



MongoDB Workshop

Bryan Nehl – Copyright 2013



MongoDB for Java Developers



■ \$: whoami

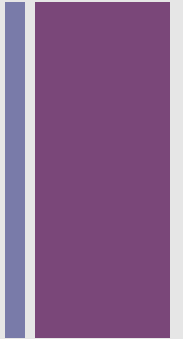
■ Bryan Nehl

■ Systems Developer

■ @k0emt

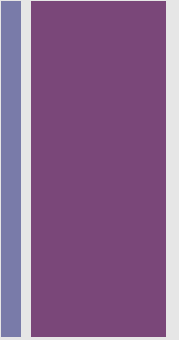
■ dbBear.com

+ You



- Familiar with JSON
- Able to work at the command prompt / terminal
- Developer
 - Able to create Java code
 - Able to add a jar/library
 - Able to compile and run code
- Curious, Engaged, Respectful

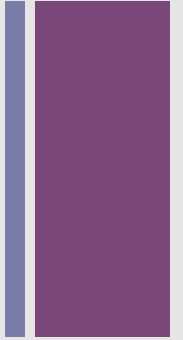
+ You



- What is your background?
- What do you want to get out of today?



Your computer



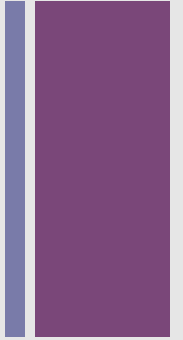
- Windows 7 or 8, Linux or OS X
 - WHY?
- Java Development Kit
 - Please reserve your alternate JVM language experimentation for lab time
- Editor or IDE of your choice

+ Workshop Primary Goal

It is my primary goal that you leave the workshop with a functioning MongoDB environment and knowledge of the fundamentals with the skills to do routine development work.

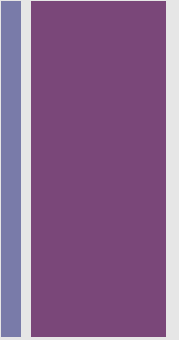


Workshop Topics



- Introduction and Installation
- Schema – databases, collections and documents
- Creating, Reading, Updating and Deleting (CRUD)
- Advanced CRUD – sub documents, arrays, sorting, limiting...
- Backups
- Performance/Indexes
- Aggregation Framework
- GridFS
- Replication
- Sharding Overview
- Open Lab Time

+ Why MongoDB?



- Document Oriented Schema
- Scalable
 - Commodity Hardware
 - Horizontal
- Fast – memory mapped files
- GridFS

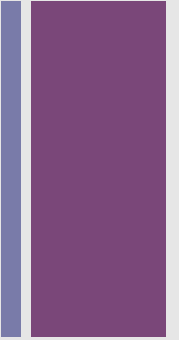


+

`{"section": "Installation" }`

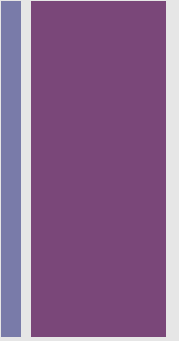
www.mongodb.org/downloads

+ File Setup



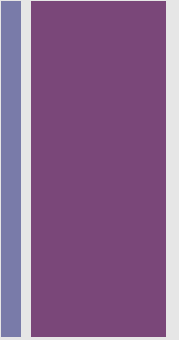
- Copy the flash drive contents to your computer
- Create a project working directory “kcdc”

+ unzip install



- Easy
- Manual updates
- Manual path setup

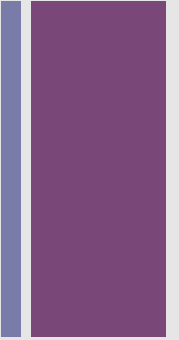
+ Windows



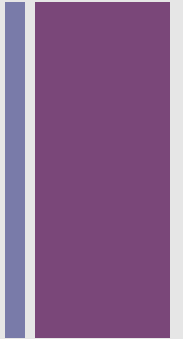
- Which zip to get?
- --smallfiles
 - Initial size reduced
 - Journal files from 1G to 128M
- Unzip
- Set your PATH
- `mkdir -p c:\data\db`

+ Linux

- `--smallfiles`
- Unzip
- Untar
- Set up your path
- `mkdir -p /data/db`
 - Set permissions
- Packages

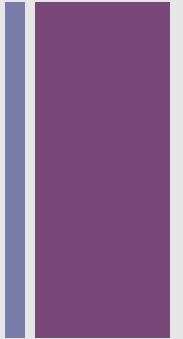


+ OS X



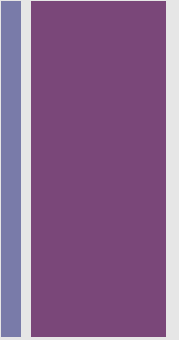
- Take the unzip, path, /data/db approach
- Home Brew
 - Easy updates
 - Don't do it if you use Mac Ports
 - <http://mxcl.github.io/homebrew/>
 - `ruby -e "$ (curl -fsSL https://raw.githubusercontent.com/mxcl/homebrew/go) "`
 - `brew update; brew install mongodb`
- Mac Ports

+ Config File



- `--config C:\mongodb\mongod.cfg`
- `/etc/mongodb.conf`
 - `smallfiles = true`
- Home brew
 - `/usr/local/etc/mongod.conf`
- <http://docs.mongodb.org/manual/reference/configuration-options/>

+ Security



- Basic
- User Privilege Roles
- The Network

+ Verifying Installation

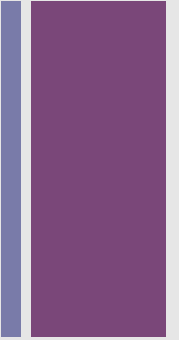
■ mongod

- --version
- Windows: start & <ctrl><c>
- OS X/Linux: --fork, kill, <ctrl><c>

■ mongo

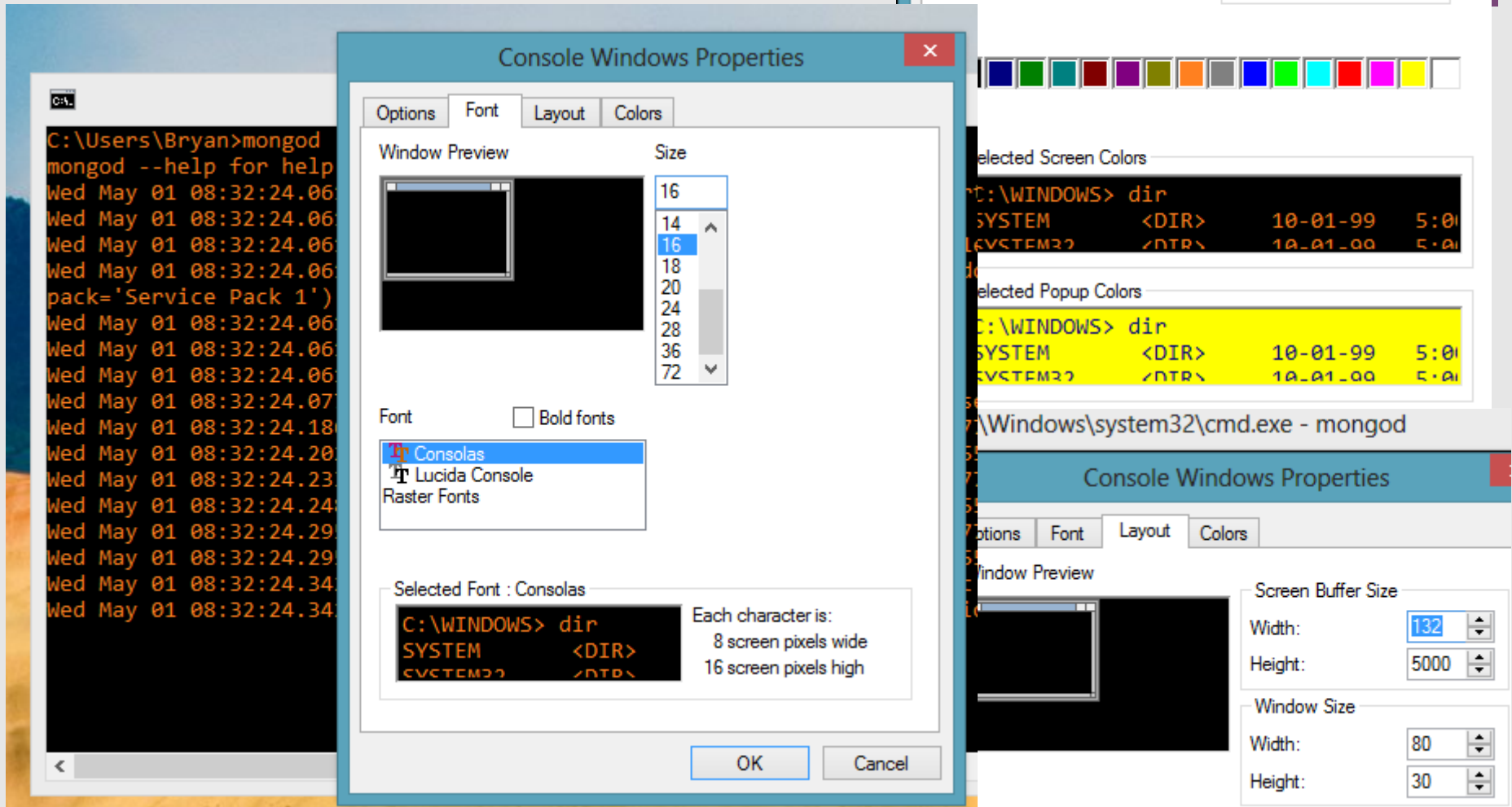
- The prompt
- db.version()
- show dbs
- quit()

+ Java Driver



- Start at MongoDB.org -> drivers -> java
- <http://docs.mongodb.org/ecosystem/drivers/java/>
- Pay careful attention to version numbers in the maven repo
- We are using version 2.11.1
- Add to your Java Project in your IDE / know your classpath

+ Configure your console

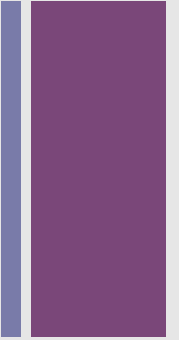




+

{“section” : “Schema”}

+ JSON Review



- json.org

- {"key" : "values"}

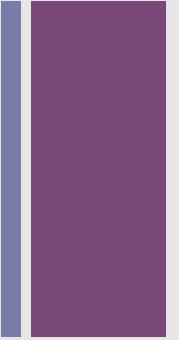
- JSON types

 - string, number, object, array, true, false, null

- Lists

- Sub-documents

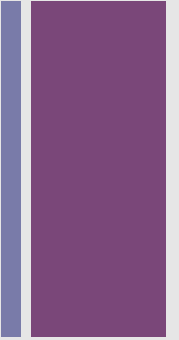
+ Structure



- Databases
- Collections
- Documents
- Fields



Document Oriented Schema Design



■ Naming

- avoid the . (dot)
- key name length matters

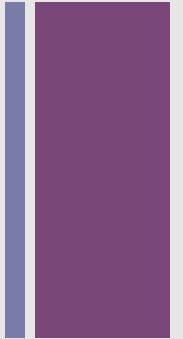
■ No Joins

■ Consider the Access Pattern



Relational Design Exercise

- Using standard relational techniques design an inventory management system that tracks assets.
- Example assets are: vehicles, computers, tables and chairs.
- I want to be able to store a lot of detail.
 - Where is the asset?
 - To whom is an asset assigned?
 - Vehicle detail like: make, model, VIN, color, etc.
 - Table detail like: material type, size, condition, color, etc.



+ Document Design

- How would we organize this same sort of information in a document oriented system like MongoDB?
- *Consider the access pattern(s).*

+ {“section” : “**CRUD**”,
“**alternateText**” : “Create,
Read, Update, Delete” }

+ Start your engines!

- Start up mongod

 - --smallfiles

- Start up mongo

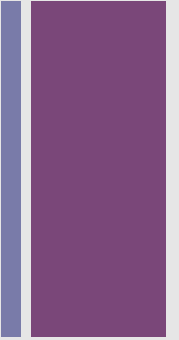
- Open a shell or IDE for java

+ mongo – the shell

■ Javascript

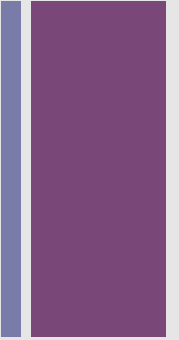
■ JSON

■ Variance from strict JSON





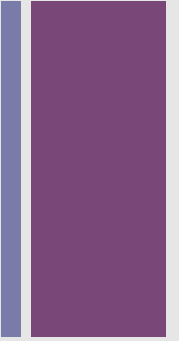
Create -- insert



- use workshop
- `db.stuff.insert({"hello": "world"})`
- `db.stuff.insert({"greeting":
"people", "name": "me"})`
- show collections

+ Read -- findOne

■ **db.stuff.findOne()**



+ `_id`

■ Unique

- Default is `ObjectID`
- A document may be fully duplicated except for the `_id`

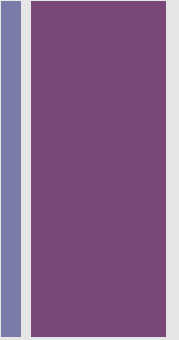
■ Any Type

■ **ObjectId** is

- Globally Unique Identifier (GUID)
- A 12-byte BSON type, constructed using:
 - A 4-byte value representing the seconds since the Unix epoch
 - A 3-byte machine identifier
 - A 2-byte process id
 - A 3-byte counter, starting with a random value

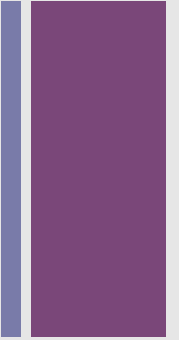


Read -- find



- `db.stuff.find()`
- `db.stuff.find().pretty()`
- `db.stuff.find({"name": "me"})`
- `db.stuff.find({"_id": ObjectId("...")})`

+
it

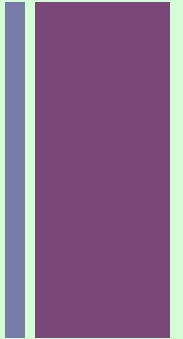


- `find()` returns a cursor

- *it* gets the next set

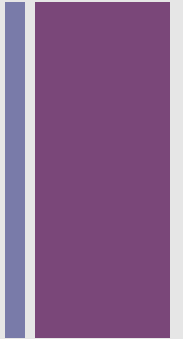
```
for(var i = 0; i < 30; i++ )  
{ db.stuff.insert({"counter":i}) }  
  
db.stuff.find()
```

+ Let's write some Java!



- What do we need to do?
- Connect to the database server
- Use a database
- Work with a collection
 - **inserts**
 - **finds**
- Close the connection

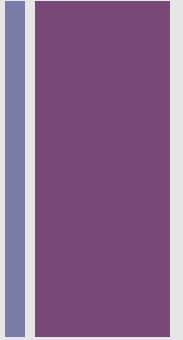
+ Update



- `db.stuff.insert({"name":"joe"})`
- `db.stuff.insert({"name":"jo"})`
- `db.stuff.update({"name":"jo"}, {"city":"COU"})`
- `db.stuff.find({"name":"jo"})` ???
- `db.stuff.find()`
- `db.stuff.insert({"name":"jo"})`
- `db.stuff.update({"name":"jo"}, { $set: {"city":"COU"} })`
- `db.stuff.find({"name":"jo"})`

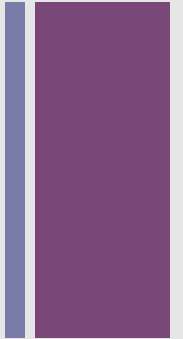


Update multiple documents



- `for(var i = 0; i < 5; i++) { db.stuff.insert({"_id":i, "multiDemo":1}) }`
- `db.stuff.find({"multiDemo":1})`
- `db.stuff.update({"multiDemo":1}, {$set: {"updated":1} })`
- `db.stuff.find({"multiDemo":1})` ???
- `db.stuff.update({"multiDemo":1}, {$set:{"updated":1} },
{multi : true})`
- `db.stuff.find({"multiDemo":1})`
- <http://docs.mongodb.org/manual/core/update/#update-multiple-documents>

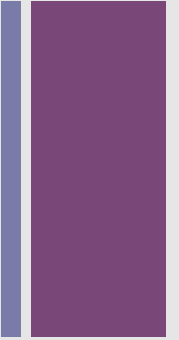
+ Upserting



- `db.stuff.find({"color":"blue"})`
- `db.stuff.update({"color":"blue"},
{$set:{"iDidIt":true}}, {upsert:true})`
- `db.stuff.find({"color":"blue"})`
- `db.stuff.update({"color":"blue"},
{$set:{"primary":true}}, {upsert:true})`
- `db.stuff.find({"color":"blue"})`

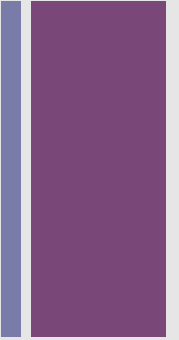


To the Java!



- Match query
- Update document via replacement
- Multi update combined with \$set

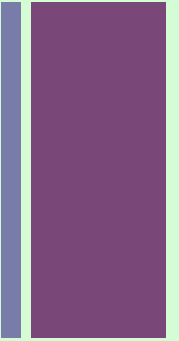
+ Delete – remove()



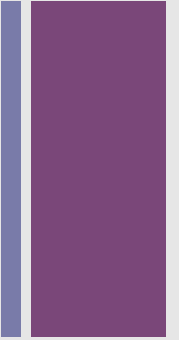
- `db.stuff.find({"counter":1})`
- `db.stuff.remove({"counter":1})`
- `db.stuff.find({"counter":1})`

+ To the Java!

- `remove()`



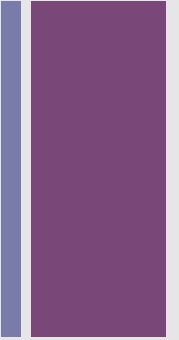
+ Atomic operations



- No Transactions
- Single write operations are atomic
- <http://docs.mongodb.org/manual/tutorial/isolate-sequence-of-operations/>

```
{“section” : “Advanced  
CRUD”, “tags” :  
+ [ “Advanced”, “Create”,  
“Read”, “Update”,  
“Delete” ] }
```

+ Regular expressions



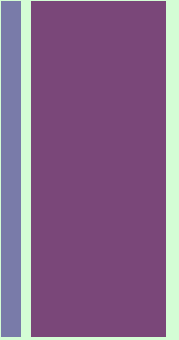
- `db.stuff.find({"name" : /^[m]/})`

- value in name starts with m

- `db.stuff.find({"name" : {$regex:/[o]$/}})`

- value in name ends with o

+ To the Java!



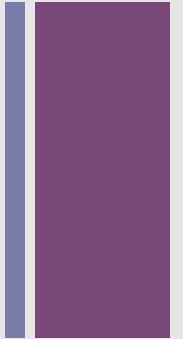
- Regular Expressions
- FindRegularExpressionsDemo.java
- Use of `java.util.regex.Pattern`

+ Arrays – the basics

- `db.junk.insert({"section": "Advanced CRUD",
"tags":
["Advanced", "Create", "Read", "Update", "Delete"]})`
- Keep their order
- `db.junk.find({tags: "Read"})`
- `db.junk.find({tags: "READ"})`



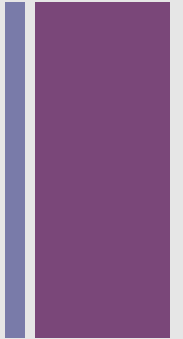
Arrays – \$push



- `db.demo.insert({"demo":"array"})`
 - `{"demo":"array"}` ← there is no magic here. It's just a document.
- `db.demo.update({"demo":"array"},
 { $push: { "movies": "dune" } })`
- `db.demo.findOne({"demo":"array"})`
- `db.demo.update({"demo":"array"},
 { $push: { "movies": "dune" } })`
- `db.demo.findOne({"demo":"array"})`
- <http://docs.mongodb.org/manual/reference/operator/update-array/>



Arrays – \$push and \$pull



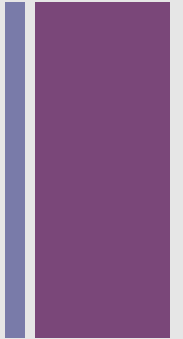
- `db.demo.update({"demo":"array"},
 { $pull: { "movies": "dune" } })`
- `db.demo.findOne({"demo":"array"})`
- `db.demo.update({"demo":"array"},
 { $push: { "movies": { $each: ["dune","tron"] } } })`
- `db.demo.findOne({"demo":"array"})`
- `db.demo.update({"demo":"array"},
 { $pull: { "movies": ["dune"] } })`
- `db.demo.findOne({"demo":"array"})`

+ Arrays – \$addToSet

- `db.demo.update({"demo":"array"},
{$addToSet: {"movies":"dune"}})`
- `db.demo.findOne({"demo":"array"})`
- `db.demo.update({"demo":"array"},
{$addToSet: {"movies":"aliens"}})`
- `db.demo.findOne({"demo":"array"})`
- `db.demo.update({"demo":"array"},
{$addToSet: {"books":"ghost"}})`
- `db.demo.findOne({"demo":"array"})`



\$set and \$unset



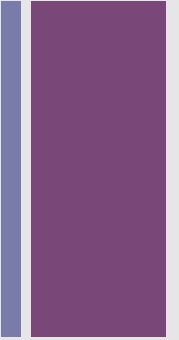
- `db.demo.insert({"demo":"setting"})`
- `db.demo.findOne({"demo":"setting"})`
- `db.demo.update({"demo":"setting"},
{$set:{"newkey":true}})`
- `db.demo.findOne({"demo":"setting"})`
- `db.demo.update({"demo":"setting"},
{$unset:{"newkey":true}})`
- `db.demo.findOne({"demo":"setting"})`



+

{“section” : “Backups”}

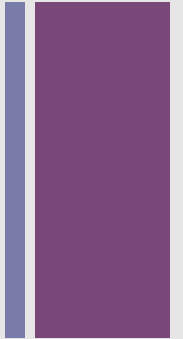
+ Database Size



- `show dbs`
- `db.stats()`
- `db.stuff.stats(1024)`
- `db.runCommand({listDatabases:1})`
- `use admin`
- `db.runCommand({listDatabases:1})`
- `db.runCommand({listCommands:1})`



MongoImport



- **mongoimport**

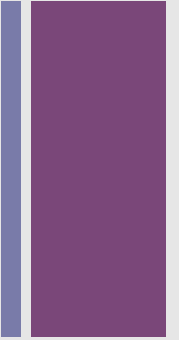
- Unicode / UTF-8 and CP1252
- Types of imports – external file types
- --drop
- --upsert

- **mongoimport --db workshop --collection names < names.json**

- **db.names.count()**

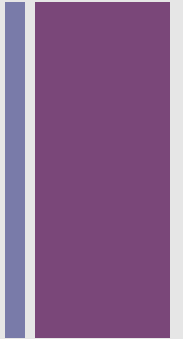
- **db.names.findOne()**

+MongoExport



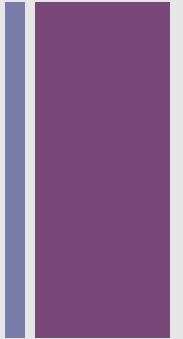
- **mongoexport**
 - Types supported – csv, json, etc.
- **mongoexport --db workshop --collection names > new_names.json**
- **Compare the names.json and new_names.json files**

+ MongoDB



- File System Snapshots are the recommended backup approach.
- `mongodump` and `mongorestore` create and restore
 - Can be run directly against the data files – no mongod running
- `mongodump --help`
 - Can use cross machine
 - Has default no argument behavior
- `mongodump --db workshop --out workshop_dump`
 - Check the directory

+MongoRestore



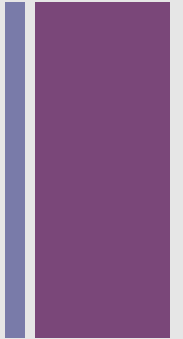
- `mongorestore --help`
- `mongorestore --db new_workshop workshop_dump/workshop`
 - Note the `--drop` option
- In the mongo shell
 - `show dbs`
 - Examine `new_workshop...`

+

```
{“section” : “Advanced CRUD”,  
  “part” : “deux”,  
  “tags” : [ “Advanced”, “Create”,  
             “Read”, “Update”, “Delete” ] }
```



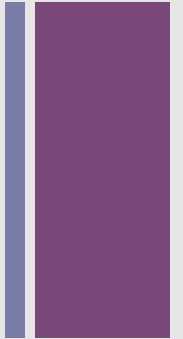

find – limiting the returned fields



- use workshop
- `db.names.findOne()`
- `db.names.find({}, {city: 1, name: 1, district: 1})`
- `db.names.find({}, {_id:0, scores:0})`
- `db.names.find({}, { city:1, scores:0})`



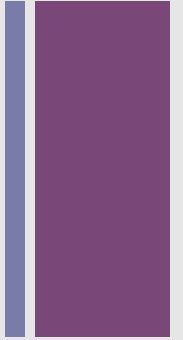
sort, skip and limit



- `db.names.find({}, {scores:0}).limit(5).pretty()`
- Skip and Limit are always applied after Sort
- `db.names.find({}, {scores:0}).limit(5).sort({city:1}).pretty()`
- `db.names.find({}, {scores:0}).limit(5).sort({city:1, name:1}).pretty()`
- `db.names.find({}, {scores:0}).limit(5).sort({city:1, name:1}).skip(2).pretty()`



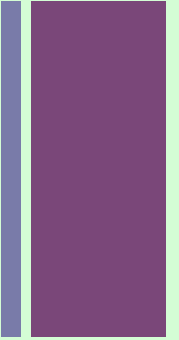
\$and or comma? And \$or



- `db.names.findOne({"age":87,"scores":87})`
- What if I want to find someone that has a score of 0 and a score of 100?
- `db.names.findOne({"scores":0,"scores":100})`
- `db.names.findOne({$and: [{"scores": 100}, {"scores": 0}] })`
- `db.names.findOne({$or: [{"scores": 100}, {"scores": 0}] })`



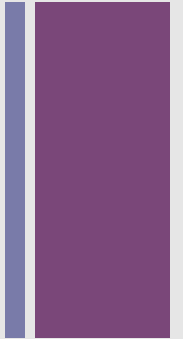
To the Java!



- Find with BasicDBObjectBuilder
- limit()
- FinderBuildAndLimitsDemo.java



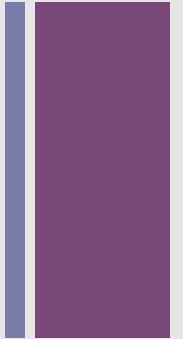
Subdocuments



- `db.demo.insert({"demo":"subdocs",
subdoc: {"sub": "one"} })`
- `db.demo.findOne({"demo":"subdocs"})`
- `db.demo.update({"demo":"subdocs"},
{$set:{"subdoc.ver":1}})`
- `db.demo.findOne({"demo":"subdocs"})`
- `db.demo.findOne({"subdoc.ver":1})`



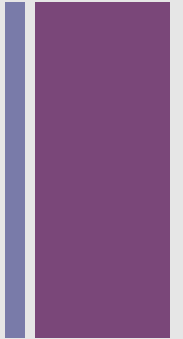
Count, \$gt, \$lte, \$not, and \$ne



- `db.names.find({ age: { $lte: 21 } }).count()`
- `db.names.find({ age: { $not : { $gt: 21 } } }).count()`
- `db.names.find({ madeup: { $gt: 21 } }).count()`
- `db.names.find({ madeup: { $not : { $lte: 21 } } }).count()`
- `db.names.find({ age: { $ne: 21 } }).count()`
- `db.names.find({ color: { $ne: "purple" } }, {scores:0}).count()`



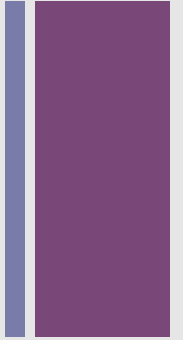
\$in and \$exists



- `db.names.find({name:"STOUT"}).count()`
- `db.names.find({name:"BROCK"}).count()`
- `db.names.find({name: { $in:["STOUT","BROCK"] } }).count()`
- `db.demo.find({movies: { $exists: true } })`
- <http://docs.mongodb.org/manual/reference/operator/>



getLastError



- `db.runCommand({getLastError:1})`
- The `getLastError` command returns the error status of the last operation on the current connection.

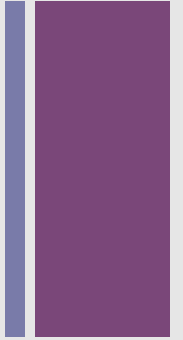


+

{“section” : “**Dropping stuff**”}



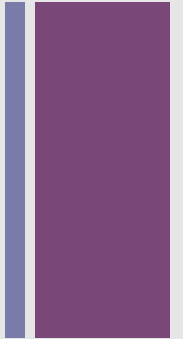
Dropping – Document Review



- use habit
- `db.badhabit.insert({"sneeze": "spray"})`
 - show dbs
 - show collections
- `db.badhabit.remove({"sneeze": "spray"})`
 - show dbs
 - show collections



Dropping – Collections and DB



- `db.badhabit.drop()`

- `show collections`

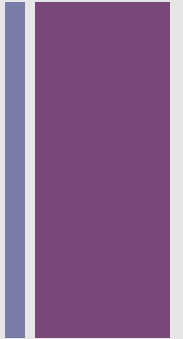
- `db`

- `db.dropDatabase()`

- `show dbs`

- `db` `???`

+ Dropping – DB



- use new_workshop

- db.dropDatabase()

 - show dbs

 - db ???

- use workshop

+ Tab completion!

- `db.<TAB>`

- `db.collection.<TAB>`

- What's the method do?

 - `db.collection.insert`

 - Notice NO ()'s after the insert

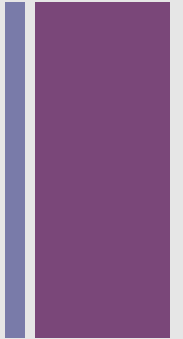


+

{“section” : “Performance”}



explain()

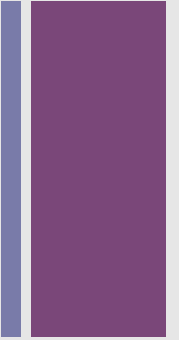


- `db.names.find({"age":55}).explain()`
- `scanAndOrder` is a boolean that is true when the query *cannot* use the index for returning sorted results.
- <http://docs.mongodb.org/manual/reference/explain/>

+ Indexes

- Enhance query performance
 - Introduces insert overhead
 - Use a B-tree data structure
- Only one index is used per operation
 - find, sort, update, etc.
- When can an index be used?
 - Index is a,b,c and query is a or -a or a,b or a,b,-c (yes)
 - Index is a and query is a,b,c (no)
 - The query has to use some **left** subset of the index
- <http://docs.mongodb.org/manual/core/indexes/>

+ Covers



- An index “covers” a query if:
 - All the fields in the query are part of the index **and**
 - All the fields returned in the documents that match the query are in the same index

+ The _id index

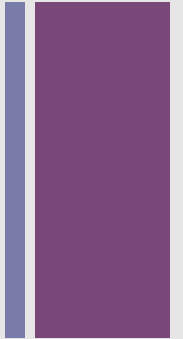
- `db.names.find({"_id":65}).explain()`

+ Indexes & getIndexes

- `db.junk.insert({"boo":"bunny","days":42})`
- `db.junk.insert({"boo":"bear","days":33})`
- `db.junk.ensureIndex({"boo":1,"days":1})`
- `db.junk.getIndexes()`
- `db.junk.find().sort({boo:1,days:-1}).explain()`
- `db.junk.find().sort({boo:-1,days:-1}).explain()`
- `db.junk.ensureIndex({"days":1})`
- `db.junk.getIndexes()`
- `db.junk.find().sort({days:-1}).explain()`
- `db.junk.find({}, {_id:0,days:1}).sort({days:-1}).explain()`



Unique indexes



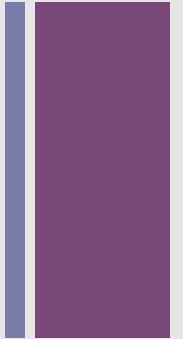
- `db.junk.ensureIndex({"boo":1},{unique:true})`
- `db.junk.insert({"boo":"bear"})`
- **Multiple unique indexes?**
- `db.junk.ensureIndex({"days":1, "boo":1},{“unique”:true})`
- <http://docs.mongodb.org/manual/core/indexes/>

+ Revise existing index

- `db.junk.ensureIndex({"days":1},{unique:true})`
- `db.junk.insert({"days":42})`
- What happened?
- `db.junk.getIndexes()`
 - To revise an index, you must drop it and create it with the new specification.



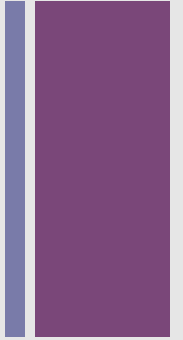
dropIndex / dropIndexes



- `db.collection.dropIndex({full:1, index:1, specification:1})`
- `db.junk.dropIndex("boo":1})`
- `db.junk.getIndexes()`
- `db.junk.dropIndexes()`
- `db.junk.getIndexes()`



Indexes & getIndexes (names)

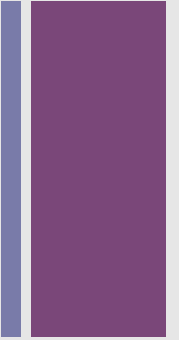


- `db.names.ensureIndex({"name":1})`
- `db.names.find({"name":"HUMPHREY"}).explain()`
 - How many HUMPHREYs are there?
 - Check `nscannedObjects` and `nscanned`
 - `Millis` probably 0 now too
- `db.names.find({meh:"meh"}).sort({name: 1}).explain()`

+ Multikey indexes

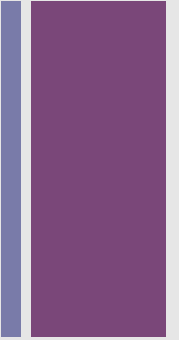
- Only one Array type field per index
- Order is important
- `db.names.ensureIndex({"city":1,"name":1})`
- `db.names.find({"city":"Munich","name":"HUMPHREY"}).explain()`
- `db.names.find({"name":"HUMPHREY","city":"Munich"}).explain()`
- `db.names.find({"name":"HUMPHREY"}).sort({"city":1,"name":1}).explain()`
- `db.names.find({"meh":"HUMPHREY"}).sort({"city":1,"name":1}).explain()`
- `db.names.find({}).sort({"city":-1,"name":1}).explain()`
- <http://docs.mongodb.org/manual/core/indexes/#multikey-indexes>

+ Mongostat



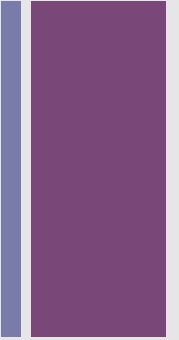
- mongostat
- What's going on with mongod / mongos?
- <http://docs.mongodb.org/manual/reference/mongostat/>

+ mongotop



- mongotop
- Where are the reads and writes happening?
- <http://docs.mongodb.org/manual/reference/mongotop/>

+ `.stats()`

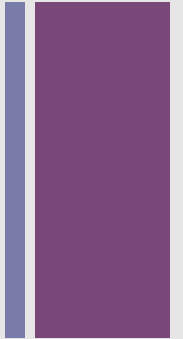


- `db.stats()`

- `db.names.stats()`

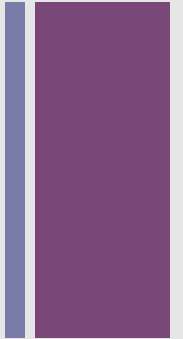


system.profile



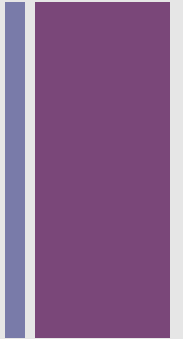
- `db.getProfilingStatus()`
- `db.setProfilingLevel(0)`
- `db.getProfilingStatus()`
- `db.runCommand({ profile: 1, slowms: 200 })`
- `db.getProfilingStatus()`
- `db.names.find().sort({"age":1})`
- `db.system.profile.find().pretty()`
- `db.setProfilingLevel(0)`

+ Logs



- **mongo.log**
 - /usr/local/var/log/mongodb
- **Windows**
 - By default no log, check the console
- **Slow queries are in the log**

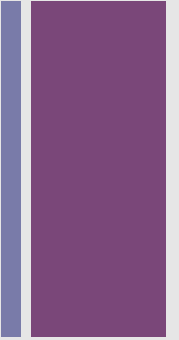
+ Mongo Monitoring Service (MMS)



- What is it?
- `mms.10gen.com`
- `http://www.10gen.com/products/mongodb-monitoring-service`

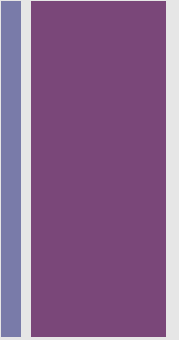


Further research



- Sparse indexes
- Query hint() the index to use
- mongod
 - --profile level
 - --slowms

+ Experiment



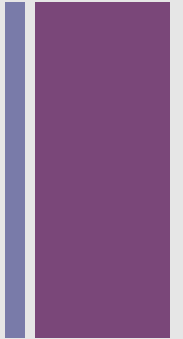
- Take a couple minutes to:
- Gather your thoughts and questions you want to research
- Experiment with the things you just learned.



+

{“section” : “**Aggregation**”}

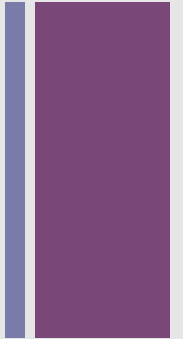
+ Stages



- Pipeline style architecture
- \$project, \$match, \$limit, \$skip, \$unwind, \$group, \$sort
- `db.collection.aggregate({pipeline operations}, {}...)`
- `db.collection.aggregate([{pipeline operations}, {}...])`
- <http://docs.mongodb.org/manual/core/aggregation/>
 - Also contains information on optimization
- <http://docs.mongodb.org/manual/reference/aggregation/>



\$limit, \$match and \$project

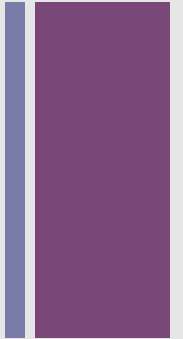


- `db.names.aggregate({$limit : 3})`
- `db.names.aggregate({$match: {"city": "Columbia"}}, {$limit:3})`
- `db.names.aggregate({$match:{ "city": "Columbia"}}, {$limit:3}, {$project:{city:1,district:1 }})`
- `db.names.aggregate({$match:{ "city": "Columbia"}}, {$limit:3}, {$project:{city:1, schoolDistrict : "$district" }})`
- `db.names.aggregate({$match:{ "city": "Columbia"}}, {$limit:3}, {$project:{city:1, district:1, higherDistrict: {$gt: ["$district", "M"] } }})`

+ \$skip

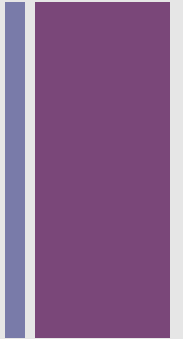
- `db.names.aggregate({$match:{"city":"Columbia"}},
{$limit:3})`
- `db.names.aggregate({$match:{"city":"Columbia"}},
{$limit:3}, {$skip:2},
{$project:{city:1, district:1 }})`
- `db.names.aggregate({$match:{"city":"Columbia"}},
{$skip:2}, {$limit:3},
{$project:{city:1, district:1 }})`

+ \$group, \$sum and \$sort



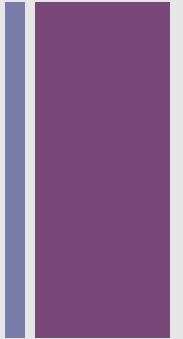
- `db.names.aggregate({$group:{_id: "$city"}})`
- `db.names.aggregate({$group: {_id: "$city", population: {$sum: 1}} })`
- `db.names.aggregate({$group: {_id: null, population: {$sum: 1}} })`
- `db.names.aggregate({$group: {_id: {}}, population: {$sum: 1}} })`
- `db.names.aggregate({$group: {_id: "$city", population: {$sum: 1}}}, {$sort: {"_id": 1}})`
- `db.names.aggregate({$group: {_id: "$city", population: {$sum: 1}}}, {$match: {"_id": "Zurich"}})`

+ \$unwind



- `db.names.aggregate({$match:{"city":"Columbia"}},
{$limit:1}, {$unwind: "$scores"})`
- `db.names.find({_id:3})`
- `db.names.aggregate({$match:{"_id":3}},
{$unwind: "$scores"},
{$group: {_id:"$_id", totalPoints:{$sum:"$scores"}} })`
- `db.names.aggregate({$match:{"_id":3}},
{$unwind: "$scores"},
{$group: { _id: "$_id", max: {$max: "$scores"}} })`

+ Aggregation Exercise



- What are the top 5 occurring names?
- What is the `_id` of the person with the *highest* overall score *average* in Columbia's "Z" district?
 - Person with "`_id`" : 58737 has the *lowest* score
- HINTS:
 - Build the query in stages!
 - Start with `$limit` and `$match`
 - `$avg`
 - Verify expected results at each stage

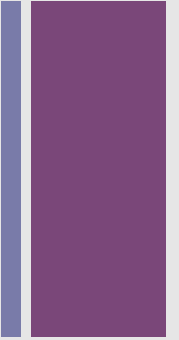
+ Aggregation Answers

- `db.names.aggregate(
 {$group: {_id: "$name", frequency: {$sum: 1} } },
 {$sort: {"frequency": -1}},
 {$limit: 5})`
- `db.names.aggregate(
 {$match: {city: "Columbia", "district": "Z" }},
 {$unwind: "$scores"},
 {$group: {_id: "$_id", average: {$avg: "$scores"}}},
 {$sort: {"average": -1}},
 {$limit: 5})`

+ {"section": "GridFS",
 "alternate": "where did I put
 that file?"}

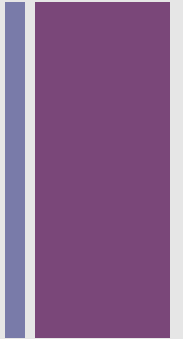


Introduction



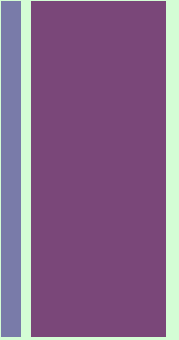
- Store and Retrieve *files*
- Uses two collections
- Meta data and pointer stored in `.files`
- Chunks of binary data stored in `.chunks`

+ mongofiles



- `$: mongofiles --db workshop put census_surnames.xls`
- `show collections`
- `db.fs.files.findOne()`
- `db.fs.chunks.findOne()`
- `mongofiles --db workshop get census_surnames.xls`
`--local surnames.xls`
- <http://docs.mongodb.org/manual/reference/mongofiles/>

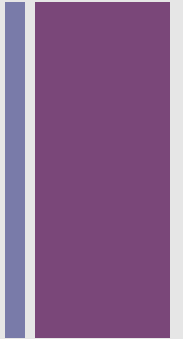
+ GridFS with Java



- Puts in a file with metadata
- Retrieve a file by metadata search
- Save the retrieved file
- Examine the collections in the shell

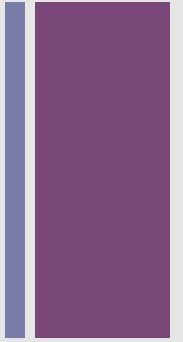
+ {"section": "Replication",
"alternate": "Department of
Redundancy Department"}

+ Replication



- Ensures redundancy
- Backup
- Automatic failover
- Replication is implemented with groups of servers known as replica sets.
- <http://docs.mongodb.org/manual/replication/>

+ Node Types and Attributes



- Primary

- Secondary

 - Hidden

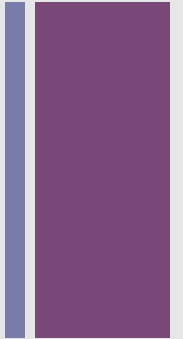
 - Delayed

 - Arbiter

 - Non-voting

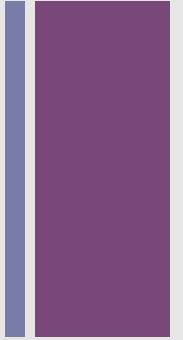
- <http://docs.mongodb.org/manual/core/replication/>

+ Fail Over



- The Primary Server goes down, now what?
- Voting Secondary's elect new Primary
- Elected Secondary Promotes to Primary
- Old primary comes back on line
- It rejoins as secondary
 - Unless it has a higher priority attribute set
- It syncs back up
- Rollback?

+ Configuration



■ Development Environment

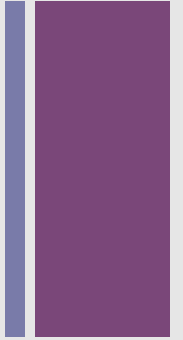
- Multiple mongod on a single machine
- *Different* port number per mongod
- What does replication on a single machine accomplish?

■ Production Environment

- One mongod per server
- Default port number
- Odd number of mongod
- Virtual Server versus Physical servers



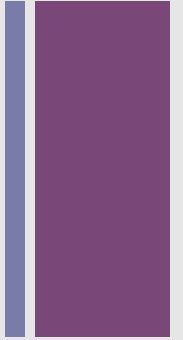
Hands on, Spin up a Replica Set



- From your flash drive backup files, copy **repset** to your **kcdc** working directory
- Start up the nodes
 - `mongod --port 27001 --dbpath n1 --logpath n1/node.log --config repset.conf`
 - `mongod --port 27002 --dbpath n2 --logpath n2/node.log --config repset.conf`
 - `mongod --port 27003 --dbpath n3 --logpath n3/node.log --config repset.conf`
- Windows
 - Use `repset_windows.conf` and start at the beginning of the line
 - **start** `mongod --port 27001 --dbpath n1 --logpath n1/node.log --config repset_windows.conf`



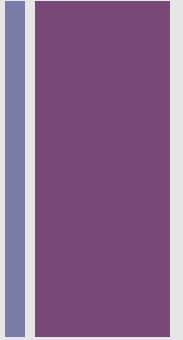
Hands on, Spin up a Replica Set



- Initiate the replication set
 - `mongo --port 27001 --shell config.json`
 - `rs.initiate(cfg)`
 - `rs.status()`
 - Notice prompt SECONDARY or PRIMARY?
- `show dbs`

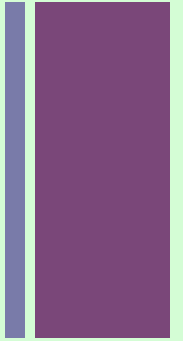


Hands on, Spin up a Replica Set



- Insert some data on PRIMARY
 - `db.grass.insert({keepOff:true})`
- Connect to a SECONDARY
 - `mongo -port 27002`
- Query for the data – show collections
 - Secondary reads not OK, why?
- Reconfigure to allow SECONDARY read
 - `rs.slaveOk()`
 - `db.grass.findOne()`
 - `db.grass.insert({mow:true})`

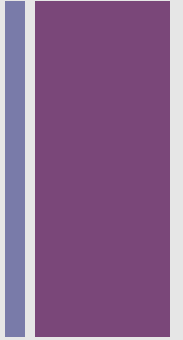
+ To the Java!



- Connecting to a replica set
- Handling of Primary node failure
- Write concern
- Safe Writes – `RepSetWriting.java`
- Safe Reads – `RepSetReading.java`



Shutdown your replica set



- Poke around in the database directories `nl`
 - Check out the log file
 - Check out the lock file
- Close running consoles
- Find and kill processes

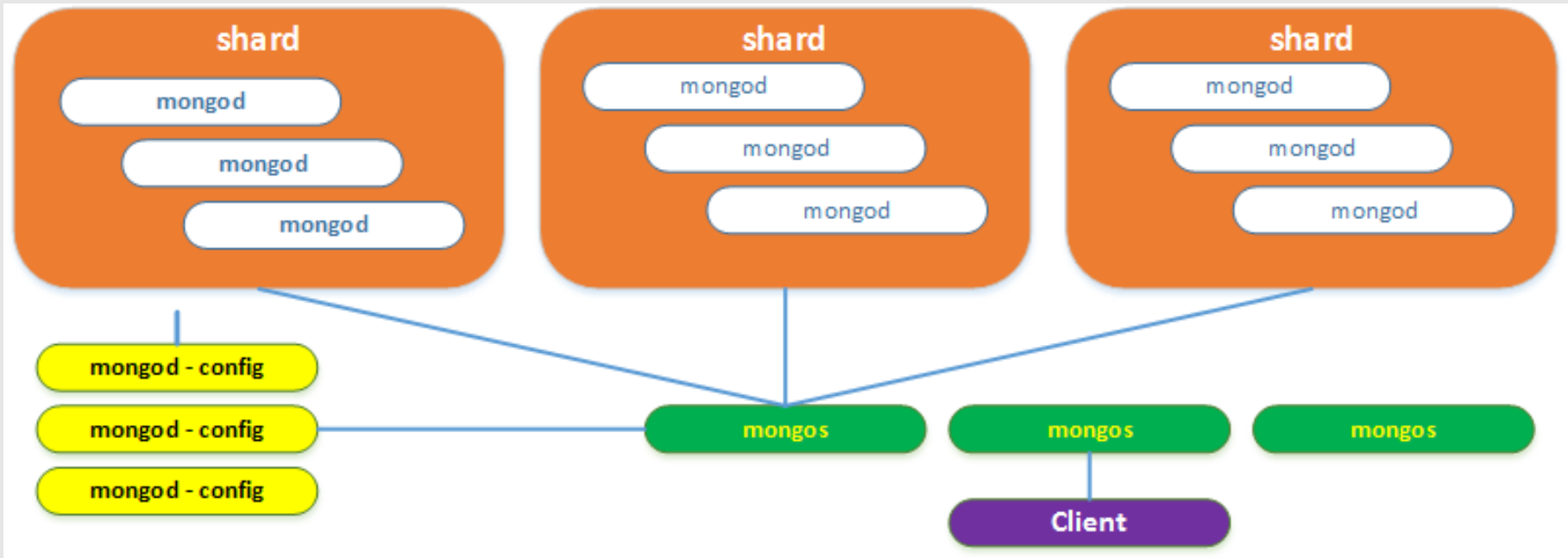
+ {“section” : “**Sharding**”,
“alternate” : “distributed read-
write scaling” }

+ Sharding

- Sharding distributes a single logical database system across a cluster of machines
- Shards
 - Store a portion of the collection – size scalability
 - Balance read/write load and data across machines
 - Enabled per database and collection
- mongos
 - used to access the shards
 - Utilize config servers which have metadata
 - About the cluster
 - About where the chunks are for the shards
- <http://docs.mongodb.org/manual/sharding/>



Production Sharding Environment



+ {“section” : “Open Lab Time”,
“alternate” : [“experiment”,
“ask questions”]}

+ Where do I go from here?

■ Check out the MongoDB documentation

- Capped Collections
- Capped Arrays (New in 2.4)
- Geospatial / GIS, GeoJSON support (New in 2.4)
- Role-based privileges (New in 2.4)
- Full Text Search (Beta in 2.4)
- Learn more about sharding
- Query Operators
- Map-Reduce

■ Experiment

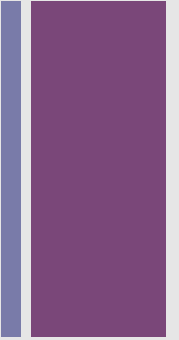
- Share your experience – blog, tweet, present
- GitHub and Gists

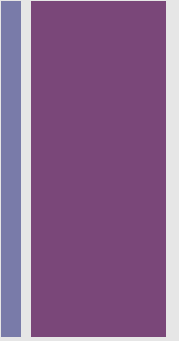
+ Other languages

- Groovy

- Python

 - generate_names.py





Question & Answer

+ Practice – open lab time

■ Experimenting ideas

- Personal Journal
- Personnel System
- Asset Management System
- Advanced Queries & Aggregation Framework

