IoT-Kommunikation mit dem Adafruit-IO-API

Dinge im Gespräch

Über den Adafruit-IO-Webservice kommunizieren IoT-Sensoren und -Clients mit Hilfe eines REST-API über das Internet. Der Artikel zeigt, wie die nicht-menschlichen Akteure in der Praxis zusammenarbeiten und wie die Daten per Skript auf den Linux-Desktop gelangen. *Marcus Nasarek*

Biologen würden wohl von einer Symbiose sprechen, um die Beziehung technikaffiner Menschen mit elektronischen Sensoren und Mikrocontrollern zu beschreiben. Ihnen hilft heute ein ganzer Zoo elektronischer Systeme durch den Alltag, steuert ihre Heizung, dimmt das Licht, kocht Kaffee oder sorgt dank Fehlfunktionen oder Sicherheitsmängeln für unterhaltsame Stunden [1]. Wer aber in die Tiefen der Elektronik und Mikrocontroller eintauchen möchte, braucht Enthusiasmus und elektronisches Grundwissen. Am Geldbeutel scheitert es heute hingegen nicht mehr unbedingt.

Tritt auf: Arduino

Die Entwicklung der Arduino-Plattform [2] schlug vor etwas mehr als zehn Jahren für ein größeres Publikum endlich eine auch finanzielle Brücke in das Paralleluniversum elektronischer Sensoren. Das kompakte und günstige Experimentier-Board lässt sich in C++ programmieren. Zeitgleich gründete die amerikanische Ingenieurin Limor Fried, auch

bekannt als "Ladyada", das Unternehmen Adafruit Industries [3]. Ihr Ziel: Den Umgang mit elektronischen Komponenten zu vereinfachen.

Adafruits kompakte Platinen fassen elektronische Baugruppen zu handlichen Breakout-Boards zusammen und ersparen dem Nutzer die sonst notwendigen Lötarbeiten. Selbst für komplexe Schaltungen und das Auslesen von Sensoren auf Basis von Industrieprotokollen wie I2C oder SPI braucht es nicht mehr als ein Programm in C++ und die passenden Bibliotheken.

Eilt herbei: Internet

Ein weiterer Trend geht inzwischen dahin, immer mehr Sensoren und Steuerkomponenten über das Internet miteinander zu verbinden. Ein Vorteil: Einfache Webservices stellen die Messdaten bereit, ein Browser stellt sie dar. Derselbe Browser oder ein Skript senden zugleich auch Schaltsignale an Komponenten oder übermitteln Konfigurationen.

Daneben gibt es heute für die meisten Elektronikplattformen auch WLAN-Erweiterungen. Arduino-Boards lassen sich seit einiger Zeit mit Ethernet- und Wifi-Shields bestücken. Geräte wie der Arduino Yún [4] verfügen über integrierte Netzwerkkomponenten. Das Spektrum ist groß und reicht von besonders kompakten Systemen wie dem ESP8266 mit Node-MCU [5] bis hin zum Raspberry Pi Zero W mit einem Linux [6].

Allerdings braucht es einigen Aufwand, um auf Basis dieser Komponenten und mit geeigneten Softwarebibliotheken eigene Internetanwendungen zu stricken. Genau hier springt Adafruit in die Bresche und bietet mit Adafruit IO [7] ein sehr komfortables API für netzwerkfähige Sensoren und Mikrocontroller an.

Kommt hinzu: Adafruit IO

Adafruit IO ist eine Programmierschnittstelle, die in zwei Geschmacksrichtungen zur Verfügung steht: einem REST- und einem MQTT-API (Abbildung 1). Das REST-API folgt den bekannten Prinzipien des REST-Architektur-Stils [8], während das MQTT-Protokoll direkt aus dem Industriebereich kommt und sich speziell zum Steuern von Sensoren über Netzwerke eignet (siehe eigenen Artikel).

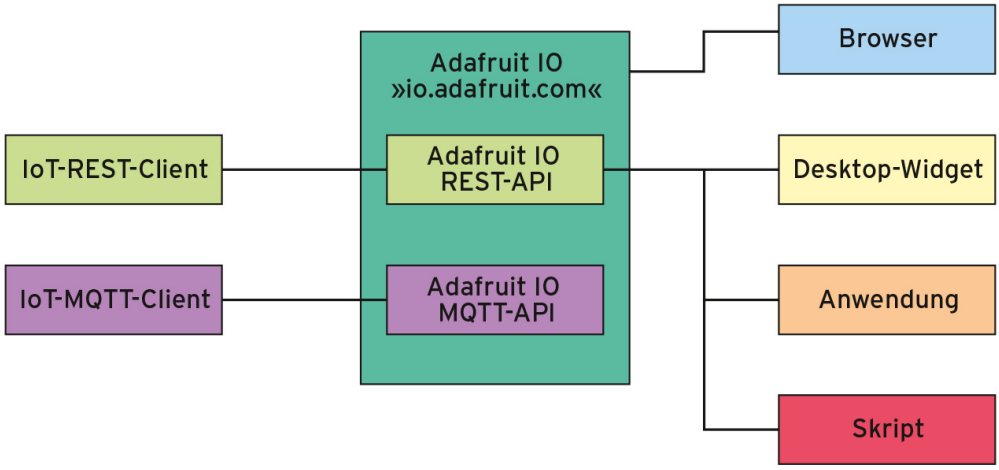

Abbildung 1: IoT-Clients und Desktop-Apps kommunizieren via REST oder MQTT mit dem Adafruit-IO-API.

Früchte ernten

Grundsätzlich gibt es zwei Wege, das Adafruit-IO-API zu nutzen. Einer führt über die Registrierung auf der Adafruit-IO-Webseite [7] und erlaubt anschließend den Zugriff auf das sehr gut dokumentierte API [9]. Ein zweiter besteht darin, den Adafruit-IO-Node.js-Server [10] von Github zu holen und ihn lokal zu installieren.

Angemeldeten Benutzern zeigt die Adafruit-IO-Webseite im Benutzerkonto zunächst eine Übersicht der vorhandenen Dashboards. Links erscheint ein Menü zur Auswahl weiterer Ansichten. Hinter dem Eintrag »Profil« verbirgt sich eine Liste der letzten Aktivitäten.

Zu den API-relevanten Übersichten gelangt der Benutzer über die Auswahl der Einträge »Feeds«, »Groups«, »Dashboards«, »Triggers« und »Settings«. Zudem stößt er dort auf Links zur »API Documentation« und zu weiteren Informationen (»Guide and Tips«, »Adafruit IO Forum« und »Blog/Changelog«).

Das Beispiel eines Temperatursensors soll zeigen, wie ein WLAN-fähiger Mikrocontroller die Sensorwerte via Internet an das Adafruit-IO-API schickt. Als IoT-Gerät dient dafür das Adafruit-Huzzah-Board auf Basis des ESP8266 [11].

Der IoT-Client

Als Linux-Client dient ein Python-Skript, das die Daten aus dem API ausliest und mit Hilfe des Systemmonitors Conky [12] auf dem Desktop darstellt. Die Schaltung besteht aus wenigen Bauteilen (siehe Tabelle 1), den Aufbau skizziert grob die Abbildung 2.

Tabelle 1: Bauteile für den IoT-Temperatursensor

Bauteil	Preis
Adafruit Huzzah	13 Euro
FTDI-Kabel	6 Euro
DHT22	6 Euro
10-kOhm-Widerstand	0,10 Euro
220-Ohm-Widerstand	0,10 Euro
LED	0,10 Euro
Steckbrett	3 Euro
Kabelbrücken	3 Euro

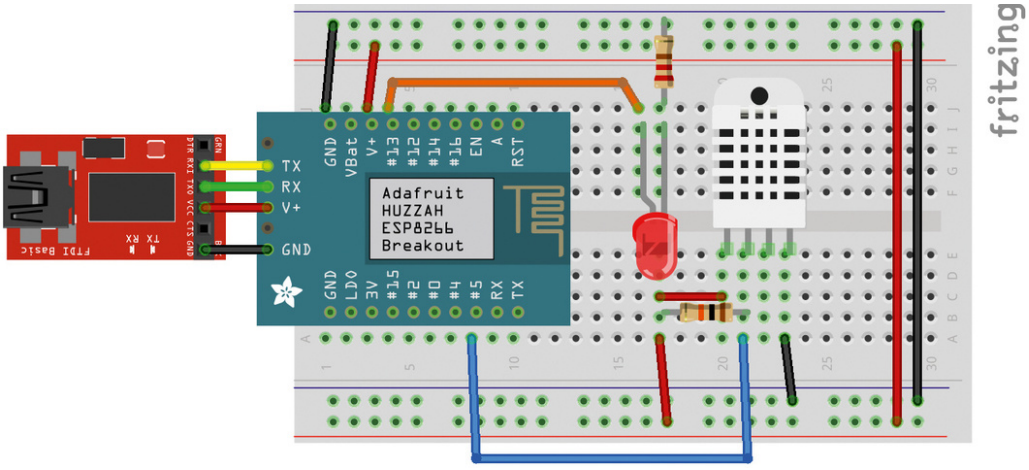

Abbildung 2: So sieht der Aufbau des IoT-Temperatursensors etwa aus, er besteht aus ein paar Standardkomponenten aus dem Elektronikmarkt.

Ist die Hardware einsatzbereit, erzeugt der Nutzer einen Datenfeed im Adafruit-IO-Benutzerkonto. Feeds nehmen Daten eines Senders entgegen und stellen sie für den Abruf durch einen Client bereit. Ein Klick auf »Actions | Create a New Feed« generiert einen neuen Feed in der Feed-Übersicht. Der IoT-Anwender wählt einen geeigneten Namen sowie eine Beschreibung für ihn aus, dann steht der Feed auf Empfang.

Der Nutzer programmiert den IoT-Temperatursensor am einfachsten mit der Arduino-IDE [13], die es sowohl für 32- als auch für 64-Bit-Linux-Systeme gibt. Die Adafruit-Huzzah-Boards warten mit integrierter USB-Schnittstelle und seriellen Anschlüssen auf. Ist lediglich ein serieller UART-Port vorhanden, erfordert dies den Einsatz eines FTDI-Kabels, um eine serielle Schnittstelle über USB zur Verfügung zu stellen.

Hat der User mit Hilfe der Anleitung unter [14] das entsprechende Adafruit-Huzzah-Board in die Arduino-IDE integriert, installiert er noch über »Sketch | Bibliotheken einbinden | Bibliotheken verwalten« die fünf Bibliotheken »DHT«, »ArduinoHttpClient«, »Adafruit Unified Sensor«, »Adafruit MQTT Library« sowie »Adafruit IO Arduino«.

Listing 1: *AdaIOClient.ino*

```
01 #include <Adafruit_Sensor.h>
02 #include <DHT.h>
03 #include "config.h"
04 #define DHTPIN 5
05 #define DHTTYPE DHT22
06 #define LED 13
07 AdafruitIO_Feed *temperatur = io.feed("Temperatur");
08 AdafruitIO_Feed *luftfeuchte = io.feed("Luftfeuchte");
09 AdafruitIO_Feed *led = io.feed("LED");
10 DHT_Unified dht(DHTPIN, DHTTYPE);
11 uint32_t delayMS;
12 void setup() {
13   io.connect();
14   led->onMessage(handleMsg);
15   while(io.status() < AIO_CONNECTED) {
16     delay(500);
17   }
18   dht.begin();
19   sensor_t sensor;
20   dht.temperature().getSensor(&sensor);
21   dht.humidity().getSensor(&sensor);
22   delayMS = sensor.min_delay / 1000;
23   pinMode(LED, OUTPUT);
24 }
25 void loop() {
26   delay(delayMS);
27   io.run();
28   sensors_event_t event;
29   dht.temperature().getEvent(&event);
30   if (!isnan(event.temperature))
31     temperatur->save(event.temperature);
32   dht.humidity().getEvent(&event);
33   if (!isnan(event.relative_humidity))
```

```
34     luftfeuchte->save(event.relative_humidity);
35   delay(10000);
36 }
37 void handleMsg(AdafruitIO_Data *data) {
38   if(data->value() == "ON")
39     digitalWrite(LED, HIGH);
40   else
41     digitalWrite(LED, LOW);
42 }
```

Das Programm basiert auf dem Beispielcode aus der Adafruit-IO-Bibliothek und besteht aus zwei Teilen. Listing 1 zeigt den ersten Teil mit dem Programmcode. Die Datei »config.h« in Listing 2 enthält den zweiten Teil mit den Zugangsdaten zum WLAN und zum Adafruit-IO-API. Die Einstellungen in der Datei passt der User entsprechend an. Anschließend gilt es, den Code und die Konfiguration zu kompilieren und dann auf das Board zu verschieben.

Listing 2: *config.h*

```
01 #define IO_USERNAME    "Benutzername"
02 #define IO_KEY         "Schlüssel"
03 #define WIFI_SSID       "SSID"
04 #define WIFI_PASS       "Passwort"
05 #include "AdafruitIO_WiFi.h"
06 AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);
```

In Sichtweite

Die Feed-Ansicht verrät, ob der Sensor erfolgreich Daten an das API liefert. Klappt das, richtet der User im nächsten Schritt den Client auf dem Linux-Desktop ein. Dafür dienen im hier vorgestellten Beispiel die Python-Skripte aus den Listings 3 und 4. Der Aufruf

Listing 3: *getTemp.py*

```
01 from Adafruit_IO import Client
02 aio = Client('AIO-API-KEY')
03 temp = aio.receive('Temperatur')
04 print("{0:.1f}".format(float(temp.value)))
```

```
sudo pip install adafruit-io
```

installiert die Python-Bibliotheken für das Adafruit-IO-API mit Hilfe der Paketverwaltung Pip. Die Skripte benötigen jeweils in Zeile 2 den API-Schlüssel, der im Benutzerkonto unter »Settings | Manage AIO Keys | View AIO Key« wartet. Einmal ausgeführt holt das Skript die Daten aus dem Feed des Adafruit-IO-API und gibt den aktuellen Wert aus.

Listing 4: *getHum.py*

```
01 from Adafruit_IO import Client
02 aio = Client('AIO-API-KEY')
03 hum = aio.receive('Humidity')
04 print("{0}".format(hum.value))
```

Es gibt mehrere Möglichkeiten, die Ausgabe des Skripts auf den Desktop zu bringen. Im Gnome-Panel zeigt zum Beispiel das Plugin Argos [15] die Werte oder entsprechende Symbole auf dem Panel an. Eine andere Möglichkeit bieten KDEs Plasma-Widgets, die der User bei Bedarf auf dem Desktop platziert. Ein Widget zu erstellen, das die Feed-Daten anzeigt, erfordert allerdings etwas Einarbeitung in das Thema [16].

Besonders einfach klappt die Anzeige der Werte mit Hilfe von Conky [12] und der Funktion »execi«, die Skriptausgaben anzeigt, die Conky zudem regelmäßig aktualisiert. Ein solcher Konfigurationseintrag für die beiden Skripte aus den Listings 2 und 3 mitsamt einer Aktualisierung alle 600 Sekunden sieht wie folgt aus:

```
Temperatur: ${execi 600 python getTemp.py}
Luftfeuchte: ${execi 600 python getHum.py}
```

Für Abbildung 3 zeigt dabei die Conky-Konfiguration »Simple Conky« [17]. Die Konfigurationsdatei ».conkyrc« kommt in das Verzeichnis »~/.conky/«. Damit das Anzeigeprogramm beim Systemstart automatisch den Dienst antritt, bindet es der Admin in die jeweilige Autostart-Konfiguration des Desktops ein. Optional kopiert er die passende Schriftart von [18] in das System-Font-Verzeichnis und installiert sie mit »fc-cache -v«.

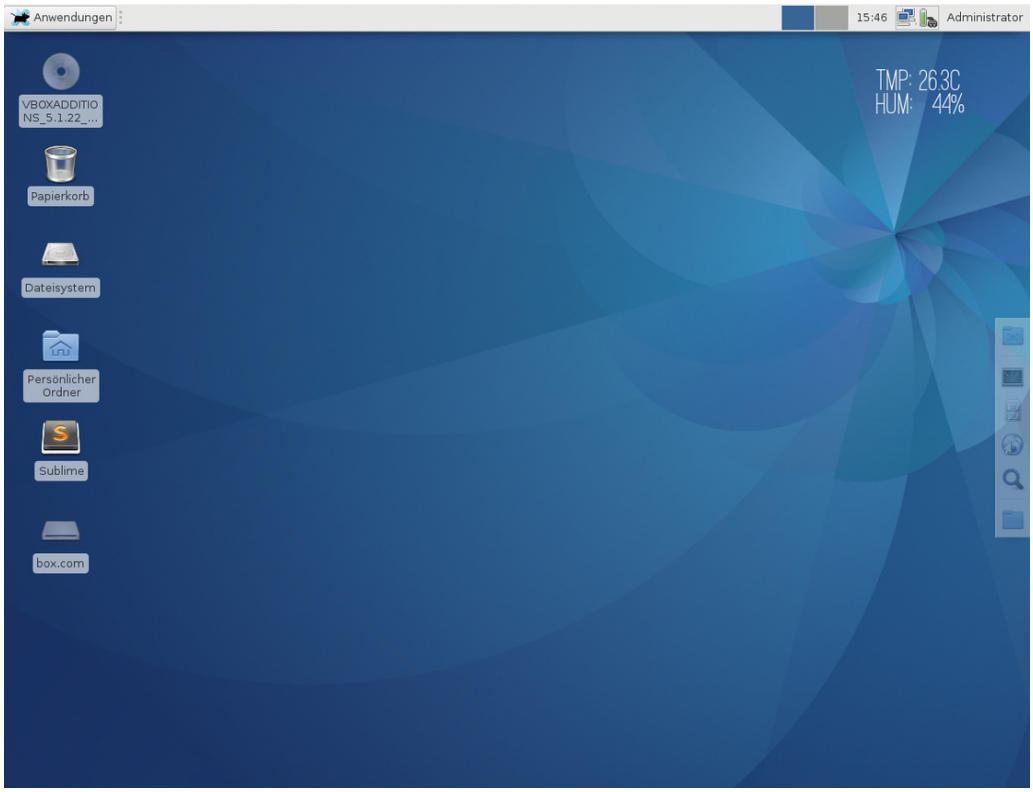

Abbildung 3: Die Feed-Daten erscheinen im Conky-Widget, hier in der Konfiguration »Simple Conky«.

Schalten übers Dashboard

Der Programmcode für das Huzzah-Board in Listing 1 und die Schaltung in Abbildung 1 enthalten bereits eine passende Funktion, um Nachrichten vom Dashboard zu empfangen. Das macht es dem Entwickler einfach, Aktionen auf dem IoT-Gerät

auszulösen. Die LED an GPIO #13 steuert ein Event-Handler, der Nachrichten aus einem Feed empfängt. Das Beispiel zeigt einen auf den Namen »LED« getauften Feed.

Das Dashboard in Abbildung 4 erstellt der Benutzer mit »Dashboard | Actions | Create a New Dashboard«. Anschließend fügt er diesem Dashboard mit einem Klick auf das blaue Plus-Symbol einen neuen Block hinzu. Hat er den Toggle-Button ausgewählt und ihn mit dem LED-Feed verknüpft, erzeugt er den Block über »Create Block«. Das Dashboard enthält neben dem Schalter für die LED zusätzlich ein Liniendiagramm, um die Temperatur und die Feuchtigkeitswerte darzustellen.

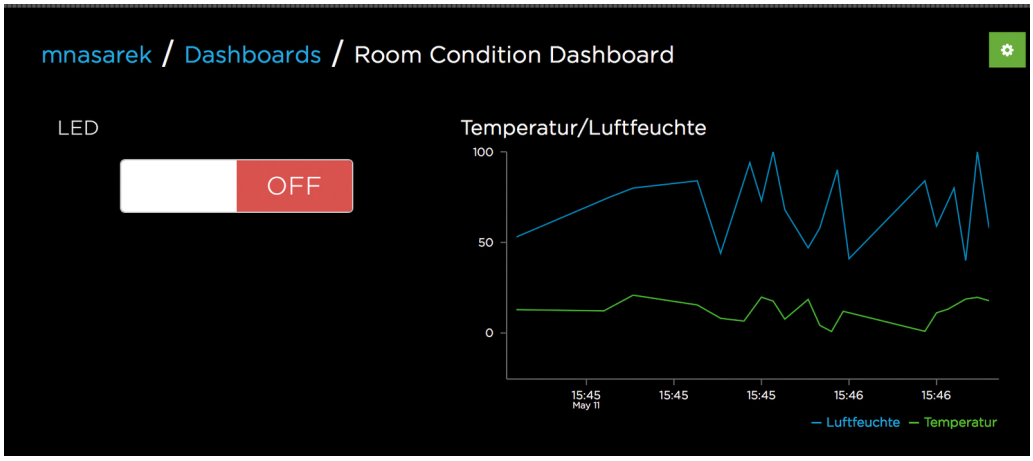

Abbildung 4: Das Dashboard zeigt nicht nur Feed-Daten an, es steuert auch IoT-Geräte, konkret die LED.

Klickt der Admin schließlich auf den Toggle-Button im Dashboard, bemerkt das Sensorboard dies und schaltet die LED ein oder aus. Dafür leitet die Funktion »led->onMessage();« in Listing 1 eingehende Benachrichtigungen an die Funktion »handleMsg()« weiter. Letztere wertet den Inhalt der Nachricht aus und schaltet die LED entsprechend.

Fazit

Open-Hardware-Plattformen wie Arduino und Adafruit senken die Hürden für kreative IoT-Projekte erheblich. Das Adafruit-IO-API macht es inzwischen besonders einfach, netzwerkfähige Sensoren mit Webdashboards zu verschränken und über das Internet zu steuern. Um sie in Anwendungen auf dem Linux-Desktop einzubinden, bedarf es nur einiger weniger Skriptzeilen.

Dabei zeigen die Dashboards nicht nur Messdaten an, sondern eignen sich auch zum Steuern der Hardware. Bestimmte Systemwerte wie CPU- und Speicherauslastung oder die Ausgaben von Anwendungen können gezielt Events auf der IoT-Hardware triggern. In der Folge erscheinen Texte auf Displays oder beeinflussen vordefinierte Events weitere vernetzte Hardware, etwa Hue-Lampen oder Wifi-Steckdosen. (kki)

Infos

1. Smarthome mit Hindernissen: [https://www.forbes.com/sites/kashmirhill/2013/07/26/smart-homes-hack/]
2. Arduino-Webseite: [https://www.arduino.cc]

3. Adafruit Industries: [https://www.adafruit.com]
4. Arduino Yún: [https://www.arduino.cc/en/Main/ArduinoBoardYun]
5. Node-MCU: [http://nodemcu.com/index_en.html]
6. Rasperry Pi Zero W: [https://www.raspberrypi.org/products/pi-zero-w/]
7. Webseite zum Adafruit-IO-API: [http://io.adafruit.com]
8. Marcus Nasarek, "Spätzünder": Linux-Magazin 06/17, S. 66
9. Dokumentation zum Adafruit-IO-REST-API: [https://io.adafruit.com/api/docs/]
10. Adafruit-IO-Node.js-Server: [https://github.com/adafruit/adafruit-io-node]
11. Adafruit Huzzah: [https://www.adafruit.com/product/2471]
12. Desktopwidget Conky: [https://github.com/brndnmtthws/conky]
13. Entwicklungsumgebung für den Arduino: [https://www.arduino.cc/en/Main/Software]
14. Adafruit Huzzah in die Arduino IDE einbinden: [https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/using-arduino-ide]
15. Argos-Plugin: [https://github.com/p-e-w/argos]
16. KDE Plasma Widget Development: [https://techbase.kde.org/Development/Tutorials/Plasma5/QML2/GettingStarted]
17. Simple Conky: [http://custom-linux.deviantart.com/art/Simple-Conky-454759768]
18. Ostrich-Sans-Font: [https://github.com/theleagueof/ostrich-sans]

Der Autor

Marcus Nasarek ist Linux seit Langem treu und schwer begeistert von Scripting, von Ruby und von Projekten mit dem Raspberry Pi.