Source: Shutterstock

compteur d'énergie basé sur le cloud

avec un module ESP32 et un capteur de tension/courant PZEM-004T

Un projet d'Elettronica In
https://elettronicain.it

Emanuele Signoretta (Italie)

Avec l'augmentation constante des tarifs de l'électricité, la consommation rationnelle et les économies sont devenues une nécessité. Avec un module ESP32 et quelques autres composants, il est possible de réaliser un compteur d'énergie qui envoie nos données énergétiques à la plateforme InfluxDb Cloud via une connexion sur le réseau wifi.

L'Italie, comme beaucoup d'autres pays, dépend fortement des sources d'énergie étrangères parce que sa propre production d'énergie renouvelable n'est pas suffisante pour répondre à la demande nationale. Le problème s'aggrave lorsque des facteurs externes limitent la disponibilité des sources d'énergie fossiles, ce qui entraîne une hausse des prix. Par conséquent, les Italiens, tout comme nous, doivent réduire leur consommation d'énergie, sacrifier certains confort et surveiller de près leur consommation d'énergie afin d'éviter les surprises désagréables sur leurs factures d'électricité et de gaz. La principale motivation pour réduire la consommation d'énergie est la peur de dépenser trop, plutôt que les préoccupations liées au changement climatique.
Pour aider les personnes soucieuses de leur consommation d'électricité, cet article présente un compteur de consommation d'énergie basé sur un module ESP32 d'Espressif et un capteur de tension/courant PZEM-004T. Le compteur collecte des données et les envoie au cloud, plus précisément aux serveurs du service en ligne InfluxDB.

Matériel

Pour le matériel du compteur d'énergie, nous avons besoin d'un module ESP32 Wi-Fi (**figure 1**) et d'un capteur PZEM-004T (que l'on voit encapsulé dans son boîtier en plastique transparent dans la **figure 2**). Nous avons également besoin d'une alimentation à découpage avec une sortie de 5 V (si vous en choisissez une avec une sortie Micro USB, vous pouvez l'utiliser pour alimenter la carte ESP et fournir de l'énergie

Figure 1. Carte ESP32.

Figure 2. Le capteur PZEM-004T avec un transformateur.

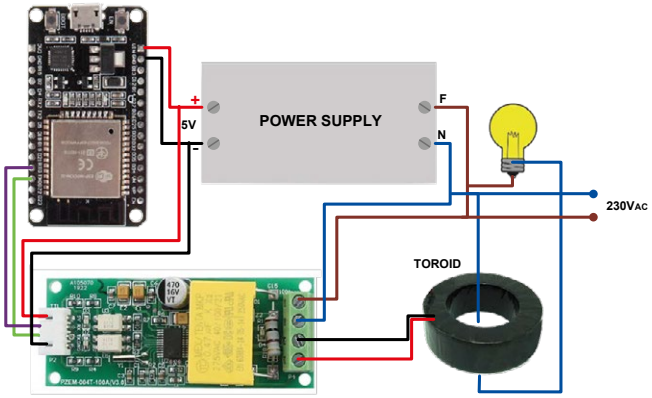

Figure 3. Schéma du circuit.

au PZME-004T à partir du VIN), des cavaliers femelle-femelle pour l'Arduino, des borniers, quelques câbles électriques, et un boîtier en plastique [2]. La fiche technique du capteur est disponible sur [3], tandis que la **figure 3** montre le câblage général du circuit – nous donnons plus de détails à ce sujet dans la partie **Installation** ci-dessous.

Le module comprend un circuit de détection du courant et de tension, logé dans un boîtier en plastique avec un tore ouvert. Il intègre une électronique simple, isolée par des optocoupleurs, et dispose d'une interface série. Le module peut mesurer avec précision des tensions allant de 80 V à 260 V, avec une résolution de 0,1 V et une précision de 0,5 %. De même, il peut mesurer des courants de 0 A à 100 A avec une précision de 0,5 % et une résolution de 0,001 A. En outre, le capteur peut déterminer l'angle de phase (facteur de puissance) entre les vecteurs de tension et de courant avec une précision de 1 %, fournissant des relevés de puissance active et réactive. Cette fonctionnalité nous permet d'évaluer l'efficacité électrique de l'appareil testé. Les composants du circuit ont été sélectionnés selon des critères de rentabilité, compte tenu de la crise actuelle des semi-conducteurs, et des spécifications impressionnantes de l'ESP32.

Configuration d'InfluxDB

De multiples logiciels sont disponibles pour recevoir et analyser les données transmises par les appareils IdO. Parmi les options les plus remarquables, on peut citer Home Assistant, Grafana, Blynk, Thingspeak, et bien d'autres encore. Dans cet article, nous nous concentrerons sur InfluxDB, dont le logo est présenté dans la **figure 4**.

InfluxDB est un logiciel polyvalent qui offre des fonctionnalités telles que la création de tableaux de bord, l'exécution de requêtes et l'envoi d'alertes. Il offre de multiples options de déploiement, notamment des versions exécutables pour diverses architectures, des conteneurs Docker et des solutions basées sur le cloud. Bien qu'initialement nous

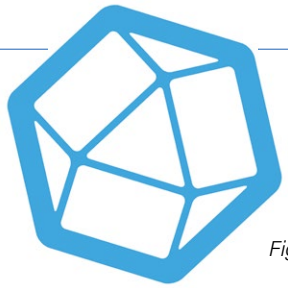

Figure 4. Logo d'InfluxDB.

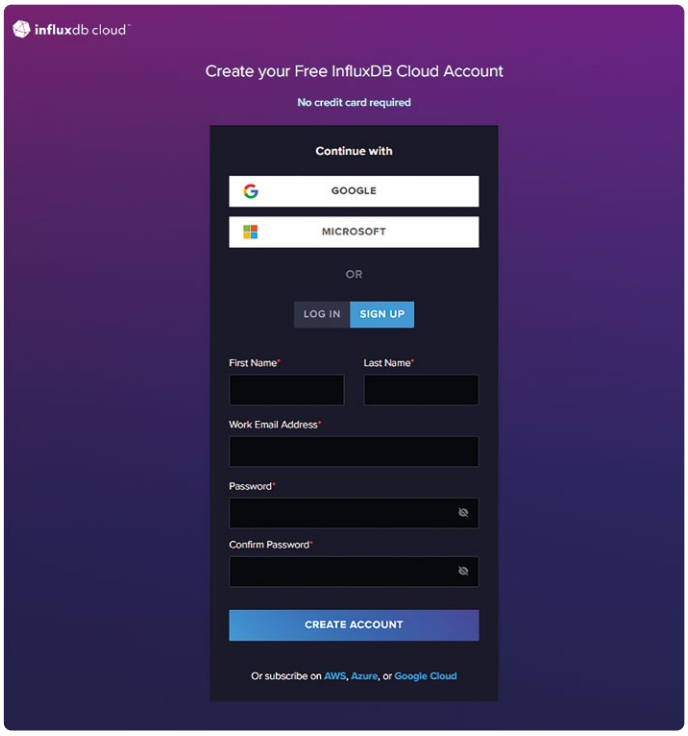

Figure 5. Page d'inscription à InfluxDB

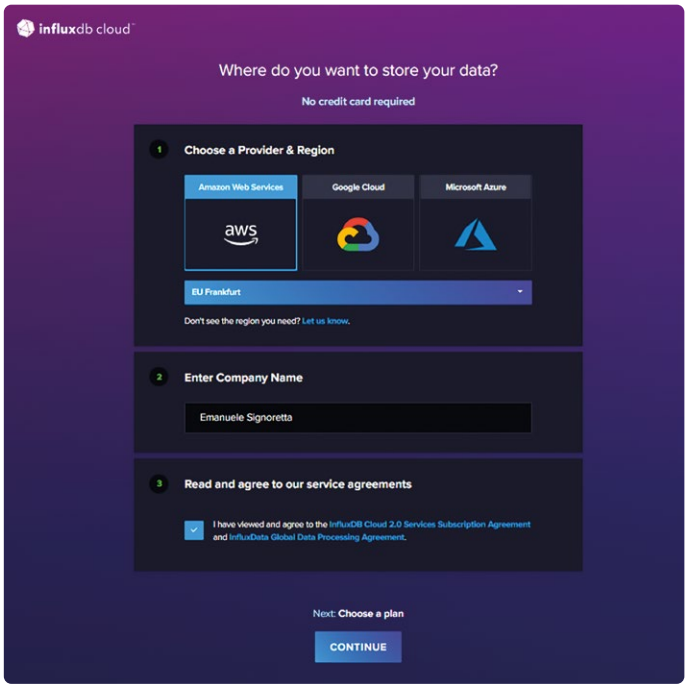

Figure 6. Choix du fournisseur et acceptation des conditions de service.

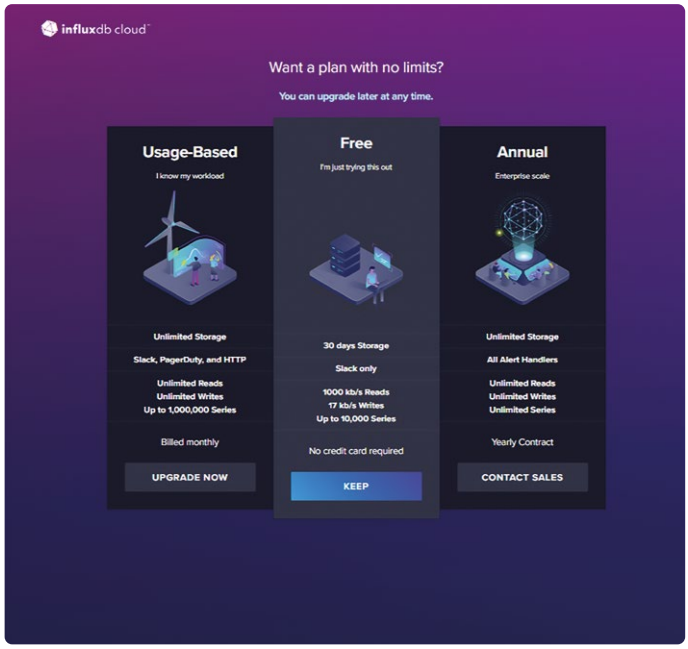

Figure 7. Choix de la formule d'abonnement.

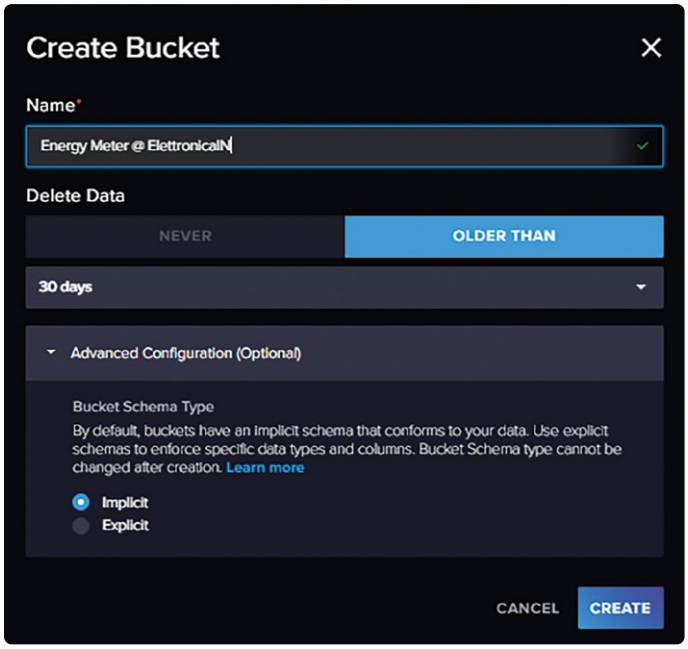

Figure 8. Création d'un nouveau bucket.

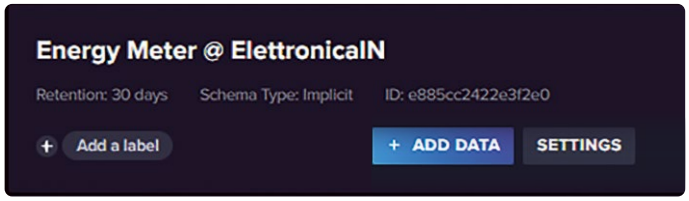

Figure 9. Un bucket récemment créé.

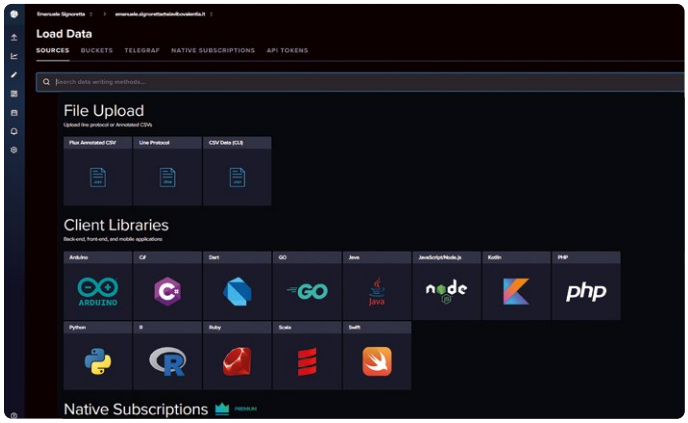

Figure 10. Sélection de la source de chargement de données.

Figure 11. Extrait d'un croquis de démonstration.

ayons eu l'intention d'installer une instance sur un Raspberry Pi pour des raisons de propriété de données, la rareté des composants nous a obligés à adopter plutôt le service InfluxDb Cloud v2.
Pour démarrer, visitez le lien [4] et enregistrez-vous. Vous pouvez choisir de vous inscrire en utilisant vos identifiants Microsoft ou Google, ou de remplir manuellement les champs requis, comme le montre la **figure 5**. Après avoir terminé le processus d'inscription, vous devrez voir une page ressemblant à la **figure 6**, où il vous sera demandé de fournir divers détails, y compris le nom du fournisseur de services. Pour notre projet, nous avons sélectionné Amazon Web Services (AWS) comme fournisseur.
Sur la page suivante (illustrée à la **figure 7**), vous devrez sélectionner un plan d'utilisation. Dans notre cas, nous avons opté pour le plan gratuit, qui permet de stocker les données pour une durée de 30 jours et de recevoir des notifications via Slack.
Après avoir confirmé notre sélection, le tableau de bord personnel s'affiche. Pour créer un bucket (conteneur), naviguez vers *Load Data Buckets Create new bucket*. En procédant ainsi, une page similaire à celle de la **figure 8** s'ouvrira. Remplissez le champ *Name* et cliquez sur le bouton *Create*. Une fois le bucket créé avec succès, il apparaîtra parmi les sources de données disponibles, comme le montre la **figure 9**. Cliquez ensuite sur *Add data Client library*. Une nouvelle page s'affiche, proposant différentes options pour le chargement de données. Parmi ces options, sélectionnez *Arduino* (**figure 10**). Dans la fenêtre suivante (**figure 11**), vous verrez une série d'extraits de code qui constituent un croquis de démonstration. Copiez les données fournies pour `INFLUXDB_URL`, `INFLUXDB_ORG`, et `INFLUXDB_BUCKET` à partir du premier extrait.
Pour terminer la configuration, nous devons générer un jeton d'accès pour le bucket. Pour ce faire, accédez à la barre latérale de gauche et naviguez à *Load Data API Tokens*.
Dans la fenêtre qui vient de s'ouvrir, cliquez sur *Generate API Token* (Générer un jeton API). Après avoir sélectionné le bucket, activez les autorisations de lecture et d'écriture, comme indiqué dans la **figure 12**. Une fois le jeton généré, copiez-le et enregistrez-le, car nous en aurons besoin dans les étapes suivantes.

Croquis

Le croquis de notre projet est une combinaison de plusieurs autres croquis provenant des bibliothèques que nous avons utilisées. Vous pouvez télécharger les fichiers du croquis depuis le dépôt GitHub [5]. Analysons maintenant notre code, qui est divisé en trois sections représentées avec des listages correspondants.

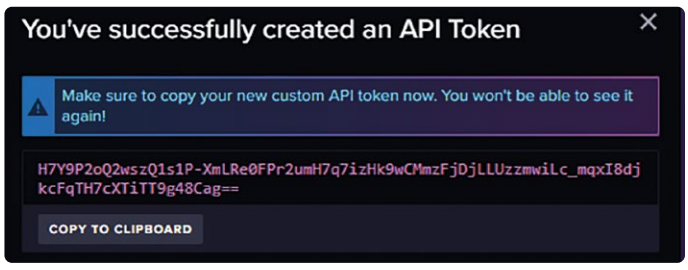

Figure 12. Création d'une clé API.

Listage 1. Inclusion des bibliothèques

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

Commençons par l'inclusion des bibliothèques et dépendances nécessaires, comme le montre le **listage 1**.
À cet égard, il est important de noter que la bibliothèque *WiFiMulti* est utilisée pour gérer la communication wifi entre l'ESP32 et le point d'accès. Les bibliothèques *ESPmDNS*, *WiFiUDP*, et *ArduinoOTA* [6] sont utilisées pour le chargement à distance (*OTA*) des croquis et la gestion des noms d'hôtes. Veillez à ce que la version 2.7.x de Python soit installée sur votre PC pour utiliser cette fonctionnalité. Les bibliothèques *InfluxDbClient* et *InfluxDbCloud* sont utilisées pour le transfert de données vers InfluxDB Cloud. La bibliothèque *PZEM004Tv30* [7] facilite la communication série avec le capteur. Enfin, la bibliothèque *Every* permet l'exécution des blocs de code à intervalles réguliers sans avoir recours à la fonction `delay()`.
Examinons maintenant les extraits de code suivants, en commençant par le **listage 2**, qui contient la section pré-processeur. En utilisant `#define REFRESH_TIME 5000`, nous définissons l'intervalle (en millisecondes) pour le chargement de données vers le cloud. Nous définissons ensuite le nom de l'appareil, les broches de communication série et le port série choisi. Les objets relatifs au réseau wifi et au capteur sont instanciés. Les adresses IP sont définies, et si vous souhaitez configurer une connexion au point d'accès avec une IP statique, vous pouvez modifier les paramètres en fonction du sous-réseau de votre réseau. Enfin, nous définissons les paramètres pour la connexion wifi et l'accès à InfluxDB Cloud. Remplacez ces lignes de code par l'extrait copié et saisissez les données manquantes.
Nous spécifions respectivement le SSID et la phrase de passe associés au réseau wifi avec `#define WIFI_SSID` et `#define WIFI_PASSWORD`. Ensuite, nous définissons les paramètres de connexion à InfluxDB Cloud. Le seul paramètre manquant est `INFLUXDB_TOKEN`. Vous pouvez le récupérer à partir de la clé API que nous avons créée précédemment. Insérez la clé API entre les guillemets. Nous spécifions le fuseau horaire de l'Europe centrale pour les horodatages avec `#define TZ_INFO "CET-1CEST,M3.5.0,M10.5.0/3"`.
Pour créer l'objet client permettant de se connecter au serveur InfluxDB, utilisez le code suivant :

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

Listage 2. Définitions

```
#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");
```

Listage 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());
  }
}
```

Enfin, l'objet *sensor* est créé avec `Point sensor("EnergyMeter")`. Cet objet, nommé *EnergyMeter*, sera associé à toutes les données collectées par le capteur. Le **listage 3** représente le code de la configuration de la carte et initialise le port série à une vitesse de transmission de 115 200.
Le bloc de code commence par `WiFi.mode(WIFI_STA)`, qui initialise le wifi en mode *station* afin de se connecter à un point d'accès. Le bloc de code suivant :

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

définit une adresse IP statique au lieu d'utiliser le protocole DHCP. En cas d'erreur, un message d'avertissement est affiché via le port série. Si vous souhaitez utiliser DHCP, vous pouvez mettre cette partie du code en commentaire.
L'appel de la fonction `wifiMulti.addAP(WIFI_SSID, WIFI_PASSWORD)` effectue une tentative de connexion au point d'accès avec SSID et la phrase de passe spécifiés.
Une connexion à l'AP est alors tentée et, en cas d'échec, une erreur est d'abord affichée via le port série, puis la carte redémarre après un délai de 5 secondes.
Le nom d'hôte est défini avec `ArduinoOTA.setHostname("ESP32-Energy Meter")`, qui sera transmis via mDNS et affiché sur l'EDI Arduino pour le chargement du croquis à distance (OTA). Il y a du code commenté pour restreindre le chargement OTA du croquis aux utilisateurs possédant un nom d'utilisateur et un mot de passe. Décommentez ces lignes de code pour ajouter cette fonction de sécurité.
Les paramètres du mode OTA sont gérés par la séquence de code suivante :

```
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();
```

Le bloc de code gère le démarrage, la fin, le chargement du croquis et les erreurs de la transmission OTA dans l'ordre. `ArduinoOTA.begin()` démarre le processus de chargement OTA.
Avec `sensor.addTag("Device", DEVICE)` ; un tag (étiquette) est défini, qui dans ce cas correspond au nom de l'appareil. Cette fonctionnalité peut être utile pour distinguer un ou plusieurs capteurs au sein d'une flotte.
L'ESP se connecte avec `timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov")` aux deux serveurs NTP (Network Time Protocol) *pool.ntp.org* et *time.nis.gov* afin d'obtenir la date et l'heure.
Enfin, le bloc de code tente de se connecter aux serveurs InfluxDB :

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

Si la connexion est réussie, l'URL du serveur s'affiche sur le port série. Sinon, le message d'erreur correspondant s'affiche.
Passons maintenant au **listage 4**, qui contient le code de la boucle `loop()` de notre croquis. En utilisant `ArduinoOTA.handle()`, l'ESP32 attend la réception OTA de tout croquis compilé.
Dans `EVERY(REFRESH_TIME) { … }` le code entre crochets est exécuté à intervalles réguliers et sans délai. L'intervalle de temps a été défini au préalable. Dans le bloc `EVERY`, l'adresse du PZEM est d'abord lue et affichée sur le port série.
Le code du bloc `EVERY(REFRESH_TIME) { … }` est exécuté à des intervalles réguliers définis précédemment, sans utiliser de délais. Dans ce bloc, l'adresse du PZEM est lue et affichée sur le port série. Ensuite, on crée et initialise les variables des capteurs avec les fonctions de la bibliothèque du capteur. Ces variables sont notamment la tension, le courant, la puissance active (en Wh), l'énergie active (en kWh), la fréquence et le facteur de puissance (un rapport entre -1 et 1, représentant le rapport entre la puissance active et la puissance apparente). Si l'une des variables n'est pas un nombre, une erreur de lecture est affichée sur le port série. Dans le cas contraire, les relevés des capteurs seront affichés. Ensuite, les champs initialisés et les horodatages sont effacés et les variables sont préparées pour le chargement vers le cloud. Enfin, la connexion wifi et le bon chargement de données sont vérifiées. Si le chargement échoue, la dernière erreur est affichée sur le port série.

Installation

Nous avons prévu de placer l'alimentation, l'ESP32 et le module principal du capteur dans une boîte en plastique, en ne laissant passer que les fils de l'alimentation et ceux qui relient le tore du capteur de courant à la carte. Si vous voulez mesurer la consommation globale d'énergie de votre maison, l'entrée d'alimentation de notre système de mesure de l'énergie doit être connectée aussi près que possible de la ligne CA sortant du panneau électrique de la maison. Sinon, comme l'illustre la figure 3, il est toujours possible de mesurer l'énergie consommée par un seul appareil électrique. L'ouverture à charnière du tore permet de serrer très commodément le câble électrique à mesurer. Dans le cas de câbles électriques tripolaires (phase, neutre et terre), la gaine isolante extérieure doit être soigneusement coupée dans le sens de

Listage 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();
  }
}
```

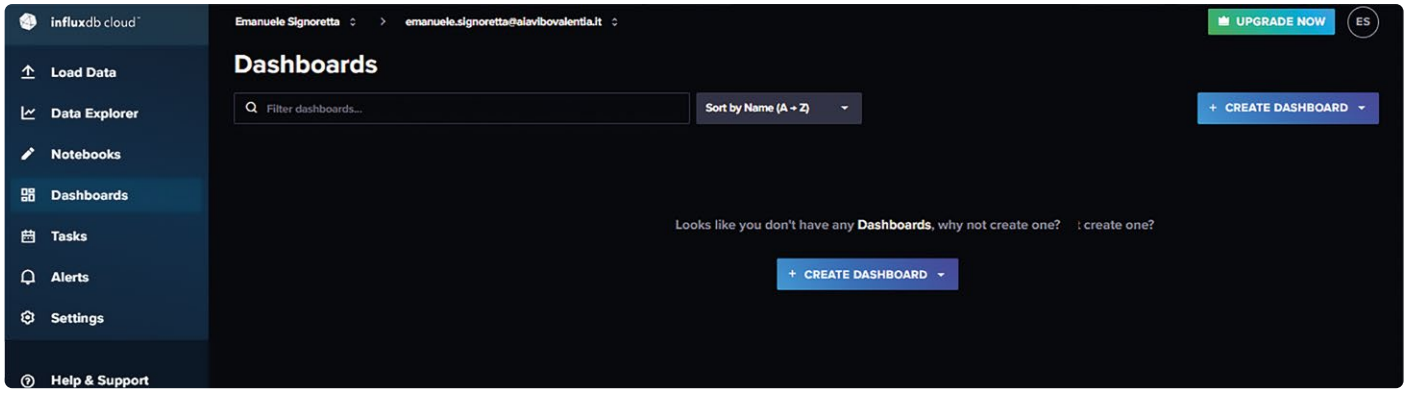

Figure 13. Création du tableau de bord.

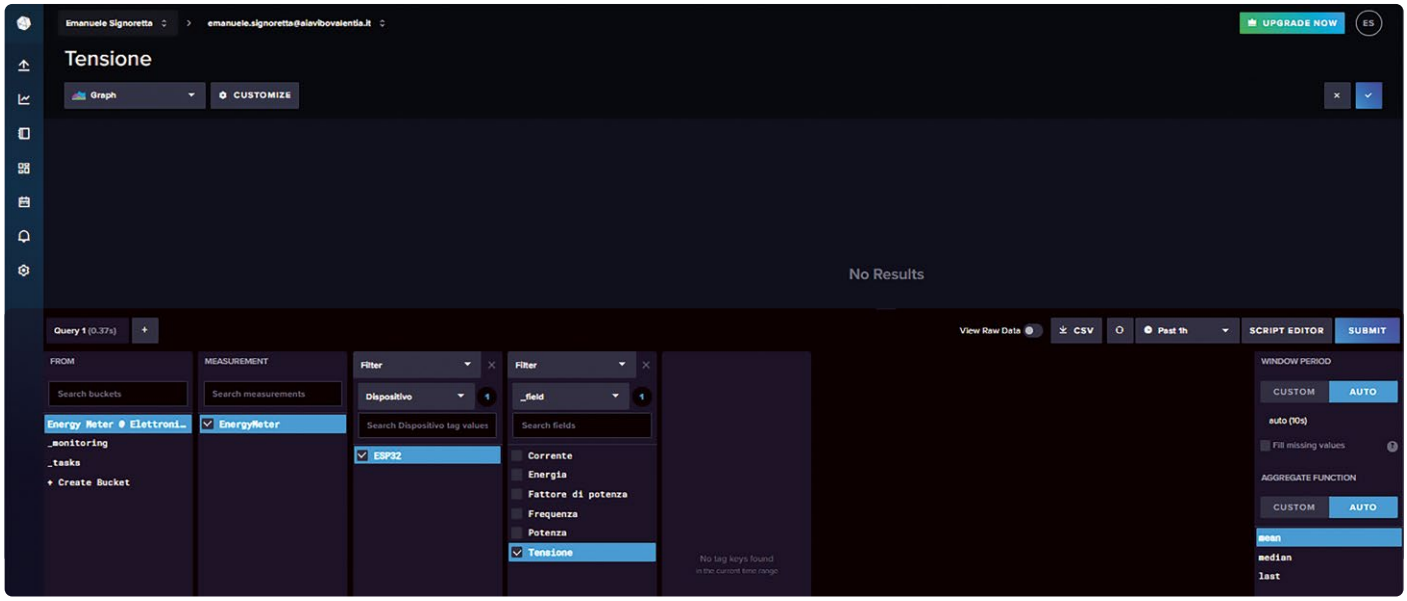

Figure 14. Ajout des graphiques du tableau de bord.

la longueur (**débranchez le au préalable !**), sans affecter l'isolation des trois câbles intérieurs. Ils seront séparés et seul le fil sous tension (normalement noir ou marron) sera placé à l'intérieur du tore de mesure. Pour installer le compteur d'énergie, il suffit de suivre le schéma déjà présenté au début de cet article, dans la figure 3.

En particulier, vous devez connecter deux fils en parallèle à l'entrée de l'alimentation et entre les broches N et L du PZEM-004T. **Soyez toujours très prudent, vous travaillez sur les fils du secteur 230 V(CA) !**

Ensuite, nous libérons le cran de sécurité du tore et faisons passer l'un des deux fils reliés au réseau électrique, puis nous connectons les fils du tore aux broches CT du capteur. En outre, nous connectons le module principal du capteur à l'ESP32, comme indiqué dans le **tableau 1**. Pour alimenter l'ESP, vous pouvez utiliser une alimentation avec une sortie Micro USB et fournir les 5 V pour le capteur à partir de VIN et GND. Une fois le croquis chargé, nous pouvons enfermer toute l'électronique dans la boîte et démarrer le système.

Démarrage et configuration du tableau de bord

Une fois la carte allumée, nous retournons sur le site web d'InfluxDB et, à partir de notre tableau de bord, nous accédons à *Dashboards Create new dashboard*. En procédant ainsi, nous obtenons un écran comme celui de la **figure 13**, où nous attribuons un nom à notre tableau de bord et cliquons sur *Add Cell*. Dans la nouvelle fenêtre qui apparaît (**figure 14**), nous allons sélectionner les paramètres à inclure dans chacun des graphiques constituant le tableau de bord. Nous sélectionnons, dans l'ordre, le bucket, les mesures, l'appareil et enfin les valeurs à afficher. Vous pouvez choisir les graphiques à utiliser pour les valeurs : carte thermique, jauge, graphique simple, tableau, etc. ainsi que les valeurs de mise à l'échelle à utiliser. On personnalise les cellules selon nos préférences et on répète la procédure pour chaque valeur que l'on souhaite afficher jusqu'à obtenir un résultat comme celui visible dans la **figure 15**, qui illustre notre tableau de bord complet.

Tableau 1. Connexions entre le compteur d'énergie et les modules ESP32

Broche du PZEM-004	Broche de l'ESP32
VCC	V5
GND	GND
RX	26
TX	27

Figure 15. Le tableau de bord complet.

Conclusion

Ceci conclut la description de notre simple mais puissant compteur d'énergie. La polyvalence de la carte ESP32 et les nombreuses caractéristiques des protocoles et plates-formes réseau pris en charge nous permettent de développer ses applications en fonction de nos besoins, par exemple en intégrant le compteur dans un système de gestion de charge électronique, tout en conservant sa capacité d'action autonome.

230279-04

Produits

- **ESP32-C3-DevKitM-1**
  https://elektor.fr/20324
- **Koen Vervloesem, *Getting Started with ESPHome*, Elektor 2021**
  https://elektor.fr/19738
- **Offre groupée : Livre « *Getting Started with ESPHome* » + LILYGO TTGO T-Display ESP32 (16 Mo)**
  https://elektor.fr/19896

Questions ou commentaires?

Contactez Elektor (redaction@elektor.fr).

À propos de l'auteur

Emanuele Signoretta est né en 2000 à Vibo Valentia, une petite ville du sud de l'Italie. Il est passionné par les TIC. Étudiant, il a découvert Arduino et sa simplicité de programmation. Il a commencé par écrire du code pour les cartes Arduino et Fishino. Aujourd'hui, il s'intéresse aux cartes STM32 et ESP32. Il croit en l'éthique des logiciels libres et utilise des distributions basées sur Linux. Emanuele termine actuellement sa licence en ingénierie électronique et de communication au Politecnico di Torino. Il travaille également pour Rai, le radiodiffuseur national italien.

LIENS

[1] Carte ESP32 : https://futuranet.it/prodotto/esp32-scheda-di-sviluppo-32-gpio-con-wifi-e-bluetooth
[2] Boîtier en plastique : https://futuranet.it/prodotto/contenitore-plastico-ermetico-546x784x1182-mm
[3] Fiche technique du PZEM-004 Datasheet sur Github : https://bit.ly/3qTZdHf
[4] Compte InfluxDB : https://cloud2.influxdata.com/signup
[5] Le dépôt GitHub du projet : http://github.com/signorettae/ESP32-EnergyMeter
[6] Bibliothèque ArduinoOTA : https://github.com/jandrassy/ArduinoOTA
[7] Bibliothèque Arduino pour le PZEM-004T mis à niveau : http://github.com/mandulaj/PZEM-004T-v30