**Definition**

**db.fsyncLock()**

**Forces the mongod to flush all pending write operations to disk and locks the entire mongod instance to prevent additional writes until the user releases the lock with a corresponding db.fsyncUnlock() command.**

**Update packages in Ubuntu machine:**

$ Sudo apt-get update -- This will update all the latest packages

**Before installing docker install the recommended packages:**

$ Sudo apt-get install linux-image-extra-$(uname –r) linux-image-extra-virtual

**Now we can install Docker:**

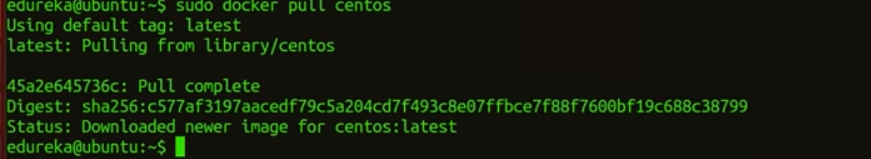
$ Sudo apt-get install docker-engine

Now start the docker engine.

$ Sudo service docker start

Now we can pull the centos image from docker hub

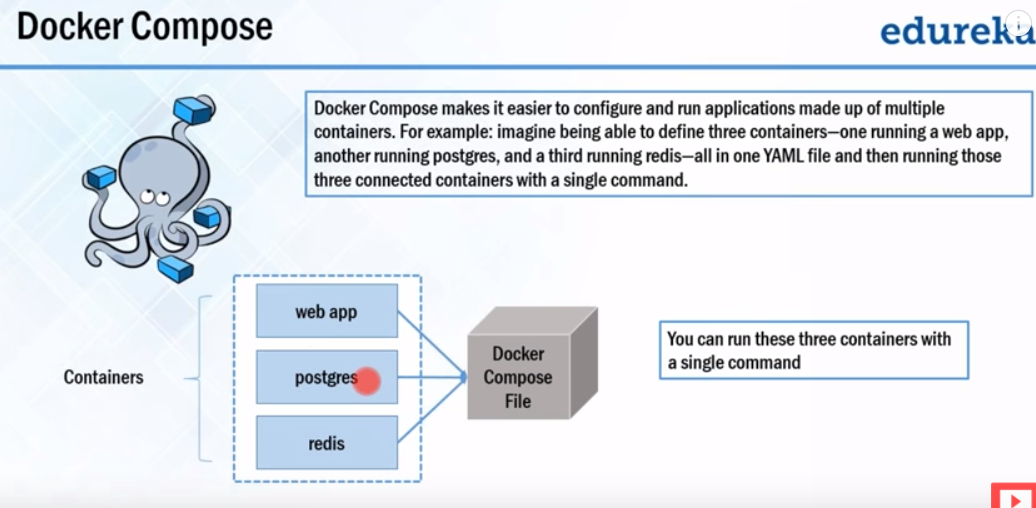
$ Sudo docker pull centos



$ Sudo docker run –it centos

This will move us into centos container.

**DOCKER COMPOSE:**



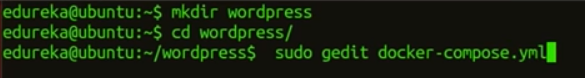
Now to install Docker-compose we need python-pip

$ sudo apt-get install python-pip

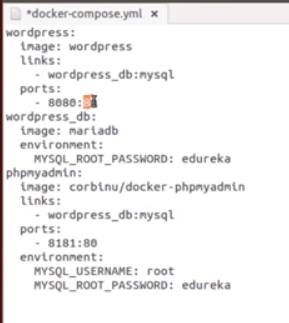
Below will install docker-compose

$ sudo pip install docker-compose

Now create a path for docker-compose.yml file and edit the docker-compose.yml file.

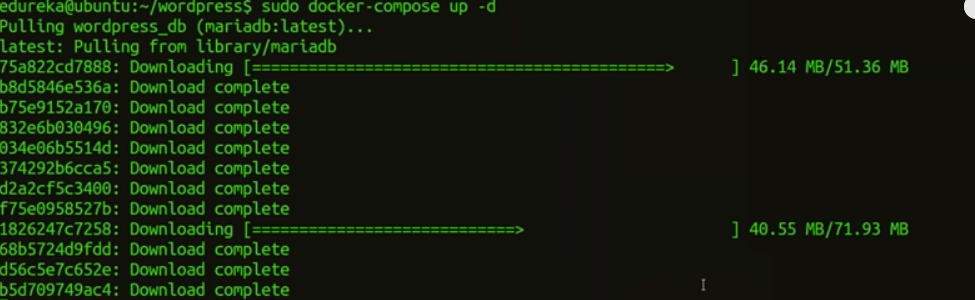


Below is an example of coker-compose.yml file



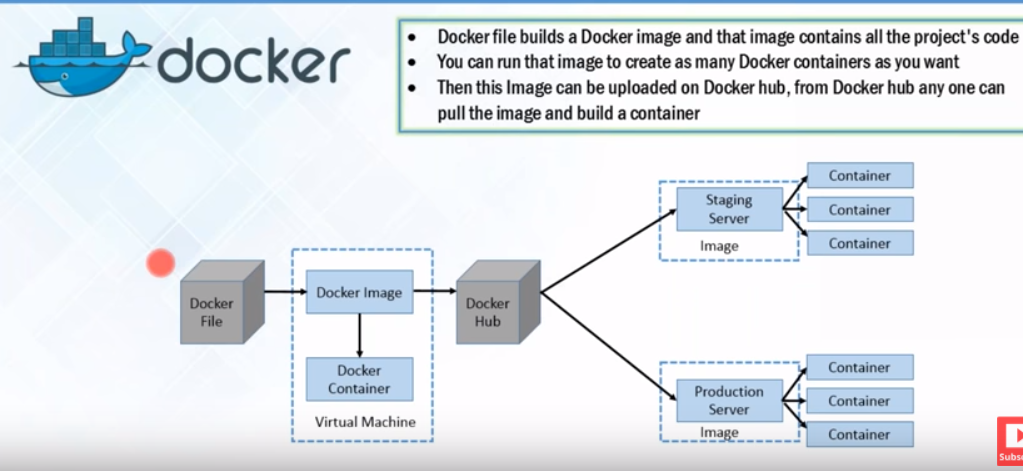
Now we can run a command to run and execute the docker-compose file.

$ sudo docker-compose up –d



After execution the docker-compose will start the application.

**DOCKER SWARM**



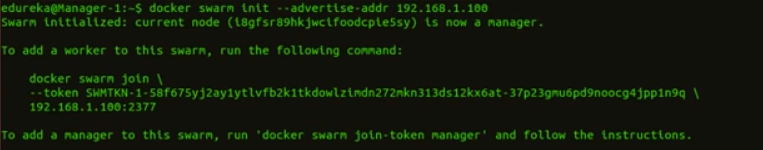
[**https://www.youtube.com/watch?v=lcQfQRDAMpQ**](https://www.youtube.com/watch?v=lcQfQRDAMpQ)

[**https://www.youtube.com/watch?v=Ceqb53EXANk**](https://www.youtube.com/watch?v=Ceqb53EXANk)

1. To create Docker swarm in Manager machine.

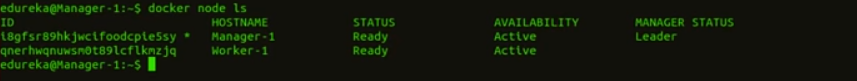
$ docker swarm init –advertise—addr 192.168.1.100

The above command will create a token and we need to copy this token and paste it in the worker/node machine so that the worker node will be added to this swarm.



To list the nodes in the swarm, issue below command:

$ docker node ls



**Now start creating our services:**

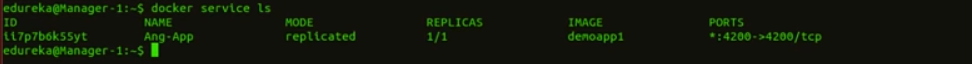
$ docker service create –name “Ang-App” –p 4200:4200 demoapp1

Where –p = browser port 4200 points to container port 4200 where the service is binded

Where –name = specifies the name of the service

Where “demoap1” specifies the name of the image

$ docker service ls



This will show us that the service is created and up and running

**At this point it is still not replicated as it is 1/1 in REPICAS**

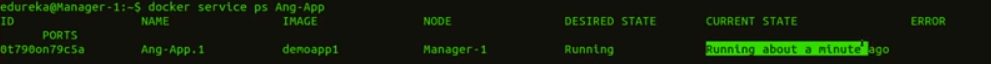
**If you go to worker-1 and then do**

**$ docker ps**



You can see that there is no container that has stated inside this node.

$ docker service ps Ang-App



This will show that the service is running in Manager node

Now the application will be running in all the 3 nodes even though it is still not replicated as all the nodes in this cluster can see this application as they are all in the same Docker Swarm.



No we need to host the service/application in multiple nodes.

$ docker service rm Ang-App

Above will remove the service Angular App so that the application stops.

$ docker service create –name “Ang-App” –p 4200:4200 –mode global demoapp1

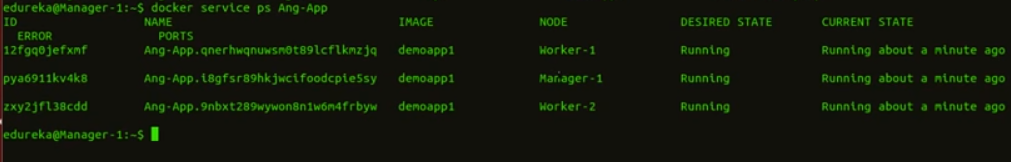
The above –global parameter will make the application service to run in all the nodes.

Now the application is hosted in all the nodes in the cluster.



Now to see the detail of the Ang-App services , issue below command:

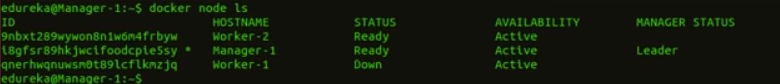
$ docker service ps Ang-App



**Now comes the concept of High-Availability**

If we want to see all the nodes status, then issue below command:

$docker node ls



As soon as the 3rd node comes up, it will automatically comes and joins the cluster

Now, if we want to run the application in only 2 of the nodes in a cluster instead of all the 3 nodes then we can issue below command where we have replaced –global mode to –replicas option.

$ docker service create –name “Ang-App” –p 4200:4200 –replicas 2 demoapp1

Now, the Manager will assign the service to 2 of the best performing nodes out of the 3 nodes.



Now if you go the 3rd node and issue

$ docker ps

It will not show any service any running in it but the application will be still up



Now Scaling up and down the services in the Docker Swarm.

TO scale up:

$ docker service scale Ang-App=5



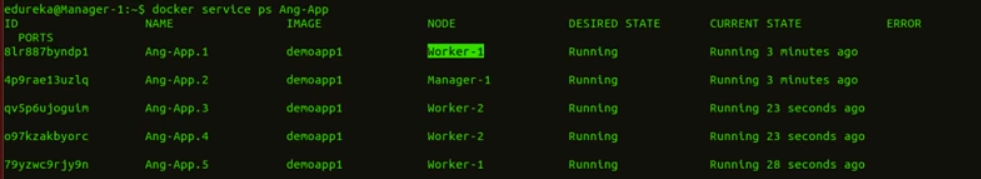
This will add 3 more services within the nodes in the cluster.

Now after some time, it will updrain

the services in all the nodes.

In order to see where the 5 services are running, we can issue below:

$ docker service ps Ang-App



Now all the services are up and running.



$ docker node update –availability drain Manager-1

Above will drain the services from Manager node and will create another service in one of the worker node

$ docker node update –availability active Manager-1

Above will enable the services in Manager node

**USERS AND ROLES:**

**Revoke pvileges from a USER**

[**https://youtu.be/ufV8W2-4wE8**](https://youtu.be/ufV8W2-4wE8)

**CLUSTER SETUP:**

**Add Other members in Relica set:**

$> rs.initiate({\_id: ‘myreplset’, version: 1, members: [{\_id: 0,host: ”localhost:27017”},{\_id: 1, host: ”localhost:27018”}, {\_id: 2, host: ”localhost:27019”}]});

**RECONFIGURE:**

**To reconfigure Replica set for slave delay:**

ABC:PRIMARY>var cfg = rs.conf()

ABC:PRIMARY>cfg and then press enter

ABC:PRIMARY>cfg.members[2].slaveDelay = true;

ABC:PRIMARY>cfg.members[2].slaveDelay = 8\*3600; -- This will make the slave with 8 hours delay

ABC:PRIMARY>cfg.members[2].hidden = true; -- This will make client not to read from this database by mistake as it is in delayed mode.

**Below will apply the reconfig to the replica set**

ABC:PRIMARY>rs.reconfig(cfg);

**To make a member not a PRIMARY meaning setting priority to Zero:**

**It will sti vote and it is still Secondary but it will not become primary unless we manually make it primary:**

ABC:PRIMARY>cfg.members[2].priority = 0;

ABC:PRIMARY>cfg and then press enter

ABC:PRIMARY>rs.reconfig(cfg);

Now we need to bring down this member to make the reconfig takes place effectively

ABC:PRIMARY>rs.isMaster();

It will not list the 3rd member in above command as it is hidden and even after we bring this member up it will not show.

Issue below command it make sure it is slave

ABC:PRIMARY>rs.slaveOk();

To connect other server form 1st node, we can do below:

ABC:PRIMARY>help connect

ABC:PRIMARY>var server2 = new Mongo(‘localhost:27002’);

Make secondary node as slave from first node

ABC:PRIMARY>server2\_test.getMongo().setslaveOk();

ABC:SECONDARY> (new Mongo(‘localhost:27001’)).getDB(‘test’).foo.count();

To find which node that you are connected to:

ABC:SECONDARY>rs.isMaster().me;

ABC:SECONDARY>server3\_test.isMaster().me;

ABC:SECONDARY> server3\_test.isMaster().me;

To check oplog for recent activity:

ABC:SECONDARY> db.oplog.rs.count();

ABC:SECONDARY> db.oplog.rs.find().sort({$natural:-1}).limit(5);

**VOTING:**

In General, Avoid using Votes .

1. If it is a 2 node cluster like one primary and one secondary then if primary goes down failover doesn’t happen as there is no majority votes , In that case use an Arbiter.
2. Else use a Vote option of 0 to Secondary and 1 to primary so that when primary goes down then we can manually switch over to secondary instead of automatic failover.

**PRIORITY:**

In General, Everyone gets a priority of ‘1’, ‘0’ indicates ‘never Primary’. Higher the number, higher is the priority of the server.

1. You can even set priority of 0.5 so that they get the least preference to become Primary.
2. IF we want to forcefully make the secondary as primary due to some network slowness or iues with Primary , then set the priority of secondary node to higher value of ‘2’ and then apply reconfigure so that the secondary becomes the new Primary now.

ABC:SECONDARY>var cfg = rs.conf()

ABC: SECONDARY >cfg and then press enter

ABC: SECONDARY >cfg.members[2].priority = <n>;

ABC: SECONDARY >cfg.members[2].arbiterOnly = true ;

**Below will apply the reconfig to the replica set**

ABC: SECONDARY>rs.reconfig(cfg);

**ARBITER:**

Arbiter is not a data bearing node, it is added for Voting purpose only to make sure who becomes primary in a 2 node cluster and by adding the arbiter to the cluster we artificially make it look like a 3 node cluster.

After the hostname in the config CFG parameter file, add option for arbiterOnly = true.

ABC: SECONDARY >cfg.members[3].arbiterOnly = true ;

**Below will apply the reconfig to the replica set**

ABC: SECONDARY>rs.reconfig(cfg);

**WRITE CONCERN (or) CLUSTER WIDE COMMIT:**

In order to make a cluster wide commit, we need to have below transaction setup so that a committed transaction happens on all the nodes in the cluster setup or a minimum of 1 secondary so that the transaction is not lost.

[

db.foo.insert({x:abc});

db.getLastError({w:’majority’,wtimeout:8000})

]

Where ‘w’ can have value of 1,2,3,4 or majority or all.

1 meaning acknowledgment from 1 server and 2 means from 2 servers, ‘majority’ or ‘all’ means from all servers.

‘wtimeout’ means timeout error message and it is in milliseconds.

To set write concern in Config level:

Cfg = rs.conf();

Cfg.settings.getLastErrordefaults = { w: “majority”,wtimeout = 5000}

Rs.reconfig();

**IOPS (Input output units per second) :**

IOPS is used in 2 places

1. Load relevant indexes entries from DISK to memory
2. Retrieve documents from DISK to memory

Wiredtiger has a cache memory which stores almost roughly 50% of the RAM to store indexes and documents from file system.

Disk access is needed only if indexes or documents are not in cache.

Working set = Indexes plus frequently accessed documents

If RAM greater than working set then reduced IO.

Working set < (less than) RAM < (less than) data Size.

If there is a query to retrieve 100 documents from DISK then it is 100 IOPS, if we have 10 prallel queries to retrieve 100 documents each then it is 1000 IOPS /second

For Insert:

1. Write document to DISK
2. Update each index file

IOPS = 1 + # of indexes.

For Delete:

1. Navigate In-Memory Indexes
2. Mark document deleted
3. Update each index file

IOPS = 1 + # of indexes.

For Update: meaning a delete + Insert

1. Navigate In-Memory Indexes
2. Mark document deleted
3. Insert new document version
4. Update each index file

IOPS = 2 + # of indexes.

Simplified Model:

The Sizing can be determined in below logical views:

1. Working set
2. Checkpoints
3. Document size relative to block size
4. Indexed arrays
5. Journal,log

The Journal,log should be in separate file system to reduce IOPS else we need to account for IOPS for this as well.

[**http://grokbase.com/t/gg/mongodb-user/127fvyax31/understanding-mongo-logging-and-paths-of-the-log-files**](http://grokbase.com/t/gg/mongodb-user/127fvyax31/understanding-mongo-logging-and-paths-of-the-log-files)

**CHECKPOINTS:**

When mongodb writes the document m the document is written in the memory first rather than in the DISK.

Typically 60 seconds or 2GB whichever comes first and then only it will write to DISK until then all operations will be on Memory.

Now, Estimate sizing based on Simplified Model:

--RAM

--CPU

--Disk Space

--IOPS

Then adjust based upon working set, checkpoints, etc.

Design sharded cluster that provides these totals.

**SHARDING:**

<https://youtu.be/xvWzS9j7IIY>

<https://youtu.be/K7nWCuA5wF4>

<https://www.youtube.com/embed/v7Gibss9JAk?showinfo=0&amp;autoplay=1&amp;autohide=1&amp;modestbranding=1&amp;rel=0&amp;enablejsapi=1>

<https://youtu.be/v7Gibss9JAk>

<https://youtu.be/XMnxjcdiMJs>

<https://youtu.be/ZyDNnN8-4Ak>

<https://youtu.be/lQ-tf4XQlQ8>

<https://youtu.be/WVXmIVUGBm4>

<https://youtu.be/-Ijt60KkWpc>

<https://youtu.be/G2GYYDTfq-M>

<https://youtu.be/DSAtoCcFdWI>

<https://youtu.be/FANoi-xQJp4>

<https://youtu.be/ujlNVJK5dMc>

<https://youtu.be/j43kGempShk>

<https://youtu.be/rYGOF-MjRQU>

<https://youtu.be/9QXa9Uz8bjc>

Creating a Shard:

1. Create all the data directory for the shards:

Example : mkdir a1

Mkdir a1

Mkdir b1

Mkdir b2 etc.

1. Make data directory for the config servers to store the metadata of the shard.

Mkdir cfg0

Mkdir cfg1

Mkdir cfg2

1. Setup config servers:

Mongod –configsvr –dbpath cfg0 –port 26050 –fork –logpath log.cfg0 –logappend

Mongod –configsvr –dbpath cfg1 –port 26051 –fork –logpath log.cfg1 –logappend

Mongod –configsvr –dbpath cfg2 –port 26052 –fork –logpath log.cfg2 –logappend

Setup in a replica set option:

mongod --configsvr --replset csReplset--dbpath cfg0 --port 26050 --fork --logpath log.cfg0 --logappend

mongod --configsvr --replset csReplset--dbpath cfg1 --port 26051 --fork --logpath log.cfg1 --logappend

mongod --configsvr --replset csReplset--dbpath cfg2 --port 26052 --fork --logpath log.cfg2 --logappend

1. Setup shard servers

Mongod –shardsvr –replset a –dbpath a0 –logpath log.a0 –port 27000 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset a –dbpath a1 –logpath log.a1 –port 27001 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset a –dbpath a2 –logpath log.a2 –port 27002 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset b –dbpath b0 –logpath log.b0 –port 27100 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset b –dbpath b1 –logpath log.b1 –port 27101 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset b –dbpath b2 –logpath log.b2 –port 27102 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset c –dbpath c0 –logpath log.c0 –port 27300 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset c –dbpath c1 –logpath log.c1 –port 27301 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset c –dbpath c2 –logpath log.c2 –port 27302 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset d –dbpath d0 –logpath log.d0 –port 27400 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset d –dbpath d1 –logpath log.d1 –port 27401 –fork –logallfiles—oplogsize 50

Mongod –shardsvr –replset d –dbpath d2 –logpath log.d2 –port 27402 –fork –logallfiles—oplogsize 50

Oplogsize should be a big sze for PROD.

1. Create MONGOS processes in clint or host machines, this is where the clients would connect to:

Mongos –configdb 10gen.local:26050, 10gen.local:26051, 10gen.local:26052 –fork –logappend –logpath log.mongos0 -- This will say to connect to MONGO to default port 27017

Mongos --configdb 10gen.local:26050, 10gen.local:26051, 10gen.local:26052 –fork –logappend –logpath log.mongos1 –port 26061

Mongos –configdb 10gen.local:26050, 10gen.local:26051, 10gen.local:26052 –fork –logappend –logpath log.mongos2 –port 26062

Mongos –configdb 10gen.local:26050, 10gen.local:26051, 10gen.local:26052 –fork –logappend –logpath log.mongos3 –port 26063

CLUSTER TOPOLOGY:

Cluster setup:

* How many shards initially = 4
* Replication factor RF = 3
* Then 3\*4 = 12 mongod processes
* 4 mongos processes
* 3 config servers

CONFIG DB :

When you connect to default port 27017 , you will connect to MONGOS

When you connect to default port it will automatically connect to MONGOS and the shell promt would show you as MONGOS>

Once we are inside MONGOS, create congig database

MONGOS> use config

It will automatically create the confg database and it will show al below:

MONGOS> show collections

Chunks

Databases

Lockpings

Locks

Mongos

Settings

Shards

System.indexes

Version

MONGOS> db.chunks.find()

MONGOS> db.shards.find()

There will be zero shards

MONGOS> sh.help()

Above will help shards command options

In order to add a shard to a cluster, we need to wait for the members to replicate for that you need to check rs.status() , once the members are recovered and made secondary then you can start adding the shards for the primary.

MONGOS> sh.addShard(“a/hostname:27000”); -- This will add shard to the cluster replicaset abc

MONGOS> sh.status(); -- This will show the shard status

After you run the addshard ‘add\_shard\_json.txt’ in bulk script, it will ask you to add more shards to that replica set:

MONGOS> ourAddShard(“b”,27100); --- WHERE ‘b’ IS SHARD NAME that you are adding to the replica set

MONGOS> ourAddShard(“c”,27200); --- WHERE ‘c’ IS SHARD NAME that you are adding to the replica set

MONGOS> ourAddShard(“c”,27300); --- WHERE ‘a’ IS SHARD NAME that you are adding to the replica set

MONGOS> sh.status();

MONGOS> db.shards.find(); -- This will show you all the shards in the cluster.

SHARDING A collection “

Unshared collection will reside fully in first shard or shard0 in the cluster

First thing that we need to do is turn on sharding for the database.

MONGOS> sh.enableSharding(“dbname”);

MONGOS> sh.status();

You will see the database status as “partitioned” = true.

Now we need to shard the collection within the database.

MONGOS> sh.shardCollection(“dbname.collectionname”,{\_id:1},true);

Above will shard the collection into the shards that we created like a,b and c

MONGOS> Collection.stats()

Above will give you the chunk and shard details for that collection.

**Working with a Sharded Custer:**

MONGOS> db.example.getIndexes();

You can see the shard key in the indexes.

Now create a new shard key .

MONGOS> db.example.createIndex({e:1,f:1,a:1}); -- Where e,f,a are shard names.

TO see stats for a query: -- To check a targeted query

MONGOS> var exp= db.people.explain('executionStats');

To find a document using shard key – Targeted Query

MONGOS> exp.find({b:5,c:5,d:5});); --- if your query has fields that exactly matches the shard key then it will directly find that document by going to that shard without even going to check in other shards, this is called Target query.

For example: if shard key is MONGOS> db.example.createIndex({e:1,f:1,a:1});

And if query is below:

exp.find({e:5}); then this a targeted query as it used the shard prefix.

exp.find({f:5}); then this a scattered gather query as it did not use either prefix nor the exact shard key.

To find a document using index key – Scatter gather Query

MONGOS> exp.find({a:5,e:5,f:5});); --- if your query has fields that exactly matches the shard key but in different order then it will use that shard key as index and it will scan through all the shards to find that document rather than the exact shard and it is expensive.

**SHARD KEY SETUP:**

**SECURITY:**

Types of Authentication:

1. SCRAM-SHA-1,MONGODB-CR(deprecated as of MONGO 3.0) --Mongodb challenge/response ( authenticating via username/password)
2. X509 / SSL – It is a certificate based TLS support connection.

Below available only in Enterprise model and above in community model

1. Kerberos – This is available only in paid version
2. LDAP – Light weight access protocol -- This is available only in paid version

--auth is basic thing to tun on security and it can be done in config file or in the mongod or mongos startup process as well.

Admin database:

It is a reserved database used for administrative purpose.

There is database.system.users collection—this stores all the users ad roles for that users info.

Db.help() – This will help to see all top level helper methods commands used.

Db.createUser({“user” : “gopi” ,”pwd” : “Password123” ,”roles”: [ “userAdminAnyDatabase”]});

Now above will just create the user in admin database and then we need to authenticate it.

$ mongo localhost/admin –u username –p password

Above will connect to the database using the username that we just created but it cannot read or write to the DB , it cn be used only to administer the database.

Create below user in order to read or write to any database:

Db.createUser({“user” : “will” ,”pwd” : “Password123” ,”roles”: [ “readWriteAnyDatabase”]});

Create below user in order to read or write to current/one database:

Db.createUser({“user” : “andrew” ,”pwd” : “Password123” ,”roles”: [ “readWrite”]});

Db.system.users.find() – This will display all the users in database.

Db.auth(‘kirby’,’password’);

$ Mongo admin –u username –p password

To make authorization enabled through config file , we can add authorization key value in config file.

$ vim config

1 security:

2 authorization: ‘enabled’

$ mongod –config config -- This will use the mongod to use the config file while starting up

**Enabling x509 Authentication:**

There is no difference between enabling tls and enabling x509 authentication because x509 authentication is enabled by default ,we just need to set the server to use TLS.

Now create the certificate files:

Public certificate: ca.pem – created on mycertificateauthority

And then create pem files for client and server

Client pem file : client.pem

Server pem file : server.pem

Each of the above has the public certificate and the corresponding private key.

$ mongod –sslMode requireSSL –sslPEMKeyFile server.pem –sslCAFile ca.pem --auth

--client.pem file – This file has a private key and public certificate, inside that public certificate, there is a subject line which describes who is the owner of this certificate is rather to whom the certificate was issued to.

Let’s see the subjectline inside the client.pem file

$ openssl x509 –in client.pem –inform PEM –subject –nameopt RFC2253 –noout

Above will give the details of the client.

subject= C=US, ST=New York, L=New York City, 0=MongoDB, OU=KernelUser, CN=Client

copy the above subjectline and we will create the above as a client user account in the database to identify the client

Now, connect to the server through a mongo shell and to identify the user in the server database

$ mongo –ssl --sslPEMKeyFile client.pem –sslCAFile ca.pem

**X509 is an external authentication mechanism because the credentials aren’t stored directly in mongodb database and that is why we need to create this client certificate as a user in the DB to authenticate.**

$ db.getSiblingDB(“$external”).runCommand({createUser: “C=US, ST=New York, L=New York City, 0=MongoDB, OU=KernelUser, CN=Client”, roles[{role” ‘root’,db” ‘admin’}]})

**After creating the user, we need to authenticate the user.**

$ db.getSiblingDB(“$external”).auth({User: “C=US, ST=New York, L=New York City, 0=MongoDB, OU=KernelUser, CN=Client”, mechanism: “MONGODB-X509”});

Above will authenticate the user.

**X509 Clfuster setup:**

You need to mention the alternative name on the certificate for each member of the replica set:

In order to find alternateive name use below command:

openssl x509 –in server.pem –text

**Below worked:**

[**https://docs.mongodb.com/manual/tutorial/configure-ssl/**](https://docs.mongodb.com/manual/tutorial/configure-ssl/)

openssl req -newkey rsa:2048 -new -x509 -days 365 -nodes -out mongodb-cert.crt -keyout mongodb-cert.key

cat mongodb-cert.key mongodb-cert.crt > mongodb.pem

mongod --replSet myReplSet --port 57017  --sslMode requireSSL --clusterAuthMode x509 --sslPEMKeyFile mongodb.pem --sslCAFile mongodb-cert.crt

Ca.pem = certificate Authority public key.

Client.pem = key file for the client (This includes Client private key and server public certificate which shows to which server you want to authenticate to).

openssl x509 -in client.pem -inform PEM -subject -nameopt RFC2253 –noout

**Example:**

openssl x509 -in dfrdbsd10vl.state.in.us.pem -inform PEM -subject -nameopt RFC2253 –noout

Member.pem = certificates for each member of the replica set

**1st member**

$ mongod –replSet myReplSet –dbpath ./rs1/db –logpath ./rs1/mongod.log –port 27017 –fork –sslMode requireSSL –clusterAuthMode x509 –sslPEMKeyFile member1.pem –sslCAFile ca.pem

**2nd member**

$ mongod –replSet myReplSet –dbpath ./rs2/db –logpath ./rs2/mongod.log –port 27017 –fork –sslMode requireSSL –clusterAuthMode x509 –sslPEMKeyFile member2.pem –sslCAFile ca.pem

**3rd member**

$ mongod –replSet myReplSet –dbpath ./rs3/db –logpath ./rs3/mongod.log –port 27019 –fork –sslMode requireSSL –clusterAuthMode x509 –sslPEMKeyFile member3.pem –sslCAFile ca.pem

We can also start this by using Config file

Congig file looks like below:

Vi config.yaml

1 security

2 clusterAuthMode x509

3 net

4 ssl

5 mode requireSSL

6 CAFile ca.pem

7 clusterFile member1.pem

Then start member using config file

$ mongod –config confog.yaml –replSet myReplSet –dbpath ./rs3/db –logpath ./rs3/mongod.log –port 27019 –fork

**Enabling TLS/SSL within Replicaset:**

[**https://youtu.be/k5N5dQTs\_E0**](https://youtu.be/k5N5dQTs_E0)

[**https://youtu.be/vmXPFUQsoFE**](https://youtu.be/vmXPFUQsoFE)

[**https://youtu.be/lB6kbRCq3HU**](https://youtu.be/lB6kbRCq3HU)

[**https://youtu.be/MNfbI81FUr0**](https://youtu.be/MNfbI81FUr0)

[**https://youtu.be/TI\_2jMfyl5A**](https://youtu.be/TI_2jMfyl5A)

$ mongod –replSet myReplSet –dbpath ./rs1/db –logpath ./rs1/mongod.log –port 27017–fork –sslMode requireSSL –clusterAuthMode x509 –sslPEMKeyFile server.pem –sslCAFile ca.pem

$ mongod –replSet myReplSet –dbpath ./rs3/db –logpath ./rs3/mongod.log –port 27018–fork –sslMode requireSSL –clusterAuthMode x509 –sslPEMKeyFile server.pem –sslCAFile ca.pem

$ mongod –replSet myReplSet –dbpath ./rs3/db –logpath ./rs3/mongod.log –port 27019–fork –sslMode requireSSL –clusterAuthMode x509 –sslPEMKeyFile server.pem –sslCAFile ca.pem

**We can use client to connect to server with or without Client CAfile**

$ mongo –ssl –sslPEMKeyFile client.pem—sslCAFile ca.pem

Above will allow client to authenticate but it will never say if the client is a CA certified client

$> rs.initiate({\_id: ‘myreplset’, version: 1, members: [{\_id: 0,host: ”localhost:27017”},{\_id: 1, host: ”localhost:27018”}, {\_id: 2, host: ”localhost:27019”}]});

$ openssl x509 –in server.pem –text

Above will let you know the servername from the server certificate, look for “subject Alternative name” to get the hostname to initialize the members of the replica set.

**Preferssl Vs requiressl**

Preferssl will allow clients to connect without ssl but members will have to authenticate through ssl.

$ mongod –sslMode preferSSL –sslPEMKeyFile server.pem –sslCAFile ca.pem –replSet myReplSet –dbpath=/rs3/db –logpath=/rs3/mongod.log –port 27019–fork

Above is an example of preferSSL mode, we can add other members and initiate them and then we can connect from client without ssl certificate and it should connect.

**DATA AT REST Encryption:**

We can do this in 2 ways

1. Data at rest encryption at storage level
2. Data at rest encryption at Application level

Create keyfile using below to encrypt the database

$openssl rand –base64 32 > mongodb-keyfile

Change permission to the key file so that mongod can use it

$chmod 600 mongodb-keyfile

Below will encrypt the database

$mongod –enableEncryption –encryptionKeyFile mongodb-keyfile

[**https://youtu.be/wtuPTXhDNZs**](https://youtu.be/wtuPTXhDNZs)

[**https://youtu.be/sdJ5bKsgmqc**](https://youtu.be/sdJ5bKsgmqc)

**Below is using KMIP server:**

[**https://youtu.be/sdJ5bKsgmqc**](https://youtu.be/sdJ5bKsgmqc)

$mongod –enableEncryption –kmipServerName localhost

--kmipServerCAFile ca.pem –kmipClientCertificateFile client.pem

**LDAP Authentication:**

**Enabling LDAP:**

[**https://youtu.be/HxgmVjRmx9s**](https://youtu.be/HxgmVjRmx9s)

**LDAP Authorization Intoduction:**

[**https://youtu.be/CLLVbbDeAhM**](https://youtu.be/CLLVbbDeAhM)

**LDAP Authorization steps:**

[**https://youtu.be/apxTmqSMcg0**](https://youtu.be/apxTmqSMcg0)

**LDAP Authorization user transformations:**

[**https://youtu.be/\_xDwx\_ngHXE**](https://youtu.be/_xDwx_ngHXE)

**LDAP Authorization config options:**

[**https://youtu.be/pu\_Z6Mn-vG8**](https://youtu.be/pu_Z6Mn-vG8)

**MONGO LDAP**

[**https://youtu.be/gUMz4ZAE7rA**](https://youtu.be/gUMz4ZAE7rA)

**LDAP Authorization setup**

[**https://youtu.be/cuR6pyjJnsw**](https://youtu.be/cuR6pyjJnsw)

**LDAP Authorization miscellaneous:**

[**https://youtu.be/Ue\_kBqkeFWU**](https://youtu.be/Ue_kBqkeFWU)

**Kerberos Authentication:**

**Enabling Kerberos:**

[**https://youtu.be/FGmvAZZr4VM**](https://youtu.be/FGmvAZZr4VM)

**Enabling Internal Authentication:**

[**https://youtu.be/x9JaxAAIV0c**](https://youtu.be/x9JaxAAIV0c)

**MONGO IMPORT**

mongoimport

mongoimport --help

mongoimport --db pcat --collection products < products.json

mongoimport -d pcat -c people --file c:\Data\people.JSON --type JSON

mongoimport --db pcat --collection people < --file c:\Data\people.jsonexplain

**Types of roles:**

Read,readWrite ,dbAdmin ,userAdmin ,clusterAdmin ,readAnyDatabase ,readWriteAnyDatabase,userAdminAnyDatabase ,dbAdminAnyDatabase

Encryption:

Scons –ssl – we need to build it ourselves.

<https://docs.mongodb.com/manual/tutorial/configure-ssl/>

Create key file. This key file is used by servers within the cluster to authenticate on eanother, it does not encrypt the data at rest or at transit, all it does is to make sure the servers with in the cluster are trstes servers by authenticating among themselves through the keyfile.

$ touch keyfile

$ chmod 600 keyfile

$ ls –la keyfile

$ openssl rand –base64 60 >> keyfile

Just start all the mongod with in a cluster by adding --keyfile option in them

$ mongod –keyfile keyfile –replset z

<https://youtu.be/BAvTSOJGynA>

**BACKUP and RESTORE:**

**Simple backup and restore:**

$ mongodump – port 27017 –db dbname –out E:\backups

Mongodump –port 27017 –out /backups

We need to restore the DB in below new path database

$ mongod – port 27018 --dbpath E:\DB\Data

Now do the restore

$ mongorestore – port 27018 E:\backups

**mongodump --authenticationDatabase admin --username gtwsysbrootadmin --password 'Pa$$word12345678!' --host achsysb --out /backupdata**

**mongorestore --authenticationDatabase admin --username gtwsysbrootadmin --password 'Pa$$word12345678!' /backupdata**

--mongodump –oplog

Shutdown the Secondary node. Connect to mongo shell and run

|  |  |
| --- | --- |
| 1  2 | use admin  db.shutdownServer() |

Restart theoplog

node with different port and without replSetName option.

Take backup of the current oplog collection. This step is only for precaution.

|  |  |
| --- | --- |
| 1 | mongodump --db local --collection 'oplog.rs' --out /path/to/dump |

Recreate the oplog collection with new size (In this case we are creating 80GB oplog)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | use local  db.temp.drop()  db.temp.save( db.oplog.rs.find( { }, { ts: 1, h: 1 } ).sort( {$natural : -1} ).limit(1).next() )  db.oplog.rs.drop()  db.runCommand( { create: "oplog.rs", capped: true, size: (80 \* 1024 \* 1024 \* 1024) } )  db.oplog.rs.save( db.temp.findOne() ) |

--mongorestore –oplogreplay 🡪 this will do th PIT restore.

-- filesystem snapshot

- db.fsyncLock() 🡪 This will prevent wite to the system like put the entire MongoDB directory filesystem in write suspended mode and then we can take a filesystem snapshot. We will do this only if we don’t have a way to snapshot the true volume like all subvolumes.

Then we can put the DB back to write mode by db.fsyncUnLock() command.

--backup from secondary

--shutdown,copy file and then restart

**SHARD BACKUP:**

In a Sharded Cluster, we need to backup each of the shards in the replica set and then also backup the config srver.

1. First step in backup is turnoff the balancer because we do not want any chunk migration or metadata migration in config servers.

In the shell sh.stopBalancer(), this might take some time as it needs to complete migration.

1. Backup config databases

– This can be done by using mongodump –db configdb

--another way is to stop one of the secondary config server and then copy its files.

c) backup each of the shard in the replica set. – Here you wil backup each shard in the replica set using file copy technique.

d) sh.startBalancer() – This will start back the Balancer

Additional Resources:

For Docs – refer mongodb.org

--driver docs

--support forums

--mongodb-user in google groups

--IRC

--github.com source code

--blog.mongodb.org

**Hardware tips:**

--Disable NUMA – Non Uniform Memory access

--Fast CPU Clock is better than number of Cores.

-- RAM is good

--64 bit is good

--SSDs are good

--check readahead – This should be in sectors and not in bytes in unix/linux settings.

The readahead should be smaller value if there is going to be lot of Reads and writes like Random IO.

In linux, check it with blockdev—report.

Mongodb.org/Docs/ProductionNotes – check notes for this setup.

32/64 GB of RAM with SSD is good.

**Repairing MONGODB**

First backup the mongodb form dbpath /data/db

mongod --dbpath /data/db --repair

Above will repair the MONGODB standalone DB in the event of a unclean shutdown

**PERFORMANCE:**

**INDEXES:**

db.collection.createIndex({a:1});

db.collection.ensureIndex({a:1});

To create composite index:

db.people.createIndex({"first\_name": 1, "address.state": -1, "address.city": -1, "ssn": 1});

To see which indexes used in collection:

db.people.getIndexes();

To drop an index:

db.people.dropIndex("people\_indx1");

Unique key :

db.collection.ensureindex({a:1},{unique:true});

sparse key : to build keys only for fewer occurrence of name value pair

db.collection.ensureindex({a:1},{sparse:true});

TTL Index: to Delete a document which are untouched for a longer time of seconds

db.collection.createindex({"lastmodifieddate":1,expiredafterseconds: 3600});

Text Index: search by test

db.collection.ensureindex({words : "text"});

Text Index: search by text variable

db.collection.find({$words : { $search : "trees cat" }} ,{score:{$meta:"text score"} ,\_id=0 });

T0 see explain plan:

var exp= db.people.explain('executionStats');

exp.find({ "first\_name": { $gt: "J" } }).sort({ "address.city": -1 })

**To get Executionstats:**

db.collection.find({endstation\_name : "Milroy Walk,south Bank" }).explain(‘executionStats’);

**Resource allocation for Indexes:**

1. Determine Index Size:

**Performance on Sharded Cluster:**

<https://youtu.be/O3pF4Rj4MmM>

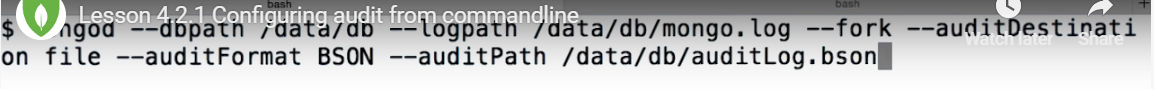
<https://youtu.be/fLe41bWKmro>

**AUDITING:**

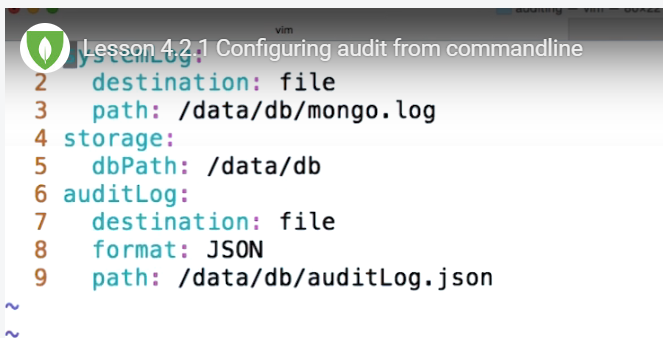
Auditing can record 4 different categories of operations.

1. We can record schema modifications or DDL
2. We can audit authorizations and authentications
3. We can audit management and configurations operations used to configure the sharded cluster and replica sets
4. We can audit all the CRUD operations

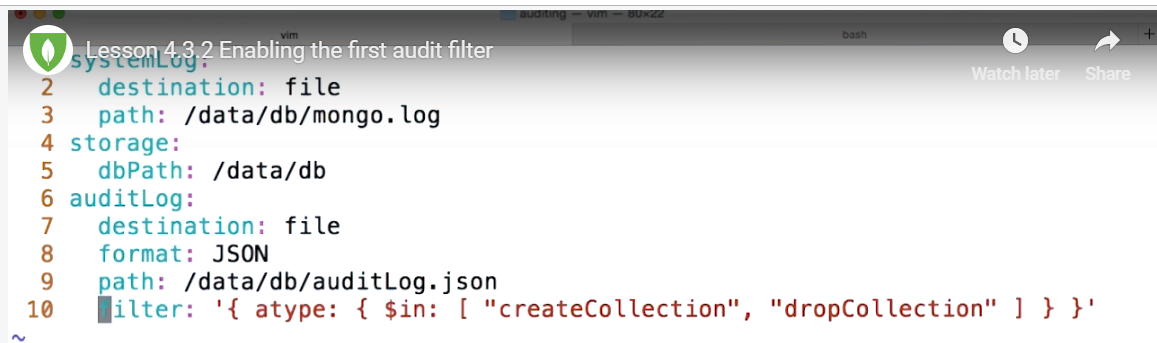
To write Audit to a file on a separate path.



To COnfig file:



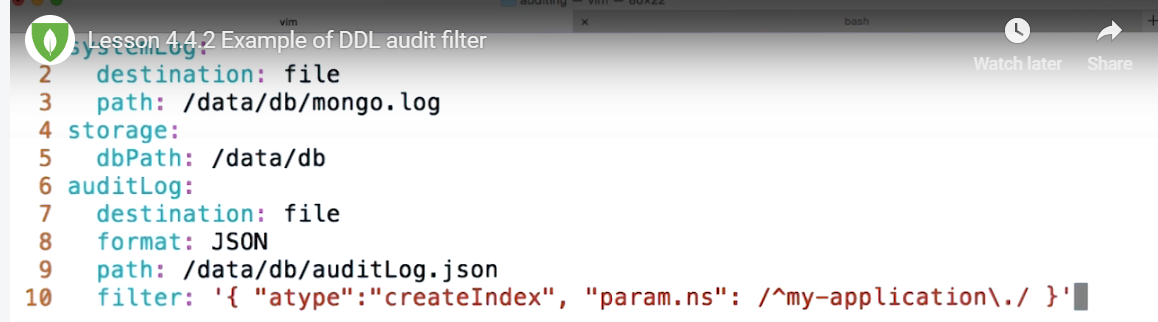
Add filter for which you want to Audit in Config file or as arguments:



Or you can pass it as an argument in command line with filter string along it like below:



To see objects getting created on a particulat database by using regex filter where it will collect audit information only about that particular database:



To Enable CRUD operaions in Audit Log:

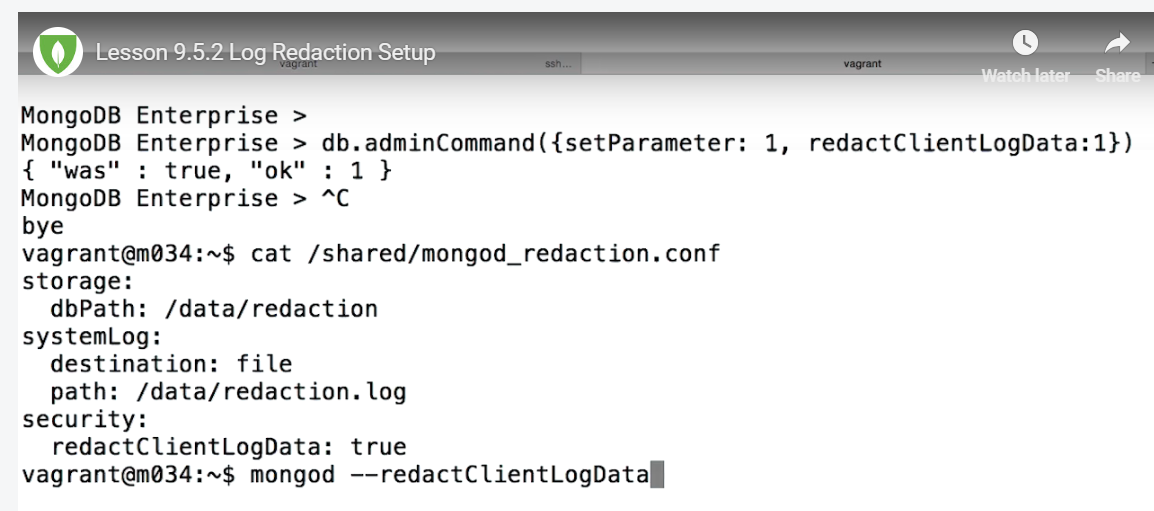


Redact Logs:

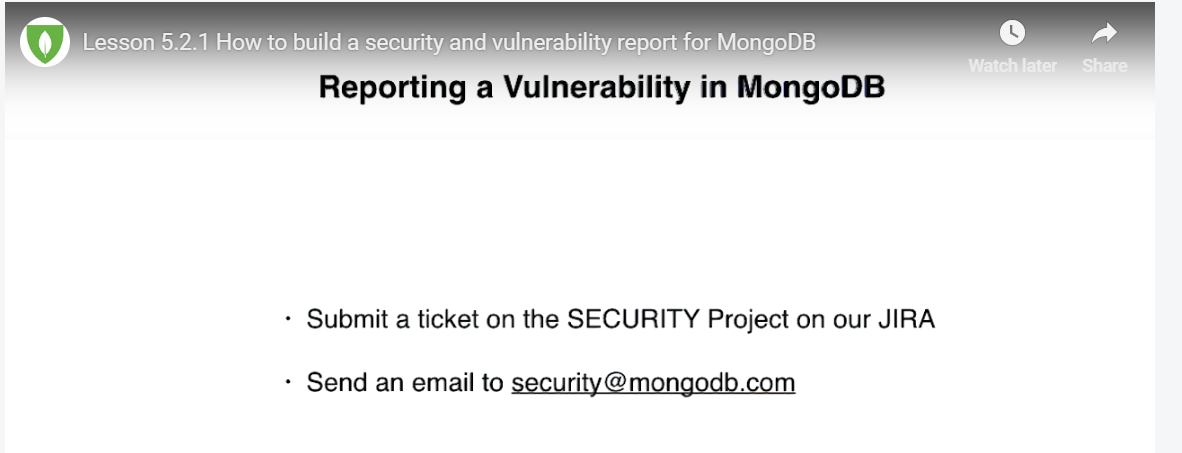
Telling mongod not to collect sensitive information like PII data in the logs:

You can add this a aparameter in Config file or as an argument.

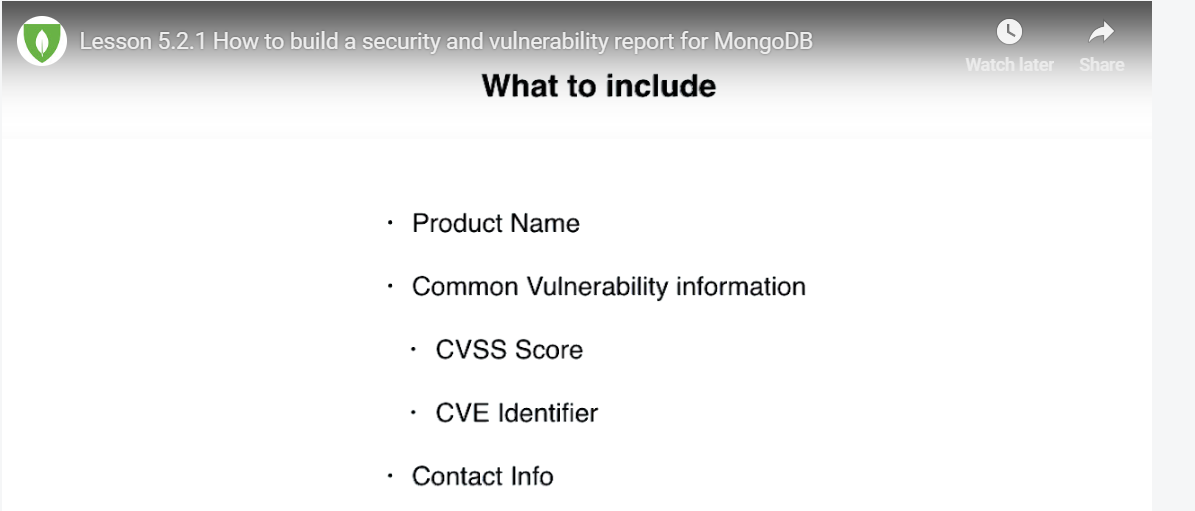
If you pass it as an agument you need to set it to true each and every time you restart the mongod.



Vulnerability:



Send the report to above link and it will ne answered in 48 Hours.



Send report in above format where CVSS is scale of the issue from 1 to 10.

CVE identifier is any links for this issue from online.

<https://youtu.be/HEWGTBmLUmU>

<https://youtu.be/g0i0txbGRwI>

<https://youtu.be/4vfhYn5BDAc>

<https://youtu.be/HIbMlZLRK3k>

<https://youtu.be/2y1RpcKbW-s>

<https://youtu.be/k3V_LpIIZ_Y>

<https://youtu.be/M9l5vClC6nU>

<https://youtu.be/oZPg7u02hfg>

<https://youtu.be/ud2HTzE7dTk>

<https://youtu.be/wSkkDSDwzS8>

<https://youtu.be/hDPl-tlCeSY>

<https://youtu.be/4AU0DmWjSac>

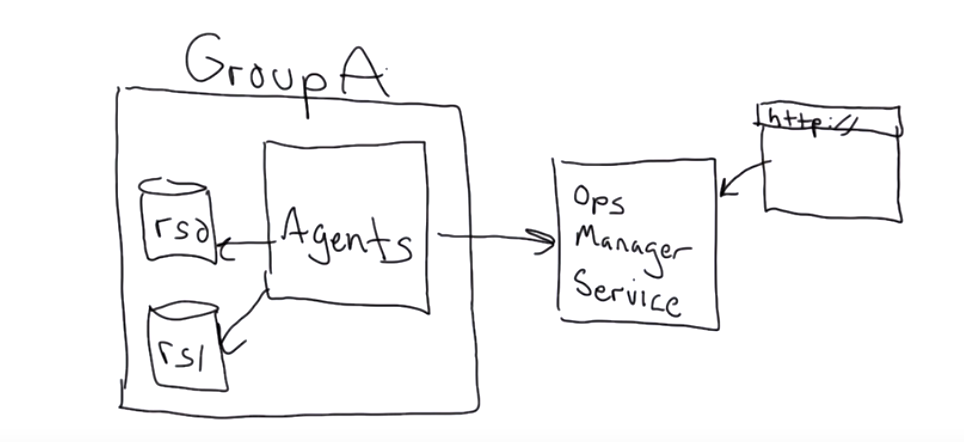
<https://youtu.be/jKjONXYxvGM>

**MONGODB OPS MANAGER:**

Overview of OPS Manager:

<https://youtu.be/qfIZfS34vKI>

**Architecture:**



**Basic Architecture:**

<https://youtu.be/ItqU7rCd4TY>

Once we install Ops Manager, all the files will be stored in below path:

/opt/mongodb/mms

Within this folder there are 2 important subfolders, they are **logs** and **conf**

The ‘Automation Agent’ has to be installed in every mongodb server, These Agents will also talk to Ops Manager server through port 8080 and also should be able to talk to other mongod /momgos processes in each server in the cluster.

The ‘Monitoring Agent’ will be automatically installed by ‘Automation Agent’ and the only difference is there can only **one Active** ‘Monitoring Agent’ at any point in time which will monitor all the nodes in the cluster and others will be passive.

**Admin Commands:**

>db.getLastError();

>db.getLastErrorObj();

>db.serverStatus();

>db.stats(); -- This will give DB stats

>db.collection.stats(); -- This will give Collection stats

**To Export and Import:**

>mongoexport –db dbname –collectionname –out collectionname.json

>mongoimport –collection imported –file collectionname.json

To import to a particular database

>mongoimport –db dbname --collection imported –file collectionname.json

Collation:

mongodump --host hostname --port 32017 --username usr --password pwd --out c:\backup --db my\_database --collection my\_collection

That will generate two files and one of them named my\_collection.metadata.json. Open this file and modify options property according to MongoDB [docs](https://docs.mongodb.com/manual/reference/bson-type-comparison-order/#collation).

{

"options": {

"collation": {

"locale": "en",

"strength": 1

}

}

...

}

And then restore using mongorestore

mongorestore --host hostname --port 32017 --username usr --password pwd --db contactstore c:\backup\my\_database --drop

From then on, any index you create will use that specific collation by default. Unfortunately, this requires a downtime window, so make sure you get one.

Find command for like:

db.getCollection("user").find({"userName" : /^.\*tele.\*$/i});

**For Oplog query:**

oplog.rs" is found in the "local" database. So, you can try something like:

use local

db.oplog.rs.find()

or from a different database:

db.getSiblingDB("local").oplog.rs.find()

### Check MongoDB Oplog Status

In order to check if you can use the MongoDB Oplog to recover the database, you can follow the steps below:

* Create a database backup following [this guide](https://docs.bitnami.com/google-templates/infrastructure/mongodb/administration/backup-restore/).
* Connect to the primary server via ssh (refer to the [FAQ](https://docs.bitnami.com/google-templates/faq/get-started/connect-ssh/) for more information).
* Connect to the MongoDB database and run the commands below:
* mongo admin -u root -p
* replicaset:PRIMARY> use local
* replicaset:PRIMARY> db.oplog.rs.find();

c

Please note that you get “PRIMARY” when accessing the MongoDB Shell

* The previous commands should print out the local MongoDB Oplog. If any of those instructions fail, or nothing is displayed, you’ll not be able to use this method to recover the database. If you created a snapshot of your servers you can try to recover the data deploying a new server using that backup.

### Recover A Database

Once you check the MongoDB Oplog exists and it’s available, you can proceed to dump and restore your database from it. Follow the following steps to do so:

* Dump the MongoDB Oplog to a folder (e.g. ~/oplog):
* mongodump -u root --authenticationDatabase admin -d local -c oplog.rs -o ~/oplog

mongodump -u root --authenticationDatabase admin -d local -c oplog.rs -o ~/oplog

mongodump --authenticationDatabase admin -d local -c oplog.rs -o /var/mongo/TEST/LOGS/oplog.txt

* Create a copy of the dump file to ensure you can manipulate it safely:
* mkdir ~/oplog2 && cp ~/oplog/local/oplog.rs.bson ~/oplog2/oplog.bson
* Use the command bsondump to inspect the .bson files and find the restoring limit:
* bsondump ~/oplog2/oplog.bson
* You should obtain an output similar to the one below:
* {"ts":{"$timestamp":{"t":1484002910,"i":1}},"t":{"$numberLong":"1"},"h":{"$numberLong":"7714671726931207910"},"v":2,"op":"c","ns":"test.$cmd","o":{"drop":"testData"}}
* Note the “t” and “i” attributes of the restore point (Note that this point won’t be included in the restore), as they are the ones used to restore the log (In this example: “t”:1484002910,“i”:1).
* You can also use that tool to find offending commands (for example you can look for “drop” commands, but also “remove”, “deleteOne”, and “deleteMany” would be suspicious). If the dump is too big, you can use grep or any other tool to find the commands:
* bsondump ~/oplog2/oplog.bson | grep drop
* If running mongorestore with --oplogReplay, the restore role is insufficient to replay the oplog. To replay the oplog, create a user-defined role that has anyAction on anyResourceand grant only to users who must run mongorestore with --oplogReplay.
* mongo admin -u root -p
* replicaset:PRIMARY> db.createRole( { role: "executeFunctions", privileges: [ { resource: { anyResource: true }, actions: [ "anyAction" ] } ], roles: [] } )
* replicaset:PRIMARY> db.grantRolesToUser("root", [ { role: "executeFunctions", db: "admin" } ])
* Restore your database using that point as the restore limit:
* mongorestore -u root --authenticationDatabase=admin --oplogReplay --oplogLimit 1484002910:1 ~/oplog
* This should restore your database to the state previous to data loss.
* Revoke the previous created role from the user root.
* mongo admin -u root -p
* replicaset:PRIMARY> db.revokeRolesFromUser("root", [ { role: "executeFunctions", db: "admin" } ])

This last step is optional and providing such access is not recommended, but don’t revoke it if your organization requires an user to run eval.

#### Troubleshooting

* Copy the application folder to have a backup of your upload files. Files that you have been uploaded to your database may not be included in the full backup described above.
* Check the users in the admin database to make sure it does not exist any unknown user in case your server has been attacked.

mongo admin -u root -p --eval "db.getUsers()"

**MONGORESTORE POINTINTIME:**

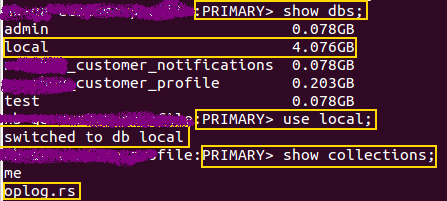
While working with databases sometimes there is a need to have data till a specific time or date across all the secondaries or database peers which is useful for testing a particular functionality. Also, in the event of outage, few folks would like to restore it to a specific weekend or month end just to have uniformity across all the dependent applications.

Here I am going to discuss how it can be achieved in [MongoDB](http://www.tothenew.com/mean-stack-web-development-consulting). I will be using MongoDB’s native tools – mongodump & mongorestore. Although these tools are heavily used to take database/table dumps of mongo data, these can also be used to have Oplog backup. Before proceeding, you must have idea of Oplog.

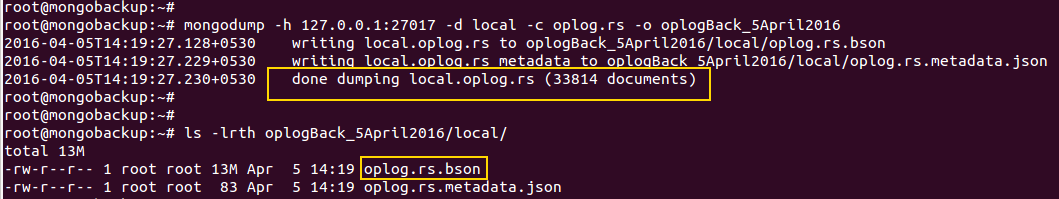
Oplog is like a normal collection (or table) in MongoDB which keeps all the operations that are received by MongoDB on its port. These operations can be updates, inserts, deletes etc. That means a working or executing copy of all the transactions that are stored in the normal collections will also be there in Oplog. However, this cannot be treated as a backup. You must keep your backup copies as well since Oplog is a capped collection, which means that it will keep overwriting the oldest data as new data comes in, once its capacity is full. So, choose Oplog size accordingly.

To have mongo restored to point in time, you must take its backup accordingly. This assumes you are already taking mongo data backups regularly. Just in addition to that, take Oplog dump as well:

Here is what you need to take backup of – **oplog.rs** collection:

****

Take the backup using mongodump:

****

From above command,  **local** is the database and **oplog.rs** is the collection name. oplogBack\_5April2016 is the non-existent directory where you will take backup. **oplog.rs.bson** is the required dump collected after running this command and which will be used to have point-in-time restored data.

To restore to point in time, you must have your ‘time’ handy. What that means is you must know the date and time up to which you want to actually restore. That is calculated on the basis of [epoch](https://en.wikipedia.org/wiki/Epoch_(reference_date)) value. Oplog keep all the transactions on the basis of this epoch value available as **timestamp**.

Now suppose you want to restore up to a date, say 31st March 2016 23:59:00. Its epoch value will be calculated as ***1459468740***. Having said that, it is not necessary Oplog will have a record created exact on the same time, it might create a record earlier or later to this time. And obviously that will happen most of the time. In that scenario, you must find an epoch value just less than the required epoch time and you will restore up to that timestamp. Your goal is to find **“”ts”:{“timestamp”:{“t”:x,”i”:y}}”** entry and note its values from Oplogs. X is the epoch value we are interested in and Y is an incrementing ordinal for operations within a given second. In case you find more than 1 value of Y for the same value of X, you should use the largest Y value.

In my case, I created and used this [script](https://gist.github.com/Amit-Naudiyal/553dd057c5d97ae9c4bec0433ab0c0f4) to get an epoch just less than my desired point-in-time epoch value from Oplogs. To be able to use this script you must have your Oplog dump in human readable format. [bsondump](https://docs.mongodb.org/manual/reference/program/bsondump/) is such utility to convert your bson dump to json values.

**bsondump**

Say or For example, I have got**“ts” : timestamp(1459464310, 1)**& **“ts” : timestamp(1459464310, 2)** as epoch just less than my desired timestamp value. This corresponds to *31 Mar 2016 22:45:10* and this is the last record my Oplog had just before 31st March 23:59:00. The ordinal value I choose will be 2.

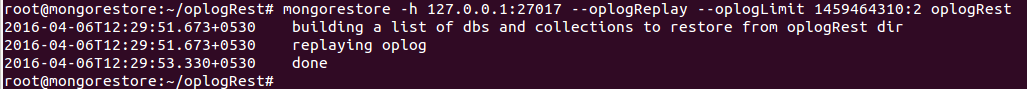
Once you are ready with the Oplog dump and the correct epoch and ordinal values, use following procedure for restoration:

**Step1**. Restore the normal data from latest dump just before the point-in-time timestamp. Say you do weekly backup and so you had backup only till 27th March 2016 (*Sunday*). So, restore the data on new node up to 27th March 2016. It will be done using your normal restoration procedure.

**Step2**: Rename your dump with **oplog.bson** as mongorestore will look for oplog.bson in the specified directory or in the root of dump directory if no directory specified.

**rename**

**Step3**. Replay your Oplogs until **1459464310:2.**

****

**–oplogReplay:**Replays the oplog after restoring the dump to ensure that the current state of the database reflects the point-in-time backup.

**–oplogLimit:**Prevents mongorestore from applying oplog entries with timestamp newer than or equal to <timestamp>.

**oplogRest**: This is the directory where I have kept my oplog.bson.

Just note that restoring Oplogs is an idempotent activity and it will do no harm if you run this multiple times. This is the time you can verify if you have restored correctly upto your desired timestamp. That should be all.

**===================================MONGO 4.0 UPGRADE========================**

**>db.adminCommand({usersInfo: {forAllDBs: true}, filter:{ mechanisms: “SCRAM-SHA-256” }})**

**Downgrading to 3.6**

**>db.adminCommand({setFeatureCompatibilityVersion” “3.6” })**

**Logon to SECONDARY NODE and do a rollover update in all 3 nodes:**

**SECONDARY>use admin**

**SECONDARY>db.shutdownServer()**

**Verify in Linux machine:**

* + **Ps –ef | grep mongod**

**Now run the mongod from 3.6 Binaries:**

**>/opt/mongodb/mongodb-linux-x86\_64-enterprise-ubuntu1404-3.6.5/bin$ ./mongod –f /shared/rs2\_unix.conf**

**Above should doengrade the mongod instance.**

**>rs.slaveOk()**

**>rs.config()**

**Now do the same in second Secondary Node and then go to Primary**

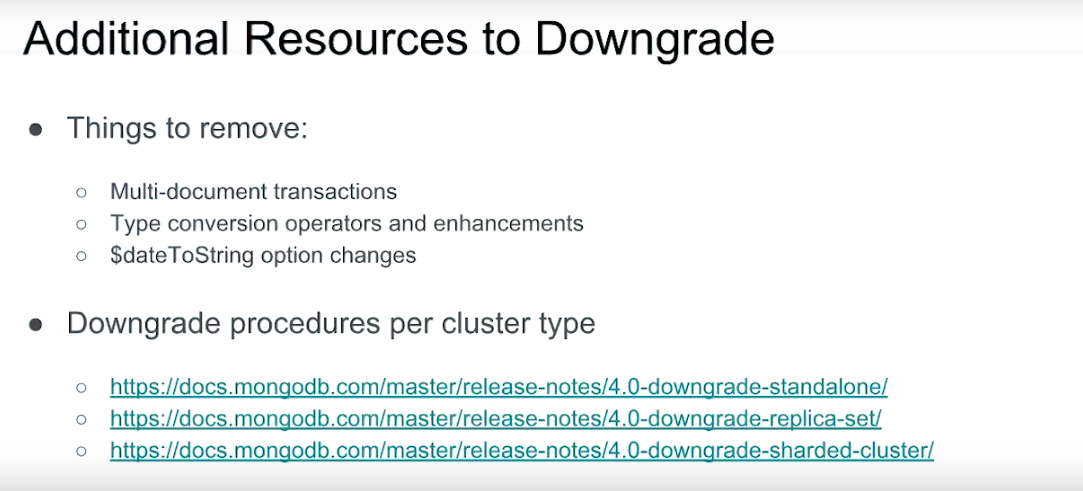
**PRIMARY> rs.stepDown()**



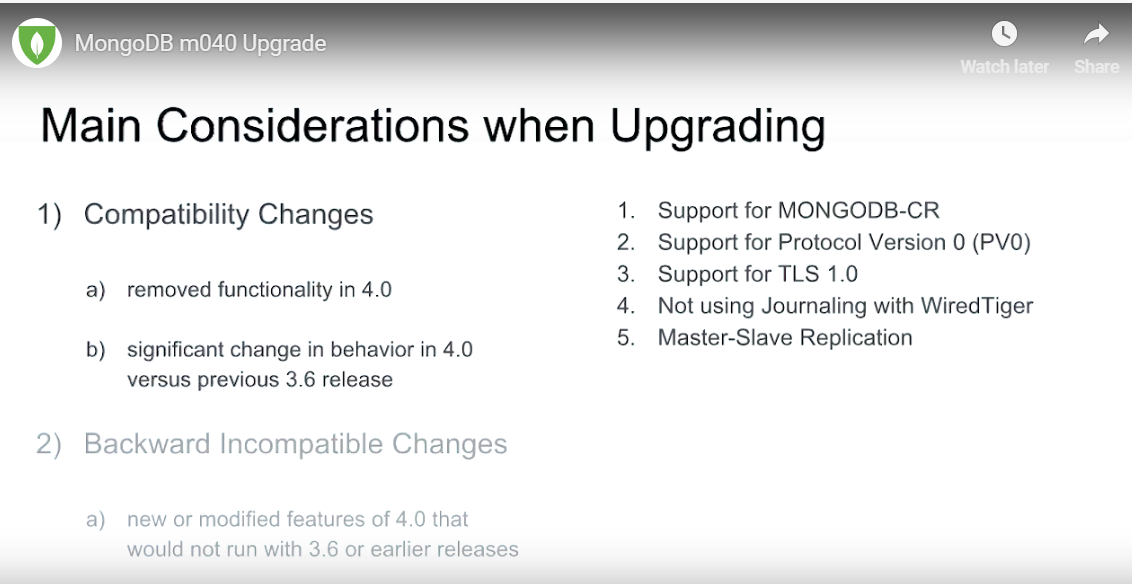
**Check if there is another PRIMARY elected:**

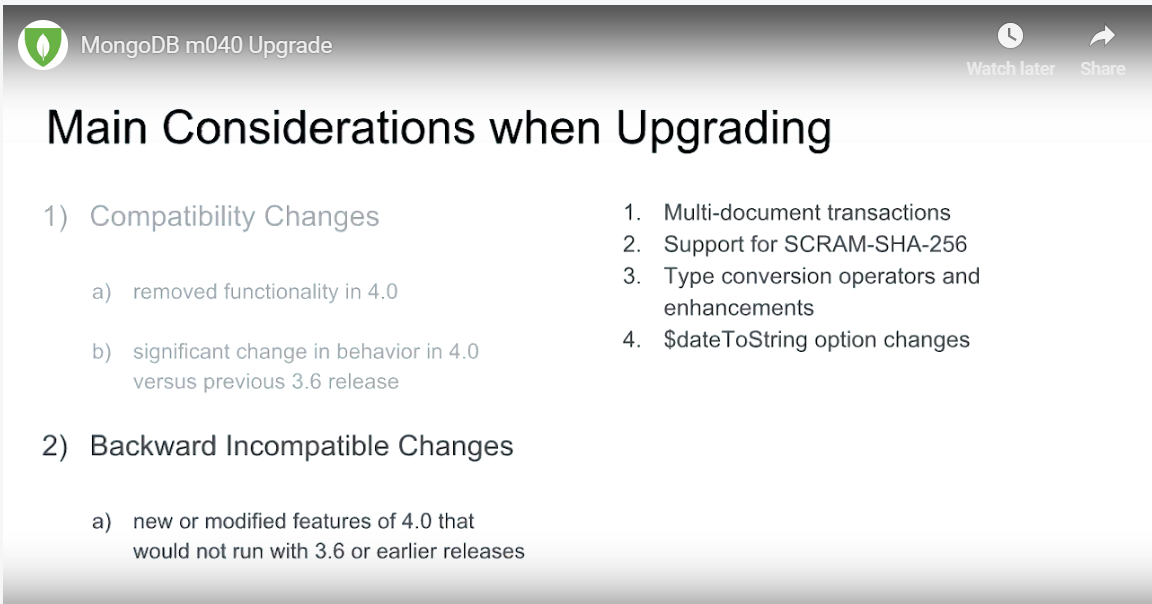
**>rs.status()**

**Then do the downgrade steps like shutdown and then downgrade:**

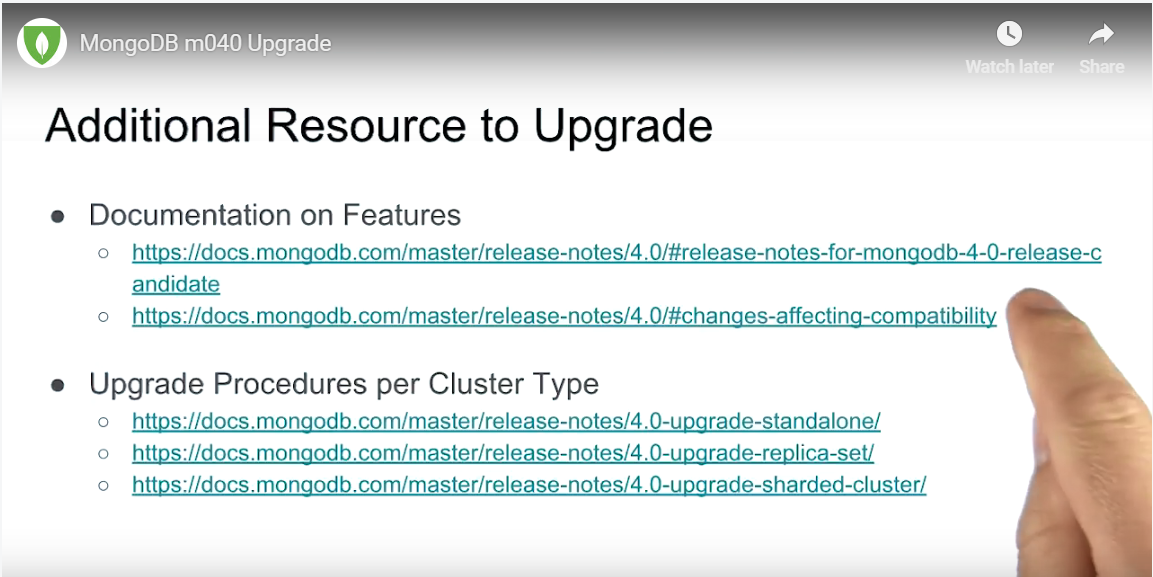


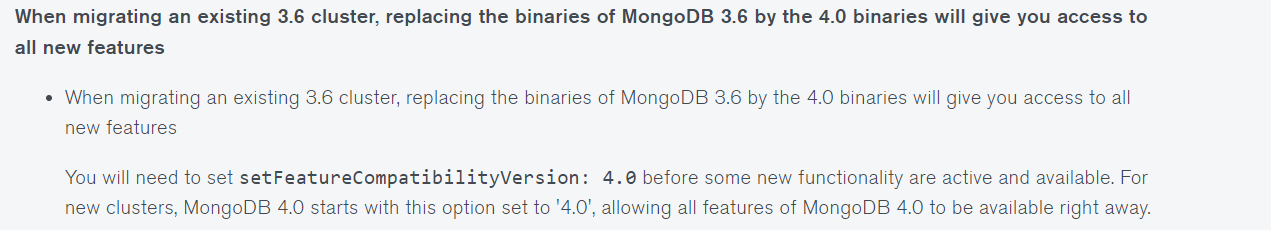
**=============================UPGRADE===================**





For additional details, please see below:





**Use the links above to see the upgrade functionality and tutorials .**

**======================END of UPGRADE======================================**

MongoDB\_Compass:

For aggregation pipeline, pelae see below video :

<https://youtu.be/aVIoeb6hnbA>

MONGO BACKUP AND RESTORE

|  |  |  |
| --- | --- | --- |
| Mongodump - full db |  | docker exec -it newtest\_newdbtest\_1 mongodump -u testadmin --password  'Password123!'  --out /testbackup --authenticationDatabase admin |
| Record insert | 6,7 | Inserted |
| oplog backup |  | docker exec -it newtest\_newdbtest\_1 mongodump -u testadmin --password  'Password123!' --authenticationDatabase admin -d local -c oplog.rs -o /testbackup |
| Record insert | 8,9 | Inserted |
| oplog backup |  | docker exec -it newtest\_newdbtest\_1 mongodump -u testadmin --password  'Password123!' --authenticationDatabase admin -d local -c oplog.rs -o /testbackup |
| Record insert | 10,11 | Inserted |
| oplog backup |  | docker exec -it newtest\_newdbtest\_1 mongodump -u testadmin --password  'Password123!' --authenticationDatabase admin -d local -c oplog.rs -o /testbackup |
| Record insert | 12,13 | Inserted |
|  |  | docker exec -it newtest\_newdbtest\_1 mongodump -u testadmin --password  'Password123!' --authenticationDatabase admin -d local -c oplog.rs -o /testbackup |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | mkdir |
|  |  | mv |
|  |  |  |
|  |  | docker exec -it newtest\_newdbtest\_1 mongorestore  /testbackup |
|  |  | docker exec -it newtest\_newdbtest\_1 mongorestore --oplogReplay --oplogLimit 1547060520:2   /testbackup |
|  |  | docker exec -it newtest\_newdbtest\_1 mongorestore --oplogReplay --oplogLimit 1547060640:2   /testbackup |