mongoDB

{ name: mongo, type: DB }

mongoDB

MongoDB

The Complete Developer's Guide

What is MondoDB?

MongoBD is a database which is created by the company who is also called MongoDB. The name stems from the word "humongous". This database is built to store a lot of data but also being able to work with the huge data efficiently. Ultimately, this is a database solution.

There are many database solutions such as mySQL, PostgresSQL, TSQL etc.

MongoDB is most importantly a database server that allows us to run different databases on it for example a Shop database. Within the database we would have different collections such as a Users collection or a Orders collection. We can have multiple databases and multiple collections per database.

Inside of the collection we have something called documents. Documents look like JavaScript JSON objects. Inside of a collection the documents are schema-less and can contain different data. This is the flexibility that MongoDB provides us with whereas SQL based database are very strict about the data stored within the database tables. Therefore, the MongoDB database can grow with the application needs. MongoDB is a No-SQL database.

Typically we will need some kind of structure in a collection because applications typically requires some type of structure to work with the data.

Diagram 1.1:

Database	Shop

Collections	Users	Orders

Documents	{name: 'Max', age: 28}	{product: 'pen', price: 1.99}
	{name: 'Sam'}	{product: 't-shirt'}

JSON (BSON) Data Format:

```
{
    "name": "Alex",
    "age": 29,
    "address": {
        "city": "Munich"
    },
    "hobbies": [
        { "name": "Cooking" },
        { "name": "Football" }
    ]
}
```

Key

"name": "Alex"

Name of the Key   Key Value

The above is an example of the JSON data format. A single document is surrounded by curly brackets. The data is normally structured with a Keys. Keys consist of a Name of the Key and a Key value. The Name of the Key (*which will be referred to as Key from now on*) and the Key Value must be wrapped around quotation marks (unless if the data is a type of number).

There are different types of values we can store such as: string, number, booleans and arrays.

We can also nest documents within documents. This allows us to create complex relations between

data and store them within one document, which makes working with the data and fetching data more efficient because it is contained in one document in a logical way. SQL in contrast requires more complex method of fetching data which require joins to find data in table A and data in table B to retrieve the relevant data.

Behind the scenes on the server, MongoDB converts the JSON data to a binary version of the data which can be stored and queried more efficiently. We do not need to concern ourselves with BSON as we would tend to work with JSON data.

The whole theme of MongoDB is flexibility, optimisation and usability and it is what really sets MongoDB apart from other database solutions because it is so efficient from a performance perspective as we can query data in the format we need it instead of running complex restructuring on the server.

The Key MongoDB Characteristics.

MongoDB is a no SQL solution because it is following an opposite concept/philosophy to SQL based databases. Instead of normalising the data i.e. storing data distributed across multiple tables where every table has a clear schema and then using relations, MongoDB goes for storing data together in a document. It does not force a schema hence schema-less/No-SQL.

We can have multiple documents in a single collection and they can have different structures as we have seen in Diagram 1.1. This is important, it can lead to messy data but it still our responsibility as developers to work with clean data and to implement a solution that works. On the other hand this provides us with a lot of flexibility. We could use mongoDB for applications that might still evolve, where the exact data requirements are not set yet. MongoDB allows us to started and we could always add data with more information in the same collection at a later point in time.

We also work with less relations. There are some relations, but with these embedded (nested) documents, we have less collections (tables) which we connect but instead we store data together. This is where the efficiencies is derived from, since data is stored together and when we fetch data from our application it does not require to reach out to multiple tables and merge the data because all the data is already within the single collection. This is where the speed, performance and flexibility comes from and can be seen beneficial for when building applications. This is the main reason why No-SQL solutions are so popular for read and write heavy applications.

MongoDB Ecosystem

The below Diagram 1.2 is the current snapshot of the MongoDB companies ecosystem and product offerings. The focus of this guide is on the MongoDB database used locally on our machines and on the cloud using Atlas. We will also dive in Compass and the Stitch world of MongoDB.

Diagram 1.2:

MongoDB Database	Stitch
Self-Managed/Enterprise · Atlas (Cloud) · Mobile	Serverless Query API
CloudManager/OpsManager	Serverless Functions
Compass	Database Triggers
BI Connectors	Realtime Sync
MongoDB Charts	

Installing MongoDB

MongoBD runs on all Operating Systems (OS) which include Windows/Mac/Linux. To install MongoDB we can visit their webpage on:

https://www.mongodb.com/

Under products select MongoDB server and download the MongoDB Community Server for our OS platform of choice. Install the MongoDB Server by following the installation steps.

Important Note: On Windows when installing click on the Custom Setup Type. MongoDB will be installed as a service which will be slightly different to how MongoDB runs on Mac & Linux.

On Mac and Linux we simply have a extracted folder which contains files. We would copy all the contents within this folder and paste them into any place within our OS i.e wherever we would want to install MongoDB.

We would then want to create a folder called data and a sub-folder called db anywhere within our OS, preferably in the root of the OS.

On Windows open up the command prompt or on Mac/Linux open up the terminal. This is where

we are going to spend most of out time using special commands and queries. Run the following command:

$ mongo

This should return command not found.

To fix this problem on a Mac go to the user folder and find a file called .bash_profile file (*if this does not exist we could simply create it*). Edit the file using a text editor. Add the following line:

export PATH=/Users/Username/mongobd/bin:$PATH

The path should be wherever we placed the mongoDB binary zip files. We need to add :$PATH at the end on Mac/Linux to make sure all our other commands work on our OS. Save the file and close the file.

Important Note: if you run into a problem on not being able to edit the .bash_profile using text editor use the following command to edit it within the terminal:

$ sudo nano ~/ .bash_profile

This will allow you to edit the file within the terminal and enter the mongo bin file path. Press CRTL + o to save and CTRL + x to exit the nano edit.

To fix this on a Windows OS, we need to create an environment variable. Press the windows key and type environment which should suggest the Edit Environment Variable option. Under the user variables edit Path to add the directory path to where we installed the mongoDB files:

C:\Program Files\MongoDB\Server\4.0\bin

Restart the terminal/command prompt and now run the command:

$ mongo

This should now return a error of connect failed on Mac/Linx.

On Windows it will connect because MongoDB is running as a service and has already started as a background service because we would have checked this during the installation. If we open the command prompt as administrator and ran the command 'net stop MongoDB' this will stop the background service running automatically and we can manually start and stop the MongoDB service running on windows. DO NOT RUN THIS COMMAND ON MAC/LINUX.

The mongo command is the client which allows us to connect to the server and then run commands on the databases. To start the service on Mac/Linux we would use the following command:

$ mongod

When running this command to start the server it may fail if we chose a different default /data/db folder. If we used a different folder and not within the root of our OS we would need to start the mongod command instance followed by the --dbpath flag and the place where the /data/db is located if not within the root directory.

$ sudo mongod –dbpath "/data/db"

On Mac we would need to run the mongod command every time we wish to run the mongoDB service whereas on Windows this will run automatically even after restarting the system.

Now that we have the mongod server running minimise the terminal on Mac/Linux and open up a new terminal. We cannot close the mongod server terminal because it is running the service and if closed the mongoDB server everything will stop working and we cannot continue to work with the database server. Pressing the CTRL + C keys within the terminal will quit the mongod service, but we would need to re-run the mongod command again should we wish to run the server again.

We are now in the mongo shell which is the environment where we can run commands against our database server. We can create new databases, collections and documents which we will now focus on in the following sections.

Time to get Started

Now that we have the mongod server running and we can now connect to it using the mongo shell we can now enter the following basic commands in the mongo terminal:

Command	Description
$ cls	Clear the terminal.
$ show dbs	Display existing databases (there are three default databases: admin, config and local which store meta data).
$ use databaseName	Connect/Switch to a database.
	If the database does not exist it will implicitly create a new database using the databaseName. It will not create the database until a collection and document is added.
$ db.collectionName.insertOne({"name of key": "key value"})	Create a new collection. The db relates to the current connected database. This will implicitly create a new collection if it does not exist. We must pass at least one new data in the collection using the .insert() command passing in a JSON object. This will return the object to confirm the data was inserted into the database.

Important Note: we can omit the quotes around the name of the key within the shell but we must contain the quotes for the key value unless the key value is type of number. This is a feature within the mongo shell which work behind the scenes. MongoDB will also generate a uniqueId for new documents inserted into the collection.

Command	Description
$ db.collectionName.find()	Display the document within the database collection.
$ db.collectionName.find().pretty()	Display the documents within the database collection but prettify the data in a more humanly readable format.

This is a very basic introductory look at the following shell commands we can run in the mongo terminal to create a new database, switch to a database, create a collection and documents and display all the documents within a database collection either in the standard or pretty format.

Tip: to run the mongod server on a different port to the default port 27017 by run the following command. Note you would need to specify the port when running the mongo shell command as well. You would use this in case the default port is being used by something else.

$ sudo mongo --port 27018

$ mongo --port 27018

Shell vs Drivers

The shell is a great neutral ground for working with mongoDB. Drivers are packages we install for different programming languages the application might be written in. There are a whole host of drivers for the various application server languages such as PHP, node, C#, python etc. Drivers are the bridges between the programming language and the mongoDB server.

As it turns out, in these drivers, we would use the same command as we use in the shell, they are just slightly adjusted to the syntax of the language we are working with.

The drivers can be found on the mongoDB website:

https://docs.mongodb.com/ecosystem/drivers/

Throughout this document we will continue to use the Shell commands as it is the neutral commands. We can take the knowledge of how to insert, configure inserts, query data, filter data, sort data and many more shell commands. These commands will continue to work when we use the drivers but we would need to make reference to the driver documentation to understand how to use the shell commands but using the programming language syntax to perform the commands using the drivers. This will make us more flexible with the language we use when building applications that uses mongoDB.
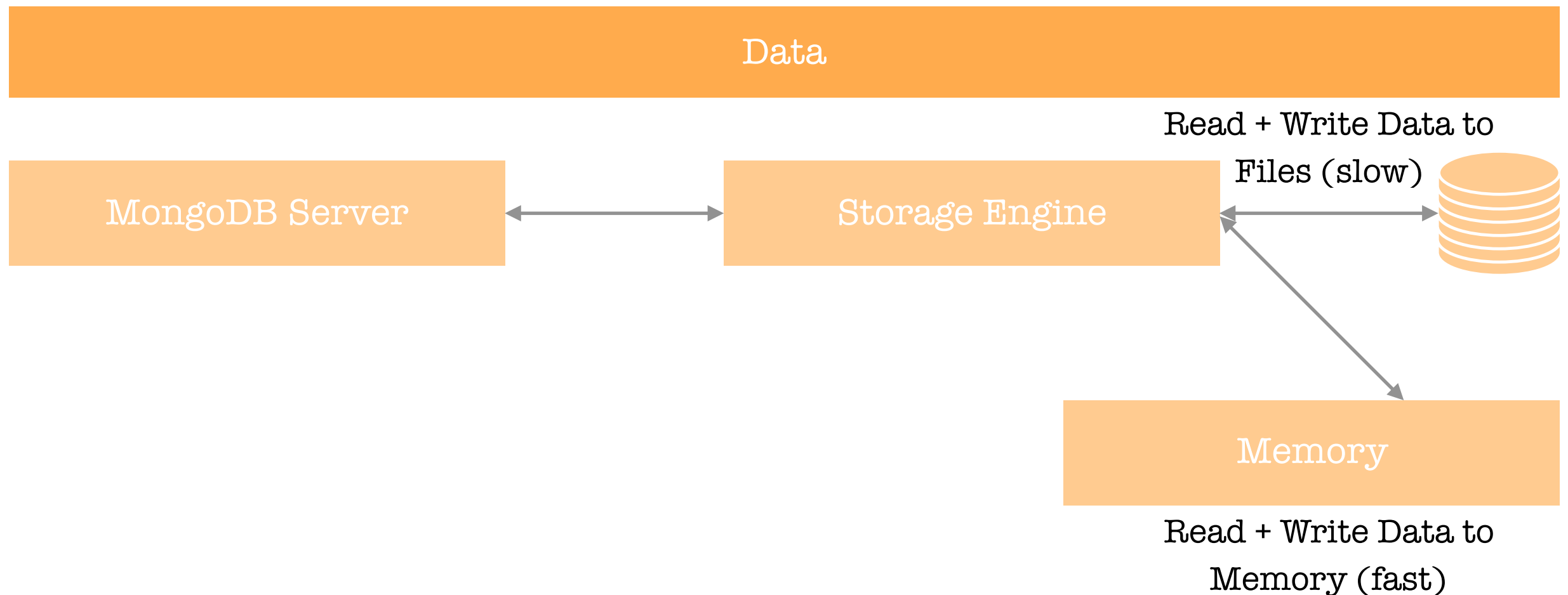
MongoDB & Clients: The Big Picture

Diagram 1.3:

Application				Data	

Frontend (UI)	Backend (Server)	Drivers		MongoDB Server	Storage Engine

Queries

Node

Java

Python

Communicate   File/Data Access

MongoDB Shell

Playground, Administration

MongoDB & Clients: The Big Picture

Diagram 1.4:

As we can see in Diagram 1.3 the application driver/shell communicates to the mongoDB server. The MongoDB server communicates with the storage engine. It is the Storage Engine which deals with the data passed along by the MongoDB Server, and as Diagram 1.4 depicts it will read/write to database and/or memory.

Create, Read, Update & Delete (CRUD)

We could use MongoDB to create a variety of things such as an application, Analytics/BI Tools or data administration. In an application case, we may have an app where the user interacts with our code (the code can be written in any programming language) and the mongoDB driver will be included in the application. In the case of a Analytics/BI Tools we may use the BI Connector/Shell provided by mongoDB or another import mechanism provided by our BI tool. Finally, in the database administrator case we would interact with the mongoDB shell.

In all the above cases we would want to interact with the mongoDB server. In an application we would typically want to be able to create, read, update or delete elements e.g. a blog post app. With analytics, we would at least want to be able to read the data and as an admins we would probably want to do all the CRUD actions.

CRUD are the only actions we would want to perform with our data i.e. to create it, manage it or read it. We perform all these actions using the mongoDB server.

Diagram 1.5:

CREATE
insertOne(data, options)
insertMany(data, options)

UPDATE
updateOne(filter, data, options)
updateMany(filter, data, options)
replaceOne(filter, data, options)

READ
findOne(filter, options)
find(filter, options)

DELETE
deleteOne(filter, options)
deleteMany(filter, options)

The above are the four CRUD operations and the commands we can run for each action. In later sections we will focus on each CRUD action individually to understand in-depth each of the actions and syntax/command we can use when performing CRUD operation with our mongoDB data collection and documents.

Finding, Inserting, Updating & Deleting Elements

To show all the existing databases within the mongoDB server we use the command "show dbs" while we use the use followed by the database name to switch to a database. The db will then relate to the switched database.

To perform any CRUD operations, these commands must always be performed/executed on a collection where you want to create/update/delete documents. Below are example snippets of CRUD commands on a fictitious flights database (where the collection is called flightData).

$ db.flighData.insertOne({distance: 1200})

This will add a single document to the collection as we have seen previously.

$ db.flightData.deleteOne({departureAirport: "LHR"})
$ db.flightData.deleteMany({departureAirport: "LHR"})

The delete command takes in a filter. To add a filter we would use the curly brackets passing in

which name of key and key value of the data we wish to filter an delete. In the above example we used the departureAirport key and the value of TXL. The deleteOne command will find the first document in our database collection that meets the criteria and deletes it. The command will return:

{ "acknowledged" : true, "deleteCount" : 1 }

If a document was deleted in the collection this will show the number of deleted documents (the deleteOne command will always return 1). If no documents matched the filter and none were deleted the returned deleteCount value will be 0.

The deleteMany in contrast will delete many documents at once where the documents matches the filter criteria specified.

Note: The easiest way to delete all data in a collection is to delete the collection itself.

$ db.flightData.updateOne({distance: 1200}, { $set: {marker: "delete"} })
$ db.flightData.updateMany({distance: 1200}, { $set: {marker: "delete"} })

The update command takes in 3 argument/parameters. The first is the filter which is similar to the delete command. The second is how we would want to update/change the data. We must use the

{ $set: { } } keyword (*anything with a $ dollar sign in front of the keyword is a reserved word in mongoDB*) which lets mongoDB know how we are describing the changes we want to make to a document. If the update key:value does not exist, this will create a new key:value property within the document else it will update the existing key:value with the new value passed in. The third parameter is options which we will analyse in great detail in the latter sections.

Important Note: when passing in a filter we can also pass in empty curly brackets { } which will select all documents within the collection.

If successful with updating many this will return within the terminal an acknowledgement as seen below, where the number of matched the filter criteria and the number of data modified:

{ "acknowledged" : true, "matchedCount" : 2, "modified" : 2 }

If we were to delete all the documents within a collection and use the command to find data in that collection i.e using the db.flightData.find().pretty() command, the terminal will return empty/nothing as there are no existing documents to read/display.

The above demonstrates how we can find, insert, update and delete elements using the update and delete command.