AARHUS UNIVERSITET

Microservices and DevOps

DevOps and Container Technology

Document-based NoSQL: MongoDB

Henrik Bærbak Christensen

AARHUS UNIVERSITET

The Mongo Shell

- Mongo is a *document based* NoSQL.
  - A document is just a JSON object.
  - A collection is just a (large) set of documents
  - A database is just a set of collections.
  - MongoDB server is just a set of databases

- JSON = JavaScript Object Notation
  - Actually BSON = binary encoded JSON

- Mongo shell is a JavaScript interpreter!
  - (I have never coded in JavaScript, so…)

JSON

- Get used to key/value pairs!

- A simple **document**
- { course: ”MSDO”, semester:”E21”, students: 17 }

AARHUS UNIVERSITET

Basic commands…

- MongoDB creates objects and collections in the fly…

```
cs@ubuntu:~$ mongo
MongoDB shell version: 2.0.4
connecting to: test
> show collections
movies
ratings
system.indexes
> db.courses.insert( { course: "SAiP", teacher: "hbc", semester: "E12"} );
> db.courses.find();
{ "_id" : ObjectId("50b9e282528ddf0f4c544af9"), "course" : "SAiP", "teacher" : "
hbc", "semester" : "E12" }
{ "_id" : ObjectId("50b9e292528ddf0f4c544afa"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E12" }
> db.courses.insert( { course: "dOpSys", teacher: "ee", semester: "E12"} );
> db.courses.insert( { course: "PaSOOS", teacher: "ck", semester: "E12"} );
> db.courses.insert( { course: "dSoftArk", teacher: "hbc", semester: "E11"} );
> db.courses.insert( { course: "dSoftArk", teacher: "hbc", semester: "E10"} );
```

AARHUS UNIVERSITET

find()

- You can formulate simple queries using ‘find()’ on a collection.

- db.collection.find(); Return cursor
  - Cursor = iterator pattern, type ‘it’ to continue iterating

```
>
> db.courses.find();
{ "_id" : ObjectId("50b9e282528ddf0f4c544af9"), "course" : "SAiP", "teacher" : "
hbc", "semester" : "E12" }
{ "_id" : ObjectId("50b9e292528ddf0f4c544afa"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E12" }
{ "_id" : ObjectId("50b9e2fb528ddf0f4c544afb"), "course" : "dOpSys", "teacher" :
 "ee", "semester" : "E12" }
{ "_id" : ObjectId("50b9e308528ddf0f4c544afc"), "course" : "PaSOOS", "teacher" :
 "ck", "semester" : "E12" }
{ "_id" : ObjectId("50b9e319528ddf0f4c544afd"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E11" }
{ "_id" : ObjectId("50b9e31d528ddf0f4c544afe"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E10" }
>
```

find()

- You use JSON objects to formulate queries
  - where (_id == 733)
  - AND
  - Ranges
  - Regular expressions

- Counting .count();
- Sorting .sort(criteria);
- Limiting .limit(#);

AARHUS UNIVERSITET

find()

- You use JSON objects to formulate queries
  - ”where” AND regexp

```
> db.courses.find( { teacher: "hbc" } );
{ "_id" : ObjectId("50b9e282528ddf0f4c544af9"), "course" : "SAiP", "teacher" : "
hbc", "semester" : "E12" }
{ "_id" : ObjectId("50b9e292528ddf0f4c544afa"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E12" }
{ "_id" : ObjectId("50b9e319528ddf0f4c544afd"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E11" }
{ "_id" : ObjectId("50b9e31d528ddf0f4c544afe"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E10" }
>
>
> db.courses.find( { teacher: "hbc", semester: "E10" } );
{ "_id" : ObjectId("50b9e31d528ddf0f4c544afe"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E10" }
>
>
> db.courses.find( { teacher: /hb/ } );
{ "_id" : ObjectId("50b9e282528ddf0f4c544af9"), "course" : "SAiP", "teacher" : "
hbc", "semester" : "E12" }
{ "_id" : ObjectId("50b9e292528ddf0f4c544afa"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E12" }
{ "_id" : ObjectId("50b9e319528ddf0f4c544afd"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E11" }
{ "_id" : ObjectId("50b9e31d528ddf0f4c544afe"), "course" : "dSoftArk", "teacher"
 : "hbc", "semester" : "E10" }
>
```

AARHUS UNIVERSITET

Switching databases

- Let us switch to the database ‘movielens’
  - Movies Users Ratings

‘mydb’ in slides

```
cs@ubuntu: ~
>
>
> use mydb
switched to db mydb
> db.movies.findOne();
{
        "_id" : 1,
        "title" : "Toy Story (1995)",
        "genres" : [
                "Animation",
                "Children's",
                "Comedy"
        ]
}
> db.users.findOne();
{
        "_id" : 1,
        "gender" : "F",
        "age" : 1,
        "occupation" : 10,
        "zip_code" : "48067"
}
> db.ratings.findOne();
{
        "_id" : ObjectId("50b5db491d41c80f92000001"),
        "user_id" : 1,
        "movie_id" : 1193,
        "rating" : 5,
        "timestamp" : "978300760"
}
>
```

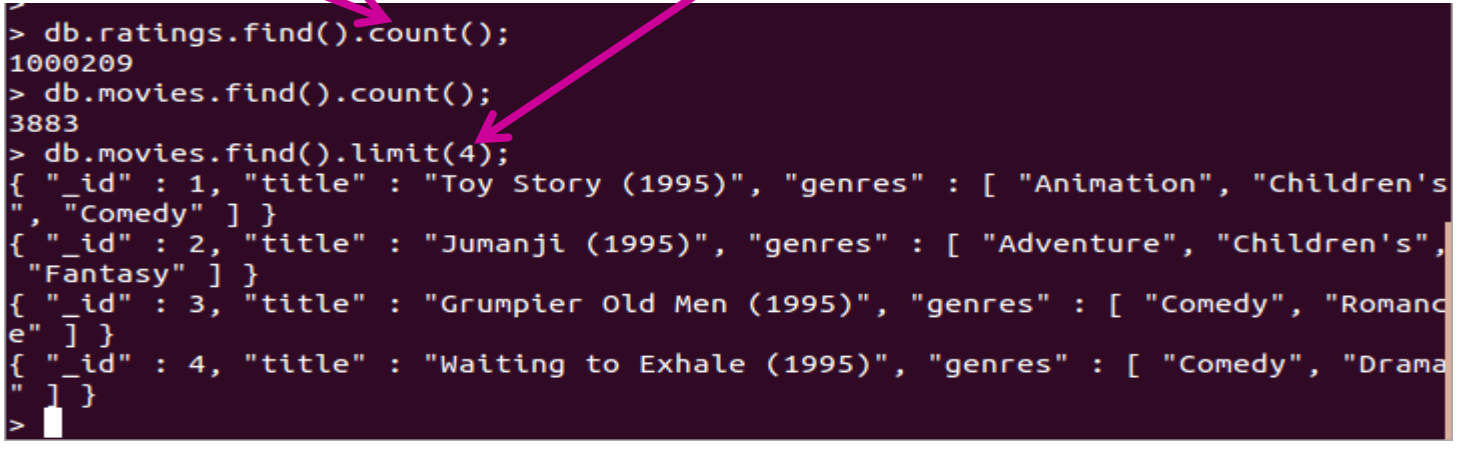
Note: A much more OO like datamodel than RDB! Arrays and nested objects!

AARHUS UNIVERSITET

Cursor functions

- Counting

Limiting



AARHUS UNIVERSITET

Sorting

```
> db.movies.find().sort( { title: -1 } ).limit(5);
{ "_id" : 2600, "title" : "eXistenZ (1999)", "genres" : [ "Action", "Sci-Fi", "T
hriller" ] }
{ "_id" : 2698, "title" : "Zone 39 (1997)", "genres" : "Sci-Fi" }
{ "_id" : 1426, "title" : "Zeus and Roxanne (1997)", "genres" : "Children's" }
{ "_id" : 1364, "title" : "Zero Kelvin (Kjaerlighetens kjoetere) (1995)", "genre
s" : "Action" }
{ "_id" : 1845, "title" : "Zero Effect (1998)", "genres" : [ "Comedy", "Thriller
" ] }
> db.movies.find( { title: /A/ } ).sort( { title: 1 } ).limit(5);
{ "_id" : 3420, "title" : "...And Justice for All (1979)", "genres" : [ "Drama",
 "Thriller" ] }
{ "_id" : 2572, "title" : "10 Things I Hate About You (1999)", "genres" : [ "Com
edy", "Romance" ] }
{ "_id" : 1203, "title" : "12 Angry Men (1957)", "genres" : "Drama" }
{ "_id" : 924, "title" : "2001: A Space Odyssey (1968)", "genres" : [ "Drama", "
Mystery", "Sci-Fi", "Thriller" ] }
{ "_id" : 1511, "title" : "A Chef in Love (1996)", "genres" : "Comedy" }
> db.movies.find( { title: /^A/ } ).sort( { title: 1 } ).limit(5);
{ "_id" : 1511, "title" : "A Chef in Love (1996)", "genres" : "Comedy" }
{ "_id" : 3928, "title" : "Abbott and Costello Meet Frankenstein (1948)", "genre
s" : [ "Comedy", "Horror" ] }
{ "_id" : 3648, "title" : "Abominable Snowman, The (1957)", "genres" : [ "Horror
", "Sci-Fi" ] }
{ "_id" : 3862, "title" : "About Adam (2000)", "genres" : "Comedy" }
{ "_id" : 2262, "title" : "About Last Night... (1986)", "genres" : [ "Comedy", "
Drama", "Romance" ] }
>
```

AARHUS UNIVERSITET

Ranges?

- Weird operators: $lt $ne etc…

```
> db.movies.find( { _id: { $lt : 4 } } );
{ "_id" : 1, "title" : "Toy Story (1995)", "genres" : [ "Animation", "Children's
", "Comedy" ] }
{ "_id" : 2, "title" : "Jumanji (1995)", "genres" : [ "Adventure", "Children's",
 "Fantasy" ] }
{ "_id" : 3, "title" : "Grumpier Old Men (1995)", "genres" : [ "Comedy", "Romanc
e" ] }
> db.movies.find( { _id: { $ne : 4 } } ).count();
3882
> db.movies.find().count();
3883
>
```

AARHUS UNIVERSITET

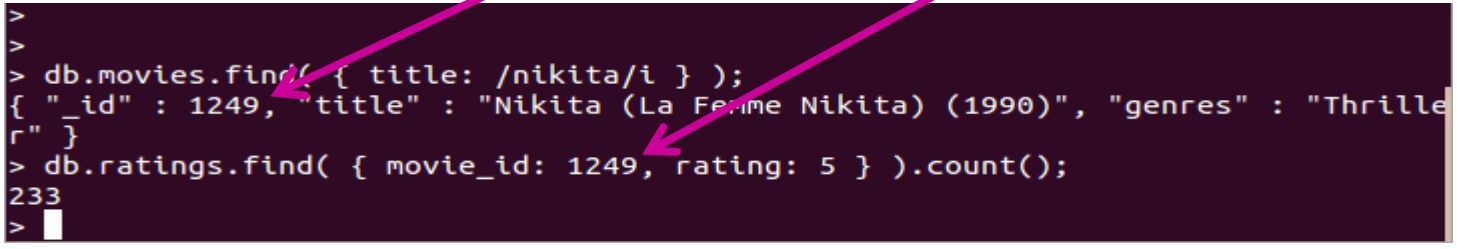
Living without Join

- Mongo does not have Join! (NoSQL have no join)
  - Select * from ratings r, movies m
    where r.movie_id = m._id AND
    m.title=”Nikita” AND r.rating=5 ☹
- Then what?
  - One answer **Denormalize**
    - However, denormalizing is very dependent on query patterns
    - Here: put the ratings *into* the movie document

AARHUS UNIVERSITET

Living without Join

- Then what?
  - Second answer **Manual references (= join on client side!)**
    - Find movie id using query 1; next find ratings using query 2
    - (§ 23.2.1 in MongoDB manual)



  - Third answer use DBRef (not NoSQL ☺)

AARHUS UNIVERSITET

Functional features

- Documents/cursors have methods, taking functions as parameters
  - cursor.forEach(function(value) { … });

```
> printjson
function (x) {
    print(tojson(x));
}
>
```

```
> db.ratings.find( { movie_id: 1249, rating: 5 } ).limit(2);
{ "_id" : ObjectId("50b5db4b1d41c80f92000b1d"), "user_id" : 23, "movie_id" : 124
9, "rating" : 5, "timestamp" : "978459672" }
{ "_id" : ObjectId("50b5db4d1d41c80f9200139a"), "user_id" : 36, "movie_id" : 124
9, "rating" : 5, "timestamp" : "978064419" }
> db.ratings.find( { movie_id: 1249, rating: 5 } ).limit(2).forEach(printjson);
{
        "_id" : ObjectId("50b5db4b1d41c80f92000b1d"),
        "user_id" : 23,
        "movie_id" : 1249,
        "rating" : 5,
        "timestamp" : "978459672"
}
{
        "_id" : ObjectId("50b5db4d1d41c80f9200139a"),
        "user_id" : 36,
        "movie_id" : 1249,
        "rating" : 5,
        "timestamp" : "978064419"
}
>
```

AARHUS UNIVERSITET

Nifty method

- Which is identical to
  - pretty()

Finding within Substructure

- Complex documents
- *Find all docs whose pdf slides contains 'mongo'*

```
emacs@m51
File  Edit  Options  Buffers  Tools  Javascript  Help
db.courses.insert({ course: "msdo", teacher: "hbc",
                    exercises: [
                        "exercise1",
                        "exercise2",
                        "exercise3" ],
                    slides: [
                        { pdf: "mongo.pdf", ppt: "mongo.pptx" },
                        { pdf: "mongo-ex.pdf", ppt: "mongo-ex.pptx" } ],
                    seminardate: "28/2 2020" })


db.courses.insert({ course: "saip", teacher: "ppe",
                    exercises: [
                        "exercise-fisk",
                        "exercise-sovs" ],
                    slides: [
                        { pdf: "fisk.pdf", ppt: "fisk.pptx" } ],
                    seminardate: "28/2 2021" })
```

```
> db.courses.find( { "slides.pdf": /mongo/ }).pretty()
{
        "_id" : ObjectId("5e566739f7f8dfd71765e639"),
        "course" : "msdo",
        "teacher" : "hbc",
        "exercises" : [
                "exercise1",
                "exercise2",
                "exercise3"
        ],
        "slides" : [
                {
                        "pdf" : "mongo.pdf",
```

AARHUS UNIVERSITET

Finding within Substructure

- Complex documents
- *Find docs that have certain no. of exercises*

emacs@m51

File Edit Options Buffers Tools Javascript Help

```
db.courses.insert({ course: "msdo", teacher: "hbc",
                    exercises: [
                        "exercise1",
                        "exercise2",
                        "exercise3" ],
                    slides: [
                        { pdf: "mongo.pdf", ppt: "mongo.pptx" },
                        { pdf: "mongo-ex.pdf", ppt: "mongo-ex.pptx" } ],
                    seminardate: "28/2 2020" })


db.courses.insert({ course: "saip", teacher: "ppe",
                    exercises: [
                        "exercise-fisk",
                        "exercise-sovs" ],
                    slides: [
                        { pdf: "fisk.pdf", ppt: "fisk.pptx" } ],
                    seminardate: "28/2 2021" })
```

```
> db.courses.find( { "exercises": { $size : 3} }).count()
1
> db.courses.find( { "exercises": { $size : 2} }).count()
1
> db.courses.find( { "exercises": { $size : 4} }).count()
0
```

And…

- We mostly ‘play around’ without large datasets, but…

- Queries are linear search unless you have indices…

```
> db.courses.createIndex( { course: 1} )
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

AARHUS UNIVERSITET

Discussion

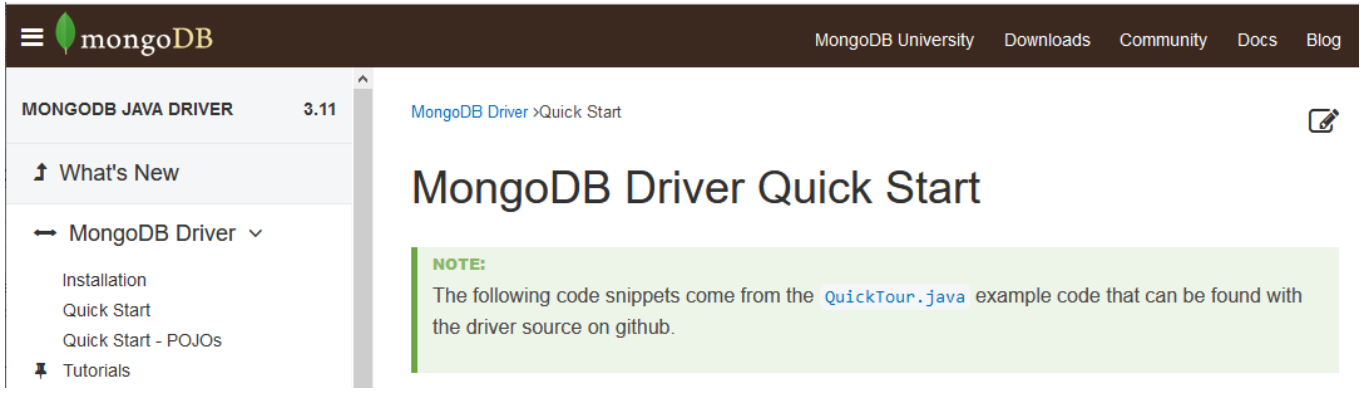
- MongoDB is a Document database
  - Much better OO-like modeling compared to RDB
  - Advanced (and complex!) query language
    - (And what you learn now, cannot be carried over to the next NoSQL you may use ☹. Compare to learning SQL).
  - “Schema less”
    - Easy to get started, and hell once people start storing different versions of documents in the same collection ☹
      - Classic versioning handling tactics must be applied…
    - And you **have to define schemas** anyway, to demarshall the stored JSON correctly ☺

The Java API

AARHUS UNIVERSITET

The Good News

- MongoDB is choosen for this course basically because their tutorial are pretty good



AARHUS UNIVERSITET

Getting the Driver

- Help me Gradle ☺

```
dependencies {
    compile project(':common')

    // Bind the SLF4J logging framework to Log4J
    compile 'org.slf4j:slf4j-log4j12:1.7.26'

    // Minimal library for handling JSON
    compile 'com.googlecode.json-simple:json-simple:1.1.1'

    // BCrypt for local password encryption
    compile 'org.mindrot:jbcrypt:0.4'

    // FRDS.Broker library
    compile group: 'com.baerbak.maven', name: 'broker', version: '1.7'

    testCompile 'junit:junit:4.12'

    // MongoDB
    compile group: 'org.mongodb', name: 'mongodb-driver', version: '3.10.2'
}
```

AARHUS UNIVERSITET

Initialization

```java
public static void main(final String[] args) {
    MongoClient mongoClient;

    if (args.length == 0) {
        // connect to the local database server
        mongoClient = MongoClients.create();
    } else {
        mongoClient = MongoClients.create(args[0]);
    }

    // get handle to "mydb" database
    MongoDatabase database = mongoClient.getDatabase("mydb");


    // get a handle to the "test" collection
    MongoCollection<Document> collection = database.getCollection("test");

    // drop all the data in it
    collection.drop();
```

CRUD

```
// make a document and insert it
Document doc = new Document("name", "MongoDB")
                .append("type", "database")
                .append("count", 1)
                .append("info", new Document("x", 203).append("y", 102));

collection.insertOne(doc);

// get it (since it's the only one in there since we dropped the rest earlier on)
Document myDoc = collection.find().first();
System.out.println(myDoc.toJson());
```

```
// now, lets add lots of little documents to the collection so we can explore queries and cursors
List<Document> documents = new ArrayList<Document>();
for (int i = 0; i < 100; i++) {
    documents.add(new Document("i", i));
}
collection.insertMany(documents);
System.out.println("total # of documents after inserting 100 small ones (should be 101) " + collection.countDocuments());
```

CRUD

```
for (Document cur : collection.find()) {
    System.out.println(cur.toJson());
}
```

- Searching

```
// now use a query to get 1 document out
myDoc = collection.find(eq("i", 71)).first();
System.out.println(myDoc.toJson());

// now use a range query to get a larger subset
cursor = collection.find(gt("i", 50)).iterator();
```

```
// range query with multiple constraints
cursor = collection.find(and(gt("i", 50), lte("i", 100))).iterator();
```

- Sorting

```
// Sorting
myDoc = collection.find(exists("i")).sort(descending("i")).first();
System.out.println(myDoc.toJson());
```

AARHUS UNIVERSITET

‘lambda’ like procesing

- Apply Ops to each element in query: forEach()

```
Consumer<Document> assignBlock = new Consumer<Document>() {
  public void accept(final Document document) { theList.add(p
};

playerCollection
        .find(
                and(
                        eq(POSITION_KEY, positionString),
                        ne(SESSION_ID_KEY,  value: null)
                ))
        .forEach(assignBlock);
```