
Install and Configure MongoDB on Ubuntu 14.04

Steps :

- 1) Install and configure mongodb master and slave node*
- 2) Arbiter node configure for failover*
- 3) testing*

Difference between “master-slave replication” and “replication set” in mongodb

The difference between a replication set and master-slave replication is that a replication set has an intrinsic automatic failover mechanism in case the primary member becomes unavailable.

Install and configure mongodb master and slave node

my configuration ::

192.168.1.253 ubuntu-server.com
192.168.1.254 ubuntu-client.com
192.168.1.234 biz.com

in initial condition;

<i>192.168.1.253</i>	<i>ubuntu-server.com</i>	<i>>></i>	<i>Primary Server</i>
<i>192.168.1.254</i>	<i>ubuntu-client.com</i>	<i>>></i>	<i>Secondary Server</i>
<i>192.168.1.234</i>	<i>biz.com</i>	<i>>></i>	<i>Arbiter</i>

<i>master node</i>	<i>>></i>	<i>primary member</i>
<i>slave node</i>	<i>>></i>	<i>secondary member</i>
<i>arbiter node</i>	<i>>></i>	<i>Arbiter</i>

Primary member: The primary member is the default access point for transactions with the replication set. It is the only member that can accept write operations.

Each replication set can have only one primary member at a time. This is because replication happens by copying the primary's "oplog" (operations log) and repeating the changes on the secondary's dataset. Multiple primaries accepting write operations would lead to data conflicts.

Secondary members: A replication set can contain multiple secondary members. Secondary members reproduce changes from the oplog on their own data.

Although by default applications will query the primary member for both read and write operations, you can configure your setup to read from one or more of the secondary members. A secondary member can become the primary if the primary goes offline or steps down.

Arbiter: An arbiter is an optional member of a replication set that does not take part in the actual replication process. It is added to the replication set to participate in only a single, limited function: to act as a tie-breaker in elections.

Consider you have 2 MongoDB nodes. One act as primary and the other once act as secondary node. If any one of the node goes down then MongoDB cannot able to choose a primary node by itself because MongoDB needs majority of members (at least two active nodes) to choose a primary node in a multi node replica set. As a result of this all write operations are failed and a lock occurs. In this scenario if an arbiter exists then the arbiter will vote for the available node to become primary.

An arbiter does **not** have a copy of data set and **cannot** become a primary. Replica sets may have arbiters to add a vote in [elections of for primary](#). Arbiters *always* have exactly 1 vote election, and thus allow replica sets to have an uneven number of members, without the overhead of a member that replicates data.

Do not run an arbiter on systems that also host the primary or the secondary members of the replica set.

In the event that the primary member becomes unavailable, an automated election process happens among the secondary nodes to choose a new primary. If the secondary member pool contains an even number of nodes, this could result in an inability to elect a new primary due to a voting impasse. The arbiter votes in these situations to ensure a decision is reached.

Set Up DNS Resolution

In order for our MongoDB instances to communicate with each other effectively, we will need to configure our machines to resolve the proper hostname for each member. You can either do this by [configuring subdomains for each replication member](#) or through editing the /etc/hosts file on each computer.

nano /etc/hosts

```
root@ubuntu-server: ~
root@ubuntu-server: ~
root@ubuntu-client: ~
root@biz: ~
GNU nano 2.2.6 File: /etc/hosts Modified
127.0.0.1 localhost
127.0.1.1 ubuntu-server.com
192.168.1.234 akhil.bizruntime.com
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

192.168.1.253 ubuntu-server.com
192.168.1.254 ubuntu-client.com
192.168.1.234 biz.com

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

do the above steps on all machines.

Prepare for Replication in the MongoDB Configuration File

The first thing we need to do to begin the MongoDB configuration is stop the MongoDB process on each server.

On each sever, type:

service mongod stop

may be sometime this will work but status will showing that service is stopped., but don't worry, it will work.

Now, we need to configure a directory that will be used to store our data. Create a directory with the following command:

mkdir -p /home/mongo/mongo-metadata

Now that we have the data directory created, we can modify the configuration file to reflect our new replication set configuration:

nano /etc/mongodb.conf

In this file, we need to specify a few parameters. First, adjust the dbpath variable to point to the directory we just created:

dbpath=/home/mongo/mongo-metadata

Remove the comment from in front of the port number specification to ensure that it is started on the default port:

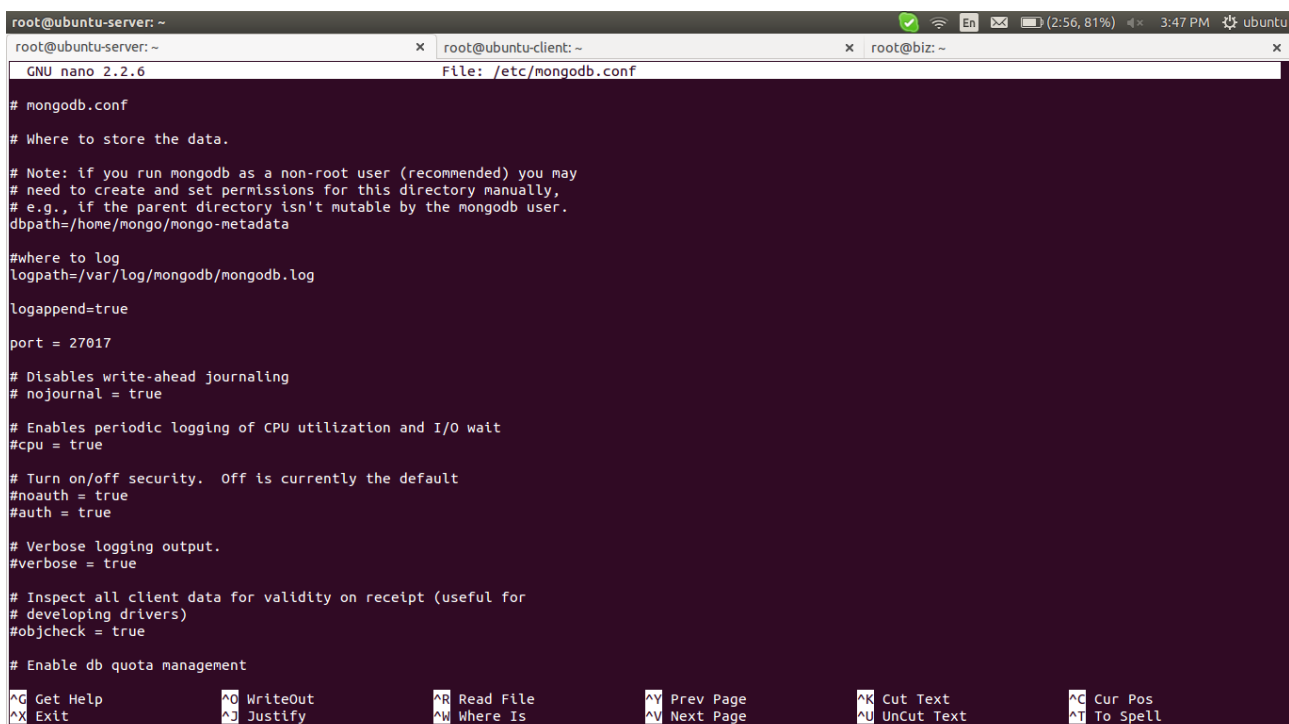
port = 27017

Towards the bottom of the file, remove the comment form in front of the replSet parameter. Change the value of this variable to something that will be easy to recognize for you.

replSet = rs0

Finally, you should make the process fork so that you can use your shell after spawning the server instance. Add this to the bottom of the file:

fork = true

A screenshot of a terminal window with three tabs: 'root@ubuntu-server: ~', 'root@ubuntu-client: ~', and 'root@biz: ~'. The active tab is 'root@ubuntu-server: ~', which shows the 'nano' text editor editing the file '/etc/mongodb.conf'. The file content is as follows:

```
# mongodb.conf
# Where to store the data.
# Note: if you run mongodb as a non-root user (recommended) you may
# need to create and set permissions for this directory manually,
# e.g., if the parent directory isn't mutable by the mongodb user.
dbpath=/home/mongo/mongo-metadata

#where to log
logpath=/var/log/mongodb/mongodb.log
logappend=true

port = 27017

# Disables write-ahead journaling
# nojournal = true

# Enables periodic logging of CPU utilization and I/O wait
#cpu = true

# Turn on/off security.  Off is currently the default
#noauth = true
#auth = true

# Verbose logging output.
#verbose = true

# Inspect all client data for validity on receipt (useful for
# developing drivers)
#objcheck = true

# Enable db quota management
```

The bottom of the terminal window shows the nano editor's command palette with various shortcuts like '^G Get Help', '^O WriteOut', '^R Read File', etc.

Save and close the file. Start the replication member by issuing the following command:

mongod --config /etc/mongodb.conf

These steps must be repeated on each member of the replication set.

Start the Replication Set and Add Members

Primary server Configuration :

Now that you have configured each member of the replication set and started the mongod process on each machine, you can initiate the replication and add each member. On one of your members, type:

```
# mongo
```

This will give you a MongoDB prompt for the current member. Start the replication set by entering:

```
> rs.initiate()
```

This will initiate the replication set and add the server you are currently connected to as the first member of the set. You can see this by typing:

```
> rs.conf()
```

Now, you can add the additional nodes to the replication set by referencing the hostname you gave them in the /etc/hosts file:

```
> rs.add("ubuntu-client.com")
```

Do this for each of your remaining replication members. Your replication set should now be up and running.

Now add the arbiter server with following command

```
# rs.addArb("biz.com")
```

This will initiate the replication set and add the server you are currently connected to as the first member of the set. You can see this by typing:

```
> rs.status()
```

```
rs0:PRIMARY> rs.status()
```

```
{
  "set" : "rs0",
  "date" : ISODate("2016-11-21T10:23:20Z"),
  "myState" : 1,
  "members" : [
    {
      "_id" : 0,
      "name" : "ubuntu-server.com:27017",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "uptime" : 7901,
      "optime" : Timestamp(1479723778, 1),
      "optimeDate" : ISODate("2016-11-21T10:22:58Z"),
      "self" : true
    },
    {
      "_id" : 1,
      "name" : "ubuntu-client.com:27017",
      "health" : 1,
      "state" : 2,
      "stateStr" : "SECONDARY",
      "uptime" : 56,
      "optime" : Timestamp(1479723778, 1),
      "optimeDate" : ISODate("2016-11-21T10:22:58Z"),
      "lastHeartbeat" : ISODate("2016-11-21T10:23:18Z"),
      "lastHeartbeatRecv" : ISODate("2016-11-21T10:23:20Z"),
      "pingMs" : 5,
      "lastHeartbeatMessage" : "syncing to: ubuntu-server.com:27017",
      "syncingTo" : "ubuntu-server.com:27017"
    },
    {
      "_id" : 2,
      "name" : "biz.com:27017",
      "health" : 1,
      "state" : 7,
      "stateStr" : "ARBITER",
      "uptime" : 1984,
      "lastHeartbeat" : ISODate("2016-11-21T10:23:19Z"),
      "lastHeartbeatRecv" : ISODate("2016-11-21T10:23:19Z"),
      "pingMs" : 8
    }
  ],
  "ok" : 1
}
```

Now restart the MongoDB instance in all the three servers

service mongod restart

do this on all servers

Testing Replication and Failover

Replication Test :

in Primary server :

create database,

> use <test1>

add some content to this database

> db.test1.insert({item: "name1"})

check whether the database created or not,

> show dbs

Do all this steps in Secondary server, it will not create database, (only Read permission)

Failover test :

in primary server :

stop the service or break the network connection to that server

after sometime check in Secondary server by enter, it will shows that it changed from Secondary to primary

in secondary server :

```
root@ubuntu-server: ~
root@ubuntu-server: ~
root@ubuntu-client: ~
root@biz: ~
root@ubuntu-server:~# mongo
MongoDB shell version: 2.4.14
connecting to: test
rs0:SECONDARY>
rs0:PRIMARY>
rs0:PRIMARY>
rs0:PRIMARY>
rs0:PRIMARY>
rs0:PRIMARY>
```

It can also visible in Arbitary node,

> *rs.conf()*

Links ::

failover with arbitary ::

<https://docs.mongodb.com/v3.0/core/replica-set-arbiter/>

<http://www.congruentsolutions.com/mongodb-replication-approach-setup-using-arbiter/>

installation and configuration::

<https://www.digitalocean.com/community/tutorials/how-to-install-mongodb-on-ubuntu-12-04>

<https://www.digitalocean.com/community/tutorials/how-to-implement-replication-sets-in-mongodb-on-an-ubuntu-vps>
