Der sprechende Kickertisch

Marco Buss, Opitz Consulting Deutschland GmbH

Was passiert, wenn man einen Kicker digitalisiert und mit Amazon Alexa in einen Raum sperrt? Dieser Artikel geht genau dieser Frage nach und zeigt an einem spaßigen Beispiel, wie man Alexa zu einer virtuellen Stadionsprecherin macht.

Der Tischkicker im Büro des Autors ist in seinem Unternehmen ein beliebter Pausentreffpunkt, an dem regelmäßig Matches und Turniere stattfinden. Die Punkte werden manuell gezählt – in der heutigen Zeit, in der alles digitalisiert wird, für den Autor ein unhaltbarer Zustand! Er begab sich also auf die Suche: Anfang 2016 wurde er auf die Amazon-IoT-Buttons aufmerksam und fand auch schon einige interessante und witzige Projekte im Internet, die mit diesen Buttons umgesetzt wurden. Bei einem dieser Projekte wurden von Alexa die Punkte beim Tischtennisspiel gezählt. Genau das war es, was ihm für den Kicker vorschwebte. Mitte Juni 2016 waren die IoT-Buttons auch in Deutschland verfügbar und er begann umgehend mit der Umsetzung.

Grobe Architektur-Skizze

Die Architektur der Lösung ist nicht ganz einfach: Die Punkte werden durch das Betätigen der IoT-Buttons gezählt. Ein in NodeJS geschriebener Server reagiert auf diese Events und interagiert mit dem Alexa-Voice-Service. Dieser persistiert die Punkte mithilfe eines Alexa-Skills als Lambda-Funktion in einer Dynamo DB und liefert eine Sprachantwort zurück. Die Antworten werden auf einem angeschlossenen Lautsprecher ausgegeben *(siehe Abbildung 1)*.

Ein Amazon Echo kann genutzt werden, um neue Spiele zu starten oder um den aktuellen Punktestand abzufragen. Der Aufbau war am Ende leider etwas umständlicher als vom Autor angenommen. Bevor er mehr über Alexa-Skills und den Alexa-Voice-Service wusste, dachte er, über die IoT-Buttons einen Event auslösen zu können und dann direkt auf einem Echo auszugeben. Das war allerdings so nicht umsetzbar.

AWS IoT

Mit der Amazon Internet of Things Plattform (AWS IoT) bietet Amazon einen komplett gemanagten Service für IoT-Anwendungen beliebiger Größe. Wie für Serverless-Angebote üblich, müssen sich die Anwender dabei keine Gedanken über den Betrieb und die Skalierung machen. All diese Aufgaben werden durch Amazon selbst abgedeckt. Es entstehen lediglich Kosten, wenn der Service auch wirklich verwendet wird.

Das Herzstück von AWS IoT ist ein MQTT-Broker für den Nachrichtenaustausch. Abgerechnet wird nach der Anzahl der Nachrichten und nach dem verbrauchten Datenvolumen. Beim Thema „Sicherheit", das ja für alle IoT-Anwendungen ein wichtiger Punkt ist, bietet Amazon hier die Authentifizierung und Autorisierung basierend auf Zertifikaten an. Ohne ein gültiges Zertifikat ist also keine Kommunikation möglich.

Neben diesen Features bietet die Plattform noch eine Rules Engine sowie sogenannte „Device Shadows". Mit der Rules Engine können Nachrichten der Devices bereits vorgefiltert werden. Die Device Shadows sind eine Möglichkeit, mit Devices zu kommunizieren, die nicht permanent online sind. Ist ein Device nicht online, kann sein Zustand trotzdem angepasst werden. In diesem Fall wird lediglich das Device Shadow verändert und es erfolgt eine Synchronisation des Zustands, sobald das Device wieder online ist *(siehe Abbildung 2)*.

So funktioniert das Zählen der Punkte im AWS IoT: Jeder IoT-Button besitzt eine eindeutige ID und ist vor der Benutzung einzurichten. Zum einen muss er in ein WLAN eingebunden sein, zum anderen müssen entsprechende Zertifikate gespeichert werden. Die benötigten Zertifikate hat der Autor einfach auf der AWS-IoT-Website erzeugt. Anschließend ist der Button direkt einsatzbereit. Bei jedem Klick wird eine Nachricht an „MQTT Topic iotbutton/<ID des Buttons>" versendet. *Listing 1* zeigt eine Beispielnachricht.

Neben der Seriennummer ist der „clickType" interessant. Unterschieden wird zwischen einem langen Druck auf die Taste, einem normalen Klick und einem Doppelklick. Für den Kicker wurde die NodeJS-Anwendung per MQTT mit AWS IoT verbunden. Sobald die Nachricht eines IoT-Buttons empfangen wird, startet die entsprechende Routine, je nachdem, ob der entsprechende Button von Spieler 1 oder von Spieler 2 betätigt wurde.

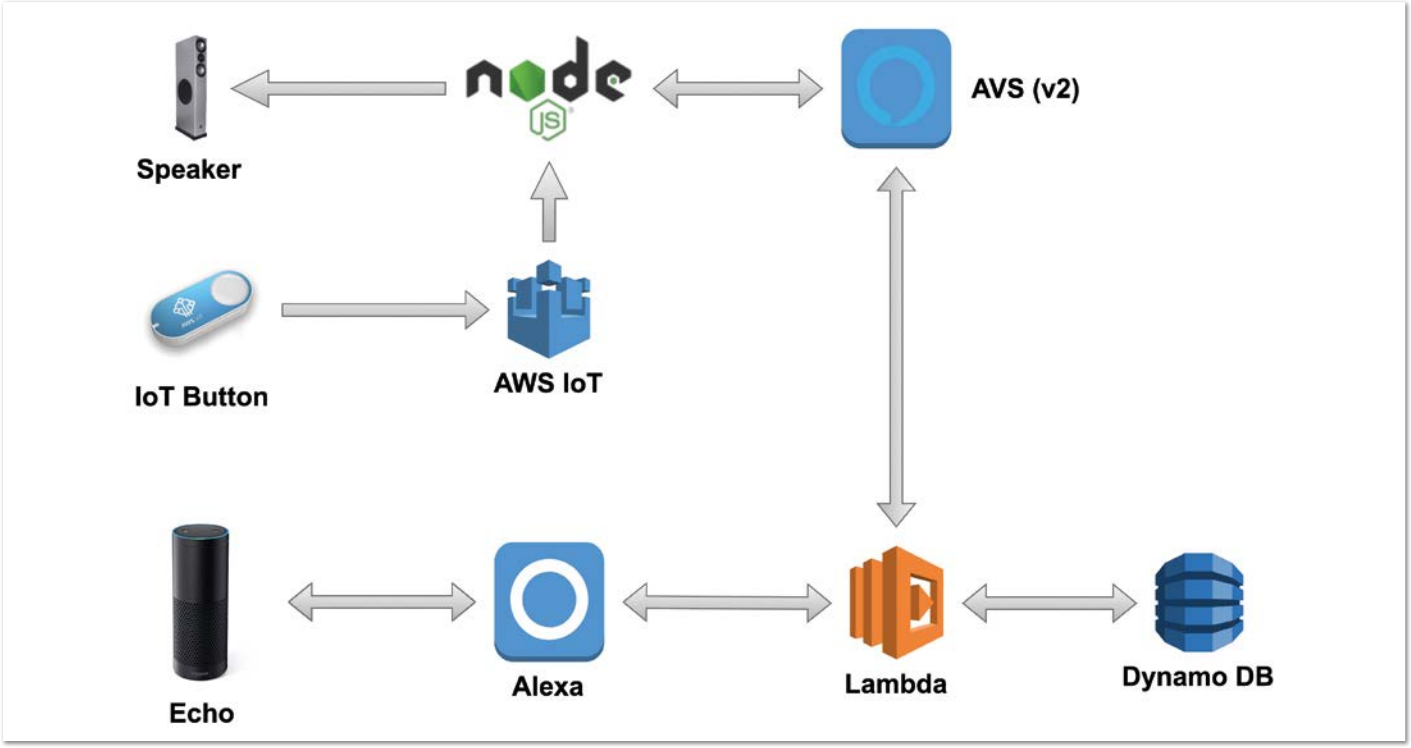

Abbildung 1: Die Architektur des Use Case „Sprechender Kickertisch"

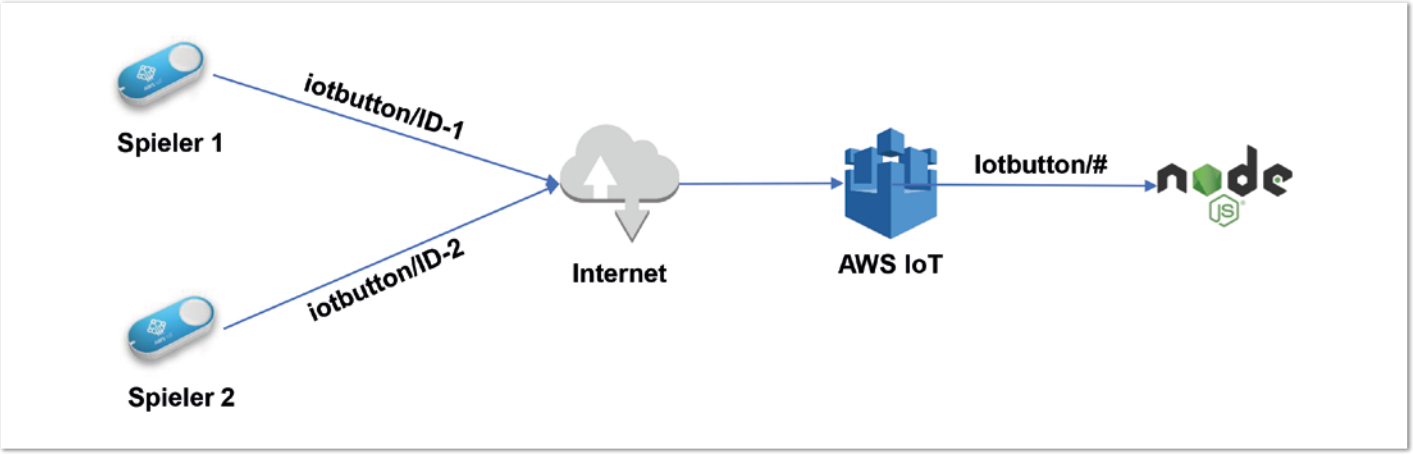

Abbildung 2: Aufbau des AWS IoT zur Punktezählung

Alexa Voice Service

Viele Anwender wissen nicht, dass Alexa nicht nur aus den Alexa-Skills besteht, sondern auch den Alexa-Voice-Service (AVS) beinhaltet. Dieser ist das eigentliche Herzstück jedes Echos und ist beispielsweise für die Auswertung der Sprache zuständig.

Anstatt diesen Service nur für eigene Produkte zu verwenden, bietet Amazon mit AVS für jeden Anbieter die Grundlagen, um Alexa in eigene Produkte zu integrieren. Diese Technik wird von immer mehr Herstellern genutzt: Seat plant beispielsweise, Alexa in einige Autos zu integrieren. Auch einige Lautsprecher-Hersteller haben eine Integration vor oder bereits entsprechende Produkte auf den Markt gebracht.

```
{
  "serialNumber": "ABCDEFG12345",
  "clickType": "SINGLE",
  "batteryVoltage": "2000 mV"
}
```

Listing 1: Nachricht des IoT-Buttons

Für den leichten Einstieg bietet Amazon ein C++-SDK an sowie detaillierte Anleitungen, wie dieses auf einem Raspberry Pi oder Linux/Mac-Rechner zu installieren ist. Für die Kommunikation mit AVS verwendet das SDK eine HTTP-basierte Schnittstelle. Mit deren Hilfe ist es möglich, diesen Service mit sehr vielen Programmiersprachen anzubinden. Neben dem C++-SDK existiert im Alexa-Git-Repository auch eine Java-Demo-Applikation *(siehe „https://github.com/alexa/alexa-avs-sample-app“).*

AVS-Security

Authentifizierung und Autorisierung erfolgen bei AVS Token-basiert. Für jede Interaktion ist ein Access-Token erforderlich. Um diesen zu erhalten, muss ein Device die Möglichkeit bieten, mit dem Dienst „Login with Amazon“ zu interagieren. Dafür stehen verschiedene Möglichkeiten zur Verfügung. Im einfachsten Fall kann man eine Companion Site oder App verwenden.

Durch „Login with Amazon“ ist auch die Verbindung vom Device zum Alexa-Nutzer festgelegt. Nach erfolgreicher Authentifizierung erhält die Companion Site oder App alle benötigten Tokens, um ein neues Access-Token anzufragen. Diese Tokens haben im Gegensatz zum

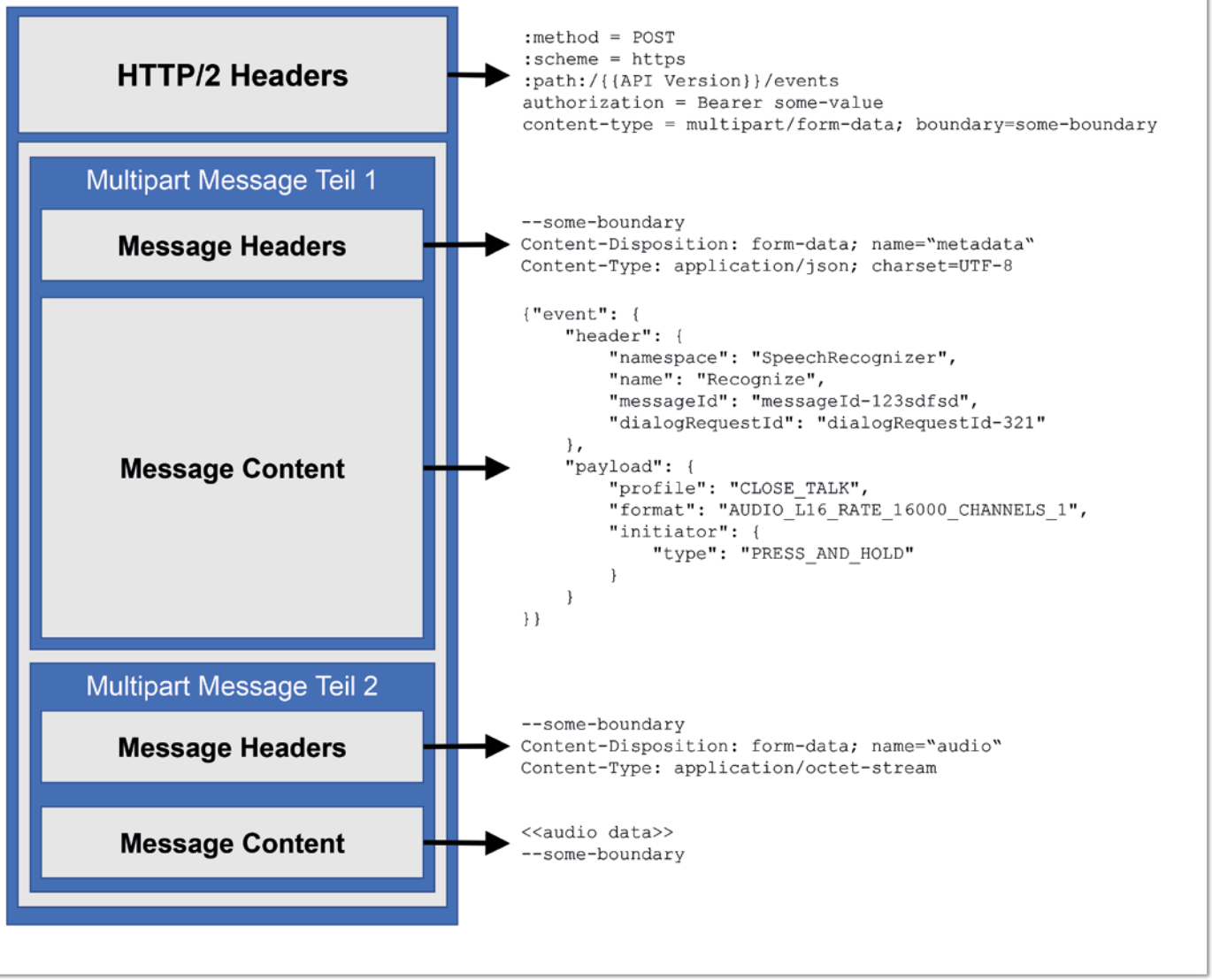

Abbildung 3: Aufbau einer AVS-Nachricht

Access-Token eine lange Lebenszeit und sollen auf dem Device gespeichert sein. Das Access-Token hingegen hält nur eine Stunde und ist daher in entsprechenden Abständen zu erneuern.

AVS v2

Anfang 2016 kam die Version 2 der AVS-Schnittstelle heraus. Sie brachte einen Umstieg auf HTTP/2 sowie eine signifikante Erweiterung des API. Während die Version 1 noch auf einem klassischen Request/Response-Prinzip basierte, führte Version 2 ein neues Kommunikationsmodell ein, das auf Events und Direktiven aufbaut.

Events sind dabei die Nachrichten, die vom Endgerät zum AVS-Service gesendet werden. Direktiven wiederum sind Anweisungen, die der AVS-Service initiiert. Diese Art der Kommunikation ist notwendig, um beispielsweise die neuen Notifications und die Steuerung des Endgeräts mit der Alexa-App zu ermöglichen.

Um die Schnittstelle zu nutzen, wird eine einzige HTTP/2-Verbindung zum AVS aufgebaut. Sobald die Verbindung steht, muss innerhalb von zehn Sekunden ein sogenannter „Downchannel-Stream" erzeugt werden. Dieser wird vom AVS verwendet, um die Direktiven zum Client zu senden. Der Stream befindet sich clientseitig in einem „half-closed State". Das bedeutet, dass der Client keine weiteren Daten sendet, aber weiterhin Daten vom Server empfangen kann.

Sobald der Downchannel-Stream erzeugt wurde, muss ein Sync erfolgen. Dieser teilt dem AVS den Zustand des Endgeräts mit. Er enthält beispielsweise Informationen zur aktuell eingestellten Lautstärke oder zum Mediafile, das gerade abgespielt wird. AVS-Messages sind HTTP/2-Multipart-Messages *(siehe Abbildung 3)*. Im HTTP/2-Header befinden sich allgemeine Informationen wie die Autorisierungsinformationen in Form des Access-Tokens.

Jeder Teil der Multipart Message besteht ebenfalls aus einem Header und dem eigentlichen Content. Im ersten Teil besteht der Content aus Informationen, die besagen, um welchen Event es sich handelt, und aus Parametern, die den Event näher beschreiben. Im zweiten Teil folgen dann die eigentlichen Nutzungsdaten wie im Beispiel ein Audio File.

```
{"intents": [
  {"intent": "StartGame"},
  {"intent": "Punktestand"},
  {"intent": "PlayerPoint",
     "slots": [
        {
           "name": "Player",
           "type": "SPIELER_PUNKT"
        }
     ]},
  {"intent": "AMAZON.HelpIntent"},
  {"intent": "AMAZON.StopIntent"},
  {"intent": "AMAZON.CancelIntent"},
  {"intent": "AMAZON.YesIntent"},
  {"intent": "AMAZON.NoIntent"}
]}
```

Listing 2: Intent-Schema des Kickertisch-Skills

```
Punktestand wie ist der punktestand
Punktestand wie steht es
Punktestand gib mir den punktestand
PlayerPoint Punkt für {PLAYER_POINT}
```

Listing 3: Sample Utterances

AVS und der Kicker

Nachdem die Grundlagen für den AVS nun geklärt sind, können wir uns der Frage nähern, wie der AVS genutzt werden kann, um die Punkte eines Kickerspiels zu zählen.

Wie bereits erwähnt, bekommt die NodeJS-Anwendung eine Nachricht per MQTT. Diese informiert sie darüber, welcher Button gedrückt wurde. Damit weiß sie, für welches Team der Punkt gezählt werden soll. An dieser Stelle kommt das Speech-Recognizer-Interface des AVS zum Einsatz; im Übrigen das einzige Interface des AVS, das für den Use Case benötigt wird. Es wird verwendet, um Sprachbefehle zu analysieren und die Befehle gemäß den Informationen des Alexa-Skills zu starten. Dafür benötigt das Speech-Recognizer-Interface ein Sprach-Sample. Das könnte beispielsweise „Alexa, wie spät ist es?" sein, wie es in der Interaktion mit einem Amazon-Echo üblich ist. Im Kickerspiel wäre ein Beispiel für ein Sprach-Sample „Punkt für Rot" oder „Punkt für Blau".

Das Speach-Recognizer-Interface interessiert nicht die Quelle des Sprach-Samples. Amazon Echo verarbeitet Sprach-Samples, die per Mikrofon aufgenommen wurden. Bei der Kicker-Anwendung wurde für jede Seite im Vorfeld ein entsprechendes Audiofile erzeugt und an den AVS gesendet. Die Samples müssen dabei folgende Eigenschaften erfüllen:

- 16 Bit Linear PCM
- 16 kHz Sample Rate
- Single Channel
- Little Endian Byte Order

Die verwendeten Sprach-Samples müssen übrigens nicht selbst gesprochen und entsprechend umgewandelt werden, dafür gibt es einen weiteren Service von AWS: Amazon Polly, ein Text-to-Speach-Service, den man beispielsweise in einen Blog einbinden kann, um die Posts vorlesen zu lassen, erzeugt auf einfache Weise die benötigten Dateien.

Im Gegenzug liefert der Speech-Recognizer-Service als Antwort ein Audiofile mit dem Ergebnis des Alexa-Skills. Beim Kickerbeispiel beinhaltet die Antwort unter anderem den aktuellen Spielstand. Die NodeJS-Anwendung gibt das Ergebnis über die angeschlossenen Lautsprecher aus.

Alexa-Skills-Kit

Die eigentliche Verarbeitungslogik ist im vorgestellten Beispiel als Alexa-Skill implementiert. Der AVS erkennt im Sprach-Sample, dass der Tischkicker-Skill aufgerufen werden soll, um den Punkt für einen Spieler zu zählen.

Alexa-Skills sind mit dem Alexa-Skills-Kit implementiert und bestehen aus zwei Teilen: einem Interaction Model und der eigentlichen Service-Funktion. Das Interaction Model enthält ein Intent-Schema und die Sample Utterances. Das Intent-Schema definiert, welche

Befehle der Skill unterstützt. Es wird zwischen eingebauten Intents und selbst definierten Intents unterschieden.

Listing 2 zeigt das Intent-Schema des Tischkicker-Skills. Die eingebauten Intents erkennt man am „AMAZON."-Präfix. Die restlichen Intents sind eigene Intents, die nur für den Skill gelten. Jeder Intent kann auch sogenannte „Slots" enthalten, wie beim „PlayerPoint"-Intent zu sehen ist. Slots definieren die Parameter eines Intent. Auf diese Parameter kann das System auch innerhalb der Service-Funktion zugreifen.

In den Sample Utterances sind Beispielsätze für die eigenen Intents abgelegt. Für jedes Intent sollten mehrere Beispielsätze definiert werden, damit der AVS-Service Intents und Slots entsprechend erkennen kann. In *Listing 3* sind die Sample Utterances für die beiden Intents „PlayerPoint" und „Punktestand" zu sehen.

Die Service-Funktion kann ein Web-Service sein, wird in den meisten Fällen jedoch als AWS-Lambda-Funktion implementiert. Für die Implementierung des Skills als Lambda-Funktion existieren verschiedene SDKs. Der Tischkicker-Skill ist in NodeJS implementiert.

Mithilfe eines Amazon Echo kann ein neues Spiel gestartet und der Punktestand unabhängig von einem gefallenen Tor erfragt werden. Da die Punkte ebenfalls mit diesem Skill gezählt werden, kann ein Tor auch ohne die IoT-Buttons mit den Kommandos „Punkt für Blau" oder „Punkt für Rot" gezählt werden.

Fazit

Der Artikel zeigt, wie einfach es sein kann, eine neue Idee mit sprachbasierten Systemen zu verwirklichen. Amazon bietet dafür mit seinen diversen Services – allen voran den Amazon-Voice-Services – eine gute Grundlage. Der Anbieter begünstigt zudem mit recht großzügigen Frei-Kontingenten für viele Services auch das Experimentieren mit den einzelnen Diensten.

Marco Buss
marco.buss@opitz-consulting.de

Marco Buss ist Senior Consultant bei Opitz Consulting und beschäftigt sich seit mehr als zehn Jahren mit der Java-Entwicklung im Enterprise-Umfeld. Seine aktuellen Themenschwerpunkte sind Cloud, Big Data und IoT.