

# Using MongoDB Responsibly



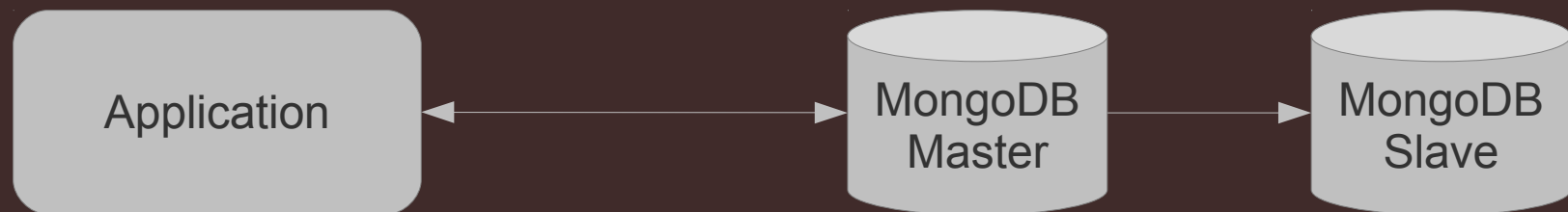
Jeremy Mikola  
[jmikola.net](http://jmikola.net)

# Topics

- Infrastructure
- Concurrency
- Indexing
- Query Optimization
- General Advice
- Case Study
- Map/Reduce
- Aggregation Framework
- Doctrine ODM
- MMS

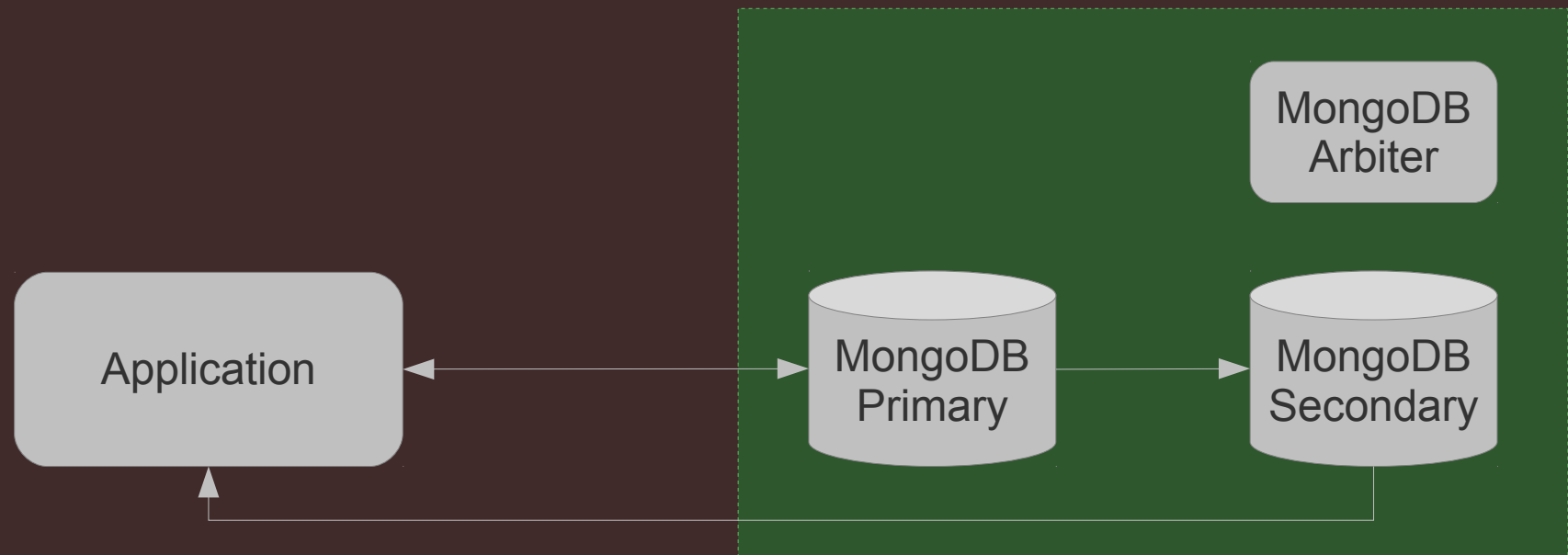
# Infrastructure: Master/Slave

- Deprecated in favor of replica sets
- **slaveOk** allows queries to target slave
- Manual failover process



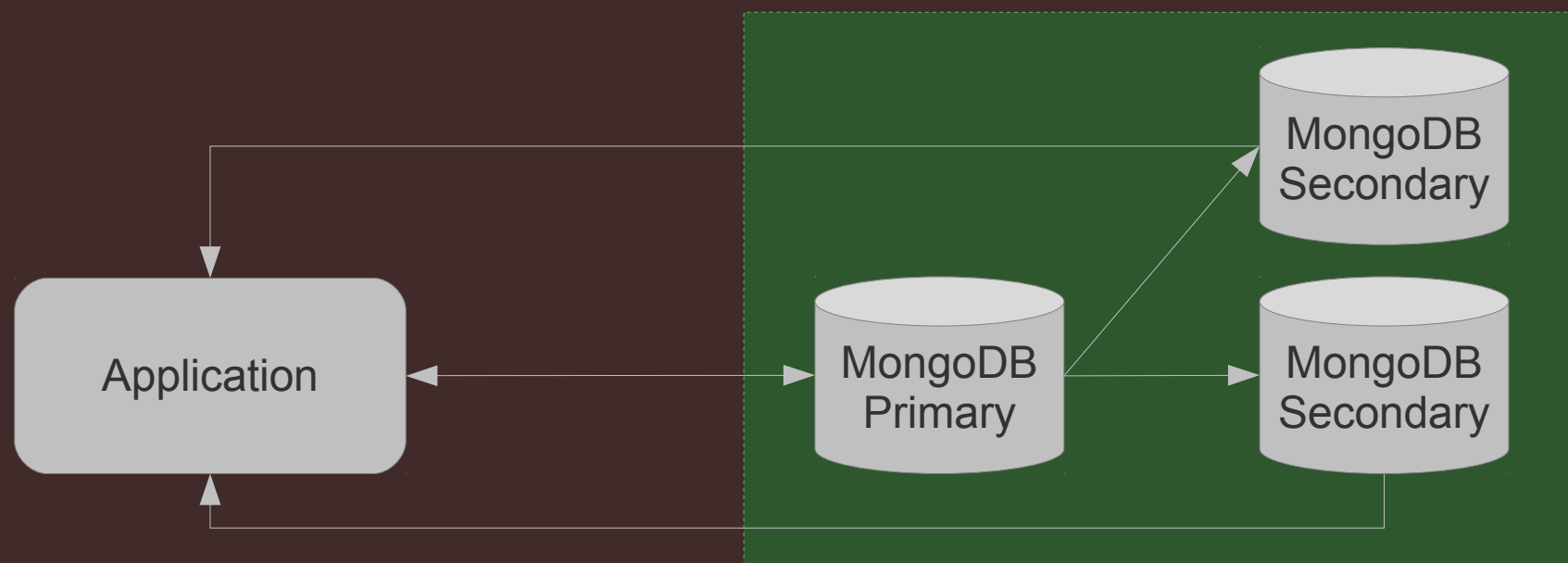
# Infrastructure: Replica Set

- Primary, secondary and arbiter
- **slaveOk** allows queries to target secondary
- Automatic failover

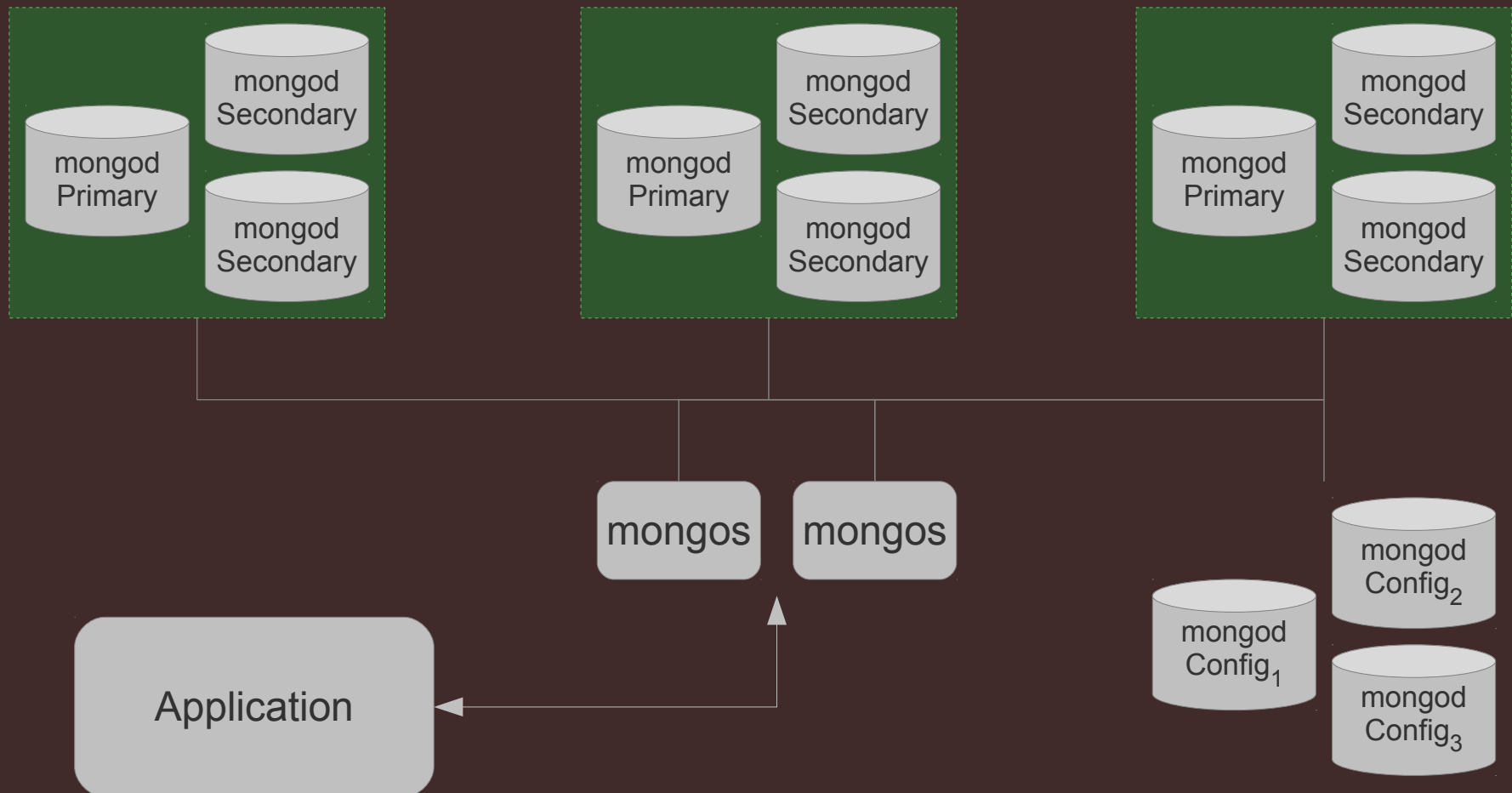


# Infrastructure: Replica Set

- Primary with two secondaries
- Arbiter unnecessary for odd number of nodes



# Infrastructure: Sharding



# Infrastructure: Sharding

- **mongos** processes
  - Route queries to shards and merges results
  - Lightweight with no persistent state
- Config servers
  - Launched with **mongod --configsvr**
  - Store cluster metadata (shard/chunk locations)
  - Proprietary replication model

# Sharding vs. Replication

Sharding is the tool for scaling a system.

Replication is the tool for data safety, high availability, and disaster recovery.

Source: [Sharding Introduction \(MongoDB docs\)](#)



# Concurrency: Locks

- Read/write locks yielded periodically
  - Long operations (queries, multi-document writes)
  - Page faults (2.0+)
- Write locks
  - Greedy acquisition (priority over read locks)
  - Global or database-level (2.2+)
  - Collection-level forthcoming (**SERVER-1240**)

# Concurrency: JavaScript

- JavaScript executes in a single mongod thread
  - `$where` queries
  - `db.eval()` commands
  - Map/reduce
- SpiderMonkey JS interpreter
  - Single-threaded
  - Possible multi-threading with V8 (`SERVER-4258`)

# Concurrency: JavaScript

- `db.eval()` takes a write lock by default
  - Cannot execute other blocking commands
  - Can be used to perform atomic operations
    - Swapping two collection names
    - Complex find/modify
- Executing JS without blocking the node
  - `{nolock: true}` option with `db.runCommand()` (1.8+)
  - Use `mongo` command-line client

# Concurrency: Map/Reduce

- JavaScript functions (lock yielded between calls)
- Collection reads (lock yielded every 100 documents)
- Write locks for incremental result storage
  - Temporary collection used between map and reduce
  - `jsMode` flag may bypass this for small datasets
- Write lock for atomic output of final collection
  - `merge` and `reduce` modes can take longer than `replace`
  - Consider `{nonAtomic: true}` output option (2.2+)

# Concurrency: Indexing

- Foreground indexing
  - Default for index creation
  - Blocks all other database operations
- Background indexing
  - Use the `{background: true}` option with `ensureIndex()`
  - Slower than foreground indexing, but doesn't block
  - Watch `db.currentOp()` to track progress

# Concurrency: Indexing

- Replication uses foreground indexing (pre-2.2)
  - Manually swap out members for indexing
  - Documentation: [Building Indexes with Replica Sets](#)

# Monitoring Foreground Indexing

```
> db.currentOp()
{
  "inprog" : [
    {
      "opid" : 10000054,
      "active" : true,
      "lockType" : "write",
      "waitingForLock" : false,
      "secs_running" : 4,
      "op" : "insert",
      "ns" : "test.system.indexes",
      "query" : {},
      "client" : "127.0.0.1:52340",
      "desc" : "conn",
      "threadId" : "0x7f4ce7f50700",
      "connectionId" : 1,
      "msg" : "index: (1/3) external sort 3685454/10000000 36%",
      "progress" : {
        "done" : 3685457,
        "total" : 10000000
      },
      "numYields" : 0
    }
  ]
}
```

# Monitoring Foreground Indexing

```
> db.currentOp()
{
  "inprog" : [
    {
      "opid" : 10000054,
      "active" : true,
      "lockType" : "write",
      "waitingForLock" : false,
      "secs_running" : 15,
      "op" : "insert",
      "ns" : "test.system.indexes",
      "query" : {},
      "client" : "127.0.0.1:52340",
      "desc" : "conn",
      "threadId" : "0x7f4ce7f50700",
      "connectionId" : 1,
      "msg" : "index: (2/3) btree bottom up 1721606/10000000 17%",
      "progress" : {
        "done" : 1721606,
        "total" : 10000000
      },
      "numYields" : 0
    }
  ]
}
```



# Monitoring Foreground Indexing

```
> db.currentOp()
{
  "inprog" : [
    {
      "opid" : 10000054,
      "active" : true,
      "lockType" : "write",
      "waitingForLock" : false,
      "secs_running" : 25,
      "op" : "insert",
      "ns" : "test.system.indexes",
      "query" : {},
      "client" : "127.0.0.1:52340",
      "desc" : "conn",
      "threadId" : "0x7f4ce7f50700",
      "connectionId" : 1,
      "msg" : "index: (3/3) btree-middle",
      "numYields" : 0
    }
  ]
}
```

# Monitoring Background Indexing

```
> db.currentOp()
{
  "inprog" : [
    {
      "opid" : 10000075,
      "active" : true,
      "lockType" : "write",
      "waitingForLock" : false,
      "secs_running" : 12,
      "op" : "insert",
      "ns" : "test.system.indexes",
      "query" : {},
      "client" : "127.0.0.1:52340",
      "desc" : "conn",
      "threadId" : "0x7f4ce7f50700",
      "connectionId" : 1,
      "msg" : "bg index build 3258205/10000000 32%",
      "progress" : {
        "done" : 3258206,
        "total" : 10000000
      },
      "numYields" : 53
    }
  ]
}
```

# Background Indexing: PHP

```
$mongo = new MongoDB();  
$collection = $mongo->example->foo;  
  
$collection->ensureIndex(  
    array('bar' => 1),  
    array('background' => true)  
);
```

Not to be confused with the **safe** option, which blocks until the operation succeeds or fails

# Background Indexing: PHP

```
> db.foo.count()
1000000
> db.foo.find()
{ "_id" : ObjectId("4fc5136b22f0e13f6f000000"), "x" : 1 }
{ "_id" : ObjectId("4fc5136b22f0e13f6f000001"), "x" : 2 }
{ "_id" : ObjectId("4fc5136b22f0e13f6f000002"), "x" : 3 }
{ "_id" : ObjectId("4fc5136b22f0e13f6f000003"), "x" : 4 }
```

```
$ php benchmark.php
Insertion took 17.095013 seconds
Indexing with [] took 0.000175 seconds
Indexing with {"background":true} took 0.000159 seconds
Indexing with {"safe":true} took 1.649953 seconds
Indexing with {"background":true,"safe":true} took 3.877397 seconds
```

Benchmarking single-field index generation with **safe** and **background** options (<https://gist.github.com/2829859>)

# Background Indexing: ODM

```
<?php

use Doctrine\ODM\MongoDB\Mapping\Annotations as ODM;

/**
 * @ODM\Document(collection="foo")
 * @ODM\Indexes({
 *     @ODM\Index(keys={"x"="asc"}, options={"background"="true"})
 * })
 */
class Foo
```

```
$ app/console doctrine:mongodb:schema:create --index
Created index for all classes
```

Doctrine ODM adds **safe** by default (since 06d8bb1)

# Indexing: Advice

- Kill 2+ ~~birds~~ queries with one ~~stone~~ index
- Compound key and multi-key indexes
- Avoid single-key indexes with low selectivity
- Mind your read/write ratio
- `$exists`, `$ne` and `$nin` can be inefficient
- When in doubt, `explain()` your cursor

# Indexing: Memory Usage

```
$ free -b
```

	total	used	free	shared	buffers	cached
Mem:	7307489280	6942613504	364875776	0	229281792	5872500736
-/+ buffers/cache:	840830976	6466658304				
Swap:	0	0	0			

```
$ mongo example --quiet
> var s = 0
0
> for each (var c in db.getCollectionNames()) {
... s += db[c].totalIndexSize();
... }
9784055680
```

7.3GB of RAM cannot hold 9.7GB of indexes, so expect intermittent page faults and disk access

# Query Optimization with `explain()`

```
> for (i=0; i<1000000; i++) db.foo.insert({ x:i, y:Math.random() });

> db.foo.count()
1000000

> db.foo.find({x:5}).explain()
{
  "cursor" : "BasicCursor",
  "nscanned" : 1000000,
  "nscannedObjects" : 1000000,
  "n" : 1,
  "millis" : 301,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "isMultiKey" : false,
  "indexOnly" : false,
  "indexBounds" : {},
  "server" : "localhost:27017"
}
```



# Query Optimization with `explain()`

```
> for (i=0; i<1000000; i++) db.foo.insert({ x:i, y:Math.random() });
```

```
> db.foo.count()  
1000000
```

```
> db.foo.find({x:5}).explain()  
{  
  "cursor" : "BasicCursor",  
  "nscanned" : 1000000,  
  "nscannedObjects" : 1000000,  
  "n" : 1,  
  "millis" : 301,  
  "nYields" : 0,  
  "nChunkSkips" : 0,  
  "isMultiKey" : false,  
  "indexOnly" : false,  
  "indexBounds" : {},  
  "server" : "localhost:27017"  
}
```

- Table scan or index-enabled?
- Documents + index entries scanned
- Documents scanned
- Documents matched
- Query time
- Read lock yields
- Docs skipped due to active chunk migrations
- Was multi-key index used? (array values)
- Was query covered by the index?
- Key bounds used in index scanning

# Query Optimization with `explain()`

```
> db.foo.ensureIndex({x:1, y:1})

> db.foo.find({x:5}).explain()
{
  "cursor" : "BtreeCursor x_1_y_1",
  "nscanned" : 1,
  "nscannedObjects" : 1,
  "n" : 1,
  "millis" : 0,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "isMultiKey" : false,
  "indexOnly" : false,
  "indexBounds" : {
    "x" : [ [5, 5] ],
    "y" : [ [{"$minElement" : 1}, {"$maxElement" : 1}] ],
  },
  "server" : "localhost:27017"
}
```

# Query Optimization with `explain()`

```
> db.foo.find({x:5, y:{ $gt:0.5}}).sort({y:1}).explain()
{
  "cursor" : "BtreeCursor x_1_y_1",
  "nscanned" : 1,
  "nscannedObjects" : 1,
  "n" : 1,
  "millis" : 0,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "isMultiKey" : false,
  "indexOnly" : false,
  "indexBounds" : {
    "x" : [ [5, 5] ],
    "y" : [ [0.5, 1.7976931348623157e+308] ],
  },
  "server" : "localhost:27017"
}
```

# General Advice

- Don't be afraid of denormalization
  - Dedicated collection, embedded document, both?
  - Make frequently needed data more accessible
- Store computed data/fields for querying
  - Count and length fields can be indexed and sorted
  - Easily updated with `$set` and `$inc`

# General Advice

- Simple references (ObjectId only) over DBRefs
  - Concise storage if referenced collection is constant
- Use range queries over `skip()` for pagination
  - `skip()` walks through documents or index values
  - Range queries are limited to next/prev links
  - <http://stackoverflow.com/a/5052898/162228>

# General Advice

- B-trees do not track counts for nodes/branches
  - Filtered counts require walking the index (at best)
  - Non-filtered collection counts are constant time
- Use `snapshot()` for find-and-update loops
  - Ensures documents are only returned once
  - Avoids duplicate processing of updated documents
  - No guarantee for inserted/deleted documents

# DBRefs, Discriminators and mongo

```
> db.users.insert({
...   name: "bob",
...   address: {
...     $ref: "addresses",
...     $id: new ObjectId("4fcea14854298292394bd20a"),
...     $db: "test",
...     type: "shipping"
...   })

> db.users.findOne({name: "bob"}, {_id: 0, address: 1})
{ "address" : DBRef("addresses", ObjectId("4fcea14854298292394bd20a")) }

> db.users.findOne({name: "bob"}, {_id:0, "address.$db": 1})
{ "address" : { "$db" : "test" } }

> db.users.findOne({name: "bob"}, {_id:0, "address.type": 1})
{ "address" : { "type" : "shipping" } }
```

# DBRefs, Discriminators and mongo

```
> db.users.findOne({name: "bob"}, {_id: 0, address: 1})
{ "address" : DBRef("addresses", ObjectId("4fcea14854298292394bd20a")) }

> db.users.findOne({name: "bob"}, {_id:0, "address.$db": 1})
{ "address" : { "$db" : "test" } }

> db.users.findOne({name: "bob"}, {_id:0, "address.type": 1})
{ "address" : { "type" : "shipping" } }
```

Although **\$db** is a valid, optional field for DBRefs, the **mongo** shell hides it by default; likewise for ODM discriminators.

Be mindful of this if you ever need to write data migrations!



# Refactoring OrnicarMessageBundle

- User-to-user messaging (2+ participants)
- Message and thread documents
- Embedded metadata fields (**hash** type in ODM)
  - message.isReadByParticipant
  - thread.datesOfLastMessageWrittenByOtherParticipant
  - thread.datesOfLastMessageWrittenByParticipant
  - thread.isDeletedByParticipant

# Refactoring OrnicarMessageBundle

```
function getNbUnreadMessageByParticipant($participant)
{
    $fieldName = 'isReadByParticipant.' . $participant->getId();

    return $this->repository->createQueryBuilder()
        ->field($fieldName)->>equals(false)
        ->getQuery()
        ->count();
}
```

Counting the number of unread messages for a user entails scanning the entire collection from disk.

# Refactoring OrnicarMessageBundle

```
> db.messages.findOne({}, {isReadByParticipant: 1})
{
  "_id" : ObjectId("4fce28482516ed983884b158"),
  "isReadByParticipant" : {
    "4fce05e42516ed9838756f17" : false,
    "4fce05e42516ed9838756f18" : true,
    "4fce05e42516ed9838756f19" : true,
    "4fce05e42516ed9838756f1a" : false,
    "4fce05e42516ed9838756f1b" : false
  }
}
```

Index isReadByParticipant? Entire object is indexed.

Index isReadByParticipant keys? We'd need 5+ indexes.

# Refactoring OrnicarMessageBundle

```
> db.messages.findOne({}, {unreadForParticipants: 1})
{
  "_id" : ObjectId("4fce28482516ed983884b158"),
  "unreadForParticipants" : [
    "4fce05e42516ed9838756f17",
    "4fce05e42516ed9838756f1a",
    "4fce05e42516ed9838756f1b"
  ]
}
```

Index unreadForParticipants? One multi-key index.

# Refactoring OrnicarMessageBundle

```
function getNbUnreadMessageByParticipant($participant)
{
    return $this->repository->createQueryBuilder()
        ->field('unreadForParticipants')->equals($participant->getId())
        ->getQuery()
        ->count();
}
```

Counting the number of unread messages for a user is now a single indexed query.

# Map/Reduce

```
> db.articles.save({author: "bob", tags: ["business", "sports", "tech"]})  
> db.articles.save({author: "jen", tags: ["politics", "tech"]})  
> db.articles.save({author: "sue", tags: ["business"]})  
> db.articles.save({author: "tom", tags: ["sports"]})
```

Generate a report with the set of authors that have written an article for each tag.

# Map/Reduce

```
> db.articles.mapReduce(
...   function() {
...     for (var i = 0; i < this.tags.length; i++) {
...       emit(this.tags[i], { authors: [this.author] });
...     }
...   },
...   function(key, values) {
...     var result = { authors: [] };
...     values.forEach(function(value) {
...       value.authors.forEach(function(author) {
...         if (-1 == result.authors.indexOf(author)) {
...           result.authors.push(author);
...         }
...       });
...     });
...     return result;
...   },
...   { out: { inline: 1 } }
... )
```

# Map/Reduce

```
{
  "results" : [
    { "_id" : "business", "value" : { "authors" : ["bob", "sue"] } },
    { "_id" : "politics", "value" : { "authors" : ["jen"] } },
    { "_id" : "sports", "value" : { "authors" : ["bob", "tom"] } },
    { "_id" : "tech", "value" : { "authors" : ["bob", "jen"] } }
  ],
  "timeMillis" : 0,
  "counts" : {
    "input" : 4,
    "emit" : 7,
    "reduce" : 3,
    "output" : 4
  },
  "ok" : 1,
}
```

Is there an easier way?



# Aggregation Framework

- Pipeline
  - Operators process a stream of documents
  - Transformations are applied in sequence
  - Expressions calculate values from documents
  - Defined in JSON (no JavaScript code)
- Invoked on collections
  - Use the **\$match** operator for early filtering
  - Compatible with sharding

# Aggregation Framework

- Operations
  - Projection (altering)
  - Match (filtering)
  - Limit
  - Skip
  - Unwind (array values)
  - Group
  - Sort
- Expressions
  - Boolean
  - Comparison
  - Arithmetic
  - String manipulation
  - Date handling
  - Accumulators
  - Conditionals

# Aggregation Framework

```
> db.articles.aggregate(
...   { $project: { author: 1, tags: 1 } },
...   { $unwind: "$tags" },
...   { $group: {
...     _id: { tags : 1 },
...     authors: { $addToSet : "$author" }
...   }}
... )

{
  "result" : [
    { "_id" : { "tags" : "politics" }, "authors" : ["jen"] },
    { "_id" : { "tags" : "tech" }, "authors" : ["jen", "bob"] },
    { "_id" : { "tags" : "sports" }, "authors" : ["tom", "bob"] },
    { "_id" : { "tags" : "business" }, "authors" : ["sue", "bob"] }
  ],
  "ok" : 1
}
```

# Benchmarking Doctrine ODM

- Benchmark bulk document creation
  - Persist and flush
  - Query builder
  - Collection (Doctrine class)
    - Wraps driver class with event dispatching
  - MongoClient (driver class)
- Track insertion time and memory usage

# Benchmarking Doctrine ODM

```
$ ./benchmark-odm-flush.php 100000
Flushing 100000 documents took 47.423843 seconds and used 576978944 bytes
150732800 bytes still allocated after benchmarking

$ ./benchmark-odm-query.php 100000
Inserting 100000 documents took 15.918296 seconds and used 3670016 bytes
8126464 bytes still allocated after benchmarking

$ ./benchmark-odm-driver.php 100000
Inserting 100000 documents took 4.305500 seconds and used 524288 bytes
6029312 bytes still allocated after benchmarking

$ ./benchmark-driver.php 100000
Inserting 100000 documents took 1.120347 seconds and used 524288 bytes
6029312 bytes still allocated after benchmarking
```

Source: <https://gist.github.com/2725976>

# Mongo Monitoring Service (MMS)

- SaaS solution for monitoring MongoDB clusters
- Speeds up diagnosis for support requests
- MMS agent (lightweight Python script)
  - Reports all sorts of database stats
  - Additional hardware reporting with Munin
- Free!

# Mongo Monitoring Service (MMS)

10gen MMS Demo



















Dashboard Hosts Events Alerts Settings Admins FAQ Sign Out

## Hosts +

Hosts all hosts Mongos Configs Agents Agent Log Pings Host Aliases

America/Los\_Angeles

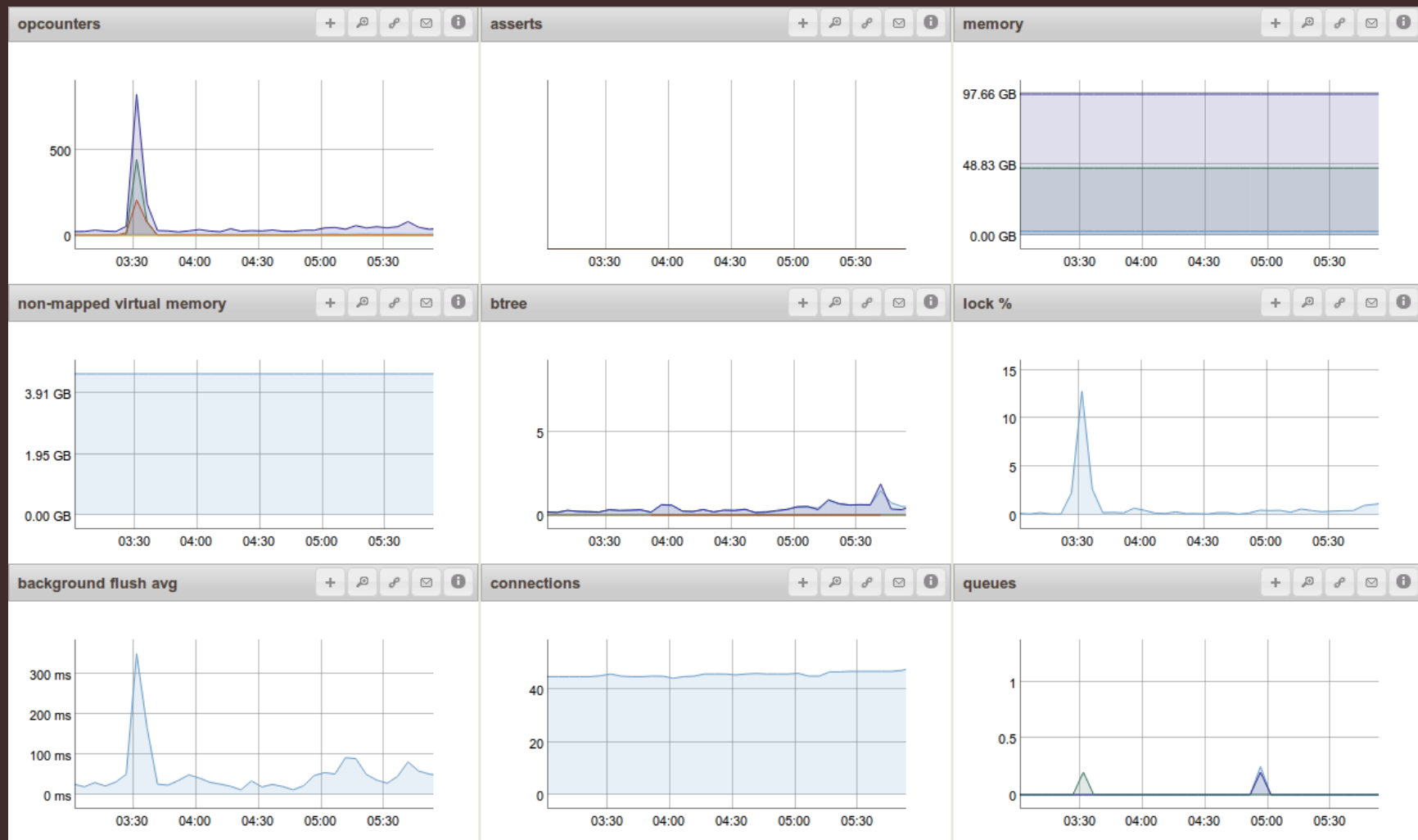
Search:

Name	Type	Cluster	Shard	Repl Set	Last Ping	Up Since	Version	
standalone:27017	standalone				06-05-12 - 10:10	03-28-12 - 15:29	2.0.2	  
shard1a:27218	primary	Cluster 0	shard0000		06-05-12 - 10:10	03-28-12 - 15:29	2.0.2	  
shard1b:27219	primary	Cluster 0	shard0001		06-05-12 - 10:10	03-28-12 - 15:29	2.0.2	  
demo-repl1b:27118	primary	demo-repl		demo-repl	06-05-12 - 10:10	03-28-12 - 15:29	2.0.2	  
demo-repl1a:27117	secondary	demo-repl		demo-repl	06-05-12 - 10:10	03-28-12 - 15:29	2.0.2	  
demo-repl1c:27119	secondary	demo-repl		demo-repl	06-05-12 - 10:10	03-28-12 - 15:29	2.0.2	  

Showing 1 to 6 of 6 entries

First Previous 1 Next Last

# Mongo Monitoring Service (MMS)





# Mongo Monitoring Service (MMS)

