

Bild 1: Wir bauen eine komplette Testumgebung auf – von der Erfassung der Messwerte bis zu ihrer grafischen Darstellung.

oder eine Produktionsanlage im betrieblichen Umfeld einbinden möchten.

### Kompakte Bausteine

Den Anfang machen die sieben Sinne des Systems: Damit es mit der realen Welt interagieren kann, braucht es Sensor-Aktor-Knoten, die sehen, hören, fühlen und Aktionen ausführen können. Konzentrieren wir uns im einfachsten Fall auf das Fühlen, können wir aus einer Vielzahl von Sensoren wählen, die etwa Raumtemperatur oder Luftfeuchtigkeit messen. Damit diese Sensoren Werte erfassen und weiterleiten können, benötigen sie ein Mindestmaß an Intelligenz. Diese Intelligenz liefert ein Mikrocontroller. Zusammen mit der notwendigen Übertragungstechnik bilden die Sensoren und der Mikrocontroller einen Sensorknoten.

Großer Beliebtheit erfreut sich der Raspberry Pi. Wenngleich sehr klein, handelt es sich bei dem Winzling mit seiner ARM-Architektur doch um einen ausgewachsenen Computer mit einem multitaskingfähigen Betriebssystem, in der Regel ein Linux-Derivat. Als Vorteil ergibt sich daraus, dass der Raspi sehr flexibel in der Programmierung ist. In der Regel ist er jedoch zu groß und zu komplex in der Handhabung, um ihn massenhaft dezentral auszurollen. Insbesondere ist der Raspi nicht über einen längeren Zeitraum batteriebetrieben einsetzbar. In unserem An-

wendungsfall wird er uns folglich erst später wieder begegnen, da wir ihn zentral als Backend einsetzen wollen.

In der Peripherie setzen wir stattdessen auf kleinere, spezialisiertere Komponenten. Hier sei allen voran die Arduino-Familie erwähnt. Es handelt sich dabei um eine quelloffene Mikrocontroller-Plattform, die mittels einer passenden IDE in C++ programmiert werden kann. Sowohl Hard- als auch Software stehen unter einer Open-Source-Lizenz. Der Arduino ist daher in verschiedenen Ausprägungen sehr günstig verfügbar.

Ebenso kostengünstig und sparsam im Energiebedarf sind die Mikrocontroller-Familien espressif ESP8266 sowie ESP32, die wir für unser Beispiel verwenden wollen. Auch diese beiden Vertreter sind Freunden eines smarten Zuhauses keine Unbekannten, sie haben ihre Ursprünge in der funkbasierten Steuerung von Lampen. Der ESP8266 bringt ab Werk einen WLAN-Chipsatz mit, der Nachfolger ESP32 auch Ethernet und Bluetooth. Es handelt sich bei beiden jeweils um ein integriertes System-on-a-Chip (SOC), das beispielsweise über die Arduino-IDE direkt in C++ oder per MicroPython programmierbar ist. Letzteres gelingt etwa in der IDE VisualStudio Code mittels Plug-in, das mit dem Bootloader des Mikrocontrollers kommuniziert.

Der eigentliche Controller ist meist auf einem Entwicklungsboard verbaut, das direkt per USB anschließbar ist. Alternativ zu einer eigenen Software-Entwicklung kann der Zwerg mit einer vorgefertigten Firmware betankt und anschließend über das WLAN konfiguriert werden. Nachteilig erweist sich allerdings, dass eine Absicherung der drahtlosen Kommunikation über Verschlüsselung und die Verwendung von Zertifikaten aufgrund der geringen Ressourcen der SOCs nur eingeschränkt möglich sind. Generell empfiehlt es sich folglich, IoT-Experimente wie auch den produktiven Betrieb in einem separaten (W)LAN-Segment zu betreiben. Das bringt uns zum zweiten Punkt unserer Übersicht, nämlich der Frage, wie die Daten vom Sensorknoten zur zentralen Verarbeitung gelangen.

### Nah und fern kostenfrei Daten übertragen

Um IoT-Geräte in großer Stückzahl einzusetzen, greifen Unternehmen sehr häufig auf eine Datenübertragung per Funktechnologie und eine Energieversorgung der Sensor-/Aktorknoten aus Batterie, Akku oder Solarzellen. Dies ist erforderlich, da eine kabelgebundene Umsetzung wegen der damit verbundenen Aufwände in vielen Fällen das frühe Aus für ein solches Projekt bedeuten würde. Der Sammelbegriff der "Low Power Wide Area Networks" (LPWAN) fasst relevante Netzwerkprotokolle und Technologien zur Übertragung von Daten im IoT zusammen. Mit IEEE 802.11ah alias Wi-Fi HaLow hat sich in den letzten Jahren ein WLAN-Standard etabliert, der mit niedrigem Energiebedarf besonders für Sensornetzwerke optimiert ist. Doch natürlich ist WLAN längst nicht der einzige Weg zur Kommunikation im IoT.

Im Nahbereich tummeln sich etwa mit Bluetooth LE, ZigBee oder auch Z-Wave weitere Übertragungstechniken. Allen gemeinsam ist, dass sie in den lizenzfreien ISM-Bändern (Industrial, Scientific and Medical), also kostenlos und ohne Genehmigung nutzbaren Frequenzbereichen, funken und mit sehr wenig Energie auskommen. Verteilte Sensoren sollen so batteriegestützt jahrelang wartungsfrei arbeiten können.



Bild 2: Header-Leisten (links) verbinden Mikrocontroller-Boards (Bildmitte unten) und verschiedene Shields (Bildmitte oben) zu kompletten Sensorknoten (rechts im Bild).

Weiterhin existieren Funktechnologien und Protokolle, die deutlich größere Entfernungen bis hin zu vielen Kilometern überwinden und so lizenzfrei eine breite Abdeckung in der Fläche realisieren können. Prominente Vertreter in diesem Bereich sind SIGFOX sowie LoRaWAN. Letzteres steht unseren Nachbarn in der Schweiz und den Niederlanden bereits nahezu flächendeckend zur Verfügung, während innerhalb Deutschlands noch Nachholbedarf besteht. Regionale LoRaWAN-Projekte werden hierzulande vor allem von Städten und Kommunen vorangetrieben, die trotz klammer Kassen Messwerte in ihrem geografisch verteilten Zuständigkeitsbereich erfassen möchten. Als Alternative zu eigenen regional begrenzten LoRaWAN-Netzen bietet sich die weltweit erfolgreiche Initiative "The Things Network" [2] an. In deren freien Communities darf jedermann mitspielen, die Übertragung und anschließende Verarbeitung der Daten ist jedoch nicht per SLA gesichert.

### Kostenpflichtige Funktechnik

Für zeitkritische Anwendungsfälle lohnt sich ein Blick auf die lizenzpflichtigen Funktechnologien. Denn dahinter stehen Provider, die sich vertraglich zur Erbringung einer definierten Leistung verpflichten und diese per SLA zusichern – dafür allerdings auch einen Preis verlangen. Waren herkömmliche Mobilfunkmodule lan-

ge zu energiehungrig und mangels passender Tarife schlicht zu teuer, findet sich auf dem Markt inzwischen praxistaugliche Technik mit auf die Anwendung im IoT zielenden Tarifen – ausgelegt auf eine sehr günstige Übertragung geringer Datenmengen von nur wenigen MByte pro Monat. Den entsprechenden Erweiterungen des Mobilfunkstandards LTE, namentlich LTE Cat M1 sowie NarrowBand-IoT (NB-IoT), hatten wir bereits einen Schwerpunkt gewidmet [3].

Mit der diesjährigen Auktion für die Lizenzblöcke des neuen Mobilfunkstandards 5G versuchen sich die Provider, ihr Stück vom Kuchen zu sichern, der auch das IoT noch mehr befeuern soll. 5G glänzt gegenüber LTE mit höheren Übertragungsraten bei minimaler Latenz und empfiehlt sich so auch zur Automatisierung zeitkritischer Produktionsprozesse. Ob die Mobilfunkprovider damit gegenüber den bereits etablierten, lizenzfreien Techniken Boden gut machen können, wird die Zukunft zeigen. Derweilen bleiben wir mit unserem Beispiel im Nahbereich und nutzen für einen schnellen Einstieg der Einfachheit halber WLAN nach 802.11 als Übertragungsmedium.

### Anwendungsprotokolle und Datenbroker

Wenden wir uns der logischen Datenübertragung zu und damit der Anwen-

dungsschicht des ISO/OSI-Modells. Auch hier werben gleich mehrere Protokolle um des Admins Gunst. Allen voran sei der Klassiker HTTP erwähnt, auf dessen Basis der "Representational State Transfer" (REST) mittels gängiger HTTP-Methoden wie GET und POST die Kommunikation von und mit IoT-Gerätschaften ermöglicht. Der Vorteil liegt in der weiten Verbreitung des Protokolls. Als nachteilig, insbesondere bei sehr schmalbandiger Anbindung, erweist sich der große Overhead des Protokolls. Dem Prinzip von Request und Response folgend, müssen sich die Kommunikationspartner zudem ihre Informationen per Pull holen und können sich nicht per Push unmittelbar informieren lassen. Abhilfe kann hier etwa der Einsatz von Websockets schaffen. Während HTTP auf TCP aufsetzt, basiert das "Constrained Application Protocol" (CoAP) auf UDP. Es kommt mit deutlich weniger Overhead aus, ist insbesondere zur Anbindung eingebetteter Systeme ans IoT ausgelegt und erlaubt ebenfalls REST.

Das "Extensible Messaging and Presence Protocol" (XMPP) wurde ursprünglich für Instant Messaging entwickelt und entsprechend unter seinem früheren Namen "Jabber" bekannt. Wie HTTP auch bringt XMPP einen großen Overhead mit sich und eignet sich somit nur bedingt für den Einsatz im IoT.

Besser geeignet für die Kommunikation von und mit Sensoren sowie Aktoren im IoT sind Protokolle wie das "Advanced Message Queuing Protocol" (AMQP) und "Message Queuing Telemetry Transport" (MQTT). Letzteres kommt mit minimalem Overhead aus und ist darauf optimiert, Geräten mit wenigen Ressourcen eine zuverlässige Datenübertragung auch über eher unzuverlässige Netze zu erlauben. MQTT folgt dem Publisher-Subscriber-Funktionsprinzip [4]: Clients können dabei Nachrichten mit einem bestimmten Betreff – Topic genannt – versenden, während andere Clients diese Topics abonnieren, so wie wir es von diversen sozialen Medien her kennen. Im Zentrum der Kommunikation steht ein MQTT-Broker, der die Daten entgegennimmt und sich um die Verteilung von Nachrichten im Push-Verfahren kümmert. Für

unsere ersten Gehversuche reicht ein öffentlicher Broker, wie ihn die Eclipse Foundation betreibt [5]. Wir wenden uns nun der Praxis zu, nehmen einen ersten Sensorknoten in Betrieb und sorgen dafür, dass er per MQTT kommuniziert.

## Aufbau des Sensorknotens

Um dies nachzuvollziehen, setzen wir als Sensorknoten ein "Wemos D1 mini" ein. Das ist ein Entwicklungsboard, das im Wesentlichen mit einem ESP8266-Mikrocontroller, einer USB-Schnittstelle vom Typ CH340, 4 MByte Speicher, einem Taktgeber für 80/160 MHz, einer WLAN-Antenne und einem RESET-Button bestückt ist. Das Board liefert der Fachhandel problemlos und günstig. Es kommt mit einer Auswahl unterschiedlicher Stift- und Buchsenleisten – sogenannten "Headern".

Erste Gehversuche, wie mit den nachfolgend beschriebenen Systemwerten, können ohne diese Header erfolgen. Zur späteren Kommunikation mit Sensoren und Aktoren steht jedoch eine Vielzahl sogenannter "Shields" – Erweiterungsplatinen, die mit zusätzlicher Elektronik bestückt sind – zum Aufstecken zur Verfügung (Bild 2). Spätestens dann wird es notwendig, die passenden Header einzulöten. Dies erfordert neben einer geeigneten Lötstation etwas Erfahrung, stellt aber keine allzu große Hürde dar.

Wir setzen in unserem Beispiel ein Shield mit dem Temperatur- und Luftfeuchtesensor DHT22 der Firma Aosong ein. Doch Achtung: Manche dieser Shields kommen mit fast baugleichen Sensoren des Herstellers Asair daher, was in unseren Tests zu erheblichen Problemen führte. Das Aufstecken des Sensor-Shields auf das Prozessorboard führt zu einer kompakten Lösung, die im Dauerbetrieb jedoch noch einen Nachteil mit sich bringt: Der Prozessor erwärmt sich im Laufe der Zeit – und damit möglicherweise auch den Sensor, der sich direkt darüber befindet. Dies führt zu verfälschten Messwerten. Abhilfe kann hier neben einer anderen Anordnung der sogenannte Deep-Sleep-Modus schaffen. Dabei wird der Prozessor nach erfolgter Messung der Sensordaten und deren Übertragung zum Backend in einen Schlafmodus versetzt,

**ESP Easy Mega: flummi42**

Main Config Controllers Hardware **Devices** Notifications Tools

**Task Settings**

Device: Environment - DHT11/12/22 SONOFF2301/7021 ? i

Name:

Enabled:

**Sensor**

GPIO ⇄ Data:

Sensor model:

**Data Acquisition**

Send to Controller

Interval:  [sec]

**Values**

#	Name	Formula ?	Decimals
1	<input type="text" value="Temperature"/>	<input type="text"/>	<input type="text" value="1"/>
2	<input type="text" value="Humidity"/>	<input type="text"/>	<input type="text" value="1"/>

Powered by Let's Control It community

Bild 3: Mit Devices und Tasks definiert die Firmware ESPEasy, welche Daten der Mikrocontroller ESP8266 per MQTT verbreitet.

in dem er nur sehr wenig Energie verbraucht und somit keine nennenswerte Wärme abgibt. Sind beispielsweise fünf Minuten Tiefschlaf vergangen, erwacht der Prozessor selbständig und beginnt die Sequenz von vorne. Auf dem Prozessorboard ist dazu allerdings eine Verbindung zwischen RST und GPIO-Pin 16 zu schaffen. Neuere Boards haben dafür eine kleine Lötbrücke auf der Rückseite.

## Firmware flashen

Schließen Sie das Board per USB an einen Computer mit Windows 10 an, meldet das System "USB2.0-Serial eingerichtet und einsatzbereit". Im Geräte-Manager sollte sich die Schnittstelle nun bei den Anschlüssen als COM-Port eingetragen haben. Falls nicht, hilft der passende Treiber direkt vom Board-Hersteller [6]. Notieren Sie sich die Nummer des COM-Ports.

Den Mikrocontroller über die Arduino IDE selbst in C++ zu programmieren, wäre allein Stoff für mehrere Artikel. Wir verwenden daher mit "ESPEasy mega" [7] eine fertige Firmware, die wir nur noch konfigurieren müssen. Laden Sie dazu die tagesaktuelle Version, in unserem Beispiel das Archiv "ESPEasy\_mega-

20190409.zip", herunter und entpacken Sie diese. Der Unterordner "bin" enthält fertig kompilierte Versionen der Firmware für diverse Plattformen. Kopieren Sie die Dateien "ESP\_Easy\_mega-20190409\_normal\_ESP8266\_4M" und "blank\_4MB.bin" eine Ebene höher zu den ausführbaren Dateien. Mit dem Werkzeug "Flash-ESP8266.exe" wählen Sie in einer rudimentären grafischen Oberfläche den COM-Port, an dem sich der Mikrocontroller befindet, und die zu übertragende Firmware aus. Deutlich schneller geht es per Kommandozeile. Mittels

```
esptool.exe -vv -cd nodemcu -cb
921600 -cp COMx -ca 0x00000 -cf
blank_4MB.bin
```

löschen Sie den Controller komplett. Ersetzen Sie dabei "COMx" durch den oben notierten COM-Port. Anschließend schreibt

```
esptool.exe -vv -cd nodemcu -cb
921600 -cp COMx -ca 0x00000 -cf
ESP_Easy_mega-20190409_normal_
ESP8266_4M.bin
```

die Firmware in den Speicher. Trennen Sie nun das Board vom Strom, verbinden

Sie es wieder und betätigen Sie zur Sicherheit noch einmal dessen Reset-Knopf.

## Sensor mit dem WLAN verbinden

Begeben Sie sich dann in das vom Controller aufgespannte WLAN namens "ESP\_Easy\_0". Das Passwort für dieses WLAN lautet "configesp". Rufen Sie nun im Browser die IP-Adresse 192.168.4.1 auf. Sie gelangen daraufhin zur WLAN-Konfiguration des Controllers.

Wählen Sie aus der Liste Ihr WLAN und geben Sie dessen Passwort ein. Die SSID des WLAN darf dabei allerdings kein Hochkomma enthalten. Sobald Sie die Konfiguration speichern, verbindet sich der Controller mit Ihrem WLAN und teilt Ihnen als letztes Lebenszeichen im Browser mit, unter welcher IP-Adresse das Controller-Board künftig erreichbar ist. Anschließend schaltet er das WLAN "ESP\_Easy\_0" ab.

## MQTT-Topics konfigurieren

Verbinden Sie Ihren Computer mit demselben WLAN, in dem auch der Controller funkt, und klicken Sie im noch offenen Browser-Fenster auf den Button "Proceed to main config". Sie können den Controller nun auf mehreren übersichtlichen Seiten konfigurieren. Geben Sie dem Gerät auf dem Register "Config" im Feld "Unit Name" einen eindeutigen Namen – in un-

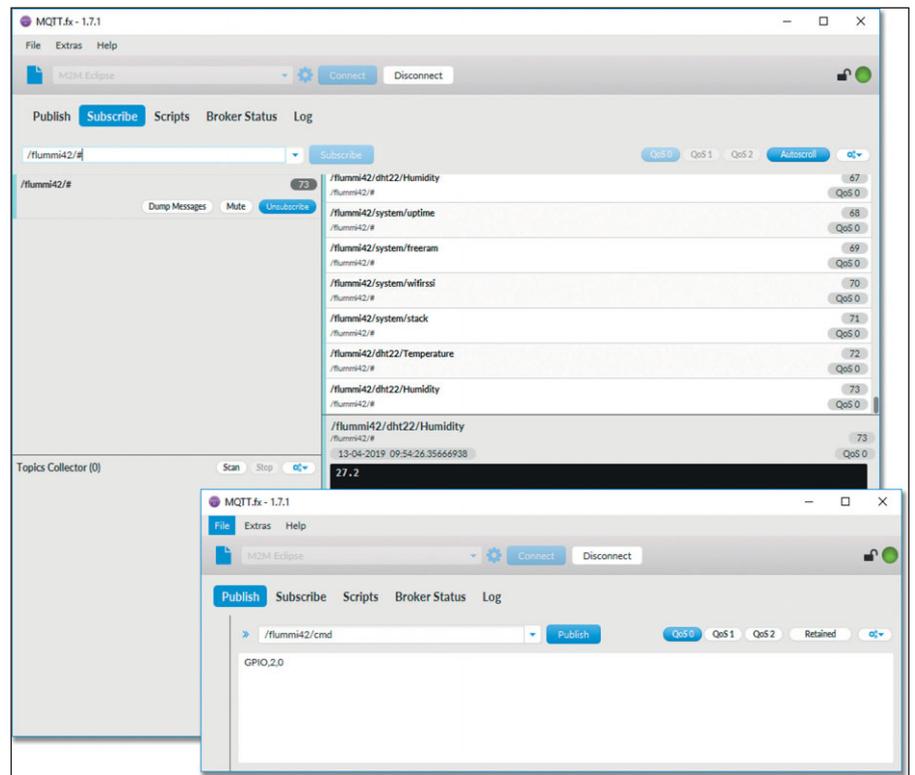


Bild 4: Der freie Client MQTT.fx kommuniziert über den MQTT-Broker bidirektional mit Sensoren und Aktoren.

serem Beispiel "flummi42" – und bestätigen Sie die Änderungen über die Schaltfläche "Submit" unten auf der Seite.

Weiter geht es auf der Seite "Controllers", deren Name vielleicht etwas irreführend ist. Sie konfigurieren dort nicht den Mikrocontroller selbst, sondern ein oder mehrere MQTT-Broker als Ziel für die Datenübertragung. Editieren Sie hier die erste Zeile und wählen Sie auf der folgenden Seite das Protokoll "OpenHAB MQTT" aus. Ändern Sie im Dropdown-Feld "Locate Controller" die Methode auf "Use Hostname" und tragen Sie im Feld darunter als Ziel "m2m.eclipse.org" ein. Weiter unten sehen Sie im Feld "Controller Publish" den Aufbau für Topics, unter denen das Gerät seine Daten per MQTT an den Broker sendet. Eine Referenz der möglichen Variablen finden Sie im Wiki der Firmware ESPEasy [8]. Standardmäßig publiziert der Sensor mit dem String "%sysname%/%tskname%/%valname%" unter seinem Namen (sysname) ein oder mehrere Tasks (tskname), die wiederum ein oder mehrere Werte (valname) umfassen. Aktivieren Sie die Checkbox "Enabled" und speichern Sie die Konfiguration des Brokers per "Submit".

Weiter geht es auf der Seite "Devices", wo Sie bis zu zwölf Messaufgaben (Tasks) anlegen können. Ein Device vom Typ "Generic - System Info" publiziert vier wählbare Werte zum Betrieb des Controllers. Geben Sie dem Task einen Namen und belegen Sie hier die vier Indikatoren etwa mit den Informationen zu "Uptime", "Free RAM", "Wifi RSSI" und "Free Stack". Weiter unten auf der Seite weisen Sie den Indikatoren vier Variablen mit beliebigem Namen zu. Wichtig ist noch, die beiden Checkboxes "Enabled:" und "Send to Controller" zu aktivieren. Das Intervall bestimmt, wie oft der Sensor Daten übermittelt. Hier eignen sich für den Anfang 10 bis 15 Sekunden, damit schnell Ergebnisse sichtbar werden. Speichern Sie das Ganze anschließend mittels "Submit".

Für das Sensor-Shield legen Sie einen zweiten Task vom Typ "Environment - DHT11/12/22..." an. Geben Sie dem Task wieder einen Namen und aktivieren Sie ihn mittels "Enabled". Nun ist festzulegen, wie der Prozessor mit dem Sensor kommunizieren soll und um welchen Sensortyp es sich genau handelt. Wählen Sie dazu mittels Drop-down-Box unter "GPIO-Da" die Option "GPIO-2(D4)" und als Sen-

### Link-Codes

- [1] IoT Standards and Protocols  
j7p31
- [2] LoRaWAN  
j7p32
- [3] Die Qual der Wahl  
IT-Administrator August 2017
- [4] MQTT Essentials  
j7p33
- [5] Eclipse IoT  
j7p34
- [6] Download "ch341ser\_win\_3.4.zip"  
j7p35
- [7] ESPEasy  
j7p36
- [8] ESPEasy System Variables  
j7p37
- [9] MQTT.fx  
j7p38

sor "DHT22". Aktivieren Sie wiederum "Send to Controller", tragen Sie ein Messintervall von zehn Sekunden ein und bestätigen Sie mittels "Submit" (Bild 3). Wechseln Sie nun wieder auf die Übersicht der "Devices", sollten Sie dort neben den Systeminformationen auch Temperatur und Luftfeuchtigkeit bereits sehen.

### Erfolgskontrolle

Um festzustellen, ob die gesamte Konfiguration wirklich erfolgreich war, nehmen Sie nun einen der zahlreich verfügbaren MQTT-Clients zur Hand. In unseren Tests hat sich der kostenfreie Client "MQTT.fx" [9] bewährt, der Linux, macOS und Windows unterstützt. Der Client bringt die passende Konfiguration für den von uns verwendeten Broker der Eclipse Foundation bereits mit. Starten Sie das Tool, wählen Sie aus der Drop-down-Box in der Kopfzeile die Verbindung "M2M Eclipse" und nehmen Sie mittels "Connect" Kontakt mit dem Broker auf. Dazu muss in Ihrer Firewall der TCP-Port 1883 in Richtung Internet offen sein.

In der Menüleiste darunter navigieren Sie nun zum Punkt "Subscribe", wo Sie im Dropdown-Feld den String "/flummi42/#" eingeben. Das #-Zeichen fungiert dabei als Wildcard für beliebige weitere Zeichen. Sobald Sie auf die Schaltfläche "Subscribe" klicken, erscheinen in der rechten Fensterhälfte, spätestens nach der konfigurierten Intervallzeit, sämtliche Topics, die das System mit dem Namen "flummi42" publiziert, darunter unsere zuvor konfigurierten Informationen zum Systemstatus sowie die Daten des Temperatur- und Luftfeuchtesensors (Bild 4).

Der Rückkanal zum Sensor funktioniert analog über MQTT-Nachrichten. Navigieren Sie zum Bereich "Publish" und geben Sie dort im Drop-down-Feld das Topic "/flummi42/cmd" ein, um dem Mikrocontroller Befehle zu schicken. Der Freitextbereich darunter nimmt die Payload des Kommandos auf. Tragen Sie dort "GPIO,2,0" ein und klicken Sie auf "Publish". Nun sollte auf dem Board des Mikrocontrollers die blaue LED aufleuchten. Mittels "GPIO,2,1" schalten Sie die LED wieder aus.

Auf demselben Weg könnten Sie ebenso Meldungen auf lokalen OLED-Displays anzeigen oder Relais schalten. Die passenden Shields dazu liefert der Fachhandel.

### Fazit

Mit diesem ersten Teil unseres Workshops haben wir einen kleinen Schritt ins IoT gewagt und einen Mikrocontroller dazu gebracht, per MQTT Daten zu verschicken sowie auf Kommandos zu reagieren. Dass dies im Zusammenspiel mit einem öffentlichen Broker nicht für einen produktiven Betrieb taugt, liegt auf der Hand. Sämtliche Kommunikation erfolgt unverschlüsselt, die Verfügbarkeit des Brokers ist nicht garantiert und alle anderen Nutzer des Brokers könnten unsere Topics mitlesen sowie selbst Befehle an unseren Controller schicken. Es fehlen auch noch die weiteren Komponenten aus unserer Übersicht. Daher werden wir im zweiten Teil des Workshops unsere eigene IoT-Plattform aufbauen und Daten nicht nur empfangen, sondern auch per Workflow-Engine weiterverarbeiten, in einer Datenbank speichern und schließlich visualisieren. (jp) 

# IT Administrator Intensiv-Seminar

## PowerShell für Admins



Quelle: goodluz - 123RF.com

**18. bis 20. November 2019  
Hamburg**

SYMPLASSON Informationstechnik GmbH  
Holstenstraße 205  
22765 Hamburg

Anmeldung unter:  
**www.it-administrator.de/  
trainings**

### Unser Angebot:

- Maximal 10 Teilnehmer je Seminar
- Einen Rechner für jeden Teilnehmer
- 3-tägiges Intensivseminar, das am ersten Tag um 10 Uhr beginnt und am 3. Tag gegen 16 Uhr endet.

### Seminarpreise:

**Sonderpreis für  
Abonnementen des IT-Administrator:**

**1.190 €**  
(inkl. 19% MwSt.)

**für Nicht-Abonnementen:**

**1.370 €** (inkl. 19% MwSt.)

### Ihr Dozent:



Franz-Georg Clodt

### Agenda:

#### • Einführung und Basiswissen

Architektur der PowerShell und Versionsunterschiede / Cmdlets, Cmdlet-Parameter und Hilfsfunktionen / Objekt-Pipeline und Ausgabefunktionen / Navigationsmodell (PowerShell-Provider)

#### • Scripting

PowerShell Language (PSL): Variablen und Kontrollstrukturen / Objektorientiertes Programmieren mit Klassen (ab PowerShell 5.0) / Sicherheitsfunktionen (Execution Policy) / Vordefinierte Variablen, Fehlerbehandlung und Fehlersuche / Scripting mit PowerShell Integrated Scripting Environment (ISE)

#### • Aufbauwissen

Fernaufruf/Fernadministration mit WS-Management (Remoting) / Serververwaltung Active Directory und Automatisierung von Azure