### [Terminology](https://docs.mongodb.com/manual/reference/glossary/index.html" \l "glossary)

As you start working with MongoDB, you will notice various terms used in the documentation. Here is a quick table of equivalences between MongoDB and traditional relational database management systems (RDBMS) terms.

|  |  |
| --- | --- |
| **MongoDB** | **RDBMS** |
| *database* | database |
| *collection* | table |
| *document* | row |
| *field* | column |

You will also encounter concepts have no direct RDBMS equivalents, summarized here:

|  |  |
| --- | --- |
| **Concept** | **Notes** |
| *mongod* | MongoDB server daemon. |
| *mongos* | Routes queries across a sharded cluster. Otherwise, behaves as a mongod instance. |
| *mongo* | A JavaScript based shell used to manage *mongod* or *mongos* instances. |
| *replica set* | A group of servers with identical copies of the database providing high availability. |
| *sharded cluster* | A group of servers with logically divided fragments of the same collection, allowing for horizontal scalability. |

### [Installation and Configuration](https://docs.mongodb.com/manual/installation/" \l "install-mongodb)

The steps needed to install MongoDB 4.x on your local server are extremely well documented [here](https://docs.mongodb.com/manual/installation/" \l "install-mongodb). Accordingly, in this guide we add a few brief notes on installation before moving on to DevOps considerations.

***DEB-Based Linux***

Debian based Linux distributions include Ubuntu and Debian itself. It's important to note that the "official" Debian package repository generally does not have the latest version of MongoDB. Accordingly you will need to add to your *sources* list, as explained [here](https://docs.mongodb.com/manual/tutorial/install-mongodb-on-debian/" \l "create-a-etc-apt-sources-list-d-mongodb-org-4-2-list-file-for-mongodb) for Debian Linux, and [here](https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/" \l "create-a-list-file-for-mongodb) for Ubuntu.

***RPM-Based Linux***

RPM (Redhat Package Manager) based Linux distributions include Redhat, Fedora and CentOS, among others. As with DEB-based Linux distributions, it's important to update the sources list for the *yum* package manager, as explained [here](https://docs.mongodb.com/manual/tutorial/install-mongodb-on-red-hat/" \l "configure-the-package-management-system-yum) on the MongoDB website.

***Mac***

It is highly recommended you install MongoDB using the [Homebrew](https://brew.sh/) package manager. The most up-to-date instructions can be found [here](https://github.com/mongodb/homebrew-brew).

***Windows***

It should come as no surprise that Windows installations use an MSI (Microsoft Installer) package provided by MongoDB. The installation is extremely straightforward and entirely GUI driven. The most difficult part of the installation will be determining where to place the database files (i.e. so you don't immediately fill up the the wrong Windows "drive"), and whether or not to run MongoDB as a service. The Windows installer also installs *[MongoDB Compass](https://docs.mongodb.com/compass/current/" \l "mongodb-compass)*, which is a GUI for the *mongo* shell.

### [Configuring MongoDB](https://docs.mongodb.com/manual/reference/configuration-options/" \l "configuration-file-options)

Although you can start MongoDB server from the command, and supply command line options to enforce specific behavior, it's much more convenient to make this behavior permanent by placing it in a configuration file.

***Location of the Config File***

The following table provides a list of default locations of the MongoDB configuration file.

|  |  |
| --- | --- |
| **OS** | **Config File** |
| Linux | /etc/mongod.conf |
| Windows | <drive>:\<install\_dir> \bin\mongod.cfg |
| Mac | /usr/local/etc/mongod.conf |

***[Config File Directives](https://docs.mongodb.com/manual/reference/configuration-options/" \l "configuration-file-options)***

Config file directives use [YAML](https://yaml.org/) formatting. Each new directive must be on a new line. Use a colon (:) to separate key from value. Nested options are indented by at least one space.

**01** **storage**

The *storage* directive is used to instruct *mongod* where its database files are located. This directive also tells *mongod* which engine to use.   
  
**01A** storage.dbPath

This directive is used to instruct *mongod* where its database files are located. Example:

storage :  
 dbPath : /var/lib/mongodb

**01B** storage.journal

This directive is used to configure *journaling*, which ensures database integrity. Example:

storage :  
 journal :   
 enabled : true  
 commitIntervalMs : 100

**02** **net**

The *net* directive is used to control *mongod* network connectivity. This directive can be used to specify allowed IP addresses, ports, sockets as well as secured TLS communications.

**02A** net.bindIp | bindIpAll

This option indicates on which hostnames or IP address(es) mongod will listen. Example:

net :  
 bindIp : localhost, 192.168.2.20

Alternatively, if you wish to listen on all IP addresses, use this example:

net :  
 bindIp : 0.0.0.0

Or use this example:

net :  
 bindIpAll : true

**02B** net.ipv6

Make sure to set this option to *true* if your network uses IP version 6. Example:

net :  
 ipv6 : true  
 bindIp : fe80::42:feff:fe99:a0ea

**02C** net.unixDomainSocket

This option allows *mongod* to connect via a local "unix" socket. Note that this does not work on a Windows server. Example:

net :  
 unixDomainSocket :   
 enabled : true

pathPrefix :

**03 systemLog**

The *systemLog* directive configures the location and nature of MongoDB server logging

**03A** systemLog.path

Configures directory path and filename for the MongDB log file. This option is required if destination (see next) is set to file. Example:

systemLog :  
 path : /var/log/mongodb/mongod.log

**03B** systemLog.destination

The nature of the MongDB log file. Settings can be either *file* or *sysLog*. Example:

systemLog :  
 path : /var/log/mongodb/mongod.log

**03C** systemLog.verbosity

Configures how much information goes into the log. Values can range from 0 to 5 with level 0 (default) producing the least amount of information. Example:

systemLog :  
 verbosity : 1

### [Mongo Shell Operations](https://docs.mongodb.com/manual/reference/mongo-shell/" \l "mongo-shell-quick-reference)

The *mongo* shell opens an interactive command line prompt that lets you perform operations on a MongoDB database. The shell accepts JavaScript syntax and commands, which means you can define variables and functions in addition to running commands supplied by MongoDB.

**01** Open the shell

The shell can be opened using this syntax:

mongo [<options>] [<db>] [<file>]

**db** specifies the name of the database to open

**file** is a JavaScript file with shell commands to execute

The more important [options](https://docs.mongodb.com/manual/reference/program/mongo/" \l "options) are summarized here:

|  |  |
| --- | --- |
| **Option** | **Notes** |
| --quiet | Takes you into the shell without diagnostic messages |
| --host | Lets you specify a host name or IP address for the server running the mongod instance you wish to manage. |
| --username | If you have configured database user/roles, specify the desired username to use for the connection |
| --password | Password for the associated username |

### [CRUD Operations](https://docs.mongodb.com/manual/crud/" \l "mongodb-crud-operations)

Here is a quick summary of the main create / read / update and delete operations performed from the *mongo* shell. Each command operates upon the currently used database, represented by **db**. You must also specify the name of the collection upon which you wish to operate.

|  |  |
| --- | --- |
| **Alias** | **Notes** |
| db | Defaults to the current database in use. |
| collection | Must be changed to the name of the collection upon which you plan to operate. |
| document | JSON document to insert or update. |
| filter | JSON document with query parameters to be matched. |
| projection | JSON document that identifies fields to include or exclude. |
| options | JSON document with pre-defined option key/value pairs. |

**01** Insert a single document

db.collection.insertOne(<document>)

Use this method to insert a single document into the named collection. Example below inserts a document into the *users* collection:

db.users.insertOne(  
{"first":"Fred","last":"Flintstone"});

**02** Insert many documents

db.collection.insertMany(  
[<document>,<document>,etc.])

Use this method to multiple documents into the named collection. Example below inserts two documents into the *users* collection:

db.users.insertMany([  
{"first":"Barney","last":"Rubble"}

{"first":"Betty","last":"Rubble"});

**03** Delete a single document

db.collection.deleteOne(<filter>)

Use this method to delete the first document to match the filter. Example below delete a single document from the *users* collection:

db.users.deleteOne(  
{"last":"Flintstone"});

**04** Delete many documents

db.collection.deleteMany(  
<filter>)

Use this method to delete any documents matching the filter. Example below delete all documents from the *users* collection where the last name is *Rubble*:

db.users.deleteMany({"last":"Rubble"});

**05** Update a single document

db.collection.updateOne(  
<filter>,<document>,<options>)

Use this method to update the first document to match the filter. Example below updates the event\_date of a single document in the *events* collection matching *key* inserting a new document if the *key* is not found:

db.events.updateOne(  
{"key":"ABCDEF"},  
{$set:{"event\_date":"2020-01-01"},

{"upsert" : true);

**06** Update many documents

db.collection.updateMany(  
<filter>,<document>,<options>)

Use this method to update all document matching the filter. Example below sets the *active* field to 0 where the *event\_date* starts with *2021*:

db.events.updateMany(  
{"event\_date":{"$regex":/^2021/},   
{$set:{"active":0});

**07** Find a single document

db.collection.findOne(  
<filter>,<projection>)

Use this method to find the first document to match the filter. Example below returns a list of event names where the *event\_date* is between June and August 2020:

db.events.findOne(  
{"key":"ABCDEF"},  
{$set:{"event\_date":"2020-01-01},

{"upsert" : true);

**08** Find one or more documents

db.collection.find(  
<filter>,<projection>)

Use this method to find all documents matching the filter. Example below returns a list of event names where the *event\_date* is between June and August 2020:

db.events.find({$and:[  
{"eventDate":{"$lt":"2020-09-01"}},{"eventDate":{"$gte":"2020-06-01"}}]},{"eventName":1});

### [Query Operators](https://docs.mongodb.com/manual/reference/operator/query/" \l "query-selectors)

Query operators produce meaningful expressions and are used to create the *filter* JSON document used in conjunction with read, update and delete operations.

|  |  |
| --- | --- |
| **Operator** | **SQL Equivalent** |
| $eq | == |
| $ne | != |
| $lt, $lte | <, <= |
| $gt, $gte | >, >= |
| $in | IN |
| $regex | LIKE but with more power |

You can combine operators using $and, $or, $not and $nor. (See last find() example above).

### [Update Operators](https://docs.mongodb.com/manual/reference/operator/update/" \l "id1)

Update operators are used in the update document. You need to use these operators to tell MongoDB what operation to perform on fields in documents matching the filter.

|  |  |
| --- | --- |
| **Operator** | **Notes** |
| $set | Overwrites any existing value for that field with new value. If the field does not exist, creates a new field with that value. |
| $unset | Removes named field from document(s) matching the filter. |
| $rename | Renames a field. |
| $push | Adds item to array field |
| $pop | Removes first or last item from array field |

If the field contains an integer value, the $inc (increment) or $dec (decrement) update operators can be used.

### [Aggregation Framework](https://docs.mongodb.com/manual/aggregation/" \l "aggregation-pipeline)

The MongoDB aggregation framework is an extremely powerful mechanism that allows you to configure a *pipeline* into which a series of *stages* can be introduced. The output from one stage flows into the pipe, and becomes the input to the stage that follows.

The framework operates at the collection level. Here is the generic syntax:

db.collection.aggregate([  
{<stage operator>:{<expression>},  
{<stage operator>:{<expression>},etc]);

***[Stage operators](https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/" \l "aggregation-pipeline-stages)*** are summarized here:

|  |  |
| --- | --- |
| **Operator** | **Notes** |
| $match | Forms a filter that allows matching documents into the pipe. |
| $bucket | Creates a series of buffers based on the value of a specified field. The output of each buffer is in the form of a field added to the pipeline. |
| $addFields | Adds new fields to the pipe based on operations involving the value of existing fields. |
| $unwind | Outputs a single document for each element of an embedded array. |
| $facet | Creates a set of nested pipelines upon which multiple operations can be performed. |
| $geoNear | Forms pipeline output based on geo-spatial data (i.e. latitude and longitude). |
| $project | Restricts which fields appear in the pipe. |
| $out | Specifies a collection into which the pipeline results are written. |

In addition to the above, there are a series of stage operators that are similar to SQL:

|  |  |
| --- | --- |
| **Operator** | **SQL Equivalent** |
| $group | GROUP BY |
| $count | COUNT |
| $limit | LIMIT |
| $skip | OFFSET |
| $sort | SORT BY |
| $lookup | LEFT JOIN |

***Expressions*** generally involve *[pipeline operators](https://docs.mongodb.com/manual/reference/operator/aggregation/" \l "aggregation-pipeline-operators)*, summarized here:

|  |  |
| --- | --- |
| **Operator** | **Notes** |
| $add,$subtract, $multiply,$divide | Arithmetic operators |
| $arrayElemAt, $concatArrays, $size, $slice | Array operators |
| $and, $or, $not | Boolean operators |
| $eq, $ne, $lt, $lte, $gt, $gte | Comparison operators |
| $cond | Lets you create a condition and returns one value if true, the other if not |
| $dateFromString, $dayOfMonth, $month, $year, $hour, $minute, $second | Date operators |
| $concat, $trim, $regexMatch, $split, $substr | String operators |
| $convert, $toDate, $toDecimal, $toInt, $toLong, $toString | Conversion operators |
| $avg, $min, $max, $sum | Accumulation operators |

Here are some examples of common aggregation operations:

**01** Produces a list of event names, dates, and hotels held for the year 2021:

db.signups.aggregate([   
{**$match** :  
{"eventInfo.eventDate":/^2021/}},

{**$addFields** : {

"name":"$eventInfo.eventName",

"date":"$eventInfo.eventDate", "hotel":  
"$eventInfo.hotelInfo.hotelName"}},   
{**$project** : {  
"name":1,"date":1,"hotel":1,"\_id":0}},

{**$sort** : {"date":1}}]);

**02** Count of signups by event:

db.signups.aggregate([

{**$addFields**:{  
"name":"$eventInfo.eventName",

"country":"$eventInfo.hotelInfo.location.country",  
"value":1}},

{**$group**:{"\_id":"$name",   
"count":{$sum:"$value"}}},

{**$sort**:{"country":1}}]);

### [PHP MongoDB Extension](https://docs.mongodb.com/ecosystem/drivers/php/" \l "mongodb-php-driver) Installation

It's extremely important to install the correct PHP MongoDB extension. The source code for the correct extension is located here:  
<http://pecl.php.net/package/mongodb>

It is referred to as the *MongoDB Driver for PHP* and is currently maintained. The older driver called the *MongoDB database driver,* was last updated in 2017, and **does not work** with MongoDB version 4.0 and above.

***Linux and Mac servers***

To install the latest MongoDB PHP extension , proceed as follows:

**01** If not already present on your server, install *pecl*:

**01A** On RPM-based systems:

sudo yum install php-pear

**01B** On DEB-based systems:

sudo apt-get install php-pear

**01C** On a Mac the *Homebrew* PHP package includes *pecl*:

brew install php

**02** Use *pecl* to install the extension:

sudo pecl install mongodb

***Windows servers***

To install the latest MongoDB PHP extension , proceed as follows:

**01** Download the correct DLL for your version of Windows and MongoDB from this site:  
http://pecl.php.net/package/mongodb

**02** Enable the extension in your php.ini file:

extension=php\_mongodb.dll

### [PHP MongoDB Library](https://docs.mongodb.com/ecosystem/drivers/php/" \l "mongodb-php-driver) Installation

The MongoDB PHP Library is separate from and dependent upon the PHP MongoDB extension. The library provides high level support for native MongoDB shell methods. Using the library you can first model queries using the *mongo* shell, which can then be directly translated into PHP expressions.

To install the MongoDB PHP Library, proceed as follows:

**01** If not already present on your server, download *composer* from this site:

<https://getcomposer.org/download/>

**02** Use *composer* to install the ibrary:

**02A** Composer as a \*.phar file:

php composer.phar \  
 require mongodb/mongodb

**02B** Composer as an executable:

composer require mongodb/mongodb

### Using MongoDB with PHP

Here are some common CRUD operations performed using the MongoDB PHP extension and library:

**01** Get a connection to MongoDB

Example below creates a connection to a MongoDB instance running on *localhost*:

use MongoDB\Client;

$client = new Client  
('mongodb://localhost:27017');

$db = 'jumpstart';

**02** Insert many documents

Use the *insertMany()* method to insert multiple documents into the named collection. Example below inserts four documents into the *users* collection:

$data = [

['key' => 1, 'first' => 'Fred', 'last' => 'Flintstone'],

['key' => 2, 'first' => 'Wilma', 'last' => 'Flintstone'],

['key' => 3, 'first' => 'Barney','last' => 'Rubble'],

['key' => 4, 'first' => 'Betty', 'last' => 'Rubble']];

$client->$db->  
 users->insertMany($data);

**03** Delete a single document

Use the *deleteOne()* method to delete the first document to match the filter. Example below delete a single document from the *users* collection:

$filter = ['key' => 3];  
$result = $client->$db->  
 users->deleteOne($filter);

if (!$result->isAcknowledged()) {

// do something

} else {

// do something else

}

**04** Update many documents

Use the *updateMany()* method to update all document matching the filter. Example below sets the *active* field to 0 where the *event\_date* starts with *2021*:

$client->$db->events->updateMany(  
["event\_date"=>["$regex"=>"^2021"]],   
["$set"=>["active"=>0]]);

**05** Find documents

Use the *find()* method to find all documents matching the filter. Example below returns a list of event names where the *event\_date* is between June and August 2020:

$client->$db->events->find(

[ '$and'=> [

["eventDate"=>["$lt"=>"2020-09-01"],

"eventDate"=>["$gte"=>"2020-0601"]

]],

[

'projection' => ["eventName"=>1],

'sort' => ["eventDate" => 1]

]);