How To Implement Replication Sets in MongoDB on an Ubuntu

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=158*

Introduction

MongoDB is an extremely popular NoSQL database. It is often used to store and manage application data and website information. MongoDB boasts a dynamic schema design, easy scalability, and a data format that is easily accessible programmatically.

In this guide, we will discuss how to set up data replication in order to ensure high availability of data and create a robust failover system. This is important in any production environment where a database going down would have a negative impact on your organization or business.

We will assume that you have already installed MongoDB on your system.

What is a MongoDB Replication Set?

MongoDB handles replication through an implementation called "replication sets". Replication sets in their basic form are somewhat similar to nodes in a master-slave configuration. A single primary member is used as the base for applying changes to secondary members.

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.

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.

Note: Due to the fact that data is transfered asynchronously, reads from secondary nodes can result in old data being served. If this is a concern for your use-case, you should not enable this functionality.

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.

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.

If a replication set has only one secondary member, an arbiter is required.

Secondary Member Customization Options

There are instances where you may not want all of your secondary members to be beholden to the standard rules for a replication set. A replication set can have up to 12 members and up to 7 will vote in an election situation.

Priority 0 Replication Members

There are some situations where the election of certain set members to the primary position could have a negative impact on your application's performance.

For instance, if you are replicating data to a remote datacenter or a specific member's hardware is inadequate to perform as the main access point for the set, setting priority 0 can ensure that this member will not become a primary but can continue copying data.

Hidden Replication Members

Some situations require you to separate the main set of members accessible and visible to your clients from the background members that have separate purposes and should not interfere.

For instance, you may need a secondary member to be the base for analytics work, which would benefit from an up-to-date dataset but would cause a strain on working members. By setting this member to hidden, it will not interfere with the general operations of the replication set.

Hidden members are necessarily set to priority 0 to avoid becoming the primary member, but they do vote in elections.

Delayed Replication Members

By setting the delay option for a secondary member, you can control how long the secondary waits to perform each action it copies from the primary's oplog.

This is useful if you would like to safeguard against accidental deletions or recover from destructive operations. For instance, if you delay a secondary by a half-day, it would not immediately perform accidental operations on its own set of data and could be used to revert changes.

Delayed members cannot become primary members, but can vote in elections. In the vast majority of situations, they should be hidden to prevent processes from reading data that is out-of-date.

How to Configure a Replication set

To demonstrate how to configure replication sets, we will configure a simple set with a primary and two secondaries. This means that you will need three server instances to follow along. We will be using Ubuntu 12.04 machines.

You will need to install MongoDB on each of the machines that will be members of the set.

Once you have installed MongoDB on all three of the server instances, we need to configure some things that will allow our server intance to communicate with each other.

The following steps assume that you are logged in as the root user.

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.

It is probably better to use subdomains in the long run, but for the sake of getting off the ground quickly, we will do this through the hosts file.

On each of the soon-to-be replication members, edit the /etc/hosts file:

nano /etc/hosts

After the first line that configures the localhost, you should add an entry for each of the replication sets members. These entries take the form of:

ip_address mongohost0.example.com

You can get the IP addresses of the members of your set in the ASPHostPortal.com server instance. The name you choose as a hostname for that computer is arbitrary, but should be descriptive.

This file should (mostly) be the same across all of the hosts in your set. Save and close the file on each of your members.

Next, you need to set the hostname of your system to reflect these new changes. The command on each server will reflect the name you gave that specific machine in the /etc/hosts file. You should issue a command on each server that looks like:

hostname mongo0.example.com

Modify this command on each server to reflect the name you selected for it in the file.

Edit the /etc/hostname file to reflect this as well:

nano /etc/hostname

mongo0.example.com

These steps should be performed on each node.

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 mongodb stop

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

mkdir /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=/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 replSetparameter. 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

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

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()

{

"_id" : "rs0"

"version" : 1,

"members" : [

{

"_id" : 0,

"host" "mongo0.example.com:27017"

}

]

}

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("mongo1.example.com")

{ "ok" : 1 }

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

Conclusion

By properly configuring replication sets for each of your data storage targets, your databases will be protected in some degree from unavailability and hardware failure. This is essential for any production system.

Replication sets provide a seamless interface with applications because they are essentially invisible to the outside. All replication mechanics are handled internally. If you plan on implementing MongoDB sharding, it is a good idea to implement replication sets for each of the shard server components.