1. Slide Gallery . . . 8
2. .bookmarks . . . 8
3. 1.1 Development Cycle . . . 8
4. Creating and Deleting Indexes . . . 8
5. C Sharp Language Center . . . 8
6. Diagnostic Tools . . . 8
7. Django and MongoDB . . . 9
8. Getting Started . . . 9
9. International Documentation . . . 9
10. Monitoring . . . 9
11. Older Downloads . . . 9
12. PyMongo and mod_wsgi . . . 9
13. Python Tutorial . . . 10
14. Recommended Production Architectures . . . 10
15. v0.8 Details . . . 10
16. Building SpiderMonkey . . . 10
17. Documentation . . . 11
18. Dot Notation . . . 11
19. Dot Notation . . .
20. Getting the Software . . . 11
21. Language Support . . . 11
22. Mongo Administration Guide . . . 11
23. Working with Mongo Objects and Classes in Ruby . . . 12
24. MongoDB Language Support . . . 12
25. Community Info . . . 12
26. Internals . . . 12
27. TreeNavigation . . . 13
28. Old Pages . . . 13
    28.1 MongoDB - A Developer's Tour . . . 13
    28.2 Mongo Developers' Guide . . . 13
    28.3 HowTo . . . 13
    28.4 Database Profiler . . . 13
    28.5 Updating Data in Mongo . . . 13
    28.6 Queries and Cursors . . . 14
    28.7 Indexes . . . 14
    28.8 Object IDs . . . 14
    28.9 OR operations in query expressions . . . 14
    28.10 Cursors . . . 14
    28.11 Data Types and Conventions . . . 14
    28.12 Databases . . . 14
    28.13 GridFS . . . 15
    28.14 Collections . . . 15
    28.15 getLastError Command . . . 15
    28.16 SQL to Mongo Mapping Chart . . . 15
    28.17 Replica Sets . . . 15
    28.18 Driver and Integration Center . . . 15
    28.19 Optimizing Object IDs . . . 16
    28.20 Querying and nulls . . . 16
    28.21 Replica Sets - Rollbacks . . . 16
    28.22 The Database and Caching . . . 16
    28.23 Dates . . . 16
    28.24 ObjectId . . . 16
    28.25 Documents . . . 16
    28.26 Replica Set Sample Configurations . . . 16
    28.27 Replica Set Versions and Compatibility . . . 17
    28.28 Connecting to Replica Sets from Clients . . . 17
    28.29 Durability and Repair . . . 17
    28.30 Mongo Metadata . . . 17
    28.31 Advanced Queries . . . 17
    28.32 Copy Database Commands . . . 17
    28.33 List of Database Commands . . . 18
    28.34 Replica Set Commands . . . 18
    28.35 About the local database . . . 18
    28.36 Capped Collections . . . 18
    28.37 Building indexes with replica sets . . . 18
    28.38 Quickstart . . . 18
    28.39 Indexing as a Background Operation . . . 18
    28.40 Security and Authentication . . . 19
    28.41 collStats Command . . . 19
    28.42 Sharding Limits . . . 19
    28.43 Replica Set Internals . . . 19
    28.44 Replication . . . 19
    28.45 MongoDB cluster config best practices . . . 19
    28.46 Splitting Shard Chunks . . . 19
    28.47 fsync Command . . . 19

28.48 Viewing and Terminating Current Operation . . . 20
28.49 Import Export Tools . . . 20
28.50 Aggregation . . . 20
28.51 MapReduce . . . 20
28.51.1 Troubleshooting MapReduce . . . 20
28.52 Geospatial Indexing . . . 20
28.53 Verifying Propagation of Writes with getLastError . . . 21
28.54 Replica Set FAQ . . . 21
28.55 Forcing a Member to be Primary . . . 21
28.56 getLastError_old . . . 21
28.57 Doc Index . . . 21
28.58 Quickstart Unix . . . 21
28.59 Commands . . . 21
28.60 Changing Config Servers . . . 22
28.61 Updating . . . 22
28.62 --directoryperdb . . . 22
28.63 findAndModify Command . . . 22
28.64 --quiet . . . 22
28.65 --syncdelay . . . 22
28.66 A Sample Configuration Session . . . 22
28.67 Configuring Sharding . . . 23
28.68 Sharding Internals . . . 23
28.69 Sharding Introduction . . . 23
28.70 How does concurrency work . . . 23
28.71 Shard Ownership . . . 23
28.72 Backups . . . 23
28.73 Tutorial . . . 23
28.74 Upgrading to Replica Sets . . . 24
28.75 Replica set internals - idempotence . . . 24
28.76 Introduction - How Mongo Works . . . 24
28.77 Reconfiguring when Members are Up . . . 24
28.78 Replication Oplog Length . . . 24
28.79 Why Replica Sets . . . 24
28.80 Replica Sets - Basics . . . 24
28.81 Optimizing Storage of Small Objects . . . 25
28.82 GridFS Tools . . . 25
28.83 Sharding Administration . . . 25
28.84 Backing Up Sharded Cluster . . . 25
28.85 Replica Sets Limits . . . 25
28.86 TTL Monitor . . . 25
28.87 flushRouterConfig command . . . 25
28.88 removeshard command . . . 26
28.89 Storing Data . . . 26
28.90 Mongo Concepts and Terminology . . . 26
28.91 Database References . . . 27
28.92 Indexes in Mongo . . . 27
28.93 Optimizing Mongo Performance . . . 27
28.94 Components . . . 27
28.95 Locking in Mongo . . . 27
28.96 Design Overview . . . 27
28.97 Quickstart OS X . . . 27
28.98 Replica Sets slaveDelay . . . 28
28.99 What is a Name Space . . . 28
28.100 Structuring Data for Mongo . . . 28
28.101 Moving Chunks . . . 28
28.102 setParameter Command . . . 28
28.103 Why so many "Connection Accepted" messages logged? . . . 29
28.104 Sharding Design . . . 29
28.105 Choosing a Shard Key . . . 29
28.106 Sharding Config Schema . . . 29
28.107 CentOS and Fedora Packages . . . 29
28.108 Replica Sets - Priority . . . 29
28.109 Simple Initial Sharding Architecture . . . 29
28.110 Indexing Advice and FAQ . . . 29
28.111 Changing a Shard Key . . . 30
28.112 Sorting and Natural Order . . . 30
28.113 serverStatus Command . . . 30
28.114 Bulk Inserts . . . 30
28.115 getCmdLineOpts command . . . 30
28.116 Updates . . . 30
28.117 Do I Have to Worry About SQL Injection . . . 30
28.118 Replica Set Configuration . . . 31
28.119 Replica Sets - Voting . . . 31
28.120 Mongo Database Administration . . . 31
28.121 Sharding FAQ . . . 31

28.122 Index-Related Commands . . . 31
28.123 Storing Files . . . 31
28.124 Server-Side Processing . . . 31
28.125 Quickstart Windows . . . 32
28.126 SSL . . . 32
28.127 Validate Command . . . 32
28.128 Replica Pairs . . . 32
28.129 Aggregation Framework . . . 32
28.129.1 Aggregation Framework - $group . . . 32
28.129.2 Aggregation Framework - $limit . . . 32
28.129.3 Aggregation Framework - $match . . . 33
28.129.4 Aggregation Framework - $project . . . 34
28.129.5 Aggregation Framework - $skip . . . 34
28.129.6 Aggregation Framework - $sort . . . 34
28.129.7 Aggregation Framework - $unwind . . . 34
28.129.8 Aggregation Framework - Expression Reference . . . 34
28.129.9 SQL to Aggregation Framework Mapping Chart . . . 34
28.130 Resyncing a Very Stale Replica Set Member . . . 34
28.131 Document-Oriented Datastore . . . 34
28.132 Architecture and Components . . . 35
28.133 Recommended Configurations . . . 35
28.134 Compact Command . . . 35
28.135 Upgrading from a Non-Sharded System . . . 35
28.136 What is the Compare Order for BSON Types . . . 35
28.137 Removing . . . 35
28.138 Inserting . . . 35
28.139 min and max Query Specifiers . . . 36
28.140 Multikeys . . . 36
28.141 Replica Set Authentication . . . 36
28.142 Moving or Replacing a Member . . . 36
28.143 Using Multikeys to Simulate a Large Number of Indexes . . . 36
28.144 Replica Set Admin UI . . . 36
28.145 Internationalized Strings . . . 37
28.146 movePrimary Command . . . 37
28.147 two-phase commit . . . 37
28.148 getLog Command . . . 37
28.149 Mongo Usage Basics . . . 37
28.150 Windows Service . . . 37
28.151 slaveOk . . . 37
28.152 Query Optimizer . . . 38
28.153 Optimization . . . 38
28.154 Atomic Operations . . . 38
28.154.1 How to Make an Auto Incrementing Field . . . 38
28.155 Data Processing Manual . . . 38
28.156 Querying . . . 38
28.157 Contributing to the Documentation . . . 38
28.157.1 Mongo Documentation Style Guide . . . 39
28.158 Legal Key Names . . . 39
28.159 Connections . . . 39
28.160 Retrieving a Subset of Fields . . . 39
28.161 Timestamp data type . . . 39
28.162 Ubuntu and Debian packages . . . 39
28.163 Replica Set Tutorial . . . 39
28.164 mongoexport . . . 40
28.165 Replica Set Design Concepts . . . 40
28.166 Dot Notation (Reaching into Objects) . . . 40
28.167 Sharding Use Cases . . . 40
28.168 Reconfiguring a replica set when members are down . . . 40
28.169 createCollection Command . . . 40
28.170 renameCollection Command . . . 40
28.171 cloneCollection Command . . . 41
28.172 Why are my datafiles so large? . . . 41
28.173 Online API Documentation . . . 41
28.174 Command Line Parameters . . . 41
28.175 File Based Configuration . . . 41
28.176 How to do Snapshotted Queries in the Mongo Database . . . 41
28.177 Developer FAQ . . . 42
28.178 Searching and Retrieving . . . 42
28.178.1 Locking . . . 42
28.179 Sharding and Failover . . . 42
28.180 Adding a New Set Member . . . 42
28.181 Adding an Arbiter . . . 42
28.182 cookbook.mongodb.org . . . 42
28.183 mongostat . . . 43
28.184 Mongo Query Language . . . 43

28.185 Master Slave . . . 43
28.185.1 Halted Replication . . . 43
28.186 Atomic operation examples . . . 43
28.187 When to use GridFS . . . 43
28.188 mongosniff . . . 43
28.189 Geospatial Haystack Indexing . . . 43
28.190 Replica Sets Troubleshooting . . . 44
28.191 Caching . . . 44
28.192 Too Many Open Files . . . 44
28.193 Internal Commands . . . 44
28.194 Replica Sets - Oplog . . . 44
28.195 Restoring a single shard . . . 44
29. Conference Information . . . 45
29.1 CFP Submission Tips . . . 45
29.2 Conference Management Tools . . . 46
29.3 MongoDB Bangalore 2012 Speaker Info . . . 47
29.4 MongoDB Bangalore 2012 Sponsor Info . . . 49
29.5 MongoDB Beijing 2012 Speaker Info . . . 51
29.6 MongoDB Boston 2012 Speaker Info . . . 53
29.7 MongoDB Boston 2012 Sponsor Info . . . 55
29.8 MongoDB Chicago 2012 Speaker Info . . . 58
29.9 MongoDB Chicago 2012 Sponsor Info . . . 60
29.10 MongoDB Melbourne 2012 Speaker Info . . . 63
29.11 MongoDB Munich 2012 Speaker Info . . . 65
29.12 MongoDB Munich 2012 Sponsor Info . . . 66
29.13 MongoDB Pune 2012 Speaker Info . . . 70
29.14 MongoDB Pune 2012 Sponsor Info . . . 71
29.15 MongoDB Seattle 2012 Speaker Info . . . 74
29.16 MongoDB Seattle 2012 Sponsor Info . . . 76
29.17 MongoDB Sydney 2012 Speaker Info . . . 80
29.18 MongoDB Tokyo 2012 Speaker Info . . . 81
29.19 MongoDC 2012 Sponsor Info . . . 83
29.20 MongoNYC 2013 Speaker Info . . . 85
29.21 MongoSF 2013 Speaker Info . . . 86
29.22 MongoSV 2012 Speaker Info . . . 88
29.23 MongoSV Sponsor Info . . . 89
29.24 PresentationTips . . . 94
30. Home . . . 94
30.1 Introduction . . . 95
30.2 Downloads . . . 97
30.2.1 2.2 Release Notes . . . 97
30.2.2 2.0 Release Notes . . . 97
30.2.3 1.8 Release Notes . . . 97
30.2.3.1 Upgrading to 1.8 . . . 97
30.2.4 1.6 Release Notes . . . 100
30.2.5 1.4 Release Notes . . . 101
30.2.6 1.2.x Release Notes . . . 102
30.2.7 1.0 Changelist . . . 103
30.2.8 Version Numbers . . . 103
30.2.8.1 What's New by Version . . . 103
30.3 Drivers . . . 103
30.3.1 Hadoop . . . 105
30.3.2 Scala Language Center . . . 106
30.3.3 Haskell Language Center . . . 106
30.3.4 C Language Center . . . 106
30.3.5 CSharp Language Center . . . 107
30.3.5.1 CSharp Community Projects . . . 107
30.3.5.2 CSharp Driver LINQ Tutorial . . . 108
30.3.5.3 CSharp Driver Quickstart . . . 128
30.3.5.4 CSharp Driver Serialization Tutorial . . . 131
30.3.5.5 CSharp Driver Tutorial . . . 147
30.3.5.5.1 CSharp getLastError and SafeMode . . . 165
30.3.6 Erlang Language Center . . . 165
30.3.7 Tools and Libraries . . . 165
30.3.8 Driver Syntax Table . . . 165
30.3.9 Javascript Language Center . . . 166
30.3.9.1 Node.js . . . 166
30.3.10 JVM Languages . . . 167
30.3.11 Python Language Center . . . 168
30.3.12 PHP Language Center . . . 168
30.3.12.1 Installing the PHP Driver . . . 169
30.3.12.2 PHP Libraries, Frameworks, and Tools . . . 170
30.3.12.3 PHP - Storing Files and Big Data . . . 173
30.3.12.4 Troubleshooting the PHP Driver . . . 173
30.3.13 Ruby Language Center . . . 173

30.3.13.1 Ruby Tutorial . . . 175
30.3.13.1.1 Replica Sets in Ruby . . . 180
30.3.13.2 GridFS in Ruby . . . 181
30.3.13.3 Rails - Getting Started . . . 184
30.3.13.4 Rails 3 - Getting Started . . . 186
30.3.13.5 MongoDB Data Modeling and Rails . . . 188
30.3.13.6 Ruby External Resources . . . 192
30.3.13.7 Frequently Asked Questions - Ruby . . . 193
30.3.14 Java Language Center . . . 196
30.3.14.1 Java Driver Concurrency . . . 197
30.3.14.2 Java - Saving Objects Using DBObject . . . 198
30.3.14.3 Java Tutorial . . . 198
30.3.14.4 Java Types . . . 204
30.3.14.5 Read Preferences and Tagging in The Java Driver . . . 206
30.3.14.6 Replica Set Semantics . . . 208
30.3.14.7 Using The Aggregation Framework with The Java Driver . . . 209
30.3.15 C++ Language Center . . . 211
30.3.15.1 BSON Arrays in C++ . . . 211
30.3.15.2 C++ BSON Library . . . 212
30.3.15.3 C++ Driver Compiling and Linking . . . 214
30.3.15.4 C++ Driver Download . . . 215
30.3.15.5 C++ getLastError . . . 215
30.3.15.6 C++ Tutorial . . . 215
30.3.15.6.1 List of helper functions . . . 220
30.3.15.7 Connecting . . . 220
30.3.15.8 SQL to Shell to C++ . . . 220
30.3.16 Perl Language Center . . . 223
30.3.16.1 Contributing to the Perl Driver . . . 225
30.3.16.2 Perl Tutorial . . . 226
30.3.17 Writing Drivers and Tools . . . 226
30.3.17.1 Overview - Writing Drivers and Tools . . . 226
30.3.17.1.1 Mongo Driver Requirements . . . 226
30.3.17.1.2 Spec, Notes and Suggestions for Mongo Drivers . . . 230
30.3.17.1.3 Feature Checklist for Mongo Drivers . . . 230
30.3.17.1.4 Conventions for Mongo Drivers . . . 230
30.3.17.2 Driver Testing Tools . . . 231
30.3.17.3 Mongo Wire Protocol . . . 231
30.3.17.4 BSON . . . 236
30.3.17.5 Mongo Extended JSON . . . 239
30.3.17.6 GridFS Specification . . . 240
30.3.17.7 Implementing Authentication in a Driver . . . 242
30.3.17.8 Notes on Pooling for Mongo Drivers . . . 244
30.3.17.9 Connecting Drivers to Replica Sets . . . 246
30.3.17.10 Error Handling in Mongo Drivers . . . 246
30.4 Developer Zone . . . 247
30.4.1 Manual . . . 248
30.4.1.1 Schema Design . . . 248
30.4.1.1.1 Trees in MongoDB . . . 252
30.4.1.1.2 Tweaking performance by document bundling during schema design . . . 256
30.4.1.1.3 Using a Large Number of Collections . . . 257
30.4.1.2 Padding Factor . . . 258
30.4.1.3 Server-side Code Execution . . . 259
30.4.1.4 Explain . . . 263
30.4.1.5 Full Text Search in Mongo . . . 266
30.4.1.6 Tailable Cursors . . . 267
30.4.2 mongo - The Interactive Shell . . . 269
30.4.2.1 Scripting the shell . . . 269
30.4.2.2 Overview - The MongoDB Interactive Shell . . . 271
30.4.2.3 dbshell (mongo) Reference . . . 276
30.5 Admin Zone . . . 278
30.5.1 Journaling . . . 279
30.5.1.1 Journaling Administration Notes . . . 281
30.5.2 Production Notes . . . 282
30.5.2.1 32 bit . . . 284
30.5.2.2 iostat . . . 284
30.5.2.3 NFS . . . 285
30.5.2.4 NUMA . . . 285
30.5.2.5 SSD . . . 285
30.5.2.6 Virtualization . . . 287
30.5.3 Sharding . . . 287
30.5.3.1 Tag Aware Sharding . . . 288
30.5.4 Monitoring and Diagnostics . . . 289
30.5.4.1 Checking Server Memory Usage . . . 291
30.5.5 Starting and Stopping Mongo . . . 293
30.5.5.1 Logging . . . 295

30.5.6 Data Center Awareness . . . 297
30.5.7 DBA Operations from the Shell . . . 301
30.5.8 Troubleshooting . . . 303
30.5.8.1 Excessive Disk Space . . . 304
30.5.8.2 The Linux Out of Memory OOM Killer . . . 306
30.5.8.3 Index Versions . . . 306
30.6 Production Deployments . . . 307
30.7 MongoDB Ecosystem . . . 381
30.7.1 Admin UIs . . . 381
30.7.2 Http Interface . . . 388
30.7.3 mongoperf . . . 391
30.7.4 MongoDB Monitoring Service . . . 392
30.7.5 Munin configuration examples . . . 393
30.7.6 Windows . . . 396
30.7.7 Wireshark Support for MongoDB Protocol . . . 397
30.8 Hosting Center . . . 399
30.8.1 Amazon EC2 . . . 400
30.8.1.1 AWS Marketplace . . . 403
30.8.1.2 Automating Deployment with CloudFormation . . . 404
30.8.1.3 Amazon EC2 Quickstart . . . 411
30.8.1.4 EC2 Backup & Restore . . . 421
30.8.2 dotCloud . . . 426
30.8.3 Joyent . . . 428
30.8.4 MongoDB on Azure - Overview . . . 428
30.8.4.1 Deploying MongoDB Replica Sets to Linux on Azure . . . 429
30.8.4.2 MongoDB Installer for Windows Azure . . . 429
30.8.4.3 MongoDB on Azure . . . 429
30.8.4.4 MongoDB on Azure VM . . . 429
30.8.4.4.1 MongoDB on Azure VM - Linux Tutorial . . . 430
30.8.4.4.2 MongoDB on Azure VM - Windows Installer . . . 442
30.8.4.5 MongoDB on Azure Worker Roles . . . 444
30.8.4.5.1 MongoDB on Azure - Building an Application . . . 446
30.8.4.5.2 MongoDB on Azure Worker Roles - Configuration . . . 448
30.8.4.5.3 MongoDB on Azure Worker Roles - Deployment . . . 450
30.8.5 Rackspace Cloud . . . 455
30.8.6 RedHat OpenShift . . . 460
30.8.6.1 OpenShift Quickstart . . . 461
30.8.7 VMware CloudFoundry . . . 465
30.9 Contributors . . . 466
30.9.1 JS Benchmarking Harness . . . 466
30.9.2 MongoDB kernel code development rules . . . 468
30.9.2.1 Kernel class rules . . . 469
30.9.2.2 Kernel code style . . . 469
30.9.2.3 Kernel concurrency rules . . . 473
30.9.2.4 Kernel exception architecture . . . 474
30.9.2.5 Kernel logging . . . 474
30.9.2.6 Kernel string manipulation . . . 475
30.9.2.7 Memory management . . . 476
30.9.2.8 Writing tests . . . 476
30.9.3 Project Ideas . . . 477
30.9.4 Roadmap . . . 478
30.9.5 UI . . . 478
30.9.6 Source Code . . . 478
30.9.7 Building . . . 479
30.9.7.1 Building Boost . . . 479
30.9.7.2 Building for FreeBSD . . . 480
30.9.7.3 Building for Linux . . . 481
30.9.7.4 Building for OS X . . . 483
30.9.7.5 Building for Solaris . . . 488
30.9.7.6 Building for Windows . . . 488
30.9.7.6.1 Boost 1.41.0 Visual Studio 2010 Binary . . . 488
30.9.7.6.2 Boost and Windows . . . 488
30.9.7.6.3 Building the Mongo Shell on Windows . . . 489
30.9.7.6.4 Building with Visual Studio 2008 . . . 489
30.9.7.6.5 Building with Visual Studio 2010 . . . 491
30.9.7.7 Building Spider Monkey . . . 493
30.9.7.8 Building with V8 . . . 495
30.9.7.9 scons . . . 496
30.9.8 Database Internals . . . 496
30.9.8.1 Durability Internals . . . 496
30.9.8.2 Parsing Stack Traces . . . 498
30.9.8.3 Error Codes . . . 499
30.9.8.4 Replication Internals . . . 499
30.9.8.5 Smoke Tests . . . 500
30.9.8.6 Pairing Internals . . . 502

30.9.9 Emacs tips for MongoDB work . . . 502
30.10 Community . . . 503
30.10.1 MongoDB Masters . . . 504
30.10.1.1 A MongoDB Masters Book . . . 512
30.10.1.2 Custom Tools and Monitoring . . . 513
30.10.1.3 Deployment Strategies . . . 514
30.10.1.4 Evangelizing to DBAs . . . 514
30.10.1.5 Integrating with Hadoop, Elastic Search, SOLR . . . 515
30.10.1.6 Notes from 2012 MongoDB Masters Summit . . . 515
30.10.1.7 ODM Breakout Session . . . 516
30.10.1.8 Ops War Stories . . . 516
30.10.1.9 Schema Design Session . . . 516
30.10.2 Technical Support . . . 516
30.10.3 MongoDB Commercial Services Providers . . . 516
30.10.3.1 Visit the 10gen Offices . . . 520
30.10.4 User Feedback . . . 520
30.10.5 Job Board . . . 522
30.11 About . . . 522
30.11.1 Gotchas . . . 522
30.11.2 Philosophy . . . 524
30.11.3 Use Cases . . . 525
30.11.3.1 Business Intelligence . . . 528
30.11.3.2 Hadoop Quick Start . . . 528
30.11.3.3 Hadoop Scenarios . . . 532
30.11.3.4 How MongoDB is Used in Media and Publishing . . . 534
30.11.3.5 Use Case - Session Objects . . . 536
30.11.4 MongoDB-Based Applications . . . 537
30.11.5 Events . . . 538
30.11.5.1 MongoDB Study Groups . . . 543
30.11.5.2 MongoDB User Groups (MUGs) . . . 543
30.11.5.3 Video & Slides from Recent Events and Presentations . . . 545
30.11.5.4 Your Go-to Resource for Running a MongoDB User Group . . . 548
30.11.5.4.1 Presentation Tips for MongoDB User Groups . . . 549
30.11.6 MongoDB Archives and Articles . . . 550
30.11.7 Benchmarks . . . 551
30.11.8 FAQ . . . 551
30.11.9 Misc . . . 551
30.11.9.1 Demo App in Python . . . 551
30.11.9.2 MongoDB, CouchDB, MySQL Compare Grid . . . 551
30.11.9.3 nutshell . . . 552
30.11.9.4 v2.2 . . . 552
30.11.9.5 Comparing Mongo DB and Couch DB . . . 553
30.11.10 Licensing . . . 554
30.12 International Docs . . . 554
30.13 Books . . . 555
30.14 Slides and Video . . . 556
30.15 Alerts . . . 557

Slide Gallery

Redirection Notice
This page should redirect to http://www.10gen.com/presentations.

.bookmarks

Recent bookmarks in MongoDB

This page is a container for all the bookmarks in this space. Do not delete or move it or you will lose all your bookmarks.
Bookmarks in MongoDB | Links for MongoDB

The 15 most recent bookmarks in MongoDB

There are no bookmarks to display.

1.1 Development Cycle

Redirection Notice
This page should redirect to [1.2.0 Release Notes].

Creating and Deleting Indexes

Redirection Notice
This page should redirect to Indexes.

C Sharp Language Center

Redirection Notice
This page should redirect to CSharp Language Center.

Diagnostic Tools

Redirection Notice
This page should redirect to Monitoring and Diagnostics.

Django and MongoDB

Redirection Notice
This page should redirect to Python Language Center.

Getting Started

Redirection Notice
This page should redirect to Quickstart.

International Documentation

Redirection Notice
This page should redirect to International Docs.

Monitoring

Redirection Notice
This page should redirect to Monitoring and Diagnostics.

Older Downloads

Redirection Notice
This page should redirect to Downloads.

PyMongo and mod_wsgi

Redirection Notice
This page should redirect to Python Language Center.

Python Tutorial

Redirection Notice
This page should redirect to Python Language Center.

Recommended Production Architectures

Redirection Notice
This page should redirect to Production Notes.

v0.8 Details

Existing Core Functionality

- Basic Mongo database functionality: inserts, deletes, queries, indexing.
- Master / Slave Replication
- Replica Pairs
- Server-side javascript code execution

New to v0.8

- Drivers for Java, C++, Python, Ruby.
- db shell utility
- (Very) basic security
- $or
- Clean up logging
- Performance test baseline
- getlasterror
- Large capped collections
- Bug fixes (compound index keys, etc.)
- Import/Export utility
- Allow any _id that is unique, and verify uniqueness

Wanted, but may not make it

- AMI's
- Unlock eval()?
- Better disk full handling
- better replica pair negotiation logic (for robustness)

Building SpiderMonkey

Redirection Notice
This page should redirect to Building Spider Monkey.

Documentation

Redirection Notice
This page should redirect to Home.

Dot Notation

Redirection Notice
This page should redirect to Dot Notation (Reaching into Objects).

Dot Notation

Redirection Notice
This page should redirect to Dot Notation (Reaching into Objects).

Getting the Software

Placeholder - $$$ TODO

Language Support

Redirection Notice
This page should redirect to Drivers.

Mongo Administration Guide

Redirection Notice
This page should redirect to Admin Zone.

Working with Mongo Objects and Classes in Ruby

Redirection Notice
This page should redirect to Ruby Language Center.

MongoDB Language Support

Redirection Notice
This page should redirect to Language Support.

Community Info

Redirection Notice
This page should redirect to Community.

Internals

Cursors

Tailable Cursors

See *p/db/dbclient.h* for example of how, on the client side, to support tailable cursors.

Set

```
Option_CursorTailable = 2
```

in the `queryOptions int` field to indicate you want a tailable cursor.

If you get back no results when you query the cursor, keep the cursor live if cursorid is still nonzero. Then, you can issue future `getMore` requests for the cursor.

If a `getMore` request has the `resultFlag ResultFlag_CursorNotFound` set, the cursor is not longer valid. It should be marked as "dead" on the client side.

```
ResultFlag_CursorNotFound = 1
```

See the Queries and Cursors section of the Mongo Developers' Guide for more information about cursors.

See Also

- The Queries and Cursors section of the Mongo Developers' Guide for more information about cursors

TreeNavigation

Follow @mongodb

Old Pages

MongoDB - A Developer's Tour

Redirection Notice
This page should redirect to Manual.

Mongo Developers' Guide

Redirection Notice
This page should redirect to Manual.

HowTo

Redirection Notice
This page should redirect to Developer FAQ.

Database Profiler

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/manage-the-database-profiler/.

Updating Data in Mongo

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/database-references/.

Queries and Cursors

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/read-operations/#read-operations-cursors.

Indexes

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/indexes/.

Object IDs

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/object-id/.

OR operations in query expressions

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/operator/or/.

Cursors

Redirection Notice
This page should redirect to Internals.

Data Types and Conventions

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/document/#bson-types.

Databases

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/glossary/#term-database.

GridFS

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/gridfs/.

Collections

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/glossary/#term-collection.

getLastError Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/write-operations/#write-concern.

SQL to Mongo Mapping Chart

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/sql-comparison/.

Replica Sets

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/replication/.

Driver and Integration Center

Redirection Notice
This page should redirect to http://www.mongodb.org/display/DOCS/Writing+Drivers+and+Tools.

Optimizing Object IDs

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/document/#record-documents.

Querying and nulls

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/developers/#faq-developers-query-for-nulls.

Replica Sets - Rollbacks

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/replication/#replica-set-rollback.

The Database and Caching

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/fundamentals/#faq-database-and-caching.

Dates

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/document/#date.

ObjectId

Redirection Notice
This page should redirect to Object IDs.

Documents

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/document/.

Replica Set Sample Configurations

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/replication-architectures/.

Replica Set Versions and Compatibility

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/release-notes/replica-set-features/.

Connecting to Replica Sets from Clients

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/connection-string/.

Durability and Repair

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/recover-data-following-unexpected-shutdown/.

Mongo Metadata

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/system-collections/.

Advanced Queries

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/operators/.

Copy Database Commands

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/copy-databases-between-instances/.

List of Database Commands

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/command/.

Replica Set Commands

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/replica-commands/.

About the local database

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/replication-internals/#replica-set-local-database.

Capped Collections

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/capped-collections/.

Building indexes with replica sets

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/indexes/#index-building-replica-sets.

Quickstart

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/installation/.

Indexing as a Background Operation

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/indexes/#index-creation-background.

Security and Authentication

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/security/.

collStats Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/collection-statistics/.

Sharding Limits

Redirection Notice
This page should redirect to
http://docs.mongodb.org/manual/reference/limits/#Operations%20Unavailable%20in%20Sharded%20Environments.

Replica Set Internals

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/replication-internals/.

Replication

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/replication/.

MongoDB cluster config best practices

Splitting Shard Chunks

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/sharding/#splitting-chunks.

fsync Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/command/fsync/.

Viewing and Terminating Current Operation

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/current-op/.

Import Export Tools

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/import-export/.

Aggregation

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/aggregation/.

MapReduce

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/map-reduce/.

Troubleshooting MapReduce

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/map-reduce/#map-reduce-troubleshooting.

Geospatial Indexing

Redirection Notice
This page should redirect to [http://docs.mongodb.org/manual/core/geospatial-indexes/].

Verifying Propagation of Writes with getLastError

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/write-operations/#write-concern.

Replica Set FAQ

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/replica-sets/.

Forcing a Member to be Primary

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/force-member-to-be-primary/.

getLastError_old

Redirection Notice
This page should redirect to getLastError Command.

Doc Index

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/genindex/.

Quickstart Unix

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/install-mongodb-on-linux/.

Commands

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/commands/.

Changing Config Servers

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/sharding/#sharding-procedure-config-server.

Updating

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/update/.

--directoryperdb

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/mongod/#cmdoption-mongod--directoryperdb.

findAndModify Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/command/findAndModify/.

--quiet

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/mongod/#cmdoption-mongod--quiet.

--syncdelay

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/mongod/#cmdoption-mongod--syncdelay.

A Sample Configuration Session

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/deploy-shard-cluster/.

Configuring Sharding

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/sharding/.

Sharding Internals

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/sharding-internals.

Sharding Introduction

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/sharding/.

How does concurrency work

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/concurrency/.

Shard Ownership

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/sharding-internals/.

Backups

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/backups/.

Tutorial

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/getting-started/.

Upgrading to Replica Sets

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/convert-standalone-to-replica-set/.

Replica set internals - idempotence

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/glossary/#term-idempotent.

Introduction - How Mongo Works

Redirection Notice
This page should redirect to Developer Zone.

Reconfiguring when Members are Up

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/replica-configuration/#usage.

Replication Oplog Length

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/replication/#replica-set-oplog-sizing.

Why Replica Sets

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/replication/.

Replica Sets - Basics

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/replication/.

Optimizing Storage of Small Objects

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/developers/#faq-small-documents.

GridFS Tools

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/mongofiles/.

Sharding Administration

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/sharding/.

Backing Up Sharded Cluster

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/backups/#sharded-cluster-backups.

Replica Sets Limits

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/replica-sets/#replica-set-member-configurations.

TTL Monitor

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/expire-data/.

flushRouterConfig command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/commands/#flushRouterConfig.

removeshard command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/remove-shards-from-cluster/.

Storing Data

Redirection Notice
This page should redirect to Inserting.

Mongo Concepts and Terminology

See the Manual page for information on the following:

- BSON
- Collections
- Cursors
- Databases
- Documents
- GridFS (for files and very large objects)
- Indexes
- Transactions / Atomic Operations

Replication Terms

- Replication. Duplicating data on multiple servers for HA, safety, disaster recovery, and a bit of scaling. Sharding and replication are used together.
- Member. A member (server) in a replica set.
- Primary. The replica set member that is currently "master" and receives all writes. Reads from the primary are "immediately consistent".
- Secondary. A replica set member that is currently not master and is applying operations it receives from the current primary.
- Arbiter. A replica set member that votes in elections of primary, but that has no data. Arbiters are very lightweight and can be used to break ties in elections. Add an arbiter if a set has an even number of members.
- Oplog. High level log of operations used by replication.

Sharding Terms

- Sharding. The partioning / distribution of data among machines in a cluster. Each shard has different data. Sharding is the mechanism in MongoDB for building very large clusters. Note: we recommend you begin using MongoDB without sharding. It is easy to transition over to sharding later.
- Shard, Shard Key. See the sharding intro page.
- Chunk. A subset of documents in a collection that fit in a certain shard key range. See the sharding intro page.
- Config server. In a sharded environment, config servers store the metadata of the cluster. Each config server has a `mongod` process which stores metadata. Typically there are three config servers which have replicas of the same metadata (for data safety and HA). The config server mongod process is fairly lightweight and can be ran on machines performing other work. *The config server data is very important be sure to include them in your backups of the cluster.*
- mongod, mongos, mongo. MongoDB processes.

Other Concepts

- Durability / [Journaling. Write-ahead logging for crash safety.
- Object IDs. Mongo documents include an `_id` field in each document.

See Also

- Architecture and Components
- SQL to Mongo Mapping Chart

Database References

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/database-references/.

Indexes in Mongo

Redirection Notice
This page should redirect to Indexes.

Optimizing Mongo Performance

Redirection Notice
This page should redirect to Optimization.

Components

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/#manual-pages.

Locking in Mongo

Redirection Notice
This page should redirect to Developer FAQ.

Design Overview

Redirection Notice
This page should redirect to Developer Zone.

Quickstart OS X

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/install-mongodb-on-os-x/.

Replica Sets slaveDelay

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/replica-sets/#replica-set-delayed-members.

What is a Name Space

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/developers/#what-is-a-namespace.

Structuring Data for Mongo

Redirection Notice
This page should redirect to Inserting.

Moving Chunks

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/command/moveChunk/.

setParameter Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/command/setParameter.

Why so many "Connection Accepted" messages logged?

Redirection Notice
This page should redirect to Developer FAQ.

Sharding Design

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/sharding/.

Choosing a Shard Key

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/sharding-internals/#sharding-internals-shard-keys.

Sharding Config Schema

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/sharding-internals/#config-database.

h

CentOS and Fedora Packages

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/install-mongodb-on-redhat-centos-or-fedora-linux/.

Replica Sets - Priority

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/replica-sets/#replica-set-node-priority.

Simple Initial Sharding Architecture

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/sharding-architectures/.

Indexing Advice and FAQ

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/indexes/.

Changing a Shard Key

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/sharding/#faq-change-shard-key.

Sorting and Natural Order

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/glossary/#term-natural-order.

serverStatus Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/server-status/.

Bulk Inserts

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/sharding/#sharding-bulk-inserts.

getCmdLineOpts command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/command/getCmdLineOpts/.

Updates

Redirection Notice
This page should redirect to Updating.

Do I Have to Worry About SQL Injection

Redirection Notice
This page should redirect to
http://docs.mongodb.org/manual/faq/developers/#how-does-mongodb-address-sql-or-query-injection.

Replica Set Configuration

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/replica-configuration/.

Replica Sets - Voting

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/replication-internals/#replica-set-election-internals.

Mongo Database Administration

Redirection Notice
This page should redirect to Admin Zone.

Sharding FAQ

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/sharding/.

Index-Related Commands

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/indexes/.

Storing Files

Redirection Notice
This page should redirect to GridFS.

Server-Side Processing

Redirection Notice
This page should redirect to Server-side Code Execution.

Quickstart Windows

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/.

SSL

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/ssl/.

Validate Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/commands/#validate.

Replica Pairs

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/replication/.

Aggregation Framework

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/aggregation/.

Aggregation Framework - $group

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/aggregation/#_S_group.

Aggregation Framework - $limit

- Overview
- Specification
- Notes

Overview

The $limit pipeline operator limits the number of JSON documents that passes through it.

Specification

$limit takes a single numeric value as a parameter. Once that many documents have passed through the pipeline operator, no more will.

Here's a simple example:

```
db.article.aggregate(
    { $limit : 5 }
);
```

This pipeline will only pass through the first 5 documents that are seen. $limit has no effect on the content of the documents that are passed through – all fields present will be passed through as they are.

Notes

Aggregation Framework - $match

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/aggregation/#_S_match.

Overview

$match filters documents. Documents which do not match the specified predicate are filtered out and do not progress further along the aggregation pipeline. Documents which do match are passed along unchanged.

Specification

$match predicate syntax is always exactly the same as that for queries; See Querying and Advanced Queries.

Within a pipeline, the $match operator appears thusly:

```
db.article.aggregate(
    { $match : <match-predicate> }
);
```

Here is an example with a simple field equality test:

```
db.aggregate(
    { $match : { author : "dave" } }
);
```

On its own like this, this is the equivalent of db.article.find({ author : "dave" }).

Here is another example, this time with a range test:

```
db.article.aggregate(
    { $match : { score : { $gt : 50, $lte : 90 } }
);
```

Notes

$match should be placed as early in the aggregation pipeline as possible. This minimizes the number of documents after it, thereby minimizing later processing. Placing a $match at the very beginning of a pipeline will enable it to take advantage of indexes in exactly the same way as a regular query (find()/findOne()).

Aggregation Framework - $project

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/aggregation/#_S_project.

Aggregation Framework - $skip

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/aggregation/#_S_skip.

Aggregation Framework - $sort

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/aggregation/#_S_sort.

Aggregation Framework - $unwind

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/aggregation/#_S_unwind.

Aggregation Framework - Expression Reference

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/aggregation/#expressions.

SQL to Aggregation Framework Mapping Chart

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/sql-aggregation-comparison/.

Resyncing a Very Stale Replica Set Member

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/replica-sets/#replica-set-resync-stale-member.

Document-Oriented Datastore

Redirection Notice
This page should redirect to Databases.

Architecture and Components

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/#manual-pages.

Recommended Configurations

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/replication-architectures/.

Compact Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/command/compact/.

Upgrading from a Non-Sharded System

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/convert-replica-set-to-replicated-shard-cluster/.

What is the Compare Order for BSON Types

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/developers/#what-is-the-compare-order-for-bson-types.

Removing

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/delete/.

Inserting

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/create/.

min and max Query Specifiers

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/method/cursor.min/.

Multikeys

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/indexes/#index-type-multikey.

Replica Set Authentication

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/replica-sets/#replica-set-security.

Moving or Replacing a Member

Redirection Notice
This page should redirect to
http://docs.mongodb.org/manual/administration/replica-sets/#replica-set-admin-procedure-replace-member.

Using Multikeys to Simulate a Large Number of Indexes

Redirection Notice
This page should redirect to
http://docs.mongodb.org/manual/faq/indexes/#how-can-i-effectively-use-indexes-strategy-for-attribute-lookups.

Replica Set Admin UI

Redirection Notice
This page should redirect to http://www.mongodb.org/display/DOCS/Http+Interface.

Internationalized Strings

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/document/#string.

movePrimary Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/command/movePrimary/.

two-phase commit

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/perform-two-phase-commits/.

getLog Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/commands/#getLog.

Mongo Usage Basics

Redirection Notice
This page should redirect to Tutorial.

Windows Service

Redirection Notice
This page should redirect to
http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/#mongodb-as-a-windows-service.

slaveOk

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/replication/#read-preference.

Query Optimizer

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/read-operations/#query-optimization.

Optimization

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/optimization/.

Atomic Operations

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/isolate-sequence-of-operations/.

How to Make an Auto Incrementing Field

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/create-an-auto-incrementing-field/.

Data Processing Manual

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/import-export/.

Querying

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/applications/read/.

Contributing to the Documentation

Redirection Notice
This page should redirect to https://github.com/mongodb/docs/blob/master/CONTRIBUTING.rst.

Mongo Documentation Style Guide

Redirection Notice
This page should redirect to https://github.com/mongodb/docs/blob/master/meta.style-guide.rst.

Legal Key Names

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/limits/#Restrictions%20on%20Field%20Names.

Connections

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/connection-string/.

Retrieving a Subset of Fields

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/read-operations/#projection.

Timestamp data type

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/document/#timestamps.

Ubuntu and Debian packages

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/install-mongodb-on-debian-or-ubuntu-linux/.

Replica Set Tutorial

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/deploy-replica-set/.

mongoexport

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/mongoexport/.

Replica Set Design Concepts

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/replication-internals/.

Dot Notation (Reaching into Objects)

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/document/#dot-notation.

Sharding Use Cases

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/sharding/.

Reconfiguring a replica set when members are down

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/reconfigure-replica-set-with-unavailable-members.

createCollection Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/method/db.createCollection/.

renameCollection Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/command/renameCollection.

cloneCollection Command

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/command/cloneCollection/.

Why are my datafiles so large?

Redirection Notice
This page should redirect to Developer FAQ.

Online API Documentation

Redirection Notice
This page should redirect to http://api.mongodb.org/.

Command Line Parameters

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/mongod/.

File Based Configuration

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/configuration-options/.

How to do Snapshotted Queries in the Mongo Database

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/developers/#faq-developers-isolate-cursors.

Developer FAQ

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/developers/.

Searching and Retrieving

Redirection Notice
This page should redirect to Querying.

Locking

Redirection Notice
This page should redirect to Atomic Operations.

Sharding and Failover

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/sharding-architectures/#sharding-high-availability.

Adding a New Set Member

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/expand-replica-set.

Adding an Arbiter

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/replication-architectures/#replica-set-arbiter-nodes.

cookbook.mongodb.org

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorials/.

mongostat

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/mongostat/.

Mongo Query Language

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/meta-query-operators/.

Master Slave

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/master-slave/.

Halted Replication

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/master-slave/.

Atomic operation examples

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/isolate-sequence-of-operations/.

When to use GridFS

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/developers/#faq-developers-when-to-use-gridfs.

mongosniff

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/mongosniff/.

Geospatial Haystack Indexing

Redirection Notice
This page should redirect to
http://docs.mongodb.org/manual/applications/geospatial-indexes/#regospatial-indexes-haystaxk-queries.

Replica Sets Troubleshooting

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/replica-sets/#replica-set-troubleshooting.

Caching

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/storage/.

Too Many Open Files

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/administration/ulimit/.

Internal Commands

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/reference/commands/#internal-use-commands.

Replica Sets - Oplog

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/core/replication/#replica-set-oplog-sizing.

Restoring a single shard

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/tutorial/restore-single-shard/.

Conference Information

This is the home page for all information for speakers and sponsors participating in MongoDB Conferences.

- CFP Submission Tips
- Conference Management Tools
- MongoDB Bangalore 2012 Speaker Info
- MongoDB Bangalore 2012 Sponsor Info
- MongoDB Beijing 2012 Speaker Info
- MongoDB Boston 2012 Speaker Info
- MongoDB Boston 2012 Sponsor Info
- MongoDB Chicago 2012 Speaker Info
- MongoDB Chicago 2012 Sponsor Info
- MongoDB Melbourne 2012 Speaker Info
- MongoDB Munich 2012 Speaker Info
- MongoDB Munich 2012 Sponsor Info
- MongoDB Pune 2012 Speaker Info
- MongoDB Pune 2012 Sponsor Info
- MongoDB Seattle 2012 Speaker Info
- MongoDB Seattle 2012 Sponsor Info
- MongoDB Sydney 2012 Speaker Info
- MongoDB Tokyo 2012 Speaker Info
- MongoDC 2012 Sponsor Info
- MongoNYC 2013 Speaker Info
- MongoSF 2013 Speaker Info
- MongoSV 2012 Speaker Info
- MongoSV Sponsor Info
- PresentationTips

CFP Submission Tips

- Where to Submit
- Basic Guidelines
- Popular Topics
- Benefits of Submitting to our CFP
- Examples of Popular 2012 presentations

Where to Submit

We encourage people of all areas of experience submit proposals to our MongoDB days. MongoDB conferences draw a wide range of developers, sysadmins, DevOps professionals and data scientists from all levels of expertise, so we're looking for a variety of topics to cater to different audiences.

10gen Talk Proposal

Basic Guidelines

- The talk should be 30 minutes in length with 10 minutes for Q&A; your talk will be 40 minutes total
- Attendees are mostly developers, so gear your presentation towards a developer heavy audience
- Your talk should contain a 'message' or a 'theme' for attendees to take away
- Have fun with your topic! Each talk should be unique to the speaker giving it
- Have slides ready for review upon submission
- Feel free to email speakerinfo@10gen.com with questions.

Popular Topics

- Analytics
    - Using Hadoop
    - Using MongoDB Aggregation
- Optimizing Performance with MongoDB
- Gaming
- Data Modeling
- Deployment Architecture
- Migration Scenarios
- Hybrid Solutions
- Scaling MongoDB

FAQ

Q: Can I submit papers in my native language?
A: Yes, you can. We are interesting in having native language presentations for conferences in our non-English speaking cities in order to make talks more accessible to our attendees
Q: My company is interested in sponsoring the conference. Can we sign up as sponsors and have a speaking slot?
A: Yes. We do not include speakings slots with sponsorship, but we accept proposals from everyone and are happy to have speakers, sponsors and attendees attend from one company. You can email sponsor@10gen.com for more information.
Q: I have not heard back yet. How do I know if my talk has been accepted?
A: If you have not heard back, we are still working on the content for the conference you submitted for. As soon as we make a definitive decision, you will hear from us. Feel free to contact speakerinfo@10gen.com if you have questions.

Benefits of Submitting to our CFP

- Every speaker who submits gets a free ticket to the conference
- Speaking at a MongoDB Conference is a great way to promote your product, your company, and yourself!
- MongoDB Conferences are highly valued as recruiting opportunities
- Being active in the MongoDB Community enables you to grow your presence in the open source community

Examples of Popular 2012 presentations

- Get Your Spatial On with MongoDB in the Cloud
- MongoDB Schema Design Insights and Tradeoffs
- Mongodb Versatility Scaling the MapMyFitness Platform
- Mapping Flatland Using MongoDB for an MMO Crossword Game

Conference Management Tools

Tools to Effectively Plan and Execute your Conference

- Planning
- Execution
- Manage Feedback

Planning

There are a few keys to effective planning materials. They should be easy to use, non-exclusive, and they should allow you to clone projects. Below are some tools we use for our conferences.

- Mavenlink
  - use Mavenlink as an overall project management system. Syncs with Google Apps.
- Dropbox
  - use Dropbox as a collaborative document store for your team
- Jira
  - use Jira as a project management system for contracts or colleagues who only own one item for your conferences, such as design and email campains
  - Google Apps
  - use google apps to create calander invites for your speakers and staff, elimating document loss and confusion
- Fiesta.cc
  - create mailing lists easily for your speakers, sponsors and staff.
- Wiki Pages
  - Create wiki pages for your speakers, sponsors and staff to communicate key information in a living document.
- Eventbrite
  - use eventbrite for registration and ticket sales. Eventbrite is easy to use with almost any CMS and has standard promotional templates for groups without the need for customizeable options.
- Twitter
  - use twitter to promote your event and speakers, and on day of to tweet content in real time
- Qrious
  - use qrious to help your attendees network with each other, to autogenerate nametags with unique QRC (it syncs with eventbrite), and to provide your sponsors with leads.
- [Guidebook]
  - use guidebook for your mobile agenda and conference information needs

Execution

One drawback to planning conferences in multiple locations is that you often have no control over your environment, and limited knowledge of your resources until the day of the conference. Here are some ways to help keep your conference environment consistent:

- Wifi Needs:
    - Order wired lines
        - Presenters can use them and you can use them to connect a custom wireless network
    - Meraki
        - cloud managed wireless solutions put control of conference wifi in your hands
            - Tips:
            - Find out the conferences up bandwidth
            - Kill the conference's house wifi so you are not fighting with them
            - Get informed before you buy - Meraki has many different solutions for many different conferences
    - Walkie Talkies
        - These help your staff communicate immediately. Simple ones work for most environments. #protip: get on the same channel as your A/V or catering crew, and you will know immediately when there is a problem on your end or the venues
    - "Go" Bag:
        - Fill your "go" bag with these goodies and refresh immediately after a conference, and you will never be scrambling for sharpies again!
            - scissors, tape, med kit, twine, extra stickie labels, pants, pens, sharpies, batteries, usb keys, extra charges and extension cords, clickers and mini-display -> VGA adaptors
    - Film Equipment (try B&H for your needs!)
        - Record your speakers to add value to your website, or just to provide internal feedback, and avoid the cost of a film crew and the uncertainty that comes from having to use different film crews in different cities!
        - We like: Sony HXR-MC50U Ultra Compact Pro AVCHD Camcorder,Sennheiser EW112-p G3 Camera Mount Wireless Microphone System with ME2
          Lavalier Mic (G: 566-608MHz), Sony VCT-60AV Tripod with Remote in Grip
        - #protip: buy a super durable external hard drive to back up your video immediately
    - Presentation Equipment:
        - Buy 3 - 5 laptops to use at conferences, and load them with presentation software. This allows you to pre set hibernate settings, resolution etc and keeps speaker transitions smooth

Manage Feedback

Collection and managing feedback presents several difficulties. You want your feedback to be in depth, but you also want it to be low entry barrier, and you want your attendees to engage in feedback while it is fresh in their minds. Here are some tools we use.

- Twitter
    - Set up a twitter handle dedicated to your conferences. Ask people to tweet at you with feedback, and use DM to engage in dialogue. This creates a low entry barrier method of getting feedback with potential for more in depth conversation, and it prevents one negative comment (or even just a silly comment) from turning into spam on your #event twitter feed.
- Guidebook
    - Set up low entry barrier questions on guidebook with general multiple choice questions, and a link to your full survey
- [SurveyMonkey]
    - Use survey monkey to create a conference survey that ecompasses every talk, meal and any other aspect of your conference for complete feedback. Use giveaways to encourage participation
- JIRA
    - Use JIRA to encourage and promote internal feedback. Your colleagues are probably passionate about your work and have a lot to say, but may feel intimidated by addressing you directly or even feel badly for providing criticism on something you've worked very hard on. Start general and get more specific based on what feedback you get the most!

MongoDB Bangalore 2012 Speaker Info

- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

Dates and Deadlines

Date	Description
12/10/2012	Slides or Outline Due
25/10/2012	MongoDB Bangalore Workshops
26/10/2012	MongoDB Bangalore

- Show up at least 10 minutes prior to the start of your talk.

General Information

Event Page

25/10/2012 MongoDB Bangalore Workshops
26/10/2012 MongoDB Bangalore Conference
26/10/2012 MongoDB Bangalore After Party

Twitter: **#MongoDBBangalore**

Time	Description
9:15 am	Keynote
9:50 am	sessions start
1 pm - 2 pm	lunch
5:20 pm - 7:20 pm	After Party

- Show up at least 10 minutes prior to the start of your talk if you are speaking.

Location

	Conference, Hotel and After Party Location
Link	Le Meridien
Address	28, Sankey Road, P.B. No. 174, Seshadripuram Bangalore
Phone	(+91 80) 2226 2233
Contact	

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don’t wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as “raise your hand if you use MongoDB”. Your next comments will much more relevant to viewers

MongoDB Bangalore 2012 Sponsor Info

- 4 Leaf Sponsors Dates and Deadlines
- 3 Leaf Sponsors Dates and Deadlines
- 2 Leaf Sponsors Dates and Deadlines
- 1 Leaf Sponsors Dates and Deadlines
- Media Partner Sponsors Dates and Deadlines
- 10gen Contact:
- General Information
- Location
- Staff Registration
- Logo Specs
- Shipping Information

4 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of after party

Date	Description
Immediate	Submit logos - see Specs below
08 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
08 Oct	Submit Staffing form to events@10gen.com
15 Oct	Hand out your free community passes on http://mongodbpune2012.eventbrite.com/ with the code "CompanyPune" (so 10gen would be "10genPune")
14 Oct	Have your boxes arrive at venue between 9am and 5pm
26 Oct	Conference day

3 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of breakfast, coffee or lunch

Date	Description
Immediate	Submit logos - see Specs below
08 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com

08 Oct	Submit Staffing form to events@10gen.com
15 Oct	Hand out your free community passes on http://mongodbbangalore2012.eventbrite.com/ with the code "CompanyBangalore" (so 10gen would be "10genBangalore")
14 Oct	Have your boxes arrive at venue between 9am and 5pm
26 Oct	Conference day

2 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
08 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
08 Oct	Submit Staffing form to events@10gen.com
15 Oct	Hand out your free community passes on http://mongodbbangalore2012.eventbrite.com/ with the code "CompanyBangalore" (so 10gen would be "10genBangalore"))
14 Oct	Have your boxes arrive at venue between 9am and 5pm
26 Oct	Conference day

1 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- 3 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
08 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
08 Oct	Submit Staffing form to events@10gen.com
15 Oct	Hand out your free community passes on http://mongodbbangalore2012.eventbrite.com/ with the code "CompanyBangalore" (so 10gen would be "10genBangalore")
14 Oct	Have your boxes arrive at venue between 9am and 5pm
26 Oct	Conference day

Media Partner Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website, and collateral

Date	Description
Immediate	Submit logos - see Specs below

26 Oct	Conference day

10gen Contact:

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

General Information

Event Page

25/10/2012 MongoDB Bangalore Workshops
26/10/2012 MongoDB Bangalore Conference
26/10/2012 MongoDB Bangalore After Party

Twitter: **#MongoDBBangalore**

Time	Description
9:15 am	Keynote
9:50 am	sessions start
1 pm - 2 pm	lunch
5:20 pm - 7:20 pm	After Party

- Show up at least 10 minutes prior to the start of your talk if you are speaking.

Location

	Conference, Hotel and After Party Location
Link	Le Meridien
Address	28, Sankey Road, P.B. No. 174, Seshadripuram Bangalore
Phone	(+91 80) 2226 2233
Contact	

Staff Registration

Please use the attached form to submit your staffing by 01 Oct

Name	Size	Creator (Last Modifier)	Creation Date	Last Mod Date	Comment
Sponsor Staffing Sheet	33 kB	Melia Jones	Oct 04, 2012	Oct 04, 2012	

Logo Specs

Full color vector version (or high res version, if vector version not available)

Knockout vector version, preferably all white

230px wide PNG or JPEG (height dependent on shape of your trademark). Please include white space around trademark in the file.

Shipping Information

Detailed Shipping Information

TBA

MongoDB Beijing 2012 Speaker Info

- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

Dates and Deadlines

Date	Description

- Show up at least 10 minutes prior to the start of your talk.

General Information

URL/Dates
Twitter: #

Time	Description

Location

Conference Location	Speaker Dinner Location	After Party Location
Link, Address		
directions, contact	directions, contact	

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port

- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don't wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as "raise your hand if you use MongoDB". Your next comments will much more relevant to viewers

MongoDB Boston 2012 Speaker Info

- 10gen Contact
- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

10gen Contact

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

Dates and Deadlines

Date	Description
10/4/2012	Slides or Outline Due
10/15/2012	RSVP for Speaker/Sponsor dinner to events@10gen.com
10/23/2012	MongoDB Boston Workshops
10/23/2012	Speaker and Sponsor Thank you Dinner
10/24/2012	MongoDB Boston

- Show up at least 10 minutes prior to the start of your talk.

General Information

Event Page
10/23/2012 MongoDB Boston Speaker and Sponsor Thank You Dinner
10/23/2012 MongoDB Boston Workshops
10/24/2012 MongoDB Boston Conference
10/24/2012 MongoDB Boston After Party
Twitter: **#MongoDBBoston**

Time	Description
6:30 am	Event Staff Arrival, Venue Set Up
6:30 am	Staff Breakfast (at venue)

7 am	Sales Staff Arrival
7:30 am	Venue Walk Through (optional for Engineering Staff)
8 am	Engineering Staff Arrival
9:15 am	Keynote
9:50 am	sessions start
12 pm - 2 pm	lunch (during sessions)

- Show up at least 10 minutes prior to the start of your talk if you are speaking.

Location

	Conference and Hotel Location	Speaker Dinner Location	After Party Location
Link	Courtyard Boston Downtown/Tremont	647 Tremont	M.J. O'Conner's
Address	275 Tremont Street, Boston, MA 02116 USA	647 Tremont St, Boston, MA	27 Columbus Ave, Boston, MA 02116
Phone	1-617-426-1400	617-266-4600	617-482-2255
Contact			
Reservation Times	10/22 through 10/24	6:30 - 9:00	5:30 - 8:30

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don’t wear stripes or busy print

- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as “raise your hand if you use MongoDB”. Your next comments will much more relevant to viewers

MongoDB Boston 2012 Sponsor Info

- 4 Leaf Sponsors Dates and Deadlines
- 3 Leaf Sponsors Dates and Deadlines
- 2 Leaf Sponsors Dates and Deadlines
- 1 Leaf Sponsors Dates and Deadlines
- Media Partner Sponsors Dates and Deadlines
- 10gen Contact:
- General Information
- Location
- Staff Registration
- Logo Specs
- Shipping Information
- Parking and Transportation

4 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of after party

Date	Description
Immediate	Submit logos - see Specs below
12 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
12 Oct	Submit Staffing form to events@10gen.com
	Hand out your free community passes on http://mongodbboston2012.eventbrite.com/ with the code "CompanyBoston" (so 10gen would be "10genBoston")
19 Oct	RSVP for speaker and sponsor dinner at the hotel
19 Oct	Have your boxes arrive at venue between 9am and 5pm
23 Oct	MongoDB Workshops
23 Oct	Speaker and sponsor dinner at the hotel at 6:30pm
24 Oct	Conference day

3 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of breakfast, coffee or lunch

Date	Description
Immediate	Submit logos - see Specs below
12 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com

12 Oct	Submit Staffing form to events@10gen.com
	Hand out your free community passes on http://mongodbboston2012.eventbrite.com/ with the code "CompanyBoston" (so 10gen would be "10genBoston")
19 Oct	RSVP for speaker and sponsor dinner at the hotel
19 Oct	Have your boxes arrive at venue between 9am and 5pm
23 Oct	MongoDB Workshops
23 Oct	Speaker and sponsor dinner at the hotel at 6:30pm
24 Oct	Conference day

2 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
12 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
12 Oct	Submit Staffing form to events@10gen.com
	Hand out your free community passes on http://mongodbboston2012.eventbrite.com/ with the code "CompanyBoston" (so 10gen would be "10genBoston")
19 Oct	RSVP for speaker and sponsor dinner at the hotel
19 Oct	Have your boxes arrive at venue between 9am and 5pm
23 Oct	MongoDB Workshops
23 Oct	Speaker and sponsor dinner at the hotel at 6:30pm
24 Oct	Conference day

1 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- 3 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
12 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
12 Oct	Submit Staffing form to events@10gen.com
	Hand out your free community passes on http://mongodbboston2012.eventbrite.com/ with the code "CompanyBoston" (so 10gen would be "10genBoston")
19 Oct	RSVP for speaker and sponsor dinner at the hotel
19 Oct	Have your boxes arrive at venue between 9am and 5pm

23 Oct	MongoDB Workshops
23 Oct	Speaker and sponsor dinner at the hotel at 6:30pm
24 Oct	Conference day

Media Partner Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website, and collateral

Date	Description
Immediate	Submit logos - see Specs below
23 Oct	MongoDB Workshops
23 Oct	Speaker and sponsor dinner at the hotel at 6:30pm
24 Oct	Conference day

10gen Contact:

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

General Information

Event Page
10/23/2012 MongoDB Boston Speaker and Sponsor Thank You Dinner
10/23/2012 MongoDB Boston Workshops
10/24/2012 MongoDB Boston Conference
10/24/2012 MongoDB Boston After Party
Twitter: **#MongoDBBoston**

Time	Description
7:00 am	Sponsor Table Set up (Ends at 8:30
8:00 am	Registration start
8:00 am	Breakfast (ends at 9)
9:15 am	Session start
12:00 pm	Lunch (ends at 2)
All Day	Coffee Service

Location

	Conference and Hotel Location	Speaker Dinner Location	After Party Location
Link	Courtyard Boston Downtown/Tremont	647 Tremont	M.J. O'Conner's
Address	275 Tremont Street, Boston, MA 02116 USA	647 Tremont St, Boston, MA	27 Columbus Ave, Boston, MA 02116
Phone	1-617-426-1400	617-266-4600	617-482-2255
Reservation Times	10/22 through 10/24	6:30 - 9:00	5:30 - 8:30

Staff Registration

Please use the attached form to submit your staffing by 12 Oct

Name	Size	Creator (Last Modifier)	Creation Date	Last Mod Date	Comment
Sponsor Staffing Sheet	33 kB	Melia Jones	Oct 03, 2012	Oct 03, 2012	

Logo Specs

Full color vector version (or high res version, if vector version not available)

Knockout vector version, preferably all white

230px wide PNG or JPEG (height dependent on shape of your trademark). Please include white space around trademark in the file.

Shipping Information

Courtyard Boston Downtown Tremont
275 Tremont Street
Boston, MA 02116
Megan Cote (Catering Manager)
617-426-1400
Hold for: Edmond Valpoort (10gen Group)
Arrival 10/23/12

Parking and Transportation

TBA

MongoDB Chicago 2012 Speaker Info

- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

Dates and Deadlines

Date	Description
10/23/12	Slides Due
11/5/12	RSVP for Speaker/Sponsor Dinner to events@10gen.com
11/12/2012	MongoDB Chicago Workshops
11/12/2012	MongoDB Chicago Speaker and Sponsor Thank You Dinner
11/13/2012	MongoDB Chicago Conference
11/13/2012	MongoDB Chicago After Party

General Information

Event Page

MongoDB Chicago - Tuesday, November 13, 2012

Twitter: **#MongoDBChicago**

Time	Description
7:00 am	Event Staff Arrival, Venue Set Up
7:00 am	Staff Breakfast (at venue)

7:30 am	Sales Staff Arrival
8:00 am	Venue Walk Through (optional for Engineering Staff)
8:30 am	Engineering Staff Arrival
9:15 am	Keynote
10:00 am	sessions start
12 pm - 2 pm	lunch (during sessions)

- Show up at least 10 minutes prior to the start of your talk.

Location

	Conference and Hotel Location	Speaker Dinner Location	After Party Location
Link	Hyatt Chicago Magnificent Mile	The Bedford	Timothy O'Toole's Pub Chicago
Address	633 North St Clair Street, Chicago, Illninois 60611	1612 West Division Street, Chicago, Ilinois	622 North Fairbanks Ct, Chicago, IL 60611
Phone	1-312-787-1234	773-235-8800	312-642-0700
Reservation Times	11/11 through 11/14		5:30 - 8:30

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

Presentation Tips

Filming

We Are Filming You!

- Don’t wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as “raise your hand if you use MongoDB”. Your next comments will much more

relevant to viewers

MongoDB Chicago 2012 Sponsor Info

- 4 Leaf Sponsors Dates and Deadlines
- 3 Leaf Sponsors Dates and Deadlines
- 2 Leaf Sponsors Dates and Deadlines
- 1 Leaf Sponsors Dates and Deadlines
- Media Partner Sponsors Dates and Deadlines
- 10gen Contact:
- General Information
- Location
- Staff Registration
- Logo Specs
- Shipping Information
- Parking and Transportation

4 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of after party

Date	Description
Immediate	Submit logos - see Specs below
Nov 5	Submit Reminder and Thank You email copy to sponsor@10gen.com
Nov 5	Submit Staffing form to events@10gen.com
	Hand out your free community passes on http://mongodbchicago2012.eventbrite.com/ with the code "CompanyChicago" (so 10gen would be "10genChicago")
Nov 5	RSVP for speaker and sponsor dinner at the hotel
Nov 9	Have your boxes arrive at venue between 9am and 5pm
Nov 9	Activate your Qrious event code for your event reps
Nov 12	MongoDB Workshops
Nov 12	Speaker and sponsor dinner at the hotel at 6:30pm
Nov 13	Conference day

3 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of breakfast, coffee or lunch

Date	Description
Immediate	Submit logos - see Specs below
Nov 5	Submit Reminder and Thank You email copy to sponsor@10gen.com

Nov 5	Submit Staffing form to events@10gen.com
	Hand out your free community passes on http://mongodbchicago2012.eventbrite.com/ with the code "CompanyChicago" (so 10gen would be "10genChicago")
Nov 5	RSVP for speaker and sponsor dinner at the hotel
Nov 9	Have your boxes arrive at venue between 9am and 5pm
Nov 9	Activate your Qrious event code for your event reps
Nov 12	MongoDB Workshops
Nov 12	Speaker and sponsor dinner at the hotel at 6:30pm
Nov 13	Conference day

2 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
Nov 5	Submit Reminder and Thank You email copy to sponsor@10gen.com
Nov 5	Submit Staffing form to events@10gen.com
	Hand out your free community passes on http://mongodbchicago2012.eventbrite.com/ with the code "CompanyChicago" (so 10gen would be "10genChicago")
Nov 5	RSVP for speaker and sponsor dinner at the hotel
Nov 9	Have your boxes arrive at venue between 9am and 5pm
Nov 9	Activate your Qrious event code for your event reps
Nov 12	MongoDB Workshops
Nov 12	Speaker and sponsor dinner at the hotel at 6:30pm
Nov 13	Conference day

1 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- 3 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
Nov 5	Submit Reminder and Thank You email copy to sponsor@10gen.com
Nov 5	Submit Staffing form to events@10gen.com
	Hand out your free community passes on http://mongodbchicago2012.eventbrite.com/ with the code "CompanyChicago" (so 10gen would be "10genChicago")

Nov 5	RSVP for speaker and sponsor dinner at the hotel
Nov 9	Have your boxes arrive at venue between 9am and 5pm
Nov 12	MongoDB Workshops
Nov 12	Speaker and sponsor dinner at the hotel at 6:30pm
Nov 13	Conference day

Media Partner Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website, and collateral

Date	Description
Immediate	Submit logos - see Specs below
12 Nov	MongoDB Workshops
12 Nov	Speaker and sponsor dinner at the hotel at 6:30pm
12 Nov	Conference day

10gen Contact:

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

General Information

Event Page

MongoDB Chicago - Tuesday, November 13, 2012

Twitter: **#MongoDBChicago**
11/12/2012 MongoDB Chicago Speaker and Sponsor Thank You Dinner
11/12/2012 MongoDB Chicago Workshops
11/13/2012 MongoDB Chicago Conference
11/13/2012 MongoDB Chicago After Party

Time	Description
7:30 am	Sponsor Table Set up (Ends at 8:30
8:30 am	Registration start
8:30 am	Breakfast (ends at 9)
9:30 am	Session start
12:00 pm	Lunch (ends at 2)
All Day	Coffee Service

Location

	Conference and Hotel Location	Speaker Dinner Location	After Party Location
Link	Hyatt Chicago Magnificent Mile	The Bedford	Timothy O'Toole's Pub Chicago
Address	633 North St Clair Street, Chicago, Illninois 60611	1612 West Division Street, Chicago, Ilinois	622 North Fairbanks Ct, Chicago, IL 60611
Phone	1-312-787-1234	773-235-8800	312-642-0700

Reservation Times	11/11 through 11/14		5:30 - 8:30

Staff Registration

Please use the attached form to submit your staffing by 5 Nov

Name	Size	Creator (Last Modifier)	Creation Date	Last Mod Date	Comment
Sponsor Staffing Sheet	33 kB	Melia Jones	Oct 26, 2012	Oct 26, 2012	

Logo Specs

Full color vector version (or high res version, if vector version not available)

Knockout vector version, preferably all white

230px wide PNG or JPEG (height dependent on shape of your trademark). Please include white space around trademark in the file.

Shipping Information

Hyatt Chicago Magnificent Mile
633 North St. Clair, 2nd Floor
Chicago, IL 60611
Whitney Bindus (Convention Services Mgr)
312-274-4449
Hold for: Edmond Valpoort (10gen Group)
Arrival 11/09/12

Parking and Transportation

TBA

MongoDB Melbourne 2012 Speaker Info

- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

Dates and Deadlines

ALL SLIDES SHOULD BE SUBMITTED ASAP IF NOT ALREADY!

General Information

Event Page

11/08/2012 MongoDB Melbourne Speaker and Sponsor Thank You Dinner (to be organized by Tia day of)
11/08/2012 MongoDB MelbourneWorkshops
11/09/2012 MongoDB Melbourne Conference
11/09/2012 MongoDB Melbourne After Party

Twitter: **#MongoMelbourne**

Time	Description
7:00 am	Event Staff Arrival, Venue Set Up
7:00 am	Staff Breakfast (at venue)

7:30 am	Sales Staff Arrival
8:00 am	Venue Walk Through (optional for Engineering Staff)
8:30 am	Engineering Staff Arrival
8:30 am	Registration
9:30 am	Welcome
9:50 am	sessions start
12 pm - 2 pm	lunch (during sessions)

- Show up at least 10 minutes prior to the start of your talk if you are speaking.

Location

Conference and Hotel Location	After Party Location
Melbourne Marriott Hotel	Trunk in the Rintel Room
Corner Exhibition and Lonsdale Streets Melbourne, Victoria, 3000 Australia	275 Exhibition Street, Melbourne, Victoria, 3000 Australia
	5:30 pm - 8:30 pm

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don’t wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as “raise your hand if you use MongoDB”. Your next comments will much more relevant to viewers

MongoDB Munich 2012 Speaker Info

- 10gen Contact:
- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

10gen Contact:

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

Dates and Deadlines

Date	Description
01/10/2012	Slides or Outline Due
01/10/2012	RSVP for Speaker/Sponsor dinner to events@10gen.com
15/10/2012	MongoDB Munich Workshop
15/10/2012	Speaker and Sponsor Thank You Dinner (6 pm - 9 pm)
16/10/2012	MongoDB Munich

- Show up at least 10 minutes prior to the start of your talk.

General Information

Event Page
15/10/2012 MongoDB Munich Workshops
15/10/2012 MongoDB Munich Conference
Twitter: **#MongoDBMunich**
WiFi: MongoDB Munich, password Munich

Time	Description
8 am - 9 am	breakfast and registration
9:15 am - 9:45 am	keynote
10 am - 12:55 pm	sessions
12 pm - 2 pm	lunch served
1:45 pm - 5:15 pm	session

Location

Conference Location	Speaker Dinner Location	After Party Location
HILTON MUNICH PARK HOTEL http://www3.hilton.com/en/hotels/bavaria/hilton-munich-park-hotel-MUCHITW/index.html AM TUCHERPARK MUNICH, GERMANY 80538	HILTON MUNICH PARK HOTELAM TUCHERPARKMUNICH, GERMANY 80538	HILTON MUNICH PARK HOTELAM TUCHERPARKMUNICH, GERMANY 80538

49-89-38450	49-89-38450	49-89-38450

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don’t wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as “raise your hand if you use MongoDB”. Your next comments will much more relevant to viewers

MongoDB Munich 2012 Sponsor Info

- 4 Leaf Sponsors Dates and Deadlines
- 3 Leaf Sponsors Dates and Deadlines
- 2 Leaf Sponsors Dates and Deadlines
- 1 Leaf Sponsors Dates and Deadlines
- Media Partner Sponsors Dates and Deadlines
- 10gen Contact:
- General Information
- Location
- Staff Registration
- Logo Specs
- Shipping Information
- Parking and Transportation

4 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email

- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of after party

Date	Description
Immediate	Submit logos - see Specs below
01 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
01 Oct	Submit Staffing form to events@10gen.com
	Hand out your free community passes on http://mongodbmunich2012.eventbrite.com/ with the code "CompanyMunich" (so 10gen would be "10genMunich")
01 Oct	RSVP for speaker and sponsor dinner at the hotel
09 Oct	Have your boxes arrive at venue between 9am and 5pm
15 Oct	MongoDB Workshops
15 Oct	Speaker and sponsor dinner at the hotel at 6:30pm
16 Oct	Conference day

3 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of breakfast, coffee or lunch

Date	Description
Immediate	Submit logos - see Specs below
01 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
01 Oct	Submit Staffing form to events@10gen.com
01 Oct	Hand out your free community passes on http://mongodbmunich2012.eventbrite.com/ with the code "CompanyMunich" (so 10gen would be "10genMunich")
01 Oct	RSVP for speaker and sponsor dinner at the hotel
09 Oct	Have your boxes arrive at venue between 9am and 5pm
15 Oct	MongoDB Workshops
15 Oct	Speaker and sponsor dinner at the hotel at 6:30pm
16 Oct	Conference day

2 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community

Date	Description

Immediate	Submit logos - see Specs below
01 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
01 Oct	Submit Staffing form to events@10gen.com
01 Oct	Hand out your free community passes on http://mongodbseattle2012.eventbrite.com/ with the code "CompanyMunich" (so 10gen would be "10genMunich")
01 Oct	RSVP for speaker and sponsor dinner at the hotel
09 Oct	Have your boxes arrive at venue between 9am and 5pm
15 Oct	MongoDB Workshops
15 Oct	Speaker and sponsor dinner at the hotel 6:30pm
16 Oct	Conference day

1 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website and collateral
- 50 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 3 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
01 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
01 Oct	Submit Staffing form to events@10gen.com
01 Oct	Hand out your free community passes on http://mongodbmunich2012.eventbrite.com/ with the code "CompanyMunich" (so 10gen would be "10genMunich")
01 Oct	RSVP for speaker and sponsor dinner at the hotel
09 Oct	Have your boxes arrive at venue between 9am and 5pm
15 Oct	MongoDB Workshops
15 Oct	Speaker and sponsor dinner at the hotel 6:30pm
16 Oct	Conference day

Media Partner Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website, and collateral

Date	Description
Immediate	Submit logos - see Specs below
01 Oct	Submit Staffing form to events@10gen.com
01 Oct	Hand out your free community passes on http://mongodbmunich2012.eventbrite.com/ with the code "CompanyMunich" (so 10gen would be "10genMunich")
01 Oct	RSVP for speaker and sponsor dinner at the hotel
09 Oct	Have your boxes arrive at venue between 9am and 5pm
15 Oct	MongoDB Workshops
15 Oct	Speaker and sponsor dinner at the hotel 6:30pm

16 Oct	Conference day

10gen Contact:

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

General Information

Public Event URL **Sept 16**, 2012
Official Twitter Hashtag: **#MongoDBMunich**
WiFi:MongoDB Munich, password Munich

Time	Description
7:00 am	Sponsor Table Set up (Ends at 8:30
8:00 am	Registration start
8:00 am	Breakfast (ends at 9)
9:15 am	Session start
12:00 pm	Lunch (ends at 2)
All Day	Coffee Service

Location

Conference Location	Speaker Dinner Location	After Party Location
HILTON MUNICH PARK HOTEL http://www3.hilton.com/en/hotels/bavaria/hilton-munich-park-hotel-MUCHITW/index.html AM TUCHERPARK MUNICH, GERMANY 80538	HILTON MUNICH PARK HOTELAM TUCHERPARKMUNICH, GERMANY 80538	HILTON MUNICH PARK HOTELAM TUCHERPARKMUNICH, GERMANY 80538
49-89-38450	49-89-38450	49-89-38450

Staff Registration

Please use the attached form to submit your staffing by 01 Oct

Name	Size	Creator (Last Modifier)	Creation Date	Last Mod Date	Comment
Sponsor Staffing Sheet	33 kB	Melia Jones (modified by Melia Jones)	Sep 20, 2012	Sep 20, 2012	

Logo Specs

Full color vector version (or high res version, if vector version not available)

Knockout vector version, preferably all white

230px wide PNG or JPEG (height dependent on shape of your trademark). Please include white space around trademark in the file.

Shipping Information

Detailed Shipping Information

Hilton Munich Park

c/o Business Centre/ Alexandra Pasztor

(EVENT: MongoDB Munich/10gen)

Am Tucherpark 7

80538 Muenchen

Parking and Transportation

Parking is available with a valet fee

MongoDB Pune 2012 Speaker Info

- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

Dates and Deadlines

Date	Description
08/10/2012	Slides or Outline Due
21/10/2012	MongoDB Pune
22/10/2012	Public Dev Training

- Show up at least 10 minutes prior to the start of your talk.

General Information

Event Page

21/10/2012 MongoDB Pune Conference
21/10/2012 MongoDB Pune After Party

Twitter: **#MongoDBPune**

Time	Description
8 am - 9:30 am	Registration
9:30 am	Keynote
9:55 am - 1 pm	sessions
1 pm - 2 pm	lunch
2pm - 4:45 pm	sessions
4:45 pm - 6:45 pm	After Party

Location

	Conference and After Party Location
Link	Four Points By Sheraton
Address	5th Mile Stone Nagar Road, Viman Nagar Pune, India 411014
Phone	(+91 20) 39406699

Contact	

Slide and Presentation Review

- Submit your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don’t wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as “raise your hand if you use MongoDB”. Your next comments will much more relevant to viewers

MongoDB Pune 2012 Sponsor Info

- 4 Leaf Sponsors Dates and Deadlines
- 3 Leaf Sponsors Dates and Deadlines
- 2 Leaf Sponsors Dates and Deadlines
- 1 Leaf Sponsors Dates and Deadlines
- Media Partner Sponsors Dates and Deadlines
- 10gen Contact:
- General Information
- Location
- Staff Registration
- Logo Specs
- Shipping Information
- Parking and Transportation

4 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email

- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of after party

Date	Description
Immediate	Submit logos - see Specs below
08 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
08 Oct	Submit Staffing form to events@10gen.com
15 Oct	Hand out your free community passes on http://mongodbpune2012.eventbrite.com/ with the code "CompanyPune" (so 10gen would be "10genPune")
19 Oct	Have your boxes arrive at venue between 9am and 5pm
21 Oct	Conference day

3 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of breakfast, coffee or lunch

Date	Description
Immediate	Submit logos - see Specs below
08 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
08 Oct	Submit Staffing form to events@10gen.com
15 Oct	Hand out your free community passes on http://mongodbpune2012.eventbrite.com/ with the code "CompanyPune" (so 10gen would be "10genPune")
19 Oct	Have your boxes arrive at venue between 9am and 5pm
21 Oct	Conference day

2 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
08 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
08 Oct	Submit Staffing form to events@10gen.com
15 Oct	Hand out your free community passes on http://mongodbpune2012.eventbrite.com/ with the code "CompanyPune" (so 10gen would be "10genPune")
19 Oct	Have your boxes arrive at venue between 9am and 5pm
21 Oct	Conference day

1 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- 3 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
08 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
08 Oct	Submit Staffing form to events@10gen.com
15 Oct	Hand out your free community passes on http://mongodbpune2012.eventbrite.com/ with the code "CompanyPune" (so 10gen would be "10genPune")
19 Oct	Have your boxes arrive at venue between 9am and 5pm
21 Oct	Conference day

Media Partner Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website, and collateral

Date	Description
Immediate	Submit logos - see Specs below
21 Oct	Conference day

10gen Contact:

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

General Information

Event Page

21/10/2012 MongoDB Pune Conference
21/10/2012 MongoDB Pune After Party

Twitter: **#MongoDBPune**

Time	Description
8 am - 9:30 am	Registration
9:30 am	Keynote
9:55 am - 1 pm	sessions
1 pm - 2 pm	lunch
2pm - 4:45 pm	sessions
4:45 pm - 6:45 pm	After Party

http://www.10gen.com/events/mongodb-munich

Location

Date	Description
Immediate	Submit logos - see Specs below
08 Oct	Submit Reminder and Thank You email copy to sponsor@10gen.com
08 Oct	Submit Staffing form to events@10gen.com
15 Oct	Hand out your free community passes on http://mongodbpune2012.eventbrite.com/ with the code "CompanyPune" (so 10gen would be "10genPune")
19 Oct	Have your boxes arrive at venue between 9am and 5pm
21 Oct	Conference day

Staff Registration

Please use the attached form to submit your staffing by 01 Oct

Name	Size	Creator (Last Modifier)	Creation Date	Last Mod Date	Comment
Sponsor Staffing Sheet	33 kB	Melia Jones	Oct 02, 2012	Oct 02, 2012	

Logo Specs

Full color vector version (or high res version, if vector version not available)

Knockout vector version, preferably all white

230px wide PNG or JPEG (height dependent on shape of your trademark). Please include white space around trademark in the file.

Shipping Information

Detailed Shipping Information

TBA

Parking and Transportation

Parking is available with a valet fee

MongoDB Seattle 2012 Speaker Info

- 10gen Contact:
- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations
- Parking and Transportation

10gen Contact:

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

Dates and Deadlines

Date	Description

08/17/2012	Slides or Outline Due
09/01/2012	RSVP for Speaker/Sponsor dinner to events@10gen.com
09/13/2012	MongoDB Seattle Workshop
09/13/2012	Speaker and Sponsor Thank You Dinner (6 pm - 9 pm)
09/14/2012	MongoDB Seattle

- Show up at least 10 minutes prior to the start of your talk.

General Information

Event Page
09/13/2012 MongoDB Seattle Workshops
09/14/2012 MongoDB Seattle Conference
Twitter: **#MongoDB Seattle**
WiFi: Please look for Bell Harbor Conference Center and go. No password needed

Time	Description
8 am - 9 am	breakfast and registration
9:15 am - 9:45 am	keynote
10 am - 12:55 pm	sessions
12 pm - 2 pm	lunch served
1:45 pm - 5:15 pm	session

Location

	Conference	Hotel (Sleeping Rooms & Workshops)	Speaker Dinner Location	After Party Location
Link	Bell Harbor International Conference Center	Seattle Waterfront Marriott	Aqua	Tavern Law
Address	2211 Alaskan Way, Pier 66 Seattle, WA 98121	2100 Alaskan Way · Seattle, Washington 98121 USA	2801 Alaskan Way, Pier 70 Seattle, WA 98121	1406 12th Avenue, Seattle WA 98122 map
Phone	206.441.6666	1-206-443-5000	206.956.8171	206.322.9734
Reservation Times	6 am - 5:30 pm		6 pm	5 pm - 8 pm

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker

- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don't wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as "raise your hand if you use MongoDB". Your next comments will much more relevant to viewers

Parking and Transportation

complimentary all day Garage Parking is available on site to speakers and sponsor only, please
contact events@10gen.com if you would like a parking pass; For all other attendees,
all day parking is available for $15

from SeaTac International Airport| driving | public transportation

MongoDB Seattle 2012 Sponsor Info

- 4 Leaf Sponsors Dates and Deadlines
- 3 Leaf Sponsors Dates and Deadlines
- 2 Leaf Sponsors Dates and Deadlines
- 1 Leaf Sponsors Dates and Deadlines
- Media Partner Sponsors Dates and Deadlines
- 10gen Contact:
- General Information
- Location
- Staff Registration
- Logo Specs
- Shipping Information
- Parking and Transportation

4 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of after party

Date	Description
Immediate	Submit logos - see Specs below

August 27	Submit Reminder and Thank You email copy to sponsor@10gen.com
Agust 27	Submit Staffing form to events@10gen.com
August 27	Hand out your free community passes on http://mongodbseattle2012.eventbrite.com/ with the code "CompanySeattle" (so 10gen would be "10genSeattle")
Sept 1	RSVP for speaker and sponsor dinner at ()
Sept 12	Have your boxes arrive at venue between 9am and 5pm
Sept 13	MongoDB Workshops
Sept 13	Speaker and sponsor dinner at ( ) 6:30pm
Sept 14	Conference day

3 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community
- Sponsorship of breakfast, coffee or lunch

Date	Description
Immediate	Submit logos - see Specs below
August 27	Submit Reminder and Thank You email copy to sponsor@10gen.com
Agust 27	Submit Staffing form to events@10gen.com
August 27	Hand out your free community passes on http://mongodbseattle2012.eventbrite.com/ with the code "CompanySeattle" (so 10gen would be "10genSeattle")
Sept 1	RSVP for speaker and sponsor dinner at ()
Sept 12	Have your boxes arrive at venue between 9am and 5pm
Sept 13	MongoDB Workshops
Sept 13	Speaker and sponsor dinner at ( ) 6:30pm
Sept 14	Conference day

2 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
August 27	Submit Reminder and Thank You email copy to sponsor@10gen.com
Agust 27	Submit Staffing form to events@10gen.com
August 27	Hand out your free community passes on http://mongodbseattle2012.eventbrite.com/ with the code "CompanySeattle" (so 10gen would be "10genSeattle")

Sept 1	RSVP for speaker and sponsor dinner at ()
Sept 12	Have your boxes arrive at venue between 9am and 5pm
Sept 13	MongoDB Workshops
Sept 13	Speaker and sponsor dinner at ( ) 6:30pm
Sept 14	Conference day

1 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website and collateral
- 50 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 3 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
August 27	Submit Reminder and Thank You email copy to sponsor@10gen.com
Agust 27	Submit Staffing form to events@10gen.com
August 27	Hand out your free community passes on http://mongodbseattle2012.eventbrite.com/ with the code "CompanySeattle" (so 10gen would be "10genSeattle")
Sept 1	RSVP for speaker and sponsor dinner at ()
Sept 12	Have your boxes arrive at venue between 9am and 5pm
Sept 13	MongoDB Workshops
Sept 13	Speaker and sponsor dinner at ( ) 6:30pm
Sept 14	Conference day

Media Partner Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website, and collateral

Date	Description
Immediate	Submit logos - see Specs below
Agust 27	Submit Staffing form to events@10gen.com
August 27	Hand out your free community passes on http://mongodbseattle2012.eventbrite.com/ with the code "CompanySeattle" (so 10gen would be "10genSeattle")
Sept 1	RSVP for speaker and sponsor dinner at ()
Sept 12	Have your boxes arrive at venue between 9am and 5pm
Sept 13	MongoDB Workshops
Sept 13	Speaker and sponsor dinner at ( ) 6:30pm
Sept 14	Conference day

10gen Contact:

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

General Information

Public Event URL **Sept 14**, 2012
Official Twitter Hashtag: **#MongoSeattle**
WiFi: Please look for Bell Harbor Conference Center and go. No password needed

Time	Description
7:00 am	Sponsor Table Set up (Ends at 8:30
8:00 am	Registration start
8:00 am	Breakfast (ends at 9)
9:15 am	Session start
12:00 pm	Lunch (ends at 2)
All Day	Coffee Service

Location

	Conference	Hotel (Sleeping Rooms & Workshops)	Speaker Dinner Location	After Party Location
Link	Bell Harbor International Conference Center	Seattle Waterfront Marriott	Aqua	Tavern Law
Address	2211 Alaskan Way, Pier 66 Seattle, WA 98121	2100 Alaskan Way · Seattle, Washington 98121 USA	2801 Alaskan Way, Pier 70 Seattle, WA 98121	1406 12th Avenue, Seattle WA 98122 map
Phone	206.441.6666	1-206-443-5000	206.956.8171	206.322.9734
Reservation Times	6 am - 5:30 pm		6 pm	5 pm - 8 pm

Staff Registration

Please use the attached form to submit your staffing by August 27

Name	Size	Creator (Last Modifier)	Creation Date	Last Mod Date	Comment
Sponsor Staffing Template.xlsx	33 kB	Melia Jones	Aug 08, 2012	Aug 08, 2012	Sponsor Staffing Doc

Logo Specs

Full color vector version (or high res version, if vector version not available)

Knockout vector version, preferably all white

230px wide PNG or JPEG (height dependent on shape of your trademark). Please include white space around trademark in the file.

Shipping Information

Detailed Shipping Information

Bell Harbor Conference Center
2211 Alaskan Way, Pier 66
Seattle, WA 98121
Phone: 206.441.6666

- please clearly label all packages with “10gen, sponsor name” -
  *10gen holds no responsibility for shipped items

Parking and Transportation

complimentary all day Garage Parking is available on site to speakers and sponsor only, please contact events@10gen.com if you would like a parking pass; For all other attendees, all day parking is available for $15

from SeaTac International Airport| driving | public transportation

MongoDB Sydney 2012 Speaker Info

- 10gen Contact:
- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

10gen Contact:

Tia Sorenson p e. tia@10gen.com

Dates and Deadlines

ALL SLIDES SHOULD BE SUBMITTED ASAP IF NOT ALREADY!

General Information

Event Page

11/13/2012 MongoDB Sydney Speaker and Sponsor Thank You Dinner (to be organized by Tia day of)
11/13/2012 MongoDB Sydney Workshops
11/12/2012 MongoDB Sydney Conference
11/12/2012 MongoDB Sydney After Party

Twitter: **#MongoDBSydney**

Time	Description
7:00 am	Event Staff Arrival, Venue Set Up
7:00 am	Staff Breakfast (at venue)
7:30 am	Sales Staff Arrival
8:00 am	Venue Walk Through (optional for Engineering Staff)
8:30 am	Engineering Staff Arrival
8:30 am	Registration
9:30 am	Welcome
9:50 am	sessions start
12 pm - 2 pm	lunch (during sessions)

- Show up at least 10 minutes prior to the start of your talk if you are speaking.

Location

Conference and Hotel Location	After Party Location
Four Points By Sheraton Darling Harbor	Slip Inn on the Terrace

161 Sussex Street, New South Wales 2000 Asutralia	111 Sussex Street Sydney, 2000
	5:30 pm - 8:30 pm

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don’t wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as “raise your hand if you use MongoDB”. Your next comments will much more relevant to viewers

MongoDB Tokyo 2012 Speaker Info

- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

Dates and Deadlines

Date	Description

- Show up at least 10 minutes prior to the start of your talk.

General Information

URL/Dates
Twitter: #

Time	Description

Location

Conference Location	Speaker Dinner Location	After Party Location
Link, Address		
directions, contact	directions, contact	

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don’t wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as “raise your hand if you use MongoDB”. Your next comments will much more

relevant to viewers

MongoDC 2012 Sponsor Info

- 3 Leaf Sponsors Dates and Deadlines
- 2 Leaf Sponsors Dates and Deadlines
- 1 Leaf Sponsors Dates and Deadlines
- Media Partner Sponsors Dates and Deadlines
- General Information
- Location
- Logo Specs
- Shipping Information

3 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 50 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community

Date	Description
June 15th	Submit Reminder and Thank You email copy to sponsor@10gen.com
June 15th	Register your staff with eventbrite (promotional code "DCSponsor")
June 21st	RSVP for speaker and sponsor dinner at Gordon Biersch
June 21st	Hand out your free community passes on eventbrite with the code "CompanyDC" (so 10gen would be "10genDC")
June 25th	Have your boxes arrive at venue between 9am and 5pm
June 25th	Speaker and sponsor dinner at Gordon Biersch, 6:30pm
June 26th	Conference day

2 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 50 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 6 conference passes for your table
- 6 conference passes for your community

Date	Description
June 15th	Submit Reminder and Thank You email copy to sponsor@10gen.com
June 15th	Register with eventbrite (promotional code "DCSponsor")
June 21st	RSVP for speaker and sponsor dinner at Gordon Biersch
June 21st	Hand out your free community passes on eventbrite with the code "CompanyDC" (so 10gen would be "10genDC")
June 25th	Have your boxes arrive at venue between 9am and 5pm
June 25th	Speaker and sponsor dinner at Gordon Biersch, 6:30pm
June 26th	Conference day

1 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website and collateral
- 50 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- 3 conference passes for your community

Date	Description
June 21st	Hand out your free community passes on eventbrite with the code "CompanyDC" (so 10gen would be "10genDC")
June 21st	RSVP for speaker and sponsor dinner at Gordon Biersch
June 25th	Have your boxes arrive at venue between 9am and 5pm
June 25th	Speaker and sponsor dinner at Gordon Biersch, 6:30pm
June 26th	Conference day

Media Partner Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website, and collateral

Date	Description
June 15th	Register with eventbrite (promotional code 'DCSponsor')
June 21st	RSVP for speaker and sponsor dinner at Gordon Biersch
June 25th	Have your boxes arrive at venue between 9am and 5pm
June 25th	Speaker and sponsor dinner at Gordon Biersch, 6:30pm
June 26th	Conference day

General Information

Public Event URL **June 26**, 2012
Official Twitter Hashtag: **#MongoDC**

Time	Description
7:00 am	Sponsor Table Set up (Ends at 8:30
8:00 am	Registration start
8:00 am	Breakfast (ends at 10)
9:15 am	Session start
1:00 pm	Lunch (ends at 2)
All Day	Coffee Service

Location

Conference Location	Speaker Dinner Location
Woolly Mammoth Theatre	Gordon Biersch
641 D Street, NW	900 F Street, NW

Washington, DC 20004	Washington, DC 20004
(202) 289-2443	(202) 783-5454
Get Detailed Map and Directions for Conference	Map for Speaker Dinner

Logo Specs

-Full color vector version (or high res version, if vector version not available)

-Knockout vector version, preferably all white

-230px wide PNG or JPEG (height dependent on shape of your trademark). Please include white space around trademark in the file.

Shipping Information

Detailed Shipping Information

Please ship as little as possible to the venue as there is very little holding space there. Arrange for it to arrive on Friday the 22nd or Monday the 25th between 10am and 6pm, as nobody will be there to receive the package outside of those hours.

Attention: Zacory Boatwright/10gen
Woolly Mammoth Theatre
641 D Street, NW
Washington, DC 20004

Please also be aware that we have no palette dolly. All shipments must be able to be broken up into smaller parts that may be moved by one person. Please keep in mind that our standard business hours (i.e. best times for delivery) would be Monday through Friday, 10 AM to 6 PM.

MongoNYC 2013 Speaker Info

- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

Dates and Deadlines

Date	Description

- Show up at least 10 minutes prior to the start of your talk.

General Information

URL/Dates
Twitter: #

Time	Description

Location

Conference Location	Speaker Dinner Location	After Party Location
Link, Address		
directions, contact	directions, contact	

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don't wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as "raise your hand if you use MongoDB". Your next comments will much more relevant to viewers

MongoSF 2013 Speaker Info

- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

Dates and Deadlines

Date	Description

- Show up at least 10 minutes prior to the start of your talk.

General Information

URL/Dates
Twitter: #

Time	Description

Location

Conference Location	Speaker Dinner Location	After Party Location
Link, Address		
directions, contact	directions, contact	

Slide and Presentation Review

- Submitted your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don’t wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as “raise your hand if you use MongoDB”. Your next comments will much more relevant to viewers

MongoSV 2012 Speaker Info

- 10gen Contact:
- Dates and Deadlines
- General Information
- Location
- Slide and Presentation Review
- Equipment
- Presentations

10gen Contact:

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

Dates and Deadlines

Date	Description
11/13/12	Slides Due
12/03/2012	MongoSV Speaker and Sponsor Thank You Dinner
12/03/2012	MongoSV Workshops
12/04/2012	MongoSV Conference
12/04/2012	MongoSV After Party

General Information

Event Page

12/03/2012 MongoSV Speaker and Sponsor Thank You Dinner
12/03/2012 MongoSV Workshops
12/04/2012 MongoSV Conference
12/04/2012 MongoSV After Party

Twitter: **#MongoSV**

Time	Description
6:00 am	Event Staff - Marketing: Arrival, Venue Set Up
6:30 am	Staff Breakfast (at venue)
7:00 am	Event Staff - Sales: Arrival
7:30 am	Venue Walk Through (optional for Engineering Staff)
8:00 am	Engineering Staff Arrival
9:00 am	Keynote
9:50 am	sessions start
12 pm - 2 pm	lunch (during sessions)

- Show up at least 10 minutes prior to the start of your talk if you are speaking.

Location

	Staff Hotel Location	Masters and Speakers Hotel Location	Conference and Hotel Location	Speaker Dinner Location	After Party Location
Link	Hyatt Regency Santa Clara	Hilton Santa Clara	Santa Clara Convention Center	The Left Bank	Gordon Biersch
Address	5101 Great America Parkway, Santa Clara, CA 95054	4949 Great America Parkway, Santa Clara, CA 95054	5001 Great America Pkwy., Santa Clara, CA 95054	377 Santana Row, Suite 1100, San Jose, CA 95128	33 East San Fernando Street, San Jose, CA 95113
Phone	408 - 510 - 6421	408 - 562 - 6795	800-272-6822	408-984-3500	408-294-6785
Reservation Times	12/2 or 12/3 through 12/4	12/3 through 12/4		6:00 - 9:00	6:00 - 9:00

Slide and Presentation Review

- Submit your slides to speakerinfo@10gen.com.
    - An outline or partially finished content is totally fine; we just want to make sure we get you the feedback you need in a timely fashion. PowerPoint, KeyNote, Prezi, or PDF are all acceptable formats.
- We are here to help! - we are happy to set up a session (Skype, hangout, call) to go through your presentation.

Equipment

Provided

- Projector, screen, and mic for speaker
- MacBook Mini Display port - VGA adaptor
- Wireless Clickers/Pointers - please do not take the USB adaptor with you on accident!

To Bring:

- A Laptop for your presentation
- A VGA adaptor for anything that is NOT a Mac mini display port
- Conference wifi is always unreliable so we recommend against any live demos that require Internet

Presentations

- Each presenter will have a 40 minute session time
- Please factor your Q&A time into this
- Each session has a short break between sessions to allow attendees to transition
- We have staff on hand to make sure you start and end on time.
- the session schedule can be viewed at

PresentationTips

Filming

We Are Filming You!

- Don’t wear stripes or busy print
- Do repeat questions from the audience before answering them.
- Do announce results if you poll the audience - such as “raise your hand if you use MongoDB”. Your next comments will much more relevant to viewers

MongoSV Sponsor Info

- 4 Leaf Sponsors Dates and Deadlines
- 3 Leaf Sponsors Dates and Deadlines
- 2 Leaf Sponsors Dates and Deadlines
- 1 Leaf Sponsors Dates and Deadlines
- Media Partner Sponsors Dates and Deadlines
- 10gen Contact:
- General Information
- Location
- Staff Registration

- Design Specs
- Shipping Information
- Parking and Transportation

4 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- Option to participate in passport program
- Option to activate smart phones as lead retrieval devices
- Ad posted on MongoSV Sponsor URL (see below for design specs)
- 6 conference passes for your table
- 6 conference passes for your community
- Swag or Collateral in Swag Bag
- Sponsorship of after party

Date	Description
Immediate	Submit logos - see Specs below
Nov 16	Submit Reminder and Thank You email copy to sponsor@10gen.com
Nov 16	Submit Staffing form to events@10gen.com
Nov 16	Hand out your free community passes on http://mongosv2012.eventbrite.com/ with the code "CompanySV" (so 10gen would be "10genSV")
Nov 16	Click here to RSVP for speaker and sponsor dinner at The Left Bank
Nov 23	Submit your static PDF to be posted on the sponsor URL
Nov 23	Prize confirmed for Passport Program
Nov 30	Have your boxes arrive at venue between 9am and 5pm
Nov 30	Have your collateral for coffee, breakfast or lunch sponsorship arrive at venue between 9 am and 5 pm
Nov 30	Activate your Qrious event code for your event reps
Dec 3	MongoDB Workshops
Dec 3	Speaker and sponsor dinner at the hotel at 6:30pm
Dec 4	Conference day

3 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 150 word message in "Reminder" Email
- 150 word message in "Thank You" Email
- Option to participate in passport program
- Option to activate smart phones as lead retrieval devices
- Ad posted on MongoSV Sponsor URL (see below for design specs)
- 6 conference passes for your table
- 6 conference passes for your community
- Swag or Collateral in Swag Bag
- Sponsorship of coffee, breakfast or lunch

Date	Description
Immediate	Submit logos - see Specs below
Nov 16	Submit Reminder and Thank You email copy to sponsor@10gen.com
Nov 16	Submit Staffing form to events@10gen.com

Nov 16	Hand out your free community passes on http://mongosv2012.eventbrite.com/ with the code "CompanySV" (so 10gen would be "10genSV")
Nov 16	Click here to RSVP for speaker and sponsor dinner at The Left Bank
Nov 23	Submit your static PDF to be posted on the sponsor URL
Nov 23	Prize confirmed for Passport Program
Nov 30	Have your boxes arrive at venue between 9am and 5pm
Nov 30	Have your collateral for coffee, breakfast or lunch sponsorship arrive at venue between 9 am and 5 pm
Nov 30	Activate your Qrious event code for your event reps
Dec 3	MongoDB Workshops
Dec 3	Speaker and sponsor dinner at the hotel at 6:30pm
Dec 4	Conference day

2 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on website, conference emails and collateral
- 50 word message in "Reminder" Email
- 50 word message in "Thank You" Email
- Option to participate in passport program
- Option to activate smart phones as lead retrieval devices
- Ad posted on MongoSV Sponsor URL (see below for design specs)
- 6 conference passes for your table
- 6 conference passes for your community
- Swag or Collateral in Swag Bag
- Sponsorship of coffee, breakfast or lunch

Date	Description
Immediate	Submit logos - see Specs below
Nov 16	Submit Reminder and Thank You email copy to sponsor@10gen.com
Nov 16	Submit Staffing form to events@10gen.com
Nov 16	Hand out your free community passes on http://mongosv2012.eventbrite.com/ with the code "CompanySV" (so 10gen would be "10genSV")
Nov 16	Click here to RSVP for speaker and sponsor dinner at The Left Bank
Nov 23	Submit your static PDF to be posted on the sponsor URL
Nov 23	Prize confirmed for Passport Program
Nov 30	Have your boxes arrive at venue between 9am and 5pm
Nov 30	Have your collateral for coffee, breakfast or lunch sponsorship arrive at venue between 9 am and 5 pm
Nov 30	Activate your Qrious event code for your event reps
Dec 3	MongoDB Workshops
Dec 3	Speaker and sponsor dinner at the hotel at 6:30pm
Dec 4	Conference day

1 Leaf Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website and collateral
- 50 word message in "Reminder" Email

- 50 word message in "Thank You" Email
- 3 conference passes for your community

Date	Description
Immediate	Submit logos - see Specs below
Nov 16	Submit Reminder and Thank You email copy to sponsor@10gen.com
Nov 16	Submit Staffing form to events@10gen.com
Nov 16	Hand out your free community passes on http://mongosv2012.eventbrite.com/ with the code "CompanySV" (so 10gen would be "10genSV")
Nov 23	click here to RSVP for speaker and sponsor dinner at The Left Bank
Dec 3	MongoDB Workshops
Dec 3	Speaker and sponsor dinner at the hotel at 6:30pm
Dec 3	Conference day

Media Partner Sponsors Dates and Deadlines

Included:

- Linked logo on conference emails, website, and collateral

Date	Description
Immediate	Submit logos - see Specs below
Dec 3	MongoDB Workshops
Dec 3	Speaker and sponsor dinner at the hotel at 6:30pm
Dec 3	Conference day

10gen Contact:

Melia Jones p. 773-875-6867 e. melia@10gen.com OR events@10gen.com

General Information

Event Page

12/03/2012 MongoSV Speaker and Sponsor Thank You Dinner
12/03/2012 MongoSV Workshops
12/04/2012 MongoSV Conference
12/04/2012 MongoSV After Party

Time	Description
6:30 am	Sponsor Hall Open for Set Up
8:00 am	Registration Begins/Attendees Arrive
9:00 am	Keynote
9:30 - 9:50	Session transition
9:50 am	sessions start
11:15 - 11:30	coffee break
12 pm - 2 pm	lunch (during sessions)
3:10 - 3:25	coffee break

5:10 pm	closing keynote

Location

Conference Location	Speaker Dinner Location	After Party Location
Santa Clara Convention Center	The Left Bank	Gordon Biersch
5001 Great America Pkwy., Santa Clara, CA 95054	377 Santana Row, Suite 1100, San Jose, CA 95128	33 East San Fernando Street, San Jose, CA 95113
800-272-6822	408-984-3500	408-294-6785
	6:00 - 9:00	6:00 - 9:00

Staff Registration

Please use the attached form to submit your staffing by Nov 16

Name	Size	Creator (Last Modifier)	Creation Date	Last Mod Date	Comment
4 Dec Sponsor Hall.pdf	26 kB	Melia Jones	Nov 24, 2012	Nov 24, 2012	Sponsor Hall Assigments
Sponsor Staffing Template.xlsx	33 kB	Melia Jones	Nov 05, 2012	Nov 05, 2012	Staffing Form

Design Specs

Logo

Full color vector version (or high res version, if vector version not available)

Knockout vector version, preferably all white

230px wide PNG or JPEG (height dependent on shape of your trademark). Please include white space around trademark in the file.

PDF for Sponsor URL

size: 1 page

content: Any

Live links are ok

Shipping Information

Name	Size	Creator (Last Modifier)	Creation Date	Last Mod Date	Comment
4 Dec Sponsor Hall.pdf	26 kB	Melia Jones	Nov 24, 2012	Nov 24, 2012	Sponsor Hall Assigments
Sponsor Staffing Template.xlsx	33 kB	Melia Jones	Nov 05, 2012	Nov 05, 2012	Staffing Form

Shipping Info:

- All shipments are due by November 30, 2012. Please label all boxes clearly with sponsor name and table # (see expo hall map, attached). Send events@10gen.com tracking info for the packages once sent out.
  Shipping Address:
  ATTN Kaleo Cornwell
  Santa Clara Convention Center
  5001 Great America Parkway
  Santa Clara, CA 95054
  For: 10gen/ MongoSV
  Date: Tuesday, Dec 3, 2012
  Location: Mission City Ballroom

O: (408) 748-7063 C: (206) 498-6788

Parking and Transportation

TBA

PresentationTips

Slides

- Be precise in your slides - We will make them available on our website after the conference, so make sure they are correct and to the point.
- Be visual - a clever picture, easy to read chart, or even simple bullet points convey clear information.
- Use Big Font! the rooms are large and it is always hard to see from the back of the room
- Avoid including full text such as sentences and paragraphs in your slides.
- Know your audience -The audience is very technical, and generally knowledgeable about MongoDB.
- We will briefly introduce all of the sessions at the beginning of the event to give attendees an idea of what sessions their time would best be spent attending. Please familiarize yourself with the sessions as well.
- Turn off Hibernate! You don't want your computer to go to sleep in the middle of your preso ... or your audience might as well

Logistics

- **Arrive 10 minutes early** for your presentation; This gives everyone time to make sure that you and your equipment are ready to go! We will have a room monitor run through a check list to set you up.
- We will have a room monitor in each room, so if you need something, say something.
- Staging: Presenters in SkyTop, PennTop North and South, and the Madison will be presenting from a stage and podium. Presenters in Paris or London will be presenting from a podium.
- Projector and screen: You will be connecting with a Mac mini display adaptor to a VGA connection. If you need a different adaptor for your laptop, please bring it with you.
- Mic's: A room monitor will equip with a lavalier (clip on) when you arrive in your session room.
- Internet connectivity: Please do not rely on wifi for demos as we cannot guarantee it will work properly.
- Power: we will have power running to your podium. Please make sure your machine is fully charged anyway!

Questions

- If there is a microphone, make sure to wait for the person to get it, and use it.
- Repeat or rephrase the question to confirm it is what they are asking and everyone hears it
- If the question is off-topic or lengthy, table-it or ask the person to find you after the presentation
- Move towards the person and away from the podium to engage the audience

Home

News: The v2.2 release is now available.

Documentation

- "The manual" (docs.mongodb.org)
- Installation Guide
- Tutorial
- SQL to Mongo Mapping Chart
- List of Reference Pages
- Production Notes tips and best practices
- Replication | Sharding
- Security | Backups
- Developer info by language
    - C | C# | C++ | C# & .NET | ColdFusion | Erlang | Haskell | Factor
    - Java | Javascript | PHP | Python | Ruby | Perl | more...

Production

- Hosting
- MongoDB Monitoring Service (MMS) is a free way to monitor your mongodb servers

Support

- Forums: http://groups.google.com/group/mongodb-user
- IRC: irc.freenode.net/#mongodb
- Bug DB: jira.mongodb.org
- Commercial support
- Training | Consulting | Online Ed

Meta

- Use Cases | Philosophy | License | Events

Community

- Conferences and Meetups
    - Upcoming:
        - MongoDB Tokyo - Dec 12
- Blog | Twitter | MongoDB Masters | Facebook | LinkedIn | Job Board

Translations

-
-
- Español
- Português
-
-

Please help translate the documentation! Email docs@10gen.com and tell us which language you can translate.

Introduction

> *MongoDB wasn't designed in a lab. We built MongoDB from our own experiences building large scale, high availability, robust systems. We didn't start from scratch, we really tried to figure out what was broken, and tackle that. So the way I think about MongoDB is that if you take MySql, and change the data model from relational to document based, you get a lot of great features: embedded docs for speed, manageability, agile development with schema-less databases, easier horizontal scalability because joins aren't as important. There are lots of things that work great in relational databases: indexes, dynamic queries and updates to name a few, and we haven't changed much there. For example, the way you design your indexes in MongoDB should be exactly the way you do it in MySql or Oracle, you just have the option of indexing an embedded field.*
>
> *– Eliot Horowitz, 10gen CTO and Co-founder*

Why MongoDB?

- **Document-oriented**
    - Documents (objects) map nicely to programming language data types
    - Embedded documents and arrays reduce need for joins
    - Dynamically-typed (schemaless) for easy schema evolution
    - No joins and no multi-document transactions for high performance and easy scalability
- **High performance**
    - No joins and embedding makes reads and writes fast
    - Indexes including indexing of keys from embedded documents and arrays
    - Optional streaming writes (no acknowledgements)
- **High availability**
    - Replicated servers with automatic master failover
- **Easy scalability**
    - Automatic sharding (auto-partitioning of data across servers)
        - Reads and writes are distributed over shards
        - No joins or multi-document transactions make distributed queries easy and fast
    - Eventually-consistent reads can be distributed over replicated servers
- **Rich query language**

Large MongoDB deployment

1. One or more shards, each shard holds a portion of the total data (managed automatically). Reads and writes are automatically routed to the

appropriate shard(s). Each shard is backed by a replica set – which just holds the data for that shard.

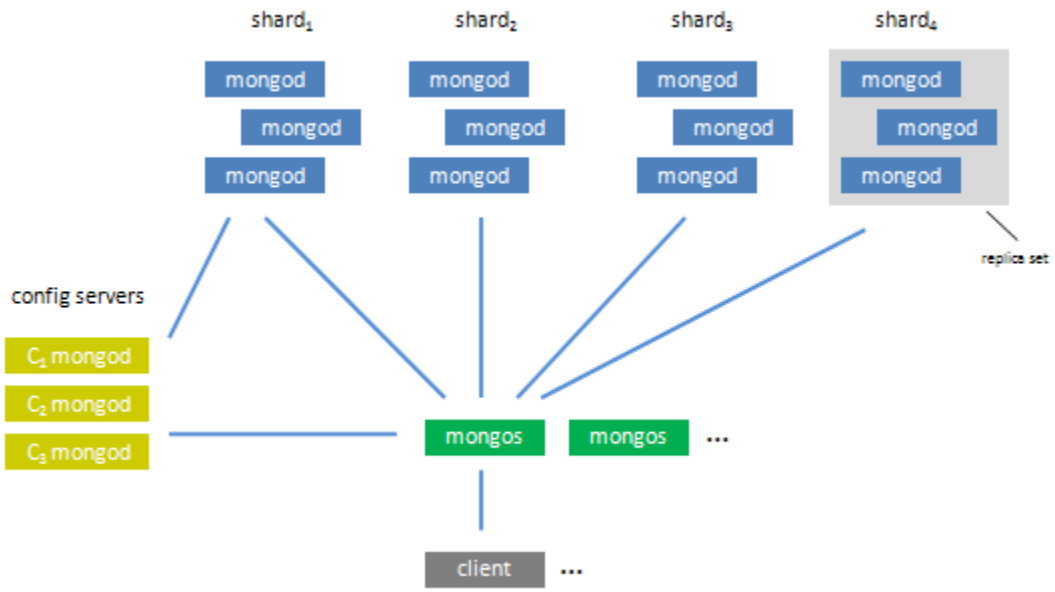
A replica set is one or more servers, each holding copies of the same data. At any given time one is primary and the rest are secondaries. If the primary goes down one of the secondaries takes over automatically as primary. All writes and consistent reads go to the primary, and all eventually consistent reads are distributed amongst all the secondaries.

2. Multiple config servers, each one holds a copy of the meta data indicating which data lives on which shard.

3. One or more routers, each one acts as a server for one or more clients. Clients issue queries/updates to a router and the router routes them to the appropriate shard while consulting the config servers.

4. One or more clients, each one is (part of) the user's application and issues commands to a router via the mongo client library (driver) for its language.

`mongod` is the server program (data or config). **`mongos`** is the router program.



Small deployment (no partitioning)

1. One replica set (automatic failover), or one server with zero or more slaves (no automatic failover).

2. One or more clients issuing commands to the replica set as a whole or the single master (the driver will manage which server in the replica set to send to).

Mongo data model

- A Mongo system (see deployment above) holds a set of databases
- A **database** holds a set of collections
- A **collection** holds a set of documents
- A **document** is a set of fields
- A **field** is a key-value pair
- A **key** is a name (string)
- A **value** is a
    - basic type like string, integer, float, timestamp, binary, etc.,
    - a document, or
    - an array of values

Mongo query language

To retrieve certain documents from a db collection, you supply a query document containing the fields the desired documents should match. For example, `{name: {first: 'John', last: 'Doe'}}` will match all documents in the collection with name of John Doe. Likewise, `{name.last: 'Doe'}` will match all documents with last name of Doe. Also, `{name.last: /^D/}` will match all documents with last name starting with 'D' (regular expression match).

Queries will also match inside embedded arrays. For example, {keywords: 'storage'} will match all documents with 'storage' in its keywords array. Likewise, {keywords: {$in: ['storage', 'DBMS']}} will match all documents with 'storage' or 'DBMS' in its keywords array.

If you have lots of documents in a collection and you want to make a query fast then build an index for that query. For example, ensureIndex({name.last: 1}) or ensureIndex({keywords: 1}). Note, indexes occupy space and slow down updates a bit, so use them only when the tradeoff is worth it.

See also:

- Philosophy

Downloads

	OS X 32-bit note	OS X 64-bit	Linux 32-bit note	Linux 64-bit	Windows 32-bit note	Windows 64-bit	Solaris i86pc note	Solaris 64	Source
Production Release (Recommended)									
1.8.1 4/6/2011 Changelog Release Notes	download	download	download *legacy-static	download *legacy-static	download	download	download	download	tgz zip
Nightly Changelog	download	download	download *legacy-static	download *legacy-static	download	download	download	download	tgz zip
Previous Release									
1.6.5 12/9/2010 Changelog Release Notes	download	download	download *legacy-static	download *legacy-static	download	download	download	download	tgz zip
Nightly Changelog	download	download	download *legacy-static	download *legacy-static	download	download	download	download	tgz zip
Development Release (Unstable)									
Nightly Changelog	download	download	download *legacy-static	download *legacy-static	download	download	download	download	tgz zip
Archived Releases									
	list	list	list	list	list	list	list	list	list

2.2 Release Notes

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/release-notes/2.2/.

2.0 Release Notes

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/release-notes/2.0/.

1.8 Release Notes

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/release-notes/1.8/.

Upgrading to 1.8

- Upgrading Replica Sets
- Upgrading Sharded Clusters

- Returning to 1.6
  - Journaling
- See Also
- Download

First, upgrade your shell (`mongo`) to the 1.8.x shell.

Upgrading Replica Sets

1.8.x secondaries **can** replicate from 1.6.x primaries.

1.6.x secondaries **cannot** replicate from 1.8.x primaries.

Thus, the trick is to replace all of your secondaries, then the primary.

For example, suppose we have a typical replica set with 1 primary, 1 secondary, and 1 arbiter. To upgrade this set, do the following:

1. For each arbiter:
   - Shut down the arbiter
   - Start it back up with the 1.8 binary
2. Change your config (optional)
   It is possible that, when you start shutting down members of the set, a new primary will be elected. If you wish to prevent this, you can give all of the slaves a priority of 0 before upgrading, then change them back afterwards.
   - Record your current config. Run rs.conf() and paste the results into a text file.
   - Update your config so that all secondaries have priority 0. For example:

```
> config = rs.conf()
{
     "_id" : "foo",
     "version" : 3,
     "members" : [
             {
                     "_id" : 0,
                     "host" : "ubuntu:27017"
             },
             {
                     "_id" : 1,
                     "host" : "ubuntu:27018"
             },
             {
                     "_id" : 2,
                     "host" : "ubuntu:27019",
                     "arbiterOnly" : true
             }
             {
                     "_id" : 3,
                     "host" : "ubuntu:27020"
             },
             {
                     "_id" : 4,
                     "host" : "ubuntu:27021"
             },
     ]
}
> config.version++
3
> rs.isMaster()
{
     "setName" : "foo",
     "ismaster" : false,
     "secondary" : true,
     "hosts" : [
             "ubuntu:27017",
             "ubuntu:27018"
     ],
     "arbiters" : [
             "ubuntu:27019"
     ],
     "primary" : "ubuntu:27018",
     "ok" : 1
}
> // for each slave
> config.members[0].priority = 0
> config.members[3].priority = 0
> config.members[4].priority = 0
> rs.reconfig(config)
```

3. For each slave:
    - Shut down the slave
    - Start it back up with the 1.8 binary
4. If you changed the config, change it back to its original state

```
> config = rs.conf()
> config.version++
> config.members[0].priority = 1
> config.members[3].priority = 1
> config.members[4].priority = 1
> rs.reconfig(config)
```

5. Shut down the primary (the final 1.6 server) and restart it with the 1.8 binary.

Upgrading Sharded Clusters

1. Turn off the balancer:

```
$ mongo <a_mongos_hostname>
> use config
> db.settings.update({_id:"balancer"},{$set : {stopped:true}}, true)
```

2. For each shard:
   - If the shard is a replica set, follow the directions above for replica sets.
   - If the shard is a single mongod process, shut it down and start it back up with the 1.8 binary.
3. For each `mongos`:
   - Shut down the mongos process
   - Restart with the 1.8 binary
4. For each config server:
   - Shut down the config server process
   - Restart with the 1.8 binary
5. Turn on the balancer

```
> use config
> db.settings.update({_id:"balancer"},{$set : {stopped:false}})
```

Returning to 1.6

If something goes wrong and you wish to move back to 1.6, follow the steps above in reverse. Please be careful that you have not inserted any documents larger than 4MB while running on 1.8 (where the max size has increased to 16MB); if you have you will get errors when the server tries to read those documents.

Journaling

Returning to 1.6 after using 1.8 journaling works fine, as journaling does not change anything about the data file format. Suppose you are running 1.8.0 with journaling enabled and something isn't working for you, so you decide to switch back to 1.6. There are two scenarios:

1. If you shut down cleanly with 1.8.x, just restart with the 1.6 mongod binary.
2. If 1.8.x shut down uncleanly, start 1.8.x up again and let the journal files run to fix any damage (incomplete writes) that may have existed at the crash. Then shut down 1.8.0 cleanly and restart with the 1.6 mongod binary.

See Also

- 1.8 Release Notes page for details on changes in v1.8 to map/reduce.

Download

- Download v1.8

1.6 Release Notes

MongoDB 1.6 is a drop-in replacement for 1.4. To upgrade, simply shutdown `mongod` then restart with the new binaries.*

> ⚠ MongoDB v1.8 is now available.

** Please note that you should upgrade to the latest version of whichever driver you're using. Certain drivers, including the Ruby driver, will require the upgrade, and all the drivers will provide extra features for connecting to replica sets.*

Sharding

Sharding is now production-ready, making MongoDB horizontally scalable, with no single point of failure. A single instance of *mongod* can now be upgraded to a distributed cluster with zero downtime when the need arises.

- Sharding Tutorial
- Sharding Documentation
- Upgrading a Single Server to a Cluster

Replica Sets

Replica sets, which provide automated failover among a cluster of *n* nodes, are also now available.

Plese note that replica *pairs* are now deprecated; we strongly recommend that replica pair users upgrade to replica sets.

- Replica Set Tutorial
- Replica Set Documentation
- Upgrading Existing Setups to Replica Sets

Other Improvements

- The w option (and wtimeout) forces writes to be propagated to *n* servers before returning success (this works especially well with replica sets)
- $or queries
- Improved concurrency
- $slice operator for returning subsets of arrays
- 64 indexes per collection (formerly 40 indexes per collection)
- 64-bit integers can now be represented in the shell using NumberLong
- The findAndModify command now supports upserts. It also allows you to specify fields to return
- $showDiskLoc option to see disk location of a document
- Support for IPv6 and UNIX domain sockets

Installation

- Windows service improvements
- The C++ client is a separate tarball from the binaries

1.6.x Release Notes

- 1.6.5

1.5.x Release Notes

- 1.5.8
- 1.5.7
- 1.5.6
- 1.5.5
- 1.5.4
- 1.5.3
- 1.5.2
- 1.5.1
- 1.5.0

You can see a full list of all changes on Jira.

Thank you everyone for your support and suggestions!

1.4 Release Notes

We're pleased to announce the 1.4 release of MongoDB. 1.4 is a drop in replacement for 1.2. To upgrade you just need to shutdown mongod, then restart with the new binaries. (Users upgrading from release 1.0 should review the 1.2 release notes, in particular the instructions for upgrading the DB format.)

Release 1.4 includes the following improvements over release 1.2:

Core server enhancements

- concurrency improvements
- indexing memory improvements
- background index creation
- better detection of regular expressions so the index can be used in more cases

Replication & Sharding

- better handling for restarting slaves offline for a while
- fast new slaves from snapshots (--fastsync)
- configurable slave delay (--slavedelay)

- replication handles clock skew on master
- $inc replication fixes
- sharding alpha 3 - notably 2 phase commit on config servers

Deployment & production

- configure "slow threshold" for profiling
- ability to do fsync + lock for backing up raw files
- option for separate directory per db (--directoryperdb)
- `http://localhost:28017/_status` to get serverStatus via http
- REST interface is off by default for security (--rest to enable)
- can rotate logs with a db command, logRotate
- enhancements to serverStatus command (db.serverStatus()) - counters and replication lag stats
- new mongostat tool

Query language improvements

- $all with regex
- $not
- partial matching of array elements $elemMatch
- $ operator for updating arrays
- $addToSet
- $unset
- $pull supports object matching
- $set with array indices

Geo

- 2d geospatial search
- geo $center and $box searches

1.2.x Release Notes

New Features

- More indexes per collection
- Faster index creation
- Map/Reduce
- Stored JavaScript functions
- Configurable fsync time
- Several small features and fixes

DB Upgrade Required

There are some changes that will require doing an upgrade if your previous version is <= 1.0.x. If you're already using a version >= 1.1.x then these changes aren't required. There are 2 ways to do it:

- --upgrade
  - stop your mongod process
  - run ./mongod --upgrade
  - start mongod again
- use a slave
  - start a slave on a different port and data directory
  - when its synced, shut down the master, and start the new slave on the regular port.

Ask in the forums or IRC for more help.

Replication Changes

- There have been minor changes in replication. If you are upgrading a master/slave setup from <= 1.1.2 you have to update the slave first.

mongoimport

- mongoimportjson has been removed and is replaced with mongoimport that can do json/csv/tsv

field filter changing

- We've changed the semantics of the field filter a little bit. Previously only objects with those fields would be returned. Now the field filter

only changes the output, not which objects are returned. If you need that behavior, you can use $exists

other notes

http://www.mongodb.org/display/DOCS/1.1+Development+Cycle

1.0 Changelist

Wrote MongoDB. See documentation.

Version Numbers

MongoDB uses the odd-numbered versions for development releases.

There are 3 numbers in a MongoDB version: *A.B.C*

- *A* is the major version. This will rarely change and signify very large changes
- *B* is the release number. This will include many changes including features and things that possible break backwards compatibility. Even Bs will be stable branches, and odd Bs will be development.
- *C* is the revision number and will be used for bugs and security issues.

For example:

- 1.0.0 : first GA release
- 1.0.x : bug fixes to 1.0.x - highly recommended to upgrade, very little risk
- 1.1.x : development release. this will include new features that are not fully finished, and works in progress. Some things may be different than 1.0
- 1.2.x : second GA release. this will be the culmination of the 1.1.x release.

What's New by Version

This is a summary of high level features only. See jira and release notes for full details.

- 1.4
  - Geospatial
  - Background indexing
  - --directoryperdb
  - Log rotate
  - $not
  - $ operator for updating arrays
  - $addToSet
  - $unset
- 1.6
  - Sharding
  - Replica Sets
  - getLastError w param
  - $or
  - $slice
  - 64 indexes per collection
  - IPv6
- 1.8
  - Journaling
  - Sparse and covered indexes
  - $rename
  - mongos (i.e., sharded environment) will route SLAVE_OK queries to secondaries in replica sets

Drivers

MongoDB has client support for most programming languages. Download the driver for the language you are coding with for your application.

News: Introducing MongoClient

mongodb.org Supported

- C
- C++
- Erlang
- Haskell
- Java (and other JVM languages)
- Javascript
- .NET (C#, F#, PowerShell, ...)
- Node.js
- Perl
- PHP
- Python
- Ruby
- Scala

Community Supported

- ActionScript3
  - http://code.google.com/p/jmcnet-full-mongo-flex-driver/
  - http://www.mongoas3.com
- C
  - libmongo-client
- C# and .NET
- Clojure
  - See the Java Language Center
- ColdFusion
  - cfmongodb
  - Blog post: Part 1 | Part 2 | Part 3
  - http://github.com/virtix/cfmongodb/tree/0.9
  - http://mongocfc.riaforge.org/
- D
  - Port of the MongoDB C Driver for D
- Dart
  - https://bitbucket.org/vadimtsushko/mongo-dart
- Delphi
  - mongo-delphi-driver - Full featured Deplhi interface to MongoDB built on top of the mongodb.org supported C driver
  - pebongo - Early stage Delphi driver for MongoDB
  - TMongoWire - Maps all the VarTypes of OleVariant to the BSON types, implements IPersistStream for (de)serialization, and uses TTcpClient for networking
- Entity
  - entity driver for mongodb on Google Code, included within the standard Entity Library
- Erlang
  - emongo - An Erlang MongoDB driver that emphasizes speed and stability. "The most emo of drivers."
  - Erlmongo - an almost complete MongoDB driver implementation in Erlang
- Factor
  - http://github.com/slavapestov/factor/tree/master/extra/mongodb/
- Fantom
  - http://bitbucket.org/liamstask/fantomongo/wiki/Home
- F#
  - http://gist.github.com/218388
- Go
  - gomongo
  - go-mongo
  - mgo
  - mongogo
- Groovy
  - gmongo
  - Also see the Java Language Center
    **Blog Post: Groovy on Grails in the land of MongoDB
- Javascript
- Lisp
  - https://github.com/fons/cl-mongo
- Lua
  - LuaMongo on Google Code
  - LuaMongo fork on Github
- MatLab
  - mongo-matlab-driver
- node.js
- Objective C
  - NuMongoDB
- Opa
  - Opa Standard Library MongoDB Driver
- PHP
  - Asynchronous PHP driver using libevent
- PowerShell

    - mosh Powershell provider for MongoDB
    - mdbc module cmdlets using official 10gen driver
    - Doug Finke's blog post on using the original community C# driver with PowerShell
- Prolog
    - https://github.com/khueue/prolongo
- Python
    - MongoEngine
    - MongoKit
    - Django-nonrel
    - Django-mongodb
    - Django-mongonaut
- R
    - rmongodb - Full featured R interface to MongoDB built on top of the mongodb.org supported C driver
    - RMongo - R client to interface with MongoDB
- REST
- Ruby
    - MongoMapper
    - rmongo - An event-machine-based Ruby driver for MongoDB
    - jmongo A thin ruby wrapper around the mongo-java-driver for vastly better jruby performance.
    - em-mongo EventMachine MongoDB Driver (based off of RMongo).
- Scala
    - See the Java Language Center
- Racket (PLT Scheme)
    - http://planet.plt-scheme.org/display.ss?package=mongodb.plt&owner=jaymccarthy
    - docs
- Smalltalk
    - Squeaksource Mongotalk
    - Dolphin Smalltalk

Get Involved, Write a Driver!

- Writing Drivers and Tools

Hadoop

The MongoDB Hadoop Adapter is a plugin for Hadoop that provides Hadoop the ability to use MongoDB as an input source and/or an output source.

The source code is available on github where you can find a more comprehensive readme

If you have questions please email the mongodb-user Mailing List. For any issues please file a ticket in Jira.

Installation

The Mongo-Hadoop adapter uses the
SBT Build Tool tool for
compilation. SBT provides superior support for discreet configurations
targeting multiple Hadoop versions. The distribution includes
self-bootstrapping copy of SBT in the distribution as `sbt`. Create a
copy of the jar files using the following command:

```
./sbt package
```

The MongoDB Hadoop Adapter supports a number of Hadoop releases. You
can change the Hadoop version supported by the build by modifying the
value of `hadoopRelease` in the `build.sbt` file. For instance, set
this value to:

```
hadoopRelease in ThisBuild := "cdh3"
```

Configures a build against Cloudera CDH3u3, while:

```
hadoopRelease in ThisBuild := "0.21"
```

Configures a build against Hadoop 0.21 from the mainline Apache distribution.

After building, you will need to place the "core" jar and the

"mongo-java-driver" in the `lib` directory of each Hadoop server.

For more complete install instructions please see the install instructions in the readme

Presentations

- MongoDB, Hadoop and HuMONGOus Data by Steve Francia at MongoSF 2012
- MongoDB + Hadoop by Brendan McAdams at MongoDB Philly 2012
- mongo-hadoop (BigData Analysis with Mongo-Hadoop) by Daichi Morifuji at MongoTokyo 2012

Blog Posts

- http://blog.10gen.com/post/20840407875/mongodb-hadoop-connector-announced

Scala Language Center

Casbah Casbah is the officially supported Scala driver for MongoDB. It provides wrappers and extensions to the Java driver meant to allow a more Scala-friendly interface to MongoDB. It supports serialization/deserialization of common Scala types (including collections and regex), Scala 2.8 collection versions of DBObject and DBList and a fluid query DSL.

- API documentation
- Tutorial
- Mailing List
- Java Driver Doc Page

Community

- Lift-MongoDB - Lift Web Framework supports MongoDB, including object mapping via the Record back-end implementation.
- Rogue: A Type-Safe Scala DSL - Foursquare's DSL for querying MongoDB alongside Lift-MongoDB-Record.
    - Tutorial/Intro
    - Source/Downloads
- Blue Eyes is a lightweight framework for building REST APIs with strong MongoDB integration including a DSL and Mock MongoDB for testing.
- mongo-scala-driver is a thin wrapper around mongo-java-driver to make working with MongoDB more Scala-like.
    - Wiki
    - Mailing list
- Reactive-Mongo: a reactive driver that allows you to design very scalable applications unleashing MongoDB capabilities like streaming infinite live collections and files for modern Realtime Web applications.
    - Repo
    - Blog post
- Hammersmith is a Scala-based, asynchronous Netty driver for MongoDB with type-class based fast custom object encoding.

Webcast - MongoDB and Scala

Haskell Language Center

The Haskell driver and its API documentation reside on Hackage

C Language Center

C Driver

The MongoDB C Driver is the 10gen-supported driver for MongoDB. It's written in pure C.

- MongoDB C Driver Documentation - Table of Contents / Index
    - Tutorial
    - Building the MongoDB C Driver
    - BSON - How to work with BSON objects
    - BCON - How to use BCON for concise, readable, and maintainable definition of BSON documents
    - API Docs for current version - or index page for all versions
- C Driver README
- History
- JIRA
- Source Code

Download and build

The C driver is hosted at GitHub. You can download the latest stable version: v0.7.

Then consult the building docs for detailed instructions on building the driver.

CSharp Language Center

MongoDB C# / .NET Driver

The MongoDB C# Driver is the 10gen-supported C# / .NET driver for MongoDB.

- C# Driver Quick-Start
- C# Driver Tutorial
- C# Driver LINQ Tutorial
- C# Driver Serialization Tutorial
- API Documentation
- C# Driver README
- Source Code

Several other C# drivers have been developed by the community. This is the "official" C# Driver supported by 10gen. It is similar in many ways to the various drivers that came before it, but it is **not** a direct replacement for any of them. Most people have found it easy to convert to using this driver, but you should expect to have to make some changes.

Downloading the Driver

The C# Driver is hosted at github.com. Instructions for downloading the source code are at: Download Instructions

You can also download binary builds in either .msi or .zip formats from:
http://github.com/mongodb/mongo-csharp-driver/downloads.

Note: if you download the .zip file Windows might require you to "Unblock" the help file. If Windows asks "Do you want to open this file?" when you double click on the CSharpDriverDocs.chm file, clear the check box next to "Always ask before opening this file" before pressing the Open button. Alternatively, you can right click on the CSharpDriverDocs.chm file and select Properties, and then press the Unblock button at the bottom of the General tab. If the Unblock button is not present then the help file does not need to be unblocked.

Visual Studio Versions Supported

The current version of the C# Driver has been built and tested using

- Visual Studio 2010
- Visual Studio 2008

Questions and Bug Reports

Questions about the C# driver (or any other MongoDB topic) can be posted at the mongodb-user Google Group:

https://groups.google.com/group/mongodb-user

Bug reports can be created in JIRA at:

https://jira.mongodb.org/browse/CSHARP

See Also

- CSharp Community Projects

Presentations

- LINQ support in C#/.NET driver (slide deck) (Aug 2012)
- What's new in the .NET driver (slide deck) (Jul 2012)
- C# Development (conference videos)

CSharp Community Projects

Community Supported C# Drivers

- See also: the 10gen supported MongoDB C# driver

- mongodb-csharp driver
- simple-mongodb driver
- NoRM

Tools

- MongoDB.Emitter Document Wrapper
- log4net appender
- ASP.NET Membership and Role Providers for MongoDB
- ASP.NET User Administration
- MongoCola Administration Tool
- MongoRepository

F#

- F# Example

Community Articles

- Experimenting with MongoDB from C#
- Using MongoDB from C#
- Introduction to MongoDB for .NET
- Using Json.NET and Castle Dynamic Proxy with MongoDB
- Implementing a Blog Using ASP.NET MVC and MongoDB
- Intro Article using a Post and Comments Example
- Using the 10gen .NET driver from PowerShell
- Tutorial MongoDB con ASP.NET MVC - Ejemplo Práctico

Support

- http://groups.google.com/group/mongodb-csharp
- http://groups.google.com/group/mongodb-user
- IRC: #mongodb on freenode

See Also

- C++ Language Center

CSharp Driver LINQ Tutorial

- Introduction
- Quickstart
- Supported LINQ query operators
    - Any
    - Any (with predicate)
    - Count
    - Count (with predicate)
    - Distinct
    - ElementAt
    - ElementAtOrDefault
    - First
    - First (with predicate)
    - FirstOrDefault
    - FirstOrDefault (with predicate)
    - Last
    - Last (with predicate)
    - LastOrDefault
    - LastOrDefault (with predicate)
    - LongCount
    - LongCount (with predicate)
    - Max
    - Max (with selector)
    - Min
    - Min (with selector)
    - OrderBy
    - OrderByDescending

        - Select
        - Single
        - Single (with predicate)
        - SingleOrDefault
        - SingleOrDefault (with predicate)
        - Skip
        - Take
        - ThenBy
        - ThenByDescending
        - Where
- Supported where clauses
    - And (&& operator)
    - Any
    - Boolean constant
    - Boolean field or property
    - Contains (Enumerable method)
    - Contains (string method)
    - ContainsAll (LINQ to MongoDB extension method)
    - ContainsAny (LINQ to MongoDB extension method)
    - Count method (array length)
    - Count property (array length)
    - EndsWith (string method)
    - Enum comparisons (==, !=, <, <=, >, >=)
    - In (LINQ to MongoDB extension method)
    - Inject
    - IsMatch (regular expression method)
    - Length (array length)
    - Mod (% operator)
    - Not (! operator)
    - Numeric comparisons (==, !=, <, <=, >, >=)
    - Or (|| operator)
    - StartsWith (string method)

Introduction

This tutorial covers the support for LINQ queries added in the 1.4 release of the C# driver.

You should already have read at least the quickstart introduction to the C# driver.

http://www.mongodb.org/display/DOCS/CSharp+Driver+Quickstart

Quickstart

First, add the following additional using statement to your program:

```
using MongoDB.Driver.Linq;
```

Then, get a reference to a collection variable in the usual way:

```
var collection = database.GetCollection<TDocument>("collectionname");
```

The basic idea behind writing a LINQ query is to start from a collection variable and begin the LINQ query by calling the AsQueryable<TDocument>() method. After that it's all standard LINQ.

For example:

```
var query =
    from e in collection.AsQueryable<Employee>()
    where e.FirstName == "John"
    select e;

foreach (var employee in query)
{
    // process employees named "John"
}
```

You can also write queries using lambda syntax. The previous query would be written using lambda syntax like this:

```
var query =
    collection.AsQueryable<Employee>()
    .Where(e => e.FirstName == "John")
    .Select(e => e); // this trivial projection is optional when using lambda syntax
```

The C# compiler translates all queries written using query syntax into lambda syntax internally anyway, so there is no performance advantage or penalty to choosing either style. You can also mix and match the styles, which can be useful when using query operators that are not supported by the query syntax.

All the code samples in this tutorial show both the query syntax and the lambda syntax for each query operator and supported where clauses.

Only LINQ queries that can be translated to an equivalent MongoDB query are supported. If you write a LINQ query that can't be translated you will get a runtime exception and the error message will indicate which part of the query wasn't supported.

Note: The 1.4 version of the C# driver requires that all where clauses that compare a field or property against a value have the constant on the right hand side. This restriction will be lifted in the next release.

Supported LINQ query operators

This section documents the supported LINQ query operators.

Any

Without a predicate Any just tests whether the collection has any documents.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c)
    .Any();
// or
var result =
    collection.AsQueryable<C>()
    .Any();
```

Any (with predicate)

With a predicate Any tests whether the collection has any matching documents.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c)
    .Any(c => c.X == 1);
// or
var result =
    collection.AsQueryable<C>()
    .Any(c => c.X == 1);
```

Note that the predicate can be provided either by a where clause or as an argument to Any, so the following are equivalent to the previous query.

```
var result =
    (from c in collection.AsQueryable<C>()
    where c.X == 1
    select c)
    .Any();
// or
var result =
    collection.AsQueryable<C>()
    .Where(c => c.X == 1)
    .Any();
```

Any with a predicate is not supported after a projection (at least not yet). So the following is not valid:

```
var result =
    collection.AsQueryable<C>()
    .Select(c => c.X)
    .Any(x => x == 1);
```

You can usually rewrite such a query by putting an equivalent where clause before the projection (in which case you can drop the projection).

Count

Without a predicate Count just returns the number of documents in the collection.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c)
    .Count();
// or
var result =
    collection.AsQueryable<C>()
    .Count();
```

Count (with predicate)

With a predicate Count returns the number of documents that match the predicate.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c)
    .Count(c => c.X == 1);
// or
var result =
    collection.AsQueryable<C>()
    .Count(c => c.X == 1);
```

Note that the predicate can be provided either by a where clause or as an argument to Count , so the following are equivalent to the previous query.

```
var result =
    (from c in collection.AsQueryable<C>()
    where c.X == 1
    select c)
    .Count();
// or
var result =
    collection.AsQueryable<C>()
    .Where(c => c.X == 1)
    .Count();
```

Count with a predicate is not supported after a projection (at least not yet). So the following is not valid:

```
var result =
    collection.AsQueryable<C>()
    .Select(c => c.X)
    .Count(x => x == 1);
```

You can usually rewrite such a query by putting an equivalent where clause before the projection (in which case you can drop the projection).

Distinct

Distinct returns the unique values of a field or property of the documents in the collection. You use a projection to identify the field or property whose distinct values you want.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c.X)
    .Distinct();
// or
var result =
    collection.AsQueryable<C>()
    .Select(c => c.X)
    .Distinct();
```

The projection must select a particular field or property of the document. If the value of that field or property is represented in MongoDB as an array you can also use array indexing to select an item from the array.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c.A[i])
    .Distinct();
// or
var result =
    collection.AsQueryable<C>()
    .Select(c => c.A[i])
    .Distinct();
```

ElementAt

ElementAt returns a particular document from a result set. Often you will combine this with a sort order.

```
var result =
    (from c in collection.AsQueryable<C>()
    where c.X > 0
    orderby c.X
    select c)
    .ElementAt(index);
// or
var result =
    collection.AsQueryable<C>()
    .Where(c => c.X > 0)
    .OrderBy(c => c.X)
    .ElementAt(index);
```

If the result set has fewer documents than index ElementAt throws an exception.

ElementAtOrDefault

ElementAtOrDefault is just like ElementAt except that if there are fewer documents than index it returns null instead of throwing an exception.

First

First returns the first document from a result set. Often you will combine this with a sort order.

```
var result =
    (from c in collection.AsQueryable<C>()
    where c.X > 0
    orderby c.X
    select c)
    .First();
// or
var result =
    collection.AsQueryable<C>()
    .Where(c => c.X > 0)
    .OrderBy(c => c.X)
    .First();
```

If the result set has no documents First throws an exception.

First (with predicate)

This overload of First allows you to provide a predicate as an argument to First. This is an alternative to using a where clause.

```
var result =
    (from c in collection.AsQueryable<C>()
    orderby c.X
    select c)
    .First(c => c.X > 0);
// or
var result =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X)
    .First(c => c.X > 0);
```

First with a predicate is not supported after a projection (at least not yet). So the following is not valid:

```
var result =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X)
    .Select(c => c.X)
    .Count(x => x > 0);
```

You can usually rewrite such a query by putting an equivalent where clause before the projection.

If the result set has no documents First with a predicate throws an exception.

FirstOrDefault

FirstOrDefault is just like First except that if there are no matching documents it returns null instead of throwing an exception.

FirstOrDefault (with predicate)

FirstOrDefault with a predicate is just like First with a predicate except that if there are no matching documents it returns null instead of throwing an exception.

Last

Last returns the last document from a result set. Often you will combine this with a sort order.

```
var result =
    (from c in collection.AsQueryable<C>()
    where c.X > 0
    orderby c.X
    select c)
    .Last();
// or
var result =
    collection.AsQueryable<C>()
    .Where(c => c.X > 0)
    .OrderBy(c => c.X)
    .Last();
```

If the result set has no documents Last throws an exception.

Last (with predicate)

This overload of Last allows you to provide a predicate as an argument to Last. This is an alternative to using a where clause.

```
var result =
    (from c in collection.AsQueryable<C>()
    orderby c.X
    select c)
    .Last(c => c.X > 0);
// or
var result =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X)
    .Last(c => c.X > 0);
```

Last with a predicate is not supported after a projection (at least not yet). So the following is not valid:

```
var result =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X)
    .Select(c => c.X)
    .Last(x => x > 0);
```

You can usually rewrite such a query by putting an equivalent where clause before the projection.

If the result set has no documents Last throws an exception.

LastOrDefault

LastOrDefault is just like Last except that if there are no matching documents it returns null instead of throwing an exception.

LastOrDefault (with predicate)

LastOrDefault with a predicate is just like Last with a predicate except that if there are no matching documents it returns null instead of throwing an exception.

LongCount

LongCount is just like Count except that the return value is a 64-bit integer instead of a 32-bit integer.

LongCount (with predicate)

LongCount with a predicate is just like Count with a predicate except that the return value is a 64-bit integer instead of a 32-bit integer.

Max

Max returns the maximum value of a field or property of the documents in the collection. You use a projection to identify the field or property whose maximum value you want.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c.X)
    .Max();
// or
var result =
    collection.AsQueryable<C>()
    .Select(c => c.X)
    .Max();
```

The projection must select a particular field or property of the document. If the value of that field or property is represented in MongoDB as an array you can also use array indexing to select an item from the array.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c.A[i])
    .Max();
// or
var result =
    collection.AsQueryable<C>()
    .Select(c => c.A[i])
    .Max();
```

Max (with selector)

This overload of Max lets you select the field or property whose maximum value you want as an argument to Max instead of to Select.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c)
    .Max(c => c.X);
// or
var result =
    collection.AsQueryable<C>()
    .Max(c => c.X);
```

Min

Min returns the minimum value of a field or property of the documents in the collection. You use a projection to identify the field or property whose minimum value you want.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c.X)
    .Min();
// or
var result =
    collection.AsQueryable<C>()
    .Select(c => c.X)
    .Min();
```

The projection must select a particular field or property of the document. If the value of that field or property is represented in MongoDB as an array you can also use array indexing to select an item from the array.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c.A[i])
    .Min();
// or
var result =
    collection.AsQueryable<C>()
    .Select(c => c.A[i])
    .Min();
```

Min (with selector)

This overload of Min lets you select the field or property whose minimum value you want as an argument to Min instead of to Select.

```
var result =
    (from c in collection.AsQueryable<C>()
    select c)
    .Min(c => c.X);
// or
var result =
    collection.AsQueryable<C>()
    .Min(c => c.X);
```

OrderBy

OrderBy is used to specify an ascending sort order for the result set.

```
var query =
    from c in collection.AsQueryable<C>()
    orderby c.X
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X);
```

OrderByDescending

OrderBy is used to specify a descending sort order for the result set.

```
var query =
    from c in collection.AsQueryable<C>()
    orderby c.X descending
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .OrderByDescending(c => c.X);
```

Select

Select is used to project a new result type from the matching documents. In the 1.4 version of the C# driver a projection must typically be the last operation (with a few exceptions like Distinct, Max and Min).

```
var query =
    from c in collection.AsQueryable<C>()
    select new { c.X, c.Y };
// or
var query =
    collection.AsQueryable<C>()
    .Select(c => new { c.X, c.Y });
```

Single

Single returns the first and only document from a result set.

```
var result =
    (from c in collection.AsQueryable<C>()
    where c.X > 0
    orderby c.X
    select c)
    .Single();
// or
var result =
    collection.AsQueryable<C>()
    .Where(c => c.X > 0)
    .OrderBy(c => c.X)
    .Single();
```

If the result set has no documents or multiple documents Single throws an exception.

Single (with predicate)

This overload of Single allows you to provide a predicate as an argument to Single . This is an alternative to using a where clause.

```
var result =
    (from c in collection.AsQueryable<C>()
    orderby c.X
    select c)
    .Single(c => c.X > 0);
// or
var result =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X)
    .Single(c => c.X > 0);
```

Single with a predicate is not supported after a projection (at least not yet). So the following is not valid:

```
var result =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X)
    .Select(c => c.X)
    .Single(x => x > 0);
```

You can usually rewrite such a query by putting an equivalent where clause before the projection.

If the result set has no documents or multiple documents Single throws an exception.

SingleOrDefault

SingleOrDefault is just like Single except that if there are no matching documents it returns null instead of throwing an exception.

SingleOrDefault (with predicate)

SingleOrDefault with a predicate is just like Single with a predicate except that if there are no matching documents it returns null instead of throwing an exception.

Skip

Use Skip to specify how many documents to skip from the beginning of the result set. Often you will combine Skip with a sort order.

```
var query =
    (from c in collection.AsQueryable<C>()
    orderby c.X
    select c)
    .Skip(100);
// or
var query =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X)
    .Skip(100);
```

Take

Use Take to specify how many documents to return from the server. When combining Take with Skip often you will also specify a sort order.

```
var query =
    (from c in collection.AsQueryable<C>()
    orderby c.X
    select c)
    .Skip(100)
    .Take(100);
// or
var query =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X)
    .Skip(100)
    .Take(100);
```

ThenBy

ThenBy is used to specify an additional ascending sort order for the result set.

```
var query =
    from c in collection.AsQueryable<C>()
    orderby c.X, c.Y
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X)
    .ThenBy(c => c.Y);
```

ThenByDescending

ThenBy is used to specify an additional descending sort order for the result set.

```
var query =
    from c in collection.AsQueryable<C>()
    orderby c.X, c.Y descending
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .OrderBy(c => c.X)
    .ThenByDescending(c => c.Y);
```

Where

A where clause is used to specify which documents the query should return. A where clause is a C# expression that maps the query document type to a boolean value. If the expression returns true the document "matches" the query and is included in the result set.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.X > 0
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.X > 0);
```

Sometimes a predicate can be supplied in other places besides a where clause, and it is also possible to have multiple where clauses. When multiple predicates are involved they are combined into a single composite predicate by combining the individual predicates with the && operator.

For example, the following queries are equivalent:

```
var query =
    (from c in collection.AsQueryable<C>()
    where c.X > 0
    where c.Y > 0)
    .First(c.Z > 0);
// or
var query =
    (from c in collection.AsQueryable<C>()
    where c.X > 0 && c.Y > 0 && c.Z > 0)
    .First();
```

Supported where clauses

This section dcouments the supported where clauses.

As mentioned earlier, not all C# expressions are supported as a where clause. You can use this documentation as a guide to what is supported, or you can just try an expression and see if it works (a runtime exception is thrown if the where clause is not supported).

Where clauses are typically introduced using the Where query operator, but the same expressions are supported wherever a predicate is called for. In some cases multiple where clauses and predicates will be combined, in which case they are combined with the && operator.

Note: The 1.4 version of the C# driver requires that all where clauses that compare a field or property against a value have the constant on the right hand side. This restriction will be lifted in the next release.

And (&& operator)

Sub-expressions can be combined with the && operator to test whether all of them are true.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.X > 0 && c.Y > 0
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.X > 0 && c.Y > 0);
```

This is translated to the following MongoDB query:

```
{ X : { $gt : 0 }, Y : { $gt : 0 } }
```

In some cases the And query can't be flattened as shown, and the $and operator will be used. The following example matches documents where X is both a multiple of 2 and a multiple of 3:

```
var query =
    from c in collection.AsQueryable<C>()
    where (c.X % 2 == 0) && (c.X % 3 == 0)
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => (c.X % 2 == 0) && (c.X % 3 == 0));
```

This is translated to the following MongoDB query using $and:

```
{ $and : [{ X : { $mod : [2, 0] } }, { X : { $mod : [3, 0] } }] }
```

Any

This method is used to test whether an array field or property contains any items.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.A.Any()
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.A.Any());
```

matches any document where A has 1 or more items.

This is translated to the following MongoDB query:

```
{ A : { $ne : null, $not : { $size : 0 } } }
```

Boolean constant

This form is mostly for completeness. You will probably use it rarely. It allows a boolean constant to be used to either match or not match the document.

```
var query =
    from c in collection.AsQueryable<C>()
    where true
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => true);
```

This is translated to the following MongoDB query:

```
{ _id : { $exists : true } }
```

Which matches all documents since the _id is a mandatory field.

Boolean field or property

A boolean field or property of the document doesn't have to be compared to true, it can just be mentioned in the where clause and there is an implied comparison to true.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.B
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.B);
```

This is translated to the following MongoDB query:

```
{ B : true }
```

Contains (Enumerable method)

This method is used to test whether an array (or array-like) field or property contains a particular value:

```
var query =
    from c in collection.AsQueryable<C>()
    where c.A.Contains(123)
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.A.Contains(123));
```

This is transalated to the following MongoDB query:

```
{ A : 123 }
```

This translation relies on the way array fields are treated by the MongoDB query language.

Contains (string method)

This method is used to test whether a string field or property of the document contains a particular substring.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.S.Contains("abc")
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.S.Contains("abc"));
```

This is translated to the following MongoDB query (using regular expressions):

```
{ S : /abc/ }
```

ContainsAll (LINQ to MongoDB extension method)

This method is used to test whether an array (or array-like) field or property contains all of the provided values.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.A.ContainsAll(new[] { 1, 2, 3 })
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.A.ContainsAll(new[] { 1, 2, 3 }));
```

This is translated to the following MongoDB query:

```
{ A : { $all : [1, 2, 3] } }
```

ContainsAny (LINQ to MongoDB extension method)

This method is used to test whether an array (or array-like) field or property contains any of the provided values.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.A.ContainsAny(new[] { 1, 2, 3 })
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.A.ContainsAny(new[] { 1, 2, 3 }));
```

This is translated to the following MongoDB query:

```
{ A : { $in : [1, 2, 3] } }
```

Count method (array length)

This method is used to test whether an enumerable field or property has a certain count of items.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.L.Count() == 3
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.L.Count() == 3);
```

This is translated to the following MongoDB query:

```
{ L : { $size: 3 } }
```

Count property (array length)

This property is used to test whether a list (or list-like) field or property has a certain count of items.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.L.Count == 3
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.L.Count == 3);
```

This is translated to the following MongoDB query:

```
{ L : { $size: 3 } }
```

EndsWith (string method)

This method is used to test whether a string field or property of the document ends with a particular substring.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.S.EndsWith("abc")
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.S.EndsWith("abc"));
```

This is translated to the following MongoDB query (using regular expressions):

```
{ S : /abc$/ }
```

Enum comparisons (==, !=, <, <=, >, >=)

Enum fields or properties can be compared to constants of the same enum type. The relative comparison are based on the value of the underlying integer type.

```
public enum E { None, A, B };

var query =
    from c in collection.AsQueryable<C>()
    where c.E == E.A
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.E == E.A);
```

This is translated to the following MongoDB query:

```
{ E : 1 }
```

The LINQ implementation takes the representation of serialized values into account, so if you have configured your class map to store enums as string values instead of integer values the MongoDB query would instead be:

```
{ E : "A" }
```

In (LINQ to MongoDB extension method)

The In method is used to test whether a field or property is equal any of a set of provided values.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.X.In(new [] { 1, 2, 3 })
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c.X.In(new [] { 1, 2, 3 }));
```

This is translated to the following MongoDB query:

```
{ X : { $in : [1, 2, 3] } }
```

Inject

Inject is a pseudo-method that is used to inject a lower level MongoDB query into a LINQ query. The following query looks for X values that are larger than 0 and are 64-bit integers.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.X > 0 && Query.Type("X", BsonType.Int64).Inject()
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.X > 0 && Query.Type("X", BsonType.Int64).Inject());
```

This is translated to the following MongoDB query:

```
{ X : { $gt : 0, $type : 18 } }
```

IsMatch (regular expression method)

This method is used to test whether a string field or property matches a regular expression.

```
var regex = new Regex("^abc");
var query =
    from c in collection.AsQueryable<C>()
    where regex.IsMatch(c.S)
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => regex.IsMatch(c.S));
```

This is translated to the following MongoDB query:

```
{ S : /^abc/ }
```

You can also use the static IsMatch method.

```
var query =
    from c in collection.AsQueryable<C>()
    where Regex.IsMatch(c.S, "^abc")
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => Regex.IsMatch(c.S, "^abc"));
```

This is translated to the following MongoDB query:

```
{ S : /^abc/ }
```

Length (array length)

This method is used to test whether an array (or array-like) field or property has a certain count of items.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.A.Length == 3
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.A.Length == 3);
```

This is translated to the following MongoDB query:

```
{ A : { $size: 3 } }
```

Mod (% operator)

This operator is used to test the result of the mod operator against a field or property of the document. The following query matches all the documents where X is odd.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.X % 2 == 1
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.X % 2 == 1);
```

This is translated to the following MongoDB query:

```
{ X : { $mod : [2, 1] } }
```

Not (! operator)

The ! operator is used to reverse the sense of a test.

```
var query =
    from c in collection.AsQueryable<C>()
    where !(c.X > 1)
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => !(c.X > 1));
```

This is translated into the following MongoDB query:

```
{ X : { $not : { $gt : 1 } } }
```

Note that !(c.X > 1) is not equivalent to (c.X <= 1) in cases where c.X is missing or does not have a numeric type.

Numeric comparisons (==, !=, <, <=, >, >=)

Numeric fields or properties can be compared using any of the above operators.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.X == 0 && c.Y < 100
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.X > 0 && c.Y < 100);
```

This is translated into the following MongoDB query:

```
{ X : 0, Y : { $lt : 100 } }
```

Or (|| operator)

Sub-expressions can be combined with the || operator to test whether any of them is true.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.X > 0 || c.Y > 0
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.X > 0 || c.Y > 0);
```

This is translated to the following MongoDB query:

```
{ $or : [{ X : { $gt : 0 } }, { Y : { $gt : 0 } }] }
```

StartsWith (string method)

This method is used to test whether a string field or property of the document starts with a particular substring.

```
var query =
    from c in collection.AsQueryable<C>()
    where c.S.StartsWith("abc")
    select c;
// or
var query =
    collection.AsQueryable<C>()
    .Where(c => c.S.StartsWith("abc"));
```

This is translated to the following MongoDB query (using regular expressions):

```
{ S : /^abc/ }
```

CSharp Driver Quickstart

- Introduction
- Downloading the C# driver
- Add a reference to the C# driver DLLs
- Add required using statements
- Get a reference to the client object
- Get a reference to a server object
- Get a reference to a database object
- Decide if you want to work with the BsonDocument object model or with your own domain classes
- Get a reference to a collection object
- Insert a document
- Find an existing document
- Save a document
- Update an existing document
- Remove an existing document
- You do NOT need to call Connect or Disconnect
- Full sample program

Introduction

This quick-start provides just enough information to get you started using the C# driver. After you have gotten started you can refer to the rest of the documentation for more information.

Downloading the C# driver

You can download the C# driver here:

http://github.com/mongodb/mongo-csharp-driver/downloads

If you downloaded the .zip file, simply unzip it and place the contents anywhere you want.

If you downloaded the .msi file, double click on the .msi file to run the setup program, which will install the C# driver DLLs in the "C:\Program Files (x86)\MongoDB\CSharp Driver 1.x" directory (the exact path may vary on your system).

Add a reference to the C# driver DLLs

Right click on the References folder in Visual Studio's Solution Explorer and select "Add Reference...". Navigate to the folder where the C# driver DLLs were installed and add a reference to the following DLLs:

1. MongoDB.Bson.dll
2. MongoDB.Driver.dll

As an alternative you could use the NuGet package manager to add the C# driver package to your solution.

Add required using statements

As a minimum you will need the following using statements:

```
using MongoDB.Bson;
using MongoDB.Driver;
```

Additionally, you will frequently add one or more of these using statements:

```
using MongoDB.Driver.Builders;
using MongoDB.Driver.GridFS;
using MongoDB.Driver.Linq;
```

There are additional namespaces that would only be required in special cases.

Get a reference to the client object

The easiest way to get a reference to a client object is using a connection string:

```
var connectionString = "mongodb://localhost";
var client = new MongoClient(connectionString);
```

If you want to store the client object in a global variable you can. MongoClient is thread-safe.

Get a reference to a server object

To get a reference to a server object from the client object, write this:

```
var server = client.GetServer();
```

Get a reference to a database object

To get a reference to a database object from the server object, write this:

```
var database = server.GetDatabase("test"); // "test" is the name of the database
```

If you use more than one database call GetDatabase again for each database you want to use.

Decide if you want to work with the BsonDocument object model or with your own domain classes

There are two ways you can work with collections:

1. using the BsonDocument object model
2. using your own domain classes

You would use the BsonDocument object model when the data you are working with is so free form that it would be difficult or impossible to define domain classes for it.

Because it is so much easier to work with your own domain classes this quick-start will assume that you are going to do that. The C# driver can work with your domain classes provided that they:

1. Have a no-argument constructor
2. Define public read/write fields or properties for the data you want stored in the database

These requirements are essentially the same as those imposed by .NET's XmlSerializer.

In addition, if your domain class is going to be used as the root document it must contain an Id field or property (typically named "Id" although you can override that if necessary). Normally the Id will be of type ObjectId.

Get a reference to a collection object

For purposes of illustration let's assume you are using a domain class called Entity. You would get a reference to a collection containing Entity documents like this:

```
var collection = database.GetCollection<Entity>("entities"); // "entities" is the name of the
collection
```

Insert a document

Inserting a document is easy:

```
var entity = new Entity { Name = "Tom" };
collection.Insert(entity);
var id = entity.Id; // Insert will set the Id if necessary (as it was in this example)
```

Find an existing document

In this example we will read back an Entity assuming we know the Id value:

```
var query = Query.EQ("_id", id);
var entity = collection.FindOne(query);
```

Query.EQ is using the Query builder class to help you build the query. "_id" is the name of the field as stored in the database (normally the name of the field in the database is exactly the same as the name of the field or property in your domain class, but Id is an exception and is mapped to "_id" in the database).

Other query operators include: GT, GTE, In, LT, LTE, Near, NE, And, Or (and a few other more specialized ones).

Save a document

You can save changes to an existing document like this:

```
entity.Name = "Dick";
collection.Save(entity);
```

Update an existing document

An alternative to Save is Update. The difference is that Save sends the entire document back to the server, but Update sends just the changes. For example:

```
var query = Query.EQ("_id", id);
var update = Update.Set("Name", "Harry"); // update modifiers
collection.Update(query, update);
```

This example uses the Update builder to easily build the update modifiers.

Remove an existing document

To remove an existing document from a collection you write:

```
var query = Query.EQ("_id", id);
collection.Remove(query);
```

You do NOT need to call Connect or Disconnect

The C# driver has a connection pool to use connections to the server efficiently. There is no need to call Connect or Disconnect; just let the driver take care of the connections (calling Connect is harmless, but calling Disconnect is bad because it closes all the connections in the connection pool).

Full sample program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MongoDB.Bson;
using MongoDB.Driver;
using MongoDB.Driver.Builders;


namespace ConsoleApplication1
{
    public class Entity
    {
        public ObjectId Id { get; set; }
        public string Name { get; set; }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var connectionString = "mongodb://localhost";
var client = new MongoClient(connectionString);
            var server = client.GetServer();
            var database = server.GetDatabase("test");
            var collection = database.GetCollection<Entity>("entities");

            var entity = new Entity { Name = "Tom" };
            collection.Insert(entity);
            var id = entity.Id;

            var query = Query.EQ("_id", id);
            entity = collection.FindOne(query);

            entity.Name = "Dick";
            collection.Save(entity);

            var update = Update.Set("Name", "Harry");
            collection.Update(query, update);

            collection.Remove(query);
        }
    }
}
```

CSharp Driver Serialization Tutorial

- Introduction
- Creating a class map
- Conventions
- Field or property level serialization options
    - Opt-In
    - Element name
    - Element order
    - Identifying the Id field or property
    - Selecting an IdGenerator to use for an Id field or property
    - Ignoring a field or property
    - Ignoring null values
    - Default values
    - Ignoring a member based on a ShouldSerializeXyz method

        - Identifying required fields
        - Specifying the serializer
        - Serialization Options
            - DateTimeSerializationOptions
            - DictionarySerializationOptions
            - RepresentationSerializationOptions
- Class level serialization options
    - Ignoring extra elements
    - Supporting extra elements
    - Polymorphic classes and discriminators
    - Setting the discriminator value
    - Specifying known types
    - Scalar and hierarchical discriminators
- Customizing serialization
    - Implementing ISupportInitialize
    - Make a class responsible for its own serialization
    - Supplementing the default serializer provider
    - Write a custom serializer
    - Write a custom attribute
    - Write a custom Id generator
    - Write a custom convention
- Handling Schema Changes
    - A member has been added
    - A member has been removed
    - A member is renamed
    - The type of a member is changed
    - The representation of a member is changed

Introduction

This document refers to version 1.6 of the C# Driver.

This section of the C# Driver Tutorial discusses serialization (and deserialization) of instances of C# classes to and from BSON documents. Serialization is the process of mapping an object to a BSON document that can be saved in MongoDB, and deserialization is the reverse process of reconstructing an object from a BSON document. For that reason the serialization process is also often referred to as "Object Mapping."

Serialization is handled by the BSON Library. The BSON Library has an extensible serialization architecture, so if you need to take control of serialization you can. The BSON Library provides a default serializer which should meet most of your needs, and you can supplement the default serializer in various ways to handle your particular needs.

The main way the default serializer handles serialization is through "class maps". A class map is a structure that defines the mapping between a class and a BSON document. It contains a list of the fields and properties of the class that participate in serialization and for each one defines the required serialization parameters (e.g., the name of the BSON element, representation options, etc...).

The default serializer also has built in support for many .NET data types (primitive values, arrays, lists, dictionaries, etc...) for which class maps are not used.

Before an instance of a class can be serialized a class map must exist. You can either create this class map yourself or simply allow the class map to be created automatically when first needed (called "automapping"). You can exert some control over the automapping process either by decorating your classes with serialization related attributes or by using initialization code (attributes are very convenient to use but for those who prefer to keep serialization details out of their domain classes be assured that anything that can be done with attributes can also be done without them).

Creating a class map

To create a class map in your intialization code write:

```
BsonClassMap.RegisterClassMap<MyClass>();
```

which results in MyClass being automapped and registered. In this case you could just as well have allowed the class to be automapped by the serializer (when first serialized or deserialized). The one case where you must call RegisterClassMap yourself (even without arguments) is when you are using a polymorphic class hierarchy: in this case you must register all the known subclasses to guarantee that the discriminators get registered.

If you want to control the creation of the class map you can provide your own initialization code in the form of a lambda expression:

```
BsonClassMap.RegisterClassMap<MyClass)(cm => {
    cm.MapProperty(c => c.SomeProperty);
    cm.MapProperty(c => c.AnotherProperty);
});
```

When your lamba expression is executed the cm (short for class map) parameter is passed an empty class map for you to fill in. In this example two properties are added to the class map by calling the MapProperty method. The arguments to MapProperty are themselves lambda expressions which identify the property of the class. The advantage of using a lambda expression instead of just a string parameter with the name of the property is that Intellisense and compile time checking ensure that you can't misspell the name of the property.

It is also possible to use automapping and then override some of the results. We will see examples of that later on.

Note that a class map must only be registered once (an exception will be thrown if you try to register the same class map more than once). Usually you call RegisterClassMap from some code path that is known to execute only once (the Main method, the Application_Start event handler, etc...). If you must call RegisterClassMap from a code path that executes more than once, you can use IsClassMapRegistered to check whether a class map has already been registered for a class:

```
if (!BsonClassMap.IsClassMapRegistered(typeof(MyClass))) {
    // register class map for MyClass
}
```

Conventions

When automapping a class there are a lot of decisions that need to be made. For example:

- Which fields or properties of the class should be serialized
- Which field or property of the class is the "Id"
- What element name should be used in the BSON document
- If the class is being used polymorphically what discriminator values are used
- What should happen if a BSON document has elements we don't recognize
- Does the field or property have a default value
- Should the default value be serialized or ignored
- Should null values be serialized or ignored

Answers to these questions are represented by a set of "conventions". For each convention there is a default convention that is the most likely one you will be using, but you can override individual conventions (and even write your own) as necessary.

If you want to use your own conventions that differ from the defaults simply create an instance of ConventionProfile and set the values you want to override and then register that profile (in other words, tell the default serializer when your special conventions should be used). For example:

```
var myConventions = new ConventionProfile();
// override any conventions you want to be different
BsonClassMap.RegisterConventions(
    myConventions,
    t => t.FullName.StartsWith("MyNamespace.")
);
```

The second parameter is a filter function that defines when this convention profile should be used. In this case we are saying that any classes whose full names begin with "MyNamespace." should use myConventions.

ConventionProfile provides the following methods to allow you to set individual conventions:

- SetDefaultValueConvention
- SetElementNameConvention
- SetExtraElementsMemberConvention
- SetIdGeneratorConvention
- SetIdMemberConvention
- SetIgnoreExtraElementsConvention
- SetIgnoreIfDefaultConvention
- SetIgnoreIfNullConvention
- SetMemberFinderConvention
- SetSerializationOptionsConvention

Field or property level serialization options

There are many ways you can control serialization. The previous section discussed conventions, which are a convenient way to control serialization decisions for many classes at once. You can also control serialization at the individual class or field or property level.

Serialization can also be controlled either by decorating your classes and fields or properties with serialization related attributes or by writing code to initialize class maps appropriately. For each aspect of serialization you can control we will be showing both ways.

Opt-In

A majority of classes will have their properties mapped automatically. There are some circumstances where this does not happen. For instance, if

your property is read-only, it will not get included in the automapping of a class by default. In order to include the member, you can use the BsonElementAttribute.

```
public class MyClass {
    private readonly string _someProperty;

    [BsonElement]
    public string SomeProperty
    {
        get { return _someProperty; }
    }
}
```

The same result can be achieved without using attributes with the following initialization code:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.MapProperty(c => c.SomeProperty);
});
```

Element name

To specify an element name using attributes, write:

```
public class MyClass {
    [BsonElement("sp")]
    public string SomeProperty { get; set; }
}
```

The same result can be achieved without using attributes with the following initialization code:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.SomeProperty).SetElementName("sp");
});
```

Note that we are first automapping the class and then overriding one particular piece of the class map. If you didn't call AutoMap first then GetMemberMap would throw an exception because there would be no member maps.

Element order

If you want precise control over the order of the elements in the BSON document you can use the Order named parameter to the BsonElement attribute:

```
public class MyClass {
    [BsonElement("sp", Order = 1)]
    public string SomeProperty { get; set; }
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.SomeProperty).SetElementName("sp").SetOrder(1);
});
```

Any fields or properties that do not have an explicit Order will occur after those that do have an Order.

Identifying the Id field or property

To identify which field or property of a class is the Id you can write:

```
public class MyClass {
    [BsonId]
    public string SomeProperty { get; set; }
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.SetIdMember(cm.GetMemberMap(c => c.SomeProperty));
});
```

When not using AutoMap, you can also map a field or property and identify it as the Id in one step as follows:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.MapIdProperty(c => c.SomeProperty);
    // mappings for other fields and properties
});
```

Selecting an IdGenerator to use for an Id field or property

When you Insert a document the C# driver checks to see if the Id member has been assigned a value, and if not, generates a new unique value for it. Since the Id member can be of any type, the driver requires the help of a matching IdGenerator to check whether the Id has a value assigned to it and to generate a new value if necessary. The driver has the following IdGenerators built-in:

- BsonObjectIdGenerator
- CombGuidGenerator
- GuidGenerator
- NullIdChecker
- ObjectIdGenerator
- StringObjectIdGenerator
- ZeroIdChecker<T>

Some of these IdGenerators are used automatically for commonly used Id types:

- BsonObjectIdGenerator is used for BsonObjectId
- GuidGenerator is used for Guid
- ObjectIdGenerator is used for ObjectId
- StringObjectIdGenerator is used for strings represented externally as ObjectId

To select an IdGenerator to use for your Id field or property write:

```
public class MyClass {
    [BsonId(IdGenerator = typeof(CombGuidGenerator))]
    public Guid Id { get; set; }
}
```

Or using intialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.IdMemberMap.SetIdGenerator(CombGuidGenerator.Instance);
});
```

You could also say that you want to use the CombGuidGenerator for all Guids. In this case you would write:

```
BsonSerializer.RegisterIdGenerator(
    typeof(Guid),
    CombGuidGenerator.Instance
);
```

The NullIdChecker and ZeroIdChecker<T> IdGenerators can be used when you don't have an IdGenerator for an Id type but you want to enforce

that the Id is not null or zero. These pseudo-IdGenerators throw an exception if their GenerateId method is called. You can select it for an individual member just like a CombGuidGenerator was selected in the previous example, or you can turn on one or both of these IdGenerators for all types as follows:

```
BsonSerializer.UseNullIdChecker = true; // used for reference types
BsonSerializer.UseZeroIdChecker = true; // used for value types
```

Note: in version 1.0 of the C# Driver NullIdChecker and ZeroIdChecker<T> were always used, but it was decided that their use should be optional, since null and zero are valid values for an Id as far as the server is concerned, so they should only be considered an error if the developer has specifically said they should be.

Ignoring a field or property

When constructing a class map manually you can ignore a field or property simply by not adding it to the class map. When using AutoMap you need a way to specify that a field or property should be ignored. To do so using attributes write:

```
public class MyClass {
    [BsonIgnore]
    public string SomeProperty { get; set; }
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.UnmapProperty(c => c.SomeProperty);
});
```

In this case AutoMap will have initially added the property to the class map automatically but then UnmapProperty will remove it.

Ignoring null values

By default null values are serialized to the BSON document as a BSON Null. An alternative is to serialize nothing to the BSON document when the field or property has a null value. To specify this using attributes write:

```
public class MyClass {
    [BsonIgnoreIfNull]
    public string SomeProperty { get; set; }
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.SomeProperty).SetIgnoreIfNull(true);
});
```

Default values

You can specify a default value for a field or property as follows:

```
public class MyClass {
    [BsonDefaultValue("abc")]
    public string SomeProperty { get; set; }
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.SomeProperty).SetDefaultValue("abc");
});
```

You can also control whether default values are serialized or not (the default is yes). To not serialize default values using attributes write:

```
public class MyClass {
    [BsonDefaultValue("abc")]
    [BsonIgnoreIfDefault]
    public string SomeProperty { get; set; }
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.SomeProperty)
        .SetDefaultValue("abc")
        .SetIgnoreIfDefault(true);
});
```

Ignoring a member based on a ShouldSerializeXyz method

Sometimes the decision whether to serialize a member or not is more complicated than just whether the value is null or equal to the default value. You can write a method that determines whether a value should be serialized. Usually the method for member Xyz is named ShouldSerializeXyz. If you follow this naming convention then AutoMap will automatically detect the method and use it. For example:

```
public class Employee {
    public ObjectId Id { get; set; }
    [BsonDateTimeOptions(DateOnly = true)]
    public DateTime DateOfBirth { get; set; }

    public bool ShouldSerializeDateOfBirth() {
        return DateOfBirth > new DateTime(1900, 1, 1);
    }
}
```

Or using initialization code instead of naming conventions:

```
BsonClassMap.RegisterClassMap<Employee>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.DateOfBirth).SetShouldSerializeMethod(
        obj => ((Employee) obj).DateOfBirth > new DateTime(1900, 1, 1)
    );
});
```

Identifying required fields

Normally, the deserializer doesn't care if the document being deserialized doesn't have a matching element for every field or property of the class. The members that don't have a matching element simply get assigned their default value (or null if they don't have a default value).

If you want to make an element in the document be required, you can mark an individual field or property like this:

```
public class MyClass {
    public ObjectId Id { get; set; }
    [BsonRequired]
    public string X { get; set; }
}
```

Or using initialization code instead attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.X).SetIsRequired(true);
});
```

Specifying the serializer

There are times when a specific serializer needs to be used rather than letting the Bson library choose. This can be done in a couple of ways:

```
public class MyClass {
    public ObjectId Id { get; set; }
    [BsonSerializer(typeof(MyCustomStringSerializer))]
    public string X { get; set; }
}
```

Or using initialization code instead attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.X).SetSerializer(new MyCustomStringSerializer());
});
```

Serialization Options

Serialization of some classes can be more finely controlled using serialization options (which are represented using classes that implement the IBsonSerializationOptions interface). Whether a class uses serialization options or not, and which ones, depends on the particular class involved. The following sections describe the available serialization option classes and the classes that use them.

DateTimeSerializationOptions

These serialization options control how a DateTime is serialized. For example:

```
public class MyClass {
    [BsonDateTimeOptions(DateOnly = true)]
    public DateTime DateOfBirth { get; set; }
    [BsonDateTimeOptions(Kind = DateTimeKind.Local)]
    public DateTime AppointmentTime { get; set; }
}
```

Here we are specifying that the DateOfBirth value holds a date only (so the TimeOfDay component must be zero). Additionally, because this is a date only, no timezone conversions at all will be performed. The AppointmentTime value is in local time and will be converted to UTC when it is serialized and converted back to local time when it is deserialized.

You can specify the same options using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.DateOfBirth)
        .SetSerializationOptions(
            new DateTimeSerializationOptions { DateOnly = true });
    cm.GetMemberMap(c => c.AppointmentTime)
        .SetSerializationOptions(
            new DateTimeSerializationOptions { Kind = DateTimeKind.Local });
});
```

DateTimeSerializationOptions are supported by the serializers for the following classes: BsonDateTime and DateTime.

DictionarySerializationOptions

When serializing dictionaries there are several alternative ways that the contents of the dictionary can be represented. The different ways are

represented by the DictionaryRepresentation enumeration:

```
public enum DictionaryRepresentation {
    Dynamic,
    Document,
    ArrayOfArrays,
    ArrayOfDocuments
}
```

A dictionary represented as a `Document` will be stored as a BsonDocument, and each entry in the dictionary will be represented by a BsonElement with the name equal to the key of the dictionary entry and the value equal to the value of the dictionary entry. This representation can only be used when all the keys in a dictionary are strings that are valid element names.

A dictionary represented as an `ArrayOfArrays` will be stored as a BsonArray of key/value pairs, where each key/value pair is stored as a nested two-element BsonArray where the two elements are the key and the value of the dictionary entry. This representation can be used even when the keys of the dictionary are not strings. This representation is very general and compact, and is the default representation when `Document` does not apply. One problem with this representation is that it is difficult to write queries against it, which motivated the introduction in the 1.2 version of the driver of the `ArrayOfDocuments` representation.

A dictionary represented as an `ArrayOfDocuments` will be stored as a BsonArray of key/value pairs, where each key/value pair is stored as a nested two-element BsonDocument of the form { k : key, v : value }. This representation is just as general as the `ArrayOfArrays` representation, but because the keys and values are tagged with element names it is much easier to write queries against it. For backward compatibility reasons this is not the default representation.

If the `Dynamic` representation is specified, the dictionary key values are inspected before serialization, and if all the keys are strings which are also valid element names, then the `Document` representation will be used, otherwise the `ArrayOfArrays` representation will be used.

If no other representation for a dictionary is specified, then `Dynamic` is assumed.

You can specify a DictionarySerializationOption as follows:

```
public class C {
    public ObjectId Id;
    [BsonDictionaryOptions(DictionaryRepresentation.ArrayOfDocuments)]
    public Dictionary<string, int> Values;
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<C>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.Values)
        .SetSerializationOptions(DictionarySerializationOptions.ArrayOfDocuments);
});
```

DictionarySerializationOptions are suppored by the serializers for the following classes: the generic classes and interfaces Dictionary, IDictionary, SortedDictionary and SortedList, and the non-generic classes and interfaces Hashtable, IDictionary, ListDictionary, OrderedDictionary and SortedList.

RepresentationSerializationOptions

For some .NET primitive types you can control what BSON type you want used to represent the value in the BSON document. For example, you can specify whether a char value should be represented as a BSON Int32 or as a one-character BSON String:

```
public class MyClass {
    [BsonRepresentation(BsonType.Int32)]
    public char RepresentAsInt32 { get; set; }
    [BsonRepresentation(BsonType.String)]
    public char RepresentAsString { get; set; }
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.GetMemberMap(c => c.RepresentAsInt32)
        .SetRepresentation(BsonType.Int32);
    cm.GetMemberMap(c => c.RepresentAsString)
        .SetRepresentation(BsonType.String);
});
```

One case that deserves special mention is representing a string externally as an ObjectId. For example:

```
public class Employee {
    [BsonRepresentation(BsonType.ObjectId)]
    public string Id { get; set; }
    // other properties
}
```

In this case the serializer will convert the ObjectId to a string when reading data from the database and will convert the string back to an ObjectId when writing data to the database (the string value must be a valid ObjectId). Typically this is done when you want to keep your domain classes free of any dependencies on the C# driver, so you don't want to declare the Id as an ObjectId. String serves as a neutral representation that is at the same time easily readable for debugging purposes. To keep your domain classes free of dependencies on the C# driver you also won't want to use attributes, so you can accomplish the same thing using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<Employee>(cm => {
    cm.AutoMap();
    cm.IdMemberMap.SetRepresentation(BsonType.ObjectId);
});
```

Class level serialization options

There are several serialization options that are related to the class itself instead of to any particular field or property. You can set these class level options either by decorating the class with serialization related attributes or by writing initialization code. As usual, we will show both ways in the examples.

Ignoring extra elements

When a BSON document is deserialized the name of each element is used to look up a matching field or property in the class map. Normally, if no matching field or property is found, an exception will be thrown. If you want to ignore extra elements during deserialization, use the following attribute:

```
[BsonIgnoreExtraElements]
public MyClass {
    // fields and properties
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.SetIgnoreExtraElements(true);
});
```

Supporting extra elements

You can design your class to be capable of handling any extra elements that might be found in a BSON document during deserialization. To do so, you must have a property of type BsonDocument and you must identify that property as the one that should hold any extra elements that are found (or you can name the property "ExtraElements" so that the default ExtraElementsMemberConvention will find it automatically). For example:

```
public MyClass {
    // fields and properties
[BsonExtraElements]
    public BsonDocument CatchAll { get; set; }
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.SetExtraElementsMember(cm.GetMemberMap(c => c.CatchAll));
});
```

When a BSON document is deserialized any extra elements found will be stored in the extra elements BsonDocument property. When the class is serialized the extra elements will be serialized also. One thing to note though is that the serialized class will probably not have the elements in exactly the same order as the original document. All extra elements will be serialized together when the extra elements member is serialized.

Polymorphic classes and discriminators

When you have a class hierarchy and will be serializing instances of varying classes to the same collection you need a way to distinguish one from another. The normal way to do so is to write some kind of special value (called a "discriminator") in the document along with the rest of the elements that you can later look at to tell them apart. Since there are potentially many ways you could discriminate between actual types, the default serializer uses conventions for discriminators. The default serializer provides two standard discriminators: ScalarDiscriminatorConvention and HierarchicalDiscriminatorConvention. The default is the HierarchicalDiscriminatorConvention, but it behaves just like the ScalarDiscriminatorConvention until certain options are set to trigger its hierarchical behavior (more on this later).

The default discriminator conventions both use an element named "_t" to store the discriminator value in the BSON document. This element will normally be the second element in the BSON document (right after the "_id"). In the case of the ScalarDiscriminatorConvention the value of "_t" will be a single string. In the case of the HierarchicalDiscriminatorConvention the value of "_t" will be an array of discriminator values, one for each level of the class inheritance tree (again, more on this later).

While you will normally be just fine with the default discriminator convention, you might have to write a custom discriminator convention if you must inter-operate with data written by another driver or object mapper that uses a different convention for its discriminators.

Setting the discriminator value

The default value for the discriminator is the name of the class (without the namespace part). You can specify a different value using attributes:

```
[BsonDiscriminator("myclass")]
public MyClass {
    // fields and properties
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {
    cm.AutoMap();
    cm.SetDiscriminator("myclass");
});
```

Specifying known types

When deserializing polymorphic classes it is important that the serializer know about all the classes in the hierarchy before deserialization begins. If you ever see an error message about an "Unknown discriminator" it is because the deserializer can't figure out the class for that discriminator. If you are mapping your classes programmatically simply make sure that all classes in the hierarchy have been mapped before beginning deserialization. When using attributes and automapping you will need to inform the serializer about known types (i.e. subclasses) it should create class maps for. Here is an example of how to do this:

```
[BsonKnownTypes(typeof(Cat), typeof(Dog)]
public class Animal {
}

[BsonKnownTypes(typeof(Lion), typeof(Tiger)]
public class Cat : Animal {
}

public class Dog : Animal {
}

public class Lion : Cat {
}

public class Tiger : Cat {
}
```

The BsonKnownTypes attribute lets the serializer know what subclasses it might encounter during deserialization, so when Animal is automapped the serializer will also automap Cat and Dog (and recursively, Lion and Tiger as well).

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<Animal>();
BsonClassMap.RegisterClassMap<Cat>();
BsonClassMap.RegisterClassMap<Dog>();
BsonClassMap.RegisterClassMap<Lion>();
BsonClassMap.RegisterClassMap<Tiger>();
```

Scalar and hierarchical discriminators

Normally a discriminator is simply the name of the class (although it could be different if you are using a custom discriminator convention or have explicitly specified a discriminator for a class). So a collection containing a mix of different type of Animal documents might look like:

```
{ _t : "Animal", ... }
{ _t : "Cat", ... }
{ _t : "Dog", ... }
{ _t : "Lion", ... }
{ _t : "Tiger", ... }
```

Sometimes it can be helpful to record a hierarchy of discriminator values, one for each level of the hierarchy. To do this, you must first mark a base class as being the root of a hierarchy, and then the default HierarchicalDiscriminatorConvention will automatically record discriminators as array values instead.

To identify Animal as the root of a hierarchy use the BsonDiscriminator attribute with the RootClass named parameter:

```
[BsonDiscriminator(RootClass = true)]
[BsonKnownTypes(typeof(Cat), typeof(Dog)]
public class Animal {
}

// the rest of the hierarchy as before
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<Animal>(cm => {
    cm.AutoMap();
    cm.SetIsRootClass(true);
});
BsonClassMap.RegisterClassMap<Cat>();
BsonClassMap.RegisterClassMap<Dog>();
BsonClassMap.RegisterClassMap<Lion>();
BsonClassMap.RegisterClassMap<Tiger>();
```

Now that you have identified Animal as a root class, the discriminator values will look a little bit different:

```
{ _t : "Animal", ... }
{ _t : ["Animal", "Cat"], ... }
{ _t : ["Animal", "Dog"], ... }
{ _t : ["Animal", "Cat", "Lion"], ... }
{ _t : ["Animal", "Cat", "Tiger"], ... }
```

The main reason you might choose to use hierarchical discriminators is because it makes it possibly to query for all instances of any class in the hierarchy. For example, to read all the Cat documents we can write:

```
var query = Query.EQ("_t", "Cat");
var cursor = collection.FindAs<Animal>(query);
foreach (var cat in cursor) {
    // process cat
}
```

This works because of the way MongoDB handles queries against array values.

Customizing serialization

There are several ways you can customize serialization:

1. Implement ISupportInitialize
2. Make a class responsible for its own serialization
3. Supplementing the default serializer
4. Write a custom serializer
5. Write a custom attribute
6. Write a custom Id generator
7. Write a custom convention

Implementing ISupportInitialize

The driver respects an entity implementing ISupportInitialize which contains 2 methods, BeginInit and EndInit. These method are called before deserialization begins and after it is complete. It is useful for running operations before or after deserialization such as handling schema changes are pre-calculating some expensive operations.

Make a class responsible for its own serialization

One way you can customize how a class is serialized is to make it responsible for its own serialization. You do so by implementing the IBsonSerializable interface:

```
public class MyClass : IBsonSerializable {
    // implement Deserialize method
// implement Serialize method
}
```

You also must implement the GetDocumentId and SetDocumentId methods. If your class is never used as a root document these methods can just be stubs that throw a NotSupportedException. Otherwise, return true from GetDocumentId if the value passed in has an Id, and set the Id value in SetDocumentId.

There is nothing else you have to do besides implementing this interface. The BSON Library automatically checks whether objects being serialized implement this interface and if so routes serialization calls directly to the classes.

This can be a very efficient way to customize serialization, but it does have the drawback that it pollutes your domain classes with serialization details, so there is also the option of writing a custom serializer as described next.

Supplementing the default serializer provider

You can register your own serialization provider to supplement the default serializer. Register it like this:

```
IBsonSerializationProvider myProvider;
BsonSerializer.RegisterSerializationProvider(myProvider);
```

You should register your provider as early as possible. Your provider will be called first before the default serializer. You can delegate handling of any types your custom provider isn't prepared to handle to the default serializer by returning null from GetSerializer.

Write a custom serializer

A custom serializer can handle serialization of your classes without requiring any changes to those classes. This is a big advantage when you either don't want to modify those classes or can't (perhaps because you don't have control over them). You must register your custom serializer so that the BSON Library knows of its existence and can call it when appropriate.

If you write a custom serializer you will have to become familiar with the BsonReader and BsonWriter abstract classes, which are not documented here, but are relatively straightforward to use. Look at the existing serializers in the driver for examples of how BsonReader and BsonWriter are used.

To implement and register a custom serializer you would:

```
// MyClass is the class for which you are writing a custom serializer
public MyClass {
}

// MyClassSerializer is the custom serializer for MyClass
public MyClassSerializer : IBsonSerializer {
    // implement Deserialize
// implement GetDefaultSerializationOptions
// implement Serialize
}

// register your custom serializer
BsonSerializer.RegisterSerializer(
    typeof(MyClass),
    new MyClassSerializer()
);
```

You can also decorate the target class with a BsonSerializer attribute instead of using the BsonSerializer.RegisterSerializer method:

```
[BsonSerializer(typeof(MyClassSerializer))]
public MyClass {
}
```

The IBsonSerializer interface is all that is necessary for serialization. However, there are some extension interfaces that will enable further use in other parts of the api such as saving a class or LINQ.

If your class is used as a root document, you will need to implement the IBsonIdProvider interface in order for "Saving" the document to function. MongoCollection.Save requires a document identity in order to know if it should generate an insert or update statement. Below is the extension to the above MyClassSerializer.

```
public MyClassSerializer : IBsonSerializer, IBsonIdProvider {
    // ...

    // implement GetDocumentId
// implement SetDocumentId
}
```

In order to enable LINQ to properly construct type-safe queries using a custom serializer, it needs access to member information or array information. If your custom serializer is for a class, as MyClassSerializer is above, then you should implement IBsonDocumentSerializer.

```
public MyClassSerializer : IBsonSerializer, IBsonDocumentSerializer {
    // ...

    // implement GetMemberSerializationInfo
}
```

If, however, your class is a collection that should be serialized as an array, it should implement IBsonArraySerializer.

```
public MyClassSerializer : IBsonSerializer, IBsonArraySerializer {
    // ...

    // implement GetItemSerializationInfo
}
```

To debug a custom serializer you can either Insert a document containing a value serialized by your custom serializer into some collection and then use the mongo shell to examine what the resulting document looks like. Alternatively you can use the ToJson method to see the result of the serializer without having to Insert anything into a collection as follows:

```
// assume a custom serializer has been registered for class C
var c = new C();
var json = c.ToJson();
// inspect the json string variable to see how c was serialized
```

Write a custom attribute

The auto mapping ability of BSON library utilizes attributes that implement IBsonClassMapModifier or IBsonMemberMapModifier for class level attributes or member level attributes respectively. Each of these interfaces have a single method called Apply that is passed a BsonClassMap or a BsonMemberMap with which it can use any of the public methods and properties to modify the map instance. One example of this would be to create an attribute called BsonEncryptionAttribute that is used to encrypt a string before sending it to the database and decrypt it when reading it back out.

View the existing attributes for examples of how these interfaces function.

Write a custom Id generator

You can write your own IdGenerator. For example, suppose you wanted to generate integer Employee Ids:

```
public class EmployeeIdGenerator : IIdGenerator {
    // implement GenerateId
// implement IsEmpty
}
```

You can specify that this generator be used for Employee Ids using attributes:

```
public class Employee {
    [BsonId(IdGenerator = typeof(EmployeeIdGenerator)]
    public int Id { get; set; }
    // other fields or properties
}
```

Or using initialization code instead of attributes:

```
BsonClassMap.RegisterClassMap<Employee >(cm => {
    cm.AutoMap();
    cm.IdMember.SetIdGenerator(new EmployeeIdGenerator());
});
```

Alternatively, you can get by without an Id generator at all by just assigning a value to the Id property before calling Insert or Save.

Write a custom convention

Earlier in this tutorial we discussed replacing one or more of the default conventions. You can either replace them with one of the provided alternatives or you can write your own convention. Writing your own convention varies slightly from convention to convention.

As an example we will write a custom convention to find the Id member of a class (the default convention looks for a member named "Id"). Our custom convention will instead consider any public property whose name ends in "Id" to be the Id for the class. We can implement this convention as follows:

```
public class EndsWithIdConvention : IIdMemberConvention {
    public string FindIdMember(Type type) {
        foreach (var property in type.GetProperties()) {
            if (property.Name.EndsWith("Id")) {
                return property.Name;
            }
        }
        return null;
    }
}
```

And we can configure this convention to be used with all of our own classes by writing:

```
var myConventions = new ConventionProfile();
myConventions.SetIdMemberConvention(new EndsWithIdConvention());
BsonClassMap.RegisterConventions(
    myConventions,
    t => t.FullName.StartsWith("MyNamespace.")
);
```

Warning: because GetProperties is not guaranteed to return properties in any particular order this convention as written will behave unpredictably for a class that contains more than one property whose name ends in "Id".

Handling Schema Changes

Just because MongoDB is schema-less does not mean that your code can handle a schema-less document. Most likely, if you are using a statically typed language like C# or VB.NET, then your code is not-flexible and needs to be mapped to a known schema.

There are a number of different ways that a schema can change from one version of your application to the next.

1. A new member is added
2. A member is deleted
3. A member is renamed
4. The type of a member is changed
5. The representation of a member is changed

How you handle these is up to you. There primary two different strategies.

1. Write an upgrade script.
2. Incrementally update your documents as they are used

The easiest and most bullet-proof of the strategies is to write an upgrade script. There is effectively no difference to this method between a relational database (SQL Server, Oracle) and MongoDB. Identify the documents that need to be changed and update them.

Alternatively, and not supportable in most relational databases, is the incremental upgrade. The idea is that your documents get updated as they are used. Documents that are never used never get updated. Because of this, there are some definite pitfalls you will need to be aware of.

First, queries against a schema where half the documents are version 1 and half the documents are version 2 could go awry. For instance, if you rename an element, then your query will need to test both the old element name and the new element name to get all the results.

Second, any incremental upgrade code must stay in the code-base until all the documents have been upgraded. For instance, if there have been 3 versions of a document, [1, 2, and 3] and we remove the upgrade code from version 1 to version 2, any documents that still exist as version 1 are un-upgradeable.

So, with that being said, let's talk about handling the schema change variations.

A member has been added

When a new member is added to an entity, there is nothing that needs to be done other than restarting the application if you are using the auto mapping features. If not, then you will manually need to map the member in the same way all the other members are getting mapped.

Existing documents will not have this element and it will show up in your class with its default value. You can, of course, specify a default value.

A member has been removed

When a member has been removed from am entity, it will continue to exist in the documents. The serializer will throw an exception when this element is seen because it doesn't know what to do with it. The 2 previously discussed items that can be used to combat this are the BsonIgnoreExtraElements class-level attribute and the ExtraElements members.

A member is renamed

When a member has been renamed, it will exist in old documents with the old name and in new docuemnts with the new name. The way to hande incremental upgrades for this rename would be to implement an ExtraElements member in conjunction with ISupportInitialize. For example, let's say that a class used to have a Name property which has now been split into a FirstName and a LastName property.

```
public class MyClass : ISupportInitialize {
    public string FirstName { get; set; }
    public string LastName { get; set; }

    [BsonExtraElements]
    public IDictionary<string, object> ExtraElements { get; set; }

    void ISupportInitialize.BeginInit() {
        // nothing to do at begin
}

    void ISupportInitialize.EndInit() {
        object nameValue;
        if(!ExtraElements.TryGetValue("Name", out nameValue)) {
            return;
        }

        var name = (string)nameValue;

        // remove the Name element so that it doesn't get persisted back to the database
ExtraElements.Remove("Name");

        // assuming all names are "First Last"
var nameParts = name.Split(' ');

        FirstName = nameParts[0];
        LastName = nameParts[1];
    }
}
```

The type of a member is changed

If the .NET type is compatible with the old type (an integer is changed to a double), then everything will continue to work. Otherwise, a custom serializer or a migration script will be required.

The representation of a member is changed

If the representation of a member is changed and the representations are compatible, then everything will continue to work. Otherwise, a custom serializer or a migration script will be required.

CSharp Driver Tutorial

- C# Driver version v1.6.x
- Introduction
- Downloading
- Building
    - Dependencies
    - Running unit tests
- Installing
- References and namespaces
- The BSON Library
    - BsonType
    - BsonValue and subclasses
        - BsonType property
        - As[Type] Properties
        - Is[Type] Properties
        - To[Type] conversion methods
        - Static Create methods
            - Implicit conversions
        - BsonMaxKey, BsonMinKey, BsonNull and BsonUndefined
        - ObjectId and BsonObjectId

        - BsonElement
        - BsonDocument
            - BsonDocument constructor
            - Create a new document and call Add and Set methods
            - Create a new document and use the fluent interface Add and Set methods
            - Create a new document and use C#'s collection initializer syntax (recommended)
            - Creating nested BSON documents
            - Add methods
            - Accessing BsonDocument elements
        - BsonArray
            - Constructors
            - Add and AddRange methods
            - Indexer
- The C# Driver
    - Thread safety
    - MongoClient class
        - Connection strings
        - SSL Support
        - Authentication
        - GetServer method
    - MongoServer class
        - GetDatabase method
        - RequestStart/RequestDone methods
        - Other properties and methods
    - MongoDatabase class
        - GetCollection method
        - Other properties and methods
    - MongoCollection<TDefaultDocument> class
        - Insert<TDocument> method
        - InsertBatch method
        - FindOne and FindOneAs methods
        - Find and FindAs methods
        - Save<TDocument> method
        - Update method
        - FindAndModify method
        - MapReduce method
        - Other properties and methods
    - MongoCursor<TDocument> class
        - Enumerating a cursor
        - Modifying a cursor before enumerating it
        - Modifiable properties of a cursor
        - Other methods
    - WriteConcern class

C# Driver version v1.6.x

This tutorial is for v1.6.x of the C# Driver. Api documentation can be found here: http://api.mongodb.org/csharp/current/.

Introduction

This tutorial introduces the 10gen supported C# Driver for MongoDB. The C# Driver consists of two libraries: the BSON Library and the C# Driver. The BSON Library can be used independently of the C# Driver if desired. The C# Driver requires the BSON Library.

You may also be interested in the C# Driver Serialization Tutorial. It is a separate tutorial because it covers quite a lot of material.

Downloading

The C# Driver is available in source and binary form. While the BSON Library can be used independently of the C# Driver they are both stored in the same repository.

The source may be downloaded from github.com.

We use msysgit as our Windows git client. It can be downloaded from: http://msysgit.github.com/.

To clone the repository run the following commands from a git bash shell:

```
$ cd <parentdirectory>
$ git config --global core.autocrlf true
$ git clone git://github.com/mongodb/mongo-csharp-driver.git
$ cd mongo-csharp-driver
$ git config core.autocrlf true
```

You must set the global setting for core.autocrlf to true before cloning the repository. After you clone the repository, we recommend you set the local setting for core.autocrlf to true (as shown above) so that future changes to the global setting for core.autocrlf do not affect this repository. If you then want to change your global setting for core.autocrlf to false run:

```
$ git config --global core.autocrlf false
```

The typical symptom of problems with the setting for core.autocrlf is git reporting that an entire file has been modified (because of differences in the line endings). It is rather tedious to change the setting of core.autocrlf for a repository after it has been created, so it is important to get it right from the start.

You can download a zip file of the source files (without cloning the repository) by clicking on the Downloads button at:

http://github.com/mongodb/mongo-csharp-driver

You can download binaries (in both .msi and .zip formats) from:

http://github.com/mongodb/mongo-csharp-driver/downloads

Building

We are currently building the C# Driver with Visual Studio 2010. The name of the solution file is `CSharpDriver-2010.sln`.

Dependencies

The unit tests depend on NUnit 2.5.9, which is included in the dependencies folder of the repository. You can build the C# Driver without installing NUnit, but you must install NUnit before running the unit tests (unless you use a different test runner).

Running unit tests

There are three projects containing unit tests:

1. BsonUnitTests
2. DriverUnitTests
3. DriverUnitTestsVB

The BsonUnitTests do not connect to a MongoDB server. The DriverUnitTests and DriverUnitTestsVB connect to an instance of MongoDB running on the default port on localhost.

An easy way to run the unit tests is to set one of the unit test projects as the startup project and configure the project settings as follows (using BsonUnitTests as an example):

- On the Debug tab:
    1. Set Start Action to Start External Program
    2. Set external program to: `C:\Program Files (x86)\NUnit 2.5.9\bin\net-2.0\nunit.exe`
    3. Set command line aguments to: `BsonUnitTests.csproj /config:Debug /run`
    4. Set working directory to: the directory where BsonUnitTest.csproj is located

Repeat the above steps for the Release configuration (using /config:Release instead) if you also want to run unit tests for Release builds.

The exact location of the nunit.exe program might vary slightly on your machine.

To run the DriverUnitTests and DriverUnitTestsVB perform the same steps (modified as necessary).

Installing

If you want to install the C# Driver on your machine you can use the setup program (see above for download instructions). The setup program is very simple and just copies the DLLs to your specified installation directory.

If you downloaded the binaries zip file simply extract the files and place them wherever you want them to be.

Note: if you download the .zip file Windows might require you to "Unblock" the help file. If Windows asks "Do you want to open this file?" when you double click on the CSharpDriverDocs.chm file, clear the check box next to "Always ask before opening this file" before pressing the Open button. Alternatively, you can right click on the CSharpDriverDocs.chm file and select Properties, and then press the Unblock button at the bottom of the General tab. If the Unblock button is not present then the help file does not need to be unblocked.

References and namespaces

To use the C# Driver you must add references to the following DLLs:

1. `MongoDB.Bson.dll`

2. `MongoDB.Driver.dll`

As a minimum add the following using statements to your source files:

```
using MongoDB.Bson;
using MongoDB.Driver;
```

Additionally you will frequently add some of the following using statements:

```
using MongoDB.Driver.Builders;
using MongoDB.Driver.GridFS;
using MongoDB.Driver.Linq;
```

In some cases you might add some of the following using statements if you are using some of the optional parts of the C# Driver:

```
using MongoDB.Bson.IO;
using MongoDB.Bson.Serialization;
using MongoDB.Bson.Serialization.Attributes;
using MongoDB.Bson.Serialization.Conventions;
using MongoDB.Bson.Serialization.IdGenerators;
using MongoDB.Bson.Serialization.Options;
using MongoDB.Bson.Serialization.Serializers;
using MongoDB.Driver.Wrappers;
```

The BSON Library

The C# Driver is built on top of the BSON Library, which handles all the details of the BSON specification, including: I/O, serialization, and an in-memory object model of BSON documents.

The important classes of the BSON object model are: BsonType, BsonValue, BsonElement, BsonDocument and BsonArray.

BsonType

This enumeration is used to specify the type of a BSON value. It is defined as:

```
public enum BsonType {
    Double = 0x01,
    String = 0x02,
    Document = 0x03,
    Array = 0x04,
    Binary = 0x05,
    Undefined = 0x06,
    ObjectId = 0x07,
    Boolean = 0x08,
    DateTime = 0x09,
    Null = 0x0a,
    RegularExpression = 0x0b,
    JavaScript = 0x0d,
    Symbol = 0x0e,
    JavaScriptWithScope = 0x0f,
    Int32 = 0x10,
    Timestamp = 0x11,
    Int64 = 0x12,
    MinKey = 0xff,
    MaxKey = 0x7f
}
```

BsonValue and subclasses

BsonValue is an abstract class that represents a typed BSON value. There is a concrete subclass of BsonValue for each of the values defined by the BsonType enum. There are several ways to obtain an instance of BsonValue:

- Use a public constructor (if available) of a subclass of BsonValue
- Use a static Create method of BsonValue

- Use a static Create method of a subclass of BsonValue
- Use a static property of a subclass of BsonValue
- Use an implicit conversion to BsonValue

The advantage of using the static Create methods is that they can return a pre-created instance for frequently used values. They can also return null (which a constructor cannot) which is useful for handling optional elements when creating BsonDocuments using functional construction. The static properties refer to pre-created instances of frequently used values. Implicit conversions allow you to use primitive .NET values wherever a BsonValue is expected, and the .NET value will automatically be converted to a BsonValue.

BsonType property

BsonValue has a property called BsonType that you can use to query the actual type of a BsonValue. The following example shows several ways to determine the type of a BsonValue:

```
BsonValue value;
if (value.BsonType == BsonType.Int32) {
    // we know value is an instance of BsonInt32
}
if (value is BsonInt32) {
    // another way to tell that value is a BsonInt32
}
if (value.IsInt32) {
    // the easiest way to tell that value is a BsonInt32
}
```

As[Type] Properties

BsonValue has a number of properties that cast a BsonValue to one of its subclasses or a primitive .NET type. It is important to note that these all are casts, not conversions. They will throw an InvalidCastException if the BsonValue is not of the corresponding type. See also the To[Type] methods which do conversions, and the Is[Type] properties which you can use to query the type of a BsonValue before attempting to use one of the As[Type] properties.

```
BsonDocument document;
string name = document["name"].AsString;
int age = document["age"].AsInt32;
BsonDocument address = document["address"].AsBsonDocument;
string zip = address["zip"].AsString;
```

Is[Type] Properties

BsonValue has the following boolean properties you can use to test what kind of BsonValue it is. These can be used as follows:

```
BsonDocument document;
int age = -1;
if (document.Contains["age"] && document["age"].IsInt32) {
    age = document["age"].AsInt32;
}
```

To[Type] conversion methods

Unlike the As[Type] methods, the To[Type] methods perform some limited conversion between convertible types, like int and double.

The `ToBoolean` method never fails. It uses JavaScript's definition of truthiness: false, 0, 0.0, NaN, BsonNull, BsonUndefined and "" are false, and everything else is true (include the string "false").

The `ToBoolean` method is particularly useful when the documents you are processing might have inconsistent ways of recording true/false values:

```
if (employee["ismanager"].ToBoolean()) {
    // we know the employee is a manager
// works with many ways of recording boolean values
}
```

The `ToDouble`, `ToInt32`, and `ToInt64` methods never fail when converting between numeric types, though the value might be truncated if it doesn't fit in the target type. A string can be converted to a numeric type, but an exception will be thrown if the string cannot be parsed as a value

of the target type.

Static Create methods

Because BsonValue is an abstract class you cannot create instances of BsonValue (only instances of concrete subclasses). BsonValue has a static `Create` method that takes an argument of type `object` and determines at runtime the actual type of BsonValue to create. Subclasses of BsonValue also have static Create methods tailored to their own needs.

Implicit conversions

Implicit conversions are defined from the following .NET types to BsonValue:

- `bool`
- `byte[]`
- `DateTime`
- `double`
- `Enum`
- `Guid`
- `int`
- `long`
- `ObjectId`
- `Regex`
- `string`

These eliminate the need for almost all calls to BsonValue constructors or Create methods. For example:

```
BsonValue b = true; // b is an instance of BsonBoolean
BsonValue d = 3.14159; // d is an instance of BsonDouble
BsonValue i = 1; // i is an instance of BsonInt32
BsonValue s = "Hello"; // s is an instance of BsonString
```

BsonMaxKey, BsonMinKey, BsonNull and BsonUndefined

These classes are singletons, so only a single instance of each class exists. You refer to these instances using the static Value property of each class:

```
document["status"] = BsonNull.Value;
document["priority"] = BsonMaxKey.Value;
```

Note that C# null and BsonNull.Value are two different things. The latter is an actual C# object that represents a BSON null value (it's a subtle difference, but plays an important role in functional construction).

ObjectId and BsonObjectId

ObjectId is a struct that holds the raw value of a BSON ObjectId. BsonObjectId is a subclass of BsonValue whose Value property is of type ObjectId.

Here are some common ways of creating ObjectId values:

```
var id1 = new ObjectId(); // same as ObjectId.Empty
var id2 = ObjectId.Empty; // all zeroes
var id3 = ObjectId.GenerateNewId(); // generates new unique Id
var id4 = ObjectId.Parse("4dad901291c2949e7a5b6aa8"); // parses a 24 hex digit string
```

Note that the first example behaves differently in C# than in JavaScript. In C# it creates an ObjectId of all zeroes, but in JavaScript it generates a new unique Id. This difference can't be avoided because in C# the default constructor of a value type always initializes the value to all zeros.

BsonElement

A BsonElement is a name/value pair, where the value is a BsonValue. It is used as the building block of BsonDocument, which consists of zero or more elements. You will rarely create BsonElements directly, as they are usually created indirectly as needed. For example:

```
document.Add(new BsonElement("age", 21)); // OK, but next line is shorter
document.Add("age", 21); // creates BsonElement automatically
```

BsonDocument

A BsonDocument is a collection of name/value pairs (represented by BsonElements). It is an in-memory object model of a BSON document. There are three ways to create and populate a BsonDocument:

1. Create a new document and call Add and Set methods
2. Create a new document and use the fluent interface Add and Set methods
3. Create a new document and use C#'s collection initializer syntax (recommended)

BsonDocument constructor

BsonDocument has the following constructors:

- `BsonDocument()`
- `BsonDocument(string name, BsonValue value)`
- `BsonDocument(BsonElement element)`
- `BsonDocument(Dictionary<string, object> dictionary)`
- `BsonDocument(Dictionary<string, object> dictionary, IEnumerable<string> keys)`
- `BsonDocument(IDictionary dictionary)`
- `BsonDocument(IDictionary dictionary, IEnumerable<string> keys)`
- `BsonDocument(IDictionary<string, object> dictionary)`
- `BsonDocument(IDictionary<string, object> dictionary, IEnumerable<string> keys)`
- `BsonDocument(IEnumerabe<BsonElement> elements)`
- `BsonDocument(params BsonElement[] elements)`
- `BsonDocument(bool allowDuplicateNames)`

The first two are the ones you are most likely to use. The first creates an empty document, and the second creates a document with one element (in both cases you can of course add more elements).

All the constructors (except the one with allowDuplicateNames) simply call the Add method that takes the same parameters, so refer to the coresponding Add method for details about how the new document is initially populated.

A BsonDocument normally does not allow duplicate names, but if you must allow duplicate names call the constructor with the allowDuplicateNames parameter and pass in true. It is **not** recommended that you allow duplicate names, and this option exists only to allow handling existing BSON documents that might have duplicate names. MongoDB makes no particular guarantees about whether it supports documents with duplicate names, so be cautious about sending any such documents you construct to the server.

Create a new document and call Add and Set methods

This is a traditional step by step method to create and populate a document using multiple C# statements. For example:

```
BsonDocument book = new BsonDocument();
book.Add("author", "Ernest Hemingway");
book.Add("title", "For Whom the Bell Tolls");
```

Create a new document and use the fluent interface Add and Set methods

This is similar to the previous approach but the fluent interface allows you to chain the various calls to Add so that they are all a single C# statement. For example:

```
BsonDocument book = new BsonDocument()
    .Add("author", "Ernest Hemingway")
    .Add("title", "For Whom the Bell Tolls");
```

Create a new document and use C#'s collection initializer syntax (recommended)

This is the recommended way to create and initialize a BsonDocument in one statement. It uses C#'s collection initializer syntax:

```
BsonDocument book = new BsonDocument {
    { "author", "Ernest Hemingway" },
    { "title", "For Whom the Bell Tolls" }
};
```

The compiler translates this into calls to the matching Add method:

```
BsonDocument book = new BsonDocument();
    book.Add("author", "Ernest Hemingway");
    book.Add("title", "For Whom the Bell Tolls");
```

A common mistake is to forget the inner set of braces. This will result in a compilation error. For example:

```
BsonDocument bad = new BsonDocument {
    "author", "Ernest Hemingway"
};
```

is translated by the compiler to:

```
BsonDocument bad = new BsonDocument();
bad.Add("author");
bad.Add("Ernest Hemingway");
```

which results in a compilation error because there is no Add method that takes a single string argument.

Creating nested BSON documents

Nested BSON documents are created by setting the value of an element to a BSON document. For example:

```
BsonDocument nested = new BsonDocument {
    { "name", "John Doe" },
    { "address", new BsonDocument {
        { "street", "123 Main St." },
        { "city", "Centerville" },
        { "state", "PA" },
        { "zip", 12345}
    }}
};
```

This creates a top level document with two elements ("name" and "address"). The value of "address" is a nested BSON document.

Add methods

BsonDocument has the following overloaded Add methods:

- `Add(BsonElement element)`
- `Add(Dictionary<string, object> dictionary)`
- `Add(Dictionary<string, object> dictionary, IEnumerable<string> keys)`
- `Add(IDictionary dictionary)`
- `Add(IDictionary dictionary, IEnumerable<string> keys)`
- `Add(IDictionary<string, object> dictionary)`
- `Add(IDictionary<string, object> dictionary, IEnumerable<string> keys)`
- `Add(IEnumerable<BsonElement> elements)`
- `Add(string name, BsonValue value)`
- `Add(string name, BsonValue value, bool condition)`

It is important to note that sometimes the Add methods **don't** add a new element. If the value supplied is null (or the condition supplied in the last overload is false) then the element isn't added. This makes it really easy to handle optional elements without having to write any if statements or conditional expressions.

For example:

```
BsonDocument document = new BsonDocument {
    { "name", name },
    { "city", city }, // not added if city is null
{ "dob", dob, dobAvailable } // not added if dobAvailable is false
};
```

is more compact and readable than:

```
BsonDocument document = new BsonDocument();
document.Add("name", name);
if (city != null) {
    document.Add("city", city);
}
if (dobAvailable) {
    document.Add("dob", dob);
}
```

If you want to add a BsonNull if a value is missing you have to say so. A convenient way is to use C#'s null coalescing operator as follows:

```
BsonDocument = new BsonDocument {
    { "city", city ?? BsonConstants.Null }
};
```

The IDictionary overloads initialize a BsonDocument from a dictionary. Each key in the dictionary becames the name of a new element, and each value is mapped to a matching BsonValue and becomes the value of the new element. The overload with the keys parameter lets you select which dictionary entries to load (you might also use the keys parameter to control the order in which the elements are loaded from the dictionary).

Accessing BsonDocument elements

The recommended way to access BsonDocument elements is to use one of the following indexers:

- `BsonValue this[int index]`
- `BsonValue this[string name]`
- `BsonValue this[string name, BsonValue defaultValue]`

Note that the return value of the indexers is BsonValue, not BsonElement. This actually makes BsonDocuments much easier to work with (if you ever need to get the actual BsonElements use GetElement).

We've already seen samples of accessing BsonDocument elements. Here are some more:

```
BsonDocument book;
string author = book["author"].AsString;
DateTime publicationDate = book["publicationDate"].AsDateTime;
int pages = book["pages", -1].AsInt32; // default value is -1
```

BsonArray

This class is used to represent BSON arrays. While arrays happen to be represented externally as BSON documents (with a special naming convention for the elements), the BsonArray class is unrelated to the BsonDocument class because they are used very differently.

Constructors

BsonArray has the following constructors:

- `BsonArray()`
- `BsonArray(IEnumerable<bool> values)`
- `BsonArray(IEnumerable<BsonValue> values)`
- `BsonArray(IEnumerable<DateTime> values)`
- `BsonArray(IEnumerable<double> values)`
- `BsonArray(IEnumerable<int> values)`
- `BsonArray(IEnumerable<long> values)`
- `BsonArray(IEnumerable<ObjectId> values)`
- `BsonArray(IEnumerable<string> values)`
- `BsonArray(IEnumerable values)`

All the constructors with a parameter call the matching Add method. The multiple overloads are needed because C# does not provide automatic conversions from IEnumerable<T> to IEnumerable<object>.

Add and AddRange methods

Bson Array has the following Add methods:

- `BsonArray Add(BsonValue value)`
- `BsonArray AddRange(IEnumerable<bool> values)`

- BsonArray AddRange(IEnumerable<BsonValue> values)
- BsonArray AddRange(IEnumerable<DateTime> values)
- BsonArray AddRange(IEnumerable<double> values)
- BsonArray AddRange(IEnumerable<int> values)
- BsonArray AddRange(IEnumerable<long> values)
- BsonArray AddRange(IEnumerable<ObjectId> values)
- BsonArray AddRange(IEnumerable<string> values)
- BsonArray AddRange(IEnumerable values)

Note that the Add method takes a single parameter. To create and initialize a BsonArray with multiple values use any of the following approaches:

```
// traditional approach
BsonArray a1 = new BsonArray();
a1.Add(1);
a2.Add(2);

// fluent interface
BsonArray a2 = new BsonArray().Add(1).Add(2);

// values argument
int[] values = new int[] { 1, 2 };
BsonArray a3 = new BsonArray(values);

// collection initializer syntax
BsonArray a4 = new BsonArray { 1, 2 };
```

Indexer

Array elements are accessed using an integer index. Like BsonDocument, the type of the elements is BsonValue. For example:

```
BsonArray array = new BsonArray { "Tom", 39 };
string name = array[0].AsString;
int age = array[1].AsInt32;
```

The C# Driver

Up until now we have been focusing on the BSON Library. The remainder of this tutorial focuses on the C# Driver.

Thread safety

Only a few of the C# Driver classes are thread safe. Among them: MongoClient, MongoServer, MongoDatabase, MongoCollection and MongoGridFS. Common classes you will use a lot that are not thread safe include MongoCursor and all the classes from the BSON Library (except BsonSymbolTable which is thread safe). A class is not thread safe unless specifically documented as being thread safe.

All static properties and methods of all classes are thread safe.

MongoClient class

This class serves as the root object for working with a MongoDB server. The connections to the server are handled automatically behind the scenes (a connection pool is used to increase efficiency).

When you are connecting to a replica set you will still use only one instance of MongoClient, which represents the replica set as a whole. The driver automatically finds all the members of the replica set and identifies the current primary.

Instances of this class are thread safe.

By default and unless set otherwise, all operations requiring a WriteConcern use w=1. In other words, by default, all write operations will block until the server has acknowledged the write.

Connection strings

The easiest way to connect to a MongoDB server is to use a connection string. The standard connection string format is:

```
mongodb://[username:password@]hostname[:port][/[database][?options]]
```

The username and password should only be present if you are using authentication on the MongoDB server. These credentials will be the default

credentials for all databases. To authenticate against the admin database append "(admin)" to the username. If you are using different credentials with different databases pass the appropriate credentials to the GetDatabase method.

The port number is optional and defaults to 27017.

To connect to multiple servers, specify the seed list by providing multiple hostnames (and port numbers if required) separated by commas. For example:

```
mongodb://server1,server2:27017,server2:27018
```

This connection string specifies a seed list consisting of three servers (two of which are on the same machine but on different port numbers). Because specifying multiple servers is ambiguous as to whether or not it is a replica set or multiple mongos (in a sharded setup), the driver will go through a discovery phase of connecting to the servers to determine their type. This has a little overhead at connection time and can be avoided by specifying a connection mode in the connection string:

```
mongodb://server1,server2:27017,server2:27018/?connect=replicaset
```

The available connection modes are automatic (the default), direct, replica set, and shardrouter. The rules for connection mode are as follows:

1. If a connect mode is specified other than automatic, it is used.
2. If a replica set name is specified on the connection string (replicaset), then replicaset mode is used.
3. If their is only one server listed on the connection string, then direct mode is used.
4. Otherwise, discovery occurs and the first server to respond determines the connection mode.

If you have multiple servers listed, and one is part of a replica set and another is not, then the connection mode is non-deterministic. Be sure that you are not mixing server types on the connection string.

Should the connection mode resolve to a replica set, the driver will find the primary server even if it is not in the seed list, as long as at least one of the servers in the seed list responds (the response will contain the full replica set and the name of the current primary). In addition, other secondaries will also be discovered and added (or removed) into the mix automically, even after initial connection. This will enable you to add and remove servers from the replica set and the driver will handle the changes automatically.

As alluded to above, the options part of the connection string is used to set various connection options. Suppose you wanted to connect directly to a member of a replica set regardless of whether it was the current primary or not (perhaps to monitor its status or to issue read only queries against it). You could use:

```
mongodb://server2/?connect=direct;readpreference=nearest
```

The full documentation for connection strings can be found at:

http://www.mongodb.org/display/DOCS/Connections

and read preferences at:

http://docs.mongodb.org/manual/applications/replication/#replica-set-read-preference

SSL Support

Support for SSL is baked into the driver. You can configure this via the connection string be adding an "ssl=true" option to the options.

```
mongodb://server2/?ssl=true
```

By default, the server certificate will get validated against the local trusted certificate store. This sometimes causes issues in test environments where test servers don't have signed certs. To alleviate this issue, you can also add an "sslverifycertificate=false" as another connection string option to ignore any certificate errors.

Authentication

MongoDB supports a simple and straightforward authentication mechanism. You can read about it on the security and authentication docs page.

The C# driver supports authentication in a couple of ways. As noted above in connection strings, you can specify default credentials on the connection string. The default credentials are always used as a fallback if no other credentials are supplied.

Supplying credentials can be done in two ways. First, they can be supplied to certain methods at runtime. These credentials will then be used to execute the desired functionality. The other, and more robust way, is to store credentials in a MongoCredentialsStore. MongoCredentials in the

store are keyed by database, so if different databases require different users, then the credentials store is consulted first and, upon a miss, will fallback to the default credentials supplied on the connection string if they exist.

The example below uses the credential store to define admin credentials and credentials for the "foo" database. Access to databases other than "admin" or "foo" will use the connection string supplied default credentials "test".

```
var url = new MongoUrl("mongodb://test:user@localhost:27017");
var settings = MongoClientSettings.FromUrl(url);
var adminCredentials = new MongoCredentials("admin", "user", true);
settings.CredentialsStore.Add("admin", adminCredentials);
var fooCredentials = new MongoCredentials("foo", "user", false);
settings.CredentialsStore.Add("foo", fooCredentials);

var client = new MongoClient(settings);
```

GetServer method

You can navigate from an instance of a MongoClient to an instance of MongoServer by using the GetServer method.

MongoServer class

The MongoServer class is used to provide more control over the driver. It contains advanced ways of getting a database and pushing a sequence of operations through a single socket in order to guarantee consistency.

GetDatabase method

You can navigate from an instance of MongoServer to an instance of MongoDatabase (see next section) using one of the following GetDatabase methods or indexers:

- `MongoDatabase GetDatabase(MongoDatabaseSettings settings)`
- `MongoDatabase GetDatabase(string databaseName)`
- `MongoDatabase GetDatabase(string databaseName, MongoCredentials credentials)`
- `MongoDatabase GetDatabase(string databaseName, MongoCredentials credentials, WriteConcern writeConcern)`
- `MongoDatabase GetDatabase(string databaseName, WriteConcern writeConcern)`

Sample code:

```
MongoClient client = new MongoClient(); // connect to localhost
MongoServer server = client.GetServer();
MongoDatabase test = server.GetDatabase("test");
MongoCredentials credentials = new MongoCredentials("username", "password");
MongoDatabase salaries = server.GetDatabase("salaries", credentials);
```

Most of the database settings are inherited from the server object, and the provided overloads of GetDatabase let you override a few of the most commonly used settings. To override other settings, call CreateDatabaseSettings and change any settings you want before calling GetDatabase, like this:

```
var databaseSettings = server.CreateDatabaseSettings("test");
databaseSettings.SlaveOk = true;
var database = server.GetDatabase(databaseSettings);
```

GetDatabase maintains a table of MongoDatabase instances it has returned before, so if you call GetDatabase again with the same parameters you get the same instance back again.

RequestStart/RequestDone methods

Sometimes a series of operations needs to be performed on the same connection in order to guarantee correct results. This is rarely the case, and most of the time there is no need to call RequestStart/RequestDone. An example of when this might be necessary is when a series of Inserts are called in rapid succession with a WriteConcern of w=0, and you want to query that data is in a consistent manner immediately thereafter (with a WriteConcern of w=0, the writes can queue up at the server and might not be immediately visible to other connections). Using RequestStart you can force a query to be on the same connection as the writes, so the query won't execute until the server has caught up with the writes.

A thread can temporarily reserve a connection from the connection pool by using RequestStart and RequestDone. For example:

```
using(server.RequestStart(database)) {
// a series of operations that must be performed on the same connection
}
```

The database parameter simply indicates some database which you intend to use during this request. This allows the server to pick a connection that is already authenticated for that database (if you are not using authentication then this optimization won't matter to you). You are free to use any other databases as well during the request.

RequestStart increments a counter (for this thread) which is decremented upon completion. The connection that was reserved is not actually returned to the connection pool until the count reaches zero again. This means that calls to RequestStart can be nested and the right thing will happen.

RequestStart returns an IDisposable. If you do no use RequestStart with a using block, it is imperative that RequestDone be called in order to release the connection.

Other properties and methods

For a reference of other properties and method, see the api documentation.

MongoDatabase class

This class represents a database on a MongoDB server. Normally there will be only one instance of this class per database, unless you are using different settings to access the same database, in which case there will be one instance for each set of settings.

Instances of this class are thread safe.

GetCollection method

This method returns an object representing a collection in a database. When we request a collection object, we also specify the default document type for the collection. For example:

```
MongoDatabase hr = server.GetDatabase("hr");
MongoCollection<Employee> employees =
    hr.GetCollection<Employee>("employees");
```

A collection is not restricted to containing only one kind of document. The default document type simply makes it more convenient to work with that kind of document, but you can always specify a different kind of document when required.

Most of the collection settings are inherited from the database object, and the provided overloads of GetCollection let you override a few of the most commonly used settings. To override other settings, call CreateCollectionSettings and change any settings you want before calling GetCollection , like this:

```
var collectionSettings = database.CreateCollectionSettings<TDocument>("test");
collectionSettings.SlaveOk = true;
var collection = database.GetCollection(collectionSettings);
```

GetCollection maintains a table of instances it has returned before, so if you call GetCollection again with the same parameters you get the same instance back again.

Other properties and methods

For a reference of other properties and method, see the api documentation.

MongoCollection<TDefaultDocument> class

This class represents a collection in a MongoDB database. The <TDefaultDocument> type parameter specifies the type of the default document for this collection.

Instances of this class are thread safe.

Insert<TDocument> method

To insert a document in the collection create an object representing the document and call Insert. The object can be an instance of

BsonDocument or of any class that can be successfully serialized as a BSON document. For example:

```
MongoCollection<BsonDocument> books =
    database.GetCollection<BsonDocument>("books");
BsonDocument book = new BsonDocument {
    { "author", "Ernest Hemingway" },
    { "title", "For Whom the Bell Tolls" }
};
books.Insert(book);
```

If you have a class called Book the code might look like:

```
MongoCollection<Book> books = database.GetCollection<Book>("books");
Book book = new Book {
    Author = "Ernest Hemingway",
    Title = "For Whom the Bell Tolls"
};
books.Insert(book);
```

InsertBatch method

You can insert more than one document at a time using the InsertBatch method. For example:

```
MongoCollection<BsonDocument> books;
BsonDocument[] batch = {
    new BsonDocument {
        { "author", "Kurt Vonnegut" },
        { "title", "Cat's Cradle" }
    },
    new BsonDocument {
        { "author", "Kurt Vonnegut" },
        { "title", "Slaughterhouse-Five" }
    }
};
books.InsertBatch(batch);
```

When you are inserting multiple documents InsertBatch can be much more efficient than Insert.

FindOne and FindOneAs methods

To retrieve documents from a collection use one of the various Find methods. FindOne is the simplest. It returns the first document it finds (when there are many documents in a collection you can't be sure which one it will be). For example:

```
MongoCollection<Book> books;
Book book = books.FindOne();
```

If you want to read a document that is not of the <TDefaultDocument> type use the FindOneAs method, which allows you to override the type of the returned document. For example:

```
MongoCollection<Book> books;
BsonDocument document = books.FindOneAs<BsonDocument>();
```

In this case the default document type of the collection is Book, but we are overriding that and specifying that the result be returned as an instance of BsonDocument.

Find and FindAs methods

The Find and FindAs methods take a query that tells the server which documents to return. The query parameter is of type IMongoQuery. IMongoQuery is a marker interface that identifies classes that can be used as queries. The most common ways to construct a query are to either use the Query builder class or to create a QueryDocument yourself (a QueryDocument is a subclass of BsonDocument that also implements IMongoQuery and can therefore be used as a query object). Also, by using the QueryWrapper class the query can be of any type that can be successfully serialized to a BSON document, but it is up to you to make sure that the serialized document represents a valid query object.

One way to query is to create a QueryDocument object yourself:

```
MongoCollection<BsonDocument> books;
var query = new QueryDocument("author", "Kurt Vonnegut");
foreach (BsonDocument book in books.Find(query)) {
    // do something with book
}
```

Another way to query is to use the Query Builder (recommended):

```
MongoCollection<BsonDocument> books;
var query = Query.EQ("author", "Kurt Vonnegut");
foreach (BsonDocument book in books.Find(query)) {
    // do something with book
}
```

Yet another way to query is to use an anonymous class as the query, but in this case we must wrap the anonymous object:

```
MongoCollection<BsonDocument> books;
var query = Query.Wrap(new { author = "Kurt Vonnegut" });
foreach (BsonDocument book in books.Find(query)) {
    // do something with book
}
```

If you want to read a document of a type that is not the default document type use the FindAs method instead:

```
MongoCollection<BsonDocument> books;
var query = Query<Book>.EQ(b => b.Author, "Kurt Vonnegut");
foreach (Book book in books.FindAs<Book>(query)) {
    // do something with book
}
```

Save<TDocument> method

The Save method is a combination of Insert and Update. If the Id member of the document has a value, then it is assumed to be an existing document and Save calls Update on the document (setting the Upsert flag just in case it actually is a new document after all). Otherwise it is assumed to be a new document and Save calls Insert after first assigning a newly generated unique value to the Id member.

For example, you could correct an error in the title of a book using:

```
MongoCollection<BsonDocument> books;
var query = Query.And(
    Query.EQ("author", "Kurt Vonnegut"),
    Query.EQ("title", "Cats Craddle")
);
BsonDocument book = books.FindOne(query);
if (book != null) {
    book["title"] = "Cat's Cradle";
    books.Save(book);
}
```

The TDocument class must have an Id member to be used with the Save method. If it does not you can call Insert instead of Save to insert the document.

Update method

The Update method is used to update existing documents. The code sample shown for the Save method could also have been written as:

```
MongoCollection<BsonDocument> books;
var query = new QueryDocument {
    { "author", "Kurt Vonnegut" },
    { "title", "Cats Craddle" }
};
var update = new UpdateDocument {
    { "$set", new BsonDocument("title", "Cat's Cradle") }
};
BsonDocument updatedBook = books.Update(query, update);
```

or using Query and Update builders:

```
MongoCollection<BsonDocument> books;
var query = Query.And(
    Query.EQ("author", "Kurt Vonnegut"),
    Query.EQ("title", "Cats Craddle")
);
var update = Update.Set("title", "Cat's Cradle");
BsonDocument updatedBook = books.Update(query, update);
```

FindAndModify method

Use FindAndModify when you want to find a matching document and update it in one atomic operation. FindAndModify always updates a single document, and you can combine a query that matches multiple documents with a sort criteria that will determine exactly which matching document is updated. In addition, FindAndModify will return the matching document (either as it was before the update or after) and if you wish you can specify which fields of the matching document to return.

Using the example documented here:

http://www.mongodb.org/display/DOCS/findAndModify+Command

the call to FindAndModify would be written in C# as:

```
var jobs = database.GetCollection("jobs");
var query = Query.And(
    Query.EQ("inprogress", false),
    Query.EQ("name", "Biz report")
);
var sortBy = SortBy.Descending("priority");
var update = Update.
    .Set("inprogress", true)
    .Set("started", DateTime.UtcNow);
var result = jobs.FindAndModify(
    query,
    sortBy,
    update,
    true // return new document
);
var chosenJob = result.ModifiedDocument;
```

MapReduce method

Map/Reduce is a way of a aggregating data from a collection. Every document in a collection (or some subset if an optional query is provided) is sent to the map function, which calls emit to produce intermediate values. The intermediate values are then sent to the reduce function to be aggregated.

This example is taken from page 87 of MongoDB: The Definitive Guide, by Kristina Chodorow and Michael Dirolf. It counts how many times each key is found in a collection.

```
var map =
    "function() {" +
    "    for (var key in this) {" +
    "        emit(key, { count : 1 });" +
    "    }" +
    "}";

var reduce =
    "function(key, emits) {" +
    "    total = 0;" +
    "    for (var i in emits) {" +
    "        total += emits[i].count;" +
    "    }" +
    "    return { count : total };" +
    "}";

var mr = collection.MapReduce(map, reduce);
foreach (var document in mr.GetResults()) {
    Console.WriteLine(document.ToJson());
}
```

Other properties and methods

For a reference of other properties and method, see the api documentation.

MongoCursor<TDocument> class

The Find method (and its variations) don't immediately return the actual results of a query. Instead they return a cursor that can be enumerated to retrieve the results of the query. The query isn't actually sent to the server until we attempt to retrieve the first result (technically, when MoveNext is called for the first time on the enumerator returned by GetEnumerator). This means that we can control the results of the query in interesting ways by modifying the cursor before fetching the results.

Instances of MongoCursor are not thread safe, at least not until they are frozen (see below). Once they are frozen they are thread safe because they are read-only (in particular, GetEnumerator is thread safe so the same cursor *could* be used by multiple threads).

Enumerating a cursor

The most convenient way to consume the results of a query is to use the C# foreach statement. For example:

```
var query = Query.EQ("author", "Ernest Hemingway");
var cursor = books.Find(query);
foreach (var book in cursor) {
    // do something with book
}
```

You can also use any of the extensions methods defined by LINQ for IEnumerable<T> to enumerate a cursor:

```
var query = Query.EQ("author", "Ernest Hemingway");
var cursor = books.Find(query);
var firstBook = cursor.FirstOrDefault();
var lastBook = cursor.LastOrDefault();
```

In the above example, the query is actually sent to the server twice (once when FirstOrDefault is called and again when LastOrDefault is called).

It is important that a cursor cleanly release any resources it holds. The key to guaranteeing this is to make sure the Dispose method of the enumerator is called. The foreach statement and the LINQ extension methods all guarantee that Dispose will be called. Only if you enumerate the cursor manually are you responsible for calling Dispose.

Modifying a cursor before enumerating it

A cursor has several properties that can be modified before it is enumerated to control the results returned. There are two ways to modify a cursor:

1. modify the properties directly
2. use the fluent interface to set the properties

For example, if we want to skip the first 100 results and limit the results to the next 10, we could write:

```
var query = Query.EQ("status", "pending");
var cursor = tasks.Find(query);
cursor.Skip = 100;
cursor.Limit = 10;
foreach (var task in cursor) {
    // do something with task
}
```

or using the fluent interface:

```
var query = Query.EQ("status", "pending");
foreach (var task in tasks.Find(query).SetSkip(100).SetLimit(10)) {
    // do something with task
}
```

The fluent interface works well when you are setting only a few values. When setting more than a few you might prefer to use the properties approach.

Once you begin enumerating a cursor it becomes "frozen" and you can no longer change any of its properties. So you must set all the properties before you start enumerating it.

Modifiable properties of a cursor

The following properties of a cursor are modifiable:

- `BatchSize` (`SetBatchSize`)
- `Fields` (`SetFields`)
- `Flags` (`SetFlags`)
- `Limit` (`SetLimit`)
- `Options` (`SetOption` and `SetOptions`)
- `SerializationOptions` (`SetSerializationOptions`)
- `Skip` (`SetSkip`)
- `SlaveOk` (`SetSlaveOk`)

The method names in parenthesis are the corresponding fluent interface methods.

The fluent interface also supports additional options that aren't used very frequently and are not exposed as properties:

- `SetHint`
- `SetMax`
- `SetMaxScan`
- `SetMin`
- `SetShowDiskLoc`
- `SetSnapshot`
- `SetSortOrder`

Other methods

MongoCursor has a few methods used for some special purpose operations:

- `Clone`
- `Count`
- `Explain`
- `Size`

WriteConcern class

There are various levels of WriteConcern, and this class is used to represent those levels. WriteConcern applies only to operations that don't already return a value (so it doesn't apply to queries or commands). It applies to the following MongoCollection methods: Insert, Remove, Save and Update.

The gist of WriteConcern is that after an Insert, Remove, Save or Update message is sent to the server it is followed by a GetLastError command so the driver can verify that the operation succeeded. In addition, when using replica sets it is possible to verify that the information has been replicated to some minimum number of secondary servers.

CSharp getLastError and SafeMode

In the C# driver SafeMode can be set at different levels.

1. At the server level via the connection string:

```
var connectionString = "mongodb://hostname/?safe=true;w=2;wtimeout=30s";
var server = MongoServer.Create(connectionString);
```

2. At the database level:

```
var safemode = SafeMode.W2; // default timeout
// or
var safeMode = SafeMode.Create(2, TimeSpan.FromSeconds(30)); // 30 second timeout
var database = server.GetDatabase("test", safeMode);
```

3. At the collection level:

```
var collection = database.GetCollection("test", safeMode);
```

4. At the operation level:

```
var safeModeResult = collection.Insert(document, safeMode);
```

Each level inherits the setting from the level above it unless overridden.

getLastError is called automatically when any SafeMode other than false is used. An exception is thrown if there was an error, otherwise the SafeModeResult has the information returned by GetLastError.

See Also

- Connections
- getLastError Command
- Replica Set Design Concepts

Erlang Language Center

- Driver Download
  - https://github.com/mongodb/mongodb-erlang
- API Docs
- Design of the Erlang Driver post on blog.mongodb.org

Third Party Frameworks and Libs

- Mongrel - A record/document mapper that maps Erlang records to MongoDB documents
  - API Documentation for Mongrel

Tools and Libraries

- Talend Adapters

Driver Syntax Table

The wiki generally gives examples in JavaScript, so this chart can be used to convert those examples to any language.

JavaScript	Python	PHP	Ruby
[]	[]	`array()`	[]
{}	{}	`new stdClass`	{}
{ x : 1 }	{"x": 1}	`array('x' => 1)`	{'x' => 1}
`connect("www.example.net")`	`Connection("www.example.net")`	`new Mongo("www.example.net")`	`Connection.new("www.exa`
`cursor.next()`	`cursor.next()`	`$cursor->getNext()`	`cursor.next_document()`
`cursor.hasNext()`	*	`$cursor->hasNext()`	`cursor.has_next?`
`collection.findOne()`	`collection.find_one()`	`$collection->findOne()`	`collection.find_one()`
`db.eval()`	`db.eval()`	`$db->execute()`	`db.eval()`

* does not exist in that language

Javascript Language Center

MongoDB can be

- Used by clients written in Javascript;
- Uses Javascript internally server-side for certain options such as map/reduce;
- Has a shell that is based on Javascript for administrative purposes.

node.JS and V8

See the node.JS page.

SpiderMonkey

The MongoDB shell extends SpiderMonkey. See the MongoDB shell documentation.

Narwhal

- http://github.com/sergi/narwhal-mongodb

MongoDB Server-Side Javascript

Javascript may be executed in the MongoDB server processes for various functions such as query enhancement and map/reduce processing. See Server-side Code Execution.

Node.js

Node.js is used to write event-driven, scalable network programs in server-side JavaScript. It is similar in purpose to Twisted, EventMachine, etc. It runs on Google's V8.

This is an overview of the available tools and suggested practices for using Node.js with MongoDB. Those wishing to skip to more detailed discussion should check out the The Node.js Driver Manual.

- Node.js Driver
  - Installing / Upgrading
- Object Mappers
  - Mongoose
- Other notable projects
- Presentations
- Tutorials
- 3rd Party Drivers

Node.js Driver

The MongoDB Node.js driver is the 10gen-supported driver for MongoDB. In spring 2012 10gen officially adopted the popular Node MongoDB

Native Project and sponsored the maintainer, Christian Kvalheim, to continue its development. It's written in pure javascript and provides a native asyncronous node.js interface to MongoDB. The driver is optimized for simplicity. It can be used on its own, but it also serves as the basis of several object mapping libraries, such as Mongoose.

- Tutorial
- Node.js Driver README
- Source Code

Installing / Upgrading

The easiest way to install is to use npm:

```
$ npm install mongodb
```

Object Mappers

Because MongoDB is so easy to use, the basic Node.js driver can be the best solution for many applications. However, if you need validations, associations, and other high-level data modeling functions then an Object Document Mapper may be helpful.

Mongoose

Mongoose is the 10gen-supported ODM for Node.js. It has a thriving open source community and includes advanced schema-based features such as async validation, casting, object life-cycle management, pseudo-joins, and rich query builder support.

Install it easily with npm:

```
$ npm install mongoose
```

- Quickstart Tutorial
- Source Code
- Google Group
- Bug reports
- irc: #mongoosejs on freenode

Other notable projects

- Mongoskin - The future layer for node-mongodb-native.
- Mongolia - Lightweight MongoDB ORM/Driver Wrapper.
- Mongojs - Somewhat mimics the MongoDB shell api.

Each of these projects build on top of the native Node.js driver and so some knowledge of that is useful, especially if you work with a custom Mongo configuration.

Presentations

- An introduction to the mongo node.js driver - June 2011
- Using MongoDB with node.js - June 2011
- Node.js & MongoDB - Webinar June 2011
- A beautiful marriage: MongoDB and node.js - MongoNYC June 2011
- Rapid Realtime App Development with Node.JS & MongoDB - MongoSF May 2011

Tutorials

Mongoskin tutorial
Mongoose Tutorial

3rd Party Drivers

A few 3rd party drivers exist. While not officially supported by 10gen, these drivers take a different approach that may be valuable given your needs.

- node-mongodb - Async Node interface to MongoDB (written in C)
- Mongolian DeadBeef - A node.js driver that attempts to closely approximate the MongoDB shell.

JVM Languages

Redirection Notice
This page should redirect to Java Language Center in about 3 seconds.

Python Language Center

This is an overview of the available tools for using Python with MongoDB. Those wishing to skip to more detailed discussion should check out the Python Driver Tutorial.

- Python Driver
- Python tools
  - ORM Like Layers
  - Framework Tools
- Alternative drivers
- Presentations
- Tutorials

Python Driver

PyMongo is the recommended way to work with MongoDB from Python.

- Installation
- Tutorial
- API Documentation
- Changelog
- Source Code

Python tools

ORM Like Layers

Because MongoDB is so easy to use the basic Python driver is often the best solution for many applications. However, if you need data validation, associations and other high-level data modeling functionality then ORM like layers may be desired.

- ORM like layers

Framework Tools

Several tools and adapters for integration with various Python frameworks and libraries also exist.

- Framework Tools

Alternative drivers

- Alternative driver list

Presentations

- MongoDB & Python - Workshop materials from PyCon 2012
- PyCon Poster - PyCon 2012
- Realtime Analytics using MongoDB, Python, Gevent, and ZeroMQ - Rick Copeland's presentation from Mongo Seattle (December 2011)
- MongoDB with Python, Pylons, and Pyramid - Niall O'Higgins' presentation from MongoSF (May 2011)
- Python Development with MongoDB - Bernie Hackett's presentation from MongoSF (May 2011)
- Building a Social Graph with MongoDB at Eventbrite - Brian Zambrano's presentation from MongoSV (December 2010)
- More Python-related presenatations

Tutorials

- Python Driver Tutorial
- Write a Tumblelog Application with Django MongoDB Engine
- Write a Tumblelog Application with Flask and MongoEngine

PHP Language Center

Using MongoDB in PHP

To access MongoDB from PHP you will need:

- The MongoDB server running - the server is the "mongo**d**" file, not the "mongo" client (note the "d" at the end)
- The MongoDB PHP driver installed

Installing the PHP Driver

***NIX**

Run:

```
sudo pecl install mongo
```

Open your php.ini file and add to it:

```
extension=mongo.so
```

It is recommended to add this to the section with the other "extensions", but it will work from anywhere within the php.ini file.

Restart your web server (Apache, nginx, etc.) for the change to take effect.

See the installation docs for configuration information and OS-specific installation instructions.

Note

`pecl` requires that pear be installed. For those using apt-get, you may need to run the following:

```
sudo apt-get install php5-dev php5-cli php-pear
```

Windows

- Download the correct driver for your environment from http://github.com/mongodb/mongo-php-driver/downloads. Thread safe is for running PHP as an Apache module (typical installation), non-thread safe is for CGI
- Unzip and add the php_mongo.dll file to your PHP extensions directory (usually the "ext" folder in your PHP installation.)
- Add to your php.ini:

  ```
  extension=php_mongo.dll
  ```

- Restart your web server (Apache, IIS, etc.) for the change to take effect

For more information, see the Windows section of the installation docs.

Using the PHP Driver

To get started, see the Tutorial. Also check out the API Documentation.

See Also

- PHP Libraries, Frameworks, and Tools for working with Drupal, Cake, Symfony, and more from MongoDB.
- Admin UIs
- If you are using Eclipse, you can get Content Assist working by downloading the mongo_version.zip package.
- MongoDB for the PHP Mind Blog Series
  - Part 1: Getting Started
  - Part 2: Queries and Indexes
  - Part 3: Data Modeling

Installing the PHP Driver

Redirection Notice
This page should redirect to http://www.php.net/manual/en/mongo.installation.php.

PHP Libraries, Frameworks, and Tools

- Libraries and Frameworks
  - CakePHP
  - Codeigniter
  - Doctrine
  - Drupal
  - Fat-Free Framework
  - Kohana Framework
  - Lithium
  - Symfony 2
  - TechMVC
  - Thundergrid
  - Vork
  - Yii
  - Zend Framework
- Stand-Alone Tools
  - Autocomplete for IDEs
  - ActiveMongo
  - Comfi
  - MapReduce API
  - Mongofilesystem
  - Mandango
  - MongoDB Pagination
  - MongoDb PHP ODM
  - Mongodloid
  - MongoQueue
  - MongoRecord
  - Morph
  - simplemongophp
  - TURBOPY
- Blogs & HOWTOs
  - How to batch import JSON data output from FFprobe for motion stream analysis

The PHP community has created a huge number of libraries to make working with MongoDB easier and integrate it with existing frameworks.

Libraries and Frameworks

CakePHP

- MongoDB datasource for CakePHP. There's also an introductory blog post on using it with Mongo.

Codeigniter

- MongoDB-Codeigniter-Driver

Doctrine

ODM (Object Document Mapper) is an experimental Doctrine MongoDB object mapper. The Doctrine\ODM\Mongo namespace is an experimental project for a PHP 5.3 MongoDB Object Mapper. It allows you to easily write PHP 5 classes and map them to collections in MongoDB. You just work with your objects like normal and Doctrine will transparently persist them to Mongo.

This project implements the same "style" of the Doctrine 2 ORM project interface so it will look very familiar to you and it has lots of the same features and implementations.

- Documentation - API, Reference, and Cookbook
- Official blog post
- Screencast
- Blog post on using it with Symfony
- Bug tracker

Drupal

- MongoDB Integration - Views (query builder) backend, a watchdog implementation (logging), and field storage.

Fat-Free Framework

Fat-Free is a powerful yet lightweight PHP 5.3+ Web development framework designed to help you build dynamic and robust applications - fast!

Kohana Framework

- Mango at github
  An ActiveRecord-like library for PHP, for the Kohana PHP Framework.
  See also PHP Language Center#MongoDb PHP ODM further down.

Lithium

Lithium supports Mongo out-of-the-box.

- Tutorial on creating a blog backend.

Symfony 2

- Symfony 2 Logger
  A centralized logger for Symfony applications. See the blog post.
- sfMongoSessionStorage - manages session storage via MongoDB with symfony.
- sfStoragePerformancePlugin - This plugin contains some extra storage engines (MongoDB and Memcached) that are currently missing from the Symfony (>= 1.2) core.

TechMVC

An extensive MVC 2 based PHP framework which supports MongoDB directly with only PHPMongo extension. Hosted at http://sourceforge.net/projects/techmvc/ and demo example available at http://demo.techmvc.techunits.com/.

Thundergrid

A GridFS Framework for PHP

Thundergrid is a simple framework written in PHP that allows you to rapidly store files in your Mongo database using the GridFS specification.

Using Thundergrid gives you the ability to control exactly how you use GridFS in your scripts, it allows you to list, filter and display objects within GridFS quickly and rapidly.

Vork

Vork, the high-performance enterprise framework for PHP natively supports MongoDB as either a primary datasource or used in conjunction with an RDBMS. Designed for scalability & Green-IT, Vork serves more traffic with fewer servers and can be configured to operate without any disk-IO.

Vork provides a full MVC stack that outputs semantically-correct XHTML 1.1, complies with Section 508 Accessibility guidelines & Zend-Framework coding-standards, has SEO-friendly URLs, employs CSS-reset for cross-browser display consistency and is written in well-documented object-oriented E_STRICT PHP5 code.

An extensive set of tools are built into Vork for ecommerce (cc-processing, SSL, PayPal, AdSense, shipment tracking, QR-codes), Google Maps, translation & internationalization, Wiki, Amazon Web Services, Social-Networking (Twitter, Meetup, ShareThis, YouTube, Flickr) and much more.

Yii

- YiiMongoDbSuite is an almost complete, ActiveRecord like support for MongoDB in Yii It originally started as a fork of MongoRecord extension written by tyohan, to fix some major bugs, and add full featured suite for MongoDB developers.

Zend Framework

- Shanty Mongo is a prototype mongodb adapter for the Zend Framework. It's intention is to make working with mongodb documents as natural and as simple as possible. In particular allowing embeded documents to also have custom document classes.
- ZF Cache Backend
  A ZF Cache Backend for MongoDB. It support tags and auto-cleaning.
- There is a Zend_Nosql_Mongo component proposal.

Stand-Alone Tools

Autocomplete for IDEs

PeclMongoPhpDoc gives IDEs information about the Pecl extension to help with autocomplete & docs.

ActiveMongo

ActiveMongo is a really simple ActiveRecord for MongoDB in PHP.

There's a nice introduction to get you started at http://crodas.org/activemongo.php.

Comfi

Comfi is a Mongo PHP ORM (part of Comfi.com Homebase Framework)

MapReduce API

A MapReduce abstraction layer. See the blog post.

- MongoDB-MapReduce-PHP at github

Mongofilesystem

Filesystem based on MongoDB GridFS. Mongofilesystem will help you use MongoDB GridFS like a typical filesystem, using the familiar PHP commands.

Mandango

Mandango is a simple, poweful and ultrafast Object Document Mapper (ODM) for PHP and MongoDB..

MongoDB Pagination

PHP MongoDB Pagination is the pagination plugin for MongoDB released under MIT License. Simple to install & use. It has been developed under TechMVC 3.0.4, but it's compatible with any 3rd party framework (e.g. Zend (tested)).

MongoDb PHP ODM

MongoDb PHP ODM is a simple object wrapper for the Mongo PHP driver classes which makes using Mongo in your PHP application more like ORM, but without the suck. It is designed for use with Kohana 3 but will also integrate easily with any PHP application with almost no additional effort.

Mongodloid

A nice library on top of the PHP driver that allows you to make more natural queries (`$query->query('a == 13 AND b >= 8 && c % 3 == 4');`), abstracts away annoying $-syntax, and provides getters and setters.

- Project Page
- Downloads
- Documentation

MongoQueue

MongoQueue is a PHP queue that allows for moving tasks and jobs into an asynchronous process for completion in the background. The queue is managed by Mongo

MongoQueue is an extraction from online classifieds site Oodle. Oodle uses MongoQueue to background common tasks in order to keep page response times low.

MongoRecord

MongoRecord is a PHP Mongo ORM layer built on top of the PHP Mongo PECL extension

MongoRecord is an extraction from online classifieds site Oodle. Oodle's requirements for a manageable, easy to understand interface for dealing with the super-scalable Mongo datastore was the primary reason for MongoRecord. It was developed to use with PHP applications looking to add Mongo's scaling capabilities while dealing with a nice abstraction layer.

Morph

A high level PHP library for MongoDB. Morph comprises a suite of objects and object primitives that are designed to make working with MongoDB in PHP a breeze.

- Morph at github

simplemongophp

Very simple layer for using data objects see blog post

- simplemongophp at github

TURBOPY

TURBOPY is a cloud based content managemant framework + IDE using new editing concepts working with mongodb.

Blogs & HOWTOs

How to batch import JSON data output from FFprobe for motion stream analysis

FFprobe is a stream analyzer that optionally reports in JSON. This example is a PHP script that reads JSON from STDIN, makes an object using json_decode, and inserts the object into a MongDB database. This script could be used with any program that outputs a JSON stream. A bash script will be used to batch process all files within the current directory. For example, the data may be used for analysis and logging of a day's shoot.

- Batch import Multimedia Stream Data into MongoDB with FFprobe web site
- Code and Sample Output at github

PHP - Storing Files and Big Data

Redirection Notice
This page should redirect to http://www.php.net/manual/en/class.mongogridfs.php.

Troubleshooting the PHP Driver

Redirection Notice
This page should redirect to http://www.php.net/manual/en/mongo.trouble.php.

Ruby Language Center

This is an overview of the available tools and suggested practices for using Ruby with MongoDB. Those wishing to skip to more detailed discussion should check out the Ruby Driver Tutorial, Getting started with Rails or Rails 3, and MongoDB Data Modeling and Rails. There are also a number of good external resources worth checking out.

- Ruby Driver
  - Installing / Upgrading
  - BSON
- Object Mappers
- Notable Projects
- Presentations

Ruby Driver

Install the bson_ext gem for any performance-critical applications.

The MongoDB Ruby driver is the 10gen-supported driver for MongoDB. It's written in pure Ruby, with a recommended C extension for speed. The driver is optimized for simplicity. It can be used on its own, but it also serves as the basis of several object mapping libraries, such as MongoMapper.

- Tutorial
- Documentation
- Source Code

Installing / Upgrading

The ruby driver is hosted at Rubygems.org. Before installing the driver, make sure you're using the latest version of rubygems:

```
$ gem update --system
```

Then install the gems:

```
$ gem install mongo
```

To stay on the bleeding edge, you can use the latest source from github:

Using your Gemfile:

```
gem 'mongo', :git => 'git://github.com/mongodb/mongo-ruby-driver.git'
```

Or Manually:

```
$ git clone git://github.com/mongodb/mongo-ruby-driver.git
$ cd mongo-ruby-driver/
$ gem build bson.gemspec; gem install bson-x.x.x.gem
$ gem build bson_ext.gemspec; gem install bson_ext-x.x.x.gem
$ gem build mongo.gemspec; gem install mongo-x.x.x.gem
```

BSON

In versions of the Ruby driver prior to 0.20, the code for serializing to BSON existed in the mongo gem. Now, all BSON serialization is handled by the required bson gem.

```
gem install bson
```

For significantly improved performance, install the bson_ext gem. Using compiled C instead of Ruby, this gem speeds up BSON serialization greatly.

```
gem install bson_ext
```

If you're running on Windows, you'll need the Ruby DevKit installed in order to compile the C extensions.

As long it's in Ruby's load path, `bson_ext` will be loaded automatically when you require `bson`.

Note that beginning with version 0.20, the mongo_ext gem is no longer used.

To learn more about the Ruby driver, see the Ruby Tutorial.

Object Mappers

Because MongoDB is so easy to use, the basic Ruby driver can be the best solution for many applications.

But if you need validations, associations, and other high-level data modeling functions then an Object Document Mapper may be needed.

In the context of a Rails application these provide functionality equivalent to, but distinct from, ActiveRecord. Because Mongo is a document-based database, these mappers are called Object Document Mappers (ODM) as opposed to Object Relational Mappers (ORM).

Several mappers are available:

- MongoMapper from John Nunemaker
- Mongoid from Durran Jordan
- Mongomatic from Ben Myles
- MongoODM from Carlos Paramio
- MongoModel from Sam Pohlenz
- DriverAPILayer from Alexey Petrushin

All the mappers build on top of the basic Ruby driver and so some knowledge of that is useful, especially if you work with a custom Mongo configuration.

Notable Projects

Tools for working with MongoDB in Ruby are being developed daily. A partial list can be found in the Projects and Libraries section of our external resources page.

If you're working on a project that you'd like to have included, let us know.

Presentations

> *Introduction to MongoDB and Ruby (September 2012)*
> *MongoDB & Ruby Presentations*

Ruby Tutorial

Redirection Notice
This page should redirect to http://api.mongodb.org/ruby/current/file.TUTORIAL.html.

This tutorial gives many common examples of using MongoDB with the Ruby driver. If you're looking for information on data modeling, see MongoDB Data Modeling and Rails. Links to the various object mappers are listed on our object mappers page.

Interested in GridFS? Checkout GridFS in Ruby.

As always, the latest source for the Ruby driver can be found on github.

- Installation
- A Quick Tour
  - Using the RubyGem
  - Making a Connection
  - Listing All Databases
  - Dropping a Database
  - Authentication (Optional)
  - Getting a List Of Collections
  - Getting a Collection
  - Inserting a Document
  - Updating a Document
  - Finding the First Document In a Collection using `find_one()`
  - Adding Multiple Documents
  - Counting Documents in a Collection
  - Using a Cursor to get all of the Documents
  - Getting a Single Document with a Query
  - Getting a Set of Documents With a Query
  - Selecting a subset of fields for a query
  - Querying with Regular Expressions
  - Creating An Index
  - Creating and querying on a geospatial index
  - Getting a List of Indexes on a Collection
  - Database Administration
- See Also

Installation

The mongo-ruby-driver gem is served through Rubygems.org. To install, make sure you have the latest version of rubygems.

```
gem update --system
```

Next, install the mongo rubygem:

```
gem install mongo
```

The required `bson` gem will be installed automatically.

For optimum performance, install the bson_ext gem:

```
gem install bson_ext
```

After installing, you may want to look at the examples directory included in the source distribution. These examples walk through some of the basics of using the Ruby driver.

The full API documentation can be viewed here.

A Quick Tour

Using the RubyGem

All of the code here assumes that you have already executed the following Ruby code:

```
require 'rubygems'  # not necessary for Ruby 1.9
require 'mongo'
```

Making a Connection

An Mongo::Connection instance represents a connection to MongoDB. You use a Connection instance to obtain an Mongo:DB instance, which represents a named database. The database doesn't have to exist - if it doesn't, MongoDB will create it for you.

You can optionally specify the MongoDB server address and port when connecting. The following example shows three ways to connect to the database "mydb" on the local machine:

```
db = Mongo::Connection.new.db("mydb")
db = Mongo::Connection.new("localhost").db("mydb")
db = Mongo::Connection.new("localhost", 27017).db("mydb")
```

At this point, the db object will be a connection to a MongoDB server for the specified database. Each DB instance uses a separate socket connection to the server.

If you're trying to connect to a replica set, see Replica Sets in Ruby.

Listing All Databases

```
connection = Mongo::Connection.new # (optional host/port args)
connection.database_names.each { |name| puts name }
connection.database_info.each { |info| puts info.inspect}
```

Dropping a Database

```
connection.drop_database('database_name')
```

Authentication (Optional)

MongoDB can be run in a secure mode where access to databases is controlled through name and password authentication. When run in this mode, any client application must provide a name and password before doing any operations. In the Ruby driver, you simply do the following with the connected mongo object:

```
auth = db.authenticate(my_user_name, my_password)
```

If the name and password are valid for the database, `auth` will be `true`. Otherwise, it will be `false`. You should look at the MongoDB log for further information if available.

Getting a List Of Collections

Each database has zero or more collections. You can retrieve a list of them from the db (and print out any that are there):

```
db.collection_names.each { |name| puts name }
```

and assuming that there are two collections, name and address, in the database, you would see

```
name
address
```

as the output.

Getting a Collection

You can get a collection to use using the `collection` method:

```
coll = db.collection("testCollection")
```

This is aliased to the [] method:

```
coll = db["testCollection"]
```

Once you have this collection object, you can now do things like insert data, query for data, etc.

Inserting a Document

Once you have the collection object, you can insert documents into the collection. For example, lets make a little document that in JSON would be represented as

```
{
   "name" : "MongoDB",
   "type" : "database",
   "count" : 1,
   "info" : {
               x : 203,
               y : 102
             }
}
```

Notice that the above has an "inner" document embedded within it. To do this, we can use a Hash or the driver's OrderedHash (which preserves key order) to create the document (including the inner document), and then just simply insert it into the collection using the `insert()` method.

```
doc = {"name" => "MongoDB", "type" => "database", "count" => 1,
       "info" => {"x" => 203, "y" => '102'}}
coll.insert(doc)
```

Updating a Document

We can update the previous document using the `update` method. There are a couple ways to update a document. We can rewrite it:

```
doc["name"] = "MongoDB Ruby"
coll.update({"_id" => doc["_id"]}, doc)
```

Or we can use an atomic operator to change a single value:

```

coll.update({"_id" => doc["_id"]}, {"$set" => {"name" => "MongoDB Ruby"}})
```

Read more about updating documents.

Finding the First Document In a Collection using `find_one()`

To show that the document we inserted in the previous step is there, we can do a simple `find_one()` operation to get the first document in the collection. This method returns a single document (rather than the `Cursor` that the `find()` operation returns).

```
my_doc = coll.find_one()
puts my_doc.inspect
```

and you should see:

```
{"_id"=>#<BSON::ObjectID:0x118576c ...>, "name"=>"MongoDB", "info"=>{"x"=>203, "y"=>102}, "type"=>
"database", "count"=>1}
```

Note the `_id` element has been added automatically by MongoDB to your document.

Adding Multiple Documents

To demonstrate some more interesting queries, let's add multiple simple documents to the collection. These documents will have the following form:

```
{
   "i" : value
}
```

Here's how to insert them:

```
100.times { |i| coll.insert("i" => i) }
```

Notice that we can insert documents of different "shapes" into the same collection. These records are in the same collection as the complex record we inserted above. This aspect is what we mean when we say that MongoDB is "schema-free".

Counting Documents in a Collection

Now that we've inserted 101 documents (the 100 we did in the loop, plus the first one), we can check to see if we have them all using the `count()` method.

```
puts coll.count()
```

and it should print `101`.

Using a Cursor to get all of the Documents

To get all the documents from the collection, we use the `find()` method. `find()` returns a `Cursor` object, which allows us to iterate over the set of documents that matches our query. The Ruby driver's Cursor implemented Enumerable, which allows us to use `Enumerable#each`, {{Enumerable#map}, etc. For instance:

```
coll.find().each { |row| puts row.inspect }
```

and that should print all 101 documents in the collection.

Getting a Single Document with a Query

We can create a *query* hash to pass to the `find()` method to get a subset of the documents in our collection. For example, if we wanted to find the document for which the value of the "i" field is 71, we would do the following ;

```
coll.find("i" => 71).each { |row| puts row.inspect }
```

and it should just print just one document:

```
{"_id"=>#<BSON::ObjectID:0x117de90 ...>, "i"=>71}
```

Getting a Set of Documents With a Query

We can use the query to get a set of documents from our collection. For example, if we wanted to get all documents where "i" > 50, we could write:

```
coll.find("i" => {"$gt" => 50}).each { |row| puts row }
```

which should print the documents where i > 50. We could also get a range, say 20 < i <= 30:

```
coll.find("i" => {"$gt" => 20, "$lte" => 30}).each { |row| puts row }
```

Selecting a subset of fields for a query

Use the `:fields` option. If you just want fields "a" and "b":

```
coll.find("i" => {"$gt" => 50}, :fields => ["a", "b"]).each { |row| puts row }
```

Querying with Regular Expressions

Regular expressions can be used to query MongoDB. To find all names that begin with 'a':

```
coll.find({"name" => /^a/})
```

You can also construct a regular expression dynamically. To match a given search string:

```
  search_string = params['search']

  # Constructor syntax
  coll.find({"name" => Regexp.new(search_string)})

  # Literal syntax
  coll.find({"name" => /#{search_string}/})
```

Although MongoDB isn't vulnerable to anything like SQL-injection, it may be worth checking the search string for anything malicious.

Creating An Index

MongoDB supports indexes, and they are very easy to add on a collection. To create an index, you specify an index name and an array of field names to be indexed, or a single field name. The following creates an ascending index on the "i" field:

```
# create_index assumes ascending order; see method docs
# for details
coll.create_index("i")
```

To specify complex indexes or a descending index you need to use a slightly more complex syntax - the index specifier must be an Array of [field name, direction] pairs. Directions should be specified as Mongo::ASCENDING or Mongo::DESCENDING:

```
# explicit "ascending"
coll.create_index([["i", Mongo::ASCENDING]])
```

Creating and querying on a geospatial index

First, create the index on a field containing long-lat values:

```
people.create_index([["loc", Mongo::GEO2D]])
```

Then get a list of the twenty locations nearest to the point 50, 50:

```
people.find({"loc" => {"$near" => [50, 50]}}, {:limit => 20}).each do |p|
  puts p.inspect
end
```

Getting a List of Indexes on a Collection

You can get a list of the indexes on a collection using `coll.index_information()`.

Database Administration

A database can have one of three profiling levels: off (:off), slow queries only (:slow_only), or all (:all). To see the database level:

```
puts db.profiling_level   # => off (the symbol :off printed as a string)
db.profiling_level = :slow_only
```

Validating a collection will return an interesting hash if all is well or raise an exception if there is a problem.

```
p db.validate_collection('coll_name')
```

See Also

- Ruby Driver Official Docs
- MongoDB Koans A path to MongoDB enlightenment via the Ruby driver.
- MongoDB Manual

Replica Sets in Ruby

Redirection Notice
This page should redirect to http://api.mongodb.org/ruby/current/file.REPLICA_SETS.html.

Here follow a few considerations for those using the Ruby driver with MongoDB and replica sets.

- Setup
- Connection Failures
- Recovery
- Testing
- Further Reading

Setup

First, make sure that you've configured and initialized a replica set.

Connecting to a replica set from the Ruby driver is easy. If you only want to specify a single node, simply pass that node to `Connection.new`:

```
@connection = Connection.new('foo.local', 27017)
```

If you want to pass in multiple seed nodes, use `Connection.multi`:

```
@connection = Connection.multi([['n1.mydb.net', 27017],
   ['n2.mydb.net', 27017], ['n3.mydb.net', 27017]])
```

In both cases, the driver will attempt to connect to a master node and, when found, will merge any other known members of the replica set into the seed list.

Connection Failures

Imagine that our master node goes offline. How will the driver respond?

At first, the driver will try to send operations to what was the master node. These operations will fail, and the driver will raise a **ConnectionFailure** exception. It then becomes the client's responsibility to decide how to handle this.

If the client decides to retry, it's not guaranteed that another member of the replica set will have been promoted to master right away, so it's still possible that the driver will raise another **ConnectionFailure**. However, once a member has been promoted to master, typically within a few seconds, subsequent operations will succeed.

The driver will essentially cycle through all known seed addresses until a node identifies itself as master.

Recovery

Driver users may wish to wrap their database calls with failure recovery code. Here's one possibility:

```
# Ensure retry upon failure
def rescue_connection_failure(max_retries=5)
    success = false
    retries = 0
    while !success
      begin
        yield
        success = true
      rescue Mongo::ConnectionFailure => ex
        retries += 1
        raise ex if retries >= max_retries
        sleep(1)
      end
    end
  end
end

# Wrapping a call to #count()
rescue_connection_failure do
  @db.collection('users').count()
end
```

Of course, the proper way to handle connection failures will always depend on the individual application. We encourage object-mapper and application developers to publish any promising results.

Testing

The Ruby driver (>= 1.0.6) includes some unit tests for verifying replica set behavior. They reside in **tests/replica_sets**. You can run them individually with the following rake tasks:

```
rake test:replica_set_count
rake test:replica_set_insert
rake test:pooled_replica_set_insert
rake test:replica_set_query
```

Make sure you have a replica set running on localhost before trying to run these tests.

Further Reading

- Replica Sets
- [Replics Set Configuration]

GridFS in Ruby

Redirection Notice
This page should redirect to http://api.mongodb.org/ruby/current/file.GridFS.html.

GridFS, which stands for "Grid File Store," is a specification for storing large files in MongoDB. It works by dividing a file into manageable chunks and storing each of those chunks as a separate document. GridFS requires two collections to achieve this: one collection stores each file's metadata (e.g., name, size, etc.) and another stores the chunks themselves. If you're interested in more details, check out the GridFS Specification.

Prior to version 0.19, the MongoDB Ruby driver implemented GridFS using the GridFS::GridStore class. This class has been deprecated in favor of two new classes: Grid and GridFileSystem. These classes have a much simpler interface, and the rewrite has resulted in a significant speed improvement. **Reads are over twice as fast, and write speed has been increased fourfold**. 0.19 is thus a worthwhile upgrade.

- The Grid class
  - Saving files
  - File metadata
  - Safe mode
  - Deleting files
- The GridFileSystem class
  - Saving files
  - Deleting files
  - Metadata and safe mode
- Advanced Users

The Grid class

The Grid class represents the core GridFS implementation. Grid gives you a simple file store, keyed on a unique ID. This means that duplicate filenames aren't a problem. To use the Grid class, first make sure you have a database, and then instantiate a Grid:

```
@db = Mongo::Connection.new.db('social_site')

@grid = Grid.new(@db)
```

Saving files

Once you have a Grid object, you can start saving data to it. The data can be either a string or an IO-like object that responds to a #read method:

```
# Saving string data
id = @grid.put("here's some string / binary data")

# Saving IO data and including the optional filename
image = File.open("me.jpg")
id2   = @grid.put(image, :filename => "me.jpg")
```

Grid#put returns an object id, which you can use to retrieve the file:

```
# Get the string we saved
file = @grid.get(id)

# Get the file we saved
image = @grid.get(id2)
```

File metadata

There are accessors for the various file attributes:

```
  image.filename
# => "me.jpg"

  image.content_type
# => "image/jpg"

  image.file_length
# => 502357

  image.upload_date
# => Mon Mar 01 16:18:30 UTC 2010

  # Read all the image's data at once
  image.read

  # Read the first 100k bytes of the image
  image.read(100 * 1024)
```

When putting a file, you can set many of these attributes and write arbitrary metadata:

```
# Saving IO data
  file = File.open("me.jpg")
  id2  = @grid.put(file,
           :filename     => "my-avatar.jpg"
           :content_type => "application/jpg",
           :_id          => 'a-unique-id-to-use-in-lieu-of-a-random-one',
           :chunk_size   => 100 * 1024,
           :metadata     => {'description' => "taken after a game of ultimate"})
```

Safe mode

A kind of safe mode is built into the GridFS specification. When you save a file, and MD5 hash is created on the server. If you save the file in safe mode, an MD5 will be created on the client for comparison with the server version. If the two hashes don't match, an exception will be raised.

```
  image = File.open("me.jpg")
  id2   = @grid.put(image, "my-avatar.jpg", :safe => true)
```

Deleting files

Deleting a file is as simple as providing the id:

```
  @grid.delete(id2)
```

The GridFileSystem class

GridFileSystem is a light emulation of a file system and therefore has a couple of unique properties. The first is that filenames are assumed to be unique. The second, a consequence of the first, is that files are versioned. To see what this means, let's create a GridFileSystem instance:

Saving files

```
  @db = Mongo::Connection.new.db("social_site")

  @fs = GridFileSystem.new(@db)
```

Now suppose we want to save the file 'me.jpg.' This is easily done using a filesystem-like API:

```
image = File.open("me.jpg")
@fs.open("me.jpg", "w") do |f|
  f.write image
end
```

We can then retrieve the file by filename:

```
image = @fs.open("me.jpg", "r") {|f| f.read }
```

No problems there. But what if we need to replace the file? That too is straightforward:

```
image = File.open("me-dancing.jpg")
@fs.open("me.jpg", "w") do |f|
  f.write image
end
```

But a couple things need to be kept in mind. First is that the original 'me.jpg' will be available until the new 'me.jpg' saves. From then on, calls to the #open method will always return the most recently saved version of a file. But, and this the second point, old versions of the file won't be deleted. So if you're going to be rewriting files often, you could end up with a lot of old versions piling up. One solution to this is to use the :delete_old options when writing a file:

```
image = File.open("me-dancing.jpg")
@fs.open("me.jpg", "w", :delete_old => true) do |f|
  f.write image
end
```

This will delete all but the latest version of the file.

Deleting files

When you delete a file by name, you delete all versions of that file:

```
@fs.delete("me.jpg")
```

Metadata and safe mode

All of the options for storing metadata and saving in safe mode are available for the GridFileSystem class:

```
image = File.open("me.jpg")
@fs.open('my-avatar.jpg', w,
           :content_type => "application/jpg",
           :metadata     => {'description' => "taken on 3/1/2010 after a game of ultimate"},
           :_id          => 'a-unique-id-to-use-instead-of-the-automatically-generated-one',
           :safe         => true) { |f| f.write image }
```

Advanced Users

Astute code readers will notice that the Grid and GridFileSystem classes are merely thin wrappers around an underlying GridIO class. This means that it's easy to customize the GridFS implementation presented here; just use GridIO for all the low-level work, and build the API you need in an external manager class similar to Grid or GridFileSystem.

Rails - Getting Started

Using Rails 3? See Rails 3 - Getting Started

This tutorial describes how to set up a simple Rails application with MongoDB, using MongoMapper as an object mapper. We assume you're using Rails versions prior to 3.0.

- Configuration
- Testing
- Coding

Using a Rails Template
All of the configuration steps listed below, and more, are encapsulated in this Rails template (raw version), based on a similar one by Ben Scofield . You can create your project with the template as follows:

```
rails project_name -m "http://gist.github.com/219223.txt"
```

Be sure to replace **project_name** with the name of your project.

If you want to set up your project manually, read on.

Configuration

1. We need to tell MongoMapper which database we'll be using. Save the following to **config/initializers/database.rb**:

```
MongoMapper.database = "db_name-#{Rails.env}"
```

Replace **db_name** with whatever name you want to give the database. The **Rails.env** variable will ensure that a different database is used for each environment.

2. If you're using Passenger, add this code to **config/initializers/database.rb**.

```
if defined?(PhusionPassenger)
   PhusionPassenger.on_event(:starting_worker_process) do |forked|
     MongoMapper.connection.connect_to_master if forked
   end
end
```

3. Clean out **config/database.yml**. This file should be blank, as we're not connecting to the database in the traditional way.

4. Remove ActiveRecord from environment.rb.

```
config.frameworks -= [:active_record]
```

5. Add MongoMapper to the environment. This can be done by opening **config/environment.rb** and adding the line:

```
config.gem 'mongo_mapper'
```

Once you've done this, you can install the gem in the project by running:

```
rake gems:install
rake gems:unpack
```

Testing

It's important to keep in mind that with MongoDB, we cannot wrap test cases in transactions. One possible work-around is to invoke a **teardown** method after each test case to clear out the database.

To automate this, I've found it effective to modify **ActiveSupport::TestCase** with the code below.

```
# Drop all columns after each test case.
def teardown
  MongoMapper.database.collections.each do |coll|
    coll.remove
  end
end

# Make sure that each test case has a teardown
# method to clear the db after each test.
def inherited(base)
  base.define_method teardown do
    super
  end
end
```

This way, all test classes will automatically invoke the teardown method. In the example above, the teardown method clears each collection. We might also choose to drop each collection or drop the database as a whole, but this would be considerably more expensive and is only necessary if our tests manipulate indexes.

Usually, this code is added in **test/test_helper.rb**. See the aforementioned rails template for specifics.

Coding

If you've followed the foregoing steps (or if you've created your Rails with the provided template), then you're ready to start coding. For help on that, you can read about modeling your domain in Rails.

Rails 3 - Getting Started

It's not difficult to use MongoDB with Rails 3. Most of it comes down to making sure that you're not loading ActiveRecord and understanding how to use Bundler, the new Ruby dependency manager.

- Install the Rails 3
- Configure your application
- Bundle and Initialize
    - Bundling
    - Initializing
- Running Tests
- Conclusion

Install the Rails 3

If you haven't done so already, install Rails 3.

```
# Use sudo if your setup requires it
gem install rails
```

Configure your application

The important thing here is to avoid loading ActiveRecord. One way to do this is with the `--skip-active-record` switch. So you'd create your app skeleton like so:

```
rails new my_app --skip-active-record
```

Alternatively, if you've already created your app (or just want to know what this actually does), have a look at `config/application.rb` and change the first lines from this:

```
require "rails/all"
```

to this:

```
require "action_controller/railtie"
require "action_mailer/railtie"
require "active_resource/railtie"
require "rails/test_unit/railtie"
```

It's also important to make sure that the reference to active_record in the generator block is commented out:

```
# Configure generators values. Many other options are available, be sure to check the documentation.
# config.generators do |g|
#   g.orm             :active_record
#   g.template_engine :erb
#   g.test_framework  :test_unit, :fixture => true
# end
```

As of this this writing, it's commented out by default, so you probably won't have to change anything here.

Bundle and Initialize

The final step involves bundling any gems you'll need and then creating an initializer for connecting to the database.

Bundling

Edit `Gemfile`, located in the Rails root directory. By default, our `Gemfile` will only load Rails:

```
gem "rails", "3.0.0"
```

Normally, using MongoDB will simply mean adding whichever OM framework you want to work with, as these will require the "mongo" gem by default.

```
# Edit this Gemfile to bundle your application's dependencies.

source 'http://gemcutter.org'

gem "rails", "3.0.0"
gem "mongo_mapper"
```

However, there's currently an issue with loading bson_ext, as the current gemspec isn't compatible with the way Bundler works. We'll be fixing that soon; just pay attention to this issue.

In the meantime, you can use the following work-around:

```
# Edit this Gemfile to bundle your application's dependencies.

require 'rubygems'
require 'mongo'
source 'http://gemcutter.org'

gem "rails", "3.0.0"
gem "mongo_mapper"
```

Requiring `rubygems` and `mongo` before running the `gem` command will ensure that `bson_ext` is loaded. If you'd rather not load `rubygems`, just make sure that both `mongo` and `bson_ext` are in your load path when you require `mongo`.

Once you've configured your `Gemfile`, run the bundle installer:

```
bundle install
```

Initializing

Last item is to create an initializer to connect to MongoDB. Create a Ruby file in `config/initializers`. You can give it any name you want; here we'll call it `config/initializers/mongo.rb`:

```
MongoMapper.connection = Mongo::Connection.new('localhost', 27017)
MongoMapper.database = "#myapp-#{Rails.env}"

if defined?(PhusionPassenger)
   PhusionPassenger.on_event(:starting_worker_process) do |forked|
     MongoMapper.connection.connect if forked
   end
end
```

Running Tests

A slight modification is required to get `rake test` working (thanks to John P. Wood). Create a file `lib/tasks/mongo.rake` containing the following:

```
namespace :db do
  namespace :test do
    task :prepare do
      # Stub out for MongoDB
    end
  end
end
```

Now the various `rake test` tasks will run properly. See John's post for more details.

Conclusion

That should be all. You can now start creating models based on whichever OM you've installed.

MongoDB Data Modeling and Rails

This tutorial discusses the development of a web application on Rails and MongoDB. MongoMapper will serve as our object mapper. The goal is to provide some insight into the design choices required for building on MongoDB. To that end, we'll be constructing a simple but non-trivial social news application. The source code for **newsmonger** is available on github for those wishing to dive right in.

- Modeling Stories
    - Caching to Avoid N+1
    - A Note on Denormalization
    - Fields as arrays
    - Atomic Updates
- Modeling Comments
    - Linear, Embedded Comments
    - Nested, Embedded Comments
    - Comment collections
- Unfinished business

Assuming you've configured your application to work with MongoMapper, let's start thinking about the data model.

Modeling Stories

A news application relies on stories at its core, so we'll start with a Story model:

```
class Story
  include MongoMapper::Document

  key :title,     String
  key :url,       String
  key :slug,      String
  key :voters,    Array
  key :votes,     Integer, :default => 0
  key :relevance, Integer, :default => 0

  # Cached values.
  key :comment_count, Integer, :default => 0
  key :username,      String

  # Note this: ids are of class ObjectId.
  key :user_id,   ObjectId
  timestamps!

  # Relationships.
  belongs_to :user

  # Validations.
  validates_presence_of :title, :url, :user_id
end
```

Obviously, a story needs a title, url, and user_id, and should belong to a user. These are self-explanatory.

Caching to Avoid N+1

When we display our list of stories, we'll need to show the name of the user who posted the story. If we were using a relational database, we could perform a join on users and stores, and get all our objects in a single query. But MongoDB does not support joins and so, at times, requires bit of denormalization. Here, this means caching the 'username' attribute.

A Note on Denormalization

Relational purists may be feeling uneasy already, as if we were violating some universal law. But let's bear in mind that MongoDB collections are not equivalent to relational tables; each serves a unique design objective. A normalized table provides an atomic, isolated chunk of data. A document, however, more closely represents an object as a whole. In the case of a social news site, it can be argued that a username is intrinsic to the story being posted.

What about updates to the username? It's true that such updates will be expensive; happily, in this case, they'll be rare. The read savings achieved in denormalizing will surely outweigh the costs of the occasional update. Alas, this is not hard and fast rule: ultimately, developers must evaluate their applications for the appropriate level of normalization.

Fields as arrays

With a relational database, even trivial relationships are blown out into multiple tables. Consider the votes a story receives. We need a way of recording which users have voted on which stories. The standard way of handling this would involve creating a table, 'votes', with each row referencing user_id and story_id.

With a document database, it makes more sense to store those votes as an array of user ids, as we do here with the 'voters' key.

For fast lookups, we can create an index on this field. In the MongoDB shell:

```
db.stories.ensureIndex('voters');
```

Or, using MongoMapper, we can specify the index in **config/initializers/database.rb**:

```
Story.ensure_index(:voters)
```

To find all the stories voted on by a given user:

```
Story.all(:conditions => {:voters => @user.id})
```

Atomic Updates

Storing the `voters` array in the `Story` class also allows us to take advantage of atomic updates. What this means here is that, when a user votes on a story, we can

1. ensure that the voter hasn't voted yet, and, if not,
2. increment the number of votes and
3. add the new voter to the array.

MongoDB's query and update features allows us to perform all three actions in a single operation. Here's what that would look like from the shell:

```
// Assume that story_id and user_id represent real story and user ids.
db.stories.update({_id: story_id, voters: {'$ne': user_id}},
  {'$inc': {votes: 1}, '$push': {voters: user_id}});
```

What this says is "get me a story with the given id whose `voters` array does not contain the given user id and, if you find such a story, perform two atomic updates: first, increment `votes` by 1 and then push the user id onto the `voters` array."

This operation highly efficient; it's also reliable. The one caveat is that, because update operations are "fire and forget," you won't get a response from the server. But in most cases, this should be a non-issue.

A MongoMapper implementation of the same update would look like this:

```
  def self.upvote(story_id, user_id)
    collection.update({'_id' => story_id, 'voters' => {'$ne' => user_id}},
      {'$inc' => {'votes' => 1}, '$push' => {'voters' => user_id}})
  end
```

Modeling Comments

In a relational database, comments are usually given their own table, related by foreign key to some parent table. This approach is occasionally necessary in MongoDB; however, it's always best to try to embed first, as this will achieve greater query efficiency.

Linear, Embedded Comments

Linear, non-threaded comments should be embedded. Here are the most basic MongoMapper classes to implement such a structure:

```
class Story
  include MongoMapper::Document
  many :comments
end
```

```
class Comment
  include MongoMapper::EmbeddedDocument
  key :body, String

  belongs_to :story
end
```

If we were using the Ruby driver alone, we could save our structure like so:

```
@stories  = @db.collection('stories')
  @document = {:title => "MongoDB on Rails",
               :comments => [{:body     => "Revelatory! Loved it!",
                              :username => "Matz"
                             }
                            ]
              }
  @stories.save(@document)
```

Essentially, comments are represented as an array of objects within a story document. This simple structure should be used for any one-to-many

relationship where the many items are linear.

Nested, Embedded Comments

But what if we're building threaded comments? An admittedly more complicated problem, two solutions will be presented here. The first is to represent the tree structure in the nesting of the comments themselves. This might be achieved using the Ruby driver as follows:

```
@stories  = @db.collection('stories')
  @document = {:title => "MongoDB on Rails",
               :comments => [{:body     => "Revelatory! Loved it!",
                              :username => "Matz",
                              :comments => [{:body     => "Agreed.",
                                             :username => "rubydev29"
                                            }
                                           ]
                             }
                            ]
              }
  @stories.save(@document)
```

Representing this structure using MongoMapper would be tricky, requiring a number of custom mods.

But this structure has a number of benefits. The nesting is captured in the document itself (this is, in fact, how Business Insider represents comments). And this schema is highly performant, since we can get the story, and all of its comments, in a single query, with no application-side processing for constructing the tree.

One drawback is that alternative views of the comment tree require some significant reorganizing.

Comment collections

We can also represent comments as their own collection. Relative to the other options, this incurs a small performance penalty while granting us the greatest flexibility. The tree structure can be represented by storing the unique path for each leaf (see Mathias's original post on the idea). Here are the relevant sections of this model:

```
class Comment
  include MongoMapper::Document

  key :body,       String
  key :depth,      Integer, :default => 0
  key :path,       String,  :default => ""

  # Note: we're intentionally storing parent_id as a string
  key :parent_id,  String
  key :story_id,   ObjectId
  timestamps!

  # Relationships.
  belongs_to :story

  # Callbacks.
  after_create :set_path

  private

  # Store the comment's path.
  def set_path
    unless self.parent_id.blank?
      parent        = Comment.find(self.parent_id)
      self.story_id = parent.story_id
      self.depth    = parent.depth + 1
      self.path     = parent.path + ":" + parent.id
    end
    save
  end
```

The path ends up being a string of object ids. This makes it easier to display our comments nested, with each level in order of karma or votes. If we specify an index on story_id, path, and votes, the database can handle half the work of getting our comments in nested, sorted order.

The rest of the work can be accomplished with a couple grouping methods, which can be found in the newsmonger source code.

It goes without saying that modeling comments in their own collection also facilitates various site-wide aggregations, including displaying the latest, grouping by user, etc.

Unfinished business

Document-oriented data modeling is still young. The fact is, many more applications will need to be built on the document model before we can say anything definitive about best practices. So the foregoing should be taken as suggestions, only. As you discover new patterns, we encourage you to document them, and feel free to let us know about what works (and what doesn't).

Developers working on object mappers and the like are encouraged to implement the best document patterns in their code, and to be wary of recreating relational database models in their apps.

Ruby External Resources

There are a number of good resources appearing all over the web for learning about MongoDB and Ruby. A useful selection is listed below. If you know of others, do let us know.

- Screencasts
- Presentations
- Articles
- Projects
- Libraries

Screencasts

Introduction to MongoDB - Part I
An introduction to MongoDB via the MongoDB shell.

Introduction to MongoDB - Part II
In this screencast, Joon You teaches how to use the Ruby driver to build a simple Sinatra app.

Introduction to MongoDB - Part III
For the final screencast in the series, Joon You introduces MongoMapper and Rails.

RailsCasts: MongoDB & MongoMapper
Ryan Bates' RailsCast introducing MongoDB and MongoMapper.

RailsCasts: Mongoid
Ryan Bates' RailsCast introducing Mongoid.

Presentations

Introduction to MongoDB (Video)
Mike Dirolf's introduction to MongoDB at Pivotal Labs, SF.

MongoDB: A Ruby Document Store that doesn't rhyme with 'Ouch' (Slides)
Wynn Netherland's introduction to MongoDB with some comparisons to CouchDB.

MongoDB (is) for Rubyists (Slides)
Kyle Banker's presentation on why MongoDB is for Rubyists (and all human-oriented programmers).

Articles

Why I Think Mongo is to Databases What Rails was to Frameworks
What if a key-value store mated with a relational database system?
Mongo Tips
John Nunemaker's articles on MongoDB and his Mongo Tips blog.

A series of articles on aggregation with MongoDB and Ruby:
1. Part I: Introduction of Aggregation in MongoDB
2. Part II: MongoDB Grouping Elaborated
3. Part III: Introduction to Map-Reduce in MongoDB

Does the MongoDB Driver Support Feature X?
An explanation of how the MongoDB drivers usually automatically support new database features.

Projects

Simple Pub/Sub

A very simple pub/sub system.

Mongo Queue
An extensible thread safe job/message queueing system that uses mongodb as the persistent storage engine.

Resque-mongo
A port of the Github's Resque to MongoDB.

Mongo Admin
A Rails plugin for browsing and managing MongoDB data. See the live demo.

Sinatra Resource
Resource Oriented Architecture (REST) for Sinatra and MongoMapper.

Shorty
A URL-shortener written with Sinatra and the MongoDB Ruby driver.

NewsMonger
A simple social news application demonstrating MongoMapper and Rails.

Data Catalog API
From Sunlight Labs, a non-trivial application using MongoMapper and Sinatra.

Watchtower
An example application using Mustache, MongoDB, and Sinatra.

Shapado
A question and answer site similar to Stack Overflow. Live version at shapado.com.

Libraries

ActiveExpando
An extension to ActiveRecord to allow the storage of arbitrary attributes in MongoDB.

ActsAsTree (MongoMapper)
ActsAsTree implementation for MongoMapper.

Machinist adapter (MongoMapper)
Machinist adapter using MongoMapper.

Mongo-Delegate
A delegation library for experimenting with production data without altering it. A quite useful pattern.

Remarkable Matchers (MongoMapper)
Testing / Matchers library using MongoMapper.

OpenIdAuthentication, supporting MongoDB as the datastore
Brandon Keepers' fork of OpenIdAuthentication supporting MongoDB.

MongoTree (MongoRecord)
MongoTree adds parent / child relationships to MongoRecord.

Merb_MongoMapper
a plugin for the Merb framework for supporting MongoMapper models.

Mongolytics (MongoMapper)
A web analytics tool.

Rack-GridFS
A Rack middleware component that creates HTTP endpoints for files stored in GridFS.

Frequently Asked Questions - Ruby

Redirection Notice
This page should redirect to http://api.mongodb.org/ruby/1.1.5/file.FAQ.html.

This is a list of frequently asked questions about using Ruby with MongoDB. If you have a question you'd like to have answered here, please add it in the comments.

- Can I run [insert command name here] from the Ruby driver?
- Does the Ruby driver support an EXPLAIN command?

- I see that BSON supports a symbol type. Does this mean that I can store Ruby symbols in MongoDB?
- Why can't I access random elements within a cursor?
- Why can't I save an instance of TimeWithZone?
- I keep getting CURSOR_NOT_FOUND exceptions. What's happening?
- I periodically see connection failures between the driver and MongoDB. Why can't the driver retry the operation automatically?

Can I run [insert command name here] from the Ruby driver?

Yes. You can run any of the available database commands from the driver using the DB#command method. The only trick is to use an OrderedHash when specifying the command. For example, here's how you'd run an asynchronous fsync from the driver:

```
# This command is run on the admin database.
@db = Mongo::Connection.new.db('admin')

# Build the command.
cmd = OrderedHash.new
cmd['fsync'] = 1
cmd['async'] = true

# Run it.
@db.command(cmd)
```

It's important to keep in mind that some commands, like `fsync`, must be run on the `admin` database, while other commands can be run on any database. If you're having trouble, check the command reference to make sure you're using the command correctly.

Does the Ruby driver support an EXPLAIN command?

Yes. `explain` is, technically speaking, an option sent to a query that tells MongoDB to return an explain plan rather than the query's results. You can use `explain` by constructing a query and calling explain at the end:

```
@collection = @db['users']
result = @collection.find({:name => "jones"}).explain
```

The resulting explain plan might look something like this:

```
{"cursor"=>"BtreeCursor name_1",
 "startKey"=>{"name"=>"Jones"},
 "endKey"=>{"name"=>"Jones"},
 "nscanned"=>1.0,
 "n"=>1,
 "millis"=>0,
 "oldPlan"=>{"cursor"=>"BtreeCursor name_1",
             "startKey"=>{"name"=>"Jones"},
             "endKey"=>{"name"=>"Jones"}
 },
 "allPlans"=>[{"cursor"=>"BtreeCursor name_1",
               "startKey"=>{"name"=>"Jones"},
               "endKey"=>{"name"=>"Jones"}}]
 }
```

Because this collection has an index on the "name" field, the query uses that index, only having to scan a single record. "n" is the number of records the query will return. "millis" is the time the query takes, in milliseconds. "oldPlan" indicates that the query optimizer has already seen this kind of query and has, therefore, saved an efficient query plan. "allPlans" shows all the plans considered for this query.

I see that BSON supports a symbol type. Does this mean that I can store Ruby symbols in MongoDB?

You can store Ruby symbols in MongoDB, but only as values. BSON specifies that document keys must be strings. So, for instance, you can do this:

```
@collection = @db['test']

boat_id = @collection.save({:vehicle  => :boat})
car_id  = @collection.save({"vehicle" => "car"})

@collection.find_one('_id' => boat_id)
{"_id" => ObjectID('4bb372a8238d3b5c8c000001'), "vehicle" => :boat}


@collection.find_one('_id' => car_id)
{"_id" => ObjectID('4bb372a8238d3b5c8c000002'), "vehicle" => "car"}
```

Notice that the symbol values are returned as expected, but that symbol keys are treated as strings.

Why can't I access random elements within a cursor?

MongoDB cursors are designed for sequentially iterating over a result set, and all the drivers, including the Ruby driver, stick closely to this directive. Internally, a Ruby cursor fetches results in batches by running a MongoDB `getmore` operation. The results are buffered for efficient iteration on the client-side.

What this means is that a cursor is nothing more than a device for returning a result set on a query that's been initiated on the server. Cursors are not containers for result sets. If we allow a cursor to be randomly accessed, then we run into issues regarding the freshness of the data. For instance, if I iterate over a cursor and then want to retrieve the cursor's first element, should a stored copy be returned, or should the cursor re-run the query? If we returned a stored copy, it may not be fresh. And if the the query is re-run, then we're technically dealing with a new cursor.

To avoid those issues, we're saying that anyone who needs flexible access to the results of a query should store those results in an array and then access the data as needed.

Why can't I save an instance of TimeWithZone?

MongoDB stores times in UTC as the number of milliseconds since the epoch. This means that the Ruby driver serializes Ruby Time objects only. While it would certainly be possible to serialize a TimeWithZone, this isn't preferable since the driver would still deserialize to a Time object.

All that said, if necessary, it'd be easy to write a thin wrapper over the driver that would store an extra time zone attribute and handle the serialization/deserialization of TimeWithZone transparently.

I keep getting CURSOR_NOT_FOUND exceptions. What's happening?

The most likely culprit here is that the cursor is timing out on the server. Whenever you issue a query, a cursor is created on the server. Cursor naturally time out after ten minutes, which means that if you happen to be iterating over a cursor for more than ten minutes, you risk a CURSOR_NOT_FOUND exception.

There are two solutions to this problem. You can either:

1. Limit your query. Use some combination of `limit` and `skip` to reduce the total number of query results. This will, obviously, bring down the time it takes to iterate.

2. Turn off the cursor timeout. To do that, invoke `find` with a block, and pass `:timeout => true`:

```
  @collection.find({}, :timeout => false) do |cursor|
    cursor.each do |document
      # Process documents here
    end
  end
```

I periodically see connection failures between the driver and MongoDB. Why can't the driver retry the operation automatically?

A connection failure can indicate any number of failure scenarios. Has the server crashed? Are we experiencing a temporary network partition? Is there a bug in our ssh tunnel?

Without further investigation, it's impossible to know exactly what has caused the connection failure. Furthermore, when we do see a connection failure, it's impossible to know how many operations prior to the failure succeeded. Imagine, for instance, that we're using safe mode and we send an `$inc` operation to the server. It's entirely possible that the server has received the `$inc` but failed on the call to `getLastError`. In that case, retrying the operation would result in a double-increment.

Because of the indeterminacy involved, the MongoDB drivers will not retry operations on connection failure. How connection failures should be handled is entirely dependent on the application. Therefore, we leave it to the application developers to make the best decision in this case.

The drivers will reconnect on the subsequent operation.

Java Language Center

Java Driver

- Java Driver
  - Basics
  - Specific Topics and How-To
- Third Party Frameworks and Libs
  - POJO Mappers
  - Code Generation
  - Misc
- Clojure
- Groovy
- JavaScript (Rhino)
- JRuby
- Scala
- Hadoop
- Presentations

Basics

- Download the Java Driver
- Tutorial
- API Documentation
- Release Notes

Specific Topics and How-To

- Concurrency
- Saving Objects
- Data Types
- Aggregation Framework
- Read Preferences and Tagging

Third Party Frameworks and Libs

POJO Mappers

- Morphia - Type-Safe Wrapper with DAO/Datastore abstractions
- Mungbean (w/clojure support)
- Spring MongoDB – Provides Spring users with a familiar data access features including rich POJO mapping.
- DataNucleus JPA/JDO – JPA/JDO wrapper
- lib-mongomapper JavaBean Mapper (No annotations)
- mongo-jackson-mapper Uses jackson (annotations) to map to/from POJOs and has a simple wrapper around DBCollection to simply this.
- Kundera - JPA compliant ORM. Works with multiple datastores.
- Jongo - Query in Java as in Mongo shell (using strings), unmarshall results into Java objects (using Jackson)

Code Generation

- Sculptor - mongodb-based DSL -> Java (code generator)
- GuicyData - DSL -> Java generator with Guice integration
  - Blog Entries

Misc

- MongoDB Asynchronous Java Driver - This driver has been created to ensure maximum utilization of resources on the client, in the network, on the server, and in the developer's IDE.
- log4mongo - a log4j appender
- mongo-java-logging - a Java logging handler
- (Experimental, Type4) JDBC driver
- Metamodel data exploration and querying library
- Mongodb Java REST server based on Jetty

Clojure

- Congo Mongo
- monger

Groovy

- Groovy Tutorial for MongoDB
- MongoDB made more Groovy
- GMongo, a Groovy wrapper to the mongodb Java driver
    - GMongo 0.5 Release Writeup

JavaScript (Rhino)

- MongoDB-Rhino - A toolset to provide full integration between the Rhino JavaScript engine for the JVM and MongoDB. Uses the MongoDB Java driver.

JRuby

- jmongo A thin ruby wrapper around the mongo-java-driver for vastly better jruby performance.

If there is a project missing here, just add a comment or email the list and we'll add it.

Scala

- Scala Language Center

Hadoop

- Hadoop

Presentations

- Building a Mongo DSL in Scala at Hot Potato - Lincoln Hochberg's Presentation from MongoSF (April 2010)
- Java Development - Brendan McAdams' Presentation from MongoNYC (May 2010)
- Java Development - James Williams' Presentation from MongoSF (April 2010)
- Morphia: Easy Java Persistance for MongoDB - Scott Hernandez' presentation at MongoSF (May 2011)
- Spring Source and MongoDB - Chris Richardson's presentation at MongoSV (December 2010)
- Using MongoDB with Scala - Brendan McAdams' Presentation at the New York Scala Enthusiasts (August 2010)
- More Java-related presentations

Java Driver Concurrency

The Java MongoDB driver is thread safe. If you are using in a web serving environment, for example, you should create a single Mongo instance, and you can use it in every request. The Mongo object maintains an internal pool of connections to the database (default pool size of 10). For every request to the DB (find, insert, etc) the java thread will obtain a connection from the pool, execute the operation, and release the connection. This means the connection (socket) used may be different each time.

Additionally in the case of a replica set with slaveOk option turned on, the read operations will be distributed evenly across all slaves. This means that within the same thread, a write followed by a read may be sent to different servers (master then slave). In turn the read operation may not see the data just written since replication is asynchronous. If you want to ensure complete consistency in a "session" (maybe an http request), you would want the driver to use the same socket, which you can achieve by using a "consistent request". Call requestStart() before your operations and requestDone() to release the connection back to the pool:

```
DB db...;
db.requestStart();
try {
   db.requestEnsureConnection();

   code....
} finally {
   db.requestDone();
}
```

DB and DBCollection are completely thread safe. In fact, they are cached so you get the same instance no matter what.

WriteConcern option for single write operation

Since by default a connection is given back to the pool after each request, you may wonder how calling getLastError() works after a write. You should actually use a write concern like `WriteConcern.SAFE` instead of calling getLastError() manually. The driver will then call getLastError() before putting the connection back in the pool.

```
DBCollection coll...;
coll.insert(..., WriteConcern.SAFE);

// is equivalent to
DB db...;
DBCollection coll...;
db.requestStart();
try {
   coll.insert(...);
   DBObject err = db.getLastError();
} finally {
   db.requestDone();
}
```

Java - Saving Objects Using DBObject

The Java driver provides a DBObject interface to save custom objects to the database.

For example, suppose one had a class called Tweet that they wanted to save:

```
public class Tweet implements DBObject {
    /* ... */
}
```

Then you can say:

```
Tweet myTweet = new Tweet();
myTweet.put("user", userId);
myTweet.put("message", msg);
myTweet.put("date", new Date());

collection.insert(myTweet);
```

When a document is retrieved from the database, it is automatically converted to a DBObject. To convert it to an instance of your class, use DBCollection.setObjectClass():

```
collection.setObjectClass(Tweet.class);

Tweet myTweet = (Tweet)collection.findOne();
```

If for some reason you wanted to change the message you can simply take that tweet and save it back after updating the field.

```
Tweet myTweet = (Tweet)collection.findOne();
myTweet.put("message", newMsg);

collection.save(myTweet);
```

Java Tutorial

- Introduction
- A Quick Tour
    - Making A Connection
    - Authentication (Optional)
    - Getting A List Of Collections
    - Getting A Collection
    - Setting write concern

    - Inserting a Document
    - Finding the First Document In A Collection using `findOne()`
    - Adding Multiple Documents
    - Counting Documents in A Collection
    - Using a Cursor to Get All the Documents
    - Getting A Single Document with A Query
    - Getting A Set of Documents With a Query
    - Creating An Index
    - Getting a List of Indexes on a Collection
- Quick Tour of the Administrative Functions
    - Getting A List of Databases
    - Dropping A Database

Introduction

This page is a brief overview of working with the MongoDB Java Driver.

For more information about the Java API, please refer to the online API Documentation for Java Driver

A Quick Tour

Using the Java driver is very simple. First, be sure to include the driver jar `mongo.jar` in your classpath. The following code snippets come from the `examples/QuickTour.java` example code found in the driver.

Making A Connection

To make a connection to a MongoDB, you need to have at the minimum, the name of a database to connect to. The database doesn't have to exist - if it doesn't, MongoDB will create it for you.

Additionally, you can specify the server address and port when connecting. The following example shows three ways to connect to the database *mydb* on the local machine :

```
import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.WriteConcern;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.ServerAddress;

import java.util.Arrays;


MongoClient mongoClient = new MongoClient();
// or
MongoClient mongoClient = new MongoClient( "localhost" );
// or
MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
// or, to connect to a replica set, supply a seed list of members
MongoClient mongoClient = new MongoClient(Arrays.asList(new ServerAddress("localhost", 27017),
                                      new ServerAddress("localhost", 27018),
                                      new ServerAddress("localhost", 27019)));

DB db = m.getDB( "mydb" );
```

At this point, the `db` object will be a connection to a MongoDB server for the specified database. With it, you can do further operations.

Note: The MongoClient instance actually represents a pool of connections to the database; you will only need one instance of class MongoClient even with multiple threads. See the concurrency doc page for more information.

The MongoClient class is designed to be thread safe and shared among threads. Typically you create only 1 instance for a given database cluster and use it across your application. If for some reason you decide to create many MonglClient instances, note that:

- all resource usage limits (max connections, etc) apply per MongoClient instance
- to dispose of an instance, make sure you call MongoClient.close() to clean up resources

Note: The MongoClient class is new in version 2.10.0. For releases prior to that, please use the Mongo class instead.

Authentication (Optional)

MongoDB can be run in a secure mode where access to databases is controlled through name and password authentication. When run in this mode, any client application must provide a name and password before doing any operations. In the Java driver, you simply do the following with a MongoClient instance:

```
MongoClient mongoClient = new MongoClient();
DB db = mongoClient.getDB("test");
boolean auth = db.authenticate(myUserName, myPassword);
```

If the name and password are valid for the database, `auth` will be `true`. Otherwise, it will be `false`. You should look at the MongoDB log for further information if available.

Most users run MongoDB without authentication in a trusted environment.

Getting A List Of Collections

Each database has zero or more collections. You can retrieve a list of them from the db (and print out any that are there) :

```
Set<String> colls = db.getCollectionNames();

for (String s : colls) {
    System.out.println(s);
}
```

and assuming that there are two collections, name and address, in the database, you would see

```
name
address
```

as the output.

Getting A Collection

To get a collection to use, just specify the name of the collection to the getCollection`(String collectionName)` method:

```
DBCollection coll = db.getCollection("testCollection");
```

Once you have this collection object, you can now do things like insert data, query for data, etc

Setting write concern

As of version 2.10.0, the default write concern is WriteConcern.ACKNOWLEDGED, but it can be easily changed:

```
m.setWriteConcern(WriteConcern.JOURNALED);
```

There are many options for write concern. Additionally, the default write concern can be overridden on the database, collection, and individual update operations. Please consult the API Documentation for details.

Note: Prior to version 2.10.0, the default write concern is WriteConcern.NORMAL. Under normal circumstances, clients will typically change this to ensure they are notified of problems writing to the database.

Inserting a Document

Once you have the collection object, you can insert documents into the collection. For example, lets make a little document that in JSON would be represented as

```
{
   "name" : "MongoDB",
   "type" : "database",
   "count" : 1,
   "info" : {
               x : 203,
               y : 102
             }
}
```

Notice that the above has an "inner" document embedded within it. To do this, we can use the BasicDBObject class to create the document (including the inner document), and then just simply insert it into the collection using the `insert()` method.

```
        BasicDBObject doc = new BasicDBObject("name", "MongoDB").
                                    append("type", "database").
                                    append("count", 1)
                                   .append("info", new BasicDBObject("x", 203).append("y", 102));

        coll.insert(doc);
```

Finding the First Document In A Collection using `findOne()`

To show that the document we inserted in the previous step is there, we can do a simple findOne() operation to get the first document in the collection. This method returns a single document (rather than the DBCursor that the find() operation returns), and it's useful for things where there only is one document, or you are only interested in the first. You don't have to deal with the cursor.

```
DBObject myDoc = coll.findOne();
System.out.println(myDoc);
```

and you should see

```
{ "_id" : "49902cde5162504500b45c2c" , "name" : "MongoDB" , "type" : "database" , "count" : 1 , "info"
: { "x" : 203 , "y" : 102}}
```

Note the `_id` element has been added automatically by MongoDB to your document. Remember, MongoDB reserves element names that start with "_"/"$" for internal use.

Adding Multiple Documents

In order to do more interesting things with queries, let's add multiple simple documents to the collection. These documents will just be

```
{
   "i" : value
}
```

and we can do this fairly efficiently in a loop

```
for (int i=0; i < 100; i++) {
    coll.insert(new BasicDBObject("i", i));
}
```

Notice that we can insert documents of different "shapes" into the same collection. This aspect is what we mean when we say that MongoDB is "schema-free"

Counting Documents in A Collection

Now that we've inserted 101 documents (the 100 we did in the loop, plus the first one), we can check to see if we have them all using the `getCount()` method.

```
System.out.println(coll.getCount());
```

and it should print `101`.

Using a Cursor to Get All the Documents

In order to get all the documents in the collection, we will use the `find()` method. The `find()` method returns a `DBCursor` object which allows us to iterate over the set of documents that matched our query. So to query all of the documents and print them out :

```
DBCursor cursor = coll.find();
        try {
            while(cursor.hasNext()) {
                System.out.println(cursor.next());
            }
        } finally {
            cursor.close();
        }
```

and that should print all 101 documents in the collection.

Getting A Single Document with A Query

We can create a *query* to pass to the `find()` method to get a subset of the documents in our collection. For example, if we wanted to find the document for which the value of the "i" field is 71, we would do the following ;

```
BasicDBObject query = new BasicDBObject("i", 71);

        cursor = coll.find(query);

        try {
            while(cursor.hasNext()) {
                System.out.println(cursor.next());
            }
        } finally {
            cursor.close();
        }
```

and it should just print just one document

```
{ "_id" : "49903677516250c1008d624e" , "i" : 71 }
```

You may commonly see examples and documentation in MongoDB which use $ Operators, such as this:

```
db.things.find({j: {$ne: 3}, k: {$gt: 10} });
```

These are represented as regular String keys in the Java driver, using embedded DBObjects:

```
BasicDBObject query = new BasicDBObject("j", new BasicDBObject("$ne", 3).
                                             append("k", new BasicDBObject("$gt", 10));

        cursor = coll.find(query);

        try {
            while(cursor.hasNext()) {
                System.out.println(cursor.next());
            }
        } finally {
            cursor.close();
        }
```

Getting A Set of Documents With a Query

We can use the query to get a set of documents from our collection. For example, if we wanted to get all documents where "i" > 50, we could write :

```
query = new BasicDBObject("i", new BasicDBObject("$gt", 50));  // e.g. find all where i > 50

        cursor = coll.find(query);

        try {
            while(cursor.hasNext()) {
                System.out.println(cursor.next());
            }
        } finally {
            cursor.close();
        }
```

which should print the documents where i > 50. We could also get a range, say 20 < i <= 30 :

```
query = new BasicDBObject("i", new BasicDBObject("$gt", 20).
                                                    append("$lte", 30));  // i.e.   20 < i <= 30

        cursor = coll.find(query);

        try {
            while(cursor.hasNext()) {
                System.out.println(cursor.next());
            }
        } finally {
            cursor.close();
        }
```

Creating An Index

MongoDB supports indexes, and they are very easy to add on a collection. To create an index, you just specify the field that should be indexed, and specify if you want the index to be ascending (1) or descending (-1). The following creates an ascending index on the "i" field :

```
coll.createIndex(new BasicDBObject("i", 1));  // create index on "i", ascending
```

Getting a List of Indexes on a Collection

You can get a list of the indexes on a collection :

```
List<DBObject> list = coll.getIndexInfo();

        for (DBObject o : list) {
            System.out.println(o);
        }
```

and you should see something like

```
{ "name" : "i_1" , "ns" : "mydb.testCollection" , "key" : { "i" : 1} }
```

Quick Tour of the Administrative Functions

Getting A List of Databases

You can get a list of the available databases:

```
MongoClient mongoClient = new MongoClient();

        for (String s : m.getDatabaseNames()) {
            System.out.println(s);
        }
```

Dropping A Database

You can drop a database by name using a `MongoClient` instance:

```
MongoClient mongoClient = new MongoClient();
mongoClient.dropDatabase("myDatabase");
```

Java Types

- Object Ids
- Regular Expressions
- Dates/Times
- Database References
- Binary Data
- Timestamp Data
- Code Data
- Embedded Documents
- Arrays

Object Ids

`com.mongodb.ObjectId` is used to autogenerate unique ids.

```
ObjectId id = new ObjectId();
ObjectId copy = new ObjectId(id);
```

Regular Expressions

The Java driver uses `java.util.regex.Pattern` for regular expressions.

```
Pattern john = Pattern.compile("joh?n", CASE_INSENSITIVE);
BasicDBObject query = new BasicDBObject("name", john);

// finds all people with "name" matching /joh?n/i
DBCursor cursor = collection.find(query);
```

Dates/Times

The java.util.Date class is used for dates.

```
Date now = new Date();
BasicDBObject time = new BasicDBObject("ts", now);

collection.save(time);
```

Database References

`com.mongodb.DBRef` can be used to save database references.

```
DBRef addressRef = new DBRef(db, "foo.bar", address_id);
DBObject address = addressRef.fetch();

DBObject person = BasicDBObjectBuilder.start()
    .add("name", "Fred")
    .add("address", addressRef)
    .get();
collection.save(person);

DBObject fred = collection.findOne();
DBRef addressObj = (DBRef)fred.get("address");
addressObj.fetch()
```

Binary Data

An array of bytes (`byte[]`) as a value will automatically be wrapped as a Binary type.
Additionally the Binary class can be used to represent binary objects, which allows to pick a custom type byte.

Timestamp Data

A timestamp is a special object used by Mongo as an ID based on time, represented by a (time in second, incremental id) pair (it is used notably in the replication oplog).
A timestamp is represented by the BSONTimestamp class.

Code Data

A code object is used to represent JavaScript code, for example when saving executable functions into the system.js collection.
The Code and CodeWScope classes are used to represent this data.
Note that some methods (like map/reduce) take Strings but wrap them into Code objects in the driver.

Embedded Documents

Suppose we have a document that, in JavaScript, looks like:

```
{
    "x" : {
        "y" : 3
    }
}
```

The equivalent in Java is:

```
BasicDBObject y = new BasicDBObject("y", 3);
BasicDBObject x = new BasicDBObject("x", y);
```

Arrays

Anything that extends List in Java will be saved as an array.

So, if you are trying to represent the JavaScript:

```
{
    "x" : [
        1,
        2,
        {"foo" : "bar"},
        4
    ]
}
```

you could do:

```
ArrayList x = new ArrayList();
x.add(1);
x.add(2);
x.add(new BasicDBObject("foo", "bar"));
x.add(4);

BasicDBObject doc = new BasicDBObject("x", x);
```

Read Preferences and Tagging in The Java Driver

MongoDB's read preferences and tagging allows application developers to target member nodes within a replica set for read or write operations. Version 2.2 brings several refinements on node tagging that give you greater control over how your data is read or written. Release 2.9.0 of the Java driver has been built in coordination with the release of Mongo 2.2 and provides full access to these newly available preferences and tagging features.

Read Preferences
A read preference provides client applications with control over which nodes of a replica set are used for reads. A client application defines its read preference by selecting one of the five behavioral modes:

PRIMARY : The default read mode. Read from primary only. Throw an error if primary is unavailable. Cannot be combined with tags.
PRIMARY PREFERRED : Read from primary if available, otherwise a secondary.
SECONDARY : Read from a secondary node if available, otherwise error.
SECONDARY PREFERRED : Read from a secondary if available, otherwise read from the primary.
NEAREST : Read from any member node from the set of nodes which respond the fastest. The responsiveness of a node is measured with pings. Any node whose ping time is within 15 milliseconds of the node with the lowest ping time is considered near.

Implementation

The Java driver implements MongoDB's read preferences with the ReadPreference class. Client applications allocate the desired read mode by calling one of ReadPreference's static factory methods. One factory method exists for each mode.

```
ReadPreference.primary();
ReadPreference.primaryPreferred();
ReadPreference.secondary();
ReadPreference.secondaryPreferred();
ReadPreference.nearest();
```

The factory method returns a private inner subclass of ReadPreference that implements the correct behavioral mode. The driver's use of polymorphism in this manner means that your code only needs to interface with the ReadPreference class alone.

example:
Suppose we are developing an application and we prefer operate on strongly consistent data, (i.e. read requests always return the most recent updates of a given document). In this case, the primary node will handle all reads and writes. Now suppose our application must be able to perform reads even if the primary becomes unavailable for some reason. Even though we prefer data consistency we can tolerate eventual consistency when the primary is down. We therefore use the primary preferred mode as our read preference.

```
ReadPreference preference = ReadPreference.primaryPreferred();
DBCursor cur = new DBCursor(collection, query, null, preference);
```

The Java driver maintains knowledge and state of the replica set by periodically pinging the member nodes and can detect the loss of a member. In this example, the driver will automatically detect the loss of the primary and route the read request to a secondary node, just as we have instructed.

Tags
As of version 2.0 of MongoDB, each node within your replica set can be marked with additional descriptors called tags. Tags can be used to indicate a node's location, membership in a designated set, or other characteristics. Tags enable your application to read from or write to specific nodes or a set of nodes in a replica set.

As an example, suppose we're running a replica set of three nodes deployed across three separate data centers. Each of these data centers is in a separate geographic location. We want to ensure that our data will persist in the event of a disaster, so we mark each node with a tag indicating the region where it lives. Our replica set configuration might look similar to this:

```
foo:SECONDARY> rs.conf()
{
        "_id" : "foo",
        "version" : 103132,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "localhost:27017",
                        "priority" : 10,
                        "tags" : {
                                "datacenter" : "Los Angeles",
                                "region" : "US_West"
                        }
                },
                {
                        "_id" : 1,
                        "host" : "localhost:27018",
                        "tags" : {
                                "datacenter" : "San Jose",
                                "region" : "US_West"
                        }
                },
                {
                        "_id" : 2,
                        "host" : "localhost:27019",
                        "tags" : {
                                "datacenter" : "Richmond",
                                "region" : "US_East"
                        }
                }
        ],
        "settings" : {
                "getLastErrorModes" : {
                        "DRSafe" : {
                                "region" : 2
                        }
                }
        }
}
foo:SECONDARY>
```

Notice the “settings” field in the replication configuration. We’ve defined a new getLastErrorModes object with the key DRSafe. When our client application uses this error mode in a write concern it is instructing the write operation to replicate to at least two regions before completion. Here’s an example in the Java driver:

```
// create a write concern with the specific getLastErrorMode
WriteConcern concern = new WriteConcern("DRSafe");

// an insert with the custom write concern
coll.insert(new BasicDBObject("name", "simple doc"), concern);
```

By allocating a write concern using the “DRSafe” error mode and passing it in on the insert, we have now ensured that our data has been backed up to two separate regions and will be available should a data center fail.

Using Tags with Read Preferences

Continuing with our sample application, we decide that we want to send our read requests to the nearest node to reduce request latency. The Java driver’s read preference API gives a couple of ways of doing this, the easiest is to simply use the nearest mode.

```
DBObject query = new BasicDBObject("name", "simple doc")
DBObject result =
coll.findOne(query, null, ReadPreference.nearest());
```

By using the nearest mode the driver will automatically send the read to one of a set of nodes with the lowest ping time relative to the client application, (the receiving node could be either a primary or secondary). But suppose our client application can determine where its requests originate. We could have explicitly tagged the read preference to use the datacenter nearest to our location.

Look again at the replica set configuration from above. Each node has been tagged by data center. Let's say that the current read request is coming from southern California. We can configure this read request to be served by the node living in our Los Angeles data center.

```
// initialize a properly tagged read preference
ReadPreference tagged_pref =
ReadPreference.secondaryPreferred(new BasicDBObject("datacenter", "Los Angeles"));

// include the tagged read preference in this request}}
DBObject result = coll.findOne(}}
new BasicDBObject("name", "simple doc"), null, tagged_pref);
```

Read preferences can also accept multiple tags. Returning to our example application, suppose we want to send our reads either the "Los Angeles" node, or failing to find a healthy member in Los Angeles a node in the "US_West" region

```
// read from either LA or US_West
DBObject tagSetOne = new BasicDBObject("datacenter", "Los Angeles"):
DBObject tagSetTwo = new BasicDBObject("region", "US_West");

ReadPreference pref =
ReadPreference.primaryPreferred(tagSetOne, tagSetTwo);
```

In this example, the driver looks first for a member tagged with datacenter "Los Angeles". If it cannot find an available member, the driver will it look for a member tagged with region of "US West". You may use a tag set to define a set of required tags a member node must have to be used for the read. For example:

```
// read from either LA or US_West
DBObject tagSetOne = new BasicDBObject("datacenter", "Los Angeles");
tagSetOne.put("rack", "1");
DBObject tagSetTwo = new BasicDBObject("region", "US_West");

ReadPreference pref =
ReadPreference.primaryPreferred(tagSetOne, tagSetTwo);
```

The difference being the driver is looking for a node that is tagged with both data center Los Angeles and on rack 1, or a node that is in region "US_West".

You can set the Read Preference at the operation, collection, DB, Mongo, MongoOptions, or MongoURI level, and the preference will be inherited similar to the way slaveOk and write concern are. Read preferences work with any server version that supports replica sets (1.6 and up). Tagged read preferences work with any version that supports tagging (2.0 and up). However, tagging will only work on a sharded cluster if you are connecting to a mongos running 2.2 or above.

Replica Set Semantics

The MongoDB Java driver handles failover in replicated setups with tunable levels of transparency to the user. By default, a `Mongo` connection object will ignore failures of secondaries, and only reads will throw `MongoExceptions` when the primary node is unreachable.

The level of exception reporting is tunable however, using a specific `WriteConcern`; you can set this on the Mongo/DB/Collection/Method level. Several levels are included as static options:

- `WriteConcern.NONE` : No exceptions thrown.
- `WriteConcern.NORMAL` : Exceptions are only thrown when the primary node is unreachable for a read, or the full replica set is unreachable.
- `WriteConcern.SAFE` : Same as the above, but exceptions thrown when there is a server error on writes or reads. Calls `getLastError()`.
- `WriteConcern.REPLICAS_SAFE` : Tries to write to two separate nodes. Same as the above, but will throw an exception if two writes are not possible.
- `WriteConcern.FSYNC_SAFE` : Same as `WriteConcern.SAFE`, but also waits for write to be written to disk.

Additional errors may appear in the log files, these are for reporting purposes and logged based on the logging settings.

Sample code is provided which illustrates some of these options. To quickly initialize a sample replica set, you can use the `mongo` shell:

```
> var rst = new ReplSetTest({ nodes : 3 })
> rst.startSet() // wait for processes to start
> rst.initiate() // wait for replica set initialization
```

Java client code demonstrating error handling is available :

- https://github.com/mongodb/mongo-snippets/blob/master/java/Test.java

Using The Aggregation Framework with The Java Driver

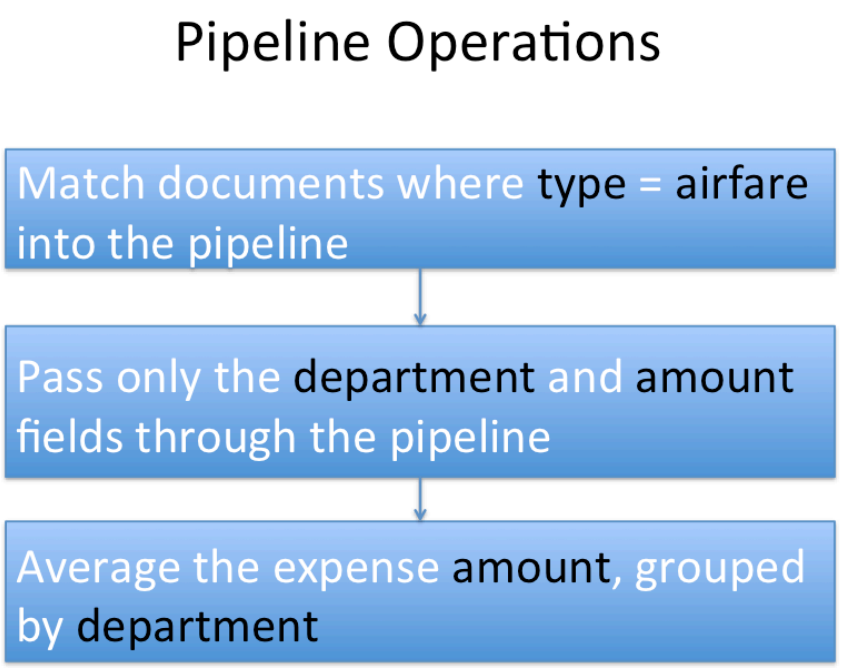
Release 2.2.0 of MongoDB introduces the aggregation framework. Designed to be both performant and easy to use, the aggregation framework calculates aggregate values, (such as counts, totals and averages), without the need for complex map-reduce operations. The aggregation framework is both multithreaded and written in C++, thus it executes natively across nodes.

Aggregation tasks are built around the concept of the aggregation pipeline. Just as UNIX-like shells use the pipe operator '|' to connect a series of command-line operations together, the aggregation framework passes documents through a pipeline of operations which transform these objects as they go. Version 2.9.0 of the Java driver provides a new helper method, DBCollection.aggregate() which can be used to create aggregation tasks.

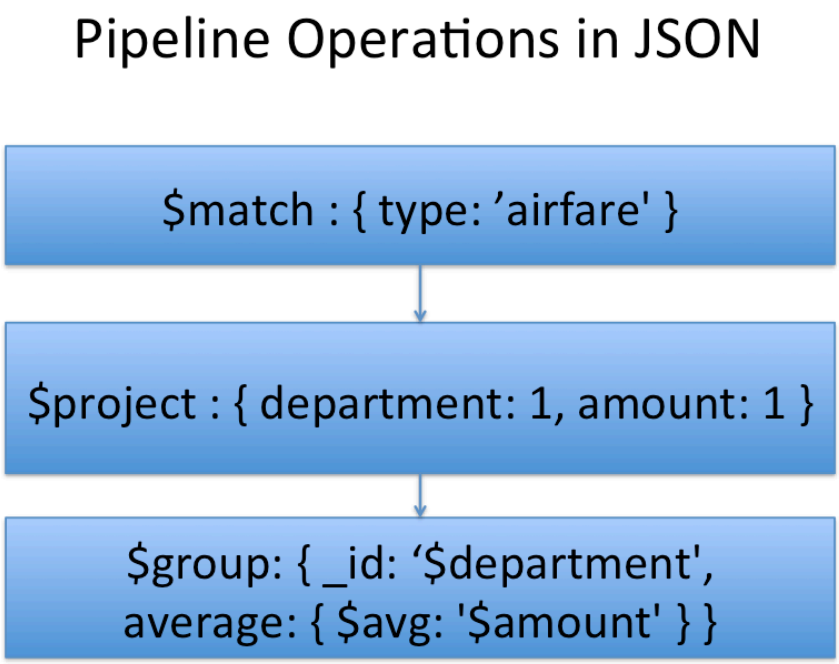
Let's use a simple example to demonstrate how the aggregation helper works. Suppose I am using MongoDB to store my employee's travel expenses. I've created a collection named "expenses", which store individual expenses by employee and by department. Here's a sample document:

```
{"_id" : ObjectId("503d5024ff9038cdbfcc9da4"),
"employee" : 61,
"department" : "Sales",
"amount" : 77,
"type" : "airfare"
}
```

I am auditing three departments: Sales, Engineering and Human Resources. I want to calculate each department's average spend on airfare. I'd like to use the Aggregation Framework for the audit, so I think of the operation in terms of a pipeline:



I will use the aggregation operators $match, $project and $group to perform each operation. Individual aggregation operations can be expressed as JSON objects, so I can think of my pipeline as:



I use the Java Driver's aggregation helper to build out this pipeline in my application. Let's take a look at the aggregate() method signature.

```
public AggregationOutput aggregate( DBObject firstOp, DBObject ... additionalOps)
```

The aggregate method uses Java varargs and accepts arbitrary number of DBObjects as parameters. These DBObjects represent aggregation operations, which will be chained together by the helper method to form the aggregation pipeline. Callers of the aggregate method must pass at least one aggregation operation. Here's the Java code we'll use to perform the aggregation task:

```
// create our pipeline operations, first with the $match
DBObject match = new BasicDBObject("$match", new BasicDBObject("type", "airfare") );

// build the $projection operation
DBObject fields = new BasicDBObject("department", 1);
fields.put("amount", 1);
fields.put("_id", 0);
DBObject project = new BasicDBObject("$project", fields );

// Now the $group operation
DBObject groupFields = new BasicDBObject( "_id", "$department");
groupFields.put("average", new BasicDBObject( "$avg", "$amount"));
DBObject group = new BasicDBObject("$group", groupFields);

// run aggregation
AggregationOutput output = collection.aggregate( match, project, group );
```

Aggregations are executed as database commands in MongoDB. These commands embed the results of the aggregation task in an object that also contains additional information about how the command was executed. The return value of aggregate() is an instance of the AggregationOutput class, which provides assessors to this information.

```
public Iterable<DBObject> results()
public CommandResult getCommandResult
public DBObject getCommand()
```

Let's take a look at the results of my audit:

```
System.out.println(output.getCommandResult());

{
"serverUsed" : "/127.0.0.1:27017" ,
"result" : [
{"_id" : "Human Resources" , "average" : 74.91735537190083} ,
{"_id" : "Sales" , "average" : 72.30275229357798} ,
{"_id" : "Engineering" , "average" : 74.1}
] ,
"ok" : 1.0
}
```

C++ Language Center

A C++ driver is available for communicating with the MongoDB. As the database is written in C++, the driver actually uses some core MongoDB code -- this is the same driver that the database uses itself for replication.

The driver has been compiled successfully on Linux, OS X, Windows, and Solaris.

- Downloading the Driver
- Compiling and Linking
- MongoDB C++ Client Tutorial
- API Documentation
- Using BSON from C++
- SQL to C++ Mapping Chart
- HOWTO
    - Connecting
    - getLastError
    - Tailable Cursors
    - BSON Arrays in C++
- Mongo Database and C++ Driver Source Code (at github). See the client subdirectory for client driver related files.

Additional Notes

- The Building documentation covers compiling the entire database, but some of the notes there may be helpful for compiling client applications too.
- There is also a pure C driver for MongoDB. For true C++ apps we recommend using the C++ driver.

BSON Arrays in C++

```
// examples

using namespace mongo;
using namespace bson;

bo an_obj;

/** transform a BSON array into a vector of BSONElements.
    we match array # positions with their vector position, and ignore
    any fields with non-numeric field names.
*/
vector<be> a = an_obj["x"].Array();

be array = an_obj["x"];
assert( array.isABSONObj() );
assert( array.type() == Array );

// Use BSON_ARRAY macro like BSON macro, but without keys
BSONArray arr = BSON_ARRAY( "hello" << 1 << BSON( "foo" << BSON_ARRAY( "bar" << "baz" << "qux" ) ) );

// BSONArrayBuilder can be used to build arrays without the macro
BSONArrayBuilder b;
b.append(1).append(2).arr();

/** add all elements of the object to the specified vector */
bo myarray = an_obj["x"].Obj();
vector<be> v;
myarray.elems(v);
list<be> L;
myarray.elems(L)


/** add all values of the object to the specified collection.  If type mismatches,
    exception.
        template <class T>
        void Vals(vector<T> &) const;
        template <class T>
        void Vals(list<T> &) const;
*/

/** add all values of the object to the specified collection.  If type mismatches, skip.
        template <class T>
        void vals(vector<T> &) const;
        template <class T>
        void vals(list<T> &) const;
*/
```

C++ BSON Library

- Standalone Usage
- API Docs
- Short Class Names

The MongoDB C++ driver library includes a bson package that implements the BSON specification (see http://www.bsonspec.org/). This library can be used standalone for object serialization and deserialization even when one is not using MongoDB at all.

Include `bson/bson.h` or `db/jsobj.h` in your application (not both). `bson.h` is new and may not work in some situations, was is good for light header-only usage of BSON (see the bsondemo.cpp example).

Key classes:

- mongo::BSONObj (aka bson::bo) a BSON object
- mongo::BSONElement (bson::be) a single element in a bson object. This is a key and a value.
- mongo::BSONObjBuilder (bson::bob) used to make BSON objects

- mongo::BSONObjIterator (bson::bo::iterator) to enumerate BSON objects

Let's now create a BSON "person" object which contains name and age. We might invoke:

```
BSONObjBuilder b;
b.append("name", "Joe");
b.append("age", 33);
BSONObj p = b.obj();
```

Or more concisely:

```
BSONObj p = BSONObjBuilder().append("name", "Joe").append("age", 33).obj();
```

We can also create objects with a stream-oriented syntax:

```
BSONObjBuilder b;
b << "name" << "Joe" << "age" << 33;
BSONObj p = b.obj();
```

The macro BSON lets us be even more compact:

```
BSONObj p = BSON( "name" << "Joe" << "age" << 33 );
```

Use the GENOID helper to add an object id to your object. The server will add an _id automatically if it is not included explicitly.

```
BSONObj p = BSON( GENOID << "name" << "Joe" << "age" << 33 );
// result is: { _id : ..., name : "Joe", age : 33 }
```

GENOID should be at the beginning of the generated object. We can do something similar with the non-stream builder syntax:

```
BSONObj p =
  BSONObjBuilder().genOID().append("name","Joe").append("age",33).obj();
```

Standalone Usage

You can use the C++ BSON library without MongoDB. Most BSON methods under the `bson/` directory are header-only. They require boost, but headers only.

See the bsondemo.cpp example at github.com.

API Docs

- http://api.mongodb.org/cplusplus/

Short Class Names

Add

```
using namespace bson;
```

to your code to use the following shorter more C++ style names for the BSON classes:

```
// from bsonelement.h
namespace bson {
    typedef mongo::BSONElement be;
    typedef mongo::BSONObj bo;
    typedef mongo::BSONObjBuilder bob;
}
```

(Or one could use `bson::bo` fully qualified for example).

Also available is `bo::iterator` as a synonym for `BSONObjIterator`.

C++ Driver Compiling and Linking

- Linux
  - Using the prebuilt library
  - Using mongo_client_lib.cpp instead of a library
- Windows
  - Windows Troubleshooting

The C++ driver is included in the MongoDB server source repository, and can also be downloaded as a separate, "standalone" tarball (see Downloads). To compile the "standalone" C++ driver, run the scons command in the top-level directory of the driver sources, e.g.:

```
cd mongo-cxx-driver-nightly/
scons
```

You may wish to compile and link client/simple_client_demo.cpp to verify that everything compiles and links fine.

Linux

Using the prebuilt library

```
$ cd mongo/client
$ g++ simple_client_demo.cpp -lmongoclient -lboost_thread-mt -lboost_filesystem
-lboost_program_options
```

Using mongo_client_lib.cpp instead of a library

If you have a compatibility problem with the library, include mongo_client_lib.cpp in your project instead. For example:

```
g++ -I .. simple_client_demo.cpp mongo_client_lib.cpp -lboost_thread-mt -lboost_filesystem
```

Windows

Note: we tend to test MongoDB with Windows using Visual Studio 2010. 2008 works, although you may have to tweak settings in some cases.

Include mongoclient.lib in your application.

To build mongoclient.lib:

```
scons mongoclient
```

Alternatively, include client/mongo_client_lib.cpp in your project.

For Windows, see also:

- bson/bsondemo/bsondemo.vcxproj
- client/examples/simple_client_demo.vcxproj
- Prebuilt Boost Libraries
  - Prebuild Boost Libraries only necessary for versions < 2.1.1
  - Boost 1.49 source is now included directly in version 2.1.1+

- Building with Visual Studio 2010

Other notes for Windows:

- Compile with /MT
- You may wish to define _CRT_SECURE_NO_WARNINGS to avoid warnings on use of strncpy and such by the MongoDB client code.
- Include the WinSock library in your application : Linker.Input.Additional Dependencies - add ws2_32.lib.

Windows Troubleshooting

- error LNK2005: ___ already defined in msvcprt.lib(MSVCP100.dll) libboost_thread-vc100-mt-1_42.lib(thread.obj)
  - The boost library being linked against is expecting a /MT build. If this is a release build, try using /MT instead of /MD for compilation (under Properties.C++.Code Generation).

C++ Driver Download

Driver tarballs

The C++ client library can be found at http://dl.mongodb.org/dl/cxx-driver/.

Note: despite the word 'linux' in the filenames, these files are mostly source code and thus should be applicable to all operating systems.

From the server source code

If you have the full MongoDB source code, the driver code is part of it. This is available on github and also on the MongoDB Downloads page.

The full server source is quite large, so the tarballs above may be easier. Also if using the full server source tree, use "scons mongoclient" to build just the client library.

Next : Compiling and Linking

C++ getLastError

- string mongo::DBClientWithCommands::getLastError();
  - Get error result from the last operation on this connection. Empty string if no error.
- BSONObj DBClientWithCommands::getLastErrorDetailed();
  - Get the full last error object. See the getLastError Command page for details.

See client/simple_client_demo.cpp for an example.

See Also

- getLastError Command

C++ Tutorial

- Installing the Driver Library and Headers
  - Unix
    - Full Database Source Driver Build
    - Driver Build
  - Windows
- Compiling
- Writing Client Code
  - Connecting
  - BSON
  - Inserting
  - Querying
  - Indexing
  - Sorting
  - Updating
  - Example
- Further Reading

This document is an introduction to usage of the MongoDB database from a C++ program.

First, install Mongo -- see the Quickstart for details.

Next, you may wish to take a look at the Developer's Tour guide for a language independent look at how to use MongoDB. Also, we suggest

some basic familiarity with the `mongo` shell -- the shell is one's primary database administration tool and is useful for manually inspecting the contents of a database after your C++ program runs.

Installing the Driver Library and Headers

A good source for general information about setting up a MongoDB development environment on various operating systems is the building page.

The normal database distribution used to include the C++ driver, but there were many problems with library version mismatches so now you have to build from source. You can either get the full source code for the database and just build the C++ driver or download the driver separately and build it.

Unix

For Unix, the Mongo driver library is `libmongoclient.a`. For either build, run `scons --help` to see all options.

Full Database Source Driver Build

To install the libraries, run:

```
scons --full install
```

`--full` tells the install target to include the library and header files; by default library and header files are installed in */usr/local*.

You can use `--prefix` to change the install path: `scons --prefix /opt/mongo --full install`.

In version 2.0, you can also specify `--sharedclient` to build a shared library instead of a statically linked library. This feature is not yet working properly in 2.2, see SERVER-6514.

Driver Build

If you download the driver source code separately, you can build it by running `scons` (no options).

Windows

For more information on Boost setup see the Building for Windows page.

Compiling

The C++ driver requires the pcre and boost libraries (with headers) to compile. Be sure they are in your include and lib paths. You can usually install them from your OS's package manager if you don't already have them.

Writing Client Code

Note: for brevity, the examples below are simply inline code. In a real application one will define classes for each database object typically.

Connecting

Let's make a tutorial.cpp file that connects to the database (see client/examples/tutorial.cpp for full text of the examples below):

```
#include <cstdlib>
#include <iostream>
#include "mongo/client/dbclient.h"

void run() {
  mongo::DBClientConnection c;
  c.connect("localhost");
}

int main() {
  try {
    run();
    std::cout << "connected ok" << std::endl;
  } catch( const mongo::DBException &e ) {
    std::cout << "caught " << e.what() << std::endl;
  }
  return EXIT_SUCCESS;
}
```

If you are using gcc on Linux or OS X, you would compile with something like this, depending on location of your include files and libraries:

```
$ g++ tutorial.cpp -pthread -lmongoclient -lboost_thread-mt -lboost_filesystem -lboost_program_options
-lboost_system -o tutorial
$ ./tutorial
connected ok
$
```

- You may need to append "-mt" to boost_filesystem and boost_program_options. If using a recent boost, "-mt" is not needed anymore.
- You may need to use -I and -L to specify the locations of your mongo and boost headers and libraries.

BSON

The Mongo database stores data in BSON format. BSON is a binary object format that is JSON-like in terms of the data which can be stored (some extensions exist, for example, a Date datatype).

To save data in the database we must create objects of class BSONObj. The components of a BSONObj are represented as BSONElement objects. We use BSONObjBuilder to make BSON objects, and BSONObjIterator to enumerate BSON objects.

Let's now create a BSON "person" object which contains name and age. We might invoke:

```
BSONObjBuilder b;
b.append("name", "Joe");
b.append("age", 33);
BSONObj p = b.obj();
```

Or more concisely:

```
BSONObj p = BSONObjBuilder().append("name", "Joe").append("age", 33).obj();
```

We can also create objects with a stream-oriented syntax:

```
BSONObjBuilder b;
b << "name" << "Joe" << "age" << 33;
BSONObj p = b.obj();
```

The macro BSON lets us be even more compact:

```
BSONObj p = BSON( "name" << "Joe" << "age" << 33 );
```

Use the GENOID helper to add an object id to your object. The server will add an _id automatically if it is not included explicitly.

```
BSONObj p = BSON(GENOID << "name" << "Joe" << "age" << 33);
// result is: { _id : ..., name : "Joe", age : 33 }
```

GENOID should be at the beginning of the generated object. We can do something similar with the non-stream builder syntax:

```
BSONObj p =
  BSONObjBuilder().genOID().append("name","Joe").append("age",33).obj();
```

Other helpers are listed here.

Inserting

We now save our person object in a persons collection in the database:

```
c.insert("tutorial.persons", p);
```

The first parameter to insert is the namespace. tutorial is the database and persons is the collection name.

Querying

Let's now fetch all objects from the persons collection, and display them. We'll also show here how to use count().

```
cout << "count:" << c.count("tutorial.persons") << endl;

  auto_ptr<DBClientCursor> cursor =
    c.query("tutorial.persons", BSONObj());
  while (cursor->more())
      cout << cursor->next().toString() << endl;
```

BSONObj() is an empty BSON object -- it represents {} which indicates an empty query pattern (an empty query is a query for all objects).

We use BSONObj::toString() above to print out information about each object retrieved. BSONObj::toString is a diagnostic function which prints an abbreviated JSON string representation of the object. For full JSON output, use BSONObj::jsonString.

Let's now write a function which prints out the name (only) of all persons in the collection whose age is a given value:

```
void printIfAge(DBClientConnection& c, int age) {
  auto_ptr<DBClientCursor> cursor =
    c.query("tutorial.persons", QUERY("age" << age));
  while (cursor->more()) {
    BSONObj p = cursor->next();
    cout << p.getStringField("name") << endl;
  }
}
```

getStringField() is a helper that assumes the "name" field is of type string. To manipulate an element in a more generic fashion we can retrieve the particular BSONElement from the enclosing object:

```
BSONElement name = p["name"];
// or:
//BSONElement name = p.getField("name");
```

See the api docs, and jsobj.h, for more information.

Our query above, written as JSON, is of the form

```
{ age : <agevalue> }
```

Queries are BSON objects of a particular format -- in fact, we could have used the BSON() macro above instead of QUERY(). See class Query in dbclient.h for more information on Query objects, and the Sorting section below.

In the mongo shell (which uses javascript), we could invoke:

```
use tutorial;
db.persons.find({age : 33});
```

Indexing

Let's suppose we want to have an index on age so that our queries are fast. We would use:

```
c.ensureIndex("tutorial.persons", fromjson("{age:1}"));
```

The ensureIndex method checks if the index exists; if it does not, it is created. ensureIndex is intelligent and does not repeat transmissions to the server; thus it is safe to call it many times in your code, for example, adjacent to every insert operation.

In the above example we use a new function, fromjson. fromjson converts a JSON string to a BSONObj. This is sometimes a convenient way to specify BSON. Alternatively, we could have written:

```
c.ensureIndex("tutorial.persons", BSON( "age" << 1 ));
```

Sorting

Let's now make the results from printIfAge sorted alphabetically by name. To do this, we change the query statement from:

```
auto_ptr<DBClientCursor> cursor = c.query("tutorial.persons", QUERY("age" << age));
```

to

```
to  auto_ptr<DBClientCursor> cursor = c.query("tutorial.persons", QUERY("age" << age ).sort("name"));
```

Here we have used Query::sort() to add a modifier to our query expression for sorting.

Updating

Use the update() method to perform a database update . For example the following update in the mongo shell :

```
> use tutorial
> db.persons.update( { name : 'Joe', age : 33 },
...                  { $inc : { visits : 1 } } )
```

is equivalent to the following C++ code:

```
db.update("tutorial.persons",
           BSON("name" << "Joe" << "age" << 33),
           BSON("$inc" << BSON( "visits" << 1)));
```

Example

A simple example illustrating usage of BSON arrays and the "$nin" operator is available here .

Further Reading

This overview just touches on the basics of using Mongo from C++. There are many more capabilities. For further exploration:

- See the language-independent Developer's Tour;
- Experiment with the `mongo` shell;
- Review the doxygen API docs;
- See connecting pooling information in the API docs;
- See GridFS file storage information in the API docs;
- See the HOWTO pages under the C++ Language Center
- Consider getting involved to make the product (either C++ driver, tools, or the database itself) better!

List of helper functions

This is a non-exhaustive list of helper functions for use in the C++ stream syntax. An exhaustive list is here: bsonmisc.h

Typical example of stream syntax:

```
BSONObj p = BSON( "name" << "Joe" << "age" << 33 );
```

GENOID -the server will add an _id automatically if it is not included explicitly.

```
BSONObj p = BSON( GENOID << "name" << "Joe" << "age" << 33 );
// result is: { _id : ..., name : "Joe", age : 33 }
```

LT,GT,LTE,GTE, NE -less than, greater than, etc

```
BSONObj p = BSON( "age" << GT << 21 );
// result is:  { age : { $gt : 21 } }
```

DATENOW -translates to current date

```
BSONObj p = BSON( "created" << DATENOW );
// result is: { created : "2009-10-09 11:41:42" }
```

MINKEY,MAXKEY

```
BSONObj p = BSON( "a" << MINKEY);
// result is: { "a" : { "$minKey" : 1 } }
```

OR

```


OR(BSON("x" << GT << 7), BSON("y" << LT << 6))
// result is: {$or: [{x: {$gt: 7}}, {y: {$lt: 6}}]}
```

BSONNULL -translates to null value (will appear in mongoDB 2.1)

```
BSONObj p = BSON( "name" << "Methuselah" << "age" << BSONNULL );
// result is: { name : "Methuselah", age : null }
```

Connecting

The C++ driver includes several classes for managing collections under the parent class DBClientInterface.

DBClientConnection is our normal connection class for a connection to a single MongoDB database server (or shard manager). Other classes exist for connecting to a replica set.

See http://api.mongodb.org/cplusplus/ for details on each of the above classes.

SQL to Shell to C++

MongoDB queries are expressed as JSON (BSON) objects. This quick reference chart shows examples as SQL, Mongo shell syntax, and Mongo C++ driver syntax.

A query expression in MongoDB (and other things, such as an index key pattern) is represented as BSON. In C++ you can use BSONObjBuilder (aka bson::bob) to build BSON objects, or the BSON() macro. The examples below assume a connection c already established:

```
using namespace bson;
DBClientConnection c;
c.connect("somehost");
```

Several of the C++ driver methods throw mongo::DBException, so you will want a try/catch statement at some level in your program. Also be sure to call c.getLastError() after writes to check the error code.

SQL	Mongo Shell	C++ Driver

`INSERT INTO USERS VALUES(1,1)`	`db.users.insert({a:1,b:1})`	`// GENOID is optional.`
		`if not done by client,`
		`server will add an _id`
		`c.insert("mydb.users",`
		`  BSON(GENOID<<"a"<<1<<`
		`"b"<<1));`
		`// then:`
		`string err =`
		`c.getLastError();`

`SELECT a,b FROM users`	`db.users.find({}, {a:1,b:1})`	`auto_ptr<DBClientCursor>`
		`cursor =`
		`  c.query("mydb.users",`
		`Query(),`
		`  0, 0, BSON("a"<<1<<`
		`"b"<<1));`

`SELECT * FROM users`	`db.users.find()`	`auto_ptr<DBClientCursor>`
		`cursor =`
		`  c.query("mydb.users",`
		`Query());`

`SELECT * FROM users WHERE age=33`	`db.users.find({age:33})`	`auto_ptr<DBClientCursor>`
		`cursor =`
		`  c.query("mydb.users",`
		`QUERY("age"<<33))`
		`// or:`
		`auto_ptr<DBClientCursor>`
		`cursor =`
		`  c.query("mydb.users",`
		`BSON("age"<<33))`

SELECT *
FROM users
WHERE age=33
ORDER BY
name
	db.users.find({age:33}).sort({name:1})	auto_ptr<DBClientCursor>
		cursor =
		  c.query("mydb.users",
		
		  QUERY("age"
		<<33).sort("name"));

SELECT *
FROM users
WHERE age>33
AND age<=40
	db.users.find({'age':{$gt:33},{$lte:40}})	auto_ptr<DBClientCursor>
		cursor =
		  c.query("mydb.users",
		
		  QUERY("age"
		<<GT<<33<<LTE<<40));

CREATE INDEX myindexname ON users(name)	db.users.ensureIndex({name:1})	c.ensureIndex("mydb.users", BSON("name"<<1));

SELECT *
FROM users
LIMIT 10
SKIP 20
	db.users.find().limit(10).skip(20)	auto_ptr<DBClientCursor>
		cursor =
		  c.query("mydb.users",
		Query(),
		  10, 20);

SELECT * FROM users LIMIT 1	db.users.findOne()	bo obj = c.findOne("mydb.users", Query());

SELECT
DISTINCT
last_name
FROM users
WHERE x=1
	db.users.distinct('last_name',{x:1})	// no helper for
		distinct yet in c++
		driver, so send command
		manually
		bo cmdResult;
		bool ok = c.runCommand(
		  "mydb", BSON(
		"distinct" << "users"
		  << "key" <<
		"last_name"
		  << "query" <<
		BSON("x"<<1)),
		  cmdResult);
		list<bo> results;
		cmdResult["values"
		].Obj().Vals(results);

SELECT COUNT(*y) FROM users where AGE > 30	db.users.find({age: {'$gt': 30}}).count()	unsigned long long n = c.count("mydb.users", BSON("age"<<GT<<30));
UPDATE users SET a=a+2 WHERE b='q'	db.users.update({b:'q'}, {$inc:{a:2}}, false, true)	c.update("mydb.users", QUERY("b"<<"q"), BSON("$inc"<<BSON("a"<<2)), false, true); // then optionally: string err = c.getLastError(); bool ok = err.empty();
DELETE FROM users WHERE z="abc"	db.users.remove({z:'abc'});	c.remove("mydb.users", QUERY("z"<<"abc")); // then optionally: string err = c.getLastError();

See Also

- Several more examples (in shell syntax) are on the SQL to Mongo Mapping Chart page.
- C++ Language Center

Perl Language Center

- Installing
  - CPAN
  - Manual (Non-CPAN) Installation
  - Big-Endian Systems
- Next Steps
- MongoDB Perl Tools
  - BSON
  - Entities::Backend::MongoDB
  - MojoX::Session::Store::MongoDB
  - MongoDB::Admin
  - Mongoose
  - MongoDBI
  - MongoDBx-Class
  - MongoX
  - OOP Perl CMS

Installing

Start a MongoDB server instance (`mongod`) before installing so that the tests will pass. The `mongod` cannot be running as a slave for the tests to pass.

Some tests may be skipped, depending on the version of the database you are running.

CPAN

```
$ sudo cpan MongoDB
```

The Perl driver is available through CPAN as the package `MongoDB`. It should build cleanly on *NIX and Windows (via Strawberry Perl). It is also available as an ActivePerl module.

Manual (Non-CPAN) Installation

If you would like to try the latest code or are contributing to the Perl driver, it is available at Github. There is also documentation generated after every commit.

You can see if it's a good time to grab the bleeding edge code by seeing if the build is green.

To build the driver, run:

```
$ perl Makefile.PL
$ make
$ make test # make sure mongod is running, first
$ sudo make install
```

Please note that the tests will not pass without a `mongod` process running.

Note that the Perl driver requires some libraries available in CPAN. As of April, 2010, these are Any::Moose, Class::Method::Modifiers, Data::Types, DateTime, File::Slurp, Test::Exception, Try::Tiny, boolean, and Module::Install. (Additionally, Tie::IxHash is required for testing.) On Debian or Ubuntu systems, these prerequisites can be easily installed with the following command:

```
$ sudo apt-get install libmodule-install-perl libany-moose-perl libclass-method-modifiers-perl
libdata-types-perl libdatetime-perl libfile-slurp-perl libtest-exception-perl libtry-tiny-perl
libboolean-perl libtie-ixhash-perl
```

Big-Endian Systems

The driver will work on big-endian machines, but the database will not. The tests assume that `mongod` will be running on localhost unless %ENV{MONGOD} is set. So, to run the tests, start the database on a little-endian machine (at, say, "example.com") and then run the tests with:

```
MONGOD=example.com make test
```

A few tests that require a database server on "localhost" will be skipped.

Next Steps

There is a tutorial and API documentation on CPAN.

If you're interested in contributing to the Perl driver, check out Contributing to the Perl Driver.

MongoDB Perl Tools

BSON

`BSON` is a pure-Perl BSON implementation.

Entities::Backend::MongoDB

`Entities::Backend::MongoDB` is a backend for the Entities user management and authorization system stores all entities and relations between them in a MongoDB database, using the MongoDB module. This is a powerful, fast backend that gives you all the features of MongoDB.

MojoX::Session::Store::MongoDB

`MojoX::Session::Store::MongoDB` is a store for `MojoX::Session` that stores a session in a MongoDB database. Created by Ask Bjørn Hansen.

MongoDB::Admin

`MongoDB::Admin` is a collection of MongoDB administrative functions. Created by David Burley.

Mongoose

`Mongoose` is an attempt to bring together the full power of Moose with MongoDB. Created by Rodrigo de Oliveira Gonzalez.

`MongoDBI`

MongoDBI is an Object-Document-Mapper (ODM) for MongoDB. It allows you to create Moose-based classes to interact with MongoDB databases.

At-a-glance, most will enjoy MongoDBI for its ability to easily model classes while leveraging the power of MongoDB's schemaless and expeditious document-based design, dynamic queries, and atomic modifier operations.

Also noteworthy is MongoDBI's ease-of-use, chainable search facilities (filters), automated indexing, moose-integration (inheritance support, etc), lean document updates via dirty field tracking, and ability for each class to be configured to use a different database and connection, etc.

`MongoDBx-Class`

`MongoDBx-Class` is an ORM for MongoDB databases. MongoDBx::Class takes advantage of the fact that Perl's MongoDB driver is Moose-based to extend and tweak the driver's behavior, instead of wrapping it. This means MongoDBx::Class does not define its own syntax, so you simply use it exactly as you would the MongoDB driver directly. That said, MongoDBx::Class adds some sugar that enhances and simplifies the syntax unobtrusively (either use it or don't). Thus, it is relatively easy to convert your current MongoDB applications to MongoDBx::Class. A collection in MongoDBx::Class isa('MongoDB::Collection'), a database in MongoDBx::Class isa('MongoDB::Database'), etc. Created by Ido Perlmuter.

`MongoX`

MongoX - DSL sugar for MongoDB

`OOP Perl CMS`

`OOP Perl CMS` is based on Khurt Williams' Object Oriented Perl methodology and can be used as a basic CMS framework or as a basis for your own CMS system. It uses Apache & mod_perl with MongoDB backend. Created by Waitman Gobble.

Contributing to the Perl Driver

The easiest way to contribute is to file bugs and feature requests on Jira.

If you would like to help code the driver, read on...

Finding Something to Help With

Fixing Bugs

You can choose a bug on Jira and fix it. Make a comment that you're working on it, to avoid overlap.

Writing Tests

The driver could use a lot more tests. We would be grateful for any and all tests people would like to write.

Adding Features

If you think a feature is missing from the driver, you're probably right. Check on IRC or the mailing list, then go ahead and create a Jira case and add the feature. The Perl driver was a bit neglected for a while (although it's now getting a lot of TLC) so it's missing a lot of things that the other drivers have. You can look through their APIs for ideas.

Contribution Guidlines

The best way to make changes is to create an account on [Github], fork the driver, make your improvements, and submit a merge request.

To make sure your changes are approved and speed things along:

- Write tests. Lots of tests.
- Document your code.
- Write POD, when applicable.

Bonus (for C programmers, particularly):

- Make sure your change works on Perl 5.8, 5.10, Windows, Mac, Linux, etc.

Code Layout

The important files:

```
| perl_mongo.c # serialization/deserialization
| mongo_link.c # connecting to, sending to, and receiving from the database
- lib
   - MongoDB
      | Connection.pm # connection, queries, inserts... everything comes through here
      | Database.pm
      | Collection.pm
      | Cursor.pm
      | OID.pm
      | GridFS.pm
      - GridFS
         | File.pm
- xs
   | Mongo.xs
   | Connection.xs
   | Cursor.xs
   | OID.xs
```

Perl Tutorial

Redirection Notice
This page should redirect to http://search.cpan.org/dist/MongoDB/lib/MongoDB/Tutorial.pod.

Writing Drivers and Tools

See Also

- Mongo Query Language
- mongosniff
- --objcheck command line parameter

Overview - Writing Drivers and Tools

This section contains information for developers that are working with the low-level protocols of Mongo - people who are writing drivers and higher-level tools.

Documents of particular interest :

BSON http://bsonspec.org	Description of the BSON binary document format. Fundamental to how Mongo and it's client software works.
Mongo Wire Protocol	Specification for the basic socket communications protocol used between Mongo and clients.
Mongo Driver Requirements	Description of what functionality is expected from a Mongo Driver
GridFS Specification	Specification of GridFS - a convention for storing large objects in Mongo
Mongo Extended JSON	Description of the extended JSON protocol for the REST-ful interface (ongoing development)

Additionally we recommend driver authors take a look at existing driver source code as an example.

- Back to driver development home page
- Drivers Home Page

Mongo Driver Requirements

This is a high-level list of features that a driver for MongoDB might provide. We attempt to group those features by priority. This list should be taken with a grain of salt, and probably used more for inspiration than as law that must be adhered to. A great way to learn about implementing a driver is by reading the source code of any of the existing drivers, especially the ones listed as "mongodb.org supported".

High priority

- BSON serialization/deserialization
- full cursor support (e.g. support OP_GET_MORE operation)
- close exhausted cursors via OP_KILL_CURSORS
- support for running database commands
- handle query errors
- convert all strings to UTF-8 (part of proper support for BSON)
- hint, explain, count, $where
- database profiling: set/get profiling level, get profiling info
- advanced connection management (replica sets, slave okay)
- automatic reconnection

Medium priority

- validate a collection in a database
- buffer pooling
- Tailable cursor support

A driver should be able to connect to a single server. By default this must be `localhost:27017`, and must also allow the server to be specified by hostname and port.

```
Mongo m = new Mongo();   // go to localhost, default port
```

```
Mongo m = new Mongo(String host, int port);
```

How the driver does this is up to the driver - make it idiomatic. However, a driver should make it explicit and clear what is going on.

Replica Sets

A driver must be able to support "Replica-Set" configurations, where more than one mongod servers are specified, and configured for hot-failover.

The driver should determine which of the nodes is the current master, and send all operations to that server. In the event of an error, either socket error or a "not a master" error, the driver must restart the determination process.

1. **Cluster Mode** Connect to master in master-slave cluster

```
ServerCluster sc = new ServerCluster(INETAddr...); // again, give one and discover?
Mongo m = new Mongo(sc);
```

Connect to slave in read-only mode in master-slave cluster

```
ServerCluster sc = new ServerCluster(INETAddr...); // again, give one and discover?
sc.setTarget(...)
Mongo m = new Mongo(sc);

or maybe make it like *Default/Simple* w/ a flag?
```

Other than that, we need a way to get a DB object :

```
Mongo m = new Mongo();

DB db = m.getDB(name);
```

And a list of db names (useful for tools...) :

```
List<String> getDBNameList();
```

Database Object

Simple operations on a database object :

```
/**
 *  get name of database
 */
String    dbName = db.getName();

/**
 * Get a list of all the collection names in this database
 */
List<String> cols = db.getCollectionNames();

/**
 * get a collection object.  Can optionally create it if it
 *  doesn't exist, or just  be strict.  (XJDM has strictness as an option)
 */
Collection coll = db.getCollection(string);

/**
 * Create a collection w/ optional options.  Can fault
 * if the collection exists, or can just return it if it already does
 */
Collection coll = db.createCollection( string);
Collection coll = db.createCollection( string, options);

/**
 * Drop a collection by its name or by collection object.
 * Driver could invalidate any outstanding Collection objects
 * for that collection, or just hope for the best.
 */
boolean  b  = db.dropCollection(name);
boolean  b  = db.dropCollection(Collection);

/**
 * Execute a command on the database, returning the
 * BSON doc with the results
 */
Document d = db.executeCommand(command);

/**
 * Close the [logical] database
 */
void  db.close();

/**
 * Erase / drop an entire database
 */
bool dropDatabase(dbname)
```

Database Administration

These methods have to do with database metadata: profiling levels and collection validation. Each admin object is associated with a database. These methods could either be built into the Database class or provided in a separate Admin class whose instances are only available from a database instance.

```
/* get an admin object from a database object. */
Admin admin = db.getAdmin();

/**
 * Get profiling level. Returns one of the strings "off", "slowOnly", or
 * "all". Note that the database returns an integer. This method could
 * return an int or an enum instead --- in Ruby, for example, we return
 * symbols.
 */
String profilingLevel = admin.getProfilingLevel();

/**
 * Set profiling level. Takes whatever getProfilingLevel() returns.
 */
admin.setProfilingLevel("off");

/**
 * Retrieves the database's profiling info.
 */
Document profilingInfo = admin.getProfilingInfo();

/**
 * Returns true if collection is valid; raises an exception if not.
 */
boolean admin.validateCollection(collectionName);
```

Collection Basic Ops

```
/**
 *   full query capabilities - limit, skip, returned fields, sort, etc
 */
Cursor        find(...);

void          insert(...) // insert one or more objects into the collection, local variants on args
void          remove(query) // remove objects that match the query
void          update(selector, modifier)  // modify all objects that match selector w/ modifier object
void          updateFirst(selector, object)   // replace first object that match selector w/ specified
object
void          upsert(selector, object)   // replace first object that matches, or insert
long          getCount();
long          getCount(query);
```

Index Operations

```
void          createIndex( index_info)
void          dropIndex(name)
void          dropIndexes()
List<info>    getIndexInformation()
```

Misc Operations

```
document      explain(query)
options       getOptions();
string        getName();
void          close();
```

Cursor Object

```
document      getNextDocument()
iterator      getIterator()  // again, local to language
bool          hasMore()
void          close()
```

Spec, Notes and Suggestions for Mongo Drivers

Assume that the BSON objects returned from the database may be up to 16MB. This size may change over time.

See Also

- Driver Requirements
- BSON
- The main Database Internals page

Feature Checklist for Mongo Drivers

Functionality Checklist

This section lists tasks the driver author might handle.

Essential

- BSON serialization/deserialization
- Basic operations: `query`, `insert`, `update`, `remove`, `ensureIndex`, `findOne`, `limit`, `sort`
- Fetch more data from a cursor when necessary (dbGetMore)
- Sending of KillCursors operation when use of a cursor has completed (ideally for efficiently these are sent in batches)
- Convert all strings to utf8
- Authentication

Recommended

- automatic _id generation
- Database $cmd support and helpers
- Detect { `$err: ...` } response from a db query and handle appropriately --see Error Handling in Mongo Drivers
- [Automatically connect to proper server, and failover], when connecting to a Replica Set
- `ensureIndex` commands should be cached to prevent excessive communication with the database. (Or, the driver user should be informed that `ensureIndex` is not a lightweight operation for the particular driver.)
- Support detecting max BSON size on connection (e.g., using `buildinfo` or `isMaster` commands) and allowing users to insert docs up to that size.

More Recommended

- lasterror helper functions
- `count()` helper function
- `$where` clause
- `eval()`
- File chunking (GridFS)
- hint fields
- explain helper

More Optional

- `addUser`, `logout` helpers
- Allow client user to specify Option_SlaveOk for a query
- Tailable cursor support
- In/out buffer pooling (if implementing in a garbage collected languages)

More Optional

- [connection pooling]
- Automatic reconnect on connection failure
- DBRef Support:
    - Ability to generate easily
    - Automatic traversal

See Also

- The Driver and Integration Center for information about the latest drivers
- The [top page] for this section
- The main Database Internals page
- The starting point for all Home

Conventions for Mongo Drivers

Interface Conventions

It is desirable to keep driver interfaces consistent when possible. Of course, idioms vary by language, and when they do adaptation is appropriate. However, when the idiom is the same, keeping the interfaces consistent across drivers is desirable.

Terminology

In general, use these terms when naming identifiers. Adapt the names to the normal "punctuation" style of your language `-- foo_bar` in C might be `fooBar` in Java.

- *database* - what does this mean?
- *collection*
- *index*

Driver Testing Tools

Object IDs

- driverOIDTest for testing toString

```
> db.runCommand( { "driverOIDTest" : new ObjectId() } )
{
        "oid" : ObjectId("4b8991f221752a6e61a88267"),
        "str" : "4b8991f221752a6e61a88267",
        "ok" : 1
}
```

Mongo Wire Protocol

- Introduction
- Messages Types and Formats
    - Standard Message Header
    - Request Opcodes
- Client Request Messages
    - OP_UPDATE
    - OP_INSERT
    - OP_QUERY
    - OP_GETMORE
    - OP_DELETE
    - OP_KILL_CURSORS
    - OP_MSG
- Database Response Messages
    - OP_REPLY

Introduction

The Mongo Wire Protocol is a simple socket-based, request-response style protocol. Clients communicate with the database server through a regular TCP/IP socket.

> **Default Socket Port**
> The default port is 27017, but this is configurable and will vary.

Clients should connect to the database with a regular TCP/IP socket. Currently, there is no connection handshake.

To describe the message structure, a C-like `struct` is used. The types used in this document (`cstring`, `int32`, etc.) are the same as those defined in the BSON specification. The standard message header is typed as `MsgHeader`. Integer constants are in capitals (e.g. ZERO for the integer value of 0).

In the case where more than one of something is possible (like in a OP_INSERT or OP_KILL_CURSORS), we again use the notation from the BSON specification (e.g. `int64*`). This simply indicates that one or more of the specified type can be written to the socket, one after another.

Byte Ordering
Note that like BSON documents, all data in the mongo wire protocol is little-endian.

Messages Types and Formats

TableOfContents

There are two types of messages, client requests and database responses, each having a slightly different structure.

Standard Message Header

In general, each message consists of a standard message header followed by request-specific data. The standard message header is structured as follows :

```
struct MsgHeader {
    int32   messageLength; // total message size, including this
int32   requestID;     // identifier for this message
int32   responseTo;    // requestID from the original request
//   (used in reponses from db)
int32   opCode;        // request type - see table below
}
```

`messageLength` : This is the total size of the message in bytes. This total includes the 4 bytes that holds the message length.

`requestID` : This is a client or database-generated identifier that uniquely identifies this message. For the case of client-generated messages (e.g. OP_QUERY and CONTRIB:OP_GET_MORE), it will be returned in the `responseTo` field of the CONTRIB:OP_REPLY message. Along with the `reponseTo` field in responses, clients can use this to associate query responses with the originating query.

`responseTo` : In the case of a message from the database, this will be the requestID taken from the CONTRIB:OP_QUERY or CONTRIB:OP_GET_MORE messages from the client. Along with the `requestID` field in queries, clients can use this to associate query responses with the originating query.

`opCode` : Type of message. See the table below in the next section.

Request Opcodes

TableOfContents

The following are the currently supported opcodes :

Opcode Name	opCode value	Comment
OP_REPLY	1	Reply to a client request. responseTo is set
OP_MSG	1000	generic msg command followed by a string
OP_UPDATE	2001	update document
OP_INSERT	2002	insert new document
RESERVED	2003	formerly used for OP_GET_BY_OID
OP_QUERY	2004	query a collection
OP_GETMORE	2005	Get more data from a query. See Cursors
OP_DELETE	2006	Delete documents

OP_KILL_CURSORS	2007	Tell database client is done with a cursor

Client Request Messages

TableOfContents

Clients can send all messages except for CONTRIB:OP_REPLY. This is reserved for use by the database.

Note that only the CONTRIB:OP_QUERY and CONTRIB:OP_GET_MORE messages result in a response from the database. There will be no response sent for any other message.

You can determine if a message was successful with a getLastError command.

OP_UPDATE

The OP_UPDATE message is used to update a document in a collection. The format of a OP_UPDATE message is

```
struct OP_UPDATE {
    MsgHeader header;             // standard message header
int32     ZERO;               // 0 - reserved for future use
cstring   fullCollectionName; // "dbname.collectionname"
int32     flags;              // bit vector. see below
document  selector;           // the query to select the document
document  update;             // specification of the update to perform
}
```

`fullCollectionName` : The full collection name. The full collection name is the concatenation of the database name with the collection name, using a "." for the concatenation. For example, for the database "foo" and the collection "bar", the full collection name is "foo.bar".

`flags` :

bit num	name	description
0	Upsert	If set, the database will insert the supplied object into the collection if no matching document is found.
1	MultiUpdate	If set, the database will update all matching objects in the collection. Otherwise only updates first matching doc.
2-31	Reserved	Must be set to 0.

`selector` : BSON document that specifies the query for selection of the document to update.

`update` : BSON document that specifies the update to be performed. For information on specifying updates see the documentation on Updating.

There is no response to an OP_UPDATE message.

OP_INSERT

The OP_INSERT message is used to insert one or more documents into a collection. The format of the OP_INSERT message is

```
struct {
    MsgHeader header;             // standard message header
int32     flags;              // bit vector - see below
cstring   fullCollectionName; // "dbname.collectionname"
document* documents;          // one or more documents to insert into the collection
}
```

`fullCollectionName` : The full collection name. The full collection name is the concatenation of the database name with the collection name, using a "." for the concatenation. For example, for the database "foo" and the collection "bar", the full collection name is "foo.bar".

`documents` : One or more documents to insert into the collection. If there are more than one, they are written to the socket in sequence, one after another.

`flags` :

bit num	name	description

0	ContinueOnError	If set, the database will not stop processing a bulk insert if one fails (eg due to duplicate IDs). This makes bulk insert behave similarly to a series of single inserts, except lastError will be set if any insert fails, not just the last one. If multiple errors occur, only the most recent will be reported by getLastError. (new in 1.9.1)
1-31	Reserved	Must be set to 0.

There is no response to an OP_INSERT message.

OP_QUERY

The OP_QUERY message is used to query the database for documents in a collection. The format of the OP_QUERY message is :

```
struct OP_QUERY {
    MsgHeader header;               // standard message header
int32     flags;                    // bit vector of query options.  See below for details.
cstring   fullCollectionName;     // "dbname.collectionname"
int32     numberToSkip;           // number of documents to skip
int32     numberToReturn;         // number of documents to return
//  in the first OP_REPLY batch
document  query;                  // query object.  See below for details.
[ document  returnFieldSelector; ] // Optional. Selector indicating the fields
//  to return.  See below for details.
}
```

`flags` :

bit num	name	description
0	Reserved	Must be set to 0.
1	TailableCursor	Tailable means cursor is not closed when the last data is retrieved. Rather, the cursor marks the final object's position. You can resume using the cursor later, from where it was located, if more data were received. Like any "latent cursor", the cursor may become invalid at some point (CursorNotFound) – for example if the final object it references were deleted.
2	SlaveOk	Allow query of replica slave. Normally these return an error except for namespace "local".
3	OplogReplay	Internal replication use only - driver should not set
4	NoCursorTimeout	The server normally times out idle cursors after an inactivity period (10 minutes) to prevent excess memory use. Set this option to prevent that.
5	AwaitData	Use with TailableCursor. If we are at the end of the data, block for a while rather than returning no data. After a timeout period, we do return as normal.
6	Exhaust	Stream the data down full blast in multiple "more" packages, on the assumption that the client will fully read all data queried. Faster when you are pulling a lot of data and know you want to pull it all down. Note: the client is not allowed to not read all the data unless it closes the connection.
7	Partial	Get partial results from a `mongos` if some shards are down (instead of throwing an error)
8-31	Reserved	Must be set to 0.

`fullCollectionName` : The full collection name. The full collection name is the concatenation of the database name with the collection name, using a "." for the concatenation. For example, for the database "foo" and the collection "bar", the full collection name is "foo.bar".

`numberToSkip` : Sets the number of documents to omit - starting from the first document in the resulting dataset - when returning the result of the query.

`numberToReturn` : Limits the number of documents in the **first** CONTRIB:OP_REPLY message to the query. However, the database will still establish a cursor and return the `cursorID` to the client if there are more results than `numberToReturn`. If the client driver offers 'limit' functionality (like the SQL **LIMIT** keyword), then it is up to the client driver to ensure that no more than the specified number of document are returned to the calling application. If `numberToReturn` is 0, the db will use the default return size. If the number is negative, then the database will return that number and close the cursor. No futher results for that query can be fetched. If `numberToReturn` is 1 the server will treat it as -1 (closing the cursor automatically).

`query` : BSON document that represents the query. The query will contain one or more elements, all of which must match for a document to be included in the result set. Possible elements include `$query`, `$orderby`, `$hint`, `$explain`, and `$snapshot`.

`returnFieldsSelector` : OPTIONAL BSON document that limits the fields in the returned documents. The returnFieldsSelector contains one

or more elements, each of which is the name of a field that should be returned, and and the integer value 1. In JSON notation, a returnFieldsSelector to limit to the fields "a", "b" and "c" would be :

```
{ a : 1, b : 1, c : 1}
```

The database will respond to an OP_QUERY message with an CONTRIB:OP_REPLY message.

OP_GETMORE

The OP_GETMORE message is used to query the database for documents in a collection. The format of the OP_GETMORE message is :

```
struct {
    MsgHeader header;             // standard message header
int32     ZERO;               // 0 - reserved for future use
cstring   fullCollectionName; // "dbname.collectionname"
int32     numberToReturn;     // number of documents to return
int64     cursorID;           // cursorID from the OP_REPLY
}
```

`fullCollectionName` : The full collection name. The full collection name is the concatenation of the database name with the collection name, using a "." for the concatenation. For example, for the database "foo" and the collection "bar", the full collection name is "foo.bar".

`numberToReturn` : Limits the number of documents in the **first** CONTRIB:OP_REPLY message to the query. However, the database will still establish a cursor and return the `cursorID` to the client if there are more results than `numberToReturn`. If the client driver offers 'limit' functionality (like the SQL **LIMIT** keyword), then it is up to the client driver to ensure that no more than the specified number of document are returned to the calling application. If `numberToReturn` is 0, the db will used the default return size.

`cursorID` : Cursor identifier that came in the CONTRIB:OP_REPLY. This must be the value that came from the database.

The database will respond to an OP_GETMORE message with an CONTRIB:OP_REPLY message.

OP_DELETE

The OP_DELETE message is used to remove one or more documents from a collection. The format of the OP_DELETE message is :

```
struct {
    MsgHeader header;             // standard message header
int32     ZERO;               // 0 - reserved for future use
cstring   fullCollectionName; // "dbname.collectionname"
int32     flags;              // bit vector - see below for details.
document  selector;           // query object.  See below for details.
}
```

`fullCollectionName` : The full collection name. The full collection name is the concatenation of the database name with the collection name, using a "." for the concatenation. For example, for the database "foo" and the collection "bar", the full collection name is "foo.bar".

`flags` :

bit num	name	description
0	SingleRemove	If set, the database will remove only the first matching document in the collection. Otherwise all matching documents will be removed.
1-31	Reserved	Must be set to 0.

`selector` : BSON document that represent the query used to select the documents to be removed. The selector will contain one or more elements, all of which must match for a document to be removed from the collection. Please see $$$ TODO QUERY for more information.

There is no reponse to an OP_DELETE message.

OP_KILL_CURSORS

The OP_KILL_CURSORS message is used to close an active cursor in the database. This is necessary to ensure that database resources are reclaimed at the end of the query. The format of the OP_KILL_CURSORS message is :

```
struct {
    MsgHeader header;            // standard message header
int32     ZERO;              // 0 - reserved for future use
int32     numberOfCursorIDs; // number of cursorIDs in message
int64*    cursorIDs;         // sequence of cursorIDs to close
}
```

`numberOfCursorIDs` : The number of cursors that are in the message.

`cursorIDs` : "array" of cursor IDs to be closed. If there are more than one, they are written to the socket in sequence, one after another.

Note that if a cursor is read until exhausted (read until OP_QUERY or OP_GETMORE returns zero for the cursor id), there is no need to kill the cursor.

OP_MSG

Deprecated. OP_MSG sends a diagnostic message to the database. The database sends back a fixed resonse. The format is

```
struct {
    MsgHeader header;  // standard message header
cstring   message; // message for the database
}
```

Drivers do not need to implement OP_MSG.

Database Response Messages

TableOfContents

OP_REPLY

The OP_REPLY message is sent by the database in response to an CONTRIB:OP_QUERY or CONTRIB:OP_GET_MORE message. The format of an OP_REPLY message is:

```
struct {
    MsgHeader header;         // standard message header
int32     responseFlags;  // bit vector - see details below
int64     cursorID;       // cursor id if client needs to do get more's
int32     startingFrom;   // where in the cursor this reply is starting
int32     numberReturned; // number of documents in the reply
document* documents;      // documents
}
```

`responseFlags` :

bit num	name	description
0	CursorNotFound	Set when getMore is called but the cursor id is not valid at the server. Returned with zero results.
1	QueryFailure	Set when query failed. Results consist of one document containing an "$err" field describing the failure.
2	ShardConfigStale	Drivers should ignore this. Only mongos will ever see this set, in which case, it needs to update config from the server.
3	AwaitCapable	Set when the server supports the AwaitData Query option. If it doesn't, a client should sleep a little between getMore's of a Tailable cursor. Mongod version 1.6 supports AwaitData and thus always sets AwaitCapable.
4-31	Reserved	Ignore

`cursorID` : The cursorID that this OP_REPLY is a part of. In the event that the result set of the query fits into one OP_REPLY message, `cursorID` will be 0. This `cursorID` must be used in any CONTRIB:OP_GET_MORE messages used to get more data, and also must be closed by the client when no longer needed via a CONTRIB:OP_KILL_CURSORS message.

BSON

- bsonspec.org
- BSON and MongoDB
- Language-Specific Examples
  - C
  - C++
  - Java
  - PHP
  - Python
  - Ruby
- MongoDB Document Types

bsonspec.org

BSON is a bin-ary-en-coded seri-al-iz-a-tion of JSON-like doc-u-ments. BSON is designed to be lightweight, traversable, and efficient. BSON, like JSON, supports the embedding of objects and arrays within other objects and arrays. See bsonspec.org for the spec and more information in general.

BSON and MongoDB

MongoDB uses BSON as the data storage and network transfer format for "documents".

BSON at first seems BLOB-like, but there exists an important difference: the Mongo database understands BSON internals. This means that MongoDB can "reach inside " BSON objects, even nested ones. Among other things, this allows MongoDB to build indexes and match objects against query expressions on both top-level and nested BSON keys.

See also: the BSON blog post and BSON and Data Interchange

Language-Specific Examples

We often map from a language's "dictionary" type – which may be its native objects – to BSON. The mapping is particularly natural in dynamically typed languages:

```
JavaScript: {"foo" : "bar"}
Perl: {"foo" => "bar"}
PHP: array("foo" => "bar")
Python: {"foo" : "bar"}
Ruby: {"foo" => "bar"}
Java: DBObject obj = new BasicDBObject("foo", "bar");
```

C

```
bson b;
bson_buffer buf;
bson_buffer_init( &buf )
bson_append_string( &buf, "name", "Joe" );
bson_append_int( &buf, "age", 33 );
bson_from_buffer( &b, &buf );
bson_print( &b );
```

See http://github.com/mongodb/mongo-c-driver/blob/master/src/bson.h for more information.

C++

```
BSONObj p = BSON( "name" << "Joe" << "age" << 33 );
cout << p.toString() << endl;
cout << p["age"].number() << endl;
```

See the BSON section of the C++ Tutorial for more information.

Java

```
BasicDBObject doc = new BasicDBObject();
doc.put("name", "MongoDB");
doc.put("type", "database");
doc.put("count", 1);
BasicDBObject info = new BasicDBObject();
info.put("x", 203);
info.put("y", 102);
doc.put("info", info);
coll.insert(doc);
```

PHP

The PHP driver includes `bson_encode` and `bson_decode` functions. `bson_encode` takes any PHP type and serializes it, returning a string of bytes:

```
$bson = bson_encode(null);
$bson = bson_encode(true);
$bson = bson_encode(4);
$bson = bson_encode("hello, world");
$bson = bson_encode(array("foo" => "bar"));
$bson = bson_encode(new MongoDate());
```

Mongo-specific objects (`MongoId`, `MongoDate`, `MongoRegex`, `MongoCode`) will be encoded in their respective BSON formats. For other objects, it will create a BSON representation with the key/value pairs you would get by running `for ($object as $key => $value)`.

`bson_decode` takes a string representing a BSON object and parses it into an associative array.

Python

```
>>> from bson import BSON
>>> bson_string = BSON.encode({"hello": "world"})
>>> bson_string
'\x16\x00\x00\x00\x02hello\x00\x06\x00\x00\x00world\x00\x00'
>>> bson_string.decode()
{u'hello': u'world'}
```

PyMongo also supports "ordered dictionaries" through the **bson.son** module. The **BSON** class can handle **SON** instances using the same methods you would use for regular dictionaries. Python2.7's collections.OrderedDict is also supported.

Ruby

There are now two gems that handle BSON-encoding: bson and bson_ext. These gems can be used to work with BSON independently of the MongoDB Ruby driver.

```
irb
>> require 'rubygems'
=> true
>> require 'bson'
=> true
>> doc = {:hello => "world"}
>> bson = BSON.serialize(doc).to_s
=> "\026\000\000\000\002hello\000\006\000\000\000world\000\000"
>> BSON.deserialize(bson.unpack("C*"))
=> {"hello" => "world"}
```

The BSON class also supports ordered hashes. Simply construct your documents using the OrderedHash class, also found in the MongoDB Ruby Driver.

MongoDB Document Types

MongoDB uses BSON documents for three things:

1. Data storage (user documents). These are the regular JSON-like objects that the database stores for us. These BSON documents are

sent to the database via the INSERT operation. User documents have limitations on the "element name" space due to the usage of special characters in the JSON-like query language.
    a. A user document element name cannot begin with "$".
    b. A user document element name cannot have a "." in the name.
    c. The element name "_id" is reserved for use as a primary key id, but you can store anything that is unique in that field. The database expects that drivers will prevent users from creating documents that violate these constraints.
2. Query "Selector" Documents : Query documents (or selectors) are BSON documents that are used in QUERY, DELETE and UPDATE operations. They are used by these operations to match against documents. Selector objects have no limitations on the "element name" space, as they must be able to supply special "marker" elements, like "$where" and the special "command" operations.
3. "Modifier" Documents : Documents that contain 'modifier actions' that modify user documents in the case of an update (see Updating).

Mongo Extended JSON

MongoDB's import and export utilitites, as well as its basic REST interface, support JSON-style access to MongoDB BSON documents. Special representations are used for BSON types that do not have obvious JSON mappings, and multiple representations are allowed for some such types. The REST interface supports three different modes for document output { Strict, JS, Shell }, which serve to control the representations used. MongoDB can understand all of these representations in REST input.

- **Strict** mode produces output conforming to the JSON RFC spec http://www.json.org.
- **JS** mode produces output that can be processed by most Javascript interpreters.
- **Shell** / "TenGen" mode produces output that the mongo shell understands. This is basically an enhanced Javascript format.

The following BSON types are represented using special conventions:

BSON Data Type	Strict	JS	Mongo Shell ("TenGen")	Notes
data_binary	`{ "$binary" : "<bindata>" , "$type" : "<t>" }`	`{ "$binary" : "<bindata>", "$type" : "<t>" }`	`{ "$binary" : "<bindata>", "$type" : "<t>" }`	<bindata> base64 representa of a binary string. <t> hexadecim representa of a single indicating t data type.
data_date	`{"$date" :<date>}`	`Date( <date> )`	`Date( <date> )`	<date> is t JSON representa of a 64 bit signed inte for millisec since epoc (unsigned before vers 1.9.1).
data_timestamp	`{ "$timestamp" : {"t": <t>, "i": <i>} }`	`{ "$timestamp" : {"t": <t>, "i": <i>} }`	`Timestamp(<t>,<i>)`	<t> is the J representa of a 32 bit unsigned integer for seconds si epoch. <i> 32 bit unsi integer for increment.

data_regex	`{ "$regex" : "<sRegex>", "$options" : "<sOptions>" }`	`/<jRegex>/<jOptions>`	`/<jRegex>/<jOptions>`	<sRegex> string of va JSON characters <jRegex> i string that contain val JSON characters unescapec characters may not cc unescapec characters <sOptions: string containing letters of th alphabet. <jOptions> string that contain on characters 'i', 'm' and (added in v Because th and TenGe representa support a limited ran options, ar nonconforn options wil dropped w converting this representa
data_oid	`{ "$oid" : "<id>" }`	`{ "$oid" : "<id>" }`	`ObjectId( "<id>" )`	<id> is a 2 character hexadecim string.
data_ref	`{ "$ref": "<name>", "$id" : "<id>" }`	`{"$ref":"<name>", "$id":"<id>"}`	`Dbref("<name>", "<id>")`	<name> is string of va JSON characters is a 24 character hexadecim string. In t Strict and representa the Strict representa for a data_ can be use place of "<
data_undefined	`{ "$undefined" :true}`	`undefined`	`undefined`	The representa for the javascript/l undefined

GridFS Specification

- Introduction
- Specification
  - Storage Collections

        - files
        - chunks
    - Indexes

Introduction

GridFS is a storage specification for large objects in MongoDB. It works by splitting large object into small chunks, usually 256k in size. Each chunk is stored as a separate document in a `chunks` collection. Metadata about the file, including the filename, content type, and any optional information needed by the developer, is stored as a document in a `files` collection.

So for any given file stored using GridFS, there will exist one document in `files` collection and one or more documents in the `chunks` collection.

If you're just interested in using GridFS, see the docs on storing files. If you'd like to understand the GridFS implementation, read on.

Specification

Storage Collections

GridFS uses two collections to store data:

- `files` contains the object metadata
- `chunks` contains the binary chunks with some additional accounting information

In order to make more than one GridFS namespace possible for a single database, the files and chunks collections are named with a prefix. By default the prefix is `fs.`, so any default GridFS store will consist of collections named `fs.files` and `fs.chunks`. The drivers make it possible to change this prefix, so you might, for instance, have another GridFS namespace specifically for photos where the collections would be `photos.files` and `photos.chunks`.

Here's an example of the standard GridFS interface in Java:

```
/*
 * default root collection usage - must be supported
 */
GridFS myFS = new GridFS(myDatabase);             // returns a default GridFS (e.g. "fs" root
collection)
myFS.storeFile(new File("/tmp/largething.mpg"));   // saves the file into the "fs" GridFS store

/*
 * specified root collection usage - optional
 */

GridFS myContracts = new GridFS(myDatabase, "contracts");             // returns a GridFS where
"contracts" is root
myFS.retrieveFile("smithco", new File("/tmp/smithco_20090105.pdf"));  // retrieves object whose
filename is "smithco"
```

Note that the above API is for demonstration purposes only - this spec does not (at this time) recommend any API. See individual driver documentation for API specifics.

`files`

Documents in the `files` collection require the following fields:

```
{
  "_id" : <unspecified>,                  // unique ID for this file
"length" : data_number,                 // size of the file in bytes
"chunkSize" : data_number,              // size of each of the chunks.  Default is 256k
"uploadDate" : data_date,               // date when object first stored
"md5" : data_string                     // result of running the "filemd5" command on this file's
chunks
}
```

Any other desired fields may be added to the files document; common ones include the following:

```
{
  "filename" : data_string,              // human name for the file
"contentType" : data_string,            // valid mime type for the object
"aliases" : data_array of data_string,  // optional array of alias strings
"metadata" : data_object,               // anything the user wants to store
}
```

Note that the _id field can be of any type, per the discretion of the spec implementor.

`chunks`

The structure of documents from the `chunks` collection is as follows:

```
{
  "_id" : <unspecified>,         // object id of the chunk in the _chunks collection
"files_id" : <unspecified>,    // _id of the corresponding files collection entry
"n" : chunk_number,            // chunks are numbered in order, starting with 0
"data" : data_binary,          // the chunk's payload as a BSON binary type
}
```

Notes:

- The `_id` is whatever type you choose. As with any MongoDB document, the default will be a BSON object id.
- The `files_id` is a foreign key containing the `_id` field for the relevant `files` collection entry

Indexes

GridFS implementations should create a unique, compound index in the `chunks` collection for `files_id` and n. Here's how you'd do that from the shell:

```
db.fs.chunks.ensureIndex({files_id:1, n:1}, {unique: true});
```

This way, a chunk can be retrieved efficiently using it's `files_id` and n values:

```
cursor = db.fs.chunks.findOne({files_id: myFileID}).sort({n:1});
```

Implementing Authentication in a Driver

The current version of Mongo supports only very basic authentication. One authenticates a username and password in the context of a particular database. Once authenticated, the user has full read and write access to the database in question.

The `admin` database is special. In addition to several commands that are administrative being possible only on `admin`, authentication on `admin` gives one read and write access to all databases on the server. Effectively, `admin` access means root access to the db.

Note on a single socket we may authenticate for any number of databases, and as different users. This authentication persists for the life of the database connection (barring a `logout` command).

The Authentication Process

Authentication is a two step process. First the driver runs a `getnonce` command to get a nonce for use in the subsequent authentication. We can view a sample `getnonce` invocation from `dbshell`:

```
> db.$cmd.findOne({getnonce:1})
{ "nonce":"7268c504683936e1" , "ok":1
```

The nonce returned is a hex String.

The next step is to run an `authenticate` command for the database on which to authenticate. The authenticate command has the form:

```
{ authenticate : 1, user : username, nonce : nonce, key : digest }
```

where

- *username* is a username in the database's system.users collection;
- *nonce* is the nonce returned from a previous getnonce command;
- *digest* is the hex encoding of a MD5 message digest which is the MD5 hash of the concatenation of (*nonce*, *username*, *password_digest*), where *password_digest* is the user's password value in the pwd field of the associated user object in the database's system.users collection. pwd is the hex encoding of MD5(username + ":mongo:" + *password_text*).

Authenticate will return an object containing

```
{ ok : 1 }
```

when successful.

Details of why an authentication command failed may be found in the Mongo server's log files.

The following code from the Mongo Javascript driver provides an example implementation:

```
DB.prototype.addUser = function( username , pass ){
    var c = this.getCollection( "system.users" );

    var u = c.findOne( { user : username } ) || { user : username };
    u.pwd = hex_md5( username + ":mongo:" + pass );
    print( tojson( u ) );

    c.save( u );
}

DB.prototype.auth = function( username , pass ){
    var n = this.runCommand( { getnonce : 1 } );

    var a = this.runCommand(
        {
            authenticate : 1 ,
            user : username ,
            nonce : n.nonce ,
            key : hex_md5( n.nonce + username + hex_md5( username + ":mongo:" + pass ) )
        }
    );

    return a.ok;
}
```

Logout

Drivers may optionally implement the logout command which deauthorizes usage for the specified database for this connection. Note other databases may still be authorized.

Alternatively, close the socket to deauthorize.

```
> db.$cmd.findOne({logout:1})
{
    "ok" : 1.0
}
```

Replica Sets and Authentication

For drivers that support replica sets, extra care with replication is required.

When switching from one server in a replica set to another, for example in a failover situation, you must reauthenticate. Clients will likely want to cache authentication from the user so that the client can reauthenticate with the new server when appropriate.

Be careful also with operations such as Logout. If you log out from only some members of a replica set, that could be an issue.

Authenticating with a server in slave mode is allowed.

See Also

- Security and Authentication

Notes on Pooling for Mongo Drivers

Note that with the db write operations can be sent asynchronously or synchronously (the latter indicating a getlasterror request after the write).

When asynchronous, one must be careful to continue using the same connection (socket). This ensures that the next operation will not begin until after the write completes.

Pooling and Authentication

An individual socket connection to the database has associated authentication state. Thus, if you pool connections, you probably want a separate pool for each authentication case (db + username).

Pseudo-code

The following pseudo-code illustrates our recommended approach to implementing connection pooling in a driver's connection class. This handles authentication, grouping operations from a single "request" onto the same socket, and a couple of other gotchas:

```
class Connection:
   init(pool_size, addresses, auto_start_requests):
      this.pool_size = pool_size
      this.addresses = addresses
      this.auto_start_requests = auto_start_requests
      this.thread_map = {}
      this.locks = Lock[pool_size]
      this.sockets = Socket[pool_size]
      this.socket_auth = String[pool_size][]
      this.auth = {}

      this.find_master()

   find_master():
      for address in this.addresses:
         if address.is_master():
            this.master = address

   pick_and_acquire_socket():
      choices = random permutation of [0, ..., this.pool_size - 1]

      choices.sort(order: ascending,
                   value: size of preimage of choice under this.thread_map)

      for choice in choices:
         if this.locks[choice].non_blocking_acquire():
            return choice

      sock = choices[0]
      this.locks[sock].blocking_acquire()
      return sock

   get_socket():
      if this.thread_map[current_thread] >= 0:
         sock_number = this.thread_map[current_thread]
         this.locks[sock_number].blocking_acquire()
      else:
         sock_number = this.pick_and_lock_socket()
         if this.auto_start_requests or current_thread in this.thread_map:
            this.thread_map[current_thread] = sock_number
```

```
      if not this.sockets[sock_number]:
         this.sockets[sock_number] = Socket(this.master)

      return sock_number

   send_message_without_response(message):
      sock_number = this.get_socket()
      this.check_auth()
      this.sockets[sock_number].send(message)
      this.locks[sock_number].release()

   send_message_with_response(message):
      sock_number = this.get_socket()
      this.check_auth()
      this.sockets[sock_number].send(message)
      result = this.sockets[sock_number].receive()
      this.locks[sock_number].release()
      return result

   # start_request is only needed if auto_start_requests is False
   start_request():
      this.thread_map[current_thread] = -1

   end_request():
      delete this.thread_map[current_thread]

   authenticate(database, username, password):
      # TODO should probably make sure that these credentials are valid,
      # otherwise errors are going to be delayed until first op.
      this.auth[database] = (username, password)

   logout(database):
      delete this.auth[database]

   check_auth(sock_number):
      for db in this.socket_auth[sock_number]:
         if db not in this.auth.keys():
            this.sockets[sock_number].send(logout_message)
            this.socket_auth[sock_number].remove(db)
      for db in this.auth.keys():
         if db not in this.socket_auth[sock_number]:
            this.sockets[sock_number].send(authenticate_message)
            this.socket_auth[sock_number].append(db)

# somewhere we need to do error checking - if you get not master then everything
# in this.sockets gets closed and set to null and we call find_master() again.
```

```
# we also need to reset the socket_auth information - nothing is authorized yet
# on the new master.
```

See Also

- The Driver and Integration Center for information about the latest drivers
- The [top page] for this section
- The main Database Internals page
- The starting point for all Home

Connecting Drivers to Replica Sets

Ideally a MongoDB driver can connect to a cluster of servers which represent a replica set, and automatically find the right set member with which to communicate. Failover should be automatic too. The general steps are:

1. The user, when opening the connection, specifies host[:port] for one or more members of the set. Not all members need be specified -- in fact the exact members of the set might change over time. This list for the connect call is the *seed list*.
2. The driver then connects to all servers on the seed list, perhaps in parallel to minimize connect time. Send an ismaster command to each server.
3. When the server is in replSet mode, it will return a *hosts* field with all members of the set that are potentially eligible to serve data. The client should cache this information. Ideally this refreshes too, as the set's config could change over time.
4. Choose a server with which to communicate.
    a. If ismaster == true, that server is primary for the set. This server can be used for writes and immediately consistent reads.
    b. If secondary == true, that server is not primary, but is available for eventually consistent reads. In this case, you can use the *primary* field to see which server the master should be. (If primary is not set, you may want to poll other nodes at random; it is conceivable that the member to which we are talking is partitioned from the other members, and thus it cannot determine who is primary on its own. This is unlikely but possible.)
5. If an error occurs with the current connection, find the new primary and resume use there.

For example, if we run the ismaster command on a non-primary server, we might get something like:

```
> db.runCommand("ismaster")
{
        "ismaster" : false,
        "secondary" : true,
        "hosts" : [
                "ny1.acme.com",
                "ny2.acme.com",
                "sf1.acme.com"
        ],
        "passives" : [
             "ny3.acme.com",
             "sf3.acme.com"
        ],
        "arbiters" : [
            "sf2.acme.com",
        ]
        "primary" : "ny2.acme.com",
        "ok" : true
}
```

There are three servers with priority > 0 (*ny1*, *ny2*, and *sf1*), two passive servers (*ny3* and *sf3*), and an arbiter (*sf2*). The primary should be *ny2*, but the driver should call ismaster on that server before it assumes it is.

Error Handling in Mongo Drivers

If an error occurs on a query (or getMore operation), Mongo returns an error object instead of user data.

The error object has a first field guaranteed to have the reserved key `$err`. For example:

```
{ $err : "some error message" }
```

The `$err` value can be of any type but is usually a string.

Drivers typically check for this return code explicitly and take action rather than returning the object to the user. The query results flags include a

set bit when $err is returned.

```
/* db response format

Query or GetMore: // see struct QueryResult
int resultFlags;
  int64 cursorID;
  int startingFrom;
  int nReturned;
  list of marshalled JSObjects;

*/

struct QueryResult : public MsgData {
  enum {
    ResultFlag_CursorNotFound = 1, /* returned, with zero results, when getMore is called but the
cursor id is not valid at the server. */
    ResultFlag_ErrSet = 2          /* { $err : ... } is being returned */
  };
  ...
};
```

See Also

- The Driver and Integration Center for information about the latest drivers
- The [top page] for this section
- The main Database Internals page
- The starting point for all Home

Developer Zone

- Shell
- Manual

See The MongoDB Manual for complete and current documentation of MongoDB.

If you have a comment or question about anything, please contact us through IRC (freenode.net#mongodb) or the mailing list, rather than leaving a comment at the bottom of a page. It is easier for us to respond to you through those channels.

Introduction

MongoDB is a collection-oriented, schema-free document database.

By *collection-oriented*, we mean that data is grouped into sets that are called 'collections'. Each collection has a unique name in the database, and can contain an unlimited number of documents. Collections are analogous to tables in a RDBMS, except that they don't have any defined schema.

By *schema-free*, we mean that the database doesn't need to know anything about the structure of the documents that you store in a collection. In fact, you can store documents with different structure in the same collection if you so choose.

By *document*, we mean that we store data that is a structured collection of key-value pairs, where keys are strings, and values are any of a rich set of data types, including arrays and documents. We call this data format "BSON" for "Binary Serialized dOcument Notation."

MongoDB Operational Overview

MongoDB is a server process that runs on Linux, Windows and OS X. It can be run both as a 32 or 64-bit application. We recommend running in 64-bit mode, since Mongo is limited to a total data size of about 2GB for all databases in 32-bit mode.

The MongoDB process listens on port 27017 by default (note that this can be set at start time - please see Command Line Parameters for more information).

Clients connect to the MongoDB process, optionally authenticate themselves if security is turned on, and perform a sequence of actions, such as inserts, queries and updates.

MongoDB stores its data in files (default location is `/data/db/`), and uses memory mapped files for data management for efficiency.

MongoDB can also be configured for automatic data replication , as well as automatic fail-over .

For more information on MongoDB administration, please see Mongo Administration Guide.

MongoDB Functionality

As a developer, MongoDB drivers offer a rich range of operations:

- Queries: Search for documents based on either query objects or SQL-like "where predicates". Queries can be sorted, have limited return sizes, can skip parts of the return document set, and can also return partial documents.
- Inserts and Updates : Insert new documents, update existing documents.
- Index Management : Create indexes on one or more keys in a document, including substructure, deleted indexes, etc
- General commands : Any MongoDB operation can be managed via DB Commands over the regular socket.

Manual

For the manual see http://docs.mongodb.org/.

More docs on specific topics:

- Schema Design
  - Trees in MongoDB
  - Tweaking performance by document bundling during schema design
  - Using a Large Number of Collections
- Padding Factor
- Server-side Code Execution
- Explain
- Full Text Search in Mongo
- Tailable Cursors

Schema Design

- Embedding and Linking
- Collections
- Atomic Operations
- Indexes
- Sharding
- Example
- Summary of Best Practices
- More Details
  - Choosing Indexes
  - How Many Collections?
- See Also
  - Books
  - Blog posts
  - Related Doc Pages
  - Videos

Schema design in MongoDB is very different from schema design in a relational DBMS. However it is still very important and the first step towards building an application.

In relational data models, conceptually there is a "correct" design for a given entity relationship model independent of the use case. This is typically a third normal form normalization. One typically only diverges from this for performance reasons. In MongoDB, the schema design is not only a function of the data to be modeled but also of the use case. The schema design is optimized for our most common use case. This has pros and cons – that use case is then typically highly performant; however there is a bias in the schema which may make certain ad hoc queries a little less elegant than in the relational schema.

As we design the schema, the questions we must answer are:

1. When do we embed data versus linking (see below)? Our decisions here imply the answer to question #2:

2. How many collections do we have, and what are they?

3. When do we need atomic operations? These operations can be performed within the scope of a BSON document, but not across documents.

4. What indexes will we create to make query and updates fast?

5. How will we shard? What is the shard key?

Embedding and Linking

A key question when designing a MongoDB schema is when to *embed* and when to *link*. Embedding is the nesting of objects and arrays inside a BSON document. Links are references between documents.

There are no joins in MongoDB – distributed joins would be difficult on a 1,000 server cluster. Embedding is a bit like "prejoined" data. Operations within a document are easy for the server to handle; these operations can be fairly rich. Links in contrast must be processed client-side by the application; the application does this by issuing a follow-up query.

Generally, for "contains" relationships between entities, embedding should be be chosen. Use linking when not using linking would result in duplication of data.

Collections

Collections in MongoDB are analogous to tables in a relational database. Each collection contains *documents*. As mentioned above these documents can be fairly rich.

There is no explicit declaration of the fields within the documents of a collection. However there is a conceptual design from the schema designer of what those fields will be and how the documents in the collection will be structured. MongoDB does not require documents in a collection to have the same structure. However, in practice, most collections are highly homogenous. We can move away from this when we need to though; for example when adding a new field. In a situation such as that, an "alter table" style operation is not necessary.

Atomic Operations

Some problems require the ability to perform atomic operations. For example, simply incrementing a counter is often a case where one wants atomicity. MongoDB can also perform more complex operations such as that shown in the pseudocode below:

```
atomically {
  if( doc.credits > 5 ) {
    doc.credits -= 5;
    doc.debits += 5;
  }
}
```

Another example would be a user registration scenario. We would never want two users to register the same username simultaneously:

```
atomically {
  if( exists a document with username='jane' ) {
    print "username already in use please choose another";
  } else {
    insert a document with username='jane' in the users collection;
    print("thanks you have registered as user jane.");
  }
}
```

The key aspect here in terms of schema design is that our scope of atomicity / ACID properties is the document. Thus we need to assure that all fields relevant to the atomic operation are in the same document.

Indexes

MongoDB supports the declaration of indexes. Indexes in MongoDB are highly analogous to indexes in a relational database : they are needed for efficient query processing, and must be explicitly declared. Thus we need to think about what indexes we will define as part of the schema design process. Just like in a relational database, indexes can be added later – if we decide later to have a new one, we can do that.

Sharding

Another consideration for schema design is sharding. A BSON document (which may have significant amounts of embedding) resides on one and only one shard.

A collection may be sharded. When sharded, the collection has a shard key, which determines how the collection is partitioned among shards. Typically (but not always) queries on a sharded collection involve the shard key as part of the query expression.

The key here is that changing shard keys is difficult. You will want to choose the right key from the start.

Example

Let's consider an example, which is a content management system. The examples below use mongo shell syntax but could also be done in any programming language – just use the appropriate driver for that language.

Our content management system will have posts. Posts have authors. We'd like to support commenting and voting on posts. We'd also like posts to be taggable for searching.

One good schema design for this scenario would be to have two MongoDB collections, one called `posts` and one called `users`. This is what we will use for the example.

Our users have a few properties - a user id they registered with, their real name, and their karma. For example we could invoke:

```
> db.users.insert( { _id : "alex", name: { first:"Alex", last:"Benisson" }, karma : 1.0 } )
```

The `_id` field is always present in MongoDB, and is automically indexed with a unique key constraint. That is perfect for our usernames so we store them in the `_id` field. We don't have to though; we could instead make a username field and let the system automically generate a unique id.

Let's now consider posts. We'll assume some posts are already populated. Let's query one:

```
> db.posts.findOne()
{
  _id : ObjectId("4e77bb3b8a3e000000004f7a"),
  when : Date("2011-09-19T02:10:11.3Z",
  author : "alex",
  title : "No Free Lunch",
  text : "This is the text of the post.  It could be very long.",
  tags : [ "business", "ramblings" ],
  votes : 5,
  voters : [ "jane", "joe", "spencer", "phyllis", "li" ],
  comments : [
    { who : "jane", when : Date("2011-09-19T04:00:10.112Z"),
      comment : "I agree." },
    { who : "meghan", when : Date("2011-09-20T14:36:06.958Z"),
      comment : "You must be joking.  etc etc ..." }
  ]
}
```

It's interesting to contrast this with how we might design the same schema in a relation database. We would likely have a users collection and a posts collection. But in addition one would typically have a tags collection, a voters collection, and a comments collection. Grabbing all the information on a single post would then be a little involved. Here to grab the whole post we might execute:

```
> db.posts.findOne( { _id : ObjectId("4e77bb3b8a3e000000004f7a") } );
```

To get all posts written by alex:

```
> db.posts.find( { author : "alex" } )
```

If the above is a common query we would create an index on the author field:

```
> db.posts.ensureIndex( { author : 1 } )
```

The post documents can be fairly large. To get just the titles of the posts by alex :

```
> db.posts.find( { author : "alex" }, { title : 1 } )
```

We may want to search for posts by tag:

```
> // make and index of all tags so that the query is fast:
> db.posts.ensureIndex( { tags : 1 } )
> db.posts.find( { tags : "business" } )
```

What if we want to find all posts commented on by meghan?

```
> db.posts.find( { comments.who : "meghan" } )
```

Let's index that to make it fast:

```
> db.posts.ensureIndex( { "comments.who" : 1 } )
```

We track voters above so that no one can vote more than once. Suppose calvin wants to vote for the example post above. The following update operation will record calvin's vote. Because of the `$nin` sub-expression, if calvin has already voted, the update will have no effect.

```
> db.posts.update( { _id : ObjectId("4e77bb3b8a3e000000004f7a"),
                     voters : { $nin : "calvin" } },
                   { votes : { $inc : 1 }, voters : { $push : "calvin" } );
```

Note the above operation is atomic : if multiple users vote simultaneously, no votes would be lost.

Suppose we want to display the title of the latest post in the system as well as the full user name for the author. This is a case where we must use client-side linking:

```
> var post = db.posts.find().sort( { when : -1 } ).limit(1);
> var user = db.users.find( { _id : post.author } );
> print( post.title + " " + user.name.first + " " + user.name.last );
```

A final question we might ask about our example is how we would shard. If the users collection is small, we would not need to shard it at all. If posts is huge, we would shard it. We would need to choose a shard key. The key should be chosen based on the queries that will be common. We want those queries to involve the shard key.

- Sharding by _id is one option here.
- If finding the most recent posts is a very frequent query, we would then shard on the `when` field. (There is also an optimization trick which might work here.)

Summary of Best Practices

- "First class" objects, that are at top level, typically have their own collection.
- Line item detail objects typically are embedded.
- Objects which follow an object modelling "contains" relationship should generally be embedded.
- Many to many relationships are generally done by linking.
- Collections with only a few objects may safely exist as separate collections, as the whole collection is quickly cached in application server memory.
- Embedded objects are a bit harder to link to than "top level" objects in collections.
- It is more difficult to get a system-level view for embedded objects. When needed an operation of this sort is performed by using MongoDB's map/reduce facility.
- If the amount of data to embed is huge (many megabytes), you may reach the limit on size of a single object. See also GridFS.
- If performance is an issue, embed.

More Details

Choosing Indexes

A second aspect of schema design is index selection. As a general rule, *where you want an index in a relational database, you want an index in Mongo.*

- `The _id` field is automatically indexed.
- Fields upon which keys are looked up should be indexed.
- Sort fields generally should be indexed.

The MongoDB profiling facility provides useful information for where an index should be added that is missing.

Note that adding an index slows writes to a collection, but not reads. Use lots of indexes for collections with a high read : write ratio (assuming one does not mind the storage overage). For collections with more writes than reads, indexes are expensive as keys must be added to each index for each insert.

How Many Collections?

As Mongo collections are polymorphic, one could have a collection `objects` and put everything in it! This approach is taken by some object databases. This is not recommended in MongoDB for several reasons, mainly performance. Data within a single collection is roughly contiguous on disk. Thus, "table scans" of a collection are possible, and efficient. Just like in relational dbs, independent collections are very important for high throughput batch processing.

See Also

Books

- Document Design for MongoDB - O'Reilly Ebook
- More Books

Blog posts

- http://blog.fiesta.cc/post/11319522700/walkthrough-mongodb-data-modeling
- Blog post on dynamic schemas

Related Doc Pages

- Trees in MongoDB
- MongoDB Data Modeling and Rails
- Advanced Queries

Videos

- Schema Design Basics - MongoSF Presentation (May 2011)
- Schema Design at Scale - MongoSF Presentation (May 2011)
- Schema Design: Data as Documents - MongoSV Presentation (December 2010)

Schema Design Basics - Presentation from MongoSF (May 2011) - Slides

Schema Design by Example - Presentation at MongoSV (December 2011)

Trees in MongoDB

- Patterns
  - Full Tree in Single Document
  - Parent Links
  - Child Links
  - Array of Ancestors
  - Materialized Paths (Full Path in Each Node)
  - acts_as_nested_set
- Use Cases
  - Find Nodes by a Partial Path
- See Also

The best way to store a tree usually depends on the operations you want to perform; see below for some different options. In practice, most developers find that one of the "Full Tree in Single Document", "Parent Links", and "Array of Ancestors" patterns works best.

Patterns

Full Tree in Single Document

```
{
  comments: [
    {by: "mathias", text: "...", replies: []}
    {by: "eliot", text: "...", replies: [
      {by: "mike", text: "...", replies: []}
    ]}
  ]
}
```

Pros:

- Single document to fetch per page
- One location on disk for whole tree
- You can see full structure easily

Cons:

- Hard to search
- Hard to get back partial results
- Can get unwieldy if you need a huge tree. Further there is a limit on the size of documents in MongoDB – 16MB in v1.8 (limit may rise in future versions).

Parent Links

Storing all nodes in a single collection, with each node having the id of its parent, is a simple solution. The biggest problem with this approach is getting an entire subtree requires several query turnarounds to the database (or use of db.eval).

```
> t = db.tree1;

> t.find()
{ "_id" : 1 }
{ "_id" : 2, "parent" : 1 }
{ "_id" : 3, "parent" : 1 }
{ "_id" : 4, "parent" : 2 }
{ "_id" : 5, "parent" : 4 }
{ "_id" : 6, "parent" : 4 }

> // find children of node 4
> t.ensureIndex({parent:1})
> t.find( {parent : 4 } )
{ "_id" : 5, "parent" : 4 }
{ "_id" : 6, "parent" : 4 }
```

Child Links

Another option is storing the ids of all of a node's children within each node's document. This approach is fairly limiting, although ok if no operations on entire subtrees are necessary. It may also be good for storing graphs where a node has multiple parents.

```
> t = db.tree2
> t.find()
{ "_id" : 1, "children" : [ 2, 3 ] }
{ "_id" : 2 }
{ "_id" : 3, "children" : [ 4 ] }
{ "_id" : 4 }

> // find immediate children of node 3
> t.findOne({_id:3}).children
[ 4 ]

> // find immediate parent of node 3
> t.ensureIndex({children:1})
> t.find({children:3})
{ "_id" : 1, "children" : [ 2, 3 ] }
```

Array of Ancestors

Here we store all the ancestors of a node in an array. This makes a query like "get all descendents of x" fast and easy.

```
> t = db.mytree;

> t.find()
{ "_id" : "a" }
{ "_id" : "b", "ancestors" : [ "a" ], "parent" : "a" }
{ "_id" : "c", "ancestors" : [ "a", "b" ], "parent" : "b" }
{ "_id" : "d", "ancestors" : [ "a", "b" ], "parent" : "b" }
{ "_id" : "e", "ancestors" : [ "a" ], "parent" : "a" }
{ "_id" : "f", "ancestors" : [ "a", "e" ], "parent" : "e" }
{ "_id" : "g", "ancestors" : [ "a", "b", "d" ], "parent" : "d" }

> t.ensureIndex( { ancestors : 1 } )

> // find all descendents of b:
> t.find( { ancestors : 'b' })
{ "_id" : "c", "ancestors" : [ "a", "b" ], "parent" : "b" }
{ "_id" : "d", "ancestors" : [ "a", "b" ], "parent" : "b" }
{ "_id" : "g", "ancestors" : [ "a", "b", "d" ], "parent" : "d" }

> // get all ancestors of f:
> anc = db.mytree.findOne({_id:'f'}).ancestors
[ "a", "e" ]
> db.mytree.find( { _id : { $in : anc } } )
{ "_id" : "a" }
{ "_id" : "e", "ancestors" : [ "a" ], "parent" : "a" }
```

ensureIndex and MongoDB's multikey feature makes the above queries efficient.

In addition to the ancestors array, we also stored the direct parent in the node to make it easy to find the node's immediate parent when that is necessary.

Materialized Paths (Full Path in Each Node)

Materialized paths make certain query options on trees easy. We store the full path to the location of a document in the tree within each node. Usually the "array of ancestors" approach above works just as well, and is easier as one doesn't have to deal with string building, regular expressions, and escaping of characters. (Theoretically, materialized paths will be *slightly* faster.)

The best way to do this with MongoDB is to store the path as a string and then use regex queries. Simple regex expressions beginning with "^" can be efficiently executed. As the path is a string, you will need to pick a delimiter character -- we use ',' below. For example:

```
> t = db.tree
test.tree

> // get entire tree -- we use sort() to make the order nice
> t.find().sort({path:1})
{ "_id" : "a", "path" : "a," }
{ "_id" : "b", "path" : "a,b," }
{ "_id" : "c", "path" : "a,b,c," }
{ "_id" : "d", "path" : "a,b,d," }
{ "_id" : "g", "path" : "a,b,g," }
{ "_id" : "e", "path" : "a,e," }
{ "_id" : "f", "path" : "a,e,f," }
{ "_id" : "g", "path" : "a,b,g," }

> t.ensureIndex( {path:1} )

> // find the node 'b' and all its descendents:
> t.find( { path : /^a,b,/ } )
{ "_id" : "b", "path" : "a,b," }
{ "_id" : "c", "path" : "a,b,c," }
{ "_id" : "d", "path" : "a,b,d," }
{ "_id" : "g", "path" : "a,b,g," }

> // find the node 'b' and its descendents, where path to 'b' is not already known:
> nodeb = t.findOne( { _id : "b" } )
{ "_id" : "b", "path" : "a,b," }
> t.find( { path : new RegExp("^" + nodeb.path) } )
{ "_id" : "b", "path" : "a,b," }
{ "_id" : "c", "path" : "a,b,c," }
{ "_id" : "d", "path" : "a,b,d," }
{ "_id" : "g", "path" : "a,b,g," }
```

Ruby example: http://github.com/banker/newsmonger/blob/master/app/models/comment.rb

acts_as_nested_set

See http://ar.rubyonrails.org/classes/ActiveRecord/Acts/NestedSet/ClassMethods.html

This pattern is best for datasets that rarely change as modifications can require changes to many documents.

Use Cases

Find Nodes by a Partial Path

Suppose we want to find a given node in a tree, given some path through a portion of the tree to that node, and then get back that node, and perhaps also everything below it.

With a materialized paths approach we can do the above. The main thing that needs tweaking is to make the operation fast if there is a path "a..b..c..d..e" to a document and we want to find documents with a path "..b..c..d..". If we are starting from the very top it is easy (and described above in the materialized paths section). However here we aren't starting at the top. One approach is to use a combination of materialized path plus an array of the node's ancestors, something like:

```
{ path : ",a,b,c,d,e,",
  ancestor : ['a','b','c','d','e'] }
```

We could index on ancestors which will create a multikey index. Then we would do a query like the following to find nodes on path "...b,c,d..." with some efficiency:

```
find({ path : /,b,c,d,/, ancestor : 'd', <more_query_expressions_optionally> })
```

In the above the index on ancestor would be used and only docs from 'd' down need be inspected. The following could be tried which might be even better depending on how smart the query optimizer is:

```
find( { path : /,b,c,d,/, ancestor : { $all : ['a','d'] }, ... } )
```

See Also

- Sean Cribbs blog post (source of several ideas on this page).

Tweaking performance by document bundling during schema design

Note: this page discusses performance tuning aspects – if you are just getting started skip this for later. If you have a giant collection of small documents that will require significant tuning, read on.

During schema design one consideration is when to embed entities in a larger document versus storing them as separate small documents. Tiny documents work fine and should be used when that is the natural way to go with the schema. However, in some circumstances, it can be better to group data into larger documents to improve performance.

Consider for example a collection which contains some documents that are fairly small. Documents are indicated in the figures below as squares. Related documents – perhaps all associated with some larger entity in our program, or else that correlate in their access, are indicated in figure 1 with the same color.

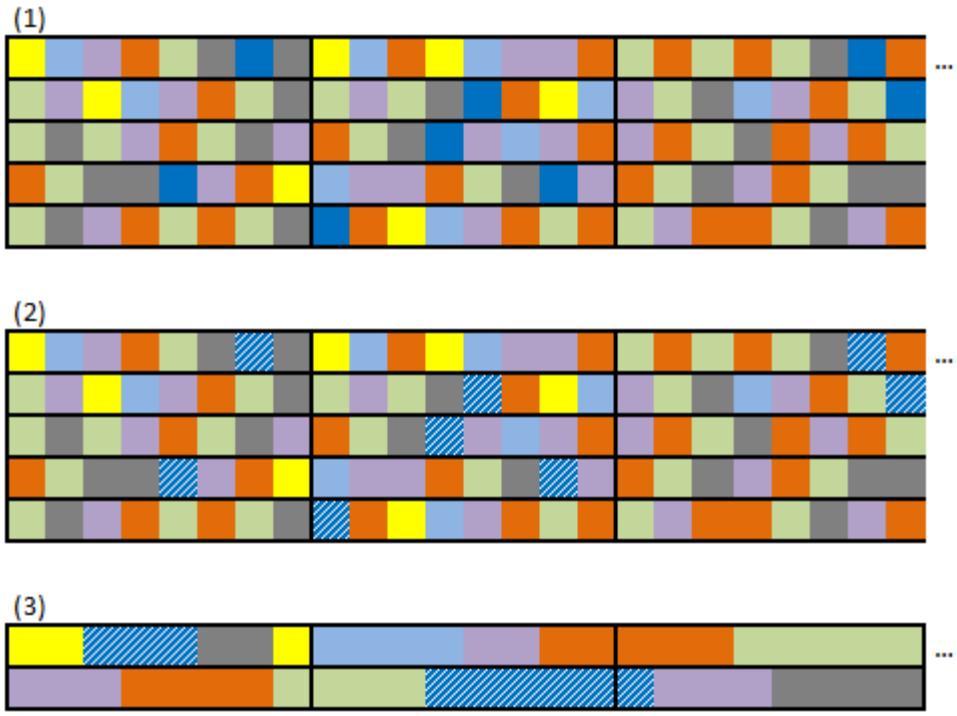
MongoDB caches data in pages, where the page size is that of the operating system's virtual memory manager (almost always 4KB). Page units are indicated by the black lines – for this example 8 boxes fit per page.

Let's suppose we wish to fetch all of the dark blue documents – indicates with stripes in figure 2. If this data is in RAM, we can (assuming we have an index) fetch them very efficiently. However note that the eight entities span eight pages, even though they could in theory fit on a single page.

With an alternate schema design we could "roll up" some of these entities into a larger document which includes an array of subdocuments. By doing that the items will be clustered together – a single BSON document in MongoDB is always stored contiguously. Figure 3 shows an example where the eight entities roll up into two documents (perhaps they could have rolled up to just one document; the point here is that it isn't essential that it be one, we are simply doing some bundling). In this example the two new documents are stored within three pages. While this isn't a huge reduction – eight to three – in many situations the documents are much smaller than a page – sometimes 100 documents fit within a single page. (The diagram example is not very granular to make reading easy.)

The benefits of this rolled-up schema design are

- Better RAM cache utilization. If we need to cache the dark blue items (but not the others), we can now cache three pages instead of eight. Note this is really only important if the data is too large to fit entirely in RAM – if it all fits, there is no gain here.
- Fewer disk seeks. If nothing was cached in RAM, less random i/o's are necessary to fetch the objects.
- Smaller index sizse. The common key the eight items contain can be stored in less copies, with less associated key entries in its corresponding index.



Caveats:

- Do not optimize prematurely; if grouping the entities would be awkward, don't do it. The goal of Mongo is to make development easier, not harder.
- Note we simply want to get to a document size that is on the order of the page cache unit size – about 4KB. If your documents are already of roughly that size, there is less benefit to the above (some still regarding random disk i/o, but no benefit for ram cache efficiency).
- If you often only need a subset of the items you would group, this approach could be inefficient compared to alternatives.

Using a Large Number of Collections

A technique one can use with MongoDB in certain situations is to have several collections to store information instead of a single collection. By doing this, certain repeating data no longer needs to be stored in every object, and an index on that key may be eliminated. More importantly for performance (depending on the problem), the data is then clustered by the grouping specified.

For example, suppose we are logging objects/documents to the database, and want to have M logs: perhaps a dev log, a debug log, an ops log, etc. We could store them all in one collection 'logs' containing objects like:

```
{ log : 'dev', ts : ..., info : ...  }
```

However, if the number of logs is not too high, it might be better to have a collection per log. We could have a 'logs.dev' collection, a 'logs.debug' collection, 'logs.ops', etc.:

```
// logs.dev:
{ ts : ..., info : ... }
```

Of course, this only makes sense if we do not need to query for items from multiple logs at the same time.

Generally, having a large number of collections has no significant performance penalty, and results in very good performance.

Limits

By default MongoDB has a limit of approximately 24,000 *namespaces* per database. Each namespace is 628 bytes, the `.ns` file is 16MB by default.

Each collection counts as a namespace, as does each index. Thus if every collection had one index, we can create up to 12,000 collections. The

`--nssize` parameter allows you to increase this limit (see below).

Be aware that there is a certain minimum overhead per collection -- a few KB. Further, any index will require at least 8KB of data space as the b-tree page size is 8KB. Certain operations can get slow if there are a lot of collections and the meta data gets paged out.

`--nssize`

If more collections are required, run mongod with the `--nssize` parameter specified. This will make the <database>.ns file larger and support more collections. Note that `--nssize` sets the size used for newly created `.ns` files -- if you have an existing database and wish to resize, after running the db with `--nssize`, run the `db.repairDatabase()` command from the shell to adjust the size.

Maximum `.ns` file size is 2GB.

Padding Factor

- Overview
- No padding after imports, repairs, compactions and initial replica syncs
- Shrinking Documents
- usePowerOf2Sizes
- Manual Padding
- See Also

Overview

When you update a document in MongoDB, the update occurs in-place if the document has not grown in size. If the document did grow in size, however, then it might need to be relocated on disk to find a new disk location with enough contiguous space to fit the new larger document. This can lead to problems for write performance if the collection has many indexes since a move will require updating all the indexes for the document.

Mongo adaptively learns if documents in a collection tend to grow, and if they do, it adds some *padding* so that the document has room to grow. This helps to prevent excessive movements on subsequent writes. This statistic is tracked separately for each collection.

You can check the collection's current *padding factor* by running the `db.<collection-name>.stats()` command helper in the shell. The padding factor indicates what the padding will be for new record allocations (for inserts or updates that grow a document, they cause a move operation to a new location).

```
> db.coll.stats()
{
        "ns" : "...", ...,
        "paddingFactor" : 1, ...,
        "ok" : 1
}
```

As each document is written at a different point in time the padding for each document will not be the same. Also, as the padding factor is relative to the size of each document you cannot calculate the exact amount of padding for a collection based on the average document size and padding factor.

The paddingfactor is 1.0 if there is no padding. 1.5 would indicate 50% padding on a new insert/moves.

No padding after imports, repairs, compactions and initial replica syncs

After compaction, repair and import operations, there is (generally) no padding as they were inserted and there were no updates (which would cause the paddingFactor to change). Thus you may see slower update performance after these cases, but the size required for storage will be lower.

Shrinking Documents

If a document gets smaller (e.g., because of an $unset or $pop), the document does not move but stays at its current location. It thus effectively has more padding. Thus space is never reclaimed if documents shrink by large amounts and never grow again. To reclaim that space run a compact operation (or repair).

usePowerOf2Sizes

v2.2+

Setting this attribute for a collection will round up allocation sizes to powers of two, resulting in much less potential for fragmentation (some extra disk space will be used for the collection's documents; the collection's index sizes will be unchanged). You can set this attribute with the collMod command.

```
> db.runCommand( { collMod : collectionname , usePowerOf2Sizes : true } )
```

Manual Padding

Padding in MongoDB is automatic. You should not have to do so manually. In exceptional cases one can do this though. The strategy is to add a faux field that assures allocation of a larger slot size for the document. The faux field is then removed; at this point there is extra room for expansion on a future update. Example below.

It is usually easier to use the v2.2 usePowerOf2Sizes attribute instead of manually padding.

```
> function checkOk() { assert( !db.runCommand({getlasterror:1,j:true,w:2,wtimeout:5000}).err ); }
>
> t = db.zcollection;
> db.setProfilingLevel(2);
>
> // let variable b be a very long string
>
> t.insert({q:1})
> t.update({q:1},{$set:{yy:b}})
> checkOk()
>
> // note the "moved:true" in the output below -- indicating not enough
> // padding to avoid moving a document.
> db.system.profile.find()
...
{ "ts" : ISODate("2011-10-13T02:45:23.062Z"), "op" : "insert", "ns" : "test.zcollection",
  "millis" : 0, "client" : "127.0.0.1", "user" : "" }
{ "ts" : ISODate("2011-10-13T02:45:33.560Z"), "op" : "update", "ns" : "test.zcollection",
  "query" : { "q" : 1 }, "updateobj" : { "$set" : { "yy" : "aaaaaaaaa..." } },
  "nscanned" : 1, "moved" : true, "millis" : 3, "client" : "127.0.0.1", "user" : "" }
>
> // not important what value of 'padding' is, only its length:
> t.insert({q:2,padding:ourpaddingstring})
> t.update({q:2},{$unset:{padding:1}})
> t.update({q:2},{$set:{yy:b}})
> checkOk()
>
> // no "moved:true" below (which is good).  Note however that the automatic adjustment
> // of a collection's padding factor normally achieves this regardless.  Manually padding
> // is rarely necessary.
> db.system.profile.find()
...
{ "ts" : ISODate("2011-10-13T02:46:34.920Z"), "op" : "insert", "ns" : "test.zcollection",
  "millis" : 1, "client" : "127.0.0.1", "user" : "" }
{ "ts" : ISODate("2011-10-13T02:46:42.775Z"), "op" : "update", "ns" : "test.zcollection",
  "query" : { "q" : 2 }, "updateobj" : { "$unset" : { "padding" : 1 } }, "nscanned" : 1,
  "millis" : 2, "client" : "127.0.0.1", "user" : "" }
{ "ts" : ISODate("2011-10-13T02:46:55.831Z"), "op" : "update", "ns" : "test.zcollection",
  "query" : { "q" : 2 }, "updateobj" : { "$set" : { "yy" : "aaaaaaaaa..." } },
  "nscanned" : 1, "millis" : 2, "client" : "127.0.0.1", "user" : "" }
```

See Also

- http://blog.mongodb.org/post/248614779/fast-updates-with-mongodb-update-in-place

Server-side Code Execution

- Map/Reduce
- Using `db.eval()`
  - Examples
  - Limitations of `eval`
    - Write locks
    - Sharding
- Storing functions server-side

- `$where` Clauses and Functions in Queries
- Notes on Concurrency
- Running .js files via a mongo shell instance on the server

Mongo supports the execution of code inside the database process.

Map/Reduce

MongoDB supports Javascript-based map/reduce operations on the server. See the map/reduce documentation for more information.

Using `db.eval()`

> **Locking**
> By default db.eval() requires a write lock so it will block all read/write operations while it runs.

`db.eval()` is used to evaluate a function (written in JavaScript) at the database server.
This is useful if you need to touch a lot of data lightly. In that scenario, network transfer of the data could be a bottleneck.

`db.eval()` returns the return value of the function that was invoked at the server. If invocation fails an exception is thrown.

For a trivial example, we can get the server to add 3 to 3:

```
> db.eval( function() { return 3+3; } );
6
>
```

Let's consider an example where we wish to erase a given field, `foo`, in every single document in a collection. A naive client-side approach would be something like

```
function my_erase() {
  db.things.find().forEach( function(obj) {
                      delete obj.foo;
                      db.things.save(obj);
                    } );
}

my_erase();
```

Calling `my_erase()` on the client will require the entire contents of the collection to be transmitted from server to client and back again.

Instead, we can pass the function to `eval()`, and it will be called in the runtime environment of the server. On the server, the `db` variable is set to the current database:

```
db.eval(my_erase);
```

Examples

```
> myfunc = function(x){ return x; };

> db.eval( myfunc, {k:"asdf"} );
{ k : "asdf"  }

> db.eval( myfunc, "asdf" );
"asdf"

> db.eval( function(x){ return x; }, 2 );
2.0
```

If an error occurs on the evaluation (say, a null pointer exception at the server), an exception will be thrown of the form:

```
{ dbEvalException: { errno : -3.0 , errmsg : "invoke failed" , ok : 0.0  } }
```

Example of using `eval()` to do equivalent of the Mongo `count()` function:

```
function mycount(collection) {
  return db.eval( function(){return db[collection].find({},{_id:ObjId()}).length();} );
}
```

Example of using `db.eval()` for doing an atomic increment, plus some calculations:

```
function inc( name , howMuch ){
    return db.eval(
        function(name, howMuch){
            var t = db.things.findOne( { name : name } );
            t = t || { name : name , num : 0 , total : 0 , avg : 0 };
            t.num++;
            t.total += howMuch;
            t.avg = t.total / t.num;
            db.things.save( t );
            return t;
        },
        name,     //first argument to function above
howMuch   //second argument to function above
);
}

db.things.remove( {} );
print( tojson( inc( "eliot" , 2 )) );
print( tojson( inc( "eliot" , 3 )) );
```

Limitations of `eval`

Write locks

It's important to be aware that by default `eval` takes a write lock. This means that you can't use `eval` to run other commands that themselves take a write lock. To take an example, suppose you're running a replica set and want to add a new member. You may be tempted to do something like this from a driver:

```
db.eval("rs.add('ip-address:27017')");
```

As we just mentioned, `eval` will take a write lock on the current node. Therefore, this won't work because you can't add a new replica set member if any of the existing nodes is write-locked.

The proper approach is to run the commands to add a node manually. `rs.add` simply queries the `local.system.replSet` collection, updates the config object, and run the `replSetReconfig` command. You can do this from the driver, which, in addition to not taking out the `eval` write lock, manages to more directly perform the operation.

In 1.7.2, a `nolock` option was added to `eval`. To use `nolock` you have to use the command interface directly:

```
db.runCommand({$eval: function() {return 42;}, nolock: true})
```

or with args

```
db.runCommand({$eval: function(x,y) {return x*y;}, args: [6,7], nolock: true})
```

Sharding

Note also that `eval` doesn't work with sharding. If you expect your system to later be sharded, it's probably best to avoid `eval` altogether.

Storing functions server-side

Note: we recommend *not* using server-side stored functions when possible. As these are code it is likely best to store them with the rest of your code in a version control system.

There is a special system collection called `system.js` that can store JavaScript functions to be reused. To store a function, you would do:

```
db.system.js.save( { _id : "foo" , value : function( x , y ){ return x + y; } } );
```

_id is the name of the function, and is unique per database.

Once you do that, you can use `foo` from any JavaScript context (db.eval, $where, map/reduce)

Here is an example from the shell:

```
>db.system.js.save({ "_id" : "echo", "value" : function(x){return x;} })
>db.eval("echo('test')")
test
```

See http://github.com/mongodb/mongo/tree/master/jstests/storefunc.js for a full example.

In MongoDB 2.1 you will also be able to load all the scripts saved in `db.system.js` into the shell using `db.loadServerScripts()`

```
>db.loadServerScripts()
>echo(3)
3
```

`$where` Clauses and Functions in Queries

In addition to the regular document-style query specification for `find()` operations, you can also express the query either as a string containing a SQL-style WHERE predicate clause, or a full JavaScript function. **Note: if a normal data-driven BSON query expression is possible, use that construction. Use $where only when you must it is significantly slower.**

When using this mode of query, the database will call your function, or evaluate your predicate clause, for each object in the collection.

In the case of the string, you must represent the object as "this" (see example below). In the case of a full JavaScript function, you use the normal JavaScript function syntax.

The following four statements in mongo - The Interactive Shell are equivalent:

```
db.myCollection.find( { a : { $gt: 3 } } );
db.myCollection.find( { $where: "this.a > 3" });
db.myCollection.find( "this.a > 3" );
db.myCollection.find( { $where: function() { return this.a > 3;}});
```

The first statement is the preferred form. It will be faster to execute because the query optimizer can easily interpret that query and choose an index to use.

You may mix data-style find conditions and a function. This can be advantageous for performance because the data-style expression will be evaluated first, and if not matched, no further evaluation is required. Additionally, the database can then consider using an index for that condition's field. To mix forms, pass your evaluation function as the `$where` field of the query object. For example, both of the following would work:

```
db.myCollection.find( { active: true, $where: function() { return obj.credits - obj.debits < 0; } } );
db.myCollection.find( { active: true, $where: "this.credits - this.debits < 0" } );
```

Do not write to the database from a `$where` expression.

Notes on Concurrency

If you don't use the "nolock" option, db.eval() requires a write lock which will block all reads and writes until it is done. Thus, its operations are atomic but prevent other operations from processing. Even with the nolock option javascript will still be limited to a single executing block at a time – multiple javascript threads cannot run concurrently.

When more concurrency is needed consider using an external client/program.

Running .js files via a mongo shell instance on the server

This is a good technique for performing batch administrative work. Run `mongo` on the server, connecting via the `localhost` interface. The connection is then very fast and low latency. This is friendlier than db.eval() as db.eval() blocks other operations.

Explain

- Basics
- Output Fields
  - cursor
  - nscanned
  - nscannedObjects
  - nYields
  - n
  - millis
  - scanAndOrder
  - indexOnly
  - isMultiKey
- With Sharding
  - clusteredType
  - shards
  - nChunkSkips
  - millisTotal
  - millisAvg
  - numQueries
  - numShards
- See Also

Basics

A great way to get more information on the performance of your database queries is to use the `$explain` feature. This will display "explain plan" information about a query from the database.

When using the mongo shell, invoke the `explain()` method on a cursor. The result will be a document that contains the explain output. *Note that explain runs the actual query to determine the result.* If the query is slow, the explain will be slow too.

Some of the options below may not appear depending on the version of mongodb you are running.

```
> db.collection.find(query).explain()
```

provides information such as the following:

```
{
    "cursor" : "BasicCursor",
    "indexBounds" : [ ],
    "nscanned" : 57594,
    "nscannedObjects" : 57594,
    "nYields" : 2 ,
    "n" : 3 ,
    "millis" : 108,
    "indexOnly" : false,
    "isMultiKey" : false,
    "nChunkSkips" : 0
}
```

Output Fields

cursor

The type of cursor used for scanning.

- BasicCursor - indicates a table scan style operation
- BtreeCursor - an index was used. When an index is used, indexBounds will be set to indicate the key bounds for scanning in the index.

nscanned

Number of items (documents or index entries) examined. Items might be objects or index keys. If a "covered index" is involved, nscanned may be

higher than nscannedObjects.

nscannedObjects

Number of documents scanned.

nYields

Number of times this query yielded the read lock to let waiting writes execute.

n

Number of documents matched (on all criteria specified).

millis

Time for the query to run, in milliseconds.

scanAndOrder

True if the index could not be used for sorting.

indexOnly

True if the results of the query can be returned using only the index. See SERVER-5759 for more information on the behavior of this value.

isMultiKey

If true, a multikey index was used.

With Sharding

In a sharded deployment, `explain` outputs some additional fields. An example:

```
"clusteredType" : "ParallelSort",
 "shards" : {
  "shard1:27017" : [
   {
    "cursor" : "BtreeCursor a_1",
    "nscanned" : 1,
    "nscannedObjects" : 1,
    "n" : 1,
    "millis" : 0,
    "nYields" : 0,
    "nChunkSkips" : 0,
    "isMultiKey" : false,
    "indexOnly" : false,
    "indexBounds" : {
     "a" : [
      [
       1,
       1
      ]
     ]
    }
   }
  ]
 },
 "n" : 1,
 "nChunkSkips" : 0,
 "nYields" : 0,
 "nscanned" : 1,
 "nscannedObjects" : 1,
 "millisTotal" : 0,
 "millisAvg" : 0,
 "numQueries" : 1,
 "numShards" : 1
```

clusteredType

- `ParallelSort` means all shards are accessed in parallel
- `SerialServer` means shards are queried one by one in order

shards

List of all the shards accessed during the query and the `explain` output for each shard.

nChunkSkips

The number of documents skipped because of active chunk migrations in a sharded system. Typically this will be zero. A number greater than zero is ok, but indicates a little bit of inefficiency.

millisTotal

Total time for the query to run, in milliseconds.

millisAvg

Average time for the query to run on each shard, in milliseconds.

numQueries

Total number of queries executed.

numShards

Total number of shards queried.

See Also

- Optimization

- Hint
- Database Profiler
- Viewing and Terminating Current Operation

Full Text Search in Mongo

- Introduction
- Multikeys (Indexing Values in an Array)
- Text Search
- Comparison to Full Text Search Engines
- Real World Examples

Introduction

Mongo provides some functionality that is useful for text search and tagging.

Multikeys (Indexing Values in an Array)

The Mongo multikey feature can automatically index arrays of values. Tagging is a good example of where this feature is useful. Suppose you have an article object/document which is tagged with some category names:

```
obj = {
  name: "Apollo",
  text: "Some text about Apollo moon landings",
  tags: [ "moon", "apollo", "spaceflight" ]
}
```

and that this object is stored in `db.articles`. The command

```
db.articles.ensureIndex( { tags: 1 } );
```

will index all the tags on the document, and create index entries for "moon", "apollo" and "spaceflight" for that document.

You may then query on these items in the usual way:

```
> print(db.articles.findOne( { tags: "apollo" } ).name);
Apollo
```

The database creates an index entry for each item in the array. Note an array with many elements (hundreds or thousands) can make inserts very expensive. (Although for the example above, alternate implementations are equally expensive.)

Text Search

It is fairly easy to implement basic full text search using multikeys. What we recommend is having a field that has all of the keywords in it, something like:

```
{ title : "this is fun" ,
  _keywords : [ "this" , "is" , "fun" ]
}
```

Your code must split the title above into the keywords before saving. Note that this code (which is not part of Mongo DB) could do stemming, etc. too. (Perhaps someone in the community would like to write a standard module that does this...)

Comparison to Full Text Search Engines

MongoDB has interesting functionality that makes certain search functions easy. That said, it is not a dedicated full text search engine.

For example, dedicated engines provide the following capabilities:

- built-in text stemming
- ranking of queries matching various numbers of terms (can be done with MongoDB, but requires user supplied code to do so)
- bulk index building

Bulk index building makes building indexes fast, but has the downside of not being realtime. MongoDB is particularly well suited for problems

where the search should be done in realtime. Traditional tools are often not good for this use case.

Real World Examples

The Business Insider web site uses MongoDB for its blog search function in production.

Mark Watson's opinions on Java, Ruby, Lisp, AI, and the Semantic Web - A recipe example in Ruby.

Full text search with MongoDB at Flowdock

Tailable Cursors

Tailable cursors are only allowed on capped collections and can only return objects in natural order. Tailable queries never use indexes.

A tailable cursor "tails" the end of a capped collection, much like the Unix "tail -f" command. They key idea is that if we "catch up" and have reached the end of the collection, our position is remembered rather than the cursor being closed. Thus, after new objects are inserted, we can resume retrieving from where we left off – which is then very inexpensive. If you reach the end of the collection and request more data, the cursor will block waiting for new documents to be inserted.

If the field you wish to "tail" is indexed, do not use tailable cursors; instead simply (re)query for { *field* : { $gt : *value* } } where `value` is where you last left off. This is normally quite efficient. Tailable cursors are for cases where having an index would be prohibitive (extremely high write collections). If performance is not problematic, use a normal query and cursor, tailable adds some complexity.

As no index will be used for the query, the initial scanning to find the first object to return will likely be quite costly. However once found, retrieving additional data from new inserts then becomes very inexpensive.

The cursor may become invalid if, for example, the last object returned is at the end of the collection and is deleted. Thus, you should be prepared to requery if the cursor is "dead". You can determine if a cursor is dead by checking its id. An id of zero indicates a dead cursor (use isDead in the c++ driver). In addition, the cursor will be in "dead" state after a query which returns no matches.

MongoDB replication uses tailable cursors to follow the end of the primary's replication op log collection. Writes to the oplog would be slower with an index. The tailable feature eliminates the need to create an index for replication's use case.

C++ example:

```
#include "client/dbclient.h"

using namespace mongo;

/* "tail" the namespace, outputting elements as they are added. Cursor blocks
 * waiting for data if no documents currently exist. For this to work something
 * field -- _id in this case -- should be increasing when items are added.
 */
void tail(DBClientBase& conn, const char *ns) {
  // minKey is smaller than any other possible value
BSONElement lastId = minKey.firstElement();

  Query query = Query().sort("$natural"); // { $natural : 1 } means in forward
// capped collection insertion order
while( 1 ) {
    auto_ptr<DBClientCursor> c =
      conn.query(ns, query, 0, 0, 0,
                 QueryOption_CursorTailable | QueryOption_AwaitData );
    while( 1 ) {
      if( !c->more() ) {
        if( c->isDead() ) {
          // we need to requery
break;
        }
        // No need to wait here, cursor will block for several sec with _AwaitData
 continue; // we will try more() again
}
      BSONObj o = c->next();
      lastId = o["_id"];
      cout << o.toString() << endl;
    }

    // prepare to requery from where we left off
query = QUERY( "_id" << GT << lastId ).sort("$natural");
  }
}
```

Javascript example:

```
var coll = db.some.capped.collection;
var lastVal = coll.find().sort({ '$natural' : 1 })
                          .limit( 1 ).next()[ 'increasing' ];
while(1){

  cursor = coll.find({ 'increasing' : { '$gte' : lastVal } });

  // tailable
cursor.addOption( 2 );
  // await data
cursor.addOption( 32 );

  // Waits several sec for more data
while( cursor.hasNext() ){
    var doc = cursor.next();
    lastVal = doc[ 'increasing' ];
    printjson( doc );
  }

}
```

See Also

- A detailed blog post on tailable cursors
- https://github.com/mongodb/mongo-snippets/blob/master/cpp/tailable_cursor.cpp
- http://api.mongodb.org/cplusplus/2.0.3/dbclient_8h_source.html

mongo - The Interactive Shell

- MongoDB Manual Shell Reference
- More Information
- Presentations

The MongoDB distribution includes `bin/mongo`, the MongoDB interactive shell. This utility is a JavaScript shell that allows you to issue commands to MongoDB from the command line. (*it is basically an extended SpiderMonkey shell*)

The shell is useful for:

- inspecting a database's contents
- testing queries
- creating indices
- maintenance scripts
- other administrative functions

When you see sample code in this wiki and it looks like JavaScript, assume it is a shell example. See the driver syntax table for a chart that can be used to convert those examples to any language.

MongoDB Manual Shell Reference

- MongoDB Manual Reference Section
- mongo Manual Page
- JavaScript Interface
- MongoDB Database Commands

More Information

- Shell Overview
- Shell Scripts
- Shell Reference
- Shell APIDocs

Presentations

- Hacking the Shell - MongoSF (May 2011)
- CRUD and the JavaScript Shell - MongoSF (April 2010)

Scripting the shell

The MongoDB shell is not just an interactive shell, it can also be scripted using JS files. In addition to specifying a Javascript file (*.js) you can also use `--eval` with a snippet of JS.

Using the shell this way allows for tasks to be performed without the need for any additional drivers or language support; it can be used in cron, or automated administrative tasks. Please be aware there are data format issues in javascript so you should be careful how much you do in Javascript.

Common uses for the scripted shell includes:

- backups
- scheduled Map-Reduce commands
- offline reports
- administration

Running a Script

```
./mongo server:27017/dbname --quiet my_commands.js
```

The syntax stems from the interactive shell. This command will execute the `my_commands.js` as if it had been entered into the shell directly, with some exceptions.

- `./mongo`: command to start the interactive shell, may vary on your shell of choice
- `server:27017/dbname`: basic connection information
- `--quiet`: this is a flag for the `mongo` command. This switch removes some header information that is not typically necessary when building unattended scripts.
- `my_commands.js`: a file containing a series of shell commands to execute

`--quiet`

This option will remove the header printed at the top when you run the shell normally:

```
$ mongo
MongoDB shell version: 2.1.0
connecting to: test
> ^C
bye
$ mongo --quiet
> ^C
```

`--eval`

In addition to using a full Javascript file you can also pass in a Javascript fragment:

```
bash-3.2$ ./mongo test --eval "printjson(db.getCollectionNames())"
MongoDB shell version: 1.8.0
connecting to: test
[
        "system.indexes",
        "t1",
        "t2",
        "test.fam",
        "test1",
        "test11",
        "testBinary",
        "testarray"
]
```

Differences between scripted and interactive

Printing

When using the shell interactively, the shell will print returned values and format where possible. This is done as a general convenience from within the shell. However, when building a script, the printing needs to be defined explicitly.

There are two functions commonly used for this:

1. `print()`: works as normal javascript
2. `printjson()`: prints a nicely formatted JSON representation of the given object

***Example**: print JSON for the first 10 objects from a `find`*

```
db.foo.find({x:1}).forEach(printjson)
```

`use dbname`

This command does not work in scripted mode. Instead you will need to explicitly define the database in the connection (`/dbname` in the example above).

Alternately, you can also create a connection within the script:

```
> help connect // for more help
> var x = new Mongo('host[:port]');
> var mydb = x.getDB('mydb');
> // or
> var mydb = connect('host[:port]/mydb');
```

Or just reassign the "db" variable.

```
> db = db.getSiblingDB("otherdb") //same as use otherdb
```

`it`

The iterator command `it` does not work outside of the interactive scripting environment.

getLastError

When running an `update/insert` command from the shell, the shell automatically awaits a reply (*i.e. runs a get last error*).

The same is not true when running from a script file. To wait for the status of an operation (such as a write), run the getLastError function after `update/insert`.

```
db.getLastErrorObj()
// or
db.getLastError()
```

Overview - The MongoDB Interactive Shell

- Running the Shell
  - .mongorc.js
- Operations
  - Help
  - Select Database
  - Querying
  - Inserting
  - Updating
  - Deleting
  - Indexes
  - Open Additional Connections
- Working from the Prompt
  - Line Continuation
  - Key Shortcuts
  - Custom Prompt
  - Using a real editor
- Some Notes on Datatypes in the Shell
  - Numbers
  - Dates
  - BinData
- See Also

Running the Shell

The interactive shell is included in the standard MongoDB distribution. To start the shell, go into the root directory of the distribution and type

```
./bin/mongo
```

It might be useful to add *`mongo_distribution_root`*`/bin` to your `PATH` so you can just type `mongo` from anywhere.

If you start with no parameters, it connects to a database named "test" running on your local machine on the default port (27017). You can see the db to which you are connecting by typing `db`:

```
./mongo
type "help" for help
> db
test
```

You can pass `mongo` an optional argument specifying the address, port and even the database to initially connect to:

`./mongo` *`foo`*	connects to the *`foo`* database on your local machine
`./mongo 192.168.13.7/`*`foo`*	connects to the *`foo`* database on 192.168.13.7

`./mongo dbserver.mydomain.com/foo`	connects to the *foo* database on dbserver.mydomain.com
`./mongo 192.168.13.7:9999/foo`	connects to the *foo* database on 192.168.13.7 on port 9999

.mongorc.js

> **1.9.1+**

When the shell is launched, it checks the user's home directory for a javascript file named `.mongorc.js`. If this file is found, its contents are interpreted and run by the shell prior to displaying the prompt for the first time. This allows the user to define variables, customize the prompt, or update information that they would like updated every time they launch a shell. This functionality can be overridden with the `--norc` flag. It should be noted that if a file is specified to be executed by the shell, the rc file will not be run until after that file has completed.

Operations

Help

```
> help                          // top level help
> db.help()                     // help on db-specific methods
> db.mycollection.help()        // help on collection methods
> db.mycollection.find().help() // cursor help
```

Select Database

The following are three basic commands that provide information about the available databases, and collections in a given database.

`show dbs`	displays all the databases on the server you are connected to
`use db_name`	switches to *db_name* on the same server
`show collections`	displays a list of all the collections in the current database

Querying

`mongo` uses a JavaScript API to interact with the database. Because `mongo` is also a complete JavaScript shell, `db` is the variable that is the current database connection.

To query a collection, you simply specify the collection name as a property of the `db` object, and then call the `find()` method. For example:

```
db.foo.find();
```

This will display the first 10 objects from the *foo* collection. Typing ***it*** after a find() will display the next 10 subsequent objects.

> By setting the shellBatchSize you can change this:
>
> ```
> DBQuery.shellBatchSize = #
> ```

> If the shell does not accept the collection name (for example if it starts with a number, contains a space etc), use
>
> ```
> db['foo'].find()
> ```
>
> instead.

Inserting

In order to insert data into the database, you can simply create a JavaScript object, and call the `save()` method. For example, to save an object {

name: "sara"} in a collection called *foo*, type:

```
db.foo.save({ name : "sara"});
```

Note that MongoDB will implicitly create any collection that doesn't already exist.

Updating

Let's say you want to change someone's address. You can do this using the following `mongo` commands:

```
person = db.people.findOne( { name : "sara" } );
person.city = "New York";
db.people.save( person );
```

Deleting

db.*foo*.drop()	drop the entire *foo* collection
db.*foo*.remove()	remove all objects from the collection
db.*foo*.remove({ name : "sara" })	remove objects from the collection where name is sara

Indexes

db.*foo*.getIndexKeys()	get all fields that have indexes on them
db.*foo*.ensureIndex({ _field_ : 1 })	create an index on *field* if it doesn't exist

Open Additional Connections

You can use the following commands to open additional connections (normally you don't need to do this, but might from a script):

```
conn = new Mongo(host);
db = conn.getDB(dbname);
db.auth(username,password);
```

where `host` is a string that contains either the name or address of the machine you want to connect to (e.g. "192.168.13.7") or the machine and port (e.g. "192.168.13.7:9999"). Note that host in an optional argument, and can be omitted if you want to connect to the database instance running on your local machine. (e.g. `conn = new Mongo()`)

Alternatively you can use the `connect` helper method:

```
> db = connect("localhost:27020/mytestdb"); // example with a nonstandard port #
```

Working from the Prompt

Line Continuation

If a line contains open '(' or '{' characters, the shell will request more input before evaluating:

```
> function f() {
... x = 1;
... }
>
```

You can press Enter twice to escape from "..." mode and terminate line entry.

Key Shortcuts

- up/down array for command history
- in v1.9+ some basic emacs keystrokes work

- ctrl-l to clear the screen
- tab for auto-complete (newer versions only)
- ctrl-c to exit
- Enter twice to break out of line continuation mode

Custom Prompt

1.9.1+

The shell's prompt can be customized by creating variable 'prompt' in the shell. It can be any arbitrary javascript, including a function that returns a string. This flexibility allows for additional information to be displayed in the prompt. For example, to have a prompt that contains the number of commands issued, type:

```
> cmdCount = 1;
> prompt = function() {
... return (cmdCount++) + "> ";
... }
1> command
2> anothercommand
3>
```

To make the prompt look a bit more familiar, we can make it database@host$:

```
> host = db.serverStatus().host; \\ since host should not change
> prompt = function() {
... return db+"@"+host+"$ ";
... }
admin@mylaptop.local$ use monkeys
switched to db monkeys
monkeys@mylaptop.local$
```

You could use the prompt to do a bit of database monitoring as well:

```
> prompt = function() {
... return "Uptime:"+db.serverStatus().uptime+" Files:"+db.stats().objects+" > ";
... }
Uptime:5897 Files:6 > db.monkeys.save({name : "James"});
Uptime:5948 Files:7 >
```

Using a real editor

2.1.0+

We've added a feature to allow you edit larger values including functions using your editor of choice. Just run `edit nameOfVariableOrFunction` and we will open whatever editor you have defined in your `$EDITOR` environment variable. Make sure that you save the file when editing. If you wish to discard your changes, you can either not save or make your editor exit with an error (`:cq` in Vim or `(kill-emacs 1)` in Emacs).

```
$ EDITOR=vim mongo --nodb
MongoDB shell version: 2.1.0-pre-
> function f() {}
> edit f
> f
function f() {
    print("this really works");
}
> f()
this really works
> o = {}
{ }
> edit o
> o
{ "soDoes" : "this" }
>
```

It is possible that the code in functions will be slightly modified by the JavaScript compiler when you try to edit it again. For example it may convert 1+1 in to 2 and strip out comments. The actual changes will vary based on the version of JavaScript used, but should not effect the semantics of the code, only its appearance.

Some Notes on Datatypes in the Shell

Numbers

By default, the shell treats all numbers as floating-point values. You have the option to work with 64-bit integers by using a class built into the shell called NumberLong() If you have long/integer BSON data from the database you may see something like this:

```
{"count" : NumberLong("575175")}
```

Setting/incrementing any number from javascript will (most likely) change the data type to a floating point value.

Here is an example of creating a document with a `long` field:

```
doc = { field: new NumberLong("123212313")}
```

Note that prior to 1.6 long numbers might be displayed like this:

```
"bytes" : {
   "floatApprox" : 5284376243087482000,
   "top" : 1230364721,
   "bottom" : 4240317554
}
```

Dates

The Date() function returns a string and a "**new** Date()" will return an object (which is what you should use to store values).

```
> Date()
Sun May 02 2010 19:07:40 GMT-0700 (Pacific Daylight Time)
> new Date()
"Sun May 02 2010 19:07:43 GMT-0700 (Pacific Daylight Time)"
> typeof(new Date())
object
> typeof(Date())
string
```

newer (1.7+) versions print this

```
> new Date()
ISODate("2010-11-29T19:41:46.730Z")
> ISODate("2010-11-29T19:41:46.730Z")
ISODate("2010-11-29T19:41:46.730Z")
```

As you can see, ISODate is a thin wrapper around the Date constructor to fix some of it's shortcomings. It returns a normal Date object with all of the normal methods that javascript Date methods support. We have also changed the way that Date objects print to make sure that they don't look like strings and that if you copy and paste the output you get the same object.

BinData

The BSON BinData datatype is represented via class BinData in the shell. Run `help misc` for more information.

```
> new BinData(2, "1234")
BinData(2,"1234")
```

See Also

- MongoDB Shell Reference

dbshell (mongo) Reference

- History
- Command Line
- Special Command Helpers
- Basic Shell Javascript Operations
- Queries
- Error Checking
- Administrative Command Helpers
- Opening Additional Connections
- Miscellaneous
- Examples

The dbshell is called mongo (or mongo.exe on windows). It is an interactive command line utility for administering and querying a mongo database.

History

You can scroll through previous commands issued to mongo with the up and down arrow keys. Command history is stored in ~/.dbshell.

Auth Commands
Authentication commands will not be saved in your history.

Windows Users
Before version 2.2.0 there was a bug and the file was not stored correctly in your home directory. It was stored in the working directory.

Command Line

`--help`	Show command line options
`--nodb`	Start without a db, you can connect later with `new Mongo()` or `connect()`
`--shell`	After running a .js file from the command line, stay in the shell rather than terminating

Special Command Helpers

Non-javascript convenience macros:

`help`	Show help

`db.help()`	Show help on db methods
`db.myColl.help()`	Show help on collection methods
`show dbs`	Print a list of all databases on this server
`use dbname`	Set the db variable to represent usage of *dbname* on the server
`show collections`	Print a list of all collections for current database
`show users`	Print a list of users for current database
`show profile`	Print most recent profiling operations that took >= 1ms

Basic Shell Javascript Operations

`db`	The variable that references the current database object / connection. Already defined for you in your instance.
`db.auth(user,pass)`	Authenticate with the database (if running in secure mode).
`coll = db.collection`	Access a specific *collection* within the database.
`cursor = coll.find();`	Find all objects in the collection. See queries.
`coll.remove(objpattern);`	Remove matching objects from the collection. *objpattern* is an object specifying fields to match. E.g.: `coll.remove( { name: "Joe" } );`
`coll.save(object)`	Save an object in the collection, or update if already there. If your object has a presave method, that method will be called before the object is saved to the db (before both updates and inserts)
`coll.insert(object)`	Insert object in collection. No check is made (i.e., no upsert) that the object is not already present in the collection.
`coll.update(...)`	Update an object in a collection. See the Updating documentation; update() has many options.
`coll.ensureIndex( { name : 1 } )`	Creates an index on `tab.name`. Does nothing if index already exists.
`coll.update(...)`	
`coll.drop()`	Drops the collection `coll`
`db.getSiblingDB(name)`	Return a reference to another database using this same connection. This allows for cross database queries. Another synonym is "getSisterDB" . Usage example: `db.getSiblingDB('production').getCollectionNames()`

Queries

`coll.find()`	Find all.
`it`	Continue iterating the last cursor returned from find().
`coll.find( criteria );`	Find objects matching *criteria* in the collection. E.g.: `coll.find( { name: "Joe" } );`
`coll.findOne( criteria );`	Find and return a single object. Returns null if not found. If you want only one object returned, this is more efficient than just `find()` as `limit(1)` is implied. You may use regular expressions if the element type is a string, number, or date: `coll.find( { name: /joe/i } );`
`coll.find( criteria, fields );`	Get just specific fields from the object. E.g.: `coll.find( {}, {name:true} );`
`coll.find().sort( {field:1[, field :1] });`	Return results in the specified order (field ASC). Use -1 for DESC.
`coll.find( criteria ).sort( { field : 1 } )`	Return the objects matching *criteria*, sorted by *field*.

`coll.find( ... ).limit(n)`	Limit result to *n* rows. Highly recommended if you need only a certain number of rows for best performance.
`coll.find( ... ).skip(n )`	Skip *n* results.
`coll.count()`	Returns total number of objects in the collection.
`coll.find( ... ).count()`	Returns the total number of objects that match the query. Note that the number ignores limit and skip; for example if 100 records match but the limit is 10, `count()` will return 100. This will be faster than iterating yourself, but still take time.

More information: see queries.

Error Checking

`db.getLastError()`	Returns error from the last operation.
`db.getPrevError()`	Returns error from previous operations.
`db.resetError()`	Clear error memory.

Administrative Command Helpers

`db.cloneDatabase(fromhost)`	Clone the current database from the other host specified. fromhost database must be in noauth mode.
`db.copyDatabase(fromdb, todb, fromhost)`	Copy fromhost/fromdb to todb on this server. fromhost must be in noauth mode.
`db.fromColl .renameCollection(toColl)`	Rename collection from fromColl to toColl.
`db.repairDatabase()`	Repair and compact the current database. This operation can be very slow on large databases.
`db.addUser(user,pwd)`	Add user to current database.
`db.getCollectionNames()`	get list of all collections.
`db.dropDatabase()`	Drops the current database.

Opening Additional Connections

`db = connect("<host>:<port>/<dbname>")`	Open a new database connection. One may have multiple connections within a single shell, however, automatic getLastError reporting by the shell is done for the 'db' variable only.
`conn = new Mongo("hostname")`	Open a connection to a new server. Use `getDB()` to select a database thereafter.
`db = conn.getDB("dbname")`	Select a specific database for a connection

Miscellaneous

`Object.bsonsize(db.foo.findOne())`	prints the bson size of a db object (mongo version 1.3 and greater)
`db.foo.findOne().bsonsize()`	prints the bson size of a db object (mongo versions predating 1.3)

For a full list of functions, see the shell API.

Examples

The MongoDB source code includes a jstests/ directory with many mongo shell scripts.

Admin Zone

- Journaling

- Production Notes
- Sharding
- Monitoring and Diagnostics
- Starting and Stopping Mongo
- Data Center Awareness
- DBA Operations from the Shell
- Troubleshooting

See The MongoDB Manual for complete and current documentation of MongoDB.

Community Admin-Related Articles

- boxedice.com - notes from a production deployment
- Survey of Admin UIs for MongoDB
- MongoDB Nagios Check
- MongoDB Cacti Graphs

See Also

- Commands in Developer Zone

Journaling

- Disabling/Enabling
- Journal Files
- Recovery
- The journal Subdirectory
- Group Commits
- Commit Acknowledgement
- FAQ
  - If I am using replication, can some members use journaling and others not?
  - How's performance?
  - Can I use the journaling feature to perform safe hot backups?
  - 32 bit nuances?
  - When did the --journal option change from --dur?
  - Will the journal replay have problems if entries are incomplete (like the failure happened in the middle of one)?
  - How many times is data written to disk when replication and journaling are both on?
- See Also

v1.8+

MongoDB supports write-ahead journaling of operations to facilitate fast crash recovery and durability in the storage engine.

Disabling/Enabling

In v2.0+, journaling is enabled by default for 64-bit platforms. You can disable journaling with the mongod `--nojournal` command line option. For versions < 2.0 or 32-bit platforms, you can enable journaling with the `--journal` command line option.

It is OK to disable journaling after running with journaling by simply shutting down `mongod` cleanly and restarting with `--nojournal`. The reverse is also OK; shutdown cleanly and restart without `--nojournal`.

MongoDB may determine that it is faster to preallocate journal files than to create them as needed. If MongoDB decides to preallocate the files, it will not start listening on port 27017 until this process completes, which can take a few minutes. This means that your applications and the shell will not be able to connect to the database immediately on initial startup. Check the logs to see if MongoDB is busy preallocating. It will print the standard "waiting for connections on port *whatever*" when it has finished.

Journal Files

With journaling enabled, journal files will be created in a `journal/` subdirectory under your chosen db path. These files are write-ahead redo logs. In addition, a last sequence number file, `journal/lsn`, will be created. A clean shutdown removes all files under `journal/`.

The MongoDB data files (*database*.ns, *database*.0, *database*.1, ...) have the same format as in previous releases. Thus, the upgrade process is seamless, and a rollback would be seamless too. (If you roll back to a pre v1.7.5 release, try to shut down cleanly first. Regardless, remove the journal/ directory before starting the pre v1.7.5 version of mongod.)

Recovery

On a restart after a crash, journal files in `journal/` will be replayed before the server goes online. This will be indicated in the log output. You do not need to run a repair.

The journal Subdirectory

You may wish, before starting `mongod` to symlink the `journal/` directory to a dedicated hard drive to speed the frequent (fsynced) sequential writes which occur to the current journal file.

Group Commits

MongoDB performs *group commits* (batch commits) when journaling. This means that a series of operations over many milliseconds are committed all at once. This is done to achieve high performance.

Group commits are performed approximately every 100ms by default. In version 1.9.2+, you can set this interval yourself using the `--journalCommitInterval` command line option. The allowed range is 2 to 300 milliseconds.

Commit Acknowledgement

You can wait for group commit acknowledgement with the getLastError Command. In versions before 1.9.0 using getLastError + fsync would do this, in newer versions the "j" option has been specifically created for this purpose.

In version 1.9.2+ the group commit delay is shortened when a commit acknowledgement (getLastError + j) is pending; this can be as little as 1/3 of the normal group commit interval.

FAQ

If I am using replication, can some members use journaling and others not?

Yes.

How's performance?

Read performance should be the same. Write performance should be very good but there is some overhead over the non-durable version as the journal files must be written. If you find a case where there is a large difference in performance between running with and without journaling, please let us know so we can tune it.

Can I use the journaling feature to perform safe hot backups?

Yes, see Backups with Journaling Enabled.

32 bit nuances?

There is extra memory mapped file activity with journaling. This will further constrain the limited db size of 32 bit builds. Thus, for now journaling by default is disabled on 32 bit systems.

When did the --journal option change from --dur?

In 1.8 the option was renamed to `--journal`, but the old name is still accepted for backwards compatibility; please change to `--journal` if you are using the old option.

Will the journal replay have problems if entries are incomplete (like the failure happened in the middle of one)?

Each journal (group) write is consistent and won't be replayed during recovery unless it is complete.

How many times is data written to disk when replication and journaling are both on?

In v1.8, for an insert, four times. The object is written to the main collection, and also the oplog collection (so that is twice). Both of those writes are journaled as a single mini-transaction in the journal file (the files in /data/db/journal). Thus 4 times total.

There is an open item in to reduce this by having the journal be compressed. This will reduce from 4x to probably ~2.5x.

The above applies to collection data and inserts which is the worst case scenario. Index updates are written to the index and the journal, but not the oplog, so they should be 2X today not 4X. Likewise updates with things like $set, $addToSet, $inc, etc. are compactly logged all around so those are generally small.

See Also

- Journaling Administration Notes

- getLastError command for controlling writes per operation
- Durability Internals
- Presentations
    - Recorded Webinar on Journaling (April 2011)
    - Journaling and the Storage Engine (May 2011)

Journaling Administration Notes

- Journal Files (*e.g.* journal/j._0)
- Prealloc Files (*e.g.* journal/prealloc.0)
- serverStatus command
- journalLatencyTest Command

Journal Files (e.g. journal/j._0)

Journal files are append-only and are written to the `journal/` directory under the dbpath directory (which is `/data/db/` by default).

Journal files are named `j._0`, `j._1`, etc. When a journal file reached 1GB in size, a new file is created. Old files which are no longer needed are rotated out (automatically deleted). Unless your write bytes/second rate is extremely high, you should have only two or three journal files.

Note: in more recent versions, the journal files are only 128MB apiece when using the `--smallfiles` command line option.

Prealloc Files (e.g. journal/prealloc.0)

`mongod` will create `prealloc` files in the journal directory under some circumstances to minimize journal write latency. On some filesystems, appending to a file and making it larger can be slower than writing to a file of a predefined size. `mongod` checks this at startup and if it finds this to be the case will use preallocated journal files. If found to be helpful, a small pool of prealloc files will be created in the journal directory before startup begins. This is a one time initiation and does not occur with future invocations. Approximately 3GB of files will be preallocated (and truly prewritten, not sparse allocated) - thus in this situation, expect roughly a 3 minute delay on the first startup to preallocate these files.

If you don't want to wait three minutes on startup, you can preallocate the files using another instance of `mongod` and then move them to your normal dbpath before starting with journaling. For example, if you had an instance of `mongod` running on port 27017 with a dbpath of */data/db* (the defaults), you could preallocate journal files for it with:

```
$ mkdir ~/tmpDbpath
$ mongod --port 10000 --dbpath ~/tmpDbpath --journal
# startup messages
# .
# .
# .
# wait for prealloc to finish
Thu Mar 17 10:02:52 [initandlisten] preallocating a journal file ~/tmpDbpath/journal/prealloc.0
Thu Mar 17 10:03:03 [initandlisten] preallocating a journal file ~/tmpDbpath/journal/prealloc.1
Thu Mar 17 10:03:14 [initandlisten] preallocating a journal file ~/tmpDbpath/journal/prealloc.2
Thu Mar 17 10:03:25 [initandlisten] flushing directory ~/tmpDbpath/journal
Thu Mar 17 10:03:25 [initandlisten] flushing directory ~/tmpDbpath/journal
Thu Mar 17 10:03:25 [initandlisten] waiting for connections on port 10000
Thu Mar 17 10:03:25 [websvr] web admin interface listening on port 11000
# then Ctrl-C to kill this instance
^C
$ mv ~/tmpDbpath/journal /data/db/
$ # restart mongod on port 27017 with --journal
```

prealloc files do not contain data, but are rather simply preallocated files that are ready to use that are truly preallocated by the file system (i.e. they are not "sparse"). It is thus safe to remove them, but if you restart mongod with journaling, it will create them again if they are missing.

serverStatus command

The serverStatus command now includes some statistics regarding journaling.

journalLatencyTest Command

You can use the `journalLatencyTest` command to measure how long it takes on your volume to write to the disk (including fsyncing the data) in an append-only fashion.

```
> use admin
> db.runCommand("journalLatencyTest")
```

You can run this command on an idle system to get a baseline sync time for journaling. In addition, it is safe to run this command on a busy system to see the sync time on a busy system (which may be higher if the journal directory is on the same volume as the data files).

In version 1.9.2+ you can set the group commit interval, using `--journalCommitInterval` command-line option, to between 2 and 300 milliseconds (default is 100ms). The actual interval will be the maximum of this setting and your disk latency as measured above.

journalLatencyTest is also a good way to check if your disk drive is buffering writes in its local cache. If the number is very low (e.g., less than 2ms) and the drive is non-ssd, the drive is probably buffering writes. In that case, you will want to enable cache write-through for the device in your operating system. (Unless you have a disk controller card with battery backed ram, then this is a good thing.)

Production Notes

- Backups
- TCP Port Numbers
- Linux
    - General Unix Notes
    - Linux Kernel Versions
    - Checking Disk IO
    - Checking Network IO
- Storage
    - Solid State Disks (SSDs)
    - RAID
    - Linux File Systems
    - Readahead
    - Remote mounts
    - Swap
- What Hardware?
- Tips

- 32 bit
- iostat
- NFS
- NUMA
- SSD
- Virtualization

Below are some MongoDB production devops recommendations and best practices.

Backups

- Backups Doc Page
- Import Export Tools

TCP Port Numbers

Default TCP port numbers for MongoDB processes:

- Standalone `mongod` : 27017
- `mongos` : 27017
- shard server (`mongod --shardsvr`) : 27018
- config server (`mongod --configsvr`) : 27019
- web stats page for `mongod` : add 1000 to port number (28017, by default)

Firewall rules for these ports and connections are available here.

Linux

General Unix Notes

- Turn off *atime* for the data volume
- Set file descriptor limit and user process limit to 4k+ (see *etc/limits* and *ulimit*)
- **Do not** use large/huge virtual memory pages (the default is small pages which is what you want)
- Use `dmesg` to see if box is behaving strangely

- Try to disable NUMA in your BIOS. If that is not possible see NUMA
- Ensure that readahead settings for the block devices that store the dbpath are acceptable Readahead
- Minimize clock skew between your hosts by using ntp; linux distros usually include this by default, but check and install the ntpd package if it isn't already installed

Linux Kernel Versions

Some have reported skepticism on behavior of Linux 2.6.33-31 and 2.6.32 kernel, at least on EC2. 2.6.36 is given a thumbs up by the community.

For those running databases on ext4 filesystems, a 2.6.23 kernel is required for efficient filesystem preallocation, 2.6.25 is required for XFS support of the same feature. High filesystem I/O following the allocation of new database files is one symptom of this problem.

Checking Disk IO

`iostat -txm 2`

If the timestamp feature is not available, `iostat -xm 2`

Checking Network IO

- atop - Holistic system top
- Munin
- ethtool eth0 - check network port speed
- bwm-ng
- iptraf

Storage

Solid State Disks (SSDs)

- See SSD page

RAID

Typically we recommend using RAID-10.

RAID-5 and RAID-6 can be slow, and are not suggested.

RAID-0 will provide good write performance but provides limited availability, and reduced performance on reads (particularly on EBS). It is not suggested.

See also the ec2 page for comments on EBS striping.

Linux File Systems

MongoDB uses large files for storing data, and preallocates these. These filesystems seem to work well:

- ext4 (kernel version >= 2.6.23)
- xfs (kernel version >= 2.6.25)

In addition to the file systems above you might also want to (explicitly) disable file/directory modification times by using these mount options:

- noatime (also enables nodiratime)

We have found **`ext3`** to be very slow in allocating files (or removing them) as well as access within large files is also poor.

Readahead

Check your volume's readahead on Linux with the following command:

```
# report
blockdev --report
# set
blockdev --setra <sectors> /dev/<devicename>
```

Note the output for the readahead (RA) column is in 512 byte sectors. For databases, a small value such as 32 (16KB) often works well. Note if the setting is small you may want to preheat some data into the filesystem cache on a server reboot.

Remote mounts

We have found that some versions of NFS perform very poorly and do not recommend using NFS. See the NFS page for more information.

Amazon elastic block store (EBS) seems to work well up to its intrinsic performance characteristics, when configured well.

Swap

It is useful for the linux kernel to have swap space to use in emergencies. Because of the way MongoDB memory maps the database files none of this data will ever end up in swap; this means that on a healthy system the swap space will rarely be used on a system only running MongoDB. Having swap can keep the kernel from killing MongoDB when physical memory limits are reached.

You may also want to look at using something which compresses swap/memory like compcache.

What Hardware?

MongoDB tends to run well on virtually all hardware. In fact it was designed specifically with commodity hardware in mind (to facilitate cloud computing); that said it works well on very large servers too. That said if you are about to buy hardware here are a few suggestions:

- Fast CPU clock speed is helpful.
- Many cores helps but does not provide a high level of marginal return, so don't spend money on them. (This is both a consequence of the design of the program and also that memory bandwidth can be a limiter; there isn't necessarily a lot of computation happening inside a database).
- NUMA is not very helpful as memory access is not very localized in a database. Thus non-NUMA is recommended; or configure NUMA as detailed elsewhere in this document.
- RAM is good.
- SSD is good. We have had good results and have seen good price/performance with SATA SSDs; the (typically) more upscale PCI SSDs work fine too.
- Commodity (SATA) spinning drives are often a good option as the speed increase for random I/O for more expensive drives is not that dramatic (only on the order of 2x) – spending that money on SSDs or RAM may be more effective.

Tips

- Handling Halted Replication
- Starting and Stopping the Database

32 bit

See also http://blog.mongodb.org/post/137788967/32-bit-limitations

The MongoDB storage engine uses memory mapped files, which limits database size on 32 bit builds of Mongo.

Use 64 bit for production. This is important as if you hit the mmap size limit (exact limit varies but less than 2GB) you will be unable to write to the database (analogous to a disk full condition).

Some notes on the 32 bit builds:

- `mongod` is the process of interest here (the one which uses memory mapped files); that is, the `mongo` shell and the `mongos` shard controller do not have this limitation.
- Replica set arbiters have virtually no data, so 32 bit `mongod` would work for an arbiter.
- If you are running a 64 bit operating system on your development desktop, use the 64 bit build though as there is no reason to make things more different than need be. That said 32 bit is generally fine for development if required.
- 32 bit `mongod` defaults to journaling off. This is because a second (private) mmap is used when journaling which lowers the amount of data which can be mapped, for example from 2GB to 1GB. If you are for some reason using 32 bit and your database is very small be sure to explicitly specify `--journal`.
- If using 32 bit `mongod` for some reason you may with to explicitly specify the --oplogSize and/or --smallFiles command line parameters.
- If in a special situation where you are using 32 bit and have small data, declaring some of your collections as capped would be a way to help ensure your database never gets too large.

iostat

On Linux, use the iostat command to check if disk I/O is a bottleneck for your database.

We generally find the form:

```
iostat -xm 2
```

to work well. (Use a number of seconds with iostat, otherwise it will display stats since server boot, which is not very useful.)

Use the `mount` command to see what device your `/data/db` directory resides on.

Fields

- %util - this is the most useful field for a quick check, it indicates what percent of the time the device/drive is in use. If the number if near 100%, your server may be physical disk I/O bound. (There are some volume situations where this statistic overstates, but most often it is correct.)
- r/s - reads per second.
- w/s - writes per second
- rMB/s - read megabytes per second
- wMB/s - write megabytes per second
- avgrq-sz - average request size. The smaller this number, the more random your IO operations are. This is in sectors : typically sectors are 512 bytes, so multiply by 0.5 to see average request size in kilobytes.

On Windows Server use the performance monitor utility.

NFS

We have found that some versions of NFS perform very poorly, or simply don't work, and do not suggest using NFS. (We'd love to hear from you if you are using NFS and what results you are getting, either great or not great.)

NUMA

- numactl
- proc settings
- Testing
- References

Linux, NUMA and MongoDB tend not to work well together. If you are running MongoDB on numa hardware, we recommend turning it off (running with an interleave memory policy). Problems will manifest in strange ways, such as massive slow downs for periods of time or high system cpu time.

numactl

To turn off NUMA, start mongod with

```
numactl --interleave=all ${MONGODB_HOME}/bin/mongod --config conf/mongodb.conf
```

proc settings

```
echo 0 > /proc/sys/vm/zone_reclaim_mode
```

You can change `zone_reclaim_mode` without restarting `mongod`. For more information on this setting see http://www.kernel.org/doc/Documentation/sysctl/vm.txt.

Testing

On Linux, `mongod` v2.0+ checks these settings on startup and prints a warning if they do not match the recommendations.

References

The MySQL “swap insanity” problem and the effects of the NUMA architecture describes the effects of NUMA on databases. This blog post was aimed at problems NUMA created for MySQL, but the issues are the same. The posting describes the NUMA architecture and goals, and how these are incompatible with the working of databases.

SSD

We are not experts on solid state drives, but tried to provide some information here that would be helpful. Comments very welcome.

- Write Endurance
  - Reserve some unpartitioned space
  - smartctl
- Speed
- Reliability
- Random reads vs. random writes
- PCI vs. SATA
- RAM vs. SSD
- FlashCache
- OS scheduler
- Run mongoperf
- Helpful links

Multiple MongoDB users have reported good success running MongoDB databases on solid state drives.

Write Endurance

Write endurance with solid state drives vary. SLC drives have higher endurance but newer generation MLC (and eMLC) drives are getting better.

As an example, the MLC Intel 320 drives specify endurance of 20GB/day of writes for five years. If you are doing small or medium size random reads and writes this is sufficient. The Intel 710 series is the enterprise-class models and have higher endurance.

If you intend to write a full drive's worth of data writing per day (and every day for a long time), this level of endurance would be insufficient. For large sequential operations (for example very large map/reduces), one could write far more than 20GB/day. Traditional hard drives are quite good at sequential I/O and thus may be better for that use case.

- Blog post on SSD lifespan

Reserve some unpartitioned space

Some users report good results when leaving 20% of their drives completely unpartitioned. In this situation the drive knows it can use that space as working space. Note formatted but empty space may or may not be available to the drive depending on TRIM support which is often lacking.

smartctl

On some devices, "smartctl -A" will show you the Media_Wearout_Indicator.

```
$ sudo smartctl -A /dev/sda | grep Wearout
233 Media_Wearout_Indicator 0x0032   099   099   000    Old_age   Always       -       0
```

Speed

A paper in ACM Transactions on Storage (Sep2010) listed the following results for measured 4KB peak random direct IO for some popular devices:

Device	Read IOPS	Write IOPS
Intel X25-E	33,400	3,120
FusionIO ioDrive	98,800	75,100

Intel's larger drives seem to have higher write IOPS than the smaller ones (up to 23,000 claimed for the 320 series). More info here.

Real-world results should be lower, but the numbers are still impressive.

Reliability

Some manufacturers specify relability stats indicating failure rates of approximately 0.6% per year. This is better than traditional drives (2% per year failure rate or higher), but still quite high and thus mirroring will be important. (And of course manufacture specs could be optimistic.)

Random reads vs. random writes

Random access I/O is the sweet spot for SSD. Historically random reads on SSD drives have been much faster than random writes. That said, random writes are still an order of magnitude faster than spinning disks.

Recently new drives have released that have much higher random write performance. For example the Intel 320 series, particular the larger capacity drives, has much higher random write performance than the older Intel X25 series drives.

PCI vs. SATA

SSD is available both as PCI cards and SATA drives. PCI is oriented towards the high end of products on the market.

Some SATA SSD drives now support 6Gbps sata transfer rates, yet at the time of this writing many controllers shipped with servers are 3Gbps. For random IO oriented applications this is likely sufficient, but worth considering regardless.

RAM vs. SSD

Even though SSDs are fast, RAM is still faster. Thus for the highest performance possible, having enough RAM to contain the working set of data from the database is optimal. However, it is common to have a request rate that is easily met by the speed of random IO's with SSDs, and SSD cost per byte is lower than RAM (and persistent too).

A system with less RAM and SSDs will likely outperform a system with more RAM and spinning disks. For example a system with SSD drives and 64GB RAM will often outperform a system with 128GB RAM and spinning disks. (Results will vary by use case of course.)

One helpful characteristic of SSDs is they can facilitate fast "preheat" of RAM on a hardware restart. On a restart a system's RAM file system cache must be repopulated. On a box with 64GB RAM or more, this can take a considerable amount of time – for example six minutes at 100MB/sec, and much longer when the requests are random IO to spinning disks.

FlashCache

FlashCache is a write back block cache for Linux. It was created by Facebook. Installation is a bit of work as you have to build and install a kernel module. Sep2011: If you use this please report results in the mongo forum as it's new and everyone will be curious how well it works.

- http://www.facebook.com/note.php?note_id=388112370932

OS scheduler

One user reports good results with the noop IO scheduler under certain configurations of their system. As always caution is recommended on nonstandard configurations as such configurations never get as much testing...

Run mongoperf

mongoperf is a disk performance stress utility. It is not part of the mongo database, simply a disk exercising program. We recommend testing your SSD setup with mongoperf. Note that the random writes it are a worst case scenario, and in many cases MongoDB can do writes that are much larger.

Helpful links

- Benchmarks of SSD drives from various manufacturers
  - http://techreport.com/articles.x/20653/5
  - http://www.anandtech.com/show/4244/intel-ssd-320-review/3
- Intel SSD Models Comparison (scroll down)
- Intel 710 and 720 series info

Virtualization

Generally MongoDB works very well in virtualized environments, with the exception of OpenVZ.

EC2

Compatible. No special configuration requirements.

VMWare

Some suggest not using overcommit as they may cause issues. Otherwise compatible.

Cloning a VM is possible. For example you might use this to spin up a new virtual host that will be added as a member of a replica set. If Journaling is enabled, the clone snapshot will be consistent. If not using journaling, stop mongod, clone, and then restart.

OpenVZ

Issues have been reported here.

Sharding

MongoDB scales horizontally via an auto-sharding (partitioning) architecture. MongoDB sharding provides:

- Automatic balancing for changes in load and data distribution
- Easy addition of new machines without down time
- Scaling to one thousand nodes
- No single points of failure
- Automatic failover

Getting Started

- Introduction. Philosophy, use cases, and its core components.
- Simple Initial Sharding Architecture
- Configuration. Setting up your cluster.
- Administration

Additional Info

- Failover How failover/HA works.
- Sharding Internals Implementation details.
- Restrictions and Limitations
- FAQ Frequently asked questions.
- HOWTO
    - Changing Config Servers

Presentations and Further Materials

- How Sharding Works - O'Reilly Webcast (February 2011)
- How queries work with sharding (PDF)
- Illustration of chunks and migration (PDF) http://www.10gen.com/video/mongosf2011/sharding
- Scaling MongoDB - O'Reilly ebook
- Schema design at scale (video)
- Mongo Sharding Architecture, Implementation, Internals (video)

See Also

- Replication

Video - How and When to Scale MongoDB with Sharding

Presentation at MongoSV (December 2011)

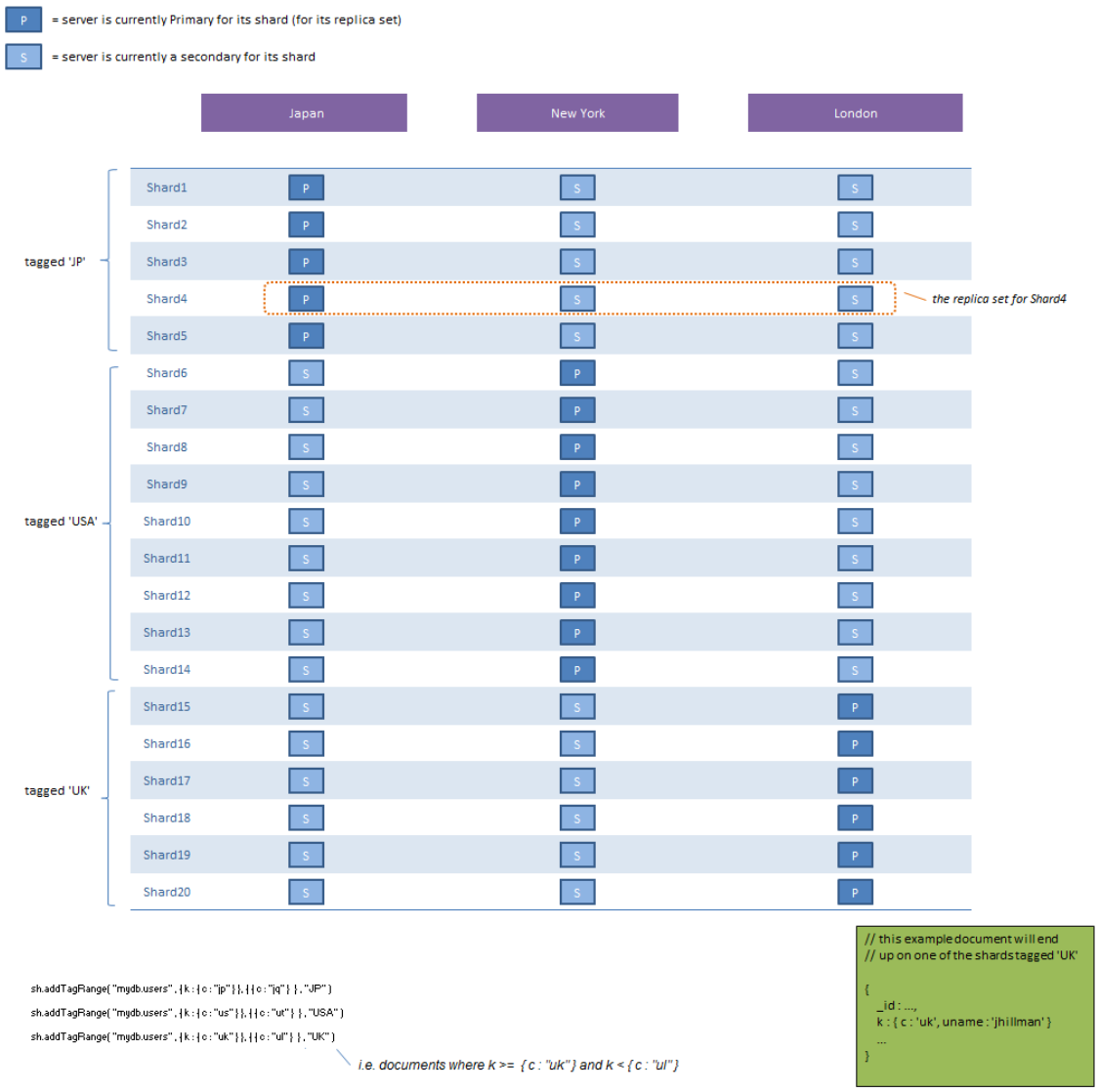
Tag Aware Sharding

In 2.2, MongoDB adds additional support for custom partitioning in sharded clusters – specifically sharded collections. By using this “tag aware” sharding and balancing, you can (automatically) ensure that data in a sharded database system is always on specific shards. This can be used to ensure data is geographically close to the systems which use it. You can also restrict sharded collections to only a few shards, therefore federating a set of shards for different uses.

Shard tagging controls data location, and is complementary but separate from replica set tagging – which can be used to enforce writing (and reading with Read Preferences within the replica set) within the replica set.

For example you can tag all “USA” data (by defining a range of your shard key which contains the "USA" data) to one or more shards (by tagging those shards with "USA").

Typically this is used to make assignment of documents to shards geographically aware. One marks a shard key range as "homed" on a certain set of shards. Those shards would then be configured such that their primary defaults to the desired data center / geography.



Monitoring and Diagnostics

- mongostat
- mongotop
- Query Profiler
- Http Console
- mongo Shell Diagnostic Commands
- Trending/Monitoring Adaptors
- Other Tools
- Hosted Monitoring
- Database Record/Replay (diagLogging command)
- Additional Resources
- Admin UIs

- Checking Server Memory Usage

- MMS (MongoDB Monitoring Service) a free hosted monitoring tool for MongoDB by 10gen

mongostat

mongostat is a great utility which exposes many internal MongoDB metrics. For any MongoDB related issues it is a good start for the analysis of performance issues.

mongotop

- http://docs.mongodb.org/manual/reference/mongotop/

Query Profiler

Use the Database Profiler to analyze slow queries.

db.currentOp() is another way to get a snapshot of what is currently happening.

Http Console

The mongod process includes a simple diagnostic screen at http://localhost:28017/. See the Http Interface docs for more information.

mongo Shell Diagnostic Commands

- db.serverStatus()
  - See the serverStatus Command page.
- db.stats()
  - Stats on the current database. Command takes some time to run, typically a few seconds unless the .ns file is very large (via use of --nssize). While running other operations may be blocked.
  - fileSize is the total size of all files allocated for the db.
- db.foo.find().explain()
  - explain plan
- help
  - db.help()
  - db.foo.help()

Trending/Monitoring Adaptors

- munin
  - Server stats: this will retrieve server stats (requires python; uses http interface)
  - Collection stats, this will display collection sizes, index sizes, and each (configured) collection count for one DB (requires python; uses driver to connect)
- Ganglia:
  - ganglia-gmond
  - mongodb-ganglia
- cacti
- Mikoomi provides a MongoDB plugin for Zabbix
- Nagios
- Motop - Realtime monitoring tool for several MongoDB servers. Shows current operations ordered by durations every second.
- mtop - A top like utility for Mongo
- Mongo Live - A Chrome extension that provides a real-time server status view (uses the rest interface).

Chris Lea from (mt) Media Temple has made an easy to install Ubuntu package for the munin plugin.

Other Tools

- Dex analyzes logs and/or system.profile collections for (slow) queries and suggests indexes which might help based on a few simple heuristics.

Hosted Monitoring

- MongoDB Monitoring Service (MMS) is a free hosted monitoring tool for MongoDB provided by 10gen
- Server Density provides hosted monitoring for your hardware and software infrastructure, and supports a number of status checks for MongoDB.
- Cloudkick
- scout app slow queries
- AppFirst

Database Record/Replay (diagLogging command)

Recording database operations, and replaying them later, is sometimes a good way to reproduce certain problems in a controlled environment.

To enable logging:

```
db._adminCommand( { diagLogging : 1 } )
```

To disable:

```
db._adminCommand( { diagLogging : 0 } )
```

Values for diagLogging:

- 0 off. Also flushes any pending data to the file.
- 1 log writes
- 2 log reads
- 3 log both
- 7 log write and some read operations

Note: if you log reads, it will record the findOnes above and if you replay them, that will have an effect!

Output is written to diaglog.*hex_time_t* in the /data/db/ directory (unless --dbpath is specified). You must shut down mongod in order to move, rename, or delete this file.

To replay the logged events:

```
nc ''database_server_ip'' 27017 < ''somelog.bin'' | hexdump -c
```

Additional Resources

- Monitoring MongoDB
- Server Status Reference
- Database Stats Reference
- Collection Stats Reference

Admin UIs

See the Admin UIs page

Checking Server Memory Usage

- How Caching Works
- Memory Mapped Files
- Windows
- Virtual Memory Size
- Swap
  - Always Have Swap
- Commands
- Working Set Size
  - Eatmem utility
  - Asymmetry
- Unix Utilities
  - See Also

How Caching Works

See Caching

Memory Mapped Files

Depending on the platform you may see the mapped files as memory in the process (see the Virtual Memory section below), but this is not strictly correct. If mapped files are counted as process memory, the `top` utility may show way more memory for `mongod` than is really appropriate for what is in physical memory.

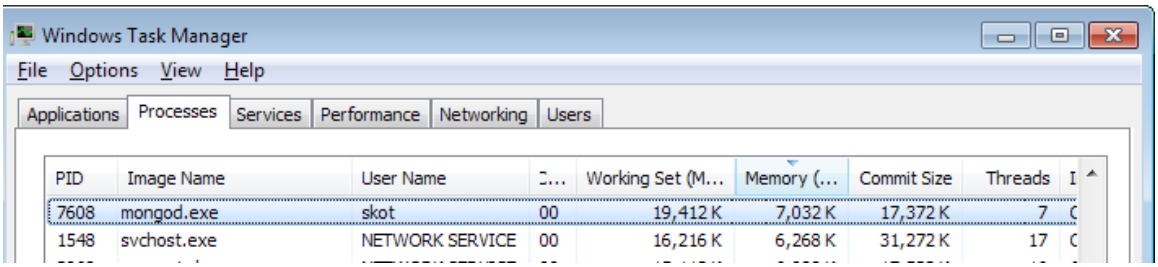
The operating system manages the memory where the memory mapped files reside (depending on the OS, this is usually done by the virtual memory manager). You can usually see mapped files using a program like `free -lmt`.

Memory mapped files are shown under "cached" memory:

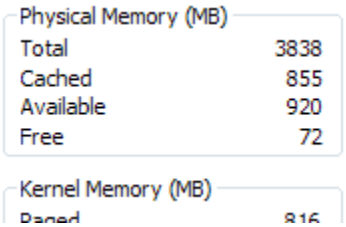
```
skot@stump:~$ free -tm
             total       used       free     shared    buffers     cached
Mem:          3962       3602        359          0        411       2652
-/+ buffers/cache:        538       3423
Swap:         1491         52       1439
Total:        5454       3655       1799
```

Windows

By bringing up the Task Manager you can see the process memory allocation for mongod.



In addition in the Performance tab the "cached" memory which represents the memory allocated by the memory mapped (data) files.



Virtual Memory Size

Virtual Memory includes the mapped memory files which can be much larger then the actual physical (or swap) available on your host. If journaling is enabled then the database files will be mapped twice leading to much higher virtual memory allocation than might be obvious.

It is important to remember that only a portion of the virtual memory number is really in physical memory. There are many utilities which will report memory usage as the virtual memory number which can be misleading.

Read below for more metrics about virtual memory allocation and how to detect related problems.

Swap

Since the database files are memory mapped and should constitute most of your mongodb memory usage it is very unlikely that mongod will ever use any swap space. The reason for this is because any of the memory mapped files can simply either be released from memory (without going to swap) to free memory for other programs or written back to the database files which means they never need to be "swapped" out to disk in the swap space (since they are already backed by files).

If you see continuous swap usage then it most likely means one of these things:

- Your system is extremely constrained by memory (see free -ltm "cached" to confirm)
- There is a memory leak (like the javascript engine, or lots of no_timeout cursors never being closed, or other internal data structures piling up)
- Some other program is stealing lots of memory

Always Have Swap

Because of these possible conditions it is always good to have some swap space available on your system. Think of the swap space as something like a steam release valve which allows excess pressure to release without blowing the system up.

Commands

The serverStatus() command provides memory usage information. Shell example:

```
> db.serverStatus()
> db.serverStatus().mem
> db.serverStatus().extra_info
```

One can verify there is no memory leak in the mongod process by comparing the mem.virtual and mem.mapped values (these values are in megabytes). If you are running with journaling disabled, the difference should be relatively small compared to total RAM on the machine. If you are running with journaling enabled, compare mem.virtual to 2*mem.mapped. Also watch the delta over time; if it is increasing consistently, that could indicate a leak.

The mem.mapped value reflects the size of all databases currently open. When replication is on, this includes the size of the local database which includes the oplog. When journaling is enabled each file is mapped twice, once as a writable memmapped view and once as a protected view. The total amount of RAM used is roughly the same; the larger virtual memory size should not be a cause for concern.

One large component of the difference between mem.virtual and mem.mapped (or 2*mem.mapped when journaling is enabled) can be stack memory. In particular, each connection that you have open has a stack frame. The size of each stack frame is determined by the stack size; in Linux this typically defaults to 8MB, which means that each connection will use 8MB on the server. If you are using many connections and are concerned with memory usage you should reduce the stack size to 1MB (this is automatic in the upcoming v2.0 release).

On Linux, extra_info will contain information about the total heap size in a heap bytes field.

You can also see the virtual size and mapped values in the mongostat utility's output.

While increasing virtual size can indicate a memory leak, increasing resident size (ie, what is reflected as RES in the output of top) indicates that the operating system is using a larger portion of available memory to hold mongodb data, which often occurs under normal operations as a system warms up.

Note: OS X includes the operating system image in virtual size (~2GB on a small program). Thus interpretation of the values on OS X is a bit harder.

Working Set Size

In MongoDB it is fine if databases (and thus virtual size) are much larger than ram (terabytes for example); however, you will want your *working set* to stay in memory to achieve good performance. Otherwise lots of random disk IO's will occur, and unless you are using SSD, this can be quite slow. One area to watch specifically in managing the size of your working set is index access patterns. If you are inserting into indexes at random locations (for example, with id's which are effectively randomly generated by hashes), you will continually be updating the whole index. If instead you are able to create your id's in approximately ascending order (for example, day concatenated with a random id), all the updates will occur at the right side of the b-tree and the working set size for index pages will be much smaller.

Eatmem utility

Measuring working set size can be difficult; even if it is much smaller than total RAM, if the db has been up for a while and the db is much larger than RAM in total, all memory will be indicated as in use for the cache. Thus we need some other way to estimate our working set size.

One technique is to use a utility which reserves a certain amount of system memory for itself. So one could run this with a certain amount specified and see if the server continues to perform well. If not, the working set is largely than (total_ram - eaten_ram). Note this test will eject some data from the file system cache which may take time to page back in after the `eatmem` utility is terminated.

Running eatmem continuously with a small percentage of total RAM (say, 20%) is a good technique to get an "early warning" of memory being too low. If disk i/o activity increases significantly, terminate eatmem to migigate the problem for the moment until further steps can be taking.

Asymmetry

In replica sets if one server is underpowered this could one again help as an early warning mechanism for server capacity. Of course the server must be receiving representative traffic to get an indication here.

Unix Utilities

`mongod` uses memory-mapped files; thus the memory stats in top require interpretation in a special way. On a large database, virtual bytes/VSIZE will tend to be the size of the entire database, and if the server doesn't have other processes running, resident bytes/RSIZE will be the total memory of the machine (as this counts file system cache contents).

`vmstat` can be useful – try running `vmstat 2`. on OS X, just `vm_stat`.

See Also

- The Linux Out of Memory OOM Killer

Starting and Stopping Mongo

- Starting mongod
  - Starting with Default Data Directory and TCP Port
  - Specifying a Data Directory
  - Specifying a TCP Port
  - Running as a Daemon
- Stopping mongod
  - Control-C
  - Sending shutdownServer() message from the mongo shell
  - Sending a Unix INT or TERM signal
- Memory Usage

MongoDB is run as a standard program from the command line. Please see Command Line Parameters for more information on those options. For information about running `mongod` or `mongos` as a service in Windows, see Windows Service.

The following examples assume that you are in the directory where the `mongod` executable resides. `mongod` is the primary database process that runs on an individual server. `mongos` is the sharding process. `mongo` is the administrative shell. This page discusses `mongod`.

Starting mongod

Starting with Default Data Directory and TCP Port

To start Mongo in default mode, where data will be stored in the `/data/db` directory (or `c:\data\db` on Windows), and listening on port 27017, just type

```
$ ./mongod
```

Specifying a Data Directory

To specify a directory for Mongo to store files, use the `--dbpath` option:

```
$ ./mongod --dbpath /var/lib/mongodb/
```

Note that you must create the directory and set its permissions appropriately ahead of time -- Mongo will not create the directory if it doesn't exist.

Specifying a TCP Port

You can specify a different port for Mongo to listen on for connections from clients using the `--port` option.

```
$ ./mongod --port 12345
```

Use this when running several mongod processes on a single machine. *It is highly recommended to use the default port number whenever possible to avoid any confusion.*

Note: if you use the --configsvr or --shardsvr command line options, those parameters will select an appropriate port number automatically (more info). Port 27017 is by convention the port to which client applications talk to the cluster. Thus if you are unsharded, mongod would be listening on 27017, but if sharded, the mongos process would be.

Running as a Daemon

Note: these options are only available in MongoDB version 1.1 and later.

This will fork the Mongo server and redirect its output to a logfile. As with `--dbpath`, you must create the log path yourself, Mongo will not create parent directories for you.

```
$ ./mongod --fork --logpath /var/log/mongodb.log --logappend
```

Stopping mongod

Control-C

If you have Mongo running in the foreground in a terminal, you can simply "Ctrl-C" the process. This will cause Mongo to do a clean exit, flushing and closing it's data files. Note that it will wait until all ongoing operations are complete.

Sending shutdownServer() message from the mongo shell

The shell can request that the server terminate.

```
$ ./mongo
> use admin
> db.shutdownServer()
```

This command only works from localhost or if one is authenticated.

From a driver (where the helper function may not exist), one can run the command

```
{ "shutdown" : 1 }
```

If this server is the primary in a replica set, it will go through the following process (version 1.9.1+):

1. Check how up-to-date the secondaries are.
2. If no secondary within 10 seconds of the primary, return that we won't shut down (optionally pass the `timeoutSecs` option to wait for a secondary to catch up.
3. If there is a secondary within 10 seconds of the primary, the primary will step down and wait for the secondary to catch up.
4. After 60 seconds or once the secondary has caught up, the primary will shut down.

If there is no up-to-date secondary and you want the primary to shut down anyway, you can use `force : true`:

```
> db.adminCommand({shutdown : 1, force : true})
> // or
> db.shutdownServer({force : true})
```

You can also specify `timeoutSecs : N`, which will keep checking the secondaries for N seconds if none are immediately up-to-date. If any of the secondaries catch up within N seconds, the primary will shut down. If no secondaries catch up, it will not shut down.

```
> db.adminCommand({shutdown : 1, timeoutSecs : 5})
> // or
> db.shutdownServer({timeoutSecs : 5})
> // sample output:
{
        "closest" : NumberLong(1307651781),
        "difference" : NumberLong(1307651808),
        "errmsg" : "no secondaries within 10 seconds of my optime",
        "ok" : 0
}
```

Sending a Unix INT or TERM signal

You can cleanly stop `mongod` using a SIGINT or SIGTERM signal on Unix-like systems. Either ^C, "`kill -2 PID`," or `kill -15 PID` will work.

Sending a KILL signal `kill -9` will probably cause damage if `mongod` is not running with Journaling. (Journaling is on by default for 64 bit mongod.) See also: repairDatabase command.)

After a hard crash, when not using `--journal`, MongoDB will say it was not shutdown cleanly, and ask you to do a repair of the database.

Memory Usage

Mongo uses memory mapped files to access data, which results in large numbers being displayed in tools like top for the mongod process. This is not a concern, and is normal when using memory-mapped files. Basically, the size of mapped data is shown in the virtual size parameter, and resident bytes shows how much data is being cached in RAM.

You can get a feel for the "inherent" memory footprint of Mongo by starting it fresh, with no connections, with an empty /data/db directory and looking at the resident bytes.

Logging

- Command Line Options
- Rotating the log files
  - From the mongo shell
  - From the unix shell
- Accessing Logs via Shell
  - Get a list of available loggers
  - Get a log

MongoDB outputs some important information to stdout while its running. There are a number of things you can do to control this

Command Line Options

- `--quiet` - less verbose output (more details)
- `-v` - more verbose output. use more v's (such as `-vvvvvv`) for higher levels of verbosity. To change the logging verbosity on a running instance, you can use the setParameter Command
- `--logpath <file>` - output to file instead of stdout
  - If you use `logpath`, you can rotate the logs by either running the `logRotate` command (1.3.4+) or sending SIGUSR1
  - You should always use `--logappend` with `--logpath` to append to the existing log file, instead of overwriting it

Rotating the log files

Log file rotation renames the current log file to the same name with a timestamp file extension appended, then continues logging to a new file with the original name. The timestamp is the time that the logRotate command was executed, expressed in UTC (GMT) and formatted as ISO but with dashes instead of colons for the time portion.

For example:

```
$ ./mongod -v --logpath /var/log/mongodb/server1.log --logappend
```

will start mongod with verbose logging to `/var/log/mongodb/server1.log`, appending to any existing log file.

In another terminal, list the matching files:

```
$ ls /var/log/mongodb/server1.log*
server1.log
```

Rotate the log file using one of the methods described below, then list the files again:

```
$ ls /var/log/mongodb/server1.log*
server1.log  server1.log.2011-11-24T23-30-00
```

This indicates a log rotation performed at exactly 11:30 pm on November 24th, 2011 UTC, which will be the local time offset by the local time zone. The original log file is the one with the timestamp, and new log lines are now being written to the `server1.log` file.

If another logRotate command is given one hour later, an additional file will appear:

```
$ ls /var/log/mongodb/server1.log*
server1.log  server1.log.2011-11-24T23-30-00  server1.log.2011-11-25T00-30-00
```

The `server1.log.2011-11-24T23-30-00` file is unchanged from before, while `server1.log.2011-11-25T00-30-00` is the previous `server1.log` file renamed and `server1.log` is a new empty file that will receive new log output.

From the mongo shell

```
> use admin
> db.runCommand("logRotate");
```

Windows
The logRotate command is available on Windows in version 2.0.3 and higher

From the unix shell

Rotate logs for a single process

```
shell> kill -SIGUSR1 <mongod process id>
```

Rotate logs for all mongo processes on a machine

```
shell> killall -SIGUSR1 mongod
```

Linux
There is currently a bug with our handling of the SIGUSR1 signal which can cause the server to deadlock in certain cases (SERVER-4739). For now we recommend using the logRotate command from the mongo shell instead.

Windows
Windows does not have an equivalent to the unix `kill -SIGUSR1` feature, but the mongo shell can be used from the Windows command line using a JavaScript command file to issue a logRotate command.

```
C:\> type logRotate.js
db.getMongo().getDB("admin").runCommand("logRotate")
C:\> mongo logRotate.js
C:\> rem Log files rotated, still at Windows command prompt
```

Accessing Logs via Shell

New in 1.9.x
See the getLog Command for more details

Get a list of available loggers

```
> show logs
> db.runCommand( { getLog : "*" } )
```

Get a log

```
> show log global
> db.runCommand( { getLog : "global" } )
```

Data Center Awareness

- Examples
  - One primary data center, one disaster recovery site
  - Multi-site with local reads
- Confirming propagation of writes with getLastError
- Replicating from nearby members
- Tagging (version 2.0+)
  - Server *X* should have a copy.
  - Make *n* backups
  - Make sure there are at least three copies of the data and it is present on at least two continents.
  - Make sure at least two servers across at least two racks in nyc have it.
  - Notes
- See Also

Examples

One primary data center, one disaster recovery site

In this example set members at the main data center (sf) are eligible to become primary. In addition we have a member at a remote site that is never primary (at least, not without human intervention).

```
{ _id: 'myset',
  members: [
   { _id:0, host:'sf1', priority:1 },
   { _id:1, host:'sf2', priority:1 },
   { _id:2, host:'ny1', priority:0 }
  ]
}
```

Multi-site with local reads

The following example shows one set member in each of three data centers. At election time, any healthy up-to-date node, arbitrarily, can become primary. The others are then secondaries and can service queries locally if the client uses slaveOk read preference modes.

```
{ _id: 'myset',
  members: [
   { _id:0, host:'sf1', priority:1 },
   { _id:1, host:'ny1', priority:1 },
   { _id:2, host:'uk1', priority:1 }
  ]
}
```

Refer to your driver's documentation for more information about read routing.

Confirming propagation of writes with getLastError

Calling getLastError (called "write concern" in some drivers) with `w:"majority"` (v2.0+) assures the write reaches a majority of the set before acknowledgement. For example, if you had a three-member replica set, calling `db.runCommand({getLastError : 1, w : "majority"})` would make sure the last write was propagated to at least 2 servers.

Once a write reaches a majority of the set members, the cluster wide commit has occurred (see Replica Set Design Concepts).

Replicating from nearby members

In v2.0+, secondaries automatically sync data from members which are nearby. You can see the latencies that the `mongod` process is observing to its peers in the replSetGetStatus command's output. If nearby members are not healthy, more distant members will be used for syncing.

Example output, highlighting new ping time and sync target fields:

```
> rs.status()
{
        ...
        "syncingTo" : "B:27017",
        "members" : [
                {
                        "_id" : 0,
                        "name" : "A:27017",
                        ...
                        "self" : true
                },
                {
                        "_id" : 1,
                        "name" : "B:27017",
                        ...
                        "pingMs" : 14
                },
                {
                        "_id" : 2,
                        "name" : "C:27017",
                        ...
                        "pingMs" : 271
                }
        ],
        "ok" : 1
}
```

Tagging (version 2.0+)

Tagging gives you fine-grained control over where data is written. It is:

- Customizable: you can express your architecture in terms of machines, racks, data centers, PDUs, continents, etc. (in any combination or level that is important to your application).
- Developer/DBA-friendly: developers do not need to know about where servers are or changes in architecture.

Each member of a replica set can be tagged with one or more physical or logical locations, e.g., `{"dc" : "ny", "rack" : "rk1", "ip" : "192.168", "server" : "192.168.4.11"}`. *Modes* can be defined that combine these tags into targets for `getLastError`'s *w* option.

For example, suppose we have 5 servers, *A*, *B*, *C*, *D*, and *E*. *A* and *B* are in New York, *C* and *D* are in San Francisco, and *E* is in the cloud somewhere.

Our replica set configuration might look like:

```
{
    _id : "someSet",
    members : [
        {_id : 0, host : "A", tags : {"dc": "ny"}},
        {_id : 1, host : "B", tags : {"dc": "ny"}},
        {_id : 2, host : "C", tags : {"dc": "sf"}},
        {_id : 3, host : "D", tags : {"dc": "sf"}},
        {_id : 4, host : "E", tags : {"dc": "cloud"}}
    ]
    settings : {
        getLastErrorModes : {
            veryImportant : {"dc" : 3},
            sortOfImportant : {"dc" : 2}
        }
    }
}
```

Now, when a developer calls `getLastError`, they can use any of the modes declared to ensure writes are propagated to the desired locations, e.g.:

```
> db.foo.insert({x:1})
> db.runCommand({getLastError : 1, w : "veryImportant"})
```

`"veryImportant"` makes sure that the write has made it to at least 3 tagged "regions", in this case, "ny", "sf", and "cloud". Once the write has been replicated to these regions, getLastError will return success. (For example, if the write was present on *A*, *D*, and *E*, that would be a success condition).

If we used `"sortOfImportant"` instead, `getLastError` would return success once the write had made it to two out of the three possible regions. Thus, *A* and *C* having the write or *D* and *E* having the write would both be "success." If *C* and *D* had the write, `getLastError` would continue waiting until a server in another region also had the write.

Below are some common examples and how you'd specify tags and `w` modes for them.

Server X should have a copy.

Suppose you want to be able to specify that your backup server (B) should have a copy of a write. Then you'd use the following tags:

Server	Tags
B	`{"backup" : "B"}`

To define a mode for "server *B* should have a copy," create the mode:

```
backedUp : {"backup" : 1}
```

You want one server with a "backup" tag to have the write.

So, your config would look like:

```
{
   _id : replSetName,
   members : [
       {
           "_id" : 0,
           "host" : B,
           "tags" : {"backup" : "B"}
       },
       ...
   ],
   settings : {
       getLastErrorModes : {
           backedUp : {backup : 1}
       }
   }
}
```

To use this mode in your application, you'd call `getLastError` with `w` set to `backedUp`:

```
> db.runCommand({getLastError : 1, w : "backedUp"})
```

In the following examples, we will skip the configuration and the usage for brevity. Tags are always added to a member's configuration, modes are always added to `getLastErrorModes`.

Make n backups

Suppose you have three backup servers (*B1*, *B2*, *B3*) and you want at least two of them to have a copy. Then you'd give each of them a unique "backup" tag:

Server	Tags
B1	`{"backup" : "B1"}`
B2	`{"backup" : "B2"}`
B3	`{"backup" : "B3"}`

Then you would create the mode:

```
backedUp : {"backup" : 2}
```

Make sure there are at least three copies of the data and it is present on at least two continents.

All of the rules up until now have only had one condition, but you can include as many and-conditions as you want. Suppose we have the following:

Server	Tags
S1	`{"continent" : "nAmerica", "copies" : "S1"}`
S2	`{"continent" : "nAmerica", "copies" : "S2"}`
S3	`{"continent" : "nAmerica", "copies" : "S3"}`
S4	`{"continent" : "Africa", "copies" : "S4"}`
S5	`{"continent" : "Asia", "copies" : "S5"}`

Then create a mode like:

```
level : {copies : 3, continent : 2}
```

Note that modes can contain as many clauses as you need.

Make sure at least two servers across at least two racks in nyc have it.

This is a complication of our original example. The key concept here is that not all tags need to be present on all servers. For example, some servers below are tagged with "nyc", others are not.

Server	Tags
S1	`{"nycRack" : "rk1", "nyc" : "S1"}`
S2	`{"nycRack" : "rk2", "nyc" : "S2"}`
S3	`{"nycRack" : "rk2", "nyc" : "S3"}`
S4	`{"sfRack" : "rk1", "sf" : "S4”}`
S5	`{"sfRack" : "rk2", "sf" : S5"}`

Now our rule would look like:

```
customerData : {“nycRack” : 2}
```

Notes

The examples above generally use hostnames (e.g., `"nyc" : "S1"`). This isn't required, it's just and convenient way to specify a server-unique tag. You could just as well use "foo", "bar", "baz" or "1", "2", "3", or any other identifiers.

Do not use "*" or "$" in tags, these characters are reserved for future use.

See Also

- Tag Aware Sharding

DBA Operations from the Shell

This page lists common DBA-class operations that one might perform from the MongoDB shell.

Note one may also create .js scripts to run in the shell for administrative purposes.

```
help                          show help
show dbs                      show database names
show collections              show collections in current database
show users                    show users in current database
show profile                  show most recent system.profile entries with time >= 1ms
use <db name>                 set curent database to <db name>

db.addUser (username, password)
db.removeUser(username)

db.cloneDatabase(fromhost)
db.copyDatabase(fromdb, todb, fromhost)
db.createCollection(name, { size : ..., capped : ..., max : ... } )

db.getName()
db.dropDatabase()

// runs the collstats] command on each collection in the database
db.printCollectionStats()

db.currentOp() displays the current operation in the db
db.killOp() kills the current operation in the db

db.getProfilingLevel()
db.setProfilingLevel(level) 0=off 1=slow 2=all

db.getReplicationInfo()
db.printReplicationInfo()
db.printSlaveReplicationInfo()
db.repairDatabase()

db.version() current version of the server

db.shutdownServer()
```

Commands for manipulating and inspecting a collection:

```
db.foo.drop() drop the collection
db.foo.dropIndex(name)
db.foo.dropIndexes()
db.foo.getIndexes()
db.foo.ensureIndex(keypattern,options) - options object has these possible
                                         fields: name, unique, dropDups

db.foo.find( [query] , [fields])       - first parameter is an optional
                                         query filter. second parameter
                                         is optional
                                         set of fields to return.
                                    e.g. db.foo.find(
                                                  { x : 77 } ,
                                                  { name : 1 , x : 1 } )
db.foo.find(...).count()
db.foo.find(...).limit(n)
db.foo.find(...).skip(n)
db.foo.find(...).sort(...)
db.foo.findOne([query])

db.foo.getDB() get DB object associated with collection

db.foo.count()
db.foo.group( { key : ..., initial: ..., reduce : ...[, cond: ...] } )

db.foo.renameCollection( newName ) renames the collection

db.foo.stats()
db.foo.dataSize()
db.foo.storageSize() - includes free space allocated to this collection
db.foo.totalIndexSize() - size in bytes of all the indexes
db.foo.totalSize() - storage allocated for all data and indexes
db.foo.validate() (slow)

db.foo.insert(obj)
db.foo.update(query, object[, upsert_bool])
db.foo.save(obj)
db.foo.remove(query)                   - remove objects matching query
                                         remove({}) will remove all
```

See Also

- collStats (stats) command
- Commands

Troubleshooting

- Excessive Disk Space
- The Linux Out of Memory OOM Killer
- Index Versions

- mongod process "disappeared"
- Socket errors in sharded clusters and replica sets
- See Also

mongod process "disappeared"

Scenario here is the log ending suddenly with no error or shutdown messages logged.

On Unix, check /var/log/messages:

```
$ sudo grep mongod /var/log/messages
$ sudo grep score /var/log/messages
```

Socket errors in sharded clusters and replica sets

If you're experiencing unexplainable socket errors between nodes in a sharded cluster or replica set, you may want to check your TCP keepalive time. The default Linux keepalive time is 7200 seconds (2 hours); we recommed that you change this to 300 seconds (five minutes).

To check your current setting:

```
cat /proc/sys/net/ipv4/tcp_keepalive_time
```

You'll usually see this:

```
7200
```

To change this setting:

```
echo 300 > /proc/sys/net/ipv4/tcp_keepalive_time
```

Note that you must alter the `tcp_keepalive_time` value on all machines hosting MongoDB processes. This include machines hosting `mongos` routers, config servers, and `mongod` servers. You should not need to restart the mongo processes themselves, the setting will take effect immediately.

See Also

- Diagnostic Tools

Excessive Disk Space

- Understanding Disk Usage
  - local.* files and replication
  - Datafile Preallocation
  - Recovering Deleted Space
  - Running out of disk space
  - Checking Size of a Collection
  - Helpful scripts

Understanding Disk Usage

You may notice that for a given set of data the MongoDB datafiles in `/data/db` are larger than the data set inserted into the database. There are several reasons for this.

local. files and replication*

The replication oplog is preallocated as a capped collection in the local database.

The default allocation is approximately 5% of disk space (*64 bit installations*).

If you would like a smaller oplog size use the `--oplogSize` command line parameter.

Datafile Preallocation

Each datafile is preallocated to a particular size. (This is done to prevent file system fragmentation, among other reasons.) The first filename for a database is <dbname>.0, then <dbname>.1, etc. <dbname>.0 will be 64MB, <dbname>.1 128MB, et cetera, up to 2GB. Once the files reach 2GB in size, each successive file is also 2GB.

Thus, if the last datafile present is, say, 1GB, that file might be 90% empty if it was recently created.

Additionally, on Unix, mongod will preallocate an additional datafile in the background and do background initialization of this file. These files are prefilled with zero bytes. This initialization can take up to a minute (less on a fast disk subsystem) for larger datafiles. Pre-filling in the background prevents significant delays when a new database file is next allocated.

On Windows prior to v2.1.2, additional datafiles are not preallocated. NTFS can allocate large files filled with zeroes relatively quickly, rendering preallocation unnecessary.

As soon as a datafile starts to be used, the next one will be preallocated.

You can disable preallocation with the `--noprealloc` command line parameter. This flag is nice for tests with small datasets where you drop the database after each test. It should not be used on production servers.

For large databases (hundreds of GB or more), this is of no significant consequence as the unallocated space is relatively small.

On Linux systems you can use `hdparam` to allocate files to get an idea of how costly allocation might be:

```
time hdparm --fallocate $((1024*1024)) testfile
```

Recovering Deleted Space

MongoDB maintains lists of deleted blocks within the datafiles when objects or collections are deleted. This space is reused by MongoDB but never freed to the operating system.

To shrink the amount of physical space used by the datafiles themselves, by reclaiming deleted blocks, you must rebuild the database by using (`db.repairDatabase()`).

repairDatabase copies all the database records to new file(s). You will need enough free disk space to hold both the old and new database files while the repair is running. Be aware that repairDatabase will block and will take a long time to complete.

Rather than compacting an entire database, you can compact just a single collection by using `db.runCommand({compact:'collectionname'})`. This does not shrink any datafiles, however; it only defragments deleted space so that larger objects might reuse it. The compact command will never delete or shrink database files, and in general requires extra space to do its work. Thus, it is not a good option when you are running critically low on disk space.

When testing and investigating the size of datafiles, if your data is just test data, use db.dropDatabase() to clear all datafiles and start fresh.

Running out of disk space

If your server runs out of disk space for data files you will see something like this in the log:

```
Thu Aug 11 13:06:09 [FileAllocator] allocating new datafile dbms/test.13, filling with zeroes...
Thu Aug 11 13:06:09 [FileAllocator] error failed to allocate new file: dbms/test.13 size: 2146435072
errno:28 No space left on device
Thu Aug 11 13:06:09 [FileAllocator]     will try again in 10 seconds
Thu Aug 11 13:06:19 [FileAllocator] allocating new datafile dbms/test.13, filling with zeroes...
Thu Aug 11 13:06:19 [FileAllocator] error failed to allocate new file: dbms/test.13 size: 2146435072
errno:28 No space left on device
Thu Aug 11 13:06:19 [FileAllocator]     will try again in 10 seconds
```

The server remains in this state forever blocking all writes including deletes. However, reads still work. To delete some data and compact (see above), you must restart the server first.

If your server runs out of disk space for journal files, the server process will abort (exit). By default, journal files are created in a subdirectory called "journal" in the data directory, but you may elect to put the journal files on another disk by using a symlink.

Checking Size of a Collection

Use the validate command to check the size of a collection -- that is from the shell run:

```
> db.<collectionname>.validate();

> // these are faster:
> db.<collectionname>.dataSize(); // just data size for collection
> db.<collectionname>.storageSize(); // allocation size including unused space
> db.<collectionname>.totalSize(); // data + index
> db.<collectionname>.totalIndexSize(); // index data size
```

This command returns info on the collection data but note there is also data allocated for associated indexes. These can be checked with validate

too, if one looks up the index's namespace name in the system.namespaces collection. For example:

```
> db.system.namespaces.find()
{"name" : "test.foo"}
{"name" : "test.system.indexes"}
{"name" : "test.foo.$_id_"}
> > db.foo.$_id_.validate()
{"ns" : "test.foo.$_id_" , "result" : "
validate
  details: 0xb3590b68 ofs:83fb68
  firstExtent:0:8100 ns:test.foo.$_id_
  lastExtent:0:8100 ns:test.foo.$_id_
  # extents:1
  datasize?:8192 nrecords?:1 lastExtentSize:131072
  padding:1
  first extent:
    loc:0:8100 xnext:null xprev:null
    ns:test.foo.$_id_
    size:131072 firstRecord:0:81b0 lastRecord:0:81b0
  1 objects found, nobj:1
  8208 bytes data w/headers
  8192 bytes data wout/headers
  deletedList: 0000000000001000000
  deleted: n: 1 size: 122688
  nIndexes:0
" , "ok" : 1 , "valid" : true , "lastExtentSize" : 131072}
```

Helpful scripts

These one-line scripts will print the stats for each db/collection:

```
db._adminCommand("listDatabases").databases.forEach(function (d) {mdb = db.getSiblingDB(d.name);
printjson(mdb.stats())})

db._adminCommand("listDatabases").databases.forEach(function (d) {mdb = db.getSiblingDB(d.name);
mdb.getCollectionNames().forEach(function(c) {s = mdb[c].stats(); printjson(s)})})
```

The Linux Out of Memory OOM Killer

The Linux out of memory killer kills processes using too much memory. On a kill event you will see a line such as the following in the system log file:

```
Feb 13 04:33:23 hostm1 kernel: [279318.262555] mongod invoked oom-killer: gfp_mask=0x1201d2, order=0,
oomkilladj=0
```

On such an event, check in future runs that memory is not leaking. This can be checked by verifying that virtualbytes - mappedbytes for mongod is bounded. Check this in serverStatus and/or mongostat.

```
> db.serverStatus()
```

ulimit

`ulimit -m` with `mongod` may cause kills that are false positives. This setting is not recommended. MongoDB uses memory mapped files. The entire data is mapped. Over time, if there is no memory pressure, the `mongod` resident bytes may approach total memory, as the resident bytes includes file system cache bytes for the file pages open and touched by `mongod`.

swap

Please see the production notes about swap.

Index Versions

- Converting an Existing Index to { v:1} format
- Rolling Back to MongoDB < v2.0
- Building a { v:0} Index

MongoDB v2.0 uses a new data structure for its indexes. The new {v:1} indexes are, on average, 25% smaller than the old {v:0} indexes. This usually results in significantly increased performance. In addition, signed (pre 1970) dates are supported in {v:1} indexes.

Backward-compatibility with {v:0} indexes is maintained. Thus, you can upgrade to MongoDB v2.0 seamlessly. However, to get the benefits of the new index format, you'll need to reindex.

Converting an Existing Index to {v:1} format

All operations that create a new index will result in a {v:1} index by default.

- Reindexing will result in a replacement of any {v:0} index of with a {v:1} type.
  - Due to SERVER-3866, reindexing on a secondary does not work in <=v2.0.0. **Do not reindex on a secondary as the drop phase of reindex is done and then the rest of the work aborts**. See jira for workaround.
- A compact command invocation will convert all indexes for the given collection to {v:1} type.

In v2.0.1, the following operations will convert {v:0} indexes to {v:1} type:

- The repair database command.
- Creation of a new replica set secondary from scratch (initial sync).

Rolling Back to MongoDB < v2.0

While MongoDB v2.0 supports the old index format, **old versions will not support the new format**. If you need to roll back to an old version, the server will run, but queries and other operations involving the newer indexes will log and return an error. Thus, you'll need to recreate any new index you'd like to use on an old server.

Versions >= 1.8.3 are aware of the index version field, but version <= 1.8.2 are not. So if you rollback a {v:1} index to 1.8.2 and re-index it its version will still be marked {v: 1} although it actual version is {v: 0}. Then if you upgrade again to 2.0 this index won't work even though they are marked as {v: 1} in db.system.indexes. So if you have to rollback to a version <= 1.8.2, you must delete the index then create it again (instead of simply re-indexing).

Building a {v:0} Index

You can still create a {v:0} index with MongoDB v2.0. To do so, add the option {v:0} in the index creation command. For example in the mongo shell:

```
> // defaults to a v:1 index
> db.foo.ensureIndex({name:1})
> db.system.indexes.find()
{ "v" : 1, "key" : { "_id" : 1 }, "ns" : "mydb.foo", "name" : "_id_" }
{ "v" : 1, "key" : { "name" : 1 }, "ns" : "mydb.foo", "name" : "name_1" }
> db.foo.dropIndex({name:1})
{ "nIndexesWas" : 2, "ok" : 1 }

> // create a v:0 index
> db.foo.ensureIndex({name:1},{v:0})
> db.system.indexes.find()
{ "v" : 1, "key" : { "_id" : 1 }, "ns" : "mydb.foo", "name" : "_id_" }
{ "v" : 0, "key" : { "name" : 1 }, "ns" : "mydb.foo", "name" : "name_1" }
```

Production Deployments

MongoDB can be used for a lot of different projects; the goal is for it to be a fairly general purpose tool. As you can see, many of the use cases are online or operational databases in which someone is writing an application in front of the database. Business intelligence / analytics use is not unusual but usually more specialized. For example, real time dashboarding is highly popular, and when the data set gets really big, things can make a lot of sense. See the use cases page for more on this and for more on when you *wouldn't use it*.

If you're using MongoDB in production, we'd love to list you here! Please complete this web form and we will add you.

- Archiving
- Cloud Infrastructure Providers

- Content Management
- Ecommerce
- Education
- Email
- Finance
- Gaming
- Government
- Healthcare
- Infrastructure
- Intranet
- Metadata Storage
- Mobile
- News & Media
- Online Advertising and Marketing
- Online Collaboration
- Real-time stats/analytics
- Scientific
- Social Networks
- Telecommunications
- More MongoDB Users
- See also

Archiving

Company	Use Case
craigslist	Craigslist uses MongoDB to archive billions of records. • Lessons Learned from Migrating 2+ Billion Documents at Craigslist - MongoSF (May 2011) • MongoDB Live at Craigslist - MongoDB blog (May 2011) • MongoDB and Craigslist's Data Storage Evolution - MongoSV (December 2010)

Cloud Infrastructure Providers

Firebase	Firebase is a cloud service that automatically synchronizes data between clients and cloud servers. Apps built with Firebase can be written entirely with client-side JavaScript, update in real-time out-of-the-box, are inherently scalable, and interoperate well with existing services. Firebase offers developers a massively scaleable real-time backend for web applications. Firebase stores all its data in MongoDB which offers the built-in capability for apps to scale automatically and gives each piece of data its own unique URL stored in JSON documents.
MongoHQ fast & easy data management	MongoHQ provides a hosting platform for MongoDB and also uses MongoDB as the back-end for its service. Our hosting centers page provides more information about MongoHQ and other MongoDB hosting options.

Content Management

Company	Use Case

	SAP uses MongoDB a core component of SAP's platform-as-a-service (PaaS) offering. MongoDB was selec for the enterprise co management (ECM) section of the platfor as its flexibility and scalability will enable SAP to scale its con management service its PaaS offering to r customers' requirem while managing data from different applications. More information available the 10gen blog. • Introduction SAP's Java Platform as Service - Di Guendisch's Presentation MongoUK (September 2011)
	MongoDB is the repository that powe MTV Networks' next-generation CMS which will eventually used to manage and distribute content for of MTV Networks' m websites. Current deployments include SpikeTV.com and Comedy Central Indecision, and MTV Networks will be rolli out MongoDB on ma other major sites wit the next year, most l including gametrailers.com, thedailyshow.com, comedycentral.com, nick.com, and nume international propert Read more on the MongoDB blog. • How MTV Leverages MongDB for CMS - Jeff Yemin's Presentation MongoBost 2011 (Octob 2011)
	MongoDB is used fo back-end storage on SourceForge front pages, project pages and download pages all projects.

- Realtime Analytics us MongoDB, Python, Gev and ZeroMC Rick Copela Presentatio MongoSV (December 2011)
- Achieving 2 Years of Mc Stability and Performanc SF.net - Ma Ramm's Presentatio MongoDalla (November 2011)
- Rapid and Scalable Developme with Mongo PyMongo a Ming - Mark Ramm's Presentatio MongoDalla (November 2011)
- Allura - An Open-Sourc MongoDB-b Document Oriented SourceForg Rick Copela Presentatio MongoSF (N 2011)
- How SourceForg Uses Mong - Rick Copeland's Presentatio MongoAtlan (February 2
- Scaling SourceForg with Mongo OSCON Presentatio (July 2010)
- MongoDB a SourceForg QCon Lond Presentatio (March 201
- How Pythor TurboGears and MongoI are Transformin SourceForg - PyCon (February 2
- SourceForg releases Mi SourceForg blog (Decer 2009)

- TurboGears
  Sourceforge
  Compound
  Thinking (Ju
  2009)

Wordnik stores its e
text corpus in Mongo
- 3.5T of data in 20 b
records. The speed
query the corpus wa
to 1/4 the time it tool
prior to migrating to
MongoDB. More abo
MongoDB at Wordni

- From the Cl and Back - Presentatio MongoSV (December 2011)
- What drove Wordnik non-relation (August 201
- Building a Directed Gr with Mongo Tony Tam's Presentatio MongoSF (M 2011)
- Managing a MongoDB Deployment Presentatio Large Scale Production Engineering Meetup (February 2
- Wordnik: 10 million API requests a d on MongoD High Scalab Blog (Febru 2011)
- Keeping the Lights on wi MongoDB - Tony Tam's Presentatio MongoSV 2 (December 2010)
- 12 Months v MongoDB - Wordnik Blo (October 20
- B is for Billio Wordnik Blo (July 2010)
- MongoDB: Migration fro Mysql at Wordnik - Scalable We Architecture (May 2010)
- Tony Tam's Presentatio MongoSF (A 2010)
- What has technology for words la - Wordnik bl (February 2

Harmony is a power web-based platform creating and managi websites. It helps developers with cont editors work togethe with unprecedented flexibility and simplic From stylesheets, images and templat pages, blogs, and comments, every pie of Harmony data is stored in MongoDB. Switching to Mongol from MySQL drastic simplified Harmony's data model and increased the speed which we can delive features.

- Real World Modeling wi MongoDB a Harmony - S Smith's Presentatio MongoBost (September 2010)
- Steve Smith presentatio about Harm at MongoSF (April 2010)

eHow, a web proper Demand Media, rece 100MM+ unique use on a large content database. MongoDB perfectly suited for eHow's content management, as the document model exa matches the delivery model of documents pages. The Demand Media team rewrote legacy inherited .NET/SQL system to PHP/Python/Mongol a cachless, tierless design, all on one machine with replica

MeteKamil uses MongoDB for browsi hundreds of thousar of videos on the websites:
YouUnbox.com
YouGamePlay.com
YouOverClock.com
DroidFather.com
YouTouchDown.com

	Omnidimensional us MongoDB to store a retrieve large numbe records. Omnidimensional is hybrid company that supplies its custome with high-end web applications, user-experience des solutions, webdesigr music or tunes for us commercials.
	Watchlater uses MongoDB as their primary database fo their Watchlater app
	Web Genius uses MongoDB to host 7C small-to-medium business sites.
	9elements uses MongoDB in their Ra projects meinpraktikum.de
	Tus Media uses MongoDB as their content managemer database.
	Amalya Tek uses MongoDB as their content repository database.

productionminds	Production Minds us MongoDB as their primary graph-like database for a comp web application.

Ecommerce

Company	Use Case
CustomInk Design T-shirts Online	At CustomInk, we be custom t-shirts have power to turn your g into a team, elevate gathering to an even make your special m more... well, momen That's why we make to create awesome s CustomInk uses Mo for a number of diffe applications includin • Supply Cha Manageme MongoDB s information t-shirt suppl local screen printers. • Content Manageme MongoDB is to store the catalog, Cus sells hundre products, nc t-shrits. • Logging - M is used for l interactions website and central store application l Learn more how Custom using Mong from their presentation at MongoSV

OpenSky is a free or platform that helps p discover amazing pr share them with thei friends, family and fo and earn money. Op uses MongoDB, Syn Doctrine 2, PHP 5.3, PHPUnit 3.5, jQuery node.js, Git (with gitl and a touch of Java Python. OpenSky us MongoDB for just ab everything (not just analytics). Along the they've developed MongoODM (PHP) a MongoDB drivers for and CAS.

- Blending Mo with RDBMS e-commerce Steve Franc presentation MongoNYC 2011)
- Augmenting RDBMS with NoSQL for e-commerce Steve Franc presentation East (March
- MongoDB & Ecommerce Perfect Combinatio Video from Francia's presentation New York MongoDB U Group (Octo 2010)

Gilt Groupe is an inv only luxury shopping Gilt uses MongoDB time ecommerce ana

- MongoDB a Presentation MongoDB S 2012
- Gilt CTO Mi Bryzek's presentation MongoSF in 2010.
- Hummingbi real-time we visualization developed b and powere MongoDB

edelight	edelight is a social s portal for product recommendations. • MongoDB a edelight: A Pain and Su in 10 Slides Fabian Schl Presentatio MongoBerli (October 20 • MongoDB: Edelight sta MySQL auf MongoDB s Exciting Ecommerce (September
totsy	Totsy offers moms on-the-go and moms access to brand-spe sales, up to 70% off Totsy was re-built up and MongoDB to co performance and sc limitations incurred v the prior relational-d platform. The transiti MongoDB has resol of Totsy's performan scaling issues. • MongoDB Ecommerce Study: Totsy Pirtle's pres at Mongo B (September
PlumWillow shop.style.share.	PlumWillow is a Soc Shopping Network w girls who like fashion build outfits by drag-and-drop, selec from thousands of to items. PlumWillow w by a "dream team" o core-developers/con to PHP, jQuery and MongoDB who utiliz Agile efficiency of M and the Vork Entepr Framework to bring PlumWillow from concept-to-launch in few months.

	Shopwiki uses Mong data store for its sho search engine, wher commit all the data generated, such as analytics. Mongo's performance is such ShopWiki uses it in where MySQL would not be practical. Sho also using it as a sto engine for all R&D a data-mining efforts MongoDB's docume oriented architecture maximum flexibility. • Avery's Talk MongoNYC ShopWiki D (June 2010)
	Yabblr uses MongoD everything, including inventory manageme orders. Yabblr is an E-Commerce platfor flips 'deals' on its he Instead of pushing r offers at you, Yabblr empowers you to gr together with friends deals on products yo select. Yabblr uses MongoDB exclusivel data store for the en platform, including: - data - Real time met are used for analyzi behavior and system - Inventory manager Customer orders - E queues and messag between different sy
	Ordoro uses Mongol store orders, invento suppliers. Ordoro de everything that happ after a shopper click checkout on the reta website: print packin and shipping labels, manage inventory le manage dropshippin manage customers, manage suppliers et Ordoro is the backof control panel for SM etailers.
	SavingStar uses Mo to store our coupon and service metadat well as all of our cou activation data. The company is the first national, fully digital, eCoupon service, av for free. There's noth clip, nothing to print.

WEDDING*PAPER divas™	Wedding Paper Diva MongoDB to store o categories and products.The compa allows users to creat custom wedding invi online using their personalization tools make unique and tre wedding invitations d and easily.
storeden	Storeden is a cloud s totally based on Mor The easiness of the popular shopping we meets the performar an advanced online solution. Storeden is revolutionary e-Com platform that allows and web-agencies to amazing on-line stor just a few clicks. Eve has the chance to cr applications, graphic themes and “suites” sell them to the othe in the Cloud. Stored be perfectly integrate the most common E CRM platforms, allo user to effectively le on Facebook, Ebay, Shopping and Kelko further sales channe is natively optimized SEO.
gitti gidiyor®	Gittigidiyor.com is th biggest e-commerce Turkey with more tha million registered us Gittigidiyor.com have replicated and shard MongoDB cluster for and for caching a wi range of data.
iPara	iPara uses MongoDI payment pages to lo kind of user activites logging, they analyz logs to determine ne designs and process iPara offers safe and convenient payment discount campaigns, providing many adva for both users and businesses.
coinbase	Coinbase uses Mong for their primary data for their web app, ap requests, etc. Coinb decentralized, digital currency that is char the world of paymen

	local.ch uses Mongo their central data sto serve complete docu for all of their phonel entries and geo data is consumed by the www.local.ch as wel their mobile apps.
	Turkishexporter.net MongoDB to list and their products, comp and leads.
	Bestcovery.com use MongoDB for MyBes pages, which allow u discover interesting products and curate their own favorite pr and services to shar the world.
	Bestekortingscode.n MongoDB as their p database.
	Wish is a social platf helping connect con to merchants with gr deals on the product want. They use Mon as their primary data to serve up hundred gigabytes of data to 10 million users in re It stands behind eve on the site, from thei personalization engi their commerce platf and supports their gr millions of new user month.

See also: http://www.slideshare.net/mongodb/mongodb-at-ebay

Education

	Codecademy is the easiest way to learn to code online. Driven by the belief that programming is one of the most important skills of the twenty-first century, Codecademy has educated more than one million people in over 100 countries.
	LearnBoost is a free and amazing gradebook web app that leverages MongoDB for its data storage needs. LearnBoost is the creator of Mongoose, a JavaScript async ORM for MongoDB that is flexible, extensible and simple to use. • Mongoose - LearnBoost blog (May 2010)
	Courseoff uses MongoDB stores student created schedules as well as the schedule listings themselves.

	Boundless Learning uses MongoDB as a rapid-prototyping technology to develop a robust content management system.
	Foothill-De Anza Community College District migrated a problematic document storage system into a web application backed by MongoDB, and using the Mongo GORM plug-in for Grails. They have indexed (and stored with GridFS) over 300,000 documents (56GB) into a single Mongo database. Mongo is much easier to get online than a comparable RDBMS, and the rich feature set helped significantly reduce the amount of required application coding. All said and done, switching to Mongo for this project has been a big win.

Email

	Sailthru is an innovative email service provider that focuses on improving the quality of emails over quantity. Moving to MongoDB from MySQL allowed us extreme flexibility in providing an API to our clients. Passing in arbitrary JSON data is easy – our customers can use objects and arrays inside of their emails. And we've launched Sailthru Alerts, which allows our customers basically whitelabeled Google Alerts: realtime and summary alerts (price, tag match, etc) that are completely up to the customer due to our schema-free data storage. Also, we can gather realtime behavioral data on a client's signed-up users (tag-based interests, geolocale, time of day, etc), and use all that information to power dynamically assembled mass emails that get higher click-through rates than static emails. Plus we just launched an onsite recommendation widget (check it out at refinery29.com), and we're using MongoDB's analytic capabilities to rapidly A/B test different behavioral algorithms. • From Cloud to Colo - Presentation at MongoDB Seattle 2012 • Two Years of MongoDB at Sailthru - Presentation at MongoNYC 2012 • Scaling and Schema Design - Presentation at MongoNYC 2011 • MongoDB in Production at Sailthru - Presentation to NY MongoDB User Group (January 2011)
	fiesta.cc makes creating mailing lists dead-simple, and fun. The goal is to bring groups together around a tool that everybody already uses and understands: email. We use MongoDB to store all of the data powering the website and the mail serving infrastructure. We're running a three node replica set. • MongoDB at fiesta.cc - Mike Dirolf's Presentation for NYC Meetup Group (November 2011) • Behind the Scenes at fiesta.cc - Mike Dirolf's Presentation at PyGotham (September 2011)

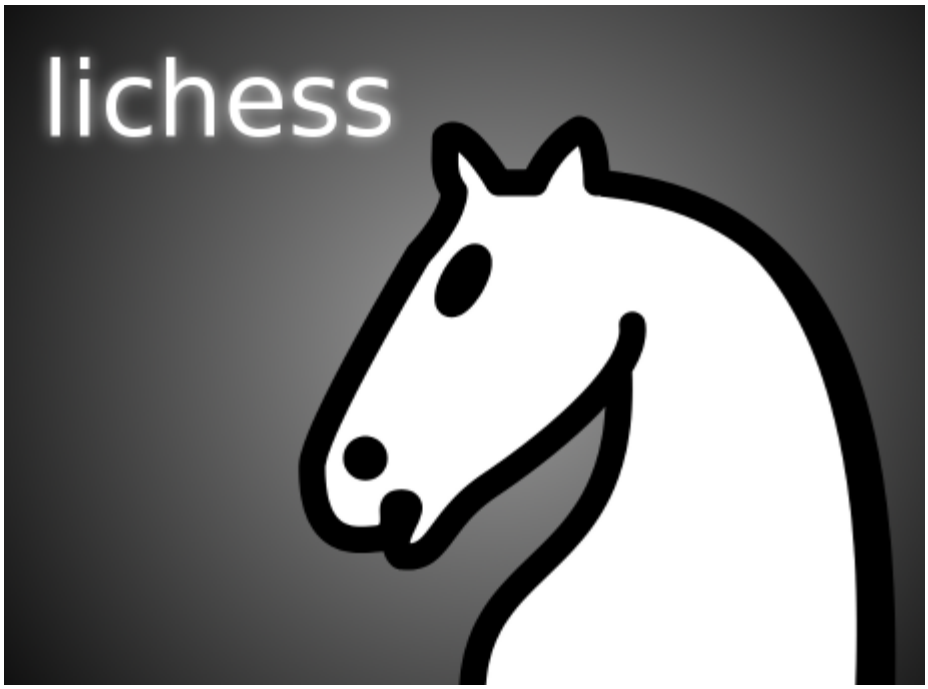
Finance

Company	Use Case
	SecondMarket is the online marketplace for alternative assets such as private company stock, structured products & bankruptcy claims. SecondMarket uses mongoDB for storing its diverse asset classes in a single collection utilizing the power of the schema free nature of mongoDB. We also store organization news and activity in mongoDB. • Mongo & Mongeez on the SecondMarket Engineering Blog - Feb 2012 • Mongeez the open source, change management tool developed for MongoDB.] • Second Market, MongoDB and Mongeez for Change Management - Michael Lysaght's Presentation at MongoNYC (June 2011) • Mongeez at SecondMarket -Michael Lysaght's presentation at the MongoDB User Group (January 2012)
	Athena Capital Research is a quantitative investment manager, specializing in automated trading strategies. • How a Hedge Fund Uses MongoDB - Roman Shtylman's Presentation at MongoNYC (June 2011) • Low Latency Event Logging with BSON - Andrew Morrow's Presentation at MongoSV (December 2010)
	Equilar uncovers the direct pathways to the most influential high net worth individuals, and delivers immediate and actionable insight into the equity wealth events that drive business development opportunities. Equilar uses Mongo DB to map and analyze the connections between over 300,000 executives and board members in companies worldwide.
	Auriga USA is a financial services company that operates in the residential mortgage space. Moving to MongoDB solved a host of problems, and assisted Auriga USA in upgrading the functionality of their loan inventory management system to handle many new features and different types of assets, including student loans, credit cards, and asset-back securities.

Gaming

Company	Use Case
	Disney built a common set of tools and APIs for all games within the Interactive Media Group, using MongoDB as a common object repository to persist state information. • A Year with MongoDB: Running Operations to Keep the Game Magic Alive - Curt Steven's Presentation at MongoSV (December 2011) • Disney Central Services Storage: Leveraging Knowledge and skillsets - MongoSF (May 2011)

	IGN Entertainment, a unit of News Corporation, is a leading Internet media and services provider focused on the videogame and entertainment enthusiast markets. IGN's properties reached more than 37.3 million unique users worldwide February 2010, according to Internet audience measurement firm comScore Media Metrix. MongoDB powers IGN's real-time traffic analytics and RESTful Content APIs. • Using MongoDB for IGN's Social Platform - Presentation to San Francisco MongoDB User Group (February 2011) • Confessions of a recovering relational addict - Presentation by Chandra Patni at MongoSV (December 2010)
	WordSquared (formerly Scrabbly) is a massively multiplayer online ~~scrabble~~ crossword. Uses MongoDB geospatial indexing. • Mapping Flatland: Using MongoDB for an MMO Crossword Game - Presentation at Mongo Seattle 2011 • Building a Scrabble MMO in 48 Hours - Startup Monkeys Blog (September 2010)
	MongoDB is being used for the game feeds component. It caches game data from different sources which gets served to ea.com, rupture.com and the EA download manager.
	Kano Games uses MongoDB to build an online gaming web app that integrates social elements. It is a gaming web app that offers a large collection of html5 and flash games with new ones added daily. MongoDB is used to store all persistent data at Kano Games. Reliability, scalability, performance, redundancy and flexibility were all considered when selecting a persistent storage solution and MongoDB beat out the competition as it best fit Kano Games needs.
	Shadelight is a unique fantasy roleplaying game where you play one of the legendary Guardians of Elumir. Set out on magical quests, battle mysterious creatures and explore a truly unique fantasy world.

adyax	Freerice is developped by World Food Porgramme, the largest UN humanitary organisation. Freerice.com is a social quizz-like game where each good answer to a question make you win 10 rice grains. Since the begining more than 85 billions of rice grains were won trough freerice. Fully connected with twitter, facebook and groups, freerice tracks down each gain of each person. This generates almost 1 MongoDB row for each right answer, wich mean 8 billions of rows currently stored in Freerice MongoDB. Numerous totals are provided to the end user, which are all stored in Mongo too.
	lichess.org uses MongoDB to store more than 3 million chess games. It also contains the analysis, users, forum posts and chat messages. Lichess.org needs blazing fast updates and efficient indexing; mongodb gives it both. Lichess.org is OpenSource! Check the code on http://github.com/ornicar/lila. MongodDB is the database behind Moshen Limited's Facebook game Gambino Poker. MongoDB is used to store all game events and player profiles. We use a 3 node replica-set and RAID 10 EBS Volumes.
moshen	MongodDB is the database behind Moshen Limited's Facebook game Gambino Poker. MongoDB is used to store all game events and player profiles. Moshen uses a 3 node replica-set and RAID 10 EBS Volumes.
	Hitcents uses MongodDB for primary data storage for their iOS/Android app PictureTHIS. The app is a cooperative social word game.

	Gamera Labs uses MongoDB for the Real-Time game High5Poker. The app is a cooperative social word game. MongoDB is used in production for data storage like players information and statistics.
	Doodle or Die uses MongoDB for almost everything in their online drawing game, including game and player information. Their logging and error tracking is also done through Mongo. More can be found in the MongoDC presentation Mongo or Die.

Government

	The National Archives (UK) made details of 11m records available through an application interface it published as part of an ongoing programme to get more official records online. It is consolidating numerous existing electronic archives, either by porting them directly or putting them in a service wrapper that can communicate with its unified system. The unified system uses MongoDB and is built on a Microsoft software stack. • National Archives releases public application programming interface for 11m records - ComputerWorld (Sept 2011) • From SQL Server to MongoDB - Presentation at MongoDB UK (Sept 2011)
	The British Government launched a beta of its GOV.UK platform, testing a single domain for that could be used throughout government. According to James Stewart, the Tech Lead on the beta of GOV.UK, "We started out building everything using MySQL but moved to MongoDB as we realised how much of our content fitted its document-centric approach," said Stewart. "Over time we've been more and more impressed with it and expect to increase our usage of it in the future." • Colophon for the GOV.UK beta - Feb 2012 • With GOV.UK, British government redefines the online government platform - O'Reilly Radar (Feb 2012)

	Sunlight Labs is a community of open source developers and designers dedicated to opening up our government to make it more transparent, accountable and responsible. MongoDB powers the National Data Catalog and the Drumbone API, which is an aggregator of data about members of Congress. • MongoDB at Sunlight - Luis Martinez's Presentation Slides • Civic Hacking - Video from Luigi Montanez's presentation at MongoNYC (May 2010) • How We Use MongoDB at Sunlight blog post (May 2010)

In addition, MongoDB has been used for various projects at the federal, state, and city levels in the U.S.

Note: If you are in Washington D.C. area, be sure to check out the annual Mongo conference there ("MongoDC"), as well as local mongo user groups.

Healthcare

Company	Use Case
	Santosoft uses MongoDB for virtually 100% of its data related stack. They run a fully blown shard cluster for transactional data, as well as analytics. They are building apps and tools for MongoDB and also hope to build the largest common care database globally.
	HolaDoctor.com is the most comprehensive health and wellness portal available in Spanish for the global online Hispanic community. MongoDB is being used to store all the content for the site, including GridFS to store article images. Session data is also being persisted on our MongoDB cluster using a custom PHP save handler.
	Vitals.com consolidates and standardizes doctor and other health provider data from over 1,000 sources to help users make informed health care choices. Our technology also powers the websites and backends of insurance companies, hospitals, and other health information brokers. In early October, we switched the datasource for our Find A Doctor location-based search functionality from PostgreSQL to a geo-indexed MongoDB collection. Since then, searches are now five to ten times as fast, and the decreased load on our dataservers permits us to serve more clients. Based on this success, we are transitioning other functionality to MongoDB datasources.
	TalkAboutHealth matches people with similar health concerns for answers and live chats. Meet others just like you, learn from experienced peers, and share your experiences.
	Earlydoc models medical data, which means things like master-identity management and data normalization occupy a lot of development time: this, specificly, is where the schemaless design of MongoDB is a **huge** win.
	D Sharp uses MongoDB to store detailed medical observations and perform complex analysis to help people with diabetes better manage their blood sugars. Built using the Express web framework for Node.js and JQuery Mobile, D Sharp is the only diabetes application that support all types of diabetes on any modern smartphone or browser. This includes iPhone, iPad, Android, Windows Mobile, Blackberry and desktop browsers.
	Vigilant Medical uses MongoDB to manage the large, complex, and often sparse metadata surrounding medical images. MongoDB also provides the entire administrative backbone of Vigilant's enterprise web application, ImageShare.

	Mini Medical Record is designed to improve medical care. While designed to help everyone, it is especially useful for travelers, and others who may receive care through multiple medical systems. Whether paper or electronic, medical records in cutting edge hospitals are often geared more for billing and medicolegal protection. Critical up-to-date information may be very challenging to discover. In addition, even simple information like emergency contact phone numbers and latest medication lists, may be inaccessible for patients who seek care in settings where they do not routinely receive their care. Mini Medical Record alleviates those issues.

Infrastructure

Company	Use Case
LexisNexis	LexisNexis Risk Solutions serves the multi-billion dollar risk information industry, which is comprised of professionals and organizations such as law enforcement, government agencies, financial services firms, collection agencies, insurance and health care providers, and hiring managers. MongoDB is used by the DevOps group to store their CMDB information, static web content, auditing data, and serves as the persistence engine for their asynchronous messaging fabric.

Intranet

	SPARC uses MongoDB as its primary store for our employee engagement platform. SPARC has been very happy with how simple data migrations has been since leaving SQL behind.

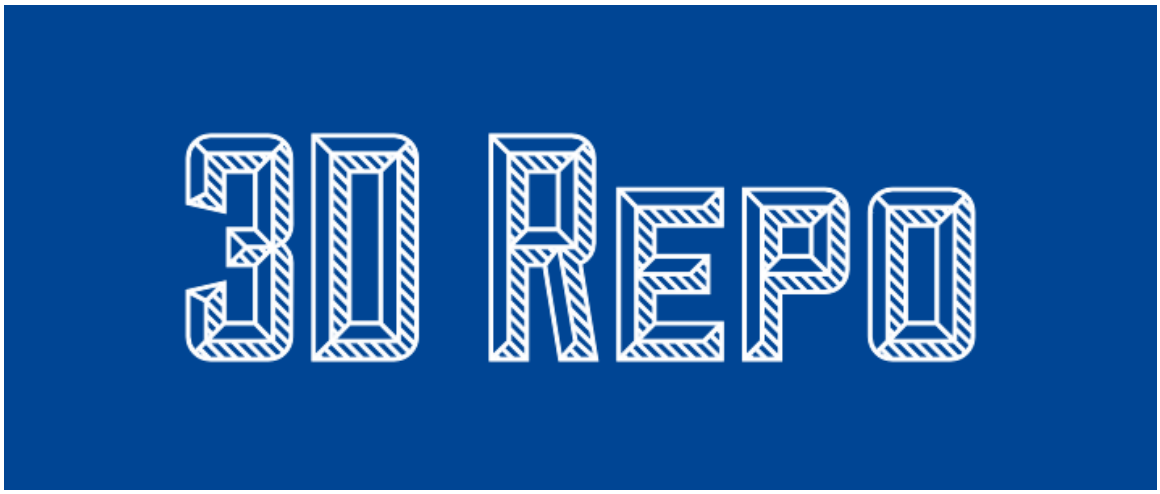
A Fortune 10 company uses MongoDB for their internal intranet app for their 100,000+ employees.

Metadata Storage

Company	Use Case

Shutterfly is an Internet-based socia expression and personal publishing service. MongoDB is used for various persistent data storage requirement within Shutterfly. MongoDB helps Shutterfly build an unrivaled service tha enables deeper, mor personal relationship between customers and those who matte most in their lives.

- Performanc Tuning and Scalability - Kenny Gorman's Presentatio at MongoS\ (December 2011)
- The Shutterfly Data Layer and Schem Definitions - Luciano Resende's Presentatio at MongoS\ (December 2011)
- MongoDB Profiling an Tuning - MongoSF (May 2011)
- Q & A with Shutterfly Data Architect Kenny Gorman (Ja 2011)
- Sharing Life's Joy using MongoDB: Shutterfly Case Study Kenny Gorman's presentatio at MongoS\ (December 2010)
- *Implementir MongoDB a Shutterfly* from MongoSF (April 2010) Slides and Video

	Lulu's open publishing platform empowers more creators to sell more content to more readers more profitably than ever before. Lulu uses MongoDB to store th bibliographic, product, and file metadata for its creators' works. MongoDB's document-based dat model makes it easy to manage the differences between print books, eBooks, photo books, and calendars and to keep up with the rapidly evolving worl of self publishing. • Mission Critical MongoDB - Presentatio from MongoDB Atlanta 201 • Why we decided NoSQL was right for us, How we came to choose MongoDB - Sept 2011 Presentatio
	3DRepo.org uses MongoDB as a domain specific repository to store scene graph components as well as associated non-linear revision history of 3D assets. The main reasons fo choosing MongoDB over alternative NoSQL solutions is the use of BSON documents for storage and retrieval automated sharding, MapReduce functionality and built-in support for geospatial indexing.

Mobile

(See also "Social" section.)

Company	Use Case

	MoPub uses MongoDB for realtime stats, budgeting systems, and user storage.
	Countly is an extensible, open source mobile analytics solution for mobile application developers. It tracks applications, customer behaviour or game mechanics - so you can focus on increasing user loyalty and engagement. With Countly, collected data is converted into meaningful information in true real-time, thanks to underlying infrastructure with MongoDB, Node.js and Nginx.
	Localytics uses MongoDB to process over 100M datapoints every day for their mobile analytics service.
	Mobile Helix uses MongoDB to provide fast data read/write for the Mobile Helix Gateway, which analyzes data that needs to be delivered from an enterprise's network to a mobile device and pre-fetches, batches and schedules its delivery to be as quick and as efficient as possible. MongoDB enables the gateway to seamlessly scale to millions of devices accessing the Mobile Helix system from all over the globe.
	Nearley is a mobile geo location application to list all facebook users located within your distance. Nearley uses MongoDB to store user interaction and activities in real-time.

	O2 uses MongoDB for several applications, including their Priority Moments location based offer service.

News & Media

Company	Use Case
	Guardian.co.uk is a leading UK-based news website. They' spent ten years fight relational database representations of ou domain model, until implementation of ou API made us realize that if only we could store documents everything would be simpler. MongoDB is key part of the Guardian's infrastructure. • Evolving fro relational to document store - MongoSF (May 2011) • Why I Chos MongoDB fo guardian.co - QCon London (August 201 • MongoDB a the Guardia MongoDB U (Sept 2011)
	Examiner.com is the fastest-growing local content network in th U.S., powered by the largest pool of knowledgeable and passionate contributi in the world. Launch in April 2008 with 60 cities, Examiner.com now serves hundred of markets across th U.S. and Canada. Examiner.com migrated their site fr Cold Fusion and SQ Server to Drupal 7 a MongoDB. Details o the deployment are outlined in an Acquia case study

Business Insider has been using MongoD since the beginning 2008. All of the site's data, including posts comments, and ever the images, are stor on MongoDB. For m information:

- An Inside Lo At The Guts Our Tech Infrastructur (June 2011 Article)
- How Busine Insider Uses MongoDB (May 2010 Presentatio
- How This W Site Uses MongoDB (November 2009 Article

Forbes has been around for nearly 10 years and on the we for more than a decade; recently, the media space has be a changing landscap One of the changes that Forbes has undertaken starting i 2010 was the “openi up” of its digital and print platforms to a global collection of content creators, marketers and audience members. This necessitated th changing of how the site worked, allowing content to flow easily through the site. Forbes began to evaluate how they store and serve their content and decided use MongoDB. They are currently using MongoDB for storing articles and compan data and are working move more of their core assets onto it.

- Supporting Distributed Global Workforce o Contributors with Mongo David Dunlo Presentatio at MongoNY (June 2011)

The New York Times	The New York Times using MongoDB in a form-building application for photo submissions. Mongo dynamic schema giv producers the ability define any combinat of custom form fields For more information • MongoDB in Production a the New Yo Times - Presentation from MongoNYC 2012 • The New Yo Times R&D Lab and MongoDB - Presentation from MongoNYC 2011 • Building a Better Submission Form - NYTimes Open Blog (May 25, 2010) • A Behind th Scenes Loo at the New York Times Moment in Time Projec Hacks/Hack Blog (July 2 2010)
Chicago Tribune	The Chicago Tribune uses MongoDB in its Illinois School Repor Cards application, which is generated from a nearly 9,000 column denormalize database dump produced annually b the State Board of Education. The application allows readers to search by school name, city, county, or district an to view demographic economic, and performance data fo both schools and districts.

	CNN Turk uses MongoDB for its infrastructure and content managemer system, including th tv.cnnturk.com, www.cnnturk.com ar video.cnnturk.com portals.

Online Advertising and Marketing

Company	Use Case
	MongoDB will be the central data store in Magnetic's system, providing low-latency, high-volume reads in a distributed cluster behind its pixel servers, and near-real-time aggregate counts to its clients for forecasting. Magnetic is excited by the speed, replication features, flexible data model, and map-reduce style query access, which will let it dramatically simplify its system relative to the current architecture using a key-value store. • Tracking & Analytics with MongoDB at Signpost; MongoDB for Online Advertising - Matt Insler's Presentation for NYC Meetup Group • MongoDB Analytics for Online Advertising at Magnetic - Webinar hosted by Mark Weiss
	Aggregate Knowledge is an IaaS company that combines data management and multi-touch attribution to make media accountable helping advertisers, agencies, ad networks, DSPs, SSPs, and publishers manage and exploit all of their data around a single view of the user. With more data to process, and less time at hand, Aggregate Knowledge pairs MongoDB with Fusion-io flash memory solutions for a winning combination.
	MediaMath is the leader in the new and rapidly growing world of digital media trading.
	Konverta.ru is the first Real Time Bidding (RTB) ad exchange on the Russian and CIS online advertising market. MongoDB is used to store all ads, ad impressions, clicks, and other data, as well as for real-time reporting and optimization.
	G5 is the largest and fastest growing provider of vertical-specific local marketing solutions that help mid-market companies get found online, generate more qualified leads, convert more leads into new customers, track marketing performance - including offline, and optimize to the marketing sources with the best return on investment. G5 migrated its analytics platform from MySQL to MongoDB due to the heavy demands of storing & processing analytical data. MongoDB has proven to be fast, scalable, flexible, & maintainable. Best of all, MongoDB is supported by a fantastic community!

	Yodle uses MongoDB to persist queues of items to be synchronized with several partner APIs. Mongo is ideally suited for this update-heavy performance sensitive workload.
	The Localstars platform makes it easy for local advertisers to build and manage locally targeted advertising campaigns. The new Localstars advert server cluster uses sharded MongoDB to provide super high performance real time ad serving decision making and campaign statistics.
	EveryScreen is built on a scalable, high-performance, real-time mobile advertising bidding platform in Amazon EC2.
	Rafflecopter is a web app that helps online publishers run sweepstakes on their own blogs & other sites. Rafflecopter logs and stores every entry to every contest that's ever been run, all contest details, user profiles, various logs/metrics, etc.
	LocalEver is a location based online market place to buy, sell & rent residential/commercial properties, automobiles, and a host of other products. MongoDB is the backbone of the system, which powers the whole range of features from geo spatial search to caching.

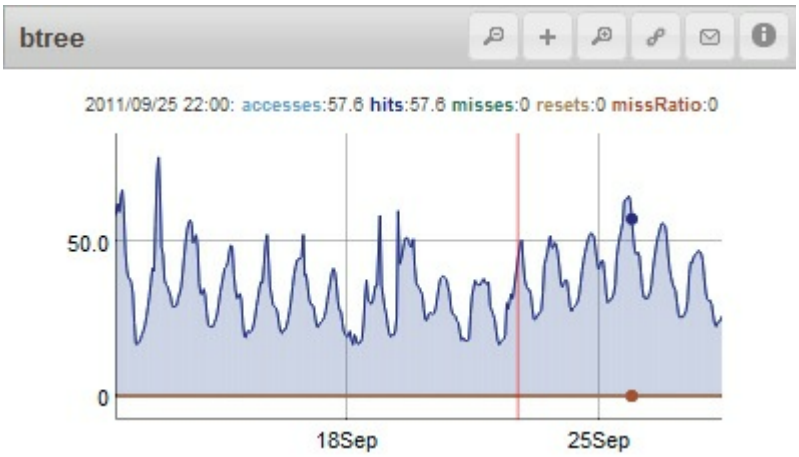
Online Collaboration

Company	Use Case
	Trello is a collaboration tool that organizes your projects into boards. In one glance, Trello tells you what's being worked on, who's working on what, and where something is in a process. Trello stores all historical and non-ephemeral data in MongoDB. The Trello team was attracted to MongoDB for its speed. MongoDB offers very fast writes, faster reads, and better denormalization support — it allows them to store the data of each individual card on the Trello board in a single document in the database and still have the ability to query into (and index) subfields of the document. They were also attracted to MongoDB's reliability: it is really easy to replicate, back up, and restore. Other benefits for the Trello team: using a loose document store makes it easy to run different versions of the Trello code against the same database without fooling around with DB schema migrations. This has a lot of benefits when they push a new version of Trello; there is seldom (if ever) a need to stop access to the app while we do a DB update or backfill. • What's in Trello Brett Keifer and the Trello team's presentation at the NYC MongoDB User Group, January 23, 2012. • The Trello Stack January 2012 from the Fog Creek Blog • Trello Architecture

Flowdock	Flowdock is a modern web-based team messenger, that helps your team to become more organized simply by chatting. Flowdock backend uses MongoDB to store all messages. • Why Flowdock migrated from Cassandra to MongoDB - Flowdock Blog (July 2010)
moxie Software™	Moxie Software™ is an innovative software company that provides integrated social enterprise software for employee and customer engagement through its Spaces™ by Moxie platform. Designed in the social media era and using MongoDB, Employee Spaces™ is an enterprise social software built for "The Way People Work." It enables employees to collaborate on projects, innovate by sharing and co-creating knowledge and accelerate innovation.
	Sherl.tv is service for sharing plain text terminal screencasts from a unix terminal that uses MongoDB as the main data storage for records.

Real-time stats/analytics

Company	Use Case
intuit	Intuit is one of the world's largest providers of software and services for small businesses and individuals. Intuit uses MongoDB to track user engagement and activity in real-time across its network of websites for small businesses. • Deriving deep customer insights using MongoDB - Presentation at MongoSV (December 2010)
BUDDY MEDIA Every brand needs a buddy™	The Buddy Media Platform gives brands the tools to manage their Facebook Pages and engage their Facebook fans. The second iteration of the Publisher tool on the Buddy Media Platform is powered by MongoDB. • Social Analytics on MongoDB (Video and Slides) - Presentation from February 2011 New York MongoDB User Group • The New Buddy Media Publisher: What You Need To Know - Buddy Media blog (November 2010)

bit.ly	bit.ly allows users to shorten, share, and track links. bit.ly uses MongoDB to store user history. For more information: • bit.ly user history, auto-sharded presentation at MongoNYC (May 2010)
	10gen is the initiator, contributor and continual sponsor of the MongoDB project. 10gen built MongoDB Monitoring Service (MMS), a scalable software as a service monitoring tool built using MongoDB. MMS displays data in charts that track performance, resource utilization, availability, and response times. 10gen built MMS to provide operational insight into thousands of MongoDB deployments, and uses it to better diagnose and resolve customer issues.
chartbeat	Chartbeat is a revolutionary real-time analytics service that enables people to understand emergent behaviour in real-time and exploit or mitigate it. Chartbeat stores all historical analytics data in MongoDB. • MongoDB & EC2: A love story? - NY MongoDB User Group (Oct 2011) • The Secret Weapons Behind Chartbeat - Kushal's coding blog (April 2010) • Kushal Dave's Presentation at MongoNYC (May 2010)

	Server Density is a server monitoring tool from Boxed Ice. They have used MongoDB since June 2009 and are now processing billions of documents every month. Server Density also includes an addon for MongoDB monitoring and they have written extensively about MongoDB itself. Blog posts: • Why we migrated from mysql to mongodb • Automating partitioning, sharding and failover with MongoDB • Notes from a production MongoDB deployment • Many more Presentations: • MongoDB Monitoring and Queueing - David Mytton's Presentation at MongoSF (May 2011) • MongoDB Monitoring and Queueing - London MUG (September 2010) • Humongous Data at Server Density: Approaching 1 Billion Documents in MongoDB - Webinar (November 2011) • Ensuring High Availability for Real-time Analytics featuring Boxed Ice / Server Density - Webinar (October 2012)
	Zuberance started using MongoDB as a reporting engine in their enterprise product. In Q3 of 2011, Zuberance decided to build their self serve product to scale it to thousands of brand marketers to have them identify, energize their advocates while giving them real-time analytics. • Why We Like MongoDB at Zuberance - Zuberance Blog (January 2012)
	ShareThis makes it easy to share ideas and get to the good stuff online. ShareThis is the world's largest sharing network reaching over 400 million users across 150,000 sites and 785,000 domains across the web • MongoDB is Powering ShareThis Count System - Lenin Gali's Presentation at MongoSV (December 2010)

	GitHub, the social coding site, is using MongoDB for an internal reporting application. • MongoDB for Analytics - John Nunemaker's presentation at MongoChicago 2012)
	Eventbrite gives you all the online tools you need to bring people together for an event and sell tickets. • Building a Social Graph with MongoDB at Eventbrite - Brian Zambrano's presentation at MongoSV (December 2010) • Tech Corner: Auto recovery with MongoDB replica sets - Eventbrite Blog (October 2010) • Why you should track page views with MongoDB - Eventbrite Blog (June 2010)
	BLiNQ Media, an employee owned social media advertising and technology company, is one of 12 companies globally with access to the Facebook advertising API and the only company that is building its technology, BAM (BLiNQ Ad Manager) in Ruby on Rails. BLiNQ works with some of the world's leading media agencies and brands, helping them place and optimize advertising on Facebook through our proprietary technology. The technology team is headquartered in Midtown Atlanta at the Advanced Technology Development Center (ATDC), one of the world's top 10 incubators. The company's sales and client services headquarters is in TechSpace in New York City's Union Square area. BAM utilizes MongoDb as an operational data store to support millions of Facebook user and advertising campaign data. The data stored in MongoDB is then used for real-time reporting, analysis and optimization against key performance indicators. Since BAM went live in July 2010, it is storing over 7 million records and averaging in excess of 30,000 transactions a day. • MongoDB Delivers Results for Facebook Advertisers - Presentation at Mongo Atlanta (February 2011)

	Yottaa offers Performance Analytics, a cloud service that monitors, ranks and analyzes the performance of millions of web sites, providing an open database to answer questions such as “why performance matters” and “how fast is my site?”. Yottaa is using Ruby on Rails and MongoDB to build its scalable analytics engine. • How Yottaa Uses MongoDB - Jared Rosoff's presentation at MongoBoston (September 2010) • Scalable Event Analytics with MongoDB and Ruby - Jared Rosoff's presentation at RubyConfChina (June 2010)
	BuzzFeed is a trends aggregator that uses a web crawler and human editors to find and link to popular stories around the web. BuzzFeed moved an analytics system tracking over 400 million monthly events from MySQL to MongoDB.
	uberVU is an intuitive social media management and analytics service used by companies of all sizes in 20+ countries. uberVU uses Mongo as its primary data store because of its fast writes, its schemaless approach and reliability. • Intelligent Stream-Filtering using MongoDB - Mihnea Giurgea's Presentation at MongoUK (September 2011)

	CopperEgg RevealCloud – Awesome Cloud Monitoring – CopperEgg is able to achieve its super real-time updates and alerts of the RevealCloud product with the help of MongoDB. Mongo was the only DB that fit the requirements of extremely rapid storage and retrieval of customer monitoring data to provide on-screen updates every few seconds (NOT minutes). CopperEgg monitors thousands of end points world-wide in real-time. RevealCloud is available to application development organizations to give the instant visibility of system performance in any stage of the development lifecycle, from initial design through production, helping ensure SLAs…and giving your customers the response times they expect – right now. • Designing Algorithms that Scale Horizontally with MongoDB - Luke Ehresman's Presentation at MongoDallas (November 2011)
	Loggly uses MongoDB as a repository of statistics and summary information for time series log data. Loggly collects and stores metrics into MongoDB including the size and count of log events, and the results of saved searches run against our customized fulltext search engine based on SOLR. These metrics are used to drive graphs, reports, dashboards and exciting visualizations for users.
	Pattern Builders built a .NET based streaming analytics engine using MongoDB. Relational databases and MSMQ were integrated with MongoDB for high performance and a great user experience.
	Go Graph Stock Photography offers over 7 million stock images and graphics. We use MongoDB to track impression data on our search results pages. It helps us to track which results are served when a user does a search query. Using that data we are able to get a better idea of what our customers are looking for, and we are able to serve up better results which will translate into a better overall user experience.
	VinTank uses MongoDB to power the analytics and reporting for their social media listening and social CRM features. The company is the only digital think tank for the wine industry.

TRUNK CLUB MEN'S OUTFITTERS	Trunk Club uses MongoDB for a variety of applications, including real-time analytics, business intelligence and machine learning.The company allows guys discover awesome clothes that are perfect for them without ever having to go shopping by combining top brands, expert service, and unparalleled convenience to deliver a highly personalized experience that helps guys look their best and saves them time. • [Lightweight Business Intelligence with MongoDB
http://www.10gen.com/presentations/mongodb-chicago-2012/lightweight-business-intelligence-mongodb]	• Cory Ehmke's presentation at MongoChicago 2012)
	MinFil.se uses MongoDB for distributed & redundant file storage capability. Using MongoDB, GridFS' service basically scales infinitely---they just add more servers as their storage requirement grows, Real-Time Analytics for all our users, all individual files, and for the service in general, tracking pageviews, downloads, unique downlods, referrers, etc
appboy	Appboy uses MongoDB for distributed & redundant file storage capability. Appboy is a customer relationship management, marketing, and user analytics platform for mobile applications. They use MongoDB as their primary datastore and analytics backend.
УКРАИНСКИЕ НАВИГАЦИОННЫЕ СИСТЕМЫ	Ukrainian Navigation Systems uses MongoDB for its main project database. The company manufactures advanced systems for GPS-monitoring.
ChannelMeter	ChannelMeter harvests hourly and daily analytics from YouTube for millions of videos and tens of thousands of channels and stores them using MongoDB. This information powers ChannelMeter's website, which provides reports, analytics, and recommendations for individual vloggers and corporate giants alike.

	GoPollGo uses MongoDB as their main data store. They've found it to be great for real-time analytics, which is a big component for them and their users, who include ABC News, ESPN, Netflix and Yahoo.
	MineWhat is an automatic shopping assistance platform for ecommerce. MongoDB powers their data collection, where they need high write speeds in the single digit milliseconds. Aggregation framework/MapReduce provide them with analytics without need of a complex architecture.
	irket Haberleri is the leading PR hub platform in Turkey. They use MongoDB to store high volumes of analytics data and present various types of realtime access data.

Scientific

- CERN uses MongoDB for Large Hadron Collider data.
  - MongoDB at the Energy Frontier - MongoNYC presentation (May 2012)
  - Holy Large Hadron Collider, Batman! - MongoDB Blog (June 2010)
- The Mount Sinai Institute for Genomics and Multiscale Biology uses MongoDB to help with various computational genomics tasks
- Realtime.springer.com is a service that aggregates together downloads of Springer journal articles and book chapters in real time and displays them in a variety of visualizations. The goal of this service is to provide the scientific community with valuable information about how the literature is being used "right now". MongoDB is used to store the details of one million downloads per day from across Springer's sites. Map reduce jobs generate collections for the last 7, 30, and 90 days for around 2,000 journal and 40,000 book titles.

Social Networks

Company	Use Case

	Foursquare is a location based social network that incorporates gaming elements. Foursquare uses MongoDB to store venues and user "check-ins" into venues, sharding the data over more than 25 machines on Amazon EC2. • Experiences Deploying MongoDB on AWS - Cooper Bethea's Presentation at MongoSV (December 2011) • Practical Data Storage: MongoDB at foursquare - MongoNYC (June 2011) • MongoDB @ foursquare - MongoSF (May 2011) • Scaling foursquare with MongoDB - New York MongoDB User Group (December 2010) • MongoDB Q&A - New York MongoDB User Group (December 2010) • MongoDB at foursquare presentation: Slides and Video (May 2010)
	musweet keeps track of what artists and bands publish on the social web. • Handling Humongous Data Sets from Social Net - Nader Cserny and Grischa Andreew's Presentation at MongoBerlin (October 2010)
	Behance uses MongoDB to power its Behance's Activity Feed, which displays what's new and what's buzzing in the Behance community from the people you follow. The Activity Feed provides a way of exploring the creative world through everyone you know as a curator in real time.
	Guildwork is a guild host and social network for massively online multiplayer games such as World of Warcraft. Guildwork stores nearly all data in MongoDB with the exception of sessions and chat history.

	Silentale keeps track of your contacts and conversations from multiple platforms and allows you to search and access them from anywhere. Silentale is using MongoDB as the back-end for indexing and searching on millions of stored messages of different types. More details on Silentale can be found in this TechCrunch article. • One Year with MongoDB presentation from MongoUK (June 2010): Slides and Video
	Squarespace is an innovative web publishing platform that consists of a fully hosted and managed GUI environment for creating and maintaining websites. Squarespace's new social modules utilize Mongo to store large amounts of social data that is pulled in from around the Internet and displayed in native widgets that are fully integrated with the platform.
	Shelby.tv Shelby uses a few physically distinct MongoDB replica sets as the primary data stores in development and production. That data is accessed by web and iOS through our internal Ruby API.
	Jorjun Technical Services uses MongoDB for caching json-based REST API calls, and increasingly for production data. The company offers solid, reliable software applications by offering focus, experience, lateral thinking and experienced evaluation of technical options.
	Tripl stores all checkin data, trips and notifications as well as queues in MongoDB. It is built from social geo-data on Facebook, Foursquare and Instagram to highlight trips and present travel stories from your friends.

	myList's Dev, Sandbox, and Production all use MongoDB for their backend. myList is a Facebook application that allows people to discover and organize the things in their lives. All back end data is in MongoDB with Solr for indexing and Hadoop for Map/Reduce jobs against logs.
	Bawaal Labs uses MongoDB for user management and storing all data related to user posts. Bawaal contains views on technology, social networking and current happenings.
	eZuce openUC features an advanced SIP Services Oriented Architecture (SSOA) which uses MongoDB to persistently store all user profile and transaction data in a distributed way. While MongoDB's stellar performance is critical to SSOA, its main benefit for SSOA is the horizontal sharding that's part of its architecture, which allows horizontal scaling across several nodes. MongoDB also holds all client registration data and makes it available to all session manager nodes. The effect to the user is completely seamless failover at a regional or global level. Many session manager nodes can participate in such a globally redundant cluster, which allows enterprises to easily build a robust communications backbone that spans the entire enterprise.
	BranchOut uses MongoDB to store profile statistics, back their feed data, endorsements and other key features of the application. BranchOut is a Facebook application designed for finding jobs, networking professionally, and recruiting employees.

	Parkuik uses MongoDB to store parking locations and perform geolocated queries.
	Meshfire uses MongoDB as the database tier for the Meshfire SaaS application.

Telecommunications

Company	Use Case
	Viber Media is using MongoDB as the core cloud infrastructure for its popular iPhone and Android applications that offer free phone calls and text messaging between users over 3G and wireless networks. Viber selected MongoDB for its ability to scale as more users join the Viber community and to manage peak and unpredictable data loads from its 30 million plus registered mobile users. • MongoDB at Viber Media: The Platform Enabling Free Phone Calls and Text Messaging for Over 18 Million Active Users - MyNoSQL (Jan 2012) • Viber Media Selects MongoDB as Database in the Cloud for Popular Mobile Apps - 10gen blog (Nov 2011) • Where NoSQL, cloud computing and free texting converge - GigaOm (Nov 2011)

	O2 uses MongoDB for several applications, including their Priority Moments location based offer service.

- Ericsson
- Telefonica

More MongoDB Users

Company	Use Case
	Sugar CRM uses Mo comments and whet Sugar.
	uses MongoDB to s uses MongoDB as c
	WuuhuuOnline.com SPHINX as full text s
	catch.com is the eas capture & find your r • MongoDB w 2011) • MongoDB a
	CollegeHumor is a c exchange applicatio
	Thrillist is using Mon Right now it's emplo our click-track data. continuous integratic
	Founded by Alibaba consumer-to-consun online retail stores th Sellers are able to p price or by auction. T sold at a fixed price; the monitoring data.
	Evite uses MongoDE • Tracking an 2010)

	Justin.tv is the easy, analytics tools for vi provide. Read more
	WHERE® is a local mobile coupons in th restaurant reviews, t from local merchants Where, Inc. uses Mc Ads™ - a hyper-loca
	PhoneTag is a servi and SMS. PhoneTag
	PiCloud enables scie power in public and computing applicatic sets in a highly distri
	Hashrocket is an ex Medication Manager resolving drug-relate
	The Mozilla open-so available on bitbucke
	Sedue is the enterpr currently uses Mong • MongoDB a
	Codaset is an open out what your friends • The awesor 2010) • Handling Da
	Punchbowl.com is a datamining. • Introducing presentatior • *Ryan Angilly* • MongoDB fo Presentatio
	photostre.am stream • MongoDB ir
	Fotopedia uses Mon timelines, a feature t tiny html fragment in • MongoDB:

	Grooveshark current
	Struq develops techn time.
	Pitchfork is using Mo
	Floxee, a web toolkit award-winning Twee
	TeachStreet helps p manage their teachin provide teachers with • Slides from
	Visibiz is a socially a productivity of busine tagged and organize information needed t activities through sin Visibiz uses MongoD • How Mongo
	Defensio is a comme
	TweetSaver is a web MongoDB for back-e
	Bloom Digital's AdG storage for AdGear.
	KLATU Networks de manage risk, reduce status, condition, an MongoDB to store te KLATU chose Mong
	This or That! is a use on comparisons. Wh user-generated com drive its leaderboard
	songkick lets you tra • Speeding up
	Crowdtap uses Mon engine. The targetin is core to its service.
	Detexify is a cool ap out the blog post for

	http://sluggy.com/ is • From MySQ 2010)
	EDITD is using Mon its sentiment analysi
	@trackmeet helps y
	eFlyover leverages t over two thousand g
	Shapado is a multi-t Rails and uses Mong
	Sifino enables stude summaries, and old
	GameChanger provi distribute real-time g • Document S • Tornado, M 2010) • GameChan (September
	soliMAP is a map-ba
	MyBankTracker iPh
	BillMonitor uses Mor used by the live site
	Tubricator allows yo Django.
	Mu.ly uses MongoDI service. MongoDB is
	Avinu is a Content M MongoDB.
	Topsy is a search er
	Cold Hard Code mal • Codepeek is • Jarvis uses

	Similaria.pl is an onl • One Year w
	ToTuTam uses Mon information about us • One Year w
	themoviedb.org is a
	OCW Search is a se uses Sphinx to inde • Full Text Se
	Mixero is the new ge noise. Mixero uses M
	Biggo is an advance collection.
	Kabisa is a web dev for many of its client
	DokDok makes it ea document - right fro Morency's presentat
	Enbil is a swedish w querying data about
	markitfor.me is a boo don't have to remem text of all of your boo pages.
	Backpage Pics is a v MongoDB is used to
	Joomla Ads uses Mo
	Eiwa System Manag projects since Janua

	Morango is an intern several client project • Building a C 2010)
	PeerPong discovers We index users acro available information
	ibibo ("I build, I bond represented as a sin million of these docu
	Zoofs is a new way t Twitter searching for
	Oodle is an online cl company behind the our millions of users
	Funadvice relaunche MongoDB User Foru
	Ya Sabe is using Mo search engine built f more than 14 million and in English.
	LoteriaFutbol.com is 2010. Mongo has be uses the PHP driver
	Kehalim switched ov contextual affiliate n MongoDB. MongoDI hadoop-like alternati
	Givemebeats.net is produced by some o profile, beats informa
	Cheméo, a search e explanation of the to chemeo.com/doc/tec
	Planetaki is place we MySQL for the stora • Planetaki P

	[ChinaVisual.com] is ChinaVisual.com mo major production an
	RowFeeder is an ea posts in a spreadshe across multiple socia • MongoDB fo (July 2010)
	Open Dining Networ web and mobile orde
	URLi.st is a small we (using the pylons fra
	Kidiso is a safe onlin setup, we are using performance (ie sea
	Carbon Calculated p everything in the wo of this platform, Carb intuitive.
	Vowch is a simple pl you. It is a platform f account. • View a vow
	Ros Spending is the 1,400,000 federal go more than 260,000 s information, stats an Development and la
	BlueSpark designs a development, we ha
	[Aghora] is a time at governmental requir information.
	Man of the House is and at home, as a fa

	PeerIndex is an algc firehose of social me
	sahibinden.com is a over 1.5 billion page caching.
	Ylastic is using Mon capability.
	BRAINREPUBLIC is like-minded people f
	Friendmaps is a tool
	The affiliate marketir search data. As of A
	Virb Looking for a pl with Virb. You provic
	Deal Machine is a st storage. It has helpe
	arrivalguides.com is launched a new site using the NoRM Driv
	The Hype Machine k and retrieval of user performance in our v
	ChatPast synchroniz computers. Search t about. Business use (Highrise, BaseCam

	Stockopedia initially 20000+ stocks, sect for building real time and investors condu
	TravelPost is a comr reviews, photos and
	SoulGoal stores or c
	Top Twitter Trends i technologies such as
	bongi.mobi is a plac include: fast docume on handset capabilit tracking, click-2-call,
	CoStore is an online importing, transform such as charts, grap wherever you need i which are MapRedu
	SnapDish is a photo social app for home and delicious. It is a taking beautiful food around the world. Pr make it social. Snap Download and enjoy
	Bakodo is a barcode while they are shopp information about th importantly, what the of products.
	noclouds.org is a on and share informatic all systems.
	CafeClimb.com is a which lets people sh traveling information from the user.
	Keekme is free mon Keekme you will eas MongoDB as a prima
	P2P Financial is Car DB for storing busine

	Qwerly is people sea links to social netwo
	phpMyEngine is a fr is MongoDB.
	vsChart allows you t
	yap.TV is the ultimat guide fused with a tu fans while watching MongoDB for analyti
	BusyConf makes gre letting them collect a multiple web platforr all the details preloa complementary to it' code is much simple • BusyConf p
	Sentimnt is a person networks and allows not related to you. S instances serve arou from MS SQL to Mo
	Workbreeze is fast a global project storag
	Kompasiana is the b biggest site In Indon
	Milaap works with es community developr
	Agent Storm is a cor and manage your Re listings on your web eFlyers all at the clic performs the search then looked up in My the results are serve updated either via th property is stored in changes. All this me • Now with 50

	Mashape is a frustra distribute an API of a • How We Sw Presentatio
	The UK Jobsite is ar We use MongoDB fo searches to user info of the reasons of wh http://www.theukjobs
	Tastebuds music da connect with their las immediately shown s popular events servi MongoDB has drama allows users to rapid
	Skimlinks enables pu equivalent affiliate lir products mentioned retailer programs in and ensures no mon impressions we get
	www.vanilladesk.com as the main databas MongoDB's docume what enables Vanilla
	Summify uses Mong (metadata and HTM only the most recent cache for URL redire
	dakwak is the easies visitors want. It is a r away from getting yo
	Kapost's online news produce content. We analytics for the new
	ContactMe is a "light many small business ContactMe.com aims social media sites lik place, making for ea
	Moontoast Impulse i from a Facebook far Their tools power co fan-building and reve Moontoast Impulse: and more. Moontoas
	Shopperhive is a soc reviews, video reviev data back-end storag

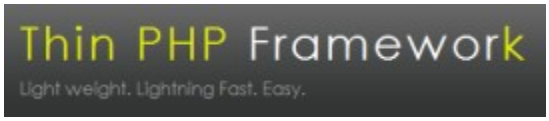
	tracknose develops reward you with disc chosen for excellent georeferenced statis
	Wusoup is a free onl handles all user and
	Fave is a local searc businesses, and to h releasing next-gener services. Infrastructu performance/respon call campaigns
	Doodle is the world's MongoDB as an adc less data via MySQL • An introduct
	FindTheBest is an o your options and dec informed decisions b products and service traffic monitoring and
	Fuseware provides r and more. This analy makers. We use Mo that we store. We ha scalability and data a
	Wherevent is a sear other cities. The enti feature.
	Skyline Innovations We deploy custom n MongoDB for data w • Time Series 2011)
	Memrise combines t of our learning and v about schema ossifi
	Dather is a content p can invite friends, fa

FUNDASTIC INFO	Fundastic.info is Cru investors and financ dedicated pages for understand the inves to store all its invest stored in MongoDB
foofind	Foofind Labs is a un and indexes every fi creation by Pablo Sc MongoDB is used to in order to allow text
BEACONPUSH BY SSN	Beaconpush is a clo real-time data with a methods for older br data is used for pres low system impact a
EASY BILL INSTANT	Easy Bill, India's first to the life of the com never-before conver biggest problems fac enterprises for bill pa grown exponentially, activity logging of ou
DeskMetrics	DeskMetrics is a rea how their users are u application, where a happened on the us We use MongoDB to have chosen Mongo important for us sinc
INTERSTATE	Interstate provides b tasks using awesom data, etc).
PROXLET	Proxlet helps you fig a Twitter API proxy, way of browser exte per-user setting docu designed to scale ho • A Beautiful 2011)
Dayload Hassle-free AWS Usage Monitoring	Dayload provides a resource usage and mail. Storing AWS re MapReduce functior
Goo.by	Avivo is creating mo mobile, web and cus Through design and interaction. Ultimatel that build brand loya is leading us to dive best for our clients. V primary database for

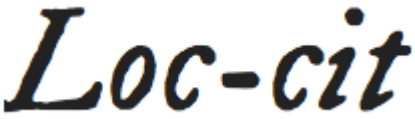
	Abusix is the networ spamfeeds(www.spa technology providers today, while handling behavior in their net major need for a solu increase customers resources in abuse d MongoDB in environ almost suits all need
	Idea2 is a versatile, create and implemer projects, customer c browser-based, fully application and work transition, we also cr
	FamilyTies is family families and friends children under 13, ba more to come. Mong various cache object relationship maps.
	Pracanowo.pl is a jo formats and finds job and in few month to
	Republika Online is Republika", a nation
	Grepler.com is a dis lists to enable single the user accounts w
	Thuisvergelijken.nl is wide range of produ all our data storage: MongoDB's flexibility
	salsadb.com uses M capabilities are used
	CHECK24 is one of compare a wide rang travel and thus save link-out the custome contract. The compa Furthermore, custom
	Wheelhouse CMS is flexible to develop si is used for all data s

	Qwk.io is the quicke backend.
	CyberAgent,Inc. has CyberAgent's major Investment Develop American office: Cyb Social Game with nc
	HeiaHeia.com provic friends, and cheer o can choose from ov HeiaHeia.com's tech provide engaging, cu
	NewsCurve is an an MongoDB as primar
	Accounting SaaS Ja accountants and sm services include, but necessary to meet th that is difficult to acc
	Fabric Structures Co in the overall constru customers about the field. MongoDB is us
	Brenden Digital uses migrated. We appre repeatedly exhibits a comprehension.
	Attachments.me is a surrounding attachm
	Thumbtack is an onl analytics, due to its details can be found
	ChirpAt.Me enables expertise with their f sessions to messag

	DianPing.com is a le objective and rich lo weddings and variou on businesses, user
	Venmo is a free and drinks, rent, grocerie activities published b
	Intelie is a software t and machine learnin centers in real time. documents per minu new rules, in additio
	First, we are using M users can create a S perform a lookup of user. Second, we ar statistics for our Ema queries in tables witl anywhere from 30 - queries. This use ca http://johnpwood.net
	Directdialogs.com is capability and a flexi powerful cache with purposes, OLAP rep MongoDB.
	DC Storm helps mar better. The intuitive t across all digital cha powers the Storm Pl redundancy.
	Bouncely uses Mong everything and store information and run
	PClicks uses Mongo then adapt to meet t
	Magnetic is the lead their most relevant a month. Beyond perfo leverage the rich dat and unify a mix of hi Magnetic a competit
	Okezone.com is usir and commenting sys
	OpenChime uses M Thousands of people instructors in Mexico

	Monoloop is a behav MongoDB is the core and deliver personal
	Yeay.me is a small s MongoDB is used fo
	Activesphere provide as well.
	SocialCityDeals use set of fields and new architecture to store effort in half for the s
	Thin PHP Framewor fast, simple, scalable users scale their dat
	Newsman App is an DB for now to build for each subscriber t all our other databas
	Sizzix uses mongod
	Gazaro uses Mongo products across a ra
	Dropzone.by uses m introduced OAuth 2.
	Cairenhuiis a Financ and use Mongodb_1
	Ez Texting uses Mor outgoing text messa
	iKeepm is a cloud-ba uploads in GridFS.

	Maansu.com is an re sells.
	www.motores24h.pt our data storage nee that we can easily ex
	Techunits seeks to v technology, Informat TechMVC is the bes architecture. Techun Buzzers.co.za, Trivia
	Buzzers.co.zahttp://l changing environme millions of products a using MongoLantern system more intellig
	Paraimpu is a social and socially share th used for all the persi sharding, heterogen
	Trivian is a location l to store quiz questio
	Business.com is the the products and ser taxonomy systems.
	General Flows is bui into some pre-baked build a booking syste plugging them into th structures, because code-generation tech (EAV) but found the Google App Engine and running; we're ju
	Art.sy is a new way t collections around th • Using Mong
	Spoondate is a socia experiences. The sit in a conversation, or data and member se rapidly and consister
	uQuery Inc. uses a s we receive from App software stack from

	46elks uses MongoD make sure the data
	Chalkboard is a daily socially over mobile of consumers across MongoDB to archive internally. With the g future.
	tisProperty is a Real realestates listed on to store our crawling
	Digital Media Excha more instead of our storage 2. We run a schema-less design. are just some of the
	Daily Gourmet provi our extensive loggin wasn't possible with
	Sponsored by Singly over their personal d as a personal data s
	Loc. cit. is a small ne can be explosive. Tr database for everyth Mongoengine for Dja
	Naver Japan utilizes recently released ph http://itunes.apple.co with MongoDB (Japa http://www.slideshar
	My eStore App uses content via Mongo.
	Fotosearch Stock Ph imagesare served w customer is looking f results.

	CloudAmp is a data many data sources a across different priva data integration for a platform from private Twitter, where we or
	Piyavate Internationa new website need so search with Solr. Mu WYSIWYG form crea these features are to way. That's why Mon
	Mindvalley is using M Mongoid. The ability speed, while its docu lead. Due to our suc developed CMS, as
	Cookfollower is a Tw meet new people. M
	Weconext develops secured web platforn allows simple collabo MongoDB stores for
	Visual.ly is building t platform for Big Data
	Trendrr provides the for media and brand curation, charting an
	Wireclub is an online are exchanged by ou
	KANO/APPS creates worldwide players ac the primary data stor
	OnePageCRM uses been using MySQL s

blabbermouth	Blabbermouth Socia for sweepstakes and
Персик	Persik.me uses Mon
AwkwardTurtle.com	Kungfuters LLC is us Users must sign in u submit stories of awl Each week, the story This site was former Stories, Users, Vote
pt PunchTab	PunchTab store all l
	Nexon Corp. is a ma
heyZap	Heyzap uses mongo iOS gaming apps, a tens of millions of ou
FISHIDY	Fishidy.com uses M
WOKEEY.com	Wokeey.com uses M hundreds records.

	ThinktankSocial use brands such as Price and mass storage w
	Data Publica uses M
	Hipster uses Mongo and analytics.
	Noktacom Medya us analytics. We track e custom video scorin suggestion system. second. We will exp usage of caching sy
	Searce we used mo very large set of data
	deegr is using Mong recommendation en architecture and sha
	AppHarbor is a .NET messages generate provision MongoDB
	Pictomist is a simple written in two basic style mongo collecti off the top elements second part is web l them in memory, an spin out the bulk of t layer without muckin
	Xperantum Xemanti structures, capture t
	4Apps uses MongoD memcached.
	GLO AB uses Mong (MES) software proc data gathered during but expecting to gro within company netv

	CMI Soft uses Mong & e-commerce store social networking ap use Rails with Mong
	Fyndlr implements M posts and users are functionality. Since c MongoDB instance.
	Exceptiontail uses M exception managem them see the entire s
	AdHui.com uses mo optimization.
	Travelmap uses Mor sources. Travelmap 5 times faster and m average our databas
	Global Advertisers u outdoor advertising a Airport. Global is one
	Melt DSP is using M
	Startup Threads Mor
	Songza uses Mongo streaming service th The schema-free de and occasionally a c

CONTACT 506°	Ciris is a new applic MongoDB to store th
socialbakers The Recipe for Social Marketing Success	Socialbakers Inc. Fo
neongrid	Neon Grid uses mor
EXMACHINA	Ex Machina develop involves tens or hun and as a result we h needed a storage sc cost-efficient regardl We never looked ba
	BellyBallot is a new MongoDB for it's pri their associated mea Rails) is also on Mor
AdMaster™ Digital Consulting	AdMaster uses Mon of over 40,000,000 f
GetVega	GetVega is a second memory relevant data from th list on GetVega whe GetVega.com uses I MySQL. MongoDB a
5Searches	5Searches uses Mo
BARBULL	barbull.co.uk uses M reliable and the abili calculations are a br

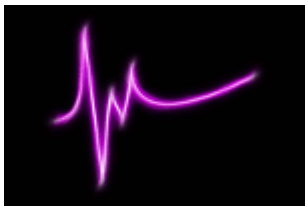
Power-lan	Power-lan uses Mon applications are : - N
Sidebuy	Sidebuy is a product find products and de properties. With Mor across our servers h everything in BSON
OviPets	OviPets uses Mongo very happy with the scaling beautifully wi storage and image c
	Tip or Skip uses Mo and job queues.
[nodegrid]	Nodegrid provides a service.
	Courseoff uses Mon
ClaimAble	ClaimAble operates use it exclusively to

	Ponder Design uses portal for a large stu implementing new fe wishes.
	Stamped is a recom favorite places and t coordinating the crea The Engineering Bel
	Hopper uses Mongo
	Recommendly INC u
	AlisverisRobotu.com including inventory n
	ITERNOVA SL prou store real-time data networks...), and veh
	tvb.com uses Mongo MongoDB is schema processing. We crea

	Ob1b.com is one of price and find produ products which they because of its simpli
	Talis uses MongoDE http://community.tali achieved an order of to simplify our stack
	Digital Flow uses Mo ease of use and dep
	Transmachin uses a sentences with loads delivering our websit
	About.me is currentl About.me turned to addition to the shard required.
	Dealer.com deploye supports configuratio on-disk size of almos

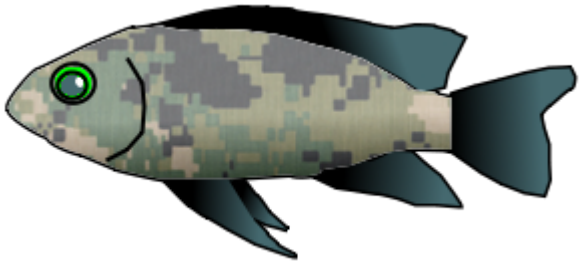
	Famigo stores all va
MATERIALS PROJECT	By scaling materials properties of over 8(Project is making the sophisticated web in schedule and track c and search the resul materials properties.
MapMyFITNESS	The MapMyFitness u traditional MySQL sc the candidate for exp application deploym for live tracking, to u
simple reach	SimpleReach power SimpleReach builds • Schemas fo • Polyglottany
stripe	MongoDB is used fo Stripe's in-house fra received, or a cron jo aggregate totals, fee

	Tuttur.com is a socia to share their betting Linux machines.
	WireLoad uses Mon narrowly focused so pleasant user experi
	Illicotravel compares and more. We love M
	MKN's (MKN Web S operations. The data
	Spideo using Mongo their own tastes.
	radi8.co.uk using Mo
	PRIJSZOEKEN.nl is products from 500 c data-feeds, our cust allows us to develop
	JustReco uses Mong
	Gimme Bar use Mor stored directly in S3 around 70g.

	Nodex use MongoDI data store for our CM and are expanding a fast and secure web outlining how we use
	FramtidsPost AB use services. The Future messages are big ar been able to use Mo
	Beanstalk Data uses
	wline uses MongoDE
	TalkOver uses Mong good for our "feeds" friendly content to us
	StoreHR uses Mong which keeps employ comes to due diliger and lets us scale hor
	Trillsy uses MongoD
	Bottomless Endless Online Gaming Engi

	Review19 uses Mon Gaming Engine Eng
	iQmetrix uses Mong customer facing, sol Microsoft's cloud pla
	Doocuments Softwa scalable sharded rep store and process al insight on readers be
	JackThreads, an ecc track products down
	Shelf9 combines the MongoDB as storag
	Luminis Technologie platform focussed or schools in the Nethe replica set running o designed to work we extra load of new sc
	Betterez is a SaaS r to improve both pass configuration spanni both the main data s with it's natural interf
	DUVRI 81.08 is a go safety risks to preve law (d.lgs 81/2008). replica sets, as docu
	Zomby is a massivel Creative Group. We user-base expands g Erlang) will allow us our game, we make efficient.
	Gusanito.com uses

	Aion Innovations use
	Support.com, Inc pro MongoDB to store m computers, devices outcomes of home te may lead to subscrib
	Footballrrating uses network, match vote pushing new feature
	Factile is a online fre survey is different in run analytics on the
	Dir uses MongoDB t Dir is a simple busin
	Socialbakers uses M for to get sophisticat
	YMIRLINK is a Toky marketing. In March mail system. In July MongoDB as our ne

	jScore lets recruiters users score candida through the Job Des datastore and also f keep their environm waiting.
	Mailgun uses Mongo stats, logs, DNS cac queries per second u APIs that allow you t
	The National Regist all data regarding th database. Others ca
	4thex Solutions offe persist users and pa
	PrivyTV uses Mongo for their products.Pri name can be shared internet-enabled tele

	Kookojais a social n MongoDB as its prin
	J-Media uses Mongo

See also

- MongoDB Apps
- Use Cases
- User Feedbackhealth

MongoDB Ecosystem

- Admin UIs
- Http Interface
- mongoperf
- MongoDB Monitoring Service
- Munin configuration examples
- Windows
- Wireshark Support for MongoDB Protocol

Admin UIs

MongoDB does not include a GUI-style administrative interface. Instead most administration is done from command line tools such as the mongo shell. However some UI's are available as separate community projects and are listed below. Some are focused on administration, while some focus on data viewing.

See also:

- The Mongo Monitoring Service (MMS) from 10gen.
- Tim Gourley's blog has a good summary of the tools.
- The built-in replica set admin UI page.

Tools

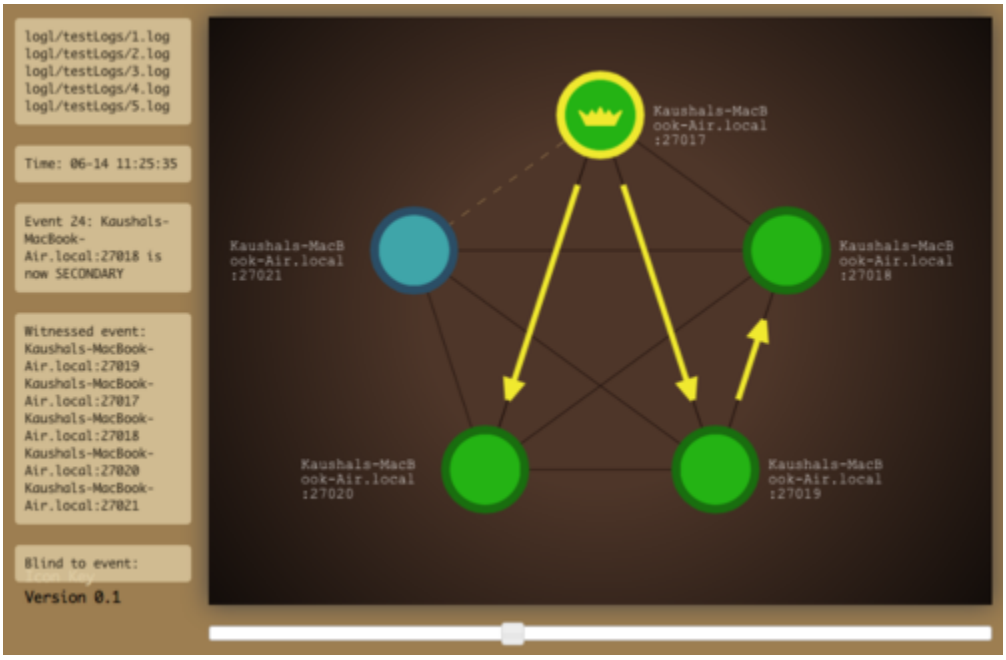
- Tools
    - Edda
    - Fang of Mongo
    - UMongo (formerly JMongoBrowser)
    - MongoExplorer
    - MongoHub
    - MongoVision
    - MongoVUE
    - mViewer
    - Opricot
    - PHPMoAdmin
    - RockMongo
    - Genghis
    - Meclipse
    - Humongous
    - MongoDB ODA plugin for BIRT
    - Monad Management for MongoDB
- Commercial
    - Database Master
- Data Viewers
    - mongs

Edda

Edda is a log visualizer. It takes logs as input and creates a timeline of notable events in the set. It can be installed via pip:

```
$ pip install edda
```

Screen shot:



See also: Github source

Fang of Mongo

- http://blueone.pl:8001/fangofmongo/
- http://github.com/Fiedzia/Fang-of-Mongo

A web-based user interface for MongoDB build with django and jquery.

It will allow you to explore content of mongodb with simple but (hopefully) pleasant user interface.

Features:

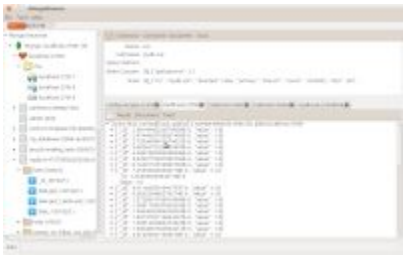
- field name autocompletion in query builder
- data loading indicator
- human friendly collection stats
- disabling collection windows when there is no collection selected
- twitter stream plugin
- many more minor usability fixes
- works well on recent chrome and firefox

To track progress on twitter: @fangofmongo

UMongo (formerly JMongoBrowser)

- home page: http://edgytech.com/umongo/
- github: http://github.com/agirbal/umongo
- download: https://github.com/agirbal/umongo/downloads

UMongo is a GUI app that can browse and administer a MongoDB cluster. It is available for Linux, Windows and Mac OSX.



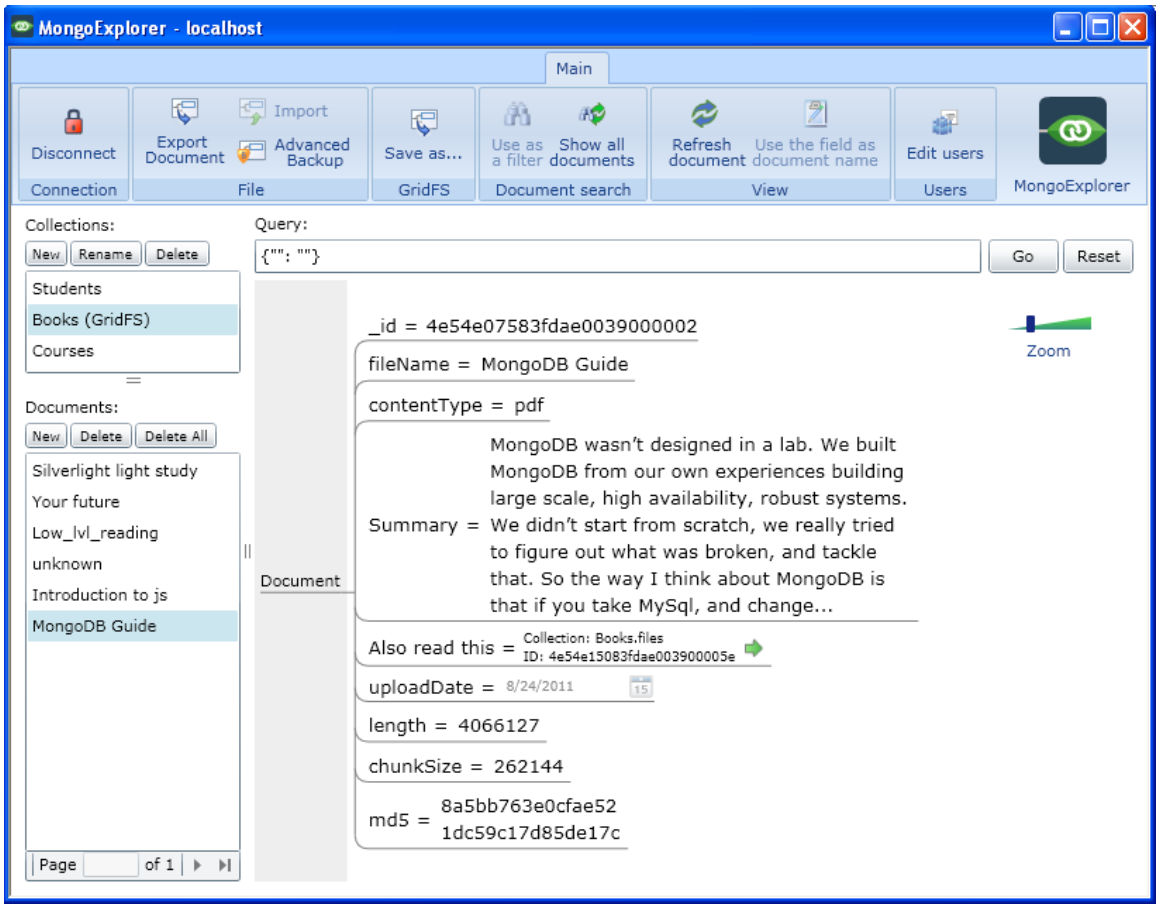
MongoExplorer

- http://mongoexplorer.com/

MongoExplorer is a MongoDB management tool, written in Silverlight (.net – works in windows/osx/?linux?).

Features:

- Easy to use
- Shows all the collections and documents of the database
- Uses a convenient tree view for documents
- Drag'n'drop is fully supported
- Document in-place editing



MongoHub

- http://mongohub.todayclose.com/

MongoHub is a native OS X GUI.

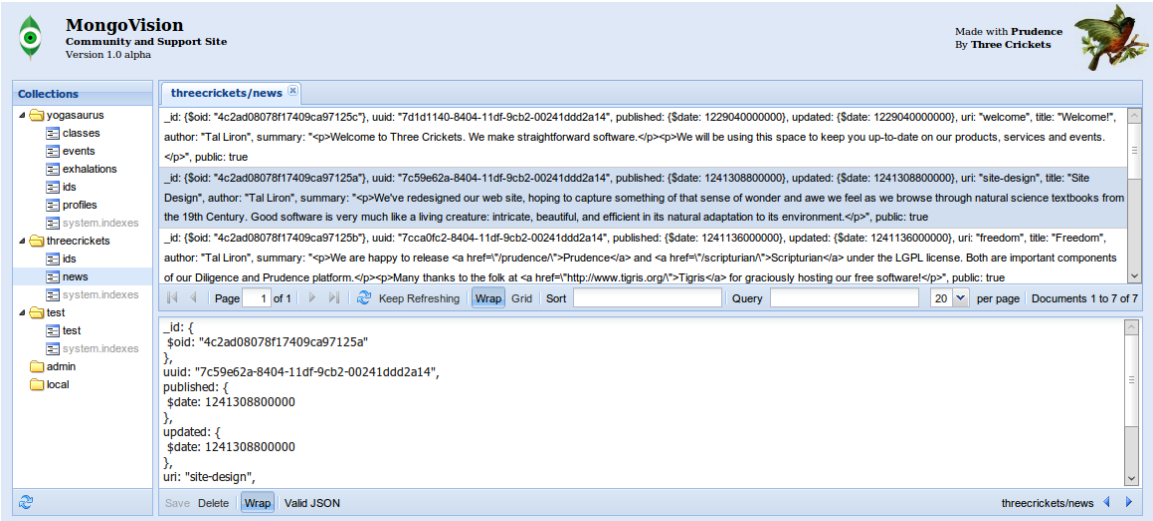
MongoVision

- http://code.google.com/p/mongo-vision/

MongoVision is a MongoDB management tool, written for Prudence.

Features:

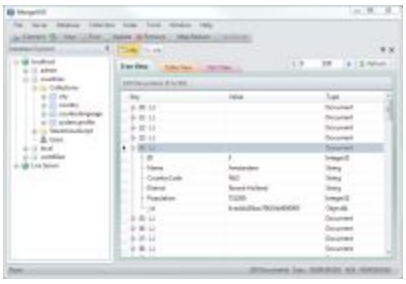
- Extended JSON support
- Tabular view
- Click to sort
- Filter boxes to alter query
- Auto-refresh



MongoVUE

- http://blog.mongovue.com

MongoVUE is a .NET GUI for MongoDB.



mViewer

- http://imaginea.com/mviewer

mViewer is a web-based MongoDB administration tool.

Opricot

- http://www.icmfinland.fi/oss/opricot/

Opricot is a hybrid GUI/CLI/Scripting web frontend implemented in PHP to manage your MongoDB servers and databases. Use as a point-and-click adventure for basic tasks, utilize scripting for automated processing or repetitive things.

Opricot combines the following components to create a fully featured administration tool:

- An interactive console that allows you to either work with the database through the UI, or by using custom Javascript.
- A set of simple commands that wrap the Javascript driver, and provide an easy way to complete the most common tasks.
- Javascript driver for Mongo that works on the browser and talks with the AJAX interface.
- Simple server-side AJAX interface for communicating with the MongoDB server (currently available for PHP).

PHPMoAdmin

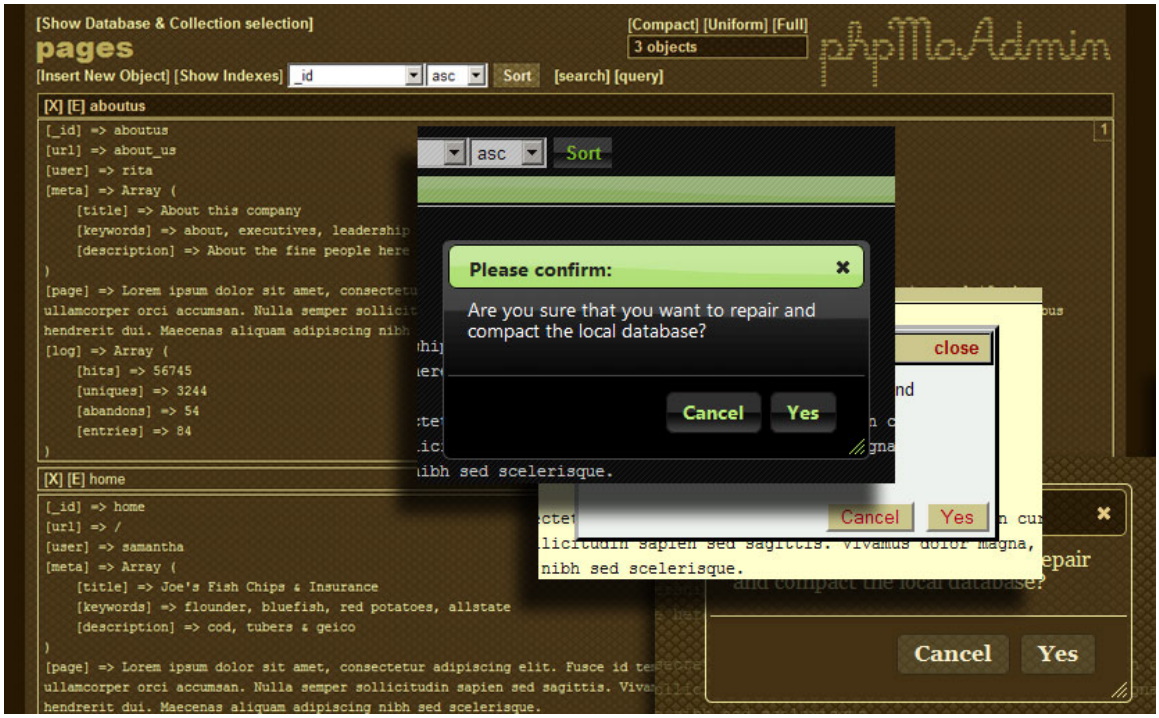
- http://www.phpmoadmin.com/

PHPMoAdmin is a MongoDB administration tool for PHP built on a stripped-down version of the Vork high-performance framework.

- Nothing to configure - place the moadmin.php file anywhere on your web site and it just works!
- Fast AJAX-based XHTML 1.1 interface operates consistently in every browser!
- Self-contained in a single 95kb file!
- Works on any version of PHP5 with the MongoDB NoSQL database installed & running.
- Super flexible - search for exact-text, text with * wildcards, regex or JSON (with Mongo-operators enabled)
- Option to enable password-protection for one or more users; to activate protection, just add the username-password(s) to the array at the top of the file.
- E_STRICT PHP code is formatted to the Zend Framework coding standards + fully-documented in the phpDocumentor DocBlock standard.
- Textareas can be resized by dragging/stretching the lower-right corner.
- Free & open-source! Release under the GPLv3 FOSS license!
- Option to query MongoDB using JSON or PHP-array syntax
- Multiple design themes to choose from
- Instructional error messages - phpMoAdmin can be used as a PHP-Mongo connection debugging tool

PHPMoAdmin can help you discover the source of connection issues between PHP and Mongo. Download phpMoAdmin, place the moadmin.php file in your web site document directory and navigate to it in a browser. One of two things will happen:

- You will see an error message explaining why PHP and Mongo cannot connect and what you need to do to fix it
- You will see a bunch of Mongo-related options, including a selection of databases (by default, the "admin" and "local" databases always exist) - if this is the case your installation was successful and your problem is within the PHP code that you are using to access MongoDB, troubleshoot that from the Mongo docs on php.net



RockMongo

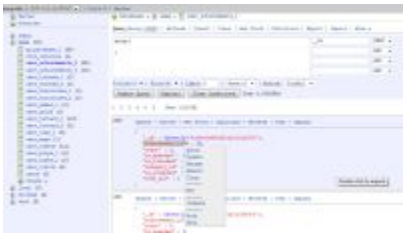
- http://code.google.com/p/rock-php/wiki/rock_mongo

RockMongo is a MongoDB management tool, written in PHP 5.

Main features:

- easy to install, and open source
- multiple hosts, and multiple administrators for one host
- password protection
- query dbs
- advanced collection query tool
- read, insert, update, duplicate and remove single row
- query, create and drop indexes
- clear collection

- remove and change (only work in higher php_mongo version) criteria matched rows
- view collection statistics



Genghis

A single-file MongoDB admin app, which is available as either a Ruby or PHP script: http://genghisapp.com/

Meclipse

Eclipse plugin for mongodb: http://update.exoanalytic.com/org.mongodb.meclipse/

Humongous

A standalone MongoDB browser built in Ruby: http://github.bagwanpankaj.com/humongous/

MongoDB ODA plugin for BIRT

The MongoDB ODA plugin for BIRT is an Eclipse based plugin which enables you to connect to a Mongo database and pull out data to display in your BIRT report. The interface is simple and an extensive user guide is also provided with the release.

http://code.google.com/a/eclipselabs.org/p/mongodb-oda-birt-plugin/

Monad Management for MongoDB

An operations and administration management tool for MongoDB, with dashboards, alerting, and monitoring graphs.

Visit their website for a demo.

Commercial

Database Master

Database Master from Nucleon Software.

Seems to be written in .net for windows (windows installer).

Features:

- Tree view for dbs and collections

- Create/Drop indexes
- Server/DB stats
- Support RDMBS (MySQL, postgres, ...)

Data Viewers

mongs

- http://www.whit537.org/mongs/

Http Interface

- REST Interfaces
  - DrowsyDromedary (Ruby)
  - MongoDB Rest (Node.js)
  - Mongodb Java REST server
- HTTP Interfaces
  - Sleepy Mongoose (Python)
- HTTP Console
  - HTTP Console Security
- Simple REST Interface
  - JSON in the simple REST interface
- Replica Set Admin UI
- See Also

REST Interfaces

DrowsyDromedary (Ruby)

DrowsyDromedary is a REST layer for Mongo based on Ruby.

MongoDB Rest (Node.js)

MongoDB Rest is an **alpha** REST interface to MongoDB, which uses the MongoDB Node Native driver.

Mongodb Java REST server

Mongodb Java REST server based on Jetty.

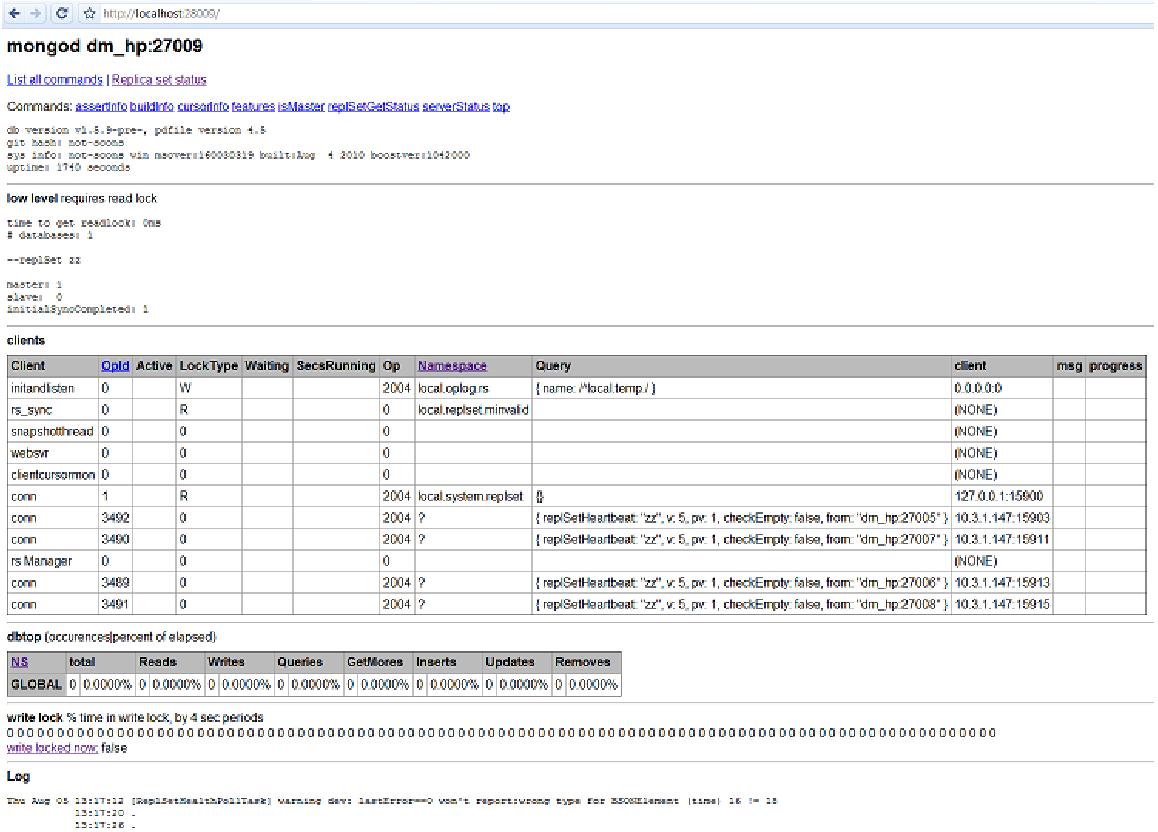
HTTP Interfaces

Sleepy Mongoose (Python)

Sleepy Mongoose is a full featured HTTP interface for MongoDB.

HTTP Console

MongoDB provides a simple http interface listing information of interest to administrators. This interface may be accessed at the port with numeric value 1000 more than the configured mongod port; the default port for the http interface is 28017. To access the http interface an administrator may, for example, point a browser to http://localhost:28017 if mongod is running with the default port on the local machine.



Here is a description of the informational elements of the http interface:

element	description
db version	database version information
git hash	database version developer tag
sys info	mongod compilation environment
dblocked	indicates whether the primary mongod mutex is held
uptime	time since this mongod instance was started
assertions	any software assertions that have been raised by this mongod instance
replInfo	information about replication configuration
currentOp	most recent client request
# databases	number of databases that have been accessed by this mongod instance
curclient	last database accessed by this mongod instance
Cursors	describes outstanding client cursors
master	whether this mongod instance has been designated a master
slave	whether this mongod instance has been designated a slave
initialSyncCompleted	whether this slave or repl pair node has completed an initial clone of the mongod instance it is replicating
DBTOP	Displays the total time the mongod instance has devoted to each listed collection, as well as the percentage of available time devoted to each listed collection recently and the number of reads, writes, and total calls made recently
dt	Timing information about the primary mongod mutex

HTTP Console Security

If security is configured for a mongod instance, authentication is required for a client to access the http interface from another machine.

Simple REST Interface

The mongod process includes a simple REST interface, with no support for insert/update/remove operations, as a convenience – it is generally used for monitoring/alerting scripts or administrative tasks. For full REST capabilities we recommend using an external tool such as Sleepy.Mongoose.

v1.4+: This interface is disabled by default. Use `--rest` on the command line to enable.

To get the contents of a collection (note the trailing slash):

```
http://127.0.0.1:28017/databaseName/collectionName/
```

To add a limit:

```
http://127.0.0.1:28017/databaseName/collectionName/?limit=-10
```

To skip:

```
http://127.0.0.1:28017/databaseName/collectionName/?skip=5
```

To query for {a : 1}:

```
http://127.0.0.1:28017/databaseName/collectionName/?filter_a=1
```

Separate conditions with an &:

```
http://127.0.0.1:28017/databaseName/collectionName/?filter_a=1&limit=-10
```

Same as `db.$cmd.findOne({listDatabase:1})` on the "admin" database in the shell:

```
http://localhost:28017/admin/$cmd/?filter_listDatabases=1&limit=1
```

To count documents in a collection:

```
http://host:port/db/$cmd/?filter_count=collection&limit=1
```

JSON in the simple REST interface

The simple ReST interface uses strict JSON (as opposed to the shell, which uses Dates, regular expressions, etc.). To display non-JSON types, the web interface wraps them in objects and uses the key for the type. For example:

```
# ObjectIds just become strings
"_id" : "4a8acf6e7fbadc242de5b4f3"

# dates
"date" : { "$date" : 1250609897802 }

# regular expressions
"match" : { "$regex" : "foo", "$options" : "ig" }
```

The code type has not been implemented yet and causes the DB to crash if you try to display it in the browser.

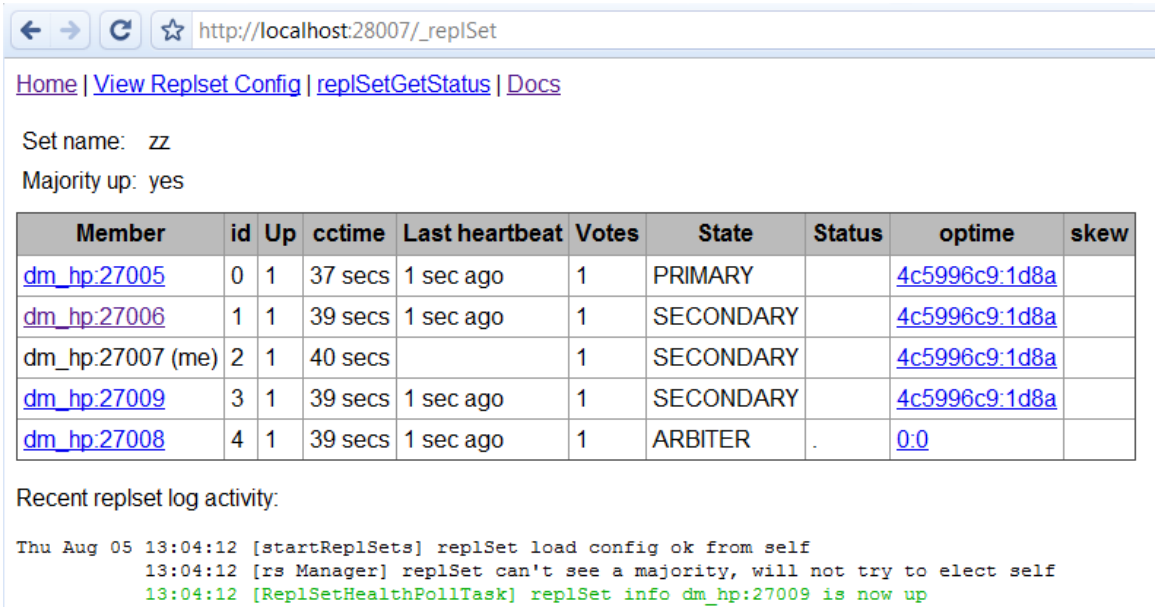
See Mongo Extended JSON for details.

Replica Set Admin UI

The `mongod` process includes a simple administrative UI for checking the status of a replica set.

To use, first enable `--rest` from the `mongod` command line. The rest port is the db port plus 1000 (thus, the default is 28017). Be sure this port is secure before enabling this.

Then you can navigate to `http://<hostname>:28017/` in your web browser. Once there, click Replica Set Status (/_replSet) to move to the Replica Set Status page.



See Also

- Diagnostic Tools

mongoperf

- Prebuilt Binaries
- Building
- Running
- mmf:false mode (direct i/o test)
- mmf:true mode
- Additional Options
- See Also

mongoperf is a utility for checking disk i/o performance of a server independent of MongoDB. It performs simple timed random disk i/o's. The utility is new and will likely be more sophisticated in the future.

mongoperf can be used to check disk subsystem performance even when MongoDB will not be used at all: its `mmf:false` mode is completely generic. In that mode is it somewhat analogous to tools such as bonnie++ (albeit mongoperf is simpler).

Prebuilt Binaries

- Win32 24Jul2012 mongoperf.exe

Building

Run `scons mongoperf` or `scons mongoperf.exe` to build. Note: mongoperf is new, so will need a recent version of the mongodb source code.

Running

```
# show help:
mongoperf -h

# example invocation:
#  test physical (direct) random read io's, using 16 concurrent reader threads,
#  and a 1GB test file.
echo "{nThreads:16,fileSizeMB:1000,r:true}" | ./mongoperf | tee out

# we might do this while it runs:
iostat -xm 2
```

mmf:false mode (direct i/o test)

In this default mode, mongoperf performs random 4KB direct (physical) disk i/o's (i.e., O_DIRECT is used on Linux). Thus this is a physical disk i/o test.

- Output form an example run

mmf:true mode

If `mmf:true` is specified as an option, tests are performed using memory-mapped files. These files are opened in a "normal" fashion and thus *caching is allowed*. This sometimes can be used to test file system cache behavior with memory mapped files.

Additional Options

```
{
    nThreads:<n>,     // number of threads (default 1)
fileSizeMB:<n>,   // test file size (default 1MB)
sleepMicros:<n>,  // pause for sleepMicros/nThreads between each operation (default 0)
mmf:<bool>,       // if true do i/o's via memory mapped files (default false)
r:<bool>,         // do reads (default false)
w:<bool>,         // do writes (default false)
syncDelay:<n>     // secs between fsyncs, like --syncdelay in mongod. (default 0/never)
}
```

- **nThreads** Number of test threads. You will need several threads to test a disk subsystem to saturation. For example, try `nThreads:16`.
- **sleepMicros** Pause for sleepMicros/nThreads between each operation.
- **r** Do read operations while testing.
- **w** Do write options while testing. w and r may be used at the same time.
- **syncDelay** For use with `mmf:true` mode only. The `syncDelay:secs` option instructs mongoperf to perform an asynchronous fsync of the test mmap file at the specified interval. mongod does something similar every 60 seconds, thus this can be useful to test basic system behavior in a simpler setting. This option is applicable only when using `mmf:true` mode. Currently in `mongoperf` this defaults to zero, which means off.
- **fileSizeMB**. Specifies the size in megabytes of the test data file. The file will be placed in the current directory. **Specify a very large test file size to create a realistic simulation.** A very small file (1MB) could be cached by your disk controller completely. A file that is say, 100MB in size would involve a small number of disk cylinders, and adjacent track-to-track seeks are much faster than a drive's average seek time. The file size is particularly important when `mmf:true` is specified, as the file system cache is then involved. A file size much larger than RAM will result in much different performance results than a file size smaller than RAM.

See Also

- The utility is quite simple, so you might wish to also take a look at the source code.
- iostat

MongoDB Monitoring Service

MongoDB Monitoring Service is a free SaaS solution for proactive monitoring of your MongoDB cluster(s). MMS's web interface features charts, custom dashboards, and automated alerting; and since it runs in the cloud, MMS requires minimal setup and configuration. Within minutes your devops and systems administration teams can manage and optimize your MongoDB deployment, and derive valuable insights from key operational metrics.

Getting Started

MMS is free and available to everyone in the MongoDB community. To get started with MMS now, visit the MMS setup page.

- Create an account at mms.10gen.com.
- Download and install the MMS agent on your cluster.
- Within minutes, data will be visible on mms.10gen.com.

Docs

Documentation for MMS is available at:

- https://mms.10gen.com/help/

Blog Posts

- Announcing the MongoDB Monitoring Service - 10gen Blog (Sept 2011)
- Getting Started with MMS - Kristina Chodorow's Blog (Sept 2011)
- Getting Started with Mongo Monitoring Service (MMS) on OpenShift - OpenShift Blog (Dec 2011)
- Monitoring mongoDB with MMS (from 10gen) - Dec 2011

Munin configuration examples

Overview

Munin can use be used for monitoring aspects of a running system. The following is a mini tutorial to help you set up and use the MongoDB plugin with munin.

Setup

Munin is made up of two components

- agent and plugins that are installed on the system you want to monitor
- server which polls the agent(s) and creates the basic web pages and graphs to visualize the data

Install

You can download from SourceForge , but prebuilt packages are also available. For example on Ubuntu you can do the following:

Agent install

To install the agent, repeat the following steps on each node you want to monitor.

```
shell> sudo apt-get install munin-node
```

Server install

The server needs to be installed once. It relies on apache2, so you will need to ensure that it is installed as well.

```
shell> apt-get install apache2
shell> apt-get install munin
```

Configuration

Both the agent(s) and server need to be configured with the IP address and port to contact each other. In the following examples we will use these nodes:

- db1 : 10.202.210.175
- db2 : 10.203.22.38
- munin-server : 10.194.102.70

Agent configuration

On each node, add an entry as follows into
for db1:

```
/etc/munin/munin-node.conf
host_name db1-ec2-174-129-52-161.compute-1.amazonaws.com
allow ^10\.194\.102\.70$
```

for db2:

```
/etc/munin/munin-node.conf
host_name db2-ec2-174-129-52-161.compute-1.amazonaws.com
allow ^10\.194\.102\.70$
```

* host_name : can be whatever you like, this name will be used by the server

- allow : this is the IP address of the server, enabling the server to poll the agent

Server configuration

Add an entry for each node that is being monitored as follows in

```
[db1-ec2-174-129-52-161.compute-1.amazonaws.com]
address 10.202.210.175
use_node_name no

[db2-ec2-184-72-191-169.compute-1.amazonaws.com]
address 10.203.22.38
use_node_name no
```

* the name in between the [] needs to match the name set in the agents munin-node.conf

- address : IP address of the node where the agent is running
- use_node_name : dtermine if the IP or the name between [] is used to contact the agent

MongoDB munin plugin

A plugin is available that provide metrics for

- B-Tree stats
- Current connections
- Memory usage
- Database operations (inserts, updates, queries etc.)

The plugin can be installed as follows on each node where MongoDB is running

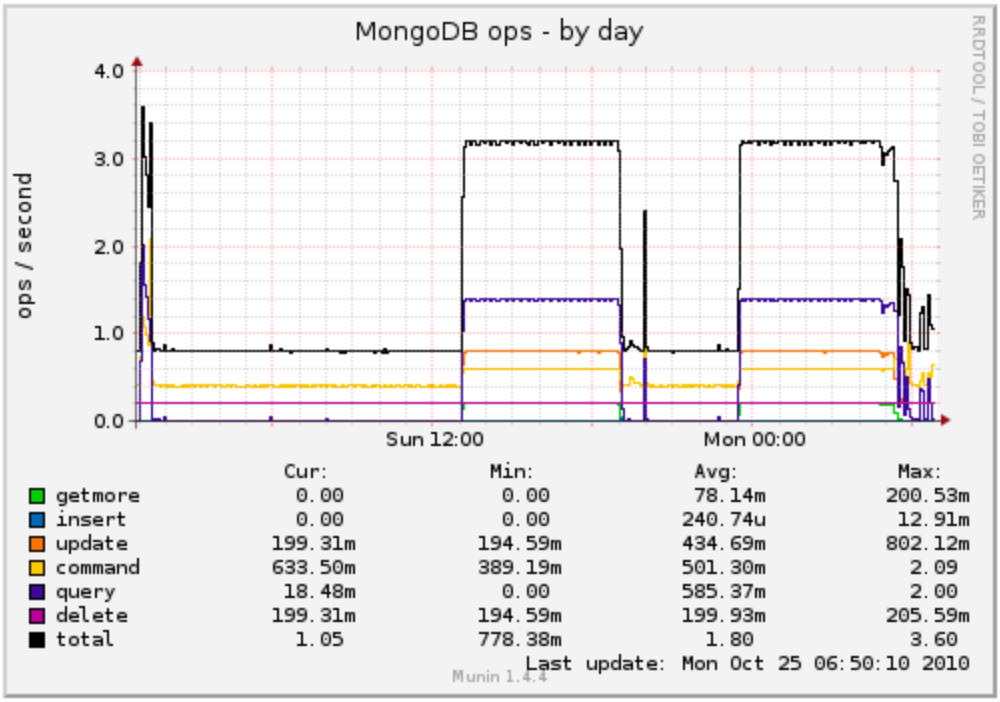
```
shell> wget http://github.com/erh/mongo-munin/tarball/master
shell> tar xvf erh-mongo-munin-*tar.gz
shell> cp erh-mongo-munin-*/mongo_* /etc/munin/plugins/
```

Check your setup

After installing the plugin and making the configuration changes, force the server to update the information to check your setup is correct using the following

```
shell> sudo -u munin /usr/share/munin/munin-update
```

If everything is set up correctly, you will get a chart like this



Advanced charting

If you are running a large MongoDB cluster, you may want to aggregate the values (e.g. inserts per second) across all the nodes in the cluster. Munin provides a simple way to aggregate.

```
/etc/munin/munin.conf
[compute-1.amazonaws.com;CLUSTER]
update no
```

* Defines a new segment called CLUSTER

- update no : munin can generate the chart based on existing data, this tell munin not to poll the agents for the data

Now lets define a chart to aggregate the inserts, updates and deletefor the cluster

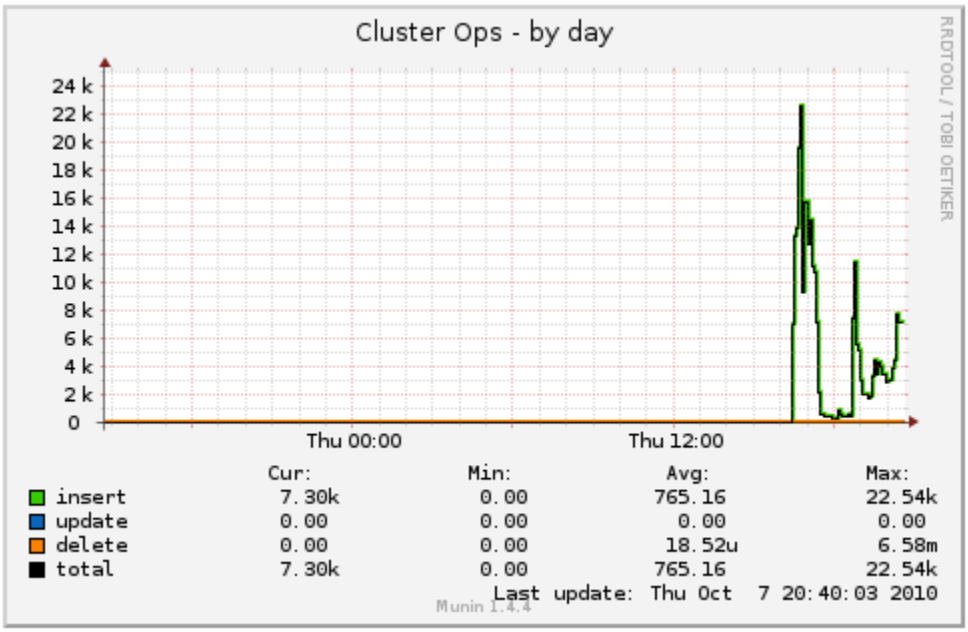
```
cluster_ops.graph_title Cluster Ops
cluster_ops.graph_category mongodb
cluster_ops.graph_total total
cluster_ops.total.graph no
cluster_ops.graph_order insert update delete
cluster_ops.insert.label insert
cluster_ops.insert.sum \
  db1-ec2-174-129-52-161.compute-1.amazonaws.com:mongo_ops.insert \
  db2-ec2-184-72-191-169.compute-1.amazonaws.com:mongo_ops.insert
cluster_ops.update.label update
cluster_ops.update.sum \
  db1-ec2-174-129-52-161.compute-1.amazonaws.com:mongo_ops.update \
  db2-ec2-184-72-191-169.compute-1.amazonaws.com:mongo_ops.update
cluster_ops.delete.label delete
cluster_ops.delete.sum \
  db1-ec2-174-129-52-161.compute-1.amazonaws.com:mongo_ops.delete \
  db2-ec2-184-72-191-169.compute-1.amazonaws.com:mongo_ops.delete
```

* cluster_ops : name of this chart

- cluster_ops.graph_category mongodb : puts this chart into the "mongodb" category. Allows you to collect similar charts on a single page
- cluster_ops.graph_order insert update delete : indicates the order of the line son the key for the chart
- cluster_ops.insert : represents a single line on the chart, in this case the "insert"
- cluster_ops.insert.sum : indicates the values are summed

- db1-ec2-174-129-52-161.compute-1.amazonaws.com : indicates the node to aggregate
- mongo_ops.insert : indicates the chart (mongo_ops) and the counter (insert) to aggregate

And this is what it looks like



Windows

Windows Quick Links and Reference Center

Running MongoDB on Windows

See the Quickstart page for info on how to install and run the database for the first time.

Running as a Service

See the Windows Service page.

The MongoDB Server

Get pre-built binaries on the Downloads page. Binaries are available for both 32 bit and 64 bit Windows. MongoDB uses memory-mapped files for data storage, so for servers managing more than 2GB of data you will definitely need the 64 bit version (and a 64 bit version of Windows).

The **Windows 2008+** MongoDB build uses newer features of Windows to enhance performance. Use this build if you are running with 64-bit Windows Server 2008 R2, Windows 7, or greater.

Writing Apps

You can write apps in almost any programming language – see the Drivers page. In particular C#, .NET, PHP, C and C++ work just fine.

- CSharp Language Center
- CSharp Community Projects

Building

We recommend using the pre-built binaries, but Mongo builds fine with Visual Studio 2008 and 2010. See the Building for Windows page.

Versions of Windows

We have successfully run MongoDB (mongod etc.) on:

- Windows Server 2008 R2 64 bit
- Windows 7 (32 bit and 64 bit)
- Windows Vista

- Windows XP

> ⚠ As of MongoDB Server 2.1.2, Windows XP is no longer supported as a server environment.

Windows Azure

Instructions for running MongoDB on Azure on Azure Worker Role (alpha)
MongoDB Installer for Windows Azure VM

AppHarbor

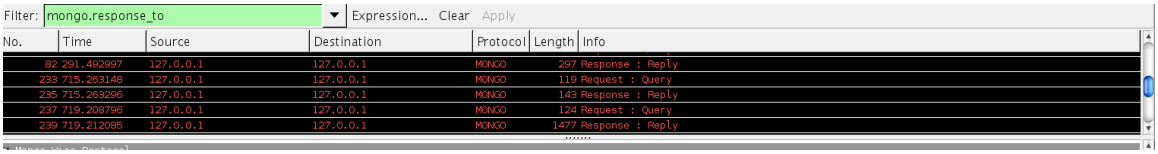
Instructions for running MongoDB on AppHarbor
Sample ASP.NET MVC app that uses MongoDB from AppHarbor

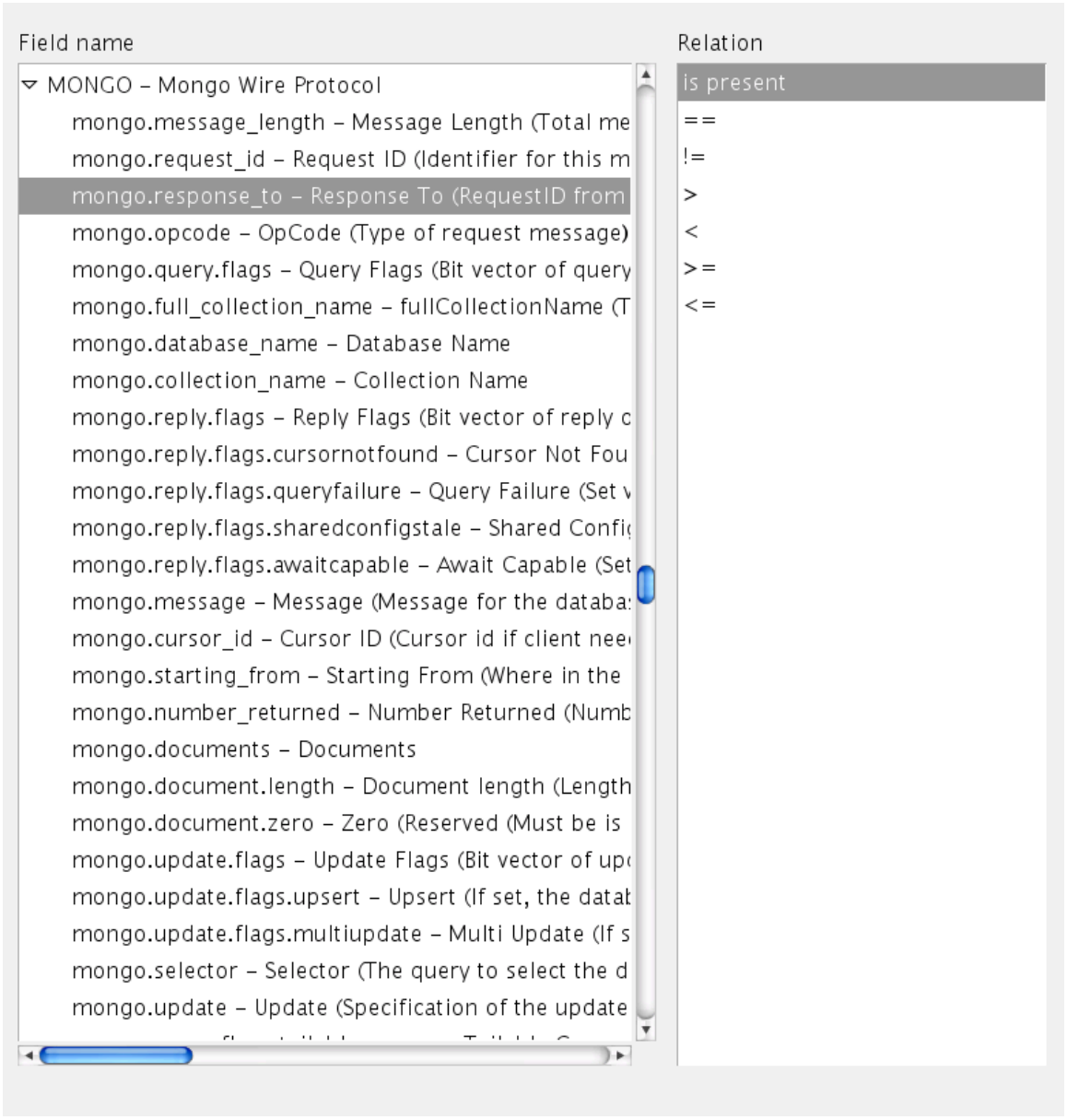
Wireshark Support for MongoDB Protocol

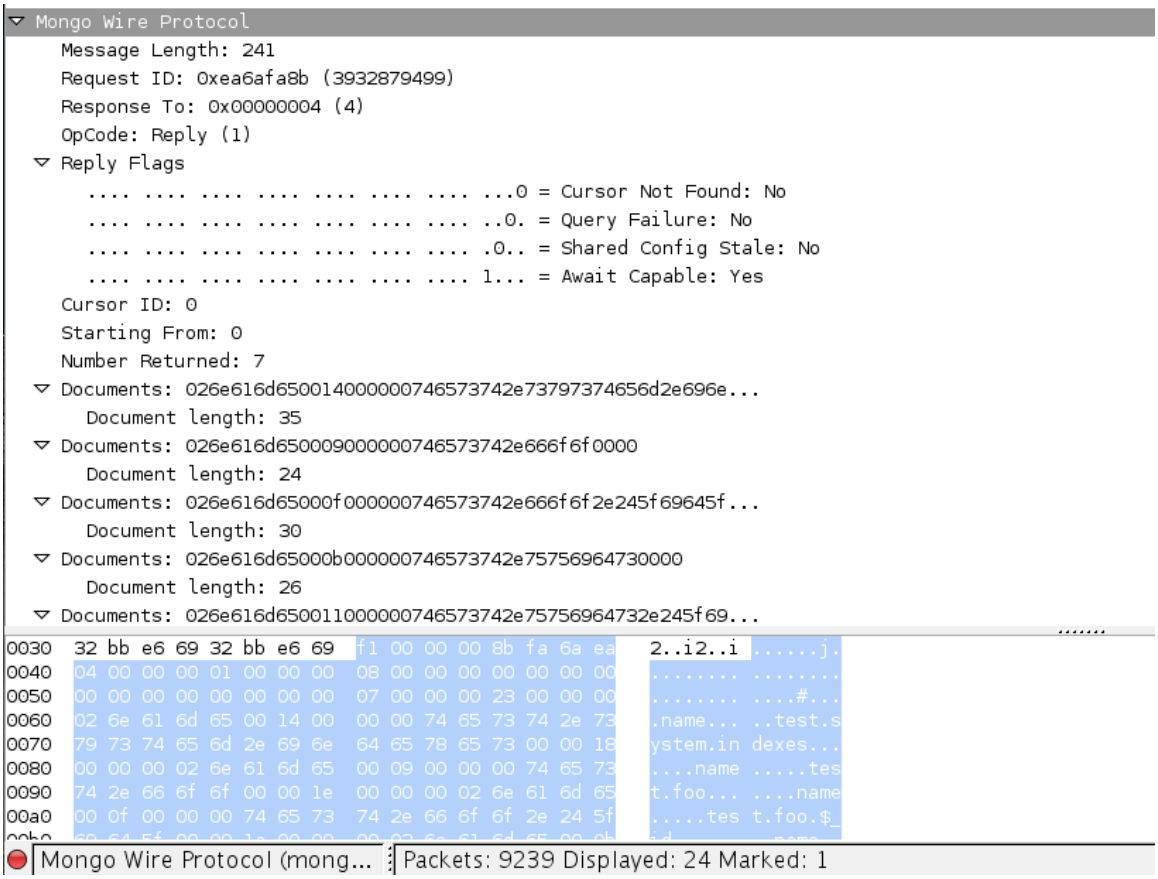
Wireshark, an advanced interactive network traffic sniffer, has full support for the MongoDB Wire protocol.

You can visually inspect MongoDB traffic, do complex filters on specific values of MongoDB wire messages and dig into individual documents both sent and received.

Note: wireshark looks for port 27017 and infers MongoDB protocol from this. If you are running on a different port number, go to Preferences...Protocols...Mongo and set your port number and it should then interpret the data.




Field name
MONGO – Mongo Wire Protocol
mongo.message_length – Message Length (Total me
mongo.request_id – Request ID (Identifier for this m
mongo.response_to – Response To (RequestID from
mongo.opcode – OpCode (Type of request message)
mongo.query.flags – Query Flags (Bit vector of query
mongo.full_collection_name – fullCollectionName (T
mongo.database_name – Database Name
mongo.collection_name – Collection Name
mongo.reply.flags – Reply Flags (Bit vector of reply c
mongo.reply.flags.cursornotfound – Cursor Not Fou
mongo.reply.flags.queryfailure – Query Failure (Set v
mongo.reply.flags.sharedconfigstale – Shared Confi
mongo.reply.flags.awaitcapable – Await Capable (Set
mongo.message – Message (Message for the databa
mongo.cursor_id – Cursor ID (Cursor id if client nee
mongo.starting_from – Starting From (Where in the
mongo.number_returned – Number Returned (Numb
mongo.documents – Documents
mongo.document.length – Document length (Length
mongo.document.zero – Zero (Reserved (Must be is
mongo.update.flags – Update Flags (Bit vector of up
mongo.update.flags.upsert – Upsert (If set, the datab
mongo.update.flags.multiupdate – Multi Update (If s
mongo.selector – Selector (The query to select the d
mongo.update – Update (Specification of the update
Relation
is present
==
!=
>
<
>=
<=



Hosting Center

What's New

- (Dec2012) SoftLayer offers servers preconfigured and optimized for MongoDB

Database-as-a-Service

- MongoOd.com
- MongoHQ
- MongoLab
- HostedMongo.com
- MongoGrid
- ObjectRocket - MongoDB service provider specializing in high performance and availability.
- MongoIC by GrandCloud (China)
- MongoHosting.com

Infrastructure-as-a-Service

- Amazon EC2
- Joyent
- Rackspace Cloud
- Windows Azure VM

Dedicated Servers / Hosting

MongoDB runs well on both virtualized and non-virtualized servers.

- ServerBeach offers preconfigured, dedicated MongoDB servers. Blog

Platform-as-a-Service

- alwaysdata
- cloudControl offers a fully managed platform-as-a-service solution with MongoDB as one of their powerful add-ons. Read the blog post MongoDB Setup at cloudControl for more information.
- dotCloud
- Heroku has add-on connectors to allow you to use from MongoDB from Heroku applications
- NodeGrid
- RedHat OpenShift supports MongoDB
- VMware CloudFoundry supports MongoDB
- Windows Azure Cloud Services

VPS

- (mt) Media Temple's (ve) server platform is an excellent choice for easy MongoDB deployment.
- A2 Hosting has a quick installer to add MongoDB to your VPS hosting account. Instructions for running the installer are on A2's wiki
- Dreamhost offers instant configuration and deployment of MongoDB
- LOCUM Hosting House is a project-oriented shared hosting and VDS. MongoDB is available for all customers as a part of their subscription plan.

More

- Linode
- Webfaction
- Presentations

Amazon EC2

- Getting Started on EC2
- Backup, Restore, Verify
- Deployment Notes
  - MongoDB via AWS Marketplace
  - Automating Deployment with CloudFormation
  - Instance Types
    - Installing MongoDB
    - Operating System
  - Networking
    - Port Management
    - Keepalive
  - Storage Configuration
    - EBS Snapshots
  - Securing instances
  - Communication across regions
- Presentations

MongoDB runs well on Amazon EC2. This page includes some notes in this regard.

Getting Started on EC2

This guide is intended to provide instructions on using the MongoDB AMI to set up production instances of MongoDB across Amazon's Web Services (AWS) EC2 infrastructure.

First, we'll step through deployment planning (instance specifications, deployment size, etc.) and then we'll set up a single production node. We'll use those setup steps to deploy a three node MongoDB replica set for production use. Finally, we'll briefly cover some advanced topics such as multi-region availability and data backups.

See the Amazon EC2 Quickstart guide for more information.

Backup, Restore, Verify

Depending upon the configuration of your EC2 instances, there are a number of ways to conduct regular backups of your data. For specific instructions on backing up, restoring and verifying refer to EC2 Backup & Restore.

Deployment Notes

MongoDB via AWS Marketplace

If you installed MongoDB via the AWS Marketplace refer to the MongoDB via AWS Marketplace guide to get a development instance up and running. If you are interested in creating a production deployment, refer to the Amazon EC2 Quickstart. Start with the section on Configuring Storage to set up a place for your data to be stored. After that refer to the Starting MongoDB section to get your MongoDB instance running. If you're interested in scaling your deployment, check out the sections on Replica Sets and Sharding.

Automating Deployment with CloudFormation

CloudFormation from Amazon Web Services provides an easy mechanism to create and manage a collection of AWS resources. To use CloudFormation you create and deploy a template which describes the resources in your stack via the AWS Management Console. We have created a series of reference templates that you can use as a starting point to build your own MongoDB deployments using CloudFormation. Check out Automating Deployment with CloudFormation for a walkthrough and the template files.

Instance Types

MongoDB works on most EC2 types including Linux and Windows. We recommend you use a 64 bit instance as this is required for all MongoDB databases of significant size. Additionally, we find that the larger instances tend to be on the freshest ec2 hardware.

Installing MongoDB

One can download a binary or build from source. Generally it is easier to download a binary. We can download and run the binary without being root. For example on 64 bit Linux:

```
[~]$ curl -O http://downloads.mongodb.org/linux/mongodb-linux-x86_64-1.0.1.tgz
[~]$ tar -xzf mongodb-linux-x86_64-1.0.1.tgz
[~]$ cd mongodb-linux-x86_64-1.0.1/bin
[bin]$ ./mongod --version
```

Before running the database one should decide where to put datafiles. Run df -h to see volumes. On some images /mnt will be the many locally attached storage volume. Alternatively you may want to use Elastic Block Store which will have a different mount point.

If you mount the file-system, ensure that you mount with the noatime and nodiratime atributes, for example

```
/dev/mapper/my_vol /var/lib/mongodb xfs noatime,noexec,nodiratime 0 0
```

Create the mongodb datafile directory in the desired location and then run the database:

```
mkdir /mnt/db
./mongod --fork --logpath ~/mongod.log --dbpath /mnt/db/
```

Operating System

Occasionally, due to the shared and virtualized nature of EC2, an instance can experience intermittent I/O problems and low responsiveness compared to other similar instances. Terminating the instance and bringing up a new one can in some cases result in better performance.

Some people have reported problems with ubuntu 10.04 on ec2.

Please read here and here for further information.

Networking

Port Management

By default the database will now be listening on port 27017. The web administrative UI will be on port 28017.

Keepalive

Change the default TCP keepalive time to 300 seconds. See our troubleshooting page for details.

Storage Configuration

For production use, we recommend raid 10 across 4-8 ebs drives for best performance. Local ec2 drives may be faster but are generally not suitable as they are ephemeral. Multiple ebs drives increase the potential number of random IO's per second (iops), but not necessarily sequential read/write throughput much. In most database applications random iops are important.

For more information, refer to EBS info at Amazon Web Services.

EBS Snapshots

If your datafiles are on a single EBS volume, you can snapshot them for backups.

If you are using journaling, simply take a snapshot (including the journal/ directory).

If not using journaling, you need to use the lock+fsync command (v1.3.1+).

Use this command to lock the database to prevent writes. Then, snapshot the volume. Then use the unlock command to allow writes to the database again. See the full EC2 Backup, Verify & Recovery guide for more information. This method may also be used with slaves / secondaries.

Securing instances

Restrict access to your instances by using the Security Groups feature within AWS. A Security Group is a set of firewall rules for incoming packets that can apply to TCP, UDP or ICMP.

A common approach is to create a MongoDB security group that contains the nodes of your cluster (replica set members or sharded cluster members), followed by the creation of a separate security group for your app servers or clients.

Create a rule in your MongoDB security group with the "source" field set to the Security Group name containing your app servers and the port set to 27017 (or whatever port you use for your MongoDB). This will ensure that only your app servers have permission to connect to your MongoDB instances.

Remember that Security Groups only control ingress traffic.

Communication across regions

Every EC2 instance will have a private IP address that can be used to communicate within the EC2 network. It is also possible to assign a public "elastic" IP to communicate with the servers from another network. If using different EC2 regions, servers can only communicate via public IPs.

To set up a cluster of servers that spans multiple regions, it is recommended to cname the server hostname to the "public dns name" provided by EC2. This will ensure that servers from a different network use the public IP, while the local servers use the private IP, thereby saving costs. This is required since EC2 security groups are local to a region.

To control communications between instances in different regions (for example, if you have two members of a replica set in one region and a third member in another), it is possible to use a built-in firewall (such as IPtables on Linux) to restrict allowed traffic to certain (elastic) IP addresses or ports.

For example one solution is following, on each server:

- set the hostname of the server

```
sudo hostname server1
```

- install "bind", it will serve as local resolver
- add a zone for your domain, say "myorg.com", and add the CNAMEs for all your servers

```
...
server1          IN     CNAME   ec2-50-19-237-42.compute-1.amazonaws.com.
server2          IN     CNAME   ec2-50-18-157-87.us-west-1.compute.amazonaws.com.
```

- restart bind and modify /etc/resolv.conf to use the local bind

```
search myorg.conf
nameserver 127.0.0.1
```

Then:

- verify that you can properly resolve server1, server2, ... using a tool like dig.
- when running mongod, db.serverStatus() should show the correct hostname, e.g. "server1:27017".
- you can then set up replica sets or shards using the simple hostname. For example connect to server1 and run "rs.initiate()", then "rs.add('server2:27017')".

Presentations

MongoDB & AWS - Free Webinar on January 20, 2012

Presentation from MongoSV (December 2011)

More Presentations

- Running MongoDB in the Cloud - MongoSF (May 2011)
- MongoDB on Amazon EC2 - Webinar (March 2011)

AWS Marketplace

- Starting MongoDB
- Security Setup
- Instance Configurations
  - MongoDB AMI
  - MongoDB with EBS RAID AMI
- More Information

If you installed MongoDB via the AWS Marketplace, refer to the instructions below for Starting MongoDB. For information on how the MongoDB with instance-storage AMI was configured, see MongoDB AMI. For information on how the MongoDB with EBS RAID storage was configured, see MongoDB with EBS RAID AMI. The Security Setup section contains information about the security group settings for these AMIs.

Starting MongoDB

The instance has been mostly configured, the only steps left are to set MongoDB to start at system boot and then start `mongod` up:

```
$ sudo chkconfig mongod on
$ sudo /etc/init.d/mongod start
Starting mongod:                                           [  OK  ]
forked process: 1234
all output going to: /log/mongod.log
```

In a few moments MongoDB will finish starting up and then you can connect to MongoDB using the `mongo` client:

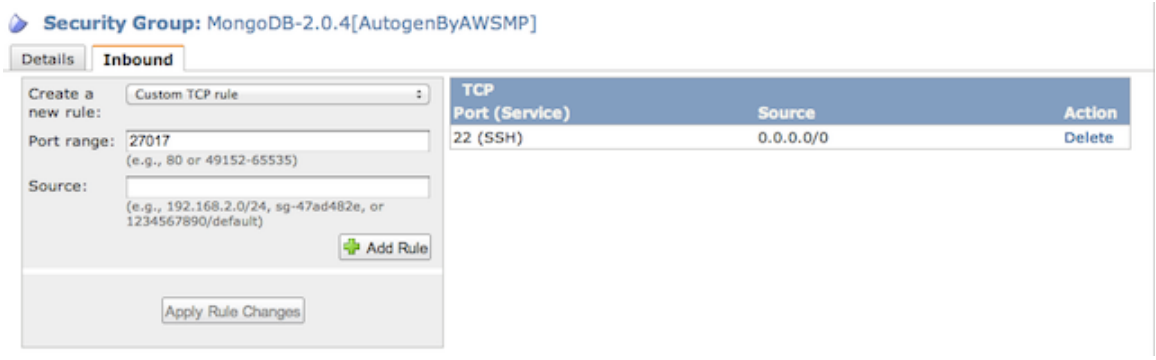
```
$ mongo
MongoDB shell version: 2.0.4
connecting to: test
>
```

If you have trouble connecting to the MongoDB daemon, check the log file (either `/var/log/mongo/mongod.log` or `/log/mongod.log`) for `mongod` console output.

Security Setup

By default, the instance starts up with a newly created security group and that group specifies that only access via port 22 (SSH) is allowed. In order to make MongoDB accessible to other instances, you'll need to add an additional rule to your security group specifying access to port 27017

along with a source (this can be a specific IP address, instances from within another security group or from an IP). Start by logging in to the AWS EC2 Management Console and navigate to the Security Groups section. Next, find the group that was created when you created your instance and add an Inbound Rule:



After the rule is applied, you'll be able to access MongoDB running in your instance from the source you specified. Consult the AWS EC2 Security Groups documentation for more information about configuring access to your instance with security groups.

Instance Configurations

MongoDB AMI

MongoDB was installed onto Amazon Linux using a subset of steps found in the Install MongoDB section of the EC2 Quickstart guide. Specifically, the 10gen repository was created and MongoDB (along with sysstat tools) were then installed via yum. This AMI uses a single EBS-backed volume as storage and does not employ RAID10 or LVM. The data directory was set to `/data` in the MongoDB configuration file `/etc/mongodb.conf`.

MongoDB with EBS RAID AMI

Using the Amazon EC2 Quickstart guide as a base, the EBS RAID AMI was configured using a subset of the steps found in the Deploying a Single Node section. This AMI was configured with block device mapping and uses 4 EBS volumes, each with 100GiB of storage. The volumes were then setup as a single RAID10 device using `mdadm` and 3 logical volumes (data, log, journal) were created using LVM. With RAID10 the total available storage is 200GiB and within that 180GiB is for `/data`, 10GiB for `/log` and 10GiB for `/journal`. The MongoDB configuration file `/etc/mongod.conf` reflects the updated locations for these elements. For more information on the specific steps, refer to the Amazon EC2 Quickstart guide.

More Information

For more information on deploying MongoDB on AWS (including advanced storage setups, backups, etc.), refer to the Amazon EC2 page or the Amazon EC2 Quickstart guide.

Automating Deployment with CloudFormation

- Template Walkthrough
  - Security
  - Storage Configuration
  - Instance Configuration
- Replica Set Stack
- Customizing Templates
  - Operating System
  - Storage
  - Instances
- Sample Template Usage

CloudFormation from Amazon Web Services provides an easy mechanism to create and manage a collection of AWS resources. To use CloudFormation you create and deploy a template which describes the resources in your stack via the AWS Management Console. CloudFormation templates are simple JSON formatted text files that can be placed under your normal source control mechanisms, stored in private or public locations such as Amazon S3.

We have created a series of reference templates (see attachments below) that you can use as a starting point to build your own MongoDB deployments using CloudFormation. The sample templates show how to build a single node MongoDB deployment and a MongoDB replica set deployment. Refer to the following sections for information on how these sample templates were developed and instructions on how to customize them for your own deployment onto AWS. The documentation provided below focuses on the sections specific to deploying MongoDB on AWS, for background on standard template sections (e.g. input parameters) refer to the AWS CloudFormation User Guide.

The sample templates we created are as follows (click to download):

- MongoDB_SingleNode.template is an example single-node MongoDB deployment with 4 EBS volumes
- MongoDB_ReplicaSetStack.template sets up a single node and references the...
- MongoDB_ReplicaSetMember.template twice to create the additional nodes (each node has 4 EBS volumes)

Template Walkthrough

Security

EC2 instances in AWS must have security groups designated during creation that specify firewall rules for each instance. In our templates, we create a simple security group `MongoSecurityGroup` that opens up port 22 (SSH) to the outside world and a separate rule (`AWS::EC2::SecurityGroupIngress`) that's used to open up the standard `mongod` port (27017) to other instances within that group:

```
"MongoSecurityGroup" : {
    "Type" : "AWS::EC2::SecurityGroup",
    "Properties" : {
        "GroupDescription" : "MongoDB security group",
        "SecurityGroupIngress" : [
            {
                "IpProtocol" : "tcp",
                "FromPort" : "22",
                "ToPort" : "22",
                "CidrIp" : "0.0.0.0/0"
            }
        ]
    }
},
"MongoSecurityGroupIngress" : {
    "Type" : "AWS::EC2::SecurityGroupIngress",
    "Properties" : {
        "GroupName" : { "Ref" : "MongoSecurityGroup" },
        "IpProtocol" : "tcp",
        "FromPort" : "27017",
        "ToPort" : "27017",
        "SourceSecurityGroupName" : { "Ref" : "MongoSecurityGroup" }
    }
},
```

Depending on the type of deployment you are creating, you may need to add additional security group ingress rules to open up additional ports, available to your instances or to the outside world (e.g. port 27018 for sharding).

Storage Configuration

When using MongoDB on AWS, we recommend using multiple EBS volumes configured as a single RAID10 storage device for your data. Configuring storage via EBS volumes using CloudFormation requires multiple steps. First the EBS volume must be created, using the `VolumeSize` input value and the same availability zone that we'll use for our EC2 instance (see Instance Configuration).

```
"MongoVolume1" : {
    "Type" : "AWS::EC2::Volume",
    "Properties" : {
        "Size" : { "Ref" : "VolumeSize" },
        "AvailabilityZone" : { "Fn::GetAtt" : [ "MongoInstance", "AvailabilityZone" ]}
    }
},
```

The next step is to attach the volume to an EC2 instance. By referencing the instance name in "InstanceId" we ensure that the EBS volumes will be created after the instance is created.

```
"MongoVolumeMount1" : {
    "Type" : "AWS::EC2::VolumeAttachment",
    "Properties" : {
        "InstanceId" : { "Ref" : "MongoInstance" },
        "VolumeId" : { "Ref" : "MongoVolume1" },
        "Device" : "/dev/sdh1"
    }
},
```

In the attached sample templates we used 4 EBS volumes as the basis for our RAID10 configuration. If you are interested in increasing the number of EBS volumes, you will need to add additional `AWS::EC2::Volume` and `AWS::EC2::VolumeAttachment` resources inside of your template. Refer to the Customizing Storage section below for more information on the steps required.

Instance Configuration

The centerpiece of the CloudFormation template is the creation of the EC2 instances that will be used as the MongoDB server. There are two main sections to be configured for your instance, the instance metadata and instance properties. In the sample provided, the metadata contains information about the packages to be installed (mdadm and sysstat) and any files to be created within the instance. In our sample, the only file to be created is a `yum` repository entry to facilitate MongoDB being installed via yum after the instance has booted.

```
"MongoInstance" : {
    "Type" : "AWS::EC2::Instance",
    "Metadata" : {
        "AWS::CloudFormation::Init" : {
            "config" : {
                "packages" : {
                    "yum" : {
                        "mdadm" : [],
                        "sysstat" : []
                    }
                },
                "files" : {
                    "/etc/yum.repos.d/10gen.repo" : {
                        "content" : { "Fn::Join" : ["", [
                            "[10gen]\n",
                            "name=10gen Repository\n",
                            "baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64\n",
"gpgcheck=0\n"
                        ] ] },
                        "mode" : "000644",
                        "owner" : "root",
                        "group" : "root"
                    }
                }
            }
        }
    },
```

For more information about the possible options for the metadata section, refer to the CloudFormation documentation for the AWS::EC2::Instance resource. The properties section in the template is uses to specify things like the instance type, AMI, security groups and a script to be run after boot (found in the "UserData" section):

```
"Properties" : {
    "InstanceType" : { "Ref" : "InstanceType" },
    "ImageId" : { "Fn::FindInMap" : [ "RegionImageZone", { "Ref" : "AWS::Region" },
        { "Fn::FindInMap" : [ "InstanceTypeArch", { "Ref" : "InstanceType" }, "Arch" ] } ] },
    "SecurityGroups" : [ { "Ref" : "MongoSecurityGroup" } ],
    "KeyName" : { "Ref" : "KeyName" },
    "UserData" : { "Fn::Base64" : { "Fn::Join" : ["", [
        "#!/bin/bash\n",
        ...
```

The instance type is determined by the `InstanceType` input parameter. The "ImageId" specifies the AMI to use, which is determined by the chosen instance type (e.g. m1.large) and region in which the instance is launched (e.g. us-east-1). Refer to the "Mappings" section inside the sample templates for a list of the available AMIs.

The "UserData" section shown above contains a bash script that be executed once the instance is launched and available. The first step is install the `aws-cfn-bootstrap` tools which are used in the script to initialize the instance, signal when errors have occurred and when the the instance creation is complete:

```
"yum update -y aws-cfn-bootstrap\n",

"## Error reporting helper function\n",
"function error_exit\n",
"{\n",
"   /opt/aws/bin/cfn-signal -e 1 -r \"$1\" '", { "Ref" : "WaitHandleMongoInstance" }, "'\n",
"   exit 1\n",
"}\n",

"## Initialize CloudFormation bits\n",
"/opt/aws/bin/cfn-init -v -s ", { "Ref" : "AWS::StackName" }, " -r MongoInstance",
"   --access-key ",  { "Ref" : "HostKeys" },
"   --secret-key ", {"Fn::GetAtt": ["HostKeys", "SecretAccessKey"]},
"   --region ", { "Ref" : "AWS::Region" }, " > /tmp/cfn-init.log 2>&1 || error_exit
$(</tmp/cfn-init.log)\n",
```

Next we include a series of sleep conditions in case our EBS volumes are not yet available. If you plan to use more than 4 EBS volumes in your CloudFormation template, you should add additional sleep conditions here:

```
"## Waiting for EBS mounts to become available\n",
"while [ ! -e /dev/sdh1 ]; do echo waiting for /dev/sdh1 to attach; sleep 10; done\n",
"while [ ! -e /dev/sdh2 ]; do echo waiting for /dev/sdh2 to attach; sleep 10; done\n",
"while [ ! -e /dev/sdh3 ]; do echo waiting for /dev/sdh3 to attach; sleep 10; done\n",
"while [ ! -e /dev/sdh4 ]; do echo waiting for /dev/sdh4 to attach; sleep 10; done\n",
```

Then we install MongoDB and create the RAID10 device:

```
"yum -y install mongo-10gen-server > /tmp/yum-mongo.log 2>&1\n",

"## Create RAID10 and persist configuration\n",
"mdadm --verbose --create /dev/md0 --level=10 --chunk=256 --raid-devices=4 /dev/sdh1 /dev/sdh2
/dev/sdh3 /dev/sdh4 > /tmp/mdadm.log 2>&1\n",
"echo ``mdadm --detail --scan`` | tee -a /etc/mdadm.conf\n",
```

With the RAID10 created, we can set some block device attributes (read-ahead) for each storage device:

```
"## Set read-ahead on each device\n",
"blockdev --setra 128 /dev/md0\n",
"blockdev --setra 128 /dev/sdh1\n",
"blockdev --setra 128 /dev/sdh2\n",
"blockdev --setra 128 /dev/sdh3\n",
"blockdev --setra 128 /dev/sdh4\n",
```

Now we use LVM to create a series of logical volumes for data, journal and logs. The values used below for each volume are specified as percentages, those may need to be changed for your deployment. After creating the volumes, we create the filesystem, mount points and entries in the filesystem table. The last storage-related step is to set the user:group ownership of each mount point to `mongod:mongod`.

```
"## Create physical and logical volumes\n",
"dd if=/dev/zero of=/dev/md0 bs=512 count=1\n",
"pvcreate /dev/md0\n",
"vgcreate vg0 /dev/md0\n",
"lvcreate -l 90%vg -n data vg0\n",
"lvcreate -l 5%vg -n log vg0\n",
"lvcreate -l 5%vg -n journal vg0\n",

"## Create filesystems and mount point info\n",
"mke2fs -t ext4 -F /dev/vg0/data > /tmp/mke2fs1.log 2>&1\n",
"mke2fs -t ext4 -F /dev/vg0/log > /tmp/mke2fs2.log 2>&1\n",
"mke2fs -t ext4 -F /dev/vg0/journal > /tmp/mke2fs3.log 2>&1\n",

"mkdir /data\n",
"mkdir /log\n",
"mkdir /journal\n",

"echo '/dev/vg0/data /data ext4 defaults,auto,noatime,noexec 0 0' | tee -a /etc/fstab\n",
"echo '/dev/vg0/log /log ext4 defaults,auto,noatime,noexec 0 0' | tee -a /etc/fstab\n",
"echo '/dev/vg0/journal /journal ext4 defaults,auto,noatime,noexec 0 0' | tee -a /etc/fstab\n",

"mount /data > /tmp/mount1.log 2>&1\n",
"mount /log > /tmp/mount2.log 2>&1\n",
"mount /journal > /tmp/mount3.log 2>&1\n",

"ln -s /journal /data/journal\n",

"chown -R mongod:mongod /data > /tmp/chown1.log 2>&1\n",
"chown -R mongod:mongod /log > /tmp/chown2.log 2>&1\n",
"chown -R mongod:mongod /journal > /tmp/chown3.log 2>&1\n",
```

Next we proceed to creating a MongoDB configuration file, specifiying the logpath and data directory (among others), and start MongoDB:

```
"## Update mongod configuration\n",
"cat <<EOF > /etc/mongod.conf\n",
"logpath=/data/log/mongod.log\n",
"logappend=true\n",
"fork = true\n",
"dbpath=/data\n",
"rest=true\n",
"EOF\n",

"## Start mongod\n",
"/etc/init.d/mongod start > /tmp/mongod-start.log 2>&1\n",
```

The final step is to signal our previously created WaitCondition that setup is complete:

```
"## CloudFormation signal that setup is complete\n",
"/opt/aws/bin/cfn-signal -e 0 -r \"MongoInstance setup complete\" '", { "Ref" :
"WaitHandleMongoInstance" }, "'\n"
```

Once this script has completed executing, the instance and associated resources have been created and our CloudFormation stack is ready to go.

Replica Set Stack

The "ReplicaSetStack" sample template first creates two "ReplicaSetMember" instances (complete with storage configuration) and then creates the overall replica set. The "ReplicaSetMember" template is modeled very closely after the "SingleNode" template except it includes additional input parameters and adds additional commands to the instance setup script in "UserData" specific to creating replica set members (adding in the `replSet` parameter to the MongoDB configuration file).

```
"## Update mongod configuration\n",
"cat <<EOF > /etc/mongod.conf\n",
"logpath=/data/log/mongod.log\n",
"logappend=true\n",
"fork = true\n",
"dbpath=/data\n",
"rest=true\n",
"replSet=", {"Ref" : "ReplicaSetName"}, "\n",
"EOF\n",
```

The "ReplicaSetStack" template also closely follows the "SingleNode" template but adds the following replica set specific steps: (1) it creates a "replicaSetConfigInit.js" file containing the replica set configuration (with hostnames for the additional members) and (2) initiates the replica set. These steps are executed just prior to signaling that the instance setup has been completed:

```
"cat <<EOF > /tmp/replicaSetConfigInit.js\n",
"config = {_id: \"", { "Ref" : "ReplicaSetName" } ,"\", members : [",
    "{_id : 0, host:\"$HOSTNAME:27017\"},",
    "{_id : 1, host:\"", { "Fn::GetAtt" : ["ReplicaSetMember1", "Outputs.ReplicaSetMemberName"] },"
:27017\"},",
    "{_id : 2, host:\"", { "Fn::GetAtt" : ["ReplicaSetMember2", "Outputs.ReplicaSetMemberName"] },"
:27017\"}",
"]};\n",
"rs.initiate(config);\n",
"EOF\n",

"/usr/bin/mongo /tmp/replicaSetConfigInit.js > /tmp/replica-setup.log 2>&1\n",
```

The child "ReplicaSetMember" instances are created from within the "ReplicaSetStack" template using the following resource definition. The inputs for the "ReplicaSetMember" instances are taken from the "ReplicaSetStack" template:

```
"ReplicaSetMember1" : {
    "Type" : "AWS::CloudFormation::Stack",
    "Properties" : {
        "TemplateURL" : "http://S3URL/MongoDB_ReplicaSetMember.template",
"Parameters" : {
            "KeyName" : { "Ref" : "KeyName" },
            "InstanceType" : { "Ref" : "InstanceType" },
            "VolumeSize" : { "Ref" : "VolumeSize" },
            "AccessKeyId" : { "Ref" : "AccessKey" },
            "SecretAccessKey" : {"Fn::GetAtt": ["AccessKey", "SecretAccessKey"]},
            "ReplicaSetName" : { "Ref" : "ReplicaSetName" },
            "SecurityGroupName" : { "Ref" : "MongoSecurityGroup" },
            "InstanceZone" : { "Fn::FindInMap" : ["RegionZone", { "Ref" : "AWS::Region"}, "AZ1"] }
        }
    }
}
```

Customizing Templates

Operating System

In the sample templates provided, we used Amazon Linux as the base operating system. If you are interested in using a recent release of Ubuntu (9.10 or greater), please use the following steps to customize the templates. The steps refer to the line numbers for the MongoDB_ReplicaSetStack.template but changes should also be made in MongoDB_ReplicaSetMember.template.

First, update the "files" content of the EC2 instance Metadata section to use the 10gen apt source:

```
"files" : {
    " /etc/apt/sources.list.d/10gen.list" : {
        "content" : { "Fn::Join" : ["", [
            "deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen\n"
] ] },
        "mode" : "000644",
        "owner" : "root",
        "group" : "root"
    }
}
```

Then in the UserData section of the EC2 instance configuration, make the following changes:

- Line 220: change `yum -y install mongo-10gen-server...` to `apt-get install mongodb-10gen`
- Line 221: add `service mongodb stop`
- Line 260, 261, 262: update to use `mongodb:mongodb` as the owner:group
- Line 265: update to `cat <<EOF > /etc/mongodb.conf`
- Line 275: change `/etc/init.d/mongod start...` to `service mongodb start`

Storage

In the sample templates provided, we used 4 EBS volumes as the basis for our RAID10 configuration. If you are interested in using additional volumes you will need to update the following items for each new volume you want to add:

- Add an "AWS::EC2::Volume" resource
- Add an "AWS::EC2::VolumeAttachment" resource and mount point
- Add an additional sleep condition
- Update the call to `mdadm` to include your additional volumes

Instances

The sample "ReplicaSetStack" template creates three instances, one from the "stack" template and two additional "ReplicaSetMember" instances, via the "AWS::CloudFormation::Stack" resource in the "ReplicaSetStack" template. If you are interested in adding additional replica set members, you'll need to create an additional member instances and edit the `replicaSetConfigInit.js` found within the "ReplicaSetStack" template. Refer to the Replica Set Stack for information about the additional resources and config file to be updated. When creating the replica set the templates spread the created instances across multiple availability zones (e.g. `us-east-1a` or `us-east-1b`). When adding additional instances be sure to specify your desired Availability Zone for increase redundancy.

Sample Template Usage

If you are interesting in launching a single MongoDB node in AWS, download the MongoDB_SingleNode.template file and edit it for your specific deployment. Once you have a completed template, login to the AWS Management Console and navigate to the "AWS CloudFormation" and "Create New Stack". There you'll be prompted to upload your template and input the necessary parameters.

If instead you are interested in launching a multi-node replica set, download the MongoDB_ReplicaSetStack.template and MongoDB_ReplicaSetMember.template. In order for a parent template ("ReplicaSetStack") to refer to child templates ("ReplicaSetMember"), the child template must be uploaded to S3 and the S3 URL of the child template must be specified in the parent template. Once you've uploaded the child template to S3, update the "TemplateURL" parameter in each "ReplicaSetMember" resource in the "ReplicaSetStack" template:

```
"ReplicaSetMember1" : {
    "Type" : "AWS::CloudFormation::Stack",
    "Properties" : {
        "TemplateURL" : "http://S3URL/MongoDB_ReplicaSetMember.template",
"Parameters" : {
            "KeyName" : { "Ref" : "KeyName" },
            "InstanceType" : { "Ref" : "InstanceType" },
            "VolumeSize" : { "Ref" : "VolumeSize" },
            "AccessKeyId" : { "Ref" : "AccessKey" },
            "SecretAccessKey" : {"Fn::GetAtt": ["AccessKey", "SecretAccessKey"]},
            "ReplicaSetName" : { "Ref" : "ReplicaSetName" },
            "SecurityGroupName" : { "Ref" : "MongoSecurityGroup" },
            "InstanceZone" : { "Fn::FindInMap" : ["RegionZone", { "Ref" : "AWS::Region"}, "AZ1"] }
        }
    }
}
```

After updating the "TemplateURL" parameters, login to the AWS Management Console and navigate to the "AWS CloudFormation" and "Create New Stack". There you'll be prompted to upload your template and input the necessary parameters.

For more information on deploying MongoDB on AWS, refer to the Amazon EC2 page and the Amazon EC2 Quickstart guide.

Amazon EC2 Quickstart

- Prerequisites
- Planning Your Deployment
  - Instance Specifications
  - Storage Configuration
  - Topology
  - Security
- Securing Your Deployment
- Deploying a Single Node
  - Launch Instance
  - Configure Storage
  - Install and Configure MongoDB
  - Starting MongoDB
- Deploying a Multi-node Replica Set
  - Replica Set Background
  - Create and Configure Instances
  - Configure Replica Set
- Deploying a Sharded Configuration
  - Simple Sharding Architecture
- Backup and Restore

This guide is intended to provide instructions on setting up production instances of MongoDB across Amazon's Web Services (AWS) EC2 infrastructure.

First, we'll step through deployment planning (instance specifications, deployment size, etc.) and then we'll set up a single production node. We'll use those setup steps to deploy a three node MongoDB replica set for production use. Finally, we'll briefly cover some advanced topics such as multi-region availability and data backups.

If you installed MongoDB via the AWS Marketplace this guide can be used to get your instance up and running quickly. Start with the section on Configuring Storage to set up a place for your data to be stored. After that refer to the Starting MongoDB section to get your instance started and allow you to get started using MongoDB. If you're interested in scaling your deployment, check out the sections on Replica Sets and Sharding below.

Prerequisites

Generally, there are two ways to work with EC2 - via the command line tools or the AWS Management Console. This guide will use the EC2 command line tools to manage security and storage and launch instances. Use the following steps to setup the EC2 tools on your system:

- Download the EC2 command line tools
- Next, refer to **Prerequisites** and **Setting Up the Tools** from Amazon's Getting Started with the Command Line Tools

Planning Your Deployment

Before starting up your EC2 instances, it's best to sketch out a few details of the planned deployment. Regardless of the configuration or the number of nodes in your deployment, we'll configure each one in roughly the same manner.

Instance Specifications

Amazon has several instance choices available ranging from low to high (based on CPU and memory) throughput. Each instance available serves a different purpose and plays a different role when planning your deployment. There are several roles to consider when deploying a MongoDB production cluster. Consider a situation where your deployment contains an even number of replicated data (`mongod`) instances, an arbiter participates in electing the primary but doesn't hold any data. Therefore a Small instance may be appropriate for the arbiter role but for data nodes you'll want to use 64-bit (standard Large or higher) instances, which have greater CPU and memory scaling. For the purposes of this guide we'll be focused on deploying `mongod` instances that use the standard Large instance. The AMI (ID: `ami-41814f28`) is the 64-bit base Amazon Linux, upon which we'll install MongoDB.

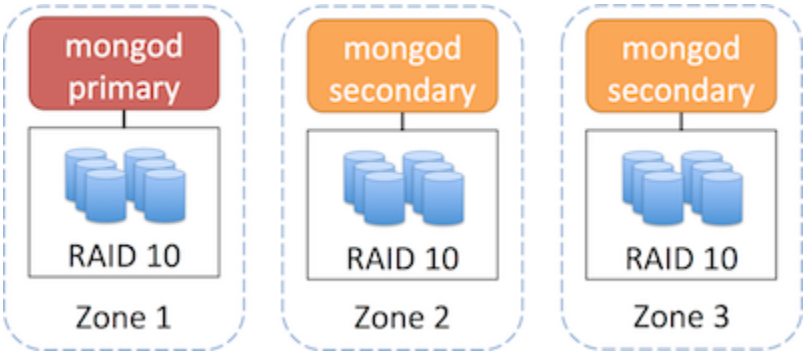
Storage Configuration

For storage we recommend using multiple EBS volumes (as opposed to instance-based storage which is ephemeral) in a RAID-based configuration. Specifically for production deployments you should use RAID 10 across 4-8 EBS volumes for the best performance. When deploying RAID 10, you'll need enough volume storage to be twice that of the desired available storage for MongoDB. Therefore for 8 GiB of available storage you'll need to have 16 GiB of allocated storage space across multiple EBS volumes.

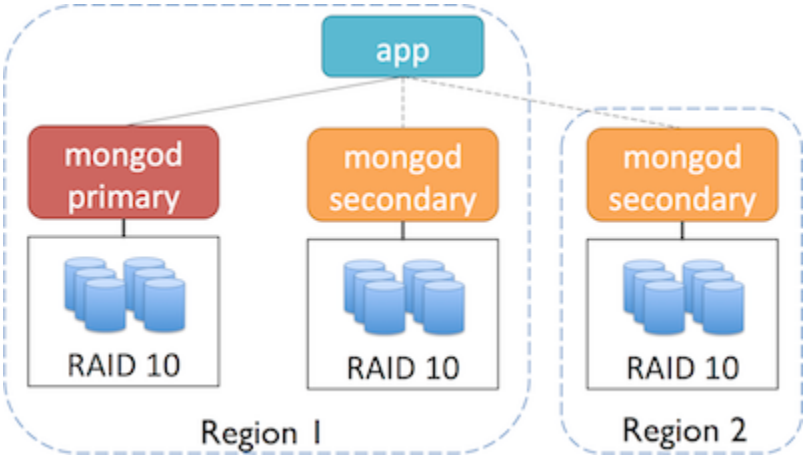
Topology

For the purposes of this guide, our topology will be somewhat simple: one to three EC2 instances, each with multiple EBS volumes attached, all located in the same availability zone (and by extension, within the same region). If you are interested in creating a deployment that spans availability zones or regions, it's best to do that planning up front and take into account security group designations (they cannot span regions) and hostname/DNS access (AWS internal IP addresses can only be used within a zone).

An example 3 node replica set with RAID 10 storage, spanning multiple availability zones would like similar to the following. Availability zones within EC2 are similar to different server racks, therefore it is recommended that you deploy your replica set across multiple zones.



For even greater redundancy and failover, you could also deploy your replica set across multiple regions (and go further with multiple zones in each region):

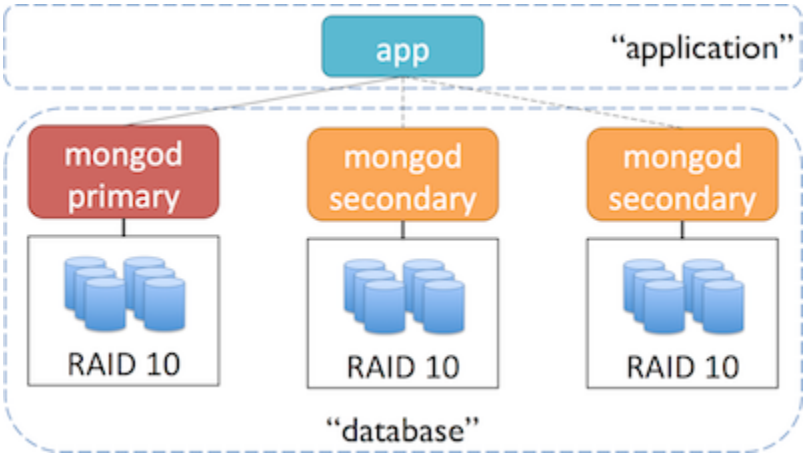


Refer to the AWS documentation on Using Regions and Availability Zones for more information.

Security

The recommended approach to securing your instances is to use multiple security groups for your MongoDB deployment, one for each type of interaction. For example, you could use one group to manage communication amongst the nodes in your cluster, another group that allows your application to communicate with the database and optionally, a group for tools and maintenance tasks.

An example setup with two security groups might look like this:



Securing Your Deployment

Before starting up instances we want to get the security groups created. As previously discussed, we recommend using multiple groups, one for each type of interaction. The following steps will show you how to create two groups (one for your app and another for your database) and provide the authorizations necessary for communication between them.

From the command line, create the `database` group and authorize SSH:

```
$ ec2-create-group database --description "security group for database"
GROUP   sg-0814f660 database    security group for database

$ ec2-authorize database -p 22
GROUP           database
PERMISSION      database    ALLOWS  tcp 22  22  FROM    CIDR    0.0.0.0/0   ingress
```

Authorize communication within the group of MongoDB instances by adding the group to itself. Note you'll need to provide the user account number (using the `-u` flag) when authorizing groups:

```
$ ec2-authorize database -o database -u [AWS_ACCOUNT_NUMBER]
GROUP           database
PERMISSION      database    ALLOWS  tcp 0   65535   FROM    USER    AWS_ACCOUNT_NUMBER  NAME database
ingress
PERMISSION      database    ALLOWS  udp 0   65535   FROM    USER    AWS_ACCOUNT_NUMBER  NAME database
ingress
PERMISSION      database    ALLOWS  icmp    -1  -1  FROM    USER    AWS_ACCOUNT_NUMBER  NAME database
ingress
```

Optionally, for testing you could also enable the port for the MongoDB web-based status interface (port `28017`):

```
$ ec2-authorize database -p 28017
GROUP           database
PERMISSION      database    ALLOWS  tcp 28017   28017   FROM    CIDR    0.0.0.0/0   ingress
```

Now create a group that will hold application servers, which will communicate with the database cluster:

```
$ ec2-create-group application --description "security group for application servers"
GROUP   sg-081bf960 application security group for application servers
```

Finally, authorize communication from the application servers (group `application`) to the MongoDB instances (group `database`):

```
$ ec2-authorize database -o application -u [AWS_ACCOUNT_NUMBER]
GROUP           database
PERMISSION      database    ALLOWS  tcp 0   65535   FROM    USER    AWS_ACCOUNT_NUMBER  NAME
application        ingress
PERMISSION      database    ALLOWS  udp 0   65535   FROM    USER    AWS_ACCOUNT_NUMBER  NAME
application        ingress
PERMISSION      database    ALLOWS  icmp    -1  -1  FROM    USER    AWS_ACCOUNT_NUMBER  NAME
application        ingress
```

Refer to the AWS guide Using Security Groups for more information on creating and managing security groups.

The next step is to generate an SSH key-pair that we'll use to connect to our running EC2 instances. Amazon's tools provide a mechanism to quickly generate a public-private key pair. Once generated, we'll need to save the private key so that we can use it to connect via SSH later (click here for more info on key pairs and AWS).

First, generate the key pair:

```
$ ec2-add-keypair cluster-keypair
KEYPAIR cluster-keypair 05:9b:b3:a6:2e:2f:54:6e:1d:a7:0a:89:56:36:b2:e8:44:37:cf:66
-----BEGIN RSA PRIVATE KEY-----
...
[private key contents]
...
-----END RSA PRIVATE KEY-----
```

Save the contents of the key to a file (including the `BEGIN` and `END` lines) and make sure that file is only readable by you:

```
$ chmod 600 private-key-file
```

Optionally, you can also the key to the SSH agent to ease connecting to our instances later:

```
$ ssh-add private-key-file
```

We're finished with the pre-deployment steps; we've covered the storage and security considerations that's necessary to setup and deploy our instances.

Deploying a Single Node

We'll start our deployment by setting up single node because later on we'll use the same steps to set up a larger cluster. The first step is to launch the instance and then setup the EBS-backed RAID 10 storage for the instance. Setting up the storage requires creating, attaching, configuring and formatting the volumes where our data will be stored.

Note: If you created a MongoDB instance via the AWS Marketplace, skip ahead to Configure Storage below.

Launch Instance

From the command line we can launch the instance. We'll need to supply an ID for an Amazon Machine Image (AMI) that we'll build our node from. We recommend using a 64-bit Amazon Linux AMI as the base of your MongoDB nodes. In this example, we are using `ami-e565ba8c` with the number of nodes (`1`), security group (`database`), authentication keypair (`cluster-keypair`), type of instance (`m1.large`) and availability zone (`us-east-1a`). Depending on the region you deploy into, a different AMI ID may be needed:

```
$ ec2-run-instances ami-e565ba8c -n 1 -g database -k cluster-keypair -t m1.large -z us-east-1a

RESERVATION r-f57f8094  711489243244    database
INSTANCE    i-a3323dc0  ami-e565ba8c            pending cluster-keypair 0       m1.large
2011-12-06T23:18:18+0000    us-east-1c  aki-825ea7eb            monitoring-disabled
ebs                 paravirtual xen     sg-0814f660, sg-1e15f776    default
```

Next, let's add some tags to the instance so we can identify it later. Tags are just metadata key-value pairs:

```
$ ec2-create-tags i-11eee072 --tag Name=QuickstartTestNode --tag Owner=Bob

TAG instance    i-11eee072  Name    QuickstartTestNode
TAG instance    i-11eee072  Owner   Bob
```

Now we can ascertain some status information about running instances at AWS (includes EBS volumes as well):

```
$ ec2-describe-instances

RESERVATION r-f57f8094  711489243244    database
INSTANCE    i-a3323dc0  ami-e565ba8c    ec2-107-20-68-113.compute-1.amazonaws.com
ip-10-2-182-67.ec2.internal running cluster-keypair 0       m1.large    2011-12-06T23:18:18+0000
us-east-1c  aki-825ea7eb            monitoring-disabled 107.20.68.113   10.2.182.67         ebs
paravirtual xen     sg-0814f660, sg-1e15f776    default
BLOCKDEVICE /dev/sda1   vol-2348cf4e    2011-12-06T23:18:43.000Z
```

Configure Storage

Now that the instance is running, let's create the EBS volumes we'll use for storing our data. In this guide we'll set up 4 volumes with 4 GiB of storage each (configured that's 16 GiB but because we're using a RAID 10 configuration that will become 8 GiB of available storage).

First off, create the EBS volumes supplying the size (4) and zone (`us-east-1a`) and save the results into a temporary file that we'll read from for the next command. Here's the command we'll use:

```
$ for x in {1..4}; do \
> ec2-create-volume -s 4 -z us-east-1a; \
> done > vols.txt
```

Here's the output of that command:

```
$ more vols.txt
VOLUME  vol-e796108a    4               us-east-1a      creating        2011-12-07T04:37:21+0000
VOLUME  vol-c39610ae    4               us-east-1a      creating        2011-12-07T04:37:30+0000
VOLUME  vol-a19610cc    4               us-east-1a      creating        2011-12-07T04:37:39+0000
VOLUME  vol-b19610dc    4               us-east-1a      creating        2011-12-07T04:37:47+0000
```

Now, let's attach those newly created volumes to our previously launched running instance from above. From the command line we'll start with the temp file (`vols.txt`), the running instance ID (`i-11eee072`), and a prefix for each attached device (`/dev/sdh`):

```
$ (i=0; \
> for vol in $(awk '{print $2}' vols.txt); do \
>   i=$((i+1)); \
>   ec2-attach-volume $vol -i i-11eee072 -d /dev/sdh${i}; \
> done)
```

Assuming the volumes attached successfully, you should see something like this:

```
ATTACHMENT  vol-e796108a    i-11eee072  /dev/sdh1   attaching   2011-12-07T04:48:22+0000
ATTACHMENT  vol-c39610ae    i-11eee072  /dev/sdh2   attaching   2011-12-07T04:48:29+0000
ATTACHMENT  vol-a19610cc    i-11eee072  /dev/sdh3   attaching   2011-12-07T04:48:37+0000
ATTACHMENT  vol-b19610dc    i-11eee072  /dev/sdh4   attaching   2011-12-07T04:48:44+0000
```

Now we'll need to connect to the running instance via SSH and configure those attached volumes as a RAID array. If you added the private key to your running SSH agent, you should be able to connect with something like (substituting your instance's hostname):

```
$ ssh ec2-user@ec2-a-b-c-d.amazonaws.com
```

And now create the RAID array using the built-in `mdadm`. You'll need the level (`10`), number of volumes (4), name of the new device (`/dev/md0`) and the attached device prefix (`/dev/sdh*`):

```
$ sudo mdadm --verbose --create /dev/md0 --level=10 --chunk=256 --raid-devices=4 /dev/sdh1 /dev/sdh2
/dev/sdh3 /dev/sdh4
$ echo 'DEVICE /dev/sdh1 /dev/sdh2 /dev/sdh3 /dev/sdh4' | sudo tee -a /etc/mdadm.conf
$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm.conf
```

Once `mdadm` is done and we've persisted the storage configuration, we'll need to tune the EBS volumes to achieve desired performance levels. This tuning is done by setting the "read-ahead" on each device. For more information refer to the blockdev man page.

```
$ sudo blockdev --setra 128 /dev/md0
$ sudo blockdev --setra 128 /dev/sdh1
$ sudo blockdev --setra 128 /dev/sdh2
$ sudo blockdev --setra 128 /dev/sdh3
$ sudo blockdev --setra 128 /dev/sdh4
```

With the RAID10 created we now turn to the Logical Volume Manager (`lvm`) which we'll use to create logical volumes for the data, log files and journal for MongoDB. The purpose of using `lvm` is to (1) safely partition differnt objects from each other and (2) provide a mechanism that we can use to grow our storage sizes later. First we start by zeroing out our RAID, creating a physical device designation and finally a volume group for that device.

```
$ sudo dd if=/dev/zero of=/dev/md0 bs=512 count=1
$ sudo pvcreate /dev/md0
$ sudo vgcreate vg0 /dev/md0
```

Once the volume group has been created, we now create logical volumes for the data, logs and journal. Depending upon the amount of available storage you may want to designate specific sizes vs. volume group percentages (as shown below). We recommend approximately 10GB for log storage and 10GB for journal storage.

```
$ sudo lvcreate -l 90%vg -n data vg0
$ sudo lvcreate -l 5%vg -n log vg0
$ sudo lvcreate -l 5%vg -n journal vg0
```

At this point we have three volumes to configure (`/dev/vg0/...`). For each volume we'll create a filesystem, mount point and an entry in the filesystem table. In the example below we used the ext4 filesystem however you could instead elect to use xfs, just be sure to edit the `mke2fs` and `sed` commands accordingly. The `/etc/fstab` entries require the partition (e.g. `/dev/vg0/data`), a mount point for the filesystem (`/data`), the filesystem type (`ext4` or `xfs`) and the mount parameters (`defaults,auto,noatime,noexec,nodiratime 0 0`, refer to the mount man page for more information on these parameters:

```
$ sudo mke2fs -t ext4 -F /dev/vg0/data
$ sudo mke2fs -t ext4 -F /dev/vg0/log
$ sudo mke2fs -t ext4 -F /dev/vg0/journal

$ sudo mkdir /data
$ sudo mkdir /log
$ sudo mkdir /journal

$ echo '/dev/vg0/data /data ext4 defaults,auto,noatime,noexec 0 0' | sudo tee -a /etc/fstab
$ echo '/dev/vg0/log /log ext4 defaults,auto,noatime,noexec 0 0' | sudo tee -a/etc/fstab
$ echo '/dev/vg0/journal /journal ext4 defaults,auto,noatime,noexec 0 0' | sudo tee -a /etc/fstab
```

Now mount all of the storage devices. By adding the entry to `/etc/fstab`, we've shortened the call to mount because it will look in that file for the extended command paramters.

```
$ sudo mount /data
$ sudo mount /log
$ sudo mount /journal
```

With the devices mounted we issue one last call to set the MongoDB journal files to be written to our new journal device, via a symbolic link to the new device:

```
$ sudo ln -s /journal /data/journal
```

Install and Configure MongoDB

Note: If you created a MongoDB instance via the AWS Marketplace, skip ahead to Starting MongoDB below.

Now that the storage has been configured, we need to install and configure MongoDB to use the storage we've set up, then set it to start up on boot automatically. First, add an entry to the local `yum` repository for MongoDB:

```
$ echo "[10gen]
name=10gen Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64
gpgcheck=0" | sudo tee -a /etc/yum.repos.d/10gen.repo
```

Next, install MongoDB and the `sysstat` diagnostic tools:

```
$ sudo yum -y install mongo-10gen-server
$ sudo yum -y install sysstat
```

Set the storage items (data, log, journal) to be owned by the user (`mongod`) and group (`mongod`) that MongoDB will be starting under:

```
$ sudo chown mongod:mongod /data
$ sudo chown mongod:mongod /log
$ sudo chown mongod:mongod /journal
```

Now edit the MongoDB configuration file and update the following parameters

```
$ sudo nano /etc/mongod.conf
...
logpath=/log/mongod.log
logappend=true
fork=true
dbpath=/data
...
```

Starting MongoDB

Set the MongoDB service to start at boot and activate it:

```
$ sudo chkconfig mongod on
$ sudo /etc/init.d/mongod start
```

When starting for the first time, it will take a couple of minutes for MongoDB to start, setup it's storage and become available. Once it is, you should be able to connect to it from within your instance:

```
$ mongo
MongoDB shell version: 2.0.4
connecting to: test
>
```

Just to confirm the system is working correctly, try creating a test database, test collection and save a document:

```
> use testdb
switched to db testdb
> db.createCollection("testCollection")
{ "ok" : 1 }
> db.testCollection.save({"name":"bob"})
> db.testCollection.find()
{ "_id" : ObjectId("4edfda1c86176ab8e27ee976"), "name" : "bob" }
```

Now that we've got a single node up and running with EBS backed RAID storage, let's move on and create a multi-node replica set.

Deploying a Multi-node Replica Set

Replica Set Background

Replica sets are a form of asynchronous master/slave replication, adding automatic failover and automatic recovery of member nodes. A replica set consists of two or more nodes that are copies of each other (i.e.: replicas). Refer to the MongoDB Replica Set documentation for more information.

Create and Configure Instances

For this guide, we'll set up a three node replica set. To set up each node use the instructions from Deploying a Single Node above. Once that's completed, we'll update the configurations for each node and get the replica set started.

First we'll need to edit the MongoDB configuration and update the `replSet` parameter:

```
$ sudo nano /etc/mongod.conf
...
replSet=exampleReplicaSetName
...
```

Save the configuration file and restart `mongod`:

```
$ sudo /etc/init.d/mongod restart
```

Configure Replica Set

Once MongoDB has started and is running on each node, we'll need to connect to the desired primary node, initiate the replica set and add the other nodes. First connect to the desired primary via SSH and then start mongo to initiate the set:

```
$ mongo
MongoDB shell version: 2.0.4
connecting to: test
> rs.initiate()
{
    "info2" : "no configuration explicitly specified -- making one",
    "me" : "ip-10-127-127-91:27017",
    "info" : "Config now saved locally.  Should come online in about a minute.",
    "ok" : 1
}
```

Next, add the other nodes to the replica set:

```
> rs.add("ec2-107-21-46-145.compute-1.amazonaws.com")
{ "ok" : 1 }
PRIMARY> rs.add("ec2-108-90-58-191.compute-1.amazonaws.com")
{ "ok" : 1 }
PRIMARY>
```

The 3 node replica set is now configured. You can confirm the setup by checking the health of the replica set:

```
PRIMARY> rs.status()
{
    "set" : "exampleReplicaSetName",
    "date" : "Tue Dec 06 2011 11:39:08 GMT-0500 (CDT)",
    "myState" : 1,
    "members" : [
        {
            "name" : "ip-10-127-127-91:27017",
            "self" : true,
        },
        {
            "name" : "ec2-107-21-46-145.compute-1.amazonaws.com:27017",
            "health" : 1,
            "uptime" : 101,
            "lastHeartbeat" : "Tue Dec 06 2011 11:39:07 GMT-0500",
        },
        {
            "name" : "ec2-108-90-58-191.compute-1.amazonaws.com:27017",
            "health" : 1,
            "uptime" : 107,
            "lastHeartbeat" : "Tue Dec 06 2011 11:39:07 GMT-0500",
        }
    ],
    "ok" : 1
}
```

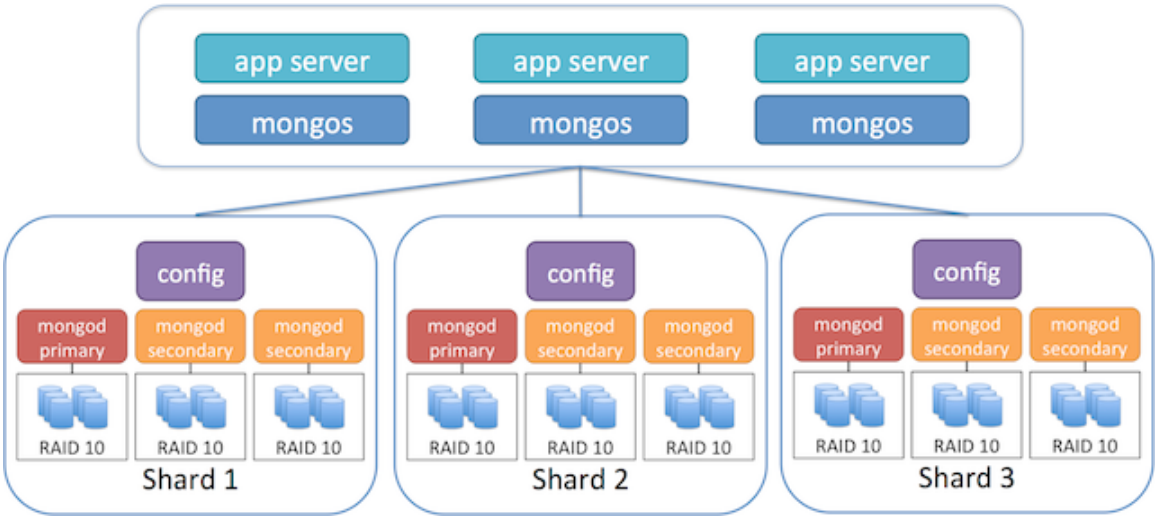
What we've completed here is a simple replica set; there are additional configurations out there that may make more sense for your deployment, refer to the MongoDB documentation for more information. If you intend to use your replica set to help scale read capacity, you'll also need to update your application's code and add the appropriate `slaveOk=true` where necessary so that read results can be returned from additional nodes more quickly.

Deploying a Sharded Configuration

MongoDB scales horizontally via a partitioned data approach known as sharding. MongoDB provides the ability to automatically balance and distribute data across multiple partitions to support write scalability. For more information, refer to the MongoDB sharding docs.

Simple Sharding Architecture

To build our simple sharded configuration, we'll be building upon the replica set steps we just worked on. To get started you'll need to create two additional replica set configurations, just the same from above. When configuring each server instance we'll set the `shardsvr` parameter inside the `mongod` configuration file. Next we'll take one node from each replica set and set it to run as a config server as well. The config server maintains metadata about the sharded data storage cluster. Finally, we'll add instances for the `mongos` router, which handles routing requests from your app to the correct shard. The recommended approach is to run this component on your application servers. The following image shows a recommended topology for use with sharding:



Use the instructions from Deploying a Single Node above to create the required nodes for each replica set (3 instances per set, for a total of 9 instances). Now start configuring the replica set but before saving `/etc/mongod.conf`, add this parameter:

```
shardsvr = true
```

Save the configuration file and restart `mongod`:

```
$ sudo /etc/init.d/mongod restart
```

Once `/etc/mongod.conf` has been updated, initiate the replica set and add the members as described in Configure Replica Set. Once that's complete, choose one instance from each replica set and start an additional `mongod` process those instances, this time as the config server component:

```
$ mongod --configsvr
```

Now that we've got N config servers running (where N is the number of running replica sets) we can set up the request router `mongos`. This process typically runs on your application servers and handles routing database requests from your app to the correct database shard. Assuming you already have your application configured and deployed, use `ssh` to connect to each instance and install MongoDB using the steps from Install and Configure MongoDB.

Before we continue, it is important to consider the role DNS plays in a sharding setup. Generally we recommend using DNS hostnames for configuring replica sets, which Amazon handles appropriately, as opposed to using specific IP addresses. Essentially, AWS knows the mapping between public and private addresses and hostnames and manages inter-region domain name resolution. Therefore, by using the public DNS name for our servers we can ensure that whether our servers are in a single region or across multiple regions, AWS will correctly route our requests. When it comes to setting up sharding, we recommend an additional step of using DNS aliases for the instances that will be acting as config servers. The routers must know the hostnames of the config servers so by using DNS aliases we gain additional flexibility if config servers ever need to change. All it takes is pointing the DNS alias to another instance and no additional update to the router configuration is needed. For more information on this topic, refer to docs on changing config servers.

Once the DNS settings have taken effect, we can proceed with the `mongos` setup. Go back to the command line on each of the instances you'll be using for `mongos` and start the service and point it to the instances running the config server using their DNS aliases (ex: `alias1.yourdomain.com`, `alias2.yourdomain.com` and `alias3.yourdomain.com`) along with the config server port `27019`:

```
$ mongos --configdb
alias1.yourdomain.com:27019,alias2.yourdomain.com:27019,alias3.yourdomain.com:27019
```

With the `mongos` routers running, we can now complete the setup for the sharding architecture. The last step is to add the previously created replica sets to the overall system. Choose one of the instances that is running `mongos`, start the `mongo` client using the hostname and port (`27017`) and connect to the `admin` database. You'll need to have the name for each replica set (ex: `replicaSetName1`) and the hostnames for each member of the set (e.g: `replicaSetHost1`)

```
$ mongo host-running-mongos:27017/admin
MongoDB shell version: 2.0.4
connecting to: admin
> db.adminCommand({addShard:
"replicaSetName1/replicaSetHost1:27018,replicaSetHost2:27018,replicaSetHost3:27018"})
```

The `addShard` command will need to be repeated for each replica set that is part of the sharded setup:

```
> db.adminCommand({addShard:
"replicaSetName2/replicaSetHost4:27018,replicaSetHost5:27018,replicaSetHost6:27018"})
> db.adminCommand({addShard:
"replicaSetName3/replicaSetHost7:27018,replicaSetHost8:27018,replicaSetHost9:27018"})
```

Once these steps have been completed, you'll have a simple sharded configuration. The architecture we used includes 3 database shards for write scalability and three replicas within each shard for read scalability and failover. This type of setup deployed across multiple regions (ex: one node from each replica located in `us-west-1`) would also provide some degree of disaster recovery as well.

In order to utilize this newly created configuration, you'll need to specify which databases and which collections are to be sharded. See Enabling Sharding on a Database and Sharding a Collection for more information.

Backup and Restore

There are several ways to backup your data when using AWS, refer to the EC2 Backup & Restore guide for more information.

EC2 Backup & Restore

Overview

This article describes how to backup, verify & restore a MongoDB running on EC2 using EBS Snapshots.

Backup

How you backup MongoDB will depend on whether you are using the --journal option in 1.8 (or above) or not.

Backup with --journal

The journal file allows for roll forward recovery. The journal files are located in the dbpath directory so will be snapshotted at the same time as the database files.

If the dbpath is mapped to a single EBS volume then proceed to the Backup the Database Files section.

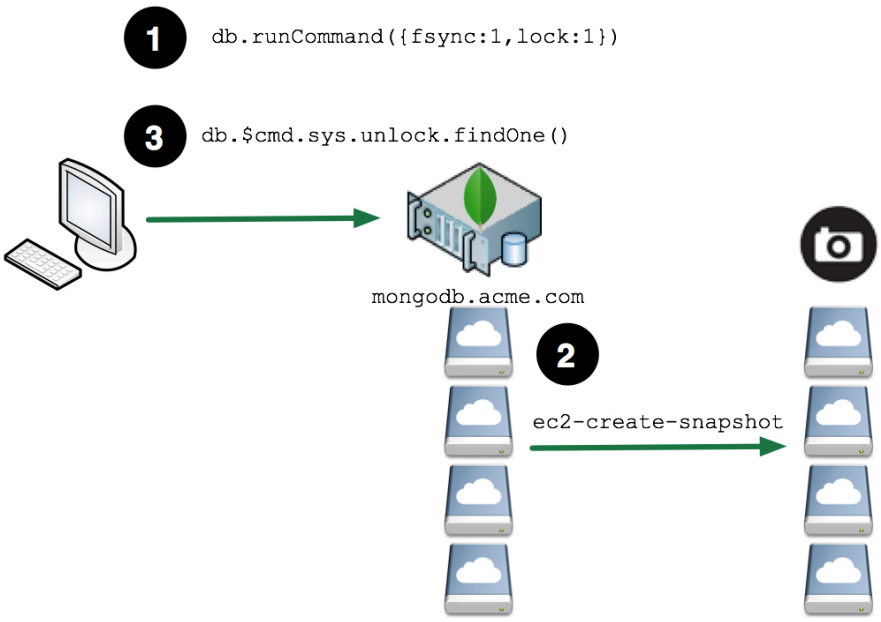
If you dbpath is mapped to multiple EBS volumes, in order to guarantee the stability of the file-system then you will need to Flush and Lock the Database section.

Note that snapshotting with the journal is only possible if the journal resides on the same volume as the data files, so that one snapshot operation captures the journal state and data file state atomically.

Backup without --journal

In order to correctly backup a MongoDB, you need to ensure that writes are suspended to the file-system before you backup the file-system. If writes are not suspended then the backup may contain partially written or data which may fail to restore correctly.

Backing up MongoDB is simple to achieve using the fsync + lock command. If the file-system is being used only by the database, then you can then use the snapshot facility of EBS volumes to create a backup. If you are using the volume for any other application then you will need to ensure that the file-system is frozen as well (e.g. on XFS file-system use xfs_freeze) before you initiate the EBS snapshot. The overall process looks like:



Flush and Lock the database

Writes have to be suspended to the file-system in order to make a stable copy of the database files. This can be achieved through the MongoDB shell using the fsync + lock command.

```
mongo shell> use admin
mongo shell> db.runCommand({fsync:1,lock:1});
{
 "info" : "now locked against writes, use db.$cmd.sys.unlock.findOne() to unlock",
 "ok" : 1
}
```

During the time the database is locked, any write requests that this database receives will be rejected. Any application code will need to deal with these errors appropriately.

Backup the database files

There are several ways to create a EBS Snapshot, for example with Elastic Fox or the AWS Command line. The following examples use the AWS command line tool.

Find the EBS volumes associated with the MongoDB

If the mapping of EBS Block devices to the MongoDB data volumes is already known, then this step can be skipped. The example below shows how to determine the mapping for an LVM volume, please confirm with your System Administrator how the original system was setup if you are unclear.

Find the EBS block devices associated with the running instance

```
shell> ec2-describe-instances
RESERVATION r-eb09aa81 289727918005 tokyo,default
INSTANCE i-78803e15 ami-4b4ba522 ec2-50-16-30-250.compute-1.amazonaws.com
ip-10-204-215-62.ec2.internal running scaleout 0  m1.large 2010-11-04T02:15:34+0000 us-east-1a
aki-0b4aa462   monitoring-disabled 50.16.30.250 10.204.215.62   ebs paravirtual
BLOCKDEVICE /dev/sda1 vol-6ce9f105 2010-11-04T02:15:43.000Z
BLOCKDEVICE /dev/sdf vol-96e8f0ff 2010-11-04T02:15:43.000Z
BLOCKDEVICE /dev/sdh vol-90e8f0f9 2010-11-04T02:15:43.000Z
BLOCKDEVICE /dev/sdg vol-68e9f101 2010-11-04T02:15:43.000Z
BLOCKDEVICE /dev/sdi vol-94e8f0fd 2010-11-04T02:15:43.000Z
```

As can be seen in this example, there are a number of block devices associated with this instance. We have to determine which volumes make up the file-system we need to snapshot.

Determining how the dbpath is mapped to the file-system

Log onto the running MongoDB instance in EC2. To determine where the database file are located, either look at the startup parameters for the mongod process or if mongod is running, then you can examine the running process.

```
root> ps -ef | grep mongo
ubuntu   10542     1  0 02:17 ?        00:00:00 /var/opt/mongodb/current/bin/mongod --port 27000
--shardsvr --dbpath /var/lib/mongodb/tokyo0 --fork --logpath /var/opt/mongodb/log/server.log
--logappend --rest
```

dbpath is set to /var/lib/mongodb/tokyo0 in this example.

Mapping the dbpath to the physical devices

Using the df command, determine what the --dbpath directory is mapped to

```
root> df /var/lib/mongodb/tokyo0
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/data_vg-data_vol
                     104802308      4320 104797988   1% /var/lib/mongodb
```

Next determine the logical volume associated with this device, in the example above /dev/mapper/data_vg-data_vol

```
root> lvdisplay /dev/mapper/data_vg-data_vol
  --- Logical volume ---
  LV Name                /dev/data_vg/data_vol
  VG Name                data_vg
  LV UUID                fixOyX-6Aiw-PnBA-i2bp-ovUc-u9uu-TGvjxl
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                100.00 GiB
...
```

This output indicates the volume group associated with this logical volume, in this example data_vg. Next determine how this maps to the physical volume.

```
root> pvscan
  PV /dev/md0   VG data_vg   lvm2 [100.00 GiB / 0    free]
  Total: 1 [100.00 GiB] / in use: 1 [100.00 GiB] / in no VG: 0 [0   ]
```

From the physical volume, determine the associated physical devices, in this example /dev/md0.

```
root> mdadm --detail /dev/md0
/dev/md0:
        Version : 00.90
  Creation Time : Thu Nov  4 02:17:11 2010
     Raid Level : raid10
     Array Size : 104857472 (100.00 GiB 107.37 GB)
  Used Dev Size : 52428736 (50.00 GiB 53.69 GB)
   Raid Devices : 4
...
           UUID : 07552c4d:6c11c875:e5a1de64:a9c2f2fc (local to host ip-10-204-215-62)
         Events : 0.19

    Number   Major   Minor   RaidDevice State
       0       8       80        0      active sync   /dev/sdf
       1       8       96        1      active sync   /dev/sdg
       2       8      112        2      active sync   /dev/sdh
       3       8      128        3      active sync   /dev/sdi
```

We can see that block devices /dev/sdf through /dev/sdi make up this physical devices. Each of these volumes will need to be snapped in order to complete the backup of the file-system.

Create the EBS Snapshot

Create the snapshot for each devices. Using the ec2-create-snapshot command, use the Volume Id for the device listed by the ec2-describe-instances command.

```
shell> ec2-create-snapshot -d backup-20101103 vol-96e8f0ff
SNAPSHOT snap-417af82b vol-96e8f0ff pending 2010-11-04T05:57:29+0000  289727918005 50 backup-20101103
shell> ec2-create-snapshot -d backup-20101103 vol-90e8f0f9
SNAPSHOT snap-5b7af831 vol-90e8f0f9 pending 2010-11-04T05:57:35+0000  289727918005 50 backup-20101103
shell> ec2-create-snapshot -d backup-20101103 vol-68e9f101
SNAPSHOT snap-577af83d vol-68e9f101 pending 2010-11-04T05:57:42+0000  289727918005 50 backup-20101103
shell> ec2-create-snapshot -d backup-20101103 vol-94e8f0fd
SNAPSHOT snap-2d7af847 vol-94e8f0fd pending 2010-11-04T05:57:49+0000  289727918005 50 backup-20101103
```

Unlock the database

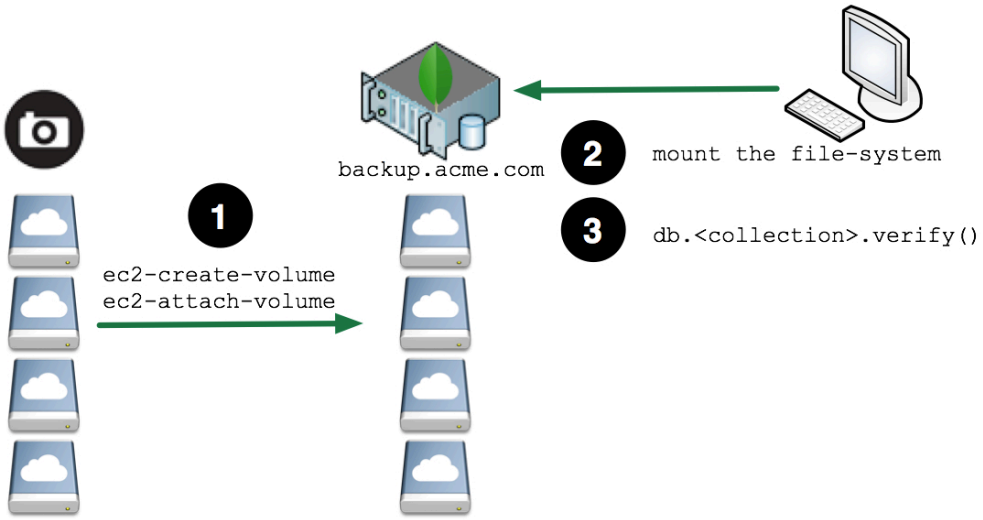
After the snapshots have been created, the database can be unlocked. After this command has been executed the database will be available to process write requests.

```
mongo shell> db.$cmd.sys.unlock.findOne();
{ "ok" : 1, "info" : "unlock requested" }
```

Verifying a backup

In order to verify the backup, the following steps need to be completes

- Check the status of the snapshot to ensure that they are "completed"
- Create new volumes based on the snapshots and mount the new volumes
- Run mongod and verify the collections



Typically, the verification will be performed on another machine so that you do not burden your production systems with the additional CPU and I/O load of the verification processing.

Describe the snapshots

Using the ec2-describe-snapshots command, find the snapshots that make up the backup. Using a filter on the *description* field, snapshots associated with the given backup are easily found. The search text used should match the text used in the *-d* flag passed to ec2-create-snapshot command when the backup was made.

```
backup shell> ec2-describe-snapshots --filter "description=backup-20101103"
SNAPSHOT snap-2d7af847 vol-94e8f0fd completed 2010-11-04T05:57:49+0000 100% 289727918005 50
backup-20101103
SNAPSHOT snap-417af82b vol-96e8f0ff completed 2010-11-04T05:57:29+0000 100% 289727918005 50
backup-20101103
SNAPSHOT snap-577af83d vol-68e9f101 completed 2010-11-04T05:57:42+0000 100% 289727918005 50
backup-20101103
SNAPSHOT snap-5b7af831 vol-90e8f0f9 completed 2010-11-04T05:57:35+0000 100% 289727918005 50
backup-20101103
```

Create new volumes based on the snapshots

Using the ec2-create-volume command, create a new volumes based on each of the snapshots that make up the backup.

```
backup shell> ec2-create-volume --availability-zone us-east-1a --snapshot snap-2d7af847
VOLUME vol-06aab26f 50 snap-2d7af847 us-east-1a creating 2010-11-04T06:44:27+0000
backup shell> ec2-create-volume --availability-zone us-east-1a --snapshot snap-417af82b
VOLUME vol-1caab275 50 snap-417af82b us-east-1a creating 2010-11-04T06:44:38+0000
backup shell> ec2-create-volume --availability-zone us-east-1a --snapshot snap-577af83d
VOLUME vol-12aab27b 50 snap-577af83d us-east-1a creating 2010-11-04T06:44:52+0000
backup shell> ec2-create-volume --availability-zone us-east-1a --snapshot snap-5b7af831
VOLUME vol-caaab2a3 50 snap-5b7af831 us-east-1a creating 2010-11-04T06:45:18+0000
```

Attach the new volumes to the instance

Using the ec2-attach-volume command, attach each volume to the instance where the backup will be verified.

```
backup shell> ec2-attach-volume --instance i-cad26ba7 --device /dev/sdp vol-06aab26f
ATTACHMENT vol-06aab26f i-cad26ba7 /dev/sdp attaching 2010-11-04T06:49:32+0000
backup shell> ec2-attach-volume --instance i-cad26ba7 --device /dev/sdq vol-1caab275
ATTACHMENT vol-1caab275 i-cad26ba7 /dev/sdq attaching 2010-11-04T06:49:58+0000
backup shell> ec2-attach-volume --instance i-cad26ba7 --device /dev/sdr vol-12aab27b
ATTACHMENT vol-12aab27b i-cad26ba7 /dev/sdr attaching 2010-11-04T06:50:13+0000
backup shell> ec2-attach-volume --instance i-cad26ba7 --device /dev/sds vol-caaab2a3
ATTACHMENT vol-caaab2a3 i-cad26ba7 /dev/sds attaching 2010-11-04T06:50:25+0000
```

Mount the volumes groups etc.

Make the file-system visible on the host O/S. This will vary by the Logical Volume Manager, file-system etc. that you are using. The example below shows how to perform this for LVM, please confirm with your System Administrator on how the original system system was setup if you are unclear.

Assemble the device from the physical devices. The UUID for the device will be the same as the original UUID that the backup was made from, and can be obtained using the *mdadm* command.

```
backup shell> mdadm --assemble --auto-update-homehost -u 07552c4d:6c11c875:e5a1de64:a9c2f2fc
--no-degraded /dev/md0
mdadm: /dev/md0 has been started with 4 drives.
```

You can confirm that the physical volumes and volume groups appear correctly to the O/S by executing the following:

```
backup shell> pvscan
  PV /dev/md0   VG data_vg   lvm2 [100.00 GiB / 0    free]
  Total: 1 [100.00 GiB] / in use: 1 [100.00 GiB] / in no VG: 0 [0   ]

backup shell> vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "data_vg" using metadata type lvm2
```

Create the mount point and mount the file-system:

```
backup shell> mkdir -p /var/lib/mongodb

backup shell> cat >> /etc/fstab << EOF
/dev/mapper/data_vg-data_vol /var/lib/mongodb xfs noatime,noexec,nodiratime 0 0
EOF

backup shell> mount /var/lib/mongodb
```

Startup the database

After the file-system has been mounted, MongoDB can be started. Ensure that the owner of the files is set to the correct user & group. Since the backup was made with the database running, the lock file will need to be removed in order to start the database.

```
backup shell> chown -R mongodb /var/lib/mongodb/toyko0
backup shell> rm /var/lib/mongodb/tokyo0/mongod.lock
backup shell> mongod --dbpath /var/lib/mongodb/tokyo0
```

Verify the collections

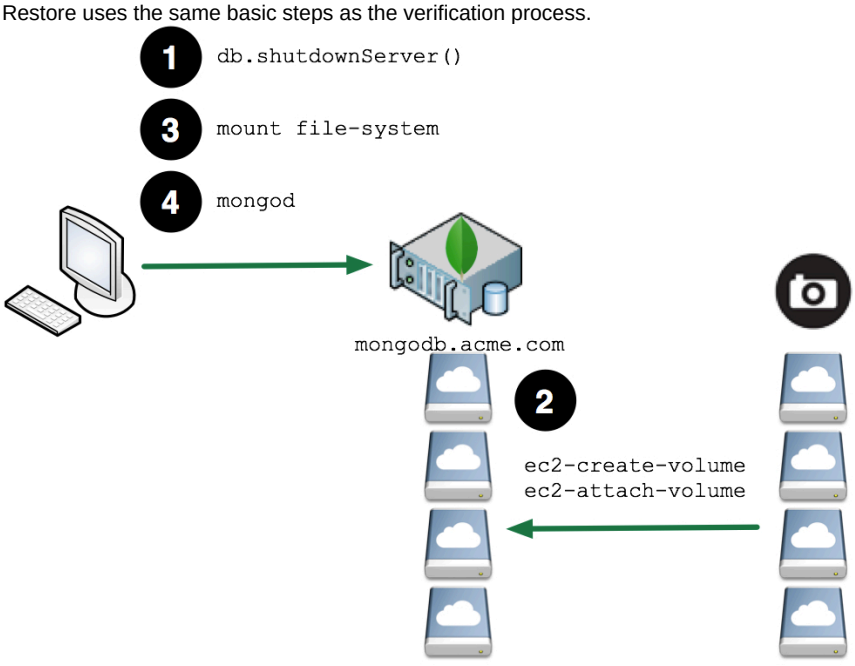
Each collection can be verified in order to ensure that it valid and does not contain any invalid BSON objects.

```
mongo shell> db.blogs.validate({full:true})
```

Validate Command

Restore

Restore uses the same basic steps as the verification process.



After the file-system is mounted you can decide to

- Copy the database files from the backup into the current database directory
- Startup mongod from the new mount point, specifying the new mount point in the --dbpath argument

After the database is started, it will be ready to transact. It will be at the specific point in time from the backup, so if it is part of a master/slave or replica set relationship, then the instance will need to synchronize itself to get itself back up to date.

dotCloud

Running MongoDB on dotCloud

MongoDB can run on dotCloud. It supports replica sets, and has alpha support for sharding.

The whole point of dotCloud is to run your apps and your databases in the same place, to optimize for latency and reliability. However, you can also deploy MongoDB on dotCloud and use it to power an app running anywhere else.

If you don't have a dotCloud account yet...

Well, what are you waiting for?

Go ahead and create one (it's free!) and install the CLI:

```
sudo easy_install pip ; sudo pip install dotcloud
```

If you need help to get the CLI running, check the dotCloud install docs and don't hesitate to ask for help.

With a dotCloud account

The following snippet will deploy MongoDB on dotCloud for you in no time:

```
mkdir mongodb-on-dotcloud
cat >mongodb-on-dotcloud/dotcloud.yml <<EOF
db:
  type: mongodb
EOF
dotcloud push mongorocks mongodb-on-dotcloud
dotcloud info mongorocks.db
```

The last command will show you the host, port, and credentials to be used to connect to your database.

Scaling

Assuming you followed the instructions of the previous section, if you want to get a replica sets of 3 servers:

```
dotcloud scale mongorocks db=3
```

Advanced use

If you want to have a closer look at your MongoDB server, nothing beats SSH access:

```
dotcloud ssh mongorocks.db
```

Moar docs

- dotCloud documentation for the MongoDB service
- generic introduction to dotCloud (in case you want to run not only MongoDB, but also Node.js, Python, Ruby, Perl, Java, RabbitMQ, Redis, MySQL, PostgreSQL, CouchDB, Riak, Erlang, or something else, on dotCloud)

Ready-to-use apps

All you need to do to run them is a `git clone` and a `dotcloud push`:

- Django setup using MongoDB to store objects
- MongoDB + Node.js sample app

Getting help

dotCloud has a Q&A site, and the dotCloud team can be reached through the FreeNode IRC network on #dotcloud.

Joyent

For the quickest start, you can use the Joyent SmartMachine for MongoDB Appliance

For installing MongoDB on a Joyent Node Smart Machine, see this article

The prebuilt MongoDB Solaris 64 binaries work with Joyent accelerators.

Some newer gcc libraries are required to run -- see sample setup session below.

```
$ # assuming a 64 bit accelerator
$ /usr/bin/isainfo -kv
64-bit amd64 kernel modules

$ # get mongodb
$ # note this is 'latest' you may want a different version
$ curl -O http://downloads.mongodb.org/sunos5/mongodb-sunos5-x86_64-latest.tgz
$ gzip -d mongodb-sunos5-x86_64-latest.tgz
$ tar -xf mongodb-sunos5-x86_64-latest.tar
$ mv "mongodb-sunos5-x86_64-2009-10-26" mongo

$ cd mongo

$ # get extra libraries we need (else you will get a libstdc++.so.6 dependency issue)
$ curl -O http://downloads.mongodb.org.s3.amazonaws.com/sunos5/mongo-extra-64.tgz
$ gzip -d mongo-extra-64.tgz
$ tar -xf mongo-extra-64.tar
$ # just as an example - you will really probably want to put these somewhere better:
$ export LD_LIBRARY_PATH=mongo-extra-64
$ bin/mongod --help
```

MongoDB on Azure - Overview

Overview

MongoDB on Azure brings the power of the leading NoSQL database to Microsoft's flexible, open, and scalable cloud.

Windows Azure is the cloud services operating system that serves as the development, service hosting, and service management environment for the Azure Services Platform. Windows Azure provides developers on-demand compute & storage to create, host and manage scalable and available web applications through Microsoft data centers.

Together, MongoDB and Azure provide customers the tools to build limitlessly scalable applications in the cloud.

Deployment Options

Users interested in deploying MongoDB on Azure can do so using Azure Worker Roles (Azure Platform-as-a-Service) or Azure VM (Azure Infrastructure-as-a-Service). For further information, please see the relevant documentation:

Azure Worker Roles (Platform-as-a-Service)

- Introduction
- Configuration
- Deployment
- Building a MongoDB Azure application

Azure VM (Infrastructure-as-a-Service)

- Introduction
- Windows Installer
- Linux Tutorial

Users deploying MongoDB on Azure may be interested in the following presentations, as well:

- MongoDB Paris 2012: MongoDB on Azure
- MongoNYC 2012: MongoDB on Windows Azure
- Hands-On: Deploying MongoDB on Azure

Deploying MongoDB Replica Sets to Linux on Azure

Redirection Notice
This page should redirect to https://wiki.10gen.com/display/DOCS/MongoDB+on+Azure+VM+-+Linux+Tutorial.

MongoDB Installer for Windows Azure

Redirection Notice
This page should redirect to http://www.mongodb.org/display/DOCS/MongoDB+on+Azure+Worker+Roles.

MongoDB on Azure

Redirection Notice
This page should redirect to http://www.mongodb.org/display/DOCS/MongoDB+on+Azure+Worker+Roles.

MongoDB on Azure VM

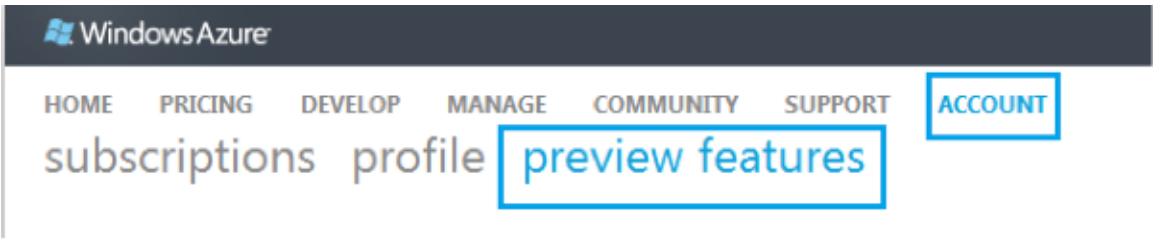
Windows Azure VMs allow you to deploy both Windows and Linux virtual machines to the Windows Azure cloud service. These VMs are similar to local instances or other IaaS services and hence can be used to run MongoDB instances.
Currently Windows Azure VMs are in preview and you need to sign up for the preview to use the feature and deploy MongoDB to it.

- Signing Up

Signing Up

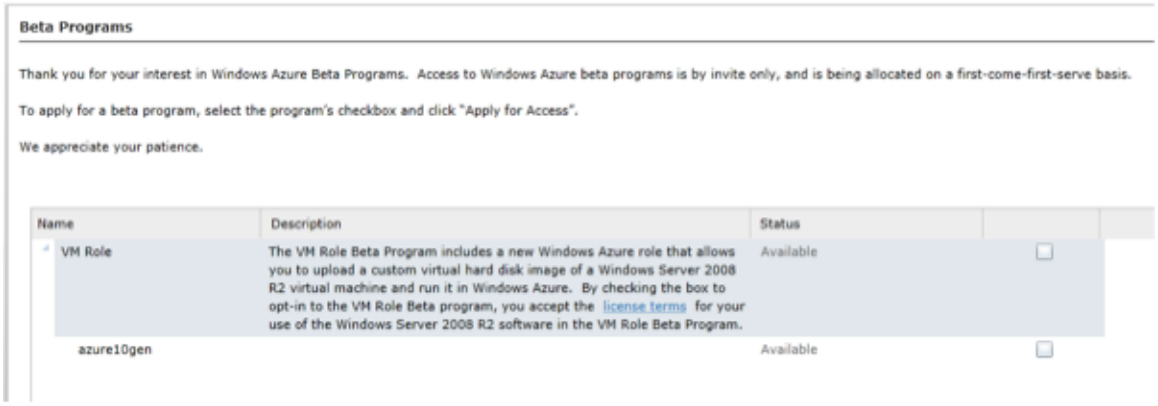
Sign up for 'Virtual Machines and Virtual Networks' preview feature from the new Azure portal*

In the 'Preview' portal, go to Account->Preview Features.



Sign up for the "Virtual Machines and Virtual Networks"

"*" – The VM Role beta program from the old Azure portal (shown below) is not the same as the preview feature above



MongoDB on Azure VM - Linux Tutorial

In this tutorial, you will learn to deploy a MongoDB replica set to CentOS VMs on Windows Azure and access it from outside Azure. The following are the tasks to achieve this:

1. Sign up for the feature on the Azure portal
2. Set up affinity group
3. Create and set up the virtual machines
4. Install and run MongoDB
5. Configure the replica set

Signing Up

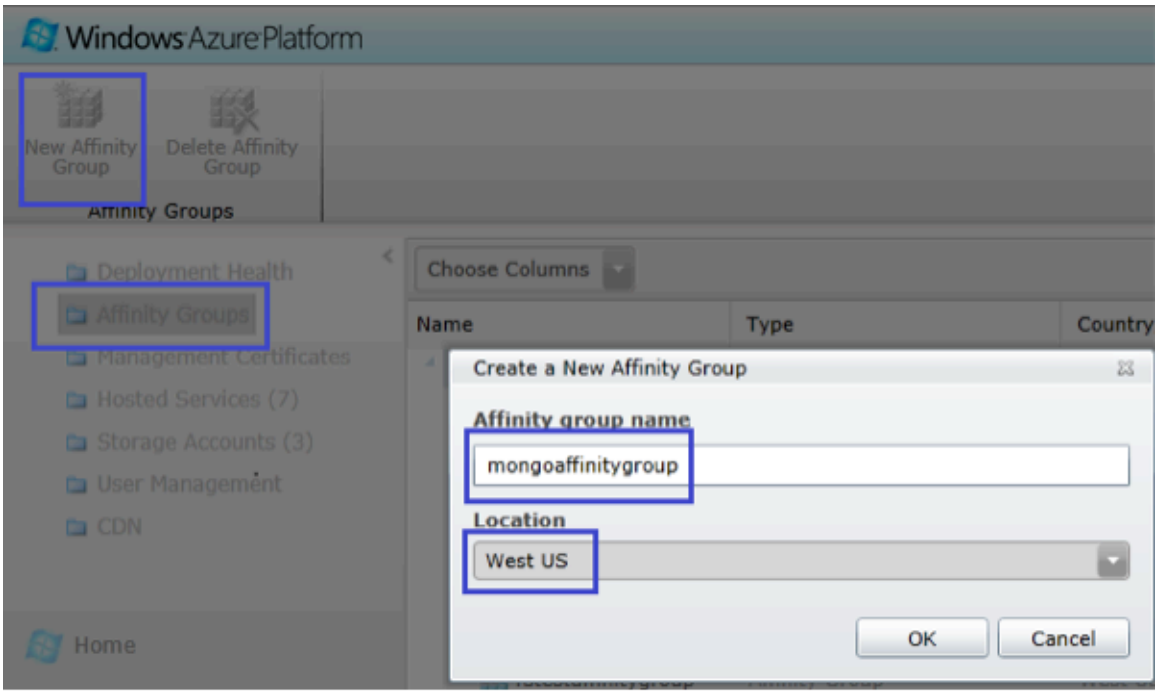
Sign up for 'Virtual Machines and Virtual Networks' preview feature from the new Azure portal*. More information can be found here.

Set Up Affinity Group

The affinity group needs to be set up from the old portal.

1. If in the new Azure Management preview portal, click you user name in the top right of the screen, then 'Previous Portal' to switch to the old portal.

2. Once in the old portal, click 'Hosted Services, Storage Accounts & CDN'
3. Select 'Affinity Groups' and then 'New Affinity Group'
4. In the "Create a New Affinity Group" dialog, enter an affinity group name such as "mongoaffinitygroup" and choose a location
    - Choose one of the newer Azure data centers such as "West US," "East US" or "West Europe"



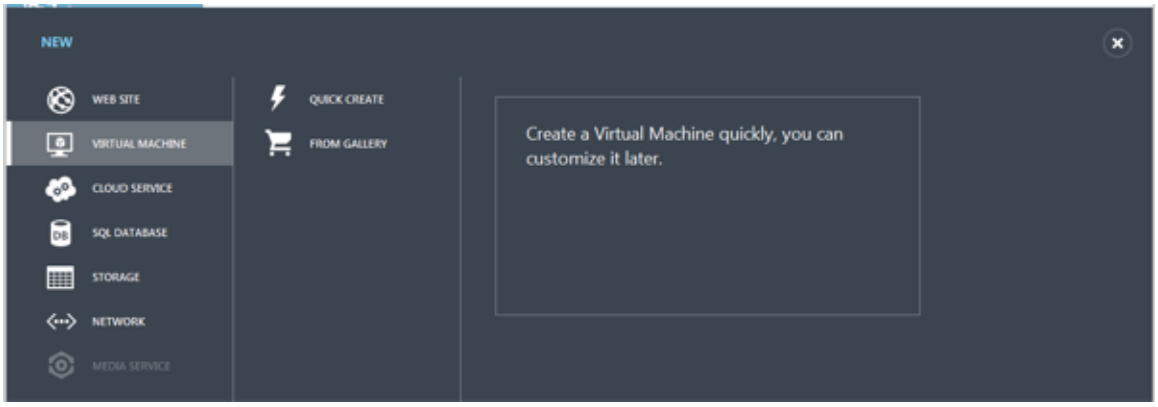
Create and Set Up the Virtual Machines

To create a 3-node MongoDB replica set you will be creating 3 VM instances. For this step, log in to the new preview portal, or from the old portal, click ‘Visit the Previous Portal’ at the bottom of the page

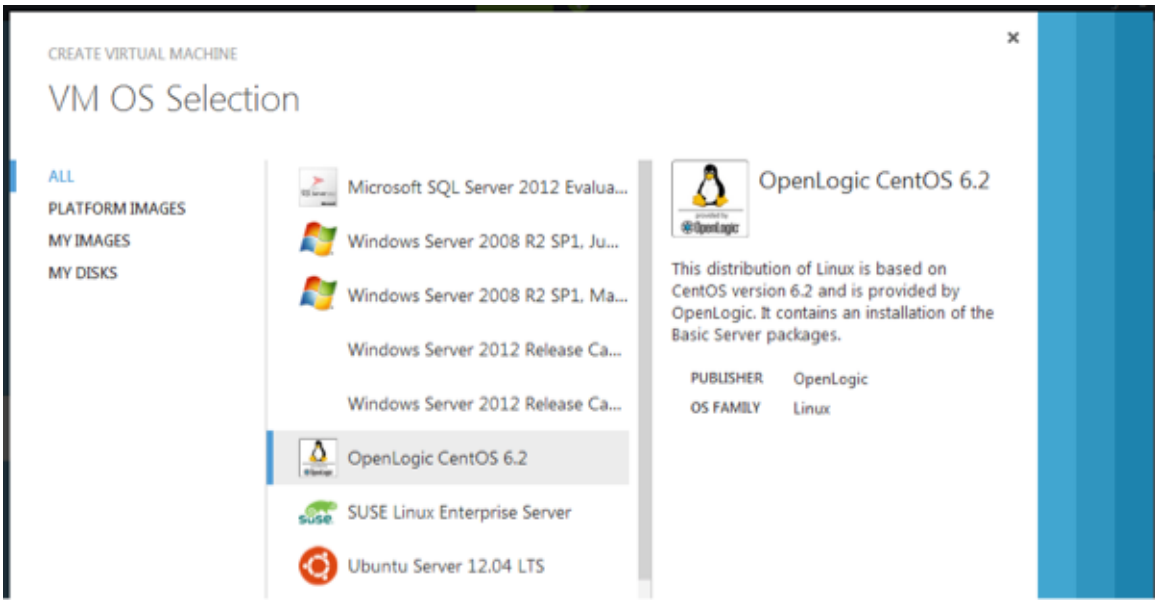
Create Instances

Instance 1

1. In the portal, click New->Virtual Machine->From Gallery

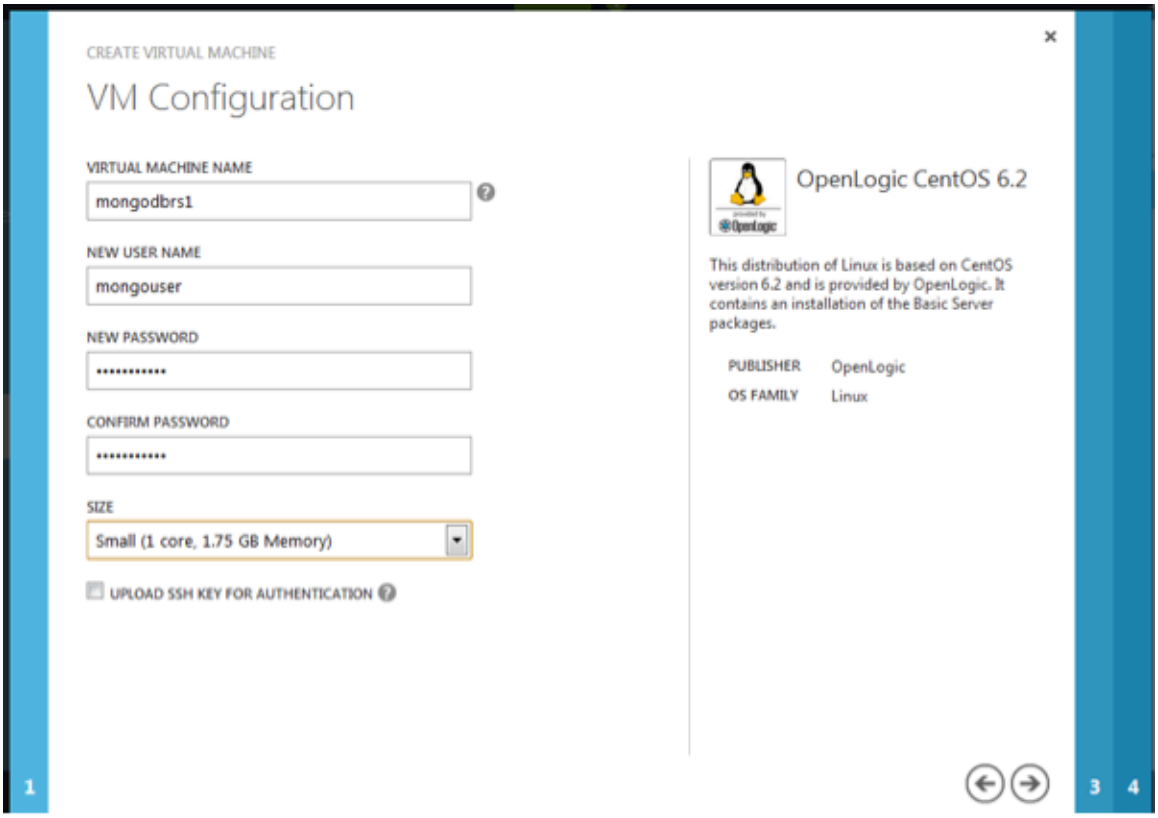


2. On the VM OS Selection page, choose ‘OpenLogic CentOS 6.2’ and click the next arrow to continue



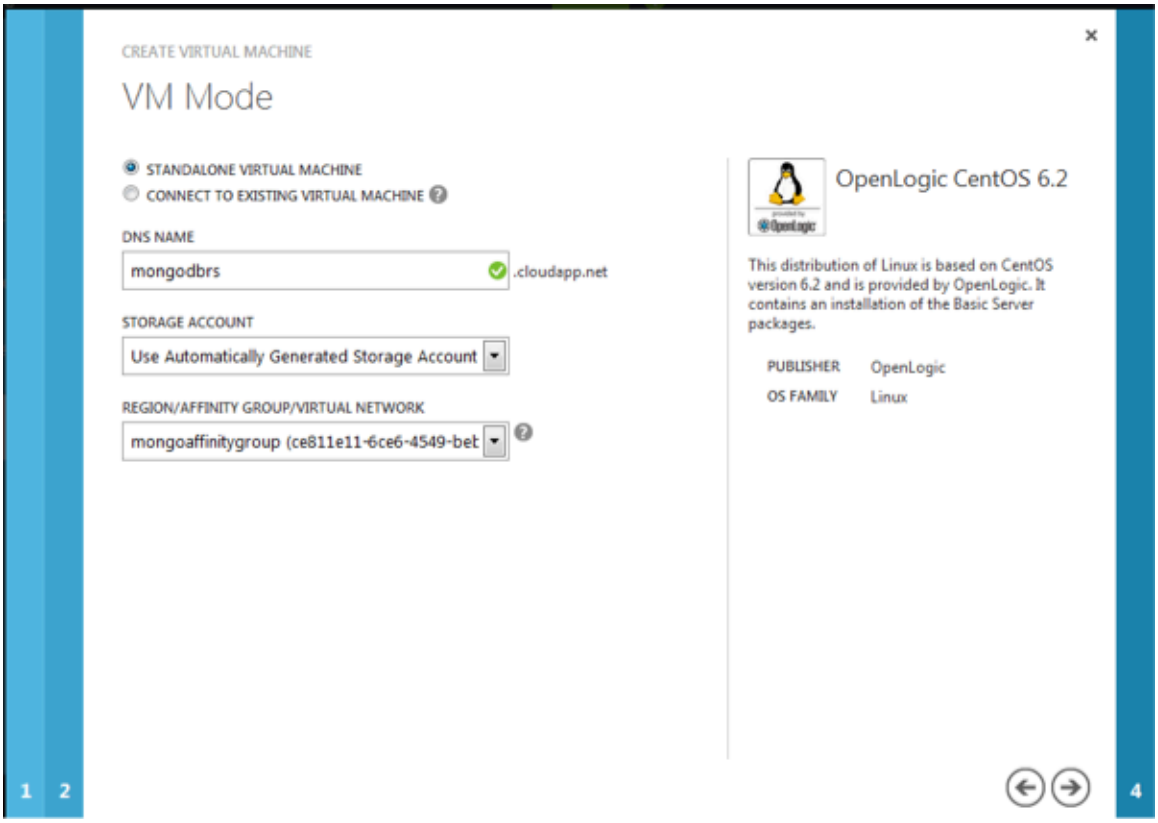
3. On the VM Configuration page, specify values for the parameters
   - Virtual Machine Name - "mongodbrs1"
   - New User Name – "mongouser." This user will be added to the ‘sudoers’ list file
   - New Password box – type a strong password that conforms to the Azure specifications
   - Retype the password to confirm
   - Size – Choose appropriate size from drop down list. For anything but small test instances, choose a size larger than Medium
   - Leave ‘Upload SSH key for Authentication’ unchecked

   Click the next arrow to continue

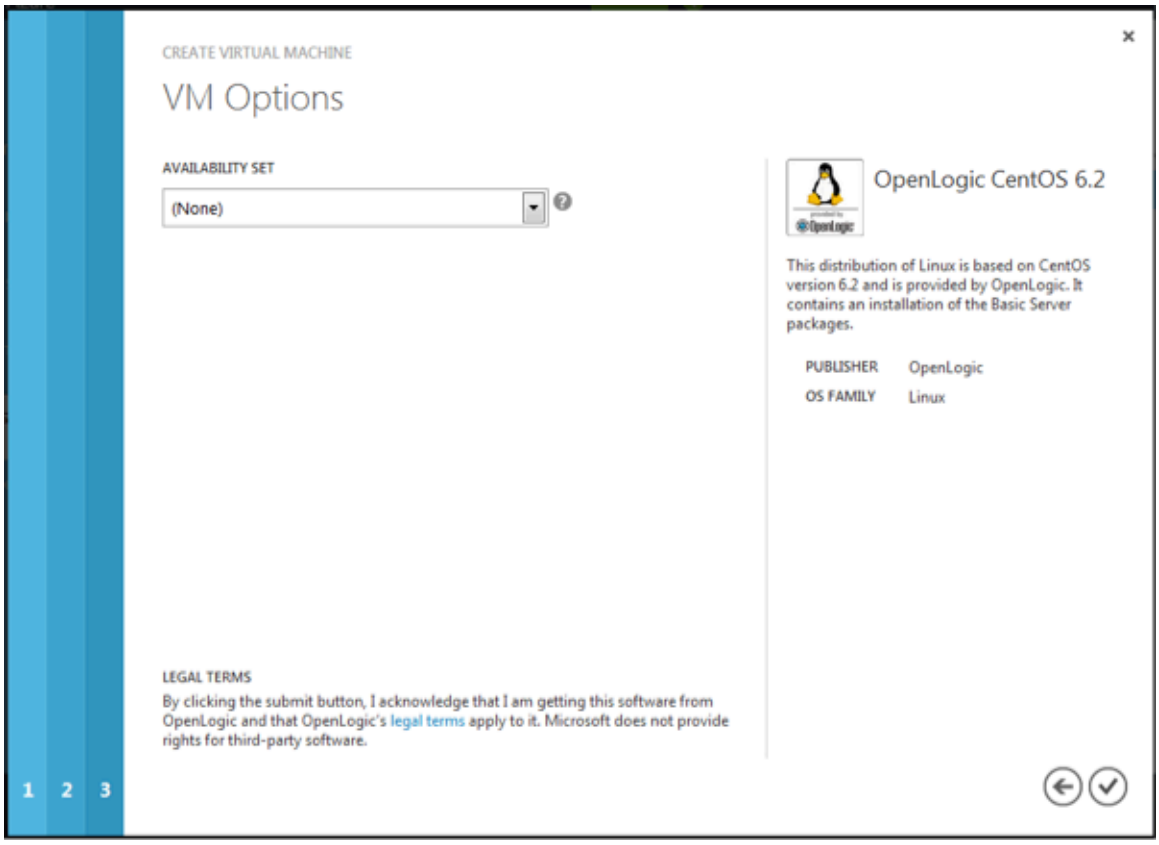


4. On the VM Mode page, specify values for the required parameters
   - Select ‘Standalone Virtual Machine’
   - DNS – a valid DNS prefix e.g., "mongodbrs"
   - Storage Account box – choose ‘Use Automatically Generated Storage Account’
   - In the ‘Region/Affinity Group/Virtual Network’ box, select the affinity group created previously “mongoaffinitygroup”

Click next arrow to continue



5. On the VM Options page select ‘None for Availability Set’



6. Click the check mark to create the VM instance

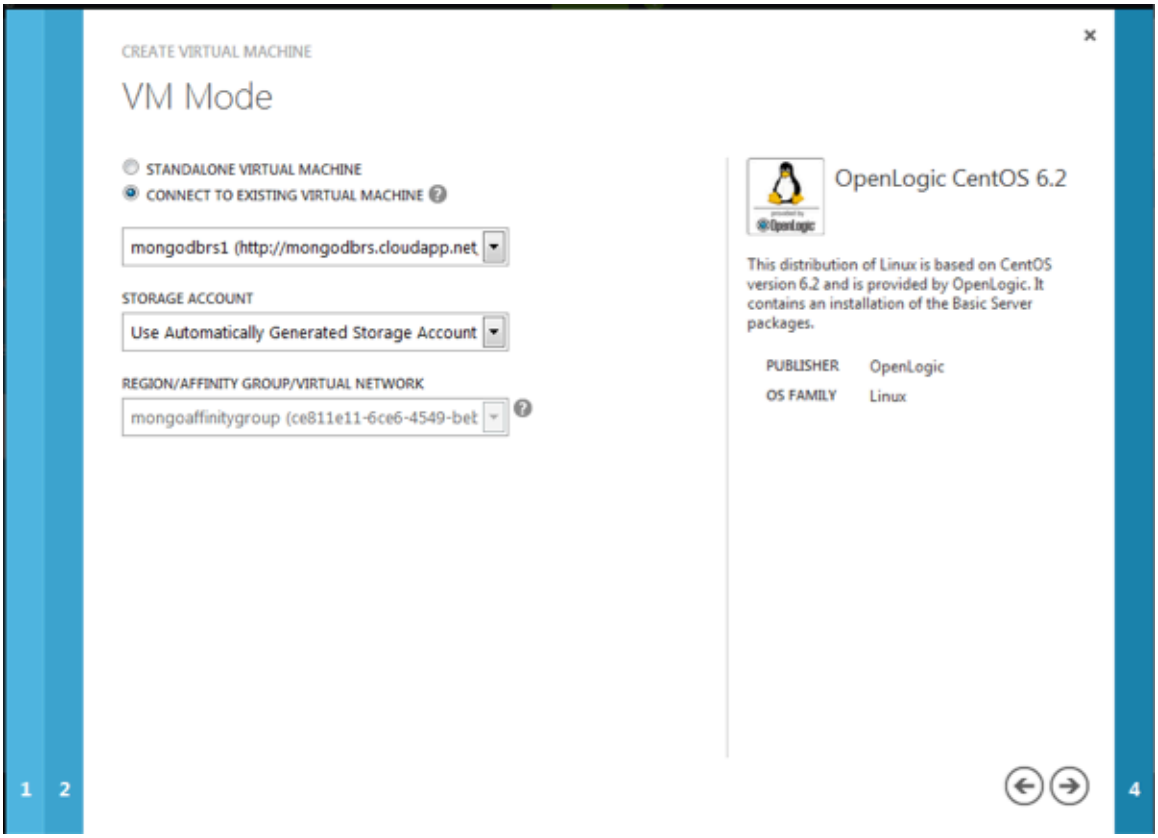
Instance 2

A similar process to creating instance 1.

1. In the portal, click New->Virtual Machine->From Gallery
2. On the VM OS Selection page, choose ‘OpenLogic CentOS 6.2’ and click the next arrow to continue
3. On the VM Configuration page, specify values for the parameters
    - Virtual Machine Name - "mongodbrs2"
    - New User Name – "mongouser." This user will be added to the ‘sudoers’ list file
    - New Password box – type a strong password that conforms to the Azure specifications
    - Retype the password to confirm
    - Size – Choose appropriate size from drop down list. For anything but small test instances, choose a size larger than Medium
    - Leave ‘Upload SSH key for Authentication’ unchecked

    Click the next arrow to continue
4. On the VM Mode page, specify values for the required parameters
    - Select ‘Connect to Existing Virtual Machine’
    - In the dropdown choose the host created in instance 1, in this case ‘mongodbrs1’
    - Storage Account box – choose ‘Use Automatically Generated Storage Account’
    - The ‘Region/Affinity Group/Virtual Network’ box should be automatically set to “mongoaffinitygroup”

    Click next arrow to continue



5. On the VM Options page select ‘None for Availability Set’
6. Click the check mark to create the VM instance

Instance 3

A similar process for creating instance 2. Choose ‘mongodbrs3’ to be the Virtual Machine Name in step 3.

Configure Endpoints

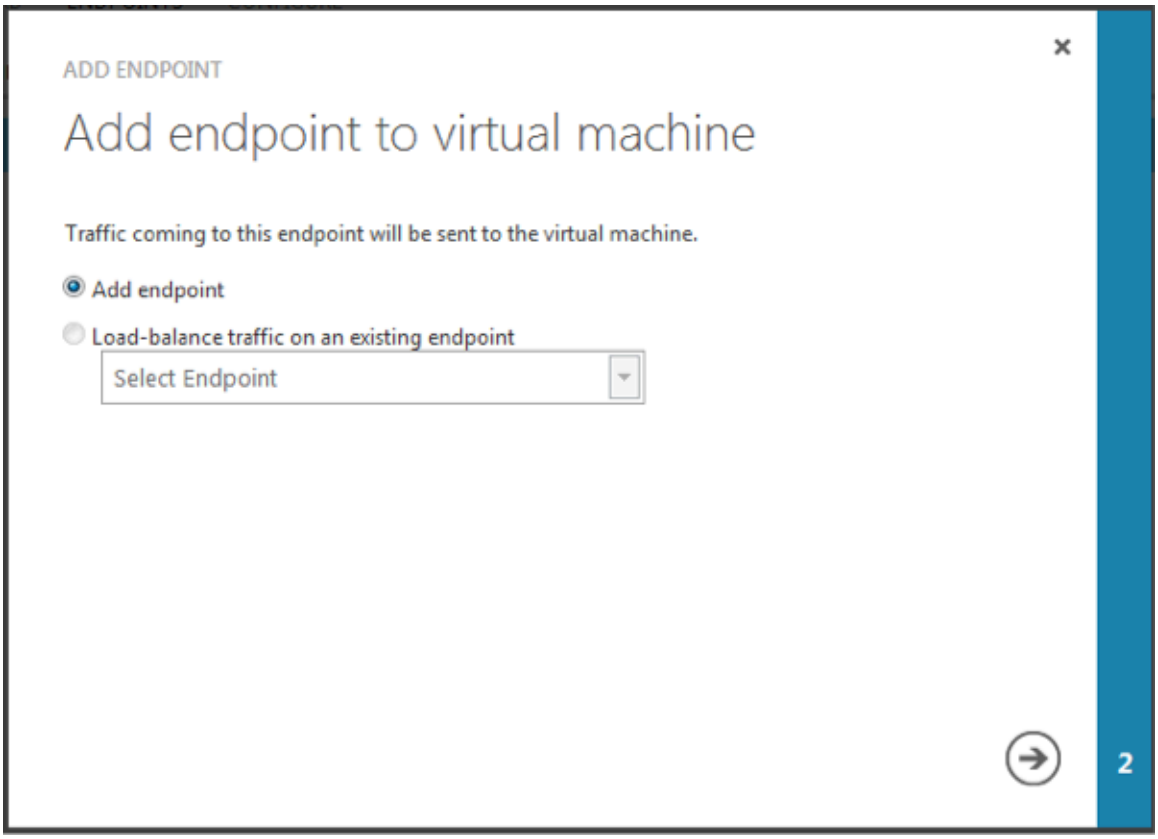
Once the virtual machines are connected you need to configure the endpoints to:

- Allow remote connection
- Allow mongo traffic

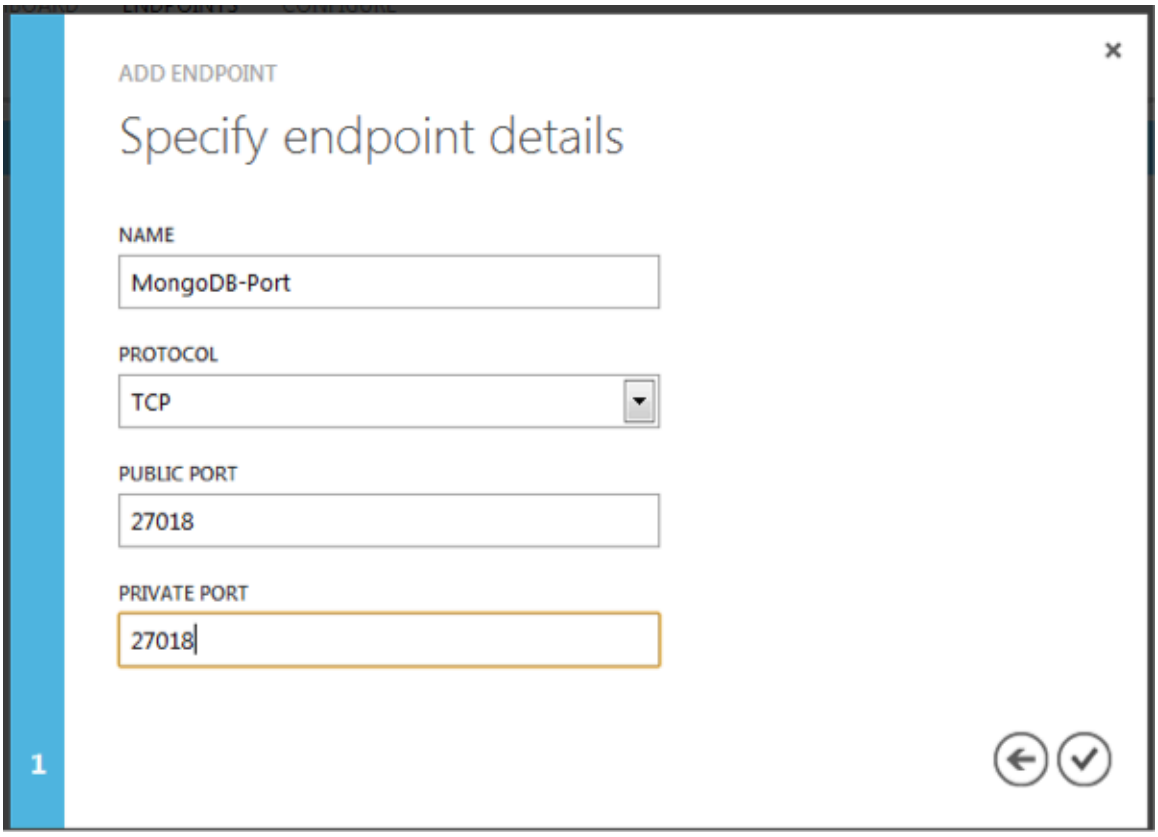
Instance 1:

1. In the management portal, click virtual machines and click the name of instance 1, ‘mongodbrs1’

2. Now click on endpoints
3. The ssh endpoint should be automatically created. For this endpoint ensure the following are set
    - Protocol – tcp
    - Public Port – 22
    - Private Port – 22
    - Load Balanced – No
4. Create a new endpoint for MongoDB by clicking on ‘Add Endpoint’ at the bottom of the screen
5. Ensure ‘Add endpoint’ is selected and click the right arrow to go to the next screen



6. Specify the endpoint details as below:
    - Name : MongoDB-Port
    - Protocol : TCP
    - Public Port : 27018
    - Private Port : 27018
    - Click on check mark to create endpoint



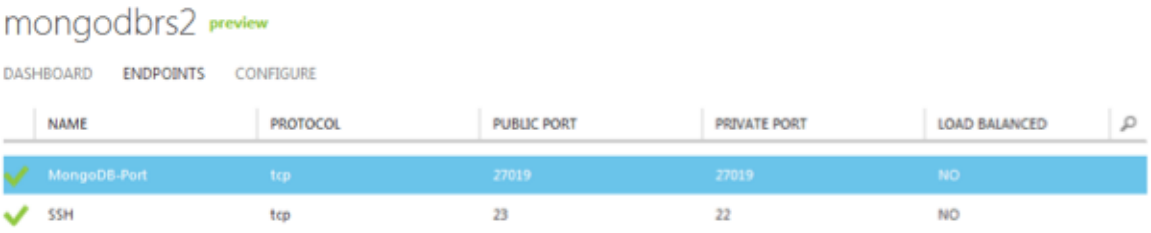
The instance now should have 2 endpoints, 1 for SSH and 1 for MongoDB

Instance 2:

We need to configure the endpoints for instance 2 similar to instance 1:

1. In the management portal, click ‘virtual machines’ and click the name of instance 2
2. Now click on ‘endpoints’
3. The ssh endpoint should be automatically created. Ensure that:
    - Name – SSH
    - Protocol – TCP
    - Private Port – 22
    - Load Balanced – No
4. Now click on Edit Endpoint at the bottom of the screen and set ‘Public Port’ to 23. Click on the ‘check mark’ to update
5. Create a new endpoint for MongoDB by clicking on ‘Add Endpoint’ at the bottom of the screen
6. Ensure ‘Add Endpoint’ is selected and click the right arrow to go to the next screen
7. Specify the endpoint details as below:
    - Name : MongoDB-Port
    - Protocol : TCP
    - Public Port : 27019
    - Private Port : 27019
    - Click on check mark to create endpoint

The instance now should have 2 endpoints, 1 for SSH and 1 for MongoDB



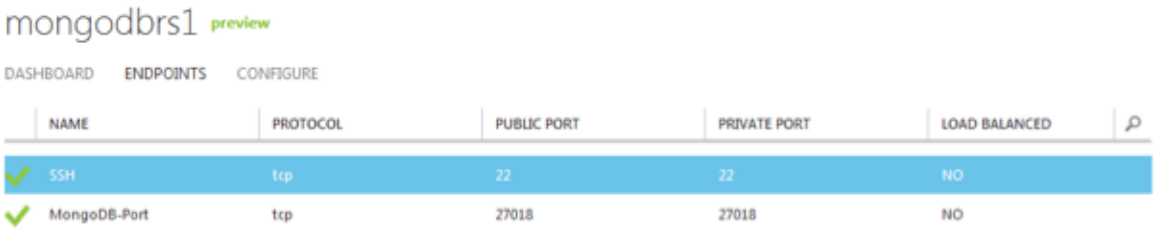
Instance 3:

Create endpoints for instance 3 similar to instance 2 with the following changes:

1. In step 4, set public port to 24

2. In step 7, set public and private ports to be 27020

The instance now should have 2 endpoints, 1 for SSH and 1 for MongoDB



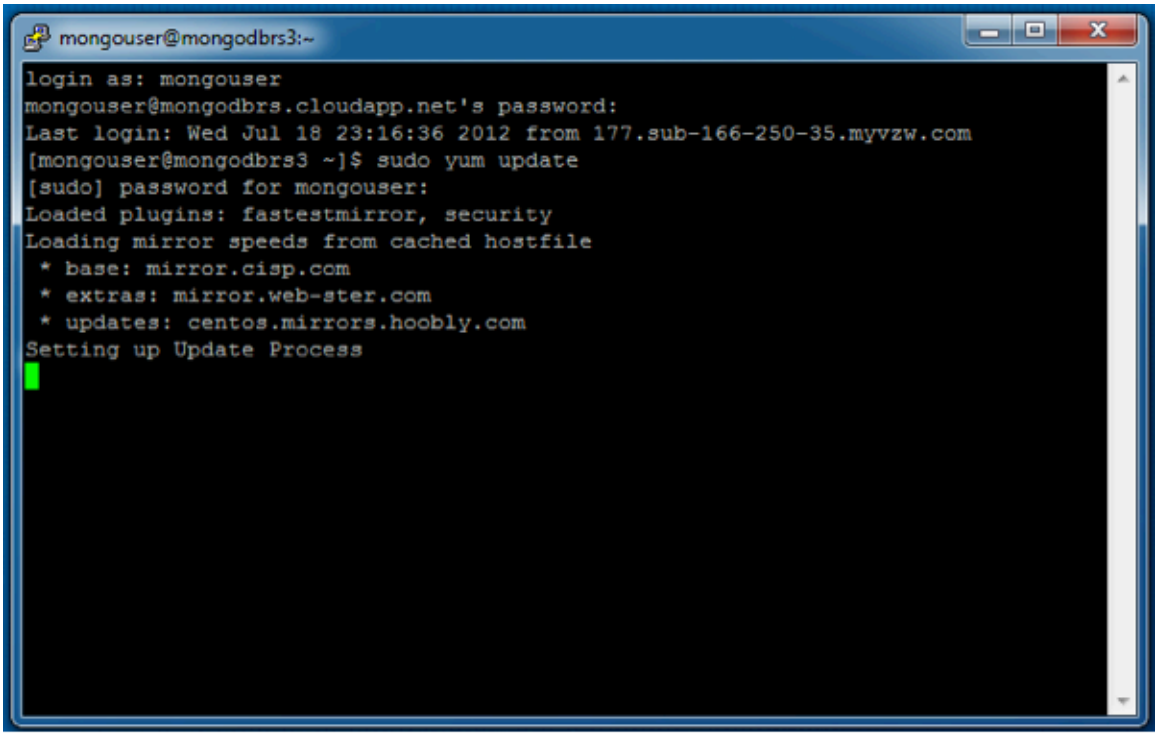
Update OS

Use this optional step to update the operating system on each of your VM instances. Once the machine endpoints have been configured above, you need to log into the machines to update them. More information on this can be found at 'How to Log on to a Virtual Machine Running Linux'

Use the username and password you used when creating the virtual machine instances.

Once you are connected to the machine, update the operating system by running: `sudo yum update` and following the prompts. This could take some time.

Note: When connecting to instances 2 and 3, remember to use ports 23 and 24 and not the default ssh port of 22.



Set up disks

Once the instances are updated, you can then attach a data disk to each of the instances. The data disks will be storing the actual mongodb data as part of --dbpath. More information on Azure data disks can be found at Data Disk Concepts.

To set up the data disk follow the steps outlined below for each of the instances you created:

1. Attach an empty disk to the instance as described in How to: Attach an empty disk
    a. Create a data disk of at least 10 GB
2. Now initialize the data disk by following the steps described at How to: Initialize a new data disk in Linux
3. Also once mounted, create a mongodb data directory by:
    a. Log on onto the instance
    b. Then run *sudo chown `id -u` /mnt/datadrive/* to make the mongouser the owner of the data directory
    c. Run the following command: *mkdir -p /mnt/datadrive/data*

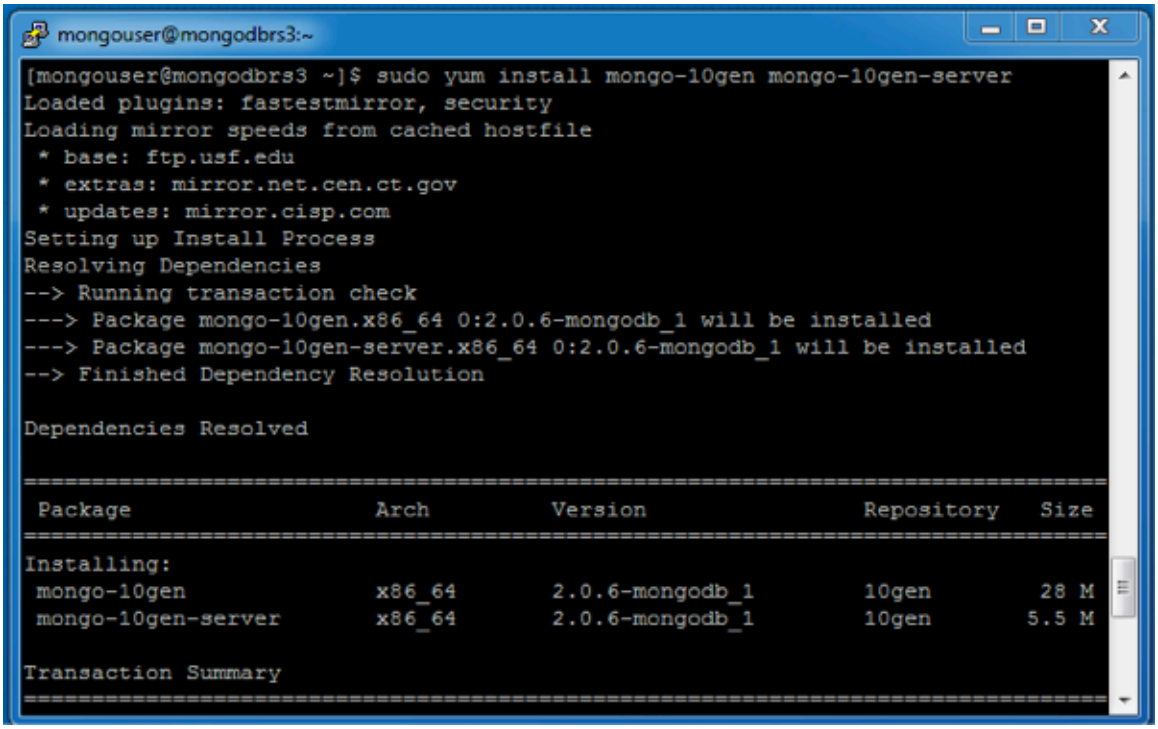
Install and Run MongoDB

As part of this step, you will be using the official 10gen supplied packages to install, configure and run MongoDB as a service using YUM. You want to install as a service since this would ensure that mongod is started on machine restart also. More information can be found at Install MongoDB on RedHat Enterprise, CentOS, or Fedora Linux doc page.

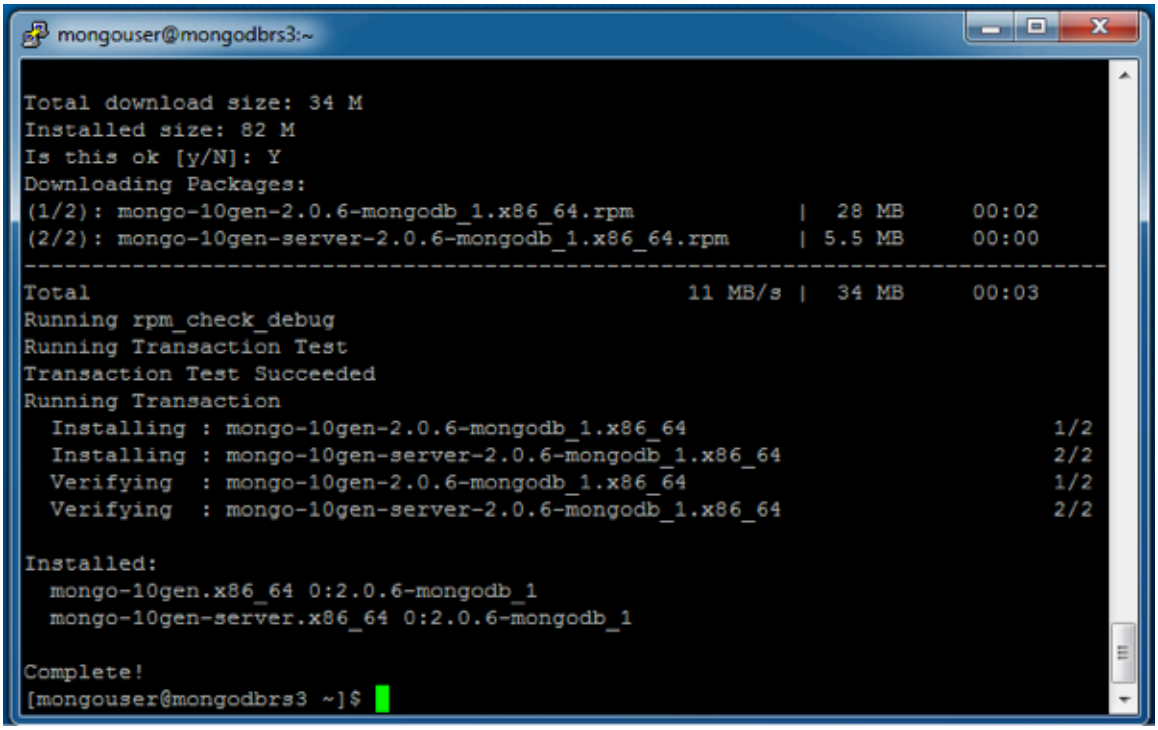
Install MongoDB

Repeat the following steps on each instance:

1. Log onto the instance
2. Create a */etc/yum.repos.d/10gen.repo* file to hold information about your repository using your favorite editor as sudo. Place the following configuration in */etc/yum.repos.d/10gen.repo* file:
   [10gen]
   name=10gen Repository
   baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64
   gpgcheck=0
   enabled=1
3. After saving the new .repo file, issue the following command to update the local package database: *sudo yum update*
4. Issue the following command (as root or with sudo) to install the latest stable version of MongoDB and the associated tools:
   sudo yum install mongo-10gen mongo-10gen-server



5. When this command completes, you have successfully installed MongoDB.



Configure MongoDB

The packages installed in the previous step configure MongoDB using the /etc/mongod.conf file in conjunction with the control script. You can find the init script at /etc/rc.d/init.d/mongod. As part of this step you will edit the mongod.conf file to set the appropriate parameters. If the parameters are commented, make sure to uncomment them.

Instance 1

1. Connect to the instance using ssh or PuTTY
2. As sudo, edit */etc/mongod.conf* to set the following parameters:
   port = 27018
   dbpath = /mnt/datadrive/data
   logpath = /mnt/datadrive/mongod.log
   replSet = mongors
3. Save the config file

Instance 2

1. Connect to the instance using ssh or PuTTY, remembering the SSH port for this node has been changed from the default of 22
2. As sudo, edit */etc/mongod.conf* to set the following parameter:
   port = 27019
   dbpath = /mnt/datadrive/data
   logpath = /mnt/datadrive/mongod.log
   replSet = mongors
3. Save the config file

Instance 3

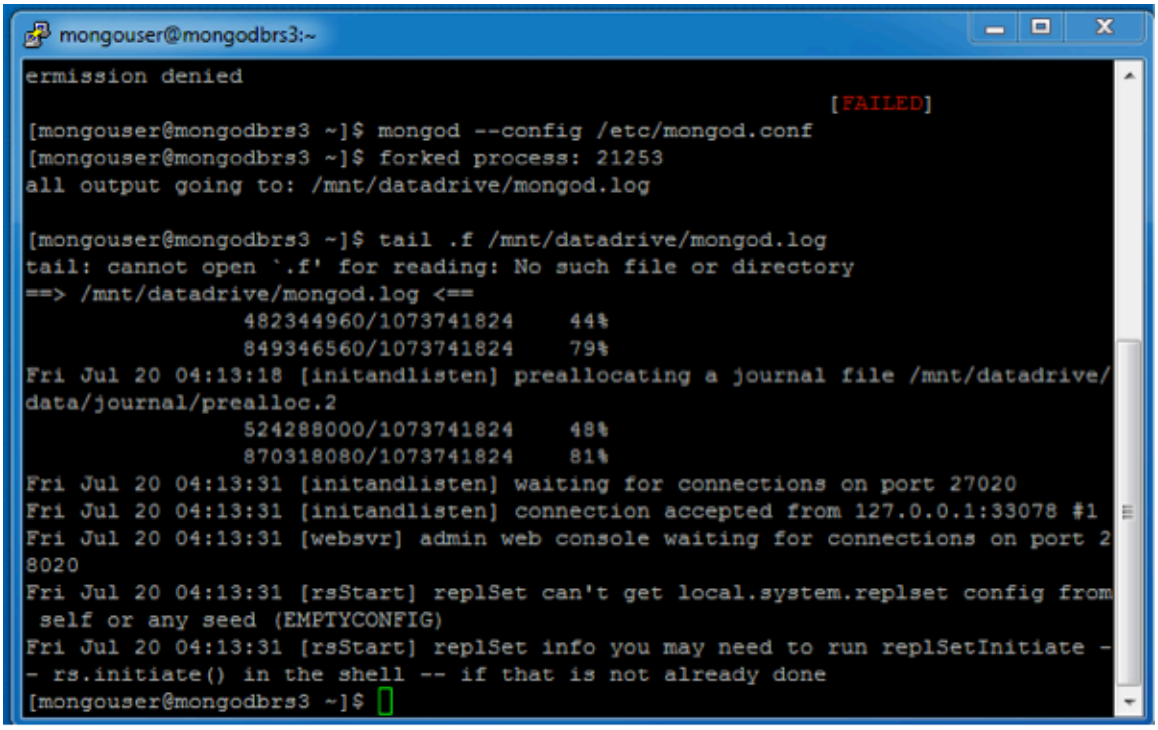
1. Connect to the instance using ssh or PuTTY, remembering the SSH port for this node has been changed from the default of 22
2. As sudo, edit */etc/mongod.conf* to set the following parameter:
   port = 27020
   dbpath = /mnt/datadrive/data
   logpath = /mnt/datadrive/mongod.log
   replSet = mongors
3. Save the config file

Start MongoDB

Once the configuration files have been edited, start the database process mongod on each instance by:

1. Log on onto the instance
2. Run the following command to start the process:
   mongod --config /etc/mongod.conf
3. This should start the mongod process

4. Verify that mongod start by tailing the log file using the command
   tail --f /mnt/datadrive/mongod.log
5. The *'waiting for connections'* message in the log file indicates mongod is up and running and waiting for client connections. This may take a while as mongod preallocates its journal files



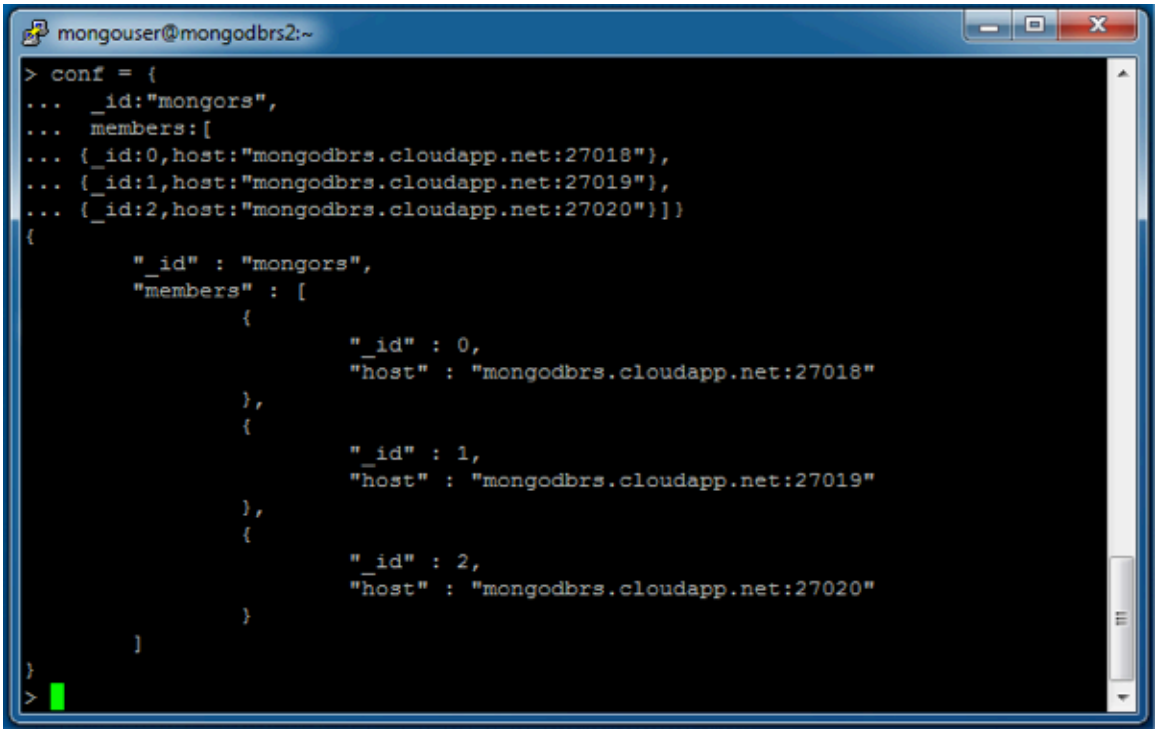
Configure the Replica Set

At this point in time, you should have mongod running on all 3 of your instances. You can now configure the 3 instances as a replica set by connecting to 1 of the 3 instances from within Azure or from outside.

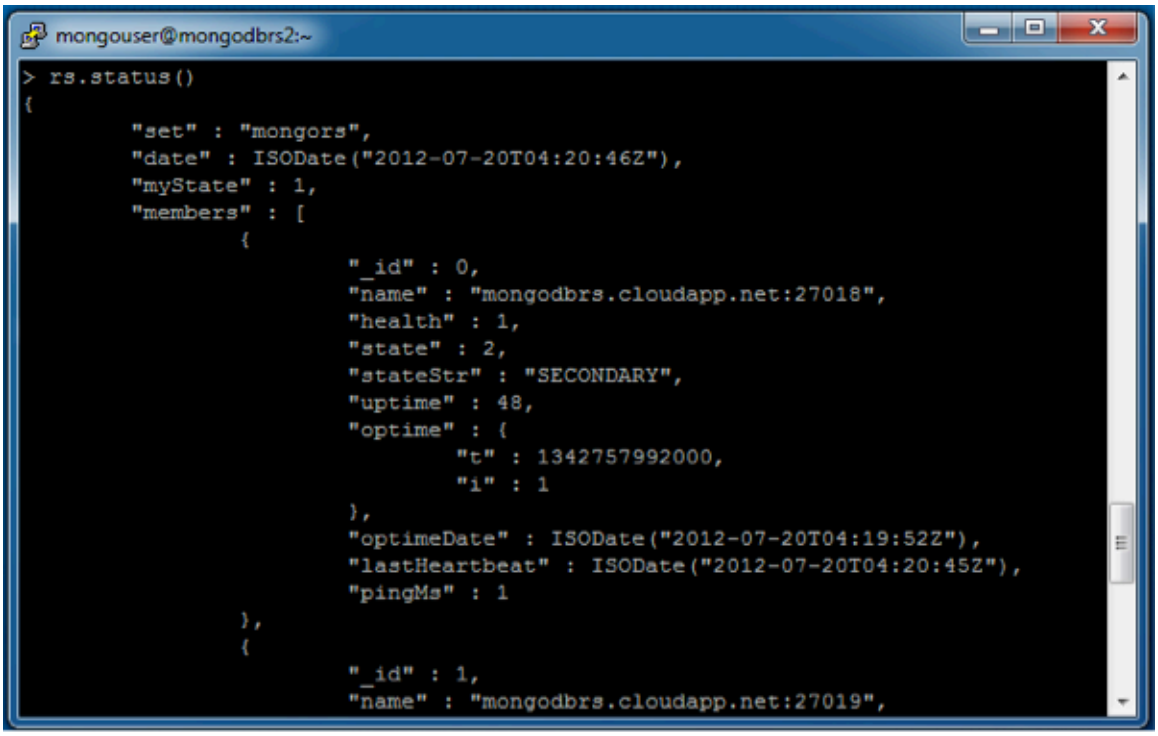
Connect to the running mongod process using the mongo shell:

1. If connected to the VM instance type the following command where port is 27018 for instance 1, 27019 for instance 2 and 27020 for instance 3: *mongo --port <port number>*
2. If connecting to the mongod process in Azure from your local machine use the following command: *mongo --host mongodbrs.cloudapp.net --port <port number>*
3. In the mongo shell type the following:

```
> conf = {
    _id = “mongors”,
     members : [
         {_id:0, host:”mongodbrs.cloudapp.net:27018},
         {_id:1, host:”mongodbrs.cloudapp.net:27019},
         {_id:2, host:”mongodbrs.cloudapp.net:27020}]}
>rs.initiate(conf)
```



4. This will start the initialization of the mongodb replica set
5. Type the command *'rs.status()'* to check the status of the replica set. Upon successful initialization, you should see 1 of the 3 instances being the 'Primary' of the set and the other 2 being the 'Secondaries'



6. You have now successfully initiated the replica set

Summary

In this tutorial you have learned how to create a set of CentOS virtual machines on Windows Azure, deploy MongoDB to them and create a replica set out of them. You can access this set from anywhere using the connection string *'mongodb://mongodbrs.cloudapp.net:27018, mongodbrs.cloudapp.net:27019, mongodbrs.cloudapp.net:27020/?replicaSet=mongors'.*

More information on MongoDB can be found at http://docs.mongodb.org/manual/.

To create and deploy a replica set to Windows Virtual Machines on Azure you can use the MongoDB Installer for Windows Azure.

MongoDB on Azure VM - Windows Installer

The MongoDB Installer for Windows Azure is a tool that can be used to provision a MongoDB replica set cluster into Window Azure VM instances.

The following are the steps to deploy a replica set using the tool

1. Install the MongoDB Installer for Windows Azure
2. Download publish settings file
3. Run the MongoDB Installer

- Prerequisites
- Install and Deployment
    - Install the tool.
    - Download your publish settings file:
    - Run the MongoDB Installer
- Troubleshooting

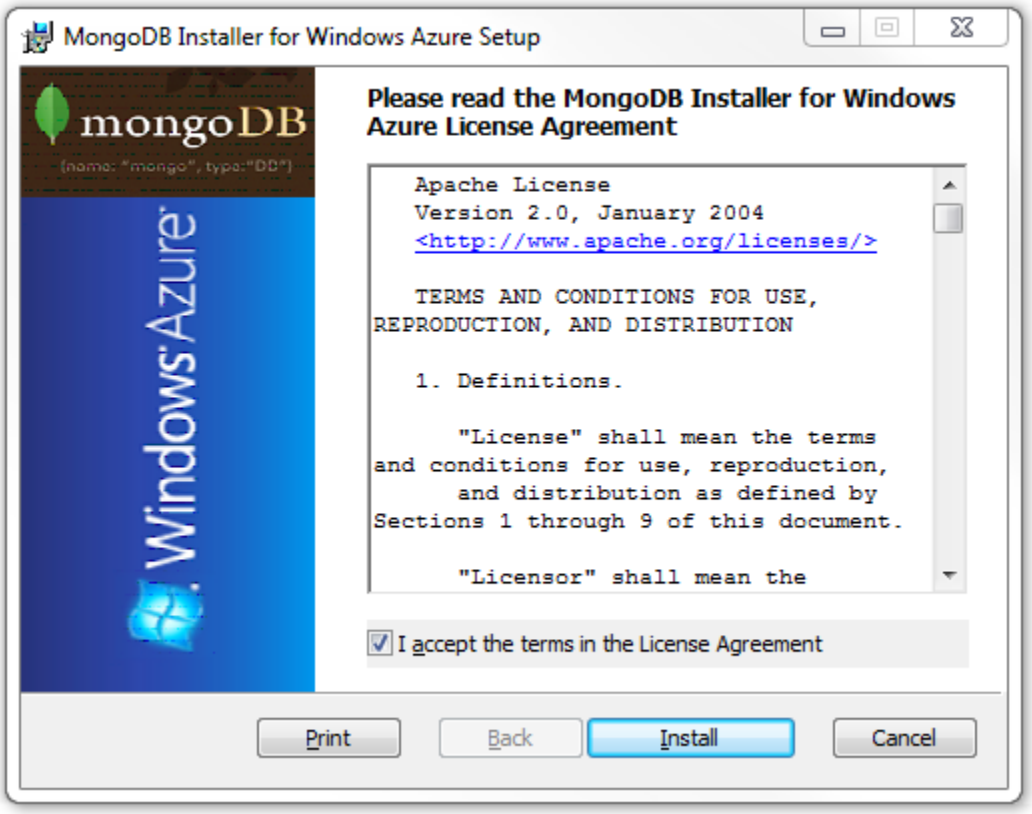
Prerequisites

- You need to be signed for Windows Azure VM Preview
- Windows versions: any 32-bit or 64-bit Windows 7 version
- Accounts: you must have administrator access to run the setup program and installer

Install and Deployment

Install the tool.

If you haven't done so already, you can download the MongoDB Installer for Windows Azure here.
Double click on the msi and accept the license agreement to complete the installation



Download your publish settings file:

The publish settings file is an xml file containing information about your subscription. This information is used by the actual script to create the VM instances in your account. To download the publish settings file, use the downloader start menu item:

```
Start -> All Programs -> MongoDB Installer for Windows Azure -> DownloadPublicSettings
```

Sign in with your Windows Live ID credentials when prompted. Remember the path to where the file was downloaded

Run the MongoDB Installer

The MongDB Installer for Azure is a powershell script. To install and configure your replica launch the installer as an Administrator

```
Start -> All Programs -> MongoDB Installer for Windows Azure -> MongoDB Installer for Windows Azure
```

Right click on the above start menu item and choose **Run as administrator**. This will open a power shell window where you can run the script to configure MongoDB on Azure VMs. You would deploy as:

```
.\deploy-mongo.ps1 <node-count> <dns-prefix> <image-name> <password> <location>
<pub-settings-file-path> [replica-set-name]
```

- **node-count** - The number of instances required in the replica set. Setting this to 1 results in a stand alone instance being deployed and not a replica set
- **dns-prefix** - The DNS prefix that will be used in the FQDN of an instance . For example, if you specify a DNS prefix of myreplicaset, the URL is http://myreplicaset.cloudapp.net.
  Note that the MongoDB Installer for Windows Azure always deploys to the production environment, so the URL never uses a GUID (as is used for deployments to the staging environment).
  The DNS prefix is not case-sensitive.
- **image-name** - The name of the VHD image in your Windows Azure storage account that will be used to create the virtual machines for the replica set. There are two possibilities:
  - You can specify the name of an existing image that you've created, which must meet these criteria:
    - It is based on the Windows Server 2008 R2 operating system
    - It has WinRM enabled with HTTP listener
    - It is hosted in your storage account
  - You can specify the name of a new image that the tool will create for you in your storage account. If the installer finds no existing image with the name you specified, it assumes that you want to create a new image and prompts you to confirm before proceeding. The new image will meet the requirements listed earlier for an existing image. Note that for this option, the name you provide cannot match any of the names of existing Windows Azure platform images
- **password** - The password to the Administrator account on the provisioned virtual machines. The same password is used for all virtual machines (nodes) in the replica set, and the password must meet Windows password complexity requirements.
  You will use this password when you connect to the virtual machines via Remote Desktop Protocol (RDP) to perform system management and troubleshooting activities. Note that RDP is enabled on all virtual machines provisioned by the MongoDB Installer for Windows Azure.
- **location** - The Windows Azure data center location where your virtual machines will be provisioned. Note that data center locations are case sensitive.
  For example, you might use "North Central US" as a location. If you don't know which locations are available for your Windows Azure subscription, you can use either of these methods to see a list of available locations:
  - Sign in to the management portal on https://windows.azure.com, click New Hosted Service, and then click Choose a Region to see a dropdown list of locations. Click Cancel when you're done.
  - You can use the **azure vm location list** command to display a list of available locations. **Note** that the installer installs the command-line tools if they are missing; however, if you want to install them before you run the installer you can do so with the Windows PowerShell script *setup-iasstool.ps1.
- **pub-settings-file-path** - The path to the publish settings file you downloaded earlier.
- **replica-set-name** - This optional can be used to specify the name of your replica set. The default name is rs. This parameter is ignored if node-count is set to 1, because a single-node deployment is not a replica set.

Troubleshooting

Qn. Error with unauthorized during deployment as seen below

> *error: The subscription is not authorized for this feature.*
> *error: vm image create command failed*

Ans. This usually indicates you do not have access to the VM preview program. Can you verify in the **new preview portal** that "Preview Features" shows account is active for Virtual machines and Virtual networks. If you have multiple accounts/subscriptions make sure you download the right pubsettings file

Qn. I am seeing a timeout error from the installer

> *Error: connect ETIMEDOUT code: 'ETIMEDOUT', errno: 'ETIMEDOUT', syscall: 'connect'*
> *Error: connect ETIMEDOUT*
> *at errnoException (net.js:670:11)*
> *at Object.afterConnect [as oncomplete] (net.js:661:19)*

Ans. This occasionally occurs due to connectivity issues with the Azure service management server or your actual deployed instances. At this point the work around is to retry. Working on making this more robust

Qn. In spite of meeting the strong password requirements the installer fails in the password setting
Ans. Ensure you do not have a **$** in the password as powershell is parsing it out as a parameter

MongoDB on Azure Worker Roles

The MongoDB Worker Role is currently a preview release. Please provide feedback, mongodb-dev, mongodb-user and IRC #mongodb are good places!

- Getting the package
- Components
- Initial Setup
- Deploying and Running
  - Running locally on compute/storage emulator
  - Deploying to Azure
- Additional Information
- FAQ/Troubleshooting
- Known issues/Where do I file bugs?

The MongoDB Worker Role project allows you to deploy and run a MongoDB replica set on Windows Azure. Replica set members are run as Azure worker role instances. MongoDB data files are stored in an Azure page blob mounted as a cloud drive. One can use any MongoDB driver to connect to the MongoDB server instance. The MongoDB .Net driver is included as part of the package.

Getting the package

The MongoDB Azure Worker Role is delivered as a Visual Studio 2010 solution with associated source files. The simplest way to get the package is by downloading it from GitHub. It is recommended using the latest tagged version.

Alternatively, you can clone the repository run the following commands from a git bash shell:

```
$ cd <parentdirectory>
$ git config --global core.autocrlf true
$ git clone git@github.com:mongodb/mongo-azure.git
$ cd mongo-azure
$ git config core.autocrlf true
```

You must set the global setting for core.autocrlf to true before cloning the repository. After you clone the repository, we recommend you set the local setting for core.autocrlf to true (as shown above) so that future changes to the global setting for core.autocrlf do not affect this repository. If you then want to change your global setting for core.autocrlf to false run:

```
$ git config --global core.autocrlf false
```

Components

Once you have unzipped the package or cloned the repository, you will see the following directories:

- **Setup** - Contains a file called solutionsetup.cmd. **Run this before opening the solution file.**
- **src** - Contains all the project's source code.
- **src/SampleApplications** - Contains sample applications that you can use to demo MongoDB on Azure. See the listing for more info.
- **lib** - Library files. Includes the MongoDB .NET driver
- **Tools** - Contains miscellaneous tools for the project.

Initial Setup

We assume you're running Windows x64 and Visual Studio. If not, install those first; Visual Studio 2010 or Visual Web Developer 2010 should work.

1. Install the latest Windows Azure SDK June 2012
2. Enable IIS on your local machine. This can be done by going to the "Turn Windows features on or off" control panel, under "Programs". Check "Internet Information Services" and also check ASP.NET under World Wide Web Services|Application Development Features.
3. Clone the project.
4. **Before opening either solution file**, run Setup\solutionsetup.cmd.
4. Open the solution you want, set the "MongoDB.WindowsAzure.(Sample.)Deploy" project as the StartUp Project, and run it!

The setup script does the following:

- Creates the cloud configs for the 2 solutions
- Downloads the MongoDB binaries to lib\MongoDBBinaries.

32-bit note: The setup script downloads the 64-bit version of MongoDB by default. If you are developing with 32-bit Windows, you will need to download the latest 32-bit MongoDB binaries and place them in lib\MongoDBBinaries yourself. Do this after running solutionsetup.cmd so it won't overwrite your work.

The prerequisites can be found in the Github readme

Once these are installed, you can open either solution MongoDB.WindowsAzure.sln for just the replica set and the monitoring application; MongoDB.WindowsAzure.Sample.sln for the replica set, monitoring application and a sample IIS app, MvcMovie, to test it.

Deploying and Running

Running locally on compute/storage emulator

The following instructions are for running the sample application.

To start, you can test out your setup locally on your development machine. The default configuration has 3 replica set members running on ports 27017, 27018 and 27019 with a replica set name of 'rs'.

In Visual Studio, run the solution using F5 or Debug->Start Debugging. This will start up the replica set, the monitoring application and the MvcMovie sample application (if you are in the sample solution).

You can verify this by using the monitoring application or MvcMovie sample application in the browser or by running mongo.exe against the running instances.

Deploying to Azure

Once you have the application running locally, you can deploy the sample app solution to Windows Azure. **Note** You cannot execute locally (on the compute emulator) with data stored in Blob store. This is due to the use of Windows Azure Drive which requires both compute and storage are in the same location.

- Detailed configuration options are outlined here
- Step-by-step deployment instructions are here

Additional Information

The MongoDB Worker Role runs mongod.exe with the following options:

```
--dbpath --port --logpath --journal --nohttpinterface --logappend --replSet
```

MongoDB creates the following containers and blobs on Azure storage:

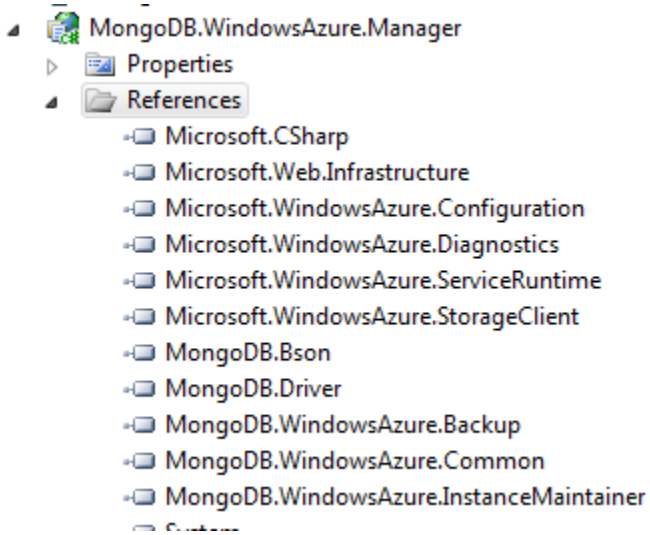
- Mongo Data Blob Container Name - mongoddatadrive(replica set name)
- Mongo Data Blob Name - mongoddblob(instance id).vhd

FAQ/Troubleshooting

- Can I run mongo.exe to connect?
    - Yes if you set up remote desktop. Then you can connect to the any of the worker role instances and run e:\approot\MongoDBBinaries\bin\mongo.exe.
- Role instances do not start on deploy to Azure
    - Check if the storage URLs have been specified correctly.
- Occasional socket exception using the .Net driver on Azure
    - This is usually due to the fact that the Azure load balancer has terminated an inactive connection. This can be overcome by setting the max idle connection time on the driver.

```
MongoDefaults.MaxConnectionIdleTime = TimeSpan.FromMinutes(1);
```

- My mongodb instances are running fine on the worker roles, the included manager app shows the instances are working fine but my client app cannot connect
    - Ensure that the Instance Maintainer exe is deployed as part of the client role. You also need to change the Service Definition for the client role to have the InstanceMaintainer started on instance start. Refer to images below:
      Instance Maintainer deployed as part of role



Instance Maintainer start defined in service definition

```
<WebRole name="MongoDB.WindowsAzure.Manager" vmsize="Small">
  <Startup>
    <Task commandLine="InstanceMaintainer.cmd" executionContext="elevated" taskType="background" />
  </Startup>
```

Known issues/Where do I file bugs?

https://jira.mongodb.org/browse/AZURE

MongoDB on Azure - Building an Application

- Overview
- Connecting to MongoDB
- Configuration
    - Specifying replica set name
    - Setting up host aliases
    - Ensure the instance maintainer is started

This page outlines the steps necessary to setup your application to connect to the MongoDB cluster running on Azure worker roles. It is not a tutorial on building an application.

Overview

When connecting to a MongoDB replica set, the client application (driver) needs to be able to connect individually to each of the members of the replica set to route reads and writes appropriately. Also with the MongoDB worker role being deployed with Internal Endpoints (only accessible from within the same service), the client application needs to be deployed as part of the same cloud service as the MongoDB application. MongoDB.WindowsAzure.Sample.MvcMovie is an example client application that follows the steps described below. This sample application along with its deployment cloud project can be found in the MongoDB on Azure Github repos.

Connecting to MongoDB

Use the ConnectionUtilities class included as part of the MongoDB.WindowsAzure.Common project. To connect to the MongoDB replica set deployed as part of you cloud service, you need to:

Use the MongoDB.WindowsAzure.Common project

```
using MongoDB.WindowsAzure.Common;
```

Use the following code snippet to obtain a replica set connection to the database

```
    MongoServerSettingsserverSettings = ConnectionUtilities.GetConnectionSettings();
    MongoServer server = MongoServer.Create(settings);
    MongoDatabase database = server["mydb"];
```

To create a connection that allows reading from secondaries use:

```
    MongoServerSettingsserverSettings = ConnectionUtilities.GetConnectionSettings();
    settings.SlaveOk = true;
    MongoServer server = MongoServer.Create(settings);
    MongoDatabase database = server["mydb"];
```

More information on the MongoDB .Net Driver ConnectionSettings object can be found in the API docs

Configuration

To account for possible changes in the IPs of the instances running MongoDB, the worker role uses the hosts file to alias each of the instances. Hence the client application instance needs the same instances. Additionally the client application needs to also have the name of the replica set.

Specifying replica set name

Ensure you client application deployment project has a setting called **ReplicaSetName** that is set to the same value as the ReplicaSetName setting of the MongoDB.WindowsAzure.MongoDB role. This can be verified by ensuring your ServiceDefinition.csdef file has the following setting as part of your client application role:

```
    <ConfigurationSettings>
      <Setting name="ReplicaSetName" />
    </ConfigurationSettings>
```

Also your ServiceConfiguration.*.cscfg files have the value set:

```
  <Role name="MongoDB.WindowsAzure.Sample.MvcMovie">
    <Instances count="1" />
    <ConfigurationSettings>
      <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString" value=
"UseDevelopmentStorage=true" />
      <Setting name="ReplicaSetName" value="rs" />
    </ConfigurationSettings>
  </Role>
```

Setting up host aliases

The included MongoDB.WindowsAzure.InstanceMaintainer utility takes care of correctly creating the hosts file. Include this as part of your client application project and ensure it is always deployed to the Azure role instance. e.g. in the sample MongoDB.WindowsAzure.Sample.MvcMovie application the following is specified in the csproj:

```
    <None Include=
"..\..\..\MongoDB.WindowsAzure.InstanceMaintainer\$(InstanceMaintainerPath)\MongoDB.WindowsAzure.Instance
>
      <CopyToOutputDirectory>Always</CopyToOutputDirectory>
      <Link>MongoDB.WindowsAzure.InstanceMaintainer.exe.config</Link>
    </None>
```

Ensure the instance maintainer is started

The InstanceMaintainer utility needs to be always running on your client application role to ensure it maintains the hosts file. To enable that it needs to be started as part of your role startup. This is done through the **Startup** setting on your csdef file. e.g. In the included MvcMovie sample app, this is specified as:

```
<WebRole name="MongoDB.WindowsAzure.Sample.MvcMovie" vmsize="Small">
  <Startup>
    <Task commandLine="InstanceMaintainer.cmd" executionContext="elevated" taskType="background" />
  </Startup>
  ....
```

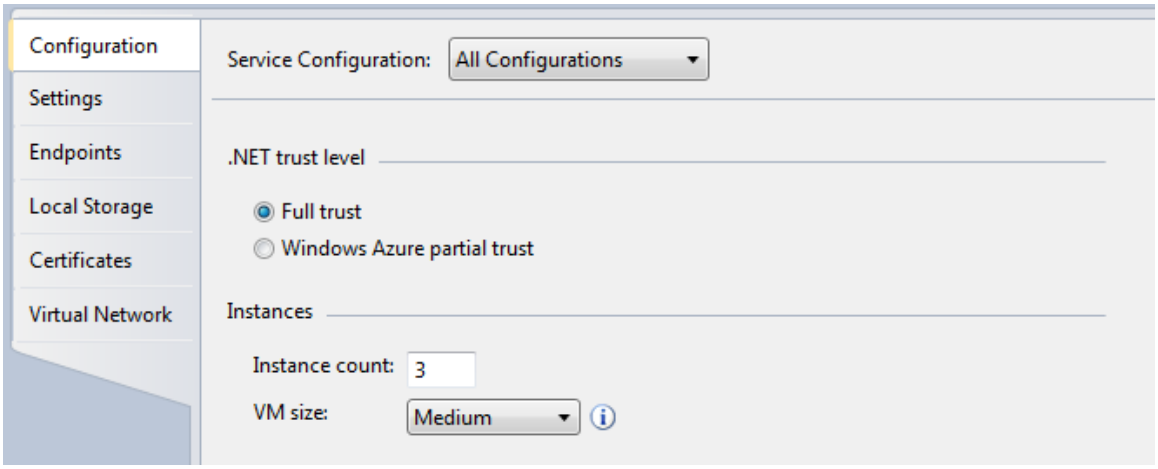
MongoDB on Azure Worker Roles - Configuration

- MongoDB.WindowsAzure.MongoDBRole configuration

The following are the configuration options available as part of the MongoDB Worker Role.

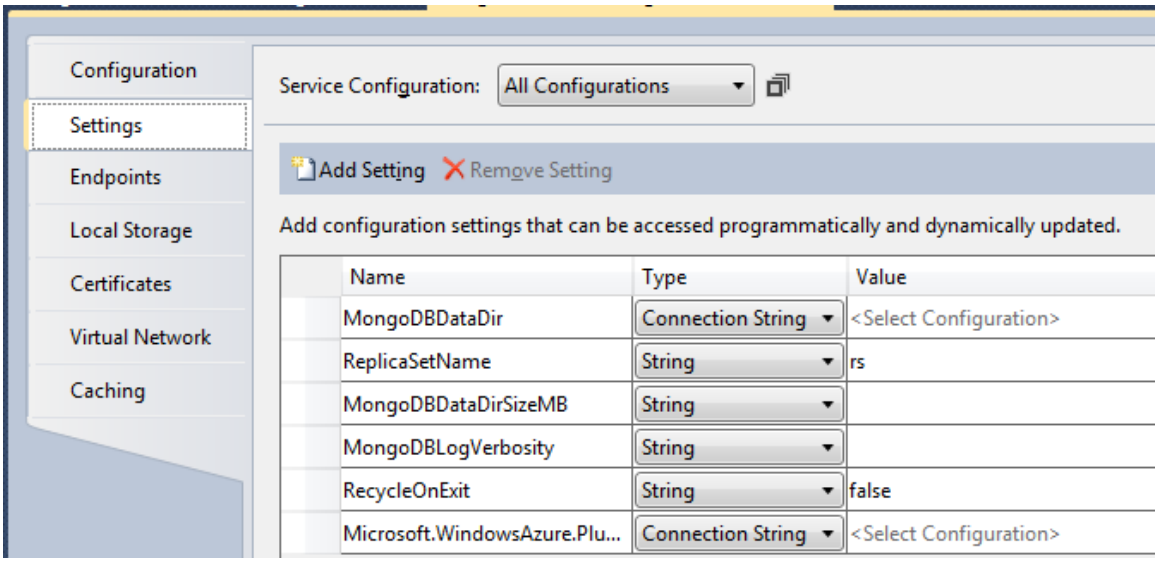
MongoDB.WindowsAzure.MongoDBRole configuration

- Configuration
  - **.Net Trust Level** - Ensure this is set to **Full Trust**
  - **Instance count** - Set to the number of replica set members you require. Default is 3.
    - Setting this to 1 would run a replica set with 1 instance (equivalent to stand alone)
  - **VM Size** - Choose size of Medium or higher. **Note** The I/O characteristics of small or extra small instance make these configurations unsuitable for MongoDB.

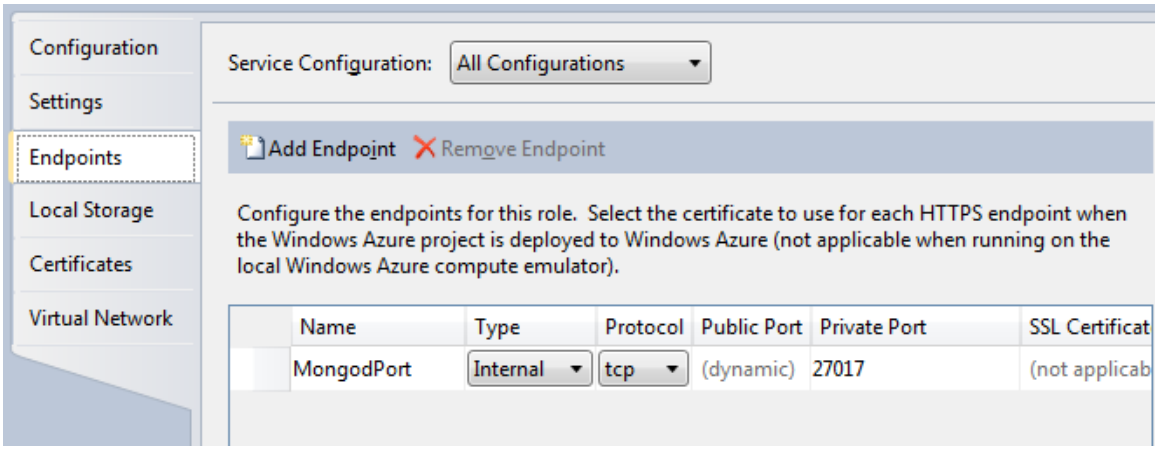


- Settings
  - **MongoDBDataDir** - Storage for mongo data files --dbpath. Configure for development or deployment. Data files are in a subdirectory called `data`. Default is local DevStorage.
  - **MongoDBDataDirSizeMB** - Size of blob (in MegaBytes) allocated for mongodb data directory. Default is 1GB for the emulator and 100GB for deployed.
  - **ReplicaSetName** - Name of the mongodb replica set. Default is rs. This also serves as the blob container name
  - **MongoDBLogVerbosity** - Verbosity to start mongod with. Default is null.
  - **RecycleOnExit** - This dictates whether the worker role is recycled when MongoDB exits. Default is false. Hence if mongodb process exits the worker role instance is still up.
  - **Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString** - Storage for diagnostics information.

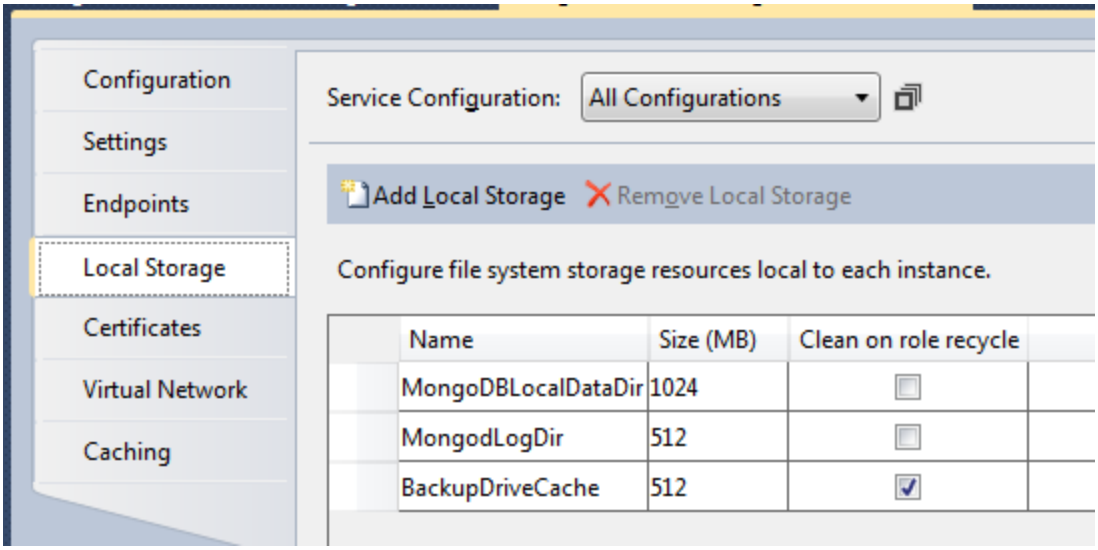
The following image shows the settings required in a local context.



- Endpoints
  - **MongodPort** - Port on which mongod listens. Default is 27017. If running locally on the Azure emulator this is port for the first instance that is started.



- Local Storage
  - **MongoDBLocalDataDir** - This specifies the size of the local storage used as a cache for the Windows Azure Drive used as the mongod data directory. Larger sizes provide better read performance.
  - **MongodLogDir** - Size of storage allocated for mongod.log. Make this large enough depending on verbosity chosen and instance size.
    Use the default settings for local storage as specified below when running on the Azure emulator. When deploying to Azure, change local storage size based on instance size chosen.
  - **BackupDriveCache** - Size of local storage used as cache during the backup process.



MongoDB on Azure Worker Roles - Deployment

> ⚠ The screenshots in this article refer to the older (pre-June 2012) Windows Azure portal, but the functionality is the same.

- In a development environment
- In the Azure environment
  - Azure setup (first time only per deployment)
    - Affinity Group
    - Storage account
    - Service
  - Deployment configuration for MongoDB.WindowsAzure.Sample.MvcMovie web role
    - Settings
  - Deployment configuration for MongoDB.WindowsAzure.Manager web role
    - Settings
  - Deployment configuration for MongoDB.WindowsAzure.MongoDBRole worker role
    - Configuration
    - Settings
    - Local Storage
    - Package and Publish

In a development environment

The solution can be built and run in a development environment using the Azure emulators as is using Visual Studio 2010 (if adequate disk space is available). Since this solution uses Cloud Drive you cannot run from a development environment against Azure cloud storage. Instead, when you run in your development environment it uses development storage:

- The mongod log file is at `C:\Users\<user>\AppData\Local\dftmp\Resources\<deploymentid>\directory\MongodLogDir` **Note** - On a development environment the port mongod listens on would be configured port (27017 by default) + instance id.
- The mongod data files are at `C:\Users\<user>\AppData\Local\dftmp\wadd\devstoreaccount1\mongoddatadrive(replica set name)\mongoddblob(instance id).vhd\data`.
  Note - Additionally when the app is running you should be able to access the drives on mounting as you would access any mounted drive.

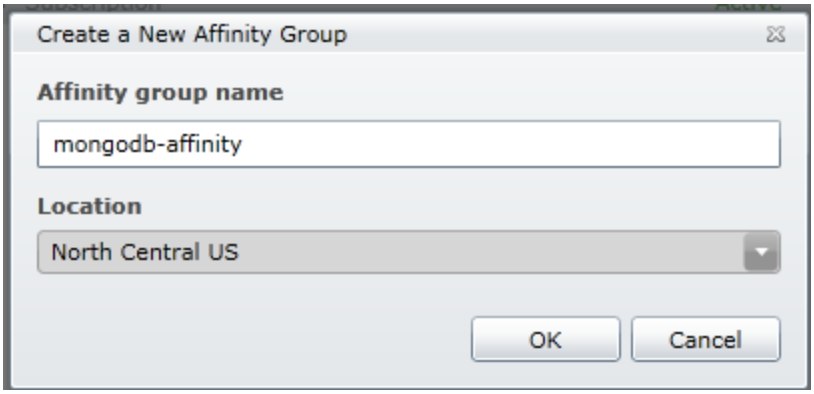
In the Azure environment

Login to the Azure management portal using your azure credentials

Azure setup (first time only per deployment)

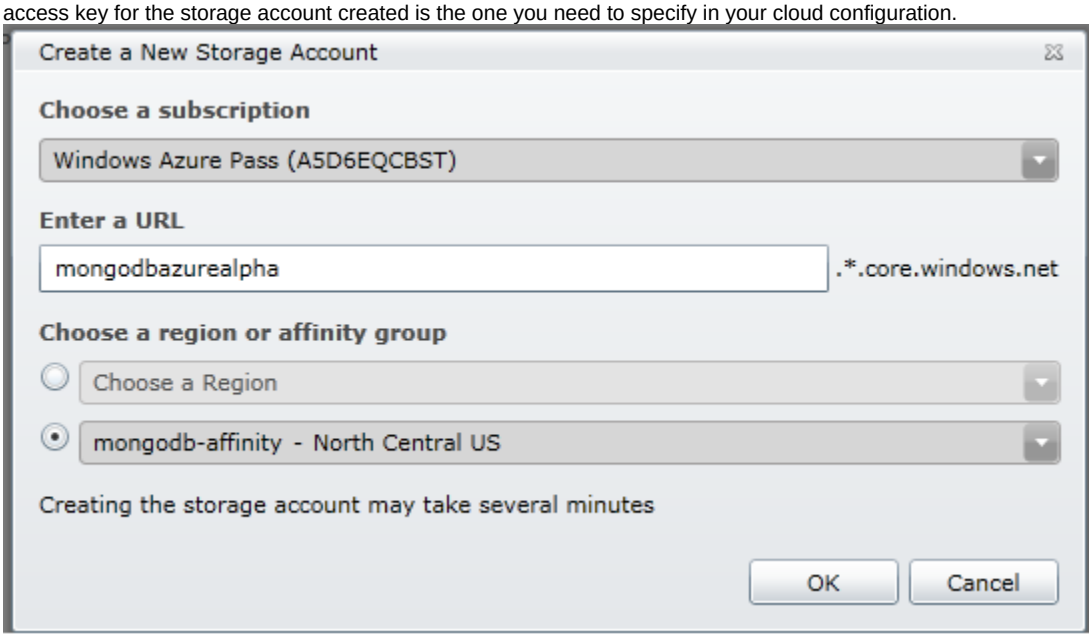
Affinity Group

Create an affinity group for your deployment. Choose your required location for the group.



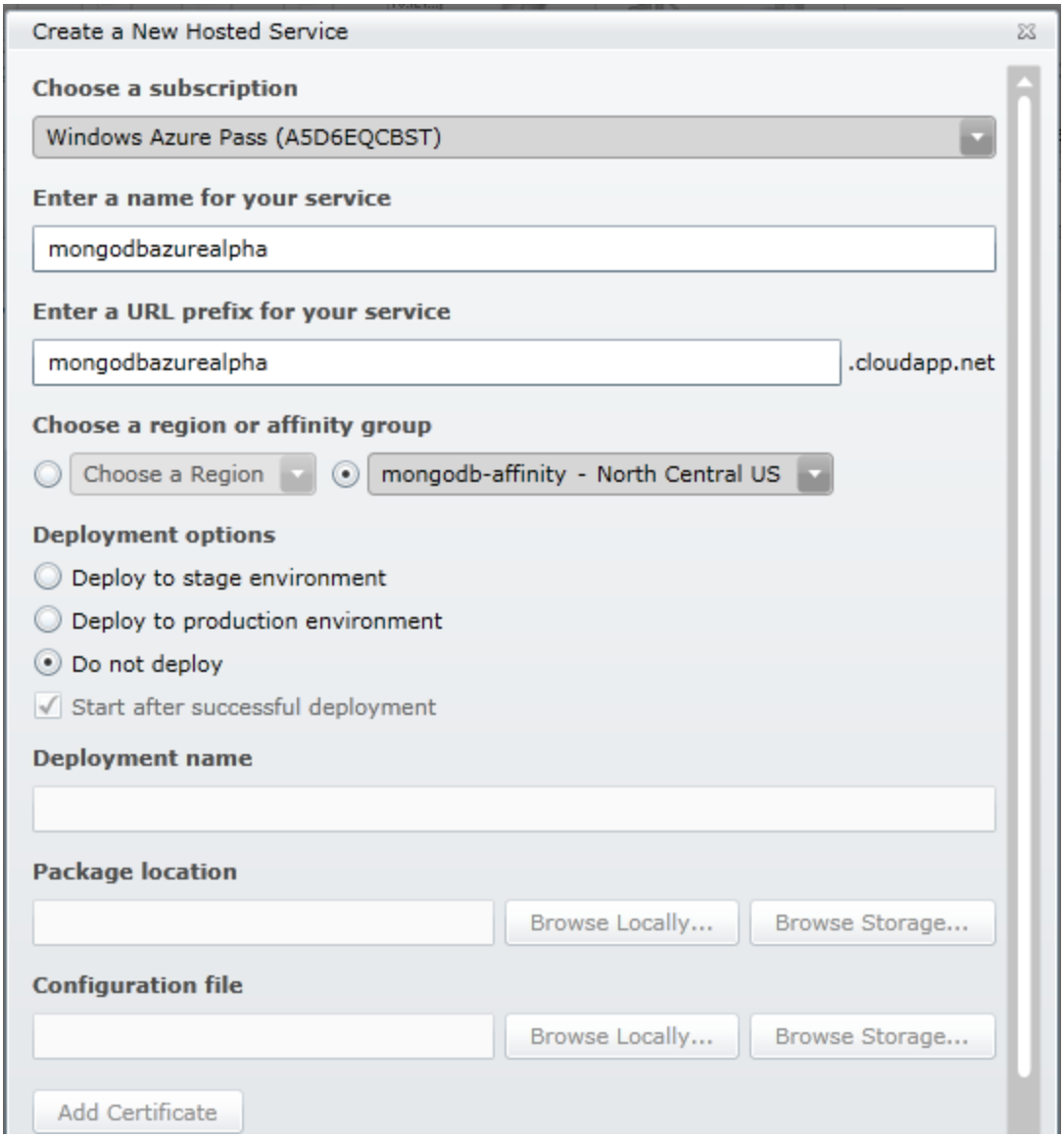
Storage account

Create the required number of storage account(s) to store the blobs. For region/affinity choose the affinity group you created earlier. **Note** The access key for the storage account created is the one you need to specify in your cloud configuration.



Service

Create a new hosted service to host your Mongo package. For region/affinity group use the same affinity group as your storage account. **Note** for cloud drives the compute and storage instances should be in the same azure domain. Choose do not deploy



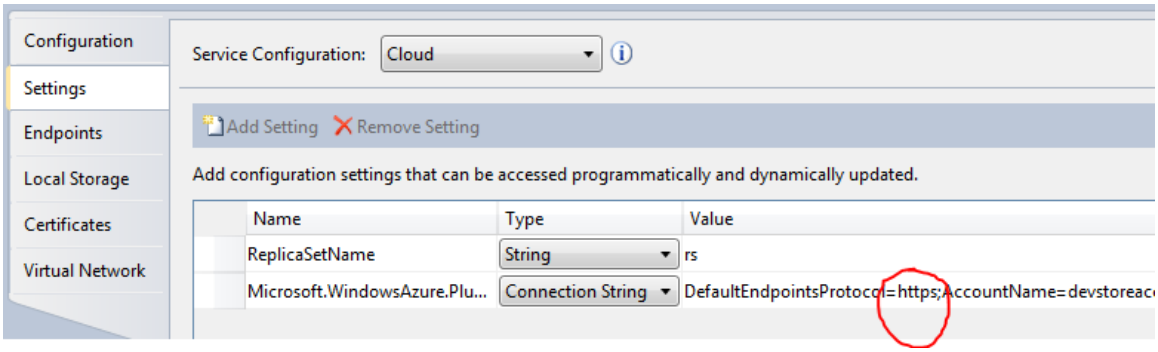
Deployment configuration for MongoDB.WindowsAzure.Sample.MvcMovie web role

If deploying the sample application you can use the default settings as is. You would only need to set the storage settings for diagnostics

Settings

In the Settings tab

- **ReplicaSetName** - This should be the same as the replica set name specified in the MongoDB.WindowsAzure.MongoDBRole
- **Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString** - Specify your azure storage credentials. Ensure that the connection mode is **https**.



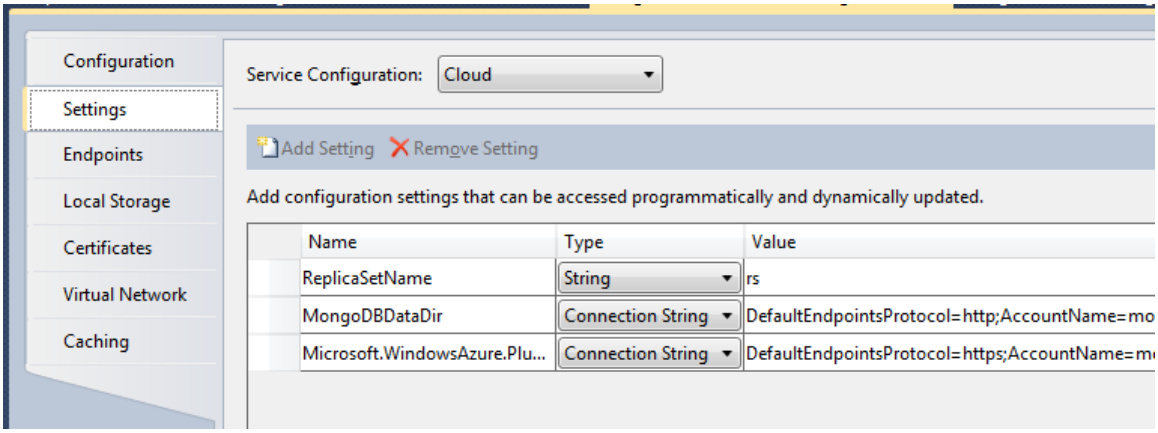
Deployment configuration for MongoDB.WindowsAzure.Manager web role

To deploy the manager application you can use the default settings as is. You would only need to set the storage settings for diagnostics and mongodb data directory.

Settings

In the Settings tab

- **ReplicaSetName** - This should be the same as the replica set name specified in the MongoDB.WindowsAzure.MongoDBRole
- **MongoDBDataDir** - Ensure that connection mode is **http**
- **Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString** - Specify your azure storage credentials. Ensure that the connection mode is **https**.



Deployment configuration for MongoDB.WindowsAzure.MongoDBRole worker role

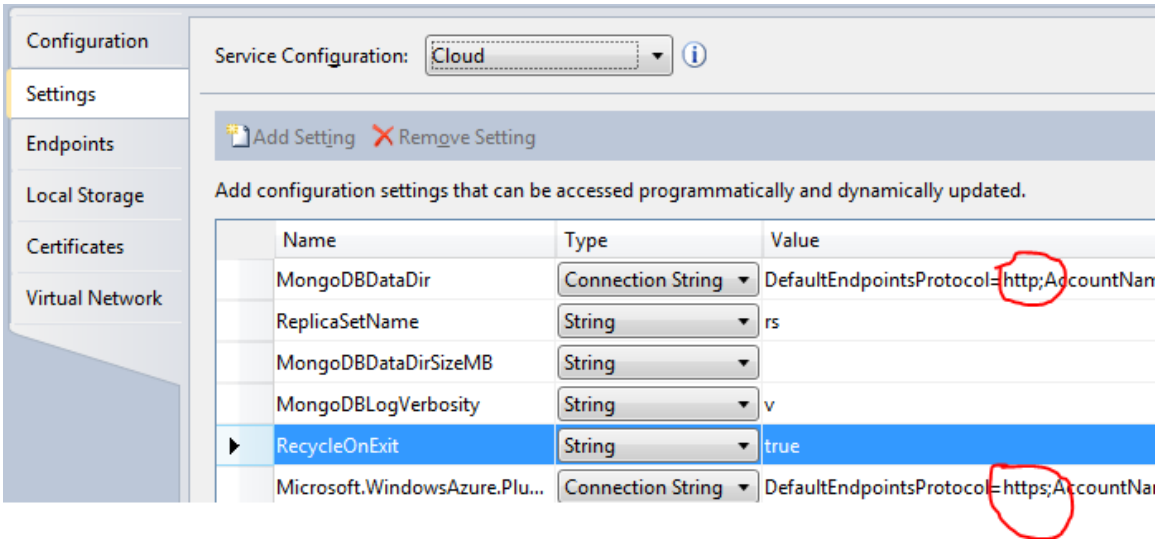
Configuration

- Ensure VM size for ReplicaSetRole is at least Medium. Larger instance sizes provide more RAM and also greater bandwidth. More information on instance sizes can be found here
- Set the Instance Count equivalent to the required number of replica set members. Default is 3.

Settings

Edit connection settings to use actual storage account credentials (created earlier). It is recommended to use different storage accounts for data and diagnostics. This would allow you to give access to external monitors for diagnostics information without giving access to your data.

- **MongoDBDataDir** - Ensure that connection mode is **http**
- **ReplicaSetName** - This is the name of the replica set in the replica set configuration. This is also the suffix to the blob container created in your storage account. **Note** - This needs to be the same as the replica set name in the client applications.
- **MongoDBDataDirSize** - Maximum size of your cloud drive where mongod data files are stored. Currently the maximum size can be 1TB.
- **MongoDBLogVerbosity** - Verbosity for mongod logging. Default is null
- **RecycleOnExit** - This dictates whether the worker role is recycled when MongoDB exits. Default is false.
- **Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString** - Ensure that the connection mode is **https**

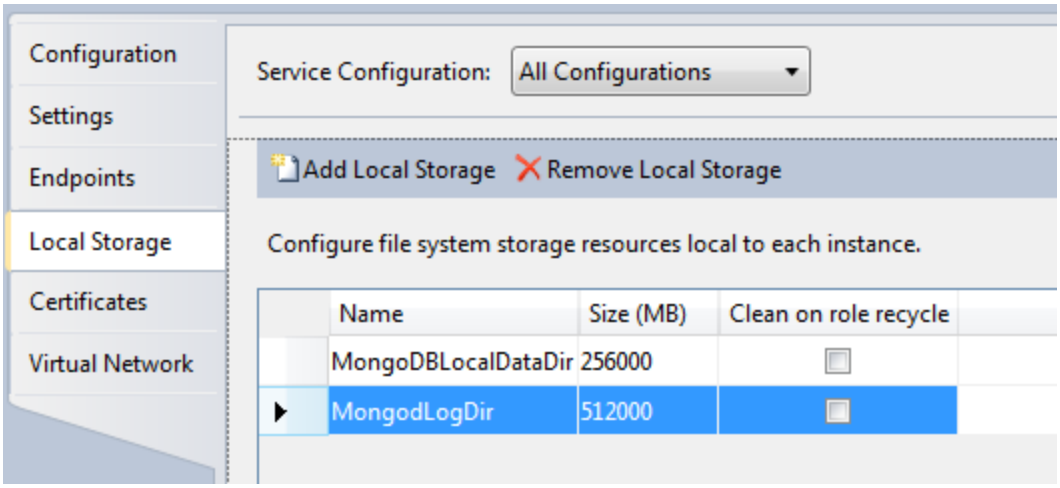


Note - If deploying multiple Azure instances make sure you use different storage accounts for each of the deployments or different replica set names if using the same storage account.

Local Storage

Configure the amount of local storage required depending on the VM size chosen. Ensure Clean on recycle role is unchecked. The following are recommendations of Local Storage **Note** All sizes are in MB.

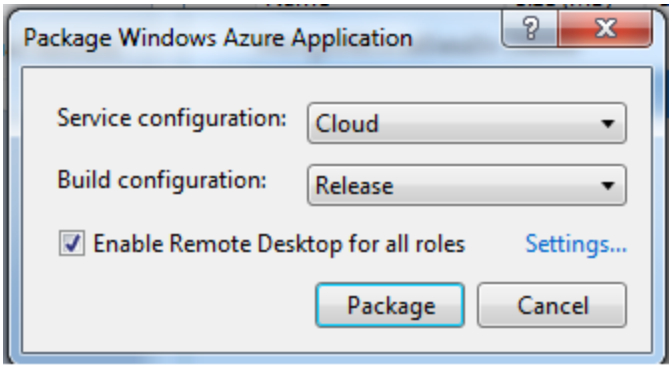
VM Size	MongoDBLocalDataDir	MongodLogDir
Medium	256000 (250 GB)	51200 (50GB)
Large	768000 (750 GB)	51200 (50GB)
Extra Large	1024000 (1000GB)	51200 (50GB)



Package and Publish

Create the Azure package by right clicking on the cloud project and choosing Package. Choose the Service Configuration as Cloud and Build Configuration as Release.

- Enable remote desktop on the role intances if required http://msdn.microsoft.com/en-us/library/gg443832.aspx



- Deploy the created package from the Azure management portal. More details on deploying can be found at http://msdn.microsoft.com/en-us/magazine/ee336122.aspx

When deploying to the Azure cloud make sure to check deployment warnings/errors to see for any breaking issues. Some common errors are

- Remote desktop is enabled but the remote desktop certificate has not been uploaded to the portal
- https is not chosen for diagnostics connection string
- If deploying the sample MvcMovie application, you can safely ignore the warning that indicates you are only running 1 instance of it.

Rackspace Cloud

- Deployment Planning
  - Instance sizes
  - Topology
  - Security
  - Monitoring
- Deploying a Single Node
  - Creating an Instance
  - Securing Your Instance
  - Installing MongoDB
  - Configuring MongoDB
- Replica Sets
- Backups
  - Cloud Servers Images
  - Built-in Tools

This guide is intended to provide instructions on getting started with MongoDB using Rackspace Cloud Servers.

First we'll step through deployment planning (instance sizes, topology, security) and then we'll set up a single MongoDB node. We'll use the same steps to create a multi-node replica set and cover the steps needed to backup your database.

Deployment Planning

Instance sizes

Rackspace Cloud offers instances with RAM ranging from 256 MB up to 64 GB. When considering instances for initial development purposes, those with 256 or 512 MB are an appropriate starting point. As development progresses towards production-level deployments we recommend moving to higher memory instances. The top-end Cloud instances are appropriate for pre-production and some production deployments. If you need to grow your deployment beyond the Cloud instances, Rackspace also offers managed dedicated servers that can be scaled to increase database performance and throughput.

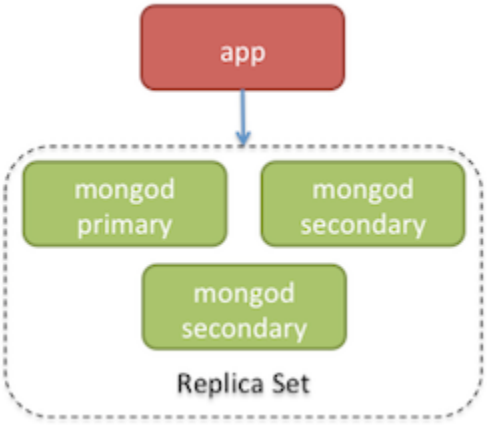
When planning your deployment, it's important to account for your resource needs for today and into the future. If you plan on growing your production systems into Rackspace's dedicated services then it may be useful to consider using RackConnect. This is a service that bridges Rackspace's public cloud infrastructure with their private dedicated servers. It makes migrating from Cloud to dedicated much easier and is also useful when creating a hybrid deployment solution combining resources from both. For more information refer to Rackspace's RackConnect site.

Topology

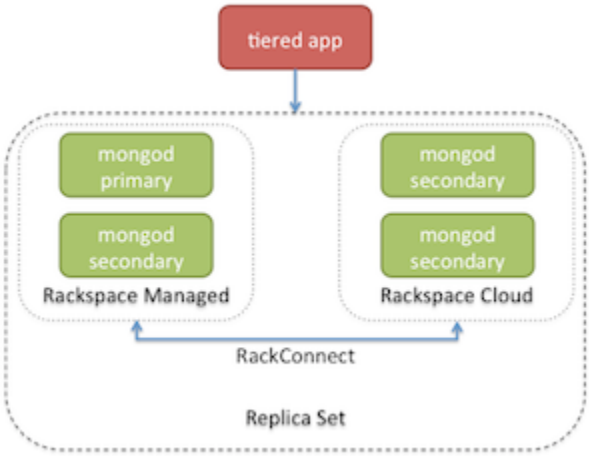
The following are example deployment scenarios for MongoDB built on Rackspace servers.

The single node example is the simplest; a single MongoDB instance can be deployed in Rackspace Cloud or Managed hosting. This deployment is useful for development and early app testing.

For production-level deployments, we'll move to a three node Replica Set running in Rackspace Cloud.



Depending on the size of your database deployment or if your app requires greater levels of database performance, another option is a hybrid deployment. In this case, a Replica Set which spans Rackspace Cloud and Managed hosting can be deployed. Cloud and Managed hosting are "connected" via RackConnect, providing a faster interconnect between the two hosting services.



Security

When deploying onto Rackspace Cloud, you'll need to secure access to your instances so that MongoDB is only available to known safe systems. To secure your instances we'll use the `iptables` command-line tool to control access to the system with a firewall. Once we've completed securing our instance, it will be using the following firewall configuration:

- port 22 (SSH) will accept any incoming TCP connections
- All other ports will reject incoming TCP connections

Later on when we deploy multiple instances we'll open up port 27017 (for mongod) to other specific IP addresses in order to facilitate a working replica set. The ports required for sharding (27018 and 27019) aren't required immediately since we'll be setting up a single node and replica set however just remember to open them up later if necessary.

Monitoring

Monitoring is a critical component of all database servers. MongoDB includes several tools to help gather statistics about data sizes, index sizes

and analyze queries. In addition, there are several third-party tools that exist to help integrate MongoDB information into other monitoring solutions. 10gen also offers MongoDB Monitoring Service, a free hosted SaaS monitoring solution that provides custom dashboards and reporting for your MongoDB instances. For more information, refer to Monitoring and Diagnostics.

Deploying a Single Node

Creating an Instance

Log in to the Rackspace Cloud control panel and navigate to Hosting > Cloud Servers > Add Server. In this example we used Ubuntu 11.10 (Oneiric Ocelot) as the OS image for our instance. Once selected, enter a server name, select a server size and continue. Be sure to record your root password or you'll have to reset it later. Once the instance is available, connect to it via SSH to secure it. By default, you'll be connecting to the server as `root`. If you'd like to create additional users/groups to use with your instance, consult the documentation for the chosen OS. For our configuration, we used the `admin` group and added an additional user, per the Ubuntu 11.10 server guide.

First we added a new user (`admin`) to the `admin` group:

```
root# adduser admin --ingroup admin
```

After stepping through the prompts to set a password (and other account information), the `admin` user will be created and added to the `admin` group.

At this point, log out of the server as `root` and login as `admin`.

Securing Your Instance

As discussed above in Security, we'll use a simple firewall setup to secure access to the instances running MongoDB. The built-in `ufw` command provides a simple mechanism to configure the system firewall. First off, enable the firewall and add a rule to allow SSH access:

```
$ sudo ufw enable
$ sudo ufw allow 22
```

Now the only port that will allow connections is port 22, attempts to connect on any other port will be dropped. With this firewall setup, the instance is secure and we can proceed with configuring MongoDB. For more information, refer to the Rackspace Introduction to IPTables and the Ubuntu firewall guide.

Installing MongoDB

Using the instructions from the MongoDB documentation on Ubuntu and Debian packages we added an additional `apt-get` source and installed MongoDB.

First add the 10gen GPG key to `apt-get` to create a "trusted" source:

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
```

Next, add the following line to `/etc/apt/sources.list`:

```
$ sudo nano /etc/apt/sources.list
...
deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen
...
```

Now update `apt-get` to pickup the new packages:

```
$ sudo apt-get update
```

Finally, install MongoDB:

```
$ sudo apt-get install mongodb-10gen
```

At this point MongoDB will be installed and started, which we can confirm via the service utility:

```
$ sudo service mongodb status
mongodb start/running, process 2308
```

You can also check the status by using `tail` to examine the MongoDB log:

```
$ sudo tail -f /var/log/mongodb/mongodb.log
```

Configuring MongoDB

By default, MongoDB is set to use `/var/lib/mongodb` as the data path. If you'd like to amend that, or update the log path, shutdown the MongoDB instance and update `/etc/mongodb.conf` with any changes.

If you change the MongoDB `dbpath` or `logpath`, be sure to set the proper ownership, etc. For example, to change MongoDB to use `/data` as the data path, use the following steps.

First, shutdown the MongoDB service:

```
$ sudo service mongodb stop
```

Next, update the MongoDB configuration file:

```
$ sudo nano /etc/mongodb.conf
...
dbpath=/data
...
```

Now create the `/data` directory and update it's ownership:

```
$ sudo mkdir /data
$ sudo chown mongodb:mongodb /data
```

Restart MongoDB with

```
$ sudo service mongodb start
```

Replica Sets

MongoDB Replica Sets are a form of asynchronous data replication between primary/secondary instances of MongoDB, adding automatic failover and automatic recovery of secondary nodes. In addition, Replica Sets can also provide the ability to distribute the read workload of your data.

To create a Replica Set, first create three instances of MongoDB using the instructions above in [Creating an Instance]. Next follow the instructions to [install] and [configure] MongoDB on each instance. Just before starting MongoDB, edit the `/etc/mongodb.conf` configuration file adding the `replSet` parameter:

```
$ sudo nano /etc/mongodb.conf
...
replSet=replicaSetName
...
```

Before proceeding, note that our current security setup has blocked access to all incoming ports (other than port 22 for SSH) therefore the MongoDB instances won't be accessible. We'll need to add rule to each instance that allows access to port 27017 (the standard `mongod` port) from the other instances in our replica set.

Collect the IP addresses of the instances in your Replica Set and execute the following `ufw` commands on each instance:

```
$ sudo ufw allow proto tcp from [IP Address 1] to any port 27017
$ sudo ufw allow proto tcp from [IP Address 2] to any port 27017
$ sudo ufw allow proto tcp from [IP Address 3] to any port 27017
```

Then start MongoDB with

```
$ sudo service mongodb start
```

Once MongoDB has started and is running on each instance we'll connect to the desired primary node, initialize the Replica Set and add the secondary nodes. First connect to the desired primary node and launch the `mongo` shell:

```
$ mongo
MongoDB shell version: 2.0.4
connecting to: test
>
```

Now initialize the Replica Set:

```
> rs.initiate()
{
    "info2" : "no configuration explicitly specified -- making one",
    "me" : "67.207.133.237:27017",
    "info" : "Config now saved locally.  Should come online in about a minute.",
    "ok" : 1
}
```

Next add the other nodes to the Replica Set. If you're executing this on the first configured instance, add the other 2 instances here:

```
> rs.add("[IP Address 2]");
{ "ok" : 1 }
PRIMARY> rs.add("[IP Address 3]");
{ "ok" : 1 }
PRIMARY>
```

Finally, you can check the status of the Replica Set from the Mongo shell with the following:

```
PRIMARY> rs.status()
...
```

In the output of the `rs.status()` command, you should see three entries, one for each instance. Each one should provide information about it's current status within the Replica Set. For more information, refer to the Replica Sets documentation.

Backups

To facilitate backups there are two main options: Cloud Servers images or MongoDB built-in backup tools.

Cloud Servers Images

Cloud Servers images are snapshots of an entire instance which are saved in Rackspace Cloud Files. Images can be created from any running instance and the process can be completed via the Rackspace Cloud control panel. When creating backups from instances in a Replica Set, be sure to image a SECONDARY instance to avoid any interruptions for your applications or data. To image a SECONDARY, first lock it against database writes from within the Replica Set, create the image and then unlock it to rejoin the Replica Set.

To lock the SECONDARY database, connect with the Mongo shell and use the following command:

```
SECONDARY> db.fsyncLock()
{
    "info" : "now locked against writes, use db.fsyncUnlock() to unlock",
    "seeAlso" : "http://www.mongodb.org/display/DOCS/fsync+Command",
"ok" : 1
}
```

While the database is locked it will not accept any incoming writes from the PRIMARY instance. Once locked, go to the Rackspace Cloud control panel and create a "New On-Demand Image". Once the image has been created and is complete, go back to the Mongo shell and issue the "unlock" command:

```
SECONDARY> db.fsyncUnlock()
{ "ok" : 1, "info" : "unlock completed" }
```

Once unlocked, any data that was written to the PRIMARY while the SECONDARY was locked will asynchronously replicate to the newly unlocked SECONDARY.

Built-in Tools

MongoDB ships with several built-in command line tools to help with database administration. One of those tools `mongodump` can be used to create file-based backups of the running database. Mongodump can be used on a live database instance however for the sake of data consistency, it is wise to use the `--oplog` option when calling `mongodump`. This allows for point-in-time backups that catch any data that was written while `mongodump` was working. For more information about live data backups, refer to the mongodump documentation.

RedHat OpenShift

- Getting Started
- Documentation
- Sample MongoDB Apps
- Additional Information

OpenShift is a Platform as a Service (PaaS) offering from RedHat, which provides support for rapid deployment and automatic scalability support for web applications developed with Java EE, Node.js, Python, PHP, Perl and Ruby, and several databases including MongoDB.

Getting Started

To get started with OpenShift and MongoDB, check out the OpenShift Quickstart guide. The guide reviews the steps necessary to create, deploy and manage your MongoDB-backed apps. The guide also covers things like application snapshots and database backups.

Documentation

- https://openshift.redhat.com/community/developers/mongodb

Sample MongoDB Apps

- Python Twitter Clone on Github
- PHP Twitter Clone on Github

Additional Information

- Getting Started with MongoDb Shell on OpenShift (video)
- Getting Started with MongoDB Monitoring Service (MMS) on OpenShift (video)
- Deploying Python Apps in the Cloud with MongoDB & OpenShift (video)
- Deploying a PHP Twitter App in the Cloud with MongoDB & OpenShift (video)
- How to Manage MongoDB on OpenShift with Your Favorite Admin Tool (video)

OpenShift Quickstart

- Cloning the App
- Deploying onto Express
- Testing the Deployment
- Advanced Functionality
    - App Administration
- Additional Information

OpenShift is a Platform as a Service (PaaS) offering from RedHat, which provides support for rapid deployment and automatic scalability support for web applications developed with Java EE, Node.js, Python, PHP, Perl and Ruby. OpenShift Express is a free shared-cloud solution for deploying your applications. With Express, you simply push your app using the command line tools to OpenShift and the platform takes care of the hosting infrastructure.

In this guide, we'll take a simple Java web application and deploy it onto OpenShift Express. We'll walk through the steps required to setup and administer MongoDB as well as backing up your data.

Cloning the App

We'll be deploying an existing Java web application onto OpenShift. The app is a simple REST-style checkin mechanism with a single endpoint. The app supports POST-ing new checkins (by supplying a comment and location) and GET-ing checkins near a given location (by supplying a pair of coordinates). The app was built using Java SE 6 and
Maven. Make sure to have those components installed, along with Git for your platform, before continuing.

First, start by cloning the repository for the web application::

```
$ git clone https://github.com/crcsmnky/openshift-checkins.git
$ cd openshift-checkins
```

Next, make sure you can build the app as-is using Maven::

```
$ mvn package
```

If that completes successfully, you're ready to move on. To prepare our app for deployment, we'll need to setup our OpenShift Express account.

Deploying onto Express

To deploy apps onto Express you'll need to create an OpenShift account from the
OpenShift sign up page and install the OpenShift command line tools.

First, follow the steps from the OpenShift Express getting started guide in the ***Install the client tools*** section for your platform. Once the tools have been installed we'll create a domain name, app name and then push the sample app to OpenShift Express (the URL for the app will be something like **http://appname-domainname.rhcloud.com**).

First, create the domain name::

```
$ rhc domain create -n your-domain -l your-openshift-login
```

This command will prompt you for your account password then for an SSH passphrase (it generates a public/private keypair to use for connecting to the OpenShift service). Next, create an entry for the app that we'll be deploying. Here you'll need to supply the app name (example, `expresscheckin`) and `jbossas-7.0` as the app type::

```
$ rhc app create -a expresscheckin -t jbossas-7
Password:
Creating application: expresscheckin
Now your new domain name is being propagated worldwide (this might take a minute)...
Warning: Permanently added 'expresscheckin-your-domain.rhcloud.com,50.16.164.248' (RSA) to the list of
known hosts.
Confirming application 'expresscheckin' is available:  Success!

expresscheckin published:  http://expresscheckin-your-domain.rhcloud.com/
git url:  ssh:
//429827960cbf4518b1785ed928db9be7@expresscheckin-your-domain.rhcloud.com/~/git/expresscheckin.git/
Successfully created application: expresscheckin
```

After the app is created the command will also clone the app's repository locally. Before continuing, we need to set up MongoDB on OpenShift and also update our app's MongoDB connection information::

```
$ rhc app cartridge add -a expresscheckin -c mongodb-2.0

RESULT:

MongoDB 2.0 database added.  Please make note of these credentials:

       Root User: admin
   Root Password: 1wbLGYAmPgDM
   Database Name: expresscheckin
  Connection URL: mongodb://127.6.85.129:27017/

You can manage your new MongoDB by also embedding rockmongo-1.1
```

Now that we've added support for MongoDB to our app (on OpenShift) we'll need to take the credentials returned and update our `openshift-checkins` app configuration. Go back to the `openshift-checkins` directory and edit `CheckinServlet.java` and add/update the following lines in the `init` function using the MongoDB details provided by OpenShift (don't forget to uncomment the `if` statement with the authentication statement)::

```
$ cd openshift-checkins
$ nano src/main/java/CheckinServlet.java
...
conn = new Mongo("127.6.85.129", 27017);
db = conn.getDB("expresscheckin");

if (db.authenticate("admin", "1wbLGYAmPgDM".toCharArray())) {
  throw new MongoException("unable to authenticate");
}
...
```

Then build the `openshift-checkins` WAR file::

```
$ mvn package
```

Now, return to the cloned repository, remove the sample code generated from the `expresscheckin` app repo that was cloned to our system and copy the `checkins.war` file into the `deployments` as `ROOT.war`::

```
$ cd ../expresscheckin
$ rm -rf pom.xml src
$ cp ../openshift-checkins/target/checkins.war deployments/ROOT.war
```

As part of the deployment process, we also need to flag our WAR file to be deployed (expanded and copied to the right places)::

```
$ touch deployments/ROOT.war.dodeploy
```

Now we can add it to the repository, commit and push::

```
$ git add -A
$ git commit -m "initial deployment of expresscheckin app onto OpenShift Express"
$ git push origin
```

After pushing the app, it will take a few minutes for the app to become available at http://expresscheckin-your-domain.rhcloud.com/. That's it, you've deployed a simple Java app that uses MongoDB to OpenShift Express. Refer to Testing the Deployment below for some notes on testing the app's functionality.

Testing the Deployment

Once you've deployed your app onto OpenShift Express, the URL for the app will be something of of the form http://expresscheckin-your-domain.rhcloud.com. We'll now use that URL to conduct some tests on our deployed app.

Since the app is a simple RESTish mechanism we can use `curl` to test it out. Let's start by posting a new comment and location to the URL (ex. http://appurl.rhcloud.com/checkin)::

```
$ curl -X POST -d "comment=hello&x=1&y=1" http://appurl.rhcloud.com/checkin
```

Now let's see if we can find the comment we just posted at that location (x = 1, y = 1)::

```
$ curl "http://appurl.rhcloud.com/checkin?x=1&y=1"
{ "_id" : { "$oid" : "4f0e068c3004bfd40822840b"} , "comment" : "hello" , "location" : [ 1.0 , 1.0]}
```

If these worked, then it looks like we've got a functional app deployed onto OpenShift.

Advanced Functionality

Once you've deployed your app, you'll probably need to connect to your server at some point so you can do things like reviewing app logs or query data in MongoDB. The following steps will cover how you can do that (and if available, set up some advanced functionality).

App Administration

Using the command line tools, we can connect to the server our app is deployed on. Run the following command to get information about the current running apps::

```
$ rhc domain show
Password:

User Info
=========
Namespace: checkins
  RHLogin: sandeep@clusterbeep.org


Application Info
================
expresscheckin
    Framework: jbossas-7
     Creation: 2012-04-14T14:04:54-04:00
         UUID: 485daa768043454d9cdcb9343018eb6e
      Git URL: ssh:
//485daa768043454d9cdcb9343018eb6e@expresscheckin-checkins.rhcloud.com/~/git/expresscheckin.git/
Public URL: http://expresscheckin-checkins.rhcloud.com/

 Embedded:
      mongodb-2.0 - Connection URL: mongodb://127.6.85.129:27017/
```

The `UDID` above shows the user ID we can use to SSH into our server::

```
$ ssh 429827960cbf4518b1785ed928db9be7@expresscheckin-your-domain.rhcloud.com

Welcome to OpenShift shell

This shell will assist you in managing openshift applications.

!!! IMPORTANT !!! IMPORTANT !!! IMPORTANT !!!
Shell access is quite powerful and it is possible for you to
accidentally damage your application.  Procede with care!
If worse comes to worse, destroy your application with rhc-ctl-app
and recreate it
!!! IMPORTANT !!! IMPORTANT !!! IMPORTANT !!!

type "help" for more info.
```

Now at the prompt type `help` to see the available commands (in addition to normal shell commands)::

```
[openshift]$ help
Help menu: The following commands are available to help control your openshift
application and environment.

ctl_app         control your application (start, stop, restart, etc)
ctl_all         control application and deps like mysql in one command
tail_all        tail all log files
export          list available environment variables
rm              remove files / directories
ls              list files / directories
ps              list running applications
kill            kill running applications
mongo           interactive MongoDB shell
```

To connect to MongoDB, we'll need to use the credentials provided to us from OpenShift above when we set it up initially. Once you have those in hand, here's how you should connect to the database::

```
[openshift]$ mongo expresscheckin -u admin -p 1wbLGYAmPgDM
MongoDB shell version: 2.0.2-rc1
connecting to: 127.1.13.1:27017/expresscheckin
>
```

From here you can query your data as needed for your application. Finally, to backup your app's data, follow the instructions found on the MongoDB Backup docs. Specifically you'll need to use the mongodump command to do a live backup of your data.

Backing up your app is a simple single step where we create a ***snapshot*** of the entirety of the application including data, which we can do locally on our development machine::

```
$ rhc app snapshot save -a expresscheckin
Password:
Pulling down a snapshot to expresscheckin.tar.gz

Running extra dump: mongodb_dump.sh
MongoDB already running
Stopping application...
Done
Creating and sending tar.gz
Running extra cleanup: mongodb_cleanup.sh
Starting application...
Done
```

Additional Information

For additional information, refer to the following:

- OpenShift Express User Guide or
- OpenShift Flex User Guide

VMware CloudFoundry

MongoDB is a supported service on VMware's Cloud Foundry.

Clound Foundry is a platform-as-a-service solution. At this time, CloudFoundry.com (the public instance of Cloud Foundry operated by VMware) supports applications written in Spring Java, Rails and Sinatra for Ruby, Node.js. Scala and other JVM languages/frameworks including Groovy and Grails.*

Starting a MongoDB service

```
vmc create-service mongodb --name MyMongoDB
```

Once you create a MongoDB service, you can bind and use it inside of Cloud Foundry applications.

Java

- MongoDB Java Language Center

Node.js

- http://docs.cloudfoundry.com/services/mongodb/nodejs-mongodb.html
- Blog post : Getting started with VMware Cloud Foundry, MongoDB and Node.js
- MongoDB Node.js language center

Ruby

- http://docs.cloudfoundry.com/services/mongodb/ruby-mongodb.html

- Getting started with VMware Cloud Foundry, MongoDB and Rails
- MongoDB Ruby language center
- Using GridFS with Ruby and Cloud Foundry

See Also

- VMware Cloud Foundry with MongoDB webinar

Contributors

- JS Benchmarking Harness
- MongoDB kernel code development rules
- Project Ideas
- Roadmap
- UI
- Source Code
- Building
- Database Internals
- Emacs tips for MongoDB work

- 10gen Contributor Agreement

JS Benchmarking Harness

This benchRun command is designed as a QA baseline perf measurement tool, not designed to be a "benchmark".

CODE:

```
db.foo.drop();
db.foo.insert( { _id : 1 } )

ops = [{op: "findOne", ns: "test.foo", query: {_id: 1}},
       {op: "update", ns: "test.foo", query: {_id: 1}, update: {$inc: {x: 1}}}]

for ( var x = 1; x <= 128; x *= 2) {
    res = benchRun( {
     parallel : x ,
        seconds : 5 ,
        ops : ops
    } );
    print( "threads: " + x + "\t queries/sec: " + res.query );
}
```

Dynamic values

```
// benchmark updates using the $inc operator
res = benchRun( {
    ops : [ {
        ns : "test.foo" ,
        op : "update" ,
        query : { _id : { "#RAND_INT" : [ 0 , 100 ] } } ,
        update : { $inc : { x : 1 } }
    } ] ,
    parallel : 2 ,
    seconds : 1 ,
    totals : true
} );
print( "threads: 2\t update/sec: " + res.update );
// benchmark inserts with random strings
res = benchRun( {
    ops : [ {
        ns : "test.foo" ,
        op : "insert" ,
        doc : { y : { "#RAND_STRING" : [ 10 ] } }
    } ] ,
    parallel : 2 ,
    seconds : 1 ,
    totals : true
} );
print( "threads: 2\t insert/sec: " + res.insert );
```

Options

- host - the hostname of the machine mongod is running on (defaults to localhost)
- username - the username to use when authenticating to mongod (only use if running with auth)
- password - the password to use when authenticating to mongod (only use if running with auth)
- db - the database to authenticate to (only necessary if running with auth)
- ops - a list of objects describing the operations to run (documented below)
- parallel - the number of threads to run (defaults to single thread)
- seconds - the amount of time to run the tests for (defaults to one second)

Operation Options

- ns - the namespace of the collection you are running the operation on, should be of the form "db.collection"
- op - the type of operation can be "findOne", "insert", "update", "remove", "createIndex", "dropIndex" or "command"
- query - the query object to use when querying or updating documents
- update - the update object (same as 2nd argument of update() function)
- doc - the document to insert into the database (only for insert and remove)
- safe - boolean specifying whether to use safe writes (only for update and insert)

Dynamic operators

- `{ "#RAND_INT" : [ min , max , <multiplier> ] }`
  - [0 , 10 , 4] would produce random numbers between 0 and 10 and then multiply by 4
- `{ "#RAND_STRING" : [ length ] }`
  - [3] would produce a string of 3 random characters

Dynamic operators generate random strings, random ints, etc but don't work in second level objects, just main level

- This is fine
  - `var complexDoc3 = { info: "#RAND_STRING": [30] } }`
- This is only going to insert a value called "#RAND_STRING" with an array as a key

```
var complexDoc3 = { info: { inner_field: { "#RAND_STRING": [30] } } }
```

More info:

http://github.com/mongodb/mongo/commit/3db3cb13dc1c522db8b59745d6c74b0967f1611c
http://github.com/mongodb/mongo/blob/master/jstests/bench_test1.js http://github.com/mongodb/mongo/blob/master/jstests/bench_test2.js
http://github.com/mongodb/mongo/blob/master/jstests/bench_test3.js

MongoDB kernel code development rules

- C++ Style Guide
- JS Style Guide
- Git Committing and Pushing
- User Facing Conventions
  - Use camelCase for about everything
  - Include units in fields

C++ Style Guide

Coding conventions for the MongoDB C++ code...

- Kernel class rules
- Kernel code style
- Kernel concurrency rules
- Kernel exception architecture
- Kernel logging
- Kernel string manipulation
- Memory management
- Writing tests

For anything not mentioned here, default to google c++ style guide

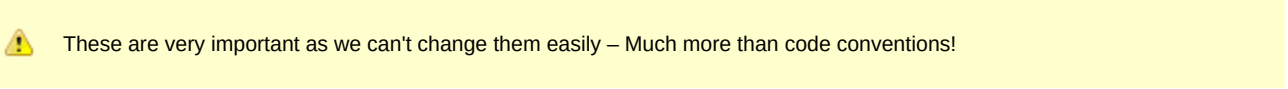
JS Style Guide

10gen follows the google javascript style guide

Git Committing and Pushing

- commit messages should have the case in the message SERVER-XXX
- commit messages should be descriptive enough that a glance can tell the basics
- commits should only include 1 thought.
- do **NOT** push until running the test suite

User Facing Conventions



Anything users see – command line options, command names, command output, we need to think hard and carefully about the name to be used, and the exact format and consistency of the items output. For example, serverStatus output is a bit of a mismash of lowercase and camelCase. Let's fix that over time, starting with new things.

Anything user facing must be ran by several team members first.

- **Do NOT add a new $operator without signoff by the entire team.**
- **Do NOT add a new command without signoff by the entire team.**

Use camelCase for about everything

- `--commandLineOptions`
- { commandNames : 1, commandOption : "abc" }
- Names of fields in objects we generate - such as command responses, profiler fields.

Include units in fields

In things like serverStatus, include the units in the stat name if there is any chance of ambiguity. For example:

- writtenMB
- timeMs

We should have standards for these – i.e. megabytes should always be "MB" and not "Mb" and "Megabytes" in different places. So the standards are:

- for bytes : use "MB" and show in megabytes unless you know it will be tiny. Note you can use a float so 0.1MB is fine to show.
- for time: use millis ("Ms") for time by default. you can also use Secs and a float for times you know will be very long.
- for microseconds, use "Micros" as the suffix, e.g., `timeMicros`.

Kernel class rules

Design guidelines

- ***Never use multiple inheritance***. If you need the service of several classes, use delegation. The only possible but highly unlikely exception to this is if your class inherits from other pure abstract classes.
- Have a comment before a class that explains its purpose. Perhaps the class name is so clear that this is obvious. Then some commentary on what you are up to.
- Only add ***members and methods to a class if they make sense*** w.r.t the bullet above. If you find yourself unsure to where to hook a piece of logic, rethink the class and surrounding classes purposes.
- Class ***names and methods names are to be descriptive*** of what they do. Avoid generic overloaded names (e.g., write, add, ...) to make grep easier (and maybe reading too).
- Don't put implementation details in the header unless the user of the class needs to know them. Sometimes single line inline implementations are good "documentation". If something needs to be inline for performance, put it at the bottom of the fine using the `inline` keyword instead of in the middle of the class definition (if the implementation is more than a line or two long).
- Assume all methods can throw a DBException. If a class should never throw (e.g can be called in a destructor), that should be clear.
- *Write a unit test for each class you create.* If you can't easily write a unit test for the class, that is a strong hint it has way too many external dependencies.
- ***Do not create early hierarchies***. An early hierarchy is a one where there is only one type of derived class. If you need to separate functionality, use delegation instead. In that case, make sure to test separately.
- Avoid `friend`.
- Default to making classes ***non-assignable and non-copyable***. (Use boost::noncopyable.)

Layout guidelines

- For classes where layout matters (anything with #pragma pack), put data members together at the top of the class. You must also have a BOOST_STATIC_ASSERT(sizeof(ClassName) == EXPECTED_SIZE) either directly under the class or in the associated .cpp file

Kernel code style

- basics
- case
- comments
- inlines
- strings
- brackets
- class members
- functions
- templates
- namespaces
- start of file
- assertions
- return early
- numeric constants
- explicit constructors
- #includes
    - For cpp files:
    - For h files:
- file names
- casting

- RAII and bare vs. smart pointers

basics

- Use spaces, no literal tabs.
- 4 spaces per indentation.
- Limit lines to 100 columns.
- Use LF (Unix-style) line endings, not CR-LF (DOS)
    - git has a config option in Windows to convert line endings automatically (core.autocrlf)
- For anything that isn't explicitly covered here, default to the Google C++ Style Guide: http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

case

Use camelCase for most `varNames`

See important notes on case on the parent page for user facing names!

comments

We follow http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml#Comments for placement of comments

As for style, we use javadoc's in classes and methods (public or private) and simple comments for variables and inside code

```
/**
 * My class has X as a goal in life
 * Note: my class is fully synchronized
 */
class DoesX {

...
  /**
   * This methods prints something and turns off the lights.
   * @param y the something to be printed
   */
   void printAndGo(const std::string& y) const;

...
 private:
    // a map from a namespace into the min key of a chunk
// one entry per chunk that lives in this server
std::map<std::string, BSONObj> _chunkMap;

    /**
     * Helper that finds the light switch
     */
    Pos findSwitch() const;

    /** @return the light switch state. */
    State getSwitchState() const;
};
```

```
void DoX(bool y) {
    // if y is false, we do not need to do a certain action and explaining
// why that is takes multiple lines.
if (!y) {
        printf("y is true!\n");
    }
}
```

Don't forget – even if a class's purpose is obvious, you can put a comment on it as to why it exists!

inlines

- Put long inline functions in a `-inl.h` file. *
- If your inline function is a single line long, put it and its decl on the same line e.g.:

```
int length() const { return _length; }
```

- If a function is not performance sensitive, and it isn't one (or 2) lines long, put it in the cpp file. Keep code out of headers.

strings

See

- util/mongoutils/str.h
- bson/stringdata.h

Use `str::startsWith()`, `str::endsWith()`, not `strstr()`.

Use `<< 'c'` not `<< "c"`.

Use `str[0] == '\0'` not `strlen(str) == 0`.

See Kernel string manipulation.

brackets

```
if (0) {
}
else if (0) {
}
else {
}

do {
} while (0);
```

class members

```
class Foo {
   int _bar;
};
```

functions

Declaration:

```
void foo(int v, MyType myItem);
```

Avoid declarations of extern functions in source files! Instead, #include a proper .h file. Be sure to match the header filename to the source filename where the function definition appears.

Definition:

```
void foo(int v, MyType myItem) {
}
```

Invocation:

```
foo(1, MyType());
```

templates

```
set<int> s;
```

namespaces

```
namespace foo {
   int foo;
   namespace bar {
      int bar;
   }
}
```

start of file

```
license (AGPL or Apache, depending on C++ driverness)
```

assertions

See Kernel exception architecture.

return early

BAD

```
int foo() {
   if (x) {
     ...
   }
}
```

GOOD

```
int foo() {
   if (!x)
     return;

   ...
}
```

Keeps indentation levels down and makes more readable.

numeric constants

Large, round numeric constants should be written in multiplied form so that you never need to count digits.

```
const int tenMillion = 10*1000*1000;
const int megabyte = 1024*1024;
```

explicit constructors

To avoid implicit type conversion, use the "explicit" keyword before constructors that take a single parameter.

#includes

- Use "double quotes" for 10gen code, <angle brackets> for 3rd party or library headers.

  ```
  examples:
  #include "mongo/pch.h"
  #include <boost/thread.h>
  #include <vector>
  ```

- Always use forward relative path from mongo/src/; do not use ".."

```
correct:
#include "mongo/db/namespace_details.h"
incorrect:
#include "../db/namespace_details.h"
```

For cpp files:

- Include mongo/pch.h first. blank line.
- Include your .h file next, if applicable. blank line.
- Include third party headers next, sorted. blank line.
- Include 10gen headers last, sorted

```
example for classy.cpp:
#include "mongo/pch.h"

#include "mongo/db/classy.h"

#include <boost/thread.h>
#include <stdio.h>
#include <string>

#include "mongo/db/db.h"
#include "mongo/db/namespace_details.h"
#include "mongo/util/concurrency/qlock.h"
```

For h files:

- #pragma once at the top
- Forward declare, if possible, in lieu of including 10gen headers in headers. Only include things that are directly used in the header itself.
- Include third party headers first, sorted. blank line.
- Include 10gen headers last, sorted.
- Be extremely careful about including pch.h in a header.

file names

- Class definitions should go in a header file with the same name as the class. Insert _ in place of a capital letter. Do not use capital letters in filenames!
  example: ClassyClass's definition goes in "classy_class.h". ClassyClass's member function implementations go in "classy_class.cpp".
- Do not be afraid to make another file, even if it is really small. This is preferable to inserting your new class into a preexisting file.

casting

- Do not use C-style casts, ever.
- Use static_cast<> or const_cast<>.
- Be aware that dynamic_cast<>, unlike other casts, is done at run-time and calls a function.
  You should always check the return status of dynamic_cast<> for null.
- reinterpret_cast<> should be used sparingly and is typically done for converting structures to raw bytes for use with I/O drivers.

RAII and bare vs. smart pointers

- Aspire to embrace RAII
- When writing functions that take or return bare pointers, document the ownership semantics in the header comment. Is the caller responsible for managing returned pointer's memory? Does the callee take ownership of the pointed-to parameter, or does the caller retain ownership. Prefer caller-retains ownership of parameters and takes ownership of returned pointers, but use the appropriate policy for each situation.
- Generally, bare calls to delete and free() are red flags
  - except in destructors
- Use smart pointers such as boost::scoped_ptr and std::auto_ptr (know the difference between them!) to avoid memory leaks and ensure all new's and malloc's are paired with delete's and free's
- Use ON_BLOCK_EXIT or ScopeGuard to protect other resources that must be released
  - e.g. fopen/fclose pairs
  - Or, write an object to do this for you via constructor and destructor

Kernel concurrency rules

All concurrency classes must be placed under `utils/concurrency`. You will find several helper libraries there.

- Do not add mutexes without discussion with others. Concurrency and correctness is very hard in the large. Great care is required. For example the concurrency model in replica sets is hard to understand and error prone (at least it was initially and probably still is).

If you think there is a real need for an exception to the list below let's have the group weigh in and get a concensus on the exception:

- Do not use/add recursive locks.
- Do not use rwlocks.
- Always acquire locks in a consistent order. In fact, the MutexDebugger can assist with verification of this. MutexDebugger is on for _DEBUG builds and will alert if locks are taken in opposing orders during the run.

Kernel exception architecture

There are several different types of assertions used in the MongoDB code. In brief:

- `uassert` checks for per-operation user errors. Operation-fatal.
- `massert` checks per-operation invariants. Operation-fatal.
- `verify` is a synonym for massert, that doesn't require an error code.
- `fassert` checks fatal process invariants. Process-fatal.
- `wassert` warn (log) and continue.

- Calling `assert` is not allowed. Use one of the above instead.
- `dassert` just calls `verify` but only in debug mode. Do not use!

When per-operation invariant checks fail, the current operation fails, but the process and connection persist. This means that `massert`, `uassert` and `verify` only terminate the current operation, not the whole process. Be careful not to corrupt process state by mistakenly using these assertions midway through mutating process state. Examples of this include uassert and massert inside of constructors and destructors. `fassert` failures will terminate the entire process; this is used for low-level checks where continuing might lead to corrupt data or loss of data on disk.

Both `massert` and `uassert` take error codes, so that all errors have codes associated with them. These error codes are assigned incrementally; the numbers have no meaning other than a way to associate a log message with a line of code. scons checks for duplicates, but if you want the next available code you can run:

```
python buildscripts/errorcodes.py
```

A failed operation-fatal assertion throws an `AssertionException` or a child of that. The inheritance hierarchy is something like:

- std::exception
  - mongo::DBException
    - mongo::AssertionException
      - mongo::UserException
      - mongo::MsgAssertionException

See `util/assert_util.h`.

Generally, code in the server should be prepared to catch a DBException. UserAssertionException's are particularly common as errors and should be expected. We use resource acquisition is initialization heavily.

Gotchas to watch out for:

- generally, don't throw a assertionexception directly. Functions like uasserted() do work beyond just that. In particular, it makes sure that the getLastError structures are set up properly.
- think about where your asserts are in constructors, as the destructor wouldn't be called. (But at a minimum, use wassert a lot therein, we want to know if something is wrong.)
- don't throw in destructors of course.

Kernel logging

- Basic Rules
  - cout/cerr should never be used
    - Except early in process startup before the log() system is initialized
      - For such cases, it would be good to use cerr, but right now there is a mix of cout and cerr
  - Include log_internal.h to use the LOG() helper macro
    - Avoid including log_internal.h in any code that may pollute C++ driver headers.
    - Use MONGO_LOG() instead of LOG() for code included by C++ driver headers.

- Normal Logging
  - Debugging with levels of verbosity. See the -v command line option (default level is 0). If the global log level is less than x, no functions in the steam are executed.

```
LOG( int x ) << ...
```

- Informational

```
log() << ...
```

- Rate limited

```
LOGSOME() << ...
```

- Warnings
  - recoverable
  - e.g. replica set node down

```
warning() << ...
```

- Errors
  - unexpected system state (disk full)
  - internal code errors

```
error() << ...
```

- Debugging Helpers
  - PRINT(x) = prints expression text and value (can also do PRINT(x.method()))
  - PRINTFL = prints file and line (good for tracing execution)
  - printStackTrace() = shows a stack trace. Alternative to using a debugger.
  - GEODEBUG, etc... = used for incredibly verbose logging for a section of code that has to be turned on at compile time

Kernel string manipulation

For string manipulation, use the `util/mongoutils/str.h` library.

`str.h`

`util/mongoutils/str.h` provides string helper functions for each manipulation. Add new functions here rather than lines and lines of code to your app that are not generic.

Typically these functions return a string and take two as parameters : string f(string,string). Thus we wrap them all in a namespace called `str`.

str::stream() is quite useful:

```
uassert(12345, str::stream() << "bad ns:" << ns, isok);
```

`StringData`

```
/** A StringData object wraps a 'const string&' or a 'const char*' without
 * copying its contents. The most common usage is as a function argument that
 * takes any of the two forms of strings above. Fundamentally, this class tries
 * go around the fact that string literals in C++ are char[N]'s.
 *
 * Note that the object StringData wraps around must be alive while the StringData
 * is.
*/
class StringData {
```

See also `bson/stringdata.h`.

mongoutils

MongoUtils has its own namespace. Its code has these basic properties:

1. are not database specific, rather, true utilities
2. are cross platform
3. may require boost headers, but not libs (header-only works with mongoutils)
4. are clean and easy to use in any c++ project without pulling in lots of other stuff
5. apache license

Memory management

Overall guidelines

- avoid using bare pointers for dynamically allocated objects. Prefer scoped_ptr, shared_ptr, or another RAII class such as BSONObj.
- do not use auto_ptr's and refactor legacy ones out whenever possible. (Careful with c++ driver and backward compatibility though.)
- If you assign the output of new/malloc() directly to a bare pointer you should document where it gets deleted/freed, who owns it along the way, and how exception safety is ensured. If you cannot answer all three questions then you probably have a leak.

Writing tests

We have four general flavors of tests currently.

General guidelines

It is important that tests can be run in parallel and still succeed.
For example, make sure that:

- try to use a unique collection name, for example named after the test

```
t = db.jstests_currentop
```

- if checking on current operations, make sure to add an NS filter

```
db.currentOp({ns: "test.mytest"})
```

- if possible, try to avoid things that are global to mongod or the database (oplog, profiling, fsync)

If the test cannot be run in parallel, add it to the blacklist in "skipTests" in shell/utils.js.

dbtests (Deprecated - avoid adding to this)

Old-style C++ unit tests are in the dbtests/ subfolder. See the code therein. To run:

```
scons test
./test --help
```

Unit tests

New-style C++ unit tests use the Unit Test framework in src/mongo/unittest. The actual unit test files appear in the same subdirectory as the source file being tested.
To run all the tests:

```
scons smokeCppUnittests
```

jstests

Many tests are written as .js scripts that are executed via the `mongo` shell. See the Smoke Tests link at the bottom for how to run comprehensive sets of tests. To run a particular test:

```
# start mongod first then run a few simple tests:
mongo jstests/basic*.js
```

Note there are several subdirectories for different test suites. slowNightly is run by the buildbots only once a night; slowWeekly only once a week. Most other tests are ran every CI cycle (all the time).

Also note that that the js tests rely on functions defined in the "shell" directory (see servers.js and utils.js in particular).

lint

`scons lint` will run cpplint with the flags we care about over the entire codebase. This is how buildbot runs lint.

To run lint on an individual file or directory, use `python buildscripts/lint.py src/mongo/platform/`

Use the `--nudge` flag to turn on warnings for rules we may turn on soon.

Lightweight startup test.

You can inherit from class `mongo::UnitTest` and make a test that runs at program startup. These tests run EVERY TIME the program starts. Thus, they should be minimal: the test should ideally take 1ms or less to run. Why run the tests in the general program? This gives some validation at program run time that the build is reasonable. For example, we test that pcre supports UTF8 regex in one of these tests at startup. If someone had built the server with other settings, this would be flagged upon execution, even if the test suite has not been invoked. Most tests are not of this sort.

See Also

- Smoke Tests
- http://buildbot.mongodb.org/

Project Ideas

If you're interested in getting involved in the MongoDB community (or the open source community in general) a great way to do so is by starting or contributing to a MongoDB related project. Here we've listed some project ideas for you to get started on. For some of these ideas projects are already underway, and for others nothing (that we know of) has been started yet.

A GUI

One feature that is often requested for MongoDB is a GUI, much like CouchDB's futon or phpMyAdmin. There are a couple of projects working on this sort of thing that are worth checking out:

http://github.com/sbellity/futon4mongo
http://www.mongodb.org/display/DOCS/Http+Interface
http://www.mongohq.com

We've also started to spec out the features that a tool like this should provide.

Try Mongo!

It would be neat to have a web version of the MongoDB Shell that allowed users to interact with a real MongoDB instance (for doing the tutorial, etc). A project that does something similar (using a basic MongoDB emulator) is here:

http://github.com/banker/mongulator

Real-time Full Text Search Integration

It would be interesting to try to nicely integrate a search backend like Xapian, Lucene or Sphinx with MongoDB. One idea would be to use MongoDB's oplog (which is used for master-slave replication) to keep the search engine up to date.

GridFS FUSE

There is a project working towards creating a FUSE filesystem on top of GridFS - something like this would create a bunch of interesting potential uses for MongoDB and GridFS:

http://github.com/mikejs/gridfs-fuse

Framework Adaptors

Working towards adding MongoDB support to major web frameworks is a great project, and work has been started on this for a variety of different frameworks (please use google to find out if work has already been started for your favorite framework).

Logging and Session Adaptors

MongoDB works great for storing logs and session information. There are a couple of projects working on supporting this use case directly.

Logging:
Zend: http://raphaelstolt.blogspot.com/2009/09/logging-to-mongodb-and-accessing-log.html
Python: http://github.com/andreisavu/mongodb-log
Rails: http://github.com/peburrows/mongo_db_logger

Sessions:
web.py: http://github.com/whilefalse/webpy-mongodb-sessions
Beaker: http://pypi.python.org/pypi/mongodb_beaker

Package Managers

Add support for installing MongoDB with your favorite package manager and let us know!

Locale-aware collation / sorting

MongoDB doesn't yet know how to sort query results in a locale-sensitive way. If you can think up a good way to do it and implement it, we'd like to know!

Drivers

If you use an esoteric/new/awesome programming language write a driver to support MongoDB! Again, check google to see what people have started for various languages.

Some that might be nice:

- Scheme (probably starting with PLT)
- GNU R
- Visual Basic
- Lisp (e.g, Common Lisp)
- Delphi
- Falcon

Write a killer app that uses MongoDB as the persistance layer!

Roadmap

Please see jira.

UI

Spec/requirements for a future MongoDB admin UI.

- list databases
    - repair, drop, clone?
- collections
    - validate(), datasize, indexsize, clone/copy
- indexes
- queries - explain() output
- security: view users, adjust
- see replication status of slave and master
- sharding
- system.profile viewer ; enable disable profiling
- curop / killop support

Source Code

Source for MongoDB and mongodb.org supported drivers is open source and hosted at Github .

- Mongo Database (includes C++ driver)
- Python Driver

- PHP Driver
- Ruby Driver
- Java Driver
- Perl Driver
- C# Driver
- Scala Driver
- Erlang Driver
- Haskell Driver

(Additionally, community drivers and tools also exist and will be found in other places.)

See Also

- Building
- License

Building

Note: see the Downloads page for prebuilt binaries, it's recommended to use those as all full QA occurs after those are built.

This section provides instructions on setting up your environment to write Mongo drivers or other infrastructure code. For specific instructions, go to the document that corresponds to your setup.

Sub-sections of this section:

- Building Boost
- Building for FreeBSD
- Building for Linux
- Building for OS X
- Building for Solaris
- Building for Windows
- Building Spider Monkey
- Building with V8
- scons

See Also

- The main Database Internals page

Building Boost

NOTE
This is not necessary when building mongo versions 2.1.1 or later.

MongoDB uses the [www.boost.org] C++ libraries.

Windows

See also the prebuilt libraries page.

By default c:\boost\ is checked for the boost files. Include files should be under \boost\boost, and libraries in \boost\lib.

First download the boost source. Then use the 7 Zip utility to extra the files. Place the extracted files in C:\boost.

Then we will compile the required libraries.

See buildscripts/buildboost.bat and buildscripts/buildboost64.bat for some helpers.

```
> rem set PATH for compiler:
> "C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\vcvarsall.bat"
>
> rem build the bjam make tool:
> cd \boost\tools\jam\src\
> build.bat
>
> cd \boost
> tools\jam\src\bin.ntx86\bjam --help
> rem see also mongo/buildscripts/buildboost*.bat
> rem build DEBUG libraries:
> tools\jam\src\bin.ntx86\bjam variant=debug threading=multi --with-program_options --with-filesystem
--with-date_time --with-thread
> mkdir lib
> move stage\lib\* lib\
```

Linux

It's common with linux to install boost via a package manager – see the Building for Linux page for specifics.

However one might also want to build from boost.org sources in some cases, for example to use a newer version of boost, to generate static libraries, etc.

The following – far from comprehensive, rather an example – shows manually building of boost libraries on Linux.

```
$ sudo ./bootstrap.sh

$ ./b2 --help

$ # now build it
$ ./b2
$ #or
$ ./b2 --with-program_options --with-filesystem --with-date_time --with-thread

$ sudo ./b2 install
```

Troubleshooting

Unresolved external get_system_category() when linking MongoDB
Try defining BOOST_SYSTEM_NO_DEPRECATED in the MongoDB SConstruct file.

Building for FreeBSD

On FreeBSD 8.0 and later, there is a mongodb port you can use.

For FreeBSD <= 7.2:

1. Get the database source: http://www.github.com/mongodb/mongo.
2. Update your ports tree:

   ```
   $ sudo portsnap fetch && portsnap extract
   ```

   The packages that come by default on 7.2 and older are too old, you'll get weird errors when you try to run the database)
3. Install SpiderMonkey:

   ```
   $ cd /usr/ports/lang/spidermonkey && make && make install
   ```

4. Install scons:

   ```
   $ cd /usr/ports/devel/scons && make && make install
   ```

5. Install boost: (it will pop up an X "GUI", select PYTHON)

```
$ cd /usr/ports/devel/boost-all && make && make install
```

6. Install libunwind:

```
$ cd /usr/ports/devel/libunwind && make && make install
```

7. Change to the database source directory
8. scons all

See Also

- Building for Linux - many of the details there including how to clone from git apply here too.

Building for Linux

Note: Binaries are available for most platforms. Most users won't need to compile mongo themselves; in addition every prebuilt binary has been regression tested. See the Downloads page for these prebuilt binaries.

- Build Prerequisites
  - Fedora
  - Ubuntu
- Building
- Building Older Versions
  - Fedora
  - Ubuntu
    - Ubuntu 8.04 and 8.10
    - Ubuntu 9.04 and Newer
  - Building
  - Troubleshooting

These instructions apply to the git master branch, and versions 2.1.1 and onward. At the end of the document are instructions for older versions.

Build Prerequisites

You'll need SCons, the gnu C++ toolchain, and glibc-devel. To get the code from github, you'll probably also want git.

Fedora

```
sudo yum -y install git-core scons gcc-c++ glibc-devel
```

Ubuntu

```
sudo apt-get install git-core build-essential scons
```

Building

1. Install any needed prerequisites (see above).
2. Get the source code

```
git clone git://github.com/mongodb/mongo.git
cd mongo
```

3. Pick a version to build (only use "master" if you're doing development work).
   a. List all tagged versions

```
git tag -l
```

b. Check out a tagged release, e.g. 2.0.4

```
git checkout r2.0.4
```

4. Compile

```
scons all
```

5. Install. Use --prefix to specify where you want to install your binaries. Defaults to /usr/local.

```
scons --prefix=/opt/mongo  install
```

Building Older Versions

Fedora

The following steps have been reported to work for Fedora versions from 11 to 16. (If they don't work on newer versions, please report this to mongodb-user@googlegroups.com.) Fedora versions 10 and earlier ship with a version of SCons that is too old to build MongoDB, but may work if you manually install SCons.

```
sudo yum -y install git-core scons gcc-c++ glibc-devel
sudo yum -y install boost-devel boost-devel-static
```

Ubuntu

Note: See SpiderMonkey note above.

Use cat /etc/lsb-release to see your version.

Ubuntu 8.04 and 8.10

```
apt-get -y install git-core build-essential
apt-get -y install libboost-dev libboost-program-options-dev libboost-thread-dev
libboost-filesystem-dev
```

Ubuntu 8.04 and 8.10 ship with a version of SCons that is too old to build MongoDB, but may work if you manually install SCons.

Ubuntu 9.04 and Newer

```
aptitude install -y git-core build-essential scons
aptitude install -y libboost-dev libboost-program-options-dev libboost-thread-dev
libboost-filesystem-dev
```

Building

1. Install prerequisites
2. get source

```
git clone git://github.com/mongodb/mongo.git
cd mongo
# pick a stable version unless doing true dev
git tag -l
# Switch to a stable branch (unless doing development) --
# an even second number indicates "stable". For example:
git checkout r2.0.4
```

3. build

```
scons all
```

4. install --prefix can be anywhere you want to contain your binaries, e.g., /usr/local or /opt/mongo.

```
scons --prefix=/opt/mongo install
```

Troubleshooting

- Link errors. If link errors occur, the `-t` gcc option is useful for troubleshooting. Try adding `-t` to the SConstruct file's `LINKFLAGS`.
- Static libraries. The `--release` scons option will build a binary using static libraries. You may need to install static boost libraries when using this option.

Building for OS X

- Prerequisites
  - Xcode
  - SCons
  - git
- Building
- Older Versions of Mongo
  - Upgrading to Snow Leopard
  - Setup
    - Sources
    - Prerequisites
    - Package Manager Setup
    - Manual Setup
      - Install Apple developer tools
      - Install libraries (32-bit option)
      - Install libraries (64-bit option)
  - Compiling
  - Troubleshooting

To set up your OS X computer for MongoDB development:

Prerequisites

Xcode

Available in the App Store. You only need to get the command line tools, if you don't want to install the whole IDE.

SCons

SCons is the build tool used to compile mongo binaries. It is available from http://www.scons.org.

If you have easy_install or pip already installed, you can use them to install scons.

```
easy_install scons
```

```
pip install scons
```

git

An installer package is available from http://git-scm.com/

Building

1. Install any needed prerequisites (see above).
2. Get the source code

```
git clone git://github.com/mongodb/mongo.git
cd mongo
```

3. Pick a version to build (only use "master" if you're doing development work).
   a. List all tagged versions

```
git tag -l
```

   b. Check out a tagged release, e.g. 2.0.4

```
git checkout r2.0.4
```

4. Compile

```
scons all
```

5. Install. Use --prefix to specify where you want to install your binaries. Defaults to /usr/local.

```
scons --prefix=/opt/mongo  install
```

Older Versions of Mongo

Upgrading to Snow Leopard

If you have installed Snow Leopard, the builds will be 64-bit -- so if moving from a previous OS release, a bit more setup may be required than one might first expect.

Setup

Sources

The mongodb source is on github. To get sources first download the git client and install it.

- Then git clone git://github.com/mongodb/mongo.git (more info)
  Note If you do not wish to install git you can instead get the source code from the Downloads page.

Prerequisites

- Install gcc. Install XCode tools for Snow Leopard. gcc version 4.0.1 (from older XCode Tools install) works, but you will receive compiler warnings. One way to get a newer gcc is to install Command Line Tools for XCode from developer.apple.com.

Package Manager Setup

1. Install Homebrew
2. Update/install dependencies:

```
brew install boost
```

3. Install SCons:

```
brew install scons
```

Manual Setup

Install Apple developer tools

Install libraries (32-bit option)

1. Download boost 1.37.0. Apply the following patch:

```
diff -u -r a/configure b/configure
--- a/configure 2009-01-26 14:10:42.000000000 -0500
+++ b/configure 2009-01-26 10:21:29.000000000 -0500
@@ -9,9 +9,9 @@

 BJAM=""
 TOOLSET=""
-BJAM_CONFIG=""
+BJAM_CONFIG="--layout=system"
 BUILD=""
 PREFIX=/usr/local
 EPREFIX=
diff -u -r a/tools/build/v2/tools/darwin.jam b/tools/build/v2/tools/darwin.jam
--- a/tools/build/v2/tools/darwin.jam   2009-01-26 14:22:08.000000000 -0500
+++ b/tools/build/v2/tools/darwin.jam   2009-01-26 10:22:08.000000000 -0500
@@ -367,5 +367,5 @@

 actions link.dll bind LIBRARIES
 {
-    "$(CONFIG_COMMAND)" -dynamiclib -Wl,-single_module -install_name "$(<:B)$(<:S)" -L
"$(LINKPATH)" -o "$(<)" "$(>)" "$(LIBRARIES)" -l$(FINDLIBS-SA) -l$(FINDLIBS-ST)
$(FRAMEWORK_PATH) -framework$(_)$(FRAMEWORK:D=:S=) $(OPTIONS) $(USER_OPTIONS)
+    "$(CONFIG_COMMAND)" -dynamiclib -Wl,-single_module -install_name
"/usr/local/lib/$(<:B)$(<:S)" -L"$(LINKPATH)" -o "$(<)" "$(>)" "$(LIBRARIES)" -l$(FINDLIBS-SA)
-l$(FINDLIBS-ST) $(FRAMEWORK_PATH) -framework$(_)$(FRAMEWORK:D=:S=) $(OPTIONS) $(USER_OPTIONS)
 }
```

   then,

```
./configure; make; sudo make install
```

2. Install pcre (must enable UTF8)

```
./configure --enable-utf8 --enable-unicode-properties  --with-match-limit=200000
--with-match-limit-recursion=4000; make; sudo make install
```

3. Install c++ unit test framework http://unittest.red-bean.com/ (optional)

```
./configure; make; sudo make install
```

Install libraries (64-bit option)

(The 64-bit libraries will be installed in /usr/64/{include,lib}.)

1. Download SpiderMonkey: ftp://ftp.mozilla.org/pub/mozilla.org/js/js-1.7.0.tar.gz

Apply the following patch:

```
diff -u  -r js/src/config/Darwin.mk js-1.7.0/src/config/Darwin.mk
--- js/src/config/Darwin.mk 2007-02-05 11:24:49.000000000 -0500
+++ js-1.7.0/src/config/Darwin.mk 2009-05-11 10:18:37.000000000 -0400
@@ -43,7 +43,7 @@
 # Just ripped from Linux config
 #

-CC = cc
+CC = cc -m64
 CCC = g++
 CFLAGS +=  -Wall -Wno-format
 OS_CFLAGS = -DXP_UNIX -DSVR4 -DSYSV -D_BSD_SOURCE -DPOSIX_SOURCE -DDARWIN
@@ -56,9 +56,9 @@
 #.c.o:
 #      $(CC) -c -MD $*.d $(CFLAGS) $<

-CPU_ARCH = $(shell uname -m)
+CPU_ARCH = "X86_64"
 ifeq (86,$(findstring 86,$(CPU_ARCH)))
-CPU_ARCH = x86
+CPU_ARCH = x86_64
 OS_CFLAGS+= -DX86_LINUX
 endif
 GFX_ARCH = x
@@ -81,3 +81,14 @@
 # Don't allow Makefile.ref to use libmath
 NO_LIBM = 1

+ifeq ($(CPU_ARCH),x86_64)
+# Use VA_COPY() standard macro on x86-64
+# FIXME: better use it everywhere
+OS_CFLAGS += -DHAVE_VA_COPY -DVA_COPY=va_copy
+endif
+
+ifeq ($(CPU_ARCH),x86_64)
+# We need PIC code for shared libraries
+# FIXME: better patch rules.mk & fdlibm/Makefile*
+OS_CFLAGS += -DPIC -fPIC
+endif
```

compile and install

```
cd src
make -f Makefile.ref
sudo JS_DIST=/usr/64 make -f Makefile.ref export
```

remove the dynamic library

```
sudo rm /usr/64/lib64/libjs.dylib
```

Download boost 1.37.0 Apply the following patch:

```
diff -u -r a/configure b/configure
--- a/configure 2009-01-26 14:10:42.000000000 -0500
+++ b/configure 2009-01-26 10:21:29.000000000 -0500
@@ -9,9 +9,9 @@

 BJAM=""
 TOOLSET=""
-BJAM_CONFIG=""
+BJAM_CONFIG="architecture=x86 address-model=64 --layout=system"
 BUILD=""
-PREFIX=/usr/local
+PREFIX=/usr/64
 EPREFIX=
 LIBDIR=
 INCLUDEDIR=
diff -u -r a/tools/build/v2/tools/darwin.jam b/tools/build/v2/tools/darwin.jam
--- a/tools/build/v2/tools/darwin.jam   2009-01-26 14:22:08.000000000 -0500
+++ b/tools/build/v2/tools/darwin.jam   2009-01-26 10:22:08.000000000 -0500
@@ -367,5 +367,5 @@

 actions link.dll bind LIBRARIES
 {
-    "$(CONFIG_COMMAND)" -dynamiclib -Wl,-single_module -install_name "$(<:B)$(<:S)" -L"$(LINKPATH)"
-o "$(<)" "$(>)" "$(LIBRARIES)" -l$(FINDLIBS-SA) -l$(FINDLIBS-ST) $(FRAMEWORK_PATH)
-framework$(_)$(FRAMEWORK:D=:S=) $(OPTIONS) $(USER_OPTIONS)
+    "$(CONFIG_COMMAND)" -dynamiclib -Wl,-single_module -install_name "/usr/64/lib/$(<:B)$(<:S)" -L
"$(LINKPATH)" -o "$(<)" "$(>)" "$(LIBRARIES)" -l$(FINDLIBS-SA) -l$(FINDLIBS-ST) $(FRAMEWORK_PATH)
-framework$(_)$(FRAMEWORK:D=:S=) $(OPTIONS) $(USER_OPTIONS)
 }
```

then,

```
./configure; make; sudo make install
```

Install pcre (must enable UTF8)

```
CFLAGS="-m64" CXXFLAGS="-m64" LDFLAGS="-m64" ./configure --enable-utf8 --with-match-limit=200000
--with-match-limit-recursion=4000 --enable-unicode-properties --prefix /usr/64; make; sudo make
install
```

Install unit test framework http://unittest.red-bean.com/ (optional)

```
CFLAGS="-m64" CXXFLAGS="-m64" LDFLAGS="-m64" ./configure --prefix /usr/64; make; sudo make install
```

Compiling

To compile 32-bit, just run:

```
scons
```

To compile 64-bit on 10.5 (64 is default on 10.6), run:

```
scons  --64
```

See the, MongoDB scons page for more details/compile options.

Troubleshooting

- Undefined symbols: "_PR_NewLock", referenced from: _JS_Init in libjs.a.
    - Try not using the scons `--release` option (if you are using it). That option attempts to use static libraries.

Building for Solaris

MongoDB server currently supports little endian Solaris operation. (Although most drivers – not the database server – work on both.)

Community: Help us make this rough page better please! (And help us add support for big endian please...)

Prerequisites:

- g++ 4.x (SUNWgcc)
- scons (need to install from source)
- spider monkey Building Spider Monkey
- pcre (SUNWpcre)
- boost (need to install from source)

See Also

- Joyent
- Building for Linux - many of the details there including how to clone from git apply here too

Building for Windows

Binaries are available for most platforms. Most users won't need to compile mongo themselves; in addition every prebuilt binary has been regression tested. See the Downloads page for these prebuilt binaries.

MongoDB can be compiled for Windows (32 and 64 bit); You will need to have the platform sdk installed. The platform sdk can be installed manually, and it comes with Visual (Studio) C++ as well. SCons is the make mechanism, although several .vcxprojs and a .sln solution file are also included in the project for convenience when using the Visual Studio 2010 IDE.

There are several dependencies exist which are listed below; you may find it easier to simply download a pre-built binary.

- Building with Visual Studio 2008
- Building with Visual Studio 2010
- Building the Shell

See Also

- Windows Quick Links
- scons

Boost 1.41.0 Visual Studio 2010 Binary

This is OLD and was for the VS2010 BETA. See the new Boost and Windows page instead.

The following is a prebuilt boost binary (libraries) for Visual Studio 2010 beta 2.

The MongoDB vcxproj files assume this package is unzipped under c:\Program Files\boost\boost_1_41_0\.

- http://downloads.mongodb.org/misc/boost_1_41_0_binary_vs10beta2.zipx

Note: we're not boost build gurus please let us know if there are things wrong with the build.

See also the prebuilt boost binaries at http://www.boostpro.com/download.

Boost and Windows

- Visual Studio 2010
    - Prebuilt from mongodb.org
    - Building Yourself
- Visual Studio 2008
    - Prebuilt from mongodb.org
    - Prebuilt from boostpro.com
    - Building Yourself
- Additional Notes

Visual Studio 2010

Prebuilt from mongodb.org

Click here for a prebuilt boost library for Visual Studio 2010. 7zip format.

Building Yourself

- Download the boost source from boost.org. Move it to C:\boost\.
  - We have successfully compiled version **1.42** – you might want to try that version or higher, but not 1.45 or later. 1.45 changed the interface to the boost::filesystem library and we've yet to catch up. See additional notes section at end of this page too.
- Run `C:\Program Files (x86)\Microsoft Visual Studio 10.0\vc\vcvarsall.bat`.
- From the MongoDB source project, run buildscripts\buildboost.bat. Or, buildboost64.bat for the 64 bit version.

Visual Studio 2008

Prebuilt from mongodb.org

Click here for a prebuilt boost library for Visual Studio 2008. 7zip format. This file has what you need to build MongoDB, but not some other boost libs, so it's partial.

Prebuilt from boostpro.com

Or, you can download a complete prebuilt boost library for 32 bit VS2008 at http://www.boostpro.com/products/free. Install the prebuilt libraries for Boost version 1.35.0 (or higher - generally newer is better). During installation, for release builds choose `static multithread libraries for installation`. The Debug version of the project uses the DLL libraries; choose all multithread libraries if you plan to do development. From the BoostPro installer, be sure to select all relevant libraries that mongodb uses -- for example, you need Filesystem, Regex, Threads, and ProgramOptions (and perhaps others).

Building Yourself

- Download the boost source from boost.org. Move it to C:\boost\.
- From the Visual Studio 2008 IDE, choose Tools.Visual Studio Command Prompt to get a command prompt with all PATH variables set nicely for the C++ compiler.
- From the MongoDB source project, run buildscripts\buildboost.bat. Or, buildboost64.bat for the 64 bit version.

Additional Notes

When using bjam, MongoDB expects

- `variant=debug` for debug builds, and `variant=release` for release builds
- `threading=multi`
- `link=static runtime-link=static` for release builds
- `address-model=64` for 64 bit

Building the Mongo Shell on Windows

You can build the mongo shell with either scons or a Visual Studio 2010 project file.

Scons

```
scons mongo.exe
```

Visual Studio 2010 Project File

A VS2010 vcxproj file is availabe for building the shell. From the mongo directory open shell/msvc/mongo.vcxproj.

The project file currently only supports 32 bit builds of the shell (scons can do 32 and 64 bit). However this seems sufficient given there is no real need for a 64 bit version of the shell.

Building with Visual Studio 2008

NOTE

These instructions are for versions of mongo prior to 2.1.1. For version 2.1.1 and newer, the instructions for Visual Studio 2010 and Visual Studo 2008 are the same.

- Get the MongoDB Source Code
- Get Boost Libraries
- Get SpiderMonkey
- Install SCons
- Building MongoDB with SCons
- Troubleshooting

MongoDB can be compiled for Windows (32 and 64 bit) using Visual C++. SCons is the make mechanism we use with VS2008. (Although it is possible to build from a sln file with vs2010.)

There are several dependencies exist which are listed below; you may find it easier to simply download a pre-built binary.

Get the MongoDB Source Code

Download the source code from Downloads.

Or install Git. Then:

- `git clone git://github.com/mongodb/mongo.git` (more info)
- `git tag -l` to see tagged version numbers
- Switch to a stable branch (unless doing development) -- an even second number indicates "stable". (Although with sharding you will want the latest if the latest is less than 1.6.0.) For example:
    - `git checkout r1.4.1`

Get Boost Libraries

- Click here for a prebuilt boost library for Visual Studio. 7zip format. This file has what you need to build MongoDB, but not some other boost libs, so it's partial.
- See the Boost and Windows page for other options.

The Visual Studio project files are setup to look for boost in the following locations:

- c:\program files\boost\latest
- c:\boost
- \boost

You can unzip boost to c:\boost, or use an NTFS junction point to create a junction point to one of the above locations. Some versions of windows come with linkd.exe, but others require you to download Sysinternal's junction.exe to accomplish this task. For example, if you installed boost 1.42 via the installer to the default location of **c:\Program Files\boost\boost_1_42**, You can create a junction point with the following command:

```
junction "c:\Program Files\boost\latest" "c:\Program Files\boost\boost_1_42"
```

This should return the following output:

```
Junction v1.05 - Windows junction creator and reparse point viewer
Copyright (C) 2000-2007 Mark Russinovich
Systems Internals - http://www.sysinternals.com

Created: c:\Program Files\boost\latest
Targetted at: c:\Program Files\boost\boost_1_42
```

Get SpiderMonkey

Build a SpiderMonkey js engine library (js.lib) – details here.

Install SCons

If building with scons, install SCons:

- First install Python: http://www.python.org/download/releases/2.6.4/.
- Then SCons itself: http://sourceforge.net/projects/scons/files/scons/1.2.0/scons-1.2.0.win32.exe/download.
- Add the python scripts directory (e.g., C:\Python26\Scripts) to your PATH.

Building MongoDB with SCons

The SConstruct file from the MongoDB project is the preferred way to perform production builds. Run scons in the mongo project directory to build.

If scons does not automatically find Visual Studio, preset your path for it by running the VS2010 vcvars*.bat file.

To build:

```
scons                  // build mongod
scons mongoclient.lib  // build C++ client driver library
scons all              // build all end user components
scons .                // build all including unit test
```

Troubleshooting

If you are using `scons`, check the file `config.log` which is generated.

- **Can't find jstypes.h when compiling.** This file is generated when building SpiderMonkey. See the Building SpiderMonkey page for more info.
- **Can't find / run cl.exe when building with scons.** See troubleshooting note on the Building SpiderMonkey page.
- **Error building program database.** (VS2008.) Try installing the Visual Studio 2008 Service Pack 1.

Building with Visual Studio 2010

Binaries are available for most platforms. Most users won't need to compile mongo themselves; in addition every prebuilt binary has been regression tested. See the Downloads page for these prebuilt binaries.

- v2.1.1+
  - Get the MongoDB Source Code
  - Building MongoDB from the IDE
  - Building with SCons
  - Troubleshooting
- Older versions
  - Get the MongoDB Source Code
  - Get Boost Libraries
  - Get SpiderMonkey
  - Building MongoDB from the IDE
  - Install SCons
  - Troubleshooting

v2.1.1+

MongoDB can be compiled for Windows (32 and 64 bit) using Visual C++. SCons is the make mechanism, although a solution file is also included in the project for convenience when using the Visual Studio IDE. (These instructions don't work using Visual Studio Express, which must be uninstalled to get Visual Studio Professional/Ultimate to work properly; VSE can only do 32 bit builds.)

These instructions are for mongo versions 2.1.1 and later.

Get the MongoDB Source Code

Download the source code from Downloads.

Or install Git. Then:

- `git clone git://github.com/mongodb/mongo.git` (more info)
- `git tag -l` to see tagged version numbers
- Switch to a stable branch (unless doing development) -- an even second number indicates "stable". For example:
  - `git checkout r1.4.1`

Building MongoDB from the IDE

Open the db\db_10.sln solution file.

Note: a separate project file exists for the `mongo` shell. Currently the C++ client libraries must be built from scons (this obviously needs to be fixed...)

Building with SCons

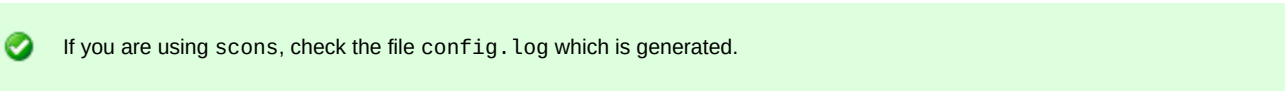
1. Install SCons:
    a. First install Python: http://www.python.org/download/releases/2.7.2/.
        - **Note** Make sure to install the 32 bit version of python and not the 64 bit as the scons binaries below are 32 bit.
    b. It is recommended you install pywin32 if you want to do parallel builds (scons -j).
    http://sourceforge.net/projects/pywin32/files/pywin32/Build216/pywin32-216.win32-py2.7.exe/download
    c. Then install SCons itself: http://sourceforge.net/projects/scons/files/scons/2.1.0/scons-2.1.0.win32.exe/download.
    d. Add the python scripts directory (e.g., C:\Python27\Scripts) to your PATH.

1. Build:

```
scons                   // build mongod
scons --release mongoclient.lib   // build C++ client driver library
scons --release core               // build all end user components
```

Add --64 or --32 to get the 64 and 32-bit versions, respectively. Replace --release with --dd to build a debug build.

Troubleshooting



If you are using `scons`, check the file `config.log` which is generated.

If scons does not automatically find Visual Studio, try using the Visual Studio Command Prompt, which will set your path for you. Alternatively, set your path manually by running the VS2010 vcvars*.bat files. Location may vary with install but usually it is something like:

- C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\Tools\vsvars32.bat
- C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\amd64\vcvars64.bat

Older versions

The following instructions are for versions of mongo prior to 2.1.1.

There are several dependencies exist which are listed below; you may find it easier to simply download a pre-built binary.

Get the MongoDB Source Code

Download the source code from Downloads.

Or install Git. Then:

- `git clone git://github.com/mongodb/mongo.git` (more info)
- `git tag -l` to see tagged version numbers
- Switch to a stable branch (unless doing development) -- an even second number indicates "stable". (Although with sharding you will want the latest if the latest is less than 1.6.0.) For example:
    - `git checkout r1.4.1`

Get Boost Libraries

- Click here for a prebuilt boost library for Visual Studio. 7zip format. This file has what you need to build MongoDB, but not some other boost libs, so it's partial. Uncompress this to the c:\boost directory. Your actual files are in c:\boost\boost
- See the Boost and Windows page for other options. Use v1.42 or higher with VS2010.

Get SpiderMonkey

- Download prebuilt libraries and headers here for VS2010. Place these files in ..\js\ relative to your mongo project directory.
- Or (more work) build SpiderMonkey js.lib yourself – details here.

Building MongoDB from the IDE

Open the db\db_10.sln solution file.

Note: a separate project file exists for the `mongo` shell. Currently the C++ client libraries must be built from scons (this obviously needs to be fixed...)

Install SCons

If building with scons, install SCons:

- First install Python: http://www.python.org/download/releases/2.7.2/.
  - **Note** Make sure to install the 32 bit version of python and not the 64 bit as the scons binaries below are 32 bit.
- Its recommended you install pywin32 if you want to do parallel builds (scons -j). http://sourceforge.net/projects/pywin32/files/pywin32/Build216/pywin32-216.win32-py2.7.exe/download
- Then install SCons itself: http://sourceforge.net/projects/scons/files/scons/2.1.0/scons-2.1.0.win32.exe/download.
- Add the python scripts directory (e.g., C:\Python27\Scripts) to your PATH.

The SConstruct file from the MongoDB project is the preferred way to perform production builds. Run scons in the mongo project directory to build.

If scons does not automatically find Visual Studio, preset your path for it by running the VS2010 vcvars*.bat files. Location may vary with install but usually it is something like:

- C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\Tools\vsvars32.bat
- C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\amd64\vcvars64.bat

To build:

```
scons                   // build mongod
scons mongoclient.lib   // build C++ client driver library
scons all               // build all end user components
scons .                 // build all including unit tests and C++ driver zip file
scons --64              // build the 64 bit version
scons --dd              // build with debug symbols
scons -jX               // perform X steps in parallel (e.g. -j16 will compile 16 files at once)
```

Troubleshooting

If you are using `scons`, check the file `config.log` which is generated.

- Can't find jstypes.h when compiling.
  - This file is generated when building SpiderMonkey. See the Building SpiderMonkey page for more info.
- Can't find / run cl.exe when building with scons.
  - Be sure to use Visual Studio Command Prompt so that your path is set correctly.
- LINK : fatal error LNK1104: cannot open file js64d.lib js64r.lib js32d.lib js32r.lib
  - Get the prebuilt spidermonkey libraries -- or copy your self-built js.lib to the above name.
  - You can also see this if you're using the wrong compiler; this is the result if you try to use Visual Studio Express instead of Visual Studio Professional/Ultimate, which is a different product.

Building Spider Monkey

- Building js.lib - Unix
  - Remove any existing xulrunner
  - Download
  - Build
  - Install
- Building js.lib - Windows
  - Prebuilt
  - Download
  - Build
  - Troubleshooting scons
- See Also

MongoDB uses SpiderMonkey for server-side Javascript execution.

Pre v2.0: MongoDB requires a js.lib file when linking. This page details how to build js.lib.

v2.0+: this is handled automatically by the Mongo build scripts via files under third_party/ in the MongoDB project directory.

Note: V8 Javascript support is under development.

Building js.lib - Unix

Remove any existing xulrunner

First find out what has been installed

```
dpkg -l | grep xulrunner
```

e.g.

```
ubuntu910-server64:mongo$ sudo dpkg -l | grep xul
ii  xulrunner-1.9.1                  1.9.1.13+build1+nobinonly-0ubuntu0.9.10.1 XUL + XPCOM
application runner
ii  xulrunner-1.9.1-dev              1.9.1.13+build1+nobinonly-0ubuntu0.9.10.1 XUL + XPCOM
development files
```

Next remove the two installed packages

```
sudo apt-get remove xulrunner-1.9.1-dev xulrunner-1.9.1
```

Download

```
curl -O ftp://ftp.mozilla.org/pub/mozilla.org/js/js-1.7.0.tar.gz
tar zxvf js-1.7.0.tar.gz
```

Build

```
cd js/src
export CFLAGS="-DJS_C_STRINGS_ARE_UTF8"
make -f Makefile.ref
```

SpiderMonkey does not use UTF-8 by default, so we enable before building.

An experimental SConstruct build file is available here.

Install

```
JS_DIST=/usr make -f Makefile.ref export
```

By default, the mongo scons project expects spidermonkey to be located at ../js/.

Building js.lib - Windows

Prebuilt

- VS2008: a prebuilt SpiderMonkey library and headers for Win32 is attached to this document (this file may or may not work depending on your compile settings and compiler version).
- VS2010 prebuilt libraries (js64d.lib, etc.)

Alternatively, follow the steps below to build yourself.

Download

From an msysgit or cygwin shell, run:

```
curl -O ftp://ftp.mozilla.org/pub/mozilla.org/js/js-1.7.0.tar.gz
tar zxvf js-1.7.0.tar.gz
```

Build

cd js/src

```
export CFLAGS="-DJS_C_STRINGS_ARE_UTF8"
make -f Makefile.ref
```

If cl.exe is not found, launch Tools...Visual Studio Command Prompt from inside Visual Studio -- your path should then be correct for make.

If you do not have a suitable make utility installed, you may prefer to build using scons. An experimental SConstruct file to build the js.lib is available in the mongodb/snippets project. For example:

```
cd
git clone git://github.com/mongodb/mongo-snippets.git
cp mongo-snippets/jslib-sconstruct js/src/SConstruct
cd js/src
scons
```

Troubleshooting scons

Note that scons does not use your PATH to find Visual Studio. If you get an error running cl.exe, try changing the following line in the msvc.py scons source file from:

```
MVSdir = os.getenv('ProgramFiles') + r'\Microsoft Visual Studio 8'
```

to

```
MVSdir = os.getenv('ProgramFiles') + r'\Microsoft Visual Studio ' + version
```

See Also

- Building MongoDB

Building with V8

Linux or OSX

```
$ pwd
~/mongo
$ cd ..
$ svn checkout http://v8.googlecode.com/svn/trunk/ v8
$ cd v8
$ scons arch=x64 debuggersupport=off snapshot=off profilingsupport=off
$ cd ../mongo
$ scons --usev8
```

Windows

```
$ pwd
~/mongo
$ cd ..
$ svn checkout http://v8.googlecode.com/svn/trunk/ v8
$ cd v8
$ scons env="PATH:%PATH%,INCLUDE:%INCLUDE%,LIB:%LIB%" arch=x64
$ cd ../mongo
$ scons --usev8 --release --64
```

scons

Use scons to build MongoDB and related utilities and libraries. See the SConstruct file for details.

Run `scons --help` to see all options.

Targets

Run `scons <target>`.

- `scons all`
- `scons mongod` build mongod (this is the default target if none is specified)
- `scons mongo` build the shell
- `scons mongoclient` build just the client library (builds libmongoclient.a on Unix)
- `scons test` build the unit test binary `test`

Options

- `--d` debug build; all this does is turns optimization off
- `--dd` debug build with _DEBUG defined (extra asserts, checks, etc.)
- `--release`
- `--32` force 32 bit
- `--64` force 64 bit
- `--clean` cleans the target specified
- `--mute` suppress compile and link command lines

Troubleshooting

scons generates a `config.log` file. See this file when there are problems building.

See Also

Smoke Tests

Database Internals

This section provides information for developers who want to write drivers or tools for MongoDB, \ contribute code to the MongoDB codebase itself, and for those who are just curious how it works internally.

Sub-sections of this section:

- Durability Internals
- Parsing Stack Traces
- Error Codes
- Replication Internals
- Smoke Tests
- Pairing Internals

Durability Internals

The main durability page (not the internals page) is the Journaling page.

- Files
- Running
- Declaring Write Intent
- Tests
- Administrative
- Diagrams

Files

The data file format is unchanged.

Journal files are placed in */data/db/journal/*.

Running

Journaling is on by default. Run with `--nojournal` to disable journaling/durable storage. Both `mongod` and `test` support this option.

Declaring Write Intent

When writing `mongod` kernel code, one must now declare an intention to write. Declaration of the intent occurs before the actual write. See `db/dur.h`. The actual write must occur before releasing the write lock.

When you do your actual writing, use the pointer that dur::writing() returns, rather than the original pointer.

```
Foo *foo;
getDur().writing(thing)->bar = something;

int *x;
getDur().writingInt(x) += 3;

DiskLoc &loc;
loc.writing() = newLoc;

void *p;
unsigned len;
memcpy( getDur().writingPtr(p,len), src, len );
```

Try to declare intent on as small a region as possible. That way less information is journalled. For example

```
BigStruct *b;

dur::writing(b)->x = 3; // less efficient

*dur::writing(&b->x) = 3; // more efficient
```

However, there is some overhead for each intent declaration, so if many members of a struct will be written, it is likely better to just declare intent on the whole struct.

Tests

`jstests/dur/` contains tests for durability.

```
mongo --nodb jstests/dur/<testname>.js
```

Administrative

```
# dump journal entries during any recover, and then start normally
mongod --journal --durOptions 1

# recover and terminate
mongod --journal --durOptions 4

# dump journal entries (doing nothing else) and then terminate
mongod --journal --durOptions 7

# extra checks that everything is correct (slow but good for qa)
mongod --journal --durOptions 8
```

Diagrams

- diagram 1 - process steps
- diagram 2 - journal file structure

Parsing Stack Traces

addr2line

```
addr2line -e mongod -ifC <offset>
```

c++filt

You can use `c++filt` to demangle function names by pasting the whole stack trace to stdin.

Finding the right binary

To find the binary you need:

- Get the commit at the header of any of our logs.
- Use git to locate that commit and check for the surrounding "version bump" commit

Download and open the binary:

```
curl -O http:
//s3.amazonaws.com/downloads.mongodb.org/linux/mongodb-linux-x86_64-debugsymbols-1.x.x.tgz
```

You can also find debugsymbols for official builds by clicking "list" on the Downloads page.

Example 1

Then, the log has lines like this:

```
/home/abc/mongod(_ZN5mongo15printStackTraceERSo+0x27) [0x689280]
```

You want the address in between the brackets [0x689280]

Note you will get more than one stack frame for the address if the code is inlined.

Example 2

Actual example from a v1.8.1 64 bit linux build:

```
$ curl http://downloads.mongodb.org/linux/mongodb-linux-x86_64-debugsymbols-1.8.1.tgz > out.tgz
$ tar -xzf out.tgz
$ cd mongodb-linux-x86_64-debugsymbols-1.8.1/
$ cd bin
$ addr2line --help
$ addr2line -i -e mongod 0x6d6a74
/mnt/home/buildbot/slave/Linux_64bit_V1.8/mongo/db/repl/health.cpp:394
$ addr2line -i -e mongod 0x6d0694
/mnt/home/buildbot/slave/Linux_64bit_V1.8/mongo/db/repl/rs.h:385
/mnt/home/buildbot/slave/Linux_64bit_V1.8/mongo/db/repl/replset_commands.cpp:111
```

Error Codes

If you have and error event and it isn't obvious what the error is, query for that error code on Jira. If still nothing please post to support forums.

This list is HIGHLY incomplete. This page is a stub.

Error Code	Description	Comments
10003	objects in a capped ns cannot grow	
11000	duplicate key error	_id values must be unique in a collection
11001	duplicate key on update	
12000	idxNo fails	an internal error
12001	can't sort with $snapshot	the $snapshot feature does not support sorting yet
12010, 12011, 12012	can't $inc/$set an indexed field	
13312	replSet error : logOp() but not primary?	Fixed in v2.0. Report if seen v2.0+
13440	bad offset accessing a datafile	Run a database --repair. If journaling is on this shouldn't happen.

Replication Internals

On the *master* mongod instance, the `local` database will contain a collection, `oplog.$main`, which stores a high-level transaction log. The transaction log essentially describes all actions performed by the user, such as "insert this object into this collection." Note that the oplog is not a low-level redo log, so it does not record operations on the byte/disk level.

The *slave* mongod instance polls the `oplog.$main` collection from *master*. The actual query looks like this:

```
local.oplog.$main.find({ ts: { $gte: ''last_op_processed_time'' } }).sort({$natural:1});
```

where 'local' is the master instance's `local` database. `oplog.$main` collection is a capped collection, allowing the oldest data to be aged out automatically.

See the Replication section of the Mongo Developers' Guide for more information.

`OpTime`

An `OpTime` is a 64-bit timestamp that we use to timestamp operations. These are stored as Javascript `Date` datatypes but are *not* JavaScript Date objects. Implementation details can be found in the `OpTime` class in *repl.h*.

Applying OpTime Operations

Operations from the oplog are applied on the slave by reexecuting the operation. Naturally, the log includes write operations only.

Note that inserts are transformed into upserts to ensure consistency on repeated operations. For example, if the slave crashes, we won't know

exactly which operations have been applied. So if we're left with operations 1, 2, 3, 4, and 5, and if we then apply 1, 2, 3, 2, 3, 4, 5, we should achieve the same results. This repeatability property is also used for the initial cloning of the replica.

Tailing

After applying operations, we want to wait a moment and then poll again for new data with our `$gte`operation. We want this operation to be fast, quickly skipping past old data we have already processed. However, we do not want to build an index on `ts`, as indexing can be somewhat expensive, and the oplog is write-heavy. Instead, we use a table scan in natural order, but use a tailable cursor to "remember" our position. Thus, we only scan once, and then when we poll again, we know where to begin.

Initiation

To create a new replica, we do the following:

```
t = now();
cloneDatabase();
end = now();
applyOperations(t..end);
```

`cloneDatabase`effectively exports/imports all the data in the database. Note the actual "image" we will get may or may not include data modifications in the time range (`t..end`). Thus, we apply all logged operations from that range when the cloning is complete. Because of our repeatability property, this is safe.

See class `Cloner` for more information.

Smoke Tests

- Test Organization
- Running all the tests
- smoke.py
- Running a jstest manually
- Running the C++ unit tests
- See Also

Test Organization

1. `dbtests/.*cpp` has C++ unit tests
2. `jstests/*.js` has core tests
3. `jstests/repl*/*.js` has replication tests
4. `jstests/sharding/*.js` has sharding tests
5. `slowNightly/*js` has tests that take longer and run only at night
6. `slowWeekly/*.js` has tests that take even longer and run only once a week

Running all the tests

```
scons smoke smokeCppUnittests smokeDisk smokeTool smokeAuth startMongod smokeClient smokeJs
scons startMongodSmallOplog smokeJs
scons startMongod smokeJsSlowNightly
scons smokeTool
scons smokeReplSets
scons smokeDur
scons mongosTest smokeSharding
scons smokeRepl smokeClone
scons startMongod smokeParallel
```

smoke.py

smoke.py lets you run a subsets of the tests in *jstests/*. When it is running tests, it starts up an instance of mongod, runs the tests, and then shuts it down again. You can run it while running other instances of MongoDB on the same machine: it uses ports in the 30000 range and its own data directories.

For the moment, *smoke.py* must be run from the top-level directory of a MongoDB source repository. This directory must contain at least the `mongo` and `mongod` binaries. To run certain tests, you'll also need to build the tools and `mongos`. It's a good idea to run `scons .` before running the tests.

To run *smoke.py* you'll need a recent version of PyMongo.

To see the possible options, run:

```
$ python buildscripts/smoke.py --help
Usage: smoke.py [OPTIONS] ARGS*

Options:
  -h, --help            show this help message and exit
  --mode=MODE           If "files", ARGS are filenames; if "suite", ARGS are
                        sets of tests (suite)
  --test-path=TEST_PATH
                        Path to the test executables to run, currently only
                        used for 'client' (none)
  --mongod=MONGOD_EXECUTABLE
                        Path to mongod to run (/Users/mike/10gen/mongo/mongod)
  --port=MONGOD_PORT    Port the mongod will bind to (32000)
  --mongo=SHELL_EXECUTABLE
                        Path to mongo, for .js test files
                        (/Users/mike/10gen/mongo/mongo)
  --continue-on-failure
                        If supplied, continue testing even after a test fails
  --from-file=FILE      Run tests/suites named in FILE, one test per line, '-'
                        means stdin
  --smoke-db-prefix=SMOKE_DB_PREFIX
                        Prefix to use for the mongods' dbpaths ('')
  --small-oplog         Run tests with master/slave replication & use a small
                        oplog
```

By default, `smoke.py` will run tests that create data in `/data/db`, which may interfere with other MongoDB instances you are running. To change the directory in which the smoke tests create databases, use `--smoke-db-prefix=/some/other/path`

To run specific tests, use the *--mode=files* option:

```
python buildscripts/smoke.py --mode=files jstests/find1.js
```

You can specify as many files as you want.

You can also run a suite of tests. Suites are predefined and include:

- *test*
- *all*
- *perf*
- *js*
- *quota*
- *jsPerf*
- *disk*
- *jsSlowNightly*
- *jsSlowWeekly*
- *parallel*
- *clone*
- *repl* (master/slave replication tests)
- *replSets* (replica set tests)
- *auth*
- *sharding*
- *tool*
- *client*
- *mongosTest*

To run a suite, specify the suite's name:

```
python buildscripts/smoke.py js
```

Running a jstest manually

You can run a jstest directly from the shell, for example:

```
mongo --nodb jstests/replsets/replsetarb3.js
```

Running the C++ unit tests

The tests under jstests/ folder are written in mongo shell javascript. However there are a set of C++ unit tests also. To run them:

```
scons test
./test
```

Build the unit tests (in src/mongo/unittest/ by running:

```
$ scons build/unittests.txt
/* build output */
Generating build/unittests.txt
        build/linux2/dd/durableDefaultOff/mongo/platform/atomic_word_test
        ...
        build/linux2/dd/durableDefaultOff/mongo/bson_template_evaluator_test
        build/linux2/dd/durableDefaultOff/mongo/balancer_policy_test
scons: done building targets.
```

Then use the line above to run the binary generated:

```
$ ./build/linux2/dd/durableDefaultOff/mongo/platform/atomic_word_test
```

See Also

- scons

Pairing Internals

Policy for reconciling divergent oplogs

pairing is deprecated

In a paired environment, a situation may arise in which each member of a pair has logged operations as master that have not been applied to the other server. In such a situation, the following procedure will be used to ensure consistency between the two servers:

1. The new master will scan through its own oplog from the point at which it last applied an operation from it's peer's oplog to the end. It will create a set C of object ids for which changes were made. It will create a set M of object ids for which only modifier changes were made. The values of C and M will be updated as client operations are applied to the new master.
2. The new master will iterate through its peer's oplog, applying only operations that will not affect an object having an id in C.
3. For any operation in the peer's oplog that may not be applied due to the constraint in the previous step, if the id of the of the object in question is in M, the value of the whole object on the new master is logged to the new master's oplog.
4. The new slave applies all operations from the new master's oplog.

Emacs tips for MongoDB work

You can edit confluence directly from emacs:

First, follow the basic instructions on http://code.google.com/p/confluence-el/

Change the confluence-url in their sample setup to http://mongodb.onconfluence.com/rpc/xmlrpc

Might also want to change the default space to DOCS or DOCS-ES or whatever space you edit the most.

etags setup (suggested by mstearn)

First, install "exuberant ctags", which has nicer features than GNU etags.

http://ctags.sourceforge.net/

Then, run something like this in the top-level mongo directory to make an emacs-style TAGS file:

```
ctags -e --extra=+qf --fields=+iasnfSKtm --c++-kinds=+p  --recurse .
```

Then you can use M-x visit-tags-table, M-., M-* as normal.

Community

- Technical Support
- Bug/Feature Tracker (Jira)
- Blog
- Mailing List
- Events
- Job Board
- Twitter etc.
- Store
- Resources for Driver and Database Developers
    - Source
    - Developer List
    - Project Ideas
- Contribute!
    - Write a book
    - Write a driver, framework, and other tools
    - Help with Free Support
    - Work on the DB

Technical Support

See the Support page, free support is available and its usage encouraged!

Bug/Feature Tracker (Jira)

File, track, and vote on bugs and feature requests. There is issue tracking for MongoDB and all supported drivers.

Blog

http://blog.mongodb.org/

Mailing List

http://groups.google.com/group/mongodb-announce - for release announcement and important bug fixes.

Events

The events page includes information about MongoDB conferences, webcasts, users groups, local meetups, and open office hours.

Job Board

- Click Here to access the Job Board. The Board is a community resource for all employers to post MongoDB-related jobs. Please feel free to post/investigate positions!
- See also the Indeed MongoDB jobs list

Twitter etc.

- @mongodb
- facebook
- linkedin

Store

- Visit the MongoDB store on Cafepress.

Resources for Driver and Database Developers

Source

The source code for the database and drivers is available at the http://github.com/mongodb.

Developer List

This mongodb-dev mailing list is for people developing drivers and tools, or who are contributing to the MongoDB codebase itself.

Project Ideas

Start or contribute to a MongoDB-related project.

Contribute!

Write a book

If interested contact info@10gen.com we'll try to get you in touch with publishers.

Write a driver, framework, and other tools

Writing Drivers and Tools

Help with Free Support

Jump in with answers on http://groups.google.com/group/mongodb-user and IRC (freenode.net#mongodb) .

Work on the DB

http://groups.google.com/group/mongodb-dev

MongoDB Masters

The MongoDB Masters are a group of MongoDB core contributors and community evangelists that are dedicated to sharing their passion and technical expertise with the MongoDB community around the world. The MongoDB Masters play a vital role in the adoption, education and advancement of MongoDB.

- The MongoDB Masters
  - **Roman Alexis Anastasini**
  - **Katia Aresti**
  - **Peter Bell**
  - **Rick Copeland**
  - **Justin Dearing**
  - **Mike Dirolf**
  - **Michael Fiedler**
  - **Kenny Gorman**
  - **Jonas Haag**
  - **Nathen Harvey**
  - **Justin Hileman**
  - **Jon Hoffman**
  - **Takahiro Inoue**
  - **Jongmyun Joo**
  - **Lennart Koopmann**
  - **Lynn Langit**
  - **Nat Luengnaruemitchai**

- **David Makogon**
- **Harry Marr**
- **David Mytton**
- **Gustavo Niemeyer**
- **John Nunemaker**
- **Niall O'Higgins**
- **Flavio Percoco**
- **Mitch Pirtle**
- **Karl Seguin**
- **Mark Smalley**
- **Russell Smith**
- **Andy Stanberry**
- **James Summerfield**
- **Tony Tam**
- **Rose Toomey**
- **Jonathan Wage**
- **Ian White**
- **Aristarkh Zagorodnikov**

- MongoDB Master Summit
    - Notes from 2012 MongoDB Masters Summit

The MongoDB Masters

(sorted alphabetically by last name)

Roman Alexis Anastasini

Roman is a Scala and C# Developer based in Beijing. Originally from Germany, he was formerly a Technical Manager at Gameforge in Kahlsrue Germany. He's now excited to encounter new adventures in Beijing.

- @foliba on Twitter

Katia Aresti

- @KAresti on Twitter
- GitHub

Katia has worked in IT since 2005, first as a consultant at Xebia and Sopra, and as a freelancer since September 2012.
She's mostly Java developer (while also dabbling in PHP). She has worked with MongoDB since 2010, becoming one of the first java teams getting MongoDB in production in France, urban dive .
She's spoken about Java and MongoDB at Java User Groups, MongoDays and MongoDB User groups about her expertise via hands-on workshops, presentations and open space Q-A's. One of her ventures is Duchess France , which focuses on educating women developers on emerging technologies. She is passionate about Open Source and enjoys getting involved on community contributions.

MongoDB Contributions

- JDuchess France
- Paris MongoDB User Group

Peter Bell

Peter is Senior VP Engineering and Senior Fellow at General Assembly, a campus for technology, design, and entrepreneurship. He's a regular presenter at national and international conferences on ruby, nodejs, NoSQL (especially MongoDB and neo4j), cloud computing, software craftsmanship, java, groovy, javascript, and requirements and estimating. He is on the program committee for Code Generation in Cambridge, England and the Domain Specific Modeling workshop at SPLASH (was ooPSLA) and reviews and shepherds proposals for the BCS SPA conference.

He has presented at a range of conferences including DLD conference, ooPSLA, QCon, RubyNation, SpringOne2GX, Code Generation, Practical Product Lines, the British Computer Society Software Practices Advancement conference, DevNexus, cf.Objective(), CF United, Scotch on the Rocks, WebDU, WebManiacs, UberConf, the Rich Web Experience and the No Fluff Just Stuff Enterprise Java tour. He has been published in IEEE Software, Dr. Dobbs, IBM developerWorks, Information Week, Methods & Tools, Mashed Code, NFJS the Magazine and GroovyMag. He's currently writing a book on managing software development for Pearson.

Github

Rick Copeland

Rick Copeland is a Lead Software Engineer at SourceForge where he joined the team that introduced MongoDB to the SourceForge technology stack with the migration of the consumer-facing pages of SourceForge from a PHP/relational database platform to Python/MongoDB. Out of that experience came Ming, an MongoDB object/document mapper for Python that he maintains and continues to develop. Rick also helped lead the effort to rewrite the developer tools (wiki, tickets, forums, repos, etc.) portion of the SourceForge site on the Python/MongoDB platform and

released that platform as Allura under the Apache License. He also created the Zarkov realtime analytics framework (also released under the Apache license) used at SourceForge to calculate project statistics. He is a frequent speaker at MongoDB events and an avid MongoDB enthusiast.

GitHub
@rick446 on Twitter
Blog
Presentations

MongoDB contributions

Ming, an object/document mapper for MongoDB in Python
Zarkov, a realtime analytics framework using MongoDB
Allura, the new MongoDB-powered platform for SourceForge

Justin Dearing

Justin Dearing has been working in IT in 2002. He started his career as a night shift AS/400 operator and rose through the ranks at a series of companies.

Justin has worked in both the development and production side of the house on Windows, Unix and Midrange Platforms. Besides MongoDB his database experience includes MySQL, Postgres and Microsoft SQL server. These days he mainly programs in C#, Powershell and PHP.

Justin's life was forever changed on 2009-10-27 when Kristinia Chodorow presented a talk on mongodb at NYPHP and destroyed everything he knew to be right, holy and true about databases. A few months later he push a small app using MongoDB to production. In addition to afflicting the world with apps that use MongoDB, he has contributed to the core server and the official .NET driver.

Justin lives in Jersey City with his wife and 3 laptops.

@Zippy1981 on Twitter
GitHub
Blog

Mike Dirolf

Mike was the original author of the PyMongo project and a maintainer of the mongo-ruby-driver. He co-authored O'Reilly's MongoDB: The Definitive Guide. He maintains several MongoDB-related open source projects, and is the founder of a web service, Fiesta (https://fiesta.cc), that uses MongoDB as its primary data store.

@mdirolf on Twitter
GitHub
Blog

MongoDB Contributions

MongoDB: The Definitive Guide
Wrote PyMongo
Mongo-ruby-driver
nginx-gridfs

Michael Fiedler

Mike is a long-time systems engineer, building a variety of platforms
with all technologies. He has spoken at MongoDB conferences, worked
with 10gen on systems-oriented features, and manages sharded clusters.
He contributes to many open source projects and writes a few of his own. On the side he's a roller derby referee and licensed skydiver.

MongoDB Contributions

- MongoDB-Chef Cookbook

- @mikefiedler on Twitter
- Github
- Blog

Kenny Gorman

Kenny Gorman has over a decade of experience with various database platforms behind some of the busiest websites in the world. He has had roles as Developer, DBA, Architect, Manager and Director. He was an early adopter of MongoDB in 2009 using it for various projects at Shutterfly. He wrote an early python version of the Mongostat tool that is distributed with MongoDB today. He enjoys performance tuning, large scale systems development, and tricky database problems.

Github

@kennygorman
Blog

Contributions
Wrote the original mongostat in python, since then it's moved to core distribution.

Jonas Haag

Jonas Haag is a passionate Python programmer and free software enthusiast from Stuttgart, Germany. He maintains Django MongoDB Engine, a MongoDB backend for the Python Web framework Django.

GitHub
Blog

MongoDB Contributions

Django MongoDB Engine
PyMongo

Nathen Harvey

Nathen Harvey is a Technical Community Manager at Opscode, the company behind Chef. Nathen is the co-organizer of the Washington DC MongoDB Users' Group and DevOpsDC and co-host of the Food Fight Show, a podcast about Chef and DevOps. Like many others who blog, Nathen updates his blog on a very irregular basis. When not working or hosting meetups, Nathen enjoys going to concerts, drinking craft beer, and over sharing on sites like twitter,untappd, and foursquare.

- **MongoDB Contributions**

- Github
- @nathenharvey on twitter
- Blog
- Untappd
- MongoDC User Group
- Presentations about MongoDB
- MongoDC User Group

Justin Hileman

- GitHub
- Blog
- @bobthecow on Twitter

Justin is a maker of the internet. He is the author of Genghis, the single-file MongoDB admin app, available in two delicious Ruby and PHP flavors. He has also contributed to the Doctrine2 MongoDB ODM, along with various and sundry other Open Source projects.

Jon Hoffman

Jon is a software engineer and currently runs the infrastructure engineering team at foursquare. His team is tasked with designing the backends of new features, building robust systems to keep the platform running smoothly and helping other engineers get things done. In the course of his tenure at foursquare Jon has made contributions to the mongoDB's core server, mongo-java-driver, Lift Framework, and scalaj-http projects. Jon recently presented work the foursquare infrastructure team built to add robustness to mongo on top of unreliable disk. Before foursquare Jon spent a brief time working at a three person startup, built distributed systems at Goldman Sachs, worked on VoIP apps for the telephone company, created a medical record app for palm pilot, and graduated from Carnegie Mellon.

Github
@hoffrocket on Twitter

MongoDB Contributions

Core Server
MongoDB Java Driver
MongoDB in Production: Data and Availability, March NYC MongoDB User Group

Takahiro Inoue

Takahiro is a Chief Data Scientist at Treasure-Data Inc where he uses MongoDB for log data analysis. He is a frequent speaker on MongoDB and Data and the organizer of the Tokyo MongoDB User Group

GitHub
@doryokujin on Twitter

Slideshare
Blog

MongoDB Contributions

Organizer of the Tokyo MongoDB User Group

Jongmyun Joo

- [South Korea MongoDB User Group]
- [Plan Information Technology]

Jongmyun is and active community evangelist and the leader of the South Korea MongoDB User Group. He is a certified Oracle ACE and Member of the expert committee on Korea Database Promotion Agency and Full-time lecturer on Database technology at the The Korea SW Technology Association.

Lennart Koopmann

Lennart Koopmann is a developer from Hamburg, Germany and author of Graylog2 - A free and open source log mangement system that uses MongoDB as database. He also wrote mongo_analyzer, a little web frontend on top of the MongoDB profiler that helps you optimizing your queries.

@_Lennart on Twitter
GitHub
Blog

MongoDB Contributions

Graylog
Mongo Analyzer

Lynn Langit

Lynn Langit was a developer evangelist for the Microsoft MSDN team for the previous 4 years.. Prior to working at Microsoft, she founded and served as lead architect of a development firm that created BI solutions. She holds a number of Microsoft certifications, including MCITP, MCSD, MCDBA, and MCT. Lynn left Microsoft to do consulting and training in October 2011. Lately she's authored and taught for DevelopMentor (SQL Server 2012 and Google App Engine) and she's created a series of talks around 'NoSQL for the RDBMS professional' and 'MongoDB for the .NET Developer'.

- @lynnlangit On Twitter
- blog
- MongoDB YouTube playlist |http://www.youtube.com/playlist?list=PLCE5902FB1D284880&feature=view_all]

Nat Luengnaruemitchai

Bio: working in financial industry. Help out on a couple projects such as ikvm, dnanalytics, mongodb
Github

MongoDB Contributions
Bug fixes/small enhancement in mongodb core, c# driver, java driver
Over 2,700 posts on the mongodb-user group (free support forum)

David Makogon

David Makogon has been a software creationist and architect for over 25 years. He's currently a Senior Cloud Architect at Microsoft specializing in Windows Azure.

Since 2010, David has been working with MongoDB, specifically in Windows Azure. He built both standalone and replica set samples, presenting these at MongoDC and MongoSV in 2010. He's also provided architectural guidance to several ISV's as they build Windows Azure solutions coupled with MongoDB.

Outside of computing, David is an avid photographer and family man, with a penchant for puns and an uncanny ability to read backwards.

Twitter
Blog
Presentations

Harry Marr

Harry Marr (@harrymarr) is the author of MongoEngine, a popular Python ODM (Object-Document Mapper). He hails from London, where he spends most of his time working in Python and Ruby. He was previously employed at Conversocial, where he drove a migration from MySQL to MongoDB using MongoEngine. He currently works at GoCardless, an early-stage startup with some exciting ambitions in the payments space.

When he's not working on disrupting the payments industry, he can be found hacking on various open source projects (https://github.com/hmarr).

Github
@harrymarr
Blog

MongoDB Contributions

MongoEngine
MongoEngine source

David Mytton

David has been a PHP/Python programmer for 11 years. He is the founder of Server Density a hosted server monitoring service where he built the original code and server infrastructure behind the application which is now processing over 1bn documents (12TB data) each month. Server Density uses MongoDB extensively as our primary data store since 2009, and it now deployed across 50 servers on Softlayer Cloud. He is a regular speaker on MongoDB and runs the London MongoDB User Group.

@Davidmytton on Twitter
GitHub
Blog

MongoDB Contributions

Server Density
MongoDB Monitoring Tool
London MongoDB User Group

Gustavo Niemeyer

Gustavo acts as the technical lead behind projects from Canonical such as the Landscape systems management platform, the juju orchestration framework, and the Storm object-relational mapper for Python. In his free time, among other things Gustavo is a contributor to Google's Go language, is the author of the mgo (mango) MongoDB driver for Go, and also designed the Geohash concept that is used internally by MongoDB.

@Gniemeyer on Twitter
Code Repository
Blog

MongoDB Contributions

mgo
Geohash

John Nunemaker

John Nunemaker develops simple and beautiful software at Github. His startup, Ordered List, recently acquired by Github, has several MongoDB backed applications in production – Gauges, Harmony and Speaker Deck. He is also the creator of MongoMapper, a popular Ruby object mapping library for MongoDB.

@Jnunemaker on Twitter
GitHub
Blog

MongoDB Contributions
MongoMapper

Niall O'Higgins

Niall O'Higgins is the co-founder of a software product & services company specializing in NoSQL, mobile and cloud computing. He is the author of the book "MongoDB and Python" published by O'Reilly. He is the founder and organizer of both the San Francisco Python Web Technology Meet-up, PyWebSF and the Bay Area Tablet Computing Group, We Have Tablets. He has published quite a bit of Open Source software - contributing to OpenBSD and Pyramid among others - and frequently speaks at conferences and events.

@niallohiggins on Twitter
GitHub
Blog

MongoDB Contributions
MongoDB and Python

Flavio Percoco

Flavio works in the Research and Development department at The Net Planet Europe and is an avid MongoDB community contributor. His host of contributions include Pymongo, the Django Database Engine (co-author and maintainer), the MongoDB plugin for eclipse and the python virtual machine for MongoDB. He lives in Milan, Italy and is a frequent speaker at MongoDB and Europe technology conferences.

@flaper87 on Twitter
GitHub
BitBucket
Blog

MongoDB Contributions

Django Database Engine for MongoDB
Python Virtual Machine inside MongoDB
MongoDB Plugin for Eclipse
MongoDB CDR Backend for Asterisk
MongoDB Transport for Kombu

Mitch Pirtle

Mitch is currently CTO at Sounday Music, a social and services platform catering to the music industry. There he maintains the core platform comprised of MongoDB and the Lithium framework for PHP.

He was first corrupted by Dwight Merriman while launching Jetsetter for Gilt Groupe, which went on to be the first e-commerce site powered by MongoDB. He then followed that up by launching Totsy, the first e-commerce site to rely solely on MongoDB for all data storage. He is also an original core member for the Mambo content management system, where he went on to found Joomla! and Open Source Matters. Before that he was contributing to many open source projects, and was an outspoken advocate for PostgreSQL, which still remains his favorite relational database.

He is based in Turin Italy with his wife, kids, and rapidly proliferating devices.

GitHub: http://github.com/spacemonkey
@mitchitized on Twitter

Karl Seguin

Karl Seguin is a developer with experience across various fields and technologies. He's an active contributor to OSS projects, a technical writer and an occasional speaker. With respect to MongoDB, he was a core contributor to the C# MongoDB library NoRM, wrote the interactive tutorial mongly, the Mongo Web Admin and the free Little MongoDB Book. His service for casual game developers, mogade.com, is powered by MongoDB.

@KarlSeguin on Twitter
GitHub
Blog

MongoDB Contributions

Mongoly.com
The Little MongoDB Book

Mark Smalley

Mark Smalley is a Brit on a mission. Currently based out of Kuala Lumpur, Malaysia, he roams around Asia making every effort he can to convert anyone and everyone into avid MongoDB enthusiasts. He is also one of the lead organizers for the monthly Kuala-Lumpur MongoDB User-Group and lead-developer on several MongoDB powered OpenSource initiatives.

Twitter
GitHub
Blog

MongoDB Contributions

MongoBase
Geoply
MongoPress

Russell Smith

Russell is a passionate developer and ops lover. He consulted for companies like Experian and 10gen on development, ops, architecture design and capacity planning. He was instrumental in scaling the Guardian's MongoDB powered social reader. He has spoken about MongoDB at many London conferences and meetups, including Seedcamp's Seedhack.

Russell is currently CTO of RainforestQA.com

- Rainforest QA
- Presentations
- MongoDB Backups Script Helper
- London MongoDB User Group

Andy Stanberry

Github
[Blog]
@cranberyxl on Twitter

Andy is the Director of Development at Sailthru, Sailthru, a behavioral communications platform. Andy has been coding since the late 90's and using MongoDB since early 2010. Sailthru is his 5th startup and second time leading a technology team. In his spare time he enjoys swing dancing, traditional jazz, and hacking.

James Summerfield

@jsummerfield on twitter
Github

James is a development manager at Rangespan where he divides his time between architecting, building and supporting large scale distributed systems using MongoDB and Hadoop. He founded Interview Zen to help find and hire great developers. He previously led software development projects at HM Treasury, where he designed systems to analyse economic recovery, and at UBS Investment Bank, where he developed real-time trade analytics. He has an MA in Computer Science from Oxford University. James enjoys presenting his work with MongoDB, most recently at the MongoDB UK conference and a number of London MongoDB User Groups.

Tony Tam

@fehguy on Twitter
GitHub
Presentations
Swagger

MongoDB Contributions

mongodb oss admin tools

Tony is a San Francisco Bay Area native. He received his undergraduate degree in Mechanical Engineering from UC Santa Barbara and his MBA from Santa Clara University. He is currently the CEO of Wordnik. He was the founding engineer and SVP of Engineering at Think Passenger, the leading provider of customer collaboration software. Prior to joining Passenger, he was lead engineer at Composite Software of San Mateo, California. At Composite Software he helped developed the company's first- and second-generation query processing engines and led the research and implementation of their patented cost-based federated query optimizer. Prior to that he led software development in the bioinformatics group at Galileo Labs, a drug-discovery company based in the Silicon Valley.

Rose Toomey

Rose Toomey is the creator of Salat, a simple serialization for Scala and MongoDBSalat was developed to make using Scala with Casbah and MongoDB as simple as possible. While Casbah increased the usability of the mongo-java-driver in Scala, there was no correspondingly elegant solution for serializing and deserializing objects. The new horizons opened up by using MongoDB as a document store demanded something better than the complexity and ceremony of the ORMs I'd worked with in the past. I also faced the challenge that my company, Novus Partners, is a financial startup that needs to process massive amounts of data very quickly. What to do? Enter Salat: it not only serializes to and from Mongo documents quickly, but uses hi-fi type information provided by the Scala compiler instead of explicit mappings. No fuss, no muss: my goal is that someone who wants to use Scala and MongoDB can be up and running with Salat in fifteen minutes.

GitHub
@Prasinous on Twitter

MongoDB Contributions

Salat

Jonathan Wage

Software engineer from Nashville, TN currently working for OpenSky.com
@Jwage on Twitter
GitHub
Blog

MongoDB Contributions

Doctrine MongoDB Object Document Mapper for PHP open source project

Ian White

Ian is the co-founder and CTO of Sailthru, a company that automatically tailors email, web and advertising content down to the unique user. He was the first non-10gen employee to use MongoDB in production, and built both Business Insider and Sailthru using MongoDB as the datastore.

@EonWhite on Twitter
GitHub

MongoDB Contributions

SimpleMongoPHP

Aristarkh Zagorodnikov

Started using MongoDB about half a year ago, made it the default database (we still have most of our stuff using PostgreSQL, but all new development except billing services is done with MongoDB) for our company Bolotov .

GitHub
BitBucket
Blog

MongoDB Contributions

MongoDB C# Driver

MongoDB Master Summit

Since 2011, 10gen has hosted an annual summit for the MongoDB Masters the day before MongoSV.

Notes from 2012 MongoDB Masters Summit

- A MongoDB Masters Book
- ODM Breakout Session
- Schema Design Session
- Deployment Strategies
- Integrating with Hadoop, Elastic Search, SOLR
- Ops War Stories
- Custom Tools and Monitoring
- [Capacity Planning]
- Evangelizing to DBAs

A MongoDB Masters Book

The MongoDB Masters are working together to write a book!

Focus and Audience

- Book should focus on MongoDB Pitfalls
- Make a "Bitter Mongo" like the "Bitter Java" book
    - Couching it right to ensure it doesn't prematurely prepare for problems is difficult
        - i.e. "This will bite you"
- Audience: MongoDB for the Accidental DBA
- 50 Tips and Tricks for MongoDB Developers
    - Kristina's book is really good
    - Good model for the MongoDB Masters Book

Content

- Support Tree for MongoDB Schemas
- Patterns that would be valuable
- Goldilocks versions i.e. Large, happy medium, small. What is just right.
- Schema Design Cookbook
    - (Blogging, logging and Cataloging)

Structure

- Let's get started
- Let's Avoid
- Let's Dig Ourselves Out
    - Blog + Comment
    - Simple models that people understand
    - Logging
    - Product Catalogues
    - Geospatial (geolocation)
    - Things you wouldn't think of
    - Quirky Use Cases
    - Choosing a Shard Key
    - Data Migration, Schema Migration
    - Managing Object Constellations for keeping quick queries

- Do the same piece in all different languages or frameworks?
  Node+express
  Python+Flask
  Skeletons to get started

Type and contributing

- AskiDoc as the type
    - PanDoc to convert formats.
    - Emacs
    - MOU for Mac.
    - Github pages to gather content

Quirky Ideas:

- Choose Your Own Adventure comic book?
    - Randall Schwartz

Custom Tools and Monitoring

Custom Tools & Monitoring
==========

Scope of discussion: Monitoring & Custom tools

Items most often monitored
~~~~~~~~~~~~~~~~~~~~~~~~~~
These items are the best indicators of a problem with MongoDB

- Replication lag
- page faults
- lock percentage
- IO usage

Additional Helpful Monitoring
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

- Watch for long running queries
- Network Flakyness / DNS monitoring
- Chunk Imbalances
- Checking that queries are distributed evenly across shards

log scraping
~~~~~~~~~~~~

- looking or indications of fragmentation, (empty extents)
- Bursts of NotMaster

Long Term Strategies

ODM Level Monitoring
~~~~~~~~~~~~~~~~~~~~

- looking for unsafe operations. Queries that do use an index
- Making ODM instrumented
- Query execution history

Profile queries as part Integration Testing that profiles queries
~~~~~~

- Write queries predefined as application objects

Possible mailing list of hard to diagnose problems
~~~~~~

- The type of problems which don't necessarily manifest themselves in the traditional indicators.

Deployment Strategies

Deployment

- Hardware
  - 12 x boxes, each with 512G RAM, SSD's, 4 x mongod each
  - Moving to 128G, 1 x mongod to avoid failures
  - Amazon 68G machines
- OS
  - Linux
    - 3 x people
  - Solaris
    - 0 x people
- SSL
  - commercial only
  - people using hacks; openvpn
- repos
  - ubuntu packages massively out of date
  - official packages always point to the latest version; how to pin to 2.x
  - missing init scripts / upstart scripts
- Mongos
  - most people 1 per application server
  - no init / upstart for mongos!
- CFG
  - on tiny boxes
  - spinning disks
- Fabric
  - niche
- Puppet x 2
  - not good for chaining stuff
  - configuration for different node types
-
  - useful to enforce defaults + install
- DNS configuration scripts
  - Replacing nodes
  - if no-DNS, add instance, sync and remove old node
- rs.addNode
  - copy config from other node if first startup?

Backups

- EBS snapshots
- LVM snapshots
- Mongodump
- SAN attached
- Testing of backups
  - Not common
  - Not understood

Evangelizing to DBAs

Pushback from DBAs

Fanboi's and the haters:

- MongoDB is a hipster OS
- feel their pain
- Not wanting to give up control of schema design
- The converts are the most powerful evangelists
- Proactive monitoring

Marketing and messaging:

- call the talk: why not to use MongoDB (trick people into coming)
- safe zone for DBAs
- Saturday events for the Operations guy
- Chained to your desk online conferences for the overworked operations guys.

Culture issue:

- Fear of new technologies
- median age difference between NoSQL and RDBMS people
- core values: we will do it in code versus they do it in the DB

Ways to fixing it:

- explain the tradeoffs
- explain that there are distinct versions and formal releases

Good things:

- commercial support
- free community support
- knowledgebase
- its transparent open source
- Accept that training becomes consultation

Concrete action items:

- A #mongodbhelp blog post – DONE!
- More operations talks and docs
- SAS70 compliance
- Identifying a roadmap for tickets that would make DBAs happier
- Improving Instrumentation (e.g. performance counters)

Integrating with Hadoop, Elastic Search, SOLR

- 10gen has unofficially provided some connectors: Hadoop, Storm, Spark, Elastic Search, SOLR, Sphinx
- For search integration, seems like the common approach is to tail the oplog and send that to the service.

From Flavio:

- One approach was to start hacking on MongoDB's index API, which produced redundant indexing and inefficiency
- Two approach was to split the data at the application level
- Three was Elastic Search River
- One issue is based on timing - maybe tailing the oplog is not the right place to be triggering your index updates to the external application. If you do that work in your application, you can control the timing.
- An interesting use is Hive - non-developers will run hive queries as a sort of BI system. They also have daily mapreduce jobs that power the recommendation engine.
- There was a little integration with Pentaho, using the aggregation framework. Complaints with the moped driver and aggregation.
- Discussing 10gen-labs / mongo-connector as a framework for integration with external systems via tailing oplogs. Has examples for SOLR.
- We need an online community platform, as 10gen is doing great at in-person networking but there's little online.

Notes from 2012 MongoDB Masters Summit

Notes from 2012 MongoDB Masters Summit

- A MongoDB Masters Book
- ODM Breakout Session
- Schema Design Session

- Deployment Strategies
- Integrating with Hadoop, Elastic Search, SOLR
- Ops War Stories
- Custom Tools and Monitoring
- [Capacity Planning]
- Evangelizing to DBAs

ODM Breakout Session

- Lots of ODMs in the ecosystem
- General consensus: Mongoose is leading the way.
- Balance: the right balance of features and what should be handled by the drivers.
- Mapping and validation will never be handled by the drivers, should be the job of the ODMs.
- It became clear that future ODMs should be split into as many components / modules as possible to allow people to choose how much they want from their ODM.

Ops War Stories

- sync from master is bad, lots of folks playing with iptables to block access to primaries for sync()
- mongodb crashes fast when there are problems goes into the ground quickly. Query killer is key.
- be sure not to wait to long to shard, causes massive headaches. Can't get balancer to run when already out of capacity. Both space and OPS causes this problem.
- pre splitting is key
- better mms docs, where is the spot where space utilization is shown over time?
- java thread growth, general sprawl over time. no good way to identify and kill session with open socket but dead in mongodb.
- People haven't really used client side timeouts, see previous bullet.
- array ops trigger crazy bugs in replication
- lots of frag problems, hopefully power of 2 allocation fixes
- need good way to measure fragmentation
- tailable cursor on oplog works ok on low latency connections, but not on wan
- load induced elections, load can cause primary to stop saying hello, and on top of high load an election happens.
- can't realistically perform bulk updates, no throttling mechanism built in folks build it into the app or don't do them. Floods oplog/replication.
- cluster wide backups/consistency is non-existent.
- if index key size is exceeded it fails quietly. really bad.
- if you kill any replication process it can leave holes in replication, really bad.
- no simple recovery tool for roll forward. Folks mentioned new oplog tool, but not really geared towards recovery.

Schema Design Session

- The session started with group concerns regarding the locking of deeply embedded documents and premature optimisation. Most people started out by embedding everything, but were quickly to realise that data without bounds (such as blog comments) should probably be broken out into separate collections (depending on use case).

- However, splitting things leads us a lack of atomicity.

- Lots of solutions and home-brewed 2 phase commits, logs and nonces were discussed - and doing things this way (rather than Hybrid applications, which are also 2 phase) makes more sense, as it can then be switched on and off depending on the use case allowing us to get the best of both worlds.

- Cool tips were also brought up regarding the use of collection names as ways to better perform mass deletion.

Technical Support

- Free Support Forum - http://groups.google.com/group/mongodb-user

- IRC Chat and Support - irc://irc.freenode.net/#mongodb

- Commercial Support

MongoDB Commercial Services Providers

10gen

10gen is the initiator, contributor and continual sponsor of the MongoDB project. 10gen offers

subscriptions that include production support, commercial licensing, and MongoDB Monitoring Service . In addition, 10gen offers advisory consulting and training.

Ready to learn more about 10gen? Fill out our contact form, and a 10gen representative will get in touch to learn about how 10gen services can help your organization.	**Need support right away?** If you are having a production issue and need support, please contact us, call us at (866) 237-8815, or visit the community Technical Support page.	**Want to be listed here?** If you provide consultative or support services for MongoDB and wish to be listed here, just let us know.

Hosting and Cloud

See the MongoDB Hosting Center.

Official 10gen Partners

10gen provides training to its systems integrator partners. To learn more about the partner program and what 10gen provides, or for information about joining the program, visit 10gen.com.

Company	More Information
CIGNEX DATAMATICS Making Open Source Work™	CIGNEX Datamatics (a subsidiary of Datamatics Global Services Ltd.) is the global leader in Commercial Open Source enterprise solutions and a global partner of 10gen (MongoDB) offering advisory consulting, implementation, and other services around the MongoDB application.
comSysto sense and respond	comSysto is a Munich based software company specialized in lean business and technology development. While supporting all three steps of a well known Build-Measure-Learn lean feedback loop, comSysto focuses on open source frameworks and software as major enablers of short, agile Build-Measure-Learn iterations and fast gains in validated learning. Powerful MongoDB technology provides the needed flexibility and agility for turning ideas into products as well as performance for handling Big Data while turning data into knowledge. We also enjoy developing with Spring framework and its sub-projects, Apache Wicket, Gradle, Git, Oracle DB and Oracle BI. comSysto is dedicated to eliminating waste in both business and technology since 2005.
codecentric	codecentric AG specialises in developing customised IT-solutions. The company is one of the leading German providers in the areas of agility, architecture, Java performance, Java and Enterprise Content Management. codecentric offers development, IT-consulting and services throughout the complete life cycle of applications and infrastructures: from individual software solutions to performance optimisation of Java-applications, to support of organisational processes within the company. The more than 120 codecentric employees work at various locations in Germany and Europe.
<epam>	EPAM's core competencies include complex software product engineering for leading global software and technology vendors as well as development, testing, maintenance, and support of mission critical business applications and vertically oriented IT consulting services for global Fortune 2000 corporations.
EQUAL EXPERTS	Since its inception in 2005, Equal Experts has been providing bespoke application development services to blue chip clients, including major investment banks and leading Telecom companies. Equal Experts has defined a unique approach to Agile software development that is both pragmatic and Enterprise-friendly. By harnessing the potential of modern application development techniques, our goal is to radically improve our customer's efficiency.

	iTechArt Group Since 2003 iTechArt Group, a leading consulting firm, has been providing R&D Services for emerging technology companies with heavy focus on web and mobile development. iTechArt offers a traditional IT outsourcing model complete with project-based services giving our clients the ability to build and manage their own dedicated team in one of our development centers in Eastern Europe. iTechArt specializes in product research and conceptualization, helping our clients select the correct architecture design & platform, implementation & customization, testing & QA, product installation, migration & porting, as well as modernization & product extension.
	NewWave Telecom & Technologies Inc. is SEI CMMI® Maturity Level 2 appraised; a GSA Schedule 70 IT Service Provider and 8(a) STARS II holder, service Information Technology and Business Services firm who is a leading implementer of business solutions using service-oriented technologies. We specialize in providing end-to-end, mission critical information technology solutions for healthcare, finance, telecoms, government agencies and private sector clients.
	OpenCredo is a team of highly experienced consultants who are amongst the best in the business. Dedicated to maximizing the ability to deliver value from any enterprise software development and delivery capability. They help their clients use leading-edge technologies and practices to drive waste from their software delivery projects.
	PalominoDB is a boutique consultancy, known for being top MySQL experts, and specializing in open source DB technologies including MongoDB, MySQL, Postgres, Cassandra and HBase.
	Thermopylae Sciences + Technology (TST) is a Service-Disabled Veteran-Owned Small Business (SDVOSB) that focuses on excellence in all we do for our government and private sector customer base. We particularly excel in software development, cyber security, geospatial engineering, cloud/grid computing in cleared environments, and mobile application development.
	Thumbtack is a full service application development partner specializing in highly scalable and cloud-based solutions. We provide strategic advice, technical architecture, software development, as well as quality assurance and deployment support to companies in Media, Publishing, and Financial Services. With a core competency in NoSQL, Thumbtack has brought robust MongoDB solutions to production for a variety of clients and industries.
	Rothbury Software is a very experienced and successful boutique software consulting company. Our impressive customer list says more about us than we can state here.
	Xebia is an international IT consultancy and project organization focused on Enterprise Java technology, Agile development methods and outsourcing services. Xebia consists of over 250 professionals, all committed to be the best in their field of expertise. Passion for in depth technology, in combination with Lean, Agile and Scrum practices are Xebia's driving factors and competitive edge.

	FastConnect FastConnect is a french consulting and engineering company specialised in distributed architectures. Our expertise spans the domains of SOA and EDA, Process and Decision Management, Cloud, Big Data and Analytics for business-critical applications. We bring to our clients our significant and strategic investments in specific technologies and development methodologies. Our consultants are helping our clients in designing and delivering flexible and linearly performing applications at the best price/performance ratio using state of the art architecture and enterprise-ready tools. For more information visit http://www.fastconnect.fr/.
	Silpion IT-Solutions GmbH is an expanding consultancy in Germany. Founded in 2000, today 97 highly qualified employees work for us. Of these, 57 are employed directly and 40 as freelancers. Our focus is safety and functionality, while we focus on innovative concepts and advanced technologies.
	Orange11 is a leading full service supplier of high-quality custom-built applications. Our specialist teams provide end-to-end project delivery services to support the full project life-cycle: from the first design stages to ongoing maintenance. In addition, Orange11 can provide high-end consulting and training services.
	KMMX Located in Mexico City, KMMX provides innovative training on web and mobile technologies. KMMX's mission is to bring the best tools and learning environment to meet the needs of Mexico's IT professionals.
	Rivet Logic is an award-winning consulting and systems integration firm that helps organizations better engage with customers, improve collaboration and streamline business operations. Through a full suite of solutions for content management, collaboration and community, Rivet Logic enables organizations to fully leverage the power of industry-leading open source software. Rivet Logic-Artisans of Open Source. Visit www.rivetlogic.com.
	RadiantBlue Technologies is a small business offering specialized information technology solutions, consulting, and program support services for national security, counter-terrorism, and other federal government customers. RadiantBlue has offices in Chantilly, VA, Colorado Springs, CO, and Melbourne Beach, FL. Areas of expertise include: open source software support and design, geospatial software, modeling and simulation and signal processing.
	El Taller Web is a knowledge center focused on laying the necessary foundations to increase innovation, awareness and the use of web related technologies, leveraging that into IT solutions for both our clients and our community. For our customers, our certified staff works on helping them obtain solutions that deliver the most cost-effective value for their needs, reducing TCO and time to market, and increasing ROI. For our community, we strive to give back by sharing what we've learned in order to advance individual and collective knowledge in our midst.
	Circar Consulting is a global leader in Open Source product development, web application development, Saas product development and mobile apps development. We are big on Big data and a global partner of 10gen (MongoDB) offering advisory consulting, implementation, and other services around the MongoDB application. Headquartered in the heart of Silicon Valley, United States, Circar Consulting has two development centers in USA and India. We provide services to companies of all sizes, from Fortune 500 to Silicon Valley start ups.

	PROTEUS Technologies is a woman-owned, small business providing high-end software and systems engineering services to the Intelligence Community, Federal Executive Departments, HealthCare, and Commercial Industry. Our systems and software engineering staff base is highly educated and experienced, we are a multi-million dollar company with a proven track record of excellence and commitment to mission success. PROTEUS creates highly scalable Big Data solutions for the Cloud, using traditional and non-traditional NoSQL data stores, and develops complex, high performance Analytics to enable real time decisions.

Other Resources

- Hashrocket is a full-service design and development firm that builds successful web businesses. Hashrocket continually creates and follows best practices and surround themselves with passionate and talented craftsmen to ensure the best results for you and your business.
- LightCube Solutions provides PHP development and consulting services, as well as a lightweight PHP framework designed for MongoDB called 'photon'.
- Squeejee builds web applications on top of MongoDB with multiple sites already in production.

Analytics and BI Tools

- Business Intelligence

Visit the 10gen Offices

Bay Area

10gen's West Coast office is located in downtown Palo Alto.

555 University Ave.
Palo Alto, CA 94301

View Larger Map

Directions:

From 101:

- Take 101 to the University Ave Exit.
- Drive West to downtown Palo Alto. The 10gen office is at the northeast corner of the intersection of Tasso St. and University Ave.

From 280:

- Driving southbound, take 280 to the Sand Hill Road exit. Driving northbound, take 280 to the Alpine Road exit; make a right onto Sand Hill Road from Alpine Road.
- Take Sand Hill Road east to the end at El Camino Real
- Make a right onto El Camino Real, then veer right towards University Ave.
- Make a left onto University Ave. The 10gen office is at the northeast corner of the intersection of Tasso St. and University Ave.

New York City

10gen's East Coast office is located in the SOHO neighborhood of NYC.

578 Broadway
7th Floor
New York, NY 10012

View Larger Map

User Feedback

"I just have to get my head around that mongodb is really _this_ good"
-muckster, #mongodb

"Guys at Redmond should get a long course from you about what is the software development and support "
-kunthar@gmail.com, mongodb-user list

"#mongoDB keep me up all night. I think I have found the 'perfect' storage for my app "
-elpargo, Twitter

"Dude, you guys are legends!"
-Stii, mongodb-user list

"Times I've been wowed using MongoDB this week: 7."
-tpitale, Twitter

Community Blog Posts

B is for Billion
-Wordnik (July 9, 2010)

[Reflections on MongoDB]
-Brandon Keepers, Collective Idea (June 15, 2010)

Building a Better Submission Form
-New York Times Open Blog (May 25, 2010)

Notes from a Production MongoDB Deployment
-Boxed Ice (February 28, 2010)

NoSQL in the Real World
-CNET (February 10, 2010)

Why I Think Mongo is to Databases what Rails was to Frameworks
-John Nunemaker, Ordered List (December 18, 2009)

MongoDB a Light in the Darkness...
-EngineYard (September 24, 2009)

Introducing MongoDB
-Linux Magazine (September 21, 2009)

Choosing a non-relational database; why we migrated from MySQL to MongoDB
-Boxed Ice (July 7, 2010)

The Other Blog - The Holy Grail of the Funky Data Model
-Tom Smith (June 6, 2009)

GIS Solved - Populating a MongoDb with POIs
-Samuel

Community Presentations

Scalable Event Analytics with MongoDb and Ruby on Rails
Jared Rosoff at RubyConfChina (June 2010)

How Python, TurboGears, and MongoDB are Transforming SourceForge.net
Rick Copeland at PyCon 2010

MongoDB
Adrian Madrid at Mountain West Ruby Conference 2009, video

MongoDB - Ruby friendly document storage that doesn't rhyme with ouch
Wynn Netherland at Dallas.rb Ruby Group, slides

MongoDB
jnunemaker at Grand Rapids RUG, slides

Developing Joomla! 1.5 Extensions, Explained (slide 37)
Mitch Pirtle at Joomla!Day New England 2009, slides

Drop Acid (slide 31) (video)
Bob Ippolito at Pycon 2009

Python and Non-SQL Databases (in French, slide 21)
Benoit Chesneau at Pycon France 2009, slides

Massimiliano Dessì at the Spring Framework Italian User Group

- MongoDB (in Italian)
- MongoDB and Scala (in Italian)

Presentations and Screencasts at Learnivore
Frequently-updated set of presentations and screencasts on MongoDB.

Benchmarking

We keep track of user benchmarks on the Benchmarks page.

Job Board

Redirecting...

Redirection Notice
This page should redirect to http://jobs.mongodb.org/ in about 2 seconds.

About

- Gotchas
- Philosophy
- Use Cases
- MongoDB-Based Applications
- Events
- MongoDB Archives and Articles
- Benchmarks
- FAQ
- Misc
- Licensing

Gotchas

Note: This page was inspired by a blog post by rsmith. Thanks. This page is intended to be a bit of a living document over time. At the time of its creation it is still a bit of a draft...

Always use 64 bit builds for production.

(MongoDB uses memory mapped files.) See the 32 bit doc page for more information.

32 bit builds exist to support use on development machines and also for other miscellaneous things such as replica set arbiters.

BSON document size limit

There is a limit – at the time of this writing 16MB per document. If you have large objects, use GridFS instead.

Set an appropriate WriteConcern for your write operations

MongoDB requires an explicit request for acknowledgement of the results of a write. This is the getLastError command. If you don't call it at all, that is likely bad – unless you are doing so with intent. The intent is to provide a way to do batch operations without continuous client/server turnarounds on every write of a batch of say, a million. The drivers support automatically calling this if you indicate a "write concern" for your connection to the database.

For example, if you try to insert a document above the BSON size limit indicated in the above section, getLastError/writeconcern would return an error – **if** you ask for those.

While this can be very useful when used appropriately, it is acknowledged this can be confusing at first as this is an untraditional pattern.

See also the getLastError/WriteConcern parameters – particularly 'j' and 'w' parameters.

Schemaless does not mean you have no Schema

(Headline here is quoting rsmith, thanks.)

MongoDB is referred to as "schemaless" as fields and the types of the values in fields are dynamic and flexible, even in a single collection. This makes iterative development and polymorphism much easier.

However, one *does* do schema design for applications using MongoDB. This can be very important. For fields there is an implicit schema: it is not unusual for collection's to have highly homogenous document structures within them. Further, other aspects of the schema are:

- the exact set of collections to be used
- the indexes to be used, which are created explicitly except for the _id index
- shard key declarations, which are explicit and quite important as it is hard to change shard keys later

One very simple rule-of-thumb is to not verbatim import data from a relational database unmodified: you will generally want to "roll up" certain data into richer documents that use some embedding of nested documents and arrays (and/or arrays of subdocuments).

Updates by default affect only one document

Perhaps this should not have been the default. If you with to modify many documents, set the multi update parameter to `true`. The mongo shell syntax is:

```
> db.my_collection_name.update(my_query, my_update_expression, bool_upsert, bool_multi)
```

Set bool_multi to true when updating many documents. Otherwise only the first matched will update!

MongoDB strings are case sensitive

So searching for "joe" will not find "Joe".

Consider:

- storing data in a normalized case format, or
- using regular expressions ending with /i
- and/or using $toLower or $toUpper in the aggregation framework

Type sensitive fields

MongoDB data – which is JSON-style, specifically, BSON format – has several data types. Thus if you have

```
{ x : "123" }
```

the query

```
db.mycollection.find( { x : 123 } )
```

will not return that document as a result as the number 123 does not equal the string "123".

Locking

Older versions of MongoDB used a "global lock"; use MongoDB v2.2+ for better results. More info.

Packages

Be sure you have the latest stable release if you are using a package manager. You can see what is current on the Downloads page, even if you then choose to install via a package manager.

Don't use an even number of Replica Set members

Replica sets perform consensus elections. Use either an odd number of members (e.g., three) or else use an arbiter to get up to an odd number

of votes.

Don't disable journaling

More info

Watch for lagging replica set members.

This is important as MongoDB replica sets support automatic failover. Thus you want your secondaries to be up-to-date. You have a few options here:

1. Monitoring and alerts for any lagging can be done via various means. MMS shows a graph of replica set lag
2. Using getLastError with w:'majority', you will get a timeout or no return if a majority of the set is lagging. This is thus another way to guard against lag and get some reporting back of its occurrence.
3. Or, if you want to fail over manually, you can set your secondaries to `priority:0` in their configuration. Then manual action would be required for a failover. This is practical for a small cluster; for a large cluster you will want automation.

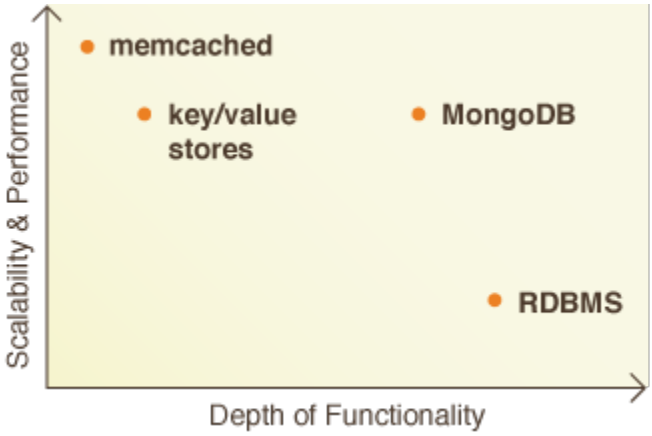
More info

More

- Pick your shard keys carefully! They cannot be changed except manually by making a new collection.
- You cannot shard an existing collection over 256G. This will eventually be removed but is a limit in the system today. You could as a workaround create a new sharded collection and copy over the data via some script – albeit that will likely take a while at this size.
- Unique indexes are not enforced across shards except for the shard key iteslef.
- Consider pre-splitting a sharded collection before a massive bulk import. Usually this isn't necessary but on a bulk import of size it is helpful.
- Use security/auth mode if you need it. It is not on by default; rather a trusted environment is assumed.
- You do not have fully generalized transactions. Create rich documents and read the preceeding link and consider the use case – often there is a good fit.
- Disable NUMA - we find that works better. This will be printed as an informational warning at `mongod` startup on a NUMA box usually.
- Avoid excessive prefetch/readahead on the filesystem. Check your prefetch settings. Note on linux the parameter is in *sectors*, not bytes. 32KBytes (a setting of 64 sectors) is pretty reasonable.
- Check "limits" settings.
- Use SSD if available and economic. Spinning disks can work well but SSDs and MongoDB are a nice combo. See also: Production Notes for more info.
- Don't go too crazy with pool sizes on clients. If you have 100 client machines and each has a pool size of 1000 connections, that would be a worst case of 100,000 connections to mongodb – which would be too much if to a single `mongod` or `mongos` (if to a cluster with many mongos, might be ok). So do the math and try to maintain a reasonable #. One thousand is fine, 100K, too much, 10K is ok.

Philosophy

Design Philosophy



- New database technologies are needed to facilitate horizontal scaling of the data layer, easier development, and the ability to store order(s) of magnitude more data than was used in the past.
- A non-relational approach is the best path to database solutions which scale horizontally to many machines.
- It is unacceptable if these new technologies make writing applications harder. Writing code should be faster, easier, and more agile.
- The document data model (JSON/BSON) is easy to code to, easy to manage(schemaless), and yields excellent performance by grouping relevant data together internally.
- It is important to keep deep functionality to keep programming fast and simple. While some things must be left out, keep as much as possible – for example secondaries indexes, unique key constraints, atomic operations, multi-document updates.
- Database technology should run anywhere, being available both for running on your own servers or VMs, and also as a cloud pay-for-what-you-use service.

Focus

MongoDB focuses on four main things: flexibility, power, speed, and ease of use. To that end, it sometimes sacrifices things like fine grained control and tuning, overly powerful functionality like MVCC that require a lot of complicated code and logic in the application layer, and certain ACID features like multi-document transactions.

Flexibility

MongoDB stores data in JSON documents (which we serialize to BSON). JSON provides us a rich data model that seamlessly maps to native programming language types, and since its schema-less, makes it much easier to evolve your data model than with a system with enforced schemas such as a RDBMS.

Power

MongoDB provides a lot of the features of a traditional RDBMS such as secondary indexes, dynamic queries, sorting, rich updates, upserts (update if document exists, insert if it doesn't), and easy aggregation. This gives you the breadth of functionality that you are used to from an RDBMS, with the flexibility and scaling capability that the non-relational model allows.

Speed/Scaling

By keeping related data together in documents, queries can be much faster than in a relational database where related data is separated into multiple tables and then needs to be joined later. MongoDB also makes it easy to scale out your database. Autosharding allows you to scale your cluster linearly by adding more machines. It is possible to increase capacity without any downtime, which is very important on the web when load can increase suddenly and bringing down the website for extended maintenance can cost your business large amounts of revenue.

Ease of use

MongoDB works hard to be very easy to install, configure, maintain, and use. To this end, MongoDB provides few configuration options, and instead tries to automatically do the "right thing" whenever possible. This means that MongoDB works right out of the box, and you can dive right into developing your application, instead of spending a lot of time fine-tuning obscure database configurations.

See also:

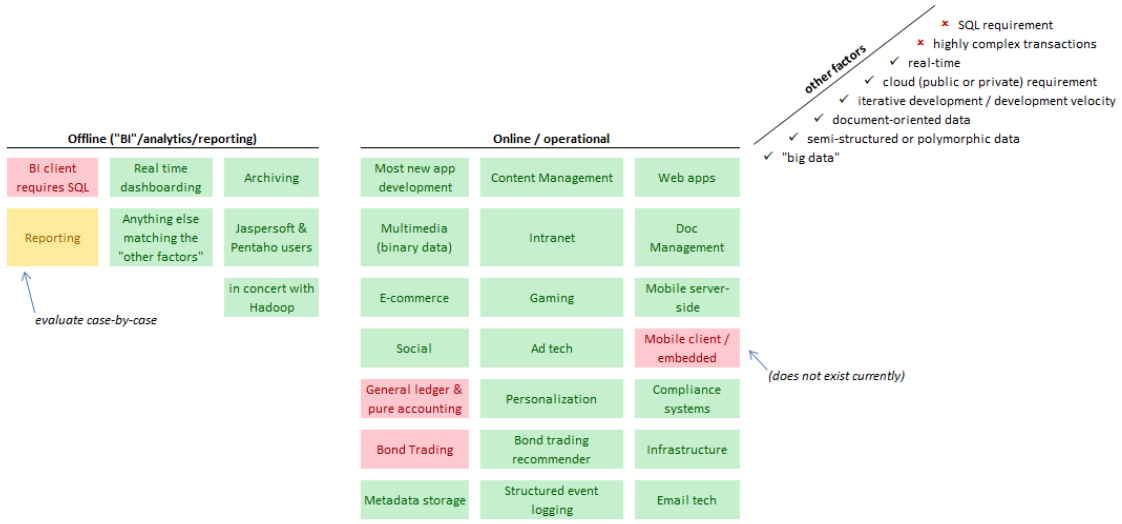
- Introduction

Use Cases

- Why do people use MongoDB?
- When should you consider using MongoDB?
- When should you use something else?
- If you had to pick one thing you wouldn't use it for, what's the first thing that comes to mind?
- Do big companies use MongoDB?
- Should I think of it as a system-of-record or as a caching tier?
- How may databases should my organization standardize on? Is one-size-fits-all over?
- Common Use Cases: Articles and Videos

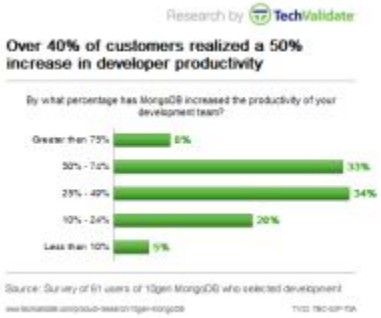
The goal of MongoDB is to be a fairly general purpose tool that can be used for a variety of applications. Common uses include online/operational tasks and applications, as well as select business intelligence / analytics use cases.

The Production Deployments page provides hundreds of examples of real world use cases; see also the use case docs in the MongoDB Manual.



Why do people use MongoDB?

Two reasons: *easy scale-out*, and *coding velocity* ("agility").



The original catalyst for the NoSQL space is the growth of "big data" and the need for scalable databases to store that data. However, a secondary but often equally important factor in MongoDB adoption is increased software development "agility."

MongoDB developers report big productivity gains writing their apps, and often use Mongo regardless of whether scaling out is required. While there are MongoDB systems that exceed one million operations per second, MongoDB is also used for many apps that *easily run on a single server*.

MongoDB makes app development faster because it

1. eliminates object-relational-mapping work and the so-called "impedance mismatch". (Mongo uses JSON, which maps well to object-style data.)
2. allows dynamic schemas ("schemaless" operation), which is synergistic with agile software development methodologies, which themselves hugely speed project completion.
3. makes it much, much easier for the developer to store and manipulate complex data and polymorphic data.
4. reduces the amount of work required to scale out the application and increase system speed.

When should you consider using MongoDB?

The following attributes are good indicators you should consider MongoDB for your application:

- You find yourself coding around database performance issues – for example adding lots of caching.
- You are storing data in flat files.
- You are batch processing yet you need real-time.
- You are doing agile development, for example, Scrum. (MongoDB's flexible schema enables iterative development.)
- Your data is complex to model in a relational db. For example a complex derivative security at an investment bank might be hard to store in a traditional relational format. Electronic health records is another example. If you were considering using an XML store, that's a strong sign to consider MongoDB and its use of JSON/BSON.
- Your project is very late, and it is database intensive.
- You have been forced to use expensive SANs, proprietary servers, or proprietary networks for your existing database solution.
- You are deploying to a public or private cloud.
- Your application has simple requirements for transactions. MongoDB models data as documents, and single document updates are atomic and durable in MongoDB, unlike many other NoSQL products. This level of transactional functionality is sufficient for many applications. For example many users have built e-commerce systems using MongoDB.
- You have a workload that entails high volume 'matching' of records such as trade clearing, transaction reconciliation, fraud detection or system/software security applications.
- Several types of analytical workloads where one or more of the following are true:
    - the analytics are realtime
    - the data is very complicated to model in a relational schema
    - the data volume is huge
    - the source data is already in a mongo database
    - reporting can be addressed by the declarative aggregation functionality available in the aggregation framework (vv2.2+), or where more elaborate needs can be addressed by MongoDB's MapReduce functionality
    - popular BI tools such as Jaspersoft and Pentaho can be used as these vendors have built integrations to MongoDB for reporting and visualization. Over time we expect other vendors to add support for MongoDB.

When should you use something else?

The following attributes are good indicators MongoDB may *not* be right for your application:

- *Applications that require SQL.* MongoDB supports ad hoc queries and has its own query language, but does not support SQL. This is likely a show stopper for many legacy apps until they are due for a refresh.
- *Applications with a heavy emphasis on complex transactions.* These systems typically require updates to multiple documents within a single transaction, XA transactions, multi-statement transactions, or other complex transactional operations , which MongoDB doesn't support.
- *Traditional Data Warehousing.* Traditional relational data warehouses are well suited for certain business intelligence problems and include complex or legacy SQL-based analytical workloads, or they require SQL for specific BI or reporting tools.

If you had to pick one thing you wouldn't use it for, what's the first thing that comes to mind?

A double-entry bookkeeping accounting system is probably the perfect anti-example.

- The intrinsically tabular application data maps well to relational data models
- The application requires very complex transactions at times
- The application requires a good amount of reporting. Relational and SQL are quite good at reporting
- The volume of data is probably relatively small

However, MongoDB is a good fit for reconciling data *between two bookkeeping systems* because 1) it speeds application development, 2) it stores and can query on data in different schemas, 3) it can rapidly execute the queries necessary for reconciliation, and 4) because this application does not require complex transactions.

Do big companies use MongoDB?

Yes. Quite a bit actually, from most industries including for example financial services, and including some companies with over $100B in annual revenues.

Should I think of it as a system-of-record or as a caching tier?

Both can be good use cases. Many, many systems are deployed by users where MongoDB is the system of record. But in addition, it can be a very good caching tier, it is fast, and in addition has good properties on a server or cluster restart as there is data persisted at the cache server.

How may databases should my organization standardize on? Is one-size-fits-all over?

Historically there is already some diversity: most large enterprises use an RDBMS for OLTP (e.g., Oracle), a data warehouse technology (e.g., Netezza), and some niche tools for special problems (e.g. a time series db).

Today we are seeing organizations add one more db tool to their toolbox: a NoSQL database.

We are also seeing some organizations (for example, one very large UK media company) adopt a **Mongo First** policy. That is to say, MongoDB is their *default* for building new applications. They use other tools, but by default, MongoDB. This is a good example that MongoDB is fairly general purpose. Note the word "application": they might at times use other things for pure reporting. Other companies are beginning to do this too; obviously these organizations have already done a few projects with MongoDB and are quite comfortable with it at this point.

Oh, and if you are a startup, just use MongoDB. :-)

Common Use Cases: Articles and Videos

Note: You'll find more on the production deployments page and the 10gen videos page.

- Blog post: on why flexible schemas are useful
- Analytics / BI
  - Analytics talks
  - Blog post: real-time dashboarding
- Archiving blog post
- Blog post : structured event logging
- Content Management Presentations Systems - as a document-oriented (JSON) database, MongoDB's flexible schemas are a good fit for this.
- E-Commerce
  - Blog post : E-Commerce
  - E-Commerce user talks
- Finance user talks
- Gaming user talks
- Gaming Presentations. High performance small read/writes are a good fit for MongoDB; also for certain games geospatial indexes can be helpful.
- Government user talks
- Media user talks
- Mobile example - FourSquare talk video. Specifically, the server-side infrastructure of mobile systems.. Geospatial key here.
- Web infrastructure presentations MongoDB is very good at real-time inserts, updates, and queries. Scalability and replication are

provided which are necessary functions for large web sites' real-time data stores.

Business Intelligence

- Pentaho
- Jaspersoft
- Precog

Pentaho

Pentaho develops Enterprise and Community tools for business analytics and data integration. Pentaho's products integrate well with MongoDB, bringing Pentaho's visual interfaces for high-performance data ingestion, extraction and manipulation, data discovery, visualization, and predictive analytics to MongoDB. Specifically, Pentaho's tools provide a complete business analytics suite, visual tools to explore data in MongoDB, quick insights and reporting from MongoDB data and integration into enterprise-level analytics solutions.

To get started,

- Visit the Pentaho for MongoDB resource center.
- Once you've got MongoDB setup and running, download and install either the Pentaho Business Analytics Enterprise Edition or the Pentaho Kettle Community Edition.
- Next, review the Pentaho Big Data toolset and then learn about how to integrate MongoDB and Pentaho's tools.

Jaspersoft

Jaspersoft develops business intelligence tools and integration tools for data from several sources. Jaspersoft's intelligent connector integrates MongoDB with the full Jaspersoft BI Suite, providing flexible and affordable reporting, ad hoc analysis, and dashboarding of MongoDB data. As a result, both business and technical users can visualize and analyze MongoDB data more rapidly for competitive advantage. JasperReports Server supports native connections to MongoDB, both on-premises and in the cloud.

- First, install and setup MongoDB
- Next you can download and install JasperReports Server (Enterprise or Community) to create a centralized reporting infrastructure.
- To design reports download the iReport Report Designer.
- To get started, visit Simple Reporting on MongoDB for steps on using iReport to design reports with data from MongoDB.

Precog

Precog is a data science platform designed for developers and data scientists to build sophisticated big data capabilities into their applications. The Precog platform offers an end-to-end solution for programmatic big data analysis: from capture and storage, to cleaning and enrichment, to deep analysis designed to power intelligent, insightful features inside applications.

Precog has released a MongoDB implementation of the Precog platform, which bundles all of the Precog technology into a free package that anyone can download. Precog for MongoDB empowers companies to quickly and easily build advanced big data analytics features into their applications on top of their existing MongoDB database. With Precog, you can now analyze all the data in your MongoDB database, without forcing you to export data into another tool or write any custom code. To get started:

- Visit the Precog for MongoDB webpage.
- Complete the short form to download the zipped file (includes JAR, scripts and config file).
- Visit our Developer Center for the installation and configuration instructions. If you run into any trouble, please email the Precog team at support@precog.com.

Hadoop Quick Start

- Prerequisites
  - Hadoop
  - MongoDB
  - Miscelllaneous
- Building MongoDB Adapter
- Examples
  - Load Sample Data
  - Treasury Yield
  - UFO Sightings

MongoDB and Hadoop are a powerful combination and can be used together to deliver complex analytics and data processing for data stored in MongoDB. The following guide shows how you can start working with the MongoDB-Hadoop adapter. Once you become familiar with the adapter, you can use it to pull your MongoDB data into Hadoop Map-Reduce jobs, process the data and return results back to a MongoDB collection.

Prerequisites

Hadoop

In order to use the following guide, you should already have Hadoop up and running. This can range from a deployed cluster containing multiple nodes or a single node pseudo-distributed Hadoop installation running locally. As long as you are able to run any of the examples on your Hadoop installation, you should be all set. The following versions of Hadoop are currently supported:

- 0.20/0.20.x
- 1.0/1.0.x
- 0.21/0.21.x
- CDH3
- CDH4

MongoDB

The latest version of MongoDB should be installed and running. In addition, the MongoDB commands should be in your `$PATH`.

Miscelllaneous

In addition to Hadoop, you should also have `git` and JDK 1.6 installed.

Building MongoDB Adapter

The MongoDB-Hadoop adapter source is available on github. First, clone the repository and get the `release-1.0` branch:

```
$ git clone https://github.com/mongodb/mongo-hadoop.git
$ git checkout release-1.0
```

Now, edit `build.sbt` and update the build target in `hadoopRelease in ThisBuild`. In this example, we're using the CDH3 Hadoop distribution from Cloudera so I'll set it as follows:

```
hadoopRelease in ThisBuild := "cdh3"
```

To build the adapter, use the self-bootstrapping version of `sbt` that ships with the MongoDB-Hadoop adapter:

```
$ ./sbt package
```

Once the adapter is built, you will need to copy it and the latest stable version of the MongoDB Java driver to your `$HADOOP_HOME/lib` directory. For example, if you have Hadoop installed in `/usr/lib/hadoop`:

```
$ wget --no-check-certificate https://github.com/downloads/mongodb/mongo-java-driver/mongo-2.7.3.jar
$ cp mongo-2.7.3.jar /usr/lib/hadoop/lib/
$ cp core/target/mongo-hadoop-core_cdh3u3-1.0.0.jar /usr/lib/hadoop/lib/
```

Examples

Load Sample Data

The MongoDB-Hadoop adapter ships with a few examples of how to use the adapter in your own setup. In this guide, we'll focus on the UFO Sightings and Treasury Yield examples. To get started, first load the sample data for these examples:

```
$ ./sbt load-sample-data
```

To confirm that the sample data was loaded, start the `mongo` client and look for the `mongo_hadoop` database and be sure that it contains the `ufo_sightings.in` and `yield_historical.in` collections:

```
$ mongo
MongoDB shell version: 2.0.5
connecting to: test
> show dbs
mongo_hadoop    0.453125GB
> use mongo_hadoop
switched to db mongo_hadoop
> show collections
system.indexes
ufo_sightings.in
yield_historical.in
```

Treasury Yield

To build the Treasury Yield example, we'll need to first edit one of the configuration files uses by the example code and set the MongoDB location for the input (`mongo.input.uri`) and output (`mongo.output.uri`) collections (in this example, Hadoop is running on a single node alongside MongoDB). :

```
$ emacs examples/treasury_yield/src/main/resources/mongo-treasury_yield.xml
...
  <property>
    <!-- If you are reading from mongo, the URI -->
    <name>mongo.input.uri</name>
    <value>mongodb://127.0.0.1/mongo_hadoop.yield_historical.in</value>
</property>
  <property>
    <!-- If you are writing to mongo, the URI -->
    <name>mongo.output.uri</name>
    <value>mongodb://127.0.0.1/mongo_hadoop.yield_historical.out</value>
</property>
...
```

Next, edit the main class that we'll use for our MapReduce job (`TreasuryYieldXMLConfig.java`) and update the class definition as follows:

```
$ emacs
examples/treasury_yield/src/main/java/com/mongodb/hadoop/examples/treasury/TreasuryYieldXMLConfig.java
...
public class TreasuryYieldXMLConfig extends MongoTool {

    static{
        // Load the XML config defined in hadoop-local.xml
// Configuration.addDefaultResource( "hadoop-local.xml" );
Configuration.addDefaultResource( "mongo-defaults.xml" );
        Configuration.addDefaultResource( "mongo-treasury_yield.xml" );
    }

    public static void main( final String[] pArgs ) throws Exception{
        System.exit( ToolRunner.run( new TreasuryYieldXMLConfig(), pArgs ) );
    }
}
...
```

Now let's build the Treasury Yield example:

```
$ ./sbt treasury-example/package
```

Once the example is done building we can submit our MapReduce job:

```
$ hadoop jar examples/treasury_yield/target/treasury-example_cdh3u3-1.0.0.jar
com.mongodb.hadoop.examples.treasury.TreasuryYieldXMLConfig
```

This job should only take a few moments as it's a relatively small amount of data. Now check the output collection data in MongoDB to confirm that the MapReduce job was successful:

```
$ mongo
MongoDB shell version: 2.0.5
connecting to: test
> use mongo_hadoop
switched to db mongo_hadoop
> db.yield_historical.out.find()
{ "_id" : 1990, "value" : 8.552400000000002 }
{ "_id" : 1991, "value" : 7.8623600000000025 }
{ "_id" : 1992, "value" : 7.008844621513946 }
{ "_id" : 1993, "value" : 5.866279999999999 }
{ "_id" : 1994, "value" : 7.085180722891565 }
{ "_id" : 1995, "value" : 6.573920000000002 }
{ "_id" : 1996, "value" : 6.443531746031742 }
{ "_id" : 1997, "value" : 6.353959999999992 }
{ "_id" : 1998, "value" : 5.262879999999994 }
{ "_id" : 1999, "value" : 5.646135458167332 }
{ "_id" : 2000, "value" : 6.030278884462145 }
{ "_id" : 2001, "value" : 5.02068548387097 }
{ "_id" : 2002, "value" : 4.61308 }
{ "_id" : 2003, "value" : 4.013879999999999 }
{ "_id" : 2004, "value" : 4.271320000000004 }
{ "_id" : 2005, "value" : 4.288880000000001 }
{ "_id" : 2006, "value" : 4.7949999999999955 }
{ "_id" : 2007, "value" : 4.634661354581674 }
{ "_id" : 2008, "value" : 3.6642629482071714 }
{ "_id" : 2009, "value" : 3.2641200000000037 }
has more
>
```

UFO Sightings

This will follow much of the same process as with the Treasury Yield example with one extra step; we'll need to add an entry into the build file to compile this example. First, open the file for editing:

```
$ emacs project/MongoHadoopBuild.scala
```

Next, add the following lines starting at line 72 in the build file:

```
...
  lazy val ufoExample = Project( id = "ufo-sightings",
                                 base = file("examples/ufo_sightings"),
                                 settings = exampleSettings ) dependsOn ( core )
...
```

Now edit the UFO Sightings config file and update the `mongo.input.uri` and `mongo.output.uri` properties:

```
$ emacs examples/ufo_sightings/src/main/resources/mongo-ufo_sightings.xml
...
  <property>
    <!-- If you are reading from mongo, the URI -->
    <name>mongo.input.uri</name>
    <value>mongodb://127.0.0.1/mongo_hadoop.ufo_sightings.in</value>
</property>
  <property>
    <!-- If you are writing to mongo, the URI -->
    <name>mongo.output.uri</name>
    <value>mongodb://127.0.0.1/mongo_hadoop.ufo_sightings.out</value>
</property>
...
```

Next edit the main class for the MapReduce job in `UfoSightingsXMLConfig.java` to use the configuration file:

```
$ emacs
examples/ufo_sightings/src/main/java/com/mongodb/hadoop/examples/ufos/UfoSightingsXMLConfig.java
...
public class UfoSightingsXMLConfig extends MongoTool {

    static{
        // Load the XML config defined in hadoop-local.xml
// Configuration.addDefaultResource( "hadoop-local.xml" );
Configuration.addDefaultResource( "mongo-defaults.xml" );
        Configuration.addDefaultResource( "mongo-ufo_sightings.xml" );
    }

    public static void main( final String[] pArgs ) throws Exception{
        System.exit( ToolRunner.run( new UfoSightingsXMLConfig(), pArgs ) );
    }
}
...
```

Now build the UFO Sightings example:

```
$ ./sbt ufo-sightings/package
```

Once the example is built, execute the MapReduce job:

```
$ hadoop jar examples/ufo_sightings/target/ufo-sightings_cdh3u3-1.0.0.jar
com.mongodb.hadoop.examples.UfoSightingsXMLConfig
```

This MapReduce job will take just a bit longer than the Treasury Yield example. Once it's complete, check the output collection in MongoDB to see that the job was successful:

```
$ mongo
MongoDB shell version: 2.0.5
connecting to: test
> use mongo_hadoop
switched to db mongo_hadoop
> db.ufo_sightings.out.find().count()
21850
```

Hadoop Scenarios

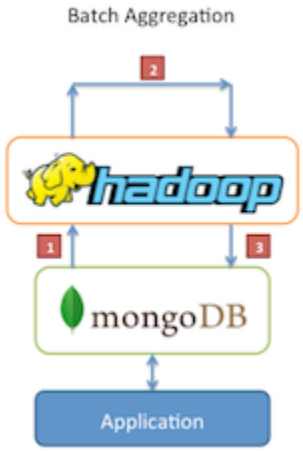
- Batch Aggregation
- Data Warehouse
- ETL Data

The following are some example deployments with MongoDB and Hadoop. The goal is to provide a high-level description of how MongoDB and Hadoop can fit together in a typical Big Data stack. In each of the following examples MongoDB is used as the "operational" real-time data store and Hadoop is uses for offline batch data processing and analysis.

Batch Aggregation

In several scenarios the built-in aggregation functionality provided by MongoDB is sufficient for analyzing your data. However in certain cases, significantly more complex data aggregation may be necessary. This is where Hadoop can provide a powerful framework for complex analytics.

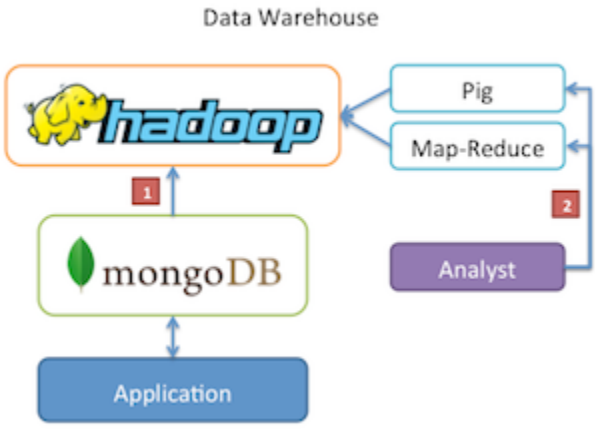
In this scenario data is pulled from MongoDB and processed within Hadoop via one or more Map-Reduce jobs. Data may also be brought in from additional sources within these Map-Reduce jobs to develop a multi-datasource solution. Output from these Map-Reduce jobs can then be written back to MongoDB for later querying and ad-hoc analysis. Applications built on top of MongoDB can now use the information from the batch analytics to present to the end user or to drive other downstream features.



Data Warehouse

In a typical production scenario, your application's data may live in multiple datastores, each with their own query language and functionality. To reduce complexity in these scenarios, Hadoop can be used as a data warehouse and act as a centralized repository for data from the various sources.

In this situation, you could have periodic Map-Reduce jobs that load data from MongoDB into Hadoop. This could be in the form of "daily" or "weekly" data loads pulled from MongoDB via Map-Reduce. Once the data from MongoDB is available from within Hadoop, and data from other sources are also available, the larger dataset data can be queried against. Data analysts now have the option of using either Map-Reduce or Pig to create jobs that query the larger datasets that incorporate data from MongoDB.

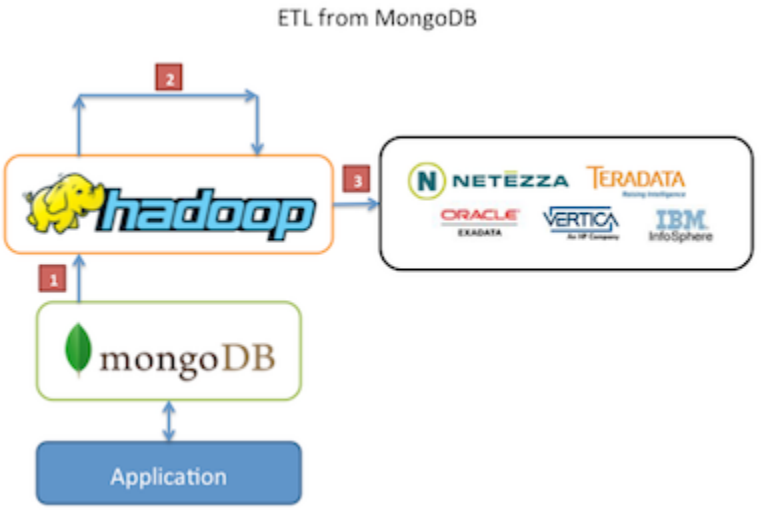


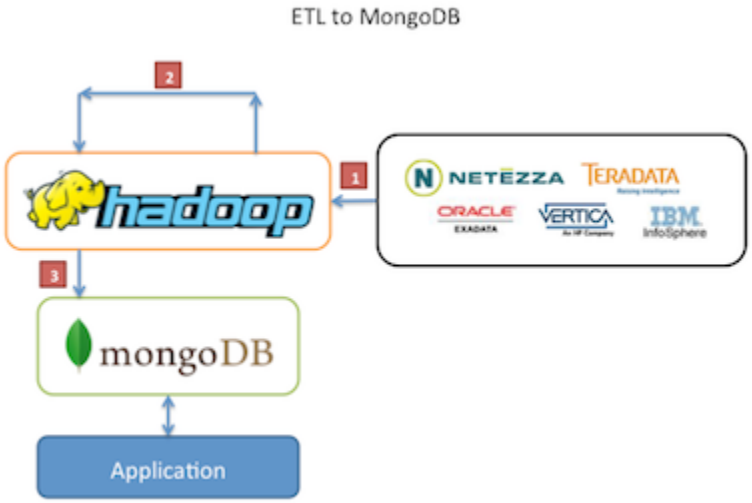
ETL Data

MongoDB may be the operational datastore for your application but there may also be other datastores that are holding your organization's data. In this scenario it is useful to be able to move data from one datastore to another, either from your application's data to another database or vice

versa. Moving the data is much more complex than simply piping it from one mechanism to another, which is where Hadoop can be used.

In this scenario, Map-Reduce jobs are used to extract, transform and load data from one store to another. Hadoop can act as a complex ETL mechanism to migrate data in various forms via one or more Map-Reduce jobs that pull the data from one store, apply multiple transformations (applying new data layouts or other aggregation) and loading the data to another store. This approach can be used to move data from or to MongoDB, depending on the desired result.





How MongoDB is Used in Media and Publishing

We see growing usage of MongoDB in both traditional and new media organizations. In these areas, the challenges for application developers include effectively managing rich content (including user-generated content) at scale, deriving insight into how content is consumed and shared in real-time, weaving personalization and social features into their applications and delivering content to a wide variety of browsers and devices.

From a data storage perspective, there is a need for databases that make it easy to rapidly develop and deploy interactive, content-rich web and mobile application, while cost-effectively meeting performance and scale requirements. Specifically, MongoDB is good fit for application across the media and publishing world for the following reasons:

- The document model is natural fit content-rich data
- Schema-free JSON-like data structures make it easy to import, store, query, and deliver structured and semi-structured content
- High-performance and horizontal scalability for both read and write intensive applications

MongoDB for Content management

MongoDB's document model makes it easy to model content and associated metadata in flexible ways. While the relational model encourages dividing up related data into multiple tables, the document model (with its support for data structures such as arrays and embedded documents)

encourages grouping related pieces of data within a single document. This leads to a data representation that is both efficient and closely matches objects in your application
As an example, the following document represents how a blog post (including its tags and comments) can be modeled with MongoDB:

```
{
_id:
author: “Nosh”
title: “Why MongoDB is good for CMS”
date:11/29/2010
body: “MongoDB’s document model makes it easy to model....”
tags: [“media”, “CMS”, “web”, “use case”]
comments: [{author:bob, date:1/1/2011, comment:"I agree"},{..}, {..}]
}
```

Modeling content elements with these patterns also simplifies queries. For example, we can retrieve all blog posts by the author ‘nosh’ which have the tag mongodb with the query,

```
find({author:”nosh”, tags:”mongodb”)
```

Flexible document-based representation, efficient and simple queries and scalability makes MongoDB a well suited as a datastore for content management systems. The Business Insider has built their content management system from the ground up using MongoDB and PHP, which serves over 2 million visits/month. For sites based on Drupal, Drupal 7 now makes it easier to use MongoDB as a datastore. Examiner.com, ported their legacy CMS (based on ColdFusion and Microsoft SQLServer) to Drupal 7 and a hybrid of MySQL and MongoDB. You can read a case study about the how the examiner.com (a top 100 website, and one of most trafficked Drupal deployments) made the transition.

MongoDB can also be used to augment existing content management systems with new functionality. One area that we see MongoDB used increasingly is as a metadata store for rich media. MongoDB’s document model makes it simple to represent the attributes for an asset (e.g. author, dates, categories, versions, etc) and a pointer to the asset (e.g. on a filesystem or on S3) as document and then efficiently search or query the metadata for display. Additionally, because MongoDB is schema-free, new metadata attributes can be added without having to touch existing records. IGN uses MongoDB as a the metadata store for all videos on IGN Videos and serves up millions of video views per month. Another similar use case for MongoDB is in storing user-submitted content. The New York Times, uses MongoDB as the backend for ‘Stuffy’, their tool for allowing users and editors to collaborate on features driven by user-submitted photos. A brief overview on the tool is here.

How-to Guides

Modeling content, comments, and tags with MongoDB (coming soon)
Modelling image and video metadata with MongoDB (coming soon)
Using MongoDB with Drupal 7 (coming soon)

Roadmap tip: Watch the full-text search ticket

MongoDB for Real-time analytics

The ability to track and change content based on up-to-minute statistics is becoming increasingly important. MongoDB’s fast write performance and features such as upsert and the $inc operator, make it well suited for capturing real time analytics on how content is being consumed. This blog post outlines some basic patterns you can use to capture real-time pageview statistics for your content.

A number of companies are using MongoDB, either internally to track their own content in real-time, or are building platforms based on MongoDB to help other companies get real time statistics on their content: Chartbeat provides analytics for publishers, with live dashboards and APIs showing how users are engaging with their content in real-time. BuzzFeed uses MongoDB to help understand how and when content will go viral, while ShareThis uses MongoDB to power its API that gives publishers insight into how content is shared across the web and social media

How-to guides

Real-time analytics with MongoDB (coming soon)

MongoDB for Social Graphs & Personalization

While systems such as graph databases excel at complex graph traversal problems, MongoDB’s document structure is well suited for building certain types of social and personalization features. Most often, this involves building user profile documents that include a list of friends, either imported from external social networks or in site. For example,

```
{
_id:
user: nosh
email: nosh@10gen.com
friendIDs: [....., …..., …...]
}
```

In this case the friendIDs field is an array with a list of IDs corresponding to profiles of users that are my friends. This data can then be used to generate personalized feeds based on content that my friends have viewed or liked. IGN's social network, MY IGN, uses MongoDB to store profiles of users and generate personalized fields. Users have the ability to import their friends from facebook, 'follow' IGN authors, or follow specific game titles they are interested in. When they log in, they are presented with a personalized feed composed from this data.

How-to guides:

Storing user profiles with MongoDB (coming soon)
Importing social graphs into MongoDB (coming soon)
Generating personalized feeds with MongoDB (coming soon)

MongoDB for Mobile/Tablet Apps

Serving similar content across desktop browsers, mobile browsers, as well as mobile apps is driving developers to build standardized API layers that can be accessed by traditional web application servers, mobile client applications, as well as 3rd party applications. Typically these are RESTful APIs that serve JSON data. With MongoDB's JSON-like data format, building these APIs on top of MongoDB is simplified as minimal code is necessary to translate MongoDB documents and query to JSON representation. Additionally, features such as in-built two-dimensional geospatial indexing allow developers to easily incorporate location-based functionality into their applications.

MongoDB for Data-driven journalism

One of the strengths of MongoDB is dealing with semi-structured data. Data sources such as those produced by governments and other organizations are often denormalized and distributed in formats like CSV files. MongoDB, with its schema-free JSON-like documents is an ideal store for processing and storing these sparse datasets.

The Chicago Tribune uses MongoDB in its Illinois School Report Cards application, which is generated from a nearly 9,000 column denormalized database dump produced annually by the State Board of Education. The application allows readers to search by school name, city, county, or district and to view demographic, economic, and performance data for both schools and districts.

How-to guides:

Importing and Exporting data from MongoDB (coming soon)
Reporting and Visualization with MongoDB (coming soon)

Presentations

- How MTV Networks leverages MongoDB for CMS - MongoNYC Presentation (June 2011)
- Schema Design for Content Management: eHow on MongoDB - MongoSF Presentation (May 2011)
- More Presentations

Use Case - Session Objects

MongoDB is a good tool for storing HTTP session objects.

One implementation model is to have a sessions collection, and store the session object's _id value in a browser cookie.

With its update-in-place design and general optimization to make updates fast, the database is efficient at receiving an update to the session object on every single app server page view.

Aging Out Old Sessions

The best way to age out old sessions is to use the auto-LRU facility of capped collections. The one complication is that objects in capped collections may not grow beyond their initial allocation size. To handle this, we can "pre-pad" the objects to some maximum size on initial addition, and then on further updates we are fine if we do not go above the limit. The following mongo shell javascript example demonstrates padding.

(Note: a clean padding mechanism should be added to the db so the steps below are not necessary.)

```
> db.createCollection('sessions', { capped: true, size : 1000000 } )
{"ok" : 1}
> p = "";
> for( x = 0; x < 100; x++ ) p += 'x';
> s1 = { info: 'example', _padding : p };
{"info" : "example" , "_padding" :
"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
}
> db.sessions.save(s1)
> s1
{"info" : "example" , "_padding" :
"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
, "_id" :  ObjectId( "4aafb74a5761d147677233b0") }

> // when updating later
> s1 = db.sessions.find( { _id : ObjectId( "4aafb74a5761d147677233b0") } )
{"_id" :  ObjectId( "4aafb74a5761d147677233b0")  , "info" : "example" , "_padding" :
"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
}
> delete s._padding;
true
>  s.x = 3; // add a new field
3
> db.sessions.save(s);
> s
{"_id" :  ObjectId( "4aafb5a25761d147677233af")  , "info" : "example" , "x" : 3}
```

MongoDB-Based Applications

Please list applications that leverage MongoDB here. If you're using MongoDB for your application, we'd love to list you here! Email meghan@10gen.com. Also, check out our Contributor Hub project for a list of the most popular projects supporting MongoDB.

See Also

- Production Deployments - Companies and Sites using MongoDB
- Hosting Center

Applications Using MongoDB

c5t

Content-management using TurboGears and Mongo

Calipso

Content management system built using NodeJS and MongoDB

Cube

Cube is an open-source system for visualizing time series data, built on MongoDB, Node and D3.

ErrorApp

ErrorApp tracks errors from your apps. It reports them to you and gathers all information and make reports available to you.

Forward

A full-featured, developer centric open source e-commerce platform that makes custom code easy, with powerful templates & expressive syntax.

Graylog2

Graylog2 is an open source syslog server implementation that stores logs in MongoDB and provides a Rails frontend.

HarmonyApp

Harmony is a powerful web-based platform for creating and managing websites. It helps connect developers with content editors, for unprecedented flexibility and simplicity. For more information, view Steve Smith's presentation on Harmony at MongoSF (April 2010).

Hummingbird

Hummingbird is a real-time web traffic visualization tool developed by Gilt Groupe

Locomotive

Locomotive is an open source CMS for Rails. It's flexible and integrates with Heroku and Amazon S3.

Mogade

Mogade offers a free and simple to use leaderboard and achievement services for mobile game developers.

MongoLantern

MongoLantern is an open source full text search server using MongoDB as index storage, which allows MongoLantern to migrate any changes very easily into account using MongoDB API. It's written originally written in PHP can be migrated to any desired language as required using it's future APIs.

MongoPress

A flexible CMS that uses MongoDB and PHP.

Mongs

A simple, web-based data browser for MongoDB.

Mongeez

Mongeez is an opensource solution allowing you to manage your mongo document changes in a manner that is easy to synchronize with your code changes. Check out mongeez.org.

NewsBlur

NewsBlur is an open source visual feed reader that powers http://newsblur.com. NewsBlur is built with Django, MongoDB, Postgres and RabbitMQ

Quantum GIS

Plugin for Quantum GIS that lets you plot geographical data stored in MongoDB

Scribe

Open source image transcription tool

Shapado

Free and open source Q&A software, open source stackoverflow style app written in ruby, rails, mongomapper and mongodb.

Strider

Strider: Open Source Continuous Integration & Deployment Server.

Thundergrid

Thundergrid is a simple framework written in PHP that allows you to store large files in your Mongo database in seconds.

Websko

Websko is a content management system designed for individual Web developers and cooperative teams.

Events

- Upcoming Meetups and Conferences
- Webinars
- MongoDB User Groups
  - MongoDB Study Groups
  - Training
  - "Office Hours"
- See Also

There are MongoDB User Groups (MUGs) all over the world. Please check out the full listing.

Upcoming Meetups and Conferences

Date	Event	What/Where
December 11-12, 2012	AP Tech Summit	Canbury, NJ
December 12, 2012	MongoDB Tokyo	Tokyo, Japan
December 12, 2012	A Morning with MongoDB Tel Aviv	Petah Tikvah, Israel
December 13, 2012	A Morning with MongoDB Tel Aviv	Petah Tikvah, Israel
December 14, 2012	Open Source Conference mongoDB: Driving a Data Revolution Tim Marston 10gen	Amsterdam, Netherlands
December 17, 2012	MongoDB Israel User Group Moving from SQL to MongoDB Norberto Leite 10gen	Tel Aviv, Israel
January 8-11, 2013	CodeMash	Sandusky, Ohio
January 9, 2013	MongoDB Detroit MongoDB Detroit is a free evening event dedicated to the Open Source Database, MongoDB	Detroit, MI
February 20, 2013	MongoDB LA MongoDB LA is the annual one-day conference dedicated to the open source, non-relational database, MongoDB	Los Angeles, CA
February 25-26, 2013	MongoDB Berlin	Berlin, Germany
May 3-4, 2013	MongoDB Stockholm	Stockholm, Sweden

Webinars

Date	Sessions	Topic	Description
December 20, 2012	11:00am EDT/ 8:00am PDT/ 4:00pm UTC and 2:00pm EDT/ 11:00am PDT/ 7:00pm UTC	**Sharding Best Practices**	Sharding allows you to distribute load across multiple servers and keep your data balanced across those servers. This session will review MongoDB's sharding support, including an architectural overview, design principles, and automation. Also will be presented varied scenarios of handling large amount of data, along with their best sharding strategies.
January 3, 2013	11:00am EDT/ 2:00pm EST/ 7:00pm UTC	**Building Your First Application with MongoDB**	This webinar will introduce the features of MongoDB by walking through how one can building a simple location-based checkin application using MongoDB. The talk will cover the basics of MongoDB's document model, query language, map-reduce framework and deployment architecture.

January 17, 2013	11:00am PST / 2:00pm EST / 7:00pm UTC	**A Total Cost of Ownership Comparison - MongoDB vs. Oracle**	In this webinar, we compare the total cost of ownership (TCO) of MongoDB and Oracle. It can be faster and cheaper to develop and deploy applications on MongoDB than on Oracle Database, yielding both bottom-line benefits – lower developer and administrative costs – and topline advantages – it is easier and faster to evolve applications to meet changing business and market conditions.
January 24, 2013	8:00am PST / 11:00am EST / 4:00pm UTC AND 11:00am PST / 2:00pm EST / 7:00pm UTC	**User Profile Management with MongoDB**	Storing data about users is central to most applications. Whether you run a social network, an identity management system, or an online game, your users are the core of your business. In this webinar, we will discuss MongoDB's capability to accommodate your evolving user data model and sustain the transaction load associated with accessing and updating user data.
January 31, 2013	8:00am PST / 11:00am EST / 4:00pm UTC AND 11:00am PST / 2:00pm EST / 7:00pm UTC	**What's New in Aggregation**	In this webinar, we will introduce MongoDB's new aggregation system that simplifies tasks like counting, averaging, and finding minima or maxima while grouping by keys in a collection. The new aggregation features are not a replacement for map-reduce but will make it possible to do a number of things much more easily without having to resort to the big hammer that is map-reduce. After introducing the syntax and usage patterns, we will give some demonstrations of aggregation using the new system.

MongoDB User Groups

Your Go-To Guide to Running A MongoDB User Group

North America	*MEETUPS*
Atlanta	
Boston	
Chicago	
Cleveland	
Cincinnatti	
Charlotte	
Denver	
DC	
Hawaii	
Huntsville	
Houston	
Los Angeles	
Madison, WI	*None currently scheduled. Join us to be notified!*
Mexico	
Milwaukee	
New York	
New Jersey	
Philadelphia	
Phoenix, AZ	
Raleigh	
San Francisco Bay Area	
Seattle	Date TBD Mapping Flatland: Using MongoDB for an MMO Crossword Game

St. Louis	
Toronto	
South America	***MEETUPS***
Sao Palo	
*Santa Caterina	
Belo Horizonte	
EUROPE	***MEETUPS***
Amsterdam	
Athens	
Barcelona	
Berlin	
Brussels	
Dublin	
East Anglia	
Finland	
Hamburg	
Lisbon	
London	
Mallorca	
Minsk	Join to get updates
München	
Paris	
Rome	
Saint Petersburg	Join us to get updates
Sophia-Antipolis, FR	
Stockholm	
Switzerland	
Thames Valley, UK	
Thessaloniki, Greece	

Middle East	***MEETUPS***
Beirut	
Islamabad	
Israel	
Africa	***MEETUPS***
Accra, Ghana	
Cape Town, South Africa	
Nairobi, Kenya	
Asia and Pacific	***MEETUPS***

Bangalore, India	
Delhi, India	
*Hanoi, Vietnam	_No meetups currently scheduled. Like the facebook page to receive updates!_\
Kuala Lumpur	*None currently scheduled. Please check back*
Melbourne	
Pune, India	
Sydney	

If you're interested in having someone present MongoDB at your conference or meetup, or if you would like to list your MongoDB event on this page, contact meetups at 10gen dot com. Want some MongoDB stickers to give out at your talk? Complete the Swag Request Form.

MongoDB Study Groups

10gen will be sponsoring a number of community-run MongoDB study groups around the globe to go along with MongoDB's Free Online Education initiative. Find a study group near you or find out how to start your own!

Training

MongoDB training from 10gen is the best way to get you and your team up to speed quickly. Our MongoDB Developer Training and MongoDB Administrator Training are comprehensive, two day courses.

Contact us at training@10gen.com if you would like to buy training credits for multiple students or classes, schedule an on-site training, or discuss a custom training option.

MongoDB Essentials	New York, NY	November 27-29, 2012	Register
MongoDB for Developers	London, UK	November 28-29, 2012	Register
Xebia Presents MongoDB for Administrators	Paris, FR	December 06-07, 2012	Register
MongoDB for Developers	Palo Alto, CA	December 11-12, 2012	Register
MongoDB for Administrators	New York, NY	December 18-19, 2012	Register
Xebia Presents MongoDB for Developers	Amsterdam, NL	December 18-19, 2012	Register

"Office Hours"

City	Date	Time	Location	Look For
Atlanta, GA		4-6pm	Please check the Atlanta MongoDB User Group page for upcoming office hours	Look for a MongoDB logo!
New York, NY	Wednesdays	4-6:30pm	10gen Headquarters, 578 Broadway, 7th Floor	10gen holds weekly open "office hours" with whiteboarding and hack sessions at 10gen headquarters. *Please note that December 26 Office Hours will be cancelled due to the Christmas Holiday*
Palo Alto, CA	Thursdays	4-6pm	10gen CA office, 555 University Avenue, Palo Alto, CA 94301	Have questions about MongoDB? Visit the 10gen office in Palo Alto to speak directly with the MongoDB engineers (or just come say hi!).
San Francisco, CA	Every other Monday	5-7pm	Epicenter Cafe, 764 Harrison St, Between 4th St & Lapu St, San Francisco, CA	Stop by the Epicenter Cafe in San Francisco to meet 10gen Software Engineers Sridhar Nanjundeswaran. Ask questions, hack, have some coffee. Look for a laptop with a "Powered by MongoDB" sticker. **Click here for more info and signup.**
London, UK	Every other Thursday	5-7pm	Zetland House, Unit 2G 5-25 Scrutton Street, London, EC2A 4HJ	Stop by the Zetland House, 2nd Floor! Meet 10gen Engineers Ross Lawley, Chris Harris and Dan Roberts. Ask questions, hack, have some coffee. Look for a laptop with a MongoDB leaf sticker. **Click here for more info and signup.**

München, DE	Scheduled through Meetup.com	comSysto GmbH Lindwurmstr. 97, 80337 Munich	Look for the MongoDB Logo!

See Also

- http://www.10gen.com/presentations
- http://www.10gen.com/events
- http://lanyrd.com/topics/mongodb/

MongoDB Study Groups

In order to facilitate community through our education courses, 10gen will be sponsoring a number of MongoDB study groups around the globe so everyone from novices to experts can support one another as they learn. You can meet weekly, monthly or whenever you like to guide one another through the lessons and walk away better prepared to put MongoDB into production.

Interested in creating a MongoDB study group in your area? Send a note to our community team and we'll help you get started. If you're a user group member, consider organizing a MongoDB study group through your local MUG .

New York MongoDB Study Group

- Next Meeting November 26

Hamburg MongoDB Study Group

St Petersburg MongoDB Study Group

- Join the Russian MongoDB Mailing List to be notified of future Meetings
-

MongoDB Lima Study Group

- October 28 (First week)
- November 4 (Second week)
- November 11 (Third week)
- November 25
- December 9 (Sixth Week)

MongoDB User Groups (MUGs)

MongoDB User Groups (MUGs) are a great way for the community to learn from one another about MongoDB best practices, to network, and to have fun. Interested in starting a user group in your city? Submit a proposal and check out our guide to starting a MongoDB User Group to get started! Need some ideas? Check out our go-to-guide for running a MongoDB user group .

MongoDB User Groups

North America

- Atlanta, GA
- Boston, MA
- Chicago, IL
- Cleveland, OH
- Cincinnati, OH
- Charlotte, NC
- Denver, CO
- Los Angeles, CA
- Huntsville, AL
- Hawaii
- Mexico City, MX
- Milwaukee, WI
    - Google+
    - @BrewMUG on Twitter
- New Jersey
    - @sjmug on twitter
- New York, NY

- Philadelphia, PA
- Phoenix, AZ
    - @PHXMUG on Twitter
- Raleigh, NC
- Richmond, VA
- San Diego
- San Francisco, CA
- Seattle, WA
- Silicon Valley, CA
- St. Louis, MO
- Toronto, Ontario
- Washington, DC

South America

- Belo Horizonte, Brazil
- Sao Paulo, Brazil
- Lima, Peru
- Santa Caterina

Europe

- Athens, GR
- Barcelona, ESP
- Berlin, DE
    - @MUGBerlin on Twitter
- Bristol, UK
- Brussels, Belgium
- Sofia, Bulgaria
- Dublin
    - @MongoDBDublin on Twitter
- Dusseldorf
    - @MongoDUS on Twitter
- East Anglia, UK
- Frankfurt, Germany
- Hamburg, Germany
- Helsinki, Finland
- Paris, France
- London, UK
    - @MongoDBLondon on Twitter
- Mallorca, ESP
- Milan, Italy
- Minsk, Belarus
- München, Germany
    - @Mongomuc on Twitter
- Omsk, Russia
    - Google+
    - Vkontakte
- Lisbon, Portugal
    - @MongoDBLisbon on Twitter
- Roma, Italia
    - @MongoDBRoma on Twitter
- Sophia-Antipolis, France
- St. Petersburg
- Stockholm, Sweden
- Thames Valley, UK
- Thessaloniki, GR
- Zurich, Switzerland

**German MongoDB User Group on Xing

Middle East

- Beirut, LB
- Israel
- Islamabad, PK

Africa

- Cape Town
- Ghana
    - AccraMongodbUserGroup on Facebook
- Nairobi

- @MUGNairobi on Twitter

Asia

- Bangalore
- Colombo, Sri Lanka
- China
- Delhi
- Hanoi
- Indonesia
- Manila
- Nepal
- Tokyo, Japan
- Thailand
- Turkey
- Kuala Lumpur, Malaysia
- Pune
- Singapore
- South Korea

Australia

- Melbourne
- Sydney
- Canberra

Video & Slides from Recent Events and Presentations

Table of Contents:

[MongoDB Conferences] [Ruby/Rails] [Python] [Alt.NET] [User Experiences] [More about MongoDB]

MongoDB Conferences

One-day conferences hosted by 10gen. 10gen develops and supports MongoDB.

MongoUK Video (June 2010)
MongoFR Video (June 2010)
MongoNYC (May 2010) and MongoSF (April 2010) Video
MongoSF (April 2010) Slides & Video

Ruby/Rails

Practical Ruby Projects with MongoDB
Alex Sharp, OptimisCorp
Ruby Midwest - June 2010

Scalable Event Analytics with MongoDB and Ruby
Jared Rosoff, Yottaa
RubyConfChina - June 26, 2010

The MongoDB Metamorphosis (Kyle Banker, 10gen)
Million Dollar Mongo (Obie Fernandez & Durran Jordan, Hashrocket)
Analyze This! (Blythe Dunham)
RailsConf
Baltimore, MD
June 7-10

MongoDB
Seth Edwards
London Ruby Users Group
London, UK
Wednesday April 14
Video & Slides

MongoDB: The Way and its Power
Kyle Banker, Software Engineer, 10gen
RubyNation
Friday April 9 & Saturday April 10
Reston, VA
Slides | Video

MongoDB Rules
Kyle Banker, Software Engineer, 10gen
Mountain West Ruby Conference
Salt Lake City, UT
Thursday March 11 & Friday March 12
Slides

MongoDB Isn't Water
Kyle Banker, Software Engineer, 10gen
Chicago Ruby
February 2, 2010
Video | Slides | Photos

Introduction to Mongo DB
Joon Yu, RubyHead
teachmetocode.com
Nov-Dec, 2009
Screencasts

Python

How Python, TurboGears, and MongoDB are Transforming SourceForge.net
Rick Copeland, SourceForge.net
PyCon - Atlanta, GA
February 21, 2010
Slides

Alt.NET

.NET and MongoDB - Building Applications with NoRM and MongoDB
Alex Hung
July 28, 2010

User Experiences

The Future of Content Technologies
Scaling Web Applications with NonSQL Databases: Business Insider Case Study
Ian White, Lead Developer, Business Insider
Gilbane Conference
San Francisco, CA
Thursday, May 20
Slides

Chartbeat and MongoDb - a perfect marriage
Kushal Dave, CTO, Chartbeat & Mike Dirolf, Software Engineer, 10gen
New York City Cloud Computing Meetup
New York, NY
May 18
Slides

Why MongoDB is Awesome
John Nunemaker, CTO, Ordered List
DevNation Chicago
May 15
Slides

Humongous Data at Server Density: Approaching 1 Billion Documents in MongoDB
David Mytton, Founder, Boxed Ice
Webinar
Wednesday May 5
Recording & Slides

Humongous Drupal
DrupalCon San Francisco
Karoly Negyesi, Examiner.com
Saturday April 17
Slides | Video

MongoDB: huMONGOus Data at SourceForge
Mark Ramm, Web Developer, SourceForge
QCon London
Thursday March 11
Slides

Migrating to MongoDB
Bruno Morency, DokDok
Confoo.ca
March 10 - 12
Slides

More about MongoDB

Recording of Michael Dirolf on MongoDB @ E-VAN 07 June 2010

NoSQL-Channeling the Data Explosion
Dwight Merriman, CEO, 10gen
Inside MongoDB: the Internals of an Open-Source
Mike Dirolf, Software Engineer, 10gen
Gluecon
Denver, CO
Wednesday May 26 & Thursday May 27

Schema Design with MongoDB
Kyle Banker, Software Engineer, 10gen
Webinar
Tuesday April 27
Recording and Slides

Dropping ACID with MongoDB
Kristina Chodorow, Software Engineer, 10gen
San Francisco MySQL Meetup
San Francisco, CA
Monday, April 12
Video

Introduction to MongoDB
Mike Dirolf, Software Engineer, 10gen
Emerging Technologies for the Enterprise Conference
Philadelphia, PA
Friday, April 9
Slides

Indexing with MongoDB
Aaron Staple, Software Engineer, 10gen
Webinar
Tuesday April 6, 2010
Video | Slides

TechZing Interview with Mike Dirolf, Software Engineer, 10gen
Monday, April 5
Podcast

Hot Potato and MongoDB
New York Tech Talks Meetup
Justin Shaffer and Lincoln Hochberg
New York, NY
Tuesday March 30
Video

MongoDB Day
Geek Austin Data Series
Austin, TX
Saturday March 27
Photo

Mongo Scale!
Kristina Chodorow, Software Engineer, 10gen
Webcast
Friday March 26
Webcast

NoSQL Live Boston
Boston, MA
Thursday March 11
Recap with slides and MP3

MongoDB: How it Works
Mike Dirolf, Software Engineer, 10gen

Monday March 8, 12:30 PM Eastern Time
Slides

Intro to MongoDB
Alex Sharp, Founder / Lead Software Architect, FrothLogic
LA WebDev Meetup
February 23, 2010
Slides

Introduction to MongoDB
Kristina Chodorow, Software Engineer, 10gen
FOSDEM - Brussels, Belgium
February 7, 2010
Video | Slides | Photos

If you're interested in having someone present MongoDB at your conference or meetup, or if you would like to list your MongoDB event on this page, contact meghan at 10gen dot com.

Your Go-to Resource for Running a MongoDB User Group

- **Tips for Running a Successful User Group**
- **Interested in Starting a User Group?**
- **Logistics**
- Working with Speakers
- **Once you Meet Up...**
- **Stay Engaged**
- ***More Tips and Tricks***

Tips for Running a Successful User Group

Interested in Starting a User Group?

Organizing a user group is a fantastic way to meet and learn from other MongoDB fans in your local community. Interested in starting up a user group in your city? Submit a proposal!

Logistics

Use Meetup.com: This one may be obvious but important to emphasize. Not only does meetup have a lot of great tools for event organizers, but they do a really good job of making it easy for meetup members to find relevant groups. Make sure to tag your group and include a description with keywords so that your group appears in meetup searches!

Consistency is important: It's important to establish a routine early on. If you consistently meet on, the second Tuesday of every month, your members will come to expect the meetup. The first few meetings of any user group will be small, but at every meetup, new members will join your group. So meeting at least on a monthly basis is very important. We have all the NY MUG meetups listed far in advance, even if we don't have a speaker lined up. This also makes your life easier when you are approaching speakers and hosts. It's much easier to ask a speaker "Can you present at the May 19 NY MUG?" than going back and forth coordinating dates. And hosts will appreciate having the events reserved far in advance.

Cross promote: Consider partnering with other technology meetups. This is a great way for communities to learn from one another and gain exposure to new technologies. It could be as simple as ocassionally posting on other meetup lists. For example, when we had a presentation on Scalable Event Analytics with Ruby on Rails and MongoDB, I cross-posted the meetup on the NYC Ruby mailing list and we soon had a dozen new members in the group. I also typically list our events in Startup Digest, LinkedListNYC, Gary's Guide, Charlie O'Donnell's newsletter, Mashable, and more.

Working with Speakers

Create a Speaker Checklist (tip courtesy of Joe Devon)

- Get the full details about your speakers before putting it on the event page
  1. Speaker bio and small photo.
  2. Company "about us" and small logo.
  3. Title of Talk.
  4. Abstract of Talk.
  5. Social Media / Corp homepage / etc... links.
  6. Phone numbers (to coordinate the day of the event)

Once you Meet Up...

Get great speakers: Make a wish list of speakers, and then just start asking people! After organizing dozens of MongoDB events, I've been amazed at how willing people are to present. Most of the time, it's just a matter of asking. And if the person says no, ask them to refer someone

else.

Get Clever About Getting Great Speakers: Are the speakers on your wish list 3,000 miles away from you? That's okay. We live in a wired world. Use Skype or WebX to bring your speakers to you. You can do screen shares to see demos, and provide your meetup members with a great learning experience.

Host Lightning Talks: Sometimes your user group members will not have the chance to pull together a full 30 minute presentation on a product or feature, but some of them may be interested in giving a 5-10 minute lightning talk on something they've designed, an issue they're having with MongoDB, or their dream project. Offer the opportunity when you send out the Meetup invite. If you don't get any submissions, don't worry. Deliver the first lightning talk at the event. Someone might step up to the plate.

Start a Study Group: Does your meetup have a lot of inexperienced MongoDB Users? Start a study group and learn together. Learning in a group is a great experience, and study groups are very common in different developer communities--particularly Ruby on Rails and Node.js

Host a Helpathon: The NYC On Rails Meetup is famous for innovating on the Hackathon to produce the Helpathon , a four hour session for helping one another get through programming hurdles.

Start a Book Club: If your group is at a loss for topics to discuss you could start a reading group. This is similar to a study group and can be a great way to create a more tightly knit User Group for the future.

Raffle off prizes: Prizes are an excellent way to get people to come to your meetup. An easy and free way to get great prizes is to join the O'Reilly User Group program. They will send you free books to give out at your group, as well as discounts on upcoming conferences.

Host a Coding Contest: Google is planning a coding competition for tickets to Google I/O, their largest developer contest. Why not do the same before your first user group? Present a small problem to your members and ask them to bring their files the day of to show off their solution. You can then have judges or the audience choose the winning solution(s).

Stay Engaged

Social media: Consider creating a twitter handle or hashtag for the group. Ask the presenter to tweet or blog about the event, and ask the key members of the group to do the same. Post the hashtag or twitter handle at the event so that members know to use it.

Continue the discussion after the meetup: You can easily record videos and take photos at sessions to share information after the meetup. Encourage the presenter to send a message the group with the slides to start a conversation among the entire meetup.

Send out Group Polls: Get a better understanding of what your user group wants to hear by sending out polls. If you use Meetup.com, you can try out the Polls feature. Otherwise, use SurveyMonkey.

More Tips and Tricks

Agenda Tips for User Groups by Nathen Harvey , organizer of the MongoDC User Group, 2011 Community Champion

Tips and Tricks for Running a Successful Tech Meetup by Meghan Gill , Director of Community Marketing at 10gen

How to Run a Successful Tech Meetup by Chris Westin , Software Engineer at 10gen

Presentation tips for speaking at a MongoDB User Group

Presentation Tips for MongoDB User Groups

Presenting at a MongoDB User Group (MUG) is a great way to share your experiences working with MongoDB, drive adoption of your product and create a closer connection between your company's services and MongoDB. If you're new to public speaking, it's a great place to get presentation experience or test out a talk for a future conference. Here are some tips for giving presentations at a user group.

Have a suggestion that's not listed here? Email our community team with your ideas.

Content: What Should I Talk About At a User Group?

- Teach them something: MongoDB User Group members have different levels of experience with MongoDB but everyone has the same goal: they want to learn more. Offering your insights, tips and tricks often turns into a great presentation with a lot of discussion
    - Many people in our user groups are new to MongoDB, while others are vetrans and have years of experience under their belt but are interested in expanding their knowledge base. It's sometimes a good idea to ask the user group organizer about the experience level of their members so you can cater your presentation their interests.
- Some great topics for a MongoDB User Group (with some examples)
    - MongoDB in the Cloud
        - Growing MongoDB on Amazon Web Services
        - MongoDB on AWS
    - Monitoring Your MongoDB Deployment
    - How I use MongoDB in Production
        - Krossover + MongoDB
        - Fiesta.cc + MongoDB
        - Inside Wordnik's Architecture

        - MongoDB in a Specific Language Library
            - MongoDB and PHP
            - Getting Started with MongoDB and Scala
        - MongoDB and Big Data
            - SQL, NoSQL and Big data Architecture , by Venu Anuganti
            - MongoDB on the JVM
- *Want to get your feet wet?* Offer to give a 5-10 minute lightening talk or demo of a MongoDB tool to build up your presentation skills.

Slides

- Be precise in your slides - Most user group organizers make slides available online after the user group, so make sure they are correct and to the point.
- Be visual - a clever picture, easy to read chart, or even simple bullet points convey clear information.
- Use Big Font! Some meeting rooms are large and it is always hard to see from the back of the room
- Avoid including full text such as sentences and paragraphs in your slides. This makes it difficult for people to pay attention to your thoughts and expertise. You should be the center of the presentation, the slides should be there to guide you.
- Know your audience -The audience is very technical, and generally knowledgeable about MongoDB and related technologies. They are interested in hearing your insights and experience so they can become better MongoDB developers.
- Some great resources to help you build well-designed slides:
    - Duarte have great insight into presentation design and storytelling. Check out their blog for weekly tips and their Tools for Revolutionaries with free video guides to reworking your presentations.

Logistics

- Ask the organizer what time you should arrive (also be courteous and arrive on time)
- Make sure your computer or presentation will be compatible with the presentation setup available at the meetup group. Ask the organizer to make sure you have the correct connecting equipment. If you need a different adaptor for your laptop, please try to bring it with you.
- Send you presentations to the organizer so they can distribute it with the group. It's great to share this information
- Internet connectivity: wifi can be tedious in some places. To avoid any painful situations with wifi, try to make any demos in your presentations available offline.
- Invite your friends! It's always fun to bring along friends, family and coworkers to a user group--but be sure to check with the organizer to ensure the meeting location has enough capacity.

MongoDB Archives and Articles

See also the User Feedback page for community presentations, blog posts, and more.

Planet MongoDB

For the most up-to-date news, visit Planet MongoDB, the aggregator of the best community blog posts on MongoDB.

Best of the MongoDB Blog

- What is the Right Data Model? - (for non-relational databases)
- Why Schemaless is Good
- The Importance of Predictability of Performance
- Capped Collections - one of MongoDB's coolest features
- Using MongoDB for Real-time Analytics
- Using MongoDB for Logging
- mongo, the MongoDB Shell
- Other great posts
- MongoDB 2.2 Released

Best Blogs from the MongoDB Community

- Using MongoDB as a queue on OpenShift with a Twilio application
- Working with MongoDB and Powershell
- Stability in the Midst of Chaos, Jon Hoffman, foursquare
- MongoDB Schema Design at BusyConf , Ryan McGeary, BusyConf.
- How this Website Uses MongoDB , Ian White, former CTO, Business Insider.
- Lessons Learnt from using MongoDB with High Volume Data by David Henderson of Triggered Messaging.
- Forward Intel Uses MongoDB for "casual" analytics
- Mobile Geolocation App in 30 Minutes on OpenShift with MongoDB
- Creating a REST API in Python using Bottle and MongoDB

Articles / Key Doc Pages

- On Atomic Operations
- Reaching into Objects - how to do sophisticated query operations on nested JSON-style objects
- Schema Design

- Full Text Search in Mongo
- MongoDB Production Deployments

Presentations

- Presentations from MongoDB Conferences
- MongoDB for Rubyists (February 2010 Chicago Ruby Meetup)
- Introduction to MongoDB (FOSDEM February 2010)
- NY MySql Meetup - NoSQL, Scaling, MongoDB
- Teach Me To Code - Introduction to MongoDB
- DCVIE

Benchmarks

MongoDB does not publish any official benchmarks.

We recommend running application performance tests on **your** application's work-load to find bottleneck and for performance tuning.

See Also

- JS Benchmarking Harness

FAQ

Redirection Notice
This page should redirect to http://docs.mongodb.org/manual/faq/fundamentals/.

Misc

Demo App in Python

From an Interop 2009 presentation

Code: http://github.com/mdirolf/simple-messaging-service/tree/master

MongoDB, CouchDB, MySQL Compare Grid

pending...

	CouchDB	MongoDB	MySQL
Data Model	Document-Oriented (JSON)	Document-Oriented (BSON)	Relational
Data Types	string,number,boolean,array,object	string, int, double, boolean, date, bytearray, object, array, others	link
Large Objects (Files)	Yes (attachments)	Yes (GridFS)	Blobs
Horizontal partitioning scheme	CouchDB Lounge	Auto-sharding	Partitioning
Replication	Master-master (with developer supplied conflict resolution)	Master-slave and replica sets	Master-slave, multi-master, and circular replication
Object(row) Storage	One large repository	Collection-based	Table-based

Query Method	Map/reduce of javascript functions to lazily build an index per query	Dynamic; object-based query language	Dynamic; SQL
Secondary Indexes	Yes	Yes	Yes
Atomicity	Single document	Single document	Yes - advanced
Interface	REST	Native drivers ; REST add-on	Native drivers
Server-side batch data manipulation	?	Map/Reduce, server-side javascript	Yes (SQL)
Written in	Erlang	C++	C++
Concurrency Control	MVCC	Update in Place	
Geospatial Indexes	GeoCouch	Yes	Spatial extensions
Distributed Consistency Model	Eventually consistent (master-master replication with versioning and version reconciliation)	Strong consistency. Eventually consistent reads from secondaries are available.	Strong consistency. Eventually consistent reads from secondaries are available.

See Also

- Comparing Mongo DB and Couch DB

nutshell

```
{
    "_id" : ObjectId("5081c97c7833857c5588f336"),
    "name" : "mongo",
    "type" : "db",
    "doc_links" : {
        "installation" : "http://www.mongodb.org/display/DOCS/Quickstart",
        "tutorial"     : "http://www.mongodb.org/display/DOCS/Tutorial",
        "reference"    : "http://docs.mongodb.org/manual/contents/"
    },
    versions : [
        { "v" : "2.0.0", "released" : ISODate("2011-09-11T16:22:17Z"), "stable" : true },
        { "v" : "2.0.1", "released" : ISODate("2011-10-22T03:06:14Z"), "stable" : true },
        { "v" : "2.1.0", "released" : ISODate("2012-02-03T17:54:14Z"), "stable" : false },
        { "v" : "2.2.0", "released" : ISODate("2012-09-24T17:38:56Z"), "stable" : true },
    ],
    "features" : [
    ],
    md5 : BinData(5,"nhB9nTcrtoJr2B01QqQZ1g==")


}
```

v2.2

The v2.2 release is now available. In addition to hundreds of smaller improvements, the release includes these major features:

- the new aggregation framework which makes reporting-type queries much easier;
- large improvements in mongod internal concurrency (no more global lock);
- tag-aware sharding / data center awareness features
- TTL collections

You can download v2.2 here.

- v2.2 details online conference
    - Video

    - Slides
- Full release notes here.

Comparing Mongo DB and Couch DB

We are getting a lot of questions "how are mongo db and couch different?" It's a good question: both are document-oriented databases with schemaless JSON-style object data storage. Both products have their place -- we are big believers that databases are specializing and "one size fits all" no longer applies.

We are not CouchDB gurus so please let us know in the forums if we have something wrong.

MVCC

One big difference is that CouchDB is MVCC based, and MongoDB is more of a traditional update-in-place store. MVCC is very good for certain classes of problems: problems which need intense versioning; problems with offline databases that resync later; problems where you want a large amount of master-master replication happening. Along with MVCC comes some work too: first, the database must be compacted periodically, if there are many updates. Second, when conflicts occur on transactions, they must be handled by the programmer manually (unless the db also does conventional locking -- although then master-master replication is likely lost).

MongoDB updates an object in-place when possible. Problems requiring high update rates of objects are a great fit; compaction is not necessary. Mongo's replication works great but, without the MVCC model, it is more oriented towards master/slave and auto failover configurations than to complex master-master setups. With MongoDB you should see high write performance, especially for updates.

Horizontal Scalability

One fundamental difference is that a number of Couch users use replication as a way to scale. With Mongo, we tend to think of replication as a way to gain reliability/failover rather than scalability. Mongo uses (auto) sharding as our path to scalabity (sharding is GA as of 1.6). In this sense MongoDB is more like Google BigTable. (We hear that Couch might one day add partitioning too.)

Query Expression

Couch uses a clever index building scheme to generate indexes which support particular queries. There is an elegance to the approach, although one must predeclare these structures for each query one wants to execute. One can think of them as materialized views.

Mongo uses traditional dynamic queries. As with, say, MySQL, we can do queries where an index does not exist, or where an index is helpful but only partially so. Mongo includes a query optimizer which makes these determinations. We find this is very nice for inspecting the data administratively, and this method is also good when we *don't* want an index: such as insert-intensive collections. When an index corresponds perfectly to the query, the Couch and Mongo approaches are then conceptually similar. We find expressing queries as JSON-style objects in MongoDB to be quick and painless though.

Update Aug2011: Couch is adding a new query language "UNQL".

Atomicity

Both MongoDB and CouchDB support concurrent modifications of single documents. Both forego complex transactions involving large numbers of objects.

Durability

CouchDB is a "crash-only" design where the db can terminate at any time and remain consistent.

Previous versions of MongoDB used a storage engine that would require a repairDatabase() operation when starting up after a hard crash (similar to MySQL's MyISAM). Version 1.7.5 and higher offer durability via journaling; specify the `--journal` command line option

Map Reduce

Both CouchDB and MongoDB support map/reduce operations. For CouchDB map/reduce is inherent to the building of all views. With MongoDB, map/reduce is only for data processing jobs but not for traditional queries.

Javascript

Both CouchDB and MongoDB make use of Javascript. CouchDB uses Javascript extensively including in the building of views .

MongoDB supports the use of Javascript but more as an adjunct. In MongoDB, query expressions are typically expressed as JSON-style query objects; however one may also specify a javascript expression as part of the query. MongoDB also supports running arbitrary javascript functions server-side and uses javascript for map/reduce operations.

REST

Couch uses REST as its interface to the database. With its focus on performance, MongoDB relies on language-specific database drivers for access to the database over a custom binary protocol. Of course, one could add a REST interface atop an existing MongoDB driver at any time -- that would be a very nice community project. Some early stage REST implementations exist for MongoDB.

Performance

Philosophically, Mongo is very oriented toward performance, at the expense of features that would impede performance. We see Mongo DB being useful for many problems where databases have not been used in the past because databases are too "heavy". Features that give MongoDB good performance are:

- client driver per language: native socket protocol for client/server interface (not REST)
- use of memory mapped files for data storage
- collection-oriented storage (objects from the same collection are stored contiguously)
- update-in-place (not MVCC)
- written in C++

Use Cases

It may be helpful to look at some particular problems and consider how we could solve them.

- if we were building Lotus Notes, we would use Couch as its programmer versioning reconciliation/MVCC model fits perfectly. Any problem where data is offline for hours then back online would fit this. In general, if we need several eventually consistent master-master replica databases, geographically distributed, often offline, we would use Couch.
- mobile
  - Couch is better as a mobile embedded database on phones, primarily because of its online/offine replication/sync capabilities.
  - we like Mongo server-side; one reason is its geospatial indexes.
- if we had very high performance requirements we would use Mongo. For example, web site user profile object storage and caching of data from other sources.
- for a problem with very high update rates, we would use Mongo as it is good at that because of its "update-in-place" design. For example see updating real time analytics counters
- in contrast to the above, couch is better when lots of snapshotting is a requirement because of its MVCC design.

Generally, we find MongoDB to be a very good fit for building web infrastructure.

Licensing

- Database:
  - Free Software Foundation's GNU AGPL v3.0.
  - Commercial licenses are also available from 10gen, including free evaluation licenses.
- Drivers:
  - mongodb.org supported drivers: Apache License v2.0.
  - Third parties have created drivers too; licenses will vary there.
- Documentation: Creative Commons.

The goal of the server license is to require that enhancements to MongoDB be released to the community. Traditional GPL often does not achieve this anymore as a huge amount of software runs in the cloud. For example, Google has no obligation to release their improvements to the MySQL kernel – if they do they are being nice.

To make the above practical, *we promise that your client application which uses the database is a separate work*. To facilitate this, the mongodb.org supported drivers (the part you link with your application) are released under Apache license, which is copyleft free. Note: if you would like a signed letter asserting the above promise please contact us.

If the above isn't enough to satisfy your organization's vast legal department (some will not approve GPL in any form), please contact us – commercial licenses are available including free evaluation licenses. We will try hard to make the situation work for everyone.

International Docs

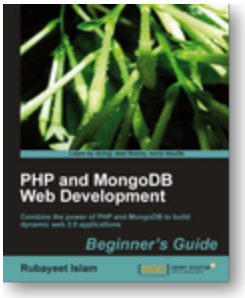
Most documentation for MongoDB is currently written in English. We are looking for volunteers to contribute documentation in other languages. If you're interested in contributing to documentation in another language please email "docs at 10gen.com".

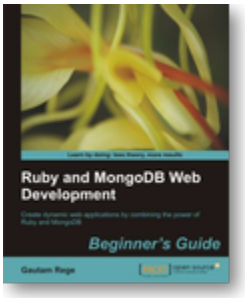
Language Homepages

- Deutsch
- Español
- Français
-

- Italiano
-
- Português
-
- Svenska
-
-
-

Books





You can download samples at 10gen.com/books.

MongoDB: The Definitive Guide
Kristina Chodorow and Mike Dirolf

The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing
Peter Membrey

Scaling MongoDB
Kristina Chodorow

MongoDB in Action
Kyle Banker

MongoDB and Python: Patterns and processes for the popular document-oriented database
Niall O'Higgins

MongoDB and PHP
Steve Francia

MongoDB for Web Development
Mitch Pirtle

MongoDB: Sag Ja zu NoSQL
Marc Boeker

The Little MongoDB Book
Karl Seguin
Free ebook

50 Tips and Tricks for MongoDB Developers
Kristina Chodorow

MongoDB with Python and Ming
Rick Copeland

PHP and MongoDB Web Development Beginner's Guide book and eBook
Rubayeet Islam

Ruby and MongoDB Web Development Beginner's Guide
Gautam Rege

Slides and Video

Upcoming Events

- http://www.10gen.com/events

Presentations from past conferences

- http://www.10gen.com/presentations

Some past webinars

- Analytics MongoDB Analytics for Online Advertising at Magnetic
- DevOps Data Center Awareness with MongoDB
- DevOps Deployment Best Practices
- DevOps Diagnostics and Performance Tuning
- DevOps Keeping your data safe: replication, backup/restore, mongodump
- DevOps MongoDB and Amazon Web Services
- DevOps MongoDB on Amazon EC2
- DevOps Scaling with MongoDB
- DevOps Tips for a Successful, Scalable MongoDB Deployment
- Geo Geospatial Indexing with MongoDB
- Hadoop Hadoop plugin for MongoDB: The Elephant in the Room
- Indexing Indexing Matters: A MongoDB Optimization Primer
- Indexing Indexing With MongoDB
- Intro Developing Applications with MongoDB: A Quickstart Tutorial
- Intro Introduction to MongoDB
- Java What's New in Java Driver Version 2.5
- Java Zero to MongoDB - Introduction for MongoDB with Java
- Journaling Journaling in MongoDB version 1.8
- Node.js Node.js and MongoDB, a Panel Discussion
- Ruby MongoDB and Ruby
- Scala MongoDB and Scala
- Schema Schema Design with MongoDB
- Use Cases How MongoDB helps Visibiz tackle social CRM
- Use Cases Humongous Data at Server Density: Approaching 1 Billion Documents in MongoDB
- Hybrid Applications with MongoDB and RDBMS
- v1.8 What's new in MongoDB v1.8.0
- v2.0 What You Need to Know About MongoDB 2.0

Alerts

This page lists critical alerts and advisories for MongoDB. This page is a work in progress and will be enhanced over time.

See http://jira.mongodb.org/ for a comprehensive list of bugs and feature requests.

Data Integrity Related

- Documents may be missing on a replication secondary after initial sync if a high number of updates occur during the sync that move the document (i.e., the documents are growing).
    - https://jira.mongodb.org/browse/SERVER-3956 Replica set case. Fixed: 1.8.4, 2.0.1
    - https://jira.mongodb.org/browse/SERVER-4270 Master/slave case. Fixed: 1.8.5, 2.0.2

- When stepping down a replica set primary, the primary's connection to clients is not automatically closed in 2.0.1. This means that if the primary steps down while an application is issuing fire-and-forget write, the application won't necessarily fail over until it issues a query. Applications that only issue safe writes are not affected by this. The issue is fixed in 2.0.2. See https://jira.mongodb.org/browse/SERVER-4405 for details.

Security Related

- Limit access of __system when running --auth without replica sets.
    - https://jira.mongodb.org/browse/SERVER-3666 Fixed: 1.8.4

Make sure to subscribe to http://groups.google.com/group/mongodb-announce for important announcements.