Installation steps of mongodb in Ubuntu/Fedora

1) Download mongodb zipfile
2) Extract file in home directory
3) Create new folder with name data/db in any drive
4) From terminal type following commands
    a. > cd (give path where extracted file is saved)
    b. > cd bin
    c. > ./mongod –dbpath “give path where your new folder created as data/db”
    d. >press entere
    e. Your systemis waiting for new connection.

Client Side

1) Open terminal
2) > cd (give path where extracted file is saved)
3) > cd bin
4) >mongo

You will be connected with server

Type db to check

>db

Test

Installation steps of mongodb in Windows

- To install the MongoDB on windows, first doownload the latest release of MongoDB from http://www.mongodb.org/downloads Make sure you get correct version of MongoDB depending upon your windows version.

- In case you have extracted the mondodb at different location, then go to that path by using command cd FOOLDER/DIR and now run

- MongoDB requires a data folder to store its files. The default location for the MongoDB data directory is c:\data\db. So you need to create this folder using the Command Prompt

- In command prompt navigate to the bin directory present into the mongodb installation folder. Suppose my installation folder is D:\set up\mongodb

- D:\set up\mongodb\bin>mongod.exe --dbpath "d:\set up\mongodb\data"

- This will show waiting for connections message on the console output indicates that the mongod.exe process is running successfully.

- Now to run the mongodb you need to open another command prompt and issue the following command

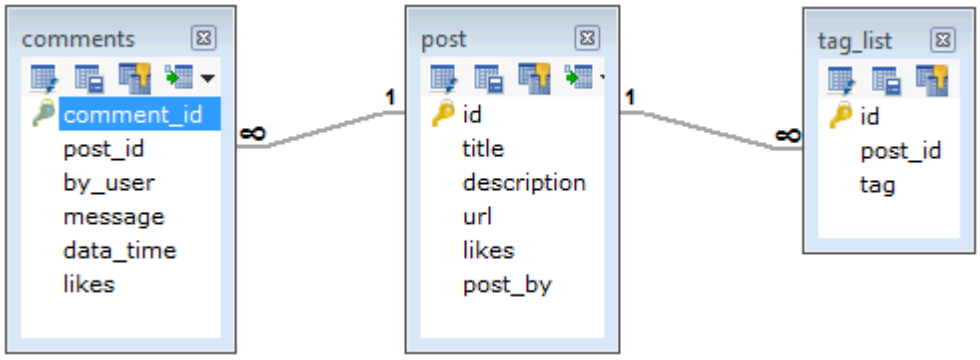
- D:\set up\mongodb\bin>mongo.exe

```
D:\set up\mongodb\bin>mongod.exe --dbpath "d:\set up\mongodb\data"
D:\set up\mongodb\bin>mongo.exe
```

- Db
- Db.test
- Db.stats()
- Db.collectionName
- Db.collectionName.insert
- Db.student.insert({name: “sambhaji”})
- Use mydb (switch db from one db to other)
- Db.student.find();

- Db.stats() : to show statistics of mongodb server

- **Example:** Suppose a client needs a database design for his blog website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.



mongoDB Query:

```
{
_id: POST_ID
title: TITLE_OF_POST,
description: POST_DESCRIPTION,
by: POST_BY,
url: URL_OF_POST,
tags: [TAG1, TAG2, TAG3],
likes: TOTAL_LIKES,
comments: [
{
user:'COMMENT_BY',
message: TEXT,
dateCreated: DATE_TIME,
like: LIKES
},
{
user:'COMMENT_BY',
message: TEXT,
dateCreated: DATE_TIME,
like: LIKES
}
]
```

So while showing the data, in RDBMS you need to join three tables and in mongodb data will be shown from one collection only.

Creating DB

- Use Database name ex: use Mydb
- Db
- Show dbs : If you want to check your databases list, then use the command
- To display database you need to insert at least one document into it.
- By default db in mongodb is test, If you not created any db by default all collections stored in test db

Drop DB

Syntax:

Basic syntax of dropDatabase () command is as follows:

```
Db.dropDatabase()
```

Create Collections

Syntax:

Basic syntax of createCollection() command is as follows:

```
db.createCollection(name, options)
```

In the command, name is name of collection to be created. Options is a document and used to specify configuration
of collection

Options: Specify options about memory size and indexing, capped, autoIndexID, size, max

Example

```
> use test
switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }
                                        >
```

You can check the created collection by using the command show collections

```
>show collections
```

Following example shows the syntax of createCollection() method with few important options:

```
>db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } )

{ "ok" : 1 }
                                                    >
```

In mongodb you don't need to create collection. MongoDB creates collection automatically, when you insert some
document.

```
>db.students.insert({"name" : "sambhaji"})
>show collections
mycol
mycollection
system.indexes
students
```

drop Collections

The drop() Method

Syntax:

MongoDB's db.collection.drop() is used to drop a collection from the database

```
db.COLLECTION_NAME.drop()
```

MongoDB Datatypes

MongoDB supports many datatypes whose list is given below:

1) String : This is most commonly used datatype to store the data. String in mongodb must be UTF-8 valid.

2) Integer : This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.

3) Boolean : This type is used to store a boolean (true/ false) value.

4) Double : This type is used to store floating point values.

5) Min/ Max keys : This type is used to compare a value against the lowest and highest BSON elements.

6) Arrays : This type is used to store arrays or list or multiple values into one key.

7) Timestamp : ctimestamp. This can be handy for recording when a document has been modified or added.

8) Object : This datatype is used for embedded documents.

9) Null : This type is used to store a Null value.

10) Symbol : This datatype is used identically to a string however, it's generally reserved for languages that use aspecific symbol type.

11) Date : This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.

12) Object ID : This datatype is used to store the document's ID.

13) Binary data : This datatype is used to store binay data.

14) Code : This datatype is used to store javascript code into document.

15) Regular expression : This datatype is used to store regular expression

ObjectID

_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows:

_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)

To insert multiple documents in single query, you can pass an array of documents in insert() command.

Example

```
>db.post.insert([
{
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
},
{
title: 'NoSQL Database',
description: 'NoSQL database doesn't have tables',
by: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 20,
comments: [
{
user:'user1',
message: 'My first comment',
dateCreated: new Date(2013,11,10,2,35),
like: 0
}
]
}
                                                    ])
```

MongoDB Query Documents

1) The find() method

To query data from MongoDB collection, you need to use MongoDB's find() method.

Syntax
Basic syntax of find() method is as follows

```
>db.COLLECTION_NAME.find()
```

find() method will display all the documents in a non structured way.

2) The pretty() method

To display the results in a formatted way, you can use pretty() method.
Syntax:

```
>db.mycol.find().pretty()
```

Apart from find() method there is findOne() method, that reruns only one document.

3) RDBMS Where Clause Equivalents in MongoDB

To query the document on the basis of some condition, you can use following operations

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50

4) AND in MongoDB

To display the results in a formatted way, you can use pretty() method.

Syntax:

```
>db.mycol.find({key1:value1, key2:value2}).pretty()
```

Example:

```
>db.mycol.find({"by":"sam","title": "MongoDB Overview"}).pretty()
```

Output

```
{
"_id": ObjectId(7df78ad8902c),
"title": "MongoDB Overview",
"description": "MongoDB is no sql database",
"by": "sam",
"url": "http://www.tutorialspoint.com",
"tags": ["mongodb", "database", "NoSQL"],
"likes": "100"
}
    >
```

5) OR in MongoDB

Syntax:
To query documents based on the OR condition, you need to **use $or keyword.**
Basic syntax of OR is shown below:

```
>db.mycol.find(
{
$or: [
{key1: value1}, {key2:value2}
]
}
    ).pretty()
```

```
>db.mycol.find({$or:[{"by":"sam"},{"title": "MongoDB Overview"}]}).pretty()
```

6) Using AND and OR together

Example:

Below given example will show the documents that have likes greater than 100 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent sql where clause is 'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'

```
>db.mycol.find("likes": {$gt:10}, $or: [{"by": "tutorials point"}, {"title":
"MongoDB Overview"}] }).pretty()
```

MongoDB Update Document

1) MongoDB Update() method

The update() method updates values in the existing document.
Syntax:

Basic syntax of update() method is as follows

```
>db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)
```

Example:

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB
Tutorial'}})
>db.mycol.find()

{ "_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
>
```

By default mongodb will update only single document, to update multiple you need to set a paramter 'multi' to true.

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}},{multi:true})
>
```

MongoDB Delete Document

The remove() Method

MongoDB's remove() method is used to remove document from the collection. remove() method accepts two
parameters. One is deletion criteria and second is justOne flag
1. deletion criteria : (Optional) deletion criteria according to documents will be removed.
2. justOne : (Optional) if set to true or 1, then remove only one document.

Syntax:
Basic syntax of remove() method is as follows

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)>
```

```
>db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Remove only one

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove All documents

If you don't specify deletion criteria, then mongodb will delete whole documents from the collection. This is equivalent of SQL's truncate command.

```
>db.mycol.remove()
>db.mycol.find()
```

MongoDB Projections

In mongodb projection meaning is selecting only necessary data rather than selecting whole of the data of a
document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

1) The find() Method

MongoDB's find() method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB when you execute find() method, then it displays all fields of a document.

To limit this you need to set list of fields with value 1 or 0. 1 is used to show the filed while 0 is used to hide the field.

Syntax:
Basic syntax of find() method with projection is as follows

```
>db.COLLECTION_NAME.find({},{KEY:1})
```

Example

Consider the collection myycol has the following data

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point
Overview"}
```

Following example will display the title of the document while quering the document.

```
>db.mycol.find({},{"title":1,_id:0})
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
```

Please note _id field is always displayed while executing find() method, if you don't want this field, then you need to set it as 0

MongoDB Limit Records

1) The Limit() Method

To limit the records in MongoDB, you need to use limit() method. limit() method accepts one number type argument, which is number of documents that you want to displayed.

Syntax:

Basic syntax of limit() method is as follows

```
>db.COLLECTION_NAME.find().limit(NUMBER)>
```

Example

Following example will display only 2 documents while querying the document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

If you don't specify number argument in limit() method then it will display all documents from the collection.

MongoDB Skip() Method

Apart from limit() method there is one more method skip() which also accepts number type argument and used to
skip number of documents.

Syntax:
Basic syntax of skip() method is as follows

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)>
```

Example

Following example will only display only second document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)
{"title":"NoSQL Overview"}
>
```

Please note default value in skip() method is 0

MongoDB Sort Document

The sort() Method

To sort documents in MongoDB, you need to use sort() method. sort() method accepts a document containing list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

Syntax:
Basic syntax of sort() method is as follows

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

Example

Following example will display the documents sorted by title in descending order.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"SQL Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

Please note if you don't specify the sorting preference, then **sort()** method will display documents in ascending order.

MongoDB Indexing

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require the mongod to process a large volume of data. Indexes are special data structures, that store a small portion of the data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in index.

The ensureIndex() Method

To create an index you need to use ensureIndex() method of mongodb.

Syntax:
Basic syntax of ensureIndex() method is as follows()

```
>db.COLLECTION_NAME.ensureIndex({KEY:1})
```

Here key is the name of filed on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

Example

```
>db.mycol.ensureIndex({"title":1})
>
```

In ensureIndex() method you can pass multiple fields, to create index on multiple fields.

```
>db.mycol.ensureIndex({"title":1,"description":-1})
>
```

ensureIndex() method also accepts list of options (which are optional)

MongoDB Aggregation

Aggregations operations process data records and return computed results.

Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In sql count(*) and with group by is an equivalent of mongodb aggregation.

The aggregate() Method

For the aggregation in mongodb you should use aggregate() method.

Syntax:

Basic syntax of aggregate() method is as follows

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

>
```

Examples:

In the collection you have the following data:

```
{
_id: ObjectId(7df78ad8902c)
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by_user: 'sam',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
},
{
_id: ObjectId(7df78ad8902d)
title: 'NoSQL Overview',
description: 'No sql database is very fast',
by_user: 'sam',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 10
},
{
_id: ObjectId(7df78ad8902e)
title: 'Neo4j Overview',
description: 'Neo4j is no sql database',
by_user: 'Neo4j',
url: 'http://www.neo4j.com',
tags: ['neo4j', 'database', 'NoSQL'],
likes: 750
},
```

Now from the above collection if you want to display a list that how many tutorials are written by each user then you will use aggregate() method as shown below:

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])

{
"result" : [
{
"_id" : "sam",
"num_tutorial" : 2
},
{
"_id" : "Neo4j",
"num_tutorial" : 1
}
],
"ok" : 1
}
>
```

Sql equivalent query for the above use case will be select by_user, count(*) from mycol group by by_user

There is a list available aggregation expressions

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

Pipeline Concept

In UNIX command shell pipeline means the possibility to execute an operation on some input and use the output as
the input for the next command and so on

Possible stages in aggregation framework are following:

1) $project: Used to select some specific fields from a collection.
2) $match: This is a filtering operation and thus this can reduce the amount of documents that are given as input to
the next stage.
3) $group: This does the actual aggregation as discussed above.
4) $sort: Sorts the documents.

5) $skip: With this it is possible to skip forward in the list of documents for a given amount of documents.
6) $limit: This limits the amount of documents to look at by the given number starting from the current position.s
7) $unwind: This is used to unwind document that are using arrays. when using an array the data is kind of prejoinded and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

Examples:

$project

```
> db.student1.aggregate({$project:{name : 1}})
```

$sort

```
> db.student1.aggregate({$sort: {name: 1}})

> db.student1.aggregate({$sort: {name: -1}})
```

MongoDB Replication

Replication is the process of synchronizing data across multiple servers.
Replication provides redundancy and increases data availability with multiple copies of data on different database servers, replication protects a database from the loss of a single server. Replication also allows you to recover from hardware failure and service interruptions. With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.

Why Replication?

- To keep your data safe
- High (24*7) availability of data
- Disaster Recovery

 No downtime for maintenance (like backups, index rebuilds, compaction)
 Read scaling (extra copies to read from)
 Replica set is transparent to the application

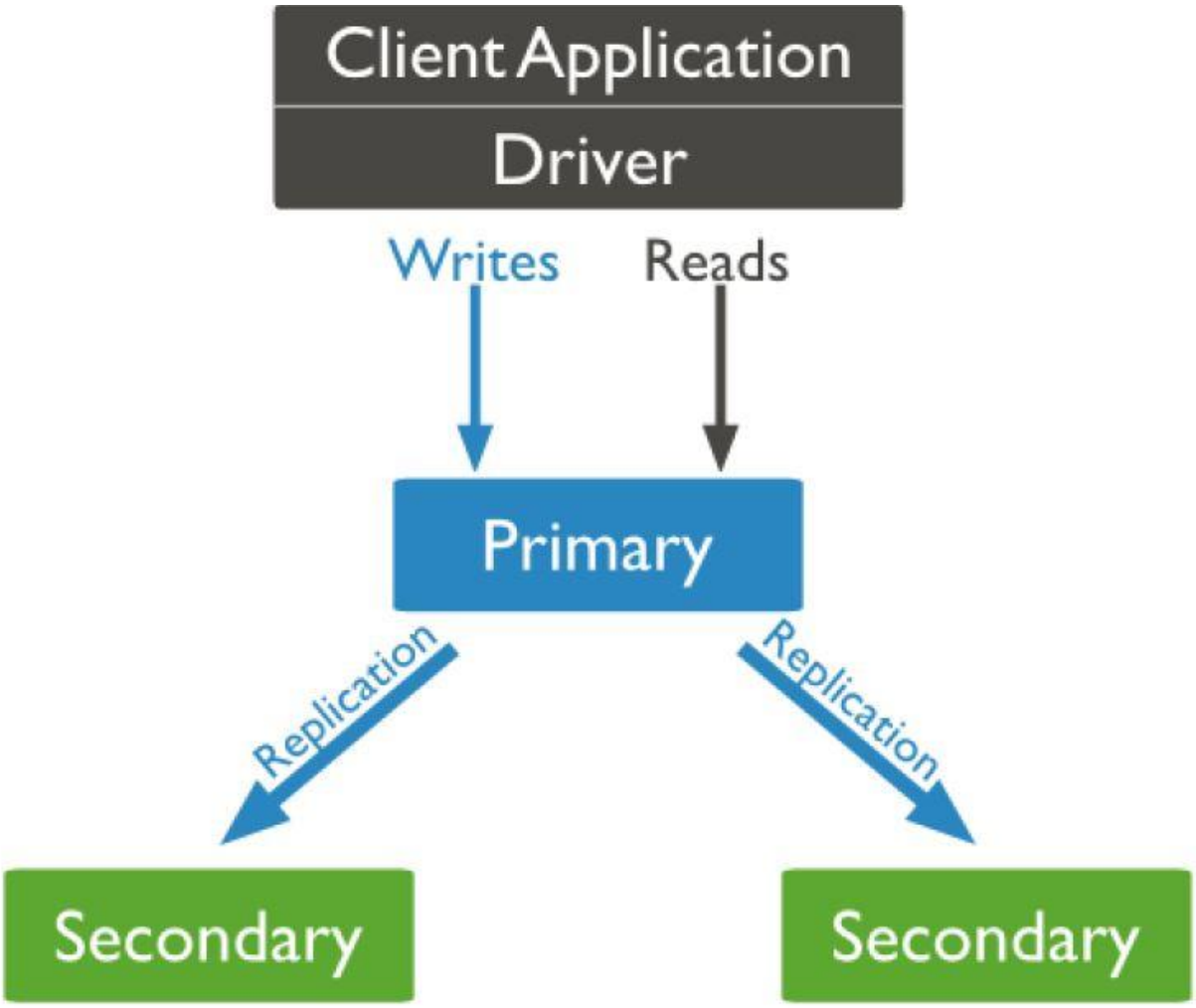


Fig. Flow of Replication in mongoDB

Replica set features

 A cluster of N nodess
 Anyone node can be primary
 All write operations goes to primary

 Automatic failover
 Automatic Recovery
 Consensus election of primary

Set up a replica set

• Shutdown already running mongodb server.

Now start the mongodb server by specifying --replSet option. Basic syntax of --replSet is given below:

```
mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet "REPLICA_SET_INSTANCE_NAME"
```

Example:

```
mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0
```

It will start a mongod instance with the name rs0, on port 27017. Now start the command prompt and connect to this mongod instance. In mongo client issue the command rs.initiate() to initiate a new replica set. To check the replica set configuration issue the command rs.conf(). To check the status of replica sete issue the command rs.status().

Add members to replica set

To add members to replica set, start mongod instances on multiple machines. Now start a mongo client and issue a command rs.add().

Syntax:

Basic syntax of rs.add() command is as follows:

```
>rs.add(HOST_NAME:PORT)
```

Example
Suppose your mongod instance name is mongod1.net and it is running on port 27017. To add this instance to replica set issue the command rs.add() in mongo client.

```
>rs.add("mongod1.net:27017")
>
```

You can add mongod instance to replica set only when you are connected to primary node. To check whether you are connected to primary or not issue the command db.isMaster() in mongo client.

MongoDB Sharding

Sharding

Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling. With sharding, you add more machines to support data growth and the demands of read and write operations.

Why Sharding?

- In replication all writes go to master node
- Latency sensitive queries still go to master
- Single replica set has limitation of 12 nodes

 Memory can't be large enough when active dataset is big
 Local Disk is not big enough
 Vertical scaling is too expensive

Sharding in MongoDB

Below given diagram shows the sharding in MongoDB using sharded cluster.

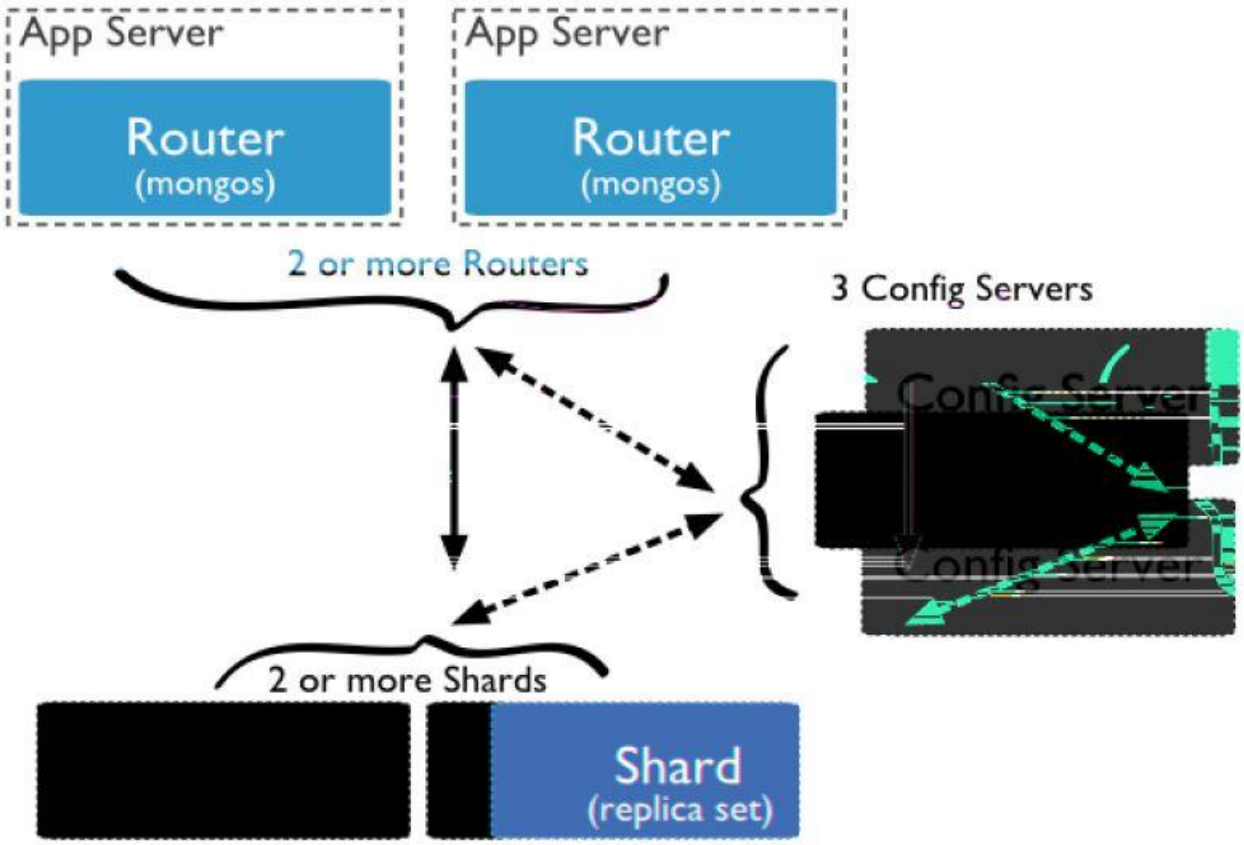


Fig: MongoDB Sharding

In the above given diagram there are three main components which are described below:

 Shards: Shards are used to store data. They provide high availability and data consistency. In production environment each shard is a separate replica set.

 Config Servers: Config servers store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. In production environment sharded clusters have exactly 3 config servers.

 Query Routers: Query Routers are basically mongos instances, interface with client applications and direct operations to the appropriate shard. The query router processes and targets operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Generally a sharded cluster have many query routers.

MongoDB using Java

Installation

Before we start using MongoDB in our Java programs, we need to make sure that we have MongoDB JDBC Driver and Java set up on the machine. Now, let us check how to set up MongoDB JDBC driver.

 You need to download the jar from the path Download mongo.jar. Make sure to download latest release of it.

 You need to include the mongo.jar into your classpath.

Create a collection

To create a collection, createCollection() method of com.mongodb.DB class is used

```
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;
public class MongoDBJDBC{
public static void main( String args[] ){
try{
// To connect to mongodb server
MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
// Now connect to your databases
DB db = mongoClient.getDB( "test" );
System.out.println("Connect to database successfully");
boolean auth = db.authenticate(myUserName, myPassword);
System.out.println("Authentication: "+auth);
DBCollection coll = db.createCollection("mycol");
System.out.println("Collection created successfully");
}catch(Exception e){
System.err.println( e.getClass().getName() + ": " + e.getMessage() );
}
}
                                                    }
```

```
Output:
Connect to database successfully
Authentication: true
Collection created successfully
Collection mycol selected successfully
```

Retrieve all documents

To select all documents from the collection, find() method of com.mongodb.DBCollection class is used. This method returns a cursor, so you need to iterate this cursor.

```
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;
public class MongoDBJDBC{
public static void main( String args[] ){
try{
// To connect to mongodb server
MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
// Now connect to your databases
DB db = mongoClient.getDB( "test" );
System.out.println("Connect to database successfully");
boolean auth = db.authenticate(myUserName, myPassword);
System.out.println("Authentication: "+auth);
DBCollection coll = db.getCollection("mycol");
System.out.println("Collection mycol selected successfully");
DBCursor cursor = coll.find();
int i=1;
while (cursor.hasNext()) {
System.out.println("Inserted Document: "+i);
System.out.println(cursor.next());
i++;
}
}catch(Exception e){
System.err.println( e.getClass().getName() + ": " + e.getMessage() );
}
}
                                                    }
```

Output:

```
Connect to database successfully
Authentication: true
Collection mycol selected successfully
Inserted Document: 1
{
"_id" : ObjectId(7df78ad8902c),
"title": "MongoDB",
"description": "database",
"likes": 100,
"url": "http://www.tutorialspoint.com/mongodb/",
"by": "tutorials point"
                                                    }
```

Delete first document

To delete first document from the collection, you need to first select the documents using findOne() method and then remove method of com.mongodb.DBCollection class.

```
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;
public class MongoDBJDBC{
public static void main( String args[] ){
try{
// To connect to mongodb server
MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
// Now connect to your databases
DB db = mongoClient.getDB( "test" );
System.out.println("Connect to database successfully");
boolean auth = db.authenticate(myUserName, myPassword);
System.out.println("Authentication: "+auth);
DBCollection coll = db.getCollection("mycol");
System.out.println("Collection mycol selected successfully");
DBObject myDoc = coll.findOne();
col1.remove(myDoc);
DBCursor cursor = coll.find();
int i=1;
while (cursor.hasNext()) {
System.out.println("Inserted Document: "+i);
System.out.println(cursor.next());
i++;
}
System.out.println("Document deleted successfully");
}catch(Exception e){
System.err.println( e.getClass().getName() + ": " + e.getMessage() );
}
}
}
```

When program is compiled and executed, it will produce the following result:

```
Output:
Connect to database successfully
Authentication: true
Collection mycol selected successfully
Document deleted successfully
```