



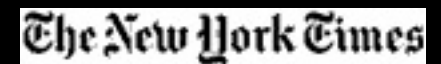
Full Metal Mongo



{name: "mongo", type: "DB"}

- Humongous: *Slang*. Extraordinary large; expressive coinage, perhaps reflecting huge and monstrous, with stress pattern of tremendous
- Open source NoSQL database
 - Written in C++
 - <https://github.com/mongodb/mongo>

Production Deployments



Outline

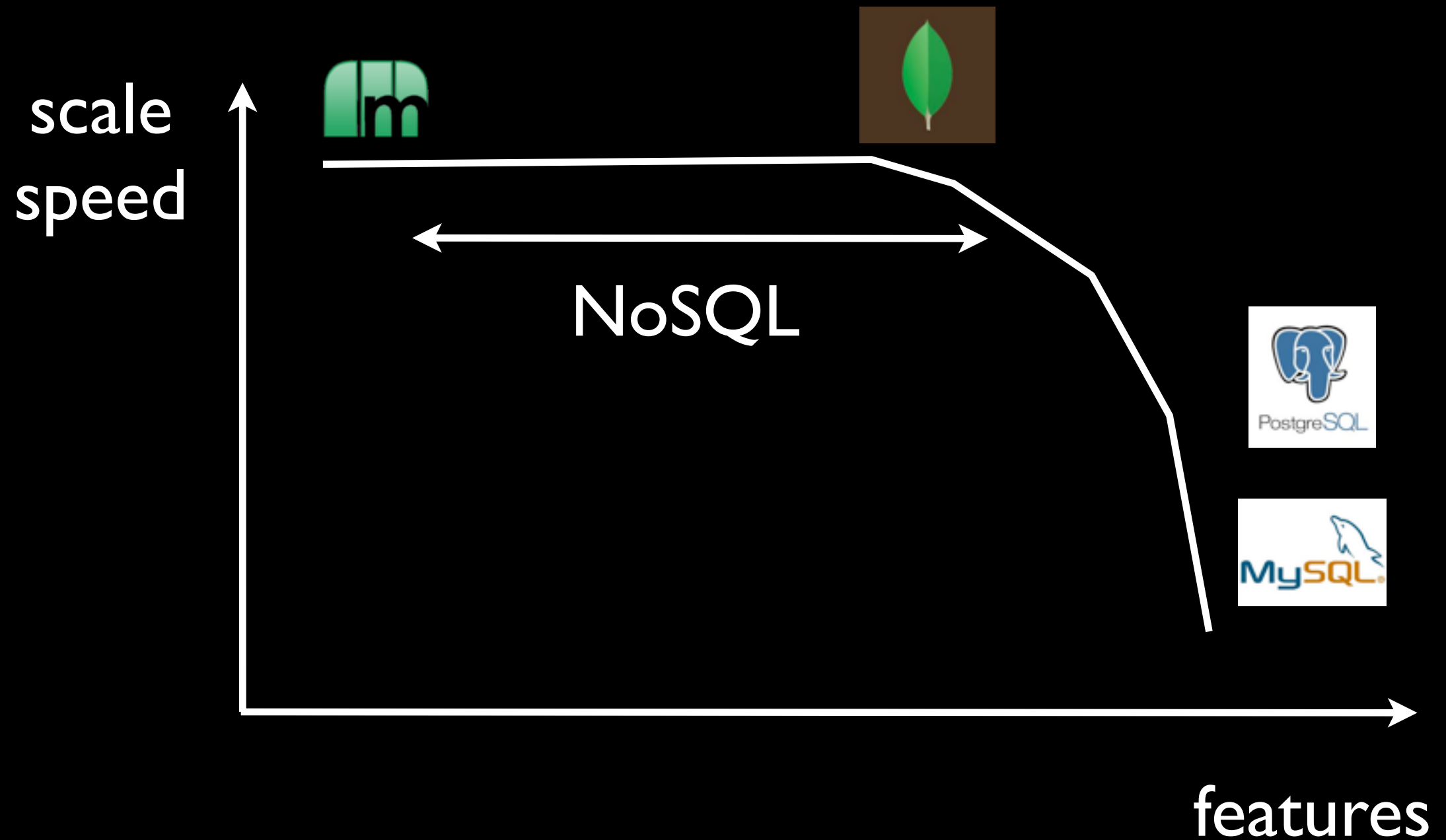
- Terminology and basics
- The mongo shell
- Insert / update / delete
- Querying
- Aggregation
- Map/reduce
- Schema design
- Indexes
- DBA stuff
- Security
- Replica sets
- Sharding

Terminology and basics

Terminology

- NoSQL is almost everything
- Schemaless is nonsense : mongoDB do have a schema
 - Flexible
 - But a schema

Scaling out



Format

- BSON: Binary encoded serialization of JSON documents
- Characteristics
 - Lightweight: minimum overhead
 - Traversable
 - Efficient: encoding and decoding

JSON

```
{
  _id : ObjectId('xxxxxx'),
  name : 'Full Metal Mongo',
  date : Date(),
  presenter: 'isra',
  attendants : [
    {name: 'ana', age: 23},
    {name: 'luis', age: 32}
  ]
}
//default _id: 24 hex chars
```

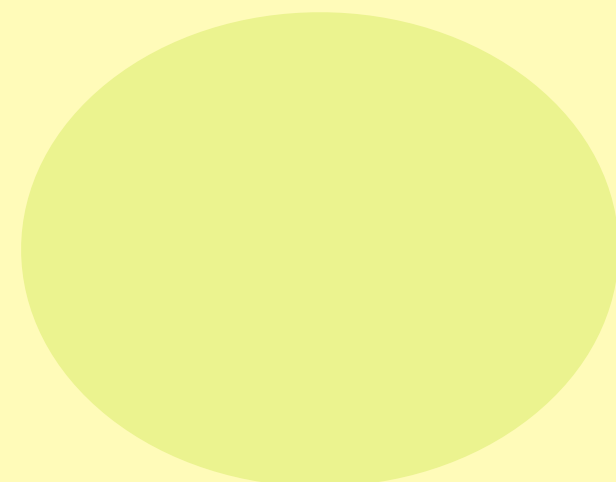
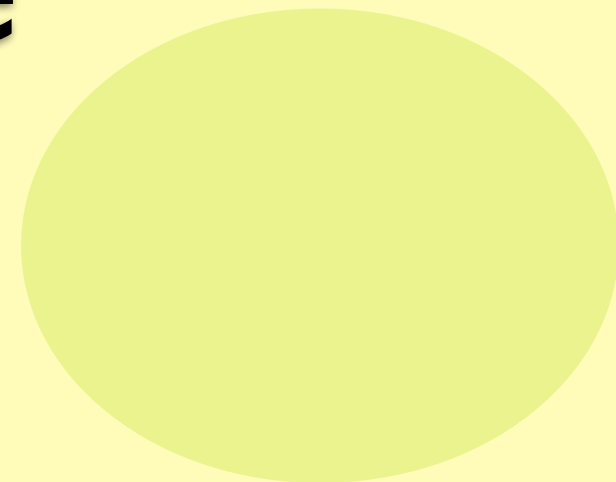
Data schema

Database

Collection

Document

```
{ user: 1, name: [] }
```



Collection

- Flexible: no fixed structure
 - ~~ALTER TABLE~~ (implicit)
- Created in the first insertion (same for dbs)
- Capped collection: maintain insert order, fixed size

Document

- JSON document
 - `_id` (ObjectId)
 - unique for the collection
 - it can be a document itself
 - Fields: numeric, string, date
 - Arrays and subdocuments

SQL to Mongo mapping

MySQL executable	Oracle executable	Mongo executable
mysqld	oracle	mongod
mysql	sqlplus	mongo

MySQL term	Mongo term/concept
database	database
table	collection
index	index
row	BSON document
column	BSON field
join	embedding and linking
primary key	_id field
group by	aggregation

MongoDB basics

- Default port: 27017
- Optional authentication
- Data location: /data/db/
- Modes
 - automatic replication
 - automatic fail-over

Drivers

- Officially supported
 - C, C++, Erlang, Haskell, Java, Javascript, .NET, Perl, PHP, Python, Ruby, Scala
- Community supported
 - ActionScript, C#, Delphi, etc.
- <http://api.mongodb.org/>

Connection

- `mongodb://username:password@host:port/database?options`
 - username and password are optional
 - port: 27017 by default
 - database: admin database by default
 - options: 'name=value' pairs

The mongo shell



Hands on: let's get started

- Run a mongod (--fork) instance
- Run a mongo shell (mongo) that connects to this instance



The mongo shell: basics

- `show dbs`
- `use db_name`
- `show collections (current db)`
- `show users (current db)`

Insertion

Suppose a collection of GUL courses.

```
db.courses.insert ({
  name : 'Full Metal Mongo',
  date : new Date(),
  presenter: 'isra',
  attendants : [
    {name: 'ana', age: 23},
    {name: 'luis', age: 32}
  ]
})
```

Querying

```
//Full Metal Mongo course
```

```
db.gul.find({name: 'Full Metal Mongo'})
```

```
//Courses attended by ana
```

```
db.gul.find({attendants.name: 'ana'})
```

```
//Course names given by isra
```

```
db.gul.find({presenter: 'isra'}, {name: 1})
```

Querying II

```
//Courses ordered by name  
db.gul.find().sort({name:1});
```

```
//The first 5 courses  
db.gul.find().limit(5);
```

```
//Next five courses  
db.gul.find().skip(5).limit(5);
```

```
//First course (natural order)  
db.gul.findOne()
```

Querying III

```
//Courses attended by any under-age  
db.gul.find({attendants.age:{$lt:18}});
```

```
//Last year courses between Monday and Thursday  
db.gul.find({date:{  
  $gt:new Date(2012,03,08) ,  
  $lt:new Date(2012,03,11) }  
});
```

Querying IV

```
//Courses attended by pedro or ana  
db.gul.find({'attendants.name':  
    {'$in':['pedro', 'ana']}  
});
```

```
//Courses attended by 10 people  
db.gul.find({'attendants':  
    {'$size:10'}  
});
```

\$ operators

- \$in / \$nin
- \$all (default is any)
- \$gt(e) / \$lt(e)
- \$ne
- \$elemMatch
(conditions in the same subdoc)
- \$exists
- \$regex
- \$natural (order)
- \$toLower / \$toUpper

More \$ expressions

- \$sum
- \$avg
- \$min
- \$max
- \$push (insert)
- \$addToSet (insert)
- \$first (sort)
- \$last (sort)

Update

```
//updates if exists; inserts if new  
db.gul.save(x)
```

```
//update speakers in the crafty course  
db.gul.update(  
  {name: 'Crafty'},  
  {$set: {presenter: ['javi', 'isra']}}  
);
```

```
//new attendant to a course (not multi)  
db.gul.update(  
  {name: 'mongoDB'},  
  {attendants:  
    {$push: {name: 'pepe', age: 19}}  
  }  
);
```

Find and Modify

- `findAndModify` (not widely used)

Argument	Description	Default
<i>query</i>	a filter for the query	<code>{}</code>
<i>sort</i>	if multiple docs match, choose the first one in the specified sort order as the object to manipulate	<code>{}</code>
<i>remove</i>	set to a true to remove the object before returning	N/A
<i>update</i>	a modifier object	N/A
<i>new</i>	set to true if you want to return the modified object rather than the original. Ignored for remove.	false
<i>fields</i>	see Retrieving a Subset of Fields (1.5.0+)	All fields
<i>upsert</i>	create object if it doesn't exist; a query must be supplied! examples (1.5.4+)	false

Remove

```
//removes all  
db.gul.remove()
```

```
//search and remove  
db.gul.remove({presenter: 'isra'})
```

Database references: direct linking

```
//Query
```

```
isra = db.gul_members.findOne()
```

```
//Response from the query
```

```
{_id: ObjectId('ad234fea23482348'),  
name: 'isra', age: 31, languages: 'js'}
```

```
//Find by id
```

```
db.gul.find({'attendants._id':isra._id})
```

Database references:

DBRef

```
//Query
```

```
isra = db.gul_members.findOne()
```

```
//Response
```

```
{_id: ObjectId('ad234fea23482348'),  
name: 'isra', age: 31, languages: 'js'}
```

```
//Insert by DBRef
```

```
db.gul.insert({  
  name: 'mongoDB',  
  presenter: new DBRef('gul_members',isra._id)  
})
```



Import example data

- Download a short courses collection from
- <http://www.it.uc3m.es/igrojas/mongo/initDB.json>

```
//Import dataset in JSON  
mongoimport --db gul --collection courses initDB.json
```



Hands on: querying

- Add a new course with data similar to the existing
- Update your course to add attendants
- Query courses with speaker “Jesús Espino”
- Query course on Friday
- Query courses tagged as “android”

Aggregation

`db.gul.aggregate([ pipeline ])`

- Pipelines (7)
 - \$match (n:1)
 - \$project (1:1)
 - \$group (n:1)
 - \$order (1:1)
 - \$limit (n:1)
 - \$skip (n:1)
 - \$unwind (1:n)

Examples: <http://docs.mongodb.org/manual/reference/sql-aggregation-comparison/>

Aggregation I

```
//Number of courses  
db.gul.count();
```

```
//Number of courses given by isra  
db.gul.count({presenter: 'isra'});
```

```
//Distinct attendants to all courses  
db.gul.distinct('attendants.name');
```

Aggregation II

```
db.grades.aggregate([
  {$unwind: "$scores"},
  {$match: {"scores.type": {$ne: "quiz"}}},
  {$group: {
    _id: {class_id: "$class_id",
      student_id: "$student_id",
      score: {$avg: "$scores.score"}}
  }},
  {$group: {
    _id: {class_id: "$_id.class_id"},
    score: {$avg: "$score"}
  }},
  {$sort: {score: -1}}
])
```



Hands on: aggregation

- Distinct course speakers
- Distinct tags and count
- Number of courses per weekday

Map/Reduce

- Batch processing of data and aggregation operations
- Where GROUP BY was used in SQL
- Input from a collection and output going to a collection

Map/reduce (II)

- Courses attended per individual

```
var map = function() {  
    for(var i in this.attendants) {  
        emit(this.attendants[i].name, 1) ;  
    }  
}
```

Map/reduce (III)

- Courses attended per individual

```
var reduce = function(key, values) {  
    var sum=0;  
    for (var i in values) {  
        sum+=values[i];  
    }  
    return sum;  
}
```

Map/reduce (IV)

- Courses attended per individual

```
db.gul.mapReduce({  
  map: map,  
  reduce: reduce,  
  {out: {inline:1}},  
  query:{initial_query}}  
});
```



Hands on: map/reduce

- Update the some courses to add attendants
- Get all the courses attended by individual
- Distinct tags and count

Schema design

Schema Design

- Function of the data and the **use case**
- Decisions
 - # of collections
 - Embedding or linking
 - Indexes
 - Sharding

Relationships

- Types
 - 1:1 (person:resume)
 - 1:n (city:person, post:comments)
 - m:n (teacher:student)
- Doc limit: 16MB
- Examples: school, blog

Transactions

- No transactions
 - Redesign schema
 - Implement in SW
 - Tolerate no transactions



Schema design: examples

- Let's design the schema for
 - courses
 - school
 - blog / twitter
 - foursquare

Indexes

Indexes

- Objective: Query optimization
- Used in the query itself and/or the ordering
- B-Tree indexes
- `_id` index is automatic (unique)

```
db.gul.ensureIndex({ name:1 })
```

```
db.gul.getIndexes()
```

```
db.gul.stats() //Size of the index
```

Indexes (II)

- For arrays, the index is **multkey** (one index entry per array element)
- Field names are not in indexes

```
//Compound indexes
```

```
db.gul.ensureIndex({ name:1, age:1})
```

```
//For nested fields (subdocs)
```

```
db.gul.ensureIndex({ attendants.name:1 })
```

Indexes types

- default
- unique

```
db.gul.ensureIndex({name:1}, {unique:1})
```

- sparse

```
db.gul.ensureIndex({name:1}, {sparse:1})
```

- TTL (time to live)
- geospatial

Indexes options

- dropDups: drop duplicate keys when creating the index (converted in unique)
- background: created in the background on primary of the replica set, in the foreground on secondaries

More about Indexes

- Covered index
 - query covered completely by the index
- Selectivity of an index
- Explain

```
db.gul.find().explain()
```

- Hints

```
db.gul.find().hint({name:1})
```

Geospatial indexes

- 2d-only
- compound indexes may be used

```
db.places.ensureIndex({'loc': '2d'})
```

```
db.places.find({loc: {  
  $near: [20, 40],  
  $maxDistance: 2}  
}).limit(50)
```



Creating indexes: examples

- Optimize our courses database
 - Think of common queries
 - Implement the convenient indexes

DBAs stuff

Backups

- mongodump / mongorestore
- copy files using your own software
(journaling enabled required)
- replica sets: backup from secondary

Commands

```
db.gul.runCommand('compact')
```

```
db.runCommand({compact: 'gul'})
```

```
//Run a script from the command line  
mongo < path/to/script.js
```

Profiler

- Log queries / commands

```
mongod --profile 0/1/2 --slowms 100
```

```
//0: no
```

```
//1: slow queries
```

```
//2: all queries
```

```
//slowms: threshold for type 1
```

Profiler (II)

- From the mongo shell

```
db.getProfilingLevel() // 0-1-2
```

```
db.getProfilingStatus() // { "was" : 0,  
"slowms" : 100 }
```

```
db.setProfilingLevel(1,1000)
```

- Data stored in system.profile collection

Kill operations

- `db.currentOp()`
 - in progress operations
- `db.killOp(op_id)`
- Don't kill
 - write ops in secondaries
 - compact
 - internal ops

Commands for dbas

- mongotop
 - time of activity per collection
 - info about total, read, write, etc.
- mongostat (command line)
 - every x seconds
 - info about insert, update, delete, getmore, command, flushes, mapped, vsize, res, faults, etc.

Security tips

Security

- mongod/mongos --auth //not from localhost
- Add user
 - use admin
 - db.addUser(user, passwd, [readOnly])
- Auth
 - use admin
- db.auth(user, passwd)

Types of users

- admin
 - created in the admin db
 - access to all dbs
- regular
 - access a specific db
 - read/write or readOnly

Intra-cluster security

- For replica sets, to use non-auth (faster) communications among the nodes
- `mongod --keyFile file --replSet`

Replica sets

What is a replica set?

- Info replicated among several nodes
- 1 primary
- n secondaries (min 3, to get a majority)
- When a node falls, there's election and a majority is needed to select a new primary

Types of nodes in a replica set

- Regular
- Arbiter: decides the primary in a election
- Delayed: cannot be elected primary
- Hidden: used for analytics (not primary)

Replica set configuration

```
rs.config({ _id: 'rs_name',  
  members: [{_id:0, host:host0}, {_id:1,  
host: host1}, {_id:2, host: host2}]}))
```

```
rs.status()
```

```
rs.slaveOk() //read from secondaries
```

```
rs.isMaster() //check primary
```

Write concern

- Journal: list of operations (inserts, updates) done, saved in disk (permanent)
- getLastError (managed by the driver)
 - w: wait until write is saved in memory (the app receives ack) Used to detect errors, like violation of a unique.
 - j: wait until write is saved in the journal

Oplog and write concern

- oplog.rs: capped collection with the operations made in the replica set, stored in natural order
- write concern
 - w: n, means wait response of n nodes in a replica set
 - w: 'majority', wait for the majority of the nodes

Sharding

What is sharding?

- Scalability
- Horizontal partitioning of a database
- A BSON document stored in ONE shard
- Shard key
 - Not unique
 - No unique fields in the collection
- Mongo offers auto-sharding

What is sharding?

- Auto balancing
- Easy addition of new machines
- Up to 1k nodes
- No single point of failure
- Automatic failover
- Select a convenient shard key

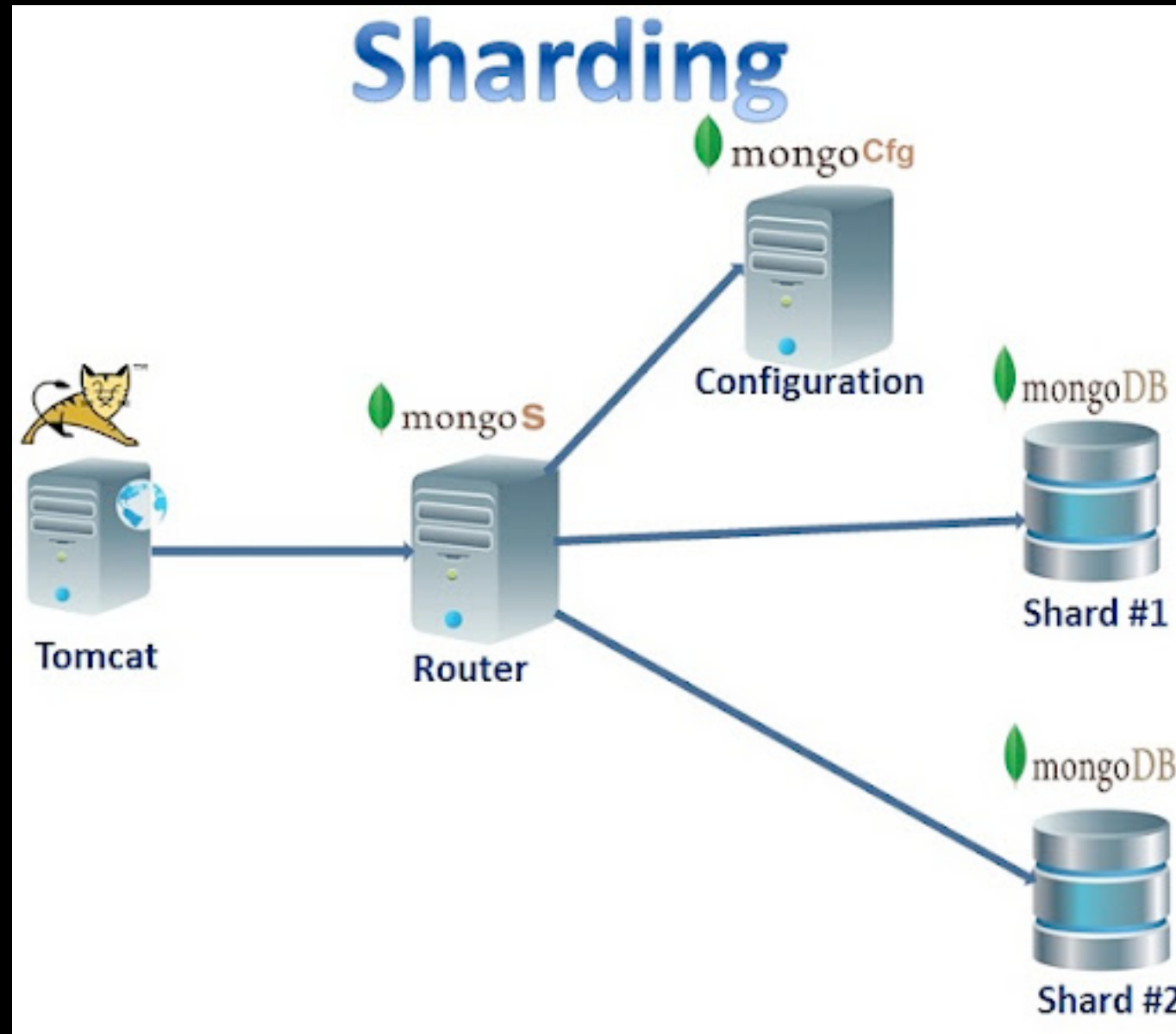
Sharding config

- Need of config servers
 - store metadata about chunks
 - mongod --configsvr
- Need mongod “routers”
 - mongos (accessed by the apps)

Sharding operations

- chunk: range of the sharding key being in a shard
- operations
 - split: dividing a chunk to balance the size of the chunks
 - migrate: moving a chunk from a shard to another

Sharding diagram



via: http://www.cloudifysource.org/2012/03/25/petclinic_deepdive.html



Shard key selection

- Examples: choose the shard key for
 - courses
 - school
 - blog / twitter
 - foursquare

References

- MongoDB devel docs: <http://www.mongodb.org/display/DOCS/Developer+Zone>
- MongoDB FAQ: <http://www.mongodb.org/display/DOCS/Developer+FAQ>
- MongoDB cookbook: <http://cookbook.mongodb.org/>

References

- Kyle Banker's blog:
 - Aggregation: <http://kylebanker.com/blog/2009/11/mongodb-count-group/>
 - e-Commerce example: <http://kylebanker.com/blog/2010/04/30/mongodb-and-ecommerce/>
- mongodb MOOCs (dbas and developers)
 - <http://education.10gen.com>

Thank you very much!

Any questions?