My IoT-Button:
de knop voor het net
deel 2: prototyping met board en cloud

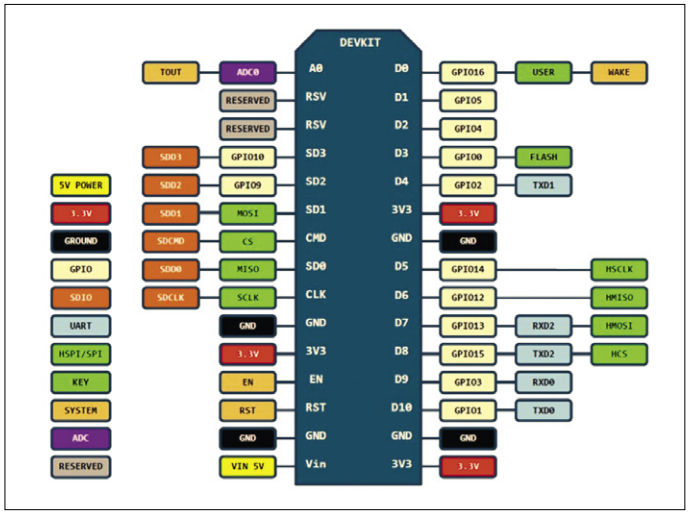
Dr. Veikko Krypczyk (Duitsland)

Het idee van een IoT-button is dat u op afstand communiceert met een dienst in de cloud met behulp van minimalistische hardware. Het is daarvoor nodig om een verbinding met het lokale netwerk tot stand te brengen. Kleine, goedkope microcontrollerboards zoals de ESP8266 NodeMCU zijn voorbestemd voor deze taak. We willen hier ingaan op de belangrijkste aspecten aan de kant van het IoT-apparaat en presenteren ideeën voor eigen projecten. Het mooie is dat het allemaal vrij gemakkelijk te realiseren is en snel tot resultaten leidt. Dat biedt ruimte voor eigen ideeën.

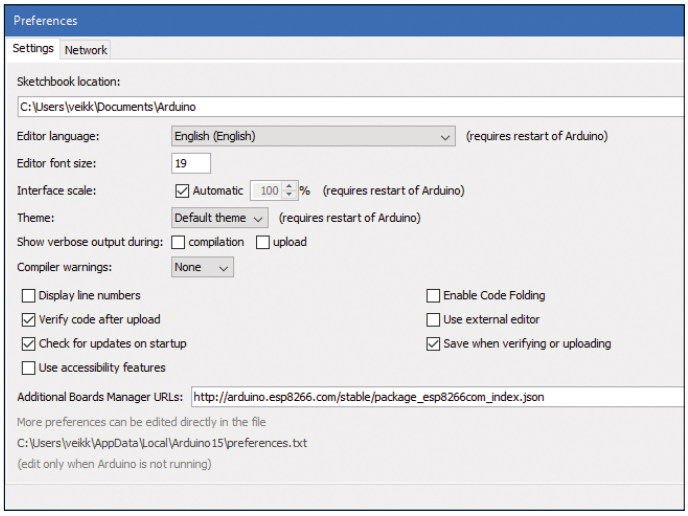
Het Internet of Things (IoT) maakt alle denkbare apparaten 'intelligent' en verbindt ze via het internet met de buitenwereld. In het eerste deel van het artikel hebben we de basisarchitectuur van IoT-toepassingen laten zien: die bestaat uit sensoren, actuatoren en een server als backend. We hebben gezien dat we meestal niet zelf de server-infrastructuur hoeven te bouwen en te programmeren: er is een heel divers server-aanbod in de cloud. De communicatie vindt plaats via het internet. Als u gebruik maakt van gevestigde standaarden, is het allemaal heel eenvoudig. Voor requests kunnen we gebruik maken van een gevestigde standaard: de REST-interface. De client, in dit geval het IoT-apparaat, kan gegevens verzenden of opvragen naar resp. van diensten (op een server). Deze tweede aflevering gaat over de hardware en software aan de kant van het IoT-apparaat.

Selectie van hardware

Welke hardware willen we gebruiken? In principe zijn er een paar mogelijkheden, bijvoorbeeld een minicomputer zoals de **Raspberry Pi**. Naast WLAN biedt de RPi ook voldoende rekenkracht voor alle toepassingen. Als we meerdere knoppen, sensoren en actuatoren met verschillende functies op één plaats



Figuur 1. Pinbezetting van het ESP8266 NodeMCU microcontrollerboard.



Figuur 2. Configuratie van de Arduino-IDE voor het programmeren van de ESP8266.

willen hebben, kunnen die allemaal worden aangesloten op de GPIO-pennen van de Raspberry Pi. De RPi gebruikt dan het netwerk om verbinding te maken met de dienst in de cloud. Maar voor eenvoudige taken is een Raspberry Pi te groot qua afmetingen. Bovendien vraagt hij om een permanente stroomvoorziening met een netvoeding.
Laten we eens kijken naar de **Arduino Uno**! Zoals bekend is dit geen complete computer, maar slechts een microcontrollerboard. Maar de hard- en software mogelijkheden zijn uitgebreid en er zijn talloze bibliotheken en toepassingsvoorbeelden. Het programmeren gaat heel eenvoudig met de Arduino-IDE. Maar een Arduino is niet geschikt om verbinding te maken met een WLAN, dus daar zouden we dan een extra module voor moeten toevoegen.
Maar het is nog gemakkelijker met een **ESP8266 NodeMCU** microcontrollerboard (zie [1] en **kader**), dat heeft alle noodzakelijke functies aan boord, met name een WLAN-interface (**figuur 1**).
We sluiten gewoon een drukknop aan tussen de D7-connector op de print en GND. Als de knop wordt ingedrukt, melden we dat via het WLAN aan de cloud. Er zit al een status-LED op het board, dus we hebben verder geen onderdelen nodig. De stroom wordt (voorlopig) geleverd via de micro-USB-poort van de computer.

Het inrichten van de ESP8266 NodeMCU

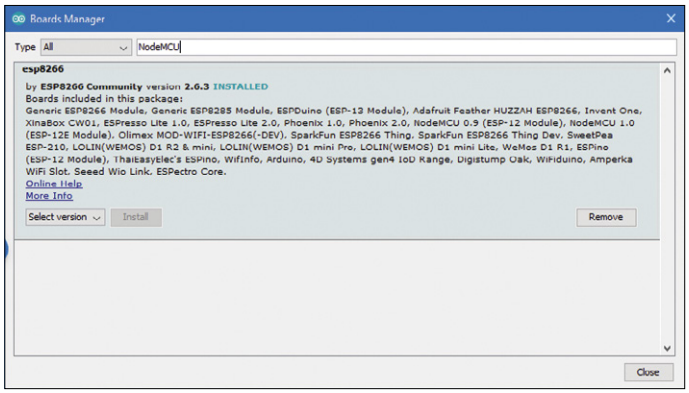
ESP8266-microcontrollers kunnen, net als Arduino's, worden geprogrammeerd met de Arduino-IDE. Sluit het board met een USB-kabel aan op de PC en installeer, indien nodig, het stuurprogramma voor de chipset CH340 [2]. Als u dat niet allang gedaan hebt, installeer dan de Arduino-IDE op uw computer [3].
De IDE moet worden uitgebreid met een extra component: Voer onder *File -> Preferences* de link [4] in voor *Additional Board Managers URL's* (**figuur 2**). Sluit het dialoogvenster met OK. Ga dan naar het menu-item Tools, waar *Generic ESP8266 Module* te vinden moet zijn. Selecteer deze en ga dan naar *Board Manager* bovenin het popup-menu, zoek dan naar *NodeMCU* en voer de installatie uit (**figuur 3**).
We kunnen testen of dat allemaal gelukt is met een eenvoudig "Hello World"-programma (**listing 1**). Kopieer de code en upload die naar de microcontroller met behulp van *Sketch | Upload*. De LED op het board wordt door de kleine lus gedurende 3 s uitgeschakeld met `digitalWrite(LED, HIGH)` en met `digitalWrite(LED, LOW)` gedurende 1 s ingeschakeld.
Voor het opvragen van een drukknop (op pin D7) is ook erg weinig code nodig. Bijvoorbeeld:

```
int button=D7;
status=digitalRead(button);
if (status==LOW)
{ …… }
```

Als de knop is ingedrukt, wordt de code in het blok in het if-statement uitgevoerd.
Nu hoeft de ESP8266 alleen nog maar toegang te krijgen tot het internet. De hardware daarvoor zit al op de ESP8266-print in de vorm van een kleine (grijsgroene) WLAN-module (**figuur 4**). Als u het WLAN van de ESP8266 wilt gebruiken, moet u bovenin de broncode een bibliotheek meenemen met de regel `#include ESP8266WiFi.h`. De bibliotheek heeft de juiste gegevens nodig om verbinding te maken met het internet: de SSID en het

Listing 1. "Hello World" (knipperende LED) op de ESP8266.

```
#define LED D4
void setup()
{
     pinMode(LED, OUTPUT);
}

void loop()
{
    digitalWrite(LED, HIGH);
    delay(3000);
    digitalWrite(LED, LOW);
    delay(1000);
}
```



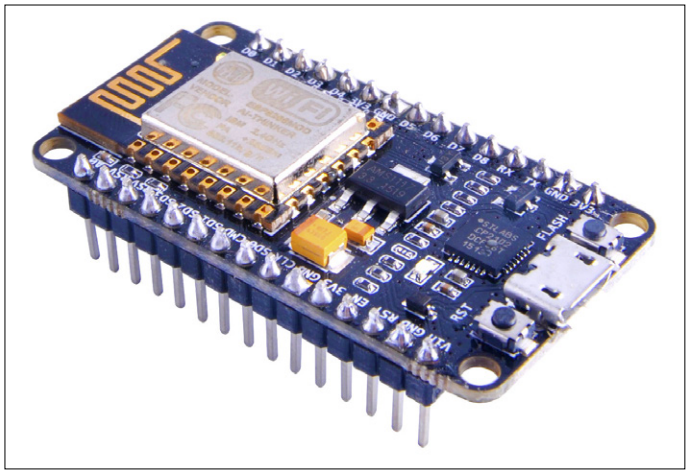
Figuur 3. Installatie van de bibliotheken voor het ESP8266 NodeMCU board.

De ESP8266 NodeMCU

Bij het microcontrollerboard ESP8266 NodeMCU gaat het om een ESP8266-chip met een kloksnelheid van 80 MHz en tot 4 MB flashgeheugen. De ESP8266 NodeMCU bevat een WLAN-module en een USB-interface inclusief UART/USB-converter (CH340G). Het kleine kaartje heeft de volgende technische gegevens:

- Ondersteunt WPA/WPA2
- Chip: ESP8266
- Processor: Tensilica L106 met een kloksnelheid van 80 MHz
- Systeemarchitectuur: 32-bit. Flashgeheugen: tot 4 MB
- Ondersteunde protocollen: SPI, UART, I^2C, I^2S, IR-afstandsbediening, PWM, GPIO
- Logisch niveau: 3,3 V
- Energieverbruik: < 10 µA in de slaapstand en tot 200 mA onder belasting
- Aansluiting voor voeding en programmering via micro-USB: 5 V
- Programmeerchip: CH340G
- Afmetingen: 5,5 x 3,1 x 1,2 cm; gewicht: 42 g

Met deze specificaties is het ESP8266 NodeMCU-board zeer geschikt voor kleine en vooral energiezuinige IoT-apparaten.



Figuur 4. ESP8266 NodeMCU met WLAN-module.

Listing 2. Verbinding maken met het internet.

```
include <ESP8266WiFi.h>
void setup()
{
    Serial.begin(115200);

    WiFi.begin(<SSID>, <Password>);

    while (WiFi.status() != WL_CONNECTED);
    {
        delay(1000);
        Serial.print("Connecting ...");
    }
}
```

wachtwoord. Voorlopig nemen we die gegevens statisch op in de broncode, maar later moet de gebruiker ze natuurlijk kunnen configureren. Het tot stand brengen van de WLAN-verbinding is heel eenvoudig (**listing 2**).
Het is nu de bedoeling dat we bij een event een bericht over het netwerk versturen. De ontvanger van dat bericht is een REST-eindpunt, zoals we hebben beschreven in het eerste deel van dit artikel. Dat eindpunt ontvangt de gegevens en kan deze verder verwerken en een reactie initiëren. Met een eenvoudige IoT-button hoeven geen andere gegevens worden verzonden dan het event “ingedrukt”. Als we de toepassing verder uitbreiden, kunnen via de REST-aanvraag aanvullende gegevens als parameters worden doorgegeven. Laten we het eens gaan proberen!

Gegevens naar de cloud

We hebben een REST-eindpunt nodig om gegevens naar de cloud, dat wil zeggen naar een externe server, te sturen. U hoeft dit niet zelf te programmeren, u kunt gebruik maken van bestaande diensten. Voor het testen en voor privétoepassingen zijn deze clouddiensten meestal gratis.
We gaan experimenteel en stap voor stap te werk. Onder [5] kunt u voor testdoeleinden een gratis REST-API aanmaken, die via WLAN toegankelijk is voor onze microcontroller. Dat gaat met een paar muiskliks, zelfs zonder voorafgaande registratie. Kijk eens naar [6]; we hebben zelfs al een eindpunt gemaakt (**figuur 5**)! Om te testen of en hoe we dit eindpunt kunnen bereiken vanuit onze microcontroller, sturen we een GET-request naar de service als de knop wordt ingedrukt. We voegen dit toe in onze programmalus (**listing 3**).
Met de volgende programmaregel controleert de microcontroller of er verbinding is met het internet:

```
if (WiFi.status() == WL_CONNECTED)
{ …… }
```

Listing 3. Een GET-request naar een REST-eindpunt zenden.

```
void loop()
{
  if (WiFi.status() == WL_CONNECTED)
  {
     status=digitalRead(button);
     if (status==LOW)
     {
      digitalWrite(LED, LOW);

      BearSSL::WiFiClientSecure client;
      client.setInsecure();
      HTTPClient https;
      https.begin(client, "https://iotbutton.free.
                           beeceptor.com");
      int httpCode = https.GET();
      if (httpCode == 200)
      {
        String payload = https.getString();
        Serial.println(payload);
      }
      else
      {
        Serial.print("Error: ");
        Serial.println(httpCode);
      }
      https.end();

     }
     else
     {
      digitalWrite(LED, HIGH);
     }

  }
}
```

Alle regels van de broncode tussen de accolades worden dus alleen verwerkt als er een verbinding met het internet is.
De volgende stap

```
status=digitalRead(button);
    if (status==LOW)
```

controleert of de knop is ingedrukt. Als dat het geval is, wordt een http-client geïnitialiseerd en wordt de optie om *https*-commando's te verzenden geactiveerd. Hiervoor is een bibliotheek nodig, die bovenin het programma met `#include` moet worden meegenomen. Als URL voor ons REST-eindpunt wordt de waarde `https://iotbutton.free.beeceptor.com` ingesteld. Daarna volgt een eenvoudig GET-request naar het eindpunt, waarvan we het resultaat in de vorm van een tekenreeks ontvangen met de regel

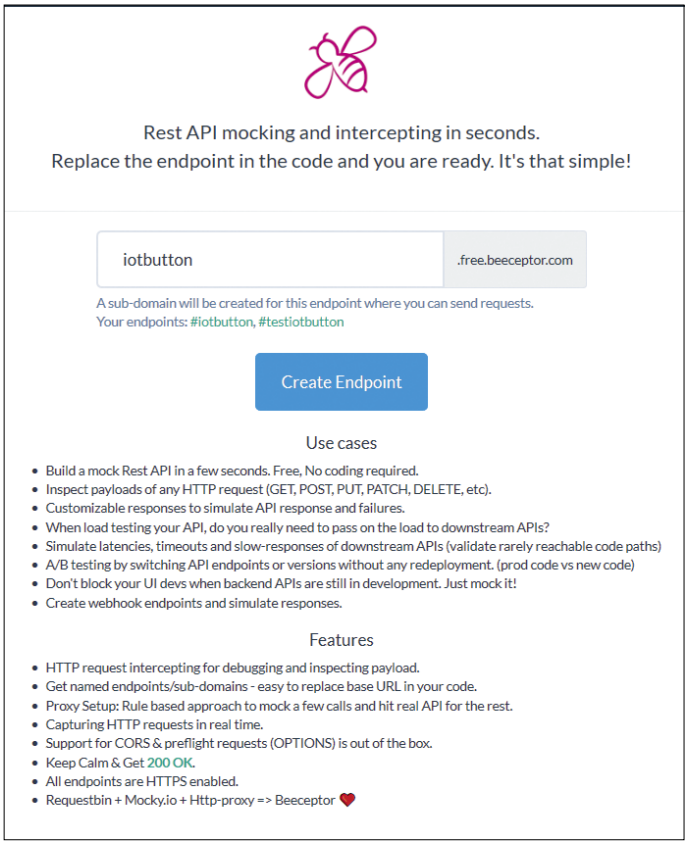
```
String payload = https.getString();
```

Als directe reactie op het indrukken van de toets moet de LED op de print worden ingeschakeld en na het loslaten moet hij weer worden uitgeschakeld (zie boven). Of de gegevens ook in de cloud zijn aangekomen, is online te controleren. Ga naar het eindpunt op de Beeceptor-pagina [6] en volg de requests aan het eindpunt (**figuur 6**).
Houd er rekening mee dat u slechts 50 aanvragen per dag gratis kunt versturen via deze service. Tijdens het experimenteren hebben we dat aantal af en toe overschreden, zodat we een foutmelding met de code 429 kregen als antwoord op onze GET-aanvraag. Maar zolang u het maximum aantal aanvragen niet overschrijdt, worden de statuscode 200 (OK) en de gegevens van de REST-aanvraag teruggestuurd naar de microcontroller. In de ontwikkelomgeving kunt u met de seriële monitor zien welke retourwaarden de microcontroller ontvangt (*Tools -> Serial monitor*) (**figuur 7**).
We hebben dus met één druk op de knop een REST-request verstuurd via het internet. Tot zover de technische kant van de IoT-button. In **listing 4** ziet u de volledige broncode. De plaatsen die van geval tot geval moeten worden aangepast zijn vetgedrukt.

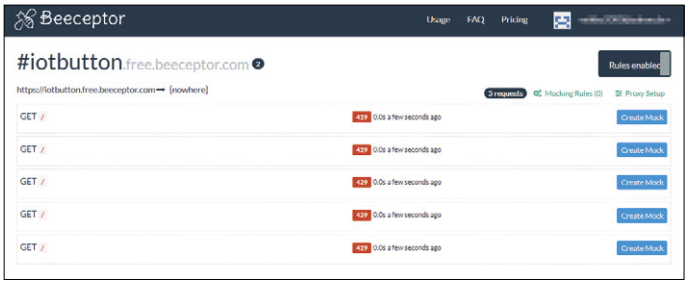
Stroomverbruik en slaapstand

We hebben vooral gekozen voor het ESP8266-microcontrollerboard vanwege het lage stroomverbruik. In de praktijk kunnen we uitgaan van de in [7] genoemde waarden. Met een eenvoudig Arduino-programma bedraagt dit ongeveer 50...70 mA, bij gebruik van WLAN neemt het stroomverbruik toe tot ongeveer 80...170 mA en gedurende enkele milliseconden kan het board zelfs tot 400 mA trekken. Daarom is het interessant dat we de energiehonger aanzienlijk kunnen verminderen met de *deep sleep*-modus. Volgens sommige datasheets en informatiebronnen op het internet gaat het dan om maar een paar microampère. Dit betekent dat een batterij (of accu) voldoende moet zijn voor maandenlang autonoom bedrijf. De voedingsspanning wordt aangesloten op +5V en GND, bijvoorbeeld in de vorm van een powerbank. Er zijn twee varianten van de deep sleep-modus.

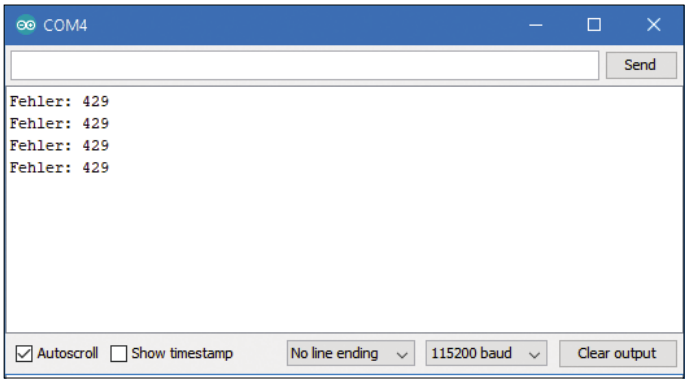
- *Software Deep Sleep*: de ESP8266 wordt na een bepaalde tijd automatisch wakker. Om dit te doen moet er een verbinding worden gemaakt van GPIO16 (D2) naar de



Figuur 5. REST-eindpunt voor testdoeleinden in de Beeceptor-cloud.



Figuur 6. Gelogde requests aan het REST-eindpunt in de Beeceptor-cloud.



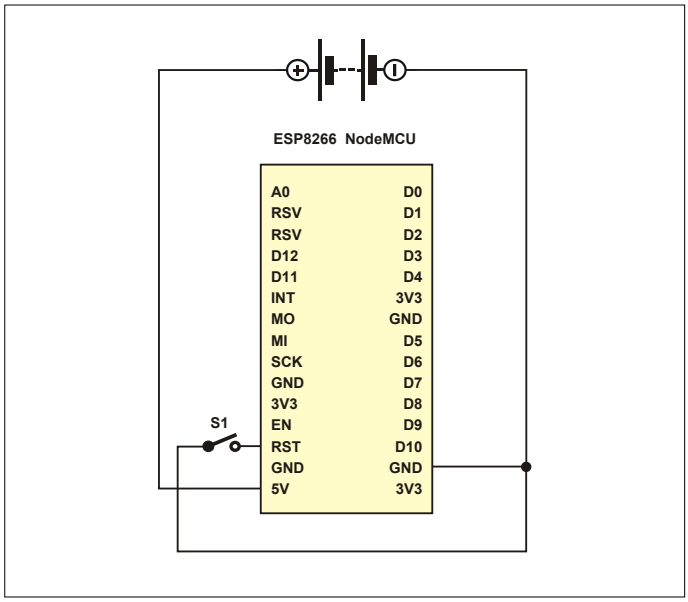
Figuur 7. De seriële monitor registreert de reacties van het REST-eindpunt. Hier wordt de foutmelding 429 gegeven omdat het maximale aantal aanvragen is overschreden.

Listing 4. Een GET-request via REST verzenden.

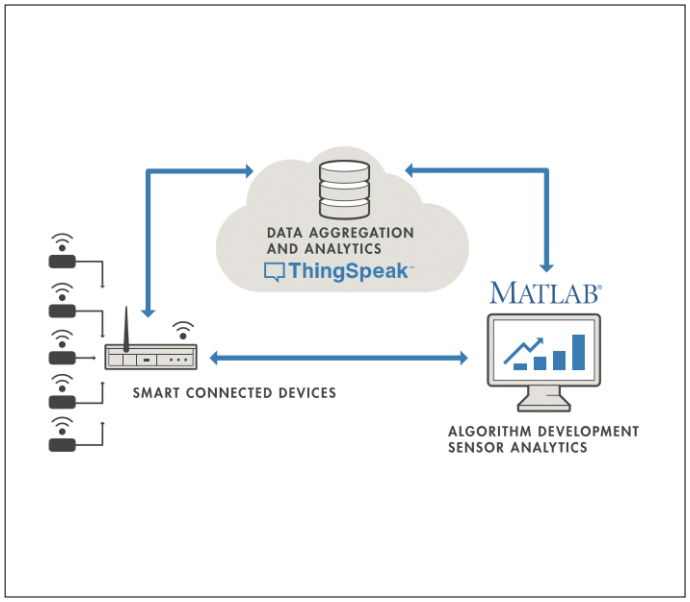
```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#define LED D4
int button=D7;
int status=1;
void setup()
{
    Serial.begin(115200);
    pinMode(LED, OUTPUT);
    digitalWrite(LED, HIGH);
    WiFi.begin(<SSID>, <Password>);
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(1000);
        Serial.print("Connecting ...");
    }
}

void loop()
{
  if (WiFi.status() == WL_CONNECTED)
  {
     status=digitalRead(button);
     if (status==LOW)
     {
      digitalWrite(LED, LOW);
      BearSSL::WiFiClientSecure client;
      client.setInsecure();
      HTTPClient https;
      https.begin(client, "https://iotbutton.free.
                           beeceptor.com");
      int httpCode = https.GET();
      if (httpCode == 200)
      {
        String payload = https.getString();
        Serial.println(payload);
      }
      else
      {
        Serial.print("Error: ");
        Serial.println(httpCode);
      }
      https.end();

     }
     else
     {
      digitalWrite(LED, HIGH);
     }
  }
}
```

Listing 5: Een GET-request met REST en deep sleep-modus verzenden.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

#define LED D4
void setup()
{
    Serial.begin(115200);
    pinMode(LED, OUTPUT);
    digitalWrite(LED, HIGH);
    WiFi.begin(<SSID>, <Password>);
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(1000);
        Serial.print("Connecting ...");
    }
}

void loop()
{
  if (WiFi.status() == WL_CONNECTED)
  {
      digitalWrite(LED, LOW);
      BearSSL::WiFiClientSecure client;
      client.setInsecure();
      HTTPClient https;
      https.begin(client, "https://iotbutton.free.
                           beeceptor.com");
      int httpCode = https.GET();
      if (httpCode == 200)
      {
        String payload = https.getString();
        Serial.println(payload);
        digitalWrite(LED, HIGH);
        delay(2000);
        ESP.deepSleep(0);
      }
      else
      {
        Serial.print("Fehler: ");
        Serial.println(httpCode);
      }
      https.end();
  }
}
```



Figuur 8. Het schema bevat naast de microcontroller slechts één drukknop.



Figuur 9. Samenwerking tussen IoT-apparaat, IoT-cloud en andere diensten aan de hand van het ThingSpeak-voorbeeld.

RST-connector. Na de ingestelde tijd wordt een signaal verstuurd dat ESP automatisch herstart. De maximale tijdsduur is beperkt tot ongeveer 71 minuten. Deze optie is een goede keuze in scenario's die bepaalde meetwaarden cyclisch doorgeven. Het commando hiervoor is: `ESP.deepSleep(SLEEP_TIME * 1000000)`

- *Interrupt Deep Sleep*: de ESP8266 start niet automatisch opnieuw op, maar wordt uit de diepe slaapstand gewekt door een interrupt. Deze vorm van diepe slaapstand wordt gestart met: `ESP.deepSleep(0)`. In dit geval mogen GPIO16 en RST niet met elkaar worden verbonden. Als er een event plaatsvindt, sturen we een LOW-signaal naar de RST-pen. Daarmee wordt de ESP8266 opnieuw opgestart en wordt de software uitgevoerd.

Voor de IoT-button betekent dit dat de knop nu niet meer is aangesloten op een GPIO-pen, maar tussen RST en massa, zoals weergegeven in **figuur 8**. In de code kan daarom de controle op het indrukken worden weggelaten. In plaats daarvan wordt de microcontroller in de slaapstand gezet (**listing 5**) nadat de informatie met succes is verzonden via de REST-interface. Compileer de broncode (sketch) en laad deze in de controller. Na de start maakt hij verbinding met het netwerk, stuurt een bericht en gaat in de slaapstand. Als de microcontroller wordt gewekt via de (Reset)-knop, wordt het proces opnieuw gestart. U kunt online volgen hoe de REST-API met succes is aangeroepen met een GET-request.

Nogmaals in de cloud

Technisch gezien hebben we onze IoT-button nu geïmplementeerd. Welke acties er moeten worden uitgevoerd, als het bericht van de microcontroller wordt ontvangen, kunnen we in een

Weblinks

[1] Gebruikershandleiding ESP8266 NodeMCU: www.handsontec.com/pdf_learn/esp8266-V10.pdf

[2] CH340-driver: https://sparks.gogo.co.nz/ch340.html

[3] Arduino-IDE: http://www.arduino.cc/en/main/software

[4] Board Manager URL: http://arduino.esp8266.com/stable/package_esp8266com_index.json

[5] Beeceptor: https://beeceptor.com/

[6] Console IoT-Button: https://beeceptor.com/console/iotbutton

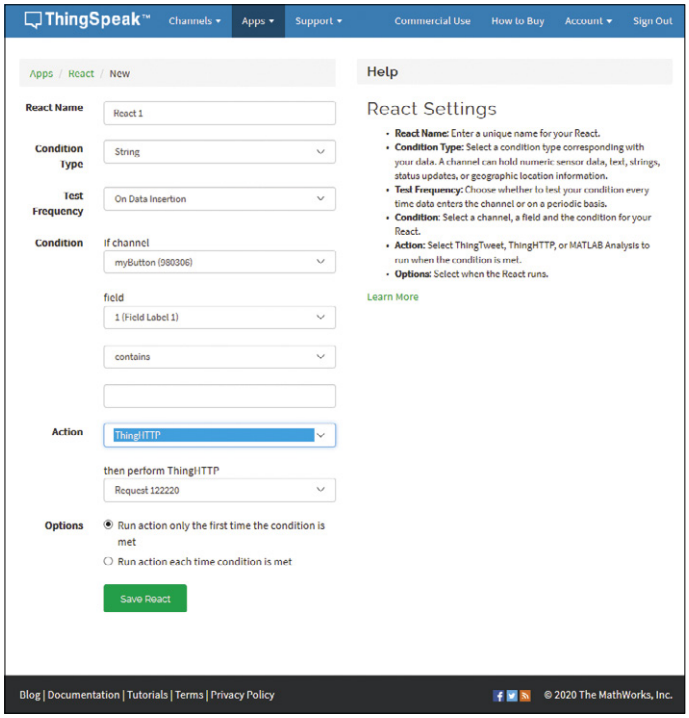
[7] Energieverbruik ESP8266: https://arduino-hannover.de/2018/07/25/die-tuecken-der-esp32-stromversorgung/

[8] Microsoft-IoT-Cloud: https://azure.microsoft.com/en-en/services/iot-hub/

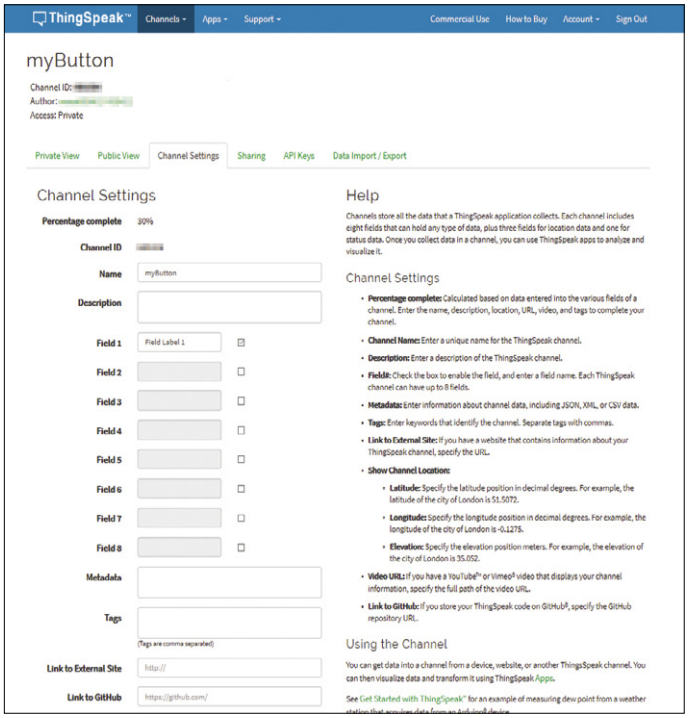
[9] ESP8266 in de Azure-Cloud: https://sandervandevelde.wordpress.com/2019/05/07/connection-a-cheap-esp8266-to-azure-iot-central/

[10] NodeMCU in de Azure-Cloud: https://www.thingforward.io/techblog/2018-05-22-connecting-nodemcu-to-microsoft-azure-iot-hub-part-1.html

[11] Thingspeak: https://thingspeak.com/



Figuur 10. Configuratie van het REST-eindpunt (channel) bij ThingSpeak.



Figuur 11. Apps bepalen de acties van ThingSpeak voor een kanaal.

paar stappen configureren. Ook voor deze scenario's zijn er verschillende clouddiensten, zoals *Microsoft Azure IoT* [8][9][10] of *ThingSpeak* [11]. De procedure bestaat in wezen uit het direct online configureren van een eindpunt in de clouddienst en het adresseren ervan via het internet vanaf het IoT-apparaat. De gewenste acties kunnen dan via de service worden geïnitieerd (**figuur 9**).

Maak een gratis account aan bij de *ThingSpeak*-service, waarvoor alleen verificatie via e-mail nodig is, en maak vervolgens een nieuw REST-eindpunt (*channel*) aan.
Vul het uitgebreide (en aan de rechterkant uitgelegde) formulier in (**figuur 10**) of verzin eerst een naam. Elk kanaal krijgt een unieke URL en de mogelijkheid om met behulp van URL-parameters extra gegevens te versturen buiten het request om. Voor een eenvoudige IoT-button zijn geen verdere parameters nodig, maar onder *My Channels/API Keys* kunt u de lees- en schrijfsleutels (en voorbeelden van het opnemen van de sleutels in de GET-requests) van het kanaal vinden. Als een kanaal correct is geconfigureerd, kunt u via het menupunt *Apps* bepalen welke acties moeten worden geïnitieerd (**figuur 11**), bijvoorbeeld het verzenden van een e-mail via een webservice of het plaatsen van een tweet op Twitter. Het IoT-backend dient als het ware als tussenlaag.

Let op: IoT-apparaten die met diensten in het netwerk communiceren, hebben toegang tot die diensten nodig. Hiervoor moeten ze zich natuurlijk wel identificeren, bijvoorbeeld met een gebruikersnaam en wachtwoord of unieke kenmerken (*keys*). Die waarden moeten worden opgeslagen in de broncode. Zorg ervoor dat u deze informatie beschermt, bijvoorbeeld als u de broncode kopieert of publiceert op het internet (voor gratis gebruik...).

Conclusie en vooruitblik

Experimenten met de ESP8266 microcontroller zijn fascinerend. Er zijn geen of nauwelijks extra componenten nodig om IoT-scenario's te implementeren. Er is niet veel werk meer nodig om onze IoT-button 'productierijp' te maken, of zelfs om er een commerciële toepassing van te maken. Meestal is het alleen een kwestie van mechanische randvoorwaarden (behuizing) en voeding (de accu permanent aansluiten). Er zijn ook uitbreidingen denkbaar: als de knop bijvoorbeeld door een bewegingsmelder wordt geactiveerd, kunt u direct 'alarm slaan' en via het internet communiceren als iemand zonder toestemming een beveiligd object nadert. Het eigenlijke werk wordt online gedaan via de configuratie in de cloud. ⏮

190303-B-03