Grundlagen und Anwendungsbeispiele für das NoSQL-Datenbankmanagementsystem DynamoDB

Seminararbeit

vorgelegt am 25. Januar 2017

Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Kurs WWI2014I

von

|  |  |
| --- | --- |
| Betreuer in der Ausbildungsstätte: | DHBW Stuttgart: |
| Prof. Dr. Nikolai Preiß DHBW Stuttgart | [Titel, Vorname und Nachname des wissenschaftl. Betreuers/Prüfers] |

Kerstin Farke  
Dhruv Mahandru  
Henning Mohr  
Käthe Vrettos

**Inhaltsverzeichnis**

[Abkürzungsverzeichnis (bei Bedarf) IV](#_Toc470864049)

[Abbildungsverzeichnis (bei Bedarf) V](#_Toc470864050)

[Tabellenverzeichnis (bei Bedarf) VI](#_Toc470864051)

[1 Einleitung 1](#_Toc470864052)

[2 Theoretische Grundlagen 2](#_Toc470864053)

[2.1 Definitionen 2](#_Toc470864054)

[2.1.1 Datenbank 2](#_Toc470864055)

[2.1.2 Datenbankmanagementsystem 2](#_Toc470864056)

[2.1.3 Software-as-a-Service 2](#_Toc470864057)

[2.1.4 Database-as-a-Service 3](#_Toc470864058)

[2.2 Datenbank Prinzipien 3](#_Toc470864059)

[2.2.1 ACID 3](#_Toc470864060)

[2.2.2 BASE 4](#_Toc470864061)

[2.2.3 CAP-Theorem 4](#_Toc470864062)

[2.3 Entwicklungen in der Datenbankbranche 5](#_Toc470864063)

[2.4 Datenbankmodelle 6](#_Toc470864064)

[2.4.1 Relationale Datenbanken 6](#_Toc470864065)

[2.4.2 NoSQL Datenbanken 7](#_Toc470864066)

[2.4.3 Zusammenfassung 9](#_Toc470864067)

[3 DynamoDB 11](#_Toc470864068)

[3.1 Generelle Informationen über AWS 11](#_Toc470864069)

[3.1.1 AWS als Infrastructre as a Serivice 11](#_Toc470864070)

[3.1.2 Gründe für die Verwendung von AWS 12](#_Toc470864071)

[3.1.3 AWS Anwendungsbeispiele 13](#_Toc470864072)

[3.1.4 Vor- und Nachteile von AWS 14](#_Toc470864073)

[3.2 DynamoDB 15](#_Toc470864074)

[3.2.1 Was ist DynamoDB 15](#_Toc470864075)

[3.2.2 Hintergrund der Entwicklung von DynamoDB 16](#_Toc470864076)

[3.2.3 Preismodell 16](#_Toc470864077)

[3.2.4 Architektur 16](#_Toc470864078)

[3.3 Art der Datenverwaltung/Datenmodell 19](#_Toc470864079)

[3.3.1 Tabelle, Elemente und Attribute 19](#_Toc470864080)

[3.3.2 Primärschlüssel 20](#_Toc470864081)

[3.3.3 Sekundäre Indizes 21](#_Toc470864082)

[3.3.4 Besonderheiten der DynamoDB 21](#_Toc470864083)

[3.3.5 Key-Value Eigenschaft 22](#_Toc470864084)

[3.3.6 Datentypen 23](#_Toc470864085)

[3.4 Vergleich mit anderes NoSQL-Lösungen 24](#_Toc470864086)

[3.4.1 DynamoDB vs. Redis 24](#_Toc470864087)

[3.4.2 Vorteile und Nachteile von DynamoDB 26](#_Toc470864088)

[4 Praxis 27](#_Toc470864089)

[4.1 Insgesamt zum Zurechtfinden 27](#_Toc470864090)

[4.2 Testdatenbank mit Testdaten 27](#_Toc470864091)

[4.3 Datenbanksprache 27](#_Toc470864092)

[4.4 GUI/API 27](#_Toc470864093)

[4.5 Ablageform der Daten 27](#_Toc470864094)

[5 Schluss 28](#_Toc470864095)

[5.1 Fazit 28](#_Toc470864096)

[5.2 Ausblick 28](#_Toc470864097)

[Anhang (bei Bedarf) 29](#_Toc470864098)

[Quellenverzeichnis 30](#_Toc470864099)

[Ehrenwörtliche Erklärung 31](#_Toc470864100)

[Literaturverzeichnis 32](#_Toc470864101)

# Abkürzungsverzeichnis (bei Bedarf)

AktG = Aktiengesetz

AWS = Amazon Web Services

BFH = Bundesfinanzhof

# Abbildungsverzeichnis (bei Bedarf)

[Abb. 1: Eine Abbildung 2](#_Toc444006772)

# Tabellenverzeichnis (bei Bedarf)

[Tabelle 1: Eine Beispieltabelle 2](#_Toc444006773)

1. Einleitung
2. Theoretische Grundlagen
   1. Definitionen
      1. Datenbank

Nach Definition von Schicker in seinem Buch „Datenbanken und SQL: Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL“ ist eine Datenbank „eine Sammlung von Daten, die untereinander in einer logischen Beziehung stehen und von einem eigenen Datenverwaltungssystem (Database Management System, DMBS) verwaltet werden“.[[1]](#footnote-2) Demnach stellt eine Datenbank eine organisierte Datenkollektion dar, welche den Zugriff zu Inhalten einfacher gestaltet.[[2]](#footnote-3)

(aus meiner PA kopiert 🡪 muss noch bearbeitet werden!)

* + 1. Datenbankmanagementsystem

Nach Leimeister besteht „[j]edes Datenbanksystem [..] in seiner Grundstruktur aus einem Datenbankmanagementsystem und mehreren, untereinander verknüpften Daten, der eigentlichen Datenbank“[[3]](#footnote-4). Das DBMS beinhaltet verschiedene Funktionen, unter anderem den Auf-bau, die Verwaltung und die Manipulation der Daten einer Datenbank, sowie die Steuerung und Kontrolle aller peripheren Geräte, das Belegen von Speicherbereichen und die Regelung von Datenkollisionen. Letzteres geschieht in Zusammenarbeit mit dem Betriebssystem.[[4]](#footnote-5)

(aus meiner PA kopiert 🡪 muss noch bearbeitet werden!)

* + 1. Software-as-a-Service

Die Art und Weise wie Software vertrieben wird hat sich durch die Popularität des Internets verändert. So wird Software nicht mehr gekauft, sondern an verschiedene Nutzer vermietet und über das Internet zur Verfügung gestellt.[[5]](#footnote-6) Software-as-a-Service (SaaS) bezeichnet die Bereitstellung von existierenden Applikationen in einem ganzen oder partiellen Remote-Service.[[6]](#footnote-7) So können Nutzer in Form von beispielsweise web-basierten Applikationen auf die Applikationen des Bereitstellers zugreifen und den Zugang der Software Applikation, welcher mehreren Nutzer gleichzeitig bereitsteht, mittels „pay-per-use“-Basis zahlen.[[7]](#footnote-8)

* + 1. Database-as-a-Service

Database-as-a-Service (DbaaS) stellt die Lieferung einer Datenbanksoftware mit verbundenem physikalischem Datenbankspeicher als Service dar. Als ein Managed Service wird dem Kunden auf einer „pay-per-usage“-Basis ein auf Anfrage bestehender Zugang zu einer Datenbank gegeben.[[8]](#footnote-9) DbaaS stellt Transaktionen und Funktionen auf einem entfernten Datenbankserver zur Verfügung, welcher mit anderen Nutzern genutzt wird, aber logisch wie eine lokale Datenbank funktioniert. Dadurch ist DbaaS eine Spezialisierung von Software-as-a-Service (SaaS).[[9]](#footnote-10)

* 1. Datenbank Prinzipien

Datenbanken folgen unterschiedlichen Prinzipien. So basieren relationale Datenbanken einem anderen Prinzip als beispielsweise NoSQL Datenbanken. Im Folgenden werden die drei meistgenutzten Prinzipien erläutert und differenziert.

* + 1. ACID

Das ACID-Prinzip steht für Atomarität, Konsistenz, Isolation und Dauerhaftigkeit der durchzuführenden Transaktionen,[[10]](#footnote-11) und wird besonders in der SQL Umgebung genutzt.[[11]](#footnote-12)

Die Atomarität steht sicher, dass eine Transaktion ganz oder gar nicht durchgeführt wird. Kann ein Teil der Transaktion also nicht komplett ausgeführt werden, so schlägt es als Gesamtes fehl.[[12]](#footnote-13) Damit wird entweder die gesamte, oder keine, Transaktion ausgeführt.[[13]](#footnote-14) Die Konsistenz sichert, dass die Datenbank sowohl vor als auch nach der Transaktion einen soliden Status vorweisen kann,[[14]](#footnote-15) während die Isolation sicherstellt, dass mehrere Transaktionen, welche zur selben Zeit ausgeführt werden, nicht die jeweils andere Transaktion beeinflussen, und somit isoliert durchgeführt werden. Dementsprechend müssen gleichzeitig ausgeführte Transaktionen „serializable“ sein.[[15]](#footnote-16) Zur Sicherstellung des permanenten Status einer Transaktion wird das ACID-Prinzip der Dauerhaftigkeit genutzt. Sobald eine Transaktion abgeschlossen worden ist und Daten gespeichert wurden, sollten die geschriebenen Daten dauerhaft verfügbar sein, auch bei einem Crash oder Fehler des Systems.[[16]](#footnote-17)

* + 1. BASE

Das BASE-Paradigma steht für „Basically Available, Soft State, Eventual Consistency“ und stellt generell das Gegenstück zu den ACID-Proportionen dar.[[17]](#footnote-18) Die Konsistenz ist nach einer Abfrage nicht mehr in einem „solid state“, sondern in einem „soft state“ und soll nicht direkt nach Ende einer Transaktion erreicht werden, sondern nach Ende einer Operation.[[18]](#footnote-19) Der Fokus der BASE-Prinzipien besteht in der permanenten Verfügbarkeit,[[19]](#footnote-20) und nicht in der stetigen Konsistenz, welche nach dem ACID-Prinzip bestehen soll.

Mit Blick auf NoSQL Datenbanken sollte die Datenbank zwischen ACID und BASE bestehen.[[20]](#footnote-21)

* + 1. CAP-Theorem

Das CAP-Theorem wurde zum ersten Mal von Eric Brewer im Jahr 2000 vorgestellt.[[21]](#footnote-22) Dabei steht das Theorem aus drei verschiedenen Faktoren: Konsistenz (Consistency), Verfügbarkeit (Availability) und Ausfalltoleranz (Partition Tolerance).[[22]](#footnote-23)

Diese Prinzipien sagen Folgendes aus:[[23]](#footnote-24)

1. Die verfügbaren Daten sind auf allen Maschinen gleich und Updates sollen frequent durchgeführt werden
2. Die Daten müssen permanent verfügbar, und ständig zugänglich sein
3. Bei Fehlern der Maschine sollte die Datenbank weiterhin laufen, ohne die Arbeit stoppen zu müssen.



Abb. 1: CAP-Theorem[[24]](#footnote-25)

Das Theorem besagt, dass nur zwei von den drei Aspekten in einem verteilten System ermöglicht werden können. Demnach muss sich im Voraus also für zwei Aspekte entschieden werden.

* 1. Entwicklungen in der Datenbankbranche

Relationale Datenbanken wurden in den 70er Jahren von E.F. Codd[[25]](#footnote-26) mit dem Ziel entwickelt, strukturierte Daten in einer organisierten Art und Weise in Tabellen zu speichern,[[26]](#footnote-27) und die hierarchischen oder netzwerkartigen Datenbanken als Vorläufer der relationalen Datenbanken abzulösen.[[27]](#footnote-28)

Die hauptsächliche Verantwortlichkeit besteht bei relationalen Datenbanken zurzeit in der Datenverarbeitung, besonders in Verbindung mit der Speicherung von finanziellen Informationen und persönlichen Daten.[[28]](#footnote-29) Beschränkungen bestehen jedoch besonders in dem starken Datenwachstum, durch welchen Datenabfragen durch das große Volumen der Daten nicht mehr so effizient gestaltet werden kann. Außerdem zeichnen sich Schwierigkeiten in der Speicherung und dem Management von größeren Datenbanken ab.[[29]](#footnote-30) Viele Unternehmen setzen aus diesen Gründen nicht-relationale Datenbanken, auch unter dem Namen NoSQL Datenbanken, in ihren Betrieben ein.[[30]](#footnote-31)

NoSQL wurde zuerst in den 1998er Jahren von Carlo Strozzi genannt, wuchs aber erst um 2009 zu einem Konkurrenten der relationalen Datenbanken.[[31]](#footnote-32) Besonders mit dem Aufkommen von des Internets und der Vielzahl von webbasierten Anwendungen gewannen die nicht-relationalen Konzepte gegenüber den relationalen an Gewicht.[[32]](#footnote-33)

* 1. Datenbankmodelle
     1. Relationale Datenbanken

Relationale Datenbanken werden primär durch die ACID-Prinzipien repräsentiert. Daraus schließend sollten relationale Datenbanken besonders dann gewählt werden, wenn Konsistenz von hoher Relevanz ist.[[33]](#footnote-34)

[[34]](#footnote-35)Vorteile bestehen besonders in der langzeitigen Nutzung dieses Modells. Als Industriestandard können relationale Datenbanken schon viel Erfahrung abzeichnen. Die Dokumentation von Professionals, welche in ihrer Karriere die gleichen Paradigmen genutzt haben, stellt weiterhin nützliche Hilfestellung bei der Entwicklung von Datenbanken dar. Ein weiterer Vorteil von relationalen Datenbanken ist die gegebene Reife. Global Player wie International Business Maschine (IBM) und Microsoft, sowie die Open-Source Community bieten stetige Verbesserungen zu dem bestehenden Datenbankmodell und vereinfacht somit die Nutzung. Auch die hohen Levels an Support und die Möglichkeit komplexe Joins zu kreieren, damit Datenbank Administratoren tiefe und komplexe Abfragen implementieren können, stellen durchaus positive Aspekte der relationalen Datenbanken dar.

Negative Aspekte bestehen besonders in der Pflicht, Daten vor der Speicherung definieren zu müssen.[[35]](#footnote-36) Weiterhin unterstützen relationale Datenbanken nicht die hohe Skalierbarkeit, soweit nicht eine bessere Hardware zu diesem Zweck implementiert werden kann.[[36]](#footnote-37) Ein weiterer Defizit besteht in der Abfragesprache in Form von SQL, welche bei strukturierten Daten zwar effizient ist, bei unstrukturierten Daten jedoch höchst komplex wird.[[37]](#footnote-38)

* + 1. NoSQL Datenbanken

NoSQL Datenbanken existieren in dem Spektrum zwischen den ACID und BASE-Prinzipien[[38]](#footnote-39) und werden primär durch die fehlenden Relationen zwischen den verschiedenen Datensätzen charakterisiert.[[39]](#footnote-40) NoSQL Implementation baiseren auf Entitäten und unterstützen die Funktionen von relationalen Datenbankmanagementsystemen wie das Sortieren, Indexen, Projektieren und Abfragen von Daten. Zusätzlich werden Joins und ACID Garantien für eine höhere Transaktionsgeschwindigkeit eingetauscht.[[40]](#footnote-41) Popularität für diesen alternativen Ansatz wurde durch die einfache Skalierbarkeit, sowie einer verbesserten Sicherheit gewonnen.[[41]](#footnote-42)

Meier und Kaufmann erläutern in ihrem Buch „SQL- & NoSQL-Datenbanken“, dass folgende Bedingungen für eine NoSQL Datenbank vorliegen müssen[[42]](#footnote-43):

1. Das zugrundeliegende Datenbankmodell ist nicht relational
2. Das Datenbanksystem erfüllt die Anforderungen für umfangreiche Datenvolumen, flexible Datenstrukturen und Echtzeitverarbeitung, auch bekannt unter den drei V’s (Volume, Variety und Velocity)
3. Das Datenbankschema unterliegt keinem fixen Schema
4. Die Datenbankarchitektur unterstützt horizontale Skalierbarkeit und verteilte Webanwendungen
5. Das Datenbanksystem unterstützt Datenreplikation
6. Durch das CAP-Theorem ist Konsistenz nur verzögert gewährleistet, falls hohe Verfügbarkeit und Ausfalltoleranz gewährleistet angestrebt wird

Generell kann man bei NoSQL Datenbanken zwischen vier verschiedenen Arten differenzieren:

* Key-Value Store
* Document Store
* Column Store
* Graphendatenbank

Bei dem Key-Value Store werden alle gelagerten Daten durch ein „key-value pair“ je Datei repräsentiert. Damit ist jeder Schlüssel einzigartig und erlaubt es die Datensatzinformationen als Werte anzufordern. Diese Struktur ist auch unter dem Namen „Hash Table“ bekannt, in dem Datenabrufe mit Hilfe des Schlüssels durchgeführt werden.[[43]](#footnote-44) Zudem können in dem System des Key-Value Stores sowohl strukturierte als auch unstrukturierte Daten gespeichert werden.[[44]](#footnote-45)

Die Document Store Datenbanken sind dafür ausgelegt, die gespeicherten Daten, welche in Dokumenten verschiedener Formate, wie beispielsweise XML oder JSON, gespeichert werden, zu handhaben.[[45]](#footnote-46) Der Nutzer kann eine unbegrenzte Anzahl von Felder jeder Länge zu einem Dokument hinzufügen[[46]](#footnote-47) und ist daher komplexer als der Key-Value Store.[[47]](#footnote-48)

Der Column Store besitzt eine Struktur, welche ähnlich zu der der relationalen Datenbank ist. Hier werden alle Daten in Reihen und Spalten gespeichert. Spalten, welche ähnliche Daten besitzen und vermehrt zusammen abgerufen werden, können zudem gruppiert werden.[[48]](#footnote-49) Der Unterschied zu relationalen Datenbanken besteht in dem Sachverhalt der unstrukturierten Daten. Während Datensätze in relationalen stark strukturiert sein müssen, so müssen diese in dem Column Store nur eng miteinander verwandt sein.[[49]](#footnote-50)

Graphendatenbanken werden genutzt, wenn gespeicherte Daten mit verflochtenen Elementen wie Social Networking oder Road Maps, als Graph dargestellt werden sollen.[[50]](#footnote-51)

[[51]](#footnote-52)Vorteile der NoSQL Datenbanklösungen bestehen zum einem in einem einfachen Applikationsmanagement, welches die Notwendung von Applikationsänderungen oder Datenbankschema-Änderungen nimmt. Weiterhin bieten NoSQL Datenbanken die Möglichkeit, Clusters und Nodes transparent zu nutzen, ohne dass das Datenbankadministrationsmanagement oder eine manuelle Distribution von Informationen benötigt wird. Zudem sind NoSQL Datenbanken dafür ausgelegt, Daten automatisch zu administrieren und wiederherzustellen.

Wird die Literatur zu NoSQL Datenbanken betrachtet, so wird als Vorteil primär die Skalierbarkeit in den Vordergrund gezogen. Die horizontale Skalierbarkeit erlaubt es zum einen Anfragen durch niedrig-preisliche Server zufriedenzustellen, wodurch ein Unternehmen seine Kosten verringern kann.[[52]](#footnote-53) Außerdem können einzelne Dokumente oder „key-value pairs“ zwischen verschiedenen Servern verschoben werden, ohne dass die Datenbank als Ganzes an Integrität verliert. Dieser Sachverhalt vereinfacht die horizontale Skalierbarkeit und Datenredundanz.[[53]](#footnote-54)

Nachteilig zu sehen ist die unzuverlässige Natur der Datenverfügbarkeit.[[54]](#footnote-55) Die letztendliche Konsistenz bedeutet, dass zwei Personen mit verschiedenen Kopien des gleichen Dokuments eine aktualisierte und nicht-aktualisierte Version eines Dokuments sehen können, wenn kurz zuvor eine write-Operation durchgeführt worden ist.[[55]](#footnote-56) Eine weitere Schwäche der NoSQL Datenbanken besteht in der Datenredundanz und der Speicherung unstrukturierter Daten, welche eine finanzielle Belastung für das Unternehmen darstellen kann.[[56]](#footnote-57) Weiterhin ist die Technologie der NoSQL-Datenbanken noch sehr jung und dadurch teilweise fehlerhaft.[[57]](#footnote-58) Eines der größten Schwachstellen ist jedoch das CAP-Theorem, bei welchem zwischen zwei der drei Eigenschaften gewählt werden muss.[[58]](#footnote-59)

* + 1. Zusammenfassung

Mahamed, Altrafi und Ismail fassen in ihrem Zeitschriftenartikel die Unterschiede von relationalen Datenbanken und noSQL Datenbanken nach verschiedenen Kriterien zusammen. In Form einer Tabelle können diese wie folgt dargestellt werden:

|  |  |  |
| --- | --- | --- |
|  | **Relationale Datenbanken** | **NoSQL Datenbanken** |
| **Transaktionsausfallsicherheit** | Garantiert sehr hohe Transaktionsausfallsicherheit | Spanne von BASE und ACID |
| **Datenmodell** | Daten als Tupel repräsentiert und durch Relationen gruppiert  Die Relationen beinhaltet Tupel (Reihen), mit Attributen (Spalten) Attribut kann durch Domain (Menge von Werten) angesprochen werden | Nehmen Modelliertechniken wie Key-Value Stores, Graphen- oder Dokumenten Stores.  NoSQL Datenbanken kann aus einem oder mehreren Datenmodellen bestehen |
| **Skalierbarkeit** | vertikal | horizontal |
| **Cloud** | Nicht gut für den Cloud-Betrieb geeignet, da sie nicht wichtige Funktionen nicht unterstützen und die nur bis zu einem bestimmten Limit skaliert werden kann | Beste Lösung für Cloud Datenbanken, da alle Charakteristiken, welche NoSQL definieren, in der Cloud Umgebung gewünscht werden |
| **Big Data Handling** | Lösung ist Skalierbarkeit und Datenverteilung in Form von horizontaler oder vertikaler Skalierbarkeit – nur begrenzt möglich | Entworfen um Big Data zu handhaben – vorhandene Methoden erleichtern und verbessern die Performance vom Speichern und Auslesen der Daten |
| **Data Warehouse** | Über Zeit nimmt die Anzahl der gespeicherten Daten stark zu, wodurch ein Big Data Problem entsteht – führt zu Performance Abbau | Design fokussiert sich auf hohe Performance, Skalierbarkeit, Verfügbarkeit und das Speichern von Big Data – dadurch eine gute Lösung für Data Warehouses |
| **Komplexität** | Erhöht sich, da Nutzer Daten in konvertieren müssen, damit sie in die Tabellen passen | Haben die Fähigkeit unstrukturierte und semi-strukturierte Daten zu speichern |
| **Crash Recovery** | Crash Recovery durch den Recovery Manager, welcher dafür verantwortlich ist sicherzustellen, dass Transaktionen atomar und dauerhaft sind [durch log-files und ARIES algorithmus] | Benötigt Replikationen als Backup, um von einem Crash zu recovern |
| **Sicherheit** | Hat sehr sichere Mechanismen um Security Services bereitzustellen – trotzdem gibt es Sicherheitsrisiken wie SQL Injections, Cross Site Scripting, Root Kits etc. | Wurde eher für die Verbesserung von Performance und Big Data auf den Markt gebracht, nicht um Sicherheitslücken zu füllen |

Tabelle 1: Relationale vs. NoSQL Datenbanken[[59]](#footnote-60)

1. DynamoDB

Das zweite theoretisch ausgelegte Kapitel beschäftigt sich mit der Einführung in die *Amazon Web Services* und stellt im weiteren Verlauf die DynamoDB im Detail vor. Der Einstieg widmet sich den Hintergründen der *Amazon Web Services*, um ein grobes Verständnis für die Infrastruktur zu erzeugen. Anschließend sollen grundlegende Informationen zu DynamoDB hinterlegt werden. Infolgedessen gilt es, die Art der Datenverwaltung und das Datenmodell der NoSQL-Datenbank zu erläutern und zu untersuchen. Abschließend soll ein Vergleich mit einer alternativen NoSQL-Lösungen, die Einschränkungen bzw. die besonderen Merkmale von DynamoDB ausarbeiten.

* 1. Generelle Informationen über AWS

Die DynamoDB ist ein Teil der *Amazon Web Services* Infrastruktur. Somit ist es relevant, die Merkmale und Besonderheiten der Infrastruktur zu erläutern, bevor eine Einführung in Datenbanklösung ausgeführt werden kann. Das folgende Unterkapitel ist auf die Einführung in AWS ausgerichtet. Im Verlauf dieses Unterkapitels werden die Ziele und Absichten der Dienstleistung aufgezeigt. Im Anschluss folgt eine Gegenüberstellung der Vorteile und Nachteile der Infrastruktur.

* + 1. AWS ein Beispiel für Infrastructre-as-a-Service (IaaS)

Das Modell von IaaS stellt laut einem Artikel von Wolfgang Sommergut „die IT-Ressourcen, wie Rechenleistung, [Speicherplatz] oder Netzwerkkapazitäten zur Verfügung“[[60]](#footnote-61). Außerdem hat der Anwender die Chance, das Betriebssystem sowie die Applikationen eigenständig zu steuern und die Infrastruktur nach eigenem Bedarf zusammenzustellen. Im Wesentlichen handelt es sich bei AWS also um ein *IaaS,* die die Möglichkeiten zur flexiblen Datenspeicherung und den Ausbau der verfügbaren Rechenleistung bereitstellt. Neben den genannten Ausprägungen bietet AWS eine Vielfalt von Ressourcen an, die in Tabelle 1 zusammengefasst sind.

|  |  |
| --- | --- |
| IT-Ressource | Beschreibung |
| Online-Speicher | Es handelt sich hierbei um Cloud-Speicher. |
| Near-line Speicher | Speichermedien mit längeren Zugriffszeiten |
| Server | Server zum Bereitstellen von diversen Diensten |
| Amazon Simple Queue Services | Dienste zur Nachrichtenübermittelung |
| Datenbanksystem | NoSQL-Lösungen zur Speicherung verschiedener Datenformate |

Tabelle 2: Einige IT-Ressourcen von AWS[[61]](#footnote-62)

* + 1. Gründe für die Verwendung von AWS

Die in 2006 gegründete Infrastruktur gehört zu den ersten seiner Art und ist mittlerweile Marktführer (siehe Abbildung 1) in diesem Bereich. Aus diesem Grund verlassen sich Plattformen, wie Netflix, AirBnB und Spotify auf die Dienstleistungen des Anbieters. Angesichts der soliden Etablierung im Markt ist das Unternehmen in der Lage vielseitige Servicepakete und Ressourcen bereitzustellen. Im Vergleich zu den Wettbewerbern kann Amazon bis zu 10-mal mehr Kapazitäten bereitstellen.[[62]](#footnote-63)

Abb. 1: IaaS Martkanteile[[63]](#footnote-64)

Laut Linton sind die „Flexibilität, Kontrolle und Skalierbarkeit“[[64]](#footnote-65) die größten Vorteile der *Amazon Web Services*. Dabei hat der Anwender die Möglichkeit Ressourcen beliebig zu reduzieren oder zu erweitern. Für den Fall, dass die Dienstleistungen nicht zufriedenstellend sind, hat der Anwender die Möglichkeit die Applikationen auf einer alternativen Plattform zu verlagern. Zusätzlich bietet AWS mit seinen 15 geografischen Standorten einen hohen Grad an Verfügbarkeit und Sicherheit für die Kunden.[[65]](#footnote-66) Das Preismodell von AWS richtet sich auf den Bedarf der Ressourcen. Hierbei können Kapazitäten beliebig zur Applikation hinzugefügt werden, sollten diese notwendig sein.

* + 1. AWS Anwendungsbeispiele

Die Infrastruktur erlaubt aufgrund der vielfältigen Dienstleistungspakete eine große Anzahl an Anwendungsbeispielen. Dieses Unterkapitel stellt einige dieser Anwendungsbeispiele vor und ermöglicht somit einen Einblick in die Möglichkeiten, die Kunden mit AWS haben.

**Hosten von Webseiten mit Elastic Compute Cloud**

Die Serverfarmen der Infrastruktur können für das Hosten von Webseiten verwendet werden.[[66]](#footnote-67) Dies ist besonders für Großhändler lukrativ, da diese mit umfangreichen Internetseiten bspw. auch mit einem Webshop auftreten. Die EC2 Infrastruktur von Amazon fängt hierbei die Anfragen ab und kollaboriert mit einer Datenbank, die weitere Details zu den Produkten des Webshops enthält**.[[67]](#footnote-68)** Zusätzlich erhält der Kunde professionelle Wartungsmöglichkeiten und technische Beratung, die die Zuverlässigkeit und Verfügbarkeit der Webseiten erhöhen.

**Ausführen von Applikationen**

Hierfür wird die Amazon Cloud eingerichtet und mit dem Unternehmensnetzwerk mit einem Virtual Private Network (VPN) verbunden.[[68]](#footnote-69) Im Weiteren werden Applikationsserver auf virtuelle Maschinen (VM) installiert, auf denen anschließend die Applikation ausgeführt werden kann. Die Applikationen laufen nun auf der Amazon Cloud und können unabhängig von geografischen Grenzen gepflegt und bedient werden.

**Datenspeicherung und Datenarchivierung mit Simple Storage Service**

Neben dem Hosten und Ausführen bietet AWS die Möglichkeit der dauerhaften Datenspeicherung und Datenarchivierung. Das bekannteste Produkt hierbei ist der S3.[[69]](#footnote-70) Dieses Angebot visiert insbesondere die gesetzlichen Vorschriften zur sicheren Datenaufbewahrung für einen gewissen Zeitraum. Dabei werden Daten dezentral gespeichert, um Verluste zu verhindern.

* + 1. Vor- und Nachteile von AWS

**Vorteile**

Die Amazon Infrastruktur erweitert das Serviceportfolio kontinuierlich, wodurch neue Servicepakete und Verbesserung stetig zur Cloud hinzukommen.[[70]](#footnote-71) Ein weiterer Vorteil des Cloud-Anbieters ist die Flexibilität und Skalierbarkeit, die es den Kunden erlaubt Ressourcen arbiträr aufzustellen.[[71]](#footnote-72) Aus diesem Grund zahlt der Kunde, wie bei SaaS und DbaaS, ausschließlich für die Ressourcen, die tatsächlich verwendet werden. Aus ökonomischer Perspektive reduziert die AWS-Cloud die Produkteinführungszeit.[[72]](#footnote-73) Das bedeutet, wenn ein Anwendungsfall zusätzliche Ressourcen benötigt, können diese in wenigen Minuten vollständig benutzt werden und das Produkt kann somit schneller zum Abnehmer gebracht werden.

**Nachteile**

Es besteht ein hoher Grad an Abhängigkeit gegenüber einem Drittanbieter. In Bezug auf die Standortwahl der Kunden ist es schwer nachvollziehen, ob sich die Daten in der Cloud tatsächlich am ausgewählten oder priorisierten Standort befinden. Für den Kunden liefert dies eine Intransparenz. Zusätzlich ist die große Auswahl von AWS für Kunden und Entwickler nicht immer überschaubar.[[73]](#footnote-74) Das führt dazu, dass die Dienstleistungen nicht optimal genutzt werden. Zum Thema Kosten ist AWS in einigen Bereichen teurer als die direkten Wettbewerber, wie bspw. Microsoft Azure.[[74]](#footnote-75)

* 1. DynamoDB

Nachdem im vorherigen Kapitel die AWS-Infrastruktur vorgestellt wurde, visiert dieses Kapitel den NoSQL-Datenbankservice von Amazon an. Dabei bietet der Cloud-Anbieter die SimpleDB und die DynamoDb als DbaaS an. Die SimpleDB ist hierbei besonders für das strukturelle Speichern von großen Mengen an Daten geeignet.[[75]](#footnote-76) In den nachfolgenden Unterkapiteln werden die Charakteristiken der DynamoDB ausgiebig analysiert. Zunächst sollen die Grundeigenschaften der DynamoDB festgehalten werden. Im Anschluss dazu werden Besonderheiten diskutiert.

* + 1. Was ist DynamoDB

Bei DynamoDB handelt es sich um eine NoSQL-Datenbanklösung, die das Ziel von Skalierbarkeit und guter Performanz verfolgt.[[76]](#footnote-77) Die DynamoDB ist ein *Key-Value Store*, aus diesem Grund besitzen Datensätze einen Primärschlüssel (Key) und die dazugehörigen Attribute. Die Abbildung 2 verdeutlicht, dass die DynamoDB als *Database as a Service* in der der AWS-Infrastruktur eingeordnet ist.

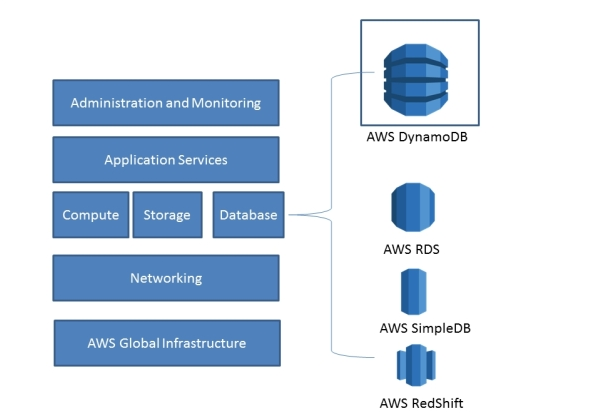


Abbildung 2 Einordnung von DynamoDB in AWS

Die Nutzung der Datenbank setzt ein AWS-Nutzerkonto voraus.[[77]](#footnote-78) Neue Tabellen können direkt in der AWS-Managementkonsole erstellt werden, sodass zusätzliche Software nicht notwendig ist. Ein weiteres Merkmal von DynamoDB ist die Tatsache, dass die Daten automatisch in AWS-Regionen repliziert werden, was sich auf die Datenbeständigkeit auswirkt.

* + 1. Hintergrund der Entwicklung von DynamoDB

Diese Dienstleistung der DynamoDB wurde in 2012 zu AWS hinzugefügt und ist seit dem bereits in verschiedenen Anwendungen integriert.[[78]](#footnote-79) Laut Vogels, dem Chief Technology Officer (CTO) von Amazon, nutzten die verschiedenen APIs des Unternehmens, vor der Entwicklung der NoSQL-Datenbanken, eigene Datenbanken. Aus diesem Grund liefen die Dienstleistungen separat und konnten nicht verkoppelt werden.[[79]](#footnote-80) Das stetige Wachstum von Amazon benötigte eine skalierbare Datenbank, die zu einem eine hohe Verfügbarkeit nachwies, zum anderen von verschiedenen APIs genutzt werden konnte. Die vorhandenen relationalen Datenbanken verfolgen in Betracht auf das CAP-Theorem, eher das Ziel der Konsistenz, jedoch lag der Schwerpunkt für Amazon bei der Verfügbarkeit der Systeme. Darüber hinaus benötigten die RDBMS kostenaufwendige Hardware und Personal.[[80]](#footnote-81) Die Entwicklung von DynamoDB galt somit als Grundstein für künftige verteilte Datenbanksysteme.

* + 1. Preismodell

Beim Erstellen des AWS-Nutzerkontos und bei der initialen Verwendung von DynamoDB fallen keine Kosten an. Somit besteht keine Grundgebühr. Das kostenfreie Kontingent bietet Neukunden 25GB Speicherplatz und jeweils 25 Schreib-und Lesekapazitätseinheiten. Mithilfe des Kontingents können laut Amazon Web Services bis zu 200 Millionen Anfrage pro Monat erzeugt werden.[[81]](#footnote-82) Zusätzliche Kapazitäten werden mit einer Stundenpauschale bestimmt. Die Übersteigung des Grundspeicherplatzes versursacht Kosten in Höhe von $0.306 monatlich pro Gigabyte. Falls für die Anwendungen weitere Schreib-und Lesekapazitätseinheiten benötigt werden, können diese für $0.00793 pro Stunde in Anspruch genommen werden.[[82]](#footnote-83) Die gesamten Kosten für die Speicherplatznutzung werden zum einem von der tatsächlichen Nutzung und zum anderen von einem Speicheraufschlag pro 100 Byte ermittelt.

* + 1. Architektur

Im folgenden Teil wird die Architektur der DynamoDB vorgestellt. Diese besteht aus verschiedenen Komponenten, die individuell gezeigt werden sollen. Einige dieser Komponenten werden bereits in bekannten Datenbanklösungen verwendet. Andere jedoch wurden speziell für DynamoDB entwickelt, um der Skalierbarkeit und Verfügbarkeit gerecht zu werden. Da es sich bei DynamoDB um ein verteiltes Datenbanksystem handelt, werden Daten auf einem *Master* und diversen *Nodes* gespeichert, die jeweils einen Bruchteil der Daten enthalten. Der *Master* enthält alle Daten und ist stets auf dem aktuellsten Standpunkt.[[83]](#footnote-84) Alle *Nodes* sind hierbei gleichzustellen, um Priorisierungen zu vermeiden und die Pflege der Daten zu erleichtern.[[84]](#footnote-85)

**Belastungsausgleich (Load balancing)**

Bei DynamoDB müssen die Daten auf den verschiedenen *Nodes* gleichmäßig balanciert werden. Die Verteilung der Daten wird mittels der Hash-Code-Methode praktiziert.[[85]](#footnote-86) Hierbei erhalten die einzelnen Nodes und die Schlüssel der Daten einen Hash-Code, der den Datenzugang gestattet und die Nodes identifiziert. Die Herausforderung hierbei ist es, die Schlüssel und die dazugehörigen Attribute gleichmäßig auf die Anzahl der Nodes zu verteilen.[[86]](#footnote-87)

**Replikation von Daten (Data replication)**

Die Replikation der Daten wird von einem sogenannten *coordinator node* gehandhabt. Die Anzahl der Replikationen muss zunächst festgelegt werden. Der koordinierende Datenbankknoten verteilt entsprechend der festgelegten Anzahl der Replikationen die Daten auf diverse Knotenpunkte.[[87]](#footnote-88) Die Replikation ermöglicht die Datenbeständigkeit und vermeidet somit den Verlust von sensiblen Daten. Diese Methode macht die DynamoDB ebenfalls resistent gegen Ausfällen von vereinzelten Datenbankkonten, da die Daten auf mehreren *Nodes* liegen.[[88]](#footnote-89)

**Versionsverwaltung von Daten**

Die DynamoDB basiert auf dem BASE-Modell. Daher sind nicht alle *Nodes* unmittelbar nach der Aktualisierung auf dem neuesten Stand, sondern das Update erreicht einige *Nodes* zu einem späteren Zeitpunkt. Um einen schwerwiegenden Datenverlust zu vermeiden, erstellt DynamoDB verschiedene Versionen von Objekten.[[89]](#footnote-90) Für die Versionsverwaltungen und für die Identifizierung von Konflikten zwischen den *Nodes*, wird eine *vector clock* verwendet.[[90]](#footnote-91) *Die vector* clock ist eine Liste *von Nodes* bzw. Servern, die jeweils die Versionsnummer eines Objekts darstellt. *Die vector clock* erlaubt es, mehrere Versionen von einem Objekt zu behalten.[[91]](#footnote-92) Beispielsweise könnte es vorkommen, dass Anfragen eines Benutzers von verschiedenen Servern behandelt werden, somit liegen auf den *Nodes* jeweils verschiede Versionen von den Daten vor. Die *vector clock* subsumiert die Veränderungen und zeigt jeweils den Stand der einzelnen Server an.

**Abwicklung von Anfragen (Request handling)**

Der Benutzer hat die Möglichkeit *get-* und *put*-Anfragen über HTTP an die Datenbank zu senden. Die Anfrage kann vom *load balancer* aufgenommen werden und entsprechend an einem *Node* mit freier Kapazität gesendet. Alternativ kann der Benutzer selbst einen Server bestimmen, der die Anfrage abarbeiten soll.[[92]](#footnote-93) Wenn ein Server die Anfrage entgegennimmt, wird zunächst der Schlüssel (Key) analysiert, bevor die Anfrage abgearbeitet werden kann.[[93]](#footnote-94)

**Zugehörigkeit von Nodes (Ring membership)**

Diese Komponente der Architektur beschreibt die Tatsache, dass die einzelnen *Nodes* der Datenbank in einer Ringtopologie eingeordnet sind. Aufgrund von Ausfällen oder Überlastungen können *Nodes* aus der ringförmigen Anordnung (siehe Abbildung 3)[[94]](#footnote-95) entnommen und hinzugefügt werden. Für den Fall, dass der *load balancer* die Anfrage des Nutzers entgegennimmt, ordnet dieser die Anfrage einem *Node* mit freien Kapazitäten zu. Wie in Abbildung 3 verdeutlicht, hat ausschließlich der Administrator das Recht die Anordnung der *Nodes* zu manipulieren. Für diese Aufgabe verwendet er ein CLI oder eine separate Plattform für den Internetbrowser. Die Veränderungen in der Anordnung werden den verbleibenden *Nodes* über ein *Gossip protocol*[[95]](#footnote-96) mitgeteilt. Dieses Protokoll ermöglicht die Weiterleitung von Informationen an weitere Nodes[[96]](#footnote-97). Wenn ein weiterer Knotenpunkt hinzugefügt wird, absorbiert dieser die Informationen der beständigen *Nodes*, um sich auf einem einheitlichen Standpunkt zu befinden.

Node 1

Node 2

Node 3

Node 4

Node 5

Administrator

Abb.: Ring membership der DynamoDB

* 1. Art der Datenverwaltung/Datenmodell

In diesem Teil soll die Datenverwaltung und das Datenmodell der DynamoDB erklärt werden. Dieses Verständnis ist notwendig für das Erstellen von Tabellen und für die Manipulation der Datensequenzen. Zunächst wird das Konzept des Datenmodells erläutert. Anschließend werden die Merkmale der DynamoDB ausgiebig vorgestellt. Auf Basis dieser Kenntnisse wird die Key-Value Eigenschaft demonstriert. Dieses Unterkapitel schließt mit den Datentypen, die von der DynamoDB unterstützt werden, ab.

* + 1. Tabelle, Elemente und Attribute

Das Datenmodell der DynamoDB besteht aus Tabellen, Elementen und Attributen. Eine Tabelle ist äquivalent zu einer Datenbanktabelle in einem relationalen Datenbanksystem. Das Schema dieser Tabelle muss in DynamoDB bei der Erstellung festgelegt werden. Dabei sind jedoch nur Tabellenname, Primärschlüssel und sekundärer Index anzugeben. Im Sinne der relationalen Datenbanken werden nicht auch alle Spalten festgesetzt.[[97]](#footnote-98) Bezug auf Tabelle 2 sind die Elemente (Items) ähnlich wie die Zeilen in einem relationalen Datenbankmodell. Die Spalten hingegen werden im Jargon der DynamodB als Attribute bezeichnet.

|  |  |
| --- | --- |
| RDBMS | DynamoDB |
| Zeilen | Elemente |
| Spalten | Attribute |

Tabelle 2: Datenmodell von DynamoDB[[98]](#footnote-99)

Die Elemente (Items) bilden die einzelnen Datensätze einer Tabelle ab. Solch ein Element kann beliebig viele Attribute enthalten[[99]](#footnote-100). Ein Beispiel für das hierarchische Konzept in der DynamoDB, wird in der folgenden Abbildung dargestellt. Die Tabelle enthält mehrere Elemente und die Elemente verfügen wiederum über mehrere Attribute bzw. Eigenschaften. Auf den Tabellen und Elementen können jeweils verschieden Operationen getätigt werden. Mithilfe des *UpdateItem*-Befehls können neue Attribute hinzugefügt und bestehende Attribute gelöscht oder aktualisiert werden[[100]](#footnote-101).

Student

Student 2

Student 1

(name,“Paul“)

(Matr.,“2“)

(name,“Nora“)

Tabelle

-create

-update

-delete

Elemente

-add

-update

-delete

Attribute

(Geb.,“01.01.1994“)

(Matr.,“3“)

Abb. : Hierarchien in DynamoDB

* + 1. Primärschlüssel

Beim Kreieren einer Tabelle muss neben dem Tabellennamen und den Datentypen auch einen Primärschlüssel festgelegt werden. Der Primärschlüssel dient dazu, die Elemente (Items) mit einem eindeutigen Schlüssel zu versehen[[101]](#footnote-102). Das Konzept der DyanomoDB unterscheidet zwei verschiedene Arten von Primärschlüsseln.

**Partition key/Hash key**

Der Partitionsschlüssel ist der klassische Primärschlüssel, der aus einem Attribut besteht. In dem Beispiel aus der Abbildung 4 wäre der Partitionsschlüssel für die Tabelle *Student* die *Matrikelnummer* der Studenten. Der Datentyp des Schlüssels muss hierbei ein *string*, *number* oder *binary* sein.[[102]](#footnote-103) Der Name eines Studenten wäre für den Partitionsschlüssel nicht geeignet, da er nicht eindeutig zuordenbar wäre.

**Partion key und Sort key**

Neben den Partitionsschlüssel gibt es auch den *sort key* bzw. *range key*, welcher in den meisten Fällen in Kombination mit dem Partitionsschlüssel verwendet wird. Der Primärschlüssel besteht dann aus der Kombination aus beiden Attributen, wobei der *sort key* auch leer sein darf.[[103]](#footnote-104) Der *sort key* wir bei einer Datenbankabfrage verwendet, um die Ausgabe der Daten zu beschränken. Im Beispiel aus der Abbildung 4 wäre die Kombination von der Matrikelnummer und dem Geburtstag des Studenten denkbar. Dadurch könnten die Matrikelnummern aller Studenten ausgegeben werden, die ein bestimmtes Geburtsdatum haben. Die Kombination aus *hash key* und *sort key* bietet sich an, um Daten schneller und effizienter abzurufen.[[104]](#footnote-105)

* + 1. Sekundäre Indizes

Abgesehen von der Kombination zwischen den Arten der Schlüssel, gibt es ebenfalls die Möglichkeit der Indizes, die die Datenabfrage beschleunigen. Bei der Erstellung einer Tabelle generiert die DynamoDB selbstständig einen Index, welches das Primärschlüsselattribut enthält. Daher wird dieser auch als primärer Index bezeichnet.[[105]](#footnote-106) Für den schnellen Zugriff auf Attribute, die nicht Bestandteil des Primärschlüssels sind, werden sekundäre Indizes verwendet. Dies ist besonders bei Tabellen mit vielen Daten hilfreich, da für die Abfrage von Nicht-Primärschlüsselattributen nicht die ganze Tabelle abgearbeitet werden muss. Hierzu bietet DynamoDB die Möglichkeit zur Erstellung von maximal fünf sekundären Indizes pro Tabelle.

* + 1. Besonderheiten der DynamoDB

**Geringer Administrationsaufwand**

Der Nutzer fokussiert sich lediglich auf die Entwicklung der Applikation und kann Hardware Angelegenheiten außer Acht lassen.[[106]](#footnote-107) Der Anbieter stellt zusätzliche Ressourcen sofort bereit, sodass hohe Kosten für die Beschaffung von Hardware entfallen. Auch die die Organisation von Replikationen und Back-ups entfällt für die Nutzer. Aus den genannten Gründen entfallen auch Kosten für Personal, welches für die manuelle Datenbankadministration allokiert wird.[[107]](#footnote-108) Dieses Personal kann somit der Entwicklung zugeordnet werden.

**Hoher Grad an Skalierbarkeit**

Die Amazon Infrastruktur setzt keine Grenzen bezüglich Speicher und Bandbreite. Falls jedoch ein gewisses Maß an Ressourcen überschritten wird, werden weitere Kapazitäten automatisch integriert, um den Bedarf zu decken. Für den Entwickler bedeutet dies, dass eine beliebige Anzahl von Daten gespeichert und abgerufen werden können, ohne jegliche Engpässe zu begegnen.

**Schneller Datenabruf**

Die DynamoDB benutzt SSDs, um Daten zu speichern. Aus diesem Grund sind Abfragen von Daten aus dem Hauptspeicher performant.[[108]](#footnote-109) Wie bereits im letzten Unterkapitel beschrieben, erstellt die Datenbank primäre Indizes selbstständig. Somit ist der Zugriff auf Primärschlüsselattribute sehr effizient.

**Integrierbarkeit mit *Amazon Elastic MapReduce* (Amazon EMR)**

Die DynamoDB lässt sich mit Amazon EMR vereinen, welches für komplexe analytische Aufgaben verwendet wird. Für diese Zwecke stellt AWS Zugang zum Hadoop Framework bereit[[109]](#footnote-110). Die Ausgangsdaten und die Ergebnisse des *MapReduce* Vorgangs können in der DynamoDB gespeichert werden, sodass die Daten für den Kunden bzw. Nutzer zu jedem Zeitpunkt zur Verfügung stehen. Diese Möglichkeit bestätigt, dass die DynamoDB ebenfalls im *Big Data Analytics* Bereich Verwendung finden kann.

* + 1. Key-Value Eigenschaft

Wie an mehreren Teilen dieser Arbeit schon angemerkt, handelt es sich bei DynamoDB um eine Key-Value-Datenbank. Der Schlüssel hierbei wird, wie in Tabelle 3 verdeutlicht, vom Namen des Attributes abgeleitet. Der tatsächliche Inhalt des Attributes wird als *Value* bezeichnet.[[110]](#footnote-111) In Bezug auf das Beispiel aus Abbildung 4 bedeutet dies, dass das Attribut *name* den *Key* darstellt und *Paul* der eigentliche Wert (value) des Attributes ist. Die Attribute der Elemente (Items) lassen sich somit mit dem *Key* verändern. Dabei verfolgt DynamoDB die Absicht, ausschließlich Objekte zu speichern, die kleiner sind als ein Megabyte.[[111]](#footnote-112) Diese Festlegung ist besonders bei der schnellen Ausgabe der Daten bemerkbar. Solch eine Speichermethode eignet sich insbesondere für Lagerbestandsführung, Produktspezifikationen und Echtzeitanalysen.[[112]](#footnote-113) Die Schlüssel-Wert-Paare können dabei vom Datentyp *string*, *number*, *boolean* oder *binary* sein.

Ein Nachteil ist, dass beim Laden von Daten nicht nur der Wert, sondern auch der Name des Attributes ausgegeben wird. Bei besonders großen Tabellen ist dies oft ineffizient.

|  |  |
| --- | --- |
| **Key** | **Value** |
| Student1:name | Paul |
| Student2:name | Nora |
| Student1:Matr. | 2 |
| Student2:Matr. | 3 |

Tabelle 3: Key-Value Beispiel

* + 1. Datentypen

**String**

Dieser Datentyp erlaubt es Zeichenketten in DynamoDB zu verwenden. Die Verschlüsselung erfolgt hierbei nach UTF-8. Es werden außerdem keine Restriktion bezüglich der Länge de Strings spezifiziert.[[113]](#footnote-114) Für den Vergleich von mehreren Zeichenketten wird die ASCII-Codierung verwendet. Ein wichtiges Merkmal bei diesem Datentyp ist, dass keine leeren Strings gespeichert werden können.[[114]](#footnote-115) Ein Beispiel für einen eingegeben String ist der Titel eines Films:



Abbildung 5: Beispiel für den Datentyp String

**Number**

Dabei können positive und negative Zahlen in der Datenbank gespeichert werden. Zusätzlich können bis zu 38-stellige Dezimalzahlen erfasst werden. Die Zahlen können hierbei als eine Zeichenkette dargestellt werden, um die Werte auf andere Anwendungen zu übertragen, die ein bestimmtes Zahlenformat nicht unterstützen.[[115]](#footnote-116) Zum Beispiel könnte somit das Jahr eines Films abgespeichert werden, wie hier das Produktionsjahr eines Films:



Abbildung 6: Beispiel für den Datentyp Number

**Binary**

Unter den binären Datentypen fallen Datentypen, wie Dokumente, Bilder, XML-Dateien und JSON-Dateien. Das Codieren und Decodieren dieser Dateien erfolgt bei DynamoDB mittels eines Base64 Encoders bzw. Decoders. Die Länge eines binären Datentyps ist laut der AWS-Dokumentation auf 400KB limitiert.[[116]](#footnote-117) Dadurch wird verhindert, dass außergewöhnlich große Dateien in der Datenbank gespeichert werden, die wiederum den Prozess der Datenabfrage entschleunigen. Wenn bspw. das Titelbild eines Films hochgeladen werden soll, muss also zuerst das Bild in binäre Kodierung umgewandelt werden, danach kann es hochgeladen werden. In der Praxis sieht diese Eingabe folgendermaßen aus:

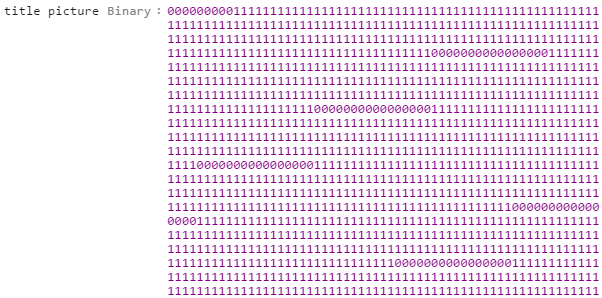


Abbildung 7: Beispiel für den Datentyp Binary

**Sets (StringSet, NumberSet, BinarySet)**

Die klassischen Datentypen können nicht nur einzeln, sondern auch als Sets abgespeichert werden. Dabei erinnert das Set an ein JSON-Array, in dem mehrere Werte des gleichen Da-tentyps gespeichert werden. Ein Set (hier in StringSet) wird in DynamoDB wie folgt dargestellt:

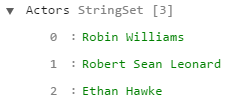


Abbildung 8: Beispiel für den Datentyp StringSet

**Boolean**

Dieser Datentyp kann gemäß der AWS Dokumentation die Werte *true* oder *false* enthalten. Dadurch lassen sich in DynamoDB Status von bestimmten Objekten speichern. Bezogen auf das Filmbeispiel, könnte ein Attribut *erhaltenOscar* auf diesem Datentypen basieren. Hierbei wäre ein *true* zu vermerken, wenn der Film einen Oscar bekommen hat und entsprechend *false*, wenn der Film keinen Oscar bekommen hat. Für Abfragen könnten dann beispielsweise alle Filme abgerufen werden, die einen Oscar erhalten haben.

**Null**

Dies ist ein weiteres Beispiel für einen skalaren Datentypen im Umfeld der DynamoDB[[117]](#footnote-118) Mit diesen Datentyp können Attribute abgebildet werden, die unbekannt oder undefiniert sind. An dieser Stelle ist es wichtig zu erwähnen, dass weder der Datentyp *Boolean*, noch *Null* für ein Primärschlüsselattribut verwendet werden können. Bezogen auf das Filmbeispiel könnten ein Attribut *Bewertung* auf diesem Datentyp aufgebaut werden. Das bedeutet, sollte einem Film keine Bewertung vorliegen oder keine bekannt sein, kann hier die Zuweisung von Null erfolgen.

**List und Map**

Neben den klassischen Datentypen können auch dokumentbasierte Datentypen, wie *List* und *Map* in der Datenbank verwendet werden. Auch hier wird die Größe der Dateien auf 400KB beschränkt. Solch eine *List* ähnelt einem JSON-Array, in welchem mehrere Werte gespeichert werden können. Die DynamoDB API ist in der Lage, auf einzelne Werte der Liste zuzugreifen[[118]](#footnote-119). Eine *Map* eignet sich für das Speichern von vollständigen JSON-Dokumenten.[[119]](#footnote-120) In einem JSON-Dokument können verschiedene Datentypen verschachtelt verwendet werden.

* 1. Vergleich mit anderes NoSQL-Lösungen

Dieses Unterkapitel stellt einen Vergleich der DynamoDB mit der Redis NoSQL-Datenbank vor. Redis ist eins der populärsten Key-Value-Datenbanken auf dem Markt und wird von Unternehmen wie Twitter und Uber genutzt.[[120]](#footnote-121) Der Vergleich dient dazu, Besonderheiten und Einschränkungen der DynamoDB hervorzuheben und ein vergleichbares System aufzuzeigen. Im Anschluss folgt ein Fazit, welches die theoretischen Grundlagen der DynamoDB zusammenfasst und die wichtigsten Aspekte herausarbeitet.

* + 1. DynamoDB vs. Redis

Die folgende Tabelle zeigt die Gemeinsamkeiten und die Unterschiede beider Datenbanken. Der größte Vorteil von Redis ist die Open-Source-Eigenschaft. Das führt zu umfassenden Nutzungsrechten der Software und das Produkt außerdem hat geringere Anschaffungskosten, da kostenaufwendige Lizenzgebühren entfallen. Die DynamoDB lässt sich mit der AWS Cloud integrieren, somit muss der Kunde keine zusätzliche physische Hardware zulegen, um das Produkt zu nutzen. Dies ist zugleich der größte Vorteil der DynamoDB. Außerdem ist DynamoDB mit *MapReduce* integrierbar, somit kann die Datenbank auch für *Big Data Analytics* genutzt werden.

|  |  |  |
| --- | --- | --- |
| Kriterien | DynamoDB | Redis |
| Datenmodell | Key-Value-Datenbank | Key-Value-Datenbank |
| Speicherform | Solid-State Drive (SSD) | In-memory (RAM) |
| Lizenz | Kommerziell (AWS) | Open source (Berkely Software Distribution) |
| Konsistenz | Eventuelle/starke Konsistenz | eventuelle Konsistenz[[121]](#footnote-122) |
| Replikationen | Master-slave Replikation | Master-slave Replikation |
| Cloud Integration | AWS Cloud | Keine Cloud Integration |
| Primärschlüssel | Zusammengesetzter Primärschlüssel kann erstellt werden | Zusammengesetzter Primärschlüssel ist nicht erlaubt[[122]](#footnote-123) |
| MapReduce Integration | Integrierbar mit AmazonEMR | Keine MapReduce Integration |
| Partitionierung | Sharding: Speichern von Datenteilen auf verschieden dezentralen Servern[[123]](#footnote-124) | Nicht unterstützt |
| Sekundäre Indizes | Sekundäre Indizes können manuell erstellt werden | Keine Möglichkeit der sekundären Indizes |

Tabelle 4: DynamoDB vs. Redis[[124]](#footnote-125)

1. Praxis

Im Praxisteil werden folgende Themen behandelt!!!!!!!

* 1. Insgesamt zum Zurechtfinden
  2. Testdatenbank mit Testdaten
  3. Datenbanksprache
  4. GUI/API
  5. Ablageform der Daten

1. Schluss
   1. Fazit
   2. Ausblick

Anhang (bei Bedarf)

# Quellenverzeichnis

**Literaturverzeichnis**

**Preiß, N. (2007):** Entwurf und Verarbeitung relationaler Datenbanken, München/Wien: Oldenbourg

**Steger, J. (2006):** Kosten- und Leistungsrechnung, 4. Aufl., München/Wien: Oldenbourg

**Stoi, R. (2003):** Management und Controlling von Intangibles, in: Studium & Praxis, 4. Jg., Nr. 1, S. 34-46

**Verzeichnis der Internet- und Intranetquellen (bei Bedarf)**

**Gesprächsverzeichnis (bei Bedarf)**

# Ehrenwörtliche Erklärung

Wir versichern hiermit, dass wir unsere Seminararbeit mit dem Thema: [ Thema einfügen ] selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ort, Datum) (Kerstin Farke)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ort, Datum) (Dhruv Mahandru)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ort, Datum) (Henning Mohr)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ort, Datum) (Käthe Vrettos)

Literaturverzeichnis

Abramova, Veronika; Bernardino, Jorge; Furtado, Pedro (2014): Experimental Evaluation of NoSQL Databases. In: *International Journal of Database Management Systems (IJDMS)* 6 (3), S. 1–16. Online verfügbar unter http://s3.amazonaws.com/academia.edu.documents/44185663/6314ijdms01.pdf?AWSAccessKeyId=AKIAJ56TQJRTWSMTNPEA&Expires=1480620894&Signature=Xuo9y8raSPFKC7sf2q3deopev5o%3D&response-content-disposition=inline%3B%20filename%3DEXPERIMENTAL\_EVALUATION\_OF\_NOSQL\_DATABAS.pdf.

Antonopoulos, N.; Gillam, L. (2010): Cloud Computing: Principles, Systems and Applications: Springer London. Online verfügbar unter https://books.google.de/books?id=SbSbdkqibwIC.

Edward, S. G.; Sabharwal, N. (2015): Practical MongoDB: Architecting, Developing, and Administering MongoDB: Apress. Online verfügbar unter https://books.google.de/books?id=7iI3CwAAQBAJ.

Ellenberg, J. (2014): Preispolitik im Software-as-a-Service Markt: Deskriptive Analyse und Bewertung: Diplomica Verlag. Online verfügbar unter https://books.google.de/books?id=7GWQAwAAQBAJ.

Hammes, Dayne; Medero, Hiram; Mitchell, Harrison (2014): Comparison of NoSQL and SQL Databases in the Cloud. In:. Information Systems Conference. Macon, GA, USA, March 21st - 22nd 2014. Proceedings of the Southern Association for Information Systems Conference. Macon, GA, USA, S. 1–8.

Jatan, Nishtha; Puri, Sahil; Ahuja, Mehak; Kathuria, Ishita; Gosain, Dishant (2012): A Survey and Comparison of Relational and Non-Relational Database. In: *International Journal of Engineering Research & Technology (IJERT)* 1 (6), S. 1–5. Online verfügbar unter http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.678.9352&rep=rep1&type=pdf.

Leimeister, Jan Marco (2015): Einführung in die Wirtschaftsinformatik. 12. Aufl. Berlin [u.a.]: Springer (Springer-Lehrbuch).

Link, Jörg (2013): Customer Relationship Management. Erfolgreiche Kundenbeziehungen durch integrierte Informationssysteme. Berlin, Heidelberg: Springer Berlin Heidelberg.

Mahamed, Mohamed A.; Altrafi, Obay G.; Ismail, Mohammed O. (2014): Relational vs. NoSQL Databases: A Survey. In: *International Journal of Computer and Information Technology* 3 (3), S. 598–601.

Meier, Andreas; Kaufmann, Michael (2016): SQL- & NoSQL-Datenbanken. 8., überarb. u. erw. Aufl. 2016. Berlin, Heidelberg: Springer Berlin Heidelberg (eXamen.press).

Narang, R. (2011): Database Management Systems. o. O.: Prentice-Hall of India. Online verfügbar unter https://books.google.com/books?id=B-cJhhsh4NAC.

Padhy, Rabi Prasad; Patra, Manas Ranjan; Satapathy, Suresh Chandra (2011): RDBMS to NoSQL. Reviewing Some Next-Generation Non-Relational Database's. In: *International Journal of Advanced Engineering Sciences and Technologies* 11 (1), S. 15–30. Online verfügbar unter http://liacs.leidenuniv.nl/~stefanovtp/courses/StudentenSeminarium/Papers/DB/3.IJAEST-Vol-No-11-Issue-No-1-RDBMS-to-NoSQL-Reviewing-Some-Next-Generation-Non-Relational-Database's-015-030.pdf.

Schicker, Edwin (2014): Datenbanken und SQL. Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL. 4., überarb. Aufl. 2014. Wiesbaden: Springer Vieweg (SpringerLink : Bücher).

Sharma, Vatika; Dave, Meenu (2012): SQL and NoSQL Databases. In: *International Journal of Advanced Research in Computer Science and Software Engineering* 2 (8), S. 20–27. Online verfügbar unter http://s3.amazonaws.com/academia.edu.documents/33559632/V2I800154.pdf?AWSAccessKeyId=AKIAJ56TQJRTWSMTNPEA&Expires=1480620329&Signature=IgyAiXJM9JNlTsXPAjBJK3dQTUQ%3D&response-content-disposition=inline%3B%20filename%3DSQL\_and\_NoSQL\_Databases.pdf.

Wang, L.; Ranjan, R.; Chen, J.; Benatallah, B. (2011): Cloud Computing: Methodology, Systems, and Applications: CRC Press. Online verfügbar unter https://books.google.de/books?id=8UbNBQAAQBAJ.

Literaturverzeichnis DynamoDB

**Abramova, V. u. a. (2014):** Experimental Evaluation of NoSQL Databases, in: „International Journal of Database Management Systems (IJDMS)", 6. Jg., Nr. 3, S. 1–16.

**Amazon Web Services (2012):** Amazon DynamoDB, Developer Guide,

**Amazon Web Services (2016): ,** https://aws.amazon.com/de/products/?nc2=h\_ql\_ny\_livestream\_blu, Abruf: 26.12.2016.

**Barr, J. (Sept. 2010):** Host Your Web Site on the Cloud, Amazon Web Services Made Easy:Amazon EC2 Made Easy, Collingwood/LaVergne:SitePoint Pty; Ingram Publisher Services [distributor].

**Bhargavi/Sharma (2012):** AWS vs. MWA: A Review, ", 1. Jg., Nr. 9, S. 7.

**Deshpande, T. (2014):** Mastering DynamoDB, Master the intricacies of the NoSQL database DynamoDB to take advantage of its fast performance and seamless scalability / Tanmay Deshpande, Birmingham, UK:Packt Pub.

**Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swami Sivasubramanian, Peter Vosshall and Werner Vogels (Hrsg.)** **(2007):** Dynamo: Amazon's Highly Available Key-Value Store, 6. Aufl.

**Grassi, V. (Hrsg.) (2012):** Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures, New York, NY:

**Habeeb, M. (2011):** A developers guide to Amazon SimpleDB, Upper Saddle River, N.J.:Addison-Wesley.

**Katarina, G. u. a. (2013):** Data management in cloud environments: NoSQL and NewSQL data stores, ", S. 41.

**Kuznetsov/Poskonin (2014):** NoSQL Data Management Systems, ", 40. Jg., Nr. 6, S. 10.

**Linton, R. (2011):** Amazon Web Services, Migrating your .NET Enterprise Application / Rob Linton,

**Madhuri/Sowjanya (2016):** Microsoft Azure v/s Amazon AWS Cloud Services: A Comparative Study, ", 5. Jg., Nr. 3, S. 7.

**Matthew, F. (2016):** Microsoft Azure vs Amazon AWS public cloud comparison: Which cloud is best for the enterprise?, http://www.computerworlduk.com/it-vendors/microsoft-azure-vs-amazon-aws-public-cloud-comparison-which-cloud-is-best-for-enterprise-3624848/, Abruf: 2512.2016.

**Nacke, K. (2016):** D web development, Leverage the power of D and the vibe.d framework to develop web applications that are incredibly fast / Kai Nacke,

**Niranjanamurthy u. a. (2014):** The Research Study on DynamoDB – NoSQL Database Service, ", 3. Jg., Nr. 10, S. 12.

**Saakshi Narula u. a. ():** Cloud Comuputing Security: Amazon Web Services, ".

**Vyas, U./Kuppusamy, P. (2014):** DynamoDB applied design patterns, Apply efficient DynamoDB design patterns for high performance of applications, Birmingham, England:Packt Publishing.

**Werner Vogels (2012):** Amazon DynamoDB – a Fast and Scalable NoSQL Database Service Designed for Internet Scale Applications, http://www.allthingsdistributed.com/2012/01/amazon-dynamodb.html, Abruf: 25.12.2016.

**Wittig, M./Wittig, A. (2016):** Amazon Web Services in action, Shelter Island NY:Manning.

**Wolfgang Sommergut (2015):** Was Sie über die Cloud wissen müssen, http://www.computerwoche.de/a/was-sie-ueber-die-cloud-wissen-muessen,2504589,2, Abruf: 15.12.2016.

1. Schicker 2014, S. 3 [↑](#footnote-ref-2)
2. Vgl. Narang 2011, S. 1 [↑](#footnote-ref-3)
3. Leimeister 2015, S. 86 [↑](#footnote-ref-4)
4. Vgl. Link 2013, S. 184 [↑](#footnote-ref-5)
5. Vgl. Ellenberg 2014, S. 2 [↑](#footnote-ref-6)
6. Vgl. Wang et al. 2011, S. 14 [↑](#footnote-ref-7)
7. Vgl. Wang et al. 2011, S. 14 [↑](#footnote-ref-8)
8. Vgl. Wang et al. 2011, S. 15 [↑](#footnote-ref-9)
9. Vgl. Antonopoulos und Gillam 2010, S. 53 [↑](#footnote-ref-10)
10. Vgl. Schicker 2014, S. 18 [↑](#footnote-ref-11)
11. Vgl. Sharma und Dave 2012, S. 21 [↑](#footnote-ref-12)
12. Vgl. Mahamed et al. 2014, S. 598 [↑](#footnote-ref-13)
13. Vgl. Schicker 2014, S. 18 [↑](#footnote-ref-14)
14. Vgl. Mahamed et al. 2014, S. 598 [↑](#footnote-ref-15)
15. Vgl. Mahamed et al. 2014, S. 598 [↑](#footnote-ref-16)
16. Vgl. Mahamed et al. 2014, S. 598 [↑](#footnote-ref-17)
17. Vgl. Sharma und Dave 2012, S. 21 [↑](#footnote-ref-18)
18. Vgl. Mahamed et al. 2014, S. 598f. [↑](#footnote-ref-19)
19. Vgl. Sharma, Dave 2012, S. 21, vgl. dazu auch Mahamed et. al. 2014, S.598f. [↑](#footnote-ref-20)
20. Vgl. Mahamed et al. 2014, S. 598–599, vgl. dazu auch Sharma, Dave 2012, S.21 [↑](#footnote-ref-21)
21. Vgl. Mahamed et al. 2014, S. 599 [↑](#footnote-ref-22)
22. Vgl. Mahamed et al. 2014, S. 599 [↑](#footnote-ref-23)
23. Vgl. auch im Folgenden Sharma und Dave 2012, S. 21–22 [↑](#footnote-ref-24)
24. Entnommen aus: Edward und Sabharwal 2015, S. 16 [↑](#footnote-ref-25)
25. Vgl. Jatan et al. 2012, S. 2 [↑](#footnote-ref-26)
26. Vgl. Abramova et al. 2014, S. 1 [↑](#footnote-ref-27)
27. Vgl. Meier und Kaufmann 2016, S. 18 [↑](#footnote-ref-28)
28. Vgl. Mahamed et al. 2014, S. 598 [↑](#footnote-ref-29)
29. Vgl. Abramova et al. 2014, S. 1 [↑](#footnote-ref-30)
30. Vgl. Mahamed et al. 2014, S. 598 [↑](#footnote-ref-31)
31. Vgl. Mahamed et al. 2014, S. 598 [↑](#footnote-ref-32)
32. Vgl. Meier und Kaufmann 2016, S. 18 [↑](#footnote-ref-33)
33. Vgl. Abramova et al. 2014, S. 1–2 [↑](#footnote-ref-34)
34. Vgl. Hammes et al. 2014, S. 1 [↑](#footnote-ref-35)
35. Vgl. Abramova et al. 2014, S. 1–2 [↑](#footnote-ref-36)
36. Vgl. Jatan et al. 2012, S. 3 [↑](#footnote-ref-37)
37. Vgl. Jatan et al. 2012, S. 3 [↑](#footnote-ref-38)
38. Vgl. Mahamed et al. 2014, S. 599 [↑](#footnote-ref-39)
39. Vgl. Abramova et al. 2014, S. 1–2 [↑](#footnote-ref-40)
40. Vgl. Hammes et al. 2014, S. 1 [↑](#footnote-ref-41)
41. Vgl. Hammes et al. 2014, S. 1 [↑](#footnote-ref-42)
42. Vgl. Meier und Kaufmann 2016, S. 20 [↑](#footnote-ref-43)
43. Vgl. Abramova et al. 2014, S. 3–4 [↑](#footnote-ref-44)
44. Vgl. Padhy et al. 2011, S. 16 [↑](#footnote-ref-45)
45. Vgl. Abramova et al. 2014, S. 3–4 [↑](#footnote-ref-46)
46. Vgl. Padhy et al. 2011, S. 16 [↑](#footnote-ref-47)
47. Vgl. Abramova et al. 2014, S. 3–4 [↑](#footnote-ref-48)
48. Vgl. Abramova et al. 2014, S. 3–4 [↑](#footnote-ref-49)
49. Vgl. Padhy et al. 2011, S. 16 [↑](#footnote-ref-50)
50. Vgl. Abramova et al. 2014, S. 3–4 [↑](#footnote-ref-51)
51. Vgl. Abramova et al. 2014, S. 3–4 [↑](#footnote-ref-52)
52. Vgl. Abramova et al. 2014, S. 3–4 [↑](#footnote-ref-53)
53. Vgl. Hammes et al. 2014, S. 3 [↑](#footnote-ref-54)
54. Vgl. Jatan et al. 2012, S. 4–5 [↑](#footnote-ref-55)
55. Vgl. Hammes et al. 2014, S. 3–4 [↑](#footnote-ref-56)
56. Vgl. Hammes et al. 2014, S. 3–4 [↑](#footnote-ref-57)
57. Vgl. Hammes et al. 2014, S. 3–4 [↑](#footnote-ref-58)
58. Vgl. Hammes et al. 2014, S. 3–4 [↑](#footnote-ref-59)
59. Mahamed et al. 2014, S. 600 [↑](#footnote-ref-60)
60. Sommergut 2015 [↑](#footnote-ref-61)
61. # Vgl. Linton 2011, S.8a

    [↑](#footnote-ref-62)
62. Finnegan 2016 [↑](#footnote-ref-63)
63. Entommen aus: Statista 2015 [↑](#footnote-ref-64)
64. Vgl. Linton 2011, S.8b [↑](#footnote-ref-65)
65. Vgl. Wittig/Wittig 2016, S.5 [↑](#footnote-ref-66)
66. Vgl. Wittig/Wittig 2016, S.6a [↑](#footnote-ref-67)
67. Vgl. Barr 2010, S.99 [↑](#footnote-ref-68)
68. Vgl. Wittig 2016, S.6b [↑](#footnote-ref-69)
69. Vgl. Barr 2010, S. 59 [↑](#footnote-ref-70)
70. Vgl. Wittig/Wittig 2016, S.10 [↑](#footnote-ref-71)
71. Vgl Narula/Jain/Prachi 2015, S.3 [↑](#footnote-ref-72)
72. Vgl. Wittig/Wittig 2016, S.12 [↑](#footnote-ref-73)
73. Vgl. Bhargavi/Sharma 2012, S.6 [↑](#footnote-ref-74)
74. Vgl. Madhuri/Sowjanya 2016, S.6 [↑](#footnote-ref-75)
75. Habeeb 2011, S.1 [↑](#footnote-ref-76)
76. Deshpande 2014, S.9a [↑](#footnote-ref-77)
77. Vgl. Deshpande 2014, S.9b [↑](#footnote-ref-78)
78. Vgl. Kuznetsov/Poskonin 2014, S.4 [↑](#footnote-ref-79)
79. Vogels 2012 [↑](#footnote-ref-80)
80. DeCandia/Hastorun/Jampani/Kakulapati/Lakshman/Pilchin/Sivasubramanian/Vosshall/Vogels 2007, S.2 [↑](#footnote-ref-81)
81. Vgl.Amazon Web Services 2016 [↑](#footnote-ref-82)
82. Vgl.Amazon Web Services 2016 [↑](#footnote-ref-83)
83. Vgl. Kuznetsov/Poskonin 2014, S.2a [↑](#footnote-ref-84)
84. DeCandia/Hastorun/Jampani/Kakulapati/Lakshman/Pilchin/Sivasubramanian/Vosshall/Vogels 2007, S.4 [↑](#footnote-ref-85)
85. Deshpande 2014, S.77a [↑](#footnote-ref-86)
86. Deshpande 2014, S.28 [↑](#footnote-ref-87)
87. Deshpande 2014, S.77b [↑](#footnote-ref-88)
88. Vgl. Kuznetsov/Poskonin 2014, S.2b [↑](#footnote-ref-89)
89. Vgl. Niranjanamurthy/Archana/Niveditha/Abdul Jafar/Shravan 2014, S.1 [↑](#footnote-ref-90)
90. Vgl. Kuznetsov/Poskonin 2014, S.2c [↑](#footnote-ref-91)
91. Vgl. Volgels 2012 [↑](#footnote-ref-92)
92. Vgl. DeCandia/Hastorun/Jampani/Kakulapati/Lakshman/Pilchin/Sivasubramanian/Vosshall/Vogels 2007, S.7 [↑](#footnote-ref-93)
93. Deshpande 2014, S.81 [↑](#footnote-ref-94)
94. Entommen aus: Deshpande 2014, S.85a [↑](#footnote-ref-95)
95. Vgl. Grassi 2012, p.2 [↑](#footnote-ref-96)
96. Vgl. Deshpande 2014, S.85b [↑](#footnote-ref-97)
97. Vgl. Deshpande 2014, S.10a [↑](#footnote-ref-98)
98. Entnommen aus: Vyas/Kuppusamy 2014, p.8 [↑](#footnote-ref-99)
99. Vgl. Deshpande 2014, S.10b [↑](#footnote-ref-100)
100. Vgl. Deshpande 2014, S.10c [↑](#footnote-ref-101)
101. Vgl. Amazon Web Services 2012, S.13 [↑](#footnote-ref-102)
102. Vgl. Amazon DynamoDB 2012, S.13a [↑](#footnote-ref-103)
103. Vgl. Amazon DynamoDB 2012, S.13b [↑](#footnote-ref-104)
104. Vgl. Deshpande 2014, S.28ff. [↑](#footnote-ref-105)
105. Vgl. Vyas/Kuppusamy 2014, S.16ff. [↑](#footnote-ref-106)
106. Vgl. Niranjanamurthy/Archana/Niveditha/Abdul Jafar/Shravan 2015, S.3 [↑](#footnote-ref-107)
107. Vgl. Volgels 2012 [↑](#footnote-ref-108)
108. Vgl. Deshpande 2014, S.14 [↑](#footnote-ref-109)
109. Vgl. Niranjanamurthy/Archana/Niveditha/Abdul Jafar/Shravan 2015, S.4 [↑](#footnote-ref-110)
110. Vgl. Vyas/Kuppusamy 2014, S.19 [↑](#footnote-ref-111)
111. Vgl. Deshpande 2014, S.45 [↑](#footnote-ref-112)
112. Abramova/Bernardino/Furtado 2014, S.4 [↑](#footnote-ref-113)
113. Vgl. Deshpande 2014, S.34a [↑](#footnote-ref-114)
114. Vgl. Deshpande 2014, S.34b [↑](#footnote-ref-115)
115. Vgl.Deshpande 2014, p.35 [↑](#footnote-ref-116)
116. Vgl. Amazon DynamoDB 2012, S.545a [↑](#footnote-ref-117)
117. Vgl. Amazon DynamoDB 2012, S.21a [↑](#footnote-ref-118)
118. Vgl. Amazon DynamoDB 2012, S.21b [↑](#footnote-ref-119)
119. Vgl. Amazon DynamoDB 2012, S.22 [↑](#footnote-ref-120)
120. Vgl. Nacke 2016, p.86 [↑](#footnote-ref-121)
121. Vgl.  Grolinger/A. Higashino/Tiwari/A. CapretzM 2013, p.15 [↑](#footnote-ref-122)
122. Vgl. Vyas and Kuppusamy 2014, S.176 [↑](#footnote-ref-123)
123. Vgl. Grolinger/Higashino/Tiwari/CapretzM 2013, p.17 [↑](#footnote-ref-124)
124. Mit Änderungen entnommen aus: Vyas/Kuppusamy 2014, S.176 [↑](#footnote-ref-125)