Replica

mongod --port 2001 --dbpath G:\Replica\data1\db --replSet withlove

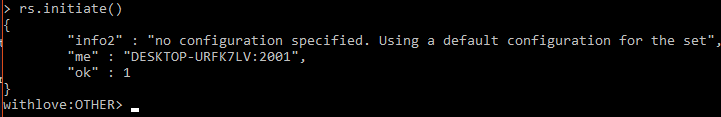
mongod --port 2002 --dbpath G:\Replica\data2\db --replSet withlove

mongod --port 2003 --dbpath G:\Replica\data3\db --replSet withlove

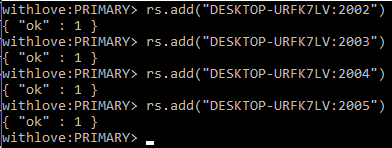
mongod --port 2004 --dbpath G:\Replica\data4\db --replSet withlove

mongod --port 2005 --dbpath G:\Replica\data5\db --replSet withlove

mongo --port 2001



Add members



Remove member



Add member as arbitrary



withlove:PRIMARY> rs.conf()

{

"\_id" : "withlove",

"version" : 7,

"protocolVersion" : NumberLong(1),

"members" : [

{

"\_id" : 0,

"host" : "DESKTOP-URFK7LV:2001",

"arbiterOnly" : false,

"buildIndexes" : true,

"hidden" : false,

"priority" : 1,

"tags" : {

},

"slaveDelay" : NumberLong(0),

"votes" : 1

},

{

"\_id" : 1,

"host" : "DESKTOP-URFK7LV:2002",

"arbiterOnly" : false,

"buildIndexes" : true,

"hidden" : false,

"priority" : 1,

"tags" : {

},

"slaveDelay" : NumberLong(0),

"votes" : 1

},

{

"\_id" : 2,

"host" : "DESKTOP-URFK7LV:2003",

"arbiterOnly" : false,

"buildIndexes" : true,

"hidden" : false,

"priority" : 1,

"tags" : {

},

"slaveDelay" : NumberLong(0),

"votes" : 1

},

{

"\_id" : 3,

"host" : "DESKTOP-URFK7LV:2004",

"arbiterOnly" : false,

"buildIndexes" : true,

"hidden" : false,

"priority" : 1,

"tags" : {

},

"slaveDelay" : NumberLong(0),

"votes" : 1

},

{

"\_id" : 4,

"host" : "DESKTOP-URFK7LV:2005",

"arbiterOnly" : true,

"buildIndexes" : true,

"hidden" : false,

"priority" : 1,

"tags" : {

},

"slaveDelay" : NumberLong(0),

"votes" : 1

}

],

"settings" : {

"chainingAllowed" : true,

"heartbeatIntervalMillis" : 2000,

"heartbeatTimeoutSecs" : 10,

"electionTimeoutMillis" : 10000,

"catchUpTimeoutMillis" : 60000,

"getLastErrorModes" : {

},

"getLastErrorDefaults" : {

"w" : 1,

"wtimeout" : 0

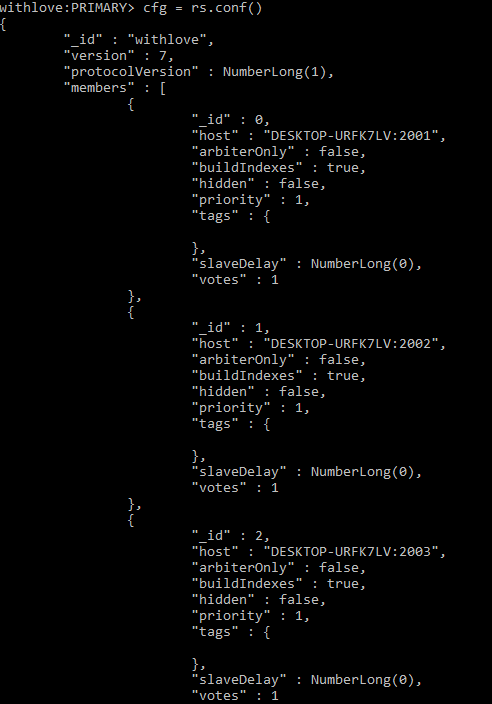
},

"replicaSetId" : ObjectId("59a1b4efa039af8b3da4df5b")

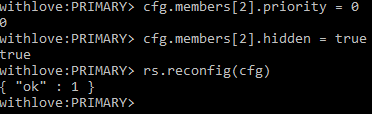
}

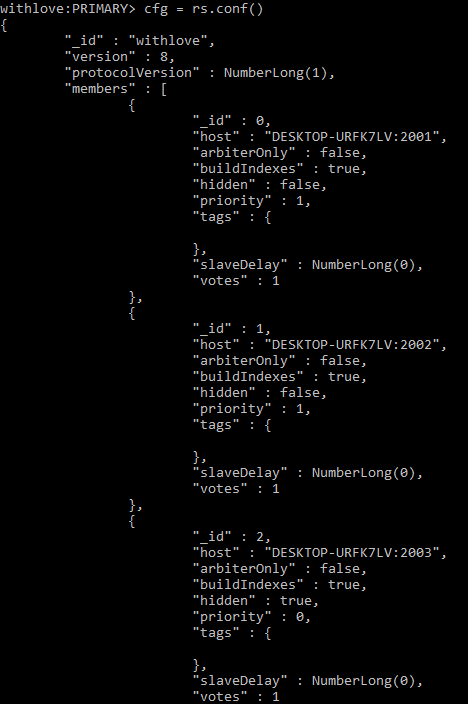
}

>>Member Configuration

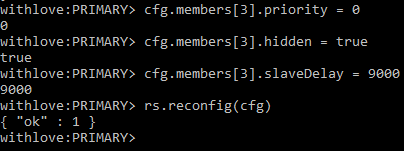






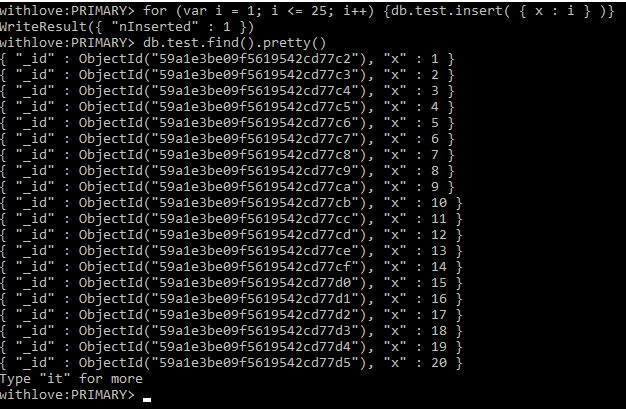
****











**Premise 1**

MongoDB releases a new version. How do we upgrade our production systems with minimal downtime?  
1) On a single node ( one node = one mongod instance )  
a. Back up the data  
b. Shutdown the mongod instance  
c. Replace the existing bins by the latest ones (that you would have downloaded from mongodb.org/downloads  
d. Restart the instance  
2) Some details (you are in a putty command line or in a unix terminal)  
a. Starting a mongod instance  
i. mongod –fork –logpath /var/log/mongodb.log –dbpath /srv/mongodb/db this will record log output to /var/log/mongodb.log and store the data into /srv/mongodb/db  
ii. if then you want to add this instance as a member of a replica set, use the replSet option ( for example “—replset rs0” means that this instance is usable as a member of the replicaset that has “rs0”as setname. In this case, you also need to note the host name and port information of this new host that is returned to you when you issue the command.  
b. How to shutdown a mongod instance.  
i. mongod –shutdown  
c. How do I access the mongo shell ?  
i. <path to bin>/mongo  
d. How do I add up a created mongod instance as a member of a replica set ?  
i. Connect to the replica set primary (if you don’t know which one it is, use db.isMaster() or rs.status() in the mongo shell )  
ii. rs.add(“<hostname>:<ports why you needed to note them when creating the mongod.  
3) On a replica set (multiple nodes, as many mongod instances), that’s what you have in the general case and that’s where you can minimize downtime  
a. Checkout the running nodes : ps aux | grep mongod  
b. Typically, there are one primary ( to which go the reads and the writes by default by the way ), and one or several secondaries that insure redundancy. You could configure the replica set to let the reads target secondaries but you wouldn’t be sure of the exactness of the data you get as the secondaries are not exactly up to date. Let’s assume here that you have one primary and two secondaries.  
c. Upgrade the version of one of your secondaries as explained above. Don’t forget to add up the new instance to the replica set again. Then do it for the other secondary.

d. You could just go on and do it for the primary. But the thing is if you kill the primary, the system will wait for 5 heartbeats ( 1 heartbeat = 2 seconds by default ) to elect one of the secondaries as the new primary. That means, you’ll have a 10 secs downtime. And there is a better solution : before killing the last mongod, you can manually trigger the election of a new primary by doing “ db.runCommand({replSetStepDown : 10})” on the primary in the mongo shell. Once it’s done, the old primary will be a secondary and one of the secondaries will be the primary. Then you can do the previous process on the last mongod without any downtime but the small gap you’ll have had during the election of the new primary

**Premise 2**

We have a new set of queries we want to run against the production system. But whenever we try to build the indexes for these queries, we experience a massive slowdown. How can we build indexes on production while minimizing the performance penalty?  
It’s quite the same idea as what’s above. From the mongo shell, building an index is as easy as : db.records.ensureIndex( { userid: 1 } )  
(this guy builds an index on the “userid” field of the collection “records”.  
But doing this would use some RAM on the machine that does it of course and it might slow down the queries if it’s done on the primary. What you can do is doing it on the secondaries first, then trigger the election of one of them as primary as described previously, and then do it on the old primary.

**Premise 3**

Your primary has become unavailable because of a power surge. Fortunately, automatic failover kicked in and the system never went down. Unfortunately, you never set alerts to inform you when a node becomes unavailable, so weeks go by until you notice. When you bring it back up, the node does not restart cleanly. Find out why and fix it.  
Here it was about the Oplog. What’s that? Remember that all the writes go to the primary? Well, the oplog is basically the buffer between primary and secondaries. When something is written in the primary, it also gets in the oplog and the secondaries pull data from it regularly. The oplog size is to be defined when the instance of the primary is created and of course the more updates you’re going to have by unit of time, the bigger the oplog needs to be and it should be large enough to contain at least a few hours of updates. In case you need to change the size of your oplog, it is possible and the procedure is described in mongo’s tutorials.

To come back to the exercise, in this scenario what happened is that the primary was down for a moment and so one of the secondaries became primary but the old primary was down long enough for the updates to be too numerous in-between to fit in the oplog. And in this case you have this node that cannot retrieve a part of the data simply because they have already been overwritten in the oplog when the node “woke up” and it might cause some malfunctions with it. To solve that we just killed it and made a new one.