Seminararbeit

Evaluierung einer Amazon Web Services Lösung zur Erfassung und Verarbeitung von Sensordaten

An der Fachhochschule Dortmund
im Fachbereich Informatik
Studiengang SWT Dual
erstellte Seminararbeit

von
Hendrik Hagmans
geb. am 02.04.1992
Matr.-Nr. 7082973

Betreuer:
Prof. Dr. Martin Hirsch
Prof. Dr. Sabine Sachweh

Dortmund, 26. Januar 2016

Zusammenfassung

Amazon Web Services gibt es nun schon seit einigen Jahren und bietet verschiedenste Cloud Services für unterschiedlichste Anforderungen. Amazon Web Services unterscheidet sich von vielen anderen Anbietern vor allem durch die variable Leistungsabrechnung und die Vielfalt der Angebote. In dieser Arbeit soll nun mittels eines Beispiels evaluiert werden, ob Amazon Web Services auch für große Datenströme wie beispielsweise Sensordaten geeignet ist.

Abstract

Amazon Web Services are present for quite a few years and offer several services for different requirements. The most important differences between Amazon Web Services and other cloud computing providers is flexible service billing and the diversity of their offers. In this work it will be evaluated by means of an example if Amazon Web Services are suitable for use with big data streams like sensor data.

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation

Cloud Computing ist ein immer größer werdendes Thema in der IT. Immer mehr Daten werden nicht mehr lokal, sondern in der Cloud gespeichert und werden somit für jeden überall verfügbar. Auch Anwendungen werden immer häufiger nicht mehr lokal betrieben, sondern nutzen die großen Rechnerleistungen von Cloud Lösungen, um möglichst skalierbar zu sein und mittels Redundanz höhere Verfügbarkeiten und bessere Latenzzeiten zu erreichen. Zudem bieten Cloud Lösungen die Möglichkeit, Investitionskosten in Betriebskosten umzuwandeln, da keine neuen Server und andere Geräte gekauft werden müssen, sondern für die Nutzung bezahlt wird. Webapplikationen bilden hier keine Ausnahme. Anstatt eigene Rechenzentren aufbauen zu müssen, werden Webapplikationen immer häufiger bei externen Cloudanbietern betrieben um Kosten und den Administrationsaufwand solcher Systeme zu reduzieren. Ein großer Anbieter solcher Plattformen ist Amazon, die mit ihren Amazon Web Services [1] eine Reihe von Cloud Services bieten, die von vielen weltweit operierenden Unternehmen genutzt wird. Doch wie kann man mittels Amazon Web Services große Datenflüsse wie beispielsweise die Aufzeichnung von Sensordaten in einem automatisierten Heimsystem am besten verwalten?

1.2 Zielsetzung

Es soll eine Wettersimulation erstellt werden, die aus mehreren Services besteht, die dauerhaft Daten liefern. Beispielsweise mehrere virtuelle Temperatursensoren, die Zufallszahlen innerhalb eines bestimmten Werteintervalls liefern. Die Temperaturwerte

können sich auch gegenseitig beeinflussen. So beeinflusst eine höhere Außentemperatur auch die Innentemperatur. Diese Komponenten werden jeweils in einem Docker Container [2] auf Amazon ECS [3] deployed und liefern einen konstanten Datenstrom von Temperaturdaten. Der Datenstrom soll skaliert werden können und bspw. die Daten eines ganzen Tages in einer Stunde erzeugen. Die Daten werden in einer SQL Datenbank auf Amazon RDS [5] oder Amazon DynamoDB [6] gespeichert. Eine eigene Lösung mit Apache Cassandra [17] wäre auch möglich. Im Dashboard soll eine Übersicht der Daten in Verlaufsdiagrammen verfügbar sein. Hier ist auch eine Anpassung der Zeitskalierung möglich. Der Fokus liegt hierbei nicht auf einer akkuraten Wettersimulation, sondern auf der Evaluation der AWS Services für große Datenströme und der Vergleich zwischen den einzelnen AWS Services. Bspw. könnten Teile der Architektur auch mit Amazon Kinesis [7] umgesetzt werden, das speziell für große Datenströme konzipiert wurde. Hier könnte man beide Architekturentwürfe vergleichen. Dieses Projekt wurde in einer Projektarbeit bereits geplant und analysiert und soll in dieser Bachelorarbeit nun implementiert werden. Daraufhin soll eine Evaluation der verwendeten AWS Komponenten durchgeführt werden.

1.3 Vorgehensweise

- Wie wird vorgegangen, um das Ziel zu erreichen?
- Warum ist die Arbeit so gegliedert, wie sie gegliedert ist?
- Welche Aspekte werden nicht behandelt **und** warum?

2. Projektplanung

Das Projekt wurde bereits in einer vorher bearbeiteten Projektarbeit geplant. Dabei wurden mehrere Analysen durchgeführt. Als Ergebnis der Anforderungsanalyse wurde folgende Liste von Anforderungen erstellt:

Thema	Beschreibung	Kano Bewertung
Dauerhafter Datenstrom	Producer sollen dauerhaft Temperaturdaten liefern	Basismerkmal
Daten Persistenz	Die Daten sollen dauerhaft in einer Datenbank persistiert werden	Basismerkmal
Start und Stopp des Datenstroms	Der Datenstrom soll vom Nutzer gestartet und gestoppt werden können	Basismerkmal
Datenstrom Skalierung	Der Datenstrom soll skaliert werden können. Beispielsweise sollen die Daten eines Tages in einer Stunde ausgegeben werden können	Basismerkmal
Dashboard	Nutzer soll die aktuellen Daten in einem Dashboard einsehen können	Basismerkmal
Mehrere AWS Services	Die Applikation soll auf mehreren AWS Services deployed werden, um diese vergleichen zu können	Basismerkmal
Dashboard Darstellung	Die Daten werden im Dashboard in verschiedenen Diagrammen wie bspw. Verlaufsdiagrammen dargestellt sowie in Tabellen.	Leistungsmerkmal

Tabelle 2.1: Anforderungen an das Projekt Teil 1

Thema	Beschreibung	Kano Bewertung
Verfügbarkeit	Die Services sollen hochverfügbar sein	Leistungsmerkmal
Aktualität der Daten	Die Daten sollen im Dashboard aktuell gehalten werden, auch wenn die Seite nicht neu geladen wird	Begeisterungsmerkmal
Temperaturen beeinflussen sich gegenseitig	Die Temperaturen sollen sich gegenseitig beeinflussen, z.B. bedingt eine höhere Außentemperatur eine höhere Innentemperatur	Leistungsmerkmal

Tabelle 2.2: Anforderungen an das Projekt Teil 2

Die Anforderungsliste ist nach der Priorisierung sortiert, die im Rahmen einer Kano Bewertung der einzelnen Punkte erstellt wurde. Wie in den Tabellen 2.1 und 2.2 zu sehen, wurden die Basismerkmale am höchsten priorisiert, da diese den Erfolg des Projektes ausmachen. Auf der Basis dieser Anforderungsanalyse wurde zudem folgendes Pflichtenheft erstellt:

Thema	Beschreibung	Aufwand
AWS kennenlernen	Kennenlernen der AWS Services und erste Deployments von Containern	10
Docker Image	Docker Image mit allen benötigten Ressourcen erstellen	3
Docker Container	Docker Container aus dem Image mit der fertigen Applikation erzeugen	3
Producer	Es müssen mehrere Producer geschrieben werden, die konstant Temperaturdaten liefern. Die Producer werden in Java geschrieben	20
Consumer	Es muss mindestens ein Consumer geschrieben werden, der die Daten der Producer verarbeitet. Der Consumer wird in Java geschrieben	20

Tabelle 2.3: Pflichtenheft Teil 1

Thema	Beschreibung	Aufwand
Consumer DB	Es muss eine Datenbank entweder auf Amazon RDS oder Amazon DynamoDB (oder andere Lösungen bspw. mit Cassandra) eingerichtet werden	10
Consumer DB Zugriff	Der Consumer muss die Temperaturdaten in die DB schreiben können	10
Mehrere AWS Services	Die Applikation für mehrere AWS Services kompatibel machen und auf mehreren Services deployen	20
Dashboard	Es muss ein Dashboard geschrieben werden, das die Temperaturdaten anzeigen kann	15
Dashboard Start Stopp	Es muss im Dashboard die Funktion geben den Datenstrom anhalten oder wieder starten zu können	3
Dashboard Zeitskalierung	Es muss im Dashboard die Funktion geben den Datenstrom verschnellern oder verlangsamen zu können	7
Dashboard Diagramme	Die Daten sollten im Dashboard in Form von Diagrammen dargestellt werden	10
Dashboard Diagramme Aktualität	Die Daten sollten im Dashboard immer aktuell gehalten werden, auch wenn der Nutzer die Seite nicht aktualisiert	5
Producer Temperatur Beeinflussung	Die Temperaturwerte der Producer müssen sich gegenseitig beeinflussen. Die Producer müssen also untereinander kommunizieren und zumindest Wechsel in der Temperaturtendenz interessierten anderen Producern mitteilen.	10

Tabelle 2.4: Pflichtenheft Teil 2

Das Pflichtenheft ist genau wie die Anforderungsliste nach der Priorisierung sortiert, die im Rahmen einer Kano Bewertung der einzelnen Punkte erstellt wurde. Genauere

Erläuterungen der einzelnen Anforderungen und Punkte des Pflichtenhefts finden sich in der Projektarbeit.
Des weiteren wurde im Rahmen der Projektplanung eine Risikoanalyse durchgeführt, um die größten Risiken des Projekts zu erkennen und dementsprechende Gegenmaßnahmen anwenden zu können. Hierbei haben sich folgende 4 Risiken herausgestellt:

Nummer	Risiko	Eintrittswahrscheinlichkeit in %	geschätzer Schaden in €	Risikofaktor
1	AWS Zugriff zu spät bekommen	30	300	90
2	Producer erzeugt zu viele Daten und damit zu viele Kosten bei AWS	150	20	75
3	Teile des Projekts werden nicht rechtzeitig vor Abgabe erstellt	10	1000	100
4	Unterschätzen des Umfangs oder der Schwierigkeit des Projektes	10	1000	100

Tabelle 2.5: Risikoliste

Der Risikofaktor entspricht der Formel $\frac{Eintrittswahrscheinlichkeit \cdot Schaden}{100}$.Dementsprechend befinden sich gerade die Risiken Nummer 3 und 4 durchaus in einem gefährlichen Bereich, der das Projekt gefährden könnte. In der Projektarbeit wurden allerdings Maßnahmen zur Verhinderung des Eintretens der Risiken ermittelt, die in der Ausführung des Projektes auch umgesetzt wurden.
Zu guter Letzt wurde eine Kostenplanung erstellt, da die Nutzung von Amazon Web Services Kosten verursachen kann und diese nicht zu hoch ausfallen sollten. Es wurde ein Budget von bis zu 200 € gewährt, das nicht überschritten werden sollte. Daher mussten die aktuellen Kosten ständig überwacht werden.
Im Rahmen der Projektarbeit wurden zudem die Themen und Komponenten, die Teil der Bachelorarbeit sein sollten, behandelt und beschrieben. Darunter zählten die Themen Cloud Computing, Amazon Web Services und deren einzelne Komponenten sowie Docker. Daher müssen diese Themen in dieser Bachelorarbeit nicht mehr ausführlich behandelt werden.
Eine erste Evaluation der in Frage kommenden Amazon Web Services wurde ebenfalls durchgeführt und dabei eine Kombination aus Amazon EC2 als Infrastruktur für die Producer und Consumer, Amazon Kinesis als Übertragungskanal für die Temperatur-

daten sowie Amazon RDS oder Amazon DynamoDB für die Persistierung der Daten als passende Services ausgemacht.

3. Implementierung

3.1 Projektarchitektur

Um die Abhängigkeitsverwaltung und das Bauen des Projektes möglichst simpel zu gestalten, wurde für das Projekt Apache Maven [18] genutzt und das Projekt dementsprechend als Maven Projekt erstellt.
Abhängigkeiten des Projekts sind der Amazon Kinesis Client in der Version 1.6.1, der Amazon Kinesis Producer in der Version 0.10.2 sowie Eclipse Jetty Servlet [20] in der Version 9.2.14.v20151106.
Die Projektarchitektur ist im Grunde genommen genau so, wie sie von Amazon in der Dokumentation von AWS Kinesis vorgestellt wird.

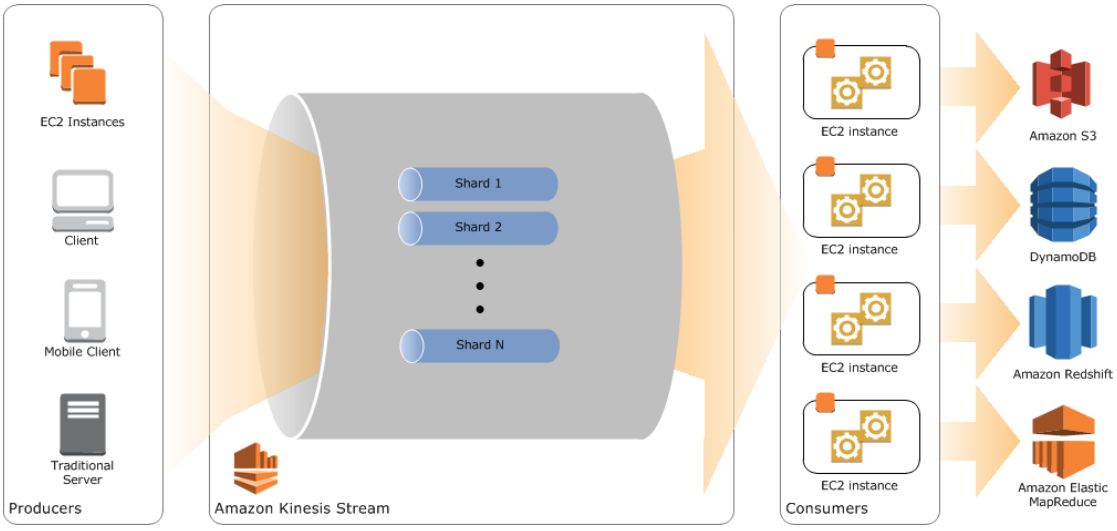

Abbildung 3.1: Kinesis Architektur, wie sie von Amazon vorgegeben wird. Quelle: [23]

Wie in Abbildung 3.1 zu sehen, setzt Amazon in der Architektur 4 Schichten voraus. Die Producer, den Kinesis Stream, die Consumer sowie weitere Services außerhalb von Kinesis. Die Daten werden von links nach rechts in der Architektur übertragen. Zunächst einmal werden die Daten in den Producern erzeugt und in den Kinesis Stream geschrieben, in denen sie in einem oder mehreren Shards einige Tage gespeichert bleiben. Ein Shard ist eine Gruppe von Datensätzen in einem Kinesis Stream, die eine feste Menge an Daten aufnehmen können.
Auf der anderen Seite des Kinesis Streams befinden sich ein oder mehrere Consumer, die die Daten aus den Shards des Kinesis Streams lesen. Nach dem Lesen können die Daten zudem an andere Services weitergeleitet werden, wie beispielsweise Amazon DynamoDB, mit dem die Daten in der NoSQL-Datenbank von DynamoDB persistiert werden können.
Genau diese Architektur wurde auch im Projekt umgesetzt. Es gibt eine oder mehrere Instanzen des Producers, der Temperaturdaten erzeugt. Der Producer schreibt die Daten in einen Kinesis Stream, meist nur mit einem Shard, da ein Shard für die Datenmengen dieses Projekts ausreicht. Zudem gibt es eine oder mehrere Instanzen eines Consumers, der die Daten aus dem Kinesis Stream liest und dann in eine DynamoDB Datenbank schreibt. Darüber hinaus enthält dieses Projekt zudem eine Webapplikation, die die Temperaturdaten aus DynamoDB liest und in Diagrammen ausgibt.
Zudem enthält das Projekt Utility-Klassen für DynamoDB, Kinesis sowie zur Temperaturgenerierung, in denen Methoden für die entsprechenden Anforderungsgebiete ausgelagert wurden. Darüber hinaus enthält das Projekt eine "DeleteResources"Klasse, die eine Methode zur Löschung aller verwendeten Ressourcen auf Amazon Webservices bereitstellt.
In der pom.xml des Projekts sind mehrere Profile eingetragen, die es ermöglichen, einzelne Klassen mit Startparametern auszuführen um somit beispielsweise den Producer mit anderen Parametern zu starten (siehe Kapitel 3.2).

3.2 Producer

3.3 Consumer

3.4 Webapplikation

3.5 Löschung der AWS Ressourcen

Eine weitere Klasse ist die Klasse “DeleteResources”, mittels der man die gestarteten Ressourcen auf Amazon Webservices wieder löschen kann, um keine weiteren Kosten zu verursachen.
Die Klasse hat eine main Methode, die zwei Argumente annimmt: Den Streamnamen sowie den Datenbank Namen der Dynamo DB Tabelle.

```
1       String streamName = TemperatureProducer.streamName;
2       String db_name = TemperatureConsumer.db_name;
3
4       if (args.length == 2) {
5           streamName = args[0];
6           db_name = args[1];
7       }
```

Quelltext 3.1: DeleteResources.java (24-30): Auslesen der übergebenen Argumente

In Quelltext 3.1 sieht man, wie die Argumente ausgelesen und gesetzt werden. Wenn nicht genau 2 Argumente übergeben werden, wird ein Standardwert für die beiden Variablen genutzt.

```
1       Region region = RegionUtils.getRegion(TemperatureProducer.REGION);
2       AWSCredentialsProvider credentialsProvider = new DefaultAWSCredentialsProviderChain();
3       AmazonDynamoDB amazonDynamoDB = new AmazonDynamoDBClient(
4               credentialsProvider, new ClientConfiguration());
5       AmazonDynamoDBClient client = new AmazonDynamoDBClient();
6       client.setRegion(region);
7       DynamoDB dynamoDB = new DynamoDB(client);
8       amazonDynamoDB.setRegion(region);
9       DynamoDBUtils dbUtils = new DynamoDBUtils(dynamoDB, amazonDynamoDB,
10              client);
11      dbUtils.deleteTable(db_name);
12      dbUtils.deleteTable(TemperatureConsumer.tableName);
```

Quelltext 3.2: DeleteResources.java (32-43): Initialisierung der Dynamo DB Utilklasse und Löschen der Tabellen

In Quelltext 3.2 wird die DynamoDB Utilklasse initalisiert und dazu werden zunächst die benötigten Amazon Client Klassen erzeugt, die der Utilklasse bei der Initialisierung übergeben werden. Daraufhin wird die Übersichtstabelle sowie die Tabelle, die die Temperaturdaten enthält, gelöscht.

```
1        AmazonKinesis kinesis = new AmazonKinesisClient(credentialsProvider,
2                new ClientConfiguration());
3        kinesis.setRegion(region);
4        StreamUtils streamUtils = new StreamUtils(kinesis);
5        streamUtils.deleteStream(streamName);
```

Quelltext 3.3: DeleteResources.java: Initialisierung der Stream Utilklasse und Löschen des Streams (45-49)

Im nächsten Abschnitt in Quelltext 3.3 wird dann die Stream Utilklasse initialisiert und daraufhin der Stream gelöscht.
Damit sind alle Ressourcen, die auf Amazon Web Services genutzt wurden, gelöscht und es werden keine Kosten mehr zum Beispiel durch die temporäre Speicherung der Daten im Kinesis Stream verursacht.

3.6 Docker

Um die Applikation auf ein Deployment auf Amazon EC2 vorzubereiten, wurde ein Docker Image erstellt, um die Applikation letztendlich über ECS auf EC2 zu deployen.

```
1 FROM maven:3.2-jdk-7-onbuild
2 CMD ["mvn clean package"]
```

Quelltext 3.4: Dockerfile des Projekts

Dieses Dockerfile ist sehr simpel gehalten und enthält nur 2 Zeilen. Wie in Quelltext 3.4 zu sehen, basiert dieses Image auf dem Apache Maven Image [19]. Dieses Image enthält neben einer Java 7 Installation auch eine Maven 3.2 Installation. Zudem kopiert

es bei der Generierung des Images das Projekt, wenn die Dockerfile im gleichen Ordner wie die pom.xml ist, nach “/usr/src/app” und führt “mvn install” aus. Zusätzlich dazu wird, wie in Zeile 2 beschrieben, noch “mvn clean package” ausgeführt, was die Applikation endgültig initialisiert.
Mithilfe dieses Dockerfiles kann nun mit “docker build -t project-image .” ein Docker Image erzeugt werden. Über dieses Image kann dann mit dem Befehl in Quelltext 3.5 der Producer gestartet werden.

```
1 docker run -it --name project-image project-container bash -c 'MAVEN_OPTS="-Daws.
      accessKeyId=ACCESSKEY -Daws.secretKey=SECRETKEY -Dstream.name=StreamTest -
      Dsensor.name=SensorTest -Drun.seconds=10 -Drecord.second=1'' mvn compile -Pproducer
      exec:java'
```

Quelltext 3.5: Konsolenbefehl zum Starten des Docker Images

Auch hier müssen wieder der AWS Access Key sowie der AWS Secret Key übergeben werden sowie die weiteren Parameter, die der Producer annimmt. In diesem Beispiel wird ein Stream namens “StreamTest” erzeugt, der Sensor Name ist “SensorTest” und der Producer erzeugt für 10 Sekunden 10 Records pro Sekunde.
Es ist natürlich auch möglich die anderen Klassen mithilfe dieses Images zu starten, indem die Übergabeparameter geändert werden.

3.7 Abweichungen im Vergleich zur Projektplanung

4. AWS IoT

4.1 Einführung

AWS IoT [9] ist einer der neueren Web Services von Amazon, welcher zunächst im Oktober 2015 in eine Beta Phase gestartet ist und seit Januar 2016 in den regulären Betrieb gewechselt ist. Es ist eine verwaltete Cloud Plattform, mit der verbundene Geräte mit Cloud Anwendungen und anderen Geräten zusammenarbeiten können.
AWS IoT kann Millionen von Nachrichten von typischen IoT Geräten wie bspw. Sensoren annehmen und an weitere Amazon Web Services oder andere IoT Geräte verteilen. Dies bietet zum einen die Möglichkeit der Kommunikation von IoT Geräten untereinander sowie die Möglichkeit der IoT Geräte mit der Cloud zu kommunizieren um beispielsweise Daten zu übermitteln, die dann in Cloud Services verarbeitet und/oder persistiert werden. Beispiele wären zum Beispiel ein Temperatursensor, der mittels AWS IoT der Klimaanlage des Hauses die Nachricht übermittelt, dass die Temperatur höher geregelt werden muss oder der Temperatursensor schickt seine Temperaturdaten an andere Amazon Web Services wie bspw. Amazon DynamoDB, welches die Daten dann persistieren kann. Amazon nennt als Beispiel eine Reihe von Temperatursensoren, die ihre Daten an AWS IoT senden. AWS IoT sendet dann bei Überschreiten eines bestimmten Grenzwertes ein Kommando an einen Ventilator im Haus, der sich daraufhin einschaltet. Der Ventilator kann aber auch bspw. über eine Mobilapplikation von einem Benutzer über AWS IoT manuell gestartet werden (s. Abbildung 4.1). Dementsprechend ist AWS IoT eine gute Alternative, sollte man dieses Projekt mit echten Temperatursensoren ausführen wollen.

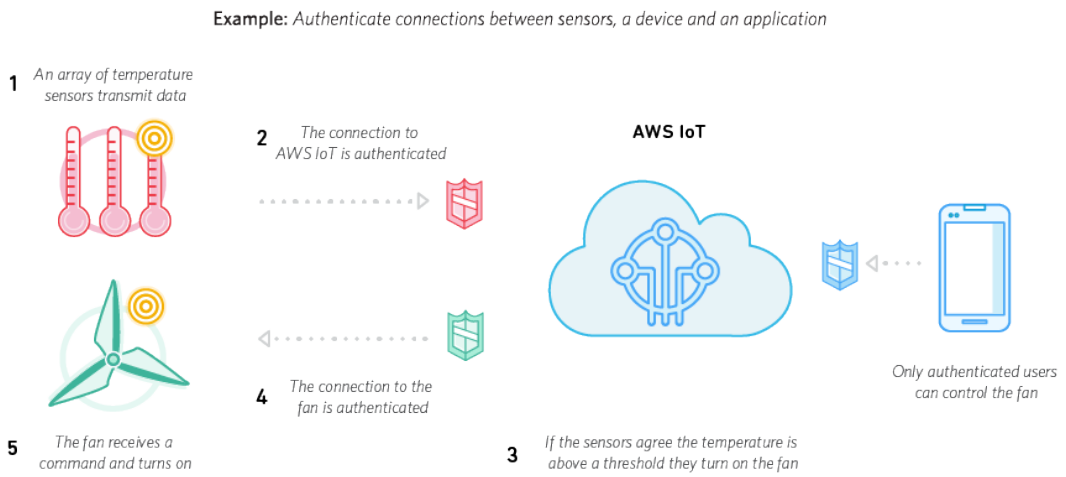

Abbildung 4.1: Kommunikation und Authentifikation von Temperatursensoren und einem Ventilator mit AWS IoT. Quelle: [9]

4.2 Funktionsweise

Ein wichtiger Punkt bei AWS IoT ist die Sicherheit der Kommunikation zwischen den einzelnen Endpunkten. Sicherheit ist bei IoT generell ein großes Thema, da unter Umständen mit sensiblen Daten gearbeitet wird, die im eigenen Haus erzeugt werden. AWS IoT stellt sicher, dass keine Kommunikation über AWS IoT unverschlüsselt stattfindet, indem sich jeder Endpunkt zunächst bei AWS IoT authentifizieren muss. Jede Kommunikation wird einzeln authentifiziert und verschlüsselt. Dies sieht man z.B. in Abbildung 4.1. Hier wird in Schritt 2 die Kommunikation der Temperatursensoren mit AWS IoT authentifiziert, bevor sie die Daten an AWS IoT senden. In Schritt 4 wird die Kommunikation mit dem Ventilator ebenfalls zunächst authentifiziert. Und natürlich können nur authentifizierte Nutzer über Mobilapplikationen über AWS IoT wie in diesem Beispiel den Ventilator kontrollieren.

Abbildung 4.2 zeigt eine Funktionsübersicht von AWS IoT. Hier steht zunächst einmal auf der linken Seite die AWS Device SDK, die eine Reihe von Client Libraries bereitstellt, mit der Geräte mit AWS IoT kommunizieren können. Die Kommunikation findet dabei über die Protokolle MQTT [15] oder HTTP statt. MQTT ist ein leichtgewichtiges Nachrichtenprotokoll für die direkte Kommunikation zwischen Maschinen (M2M) und ist daher ein häufig genutztes Protokoll im Bereich des IoT. Mittels einem dieser beiden

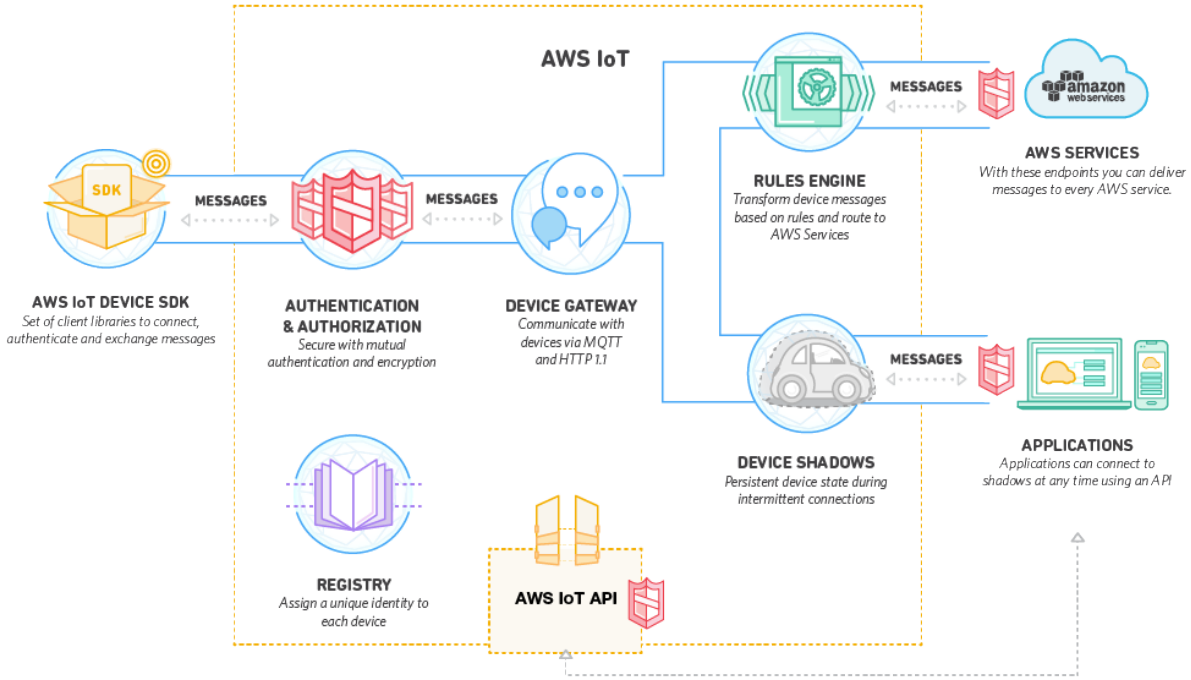

Abbildung 4.2: Funktionsübersicht von AWS IoT. Quelle: [10]

Protokolle kommuniziert das Gerät, das die AWS Service SDK nutzt, mit dem Device Gateway. Die Kommunikation wird dabei verschlüsselt und zunächst müssen sich beide Seiten auch authentifizieren, bevor überhaupt eine Kommunikation zustande kommt. AWS IoT unterstützt dabei die AWS-Methode der Authentifizierung (mit der Bezeichnung “SigV4”) sowie eine Authentifizierung auf der Basis von X.509 [10]. Den Geräten können einzelne Richtlinien vorgegeben werden, die ihren Zugriff entsprechend einschränken oder es können Rollen definiert werden, die Geräten eine genau spezifizierte Menge an Rechten geben.
Die Registry erstellt eine eindeutige Identität für die Geräte. Diese ist für alle Geräte egal welcher Art einheitlich formatiert. Außerdem werden Metadaten von Geräten gespeichert, die bspw. angeben, welche Funktionen dieses Gerät unterstützt wie zum Beispiel, dass ein Sensor Temperaturdaten meldet und in welcher Einheit diese Temperaturdaten übermittelt werden.
Das Device Gateway stellt für Applikationen REST APIs bereit, über die die Applikationen die Statusinformationen von Geräten auslesen und manipulieren können. Dazu legt das Device Gateway sogenannte “Device Shadows” bzw. “Schattengeräte” an, die

den letzten Zustand des Gerätes darstellen. Das heißt, dass der Zustand von Geräten auch dann ausgelesen werden kann, wenn das Gerät gar nicht mehr online ist, da der letzte Zustand als “Schattengerät” gespeichert wurde. Zudem können Applikationen so auch den gewünschten zukünftigen Zustand festlegen. Auch dieser wird im “Schattengeräte” persistiert und wird auf das Gerät übertragen, wenn es wieder online ist. Damit ist es möglich, auch für Geräte, die nicht dauerhaft online sind, eine REST API bereitzustellen, die dauerhaft verfügbar ist und Applikationen ermöglicht, sich immer mit dem “Schattengerät” zu synchronisieren und den Status zu verändern. Die Kommunikation erfolgt auch hier wieder über verschlüsselte Nachrichten.
Das Device Gateway kann aber auch über die Rules Engine Daten von Geräten transformieren und an andere Services weiterleiten. So könnten beispielsweise Temperaturdaten an Amazon DynamoDB oder andere Datenbank Services weitergeleitet werden. Es können aber auch Regeln erstellt werden, die bei bestimmten Konditionen Aktionen auslösen. Zum Beispiel könnte beim Überschreiten einer bestimmten Menge an Temperaturen eine Nachricht an AWS Lambda [8] geschickt werden, das daraufhin den Mittelwert der gesammelten Daten berechnet. Nachrichten können aber auch an Geräte geschickt werden wie in Abbildung 4.1. Erreicht die Temperatur der Sensoren einen Grenzwert, greift eine Regel und sendet den Befehl an den Ventilator, der sich daraufhin einschaltet.
Die Regeln werden direkt in der Management Konsole eingetragen oder über eine SQL-ähnliche Syntax definiert.
AWS IoT bietet also passende Funktionen, um das Projekt mit echten Temperatursensoren umzusetzen. Man könnte die Temperaturdaten direkt an einen Datenbank Service wie Amazon DynamoDB weiterleiten und auch die Temperatursensoren über die REST API direkt steuern. Mittels der Rules Engine könnten Regeln festgelegt werden, nach denen bei bestimmten Temperaturen bestimmte Aktionen ausgeführt werden können. Dadurch könnte man auch andere Geräte wie beispielsweise Ventilatoren oder Klimaanlagen einbinden und mittels AWS IoT steuern lassen. Man benötigt aber neben den Sensoren auch Mikrocontroller wie bspw. Raspberry PIs [16], die mit den Sensoren und anderen Geräten kommunizieren können, um AWS IoT nutzen zu können.

4.3 AWS IoT CLI, SDK und APIs

Die AWS IoT CLI bietet die Möglichkeit über Kommandozeileneingaben verschiedene Funktionen von AWS IoT zu nutzen.. Beispielsweise können Geräte zur Device Registry hinzugefügt werden und Einträge in der Registry verändert werden.

```
1 aws iot describe-thing --thing-name "MyDevice3"
```

Quelltext 4.1: Einfügen eines Geräts in die Registry. Quelle: [11]

Der Quellcode 4.1 zeigt den Befehl, mit dem ein Gerät mit dem Namen “MyDevice3” in die Registry eingefügt werden kann.

```
1 {
2     "thingName": "␣MyDevice3",
3     "defaultClientId": "MyDevice3",
4     "attributes": {
5         "Manufacturer": "Amazon",
6         "Type": "IoT␣Device␣A",
7         "Serial␣Number": "10293847562912"
8     }
9 }
```

Quelltext 4.2: Geräteeintrag in der Registry. Quelle: [11]

Der Eintrag in der Registry sieht dann wie in Quelltext 4.2 aus. Weitere Attribute wie eine einzigartige Seriennummer werden automatisch generiert.
Die AWS IoT CLI bietet zudem auch die Möglichkeit alle Geräte in der Registry anzuzeigen oder die Suchergebnisse zu filtern.
Mittels SDK lassen sich auch Regeln erstellen, einsehen und bearbeiten.

```
1 aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://my-rule.json
```

Quelltext 4.3: Erstellen einer AWS IoT Regel. Quelle: [12]

In Quelltext 4.3 wird eine Regel mit dem Namen “my-rule” erzeugt. Zudem wird eine payload Datei namens “my-rule.json” ebenfalls mitgeschickt.

```
1 {
2   "sql": "SELECT␣*␣FROM␣'iot/test'",
3   "ruleDisabled": false,
```

```
4    "actions": [{
5        "dynamoDB": {
6            "tableName": "my-dynamodb-table",
7            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
8            "hashKeyField": "topic",
9            "hashKeyValue": "${topic(2)}",
10           "rangeKeyField": "timestamp",
11           "rangeKeyValue": "${timestamp()}"
12       }
13   }]
14 }
```

Quelltext 4.4: Inhalt einer Payload Datei. Quelle: [12]

Die Payload Datei in Quelltext 4.4 enthält die Regel, alle Nachrichten, die an die Topic “iot-topic” gesendet werden, in die DynamoDB Tabelle “my-dynamodb-table” zu schreiben. Dabei wird noch eine Rolle übergeben, die in diesem Fall das Recht hat, auf die DynamoDB Tabelle zuzugreifen und in diese zu schreiben. Als HashkeyValue wird der Name der Topic übergeben und als rangeKeyValue der Timestamp der Nachricht.

```
1 HTTP GET https://endpoint/things/thingName/shadow
```

Quelltext 4.5: GET Anfrage an einen “Geräteschatten”. Quelle: [13]

Mittels einer REST API können die “Geräteschatten” von Geräten abgefragt und auch manipuliert werden. Quelltext 4.5 zeigt eine HTTP GET Anfrage an einen Geräteschatten, wobei “endpoint” und “thingName” Variablen sind, die man in der AWS IoT Konsole auslesen kann oder auch per AWS IoT SDK.

```
1 HTTP 200
2 BODY: response state document
```

Quelltext 4.6: GET Antwort auf die Anfrage an einen “Geräteschatten”. Quelle: [13]

Quelltext 4.6 zeigt die Antwort auf eine Anfrage an einen “Geräteschatten”. “response state document” ist hierbei ein größeres Dokument, das Informationen über den Status des “Geräteschattens” sowie Metadaten, einen Timestamp und andere Informationen enthält.
Über die API ist es auch möglich, “Geräteschatten” zu bearbeiten und zum Beispiel einen neuen Status zu setzen, der dann im Gerät gesetzt wird, wenn es online ist oder auch “Geräteschatten” zu löschen.

Es gibt außerdem noch eine AWS IoT SDK [14], die die Integration von Geräten in AWS IoT vereinfacht. Die SDK ist für C, Arduino Yún [22] und Node.js [21] verfügbar.
Sie bietet Funktionen zum Anmelden der Geräte bei AWS IoT, dem Aufbauen von Verbindungen und Senden von Nachrichten über die Verbindung sowie zum Bearbeiten von “Geräteschatten”.

5. Evaluation

5.1 Evaluation Kinesis

5.2 Evaluation IoT

5.3 Evaluation anderer Implementierungsmöglichkeiten

6. Fazit

Li Europan lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc, litot Europa usa li sam vocabular. Li lingues differe solmen in li grammatica, li pronunciation e li plu commun vocabules. Omnicos directe al desirabilite de un nov lingua franca: On refusa continuar payar custosi traductores. At solmen va esser necessi far uniform grammatica, pronunciation e plu sommun paroles. Ma quande lingues coalesce, li grammatica del resultant lingue es plu simplic e regulari quam ti del coalescent lingues. Li nov lingua franca va esser plu simplic e regulari quam li existent Europan lingues. It va esser tam simplic quam Occidental in fact, it va esser Occidental. A un Angleso it va semblar un simplificat Angles, quam un skeptic Cambridge amico dit me que Occidental es. Li Europan lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc, litot Europa usa li sam vocabular. Li lingues

Abkürzungsverzeichnis

ACL Access Control Lists
AES Advanced Encryption Standard

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Literaturverzeichnis

[1] Amazon Inc, “Amazon Web Services”; http://aws.amazon.com/de/, abgerufen am 28. Oktober 2015

[2] Docker Inc, “Docker”; https://www.docker.com/, abgerufen am 28. Oktober 2015

[3] Amazon Inc, “Amazon ECS”; http://aws.amazon.com/de/ecs/, abgerufen am 28. Oktober 2015

[4] Amazon Inc, “Amazon EC2”; http://aws.amazon.com/de/ec2/, abgerufen am 28. Oktober 2015

[5] Amazon Inc, “Amazon RDS”; http://aws.amazon.com/de/rds/, abgerufen am 28. Oktober 2015

[6] Amazon Inc, “Amazon Dynamo DB”; http://aws.amazon.com/de/dynamodb/, abgerufen am 28. Oktober 2015

[7] Amazon Inc, “Amazon Kinesis”; http://aws.amazon.com/de/kinesis/, abgerufen am 28. Oktober 2015

[8] Amazon Inc, “AWS Lambda”; http://aws.amazon.com/de/lambda/, abgerufen am 28. Oktober 2015

[9] Amazon Inc, “AWS IoT”; http://aws.amazon.com/de/iot/, abgerufen am 28. Oktober 2015

[10] Amazon Inc, “AWS IoT Funktionsweise”; https://aws.amazon.com/de/iot/how-it-works/, abgerufen am 28. Oktober 2015

[11] Amazon Inc, “AWS IoT Thing Registry”; http://docs.aws.amazon.com/iot/latest/developerguide/thing-registry.html, abgerufen am 28. Oktober 2015

[12] Amazon Inc, “AWS IoT Creating an AWS IoT Rule”; http://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-create-rule.html, abgerufen am 28. Oktober 2015

[13] Amazon Inc, “AWS IoT GetThingShadow”; http://docs.aws.amazon.com/iot/latest/developerguide/API_GetThingShadow.html, abgerufen am 28. Oktober 2015

[14] Amazon Inc, “AWS IoT SDK”; http://docs.aws.amazon.com/iot/latest/developerguide/iot-sdks.html, abgerufen am 28. Oktober 2015

[15] MQTT Org, “MQTT”; http://mqtt.org/, abgerufen am 28. Oktober 2015

[16] Raspberry Pi Foundation, “Raspberry PI”; https://www.raspberrypi.org/, abgerufen am 28. Oktober 2015

[17] Apache Software Foundation, “Apache Cassandra”; http://cassandra.apache.org/, abgerufen am 28. Oktober 2015

[18] Apache Software Foundation, “Apache Maven”; https://maven.apache.org/, abgerufen am 28. Oktober 2015

[19] Apache Software Foundation, “Apache Maven Image”; https://hub.docker.com/_/maven/, abgerufen am 28. Oktober 2015

[20] Eclipse Foundation, “Eclipse Jetty”; http://www.eclipse.org/jetty/, abgerufen am 28. Oktober 2015

[21] Node.js Foundation, “Node.js”; https://nodejs.org/en/, abgerufen am 28. Oktober 2015

[22] Arduino Foundation, “Arduino YÃºn”; https://www.arduino.cc/en/Main/ArduinoBoardYun, abgerufen am 28. Oktober 2015

[23] Amazon Inc, “Amazon Kinesis Key Concepts”; http://docs.aws.amazon.com/kinesis/latest/dev/key-concepts.html, abgerufen am 28. Oktober 2015