

Install a mongodb Replica Set in a Virtual Environment

Fernando G. Tinetti*, Agustín Terruzzi

Technical Report TR-BDD-01-2018

III-LIDI, Fac. de Informática, UNLP

*Also with CIC, Prov. de Buenos Aires

La Plata, Argentina

contact e-mail: fernando@info.unlp.edu.ar

February 2018

Abstract. In this report, we enhance and document a simple installation of a mongodb replica set for an initial environment of 3 nodes: 1 primary node and 2 secondary nodes from scratch. Even when there are many web sites and tutorials on mongodb replica sets, we did not find anyone simple enough for a simple environment, as well as with some methodological way of going from a non-replicated environment to a replicated one and vice-versa. Having to go back and forth from replicated to non-replicated environment enables several experiments on which we are interested in. We are particularly interested in an environment focused on the replica set (functionality and runtime) performance for analysis not affected by other details such as web applications, schema control of mongodb NoSQL databases, and web development frameworks, which are usually included in most (it not all) mongodb tutorials. We have simplified installation and configuration of a previous environment by using the Vagrant tool.

1.- Introduction

Our main goal is to have a step-by-step guide for having a three-node replicated mongo NoSQL database, as shown in Fig. 1 [1]. In terms of mongo replication, we configure a three-node replica set, where one of them is the so called Primary node (i.e. the authoritative and up-to-date one) and the two others are (peer) Secondary nodes, which replicate the Primary node write operations [2]. We expect to experiment with a relatively small set of data, so that the nodes will not require large amounts of processing power, and RAM and disk space.

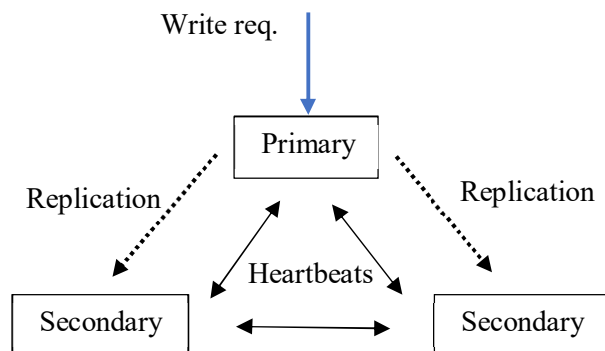


Figure 1: Three-node Replica Set, One Primary Node, ad Two Secondary Nodes.

We will use virtual machines (vm) in order to build up the whole replica set system, so that the environment is closer to a real one, in which the actual nodes usually are vm too. However, we will use a single host computer in which the three virtual guests will run. Initially, the host computer will provide only connectivity (networking), but it could be later used for running a client in a fourth/different computer, other than the three nodes handling the replica set. A 64-bit operating system will be installed in every vm, since it is required by the current mongodb version. Actually, we use a standard distribution: a 64-bit Ubuntu.

While VirtualBox will be used for handling the vm guests' environment, we have simplified installation and setting the vm via the Vagrant tool [10] [5]. As a matter of fact, this Technical Report can be considered as the enhancement of a previous one [11] from two points of view:

- Simple installation and setting of virtual machines. The complete environment will be defined via the so called "Vagrantfile" [6], without any manual setting/configuration of virtual machines and/or network interfaces. One of the most time-consuming and manual tasks for vm configuration in VirtualBox will be done automatically via the so called "Vagrant Boxes". i.e. preconfigured, preinstalled virtual machine images, from which Vagrant clones new ones [7].
- Just about 0.5GB of RAM memory required per virtual machine (vm). We expect to have several (at least three) virtual machines "up&running" in a relatively small/standard desktop/laptop computer (e.g. one with 4 or 8 GB RAM).

Besides, the complete environment is configured so that the host computer has network connectivity to the vm and the each vm has access to the internet (even when it is not necessary for most of the tasks and experiments). In the end, a cluster-like environment is built and deployed for a mongo replica set experimentation.

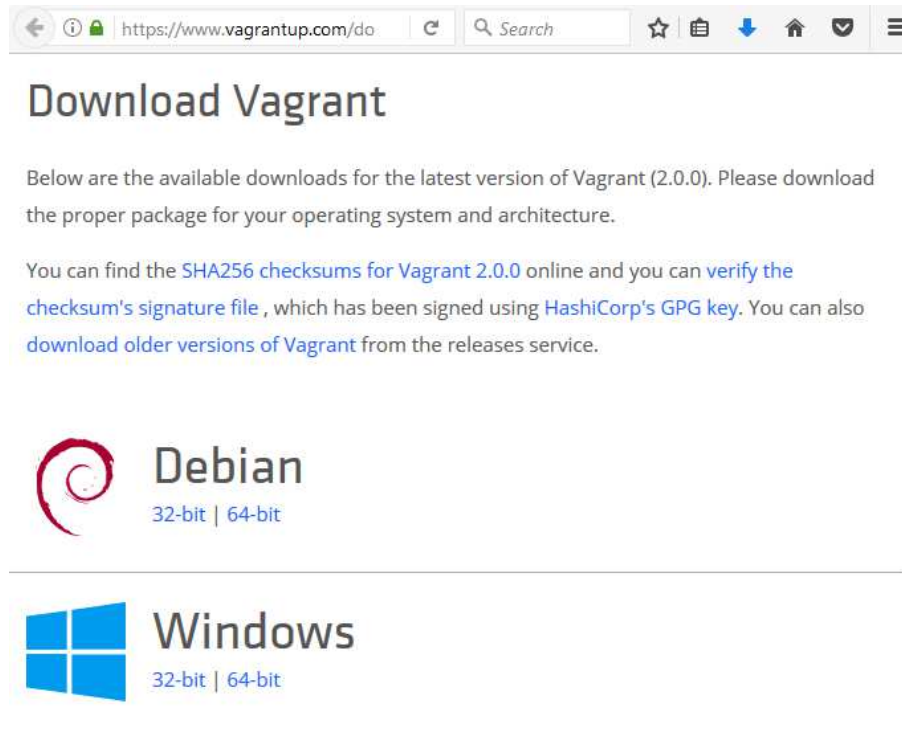
2.- Basic Virtual Environment

We will use a Windows 10 computer as the host running the virtualized environment basically provided by VirtualBox. Furthermore, we will handle the whole virtual environment via Vagrant, as explained above. Thus, we will be able to define the environment (e.g. vm and network connections) with a Vagrantfile, which in fact can be considered as "virtual hardware defined declaratively" with the addition of (pre)installed software. In this way, there will be no need to manually install and configure hardware, network connections and base software (such as the mongodb manager). We will also take advantage of the large number of predefined "Vagrant boxes", which provide functional vm with small resource requirements (mostly from the point of view of RAM requirements). Having the host with Windows 10 implies several (interesting) characteristics of the whole experimentation environment:

- Proof of concept: having a running environment in Windows 10, it should be possible to be ported to a (more usual) Linux one.
- Heterogeneity, having the mongodb replica set in Linux vm and the host with another OS, one of the simplest experiment for a replica set client would be that of having the client in the host computer making requests to the replica set.
- Even small/standard laptop and desktop computers would be able to be used for the small/simple experiments, it would not be required a "server like" computer for verifying/testing simple functionality.

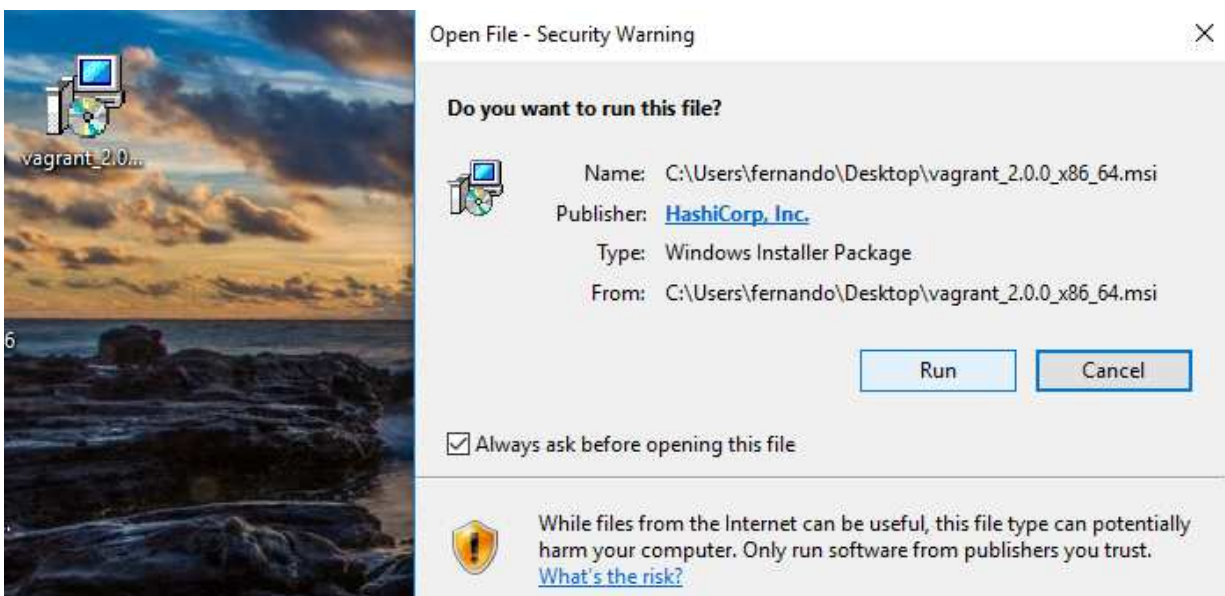
VirtualBox and Vagrant should be installed in the host computer (i.e. they are required for these experiments and environment testing). In the case of Vagrant, it is suggested that the software is installed with default settings (installation directory, in particular), since we have found errors in handling vm when changing defaults. Solamente a modo de ejemplo, se detallan los pasos más relevantes de la instalación de Vagrant:

Download: <https://www.vagrantup.com/downloads.html>



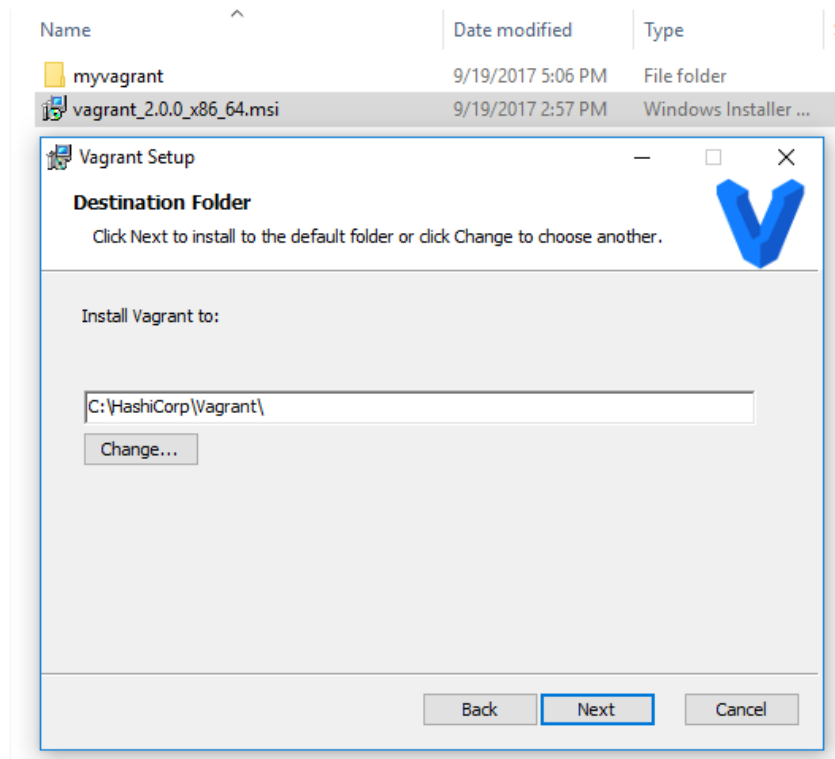
The screenshot shows a web browser window with the address bar containing <https://www.vagrantup.com/downloads.html>. The page title is "Download Vagrant". Below the title, there is a paragraph: "Below are the available downloads for the latest version of Vagrant (2.0.0). Please download the proper package for your operating system and architecture." Another paragraph follows: "You can find the [SHA256 checksums for Vagrant 2.0.0](#) online and you can [verify the checksum's signature file](#), which has been signed using [HashiCorp's GPG key](#). You can also [download older versions of Vagrant](#) from the releases service." Below this text, there are two sections for operating systems: "Debian" with subtext "32-bit | 64-bit" and "Windows" with subtext "32-bit | 64-bit".

Using the installation software downloaded from the Vagrant site:



The screenshot shows a Windows Security Warning dialog box titled "Open File - Security Warning". The dialog asks "Do you want to run this file?". The file details are: Name: C:\Users\fernando\Desktop\vagrant_2.0.0_x86_64.msi, Publisher: [HashiCorp, Inc.](#), Type: Windows Installer Package, and From: C:\Users\fernando\Desktop\vagrant_2.0.0_x86_64.msi. There are "Run" and "Cancel" buttons. At the bottom, there is a checkbox labeled "Always ask before opening this file" which is checked. A warning icon and text state: "While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. [What's the risk?](#)"

And installing in the default directory:



We will use a small yet “complete” Vagrantfile. We use the term “complete” in the sense that we will have three interconnected computers “up&running”. The Vagrantfile contains:

```
base_network = "192.168.33."
base_host = 20

Vagrant.configure("2") do |config|
  3.times do |num|
    config.vm.define ("mongo%01d" % (num + 1)) do |machine|
      machine.vm.box = "bento/ubuntu-16.04"
      machine.vm.network "private_network", ip: "#{base_network}#{base_host + num}"

      machine.vm.provider "virtualbox" do |vb|
        vb.memory = "512"
      end

      machine.vm.provision "shell", inline: <<-SHELL
        # Get updated urls/packages sources?
        apt-get update

        # The joe editor
        apt-get install -y joe

        # The alias for ls
        grep -q "alias ls='ls -la'" /home/vagrant/.bashrc || echo -e "\nalias ls='ls
-la'" >> /home/vagrant/.bashrc
      SHELL
    end
  end
end
```

Important details about the Vagrant file above:

- The network for every vm will be defined “stating at” 192.168.33.20, i.e. if 3 vm are created, then the IPs will be 192.168.33.20, 192.168.33.21, and 192.168.33.22 respectively. This is defined with aif of the variables defined in the first two Vagrantfile lines:

```
base_network = "192.168.33."  
base_host = 20
```

- The line
`3.times do |num|`
implies creating 3 identical vm, with the exception of the vm network configuration, which is defined by the line

```
machine.vm.network "private_network", ip: "#{base_network}#{base_host + num}"
```

(where the variables assigned in the first two lines are used) for each vm

- Every vm will be based on the vagrant box “bento”, according to
`machine.vm.box = "bento/ubuntu-16.04"`
which is a “small” (mostly with small RAM requirement) Ubuntu installation.
- And every vm will have
 - 512 MB RAM
 - The joe editor
 - An alias for the ls command

With the command

`vagrant up`

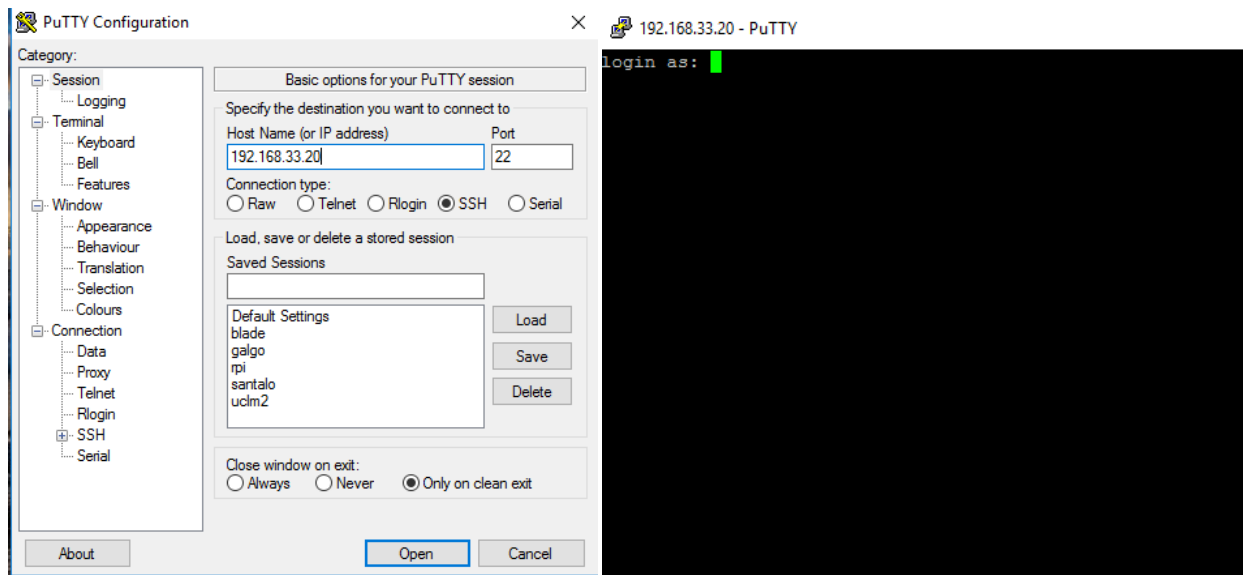
in the same directory where the Vagrantfile is located, the three vm are created and installed. As a matter of fact, the whole process took less than 10 minutes (and a lot of output text) in the command terminal where the “vagrant up” was used. Thus, vagrant solves automatically (by reading the Vagrantfile) and quickly the whole process of creating and running the three vm (a process called “provisioning” [8], which is made for the first execution of the vagrant up command). For example, the three vm now appear at the VirtualBox interface:



The name of each vm has the form `<s1>_<s2>_<num1>_<num2>`, where:

- `<s1>` is the directory name where the Vagrantfile is located, which in this example is 3bentos.
- `<s2>` is the name given to the vm in the line
`config.vm.define ("mongo%01d" % (num + 1)) do`
where the variable num will be assigned the sequence 0, 1, and 2 thus the names mongo1, mongo2, and mongo3.
- `<num1>` and `<num2>` are defined by vagrant itself the first time “vagrant up” is run.

Furthermore, if we use PuTTY (or a similar tool, or just a plain ssh command) in the host computer we are able to verify that the vm are actually running, i.e. they are not “only inside vagrant”, but they are actual (VirtualBox) vm:



There are several details to take into account about the command **vagrant up**:

- As explained above, the first time it is run it generates the complete environment defined in the Vagrantfile. It implies cloning the specified “vagrant box”, configuring the vm (e. g. assigning RAM memory space and network-related configuration definitions), and installing software (usually via one or more apt-get commands in case of Ubuntu based boxes as in the example above).
- The vm are installed only the first time, subsequent use of **vagrant up** will only start the already installed vm (described in the Vagrantfile).
- In the usual case in which **vagrant up** works as expected, we are able to ssh to each vm, e.g. with **vagrant ssh mongo1** (using the Vagrantfile given above), which will provide a **mongo1** vm terminal, and **vagrant halt** is used to shutdown every vm.
- With low probability, it is possible the **vagrant up** command does not work as expected and the vm are turned on (booted up) but not completely functional (e.g. ssh to any of them does not work). In that case, recovering is possible with the sequence of commands:
vagrant suspend
vagrant halt
- Other vagrant useful commands would be
 - **vagrant status**, which shows at least whether the vm are running or not
 - **vagrant destroy**, which completely removes the virtual environment described in the Vagrantfile, every vm is removed from the VirtualBox environment too.

All the vagrant commands should be used in the directory containing the Vagrantfile. The bento vagrant box has a predefined (Ubuntu) sudoer user, **vagrant** (with password **vagrant**), and it is expected that most (if not all) vagrant boxes have a similar one.

In part for “consistency” and in part because it will be needed for the replica set operation, we are going to explicitly define the hostname of each vm (since they are all named “vagrant” by default) and add the corresponding ips to the **/etc/hosts** in all vm. More specifically, at **mongo1** vm:

```
sudo echo mongo1 > /etc/hostname
```

At **mongo2** vm:

```
sudo echo mongo2 > /etc/hostname
```

And at mongo3:

```
sudo echo mongo3 /etc/hostname
```

Besides, in all the vm the three IPs should be added in the /etc/hosts:

```
192.168.33.20 mongo1
```

```
192.168.33.21 mongo2
```

```
192.168.33.22 mongo3
```

3.- Install mongod and Configure the Replica Set

We manually install mongod and configure a mongo replica set only for explanation purposes, but it is expected to be included in the Vagrantfile, so that the vm are automatically installed, configured, and started ready-to-go for experimentation. Once the vm are turned on with

```
vagrant up
```

and we have a terminal in one of the vm, e.g. mongo1, with

```
vagrant ssh mongo1
```

Installation of the mongod NoSQL DBM is straightforward:

a) Install package:

```
$ sudo apt-get install mongod
```

b) Test the initial install:

```
$ mongo
```

```
(exit)
```

The replica set configuration is straightforward too, but involves several steps, as follows. The first step to configure the replica set is to stop the mongod server (mongod process), because it should start configured for replication, which is not made by default (at the install step):

1) \$ sudo service mongod stop

The replica set configuration must be defined in the mongo configuration file, as well as allowing connections to the mongod server from computers other than localhost:

2) Edit (maybe save a copy of the original file is a good idea before changing it) /etc/mongod.conf (sudo <editor> /etc/mongod.conf) for defining the replica set name and commenting out the bind_ip line:

```
=====
# replica name...
replSet = myreplica
...
#bind_ip = 127.0.0.1
...
=====
```

And that's all what we need to start the mongod servers at each one of the computers:

3) \$ sudo service mongod start

3.1 Script for Starting with a Replica Set Configuration

Given that the replica set environment will be used for many different experiments/scenarios/tests, we will make a simple script for starting the mongod with a replica set. The step-by-step guide is as follows.

1) Copy the original mongo configuration file in order to save a copy

```
$ sudo cp /etc/mongodb.conf /etc/mongodb.conf.orig
```

2) Prepare a configuration file to be specifically used for the mongo replication set

```
$ sudo cp /etc/mongodb.conf /etc/mongodb.conf.repl
```

And edit the replica set configuration file as described earlier (sudo <editor> /etc/mongodb.conf.repl) for defining the replica set name and commenting out the bind_ip line:

```
=====
# replica name...
replSet = myreplica
...
#bind_ip = 127.0.0.1
...
=====
```

3) Create the script for restarting the mongo operation with the replica set, we will call such script as “repl”:

```
=====
sudo cp /etc/mongodb.conf.repl /etc/mongodb.conf
sudo service mongodb restart
=====
```

As a result of steps 1) and 2) above, there will be three mongo configuration files:

/etc/mongodb.conf (the one to be taken into account in mongo startup)

/etc/mongodb.conf.orig (the one created at mongo install, with no replication set whatsoever)

/etc/mongodb.conf.repl (the one created for mongo replication set startup)

Installation of the three vm would be straightforward, and we can take advantage of other vagrant vm facilities, mostly the shared directory, aka “Synced Folders” in vagrant terminology [9]. By default, the directory in where the Vagrantfile is located and where the vagrant up command is executed (also called the vagrant “project directory”) is shared and synchronized to /vagrant directory of the vm filesystem. Thus, the /vagrant will be shared among the vm and the host computer. Thus, in the recently installed computer, we can use

```
$ cp repl /vagrant
```

```
$ cp /etc/mongodb.conf.repl /vagrant
```

```
$ cp /etc/mongodb.conf.orig /vagrant
```

so that we will install mongodb and reuse the repl script and the mongodb configuration files in every vm with the following sequence of commands (the first one is issued in the host computer):

```
vagrant ssh mongo2
```

```
$ sudo apt-get install mongodb
```

```
$ sudo service mongodb stop
```

```
$ sudo cp /vagrant/mongodb.conf.repl /etc
```

```
$ sudo cp /vagrant/mongodb.conf.orig /etc
```

```
$ cp /vagrant/repl .
```

and the same sequence for the mongo3 vm (with the corresponding vagrant ssh mongo3 command).

3.2 Starting the Replica Set Operation

Once every mongodb process (mongod) starts with the replica set option, they are all peers and they are ready to start a NoSQL database replication, i.e.:

- a) Identify peers, all nodes in the replica set (not given yet).
- b) Vote a primary node among those in the same replica set.
- c) Recognize non-primary node/s in the same replica set as secondary node/s.
- c) Every change made in the primary node is replicated in every secondary node, as shown in Fig. 1.

From now on, the nodes are not all peers in the sense that not all the nodes are able to carry out the same operations for starting the replica set operation. More specifically, only one node should run the “initiate” operation:

```
1) $mongo
   > rs.initiate()
   ...
   > rs.status()
```

where `rs.status()` reports some details of the replication set currently operating. Even when the replication set operation has begun in the node in which `rs.initiate()` has been run, there is no actual replication because there is only one node in the replication set so far. The node in which the `rs.initiate()` has been successfully run has to include/reference the other nodes started with the same replication set in its configuration file. The `rs.add()` has to be used in so far unique node of the operating replication set. More specifically, if `rs.initiate()` has been run in the mv with IP 192.168.33.20 (which happens to be mongo1), the nodes with IP 192.168.33.21 and 192.168.33.22 can be included/started in the replication set with:

```
2) > rs.add('192.168.33.21:27017')
   ...
   > rs.add('192.168.33.22:27017')
   ...
```

Usually, the node in which the `rs.add()` is/are executed becomes the replication set primary node, while the others become secondary ones. However, there is no documentation about the way in which one node becomes the first primary node, only that an election is carried out when a replica set starts execution [3]. After adding nodes to the replica set, it is suggested to run `rs.status()` in order to show several details such as which node is the current primary node, and if every replication set node is recognized as “up & running”.

3.3 Script for Restarting the Operation without a Replica Set Configuration

We suggest a method to restart the entire system (the three vm) without replica set after having run the replica set system for a number of reasons:

- Restart the system from scratch for the cases in which some error in the configuration files has been included. Other problems arise in the case of executing `rs.initiate()` in more than one replication set node.
- Start the replica set in different scenarios, or at least in different nodes in order to test, for example, which node is elected as the first primary node (in the very first election).
- Performance analysis depending on different first-elected primary node.

The step-by-step guide is as follows.

- 1) Exit console (exit)
- 2) \$ sudo service mongod stop
- 3) Edit the mongod configuration file so that the replica set is not started, comment out the line `#replSet = myreplica` in file `/etc/mongod.conf`

- 4) \$ sudo service mongod start
- 5) \$ mongo (warning about replica set problems...)
 - > use local
 - > db.dropDatabase()
 - > exit
- 6) \$ sudo service mongod restart

If the mongo console is started, there is no more warning/indication of replica set problem/operation. We are able to automate the previous task with the following script, “norepl” which takes advantage of the configuration files already copied/generated in Section 3.1 above:

/etc/mongod.conf (the one to be taken into account in mongo startup)
 /etc/mongod.conf.orig (the one created at mongo install, with no replication set whatsoever)
 /etc/mongod.conf.repl (the one created for mongo replication set startup)

```
norepl
=====
sudo cp /etc/mongod.conf.orig /etc/mongod.conf
sudo service mongod restart
echo "Step 1:"
echo " mongo"
echo " > use local"
echo " > db.dropDatabase()"
echo " > exit"
echo ""
echo "Step 2:"
echo " nothing"
=====
```

The script (norepl) should be run twice in all the nodes in order to get a mongo console with no replication set started in any mv.

4.- The Vagrantfile for a “Complete” Installation

Most of the tasks described in previous section can be made automatically. Actually, programmatic installation and configuration of vm is always preferred (saving time, reducing possible typos/errors, etc.). Some basic tasks such as setting hostnames and /etc/hosts data as well as installation of the mongod software are the first candidates. The Vagrantfile shown below includes most of the useful instructions to install and configure a 3 vm environment for mongo replica set. Actually, the only two extra tasks to start the mongo replica set operations are:

- 1) rs.initiate()
- 2) rs.add(...) for every node running as part of the mongo replica set to be executed in the mongo console of a node (any of them).

===== Vagrantfile =====

```
base_network = "192.168.33."  
base_host = 20
```

```
Vagrant.configure("2") do |config|  
  3.times do |num|  
    config.vm.define ("mongo%01d" % (num + 1)) do |machine|  
      machine.vm.box = "bento/ubuntu-16.04"  
      machine.vm.network "private_network", ip: "#{base_network}#{base_host + num}"  
      machine.vm.hostname = "mongo#{num+1}"  
  
      machine.vm.provider "virtualbox" do |vb|  
        vb.memory = "512"  
      end  
  
      machine.vm.provision "shell", inline: <<-SHELL  
        # Adding IPs and hostnames... Warning: this is hardcoded... and has a long list  
of dependencies...  
        echo "" | sudo tee -a /etc/hosts  
        echo "192.168.33.20 mongo1" | sudo tee -a /etc/hosts  
        echo "192.168.33.21 mongo2" | sudo tee -a /etc/hosts  
        echo "192.168.33.22 mongo3" | sudo tee -a /etc/hosts  
  
        # Get updated urls/packages sources?  
        sudo apt-get update  
  
        # The joe editor  
        sudo apt-get install -y joe  
  
        # The alias for ls  
        grep -q "alias ls='ls -la'" /home/vagrant/.bashrc || echo -e "\\nalias ls='ls -  
la'" >> /home/vagrant/.bashrc  
  
        # Install mongod if needed  
        if ! which mongod > /dev/null ; then  
          sudo apt-get install -y mongod  
        fi  
  
        # Save mongod.conf  
        sudo cp /etc/mongod.conf /etc/mongod.conf.orig  
  
        # Make a copy of mongod.conf for replication  
        sudo cp /etc/mongod.conf /etc/mongod.conf.repl  
  
        # Comment bind line in mongod.conf.repl  
        sudo sed -i 's/bind_ip = 127.0.0.1/#bind_ip = 127.0.0.1/g'  
/etc/mongod.conf.repl  
  
        # Add replica conf.  
        echo "" | sudo tee -a /etc/mongod.conf.repl  
        echo "# replica name..." | sudo tee -a /etc/mongod.conf.repl  
        echo "replSet = myreplica" | sudo tee -a /etc/mongod.conf.repl  
  
        # And restart with replication... just for the first time, recently installed...  
        sudo cp /etc/mongod.conf.repl /etc/mongod.conf  
        sudo service mongod restart  
      SHELL  
    end  
  end  
end
```

=====

The name resolution data has been added by hardcoding, which is more than dangerous in general, but we have selected that method for the sake of a direct/better explanation. Beyond that Vagrantfile, we think it is useful to have the scripts for starting the replica set and restarting every mongod without any replica set. Basically, we could take advantage of the /vagrant directory and copy the corresponding scripts:

```
===== repl =====  
  
sudo cp /etc/mongodb.conf.repl /etc/mongodb.conf  
sudo service mongodb restart  
=====
```

```
===== norepl =====  
  
sudo cp /etc/mongodb.conf.orig /etc/mongodb.conf  
sudo service mongodb restart  
echo "Step 1:"  
echo "  mongo"  
echo "  > use local"  
echo "  > db.dropDatabase() "  
echo "  > exit"  
echo ""  
echo "Step 2:"  
echo "  nothing"  
=====
```

5.- Conclusions and Further Work

We have presented a simple and straightforward step-by-step guide for having a mongo replica set “up & running”, along with a script to start every node in a replica set. We have automated the replica set start and, also, the restart of every node so that no replica set remains in operation. We have added some comments we have seen by experience and not fully documented in other tutorials such as [4] or is widespread in tutorials and reference manual/s.

We have not covered all alternatives, mostly because either other ways of configuring and starting the replica set are well documented or because the guide and comments in this guide let to better understand other alternative ways of starting the system. Clearly, configuration and initial operation of a mongo replica set is not useful by itself, we expect to run a number of tests and experiments for functionality and performance analysis.

We also have taken advantage of the vagrant tool, which enables to programmatically install and configure the whole virtual environment, which in this case comprises three virtual machines. Besides, since the all the vm have network connectivity with the host we will be able to experiment in a heterogenous yet simple environment.

Acknowledgements

We would like to thank the invaluable and precise help of prof. Christian A. Rodríguez regarding every detail and complete explanation of the vagrant tool. He’s been always ready to show solutions to specific

issues and enhance the ways in which vagrant can be used for automating vm installation and configuration/s.

References

- [1] MongoDB, Inc, Replication, Replication - MongoDB Manual 3.4, 2017, <https://docs.mongodb.com/manual/replication/>
- [2] MongoDB, Inc, Replica Set Members - MongoDB Manual 3.4, 2017, <https://docs.mongodb.com/manual/core/replica-set-members/>
- [3] MongoDB, Inc, Replica Set Elections - MongoDB Manual 3.4, 2017, <https://docs.mongodb.com/manual/core/replica-set-elections/>
- [4] MongoDB, Inc, Deploy a Replica Set - MongoDB Manual 3.4, 2017, <https://docs.mongodb.com/manual/tutorial/deploy-replica-set/>
- [5] HashiCorp Vagrant, Development Environments Made Easy <https://www.vagrantup.com/>
- [6] HashiCorp Vagrant, “Project Setup”, https://www.vagrantup.com/intro/getting-started/project_setup.html
- [7] HashiCorp Vagrant, “Boxes”, <https://www.vagrantup.com/intro/getting-started/boxes.html>
- [8] HashiCorp Vagrant, “Provisioning”, <https://www.vagrantup.com/docs/provisioning/>
- [9] HashiCorp Vagrant, “Synced Folders”, <https://www.vagrantup.com/docs/synced-folders/>
- [10] M. Hashimoto, Vagrant: Up and Running: Create and Manage Virtualized Development Environments, O'Reilly Media, 2013, ISBN: 1449335837.
- [11] Fernando G. Tinetti, Agustín Terruzzi, “Install a mongodb Replica Set: 1 Primary Node and 2 Secondary Nodes”, III-LIDI, Facultad de Informática, UNLP, March 2017, Technical Report TR-BDD-01-2017, disponible en <http://fernando.bl.ee/>