<https://university.mongodb.com/>

[Are you behinf Firewall?](https://university.mongodb.com/mercury/M001/2019_February/chapter/Chapter_1_Introduction/lesson/5963b30cc1da5a32116dc5b3/tab/5963b30cc1da5a32116dc5b4)

<http://portquiz.net:27017>

telnet cluster0-shard-00-00-jxeqq.mongodb.net 27017

**ping cluster0-shard-00-00-jxeqq.mongodb.net**

**Data Types**

int32, double, string, date, array, objectid

**Asignado y Documentado**

### mongo shell ###

**Connection**:

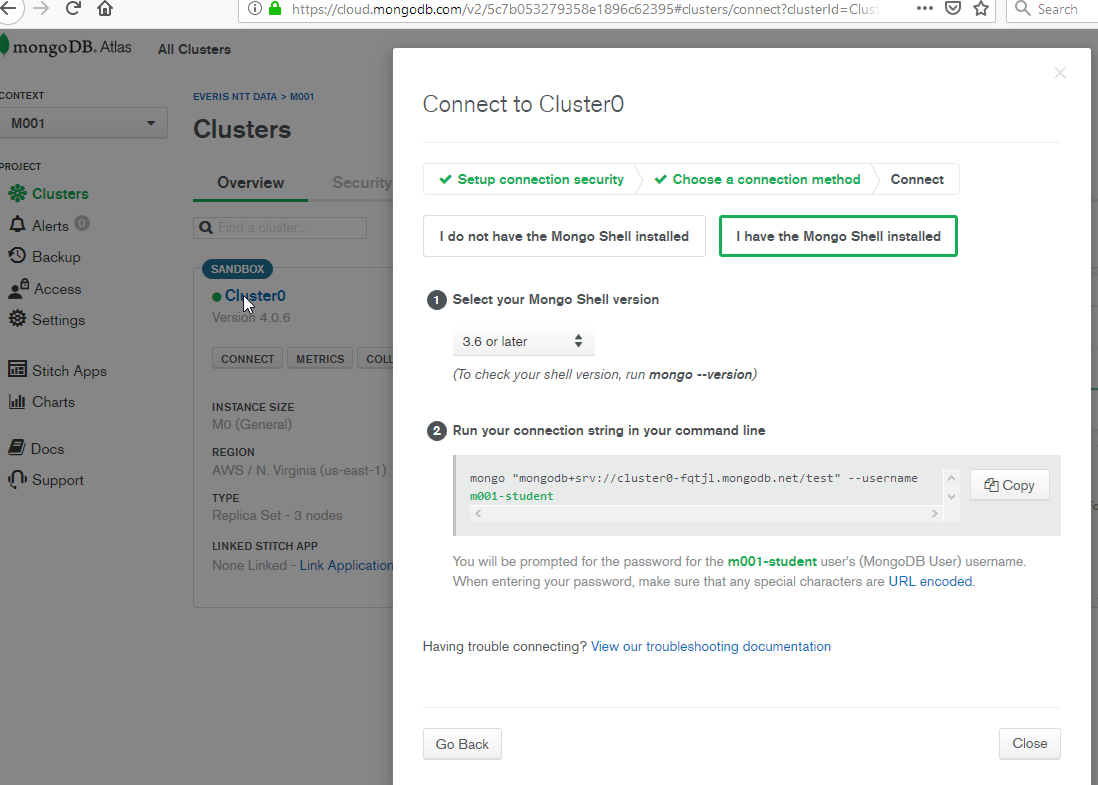
mongo "mongodb://cluster0-shard-00-00-jxeqq.mongodb.net:27017,cluster0-shard-00-01-jxeqq.mongodb.net:27017,cluster0-shard-00-02-jxeqq.mongodb.net:27017/test?replicaSet=Cluster0-shard-0" --authenticationDatabase admin --ssl --username m001-student --password m001-mongodb-basics

clusters:

cluster0-shard-00-00-jxeqq.mongodb.net:27017

cluster0-shard-00-01-jxeqq.mongodb.net:27017

cluster0-shard-00-02-jxeqq.mongodb.net:27017



**Connecting to a cluster**

mongo "mongodb+srv://cluster0-fqtjl.mongodb.net/test" --username m001-student --password m001-mongodb-basics

In this lecture video we end up getting connected to the Primary node of our Sandbox replica set. In case you get connected to a Secondary node, in order to run queries or commands, you will have to run rs.slaveOk() shell command, to list databases, collections and run queries.

Example working dir for this course

C:\Users\USER\Documents\MyApps\MongoDb\M001

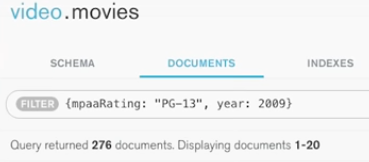
cluster demo: cluster0-shard-00-00-fqtjl.mongodb.net

SHELL COMMNADS

fully functional javascript interpreter

Todas las entradas debeb tener un campo \_id. Si no se especifica se crea automaticamente como \_id: ObjectId(“ xxxxxxx”)

**Filtros para leer los documentos**

****

**{wind.type: “C”}**, en el caso que wind es de tipo Object.

**{wind.direction.angle: 290}** : En este caso wind y direction son tambien objetos.

En el shell hay que poner comillas en el nombre del filtro: **{“wind.direction.angle”: 290}**

**quit()**

**show dbs**

**use** database\_name

**show collections**

**db.collection\_name.insertOne (** {“key”:”value”} **)**

**db.collection\_name.insertMany ( [** {“key”:”value”},{“key”:”value”},…,{“key”:”value”} **], {“ordered”: “false” / “true” } )**

**true:** Termina si encuentra error insertando datos. También se llama **ordered.** Es el valor por defecto.

**load (**“javascript\_file\_name.js”**)**

**find: Lee documentos**

use video

db.movieDetails.find ( {rated:”PG-13”}).pretty(**)**

**find in arrays**

*Match arrays wit the exact fields*

db.movies.find ( {cast: **[**"jeff Birdges", "Tim Robbins**]**}).pretty()

db.movieDetails.find (writers : "{[Ethan Coen","Joel Coen"]}").pretty()

*Match all the arrays with the field*

db.movies.find ( {cast: **[**"jeff Birdges"**]**}).pretty()

*Match the arrays with the field at the position specified*

db.movies.find ( {"cast**.0"** : **[**"jeff Birdges"**]**}).pretty()

**projection**

*returns only the field in the second argument of the find menthod*

db.movies.find ( { genre : "Action, Adventure"}), {title: 1} )

*el numero 1 indica "incluir el campo" y 0 indica "excluir el campo"*

db.movies.find ( { genre : "Action, Adventure"}), {title: 1, \_id:0} )

# Updating Documents

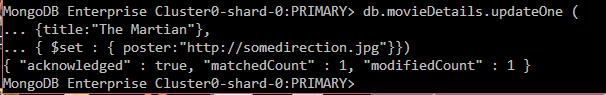
**db.movieDetails.updateOne** (

{ title : "The Martian" },  *🡺 Identify the document to update. Only the first match will be updated.*

{ **$set** : { poster : "<http://somedirection.img>" } } ***$Set*** *Add or modified the field.*

)

*example*



[update operators](https://docs.mongodb.com/manual/reference/operator/update/)

**updateOne, updateMany**

**$inc**

**db.movieDetails.updateOne** (

{ title : "The Martian" },  *🡺 Identify the document to update. Only the first match will be updated.*

{ **$inc** : { "tomato.reviews" : 3,

"tomato.userReviews" : 25 }

}

)

### Array Update

**$addToSet :**  *Adds elements to an array only if they do not already exist in the set.*

**$pop :** *Removes the first or last item of an array.*

**$pullAll :** *Removes all matching values from an array.*

**$pull :** *Removes all array elements that match a specified query.*

**$push :** *Adds an item to an array.*

#### Modifiers

**$each** *Modifies the* ***$push*** *and* ***$addToSet*** *operators to append multiple items for array updates.*

**$position** *Modifies the $push operator to specify the position in the array to add elements.*

**$slice** *Modifies the $push operator to limit the size of updated arrays.*

**$sort** *Modifies the $push operator to reorder documents stored in an array.*

db.movieDetails.**updateOne** (

{ title : " The Martian" },

{ **$push** :

{ reviews :

{ **$each** :

[ { rating : 0.5,

date : ISODate ("206-01-12T07:00:00Z"),

reviewer" :Yabo A.",

text : reviewText2 } ,

{ rating : 0.6,

date : ISODate ("206-01-12T07:00:00Z"),

reviewer" :Krsitina Z.",

text : reviewText3 }

]

}

}

}

);

*Ejemplo elimina todos los campos rated con valor de null*

db.movieDetails.**updateMany** ( { rated: null}, { **$unset** : { rated : "" } } );

#### Upsert Option

If **updateOne(),** **updateMany(),** or **replaceOne()** includes **upsert** : **true** and no documents match the specified filter, then the operation creates a new document and inserts it. If there are matching documents, then the operation modifies or replaces the matching document or documents.

db.movieDetails.updateOne ( { "imdb.id" : detail.imdb.id },

{ $set : detail},

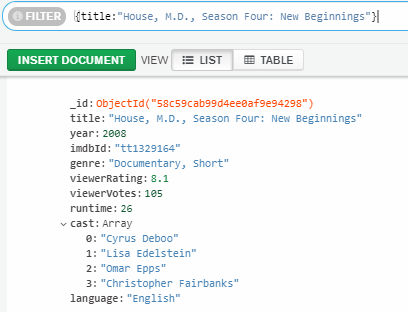
{ upsert : true } );

[replaceOne](https://docs.mongodb.com/manual/reference/method/db.collection.replaceOne/)

Important aspect to consider about replaceOne:

* The *replacement* document cannot contain [update operators](https://docs.mongodb.com/manual/reference/operator/update/#id1).
* replaceOne will apply changes to only one document, the first found in the server that matches the filter expression, using the [$natural](https://docs.mongodb.com/manual/reference/method/cursor.sort/#return-natural-order) order of documents in the collection.

Example



*Crea el filtro*

let filter = {title: "House, M.D., Season Four: New Beginnings"}

*Carga el documento*

let doc = db.movies.findOne ( filter);

*Set the new poster field*

doc.poster;

doc.poster = "https://www.imdb.com/title/tt1329164/mediaviewer/rm2619416576";

*Get the values in the array genres*

doc.genres;

*Put a new value in the array*

doc.genres.push("TV Series");

*Replace the old document with this one*

db.movies.replaceOne(filter, doc);

## Delete Operations

deleteOne and deleteMany

db.reviews.deletOne ( { \_id: ObjectId ("xxxxxx") } );

db.reviews.delet many ( { reviewer\_id: 12345 } );

# Query Operators

<https://docs.mongodb.com/manual/reference/operator/query>

db.movieDetails.find({runtime: {$**gt**: 90}})

db.movieDetails.find({runtime: {$**gt**: 90}}, {\_id: 0, title: 1, runtime: 1})

db.movieDetails.find({runtime: {$**gt**: 90, $**lt**: 120}}, {\_id: 0, title: 1, runtime: 1})

db.movieDetails.find({runtime: {$**gte**: 90, $**lte**: 120}}, {\_id: 0, title: 1, runtime: 1})

db.movieDetails.find({runtime: {$**gte**: 180}, "tomato.meter": 100}, {\_id: 0, title: 1, runtime: 1})

db.movieDetails.find({runtime: {$**gte**: 180}, "tomato.meter": {**$gte**:95} }, {\_id: 0, title: 1, runtime: 1})

db.movieDetails.find({rated: {$**ne**: "UNRATED"}}, {\_id: 0, title: 1, rated: 1})

db.movieDetails.find({rated: {$**in**: ["G", "PG"]}}, {\_id: 0, title: 1, rated: 1})

db.movieDetails.find({rated: {$**in**: ["G", "PG", "PG-13"]}}, {\_id: 0, title: 1, rated: 1}).pretty()

db.movieDetails.find({rated: {$**in**: ["R", "PG-13"]}}, {\_id: 0, title: 1, rated: 1}).pretty()

### Element Operartos

db.moviesDetails.find({mpaaRating: {**$exists**: true}})

db.moviesDetails.find({mpaaRating: {**$exists**: false}})

db.movieDetails.find({mpaaRating: **null**})

db.movieDetails.find({})

<https://docs.mongodb.com/manual/reference/operator/query/type>

db.movies.find({viewerRating: {**$type: "int"**}}).pretty()

db.movies.find({viewerRating: {**$type: "double"**}}).pretty()

### Logical Operators

or/and usan array

db.movieDetails.find({**$or**: [{"tomato.meter": {$gt: 95}},

{"metacritic": {$gt: 88}}]},

{\_id: 0, title: 1, "tomato.meter": 1, "metacritic": 1})

db.movieDetails.find({**$and**: [{"tomato.meter": {$gt: 95}},

{"metacritic": {$gt: 88}}]},

{\_id: 0, title: 1, "tomato.meter": 1, "metacritic": 1})

and operator se usa cuando comparamos un mismo campo, de lo contrario no es necesario

db.movieDetails.find({"tomato.meter": {$gt: 95},

"metacritic": {$gt: 88}},

{\_id: 0, title: 1, "tomato.meter": 1, "metacritic": 1})

db.movieDetails.find({**$and**: [{"metacritic": {$ne: null}},

{"metacritic": {$exists: true}}]},

{\_id: 0, title: 1, "metacritic": 1})

db.movieDetails.find({**$and**: [{"metacritic": null},

{"metacritic": {$exists: true}}]},

{\_id: 0, title: 1, "metacritic": 1})

{ $and : [ {"tripduration":{$exists:true} },{"tripduration":null } ] }

### Array

db.movieDetails.find({genres: {**$all**: ["Comedy", "Crime", "Drama"]}},

{\_id: 0, title: 1, genres: 1}).pretty()

db.movieDetails.find({countries: {**$size: 1** }},

{\_id: 0, title: 1, genres: 1}).pretty()

### $elemMatch

boxOffice: [ { "country": "USA", "revenue": 228.4 },

{ "country": "Australia", "revenue": 19.6 },

{ "country": "UK", "revenue": 33.9 },

{ "country": "Germany", "revenue": 16.2 },

{ "country": "France", "revenue": 19.8 } ]

*regresa todos los elementos que contenga el pais germany o revenue mayor que 17*

db.movieDetails.find({"boxOffice.country": "Germany", "boxOffice.revenue": {$gt: 17}})

db.movieDetails.find({"boxOffice.country": "Germany", "boxOffice.revenue": {$gt: 228}})

use video

martian = db.movieDetails.findOne({title: "The Martian"})

martian

delete martian.\_id;

martian

martian.boxOffice = [

{"country": "USA", "revenue": 228.4},

{"country": "Australia", "revenue": 19.6},

{"country": "UK", "revenue": 33.9},

{"country": "Germany", "revenue": 16.2},

{"country": "France", "revenue": 19.8}

]

db.movieDetails.insertOne(martian);

*Regresa los documentos cuyo array contengan "country":"germany" y "revenue" mayor que 16, que en el ejemplo seria 1*

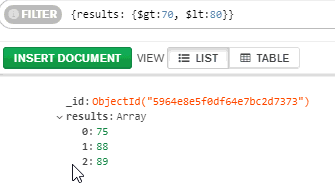
db.movieDetails.find({boxOffice: {$elemMatch: {"country": "Germany", "revenue": {$gt: 17}}}})

### $regex Operator

*/ / limitan la expresion regular*

db.movieDetails.find({"awards.text": {**$regex**: **/^Won.\* /**}}, {\_id: 0, title: 1, "awards.text": 1}).pretty()

*Busca valores dentro de un array*

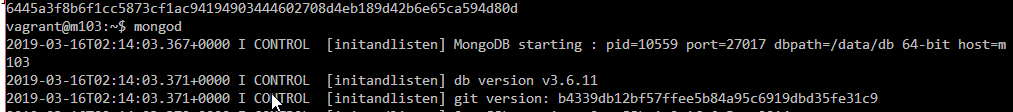


# M103

## The mongod

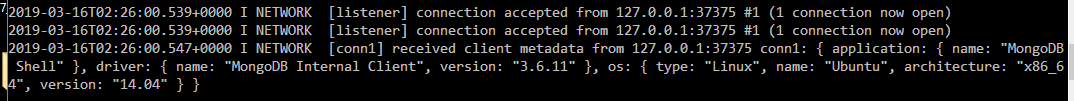
Is the main daemon process for MongoDB

Para ejecutar mongod, ejecutamos mongod, en el ambiente de vagrant:



la primera linea nos da informacion: **pid, port, dbpath y host**.

tambien podemos ver informacion sobre la coneccion:



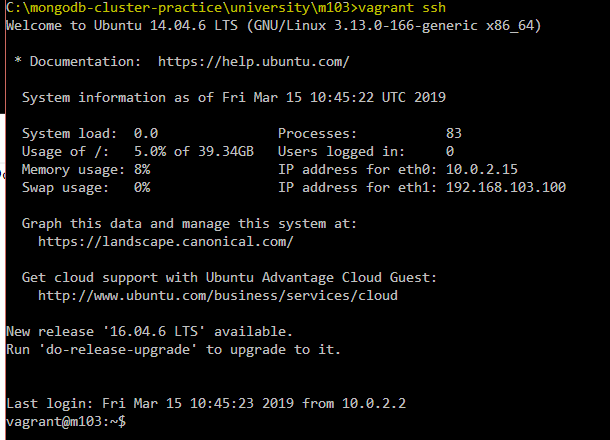
una vez que ejecutamos mongodb la terminal deja de ser accesible, por lo que es necesario abrir otra terminal:

$vagarant ssh

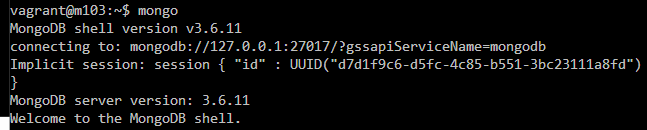
no se puede ejecutar vagrant desde cualquier lugar, hay que hacerlo desde donde hicimos la instalacion inicial:

C:\mongodb-cluster-practice\university\m103\vagrant up

C:\mongodb-cluster-practice\university\m103\vagrant ssh



Para conectarnos a mongodb ejecutamos el comando **mongo**



Command line options for start mongod

--port

--dbpath

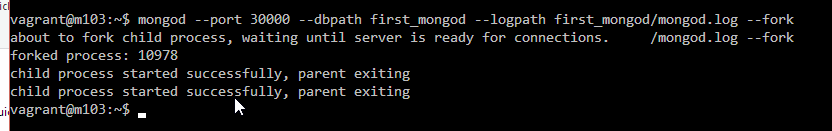
--logpath

--fork

<https://docs.mongodb.com/manual/reference/program/mongod>

Before specified diferent db path we have to create that folder.

Si se especifica **–fork** debemos especificar tambien **logpath**



## Configuration File

### Basic commands configuration options

**dbpath**

**logpath**

**bind\_ip** : *Starting with 3.6 you need to set bind ip to include a network adapter on the host that provides access to the network, if not mongod can only accept conections on that same host.*

**replSet, keyFile :** *Start mongod in replacation mode with basic intercluster, security and user authentication enabled.*

**sslPEMKey**

**sslCAKey**

**sslMode**

**fork**

### Configuration File Option

dbPath 🡪 storage.dbPath

logPath 🡪 systemLog.path and systemLog.destination

bind\_ip 🡪 net.bind\_ip

replSet 🡪 replication.replSetName

keyFile 🡪 security.keyFile

sslPEMKey 🡪 net.ssl.sslPEMKey

sslCAKey 🡪 net.ssl.sslCAKey

sslMode 🡪 net.sslMode

fork 🡪 processManagement.fork

mongod configuration file options

<https://docs.mongodb.com/manual/reference/configuration-options>

**Y**et **A**nother **M**arkup Language

Example configuration file, with the same configuration options as above:

storage:

dbPath: "/data/db"

systemLog:

path: "/data/log.mongod.log"

destination: "file"

replication:

replSetName: M103

net:

bindIp : "127.0.0.1,192.168.0.100"

ssl:

mode: "requireSSL"

PEMKeyFile: "/etc/ssl/ssl.pem"

CAFile: "/etc/ssl/SSLCA.pem"

security:

keyFile: "/data/keyfile"

processManagement:

fork : true

mongod - -config "etc/mongod.conf"

mongod -f "etc/mongod.conf"

## File Structure

**Let's recreate the user m103-admin:**

* Role: root on admin database
* Username: m103-admin
* Password: m103-pass

vagrant@m103:~$ mongo admin --host localhost:27000 --eval '

> db.createUser({

> user: "m103-admin",

> pwd: "m103-pass",

> roles: [

> {role: "root", db: "admin"}

> ]

> })

> '

**connecting to mongod**

mongo –u m103-user –p 103-pass - -host localhost - - port 2700

## Basic Commands

db.<method>

db.<collection>.<method>

rs.<method> : control set deployment and management.

sh.<method> : control shared cluster.

## User Management

db.createUser()

db.dropUser()

## Database Managament

db.dropDatabase()

db.createCollection()

## Database Status

db.serverStatus()

## Database Commands

If you want to run a database command from a driver, then you need to execute the underlying database

command instead, assuming that the driver doesn't have some other existing helper method.

db.runCommand ( { <COMMAND> } )

db.commandHelp ("<command>")

example: Creating index with Database Command:

db.runCommand(

{ "createIndexes": <collection> },

{ "indexes": [

{

"key": { "product": 1 }

},

{ "name": "name\_index" }

]

}

)

Creating index with Shell Helper:

db.<collection>.createIndex(

{ "product": 1 },

{ "name": "name\_index" }

)

## Collection Management

db.<collection>.renameCollection()

db.<collection>.createIndex()

db.<collection>.drop()

# Logging Basics

db.getLogComponents() : retrieve the log components from my current db

**-1** indicates that inherits from parent

**0**: Default Verbosity, to include informal messages

**1-5**: Increases the verbosity level to include Debug messages.

MongoDB Enterprise > db.getLogComponents()

{

"verbosity" : 0,

"accessControl" : {

"verbosity" : -1

},

"command" : {

"verbosity" : -1

},

"control" : {

"verbosity" : -1

},

"executor" : {

"verbosity" : -1

},

"geo" : {

"verbosity" : -1

},

"index" : {

"verbosity" : -1

},

"network" : {

"verbosity" : -1,

"asio" : {

"verbosity" : -1

},

"bridge" : {

"verbosity" : -1

}

},

"query" : {

"verbosity" : -1

},

"replication" : {

"verbosity" : -1,

"heartbeats" : {

"verbosity" : -1

},

"rollback" : {

"verbosity" : -1

}

},

"sharding" : {

"verbosity" : -1,

"shardingCatalogRefresh" : {

"verbosity" : -1

}

},

"storage" : {

"verbosity" : -1,

"journal" : {

"verbosity" : -1

}

},

"write" : {

"verbosity" : -1

},

"ftdc" : {

"verbosity" : -1

},

"tracking" : {

"verbosity" : -1

}

}

db.adminCommand ( {"getLog" : "global" } ) : global, return all the log activity

db.setLogLevel (0,"index")

another way to see the logs:

tail –f /var/log/mongodb/mongod.log

"2019-03-17T05:02:51.479+0000 I ACCESS [conn5] Successfully authenticated as principal m103-admin on admin",

## Log Message Severity Level

F – Fatal

E – Error

W – Warning

I – Informational (Verbosity Level 0)

D – Debug (Verbosity Level 1-5)

# Profiling the Database

Level 0 : The profiler is off and does no collect any data. This is the default profiler level.

Level 1: The profiler collects data for operations that take longer than the value of slow ms.

Level 2 : The profiler collects data for all operations.

db.getProfilingLevel() : Get profiling level.

db.setProfilingLevel(1) : Set profiling level to 1

db.setProfilingLevel( 1, { slowms: 0 } ) : Set slowms to 0:

db.system.profile.find().pretty() : Get profiling data from system.profile.

# Basic MongoDB Security

Authentication : Verifies the **Identity** of a user.

## Authentication Mechanisms

SCRAM : Default and the most basic form. Salted Challenge Response Authentication Mechanism. Password Security.

X.509 : X.509 Security for Authentication. Is a more secure, more complex.

LDAP : *Only for MongoDB Enterprise*. Lighweight Directory Access Protocol.

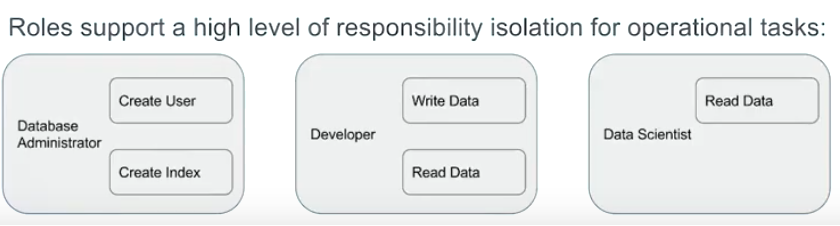
KERBEROS: *Only for MongoDB Enterprise.*

MongoDB also procides Cluster Authentication Mechanisms

Authorization

Verifies the **privileges** of a user. Role Based Access Control.

* Each user has one or more **Roles**.
* Each Role has one or more **Privileges**.
* A **Privilege** represents a group of Actions and the Resources those actions apply to.



Print Configuration File

|  |
| --- |
| cat /etc/mongod.conf |

### Localhost Exception

* Allows you to access a MongoDB server that enforces authentication but does not yet have a configured user for yo to authenticate with.
* Must run Mongo shell from the **same host** running the MongoDB server.
* The localhost exception **closes** after you create your first user.
* **Always** create a user with administrative privilges **first.**

|  |
| --- |
| MongoDB Enterprise > db.stats()  {  "ok" : 0,  "errmsg" : "not authorized on admin to execute command { dbstats: 1.0, scale: undefined, lsid: { id: UUID(\"ef8449cb-eb4b-4977-bdc4-61cdd6b92c81\") }, $db: \"admin\" }",  "code" : 13,  "codeName" : "Unauthorized"  }  MongoDB Enterprise > |

Create the first admin user

|  |
| --- |
| MongoDB Enterprise > db.createUser (  ... {  ... user:"root",  ... pwd:"root",  ... roles:["root"]  ... }  ... )  Successfully added user: { "user" : "root", "roles" : [ "root" ] }  MongoDB Enterprise > |

Authetication throw the shell

|  |
| --- |
| vagrant@m103:~$ mongo –host 127.0.0.1:27017 \  > -- username root \  > --password root \  > --authenticationDatabase admin  MongoDB shell version v3.6.11  connecting to: mongodb://127.0.0.1:27017/?authSource=admin&gssapiServiceName=mongodb  Implicit session: session { "id" : UUID("d7dcc2f0-3736-4751-bf66-7434ee335476") }  MongoDB server version: 3.6.11  Server has startup warnings:  2019-03-18T11:56:13.989+0000 I STORAGE [initandlisten]  2019-03-18T11:56:13.989+0000 I STORAGE [initandlisten] \*\* WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine  2019-03-18T11:56:13.989+0000 I STORAGE [initandlisten] \*\* See http://dochub.mongodb.org/core/prodnotes-filesystem  MongoDB Enterprise > |

# Built-In Roles: Part 1

How Role Based Access Control (RBAC) Works

Roles in MongoDB

* Role Based Access Control
* Database User granted Roles
* Custom Roles
  + Tailored roles to attend specific needs of sets of users
* Built in Roles
  + Pre-packaged MongoDB Roles
* Role Structure
  + Privileges
  + Action + Resource

Role Structure

Role is composed of

- Set of privileges: Actions 🡪 Resources

A role con inherit from anther role.

We can also define Network Authentication Restrictions, allow to connect from a clientSource or to a serverAddress by specifiying a set of clientSource and severAddress

### Resources

|  |  |
| --- | --- |
| Database  Collection  Set of Collections  Cluster  . Replica Set  . Shard Cluster | //specific database and collection  { db: "products", collection: "inventory"}  //all databases and all collections  { db: "", collection: "" }  //specific database any collection  { db: "products", collection: ""}  //or cluster resource  {cluster: true} |

Privilege is defined by: Resource and Actions allowed over a resource.

//allow to shutdown over the cluster

{resource: {cluster: true}, actions: ["shutdown"]}

## Buil-In Roles

* Database User: read, readWrite
* Database Administration: dbAdmin, userAdmin, dbOwner
* Cluster Administration: ckusterAdmin. clusterManger, clusterMonitoring, hostManager
* Backup/Restore
* Super User: root

### Buitl-In All Database Role

**Database User** : readAnyDatabase, readWriteAnyDatabase

**Database Administration**: dbAdminAnyDatabase, userAdminAnyDatabase

**Super User**: root

Lecture Notes: [M310 – MongoDB Security](https://university.mongodb.com/courses/M310/about)

La conexion a mongod –f mongod.conf es:

mongo admin –u root –p root

1-Create a securty officer

**db.createUser** ({user : 'security\_officer',pwd : 'h3ll0th3r3',roles: [{db:'admin', role:'userAdmin'}]})

|  |
| --- |
| MongoDB Enterprise > db.createUser ({user : 'security\_officer',pwd : 'h3ll0th3r3',roles: [{db:'admin', role:'userAdmin'}]})  Successfully added user: {  "user" : "security\_officer",  "roles" : [  {  "db" : "admin",  "role" : "userAdmin"  }  ]  }  MongoDB Enterprise > |

**userAdmin**: This is the first user you should always create because allows user to do all operations around user management, but in themselves is not able to do anything related with data management or data modifications. Can not create or write, cannot list databases, cannot do anything around database administration aside from creating and updating or reviewing database users. This is quite important if you want to ensure that there are specific users in your organization that are not allowed to do anything with data in your system just managing other user that themselves can create the data

changeCustomData

changePassword

createRole

createUser

dropRole

dropUser

grantRole

revokeRole

setAuthenticationRestriction

viewRole

viewUser

2- Create a user that is allowed to actually administer the database. **dbAdmin.** Doesn’t has permission to read or write data, every thing related with **DDL**, Data Definition Language, every thing related with **DML**, Data Modification Language,

collStats

dbHash

dbStats

killCursors

listIndexes

listCollections

. . .

bypassDocumentValidation

collMod

collStats

compact

convertToCapped

. . .

|  |
| --- |
| MongoDB Enterprise > db.createUser (  ... { user : 'dba',  ... pwd : 'c1lynd3rs',  ... roles : [ { db : 'm103', role : 'dbAdmin' } ] } )  Successfully added user: {  "user" : "dba",  "roles" : [  {  "db" : "m103",  "role" : "dbAdmin"  }  ]  }  MongoDB Enterprise > |

All user should be created in admin db, will be used to authenticate the users that we create

**db.grantRolesToUser**( "dba", [ { db: "playground", role: "dbOwner" } ] )

### dbOwner

The database owner can perform any administrative action on the database. This role combines the privileges granted by the **readWrite**, **dbAdmin** and **userAdmin** roles.

**db.runCommand**( { rolesInfo: { role: "dbOwner", db: "**playground**" }, showPrivileges: true} )

|  |
| --- |
| MongoDB Enterprise > db.runCommand( { rolesInfo: { role: "dbOwner", db: "playground" }, showPrivileges: true} )  {  "roles" : [  {  "role" : "dbOwner",  "db" : "playground",  "isBuiltin" : true,  "roles" : [ ],  "inheritedRoles" : [ ],  "privileges" : [  {  "resource" : {  "db" : "playground",  "collection" : ""  },  "actions" : [  "bypassDocumentValidation",  "changeCustomData",  "changePassword",  "changeStream", |

db.getUsers() show all the db users (login as root)

|  |
| --- |
| MongoDB Enterprise > db.getUsers()  [  {  "\_id" : "admin.dba",  "user" : "dba",  "db" : "admin",  "roles" : [  {  "role" : "dbOwner",  "db" : "playground"  },  {  "role" : "dbAdmin",  "db" : "m103"  }  ]  },  {  "\_id" : "admin.root",  "user" : "root",  "db" : "admin",  "roles" : [  {  "role" : "root",  "db" : "admin"  }  ]  },  {  "\_id" : "admin.security\_officer",  "user" : "security\_officer",  "db" : "admin",  "roles" : [  {  "role" : "userAdmin",  "db" : "admin"  }  ]  }  ]  MongoDB Enterprise > |

# Built-In Roles

## Database User Roles

read

readWrite

## Database Administration Roles

dbAdmin, dbOwner, userAdmin

## Cluster Administration Roles

clusterAdmin

clusterManager

clusterMonitor

hostManager

## Backup and Restoration Roles

backup

restore

## All-Database Roles

readAnyDatabase

readWriteAnyDatabase

userAdminAnyDatabase

dbAdminAnyDatabase

## Superuser Roles

root

## Internal Role

\_\_system

**Lab - Creating First Application User**

In order to create this new user, we have to be logged into your MongoDB server as a user with the privilege to create other users. Luckily, the m103-admin user from previous labs has the root role, and therefore has this privilege.

As a reminder, authenticating to MongoDB as this user can be done with the following command:

mongo --port 27000 -u "m103-admin" -p "m103-pass" --authenticationDatabase "admin"

Once we are logged in, we can create our new user with the following command:

use admin

db.createUser({

user: "m103-application-user",

pwd: "m103-application-pass",

roles: [

{role: "readWrite", db: "applicationData"}

]

})

Notice that we created our user on the admin database, but we gave that user readWrite access to the applicationData database. These are two separate actions - the user **authenticates** against admin and is **authorized** on applicationData.

What I Did

vagrant@m103:~$ mongo admin --host localhost:27000 -u m103-admin -p m103-pass

use admin

db.grantRolesToUser ("m103-application-user", [ { db:"applicationData", role:"readWrite" } ] )

error note: Cree el usuario conun espacio en blanco delante, y luego no podia hacer login

Se puede usar dropUser desde root

MongoDB Enterprise > db.createUser ( {

... user:"m103-application-user",

... pwd:"m103-application-pass",

... roles:[

... {role:"readWrite",db:"applicationData"}

... ]

... }

... )

Successfully added user: {

"user" : "m103-application-user",

"roles" : [

{

"role" : "readWrite",

"db" : "applicationData"

}

]

}

# Server Tools Overview

list mongodb binaries

|  |
| --- |
| find /usr/bin/ -name "mongo\*" |

## [mongostat](https://docs.mongodb.com/manual/reference/program/mongostat/)

The [mongostat](https://docs.mongodb.com/manual/reference/program/mongostat/#bin.mongostat) utility provides a quick overview of the status of a currently running [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) or [mongos](https://docs.mongodb.com/manual/reference/program/mongos/#bin.mongos) instance.

Run [mongostat](https://docs.mongodb.com/manual/reference/program/mongostat/#bin.mongostat) from the system command line, not the [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell.

|  |
| --- |
| mkdir –p ~/first\_mongod  mongod –port 30000 –dbpath ~/first\_mongod –logpath ~/first\_mongod/mongodb.log --fork  mongostat –port 30000 |

## [Mongorestore](https://docs.mongodb.com/manual/reference/program/mongorestore/)

The [mongorestore](https://docs.mongodb.com/manual/reference/program/mongorestore/#bin.mongorestore) program loads data from either a binary database dump created by [mongodump](https://docs.mongodb.com/manual/reference/program/mongodump/#bin.mongodump) or the standard input (starting in version 3.0.0) into a [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) or [mongos](https://docs.mongodb.com/manual/reference/program/mongos/#bin.mongos) instance.

[mongorestore](https://docs.mongodb.com/manual/reference/program/mongorestore/#bin.mongorestore) can create a new database or add data to an existing database. However, [mongorestore](https://docs.mongodb.com/manual/reference/program/mongorestore/#bin.mongorestore) performs inserts only and does not perform updates. That is, if restoring documents to an existing database and collection and existing documents have the same value \_id field as the to-be-restored documents, [mongorestore](https://docs.mongodb.com/manual/reference/program/mongorestore/#bin.mongorestore) will not overwrite those documents.

|  |
| --- |
| mongorestore –drop –port 30000 –u "m103-admin" –p "m103-pass" –authenticationDatabase "admin" dump/  --drop: drop the current collection  dump/: is where is the BSON file |

## [Mongodump](https://docs.mongodb.com/manual/reference/program/mongodump/)

Use mongodump to get BSON dump of a MongoDB collection

[mongodump](https://docs.mongodb.com/manual/reference/program/mongodump/#bin.mongodump) is a utility for creating a binary export of the contents of a database. [mongodump](https://docs.mongodb.com/manual/reference/program/mongodump/#bin.mongodump) can export data from either [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) or [mongos](https://docs.mongodb.com/manual/reference/program/mongos/#bin.mongos) instances.

[mongodump](https://docs.mongodb.com/manual/reference/program/mongodump/#bin.mongodump) can be a part of a [backup strategy](https://docs.mongodb.com/manual/core/backups/#backup-with-mongodump) with [mongorestore](https://docs.mongodb.com/manual/reference/program/mongorestore/#bin.mongorestore) for partial backups based on a query, syncing from production to staging or development environments, or changing the storage engine of a standalone. However, the use of [mongodump](https://docs.mongodb.com/manual/reference/program/mongodump/#bin.mongodump) and [mongorestore](https://docs.mongodb.com/manual/reference/program/mongorestore/#bin.mongorestore) as a backup strategy can be problematic for sharded clusters and replica sets.

|  |
| --- |
| mongodump –port 30000 –u "m103-admin" –p "m103-pass" –authenticationDatabase "admin" –db exampleDB –collection students |

## Mongoexport

The same as mongodump but works with JSON instead.

|  |
| --- |
| mongoexport –port 30000 –u "m103-admin" –p "m103-pass" –authenticationDatabase "admin" –db exampleDB –collection students –o students.json |

## Mongoimport

The same as mongorestore but works with JSON instead.

|  |
| --- |
| mongoimport --port 30000 -u "m103-admin" -p "m103-pass" --authenticationDatabase "admin" --db exampleDB --collection students.json |

# Lab - Importing a Dataset

## Problem:

Now that you have some background about MongoDB's server tools, use mongoimport to import a JSON dataset into MongoDB. You can find the dataset inside the Vagrant box in /dataset/products.json, or in the lesson handout.

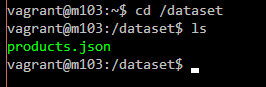
Import the whole dataset with your application's user m103-application-user into a collection called products, in the database applicationData.

When you're finished, run the following validation script in your vagrant and outside the mongo shell and enter the validation key you receive below. If you receive an error, it should give you some idea of what went wrong.

vagrant@m103:~$ validate\_lab\_import\_dataset

## Solution

1-check if the data is really there



2-Iniciar mongod en 270003

mongod -f /etc/mongod-27000-A.conf

3- Connecting to the database

mongo applicationData -u m103-application-user -p m103-application-pass --port 27000 --authenticationDatabase admin

Solution Answer

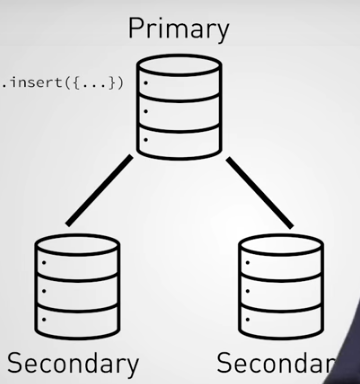
|  |
| --- |
| mongoimport --drop --port 27000 -u "m103-application-user" \  -p "m103-application-pass" --authenticationDatabase "admin" \  --db applicationData --collection products /dataset/products.json |

# What is Replication?

MongoDB uses asynchronous statement-based replication because its platform independent and allows more flexibility within a replica set.

Replication is the concept of manteining multiple copies of your data.

Replica Set.



Failover: When a node take the place of a failure node.

**Binary Replication**: Less Data, Faster

**Statement Based Replication:** Not bound by operating system, or any machine level dependency.

**Idempotence**

**Protocol Version 1 (PV1)** Default protocol. Based on the RAFT protocol.

oplock.

**Arbiter**: Holds no data. Can vote in a election. Cannot became primary. It's mere existence is to serve as a tiebreaker between secondaries in an election.

[Simple Raft Protocol](http://thesecretlivesofdata.com/raft/)

[Raft Consensus Algorithm](https://raft.github.io/).

# Setting Up a Replica Set

**Lecture Instructions**

The configuration file for the first node (node1.conf):

|  |
| --- |
| storage:  dbPath: /var/mongodb/db/node1  net:  bindIp: 192.168.103.100,localhost  port: 27011  security:  authorization: enabled  keyFile: /var/mongodb/pki/m103-keyfile  systemLog:  destination: file  path: /var/mongodb/db/node1/mongod.log  logAppend: true  processManagement:  fork: true  replication:  replSetName: m103-example |

**keyFile**: Enables authentication on our cluster, which mandates that all members of the replica set have to authenticate to each other using a key file that we create here. This is an addition to the client authentication in the previous line.

Creating the keyfile and setting permissions on it:

|  |
| --- |
| $ sudo mkdir /var/mongodb/pki  $ sudo chown vagrant:vagrant /var/mongodb/pki  $ openssl rand -base64 741 > /var/mongodb/pki/m103-keyfile  $ chmod 400 /var/mongodb/pki/m103-keyfile |

Creating the dbPath for node1.

|  |
| --- |
| $ mkdir -p /var/mongodb/db/node1 |

Starting a mongod with node1.conf:

|  |
| --- |
| mongod -f node1.conf |

Copying node1.conf to node2.conf and node3.conf:

|  |
| --- |
| $ cp node1.conf node2.conf  $ cp node1.conf node3.conf |

Editing node2.conf using **vi**:

|  |
| --- |
| $ vi node2.conf |

node2.conf, after changing the **dbpath**, **port**, and **logpath**:

|  |
| --- |
| storage:  dbPath: /var/mongodb/db/node2  net:  bindIp: 192.168.103.100,localhost  port: 27012  security:  keyFile: /var/mongodb/pki/m103-keyfile  systemLog:  destination: file  path: /var/mongodb/db/node2/mongod.log  logAppend: true  processManagement:  fork: true  replication:  replSetName: m103-example |

node3.conf, after changing the **dbpath**, **port**, and **logpath**:

|  |
| --- |
| storage:  dbPath: /var/mongodb/db/node3  net:  bindIp: 192.168.103.100,localhost  port: 27013  security:  keyFile: /var/mongodb/pki/m103-keyfile  systemLog:  destination: file  path: /var/mongodb/db/node3/mongod.log  logAppend: true  processManagement:  fork: true  replication:  replSetName: m103-example |

Creating the data directories for **node2** and **node3**:

|  |
| --- |
| $ sudo mkdir /var/mongodb/db/{node2,node3}  $ mkdir -p /var/mongodb/db/node2  $ mkdir -p /var/mongodb/db/node3  $ sudo chown vagrant:vagrant node2  $ sudo chown vagrant:vagrant node3 |

Starting mongod processes with node2.conf and node3.conf:

|  |
| --- |
| mongod -f node2.conf  mongod -f node3.conf  connecting to node 1:  mongo –port 27011  initiating the replica set  rs.initiate() |

|  |
| --- |
| mongoDB Enterprise > rs.initiate()  {  "info2" : "no configuration specified. Using a default configuration for the set",  "me" : "192.168.103.100:27011",  "ok" : 1  }  MongoDB Enterprise m103-example:OTHER> |

because we have auth enable we need to create an user to connect at that user.

|  |
| --- |
| use admin  db.createUser({  user: "m103-admin",  pwd: "m103-pass",  roles: [  {role: "root", db: "admin"}  ]  }) |

Exiting out of the Mongo shell and connecting to the entire replica set:

|  |
| --- |
| exit  mongo --host "m103-example/192.168.103.100:27011" -u "m103-admin" /  -p "m103-pass" --authenticationDatabase "admin" |

Getting replica set status:

|  |
| --- |
| rs.status() |

Adding other members to replica set:

|  |
| --- |
| rs.add("192.168.103.100:27012")  rs.add("192.168.103.100:27013") |

Getting an overview of the replica set topology:

|  |
| --- |
| rs.isMaster() |

|  |
| --- |
| MongoDB Enterprise m103-example:PRIMARY> rs.isMaster()  {  "hosts" : [  "192.168.103.100:27011",  "192.168.103.100:27012",  "192.168.103.100:27013"  ],  "setName" : "m103-example",  "setVersion" : 3,  "ismaster" : true,  "secondary" : false,  "primary" : "192.168.103.100:27011",  "me" : "192.168.103.100:27011",  "electionId" : ObjectId("7fffffff0000000000000001"),  "lastWrite" : {  "opTime" : {  "ts" : Timestamp(1553251487, 1),  "t" : NumberLong(1)  },  "lastWriteDate" : ISODate("2019-03-22T10:44:47Z"),  "majorityOpTime" : {  "ts" : Timestamp(1553251487, 1),  "t" : NumberLong(1)  }, |

Stepping down the current primary:

|  |
| --- |
| [rs.stepDown()](https://docs.mongodb.com/manual/reference/method/rs.stepDown/) |

Checking replica set overview after election:

|  |
| --- |
| rs.isMaster() |

# Lab - Initiate a Replica Set Locally

Problem:

In this lab you will launch a replica set with three members from within your Vagrant environment. To secure this replica set, you will create a keyfile for your nodes to use when communicating with each other.

For this lab, you must place this keyfile in the /var/mongodb/pki directory and change the permissions so only the owner of the file can read it or write to it:

|  |
| --- |
| sudo mkdir -p /var/mongodb/pki  sudo chown vagrant:vagrant -R /var/mongodb  openssl rand -base64 741 > /var/mongodb/pki/m103-keyfile  chmod 600 /var/mongodb/pki/m103-keyfile |

Your three mongod processes will each have their own configuration file, and now those config files can reference the keyfile you just made. These config files will be similar to the config file from the previous lab, but with the following adjustments:

| **type** | **PRIMARY** | **SECONDARY** | **SECONDARY** |
| --- | --- | --- | --- |
| config filename | mongod-repl-1.conf | mongod-repl-2.conf | mongod-repl-3.conf |
| port | 27001 | 27002 | 27003 |
| dbPath | /var/mongodb/db/1 | /var/mongodb/db/2 | /var/mongodb/db/3 |
| logPath | /var/mongodb/db/mongod1.log | /var/mongodb/db/mongod2.log | /var/mongodb/db/mongod3.log |
| replSet | m103-repl | m103-repl | m103-repl |
| keyFile | /var/mongodb/pki/m103-keyfile | /var/mongodb/pki/m103-keyfile | /var/mongodb/pki/m103-keyfile |
| bindIP | localhost,192.168.103.100 | localhost,192.168.103.100 | localhost,192.168.103.100 |

Note that the mongod does **not** automatically create the dbPath directory. You will need to create this yourself:

|  |
| --- |
| mkdir /var/mongodb/db/{1,2,3} |

Once your configuration files are complete, you can start up the replica set:

1. Start a mongod process with the first config file (on port **27001**). This mongod process will act as the primary node in your replica set (at least, until an election occurs).
2. Now use the mongo shell to connect to this node. On this node, and **only** this node, initiate your replica set with rs.initiate(). Remember that this will only work if you are connected from localhost.
3. Once you run rs.initiate(), the node automatically configures a default replication configuration and elects itself as a primary. Use rs.status() to check the status of the replica set. The shell prompt will read PRIMARY once the initiation process completes successfully.
4. Because the replica set uses a keyfile for internal authentication, clients must authenticate before performing any actions.

While still connected to the primary node, create an admin user for your cluster using the localhost exception. As a reminder, here are the requirements for this user:

* + Role: root on admin database
  + Username: m103-admin
  + Password: m103-pass

1. Now exit the mongo shell and start the other two mongod processes with their respective configuration files.
2. Reconnect to your primary node as m103-admin and add the other two nodes to your replica set. Remember to use the IP address of the Vagrant box 192.168.103.100 when adding these nodes.
3. Once your other two members have been successfully added, run rs.status() to check that the members array has three nodes - one labeled PRIMARY and two labeled SECONDARY.

Now run the validation script in your vagrant and outside the mongo shell and enter the validation key you receive below. If you receive an error, it should give you some idea of what went wrong.

|  |
| --- |
| vagrant@m103:~$ validate\_lab\_initialize\_local\_replica\_set |

Solution:

### Create the file m103-keyfile

|  |
| --- |
| vagrant@m103:~$ **ls -lha /var/mongodb/pki**  total 12K  drwxr-xr-x 2 vagrant vagrant 4.0K Mar 22 07:27 .  drwxr-xr-x 6 root root 4.0K Mar 22 09:17 ..  -r-------- 1 vagrant vagrant 1004 Mar 22 07:27 **m103-keyfile** |

### Editing the configuration files

My configuration files are in **vagrant@m103:~/etc$**

|  |
| --- |
| vagrant@m103:~/etc$ **ls -lha**  total 20K  drwxrwxr-x 2 vagrant vagrant 4.0K Mar 23 05:14 .  drwxr-xr-x 8 vagrant vagrant 4.0K Mar 22 09:50 ..  -rw-rw-r-- 1 vagrant vagrant 323 Mar 23 04:09 **mongod-repl-1.conf**  -rw-rw-r-- 1 vagrant vagrant 323 Mar 23 04:19 **mongod-repl-2.conf**  -rw-rw-r-- 1 vagrant vagrant 323 Mar 23 04:19 **mongod-repl-3.conf** |

### Editing every file with the information given

**mongod-repl-1.conf**

|  |
| --- |
| storage:  dbPath: /var/mongodb/db/1  net:  bindIp: 192.168.103.100,localhost  port: 27001  security:  authorization: enabled  keyFile: /var/mongodb/pki/m103-keyfile-2  systemLog:  destination: file  path: /var/mongodb/db/node1/mongod1.log  logAppend: true  processManagement:  fork: true  replication:  replSetName: m103-repl |

**mongod-repl-2.conf**

|  |
| --- |
| storage:  dbPath: /var/mongodb/db/2  net:  bindIp: 192.168.103.100,localhost  port: 27002  security:  authorization: enabled  keyFile: /var/mongodb/pki/m103-keyfile-2  systemLog:  destination: file  path: /var/mongodb/db/node1/mongod2.log  logAppend: true  processManagement:  fork: true  replication:  replSetName: m103-repl |

**mongod-repl-3.conf**

|  |
| --- |
| storage:  dbPath: /var/mongodb/db/3  net:  bindIp: 192.168.103.100,localhost  port: 27003  security:  authorization: enabled  keyFile: /var/mongodb/pki/m103-keyfile-2  systemLog:  destination: file  path: /var/mongodb/db/node1/mongod3.log  logAppend: true  processManagement:  fork: true  replication:  replSetName: m103-repl |

### Create de db directory

|  |
| --- |
| vagrant@m103:/var/mongodb/db$ **mkdir 1**  vagrant@m103:/var/mongodb/db$ **mkdir 2**  vagrant@m103:/var/mongodb/db$ **mkdir 3**  vagrant@m103:/var/mongodb/db$ **ls -lha**  total 644K  drwxr-xr-x 10 vagrant vagrant 4.0K Mar 23 02:06 .  drwxr-xr-x 6 root root 4.0K Mar 22 09:17 ..  drwxrwxr-x 2 **vagrant vagrant 4.0K Mar 23 02:05 1**  drwxrwxr-x 2 **vagrant vagrant 4.0K Mar 23 02:06 2**  drwxrwxr-x 2 **vagrant vagrant 4.0K Mar 23 02:06 3** |

### Start the replica set

|  |
| --- |
| **vagrant@m103:~/etc$ mongod -f mongod-repl-1.conf**  about to fork child process, waiting until server is ready for connections.  forked process: 12410  child process started successfully, parent exiting |

### Connect to the node and initiate the replica set

*(Now use the mongo shell to connect to this node. On this node, and* ***only*** *this node, initiate your replica set with* ***rs.initiate()****. Remember that this will only work if you are connected from localhost.)*

|  |
| --- |
| vagrant@m103:/var/mongodb/db$ **mongo --port 27001**  MongoDB shell version v3.6.11  connecting to: mongodb://127.0.0.1:27001/?gssapiServiceName=mongodb  Implicit session: session { "id" : UUID("53674aaa-bb9c-43f4-9540-431595ded160") }  MongoDB server version: 3.6.11  MongoDB Enterprise > **rs.initiate()**  {  "info2" : "no configuration specified. Using a default configuration for the set",  "me" : "192.168.103.100:27001",  "ok" : 1  }  MongoDB Enterprise m103-repl:OTHER> |

### Check the status of the replica set

|  |
| --- |
| MongoDB Enterprise m103-repl:OTHER> **rs.status()**  {  "set" : "m103-repl",  "date" : ISODate("2019-03-23T02:34:14.143Z"),  "myState" : 1,  "term" : NumberLong(1),  "syncingTo" : "",  "syncSourceHost" : "",  "syncSourceId" : -1,  "heartbeatIntervalMillis" : NumberLong(2000),  …  MongoDB Enterprise m103-repl:PRIMARY> |

### Create an admin user for your cluster

|  |
| --- |
| MongoDB Enterprise m103-repl:PRIMARY> **use admin**  switched to db admin  MongoDB Enterprise m103-repl:PRIMARY> **db.createUser ( {**  ... **user:"m103-admin",**  ... **pwd:"m103-pass",**  ... **roles: [**  ... **{role:"root",db:"admin"}**  ... **]**  ... **})**  Successfully added user: {  "user" : "m103-admin",  "roles" : [  {  "role" : "root",  "db" : "admin"  }  ]  }  MongoDB Enterprise m103-repl:PRIMARY> |

### Start the other two mongod processes

*(exit the mongo shell and* *start the other two mongod processes with their respective configuration files.)*

|  |
| --- |
| MongoDB Enterprise m103-repl:PRIMARY>**exit**  **vagrant@m103:~/etc$ mongod -f mongod-repl-2.conf**  about to fork child process, waiting until server is ready for connections.  forked process: 12698  child process started successfully, parent exiting  **vagrant@m103:~/etc$ mongod -f mongod-repl-3.conf**  about to fork child process, waiting until server is ready for connections.  forked process: 12729  child process started successfully, parent exiting |

### Reconnect to your primary node as m103-admin

|  |
| --- |
| **vagrant@m103:/var/mongodb/db$ mongo --host "m103-repl/192.168.103.100:27001" \**  **> -u "m103-admin" \**  **> -p "m103-pass" \**  **> --authenticationDatabase "admin"**  MongoDB shell version v3.6.11  connecting to: mongodb://192.168.103.100:27001/?authSource=admin&gssapiServiceName=mongodb&replicaSet=m103-repl  2019-03-23T02:58:16.274+0000 I NETWORK [thread1] **Starting new replica set monitor for m103-repl/192.168.103.100:27001**  2019-03-23T02:58:16.284+0000 I NETWORK [thread1] **Successfully connected to 192.168.103.100:27001** (1 connections now open to 192.168.103.100:27001 with a 5 second timeout)  Implicit session: session { "id" : UUID("76ab5706-5902-4fcd-afee-be5ff0652b92") }  MongoDB server version: 3.6.11  Server has startup warnings:  . . .  **MongoDB Enterprise m103-repl:PRIMARY>** |

### Add the other two nodes to your replica set

|  |
| --- |
| MongoDB Enterprise m103-repl:PRIMARY> **rs.add("192.168.103.100:27002")**  {  "ok" : 1,  "operationTime" : Timestamp(1553310449, 1),  "$clusterTime" : {  "clusterTime" : Timestamp(1553310449, 1),  "signature" : {  "hash" : BinData(0,"lP9M8Ms1H9WYHe0PZi5yRZWiVaA="),  "keyId" : NumberLong("6671405952513605634")  }  }  }  MongoDB Enterprise m103-repl:PRIMARY> **rs.add("192.168.103.100:27003")**  {  "ok" : 1,  "operationTime" : Timestamp(1553310642, 1),  "$clusterTime" : {  "clusterTime" : Timestamp(1553310642, 1),  "signature" : {  "hash" : BinData(0,"npD5LaEgVTwzXQA/CkkWd0HRuN8="),  "keyId" : NumberLong("6671405952513605634")  }  }  } |

### Check that the members array has three nodes

|  |
| --- |
| MongoDB Enterprise m103-repl:PRIMARY> **rs.status()**  {  "set" : "**m103-repl**",  "date" : ISODate("2019-03-23T03:12:01.033Z"),  "myState" : 1,  . . .  "**members**" : [  {  "\_id" : 0,  "name" : "**192.168.103.100:27001**",  "health" : 1,  "state" : 1,  "stateStr" : "**PRIMARY**",  . . .  },  {  "\_id" : 1,  "name" : "**192.168.103.100:27002**",  "health" : 1,  "state" : 2,  "stateStr" : "**SECONDARY**",  "uptime" : 271,  . . .  {  "\_id" : 2,  "name" : "**192.168.103.100:27003**",  "health" : 1,  "state" : 2,  "stateStr" : "**SECONDARY**",  "uptime" : 78,  . . . |

Every thing works fine but the validator is throwing error:

|  |
| --- |
| vagrant@m103:~/etc$ validate\_lab\_initialize\_local\_replica\_set  **User 'm103-admin' not configured correctly - make sure your admin user has root**  **access.**  vagrant@m103:~/etc$ |

**start from scratch**

[**https://discourse.university.mongodb.com/t/removing-a-member-from-replica-set-or-replica-set-as-whole/7209/3**](https://discourse.university.mongodb.com/t/removing-a-member-from-replica-set-or-replica-set-as-whole/7209/3)

# Replication Configuration Document

* JSON Object that defines the configuration options of our eplica set
* Can be configured manually from the shell
* There are set of mongo shell replication helper that make it easier to manage

rs.add

rs.initiate

rs.remove

rs.reconfig

rs.config

### \_id:<string>,

Is set with the name of the replica set, is a string value that matches the server defined replica set. Whenever we start our mongod and we provide a –replSet name to our mongod, meaning that this mongod will belong to the set, or by setting that same name in the configuration file

|  |
| --- |
| #/etc/mongodb.conf  …  replication:  replSetName: 103-example  … |

the same value must match the \_id field of our replica set onfiguration document. En case we have different values from the configuration \_id and the defined replica set name we end up wit an error message. We get an incorrect replica set configuration, stating that we are attempting to initiate the replica set with a different name from which it has been set as –replSet or in the config file. This is a safeguard against incorrect configurations or incorrect adding the server to the incorrect replica sets.

### version: <int>

Incremented every time the current configuration of our replica set changes,if for example we add a node to our replica set

* rs.add ('node3:27017')

and if our version used to be number 1 we increment the value every time we change the tipology, changed a replica set configuration at all pr do something like changing the number of votes:

* config [hists][0]["votes"] = 10
* rs.reconfig(config)

of a given hist, that will automatically increment the version number

### members

Is where the topology of our replica set is defined

**\_id:** Integer, unique identifier of each element in the array. When set, this value cannot be changed.

**host:** host name and port. ex.: ,103:27017

**arbiterOnly, hidden** : Roles of the node within the replica set. Set to false by default.

**priority:** Integer value that allow to seta hierarchywithin the replica set, we can set priorities between 0 and 1000, members with higher priorities tend to be elected in primaries more often a change in the priority of a node wil trigger an election because it will be perceived as a topology change. Setting priority to 0 effectively excludes that member from ever becoming a primary. In case we set a member as a **arbiteronly** that implies that the priority needs to be set to 0. The same would apply for **hidden**. falling to do that wil result in an error where a new replica set configuration is incompatible.

**salveDelay:** Integer value that determines a replication delay interval in seconds, the default is 0. This option wil enable delayed nodes. This delayed members maintain a copy of data reflecting a state in some point in the past applying that delay in seconds. For example, if we have our salveDelay option to 3600 seconds, which means 1 hour, that will mean that such member will be replicating data from the other nodes in the sets that occurred 1 hour ago. By setting this option will imples that your node will be hidden and the priority will be set to 0