Richard Hubert
(hubert@hubert-associates.eu)
ist IT-Architekt mit zahlreichen Auszeichnungen, IEEE CSDP und IASA Fellow sowie Autor.

Internet of Things (IoT)
End-to-End mit Azure IoT-Services

Das Internet of Things (IoT) macht traditionelle Steuerungssysteme mit Sensoren und Aktoren immer intelligenter und ermöglicht eine Erfassung und Kommunikation von Daten mittels moderner Cloud-Services fast in Echtzeit. In diesem Beitrag werden wir das Thema „Telemetry-to-Cloud" schrittweise ausarbeiten. Zunächst werden Definitionen, Anwendungsfelder und Architekturschwerpunkte von „Telemetry-to-Cloud-Systemen" betrachtet. Die Aspekte werden anhand eines direkt aus der Praxis stammenden Systems veranschaulicht. Ziel des Systems ist die Bereitstellung einer funktionierenden, durchgängigen „Telemetry-to-Cloud-Lösung" [Hub].

Zur Realisierung des Gesamtsystems setzen wir auf den Microsoft Azure-Stack. Mit dem hohen Innovationsgrad und der Offenheit dieses Stacks auf vielen Plattformen sowie mit den leicht integrierbaren Standard-Services der Microsoft Azure-Clouds kann das System „high level" entwickelt und erweitert werden. Abbildung 1 zeigt das **Top-Level-Systembild** des Telemetriesystems.

IoT – Perspektiven

Das Hauptmerkmal des Internet of Things ist eine exponentiell steigende Anzahl von „Datengebern" und „Datenempfängern" sowie eine entsprechend wachsende Integrierbarkeit. Diese Integrierbarkeit ermöglicht neue Kooperations- und Optimierungsszenarien, bringt aber auch ganz neue Herausforderungen mit sich, zum Beispiel in den Bereichen Sicherheits- und Identitätsmanagement.

Architektonisch kann die IoT-Evolution im übertragenen Sinne gesehen werden wie Schritte zur Vereinigung vieler einzelner Instrumente und Musiker zu einem Orchester. Hierdurch werden faszinierende, neue Formen von Musik möglich, die auch neue Koordinierungsszenarien sowie ein übergeordnetes, ganzheitliches Denken von Komponisten und Dirigenten erforderlich machen.

Um beim Bild zu bleiben: In der IT-Welt müssen sich die Entwickler von traditionellen Produkten – ähnlich wie die „One Man Bands" in der Musik – überlegen, in welchem Maße sie sich für neue Formen des Zusammenspiels öffnen. Ein Beispiel

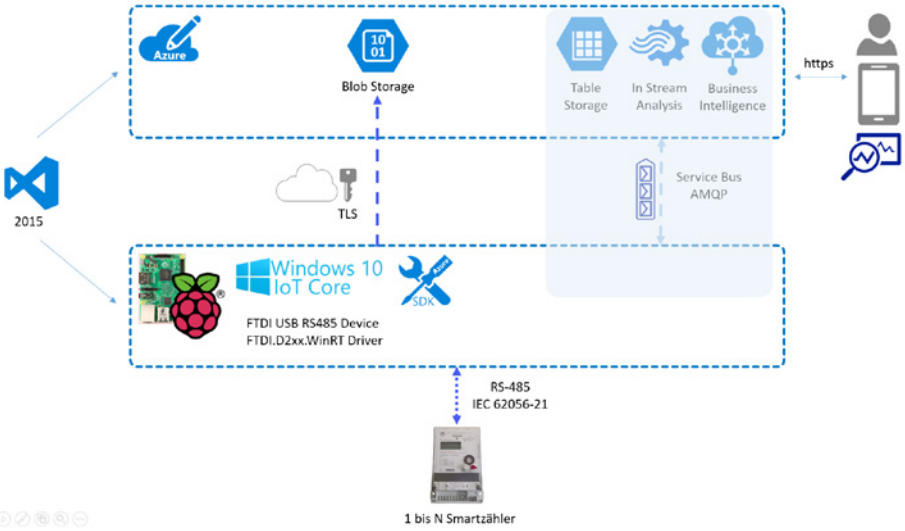

Abb. 1: Top-Level-Systembild

hierfür sind Hersteller von Heizungsanlagen.

Die Anlagen, die heute zu kaufen sind, bieten praktisch nur bei den Rohrmaßen genormte, offene Schnittstellen. Aus IT-Sicht ist die gesamte Anlage eine proprietäre, meist hermetisch geschlossene Insel des Herstellers. Aber der Druck zur offenen Beteiligung im Informationsnetz steigt. So, wie früher die Standardisierung auf die (englischen) Rohrmaße die meisten Hersteller zur Beteiligung bewogen hat, werden heute durch die Fortentwicklung des IoT-Markts zunehmend diejenigen Heizungsanlagenhersteller vom Markt belohnt, die ein Zusammenspiel im Sinne von Datengebern (Zustands- und Analysedaten) und Datenempfängern (Steuerung) ermöglichen.

Unser Praxisbeispiel ist ein Beispiel für genau diese Marktentwicklung: Wir verwenden eine neue Generation von Smart-Meter, da sie sich mit offenen und genormten Schnittstellen (RS-485/IEC-62056) einfach und sicher in die Welt des IoT einbinden lässt.

Das Thema Security (Datenintegrität und Datenschutz) steht immer im Vordergrund, vor allem bei Systemen mit Zugang zum Internet. Damit Sicherheit nicht dem Zufall überlassen bleibt, muss das Thema Security fest in der Rahmenarchitektur eines Cloud-zentrischen Systems verankert sein. Mit anderen Worten: IoT-Systeme sollen „Secure by Design“ sein. Spätestens dann, wenn ein System zahlreiche Nutzer hat oder Zugangsrechte und Nutzungsrechte beherrschen muss, wird auch ein Architekturrahmen notwendig, um Risiken nicht nur im Bereich Security kalkulierbar und beherrschbar zu machen.

Der Architekturrahmen sorgt unter anderem dafür, dass Aufwand, Risiken und Nutzen über den gesamten Lebenszyklus in ein optimales Verhältnis gebracht werden können. In diesem Zusammenhang zeigt unser Praxisbeispiel, wie die soliden Features eines ausgereiften Cloud-Service-Stacks wie Microsoft Azure konsequent wiederverwendet werden, um viele Eigenentwicklungen und die damit verbundenen Kosten und Risiken in „Standardbereichen“ zu vermeiden.

Dasselbe gilt für Themen wie „Compliance“ und „Auditability“ eines Cloud-zentrischen IoT-Systems. Wird im IoT-Architectural-Framework konsequent der Cloud-Stack eines zertifizierten Cloud-Providers verwendet, lassen sich diese Eigenschaften automatisch und ausgesprochen günstig auf das eigene System übertragen.

„Use it or lose it“

„Use it or lose it“ sagt man in den USA. Das gilt insbesondere für Technologiebereiche mit hoher Innovationsgeschwindigkeit wie Internet-Systemarchitektur, IoT und Cloud. Wenn ein IT-Designer nicht regelmäßig und „bis zu den Ellenbogen“ in der Entwicklungspraxis involviert ist, dann verliert er schnell seine „Schärfe“ („loses his edge“).

Praxisnahe IoT-/Cloud-Architekten sind aus Business-Sicht nicht länger ein „nice-to-have“, denn der Verlauf des digitalen Wandels in Unternehmen entscheidet mittelfristig über Erfolg oder Misserfolg. Das Beratungshaus McKinsey spricht in diesem Zusammenhang von der Notwendigkeit „digitaler Beiräte“ auf CEO-Ebene und stellt fest, dass der digitale Wandel bereits heute die Märkte und Wettbewerbssituation der meisten Firmen wesentlich prägt.

Firmen, die die Wichtigkeit der „digitalen Transformation“ verstehen und es als Chance begreifen, dass fast nichts mehr bleibt wie es war, werden als Gewinner aus diesem Prozess hervorgehen. McKinsey schätzt, dass aber nur drei von fünf Firmen diesen Wandel mittelfristig überleben.

Aus unserer Sicht besteht kein Zweifel, dass der digitale Wandel zu IoT und Cloud-IT stattfinden wird. Die Frage ist nur, wie diese Transformation durch das jeweilige Unternehmen gestaltet wird. Hier ist die zentrale Rolle einer Architektur-zentrischen Vorgehensweise wichtiger denn je.

Im Sinne von „use it or lose it” beginnen wir jetzt mit einem IoT End-to-End-System in der Praxis.

IoT – Praxis End-to-End

Hardware und Treiber

Neben einem gängigen Windows 10 PC für die Entwicklung werden folgende Hardwareprodukte eingesetzt:

- Als Hardware kommt ein Raspberry Pi 2B (1GB RAM) „vom Regal“ zum Einsatz.
- Als RS-485-Schnittstelle für den Raspberry Pi verwenden wir eine USB-RS-485-Bridge mit FTDI-Chip sowie Treibern [Info1].
- Der Smart-Meter ist ein Messwandlerzähler 5||1 der Firma EMH (EMH ITZ W1E4-00-STB-D3-020001-N50/Q) mit RS-485 (ANSI TIA-485-A) Schnittstelle – einer von vielen dieser Art, die wir im Labor zu Testzwecken nutzen.

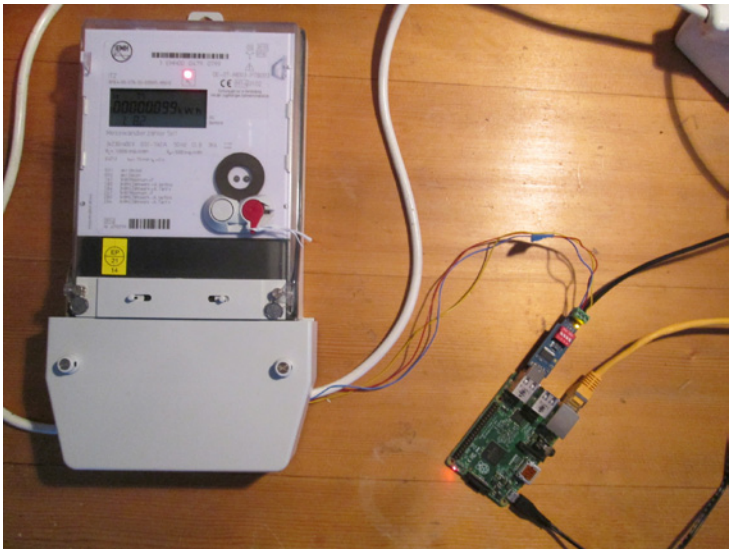

Abb. 2: Hardware-Setup im Labor

Warum RS-485 und IEC 62056-21 und wie?

RS-485/TIA-485 wird oft als nicht mehr zeitgemäß gesehen. Diesem Eindruck ist nachdrücklich zu widersprechen. TCP-IP ist auch nicht veraltet, obwohl es tatsächlich noch älter als RS-485 ist. Das genormte RS-485/TIA-485 ist nach wie vor erste Wahl für viele „last mile“-Kommunikationsverbindungen. Es ist störungsresistent und ausreichend schnell für viele Anwendungen im industriellen Umfeld, solange nicht große Datenmengen übertragen werden müssen. Es kann im Master-Slave-Mode sicher betrieben werden und kommt mit 2 Drähten für die Kommunikation aus. Es ist flexibel über eine

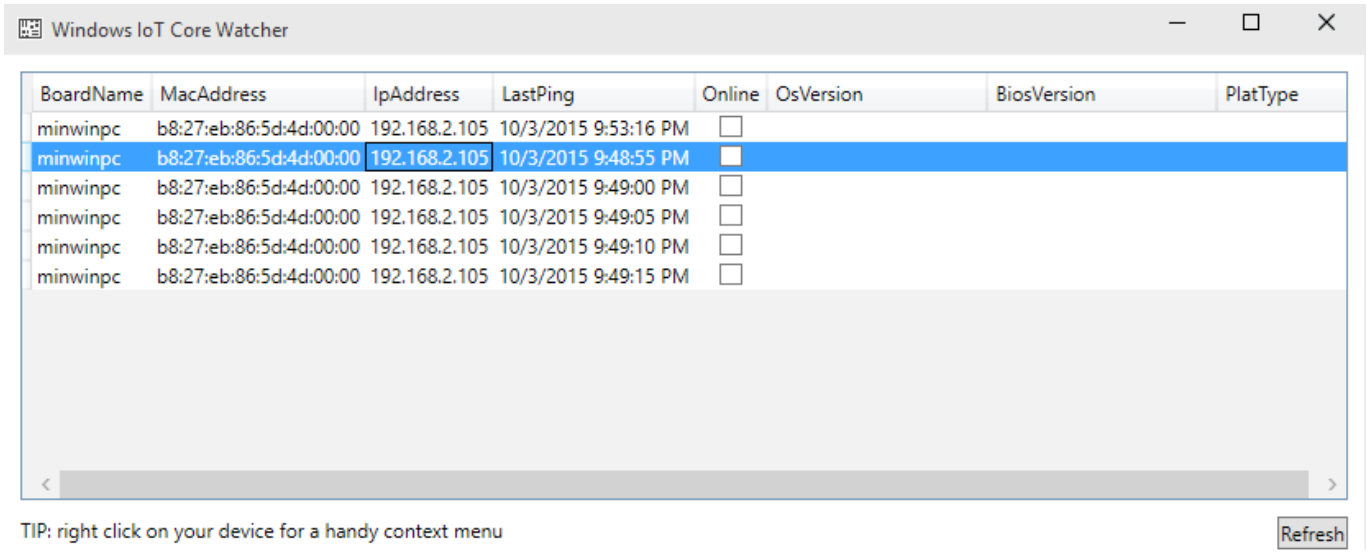

Abb. 3: Windows IoT Core Watcher

Länge von 1200 m mit bis zu 32 Slaves pro Master [Info2].

IEC 62056-21 ist nur eines von vielen Protokollen, das über RS-485 verwendet werden kann. IEC 62056-21 ist eine internationale Norm speziell für Smart-Meter und wird deshalb in unserem Praxisbeispiel verwendet [Info3]. Im Praxisbeispiel ist sie in eine „steckbare" Zustandsmaschine gekapselt und somit sauber von den RS-485- und Cloud-Repository-Schichten getrennt (vgl. Abbildung 2).

Setup für Entwicklungs-"DevOps"

Mit einem Windows 10 PC und Visual Studio 2015 ist man sowohl für die Entwicklung und Erstellung der ARM-Binary für den Raspberry Pi 2 und der Azure-Cloud-Services vorbereitet, auch für Deployment, Debugging und Monitoring in beiden Umgebungen. Damit steht eine hochgradig integrierte Entwicklungsumgebung aus einem Guss für den gesamten Lebenszyklus zur Verfügung.

Windows 10 IoT Core auf Raspberry Pi2 installieren

Es gibt inzwischen gute Online-Anleitungen von Microsoft und anderen für die Installation von Windows 10 IoT Core auf Raspberry Pi [Info4]. Den Raspberry Pi erreichen wir mit Passwortschutz von jedem Rechner im LAN. Windows 10 IoT Core liefert sogar ein paar nützliche Tools hierzu, wie z. B. IoT Core Watcher, die weiter unten kurz erläutert werden.

Die Modi „Headed", „Headless", und „Headed-Blind"

Eine Telemetry-to-Cloud-Lösung soll durchgängig manipulationssicher sein. Eine Zugangsmöglichkeit zum Raspberry Pi direkt im Feld würde eine unnötige Angriffsmöglichkeit bieten. Ohne einen „Kopf" (headless = ohne Bildschirm) kann niemand, nicht einmal mit einem gültigen Passwort, einfach auf das Gerät zugreifen.

Dieser „Headless Mode" sollte aber nicht schon bei Entwicklung und Test eingestellt sein, denn er hat gewisse Nebenwirkungen. Es gibt eigentlich drei Modi, die wir gezielt nutzen können, jeweils mit unterschiedlichen Vor- und Nachteilen:

- **„Headed Mode"** als Defaultmodus ermöglicht eine einfache Bedienung des Raspberry Pi mittels eigenes Bildschirms, einer Tastatur und Maus. Das hat in diversen Szenarien offensichtliche Vorteile. Vor allem dann, wenn eine Windows 10-App ein direktes Geräte-GUI hat – das geht mit Windows 10 IoT ohne Weiteres –, braucht man einen Bildschirm. Apps werden dann als Executable (.exe) auf den Raspberry Pi geladen, zum Beispiel direkt vom Visual Studio. Executables sind klassische Anwendungsformen, die eine sehr effektive Tool-Unterstützung bei der Entwicklung haben. Ein weiterer Vorteil des „Headed Mode" ist die Verfügbarkeit des kompletten IoT Core Feature-Sets, wie Debugger, Monitors, Watch-Dogs, FileBrowser, WebServer etc. Diese stehen im „Headless-Mode" nicht alle zur Verfügung, aber diese Features konsumieren auch CPU und RAM und sind dann bei einer Auslieferungskonfiguration nicht mehr wirklich von Nutzen.
- **„Headless Mode"** hält nur eine minimale Anzahl von Services am Laufen (keine Grafik- und Tastaturdienste), um Ressourcen zu sparen und die Angriffsfläche zu minimieren. Die ausgelieferten Apps haben dann keine GUI. Sie verfügen stattdessen über einen einfachen Task-Starter und werden nicht als „Executable" installiert, sondern als leichtgewichtige dynamische Bibliothek (project type „Library", output type „Windows Runtime Component") auf den Raspberry Pi geladen. Diese Komponenten werden dann zur Laufzeit bei Bedarf aktiviert. Nachteil dieses Modus ist der Wegfall einiger „nice-to-have"-Apps und -Features, die während der Entwicklung und je nach Szenario auch im ausgelieferten Zustand sehr nützlich sein können. Zum Beispiel laufen manche Device-Treiber nur innerhalb eines Executables, nicht als Component, also nicht im „Headless-Mode". Deshalb verwenden wir – vor allem bei Entwicklungs- und Testversionen – einen sogenannten „Blind-Headed Mode".
- **„Blind-Headed Mode"** (unsere Wortschöpfung) bietet für manche Anforderungen eine „Best of Both Worlds"-Konstellation. Hier belegen wir das GUI mit einem nicht reagierenden „blinden" GUI, um es zu blockieren, und stecken dann Bildschirm und Tastatur ab. Dadurch behält man die Vorteile des „Headed Mode" und reduziert gleichzeitig die Angriffsfläche. Man kann zwar Bildschirm und Tastatur anschließen, aber der Bildschirm bleibt schwarz und es erfolgt keine Reaktion. Man kann in diesem Modus sogar den Cloud-Monitor informieren, wenn jemand sich mit Tastatur oder Bildschirm direkt am Raspberry Pi zu schaffen macht. In diesem

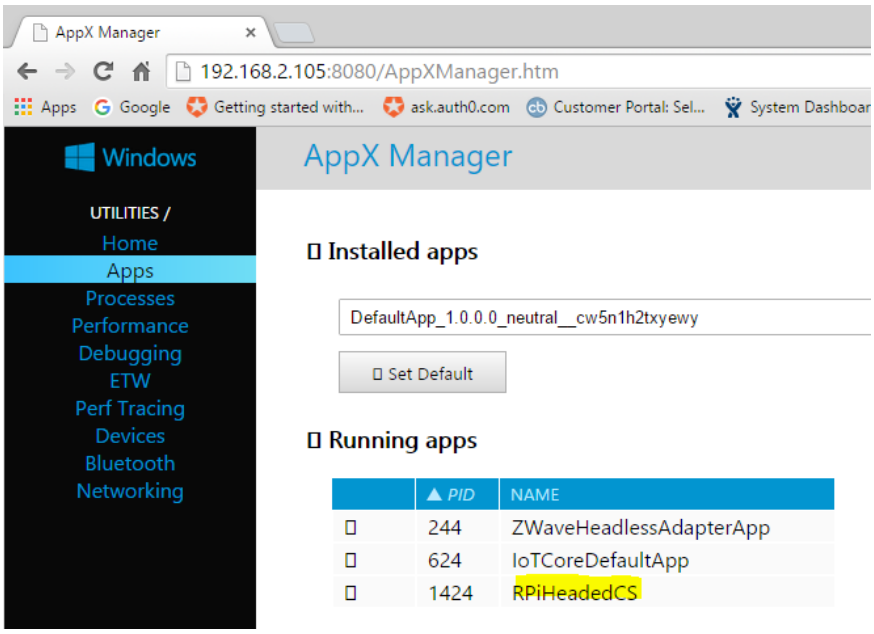

Abb. 4: Windows 10 IoT Web AppX Manager

Beitrag zum Praxisbeispiel verwenden wir den „Blind-Headed Mode“, in der weiteren Entwicklung wird dann voraussichtlich auf den „Headless Mode“ umgestiegen.

Windows IoT Core Watcher & Friends

Der Windows IoT Core Watcher ist eine Art Monitoring-Werkzeug, das man auf einem beliebigen Windows 10 PC im LAN starten kann. Wenn Sie Windows 10 IoT Core downloaden, ist die Installationsdatei “WindowsDeveloperProgramForIoT.msi” für den Core Watcher mit dabei. Wie in Abbildung 3 dargestellt, zeigt er dynamisch an, welche Windows IoT Core Devices gerade aktiv sind und ermöglicht auch eine Verbindung zum Device über den sogenannten Windows 10 IoT Web AppX Manager (Abbildung 4) oder auch über einen Windows Explorer File Browser. Die „mächtigste“ Verbindung mit dem Windows 10 IoT Core erfolgt aber über die PowerShell, die wir weiter unten nutzen werden, um die Gerätetreiber zu aktivieren.

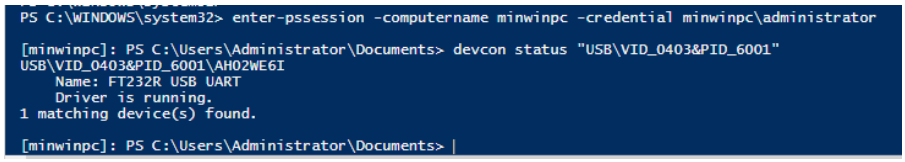

Abb. 5: IoT Core-Treiber mit PowerShell auflisten

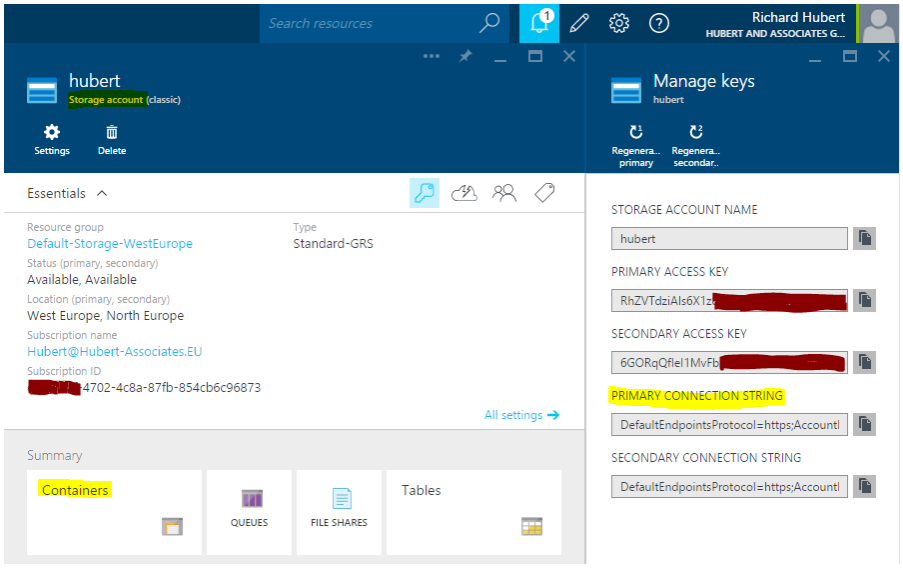

Abb. 6: Portal.Azure.Com Storage Blob Container Connect String

RS-485-Bridge-Treiber installieren und konfigurieren

Die RS-485-Bridge-Treiber müssen zunächst auf Windows 10 IoT Core installiert werden, da IoT Core in der aktuellen Ausführung nur einen RS-232-Treiber besitzt. Anschließend konfigurieren wir unsere App, um diesen Treiber zu erkennen und zu nutzen. Für den ersten Schritt wird der PowerShell-Zugang zum IoT Core-System genutzt.

Auch hier gibt es eine sehr gute Beschreibung von Microsoft, die exakt zu unserem Anwendungsfall passt. Folgen Sie den Anweisungen der Beschreibung im Link [Info5], bis Sie auf dem Raspberry Pi IoT-Core PC (minwinpc) eingeloggt sind.

Wir verwenden die D2XX Windows RT driver-Treiber für das FTDI-Chip von FTDI, weil bisher kein spezifischer Windows 10 IoT Core-Treiber existiert. Verifizieren Sie die Verfügbarkeit des Treibers im PowerShell mit dem Befehl `devcon status „USB\VID_0403&PID_6001”`. Das Ergebnis sollte wie in Abbildung 5 dargestellt aussehen:

Setup der Azure Storage

Die Telemetriedaten werden zunächst lokal auf der IoT Core-App zwischengespeichert. In Intervallen, die in der App konfiguriert werden, werden die gesammelten Daten als Microsoft Azure Blob gespeichert und dann vom lokalen Cache gelöscht. Weitere interessante Speicherarten und dynamische Verarbeitungsmechanismen werden von Microsoft Azure angeboten, wie in unserem Top-Level-Systembild (Abbildung 1) angedeutet.

Um den benötigten Azure Storage-Rahmen anzulegen, erstellen Sie zunächst ein Azure-Speicherkonto wie in der Anleitung [Info6] beschrieben. Kopieren Sie dann den Azure-Storage-Connect-String des Containers (siehe Abbildung 6) direkt in die Konfigurationsdatei der Windows 10 IoT-App.

Die Windows 10 IoT Core App deployen und testen

Sobald Sie das IoT-Core-Gerät, die Gerätetreiber und die Azure-Storage wie beschrieben eingerichtet haben, sind Sie startklar. Wir müssen nur noch unser Raspberry Pi

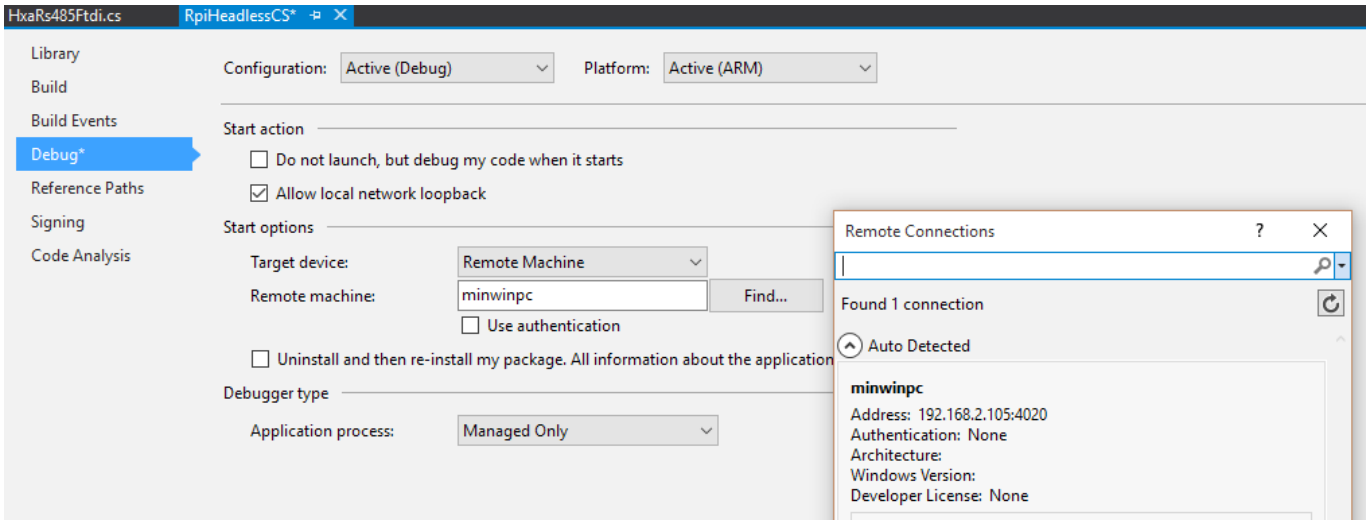

Abb. 7: Automatische Erkennung und Einrichtung des IoT-Core Device in Visual Studio

IoT mit dem Visual Studio 2015 auf unserem Entwicklungs-PC verknüpfen.

Um das IoT-Gerät einzubinden, starten Sie Visual Studio 2015 und wählen „Project => Project Properties“ in der obersten Menüleiste. Wählen Sie im jetzt angezeigten Formular den Tab „Debug“ und gehen Sie auf „Remote machine: Find…“, wie in Abbildung 7 dargestellt. Wählen Sie dann das erkannte Target-Device oder geben Sie notfalls die IP-Adresse des Devices an (diese IP-Adresse finden Sie auch auf dem Windows 10 IoT Core-Bildschirm direkt nach einem Neustart).

Jetzt können Sie die IoT-Core-App direkt von Visual Studio 2015 mit den gewohnten Tasten F5 (*Run mit Debug*) oder Strg-F5 (*Run*) starten. Das Programm wird als „Universal Windows Platform Program“ für die gesetzte Plattform (Platform: ARM) gebaut und automatisch auf die eingestellte „Remote machine“ provisioniert und dort gestartet. Sie können dann in Visual Studio 2015 die App so debuggen, als ob sie lokal auf dem Entwicklungs-PC laufen würde. Dadurch wird die Fehlersuche wesentlich erleichtert und die Testzeiten werden erheblich verringert.

Aus derselben Visual-Studio-Session können Sie dann auch über Server- oder Cloud-Explorer Ihre gesamte Azure-Cloud-Infrastruktur verwalten, provisionieren und sogar debuggen (siehe Abbildung 9). Damit hat man in einem einzigen Tool (Visual Studio) und in wenigen modernen Sprachen (C# V6, Javascript/TypeScript) auf dem Entwicklungs-PC die gesamte „End-to-End“-Strecke vom IoT-Gerät bis zu den Cloud-Services im Zugriff. Dies erhöht die Geschwindigkeit und gleichzeitig die Qualität.

Die von der IoT-App produzierte Cache-Datei können Sie mit dem „IoT-Core Windows Explorer File Browser“, wie oben beschrieben, direkt auf dem IoT-Core-Gerät einsehen. Das Senden dieser Datei ins „Azure Blob Storage“ erfolgt durch Einsatz des Microsoft „Azure Storage SDK“. Dieses SDK ist Teil der Universal Windows Platform und funktioniert deshalb quellcode-identisch zu einer „normalen“ Web- oder PC-Anwendung. Sie können das gespeicherte Azure Blob dann lesen und bearbeiten wie im nächsten Abschnitt beschrieben.

Telemetriedaten aus der „Azure Cloud Storage“ lesen

Unsere Telemetriedaten werden aktuell als

```
1 {"deviceguid":"0000000000238752","starttime":"2015-10-04T00:00:00.0000000-07:00","endtime"
  :"2015-10-05T00:00:00.0000000-07:00","readingtype":"QuarterHourly","comment":"demo only"
  ,"readings":[{"data":"..."},{"data":"..."},{"data":"..."},{"data":"..."}]}
```

Abb. 8: Telemetriedatenspeicher als „Azure Blob“ in JSON-Format

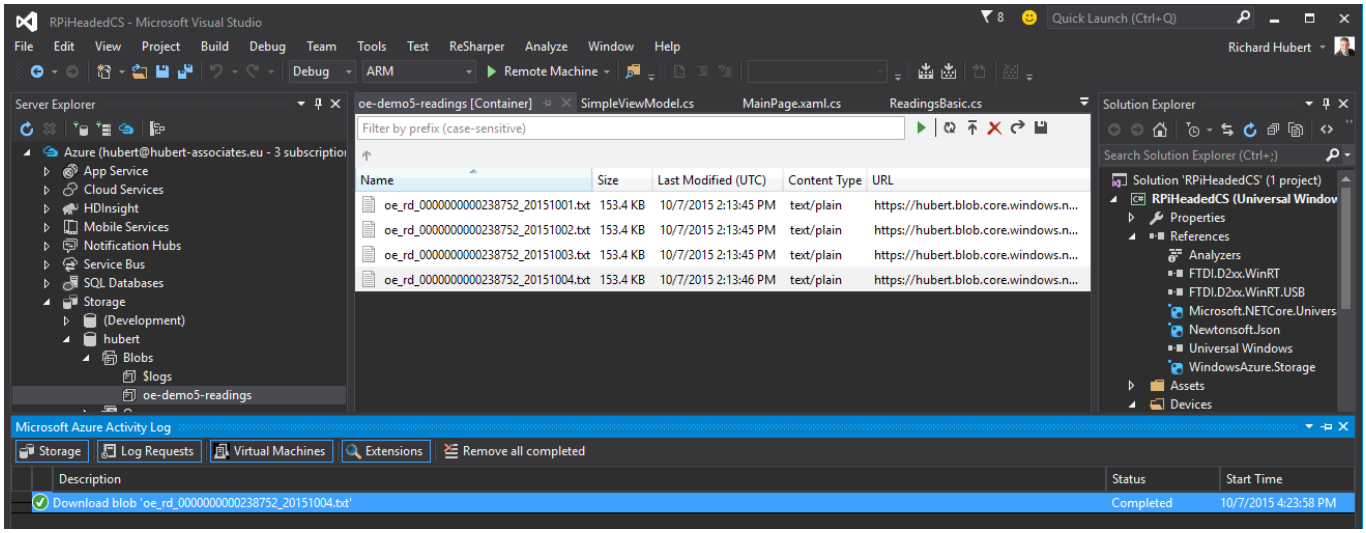

Abb. 9: Blob-Datei im Visual Studio 2015 „Azure Storage Explorer“ verwalten

Javascript JSON-Objekt im sicheren „Azure Blob Storage“ gespeichert, damit sie schnell und einfach als standardkonformes Objekt automatisch geprüft und verarbeitet werden können. Das Format enthält neben den eigentlichen Nutzdaten auch die notwendigen IoT-Endgeräteinformation und Zeitstempel (siehe Abbildung 8).

Es gibt viele Tools, um Azure Blobs sicher zu lesen oder zu bearbeiten. Nennenswert sind hier zum Beispiel das kostenlose Microsoft AzCopy, Neudesic Azure Storage Explorer oder LINQPad. Da wir Visual Studio 2015 haben, können wir, wie in Abbildung 9 gezeigt, mit dem Cloud-Explorer direkt und gesichert auf unsere „Azure Cloud Storage“ zugreifen.

Weitere Ausbaustufen

Bis zu diesem Punkt haben wir eine fertige Windows 10 Core IoT-App und eine sichere Verbindung zur Speicherung und zum Abrufen der Telemetriedaten in der Azure-Cloud. Allerdings, wie in unserem Top-Systembild oben angedeutet, nutzen wir bei Weitem nicht das gesamte Potenzial der Azure-Services. Diese Möglichkeiten sollen an anderer Stelle weiter beleuchtet werden, daher gilt: Stay tuned! ■

Referenzen

[Hub] Für den Quellcode des Praxisbeispiels gerne den Autor kontaktieren: hubert@hubert-associates.eu.

[Info1] Informationen zur USB-RS485-Bridge: Schaltung http://wiki.in-circuit.de/images/d/da/610000191A.pdf, Bezugsquelle http://shop.in-circuit.de/product_info.php?products_id=82, Treiber: http://www.ftdichip.com/Drivers/D2XX.htm.

[Info2] Informationen zu RS-485: https://en.wikipedia.org/wiki/RS-485. Weitere Informationen zum Thema RS-485 im industriellen Umfeld: The Industrial Communication Technology Handbook, Richard Zurawski, CRC Press, Taylor & Francis Group (https://www.crcpress.com/The-Industrial-Communication-Technology-Handbook/Zurawski/9780849330773). Informationen zu RS-485 vs. Ethernet: http://blog.robotiq.com/what-is-rs485communication-protocol.

[Info3] Informationen zur IEC 62056-21: https://en.wikipedia.org/wiki/IEC_62056#IEC_62056-21.

[Info4] Installation von Windows 10 IoT Core auf Raspberry Pi: http://ms-iot.github.io/content/en-US/win10/SetupPCRPI.htm.

[Info5] Zugang zu und Konfiguration eines Geräts mit Windows IoT Core über PowerShell: http://ms-iot.github.io/content/en-US/win10/samples/PowerShell.htm

[Info6] Anlegen eines Azure-Speicherkontos: https://azure.microsoft.com/de-de/documentation/articles/storage-create-storage-account/#erstellen-eines-speicherkontos