Michael Bauer
michael.bauer@maibornwolff.de)
ist Software Engineer bei MaibornWolff.

IoT meets Big Data
Anforderungen an eine IoT-getriebene Big-Data-Lösung

Unternehmen verbessern Produkte mit Informationen aus Datenanalysen und schaffen so Mehrwert für Kunden. Grundlage sind immer häufiger Daten von vernetzten IoT-Geräten. Doch ein System, das viele vernetzte IoT-Geräte betreibt und von ihnen gelieferte Daten performant auswertet, braucht eine belastbare Architektur. Der Artikel beschreibt die Herausforderungen an eine Big-Data-Architektur, die Steuerungsmöglichkeiten für eine Flotte von IoT-Geräten anbietet, Features zur Datenanalyse für Reports und Echtzeit-Anwendungen unterstützt und Schnittstellen für andere Business-Anwendungen, etwa für die Rechnungserstellung, mitbringt.

Anwendungsfall: Drohnen-Pakettaxis

Sensor-bestückte IoT-Geräte liefern Unternehmen immer mehr Nutzdaten, die zur Verbesserung eines Produkts oder eines Systems verwendet werden können. Die Informationen sind umso wertvoller, je mehr Geräte organisiert und vernetzt werden, sei es in Form von Fahrzeugen, Drohnen, Smartphones oder Haushaltsgeräten.

Stützt sich die Datenauswertung auf eine größere Zahl von verteilten Geräten oder eine Langzeitbeobachtung einer Geräteflotte, können Muster erkannt und ausgewertet werden, die weit über den Informationsgehalt einzelner Sensorwerte hinausgehen. Werden Informationen darüber hinaus aus verschiedenen Quellen zu einer Analyse kombiniert, lassen sich bis dahin unverstandene Phänomene einordnen. Beispielsweise erklären sich untypische Sensorwerte einer Flugdrohne möglicherweise durch hohe Windstärken; die Korrelation ist nur erkennbar, wenn der Drohnenbetreiber die Sensordaten der Geräte mit Wetterinformationen für die entsprechenden Flugzeiten aggregiert.

Doch wie muss eine Lösung aussehen, die die verschiedenen Anforderungen – Reports und Echtzeit-Anwendung – be-

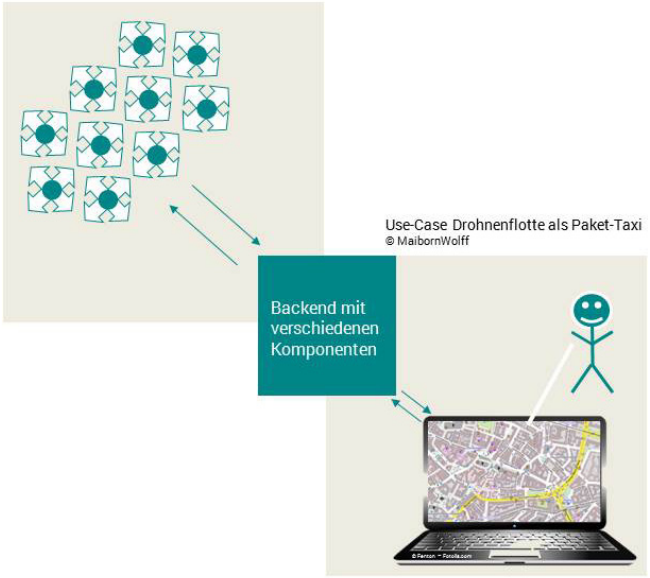

Abb. 1: Anwender kommunizieren mit dem Backend-System, das die Drohnenflotte verwaltet

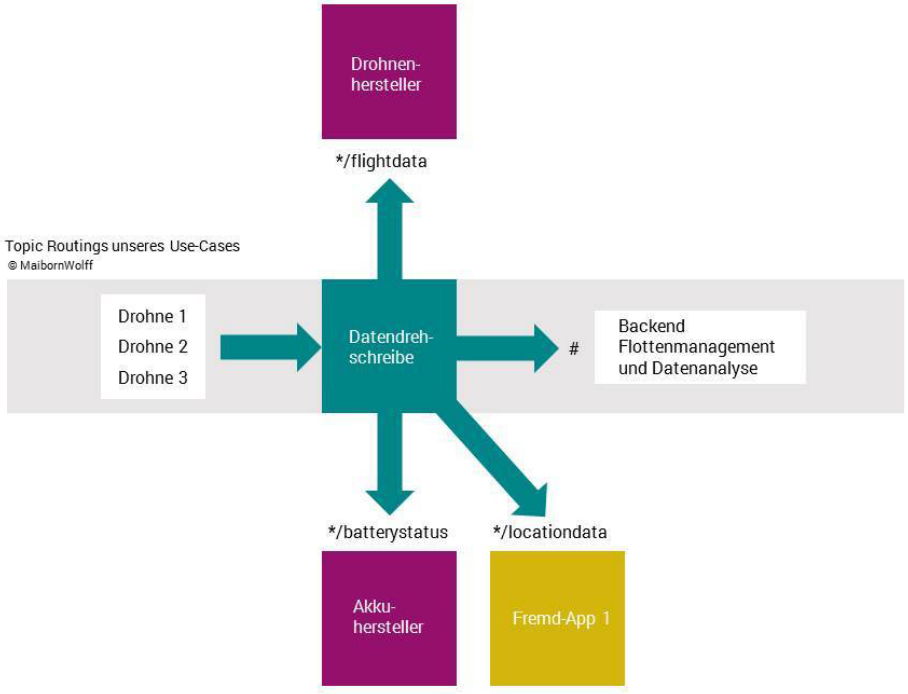

Abb. 2: Topic-Routing an verschiedene Business-Abnehmer

rücksichtigt? Wir konstruieren einen Fall: Eine Drohnenflotte liefert wie Pakettaxis Express-Briefe und Waren aus. Der Versender bucht per App eine Drohne in seiner Nähe, gibt den Empfänger in einer App ein und hängt das Paket an. Während die Drohne das Paket ausfliegt, können Absender und Empfänger auf einer Karte in Echtzeit die aktuelle Position des Pakets abrufen.

Immer auf dem Laufenden sein: Steuerungs- und Auswertungs-Applikation fürs Pakettaxi

Der Drohnen-Taxi-Anbieter möchte die Daten der Drohnen auswerten, um das Flottenmanagement zu optimieren: Reports fassen täglich oder wöchentlich die häufig frequentierten Flugrouten, Gebiete mit hoher Abdeckung oder das durchschnittliche Paketgewicht zusammen. Ad-hoc-Abfragen zeigen den aktuellen Aufenthaltsort jeder Drohne inklusive Kilometerstand, Batterieladestatus und anstehenden Aufträgen.

Fehlerfälle, etwa mechanische Probleme oder ein Verbindungsabbruch, lösen in Echtzeit Alarm beim Flottenmanager aus. Andere Business-Anwendungen rufen ebenfalls Betriebsdaten ab: Beispielsweise greift das Wartungssystem des Drohnenherstellers auf den aktuellen Kilometerstand zu, damit Verschleißteile rechtzeitig ausgetauscht werden können.

Betrieb und Steuerung einer Geräte-Flotte, die Anbindung anderer Systeme und die Auswertung der Daten stellen bei Datenübertragung, Datenverteilung und Datenverarbeitung eigene Herausforderungen an die Architektur des Systems, das diese Aufgaben erfüllen soll.

Leichtgewichtiges Übertragungsprotokoll für die kommunizierende Flotte

Aus dem konstruierten Szenario einer Drohnen-Flotte ergeben sich verschiedene Anforderungen an die Datenübertragung: Bei mehreren mobilen IoT-Geräten wie unseren Drohnen muss die Datenübertragung leichtgewichtig bleiben. Die Übertragung der Informationen muss priorisiert werden. Und die Nachrichtenübertragung soll so einfach wie möglich sein. So schont sie die Akkulaufzeit, berücksichtigt die oft geringe Leistungsfähigkeit der verbauten Hardware und minimiert die notwendige Bandbreite für die Datenübertragung zwischen Drohne und Backend.

Der Overhead für Nachrichten wird auf ein Minimum begrenzt; die reduzierte Datenmenge hilft, mit der Bandbreite auszukommen und die Kosten für Mobilfunk gering zu halten. Für einige Nachrichten muss sichergestellt werden können, dass sie den Empfänger erreichen. Das bedeutet, dass Verbindungsabbrüche abgefangen werden und die Kommunikation zu einer Drohne nicht dauerhaft verloren geht.

Das Protokoll MQTT (www.mqtt.org), [MQTT]) eignet sich besonders gut für zuverlässige und effiziente Kommunikation über instabile Mobilfunknetze und Tausender vernetzter Geräte mit einem zentralen Backend. Das Maschine-zu-Maschine-Protokoll ist inzwischen eines der wichtigsten Protokolle im Internet der Dinge. Implementierungen gibt es von verschiedenen Anbietern, sowohl als proprietäre Lösung, beispielsweise HiveMQ [HiveMQ], wie auch als Open-Source-Lösung, wie RabbitMQ [RabbitMQ].

MQTT eignet sich, um Nachrichten mit geringem Netzwerk-Overhead zwischen Client und Server zu übertragen. Das Übertragungsprotokoll arbeitet Payload-unabhängig. Das bedeutet, dass das Nachrichtenprotokoll separat definiert werden muss. Es nutzt dafür ein Publish-/Subscribe-Pattern. Es bietet drei Quality-of-Service-Levels, die es uns erlauben, Nachrichtentypen zu priorisieren:

- *at most once*: Jede Nachricht wird nur einmal verschickt, unabhängig davon, ob sie den Empfänger erreicht oder nicht. Dieser Modus eignet sich für sich schnell verändernde Sensordaten, die einander ersetzen. Vorstellbar ist er beispielsweise für GPS-Koordinaten, die im Sekundentakt verschickt werden.
- *at least once*: Jede Nachricht wird mindestens einmal verschickt und es dürfen Duplikate vorkommen. Das ist für viele Statusänderungen nützlich, die das Backend mindestens einmal erreichen müssen. Ein Beispiel ist eine Warnung der Drohne über den Batterie-Ladestatus, nach der das Backend-System die Drohne zur Ladestation schickt.
- *exactly once*: Buchungen und Zahlungen des Endkunden sind Informationen, die ankommen müssen, aber nicht dupliziert werden dürfen. Bricht die mobile Datenverbindung der Drohne ab, muss eine Fallback-Lösung über SMS greifen.

MQTT unterstützt die initiale Authentifizierung der Clients. Datenübertragung per SSL ist möglich, verursacht jedoch signifikanten Netzwerk-Overhead. Alternativ kann der Payload vor dem Versenden durch den Client verschlüsselt werden. Das verhindert Overhead beim Transport, der Berechnungsaufwand entsteht auf dem Client und dem Server.

Durch geschickte Codierung des Nachrichten-Payloads kann die Datenübertragung ebenfalls effizienter werden. Eine Möglichkeit ist die Verwendung von Google Protocol Buffers [GPP]: Um auf

die Attribute der Nachrichten zuzugreifen, werden Nachrichten in einer eigenen Syntax definiert. Aus dieser Nachrichtendefinition kann Nachrichten-bezogener Code für verschiedene Programmiersprachen erzeugt werden. Die Nachrichtendefinition kann als Schnittstellendokumentation schnell an neue Abnehmer verteilt werden. Das System ist dadurch einfach erweiterbar. Außerdem ist es rückwärtskompatibel. Das stellt sicher, dass die Daten über verschiedene Geräteversionen konsistent bleiben – Grundlage für Auswertungen und Analysen.

Eine Reduktion der Datenübertragung wird von Architekturentscheidungen unterstützt: Um Redundanz zu vermeiden, schickt nicht jedes Gerät seine Daten einzeln an die verschiedenen Abnehmer. Drohnen und Drittsysteme kommunizieren stattdessen über eine Datendrehscheibe miteinander. Das Konzept der „Datendrehscheibe“ als dem Backend vorgeschalteter Message Broker hilft, eine effiziente Kommunikation der Geräte zu einer beliebig erweiterbaren Menge von Abnehmern zu implementieren.

Datendrehscheibe bindet Business-Anwendungen an

Die Datendrehscheibe nimmt Daten von verschiedenen Publishern – in diesem Fall unseren Pakettaxis – in einem Kanal entgegen und verteilt sie nach Bedarf an weitere Anwendungen wie CRM- oder Buchhaltungssysteme. Der MQTT-Server verwendet dafür „Topics“, vereinfacht gesagt sind das Regeln, die Nachrichten nach festgelegten Kriterien filtern und verteilen.

Durch Wildcards in den Topics kann ein Subscriber eine Gruppe von Nachrichten abonnieren: In unserem Szenario ist die eindeutige ID der Drohne als erstes Topic-Level definiert. Auf dem zweiten Level werden die Subsets der versendeten Nachrichten, beispielsweise Standortdaten oder Buchungen, unterschieden. Das Topic „*/bookings“ fasst dann die Buchungsinformation aller Drohnen zusammen. Sie können an ein zentrales CRM-System geleitet werden. Rückblickend lassen sich Nachrichten über die Sender-ID eindeutig einem Absender zuordnen, ohne Platz im Payload der Nachricht zu verbrauchen.

Backend: „Volume, Velocity, Variety“

Hauptabnehmer für Nachrichten der Datendrehscheibe ist in unserem Szenario das Flottenmanagement-Backend. Es bekommt alle Daten, um die Flotte zu managen. Viele Geräte produzieren eine große Datenmenge, für die klassische Business-Anwendungen nicht ausgelegt sind. Konkret haben sie Grenzen bei

- hohen Datenmengen, da jedes Gerät Daten mit verschiedenen Sensoren erfasst (zum Beispiel Temperatur, GPS),
- hoher Frequenz der Daten, da die Geräte ihre Daten in sehr kurzen Zeitabständen versenden (zum Beispiel GPS-Koordinaten während des Flugs),
- Inhomogenität der Daten, da die verschiedenen Sensoren ihre Werte in verschiedenen Formaten versenden können, oder die Daten keinen sinnvollen Bezug zueinander haben (etwa Temperaturwerte im Vergleich zu GPSKoordinaten).

Diese Eigenschaften von Daten werden als „Volume, Velocity, Variety“ [IBMGraphic] bezeichnet. Big-Data-Architekturen nehmen sich speziell der Herausforderungen an, die aus den „drei Vs“ entstehen. Für unser Flottenmanagement wählen wir einen Technologie-Stack mit Komponenten aus dem Hadoop-Ökosystem [Hadoop]. Der Hadoop-Stack lässt miteinander konkurrierende Technologien zu. Beim Entwickler liegt die Entscheidung, welche zum Einsatz kommt. Alternativ geben Big-Data-Stack-Lösungen von Hortonworks oder Pivotal die einzelnen Komponenten schon vor.

Anforderungen an eine Big-Data-Architektur

Jenseits der Technologie- und Tool-Ebene ist die Frage spannend, wie eine technische Architektur der Big-Data-Lösung aussieht, die den fachlichen Anforderungen des Flottenmanagement-Szenarios gerecht wird.

Eine Big-Data-Anwendung macht die Daten aus den Sensoren unserer IoT-Geräte zugänglich: Unternehmen wie der Betreiber unserer fiktiven Drohnen-Flotte können schnell auf Ereignisse in der Flotte reagieren, zum Beispiel auf Geräteausfälle. Analysen und Auswertungen der entstandenen Daten zeigen im Idealfall Schwachstellen im aktuellen System, Optimierungspotenzial und vollständig neue Handlungsfelder.

Die Architektur berücksichtigt alle Anforderungen für das Datenmanagement aus unserem Anwendungsfall: Gewünscht sind Reports und Echtzeit-Abfragen. Reports werten historische Daten aus und dienen als Grundlage für Geschäftsentscheidungen. Werden etwa bestimmte Routen oder Gebiete häufig überflogen, lohnt es sich eventuell, dort mehr Drohnen zu stationieren oder weitere Aufladestationen zu installieren.

Aktuelle Vorgänge wie Positionsbestimmungen, live visualisiert auf einer Karte, brauchen dagegen eine Echtzeit-Datenverarbeitung. Die Spanne zwischen zwei Updates sollte so gering sein, dass der Flottenmanager schnell und präzise reagieren kann, etwa wenn eine unserer Drohnen meldet, dass die restliche Akkuladung nur noch knapp für den Flug zur Ladestation reicht und die Rückkehr vom Flottenmanager eingeleitet werden muss.

Die Anforderungen für Reports und Echtzeitdatenverarbeitung widersprechen sich jedoch: Während Echtzeitdaten sehr schnell verfügbar sein müssen, brauchen ausführliche Datenanalysen für Reports eine sehr große Datenbasis und einen robusten, persistenten Datensatz. Das dehnt die Berechnungszeit für Analysen schnell auf mehrere Stunden aus.

Im Unterschied zu Echtzeitanfragen müssen große Reports nicht kontinuierlich durchgeführt werden, sie können nachts als Batchverarbeitung laufen und Views erzeugen, damit Reporting-Tools schnellen Zugriff auf Ergebnisse haben. Außerdem muss die Datenbasis robust gegen Fehler in den Daten sein. Daten für Echtzeit-Anwendungen dürfen dagegen flüchtiger sein, da Datenpakete einander in der Regel ersetzen: Gespeichert wird beispielsweise nur die letzte Position der Drohne, alte GPS-Koordinaten werden nach jedem Update verworfen.

Um Informationen in Echtzeit anzuzeigen, muss die Verarbeitungsdauer sehr kurz sein. Der Begriff „Echtzeit“ ist hier nicht hart an eine Spanne von Millisekunden gekoppelt. Da eine integrierte Systemlandschaft das Ziel ist, muss das System diese unterschiedlichen Anforderungen auf einer hybriden technischen Architektur erfüllen. Dafür hat sich das Konzept der sogenannten „Lambda-Architektur“ herauskristallisiert.

Lambda-Architektur als Lösung

Nathan Marz prägte den Begriff „Lambda-Architektur“ [Marz15] als hybriden Ansatz, um Echtzeit-Anwendungen und Batch-Auswertungen über große Datenmengen zu fahren. Die Datendrehscheibe

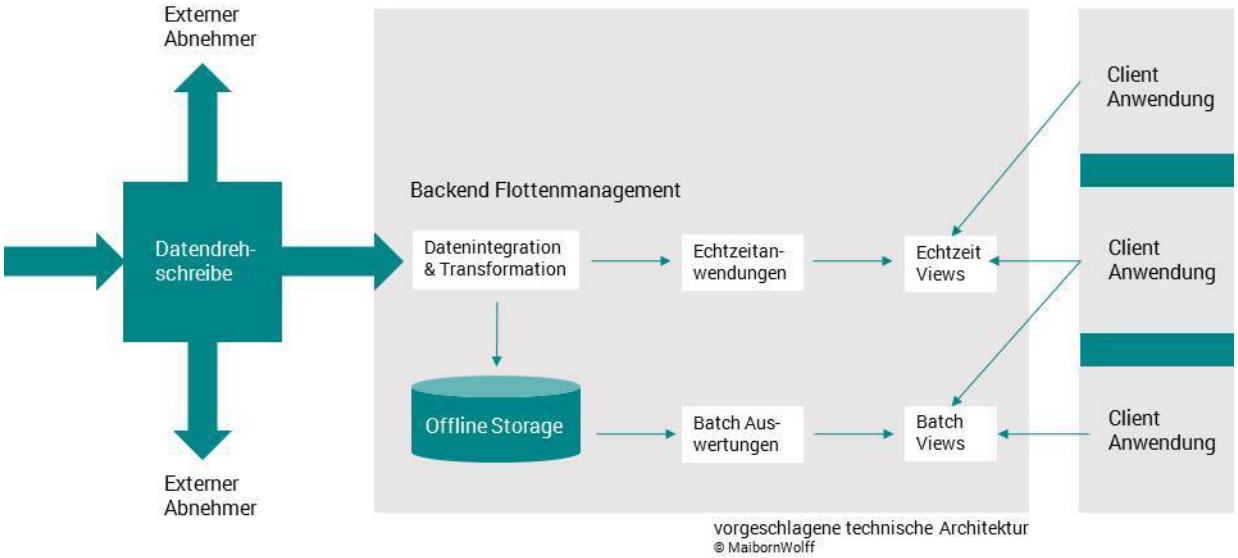

Abb. 3: Technische Architektur des Systems inklusive Datendrehscheibe

leitet Nachrichten an das Backend. Hier landen sie in einer Integrations- und Transformationskomponente, die fehlerhafte Daten für die dauerhafte Speicherung filtert. Außerdem verteilt sie benötigte Daten an andere Echtzeit-Anwendungen.

Jeder Datenfluss kann sehr einfach in einem Hadoop-basierten Offline-Storage abgelegt werden. Ein Framework, das diese Funktionalität anbietet, ist Spring XD [SpringXD]. Es unterstützt Technologien wie das eingangs beschriebene MQTT. Dadurch lassen sich leicht Streams aufsetzen, die die Daten von der Datendrehscheibe annehmen, filtern und abspeichern.

Eine mögliche Echtzeitanwendung in unserem Szenario ist ein Tool, das den eingehenden Nachrichtenstrom zu Paaren von IDs und Koordinaten kombiniert und einer Schnittstelle gesammelt anbietet. Diese Schnittstelle kann nun von einer Client-Anwendung mit Echtzeit-Anforderungen genutzt werden, um die Positionen der Flotte zu visualisieren. Ein solches Tool könnte beispielsweise als Processor Module im Spring XD Framework integriert werden.

Für große Auswertungen kann zusätzlich ein Hadoop-basierter Stack aufgebaut werden, mit einer HDFS-basierten Datenablage [HDFS] und einer Datenauswertung auf Basis von Apache Spark [Spark], die sich für das Scheduling und zur Verwaltung der Systemressourcen auf YARN [YARN] stützt.

Die batch-basierten Auswertungen bieten Ergebnisse in indizierten Views an, die wie oben beschrieben die Zugriffszeit auf Reports trotz großer Datenbasis gering hält. Um den Echtzeitstand der Informationen zwischen den Batch-Läufen genauer abzubilden, können Anwendungen auf Batch-View und Echtzeitschnittstelle zugreifen und Echtzeitdaten inkrementell zum letzten Batch-View hinzufügen.

Fazit

Die Lambda-Architektur hilft, Anforderungen aus Echtzeit-Datenverarbeitung und Batch-Auswertungen unter einen Hut zu bringen. Sie ist ein wichtiger Baustein einer belastbaren Big-Data-Architektur, die aus den enormen Datenmengen im Internet der Dinge sinnvolle Informationen destilliert.

Doch nicht nur die technische System-Architektur setzt saubere Planung voraus. Auch Übertragungsprotokolle und Datenmanagement für verschiedene Client-Systeme können zum limitierenden Faktor werden, an dem Management- und Analyse-Anwendungen im IoT mit vielen verschiedenen Daten scheitern können. ■

Literatur & Links

[GPP] https://developers.google.com/protocol-buffers/
[Hadoop] https://hadoop.apache.org/
[HDFS] Hadoop Distributed File System, http://wiki.apache.org/hadoop/HDFS
[HiveMQ] http://www.hivemq.com/
[IBMGraphic] http://www.ibmbigdatahub.com/infographic/four-vs-big-data
[Marz15] N. Marz mit J. Warren, Big Data – Principles and best practices of scalable realtime data systems, B&W, 2015
[MQTT] Message Queue Telemetry Transport, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
[RabbitMQ] https://www.rabbitmq.com/
[Spark] http://spark.apache.org/
[SpringXD] http://projects.spring.io/spring-xd/
[YARN] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Der Beitrag wurde ebenfalls in der Printausgabe von JavaSPEKTRUM 05/2015 veröffentlicht.