf
[4] InfluxDB-account: https://cloud2.influxdata.com/signup
[5] De GitHub-repository van dit project: http://github.com/signorettae/ESP32-EnergyMeter
[6] ArduinoOTA-bibliotheek: https://github.com/jandrassy/ArduinoOTA
[7] Arduino-bibliotheek voor de nieuwste PZEM-004T: http://github.com/mandulaj/PZEM-004T-v30