**MongoDB Cluster configuration**

<http://www.10gen.com/presentations/webinar/deployment-best-practices>

<http://docs.mongodb.org/master/MongoDB-Manual-master.pdf>

<http://www.10gen.com/presentations/mongosv-2011/deploying-mongodb-for-high-availability>

<http://www.10gen.com/presentations/mongosv-2011/mongodb-and-aws>

Goal of this document is to setup MongoDB in single node configuration, examing the runtime for settings and output we can monitor and setup a cluster.

Requirements; Javascript and Python language skills.

**This material is a review of the MongoDB One Server Install but is now modified to run on a CentOS Amazon AMI.**

First verify Mongodb runs in one process, a single mongod daemon which is the database server. There are several ways to start the mongo database:

1. **>sudo service mongod start** if you installed this as a service
2. Using the command line;

**>sudo mongod**

Running mongod without any parameters will run mongod with default parameters for the dbpath( /data/db), logfile path(stdout) and port(27017+1000) are used.

You should see an OK sign after starting the mongod process. If you don’t see this you have a conflict preventing mongod from starting.

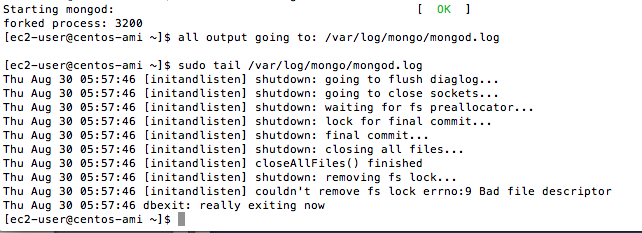
**Starting mongod: [ OK ]**

If the OK does not appear you have conflicts preventing Mongod from starting correctly. If there is no OK message then a ps –ef | grep mongo won’t show a mongod daemon running in the background.

The default output for mongod goes to /var/log/mongo/mongod.log and is set in the mongod.conf file.

**Bug:**The service daemon for Mongod is not correct. There are bugs in it. It reports the process is stated with in fact it is not. If the directory is empty, /var/lib/mongo, mongod wont start. You have to start the daemon using sudo mongod –dbpath /var/lib/mongo which will initialize the system from cold start.

E.g.



Lots of Internet posts say to remove mongod.lock to fix the no connection issue:

Macintosh HD:Users:dc:Desktop:Screen Shot 2012-08-29 at 11.33.13 PM.png

This is incorrect as seen above.

Bug: related to above. If you initialize from cold start using sudo mongod –dbpath /var/log/mongo and connect with the client:

Using the mongo CLI you can run a db.serverStatus(), these real time parameters can be output to munin.

> db.serverStatus()

{

"host" : "centos-ami",

"version" : "2.0.7",

"process" : "mongod",

"uptime" : 22,

"uptimeEstimate" : 22,

"localTime" : ISODate("2012-08-30T06:35:32.869Z"),

"globalLock" : {

"totalTime" : 22637708,

"lockTime" : 16112,

"ratio" : 0.0007117328309031992,

"currentQueue" : {

"total" : 0,

"readers" : 0,

"writers" : 0

},

"activeClients" : {

"total" : 0,

"readers" : 0,

"writers" : 0

}

},

"mem" : {

"bits" : 64,

"resident" : 14,

"virtual" : 629,

"supported" : true,

"mapped" : 0,

"mappedWithJournal" : 0

},

"connections" : {

"current" : 1,

"available" : 19999

},

"extra\_info" : {

"note" : "fields vary by platform",

"heap\_usage\_bytes" : 347984,

"page\_faults" : 0

},

"indexCounters" : {

"btree" : {

"accesses" : 0,

"hits" : 0,

"misses" : 0,

"resets" : 0,

"missRatio" : 0

}

},

"backgroundFlushing" : {

"flushes" : 0,

"total\_ms" : 0,

"average\_ms" : 0,

"last\_ms" : 0,

"last\_finished" : ISODate("1970-01-01T00:00:00Z")

},

"cursors" : {

"totalOpen" : 0,

"clientCursors\_size" : 0,

"timedOut" : 0

},

"network" : {

"bytesIn" : 80,

"bytesOut" : 92,

"numRequests" : 1

},

"opcounters" : {

"insert" : 0,

"query" : 1,

"update" : 0,

"delete" : 0,

"getmore" : 0,

"command" : 2

},

"asserts" : {

"regular" : 0,

"warning" : 0,

"msg" : 0,

"user" : 0,

"rollovers" : 0

},

"writeBacksQueued" : false,

"dur" : {

"commits" : 30,

"journaledMB" : 0,

"writeToDataFilesMB" : 0,

"compression" : 0,

"commitsInWriteLock" : 0,

"earlyCommits" : 0,

"timeMs" : {

"dt" : 3067,

"prepLogBuffer" : 0,

"writeToJournal" : 0,

"writeToDataFiles" : 0,

"remapPrivateView" : 0

}

},

"ok" : 1

}

>

Another way to check and verify if mongod is running is to do ps -ef which will list the processes running on the machine:

**[ec2-user@centos-ami ~]$ ps -ef | grep mongod**

**mongod 1261 1 0 Aug20 ? 00:00:01 /usr/bin/mongod -f /etc/mongod.conf**

**ec2-user 26587 7979 0 07:27 pts/0 00:00:00 grep mongod**

**[ec2-user@centos-ami ~]$**

1. To test the mongod daemon is functioning correctly, start the mongo client using

**>mongo**. You should see something like:

**[ec2-user@centos-ami ~]$ mongo**

**MongoDB shell version: 2.0.7**

**connecting to: test**

**>**

Like a SQL database, check you can view the databases and you can write and read data. Usually listing the databases is good enough, reading and writing data into a database should work.

**>show dbs**

**config 0.0625GB**

**local (empty)**

**>**

OK for one node it looks like everything is functioning. We can insert data using the command line interface as shown above or python to insert many records at a time.

To add/list data using the above mongo command line:

First create an object in JSON notation. JSON objects are enclosed in braces, {} and consist of key value pairs. Here is a sample record:

> record = {date:new Date('09/01/1933'),name:'a',age:'100'}

To insert the record into a database test, we don’t need to create the table or collection test, we declare it and the mongo client will create the table or collection for us automatically if it doesn’t exist. If it exists, it will check for duplicates and issue an error messsage if there is a duplciate.

>db.test.insert(record)

>db.test.find() , will display everything in the collection test in database db.

The problem with the command line interface is it doesn’t allow the user to define loops or variables like a programing language command line interface. This is an improvement in database user interfaces where users can interface to a database without having to write an ODBC program. To insert multiple rows, use pymongo.

You will have to install pymongo first if it isn’t already installed:

<http://api.mongodb.org/python/current/installation.html>

Before running the pymongo scripts below it is assumed the reader has some familiarity with python. If you are learning python with little programming background you can use the Udacity materials to learn idioms which are enough to write database administration and testing programs. There are 6 units with code, units 2-5 are useful and each unit takes about a week to do. This is a good introduction to Python : [www.udacity.com](http://www.udacity.com) cs101. There is additional practice material on Unit 2 and Unit 3 at codingbat.com. Do strings1/2 after Unit 2 and Lists1/2 after Unit 3. This approach has been tested on people with no programming background.

**Starting up a cluster:**

A mongodb cluster is one which has shards where data coming into the database is distributed across the shards. If there are 3 shards, s1, s2 and s3 then the first write would go to s1, the second to s2 and the third to s3. We should be able to see this in action as we send data to the shard.

Start a cluster on one node first; make 3 data directories for each shard and start mongod daemons on each data directory:

**mkdir –p /slave1/db**

**mkdir –p /slave2/db**

**mkdir –p /slave3/db**

**mongod –dbpath /slave1/db –logpath /slave1/log –port 10000 &**

**mongod –dbpath /slave2/db –logpath /slave2/log –port 12000 &**

**mongod –dbpath /slave3/db –logpath /slave3/log –port 14000 &**

**mongod --configsvr --dbpath /configsvr/db –logpath /configsvr/log –port 27017 &**

**The master is mongos which distributes the writes across each shard.**

**mongos –configdb localhost:10000,localhost:12000,localhost:14000 &**

(Note : MSWORD automatically creates hyphens because of autocorrect, those are double hyphens --)

All the logs above are sent to stdout. To add logs to these startup commands --logpath

**Creating a script to start the mongodb cluster**

**Script improvements:** Write a script which lists the processes so we can kill the mongo related processes before startup. The script below parses the output of a command into an array. Step through the array, strip out the process Id and kill each process before starting the mongod cluster.

#!/bin/bash

IFS='

'

ARRAY=(`ps -ef | grep mongo`)

count="0"

for i in ${ARRAY[\*]}; do

echo "$i"

done

The default shell environment for Linux distributions is bash. In your favorite text editor copy the above commands to a file and save it as startcluster. Change the permissions to 755 using sudo chmod 755 startcluster. Make sure all the mongo processes are stopped and run the script using ./startcluster.

Note: make sure there are no spaces after the commas.

Do we need a config server? yes

Start mongo shell

Use admin

Db.runCommand({“addShard”:”localhost:20000”})

What does the add shard command do? Now you can add data to localhost:27017 and you don’t get the cant find shard for db error.

**Mongodb options:**

**mongod --dbpath /master/db –bindip localhost –port 27017 –logpath /master/log –logappend true**

The ip address/port of the server is set using bindip/port. Will assume we are running mongod on the local machine.

Do you need cpu=true for the cpu utilization for MMS?

One feature of mongodb is replication which allows recovery from a single point of failure.

Configuration 2:

Setting up master/slave replication:

**Finding your data:** One of the concepts in distributed nosql databases which doesn’t exist in SQL database is the existence of a hash function. Your data is routed to some server based on a hash function. There will be scenarios where you will need to find a particular piece of data on a particular machine and you will also need to know what happens to that piece of data if nodes are added and removed either due to failure or maintenance.

In cases of replication finding data using the hash function:

**Profiling:**

**Enable Profiling db.setProfiling(Level). Stores data in mongodb. Can query this.**

**Another presentation here**

**Query explain:**

**Slide: warm start the database. (2 ways run regression tests to load dat , use dd linux command command?? Load files into memory.)**

If your data would be smaller than ram available you could try:   
something like   
db.foo.distinct( ..fieldname..) to load an index in memory   
  
or use dd to "read" all files into the os disk cache:   
  
for file in /data/db/\*; do dd if=$file of=/dev/null bs=16M; done   
  
This would be faster than mongo, since files will be read   
sequentially.

**Mongod uses memory mapped files. Files are mapped to memory.**

**Mongostat, iostat, 200-300mb on ec2.**