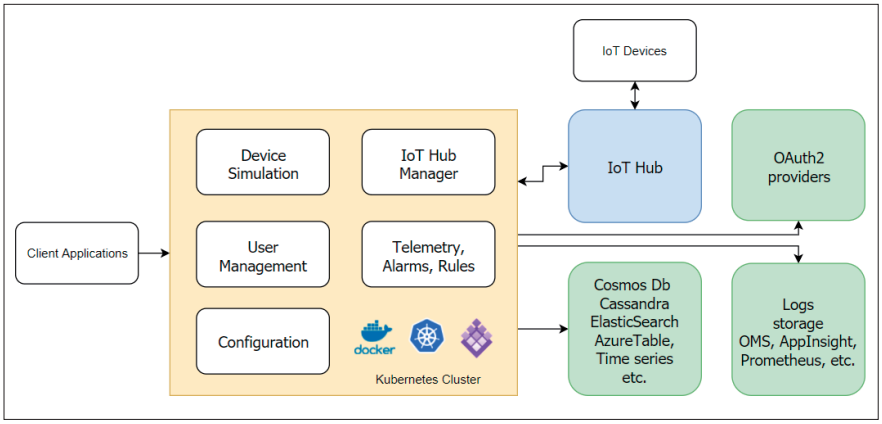
IOT-DATEN VERARBEITEN

Mehr als Azure IoT Hub

Funktionalitäten industrieller IoT-Lösungen schnell umsetzen und produktiv betreiben.

Beim Thema Internet of Things (IoT) stehen häufig die Geräte im Fokus, die ihre Daten in die Cloud senden. Wie diese Daten anschließend verarbeitet werden, tritt meist in den Hintergrund. Im Gegensatz dazu soll in diesem Artikel die Verarbeitung der Daten betrachtet werden, damit eine ganzheitliche Lösung entstehen kann.

IoT-Lösungen lassen sich mit vielen Cloud-Plattformen umsetzen. Die Dienste unterscheiden sich oft nur in geringem Maß. Das beschriebene Projekt wurde mit Komponenten der Plattform Microsoft Azure umgesetzt.



Beispiel-Implementierung der IoT-Architektur Remote Monitoring – eigene Darstellung, angelehnt an [2] (Bild 1)

Das Szenario

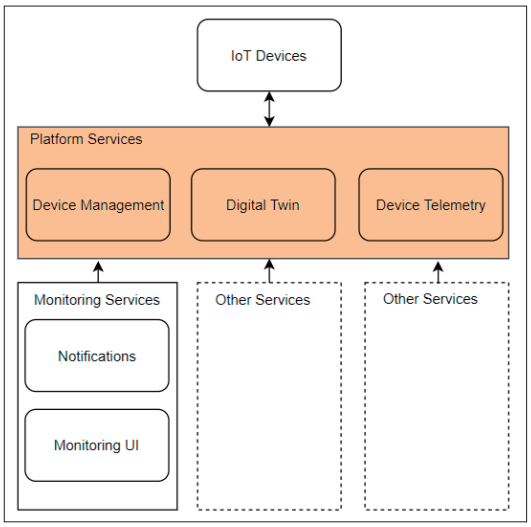
Im Unterschied zu den millionenfach gefertigten kleinen IoT-Geräten handelt es sich in diesem Szenario um Schienenfahrzeuge, welche mit der Cloud interagieren sollen. Abstrakt betrachtet sind dies sehr komplexe Endgeräte, die über fortschrittliche Steuerungs- und Wartungssoftware verfügen. Die Kommunikation mit den Systemen ist zwar ausgereift, erfordert jedoch für alle Eingriffe einen Techniker vor Ort.

In diesem Beitrag wird das „Gerät" an sich bewusst ausgeklammert und der Blick voll und ganz auf die Cloud-Seite gerichtet. Das setzt voraus, dass eine ausgereifte Sensorik vorhanden ist und somit Daten gesammelt werden können. Es wird davon ausgegangen, dass die Daten kontinuierlich an die Cloud übertragen werden – auch nach Offline-Unterbrechungen.

IoT-Architekturen auf Knopfdruck

Als Vorbereitung dienten die Architekturbeispiele von Microsoft. Wer wenig Erfahrung mit IoT-Architekturen hat, bekommt mit den vorkonfigurierten Lösungen der IoT Suite eine gute Basis für erste Prototypen und Anregungen für eigene Architektur-Ideen. Mit wenigen Aktionen lässt sich eine solche Lösung auf [1] in Betrieb nehmen. Ausgehend von den Anforderungen des vorgestellten Projekts lohnt sich ein Blick auf die Remote-Monitoring-Lösung. Sie enthält eine Fülle von Komponenten zum Verarbeiten von Daten in der Cloud. Erst im September 2017 wurde eine komplett überarbeitete Architektur dafür vorgestellt (Bild 1). Sie soll vor allem eine verbesserte Anpassbarkeit an die Bedürfnisse realer Projekte bieten.

Ein Blick auf die neue Architektur offenbart eine Vielzahl an Microservices, welche in Docker-Containern untergebracht sind. Auffällig ist auch, dass die meisten Azure-Komponenten durch beliebige andere Systeme austauschbar sind. Im Vergleich dazu konzentrierte sich das vorherige Modell darauf, eine größere Anzahl verschiedener Azure-Dienste intelligent miteinander zu kombinieren. Auch die im April 2017 vorgestellte Smart-Factory-Lösung baut auf eine vergleichbare containerbasierte Architektur.



Plattform und Anwendungen sind voneinander getrennt (Bild 2)

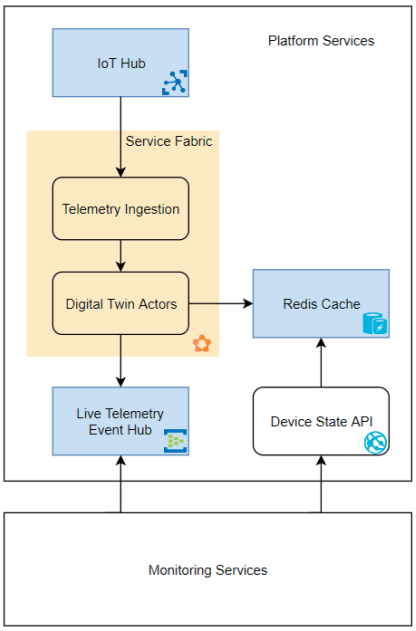
Die Zielarchitektur

Da die umgesetzte IoT-Lösung perspektivisch nicht nur auf Remote Monitoring beschränkt sein soll, wurden fachliche Komponenten in der Architektur logisch voneinander getrennt (Bild 2). Auf der einen Seite stellen die Platform Services die Basis der Lösung dar. Deren Hauptfunktionen sind die Verwaltung der Geräte sowie das Sammeln, Normalisieren und Bereitstellen der Gerätedaten. Auf der anderen Seite stehen die verschie-

denen Anwendungen, welche die Daten der Plattform nutzen, um Applikationen abzubilden, die wiederum von den Anwendern genutzt werden.

In Bild 3 wird der Fluss der Telemetrie-Daten innerhalb der Plattform-Dienste gezeigt. Der IoT Hub ist dabei der Anlaufpunkt in der Cloud. Die Telemetriedaten der Geräte werden von einem Microservice (Telemetry Ingestion) aus dem in den IoT Hub integrierten Event Hub ausgelesen. Dort werden die Daten gruppiert und zur gerätespezifischen Verarbeitung an Aktoren übergeben. Der Ingestion-Dienst lässt sich dabei anhand der Partitionen des Event Hubs gut skalieren, während Aktoren konzeptbedingt ebenfalls gut horizontal skalierbar sind.

Ein wichtiges Element der Plattform-Dienste ist der erweiterte Digital Twin des Geräts. Der ist in einem Service Fabric Actor abgebildet. Für komplexe IoT-Geräte taugt das Aktorenmodell hervorragend, sofern sich einzelne Logiken nur auf ein Gerät beziehen. Diese Abstraktion vereinfacht die Implementierung, wenn die Aktoren eigenständig mit weiteren Diensten kommunizieren sollen. Der Bedarf für den Aktor ergibt sich daraus, dass für den Hersteller die digitale Repräsentation des Gerätezustands eine wesentliche Rolle spielt und nicht mit simplen Aggregationen abgebildet werden kann. Der Device Twin des IoT Hubs bietet keine Möglichkeit, derartige Funktionen ähnlich elegant abzubilden. Der Aktor verarbeitet die Telemetrie-Daten und leitet einen Teil der Daten für die weitere Verarbeitung an einen Event Hub weiter. Zusätzlich kann der Aktor auch eigene Telemetrie-Daten erzeugen, um Statusveränderungen deutlich zu machen.



Plattform-Dienste für Digital Twin und Telemetrie, Ausschnitt (Bild 3)

Die digitale Repräsentation des Gerätezustands wird regelmäßig in einem Redis Cache persistiert. Redis Cache stellt dabei eine leichtgewichtige Alternative zu NoSQL-Datenbanken dar und trennt die schreibenden Aktoren von den lesenden Komponenten. Für die Anwendungen, welche auf die Gerätedaten zugreifen wollen, steht ein API bereit, welches den Zugriff auf den Redis Cache kontrolliert. Zusammen mit dem genannten Event Hub stehen damit die primären Datenquellen der weiteren IoT-Anwendungen zur Verfügung.

Monitoring Services

Die erste Anwendungsgruppe stellen die Monitoring Services dar. In einer Webanwendung werden die Daten der Geräte angezeigt. Dafür werden die Zustandsinformationen aus dem Plattform-API mit den Metadaten der Geräte kombiniert. Dazu gehören beispielsweise Darstellungsinformationen, welche baureihenabhängig unterschiedlich sind. Parallel hierzu werden die Live-Telemetrie-Daten über einen SignalR-Server an die Webanwendung geleitet, wodurch die Anzeige für den Benutzer permanent aktuell gehalten wird.

Für die Authentifizierung wird dienst- und applikationsübergreifend ein Azure Active Directory (AAD) genutzt. Eine Autorisierung eines Benutzers oder Dienstes im Kontext der Daten wird später hinzugefügt. Die implementierten Microservices verteilen sich damit auf zwei Azure-Dienste:

- App Service Web Apps
- Service Fabric

Für die Webanwendungen fiel die Wahl auf Web Apps, da hier bereits viele Funktionalitäten bereitstehen, welche den Einstieg und Betrieb deutlich vereinfachen. Diese in Service Fabric zu betreiben ist ebenso möglich und kann beispielsweise dann sinnvoll sein, wenn eine direkte Kommunikation mit den Diensten im Service Fabric Cluster gewünscht wird. Von einer direkten Kommunikation mit den Service Fabric Actors sei an dieser Stelle jedoch gewarnt, da diese konzeptbedingt zu jedem Zeitpunkt nur eine Anfrage bearbeiten. Viele lesende Anfragen und die wenigen schreibenden Anfragen würden im gezeigten Beispiel miteinander konkurrieren und die Funktionstüchtigkeit einschränken.

Microservices in Service Fabric

Service Fabric erlaubt einen sehr schnellen Einstieg in die Welt der Microservices, in welcher sich die Entwicklung auf die gewünschten Funktionalitäten fokussieren kann. In Service Fabric laufen die Anwendungen in einem sogenannten Cluster, welcher sich über eine variable Anzahl Nodes verteilt. Die Nodes stellen dabei einzelne virtuelle Maschinen dar. Die Erstellung und Wartung dieser Maschinen übernimmt Service Fabric. Auf jedem Node lässt sich eine Vielzahl von Diensten zugleich ausführen.

Zu den größten Herausforderungen von Microservices zählen das Überwachen und Ausrollen der Dienste sowie deren Kommunikation untereinander. Hierfür stellt Service Fabric einige Systemdienste bereit, welche sich beispielsweise darum kümmern, auf welchen Nodes die Dienste ausgerollt werden sollen und wie Ausfälle aufgefangen werden können. Ein Dienst zur Namensauflösung ermöglicht es den Diensten, sich gegenseitig unabhängig von der Verteilung auf die Nodes zu finden und miteinander zu kommunizieren. Aus Sicht der Anwendung lassen sich die meisten Konfigurationen über eine Handvoll XML-Dateien bewerkstelligen.

Ein weiterer Vorteil von Service Fabric ist die große Technologie-Unabhängigkeit. Sie können damit eigene Anwendungen als Prozesse betreiben. Zusätzlich lassen sich Container wie Docker einsetzen. Für das Host-System der Nodes des Clusters stehen Windows und Linux zur Wahl, womit Sie eine große Bandbreite unterschiedlicher Dienste in einer skalierbaren und überwachten Umgebung betreiben können. ▶

Werden Dienste von Grund auf neu entwickelt, kann es sich lohnen, speziell für Service Fabric angepasste Dienste zu schreiben. Für .NET Core, .NET Framework und Java stehen dafür spezialisierte Schnittstellen bereit. Mit diesen ist es ein Leichtes, auf weitere Funktionalitäten zuzugreifen. Die in der Architekturübersicht angesprochenen Aktoren sind ein gutes Beispiel hierfür.

Zusätzlich lassen sich mit den APIs zustandsabhängige Dienste (Stateful Services) abbilden. Der Zustand wird von Service Fabric unabhängig vom ausführenden Node abgebildet. Zustandsabhängige Dienste sollten in der Regel jedoch die Ausnahme bleiben. Soweit der Anwendungsfall es zulässt, sind zustandslose Dienste stets zu bevorzugen.

Die schnelle Einsatzfähigkeit durch bestehende APIs und die gute Entwicklerunterstützung durch lokale Emulatoren sind wesentliche Faktoren bei der Auswahl von Service Fabric. Die Unterstützung von Docker ist ein weiterer Pluspunkt für die zukünftige Entwicklung der Anwendungen.

Entkopplung der Microservices

Azure Event Hubs ist ein Dienst, der auf die Aufnahme und Bereitstellung großer Mengen an Ereignisdaten spezialisiert ist. Er eignet sich damit hervorragend für die von IoT-Geräten produzierten Telemetrie-Daten, aber beispielsweise auch für das Verarbeiten von Anwendungsprotokollen.

In der vorgestellten Architektur stellen Event Hubs ein wichtiges Element dar. Bereits im IoT Hub nimmt ein Event Hub die Daten der Geräte an, bevor diese weiterverarbeitet werden. Auch im weiteren Verlauf werden Event Hubs gezielt eingesetzt, da sie sich ideal eignen, um einzelne Dienste voneinander zu entkoppeln und die Telemetrie-Daten zu verteilen. Diese Entkopplung erfolgt dabei sowohl fachlich als auch technisch. Fachlich muss sich die schreibende Komponente nicht darum kümmern, wer die Daten als Nächstes verarbeitet. Technisch bleiben beide Seiten der Dienste von Ausfällen und Wartungen unbeeinträchtigt. Inhaltlich wird die Kommunikation über die Beschreibung der Elemente im Event Hub abgebildet und kann damit als Alternative zu üblichen APIs gesehen werden.

Sollen Daten an mehrere Dienste parallel verteilt werden, so müssen Sie dies nicht über komplexe Logiken abbilden: Für jeden zusätzlichen Abnehmer wird eine zusätzliche Consumer Group auf dem Event Hub erstellt. Die existierenden Komponenten müssen Sie nicht anpassen. Randfälle wie einen Teilausfall der lesenden Dienste lassen sich dadurch elegant lösen. Ist die Komponente wieder funktionstüchtig, kann sie auch die Nachrichten verarbeiten, welche zwischenzeitlich aufgelaufen sind.

Dass diese Entkopplung nicht nur in der Theorie funktioniert, zeigte sich bei den ersten Lasttests der Plattformdienste. Zunächst sendete ein simuliertes Gerät eine Vielzahl von Telemetrie-Daten. Das System verarbeitete diese ohne nennenswerte Latenzschwankungen. Als jedoch im nächsten Test-Schritt Hunderte Geräte gleichzeitig anfingen, Daten zu senden, wurde ein Flaschenhals im System offensichtlich. Beim erstmaligen Initialisieren eines Aktors benötigte die Implementierung der Aktoren vergleichsweise viel Zeit und CPU-Ressourcen. Aufgrund der Vielzahl gleichzeitig simulierter Geräte (und damit gestarteten Aktoren) wurde der Service Fabric Cluster komplett ausgelastet und die Verarbeitungsgeschwindigkeit brach ein. Da die lesende Komponente (Telemetry Ingestion aus Bild 3) jedoch keine Daten mehr abholte, solange die alten Daten nicht verarbeitet waren, stürzte das System nicht ab.

Die Service-Fabric-Komponenten erholten sich sukzessive und riefen wieder mehr Daten vom Event Hub ab. Nach einer gewissen Zeit hatte sich das System komplett erholt und erreichte wieder die gewünschten Latenzzeiten. Die schreibende Komponente wurde vom Flaschenhals nicht beeinträchtigt und keine Daten gingen verloren.

Die Qual der Wahl

Aufgrund der großen Fülle an Azure-Komponenten und Open-Source-Lösungen ergibt sich eine nahezu unendliche Anzahl an Möglichkeiten, einzelne Komponenten einer IoT-Architektur mit anderen Diensten abzubilden. Dies zeigt sich nicht zuletzt an den eingangs erwähnten Beispiel-Implementierungen von Microsoft, welche sich im direkten Vergleich in Bezug auf die eingesetzten Technologien radikal unterscheiden.

Für den Entwickler ergibt sich daraus die Herausforderung, für jede Anforderung aufs Neue zu fragen, wie diese am besten abgebildet werden kann. Ist Open Source das Mittel der Wahl? Der Service Fabric Cluster könnte durch einen Kubernetes Cluster ersetzt werden, welcher auf virtuellen Maschinen mit Docker-Images eine solide Lösung darstellen kann, nach Wunsch unterstützt durch den Azure Container Service. Für die Aktoren-Verwaltung stellt Akka beziehungsweise Akka.NET eine etablierte Bibliothek bereit [3]. Mit Apache Kafka [4] steht zudem eine Open-Source-Alternative für den Event Hub zur Verfügung. Ähnliches gilt für die anderen Dienste.

Auch im direkten Vergleich einzelner Azure-Komponenten stellt sich häufig die Frage, was am besten zu einem Anwendungsfall passt. Storage Queue, Event Hub, Service Bus Queue oder doch Event Grid? Microservice in Service Fabric, Serverless als Azure Function oder WebJobs als möglicher Mittelweg? Viele Dienste decken sehr spezifische Bedürfnisse ab. Auch die Kosten der verschiedenen Möglichkeiten lassen sich nicht immer leicht kalkulieren, sollten jedoch nie aus dem Blick geraten. Gleiches gilt für die Komplexität des Betriebs, die stark variieren kann.

Bei der Wahl der Komponenten sollten Sie zudem beachten, dass nicht nur Cloud-Ressourcen ihren Preis haben: Auch die Mitarbeiter besitzen Kenntnisse, welche die Wahl der Komponenten durchaus beeinflussen sollten. Erfahrungsgemäß ist es sinnvoll, neue Technologien und Konzepte schrittweise in das Team zu tragen. Das Team und die Architektur entwickeln sich damit gemeinsam weiter, während konstant Ergebnisse entstehen.

Das Team sollte sich hierfür an einige Grundregeln halten. Beim Implementieren der Geschäftslogiken sollte es beispielsweise stets darauf achten, sich nicht zu sehr an das ausführende System zu binden. Dies stärkt die Trennung der Ab-

hängigkeiten des Dienstes und lässt den Weg offen, zukünftig beispielsweise einen Docker-Container als Umgebung zu verwenden.

Wie geht es weiter?

In den Gesprächen mit dem Hersteller wurde früh klar, dass bei einer erfolgreichen Umsetzung des Remote Monitorings schnell weitere Schritte in die Welt des Internets der Dinge gemacht werden sollen.

Im Vordergrund stehen dabei Firmware-Updates für die Steuerungssoftware. Durch das Anbinden von Telemetriedaten an die Störungsmeldungen aus dem CRM wird parallel die Service-Erfahrung für die Kunden revolutioniert. Für den Hersteller ist es damit beispielsweise möglich, die Ursache der Störung unabhängig von Kundenbeschreibungen und ohne Vor-Ort-Einsätze nachzuvollziehen und Lösungsvorschläge vorzubereiten. Als Krönung ist das Ziel, eine Predictive-Maintenance-Lösung zu schaffen, welche Ausfälle durch vorausschauende Service-Einsätze reduzieren soll, womit der Nutzungsgrad der Schienenfahrzeuge beim Kunden erhöht wird.

Auch für die Kunden sollen die Daten bereitgestellt werden. Hierzu ist in einem ersten Schritt eine einfache Webanwendung geplant, welche die Schienenfahrzeuge mit ihrem aktuellen Status visualisiert. Dem Kunden ist es damit möglich, einen einfachen und schnellen Überblick über seine Fahrzeugflotte zu erhalten. Durch die zusätzliche Bereitstellung der Monitoring-Daten in Rohform sind viele weitere Anwendungsszenarien auf der Kundenseite denkbar.

Fazit

Die bereits sehr ausgereifte Hardware- und Kommunikationslandschaft bietet bei industriellen IoT-Projekten entscheidende Geschwindigkeitsvorteile bei der Umsetzung. Dank moderner Entwicklungsansätze und unterstützenden Cloud-Technologien lassen sich erste Funktionalitäten schnell realisieren und produktiv betreiben, anstatt nur technische Machbarkeiten zu beweisen. Sind die ersten Schritte gemacht, lassen sich mit Konzepten wie Predictive Maintenance bestehende Geschäftsfelder revolutionieren und neue Geschäftsfelder erschließen. ■

[1] Microsoft Azure IoT Suite, www.azureiotsuite.com
[2] Remote Monitoring, www.dotnetpro.de/SL1805IoT1
[3] Akka.NET, http://getakka.net
[4] Apache Kafka, https://kafka.apache.org

Alexander Bender
arbeitet als Consultant bei AIT als Softwareentwickler mit Schwerpunkt in der Backend-Entwicklung. Aktuell beschäftigt er sich mit IoT-Architekturen und Microsoft Azure.

Alexander.Bender@aitgmbh.de

dnpCode	A1805IoT