Install and Use mongodb nodejs Driver with a mongdb Replica Set Defined in Vagrant

Fernando G. Tinetti*, Agustín Terruzzi

Technical Report TR-BDD-02-2018

III-LIDI, Fac. de Informática, UNLP
*Also with CIC, Prov. de Buenos Aires
Argentina
contact e-mail: fernando@info.unlp.edu.ar

February 2018

Abstract. We document the installation of nodejs and the nodejs driver for mongodb operation from scratch, along with a few javascript scripts operating with a mongodb replica set. Even when there are many similar reports on nodejs and mongodb nodejs driver, we did not find anyone simple enough for a simple environment: nodejs code operating with an already installed mongodb replica set in a cluster-like environment. On one hand, some guides are too similar to a reference manual in that many details are assumed already known and only describe some commands and tools. On the other hand, there are many tutorials describing much more complex environment/s, including infrastructures/*frameworks* for web applications development and deployment, schema-like operations on mongodb, etc., none of which we are interested in, at least initially. We document in this report a step-by-step guide starting with a stand-alone computer (initially in the same LAN) without any infrastructure software other than the network connection and ending that computer being able to operate on a mongodb replica set database using javascript code. The initial goal is to have an environment for experimentation with a replicated mongodb database. Besides, the cluster-like environment is completely defined using the Vagrant tool/software so that it can be replied in any standard desktop/laptop computer at least to experiment with small amounts of data.

1.- Introduction

We are interested in having a nodejs application operating with a mongo replica set already installed and in operation in a cluster-like environment. Actually, the current configuration is described in Fig. 1: a computer with VirtualBox hosting three virtual machines (vm) on which a mongo replica set is running [1]. The whole replica set (three vm) are installed and configured using the Vagrant tool [2]. As explained in [1], only the start of the replica set and adding nodes to the replica set is explicitly made in a mongo console. Since it is expected to be made only once and further nodes can be added at runtime, we have chosen to leave that part of the operation as a manual process in [1] for the sake of simplicity. Even when the scenario in Fig. 1 is rather limited, it is not far from a general-purpose one. More specifically, we replicate the cluster-like environment in which the mongo replica set usually run. The relationship/coupling among vm host and vm guests is clearly far from being that of a production environment, but provides a single environment for carrying out several experiments in an affordable infrastructure. We specifically simplified vm installation, configuration, and network setting/s as well as mongo software installation and startup via the vagrant tool.

This Technical Report can be considered as an upgrade/enhancement of a previous one [3], where we expect to fix some errors/typos and show that the vm environment now defined with Vagarant handling VirtualBox virtual machines in fact does not change the concepts, problems and solutions in having a client of the mongo replica set running in the virtualized environment.

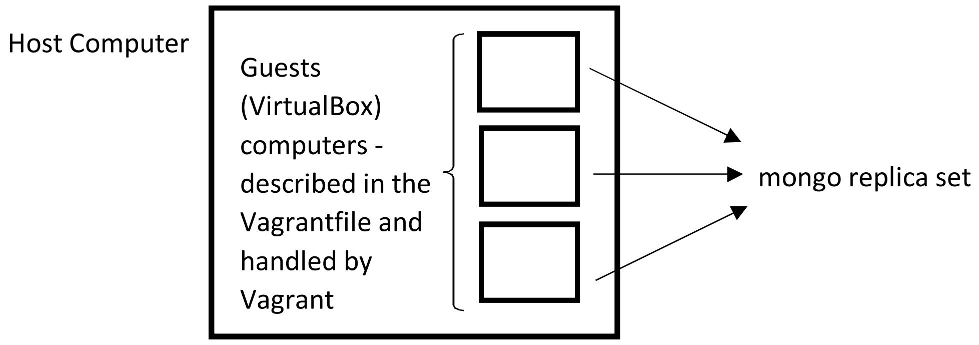
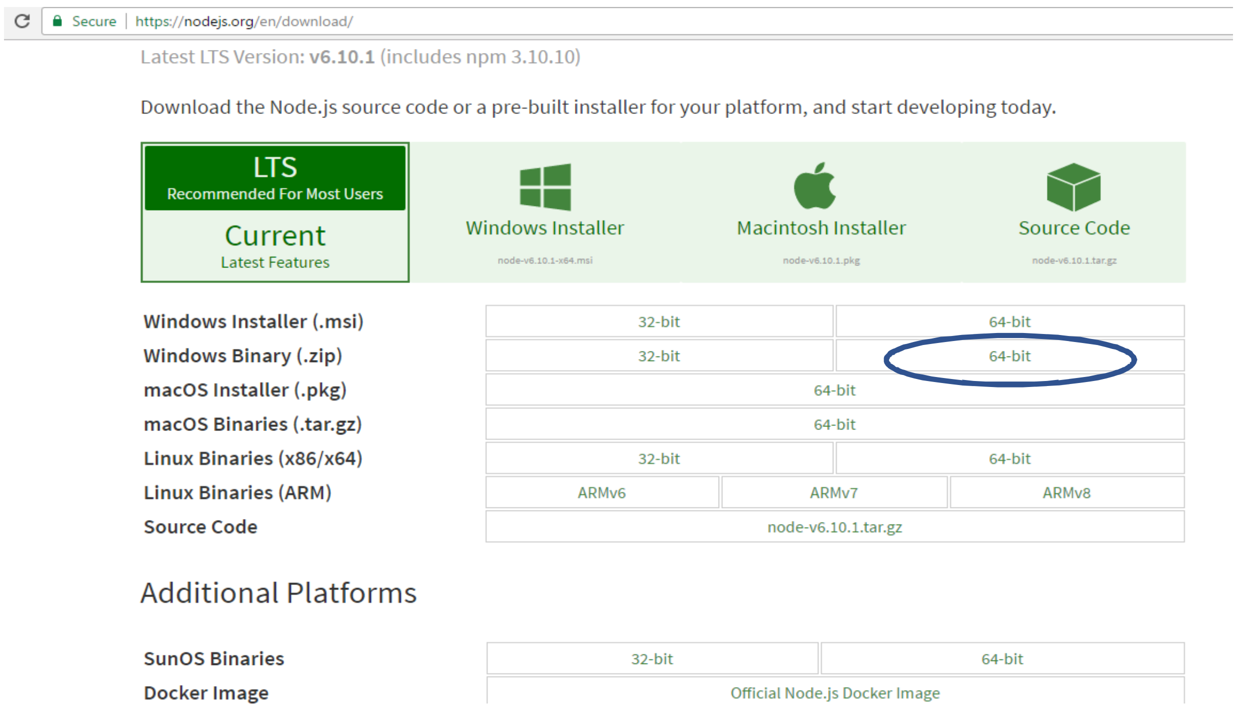

Host Computer
Guests (VirtualBox) computers - described in the Vagrantfile and handled by Vagrant
mongo replica set

Figure 1: Initial Environment.

In this report, we make a step-by-step guide for the installation, configuration, and usage of nodejs and the nodejs mongodb driver in a computer so that a nodejs application is able to operate on a mongodb replica set. Since the host computer we are using has a windows operating system (OS), the installation details will be "restricted" to that OS, but the general configuration and nodejs code is, in fact, platform independent, so we expect this report to be useful in several other scenarios as well. As a "collateral effect" of installing nodejs and the mongodb nodejs driver in a Windows computer, we will have a heterogeneous environment, which is rather usual in the context of distributed systems: the client computer an OS different than that in which the database (a mongo replica set in this specific case) is managed.

2.- Installation and Configuration of nodejs

The nodejs software infrastructure is obtained at its web site [4], as shown in Fig. 2 for the Windows OS without installer (aka binary distribution). Trying to install in a relatively "deep" directory of the directory structures (c:\users\fernando\mydir\catedras\mongodb\nodejs\v2) generates an error, as shown in Fig. 3. The error may be due to the program used to handle the .zip, 7-zip. The version shown in Fig. 3 is 6.10.1, which was the current one at the time the previous report was written, April 2017 [3], the current version is 8.9.4, thus the downloaded file is node-v8.9.4-win-x64.zip. However, the same problem is found. We are aware this is a minor error and unrelated to nodejs, but we want to document it in either case, just as an example of minor details/problems found in this kind of nodejs installation. The work-around to this problem is simple: unzip to a "short" pathname directory (c:\users\fernando\mydir\tmp) and copy de unzipped material to the original/selected one, as shown in Fig. 4 and Fig. 5.


Secure | https://nodejs.org/en/download/

Latest LTS Version: **v6.10.1** (includes npm 3.10.10)

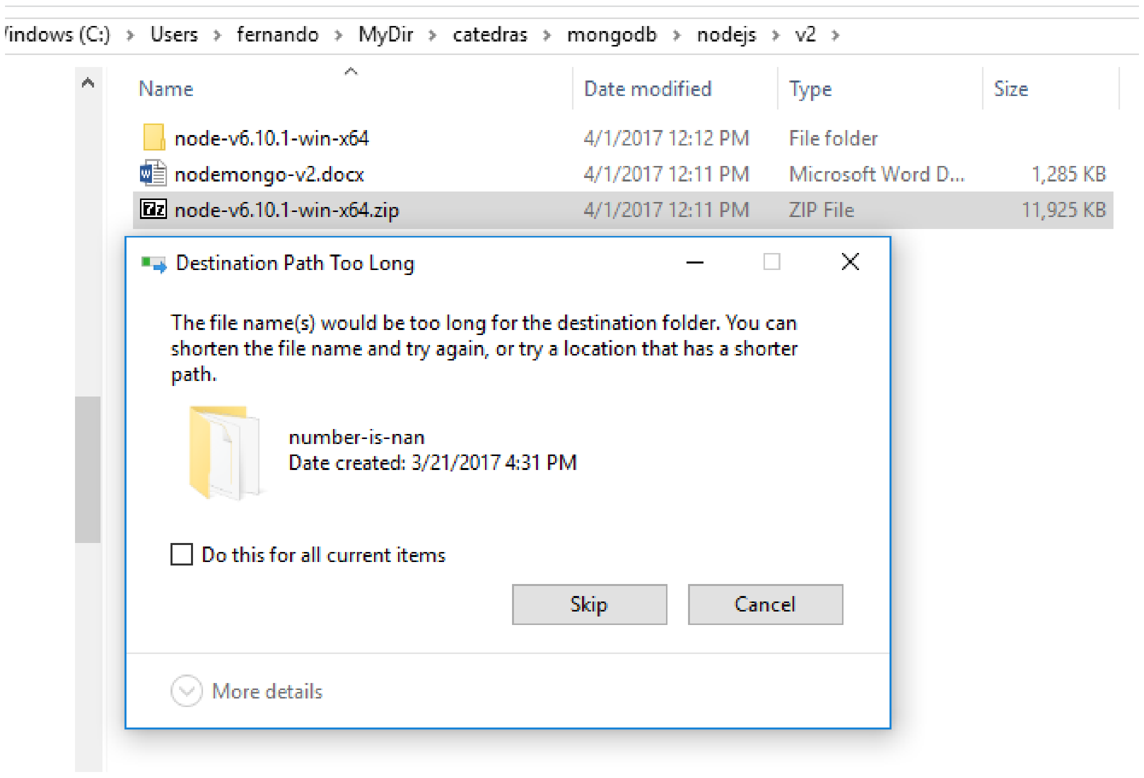
Download the Node.js source code or a pre-built installer for your platform, and start developing today.

Windows Installer (.msi)	32-bit		64-bit
Windows Binary (.zip)	32-bit		[✓ 64-bit]
macOS Installer (.pkg)	64-bit		
macOS Binaries (.tar.gz)	64-bit		
Linux Binaries (x86/x64)	32-bit		64-bit
Linux Binaries (ARM)	ARMv6	ARMv7	ARMv8
Source Code	node-v6.10.1.tar.gz		

Additional Platforms

SunOS Binaries	32-bit	64-bit
Docker Image	Official Node.js Docker Image	

Figure 2: Download Windows Binary nodejs.


'indows (C:) › Users › fernando › MyDir › catedras › mongodb › nodejs › v2 ›

Name	Date modified	Type	Size
node-v6.10.1-win-x64	4/1/2017 12:12 PM	File folder	
nodemongo-v2.docx	4/1/2017 12:11 PM	Microsoft Word D...	1,285 KB
node-v6.10.1-win-x64.zip	4/1/2017 12:11 PM	ZIP File	11,925 KB

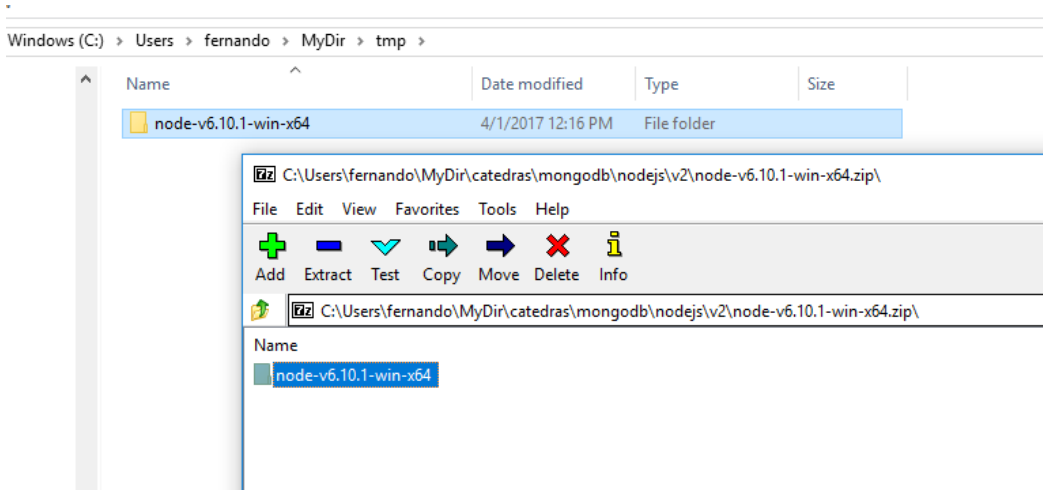
Destination Path Too Long

The file name(s) would be too long for the destination folder. You can shorten the file name and try again, or try a location that has a shorter path.

number-is-nan
Date created: 3/21/2017 4:31 PM

☐ Do this for all current items

Figure 3: Problem in the Binary Unzip and Copy.


Windows (C:) › Users › fernando › MyDir › tmp ›

Name	Date modified	Type	Size
node-v6.10.1-win-x64	4/1/2017 12:16 PM	File folder	

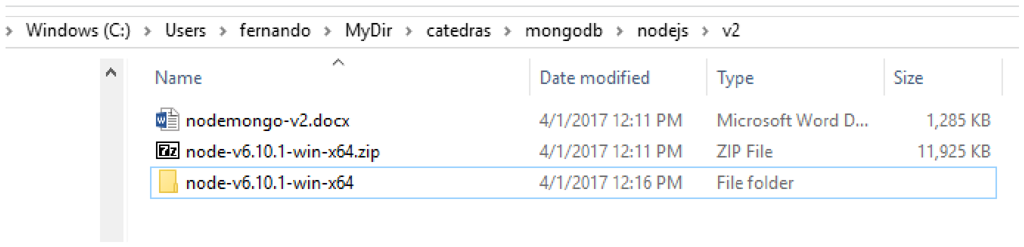
C:\Users\fernando\MyDir\catedras\mongodb\nodejs\v2\node-v6.10.1-win-x64.zip\

File Edit View Favorites Tools Help
Add Extract Test Copy Move Delete Info

C:\Users\fernando\MyDir\catedras\mongodb\nodejs\v2\node-v6.10.1-win-x64.zip\

Name
node-v6.10.1-win-x64

Figure 4: Unzip to a “Short” Pathname Directory.


› Windows (C:) › Users › fernando › MyDir › catedras › mongodb › nodejs › v2

Name	Date modified	Type	Size
nodemongo-v2.docx	4/1/2017 12:11 PM	Microsoft Word D...	1,285 KB
node-v6.10.1-win-x64.zip	4/1/2017 12:11 PM	ZIP File	11,925 KB
node-v6.10.1-win-x64	4/1/2017 12:16 PM	File folder	

Figure 5: nodejs Directory.

Actually, we suggest to have a structure directory such as:
- <base_dir>\node, for installing nodejs binaries
- <base_dir>\wrk, for developing/experimentations

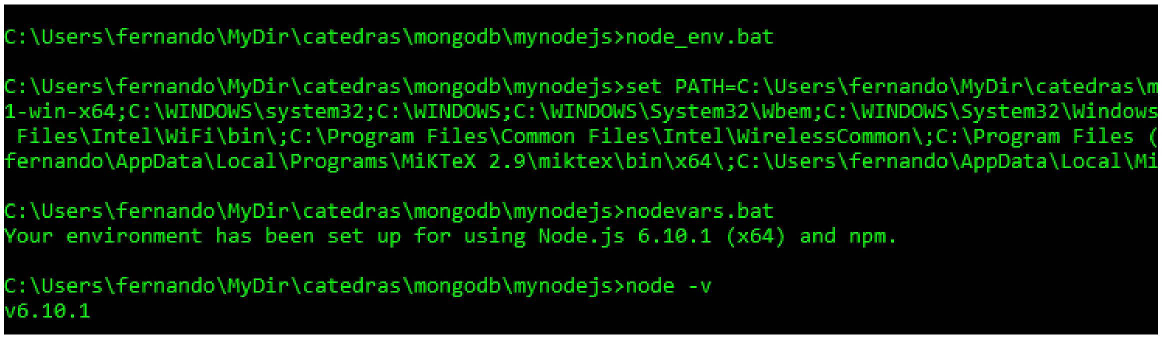
Given that we have chosen to install binaries, we have to set path and other environment variables (those defined by the nodejs in its nodevars.bat file). We have defined a node_env.bat file to do so (note that the version number should be that of the installed binaries):

```
node_env.bat
=========================================================
set PATH=<base_dir>\node\node-v6.10.1-win-x64;%PATH%
nodevars.bat
=========================================================
```

The installation or, at least, the node executable functionality can be verified with the command

```
<base_dir>\wrk>node –v
```

as shown in Fig. 6, where the node command shows the node version installed in the computer. The current version shows “v8.9.4”.



```
C:\Users\fernando\MyDir\catedras\mongodb\mynodejs>node_env.bat

C:\Users\fernando\MyDir\catedras\mongodb\mynodejs>set PATH=C:\Users\fernando\MyDir\catedras\m
1-win-x64;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\Windows
 Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Program Files (
fernando\AppData\Local\Programs\MiKTeX 2.9\miktex\bin\x64\;C:\Users\fernando\AppData\Local\Mi

C:\Users\fernando\MyDir\catedras\mongodb\mynodejs>nodevars.bat
Your environment has been set up for using Node.js 6.10.1 (x64) and npm.

C:\Users\fernando\MyDir\catedras\mongodb\mynodejs>node -v
v6.10.1
```

Figure 6: nodejs Installation Minimum Check.

3.- Installation and First Tests of the mongodb nodejs Driver

The nodejs driver installation is straightforward [6], according to “Install MongoDB Node.js Driver” (remember to run the commands at <base_dir>\wrk, where it is suggested the node_env.bat is also located)

```
<base_dir>\wrk>npm init
<base_dir>\wrk>npm install mongodb@2.2 --save
```

The npm init ask several questions, and all of them can be answered with <return>, and the npm install shows several warnings and the number of packages actually installed, as as shown in Fig. 7.

```
<base_dir>\wrk> npm install mongodb@2.2 --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN wrk@1.0.0 No description
npm WARN wrk@1.0.0 No repository field.

+ mongodb@2.2.34
added 16 packages in 10.466s
```

Figure 7: mongodb nodejs Driver Install.

The current version of the nodejs mongo driver is 3.0 and the mongo version installed by apt-get (as explained in [1]) is 2.6.10, so we decided to install the 2.2 driver version. At this point, a nodejs application is able to connect to a mongodb database, including one handled by a replica set given that the connection is made to the server acting as the replica set primary. Following the guide at [5], we can use the code in connect.js containing

```
connect.js
=================================================================
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert');

var url = 'mongodb://192.168.33.20:27017/test';
MongoClient.connect(url, function(err, db) {
   assert.equal(null, err);
   console.log("Connected to server.");
   db.close();
});
=================================================================
```

The mongo replica set primary node is running at 192.168.33.20 as explained in [1], and listens to connections in the default mongodb server port. The code in connect.js is used as an example for database connection, and having the mongo replica set environment "up&running" as explained in [1], issuing

```
<base_dir>\wrk>node connect.js
```

is successful, providing the output shown in Fig. 8

```
<base_dir>\wrk>node connect.js
Connected to server.
```

Figure 8: A Simple mongodb Connection.

It is worth noting that the connection is possible given that the computer where the javascript code is executed reaches (via the network) the mongodb replica set primary node. More specifically, this is one of the reasons why the VirtualBox host computer is in the same LAN as the vm with the mongodb replica set, given that every vm network is defined by Vagrant using the Vagrantfile as a so called "private_network" in the VirtualBox vm manager.

The guide at [5] is followed in order to insert documents in a mongodb database, i.e. the insert.js code connects to a mongodb and inserts 3 documents, where insert.js is as follows (adapting the code found in [5]).

insert.js

```
============================================================
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert');

var url = 'mongodb://192.168.33.20:27017/test';

var insertDocuments = function(db, callback) {
  // Get the documents collection
  var collection = db.collection('documents');
  // Insert some documents
  collection.insertMany([
    {a : 1}, {a : 2}, {a : 3}
  ], function(err, result) {
    assert.equal(err, null);
    assert.equal(3, result.result.n);
    assert.equal(3, result.ops.length);
    console.log("Inserted 3 documents into the collection");
    callback(result);
  });
}

// Use connect method to connect to the server
MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  console.log("Connected successfully to server");

  insertDocuments(db, function() {
    db.close();
  });
});
============================================================
```

Before using insert.js, the collection does not exist, and the insertMany method actually creates the collection and inserts 3 documents. Before executing the insert.js code the state of the database in terms of existing collection is shown in Fig. 9.

```
myreplica:PRIMARY> db.getCollectionNames()
[ "system.indexes" ]
```

Figure 9: Database Status Before Inserting any Documents.

Then, the code in insert.js inserts 3 documents executing

```
<base_dir>\wrk>node insert.js
Connected successfully to server
Inserted 3 documents into the collection
```

Once the code in insert.js is executed, the database contains a collection named “documents” with 3 documents, as shown in Fig. 10.

```
myreplica:PRIMARY> db.documents.find().count()
3
```

Figure 10: Number of Documents in Collection “documents”.

As explained above, the javascript client will be able to connect and operate on the mongodb database replica set only if it uses the mongodb replica set primary node. One of the main disadvantages of this type of clients is that the server working as the replica set primary node is defined at runtime. In our experiments, the node in which the replica set is “initiate()d” becomes the primary node, but it would change at runtime depending on runtime conditions (e.g. due to node reset/shutdown, and/or network errors). If the server explicitly referenced in the .js files is shutdown, the nodejs code reports error/s such as that in Fig. 11 (font has been reduced in order to void too many text line wraparounds):

```
<base_dir>\wrk>node connect.js

<base_dir>\wrk\node_modules\mongodb\lib\mongo_client.js:338
        throw err
        ^
AssertionError: null == { MongoError: failed to connect to server [192.168.0.160:27017] on first connect
[MongoError: connect ETIMEDOUT 192.168.0.160:27
   at …nodejs\connect.js:6:10
   at connectCallback (<base_dir>\wrk\node_modules\mongodb\lib\mongo_client.js:428:5)
   at <base_dir>\wrk\node_modules\mongodb\lib\mongo_client.js:335:11
   at _combinedTickCallback (internal/process/next_tick.js:73:7)
   at process._tickCallback (internal/process/next_tick.js:104:9)
```

Figure 11: Error when trying to Connect to a Non-Running Server.

Otherwise, if the server explicitly referenced in the .js file is not the replica set primary node, the nodejs code reports error/s too (because it will try to modify a database in a replica set node other than the primary one:

```
…>node insert.js

…\lib\utils.js:123
   process.nextTick(function() { throw err; });
                               ^
AssertionError: { MongoError: not master
   at Function.MongoError.create (C:\Users\fernando\MyDir\catedras\mongodb\mynodejs\node-v6.10.1-win-x == null
   at …\insert.js:30:12
   at … \node_modules\mongodb\lib\collection.js:431:32
   at handleCallback (…\node_modules\mongodb\lib\utils.js:120:56)
   at …\node_modules\mongodb\lib\collection.js:736:20
   at …\node_modules\mongodb\node_modules\mongodb-core\lib\connection\pool.js:461:18
   at _combinedTickCallback (internal/process/next_tick.js:73:7)
   at process._tickCallback (internal/process/next_tick.js:104:9)
```

Actually, the second error reported at the nodejs client side is similar to that reported at the mongo console of the replica set secondary nodes, as shown in Fig. 12, where the error report indicates a possible solution for reading/querying the database on secondary nodes (slaveOk=true).

```
myreplica:SECONDARY> db.documents.find()
error: { "$err" : "not master and slaveOk=false", "code" : 13435 }
```

Figure 12: Error at Replica Set Secondary Node Trying to Query the Database.

The description so far covers only basic operations (there are many more details, such as those in [6]) and it does not use/take advantage of a mongo replica set at the client side. The replica set provides high availability in case of some network and server errors and possible performance enhancement for reading operations. The given code examples do not take advantage of such high availability facilities because the connection is made with a single mongodb server, not with a replica set. The next section provides some simple examples and explanations on a nodejs client (and setting apart the javascript code, any other client) connecting to a replica set, not a single server. So far, the only "extra" functionality at the server side having a replica set is that the data is replicated/copied in all the mongo replica set secondary nodes.

4.- A nodejs Client of a mongodb Replica Set "Server"

Connecting a nodejs application to a mongodb replica set [7] implies using a specific URI format [8]. The nodejs driver has specific options too, as described at [9], some of which are related to replica set connections. The minimum suggested information in the URI provided for the connection to a replica set would be the list of servers (referred to as "a seedlist of replica set members") and the replica set name. Taking into account we have a mongodb replica set "up & running" as described in [1], i.e.:

- Computers IPs 192.168.33.20, 192.168.33.21, and 192.168.33.22,
- Every mongodb server (mongod) using the server default port,
- The replica set name is "myreplica",
- The database is 'test'

the URL we should use for the client connection should be
'mongodb://192.168.33.20,192.168.33.21,192.168.33.22/test?replicaSet=myreplica'

Thus, the “complete” javascript code for a client connection to a mongodb replica set, connectrepl.js, would be as follows
connectrepl.js

```
============================================================
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert');

var url = 'mongodb://192.168.33.20,192.168.33.21,192.168.33.22/test?replicaSet=myreplica';
MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  console.log("Connected to server.");
  db.close();
});
============================================================
```

which is identical to the one given as connect.js except for the url value (now referencing the replica whole set, including the replica set name). Take into account that the so called seedlist of replica set members is given without any identification of primary and/or secondary nodes, as expected, since it is not possible to know in advance which of the nodes will be the primary one. There is another detail for the javascript code to work in the host containing the vm environment, and it is related to the way mongo “recognizes” replica set nodes in the replica set. If the rs.status() is run in the mongo console, among the output is

```
myreplica:PRIMARY> rs.status()
{
        "set" : "myreplica",
        …
        "members" : [
                {
                        "_id" : 0,
                        "name" : "mongo1:27017",
                        …
                        "stateStr" : "PRIMARY",
```

i.e., the replica set primary “recognizes itself” as mongo1. However, the host computer (the one containing the vm environment) does not recognizes that host name, and the client connection fails. Thus, in the host computer it is necessary to have the correct hostname resolution. In the specific case of having a windows computer, we can add the lines

```
192.168.33.20     mongo1
192.168.33.21     mongo2
192.168.33.22     mongo3
```

to the file C:\Windows\System32\drivers\etc\hosts. At this point, we are able to run the code for a client connection to a replica set

```
…>node connectrepl.js
Connected to server.
```

And we are able to experiment different features of the mongo replica set. Shutting down the mongo1 vm with

```
>vagrant halt mongo1
==> mongo1: Attempting graceful shutdown of VM...
```

we are able to see the replica set status at the mongo2 vm with

```
>vagrant ssh mongo2
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-87-generic x86_64)
…
vagrant@mongo2:~$ mongo
MongoDB shell version: 2.6.10
connecting to: test
myreplica:PRIMARY> rs.status()
{
        "set" : "myreplica",
        …
        "members" : [
                {
                        "_id" : 0,
                        "name" : "mongo1:27017",
                        …
                        "stateStr" : "(not reachable/healthy)",
                        …
                },
                {
                        "_id" : 1,
                        "name" : "192.168.33.21:27017",
                        …
                        "stateStr" : "PRIMARY",
                        …
                },
                {
                        "_id" : 2,
                        "name" : "192.168.33.22:27017",
                        …
                        "stateStr" : "SECONDARY",
                        …
                }
        ],
        "ok" : 1
}
```

i.e., the replica set identifies that one of the nodes, mongo1, is down ("not reachable/healthy"), which was the initial PRIMARY node, and the current replica set is operating with 2 nodes: 192.168.33.21 (the current PRIMARY node) and 192.168.33.22 (as a SECONDARY node). In this scenario, we can try to use the same connectrepl.js and verify whether it works or not:

```
…>node connectrepl.js
Connected to server.
```

And works as expected, because it connects to a "replica set", not a specific mongo server. Besides, we are able to operate with the replica as long as there is a PRIMARY node (as it is the current state). With a small change in the url used for the mongo client connection, we can use the following javascript insertion code

insertrepl.js

```
=======================================================================
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert');

//var url = 'mongodb://192.168.33.20:27017/test';
var url = 'mongodb://192.168.33.20,192.168.33.21,192.168.33.22/test?replicaSet=myreplica';

var insertDocuments = function(db, callback) {
  // Get the documents collection
  var collection = db.collection('documents');
  // Insert some documents
  collection.insertMany([
    {a : 1}, {a : 2}, {a : 3}
  ], function(err, result) {
    assert.equal(err, null);
    assert.equal(3, result.result.n);
    assert.equal(3, result.ops.length);
    console.log("Inserted 3 documents into the collection");
    callback(result);
  });
}

// Use connect method to connect to the server
MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  console.log("Connected successfully to server");

  insertDocuments(db, function() {
    db.close();
  });
});
=======================================================================
```

And executing the code produces

```
<base_dir>\wrk>node insert.js
Connected successfully to server
Inserted 3 documents into the collection
```

in the host computing, which runs the client application, and the database is changed accordingly:

```
myreplica:PRIMARY> db.documents.find().count()
6
myreplica:PRIMARY>
```

(note the collection had 3 documents prior to the execution of the insertion code).

By default, the mongodb client operates with the replica set primary node, i.e. read and write database operations are made in the replica set primary node. Among the settings that can be changed, read operations can be made in replica set secondary nodes if required, given the proper connection setting/s option/s, e.g.
Setting the readPreference value to one of

ReadPreference.PRIMARY
ReadPreference.PRIMARY_PREFERRED
ReadPreference.SECONDARY
ReadPreference.SECONDARY_PREFERRED
ReadPreference.NEAREST

(given as a MongoClient.connect function option [9]).

5.- Conclusions and Further Work

We have set up a nodejs execution environment, installed the mongodb driver so that it is possible to connect to a mongodb database, and show how to connect to a mongodb replica set in particular (i.e. a replicated mongodb database). The nodejs installation and project management mostly through nom has a lot of details not covered in this step-by-step guide. Instead, we have restricted to the simplest environment/ installation in a Windows computer. We think each nodejs developer has a predefined development environment and it would be almost impossible to cover them all.

We have used a mongodb replica set already installed so that it was possible to show how to make connections and how the client is transparently managed by the mongodb replica set independently of the details of the current replica set primary node. We know this is just the minimum detail covered in this report regarding mongodb replica set databases. Actually, this step-by-step guide allows to have an experimentation environment we think would be very useful to analyze at least mongodb high availability facilities to clients (nodejs clients in this case). In this context, the guide we presented is only about infrastructure, the interesting details will be obtained by thoroughly thought through experiments that should provide insight on specific performance details.

Using the Vagrant software for vm management simplifies several experiments/practical tasks. The concepts behind replica sets, experiments, and nodejs clients are independent of Vagrant, however. We take advantage of vm installation, setup, and Vagrant general vm handling.

References

[1] Fernando G. Tinetti, Agustín Terruzzi, "Install a mongodb Replica Set in a Virtual Environment", Technical Report BDD-01-2018, February 2018.
http://fernando.bl.ee/reptec/2018-repl-mongo.pdf

[2] HashiCorp Vagrant, Development Environments Made Easy
https://www.vagrantup.com/

[3] Fernando G. Tinetti, Agustín Terruzzi, “Install and Use mongodb nodejs Driver with a mongdb Replica Set”, III-LIDI, Facultad de Informática, UNLP, April 2017, Technical Report TR-BDD-02-2017. Disponible en http://fernando.bl.ee/

[4] 2017 Node.js Foundation, “Download | Node,js”,
https://nodejs.org/en/download/

[5] MongoDB, Inc., Quick Start,
http://mongodb.github.io/node-mongodb-native/2.2/quick-start/quick-start/

[6] Tutorials, CRUD Operations,
http://mongodb.github.io/node-mongodb-native/2.2/tutorials/crud/

[7] Tutorials, Connect to MongoDB,
http://mongodb.github.io/node-mongodb-native/2.2/tutorials/connect/

[8] MongoDB, Inc., Connection String URI Format, MongoDB Manual 3.4,
https://docs.mongodb.com/manual/reference/connection-string/

[9] URI Connection Settings,
http://mongodb.github.io/node-mongodb-native/2.2/reference/connecting/connection-settings/