

Die Mongo DB

- Name
- Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Konkrete Problemstellung
- Individuelle Zielsetzung

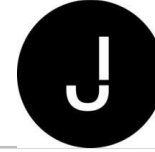
- Eigener Rechner
 - Download der Mongo-Community Edition und Installation jeweils für benutztes Betriebssystem
 - <https://www.mongodb.com/try/download/community>
 - Alternativ
 - Docker-Image
 - https://hub.docker.com/_/mongo



8:00 - 16:00

Mittagspause: 11:45 - 13:00

Pausen: 9:45 - 10:15, 14:15:14:30



Ausgangssituation

- Name der Datenbank
 - MongoDB -> “Humengous” = gigantisch/enorm
 - “Menges-DB”

- 2003
 - “Big Data”
 - Besser: “Big & Fast Data”
- Relationale Datenbanksysteme hatten im Endeffekt Skalierungsprobleme
 - Hochskalieren erfolgt vertikal, mehr CPU, mehr RAM, mehr Storage
- NoSql erfindet neue Datenbank-Typen
 - Key-Value
 - Dokumenten-orientiert -> MongoDB
 - Graphen-orientiert
 - Spalten-orientiert

- Der Name “NoSql”
 - Besser “NoRelational”
 - Präziser: “No” = “Not Only”
- Die Umsetzung eines abstrakten EntityModells erfolgt nicht zwangsläufig durch ein relationales Modell
- NoSQL-Kategorien
 - Key-Value-Modell
 - select value from store where key='key1'
 - Dokumenten-orientierte Modellierung -> Details später
 - Graphen-Modell
 - Tausende von Joins sind ohne relevanten Performance-Verlust zu realisieren
 - Spalten-orientierte Modellierung
 - basiert auf einem Objekt-Modell
 - Relationales Modell
 - Fremdschlüssel und Verknüpfungstabellen

- Jedes Dokument hat eine eindeutige Dokumenten-ID
 - Weltweite Eindeutigkeit ist zu garantieren
 - URI: Zugriffsprotokoll://host/collection/id
- Beziehungen zwischen Dokumenten erfolgen über eine Verlinkung
 - = Angabe der URI des “anderen” Dokuments
- Dokumente haben ein Schema, eine Struktur
 - “Schema on Read”
 - Eine Abfrage legt das benötigte Schema fest und bekommt dann auch nur die Dokumente, die dem Schema genügen
 - “Schema on Write”
 - Validierung beim Schreib-Vorgang, eine Datenschenke akzeptiert nur Daten, die einer vorgegebenen Struktur entsprechen

- Breite Produktpalette
 - Open Source-Produkte, z.B. Couchbase
- Kommerzielles Produkt: MongoDB
 - Community-Edition
 - Frei einsetzbar, alle kein offizieller Support
 - Bei uns im Training
 - Lizenzpflichtige Version
 - Support
 - Tooling
 - ...
 - MongoDB Atlas
 - Cloud-basierte Lösung

Training

VKB

- Rechner mit Docker-Runtime
 - Integrata-Cegos stellt hierfür Ubuntu-basierte Deskmate-Maschinen bereit

Remote Rechner der Integrata-Cegos

Deskmate-User	Deskmate Passwort	Ubuntu User-ID	Passwort
tn28.raum01@integrata-cegos.de	4023_tn28	sl01	sl01
tn29.raum01@integrata-cegos.de	4023_tn29	sl01	sl01
tn30.raum01@integrata-cegos.de	4023_tn30	sl01	sl01
tn31.raum01@integrata-cegos.de	4023_tn31	sl01	sl01
tn32.raum01@integrata-cegos.de	4023_tn32	sl01	sl01
tn33.raum01@integrata-cegos.de	4023_tn33	sl01	sl01
Deskmate-Link: https://integrata-cegos.deskmate.me/			

- BA Brännert Andreas (Gast)
Gast der Besprechung
- DP Daniel Petermeier (Gast)
Gast der Besprechung
- SS Sebastian Schumacher (Ga...
Gast der Besprechung
- TA Thomas Adamek (Gast)
Gast der Besprechung

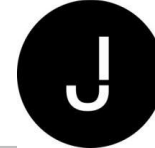
Support:

Zentrale IT

Telefon:

+49 711 62010 355

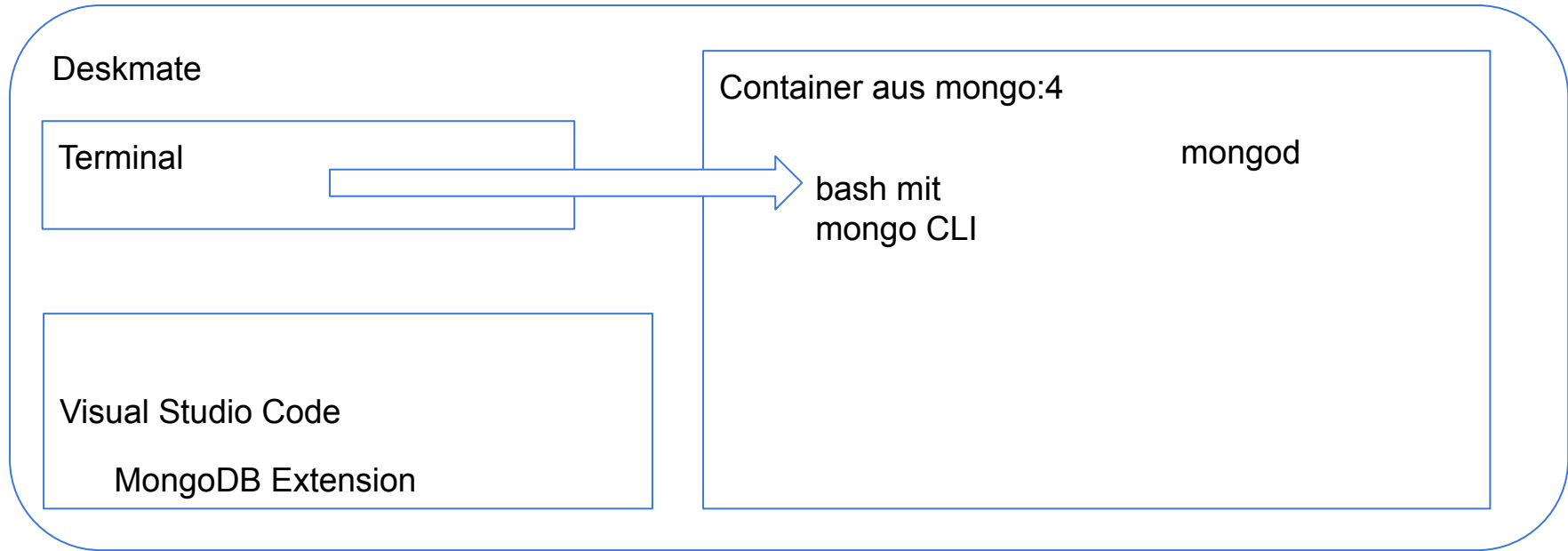
ZentraleIT@integrata-cegos.de

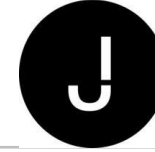


- BITTE KEINE AKTUALISIERUNGEN AKZEPTIEREN!



- `docker create --name mongodb -p 27017:27017 mongo:4`
- `docker exec -it mongodb /bin/bash`
 - `mongo`
 - Interaktive Shell
- Dazu Visual Studio Code mit Mongos Extension





MongoDB First Contact

- **Datenbank-Server mit Server-Socket auf 27017**
- **Befehlssatz ist JavaScript**
 - Das Dokumenten-Format in MongoDB ist JSON
- **Andere APIs**
 - REST-Schnittstelle ist vorhanden, allerdings eigentlich nur in der Lizenz-Version
 - Besser: GraphQL
- **MongoDB-Treiber für andere Programmiersprachen sind vorhanden**
 - Java
 - C#
 - Python

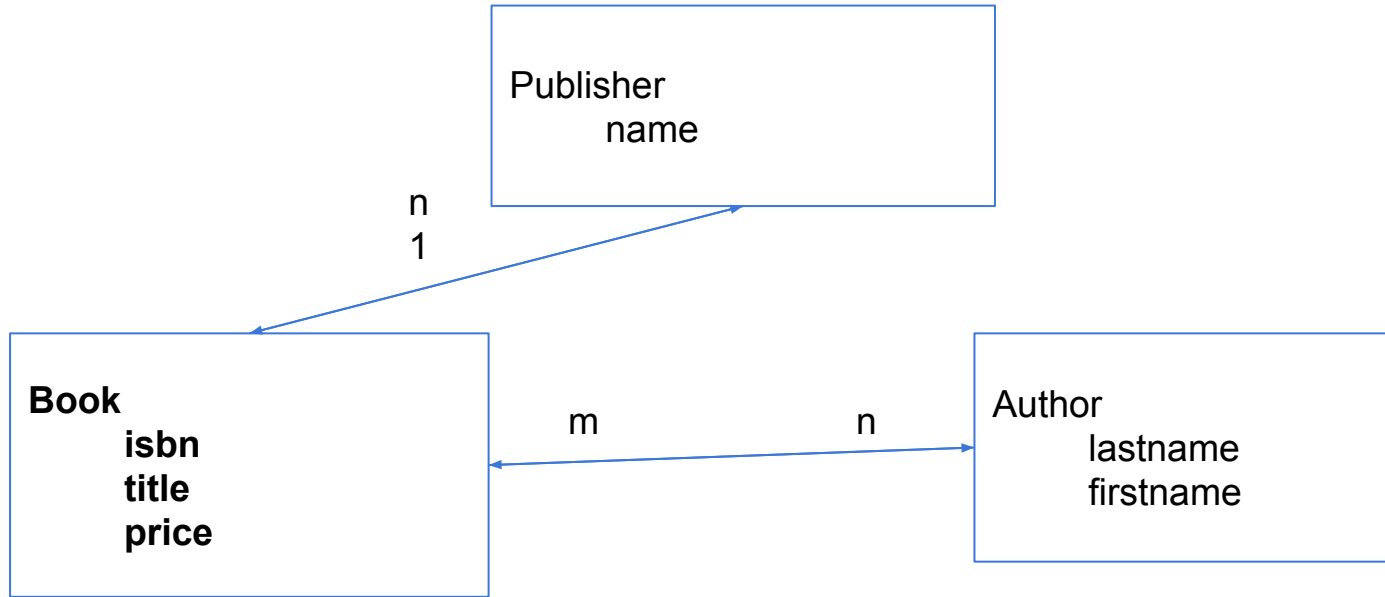
- XML ist ein **akademisch** sehr geeignetes Datenformat für Dokumente
 - Schema
 - Link-Element
- JSON-Format
 - de facto Standard im Internet
 - Für Web-Anwendungen ist damit keine Transformation nötig
 - Es fehlt
 - Schema-Beschreibung
 - Standardisierte Angabe für Links
 - MongoDB nutzt JSON in einer Erweiterung
 - BSON-Spezifikation
 - Ermöglicht Schemata und Verlinkungen

- **CRUD-Operationen**
 - **Create**
 - `insertOne(object)`
 - **Read**
 - `find`
 - Mit Abfrage-Kriterien
 - **Update**
 - `saveOrUpdate`, Details hierzu später
 - **Delete**
 - `deleteMany`
 - `deleteOne`

- Parameter ist das zu erzeugende Object
- MongoDB erzeugt intern eine relativ eindeutige Id
 - Diese ObjectID ist nur im Zusammenspiel mit host/collection weltweit eindeutig

- Ansatz 1: Collections werden wie Tabellen verwendet
 - In den allermeisten Fällen total falsch
 - Dies führt zu einem schlechten Dokumenten-Modell mit viel zu viel Joins zwischen verschiedenen Dokumenten
- Ansatz 2:
 - Es genügt pro Datenbank-Instanz eine einzige Collection
 - Das ist prinzipiell völlig in Ordnung
 - Schema on Read ist hocheffizient implementiert
- In der Realität werden Collections aber noch zusätzlich benutzt
 - Halten von Konfigurationseinstellungen
 - Berechtigungskonzept
 - Übersichtlichkeit

Das Entity-Modell für unser Training



Step 1

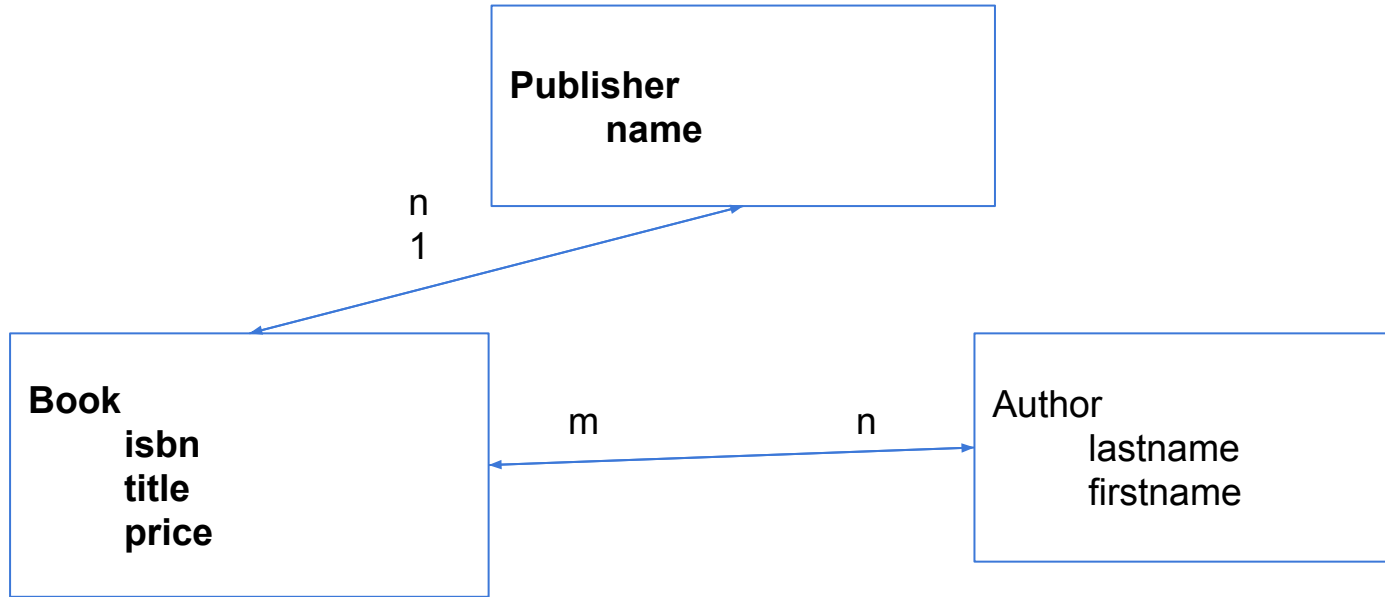


- Schreiben Sie eine Funktion, die ein paar Test-Bücher in die Datenbank legt
- Ablauf
 - drop der Collection publishing
 - create publishing
 - Erzeugen der Test-Daten
 - CHECK: DB, Collection und Daten sind vorhanden

- Such-Operationen
 - find()
 - find(criteriaObject)
 - criteriaObject: Ein JSON-Objekt bzw. in unserer Umgebung ein JavaScript-Objekt
- Hinweis
 - `_id` ist auch ein Kriterium, allerdings: ObjectId("hash")
- Exkurs: zu den ObjectIds
 - Bestandteile
 - Timestamp
 - 5 Zeichen sind Prozess-abhängig
 - Interner Counter
-

- SQL?
 - Passt nicht auf Dokumenten-basierte Abfragen
- N1QL
 - gesprochen: “Nickel”
 - SQL-Erweiterung als Standard für eine Dokumenten-basierte Abfragesprache
 - MongoDB unterstützt N1QL nicht
- Statt dessen
 - etwas proprietäres

Das Entity-Modell für unser Training



Step 2

- Publisher zusätzlich mit einem Address-Dokument {city: "", street: ""}
 - {"address.city": "Berlin"}
- Book zusätzlich mit einer Kurzbeschreibung (description)
 - {\$text: {\$search: "MongoDb"}}
- zusätzlich: auch die Projektion kann ein \$elemMatch enthalten
 - Im Endeffekt ein Subquery auf die Ergebnis-Liste

- Embedded Documents
 - Besteht also aus mehreren Dokumenten, die sich allesamt in einem Haupt-Dokument befinden
 - Als Aggregat sind alle Sub-Dokumente im Lebenszyklus an das Hauptdokument gebunden
 - Nur das Haupt-Dokument hat eine Object-ID
 - “Sawitzki”: Atomar Document
- Verlinkungen auf andere Dokumente
 - Die klassische Dokumenten-orientierte Modellierung

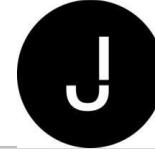
- Primärfokus
 - Embedded Documents
 - Verwaltung solcher Dokumente ist äußerst effizient möglich
 - Transaktionssicherheit auf Embedded Documents ist trivial
- Bei verlinkten Dokumenten erfolgt das Joinen
 - auf Client-Seite (der ursprüngliche Ansatz)
 - auf Server-Seite (mittlerweile unterstützt, aber aus Performance-Sicht nicht unbedenklich)
 - Transaktionssicherheit ist entweder gar nicht gewährleistet (Client-Joins) oder sehr aufwändig

- Ursprüngliche Abfrage-Sprache bezieht sich auf Embedded Documents
- Später
 - Map-Reduce
 - Aggregate-Pipeline

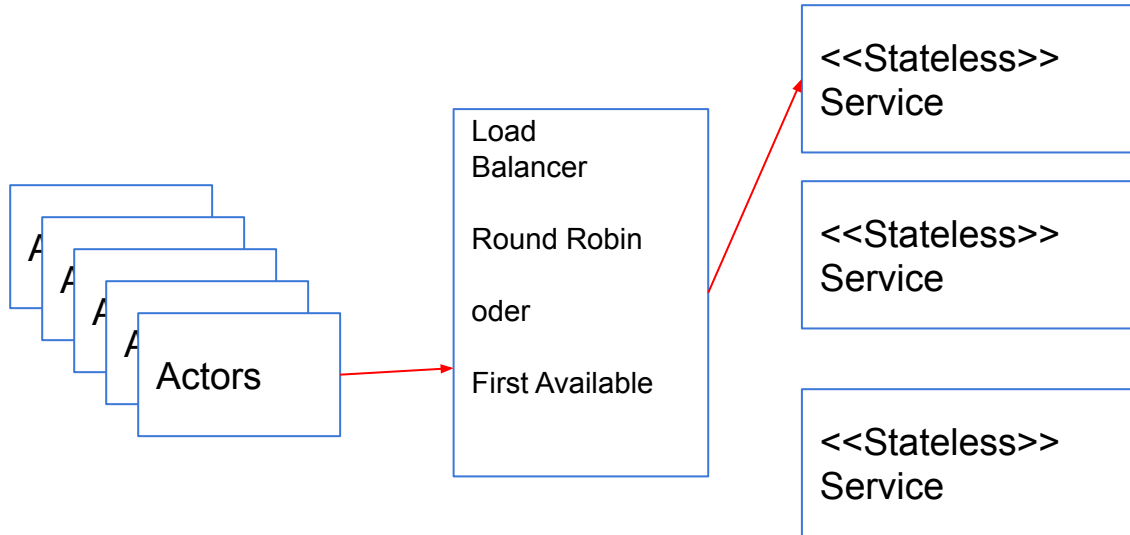


- Embedded Publisher-Books-Dokumente
- Klassische Formulierung von Kriterien
- Programmatisch ist ein Embedded Document nichts anderes als ein Objekt-Geflecht
 - Als Literal {name: "Springer", books: [{isbn: "ISBN1", ...}]}
 - let publisher = {name: "Springer", books: []}
 - publisher.books[0] = book1
- ToDo
 - 2 Publisher ("Springer", "Addison")
 - Springer bekommt die 5 Bücher, Addison ein Demo-Buch

- Update-Operationen
- Aggregate-Framework
- Von Embedded zu Verlinked
- Übersicht der Arbeitsweise eines Mongos-Clusters
- Technik
 - Indizes
 - Daten-Konsistenz
 - Transaktionen



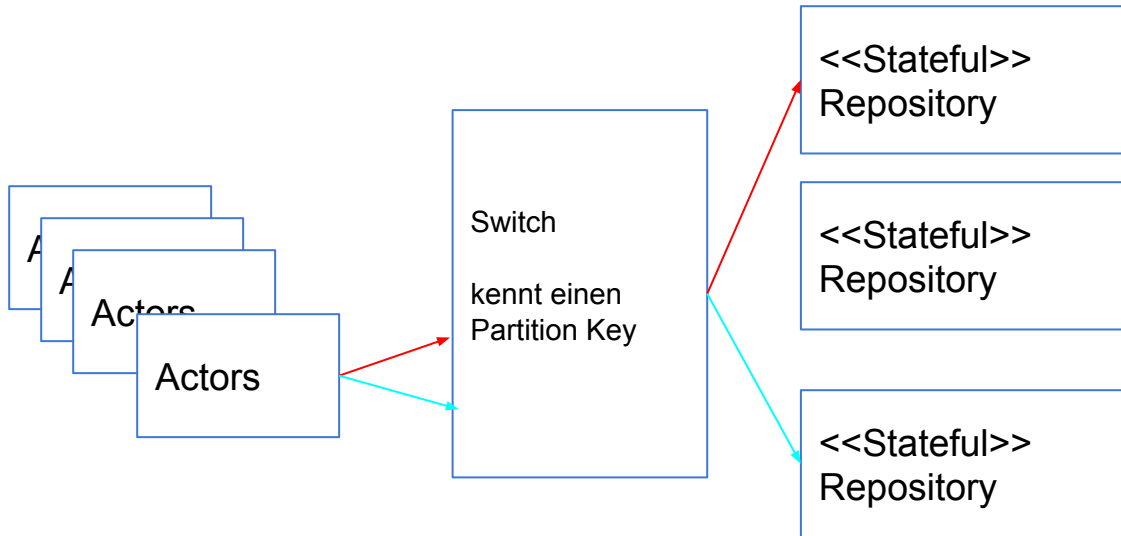
Architektur der MongoDB



Dynamisch skalierbare
Systeme

Orchestrierung und Überwachung

Partitionierung des Datenbestands

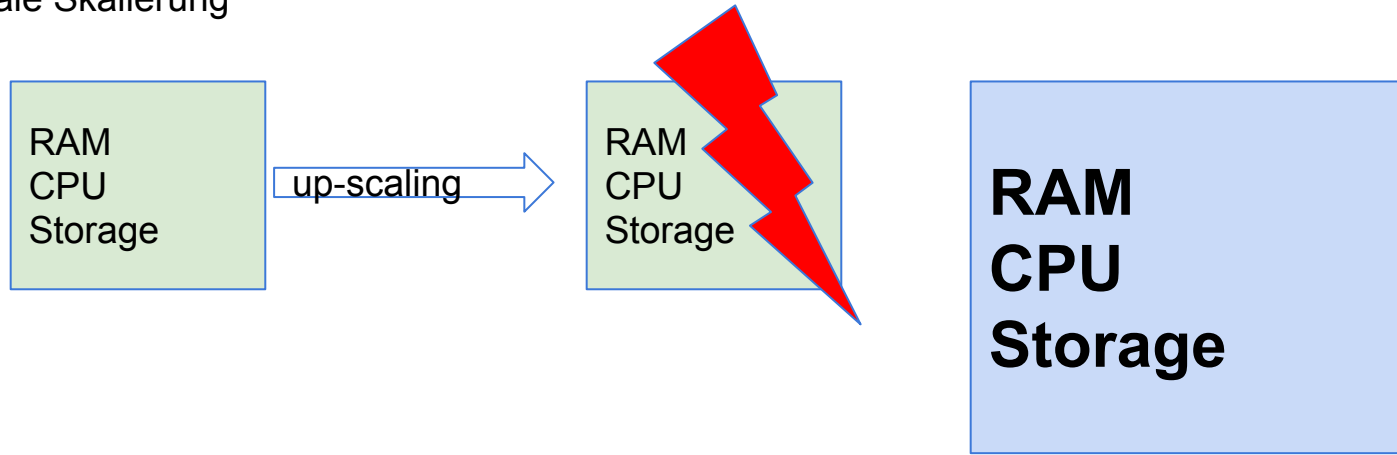


Up & Down Scaling rein administrativ

Probleme

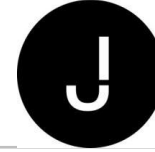
- Ein Partiton Key muss gefunden werden
- Assoziationen zwischen Partitionen sind nicht möglich

Vertikale Skalierung



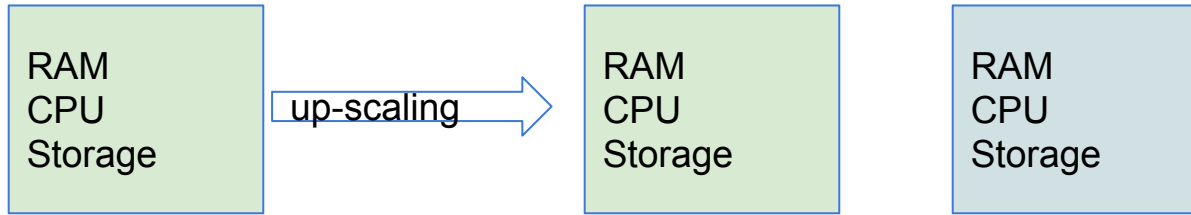
Probleme

- Wartungsfenster durch Daten-Migration
- Grenzen der Skalierung durch
 - Kostengründe
 - Technisch



- Relationale Modelle skalieren traditionell vertikal
 - Relationen zwischen einzelnen Knoten sind aufwändig

Horizontale Skalierung

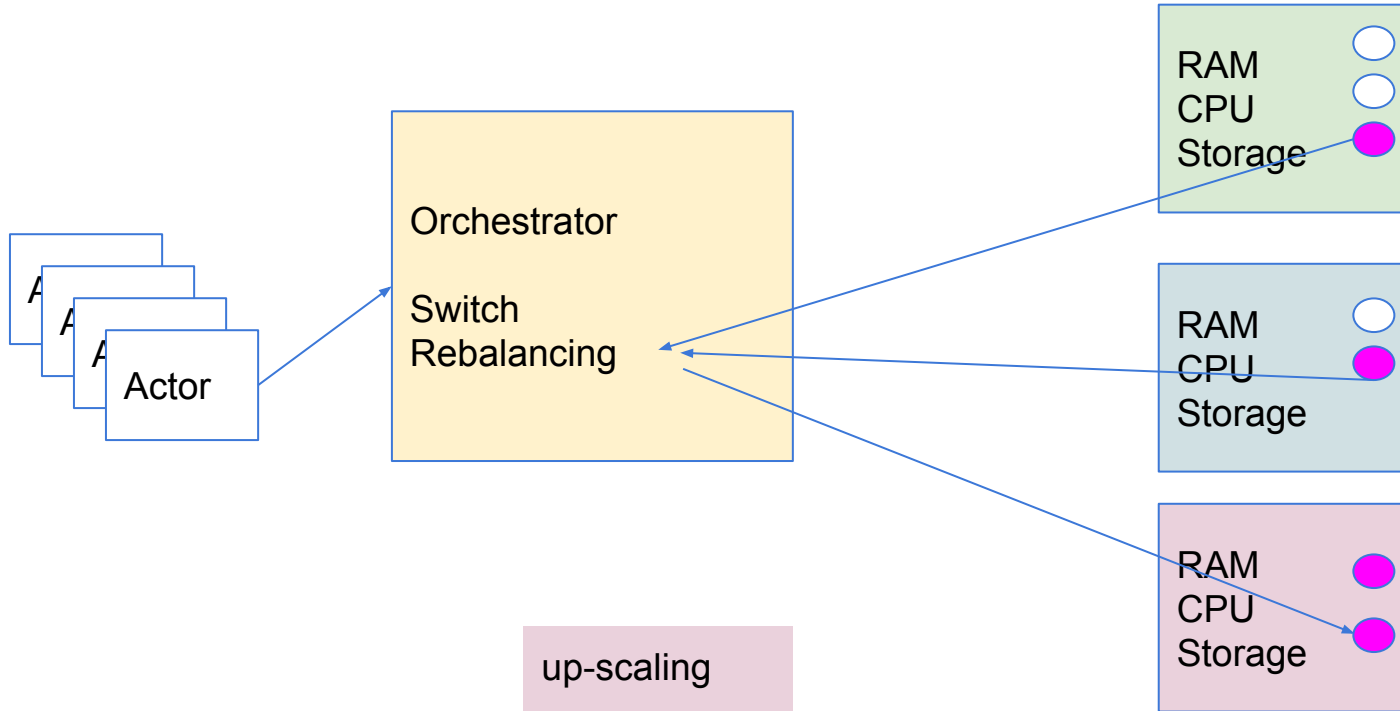


Probleme

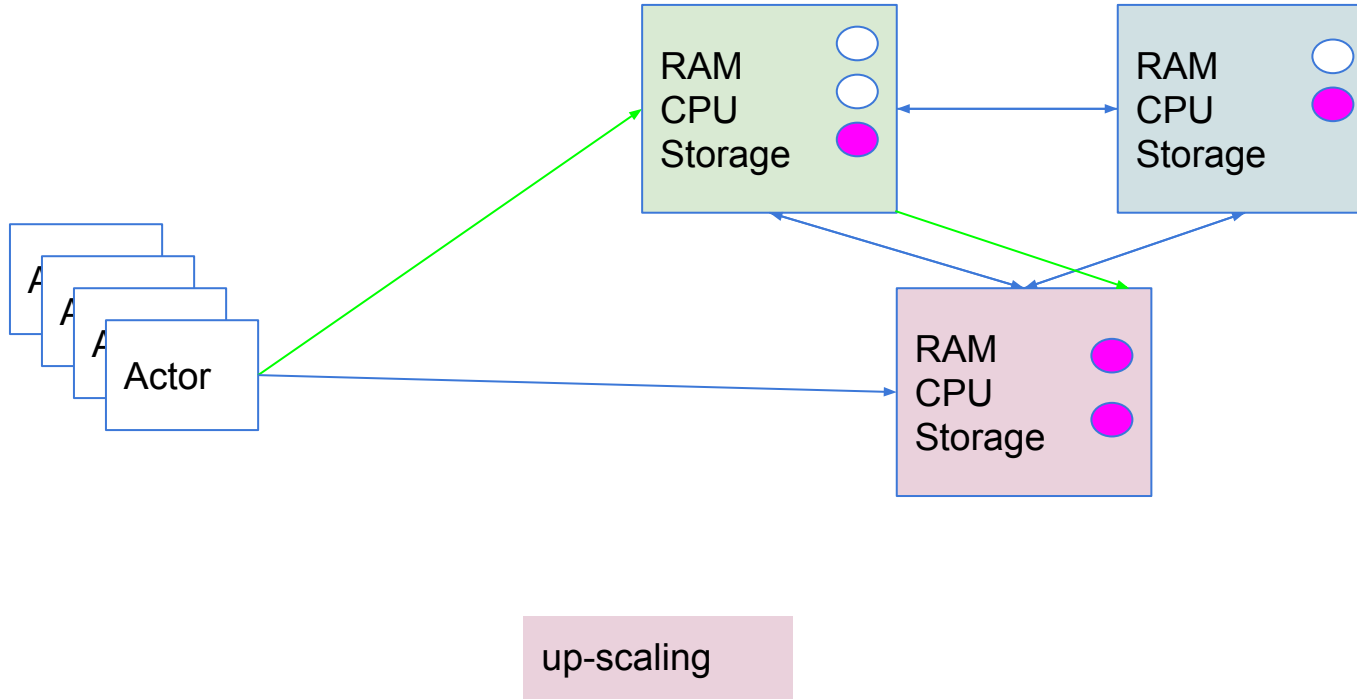
- Die Skalierung kann so nicht funktionieren!

Horizontale Skalierung mit State

Orchestrator



Ring Cluster



- Eine Kommunikation zwischen den Knoten bzw. zwischen Knoten und Orchestrator ist notwendig
 - Netzwerk
- Interne Netzwerk-Kommunikation innerhalb der Datenbank ist relativ zeitaufwändig
- Auch das interne Netzwerk kann ausfallen
 - Es ist ein “Partitionsfehler” aufgetreten

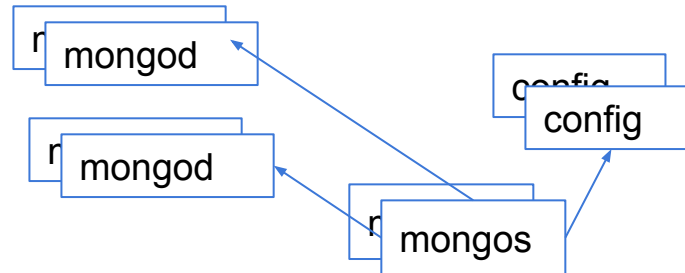
- Notwendig ist ein eindeutiger Schlüssel zur Identifikation eines Datenbestands notwendig
 - Dokumenten-ID
- Ein Datensatz sollte für Auswertungen alle Daten atomar enthalten
 - Embedded Documents
 - Daten-Redundanzen sind damit unvermeidlich
- Das Auflösen von Verlinkungen verlangt immer Netzwerk-Kommunikation
- Design-Ansatz
 - Query First



- Einstiegspunkte
 - Publisher
 - ObjectID, Books sind Embedded Document
 - suche publisher nach Name + weitere Selektionen an Hand der embedded documents
 - Book
 - ObjectID

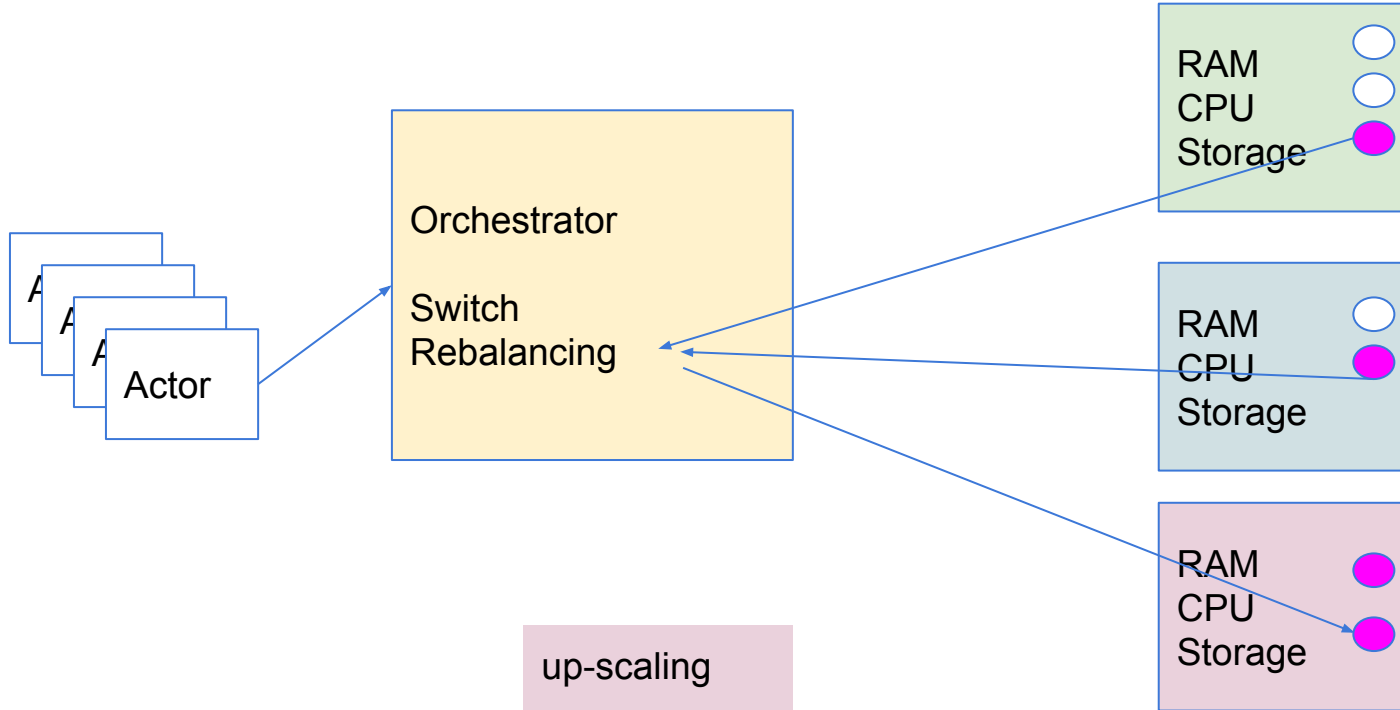
- mongod
 - der eigentliche Datenbank-Prozess
 - master-slave zur Ausfallsicherheit
- mongos
 - der Shard-Controller
- Config-Server
 - master-slave zur Ausfallsicherheit

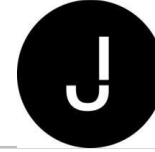
Minimal-Mongos-Cluster
besteht aus 7 Prozessen



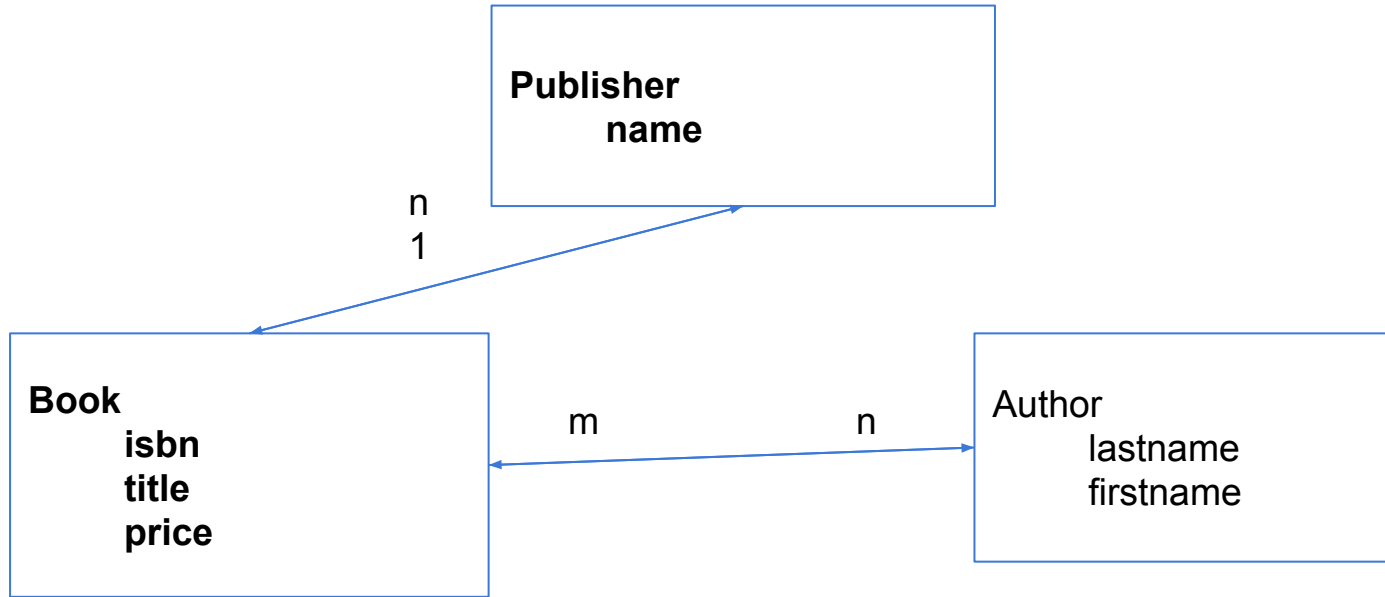
Horizontale Skalierung mit MongoDB

Orchestrator





Query First: Ein Beispiel



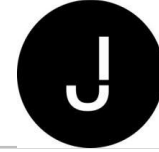
- Anhand des Namens soll ein Publisher gefunden werden
 - Rückgabe: Die Liste der Bücher ISBNs
- Anhand des Namens soll die Adresse eines Publishers gefunden werden
- Es sollen alle Bücher gefunden werden, die in einem bestimmten Preisbereich liegen
 - Book
- Suche Bücher über nach ihrer Beschreibung
 - Buch-Titel

Publisher

- name
- books
 - isbn
- address
 - city
 - street

Book

- isbn
- title
- pages
- price
- description



- Welcher Verleger verlegen “billige” Bücher?
 - Name des Publishers

Publisher

- name
- books
 - isbn
- address
 - city
 - street

Book

- isbn
- title
- pages
- price
- description
- **publisherName**

- Ausgehend von der gestrigen Modellierung ändern Sie das Dokumenten-Modell an Hand der identifizierten Queries
- Einführen der Autoren-Information
 - lastname, firstname
 - Abfrage: Wie heißen die Autoren eines Buches?
 - “Firstname Lastname”
 - Welche Bücher hat ein Autor geschrieben
 - Titel

Publisher

- name
- books
 - isbn
- address
 - city
 - street

Book

- isbn
- title
- pages
- price
- description
- publisherName
- authorNames

Authors

- lastname
- firstname
- isbns

Publisher

- name <<query>>
- books
 - isbn
- address
 - city
 - street

Book

- isbn <<query>>
- title <<query>>
- pages
- price
- description
- publisherName
- authorNames

Authors

- lastname
- firstname
- isbns

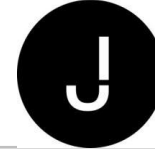
- `books = db.publishing.find()`
 - `books` ist nicht bereits die Treffermenge, sondern ein iterierbarer Proxy
- `books.explain("executionStats")`
- Daraus ergibt sich die Anlage eines neuen Index
 - `db.publishing.createIndex({isbn: 1})`
 - `db.publishing.createIndex({address.city: 1})`
 - `db.publishing.createIndex({description: "text"})`
- Options-Objekt
 - `unique: true|false`
 - `expireAfterSeconds: 60`
 - `background: true|false`

- Voraussetzung
 - Cluster mit mehreren mongod-Prozessen
- Ab nun kann jede Collection einen speziellen Index, den Shard Key definieren
 - Identifikation: Auf welchem mongod-Prozess liegt ein Dokument, das zu diesem Shard gehört
 - Optimierung von Abfragen: Enthält die Abfrage-Bedingung als Kriterium den Shared Key, sind alle weiteren Abfragen auch im Cluster sehr effizient
-

- <https://www.mongodb.com/docs/manual/core/schema-validation/>
-



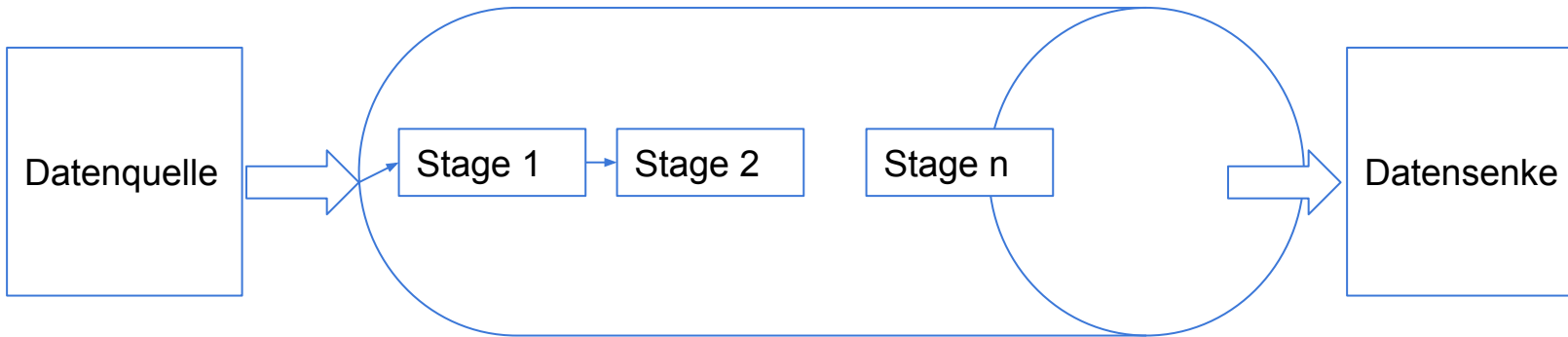
- Auftrennung in die Collections publishers, books, authors
- Fügen Sie diesen Collections noch einen Schema-Validator hinzu

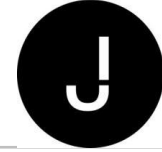


Join

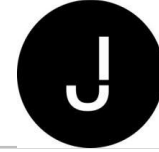
- Aufbau der Verlinkung erfolgt durch die Verwendung einer Dokumenten-ID
 - Theoretisch: Eine beliebige Endpoint-Adresse
 - MongoDB: Endpoint ist eine interne Adresse innerhalb der Datenbank
- Damit wird für die interne Verlinkung eine Kombination aus ObjectId und Collection benutzt
 - {oids: [oid1, oid2, oid3], collection: "books"}
 - diese wird einem Attribut zugewiesen
 - books: {oids: [oid1, oid2, oid3], collection: "books"}

Exkurs: Datenverarbeitung in einer Pipeline





```
output = input.pipeline(  
[  
  stage1,  
  stage2,  
  stage3  
])
```



```
let output = db.books.aggregate(  
[  
  stage1,  
  stage2,  
  stage3  
])
```

Eine simple Abfrage mit Query und Projection



```
db.books.aggregate(  
[  
{$match: {isbn: "ISBN0"}},  
{$project: {_id: 0, title: 1}}  
]  
)
```


Eine Abfrage mit Query, Join und Projection



```
db.publishers.aggregate(  
  [  
    {$match: {name: "Springer"}},  
    {$lookup: {  
      from: "books" // "books.collection"  
      localField: "books.oids",  
      foreignField: "_id",  
      as: "book"  
    }}  
  ],  
  {$project: {_id: 0, name: 1, "book.title": 1, "book.price": 1}}  
)
```



- Vorbereitung
 - Python Runtime mit PyMongo
 - <https://pymongo.readthedocs.io/en/stable/api/pymongo/>
 - Mongo 5.x
- Anwendungsentwicklung mit MongoDB
- Collections als TimeSeries
- Workshop
 - ToDo: Bitte senden Sie an training@rainer-sawitzki.de gerne eine Themenliste
-

Anwendungsprogrammierung

- Aktueller Stand
 - insert_one
 - insert_many
 - _id kann automatisch generiert werden
 - Antwort-Struktur enthält die generierten Ids
 - Query-API
 - Elementares find
 - Aggregate-Pipeline
 - Für komplexere Abfragen die Methode der Wahl
 - Insbesondere
 - \$lookup: "Joinen" auf Server-Seite
 - \$match: Erzeugt eine Ergebnis-Collection auf Server-Seite

- Neu
 - Löschen von Dokumenten
 - Aktualisieren von Dokumenten
- Problem
 - Umgang mit den Embedded Dokumenten
 - Das gesamte Dokument?
 - `replace_one`
 - Ein Embedded Dokument?
 - Ein Dokument in einer Liste?
 - Ein einzelnes Attribut
- Lösung
 - Aktualisierungs-Objekte



- \$set
- \$unset
- \$inc
- Arrays
 - \$push
 - \$addToSet
 - \$pull, \$pop

- Publisher
 - Aktualisierung der Adresse
 - Komplett neues Address-Dokument
 - Änderung der Street der vorhandenen Address

Publisher

- name <<query>>
- books
 - isbn
- address
 - city
 - street

Book

- isbn <<query>>
- title <<query>>
- pages
- price
- description
- publisherName
- authorNames
- tags

Array von
Schlüsselwörtern

Authors

- lastname
- firstname
- isbns

- Erweitern Sie das Book-Schema um eine optionale String-Liste “tags”
 - Alternativ: Validierung rausnehmen
- Schreiben Sie eine Funktion, die einem Book-Dokument ein neues Schlüsselwort hinzufügt
- Schreiben Sie eine Funktion, die einem Book-Dokument ein Schlüsselwort entfernt
- Suche Bücher nach Schlüsselwort
 - Entweder eines oder alle
-