MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT5148 - Distributed Databases and Big Data

Module 4: MongoDB

GROUP OF EIGHT AUSTRALIA

MongoDB

- A free and open-source cross-platform **document-oriented** database.
- A NoSQL database
- Uses **JSON-like documents (BSON - Binary JSON)** with schemata.
- Developed by MongoDB Inc.
- Published under a combination of the GNU Affero General Public License and the Apache License.

Relational	MongoDB
Database	Database
Table	Collection
Row	Document

Document Oriented Database

- Document Data Model (JSON Data Model)

sid	sfirstname	slastname
123	Marie	Currie
124	Albert	Einstein

ucode	uname
FIT9132	Database
FIT9131	Programming

sid	ucode	year	semester	mark
123	FIT9132	2017	1	100
123	FIIT9131	2017	2	80
124	FIT9131	2017	2	100

```
{
    "enrolment": [
        {
            "sid": 123,
            "name": {
                "first": "Marie",
                "last": "Currie"
            },
            "course": "MIT",
            "result": [
                {
                    "unit_code": "FIT9132",
                    "unit_name": "Database",
                    "semester": 1,
                    "year": 2017,
                    "result": 100
                },
                {
                    "unit_code": "FIT9131",
                    "unit_name" "Programming"
                    "semester": 2,
                    "year": 2017,
                    "result": 80
                }
            ]
        },
        {
            "sid": 124,
            "name": {
                "first": "Albert",
                "last": "Einstein"
            },
            "course": "MBIS",
            "result": [
                {
                    "unit_code": "FIT9131",
                    "unit_name": "Database",
                    "semester": 2,
                    "year": 2017,
                    "result": 100
                }
            ]
        }
    ]
}
```

Document Oriented Database

- Document Data Model (JSON Data Model)
- Terminologies
  - Object
  - Members
  - Pair
  - Value
  - Array
  - Elements

```
{
    "enrolment": [
        {
            "sid": 123,
            "name": {
                "first": "Marie",
                "last": "Currie"
            },
            "course": "MIT",
            "result": [
                {
                    "unit_code": "FIT9132",
                    "unit_name": "Database",
                    "semester": 1,
                    "year": 2017,
                    "result": 100
                },
                {
                    "unit_code": "FIT9131",
                    "unit_name" "Programming"
                    "semester": 2,
                    "year": 2017,
                    "result": 80
                }
            ]
        },
        {
            "sid": 124,
            "name": {
                "first": "Albert",
                "last": "Einstein"
            },
            "course": "MBIS",
            "result": [
                {
                    "unit_code": "FIT9131",
                    "unit_name": "Database",
                    "semester": 2,
                    "year": 2017,
                    "result": 100
                }
            ]
        }
    ]
}
```

Document Oriented Database

- Document Data Model (JSON Data Model)
- **JSON Object VS Array**
  - Object:
    - Unordered list.
    - List 1 {a,b,c,d} = List 2 {b,c,a,d}
    - Accessed by its name.
  - Array:
    - Ordered list
    - List 1 {a,b,c,d} != List 2 {b,c,a,d}
    - Can be accessed by its position.
- **JSON Data Type**
  - String
  - Number
  - Object
  - Array
  - True
  - False
  - NULL
  - BSON provides more data types, eg date, integer, float.

Here is why MongoDB is great

- Flexible Schema. If your requirements change, you can adapt.
- Unstructured data - you can store and retrieve unstructured data easily. It's just JSON. Not every document in a collection needs the same fields.
- Denormalized data - Group related content in a single document.
- Clean and simple API - Mongo is nice to talk to.

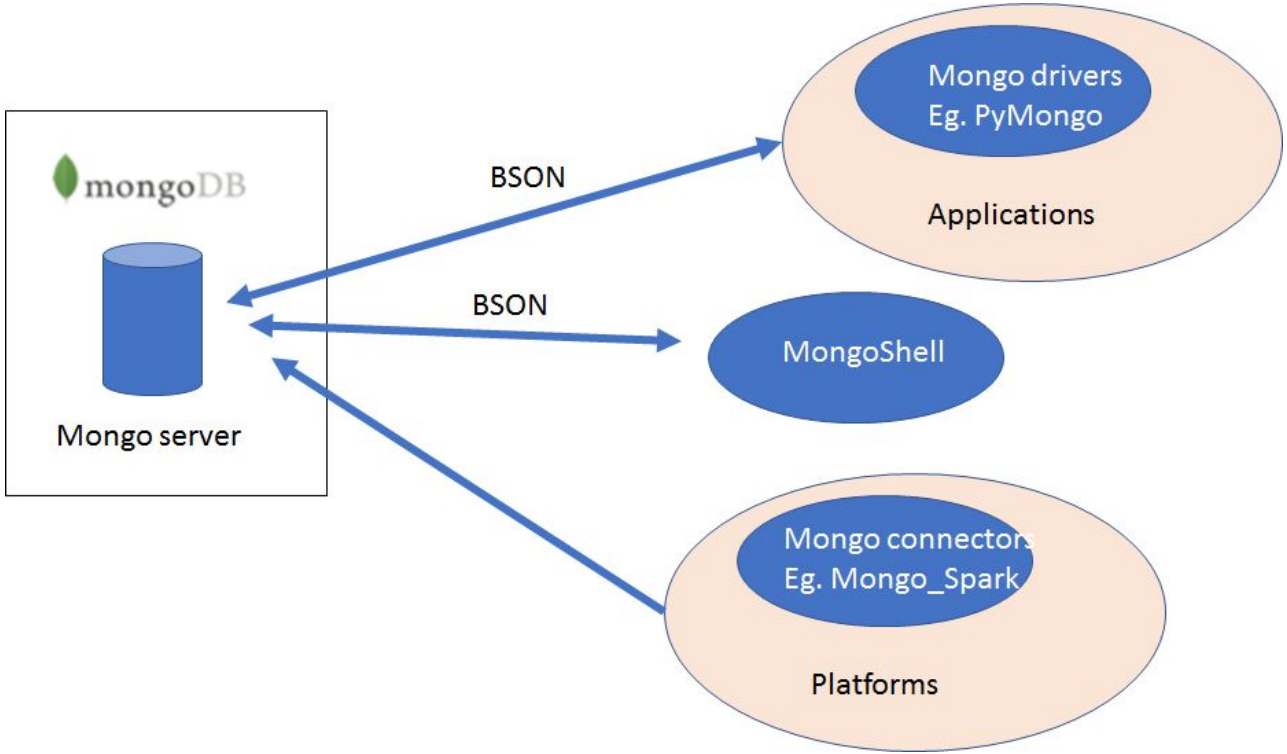
Here is why you might not like MongoDB

- Denormalized data means no joins.
- Flexible schema means no built in data validation.
- Bugs - Mongo is new and there are still issues in the tracker.
- No transactions
- Theoretical data loss

When should you use MongoDB

- Mongo represents data as a tree. If your data is tree shaped, or can be made tree shaped, Mongo is great.
- Unstructured data
- Semi-structured
- If you have big customer data to store, and each customer record contains lists of communications, subscriptions, etc, the data is tree shaped, and Mongo would again be a good choice.
- If you have big data and you want to query it in interesting and complex ways, pulling useful aggregated data out the other side in surprisingly short timeframes, Mongo is perfect.

MongoDB Architecture



Using Mongo Import

- Run the MongoDB server **(you don't need to do this in the VM)**
  - **`mongod`**

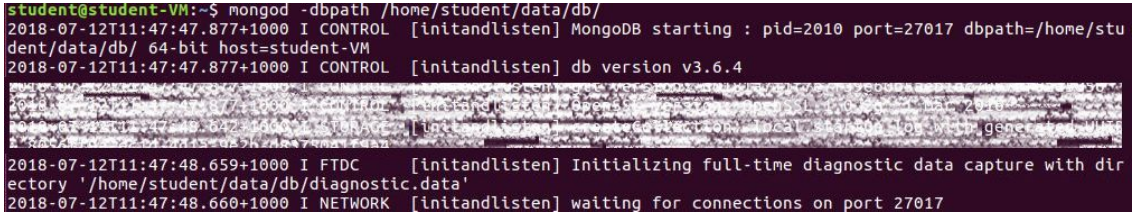
```
student@student-VM:~$ mongod -dbpath /home/student/data/db/
2018-07-12T11:47:47.877+1000 I CONTROL  [initandlisten] MongoDB starting : pid=2010 port=27017 dbpath=/home/stu
dent/data/db/ 64-bit host=student-VM
2018-07-12T11:47:47.877+1000 I CONTROL  [initandlisten] db version v3.6.4
[redacted]
2018-07-12T11:47:48.659+1000 I FTDC     [initandlisten] Initializing full-time diagnostic data capture with dir
ectory '/home/student/data/db/diagnostic.data'
2018-07-12T11:47:48.660+1000 I NETWORK  [initandlisten] waiting for connections on port 27017
```

- MongoImport **(do this in terminal or command line)**
  - **`mongoImport`**

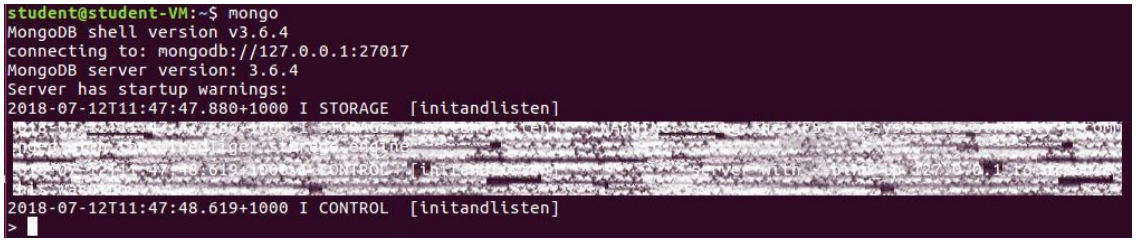
```
student@student-VM:~$ mongoimport --db petshop --collection pets --file FIT5148_
Online/pets.json
2018-09-20T11:22:10.399+1000    connected to: localhost
2018-09-20T11:22:10.427+1000    imported 3 documents
```

Using MongoDB Shell

- Run the MongoDB server **(you don't need to do this in the VM)**
  - **`mongod`**



- Run the Mongo Shell
  - **`mongo`**



https://docs.mongodb.com/manual/reference/mongo-shell/

Using MongoDB Shell

- List existing databases
  - **`show dbs`**

```
> show dbs
admin       0.000GB
config      0.000GB
fit5148_db  0.000GB
local       0.000GB
```

- List existing collections
  - **`show collections`**

```
> use fit5148_db
switched to db fit5148_db
> show collections
climate
fire
taskC_climate_fire
>
```

Using MongoDB Shell

- List contents of a collection

```
> db.climate.find().pretty()
{
        "_id" : ObjectId("5b403f30cf88174f5e0059a7"),
        "Station" : 948700,
        "Date" : "2016-12-31",
        "Air_Temperature_Celcius" : 19,
        "Relative_Humidity" : 56.8,
        "WindSpeed_knots" : 7.9,
        "Max_Wind_Speed" : 11.1,
        "Precipitation" : "0.00I"
}
{
        "_id" : ObjectId("5b403f30cf88174f5e0059a8"),
        "Station" : 948700,
        "Date" : "2017-01-02",
        "Air_Temperature_Celcius" : 15,
        "Relative_Humidity" : 50.7,
        "WindSpeed_knots" : 9.2,
        "Max_Wind_Speed" : 13,
        "Precipitation" : "0.02G"
}
{
        "_id" : ObjectId("5b403f30cf88174f5e0059a9"),
        "Station" : 948700,
        "Date" : "2017-01-03",
        "Air_Temperature_Celcius" : 16,
        "Relative_Humidity" : 53.6,
        "WindSpeed_knots" : 8.1,
        "Max_Wind_Speed" : 15,
        "Precipitation" : "0.00G"
}
```

Using MongoDB Shell

- Creating a database
  - We can switch to a database in Mongo with the use command.
  - `USE petshop`
- Dropping the database
  - `db.dropDatabase()`

Using MongoDB Shell

- **Collections**
  - Collections are sets of (usually) related documents.
  - A database can have many collections.
  - Create a collection using the `createCollection` command.
    - `USE petshop`
    - `db.createCollection('mammals')`
- View the database and collections
    - `show dbs`
    - `show collections`

Using MongoDB Shell

- **Documents**
  - Documents are JSON objects that live inside a collection.
  - They can be any valid JSON format, with the caveat that they can't contain functions.
  - The size limit for a document is 16Mb which is more than ample for most use cases.
  - Create a document by inserting it into a collection

Using MongoDB Shell

- **Documents**
  - Insert Document (ONE)
    - `db.mammals.insert({"name": "Polar Bear"})`
    - `db.mammals.insert({"name": "Star Nosed Mole"})`
  - Insert Document(MANY)
    - `db.mammals.insertMany([{"name": "Baboon"},{"name":"Opossum"}],{"ordered": false})`

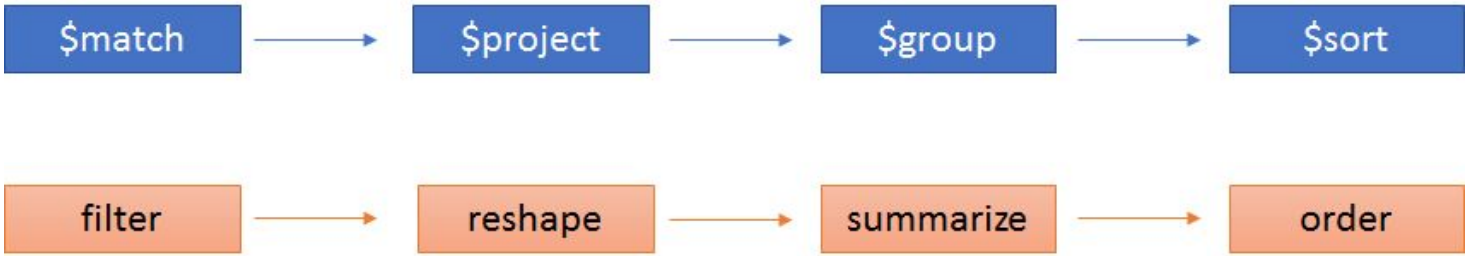
- Inserted as a collection of array elements.
- Each of the element is a document.

- **Documents**
  - **Finding documents**
    - We can find a document or documents matching a particular pattern using the find function.
      - `db.mammals.find()`
      - `db.mammals.find({})`
    - The argument for the function is a JSON document.
    - The return will be a single or many JSON documents.

Using MongoDB Shell

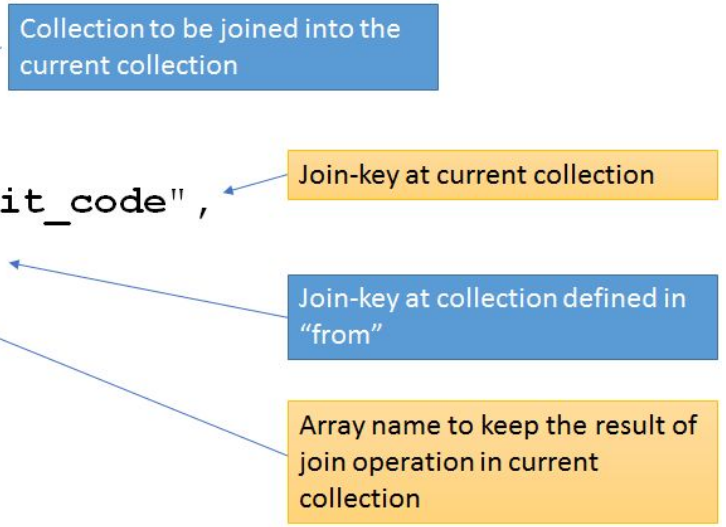
- **Documents**
  - **Finding documents**
    - Find a document by id
      - `db.mammals.find({ObjectId("557afc91c0b20703009f")})`
    - Finding by partial match
      - `db.mammals.find({"name": "Polar Bear"})`
      - `db.people.find({"name": "dave","email": "davey@aol.com"})`
      - `db.people.find({"name": "dave","age": 69, "email": "davey@aol.com"})`

Using MongoDB shell

- **Documents**
  - **Finding documents**
    - **Exercise:** Paste the following into your terminal to create a petshop with some pets in it.
      - `use petshop`
      - ```
        db.pets.insertMany([{name: "Mikey", species: "Gerbil"},
        {name: "Davey Bungooligan", species: "Piranha"},
        {name: "Suzy B", species: "Cat"},
        {name: "Mikey", species: "Hotdog"},
        {name: "Terrence", species: "Sausagedog"},
        {name: "Philomena Jones", species: "Cat"}])
        ```
        ```

Using MongoDB shell

- **Documents**
  - **Finding documents**
    - Tasks:
      - Add another piranha, and a naked mole rat called Henry.
      - Use find to list all the pets. Find the ID of Mikey the Gerbil.
      - Use find to find Mikey by id.
      - Use find to find all the gerbils.
      - Find all the creatures named Mikey.
      - Find all the creatures named Mikey who are gerbils.
      - Find all the creatures with the string "dog" in their species.

Using MongoDB shell

- **Documents**
  - **Finding Nested documents**
    - Find people with the last name Morgan.
      - `db.people.find({"name.last": "Morgan"})`
  - **Operators - comparison**
    - Find people with age more than 20
      - `db.people.find({"age": {"$gt": 20}})`
    - Find people with height less than 175 cms and age more than 20
      - `db.people.find({"height":{"$lt":175}, "age": {"$gt": 20}})`

http://docs.mongodb.org/manual/reference/operator/query/

Using MongoDB shell

- **Documents**
  - **Logical - AND Implicit**
    - Comma separator of filter acts as an AND operator
      - `db.people.find({"height":{"$lt":175}, "age": {"$gt": 20}})`
  - **Logical - AND Explicit**
    - $and
      - `db.people.find({"$and":[{"height":{"$lt":175}}, {"age": {"$gt": 20}}]})`
  - **Logical - OR**
    - $or
      - `db.people.find({"$or":[{"height":{"$lt":175}}, {"age": {"$gt": 20}}]})`

Using MongoDB shell

- **Documents**
  - **Querying Missing or NULL fields**
    - Value of **null**
      - Find people who have null value as their age
        - » `db.people.find({"age": null})`
    - **$exists**
      - Find people without any age.
        - » `db.people.find({"age": {"$exists":false}})`

Using MongoDB shell

- **Documents**
  - **Searching Text**
    - MongoDB uses PCRE(PERL Compatible Regular Expression)
      - Find people whose surname starts with letter “B”.
        - `db.people.find({“name.last”: {“$regex”: “^B”}})`
  - **Limiting Fields in the Query Output**
    - Find people whose age is 25 and display only their last name.
      - `db.people.find({“age”:25},{“name.last”:1, “_id”:0})`
  - **Limiting number of documents returned.**
    - Find 10 people whose age is 25.
      - `db.people.find({“age”:25}).limit(10)`

Using MongoDB shell

- **Documents**
  - **Aggregation**
    - Processing a group of documents.
      - Aggregated values e.g. sum, count, average
      - Reshaping document during analysis.
      - Aggregation Pipeline Example



Using MongoDB shell

- **Documents**
  - **Aggregation**
    - Exercise: Try this in the pets collection.
      - `db.pets.aggregate([{"$group":{"_id":"$species", "total":{"$sum":1}}}])`
  - **Aggregating Array Elements**
    - Need to change array elements into an object of a document using $unwind.
      - ```
        db.evaluation.aggregate([
        {$unwind:"$results"},
        {$group:{_id:"$results.item",average:{$avg:"$results.score"}}},
        {$sort:{average : 1}}])
        ```
        ```

JOIN Using Aggregation

```
db.FIT.aggregate(
   [
      {$lookup:
         {  from: "unit",
            localField: "result.unit_code",
            foreignField: "_id",
            as: "enrolled_unit"
         }
      },
      {$out:"studUnit"}
   ]
)
```



JOIN Using Aggregation

Try this:

```
db.orders.insert([
   { "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2 },
   { "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1 },
   { "_id" : 3  }])
db.inventory.insert([
   { "_id" : 1, "sku" : "almonds", description: "product 1", "instock" : 120 },
   { "_id" : 2, "sku" : "bread", description: "product 2", "instock" : 80 },
   { "_id" : 3, "sku" : "cashews", description: "product 3", "instock" : 60 },
   { "_id" : 4, "sku" : "pecans", description: "product 4", "instock" : 70 },
   { "_id" : 5, "sku": null, description: "Incomplete" },
   { "_id" : 6 }])
```

JOIN Using Aggregation

Try this:

```
db.orders.aggregate([
    {
      $lookup:
      {
            from: "inventory",
            localField: "item",
            foreignField: "sku",
            as: "inventory_docs"
      }
   }
])
```

https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/

Using MongoDB shell

- **Documents**
  - **Update**
    - `db.collection.updateOne(<filter>, <update>, <options>)`
    - `db.collection.updateMany(<filter>, <update>, <options>)`
    - `db.collection.replaceOne(<filter>, <replacement>, <options>)`
  - **Examples**
    - `db.pets.updateOne({"name": "Mikey"},{"$set":{"species": "Cat"}})`
    - `db.pets.updateMany({"species": "Cat"},{"$set":{"species": "Dog"}})`
    - `db.pets.replaceOne({"name": "Mikey"},`
      `{"name": "Mikey", "species": "Cat", "age": 2})`

Using MongoDB shell

- **Documents**
  - **Delete**
    - `db.collection.deleteMany()`
    - `db.collection.deleteOne()`
    - `db.collection.drop()`
  - **Examples**
    - `db.pets.deleteOne({“name”: “Mikey”})`
    - `db.pets.deleteMany({“species”: “Dog”})`
    - `db.pets.drop()`

Data Model

- **Modelling Documents**

Model 1

```
{ results: [ {"item": "content", "score": 9}, {"item": "presentation", "score": 6}]}

db.evaluation.find({"results":{$elemMatch:{"score":9,"item":"presentation"}}})
```

Model 2

```
{results: {"content": 9,"presentation":6}}

db.evaluation_1.find({"results.content":9})
```

Model 3

```
{ results: [ {"content": 9}, {"presentation":6}]}

db.evaluation_1.find({"results.presentation":9})
```

Data Model

- **Modelling Relationships**
  - Redundancy may exist, data typically is not normalised.
  - Connecting two or more documents.
    - Embedding.
    - Referencing
  - There is no JOIN operator.
  - Join can be performed using:
    - $lookup in aggregate => outer join
    - Embedded document structure
    - Application code

Data Model

- **Modelling One to Many Relationships**
  - Embedding Documents

```
{
   _id: "joe",
   name: "Joe Bookreader"
}

{
   patron_id: "joe",
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}

{
   patron_id: "joe",
   street: "1 Some Other Street",
   city: "Boston",
   state: "MA",
   zip: "12345"
}
```

```
{
   _id: "joe",
   name: "Joe Bookreader",
   addresses: [
                {
                  street: "123 Fake Street",
                  city: "Faketon",
                  state: "MA",
                  zip: "12345"
                },
                {
                  street: "1 Some Other Street",
                  city: "Boston",
                  state: "MA",
                  zip: "12345"
                }
              ]
}
```

Data Model

- **Modelling One to Many Relationships**
  - Referencing Documents

```
{
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   publisher: {
              name: "O'Reilly Media",
              founded: 1980,
              location: "CA"
            }
}

{
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English",
   publisher: {
              name: "O'Reilly Media",
              founded: 1980,
              location: "CA"
            }
}
```

```
{
   name: "O'Reilly Media",
   founded: 1980,
   location: "CA",
   books: [123456789, 234567890, ...]
}
```

Do this

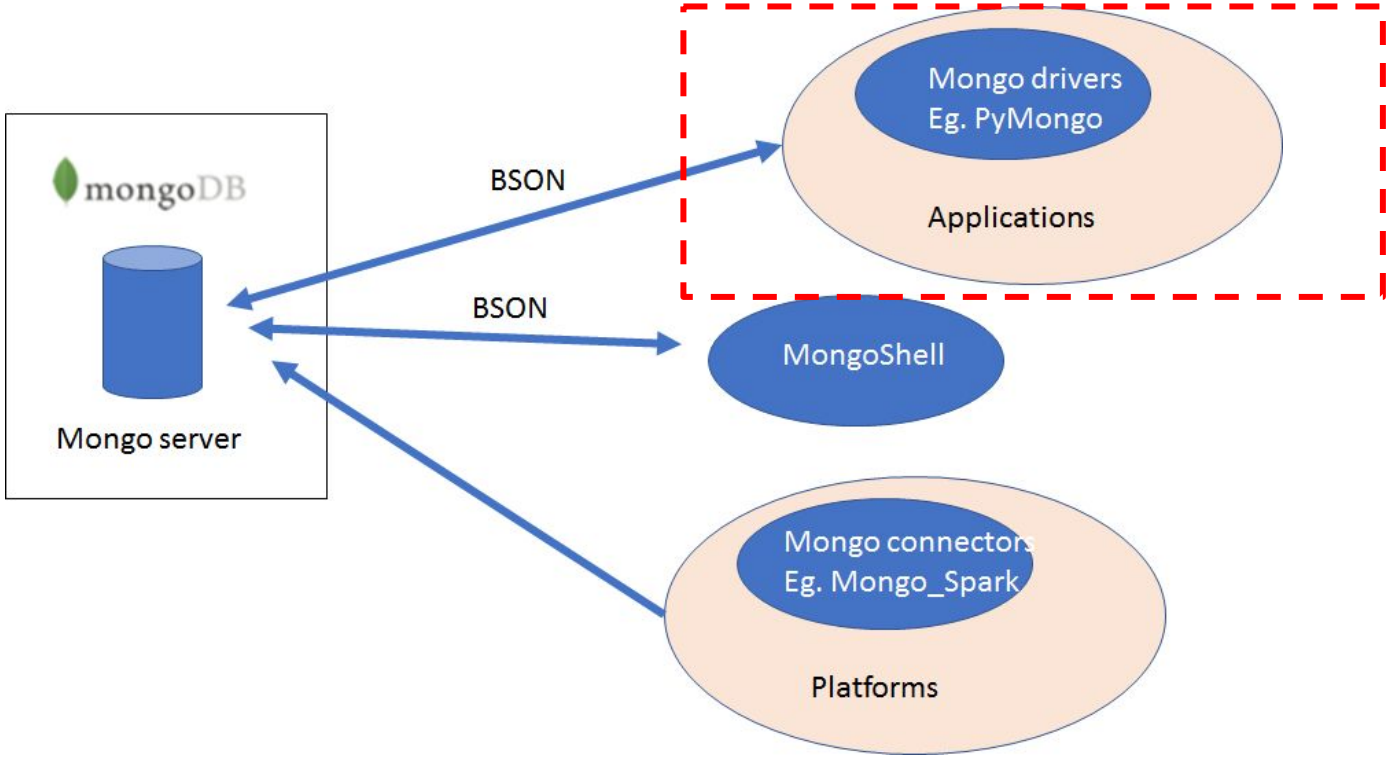
```
{
   _id: 123456789,
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English"
}

{
   _id: 234567890,
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English"
}
```

```
{
   _id: "oreilly",
   name: "O'Reilly Media",
   founded: 1980,
   location: "CA"
}
```

Or this

```
{
   _id: 123456789,
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   publisher_id: "oreilly"
}

{
   _id: 234567890,
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English",
   publisher_id: "oreilly"
}
```

MongoDB Architecture



Connecting MongoDB and Python

1. Use `PyMongo` driver from a python application.
2. Import the library.

```
import pymongo

from pymongo import MongoClient
```

3. Make connection to the server
   - `Client = MongoClient()` **=> using default host and port**
   - `Client = MongoClient('localhost',27017)` **=> explicitly specify host and port**
   - `Client = MongoClient('mongodb://localhost:27017/')` **=> using URI to specify host and port**
4. Make a connection to the database.

```
db = client.FIT5148
```

5. Make connection to the collection.

```
coll = db.FIT
```

Python with MongoDB Code Example

```
import pymongo
from pymongo import MongoClient
from pprint import pprint
client = MongoClient()
db = client.FIT5148
collEval = db.evaluation
evals = [
    {"results":[{"item": "content","score": 9},{"item": "presentation","score": 6}]},
    { "results":[{"item": "content","score": 8},{"item": "presentation","score": 8}]}
  ]
db.collEval.insert_many(evals) #using insert_many rather than insertMany().
cursor = collEval.find({})
for document in cursor:
  pprint(document)
```

References

MONASH University

- http://nicholasjohnson.com/mongo/course/workbook/
- https://docs.mongodb.com/manual/aggregation/
- https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1
- https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-2
- https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-3
- https://docs.mongodb.com/manual/crud/#read-operations
- https://docs.mongodb.com/manual/reference/operator/query/

Thank You

MONASH University

Any Simple Questions?