MongoDB

Chapter contents:

- MongoDB: queries
- MongoDB: architecture

Table of contents

2022-2023

MongoDB: Historical/economic context

Developped from 2007 by ad-backend people (with venture capital funding). Soon became an open-source NoSQL document store meant for the cloud and developers.

More flexibility than normalized relational schema (and no ORM). And easy to distribute on the cloud. Less emphasis on schema design/data organization, to shift focus towards agile software development.

Rich API. Performance and features improved a lot over time (though transaction support remains debatable).

Most popular document store (2020), 5^{th} DBMS (1^{st} among NoSQL).

Easy to deploy. Free Community edition (GNU AGPLv3), or MongoDB Enterprise with professional support, access control, BI connectors, visualizations, in-memory engine. Also available as DBaaS (Atlas).

[https://www.mongodb.com/nosql-explained]

MongoDB

Records JSON documents into *collections*.

- each doc has a unique field `"_id"`,
  automatically generated: it's the "primary key".
- the docs in a collection do not always follow the same schema (schema-less).
- the docs are actually stored in MongoDB's binary format for JSON: BSON

MongoDB is a (*document store*).

Relational	MongoDB
table	collection
line	document
column	field
join	nested doc, lookup
aggregation	aggregation pipeline
secondary index	secondary index
SQL	calling methods on a document object

[https://www.mongodb.com/compare/mongodb-mysql]

MongoDB : inserting documents

```
> help
> show dbs
> show collections
// See also: use madatabase, db.dropDatabase(), db.movies.drop(),

// insertions, creates  collection if needed
> db.movies.insertOne({"nom": "Les 7 Samurais" })

// inserting many docs at once :
> db.movies.insertMany(
  [
    {"nom": "Citizen Kane"},
    {"nom": "The Godfather",
      "acteurs": [
        {"prenom": "Marlon", "nom": "Brando"},
        {"prenom": "Al", "nom": "Pacino"} ]}
  ], {
          writeconcern : { w: "majority", wtimeout: 100 },
          j: true,
          ordered : false
          })
// if all required (a majority) replica nodes have not acknowledged
// within 100ms and been written on journal
```

https://www.mongodb.com/docs/manual/tutorial/insert-documents/

MongoDB : querying documents

```
> db.movies.find()
// result:                    ... to indent: db.movies.find().pretty()
// { "_id" : ObjectId("5a982ac71e32c4e0f52641be"), "nom" : "Les 7 Samurais" }
// { "_id" : ObjectId("5a982b3f1e32c4e0f52641bf"), "nom" : "Citizen Kane" }
// { ... }

// selection: (returns all movie docs (complete doc))
> db.movies.find({"nom": "The Godfather"})
> db.movies.find({"acteurs.nom": "Pacino"}).sort({"nom":1}).skip(0).limit(1)
```

-1 for descending

https://www.mongodb.com/docs/manual/tutorial/query-documents/

MongoDB

MongoDB Queries:

```
/*  db.collection.find(query)  */

// Selection operators: regex, in, nin (not in), all, $lt...
> db.movies.find({"nom": {$regex: '^[lc]', $options: "i"}})
> db.movies.find({"acteurs.nom": {$in: ["Pacino", "Brando"]}})
> db.movies.find({"acteurs.nom": {$nin: ["Pacino", "Brando"]}})
> db.movies.find({"acteurs.nom": {$all: ["Pacino", "Brando"]}})

// boolean combinations of conditions
> db.movies.find({"nom": { $gt: "D", $lt: "M"} })
> db.movies.find({$and : [{"acteurs.nom": "Pacino"}, {"acteurs.nom": "Brando"}]})
> db.movies.find({$or : [{"acteurs.nom": "Pacino"}, {"nom": { $lt: "D"} }]})
> db.movies.find({"acteurs.nom": { $gt: "D", $lt: "C"} })
> db.movies.find({"acteurs.nom": { $elemMatch : {$gt: "D", $lt: "C"} } })

/*  db.collection.find(query, projection)  */

// projection: field is exluded if 0 or null, returned on other values
//   but _id always in unless explicitly out.
//   Cannot specify simultaneously in and out (except _id)
> db.movies.find({}, {"acteurs.prenom": 0})    ⟶ champs prenom est exclu
> db.movies.find({"acteurs.nom": {$all: ["Pacino", "Brando"]}}, {acteurs: 1})
```

MongoDB: aggregation

```
// aggregation - easy cases:
> db.movies.distinct("nom") // returns: [ "Les 7 Samurais", "Citizen Kane"... ]
> db.movies.count()    // returns : 3

// aggregation pipeline :  aggregate( [ <step1>, <step2> ...] )
> db.movies.aggregate(
    [ {
      $group: { "_id": "$acteurs.nom", "nb_films": { $sum: 1}  }
    } ] )

> db.movies.aggregate(
    [
      { $match: { "nom": { $ne: "Citizen Kane"}}},          ⟶ match : to filter
      { $group: { "_id": "$acteurs.nom", "nb_films": { $sum: 1}  }  }
    ] )
// ... and other operations to transform, sample
```

MongoDB: joins

In the past: you would have written a loop client-side

Ex: js in mongo shell:

```
while (curs.hasNext()) {
  ...
  find(...)
}
```

Since version 3.2: `$lookup` operator in pipeline.

```
> db.movies.aggregate(
    [{
    $lookup:
       {
        from: "seances",          ⟶ the other collection we wish to join
        localField: "nom",
        foreignField: "nom_film",
        as: "infos_cine"          ⟶ field containing the array of "seance objects"
      }
    }] )
```

Similar to : `movies m LEFT OUTER JOIN seances s ON m.nom=s.nom_film`

but all seances listed in an array ⇒ `$unwind` to break the array into a collection...

MongoDB: MapReduce

```
var mapfunction = function () {
  emit (this._id, 1);
};

var reducefunction = function (key, values) {
  return Array.sum(values);
};

db.movies.mapReduce(
  mapfunction,
  reducefunction,
  { out : "fichier_nb" }
)

db.fichier_nb.find()
```

- you should preferably use aggregation pipeline
- not the best MapReduce implementation for massive data:
  - restrictions on reduce (associative, commutative, idempotente. . .)
  - performance does not match Hadoop

A priori, for heavy batch computation you would import data from MongoDB into some other computation framework.

Table of contents

MongoDB: a distributed DB. Replication

Replication: multiple instances (forming what MongoDB calls a *replica set*) maintain the same data: 1 primary node (master) receives all operations (esp. writes), and secondary nodes (slaves) copy the primary node's data.

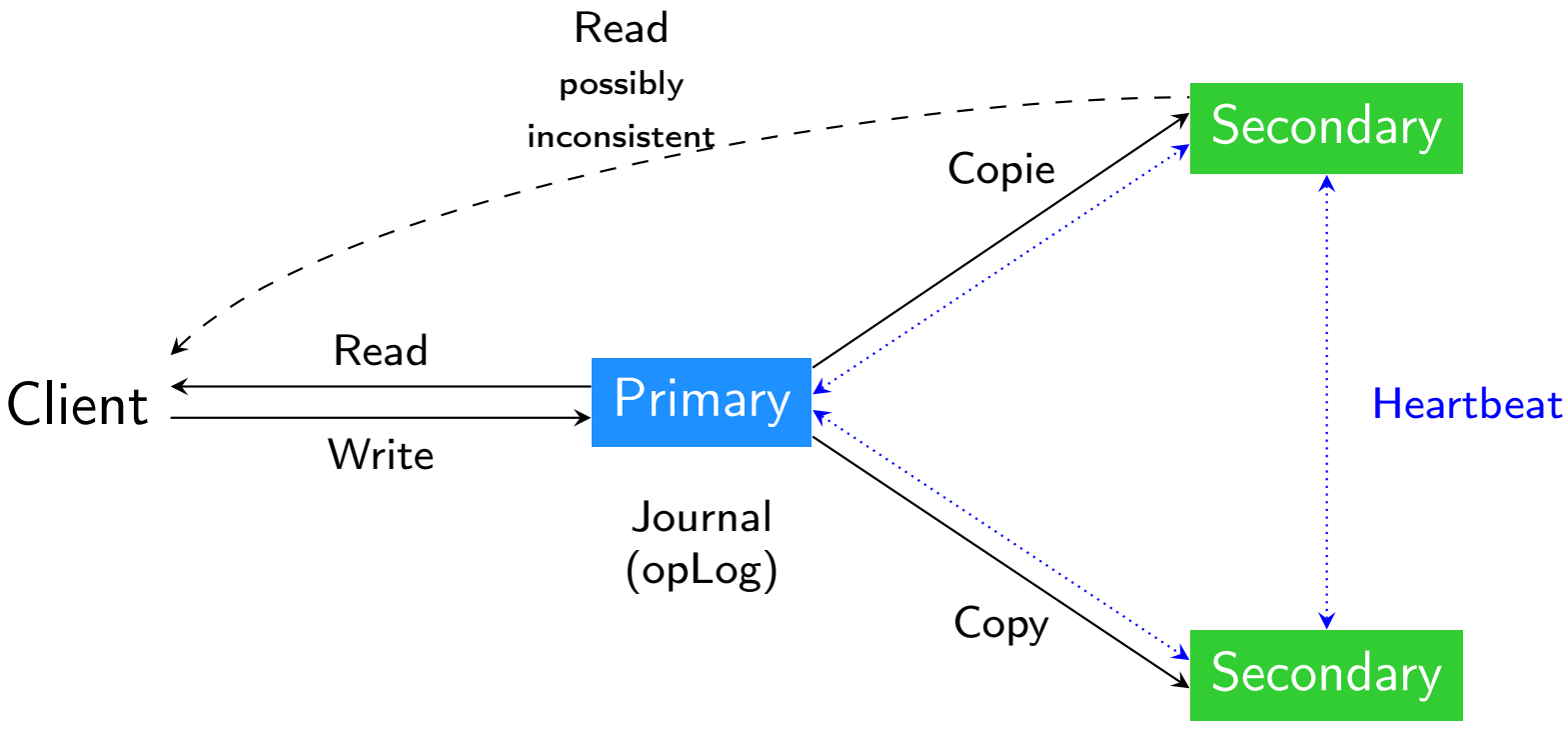
Generally, replication distributes the load (reads). But not in MongoDB with default setting.

Periodically (ex: *heartbeat*=10s), nodes exchange short messages (heartbeat) to check availability. If primary fails, secondary elect a new one (majorité absolue).

In practice (Jan 2022):

- default replica set = 3 nodes: 1 master, 2 copies
- heartbeat+election take 10-30s, so $\leq 1min$ to recover the set.
- max 50 nodes, 7 of which votes.
- many parameters can be tuned: non-voting, non electable, arbiter for elections.....
- by default, all reads are performed on the primary only!

Replication (2)



Replication in MongoDB.

- opLog collection is copied
- asynchronous copy, in batch and multithreaded (ex: grouped by document_id)
- copies may change their sync targets based on ping times and states.

Sharding

. . . (almost)just another word for partitioning (horizontal)

MongoDB Sharding:

- partitions a collection
- by default, range partitioning
- can use hash partitioning
  (just hashed through MD5 each key, then interval partitioning on hashes)
- specify partitioning key and nodes (cf architecture):
  - chunk is split if exceeds a given size ($\simeq$real time)
  - chunks balanced when #chunks too much .

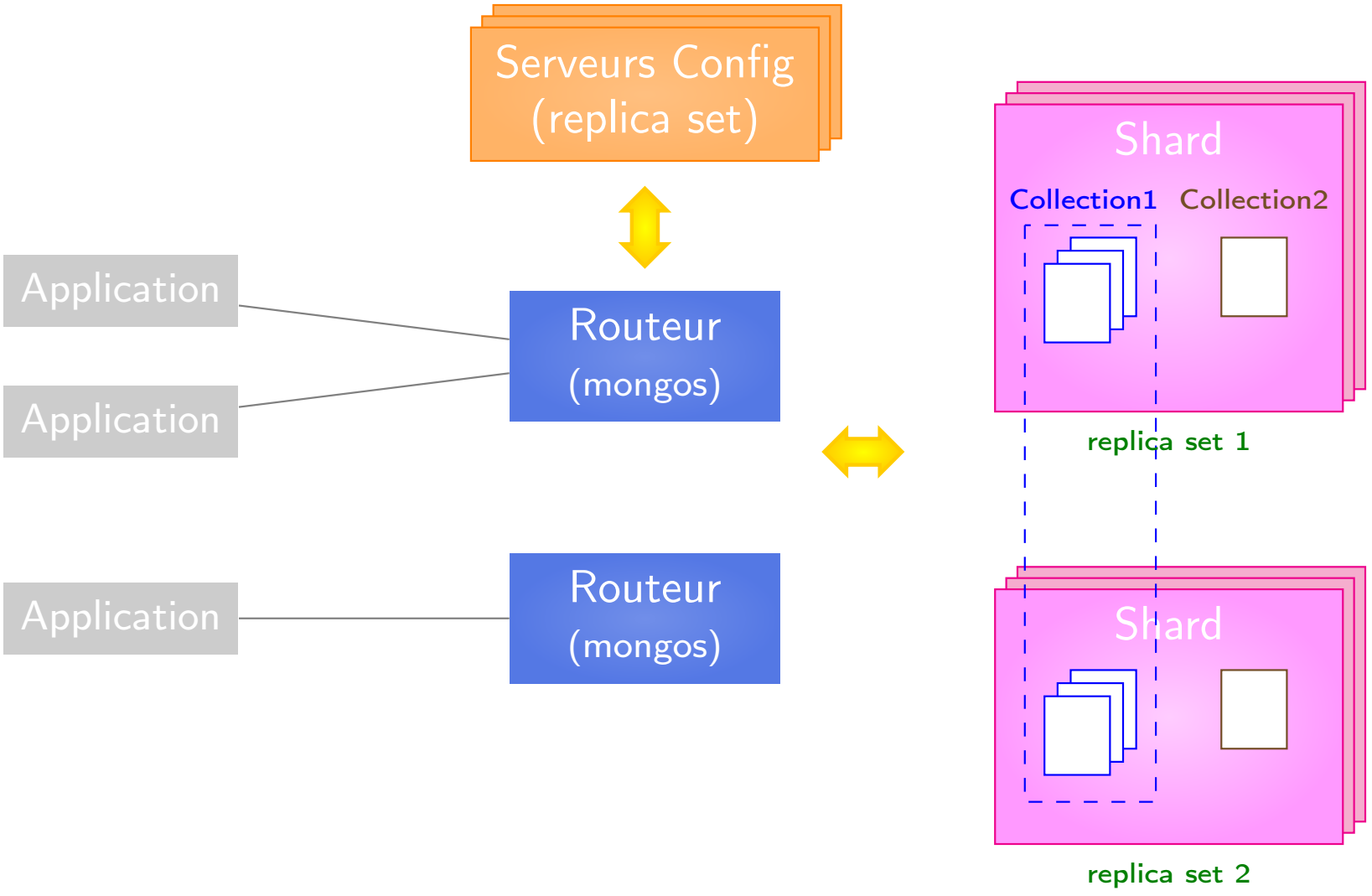
Architecture:

- each chunk stored independently on some replica set
- routeurs called "mongos" distribute queries, merge results

Careful when you pick the key!

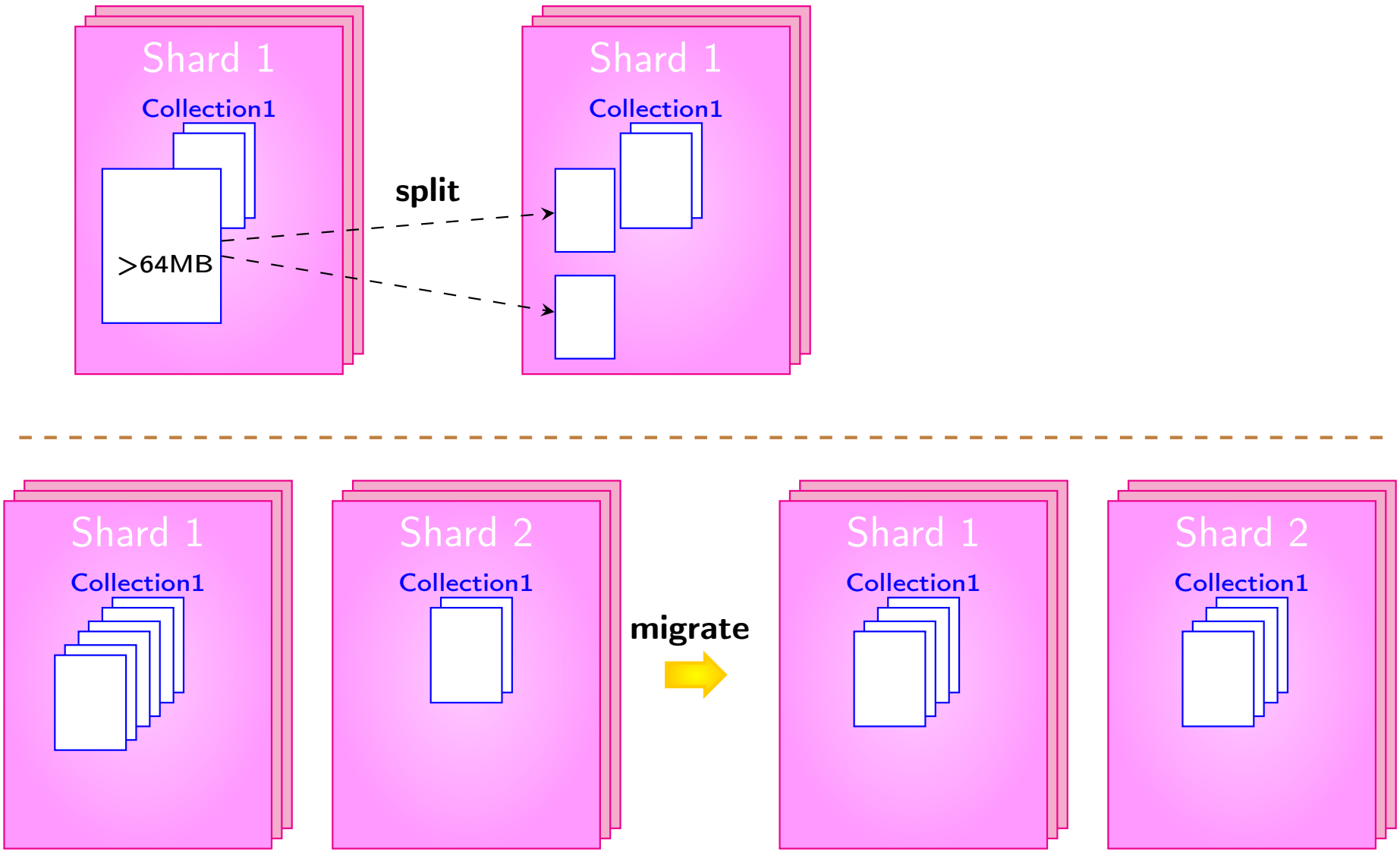
Cardinality, distributing the (write) load, index...

Sharding: architecture



Partitionning (sharding) in MongoDB: architecture.

Sharding: balancing shards



Partitionning (sharding) in MongoDB: splitting a chunk, migrating chunks.

MongoDB: concurrency, failure recovery

. . . for WiredTiger engine

Operations on a document are atomic.

Engine uses locks (on collections, db, globally).

Internally, engine uses MVCC. (NB MongoDB finally supports multidoc transactions since v4).

Checkpoint every 60s: data saved on disk

journal : a *write-ahead log* records transactions between 2 checkpoints.
(compressed with `snappy`)

MongoDB: indexes

Types of index:

- simples or composite: B-trees
  can also enforce unicity...
- (geo)spatial index (2d)
  queries such as : closest points, points in a given area...
- text: word list, adter stemming)
- hash: hashing a field (incl. content)

```
// Composite indexes use lexicographic order
db.movies.createIndex( { "acteurs.nom": 1, "acteurs.prenom": -1 } )
```

Since version 3.2, can define a partial index.

```
// Indexes only docs that satisfy condition:
db.movies.createIndex(
  { "acteurs.nom": 1, "acteurs.prenom": 1 },
  { partialFilterExpression: { annee:  {$gt: 2000} } }
)
```

Interact with MongoDB

Drivers for all common languages
(C++,Java,Python,Scala,Node.js,PHP...).
Some ODM (generally "community supported")

```
#Official Python client: pymongo

from pymongo import MongoClient

myconnection = MongoClient('mongodb://localhost:27917')
db = myconnection.movie_database
movies = db.movie_collection
movie1 ={"author": "Mike", "text": ..."}
movie2 ={"author": "Mary", "text": ..."}
newids = movies.insert_many([movie1,movie2])
newids.inserted_ids      #[ObjectId('...'), ObjectId('...')]

import pprint
for mov in movies.find({"author": "Mike"}):
    pprint.pprint(mov)
```

References

- MongoDB

  https://docs.mongodb.com/manual/
  http://b3d.bdpedia.fr/bddoc.html
  https://buzut.fr/commandes-de-base-de-mongodb/

  https://www.slideshare.net/mongodb/sharding-v-final (sharding MongoDB, avec exemples réels)