mongoDB®

MongoDB is an open-source, high-performance, document-oriented database.

Documents are JSON-like data structures stored in a format called BSON (bsonspec.org). Documents are stored in **collections**, each of which resides in its own **database**. Collections can be thought of as the equivalent of a table in an RDBMS. There are **no fixed schemas** in MongoDB, so documents with different "shapes" can be stored in the same collection.

MongoDB features **full index support** (including secondary and compound indexes); indexes are specified per collection. There is a rich, **document-based query language** (see reverse) that leverages these indexes. MongoDB also provides sophisticated atomic update modifiers (see reverse) to keep code contention-free.

Clustered setups are supported, including easy replication for high availability, as well as auto-sharding for write-scaling and large data-set sizes.

mongoDB® Resources

Learn	**In-browser tutorial** try.mongodb.org **Downloads** mongodb.org/downloads **Presentations** 10gen.com/presentations **Getting Started, Projects, Collections** mongly.com
Support	**User forum** groups.google.com/group/mongodb-user **IRC** irc://irc.freenode.net/#mongodb **Bug tracking** jira.mongodb.org **Commercial Support** 10gen.com/support
Community	**MongoDB User Groups (MUGs)** 10gen.com/user-groups **MongoDB Conferences** 10gen.com/events
Social	**Twitter** follow @mongodb and @10gen **Facebook** www.facebook.com/mongodb **LinkedIn** linkd.in/mongodb_group
Careers	**MongoDB jobs board** jobs.mongodb.org
Contact	**Contact 10gen** 10gen.com/contact

Queries and What They Match

`{a: 10}`	Docs where **a** is 10, or an array containing the value 10.
`{a: 10, b: "hello"}`	Docs where **a** is 10 and b is "hello."
`{a: {$gt: 10}}`	Docs where **a** is greater than 10. Also `$lt` (<), `$gte` (>=), `$lte` (<=), and `$ne` (!=).
`{a: {$in: [10, "hello"]}}`	Docs where **a** is either 10 or "hello."
`{a: {$all: [10, "hello"]}}`	Docs where **a** is an array containing both 10 and "**hello**".
`{"a.b": 10}`	Docs where **a** is an embedded document with **b** equal to 10.
`{a: {$elemMatch: {b: 1, c: 2}}}`	Docs where **a** is an array containing a single item with both **b** equal to 1 and **c** equal to 2.
`{$or: [{a: 1}, {b: 2}]}`	Docs where **a** is 1 or **b** is 2.
`db.foo.find({a: /^m/})`	Docs where **a** begins with the letter "m".

The following queries cannot use indexes as of MongoDB v2.0. These query forms should normally be accompanied by at least one other query term which *does* use an index:

`{a: {$nin: [10, "hello"]}}`	Docs where **a** is anything but 10 or "hello."
`{a: {$mod: [10, 1]}}`	Docs where **a** mod 10 is 1.
`{a: {$size: 3}}`	Docs where **a** is an array with exactly 3 elements.
`{a: {$exists: true}}`	Docs containing an **a** field.
`{a: {$type: 2}}`	Docs where **a** is a string (see bsonspec.org for more types).
`{a: /foo.*bar/}`	Docs where **a** matches the regular expression "**foo.*bar**".
`{a: {$not: {$type: 2}}}`	Docs where **a** is not a string. **$not** negates any of the other query operators.

Update Modifiers

`{$inc: {a: 2}}`	Increment **a** by 2.
`{$set: {a: 5}}`	Set **a** to the value 5.
`{$unset: {a: 1}}`	Delete the **a** key.
`{$push: {a: 1}}`	Append the value 1 to the array **a**.
`{$pushAll: {a: [1, 2]}}`	Append both 1 and 2 to the array **a**.
`{$addToSet: {a: 1}}`	Append the value 1 to the array **a** (if it doesn't already exist).
`{$addToSet: {a: {$each: [1, 2]}}}`	Append both 1 and 2 to the array **a** (if they don't already exist).
`{$pop: {a: 1}}`	Remove the last element from the array **a**.
`{$pop: {a: -1}}`	Remove the first element from the array **a**.
`{$pull: {a: 5}}`	Remove all occurrences of 5 from the array **a**.
`{$pullAll: {a: [5, 6]}}`	Remove all occurrences of 5 or 6 from the array **a**.

mongoDB® Commands

What are Commands?

Commands are special MongoDB operations. Most MongoDB client libraries provide a helper for running commands. For example, here's how to run the **dropDatabase** command from the shell:

```
> db.runCommand({dropDatabase:1});
```

Some commands are admin-only, and must be run on the admin database. In the list below, those commands are marked with an asterisk (*).

Available Commands

To get a list of all available commands, run **db.listCommands()** from the MongoDB shell.

Some of the most frequently used commands are listed below:

***{buildinfo: 1}**	Get version number and other build information about the MongoDB server.
{collStats: coll[, scale: 1]}	Get stats about collection **coll**. Sizes are in bytes by default but may be scaled by the provided scaling factor.
{count: coll[, query: query]}	Get the number of documents in collection **coll** that match the (optional) specified query.
{dbStats: 1}	Get stats about the current database.
{distinct: coll, key: key[, query: query]}	Get a list of distinct values for **key** in **coll** for all documents that match the (optional) specified query.
{drop: coll}	Delete collection **coll**.
{dropDatabase: 1}	Drop the current database.
{dropIndexes: coll, index: {y: 1}}	Drop the index with key pattern **{y:1}** in collection **coll**. Use **index: *** to drop all indexes in **coll**.
{getLastError: ...}	Get the status of the last operation on this connection.
{isMaster: 1}	Check if this server is a primary/master server.
{listCommands: 1}	Get a list of available commands.

Commands: Available Commands (cont'd)

*`{listDatabases: 1}`	Get a list of databases on this server.
`{profile: n}`	Set the database profiler to a given profiling level (0=disabled, 1=slow queries, 2=all queries).
`{reIndex: coll}`	Re-index collection **`coll`**.
*`{renameCollection: a, to: b}`	Rename collection **a** to **b**.
`{repairDatabase: 1}`	Repair and compact the current database.
`{replSetGetStatus: 1}`	Get the status of a replica set.
`{serverStatus: 1}`	Get a list of administrative statistics about the server.
*`{shutdown: 1}`	Shut down the MongoDB server.
*`{top: 1}`	Get a breakdown of usage by collection.
`{validate: ns}`	Validate the documents in the specified namespace (collection or index).

mongoDB® Indexing

Index Creation

`db.coll.ensureIndex(key_pattern, options)`
Create an index on collection **`coll`** with the given key pattern and options.

Indexing Key Patterns with Sample Queries

`{username: 1}` Ex: `db.users.find({username: 'smith'});`	Simple index.
`{last_name: 1, last_login: -1}` Ex: `db.users.find({last_name: 'jones'}}).sort({last_login: -1})`	Compound index with **name** ascending and **`last_login`** descending. Note that key order on compound indexes matters.
`{coord: '2d'}` Ex: `db.places.find({coord: {$near: [50, 50]}})`	Geospatial index, where **coord** is a coordinate (x,y) where -180 < x, y < 180. Note that **$near** queries return the closest points to the given coordinate.

Index Creation Options

`{unique: true}`	Create a unique index. To check insertion failures, you must use your driver's **safe mode**.
`{dropDups: true}`	Use with the **unique** option. Drop documents with duplicate values for the given key pattern on index creation.
`{background: true}`	Create this index in the background; useful when you need to minimize index creation performance impact.
`{sparse: true}`	Create entries in the index only for documents having the index key.
`{name: 'foo'}`	Specify a custom name for this index. If not specified, the name will be derived from the key pattern.

Examples

`db.users.ensureIndex({username: 1}, {unique: true})`	Create a unique index on **`username`**.
`db.products.ensureIndex({category: 1, price: -1}, {background: true})`	Create a compound index on **`category`** and **`price`** and build it in the background.
`db.places.ensureIndex({loc: '2d'})`	Create a geospatial index on **`loc`**.

Administration

`db.users.getIndexes()`	Get a list of all indexes on the **`users`** collection.
`db.system.indexes.find()`	Directly query the collection containing index definitions for this database.
`db.users.totalIndexSize()`	Get the number of bytes allocated by indexes for the **`users`** collection.
`db.stats()`	Get database stats, including total index size for current database.
`db.users.reIndex()`	Rebuild all indexes on this collection.
`db.users.dropIndex({x: 1, y: -1})`	Drop the index with key pattern **`{x: 1, y: -1}`**. Use **`db.users.dropIndexes()`** to drop all indexes on the **`users`** collection.

Tips

You can use a compound index on **`{username: 1, date: 1}`** for the following queries:

```
db.users.find({username: "Jones"});
db.users.find({username: /^Jones/});
db.users.find({username: "Jones", date: "2010-12-01"});
db.users.find({username: "Jones"}).sort({date: -1});
db.users.find({}).sort({username: 1, date: 1}).limit(100);
```

Note that with this index, a separate single-key index on **`{username: 1}`** is unnecessary.

mongoDB® Replication

About Replication

MongoDB enables sophisticated replication, with automated failover, as a configuration of servers known as a **replica set**. A replica set consists of a single master node, called the **primary**, and one or more slave nodes, called **secondaries**. If the primary becomes unreachable, the replica set will attempt to elect one of the secondary nodes as the new primary. A **majority** of servers must be able to reach the primary, both to elect it and to keep it a primary.

If you have only two servers and you want one server to be elected primary if the other goes offline, then you must also run an **arbiter** node on a third server. Arbiter nodes do not store a replica of the set's data; they exist solely to participate in elections. Therefore, these nodes can easily reside on an application server.

What is a Majority?

If your set consists of...

1 server, 1 server is a majority.

2 servers, 2 servers are a majority.

3 servers, 2 servers are a majority.

4 servers, 3 servers are a majority.

...

Setup

To initialize a three-node replica set with one arbiter, start three mongod instances, each using the **--replSet** flag followed by a name for the replica set. For example:

```
mongod --replSet cluster-foo
```

Next, connect to one of the **mongod**s and run the following:

```
rs.initiate()

rs.add("host2:27017")

rs.add("host3:27017", {arbiterOnly: true})
```

Replication: Setup (cont'd)

The second argument to **`rs.add()`** may also contain the following:

`priority: n`	Members will be elected primary in order of priority, if possible. **n=0** means this member will never be a primary.
`slaveDelay: n`	This member will always be a secondary and will lag **n** seconds behind the master.
`arbiterOnly: true`	This member will be an arbiter.
`hidden: true`	Do not show this member in **`isMaster`** output. Use this option to hide this member from clients.
`tags: [...]`	Member location description. See http://www.mongodb.org/display/DOCS/Data+Center+Awareness.

Shell Helpers

`rs.initiate()`	Create a new replica set with one member.
`rs.add("host:port")`	Add a member.
`rs.remove("host:port")`	Remove a member.

Administration

`rs.status()`	See the status of each member.
`rs.conf()`	See the current set configuration.
`rs.reconfig(newConfig)`	Change the set configuration.
`rs.isMaster()`	See which member is primary.
`rs.stepDown(n)`	Force the primary to become a secondary for **n** seconds.
`rs.freeze(n)`	Prevent a secondary from becoming the primary for **n** seconds (**n=0** means unfreeze).

Tips

- Run at least one member with journaling enabled to ensure that you always have a clean copy of your data available.
- Start replica set members with the **`--rest`** option to enable the web interface.

What is MMS?

MongoDB Monitoring Service (MMS) is a free, cloud-based monitoring and alerting solution for MongoDB deployments that offers the ability to proactively monitor your production system. MMS allows you to track performance, utilization, availability and response times all from a centralized dashboard. The service also supports monitoring advanced deployments such as those that use replication, sharding or that are hosted across multiple data centers. MMS requires minimal setup and configuration and within minutes your devops and sysadmin teams can manage and optimize your MongoDB deployment.

Register at mms.10gen.com.

How MMS Works

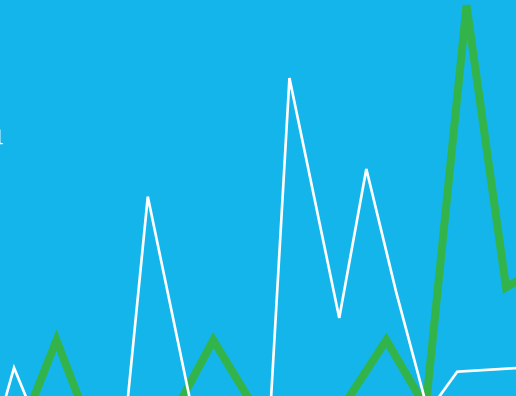
MMS is enabled by installing a lightweight agent in the server environment. From there, the MMS agent automatically discovers all MongoDB nodes in the cluster and then reports on metrics such as memory usage, ops per second, open connections, CPU load, and I/O activity. The web interface displays the status of each MongoDB instance, as well as historical graphs for all metrics. You can create custom dashboards, perform side-by-side data comparisons, and review cluster behavior at a wide variety of timescales.

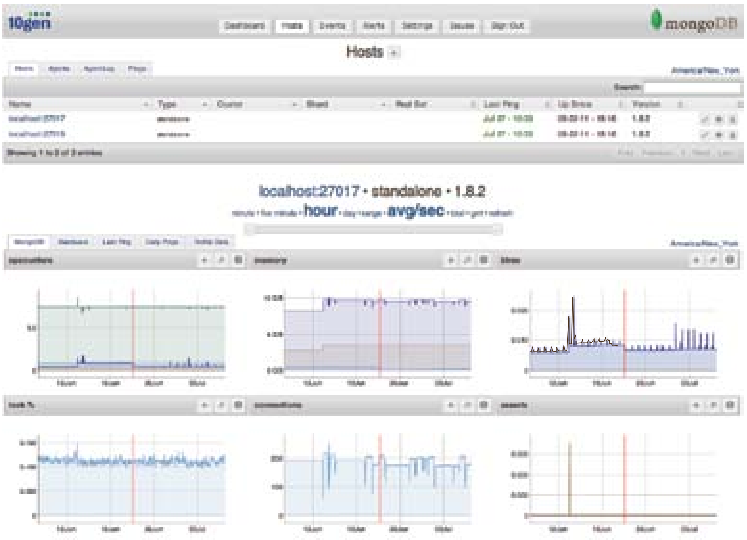
Installation

MMS can be used with all varieties of MongoDB deployments, hosted on physical hardware or in the cloud. Installing the monitoring agent is quick and simple, and meaningful results are available through the web interface in minutes. A monitoring solution you're sure to find useful, you can sign up for MMS now at mms.10gen.com.

Security

All metrics are transmitted by the agent to the MMS servers over SSL (128-bit encryption). The monitoring agent is a Python script, and you're free to examine its source code.
By default, only server metrics are recorded, and collection of hardware and application data is entirely optional. Naturally, the web interface is secured over SSL, and extensive data access controls and audits are in place to ensure the safety of your data.





"MMS adds enormous value to 10gen's support capabilities. After adding MMS to our cluster, 10gen's engineers detected an anomaly in our production deployment and proactively reached out to us to fix the problem before it became a production incident. We didn't even have to file a ticket. This gives us great confidence in 10gen's ability to support us going forward and demonstrates 10gen's commitment in partnering with Monster to ensure success."
– *Ray Howell, Vice President of Architecture at Monster.com*

"MMS helps us manage the complexity inherent in large scale online gaming. Behind every online game is a complex distributed system that's much more complicated than a standalone game. 10gen has already figured out what we need to monitor and made it really easy for us to be confident with our infrastructure."
– *Matt Levy, Architect at Playfirst*

mongoDB®

mongoDB.org

10gen

10gen.com